

ANEXO I

Roteiro de Métricas para Serviços de desenvolvimento de aplicações web

Os Story Points são uma unidade de medida utilizada em metodologias ágeis, como Scrum, para estimar o esforço necessário para desenvolver uma funcionalidade ou realizar uma tarefa em um projeto de software. Diferentemente de estimativas tradicionais baseadas em horas, os Story Points medem a complexidade, o risco e o volume de trabalho envolvido, proporcionando uma visão mais flexível e adaptável às variações do desenvolvimento.

Essa abordagem leva em conta a dificuldade técnica, o tempo estimado e as incertezas relacionadas a cada tarefa, permitindo um planejamento mais eficiente e realista das entregas. Cada tarefa ou funcionalidade é atribuída a um valor em Story Points com base em sua complexidade relativa, sendo que tarefas mais simples recebem menos pontos e tarefas mais complexas, mais pontos. Isso permite que o progresso seja mensurado de forma objetiva, ajudando a equipe de desenvolvimento a prever com mais precisão o que pode ser entregue dentro de um determinado prazo.

A adoção de Story Points como método de mensuração oferece diversas vantagens para a PPSA na gestão dos Serviços de Desenvolvimento de Aplicações Web (Fábrica de Software). Além de permitir uma estimativa clara do esforço necessário para cada funcionalidade, o uso de Story Points facilita a adaptação contínua dos serviços de desenvolvimento, à medida que novas demandas e prioridades surgem ao longo do contrato. Esse método também é essencial para promover a transparência e a previsibilidade nas entregas, visto que o progresso pode ser acompanhado de forma incremental e frequente.

Por meio da alocação de Story Points, a PPSA poderá monitorar de maneira precisa o desempenho do prestador de serviços, garantindo que o pagamento esteja diretamente relacionado ao valor entregue. Isso proporciona uma maneira eficiente de associar a quantidade de trabalho ao cumprimento dos objetivos da organização, estabelecendo uma relação direta entre esforço e resultado, e assegurando que os serviços estejam sempre alinhados às necessidades da empresa.

A produtividade média esperada é de **4 horas por Story Point**. Essa medida servirá apenas como um referencial balizador. É importante ressaltar que o pagamento não será realizado com base no número de horas trabalhadas, mas

sim pela entrega das solicitações, mensuradas em **Story Points**. A quantidade de horas referida é apenas um parâmetro auxiliar para evitar distorções de percepção durante o processo de precificação.

A solicitação de serviços de desenvolvimento de software pela PPSA será formalizada por meio da abertura de um chamado na ferramenta de ITSM. Nessa ferramenta, a PPSA deverá detalhar os requisitos específicos para o desenvolvimento de melhorias ou novas funcionalidades nos sistemas existentes, bem como iniciar o desenvolvimento de novos sistemas, sempre de acordo com as necessidades da organização.

Exemplos de Solicitações:

- Melhoria em sistemas já existentes: Como ajustes em funcionalidades, otimização de processos internos ou integração com novas ferramentas.
- Desenvolvimento de novos módulos: Expansão de funcionalidades para sistemas em operação, incluindo a adição de novos recursos.
- Criação de novos sistemas: Projetos inteiramente novos que envolvem a análise, planejamento e desenvolvimento de sistemas personalizados para atender às necessidades da PPSA.
- Correção de defeitos complexos: Solicitação para corrigir erros críticos que não puderam ser resolvidos pelo suporte técnico habitual e exigem desenvolvimento adicional.
- Desenvolvimento de Relatórios Customizados (BI): Criação de novos relatórios interativos em sistemas web de Business Intelligence (BI), como Power BI, com integração de diferentes fontes de dados e visualização de KPIs.
- Integração com Sistemas Externos via API: Desenvolvimento de integrações com sistemas externos, como plataformas de gestão de contratos ou serviços financeiros ou páginas web.
- Otimização de Desempenho de Aplicações Web: Ajustes de código e configuração de sistemas para melhorar a performance de sistemas já existentes, como tempo de resposta e eficiência de banco de dados.
- Desenvolvimento de Funcionalidades de Segurança: Implementação de medidas como autenticação multifator (MFA), criptografia de dados e controle de permissões baseado em papéis (RBAC).
- Automatização de Processos Internos: Desenvolvimento de soluções que automatizam processos manuais, como geração e envio de relatórios ou notificações automáticas.

- Desenvolvimento de Módulos de Auditoria e Conformidade: Adição de funcionalidades para monitorar alterações feitas em sistemas, garantindo conformidade com normas regulatórias.
- Personalização de Interface do Usuário (UI/UX): Ajustes no layout, design e navegação dos sistemas web para melhorar a experiência do usuário e compatibilidade com dispositivos móveis.
- Criação de Landing Pages ou Hotsites Temporários: Desenvolvimento de páginas temporárias para campanhas internas ou comunicação de eventos importantes.
- Implementação de Funcionalidades de Armazenamento em Nuvem: Integração de funcionalidades que permitam upload, armazenamento e compartilhamento de arquivos diretamente em plataformas de nuvem, como Azure.
- Desenvolvimento de Sistemas de Gerenciamento de Documentos: Criação de soluções para armazenar, organizar e gerenciar documentos com funcionalidades como controle de versão e permissões de acesso.
- Atualização Tecnológica e Migração para Novas Plataformas: Migração de sistemas legados para tecnologias modernas, como React ou Angular, ou plataformas de nuvem.
- Desenvolvimento de Ferramentas de Colaboração Online: Implementação de ferramentas para colaboração interna, como plataformas de videoconferência ou compartilhamento de documentos.
- Criação de Portais de Autosserviço para Usuários: Desenvolvimento de portais web onde os usuários possam realizar ações de forma independente, como a atualização de dados cadastrais ou submissão de pedidos de suporte.
- Implementação de Notificações Automáticas: Configuração de sistemas para enviar notificações automáticas aos usuários em eventos críticos, como prazos de vencimento ou conclusão de tarefas.
- Integração com Ferramentas de Armazenamento Externo (One Drive e Google Drive): Integração com plataformas externas de armazenamento para facilitar o compartilhamento e organização de documentos.
- Desenvolvimento de Ferramentas de Automação de Testes: Implementação de sistemas para a automação de testes unitários e de integração durante o desenvolvimento de novos módulos.
- Desenvolvimento de Ferramentas de E-learning Interno: Criação de uma plataforma interna de aprendizado para capacitação dos colaboradores da PPSA.
- Desenvolvimento de Sistemas de Planejamento Orçamentário: Criação de sistemas que facilitem o planejamento e o acompanhamento

orçamentário, permitindo a visualização de receitas e despesas em tempo real.

- Desenvolvimento de Ferramentas de Auditoria Automática: Implementação de ferramentas para auditoria de conformidade automatizada, com geração de relatórios em tempo real.
- Desenvolvimento de Funcionalidades de Georreferenciamento: Adição de mapas e localização de ativos ou áreas operacionais em sistemas web da PPSA.
- Implementação de Funcionalidades de Análise de Sentimento: Utilização de algoritmos de análise de sentimento em comentários ou feedbacks de colaboradores para obter insights.
- Desenvolvimento de Funcionalidades de Acessibilidade: Melhorias no design e usabilidade dos sistemas para atender às necessidades de usuários com deficiências visuais ou motoras.
- Criação de Ferramentas de Pesquisa Interna: Desenvolvimento de ferramentas de busca avançada para facilitar a localização de documentos e informações nos sistemas internos da PPSA.
- Desenvolvimento de Módulos de Gestão de Riscos: Implementação de sistemas para identificar, mitigar e monitorar riscos operacionais e estratégicos.
- Desenvolvimento de Funcionalidades de Gamificação: Adição de elementos de gamificação, como pontuações e metas, para incentivar a adoção de novas funcionalidades nos sistemas web.
- Desenvolvimento de Ferramentas para integração com certificados digitais
- Implementação de Funcionalidades de Assinatura Eletrônica: Adição de módulos para permitir assinaturas eletrônicas em documentos dentro dos sistemas da PPSA.
- Desenvolvimento de Ferramentas para Gestão de Ativos: Criação de sistemas que permitam o rastreamento e controle do ciclo de vida dos ativos tecnológicos e físicos da organização.

A partir da abertura do chamado, a CONTRATADA terá um prazo de **7 (sete) dias** corridos para revisar a solicitação e fornecer a estimativa inicial em Story Points, indicando o esforço necessário para realizar o desenvolvimento. A quantidade de pontos deverá ser estimada com base no roteiro de métricas estabelecido neste anexo, garantindo a coerência das estimativas com os critérios pré-definidos.

Durante esse período, o chamado poderá ser pausado pela CONTRATADA para a obtenção de maiores esclarecimentos junto à PPSA, caso a descrição original não seja suficiente para uma estimativa precisa. A CONTRATADA poderá solicitar mais

detalhes, anexos adicionais, ou rascunhos de telas e fluxos, a fim de garantir uma compreensão clara do escopo do desenvolvimento. Esse processo de esclarecimento é essencial para assegurar que o desenvolvimento seja bem direcionado e que as expectativas da PPSA sejam atendidas de forma eficiente.

O desenvolvimento do que for solicitado só terá início após a aprovação da PPSA. Após essa aprovação, um **novo chamado será aberto, contendo o tempo de resolução estimado no processo anterior** pela CONTRATADA com base na metodologia ágil adotada. Nesse momento, a CONTRATADA iniciará o desenvolvimento da solicitação conforme o escopo acordado, realizando as entregas parciais de acordo com as sprints definidas. As entregas serão feitas diretamente à PPSA, permitindo o acompanhamento contínuo do progresso e a validação de cada etapa.

Todas as interações, dúvidas e entregas realizadas após o processo de Quality Assurance (QA) serão devidamente registradas no chamado, garantindo total rastreabilidade das atividades e eventuais ajustes necessários. Durante o desenvolvimento, quaisquer dúvidas que possam surgir também serão registradas e tratadas no próprio chamado para garantir a transparência e o acompanhamento das demandas pela PPSA.

Após o fechamento do chamado, todos os indicadores de IMR descritos no Termo de Referência serão considerados para o cálculo do pagamento referente aos Story Points executados. O cumprimento desses indicadores será determinante para assegurar que a qualidade, prazos e demais critérios de avaliação sejam atendidos conforme o esperado, impactando diretamente o processo de pagamento.

Para realizar a estimativa, a CONTRATADA deverá utilizar um ou mais macros assuntos para calcular os Story Points correspondentes à solicitação da PPSA. Cada um dos macros assuntos está subdividido em tarefas organizadas do menor ao maior grau de dificuldade, possibilitando a correta avaliação do esforço necessário.

A CONTRATADA deverá escolher o grau de dificuldade e a semelhança técnica da solicitação com os exemplos fornecidos, atribuindo a pontuação de Story Points de forma adequada para cada parte da demanda. Assim, a CONTRATADA assegurará que o desenvolvimento será estimado de maneira objetiva e transparente, permitindo à PPSA um acompanhamento claro do progresso e do esforço empregado em cada fase do projeto.

Conceitos Gerais de uso

Neste roteiro de métricas, utilizamos a métrica Story Points baseada na sequência de Fibonacci (1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144) para estimar o esforço necessário para a execução de cada atividade no desenvolvimento. A escolha dessa sequência se dá pela sua ampla adoção em metodologias ágeis, como o Scrum, e por refletir de forma escalonada o aumento gradual de complexidade entre as tarefas.

Contudo, entendemos que certas atividades possuem uma variação de complexidade muito sutil entre os níveis consecutivos da sequência, enquanto outras exigem um aumento significativo de esforço. Com isso em mente, ajustamos os Story Points da seguinte forma, visando proporcionar maior clareza na diferenciação de esforço:

Muito Simples (1 Story Point): Tarefas extremamente básicas, como ajustes pequenos em interfaces ou modificações leves em funcionalidades já existentes.

Simples (5 Story Points): Tarefas que envolvem algum nível de desenvolvimento, como a implementação de componentes simples ou modificações de lógica sem grande impacto estrutural.

Intermediária (13 Story Points): Tarefas que exigem maior esforço e planejamento, como a criação de novos módulos, integrações simples entre sistemas ou ajustes complexos em lógicas existentes.

Complexa (34 Story Points): Tarefas que envolvem o desenvolvimento de sistemas ou funcionalidades com alto nível de detalhamento técnico, dependências entre diferentes módulos ou impacto direto na estrutura do sistema.

Muito Complexa (89 Story Points): Tarefas que envolvem o desenvolvimento de sistemas inteiros, integrações de alta complexidade ou funcionalidades com várias camadas de dependências, exigindo alto esforço.

Ao adotar essa categorização, pulamos alguns números da sequência de Fibonacci para evitar distorções no esforço estimado e na precificação associada. Isso garante que os valores atribuídos a cada nível de complexidade estejam alinhados com o esforço real necessário para sua execução, evitando tanto a subvalorização de tarefas complexas quanto a sobrevalorização de tarefas simples.

Dessa forma, asseguramos uma maior transparência e previsibilidade no processo de precificação e execução das atividades, além de garantir que os fornecedores compreendam de maneira clara os critérios utilizados para avaliar o esforço de cada tarefa.

Tarefas repetitivas:

Repetição – Aumento de Esforço em 10% por Instância Adicional

Quando se repete a mesma tarefa várias vezes, como a criação de várias páginas com a mesma estrutura, o esforço adicional para cada nova instância é menor. Isso se deve ao fato de que a complexidade técnica e o trabalho de configuração inicial já foram realizados, e a duplicação ou criação de mais instâncias da mesma funcionalidade não exige o mesmo nível de esforço.

Conceito de Repetição:

Cada vez que uma tarefa é repetida, o esforço aumenta apenas em 10% sobre o esforço original. Isso reflete o fato de que a maior parte do trabalho já foi realizada, e o esforço incremental é muito menor.

Fórmula:

$$\text{Esforço total} = \text{Esforço base} + (\text{Esforço base} \times \text{Número de repetições} \times 0,1)$$

Exemplo:

Se a criação de uma página com imagem estática e texto custa 1 Story Point, e a solicitação é criar 10 páginas iguais, o esforço adicional para as 9 páginas seguintes será de 10% por página.

Aplicando a fórmula:

$$\text{Esforço total} = 1 + (1 \times 9 \times 0,1) = 1 + 0,9 = 1,9 \text{ Story Points}$$

Reaproveitamento de Código

Redução de 50% no Esforço

Quando uma tarefa envolve o reaproveitamento de código existente, como um layout ou um componente já desenvolvido, o esforço necessário para implementar essa funcionalidade novamente é significativamente menor. Isso ocorre porque a maior parte do trabalho já foi realizada na primeira implementação, e reutilizar o código reduz o esforço.

Conceito de Reaproveitamento:

Ao reutilizar código ou layout já existente, o esforço original é reduzido em 50%. Isso significa que a tarefa não será contada como se fosse criada do zero, refletindo a economia de esforço trazida pela reutilização.

Fórmula:

Esforço reduzido = Esforço original × 0,5

Exemplo:

Se a criação de uma página do zero com uma imagem estática e texto custa 1 Story Point, mas o layout já foi criado e será reutilizado, o esforço para essa página será reduzido para 0,5 Story Point.

Além disso, se houver repetição (mais páginas a serem criadas usando o mesmo layout), aplicamos a regra de aumento de 10% por página adicional.

Aplicando a fórmula:

$\text{Esforço total} = 0,5 + (0,5 \times \text{Número de repetições} \times 0,1)$

Para 10 páginas, o cálculo seria:

$\text{Esforço total} = 0,5 + (0,5 \times 9 \times 0,1) = 0,5 + 0,45 = 0,95 \text{ Story Points}$

Abaixo seguem os macros assuntos, onde cada um fornece uma lista de exemplos de tarefas, organizadas do menor grau de complexidade em Story Points até o mais alto. Essas tarefas servem como referência para a CONTRATADA estimar e atribuir a pontuação adequada com base na solicitação da PPSA, assegurando que o desenvolvimento seja mensurado de maneira precisa, considerando tanto a dificuldade técnica quanto o esforço envolvido em cada etapa.

Desenvolvimento de Funcionalidades de Frontend

O Desenvolvimento de Funcionalidades de Frontend é uma área central no processo de criação de interfaces interativas, sendo responsável por transformar o design visual e as interações propostas em código funcional. Tarefas nesse macro assunto envolvem a implementação de elementos visuais, manipulação de estado, e integração com APIs e outras partes do sistema, sempre com foco na experiência do usuário e na responsividade da interface. O Frontend é, portanto, o ponto de convergência entre a visão estética e a lógica de negócios.

Embora o Frontend tenha interseções naturais com outros macros assuntos, como Design e UX/UI, Segurança, e Integração de APIs, seu foco está na aplicação

técnica dessas áreas. No caso de Design e UX/UI, por exemplo, o Design lida com a concepção visual e a experiência do usuário, enquanto o Frontend implementa essa visão no código, garantindo que o layout e as interações funcionem corretamente em diferentes dispositivos e navegadores. Da mesma forma, o Frontend aplica práticas de Segurança na interface, como autenticação e validação de entrada, mas sem adentrar os aspectos mais profundos da arquitetura de segurança que são tratados em outro macro assunto.

Quando se trata de APIs e Serviços Externos, o Frontend frequentemente consome esses serviços, integrando dados dinâmicos em tempo real à interface do usuário. No entanto, o foco aqui é garantir que a interação do usuário com esses dados seja eficiente e fluida, utilizando as melhores práticas de desenvolvimento de interface e performance. Assim, enquanto essas interseções são importantes, o Frontend tem seu papel distinto na transformação de conceitos visuais e lógicos em uma experiência de usuário prática e eficiente.

1 (um) Story Point

Tarefas de 1 Story Point no desenvolvimento de Frontend são caracterizadas por sua baixa complexidade e rápido tempo de execução. Essas tarefas envolvem ajustes simples na interface, sem exigir modificações profundas na lógica de negócios ou manipulação complexa de dados. O objetivo principal é a melhoria visual ou funcional da interface, com impacto limitado na arquitetura do sistema e pouca necessidade de testes extensivos.

Geralmente, essas tarefas incluem ajustes visuais ou pequenos incrementos na funcionalidade, como correções de layout, troca de cores ou fontes, e ajustes de espaçamento entre elementos. Por exemplo, a troca de ícones ou imagens estáticas, a aplicação de bordas ou sombras em botões, ou a adição de efeitos simples de "hover" são atividades comuns nessa categoria. Tais modificações visam aprimorar a experiência do usuário sem afetar o desempenho geral da aplicação.

Além disso, tarefas de 1 Story Point também podem incluir pequenas interações de usuário, como a implementação de um botão simples que executa uma ação básica sem exigir manipulação de estado ou interação com APIs. Correções de responsividade em dispositivos móveis, ajustes de alinhamento, e pequenas correções de bugs na interface também fazem parte desse escopo. O impacto dessas tarefas é pontual e, embora contribuam para a usabilidade, seu nível de complexidade é mínimo, o que justifica a classificação de 1 Story Point.

Exemplos:

- Criação de um botão simples: Implementar um botão com texto fixo que executa uma ação básica de clique sem lógica adicional.
- Ajuste de estilo em um componente existente: Alterar a cor, fonte ou tamanho de um componente já existente para melhorar a aparência visual.
- Adição de uma imagem estática à página: Inserir uma imagem fixa em uma posição específica da interface, sem interação ou animação.
- Atualização de texto estático em uma página: Modificar o conteúdo textual de um parágrafo ou cabeçalho para refletir informações atualizadas.
- Criação de um link para uma página externa: Implementar um link que, ao ser clicado, redireciona o usuário para um site externo.
- Implementação de um tooltip simples: Adicionar um tooltip que exibe uma mensagem fixa quando o usuário passa o mouse sobre um elemento.
- Adição de um ícone a um botão ou link: Inserir um ícone de uma biblioteca (como Font Awesome) em um botão ou link existente.
- Criação de um separador (divider) entre seções: Adicionar uma linha ou elemento visual que separa diferentes partes da interface.
- Ajuste de espaçamento (margin/padding) entre elementos: Modificar o espaçamento entre componentes para melhorar o layout da página.
- Alteração de um atributo alt em uma imagem: Atualizar o texto alternativo de uma imagem para melhorar a acessibilidade.
- Implementação de uma lista simples não ordenada: Adicionar uma lista de itens estáticos sem interação ou estilos avançados.
- Criação de um componente de texto estático: Implementar um componente que exibe um texto fixo, como um aviso ou nota.
- Atualização de links de mídia social: Modificar URLs dos ícones de mídia social para apontar para os perfis corretos.
- Adição de um favicon ao site: Inserir um ícone que aparece na aba do navegador ao acessar o site.
- Implementação de um rodapé (footer) simples: Adicionar um rodapé com texto estático, como direitos autorais ou informações de contato.
- Criação de um cabeçalho (header) com logo: Implementar um cabeçalho que exibe o logotipo da empresa sem menus ou interação.
- Ajuste de cores para seguir a paleta da marca: Atualizar cores de elementos para alinhar com as cores oficiais da empresa.
- Adição de atributos de acessibilidade básicos: Incluir atributos como aria-label em elementos para melhorar a acessibilidade.

- Implementação de um placeholder em um campo de entrada: Adicionar um texto de exemplo em um campo de input que indica ao usuário o que inserir.
- Criação de uma mensagem de erro estática: Exibir uma mensagem de erro fixa na interface sem lógica de validação.
- Adição de um botão de voltar ao topo: Inserir um botão que permite ao usuário voltar ao topo da página sem animações complexas.
- Implementação de um ícone de carregamento estático: Exibir um ícone de loading fixo sem lógica para mostrar carregamento real.
- Ajuste de alinhamento de texto: Alterar o alinhamento de um parágrafo ou cabeçalho (esquerda, direita, centro).
- Criação de um campo de entrada simples: Adicionar um input de texto sem validação ou lógica adicional.
- Adição de um título (title) em um link: Incluir um atributo title em um link para mostrar uma dica ao passar o mouse.
- Implementação de um botão desabilitado: Criar um botão que está visualmente desabilitado sem lógica para habilitar ou desabilitar.
- Ajuste de bordas em elementos: Alterar o estilo ou espessura das bordas de um componente para melhorar o design.
- Criação de um parágrafo com formatação básica: Adicionar um texto com negrito, itálico ou sublinhado.
- Adição de um comentário no código para documentação: Inserir comentários no código para explicar a funcionalidade de um trecho simples.
- Implementação de uma lista ordenada: Adicionar uma lista numerada com itens estáticos.
- Ajuste de opacidade de um elemento: Alterar a transparência de uma imagem ou div para efeitos visuais simples.
- Criação de um componente de etiqueta (label): Adicionar uma etiqueta a um campo de formulário para indicar seu propósito.
- Adição de um link de download para um arquivo estático: Implementar um link que permite ao usuário baixar um arquivo disponível no servidor.
- Implementação de um estilo hover simples: Alterar a cor ou estilo de um elemento quando o usuário passa o mouse sobre ele.
- Ajuste de tamanho de fonte em um elemento: Modificar o tamanho da fonte de um texto para melhorar a legibilidade.
- Criação de um formulário com um campo: Implementar um formulário simples com um único campo de entrada e nenhum processamento.
- Adição de um texto de direitos autorais: Inserir um aviso de copyright no rodapé da página.

- Implementação de uma imagem de fundo fixa: Adicionar uma imagem como background de um elemento sem efeitos de parallax.
- Ajuste de z-index para posicionamento: Alterar a ordem de sobreposição de elementos para corrigir problemas visuais simples.
- Criação de um campo de seleção (select) com opções estáticas: Adicionar um dropdown com opções fixas sem lógica de seleção.
- Adição de um link âncora na página: Criar um link que leva o usuário a uma seção específica da mesma página.
- Implementação de estilos responsivos simples: Utilizar média queries básicas para ajustar estilos em diferentes tamanhos de tela.
- Ajuste de formato de data estático: Exibir uma data em um formato específico sem lógica para atualizar dinamicamente.
- Criação de um ícone de busca sem funcionalidade: Inserir um ícone de lupa para representar uma busca sem implementar a lógica.
- Adição de um estilo de lista personalizada: Alterar os marcadores de uma lista para um estilo personalizado.
- Implementação de um texto em destaque (blockquote): Adicionar um bloco de citação com formatação básica.
- Ajuste de estilos de link visitado: Alterar a cor de links após serem clicados para indicar que já foram visitados.
- Criação de um componente de separador horizontal (HR): Inserir uma linha horizontal para separar conteúdo.
- Adição de uma mensagem de boas-vindas estática: Exibir uma saudação fixa ao usuário sem personalização.
- Implementação de um background colorido em um elemento: Alterar a cor de fundo de um div ou seção para melhorar o design.
- Criação de um formulário de busca sem funcionalidade: Adicionar um campo de input para busca sem implementar a lógica de pesquisa.
- Adição de uma tabela com dados estáticos: Inserir uma tabela simples com informações fixas, sem ordenação ou paginação.
- Implementação de um modal estático: Adicionar um modal que exibe informações fixas, sem lógica para abrir ou fechar.
- Ajuste de cursor em elementos interativos: Alterar o cursor para pointer quando passar o mouse sobre botões ou links.

5 (cinco) Story Points

As tarefas de 5 Story Points no desenvolvimento de Frontend envolvem uma complexidade moderada e exigem mais tempo e esforço do que ajustes simples.

Essas tarefas geralmente incluem a implementação de lógica básica, como validação de formulários, manipulação de estado, ou integração com APIs para operações simples, como leitura de dados. O foco está em funcionalidades que começam a introduzir interatividade e manipulação de dados na interface do usuário, mas que ainda não envolvem uma arquitetura complexa.

Um exemplo típico de uma tarefa de 5 Story Points seria a implementação de um formulário com validação básica. Isso inclui a verificação dos campos obrigatórios antes de permitir o envio, bem como o feedback ao usuário sobre erros de preenchimento. Outro exemplo seria a exibição de dados dinâmicos obtidos de uma API, como a listagem de produtos ou itens em uma página, com filtros ou ordenação simples, mas sem operações de escrita ou atualização de dados no backend.

Essas tarefas exigem uma maior atenção ao fluxo de dados e à experiência do usuário, mas ainda são de escopo limitado, sem a necessidade de gerenciar estados complexos ou realizar manipulação de dados avançada. O desenvolvimento de pequenos componentes interativos ou a integração com APIs para operações de leitura (GET), juntamente com ajustes de usabilidade e responsividade, se enquadram perfeitamente nessa categoria. As tarefas de 5 Story Points buscam melhorar a funcionalidade e a dinâmica da aplicação, sem envolver grande complexidade técnica.

Exemplos:

- Criação de um formulário com validação básica: Implementar um formulário que verifica se os campos obrigatórios foram preenchidos antes do envio.
- Implementação de uma chamada API GET para exibir dados: Consumir uma API externa ou interna para buscar e exibir uma lista de itens.
- Desenvolvimento de um componente de lista com iteração: Criar um componente que renderiza uma lista de itens a partir de um array ou objeto.
- Adição de navegação simples entre páginas: Implementar links ou botões que permitem ao usuário navegar entre diferentes rotas da aplicação.
- Implementação de um modal com funcionalidade de abrir e fechar: Criar um modal que pode ser aberto e fechado pelo usuário, exibindo conteúdo dinâmico.
- Criação de um carrossel de imagens básico: Implementar um slider que permite ao usuário navegar através de várias imagens ou conteúdo.

- Desenvolvimento de um componente de tabela com dados dinâmicos: Criar uma tabela que exibe dados provenientes de uma fonte, sem funcionalidades avançadas como paginação.
- Implementação de validação de formato de e-mail em um campo: Adicionar validação para verificar se um e-mail inserido está no formato correto.
- Adição de mensagens de erro para validações simples: Exibir mensagens ao usuário quando campos obrigatórios não são preenchidos corretamente.
- Criação de um menu dropdown com opções estáticas: Implementar um menu suspenso que permite ao usuário selecionar uma opção, atualizando o estado interno.
- Desenvolvimento de um sistema de abas (tabs) básico: Criar um componente que permite alternar entre diferentes seções de conteúdo clicando em abas.
- Implementação de um botão de toggle para mostrar/esconder conteúdo: Adicionar funcionalidade que permite ao usuário mostrar ou esconder uma seção de conteúdo.
- Criação de um componente de acordo (accordion): Implementar um elemento que expande e recolhe seções de conteúdo ao ser clicado.
- Adição de animações simples com CSS: Aplicar transições ou animações básicas a elementos para melhorar a experiência do usuário.
- Implementação de uma barra de progresso estática: Criar uma barra que indica o progresso de uma ação, sem lógica para atualização dinâmica.
- Desenvolvimento de um componente de cartão (card) com conteúdo dinâmico: Criar cartões que exibem informações como título, imagem e descrição a partir de dados fornecidos.
- Criação de um campo de busca que filtra itens localmente: Implementar uma caixa de busca que filtra uma lista de itens já carregados na página.
- Adição de um mapa estático utilizando uma API de mapas: Exibir um mapa fixo em uma localização específica sem interações avançadas.
- Implementação de um seletor de data simples: Adicionar um campo que permite ao usuário selecionar uma data, utilizando um componente de calendário básico.
- Desenvolvimento de um componente de alerta que pode ser fechado: Criar um alerta que exibe uma mensagem ao usuário e pode ser fechado ao clicar em um botão.
- Criação de um slider de valor numérico: Implementar um controle que permite ao usuário selecionar um valor dentro de um intervalo arrastando um seletor.

- Adição de suporte a temas claro e escuro: Implementar a alternância entre dois temas predefinidos, ajustando cores e estilos básicos.
- Implementação de um contador simples com incremento e decremento: Criar um componente que permite aumentar ou diminuir um valor numérico ao clicar em botões.
- Desenvolvimento de um componente de classificação por estrelas: Permitir que o usuário selecione uma avaliação de 1 a 5 estrelas, atualizando o estado interno.
- Criação de um componente de breadcrumb (trilha de navegação): Implementar uma trilha que mostra ao usuário o caminho de navegação atual na aplicação.
- Adição de validação de senha forte: Verificar se a senha inserida atende a critérios básicos, como tamanho mínimo e presença de números.
- Implementação de uma galeria de imagens com visualização ampliada: Permitir que o usuário clique em uma imagem em miniatura para vê-la em tamanho maior.
- Desenvolvimento de um componente de lista suspensa com busca: Criar um dropdown que permite ao usuário pesquisar opções dentro da lista.
- Criação de um botão que compartilha conteúdo em redes sociais: Implementar botões que abrem janelas de compartilhamento para Facebook, Twitter etc.
- Adição de suporte a várias línguas com textos estáticos: Implementar internacionalização básica, permitindo alternar textos entre dois idiomas.
- Implementação de um leitor de código QR simples: Integrar uma biblioteca que permite ao usuário escanear códigos QR utilizando a câmera.
- Desenvolvimento de uma animação de carregamento enquanto dados são obtidos: Exibir um spinner ou indicador enquanto uma requisição de dados está em andamento.
- Criação de um componente de tooltip avançado com posicionamento: Implementar tooltips que aparecem em diferentes posições baseadas na interação do usuário.
- Adição de um campo de upload de arquivos com pré-visualização de imagem: Permitir que o usuário selecione uma imagem e visualize uma miniatura antes do upload.
- Implementação de notificações simples no navegador: Exibir notificações básicas dentro da aplicação em resposta a ações do usuário.
- Desenvolvimento de um formulário com máscaras em campos de entrada: Adicionar máscaras a campos como CPF, CNPJ ou telefone para orientar o usuário na digitação.

- Criação de um gráfico básico utilizando uma biblioteca: Exibir um gráfico de barras ou linhas com dados estáticos utilizando uma biblioteca como Chart.js.
- Adição de um componente de scroll infinito: Implementar carregamento adicional de conteúdo conforme o usuário rola a página para baixo.
- Implementação de uma funcionalidade de favoritos: Permitir que o usuário marque itens como favoritos, armazenando o estado localmente.
- Desenvolvimento de um sistema de votação simples: Criar botões de "like" ou "dislike" que atualizam contadores de forma local.
- Criação de um componente de pergunta frequente (FAQ) interativo: Implementar uma lista de perguntas que ao serem clicadas exibem ou escondem as respostas.
- Adição de um sistema de classificação de produtos: Permitir que usuários atribuam notas a produtos, atualizando o estado interno.
- Implementação de filtros básicos em uma lista de itens: Adicionar botões ou checkboxes que filtram itens exibidos com base em categorias.
- Desenvolvimento de um componente de upload de múltiplos arquivos: Permitir que o usuário selecione vários arquivos para upload, sem lógica de envio ao servidor.
- Criação de um componente de cronômetro simples: Implementar um cronômetro que inicia, pausa e reinicia, sem necessidade de precisão alta.
- Adição de um botão de copiar para a área de transferência: Implementar um botão que copia um texto específico quando clicado.
- Implementação de uma lista de tarefas com adição e remoção: Criar uma lista onde o usuário pode adicionar e remover itens, mantendo o estado localmente.
- Desenvolvimento de um componente de legenda para imagens: Adicionar legendas que aparecem sobre ou abaixo das imagens exibidas.
- Criação de uma página de erro 404 personalizada: Desenvolver uma página que informa ao usuário que a página não foi encontrada, com design básico.
- Adição de um componente de confirmação antes de ações críticas: Exibir um diálogo de confirmação quando o usuário tenta realizar uma ação importante.
- Implementação de um menu hambúrguer simples para navegação móvel: Criar um menu que expande e colapsa ao ser clicado, adequado para dispositivos móveis.

13 (treze) Story Point

Tarefas de 13 Story Points no desenvolvimento de Frontend envolvem uma complexidade moderada, exigindo uma combinação de manipulação de estado, integração com APIs, e desenvolvimento de componentes reutilizáveis e interativos. Essas tarefas geralmente requerem mais planejamento e tempo de implementação do que tarefas simples, pois lidam com múltiplas camadas de lógica e demandam maior atenção ao design de interação e à manipulação de dados dinâmicos.

Essas tarefas frequentemente incluem a manipulação de estado local ou global por meio de ferramentas como Redux ou Context API, além de integrações com APIs para operações de leitura e escrita, como GET e POST. Um exemplo seria o desenvolvimento de um formulário completo que valida múltiplos campos e envia dados a uma API, exibindo mensagens de erro personalizadas com base nas respostas do backend. Essas tarefas também podem envolver a implementação de sistemas de roteamento completos, onde o usuário navega entre diferentes páginas ou vistas da aplicação, incluindo rotas protegidas que exigem autenticação.

Outro exemplo de uma tarefa de 13 Story Points é a criação de componentes reutilizáveis com propriedades configuráveis, permitindo maior flexibilidade e reuso em diferentes partes da aplicação. Isso pode incluir um componente de lista paginada, que carrega dados conforme o usuário navega entre páginas e permite ordenação e filtragem avançadas. Essas tarefas exigem um maior nível de controle sobre o estado da aplicação e a interação com o backend, garantindo que os dados sejam geridos de forma eficiente e que a interface responda dinamicamente.

Essas tarefas frequentemente incluem validações avançadas em formulários, integrações com bibliotecas externas para funcionalidades especializadas (como gráficos interativos), e a implementação de animações e transições que melhoram a experiência do usuário. Em resumo, tarefas de 13 Story Points no Frontend equilibram interatividade, integração e complexidade técnica, exigindo que o desenvolvedor domine tanto a lógica de negócios quanto a usabilidade e a performance.

Exemplos:

- Criação de um formulário completo com validação e envio de dados: Implementar um formulário que valida múltiplos campos, exibe mensagens de erro específicas e envia os dados para uma API via POST.
- Implementação de autenticação básica no frontend: Desenvolver funcionalidades que permitem ao usuário fazer login, armazenando o token de autenticação e ajustando a interface com base no estado de login.
- Desenvolvimento de um componente de lista com paginação: Criar uma lista que exibe itens paginados, permitindo ao usuário navegar entre as páginas e atualizando os dados conforme necessário.
- Criação de um sistema de roteamento completo: Implementar navegação entre múltiplas páginas ou vistas, incluindo rotas protegidas que requerem autenticação.
- Implementação de um gerenciador de estado global: Utilizar Redux ou Context API para gerenciar o estado compartilhado entre diferentes componentes da aplicação.
- Desenvolvimento de um componente de upload de arquivos com envio para o servidor: Permitir que o usuário selecione arquivos e envie-os para uma API, incluindo exibição de progresso e tratamento de erros.
- Criação de um componente de gráfico interativo: Integrar uma biblioteca de gráficos (como Chart.js ou D3.js) para exibir dados dinâmicos com interações básicas.
- Implementação de um sistema de busca com autocomplete: Desenvolver um campo de busca que sugere resultados conforme o usuário digita, consultando uma API para obter as sugestões.
- Desenvolvimento de um componente de mapa interativo: Integrar uma API de mapas (como Google Maps ou Leaflet) para exibir mapas com marcadores dinâmicos baseados em dados.
- Criação de um carrossel de imagens avançado: Implementar um slider que permite transições automáticas, navegação por pontos e suporte a gestos em dispositivos móveis.
- Implementação de um sistema de notificações in-app: Exibir notificações temporárias dentro da aplicação em resposta a ações do usuário ou eventos do sistema.
- Desenvolvimento de um componente de seleção múltipla com tags: Permitir que o usuário selecione múltiplas opções de uma lista, exibindo-as como tags que podem ser removidas.
- Criação de um sistema de filtros avançados em uma lista: Implementar múltiplos critérios de filtro que podem ser combinados para refinar os resultados exibidos.

- Implementação de validação de formulários com biblioteca externa: Utilizar uma biblioteca como Formik ou Yup para gerenciar validações complexas em formulários.
- Desenvolvimento de um editor de texto rico (rich text editor): Integrar um componente que permite ao usuário formatar texto com opções como negrito, itálico, listas e links.
- Criação de um componente de calendário interativo: Implementar um calendário que exibe eventos dinâmicos e permite ao usuário navegar entre meses e anos.
- Implementação de internacionalização (i18n) com suporte a múltiplos idiomas: Configurar a aplicação para suportar vários idiomas, permitindo a troca dinâmica do idioma exibido.
- Desenvolvimento de um componente de chat básico: Criar uma interface de chat que exibe mensagens e permite ao usuário enviar novas mensagens, utilizando WebSockets ou polling.
- Criação de um componente de árvore (tree view) para exibir dados hierárquicos: Implementar uma estrutura que permite expandir e recolher nós para navegar por dados em forma de árvore.
- Implementação de autenticação social no frontend: Permitir que usuários façam login utilizando contas de terceiros como Google ou Facebook, integrando com as APIs correspondentes.
- Desenvolvimento de um sistema de arrastar e soltar (drag and drop): Implementar funcionalidades que permitem ao usuário reorganizar itens em uma lista através de arrastar e soltar.
- Criação de um componente de tabelas com ordenação e filtragem: Desenvolver uma tabela que permite ordenar colunas e filtrar resultados baseados em critérios específicos.
- Implementação de um modo escuro (dark mode) com persistência: Permitir que o usuário alterne entre temas claro e escuro, salvando a preferência para futuras visitas.
- Desenvolvimento de um sistema de permissões no frontend: Ajustar a interface e a disponibilidade de funcionalidades com base nos papéis ou permissões do usuário.
- Criação de um componente de carrinho de compras: Implementar um carrinho que permite adicionar, remover e atualizar itens, calculando o total e impostos.
- Implementação de um sistema de rolagem virtualizada para listas grandes: Otimizar a exibição de lista extensas carregando apenas os itens visíveis na tela.

- Desenvolvimento de um componente de upload com pré-visualização e remoção: Permitir que o usuário visualize e remova arquivos selecionados antes do envio ao servidor.
- Criação de um sistema de cache de dados no frontend: Armazenar dados obtidos de APIs localmente para reduzir requisições e melhorar a performance.
- Implementação de gráficos interativos com zoom e pan: Desenvolver gráficos que permitem ao usuário ampliar e navegar pelos dados exibidos.
- Desenvolvimento de um componente de áudio ou vídeo player personalizado: Criar um player com controles personalizados para reprodução de mídia.
- Criação de um formulário de múltiplas etapas (wizard): Implementar um formulário dividido em várias etapas, permitindo navegação entre elas e validação de cada passo.
- Implementação de testes unitários para componentes: Escrever testes utilizando frameworks como Jest ou Mocha para garantir o funcionamento correto dos componentes.
- Desenvolvimento de um componente de mapa de calor (heatmap): Exibir dados em um mapa utilizando cores para representar intensidade ou frequência.
- Criação de um sistema de monitoramento de performance no frontend: Implementar ferramentas para rastrear e reportar métricas de desempenho da aplicação.
- Implementação de autenticação de dois fatores no frontend: Adicionar uma etapa extra de verificação após o login, como inserção de código enviado por e-mail ou SMS.
- Desenvolvimento de um componente de tabela com edição inline: Permitir que o usuário edite dados diretamente nas células da tabela, com validação e salvamento.
- Criação de um sistema de favoritos persistente: Permitir que o usuário marque itens como favoritos, armazenando essa informação localmente ou em uma API.
- Implementação de um componente de seleção de horário: Desenvolver um seletor que permite ao usuário escolher um horário específico, com validações.
- Desenvolvimento de animações avançadas com bibliotecas como GSAP ou Anime.js: Implementar animações que melhoram a experiência do usuário, como transições entre páginas.

- Criação de um sistema de notificações push utilizando Service Workers: Permitir que a aplicação receba notificações mesmo quando não estiver aberta no navegador.
- Implementação de um componente de carregamento progressivo de imagens: Carregar versões de baixa resolução das imagens e substituí-las por versões de alta resolução quando disponíveis.
- Desenvolvimento de um sistema de agendamento de eventos: Permitir que o usuário crie, edite e visualize eventos em um calendário, interagindo com uma API.
- Criação de um componente de feedback do usuário: Implementar uma funcionalidade que permite ao usuário enviar feedback ou reportar problemas, com envio para uma API.
- Implementação de otimização de recursos utilizando lazy loading: Carregar componentes ou imagens somente quando necessários, melhorando o tempo de carregamento inicial.
- Desenvolvimento de um componente de gráficos em tempo real: Exibir dados que são atualizados em tempo real, como métricas ou estatísticas.
- Criação de um sistema de comentários com threads: Permitir que usuários adicionem comentários e respondam a outros comentários, exibindo a hierarquia.
- Implementação de suporte a acessibilidade avançada: Garantir que a aplicação seja utilizável por pessoas com deficiências, seguindo as diretrizes WCAG.
- Desenvolvimento de um componente de rotação automática de banners: Implementar um carrossel que alterna automaticamente entre diferentes banners promocionais.
- Criação de um sistema de registro e login com confirmação de e-mail: Implementar fluxos que requerem que o usuário confirme seu endereço de e-mail antes de acessar a aplicação.
- Implementação de um componente de seleção de arquivos com arrastar e soltar: Permitir que o usuário arraste arquivos para uma área específica para selecioná-los para upload.
- Desenvolvimento de um sistema de votação com atualização dinâmica: Permitir que usuários votem em itens e vejam os resultados atualizados imediatamente.
- Criação de um componente de timeline interativa: Exibir eventos ao longo de uma linha do tempo, permitindo ao usuário explorar diferentes períodos.

34 (trinta e quatro) Story Point

Tarefas de 34 Story Points no desenvolvimento de Frontend envolvem um nível significativo de complexidade e exigem o uso de lógica de negócios avançada, manipulação de estado complexa, e integração completa com APIs externas para operações CRUD (Create, Read, Update, Delete). Estas tarefas geralmente demandam um esforço de coordenação entre diferentes partes da aplicação e podem impactar a experiência do usuário de forma mais profunda. Além disso, elas requerem um planejamento mais detalhado e testes abrangentes para garantir que a funcionalidade seja robusta e eficiente.

Exemplos típicos incluem a implementação de sistemas de upload de arquivos com pré-visualização e acompanhamento do progresso de envio. Neste tipo de tarefa, o desenvolvedor precisa criar uma interface que permite ao usuário selecionar arquivos, visualizar uma prévia (como imagens ou PDFs) e acompanhar o progresso do upload, incluindo a gestão de erros de envio e a possibilidade de cancelamento. Esse tipo de tarefa exige uma coordenação entre o frontend e o backend, além de uma interface que responda dinamicamente às interações do usuário.

Outro exemplo de tarefa de 34 Story Points seria o desenvolvimento de um dashboard interativo, onde o usuário pode visualizar gráficos e tabelas que refletem dados dinâmicos, permitindo a aplicação de filtros e ordenação em tempo real. Esses dashboards geralmente integram dados de várias fontes, exigem gerenciamento avançado de estado e interações contínuas com APIs. Também envolvem otimizações de performance para garantir que a interface permaneça fluida, mesmo com grandes volumes de dados.

Essas tarefas, por envolverem múltiplas camadas de interação, requerem não só a integração com o backend, mas também uma manipulação eficiente de estados globais, utilizando ferramentas como Redux ou Context API para gerenciar a complexidade do frontend. Tarefas de 34 Story Points são tipicamente voltadas para funcionalidades que oferecem alta interatividade e demandam coordenação entre várias partes do sistema.

Exemplos:

- Criação de um formulário multi-etapas (wizard) com validação complexa: Criar um formulário dividido em várias etapas, onde cada etapa tem

validações específicas, com possibilidade de navegação entre etapas, persistência de dados e envio final consolidado.

- Implementação de um sistema de upload de arquivos com pré-visualização e progressão de upload: Permitir que o usuário faça upload de arquivos, exibindo pré-visualizações (por exemplo, imagens ou PDFs), mostrando a progressão do upload e lidando com cancelamentos e erros.
- Criação de um componente de mapa interativo com múltiplos marcadores e filtros: Integrar um mapa que exibe múltiplos pontos de interesse, permitindo ao usuário aplicar filtros, visualizar detalhes ao clicar em um marcador e traçar rotas simples.
- Desenvolvimento de um dashboard com gráficos e tabelas interativos (Possivelmente usando Power BI): Criar um painel que exibe dados em gráficos e tabelas, permitindo que o usuário interaja com os elementos, aplique filtros, ordene dados e atualize as visualizações dinamicamente.
- Implementação de um sistema de notificação in-app com histórico e gerenciamento: Desenvolver notificações dentro da aplicação, permitindo que o usuário visualize um histórico de notificações, marque como lidas ou exclua, e receba novas notificações em tempo real.
- Criação de um componente de calendário com seleção de datas e eventos simples: Implementar um calendário que permite ao usuário selecionar datas, visualizar eventos associados e navegar entre meses e anos.
- Desenvolvimento de uma funcionalidade de arrastar e soltar (drag and drop) para reordenar itens em uma lista: Permitir que o usuário reordene itens em uma lista através de arrastar e soltar, atualizando a ordem no estado da aplicação.
- Implementação de um sistema de favoritos com persistência e sincronização: Permitir que o usuário marque itens como favoritos, com armazenamento local e sincronização com o servidor, refletindo as mudanças em diferentes dispositivos.
- Criação de um componente de tabela com paginação, ordenação e filtros avançados: Desenvolver uma tabela que suporta paginação, permite ao usuário ordenar por diferentes colunas e aplicar filtros avançados, incluindo busca por texto e filtros por categoria.
- Desenvolvimento de um sistema de autenticação com login social: Implementar fluxo de login que permite ao usuário autenticar-se usando contas de redes sociais como Google ou Facebook, incluindo tratamento de tokens e gerenciamento de sessão.
- Implementação de um sistema de chat básico entre usuários: Criar uma interface de chat que permite a troca de mensagens em tempo real entre

usuários, exibindo histórico de conversas e notificações de novas mensagens.

- Criação de um componente de gráfico personalizado utilizando uma biblioteca externa: Integrar uma biblioteca de gráficos para criar visualizações personalizadas de dados, incluindo interações como hover para exibir detalhes e atualização dinâmica dos dados.
- Desenvolvimento de um sistema de busca com autocomplete e sugestões: Implementar uma barra de busca que oferece sugestões em tempo real enquanto o usuário digita, incluindo consultas a uma API e tratamento de resultados.
- Implementação de um sistema de internacionalização (i18n) com suporte a múltiplos idiomas: Adaptar a aplicação para suportar múltiplos idiomas, permitindo alternância dinâmica e tradução de conteúdo estático e dinâmico.
- Criação de um componente de carrossel de imagens com animações personalizadas: Desenvolver um carrossel que exibe imagens ou conteúdo, com transições animadas personalizadas, controle manual e automático, e responsividade.
- Desenvolvimento de um sistema de gerenciamento de perfil de usuário com upload de avatar e edição de informações: Permitir que o usuário visualize e edite suas informações de perfil, incluindo upload de imagem de avatar, validações e salvamento de dados.
- Implementação de um sistema de permissões no frontend para controlar acesso a componentes: Adaptar a interface para exibir ou ocultar componentes e funcionalidades com base nas permissões do usuário, atualizando dinamicamente conforme o contexto.
- Criação de um componente de agenda com criação e edição de compromissos: Implementar uma agenda que permite ao usuário criar, visualizar e editar compromissos, com integração ao backend para persistência.
- Desenvolvimento de um sistema de notificações por e-mail com configuração no frontend: Permitir que o usuário configure preferências de notificações por e-mail, integrando com o backend para atualizar as configurações.
- Implementação de testes unitários abrangentes para componentes críticos: Escrever testes que cobrem os principais cenários de uso dos componentes, garantindo estabilidade e facilitando futuras manutenções.

89 (oitenta e nove) Story Point

Tarefas de 89 Story Points no Frontend envolvem o mais alto nível de complexidade dentro desse macro assunto. Essas tarefas normalmente exigem a criação de funcionalidades completas e altamente interativas que afetam significativamente a experiência do usuário. Elas incluem a implementação de lógica avançada de interface, manipulação complexa de estados, e integração com múltiplos sistemas externos. No entanto, é importante destacar que, mesmo com essa complexidade, o foco das tarefas de frontend está na interface e na interação do usuário, enquanto questões mais profundas de lógica de negócios, infraestrutura ou segurança são tratadas em outros macros assuntos.

Por exemplo, a implementação de um sistema de gerenciamento de conteúdo (CMS) no frontend, que permite aos usuários criar, editar e publicar conteúdo de forma dinâmica, é uma tarefa de 89 Story Points. Essa funcionalidade envolve a criação de múltiplas interfaces, o gerenciamento de permissões de usuários e a integração com APIs de backend para sincronizar dados. No entanto, o foco aqui está na criação de uma interface interativa e eficiente, com uma experiência fluida e bem adaptada ao usuário final, deixando a lógica de gerenciamento de dados e segurança para o backend e o sistema de APIs.

Outro exemplo seria o desenvolvimento de um editor de texto rico (WYSIWYG) altamente personalizável, que permite aos usuários formatar texto, inserir imagens, e adicionar elementos dinâmicos, tudo isso integrado a um sistema de versionamento e publicação. Essa tarefa exige não só a criação de uma interface robusta, mas também a integração de plugins e ferramentas que melhorem a usabilidade e permitam uma experiência de edição avançada. Novamente, o backend lidaria com o armazenamento e validação de conteúdo, enquanto o frontend se concentra em oferecer uma experiência rica e interativa.

Por fim, essas tarefas também podem incluir a implementação de sistemas de dashboard altamente interativos, com gráficos e tabelas dinâmicas que se atualizam em tempo real, permitindo que os usuários visualizem e manipulem grandes volumes de dados. A complexidade aqui está em garantir uma experiência de usuário fluida, mesmo com dados dinâmicos sendo carregados continuamente. No entanto, as otimizações de performance de backend, como o uso de cache, balanceamento de carga e escalabilidade, são responsabilidade de outros macros assuntos.

Tarefas de 89 Story Points no Frontend envolvem o desenvolvimento de interfaces complexas e interativas que demandam um nível elevado de planejamento e

execução, mas sem adentrar os domínios da lógica de negócios do backend, segurança ou otimização de infraestrutura.

Exemplos:

- Desenvolvimento de um sistema de gerenciamento de conteúdo (CMS) personalizado no frontend: Implementar uma interface completa que permite criação, edição, pré-visualização e publicação de conteúdo, com suporte a formatação avançada, gestão de mídia (imagens, vídeos), agendamento de publicações e histórico de versões.
- Implementação de um editor de texto rico (WYSIWYG) customizado com funcionalidades avançadas: Criar um editor de texto que suporta formatação complexa, inserção de tabelas, imagens, vídeos, links, emojis, e que permite extensão através de plugins ou módulos.
- Desenvolvimento de uma aplicação SPA (Single Page Application) complexa com roteamento dinâmico e lazy loading: Implementar uma aplicação que carrega módulos sob demanda, gerencia rotas dinâmicas, inclui proteção de rotas, e otimiza o desempenho com code splitting.
- Criação de um sistema de dashboards personalizados pelo usuário: Permitir que os usuários construam seus próprios dashboards, selecionando e configurando widgets, arrastando e soltando componentes, salvando layouts e compartilhando com outros usuários.
- Implementação de um sistema de autenticação multifator no frontend: Desenvolver fluxos de login que incluem senhas, tokens temporários (por e-mail ou SMS), integração com aplicativos autenticadores, e gerenciamento de sessões seguras, incluindo renovação e expiração de tokens.
- Desenvolvimento de um sistema de chat em tempo real com suporte a grupos e mídia: Implementar um chat que suporta conversas individuais e em grupo, envio de arquivos, imagens, vídeos, emojis, status de entrega e leitura, e que funciona em tempo real utilizando WebSockets.
- Criação de um sistema de agendamento com calendário complexo: Implementar um calendário que permite agendamento de eventos recorrentes, gestão de recursos (salas, equipamentos), fusos horários, convites, notificações e integração com calendários externos (Google Calendar, Outlook).
- Implementação de uma aplicação com visualização de dados geoespaciais avançados: Desenvolver um sistema que exibe dados em mapas, com suporte a camadas personalizadas, clusters, heatmaps, traçado de rotas, geofencing e interações avançadas com os dados.

- Desenvolvimento de uma interface de e-commerce completa com carrinho, checkout e integração de pagamentos: Implementar todas as etapas de um processo de compra, incluindo busca de produtos, filtros avançados, carrinho persistente, cálculo de frete, aplicação de cupons, integração com APIs de pagamento e confirmação de pedido.
- Criação de um sistema de gerenciamento de permissões e acesso no frontend: Desenvolver uma interface que permite a administradores configurar permissões granulares, atribuir papéis, gerenciar usuários e ajustar dinamicamente a exibição de componentes e funcionalidades com base nessas permissões.
- Implementação de um sistema de notificações push com Service Workers e suporte offline: Permitir que a aplicação receba notificações mesmo quando não está ativa, implementando Service Workers, gerenciando o consentimento do usuário e garantindo funcionamento offline com sincronização de dados.
- Desenvolvimento de um sistema de edição colaborativa em tempo real: Implementar funcionalidades que permitem a múltiplos usuários editar simultaneamente documentos ou dados, com atualizações em tempo real, indicadores de presença, controle de versões e resolução de conflitos.
- Criação de uma aplicação de visualização de dados complexos com gráficos customizados: Utilizar bibliotecas como D3.js para criar visualizações de dados interativas e customizadas, incluindo gráficos não convencionais, animações complexas e suporte a grandes volumes de dados.
- Implementação de um sistema de relatórios avançados com exportação em múltiplos formatos: Permitir que usuários gerem relatórios personalizados, selecionando dados, aplicando filtros complexos e exportando em formatos como PDF, Excel, CSV, com layouts customizados.
- Desenvolvimento de uma aplicação móvel híbrida ou PWA com funcionalidades avançadas: Criar uma aplicação que funciona como PWA ou usando frameworks como React Native ou Flutter, incluindo integração com recursos do dispositivo (GPS, câmera, notificações), suporte offline e otimizações para desempenho e usabilidade móvel.
- Criação de um sistema de gerenciamento de estado complexo com otimizações de performance: Gerenciar estados altamente aninhados ou grandes volumes de dados, implementando técnicas como memoization, selectors, normalização de dados e evitando re-renderizações desnecessárias.
- Implementação de um sistema de roteamento complexo com micro-frontends: Dividir a aplicação em micro-frontends, gerenciando rotas entre

diferentes módulos ou aplicações, garantindo integração fluida e experiência consistente para o usuário.

- Desenvolvimento de um sistema de configuração dinâmica da interface pelo usuário: Permitir que usuários personalizem a interface, ajustando layouts, temas, preferências e que essas configurações sejam persistidas e aplicadas em diferentes dispositivos.
- Criação de um componente de tabela com virtualização e performance otimizada: Implementar uma tabela que pode exibir milhares de linhas com performance otimizada, utilizando técnicas de virtualização, carregamento sob demanda e mantendo funcionalidades como ordenação e filtros.
- Implementação de acessibilidade avançada seguindo padrões WCAG: Adaptar a aplicação para ser totalmente acessível, incluindo navegação por teclado, suporte a leitores de tela, contrastes adequados, legendas em vídeos e testando com ferramentas de validação de acessibilidade.
- Desenvolvimento de um sistema de gamificação completo: Implementar elementos de gamificação como níveis, pontos, badges, rankings, desafios, integrando com o perfil do usuário e interagindo com diversas partes da aplicação.
- Criação de um sistema de testes automatizados abrangentes (unitários, integração, end-to-end): Desenvolver uma suíte completa de testes, cobrindo componentes críticos, fluxos de usuário, integração com o backend, utilizando ferramentas como Jest, Enzyme, Cypress.
- Implementação de otimizações de performance avançadas (caching, code splitting, prefetching): Melhorar o desempenho da aplicação através de técnicas como cache de recursos, code splitting avançado, prefetching de dados e recursos, e otimizações para primeira renderização (First Paint).
- Desenvolvimento de um sistema de integração contínua (CI) e entrega contínua (CD) para o frontend: Configurar pipelines que automatizam builds, testes, linting e deploys para ambientes de desenvolvimento, homologação e produção, integrando com sistemas de versionamento e gerenciamento de dependências.
- Criação de um sistema de monitoramento e logging no frontend com análise de performance: Integrar ferramentas que coletam logs, erros, métricas de uso, traçando o comportamento do usuário, identificando gargalos de performance e reportando para serviços de análise.

Desenvolvimento de Funcionalidades de Backend

O Desenvolvimento de Funcionalidades de Backend se concentra na criação da lógica de negócios e na manipulação de dados que sustentam a aplicação. O backend é responsável por processar, armazenar e gerenciar dados, além de fornecer APIs e serviços que conectam o frontend e outras partes do sistema. Diferente do frontend, que lida diretamente com a interface do usuário, o backend opera nos bastidores, garantindo que os dados sejam processados e fornecidos de forma eficiente e segura.

Lida com tarefas que envolvem autenticação, autorização e segurança de dados, o foco principal desse macro assunto está na lógica de negócios e no gerenciamento de dados. Questões mais profundas de segurança, como autenticação multifator, prevenção de ataques e criptografia de dados sensíveis, são abordadas no macro assunto de Segurança, garantindo que a aplicação seja protegida em um nível mais profundo. No backend, a segurança é implementada no contexto de garantir que os dados e as permissões sejam geridos corretamente.

O backend também é central na integração de APIs e na criação de endpoints que servem ao frontend ou a outros serviços. No entanto, o macro assunto de Integração de APIs e Serviços Externos é o mais apropriado para tarefas específicas de integração com sistemas externos complexos, autenticação de APIs com protocolos como OAuth, e manipulação de dados entre diferentes sistemas. Já o backend se concentra mais na construção de APIs internas, controle de fluxo de dados e na lógica de processamento.

Em termos de performance e escalabilidade, o backend desempenha um papel fundamental na otimização de bancos de dados, gerenciamento de cache e implementação de filas de processamento. No entanto, otimizações mais profundas na infraestrutura da aplicação, como balanceamento de carga, escalabilidade automática e ajustes avançados de desempenho, são tratadas no macro assunto de Otimização de Performance e Escalabilidade. O backend foca na eficiência do processamento de dados e na entrega de respostas rápidas, garantindo que as funcionalidades da aplicação sejam executadas de forma confiável.

1 (um) Story Point

Tarefas de 1 Story Point no desenvolvimento de Backend são caracterizadas por sua simplicidade e baixa complexidade. Elas envolvem a criação de

funcionalidades básicas que não demandam lógica de negócios avançada, interações com bancos de dados complexas ou integrações com APIs externas. Essas tarefas são rápidas de implementar e focam em pequenas adições ou ajustes que têm impacto limitado no sistema como um todo.

Essas tarefas frequentemente incluem a implementação de funções ou métodos utilitários simples, como funções de conversão de unidades ou geração de valores básicos, como UUIDs ou números aleatórios. Um exemplo típico seria a criação de uma função que converte graus Celsius em Fahrenheit ou uma que gera um número aleatório entre dois valores. Outro exemplo é a criação de endpoints básicos que retornam informações fixas ou estáticas, como um endpoint que responde com uma mensagem de "Olá mundo!" ou que retorna a data e hora atuais.

Essas tarefas podem também incluir pequenas adições de funcionalidade, como adicionar logs básicos para monitoramento de ações ou implementar midlleswares simples que adicionam headers padrão às respostas HTTP. O impacto dessas tarefas no sistema é mínimo, focando em melhorias incrementais ou ajustes que não envolvem modificações significativas na arquitetura ou na lógica de negócios.

Exemplos:

- Criação de uma função que retorna uma mensagem fixa: Implementar uma função que retorna "Olá, mundo!" quando chamada.
- Implementação de um endpoint que retorna o status do sistema: Criar um endpoint `/status` que retorna `{"status": "OK"}`.
- Adição de um método utilitário para conversão de unidades: Desenvolver uma função que converte graus Celsius em Fahrenheit.
- Criação de um endpoint que retorna a data e hora atuais: Implementar um serviço que retorna `{"datetime": "2023-10-10T10:00:00Z"}` quando acessado.
- Implementação de uma função para gerar identificadores únicos: Criar um método que retorna um UUID (Universally Unique Identifier) sem armazenamento ou validações adicionais.
- Implementação de uma função de cálculo simples: Criar uma função que calcula a soma de dois números fornecidos como parâmetros.
- Criação de um serviço que retorna uma lista estática de itens: Implementar um endpoint que retorna uma lista fixa de categorias ou produtos para testes.

- Adição de uma constante ou configuração simples ao sistema: Definir uma constante global para uso em outras partes do código, como um valor padrão.
- Implementação de um serviço que verifica se um número é par ou ímpar: Criar uma função que recebe um número e retorna {"numero": 5, "par": false}.
- Criação de um middleware que adiciona um header fixo às respostas: Implementar um middleware que adiciona X-App-Version: 1.0.0 a todas as respostas HTTP.
- Implementação de um manipulador de erros genérico: Criar uma função que captura exceções não tratadas e retorna um erro 500 com uma mensagem padrão.
- Adição de comentários e documentação em uma função existente: Documentar uma função utilizando padrões de comentários como Javadoc ou Docstrings.
- Criação de um endpoint que retorna informações do ambiente: Implementar um serviço que retorna o nome do ambiente (desenvolvimento, homologação, produção).
- Implementação de uma função de verificação de saúde do serviço: Criar um endpoint /healthcheck que retorna {"health": "up"}.
- Adição de logs simples em uma função: Inserir mensagens de log informando quando uma função é iniciada e finalizada.
- Implementação de um serviço que retorna um valor aleatório: Criar um endpoint que retorna um número aleatório entre 1 e 100.
- Criação de uma função que inverte uma string: Desenvolver uma função que recebe "abcde" e retorna "edcba".
- Implementação de uma função para calcular a idade a partir da data de nascimento: Criar uma função que recebe uma data e retorna a idade em anos.
- Criação de um endpoint que retorna informações sobre a versão da aplicação: Implementar um serviço que retorna {"version": "1.2.3", "build": "20231010"}.
- Implementação de uma função que formata uma data em um padrão específico: Converter uma data do formato "YYYY-MM-DD" para "DD/MM/YYYY".
- Adição de uma regra de roteamento simples: Configurar o servidor para redirecionar uma rota antiga para uma nova.
- Criação de um serviço que retorna o endereço IP do cliente: Implementar um endpoint que retorna {"ip": "192.168.0.1"}.

- Implementação de uma função que verifica se uma string está vazia ou nula: Retornar true se a string for vazia ou nula, false caso contrário.
- Criação de um endpoint que retorna um código de status personalizado: Implementar um serviço que retorna um código HTTP 204 (No Content) sem corpo na resposta.
- Implementação de uma função que converte texto para maiúsculas ou minúsculas: Receber uma string e retornar {"original": "texto", "maiuscula": "TEXTO"}.
- Adição de suporte a CORS simples: Configurar o servidor para permitir requisições de todos os domínios (Access-Control-Allow-Origin: *).
- Implementação de um serviço que retorna uma imagem ou arquivo estático: Servir um arquivo PDF ou imagem localizada em uma pasta específica do servidor.
- Criação de um manipulador para uma rota não encontrada (404): Retornar uma mensagem personalizada quando um endpoint não existe.
- Implementação de uma função que gera uma senha aleatória: Criar uma função que retorna uma string aleatória com letras e números de tamanho definido.
- Adição de suporte a um novo formato de data ou número: Configurar a aplicação para aceitar e formatar datas no padrão ISO 8601.
- Criação de uma função que calcula o fatorial de um número: Receber o número 5 e retornar 120.
- Implementação de uma função que valida o formato de um CPF ou CNPJ: Verificar se um CPF possui 11 dígitos e está no formato correto.
- Criação de um endpoint que simula um atraso na resposta: Implementar um serviço que espera 5 segundos antes de responder, útil para testes de timeout.
- Adição de um header de segurança simples: Configurar X-Content-Type-Options: nosniff em todas as respostas.
- Implementação de uma função que gera um código de verificação simples: Criar uma função que retorna um código numérico de 6 dígitos.
- Criação de um serviço que retorna uma lista de idiomas suportados: Retornar {"idiomas": ["pt-BR", "en-US", "es-ES"]}.
- Implementação de uma função que substitui caracteres especiais em uma string: Converter "ação" em "acao".
- Adição de um arquivo de configuração para variáveis de ambiente: Criar um arquivo .env e ler variáveis como PORT e HOST.
- Criação de um endpoint que retorna informações de cabeçalhos da requisição: Retornar todos os headers enviados pelo cliente.

- Implementação de uma função que formata números em moeda: Receber 1000 e retornar "R\$ 1.000,00".
- Criação de um serviço que verifica a disponibilidade de um nome de usuário: Retornar true se o nome não estiver em uma lista pré-definida de nomes reservados.
- Implementação de uma função que gera um hash simples: Criar um hash MD5 ou SHA1 de uma string fornecida.
- Adição de um endpoint para limpeza de cache simples (sem implementação real): Criar um endpoint /clear-cache que retorna "Cache limpo" sem efetivamente limpar o cache.
- Criação de um middleware que registra o tempo de execução de uma requisição: Logar o tempo que uma requisição leva para ser processada.
- Implementação de uma função que compara duas strings ignorando maiúsculas/minúsculas: Verificar se "Texto" é igual a "texto".
- Criação de um serviço que retorna uma resposta em um formato específico (XML): Retornar dados em XML ao invés de JSON.
- Adição de suporte a uma nova rota de API versão: Configurar o servidor para responder a /api/v2/ com um simples endpoint.
- Implementação de uma função que valida um número de telefone pelo formato: Verificar se o número está no formato "(11) 91234-5678".
- Criação de um endpoint que retorna um código de barras ou QR code estático: Servir uma imagem de QR code fixa armazenada no servidor.
- Implementação de um serviço que retorna dados de um arquivo de configuração: Ler um arquivo JSON de configurações e retornar seu conteúdo.

5 (cinco) Story Points

Tarefas de 5 Story Points no Backend envolvem um nível moderado de complexidade e demandam um pouco mais de esforço em termos de implementação e lógica do que tarefas simples. Essas atividades geralmente incluem validações básicas, interações simples com bancos de dados ou manipulação de arquivos e dados estáticos, sem grandes complicações de lógica de negócios ou integração profunda com sistemas externos.

Um exemplo típico de uma tarefa de 5 Story Points seria a criação de endpoints que manipulam dados básicos, como um serviço que recebe dados via requisição POST, faz uma verificação simples de integridade (por exemplo, se todos os campos obrigatórios estão presentes), e retorna uma resposta em formato JSON. Outro exemplo seria a implementação de uma validação básica de entradas,

garantindo que os dados enviados por um cliente sigam um formato específico antes de serem processados.

Essas tarefas também podem incluir leitura de arquivos estáticos, como a implementação de um endpoint que retorna dados de um arquivo JSON ou XML armazenado no servidor, ou ainda a configuração de logs para registrar diferentes níveis de eventos no sistema, como informações, avisos ou erros. As tarefas de 5 Story Points no backend exigem atenção à manipulação de dados e garantem o funcionamento básico do sistema, sem a necessidade de lidar com lógica avançada ou integrações complexas.

Exemplos:

- Criação de endpoints básicos para operações sem acesso a banco de dados: Implementar um endpoint que recebe dados via POST e os retorna no formato JSON invertido.
- Implementação de validações de entrada de dados simples: Criar uma função que verifica se os campos obrigatórios estão presentes em uma requisição e retorna um erro adequado se não estiverem.
- Desenvolvimento de um serviço que lê dados de um arquivo estático: Implementar um endpoint que retorna informações de um arquivo JSON ou XML armazenado no servidor.
- Criação de um endpoint que realiza cálculos básicos com validação: Desenvolver um serviço que recebe dois números e uma operação (soma, subtração) e retorna o resultado, validando a entrada para evitar erros.
- Implementação de autenticação básica com usuário e senha fixos: Criar um middleware que verifica se o usuário e senha enviados na requisição correspondem a valores pré-definidos.
- Adição de logs com níveis de severidade: Implementar logs que diferenciam entre informações, avisos e erros, permitindo melhor monitoramento.
- Criação de um endpoint que retorna dados em diferentes formatos: Desenvolver um serviço que retorna dados em JSON ou XML com base em um parâmetro da requisição.
- Implementação de um serviço que manipula dados em memória: Criar um serviço que armazena dados em uma estrutura em memória (ex.: array, lista) durante a execução da aplicação.
- Desenvolvimento de um middleware que verifica headers obrigatórios: Implementar um middleware que verifica se determinados headers estão presentes na requisição, como Content-Type ou Authorization.

- Criação de uma função que faz chamadas HTTP a um serviço externo simples: Implementar uma função que consome uma API pública sem autenticação e retorna os dados ao cliente.
- Implementação de um serviço que realiza parsing de dados: Criar uma função que recebe uma string em formato CSV e a converte em um objeto JSON.
- Adição de configuração para diferentes ambientes: Configurar a aplicação para ler variáveis específicas de acordo com o ambiente (desenvolvimento, homologação, produção).
- Criação de um endpoint que envia e-mails usando um serviço SMTP simples: Implementar um serviço que envia e-mails utilizando um servidor SMTP com configurações básicas.
- Implementação de cache simples em memória: Criar uma estrutura que armazena resultados de funções para melhorar a performance em chamadas subsequentes.
- Desenvolvimento de uma função que gera relatórios simples: Implementar uma função que compila dados estáticos e os formata em um relatório básico em texto ou HTML.
- Criação de um serviço que manipula arquivos no sistema: Implementar funções que leem, escrevem ou deletam arquivos em diretórios específicos do servidor.
- Implementação de um mecanismo de retry para chamadas externas: Criar uma função que tenta novamente uma chamada HTTP externa em caso de falha, com um número limitado de tentativas.
- Adição de suporte a variáveis de ambiente sensíveis: Configurar a aplicação para ler valores sensíveis como senhas ou tokens a partir de variáveis de ambiente.
- Criação de um endpoint que retorna informações filtradas: Implementar um serviço que retorna dados de uma lista filtrados por critérios simples fornecidos na requisição.
- Implementação de uma função que valida formatos complexos: Criar uma função que verifica se uma string está em formato de endereço IP, CEP ou outro padrão específico.
- Desenvolvimento de um middleware que limita o tamanho das requisições: Implementar um middleware que rejeita requisições com payloads maiores que um limite definido.
- Criação de um serviço que calcula estatísticas simples: Implementar uma função que recebe um conjunto de números e retorna média, mediana e moda.

- Implementação de um endpoint que suporta parâmetros de consulta (query params): Criar um serviço que processa parâmetros de URL para modificar o comportamento da resposta.
- Adição de suporte a compressão de respostas: Configurar o servidor para comprimir respostas HTTP usando gzip ou deflate para melhorar a performance.
- Criação de uma função que manipula datas e horários com fusos horários: Implementar uma função que converte datas entre diferentes fusos horários.
- Implementação de um serviço que gera tokens simples: Criar uma função que gera tokens únicos para identificação temporária, sem implementar um sistema completo de autenticação.
- Desenvolvimento de um endpoint que processa uploads de arquivos pequenos: Implementar um serviço que aceita arquivos de até um tamanho limitado e os armazena temporariamente.
- Criação de um middleware que implementa rate limiting básico: Implementar uma função que limita o número de requisições de um IP em um determinado período.
- Implementação de uma função que envia notificações simples: Criar uma função que simula o envio de notificações por e-mail ou SMS, sem integração real.
- Adição de suporte a agendamento de tarefas simples: Configurar a aplicação para executar uma função em intervalos regulares, como limpeza de logs.
- Criação de um endpoint que fornece dados para gráficos simples: Implementar um serviço que retorna dados formatados para uso em gráficos de linha ou barra no frontend.
- Implementação de um serviço que valida JWTs simples: Criar uma função que verifica a estrutura e validade de um token JWT sem verificar a assinatura.
- Desenvolvimento de um mecanismo simples de internacionalização (i18n): Implementar suporte para retornar mensagens em diferentes idiomas com base em um parâmetro.
- Criação de uma função que processa imagens básicas: Implementar uma função que redimensiona ou converte imagens utilizando bibliotecas padrão.
- Implementação de um serviço que gera códigos QR simples: Criar uma função que gera códigos QR a partir de textos fornecidos.

- Adição de um sistema de preferência do usuário em memória: Implementar um serviço que armazena e recupera preferências do usuário durante a sessão.
- Criação de um middleware que verifica o tipo de conteúdo da requisição: Implementar uma função que aceita apenas requisições com Content-Type específico, como application/json.
- Implementação de um serviço que realiza cálculos financeiros básicos: Criar funções que calculam juros simples, amortização ou outras operações financeiras básicas.
- Desenvolvimento de um endpoint que fornece sugestões de autocompletar: Implementar um serviço que retorna sugestões com base em uma lista estática de termos.
- Criação de uma função que verifica a disponibilidade de um serviço externo: Implementar uma função que realiza um ping ou chamada de teste a um serviço externo e retorna o status.
- Implementação de um serviço que converte unidades de medida: Criar funções que convertem unidades como metros para pés, quilogramas para libras etc.
- Adição de suporte a formatos de resposta personalizados: Configurar o servidor para retornar respostas em formatos como YAML ou CSV.
- Criação de um endpoint que retorna dados paginados simples: Implementar um serviço que retorna uma parte de uma lista de dados com base em parâmetros de página e tamanho.
- Implementação de uma função que gera códigos de barras simples: Criar uma função que gera imagens de códigos de barras a partir de números fornecidos.
- Desenvolvimento de um middleware que adiciona cookies na resposta: Implementar uma função que configura cookies simples na resposta HTTP.
- Criação de um serviço que realiza operações de criptografia básica: Implementar funções que criptografam e descriptografam textos usando algoritmos simples como base64.
- Implementação de um endpoint que retorna dados ordenados: Criar um serviço que retorna uma lista de dados ordenados com base em um campo específico.
- Adição de suporte a configurações de tempo limite (timeout): Configurar o servidor ou serviços para definir um tempo máximo de execução para requisições.
- Criação de uma função que realiza parsing de URLs: Implementar uma função que extrai componentes como protocolo, domínio e parâmetros de uma URL fornecida.

- Implementação de um serviço que verifica a integridade de arquivos: Criar uma função que calcula o hash de um arquivo e verifica se corresponde a um valor esperado.

13 (treze) Story Points

Tarefas de 13 Story Points no desenvolvimento de Backend envolvem uma complexidade intermediária, exigindo mais planejamento e integração com o banco de dados ou serviços externos. Essas tarefas incluem a criação de lógica de negócios básica, implementações CRUD (Create, Read, Update, Delete) com múltiplas operações, e integrações com APIs que necessitam de autenticação simples ou manipulação de dados moderadamente complexos. Além disso, essas tarefas começam a demandar validações mais detalhadas e o gerenciamento de estados mais dinâmicos.

Um exemplo comum de uma tarefa de 13 Story Points seria a implementação de operações CRUD completas para uma entidade simples, como produtos ou usuários, com o backend realizando as operações de leitura, escrita, atualização e exclusão de registros no banco de dados. Isso envolve a criação de endpoints e a lógica necessária para garantir a integridade dos dados, como validações de campos obrigatórios e o gerenciamento de erros comuns.

Essas tarefas também podem incluir a manipulação de dados mais complexos em consultas ao banco de dados, como a implementação de filtros e ordenação para a recuperação de dados com base em múltiplos parâmetros fornecidos pelo usuário. A integração com serviços externos, utilizando autenticação básica ou chaves de API, é outro exemplo. Isso poderia envolver a criação de um serviço que consome uma API externa, obtendo dados e processando-os de acordo com os requisitos da aplicação.

Tarefas de 13 Story Points no backend envolvem uma lógica de negócios mais elaborada, mas ainda dentro de uma escala gerenciável. Elas exigem o uso de transações simples para garantir a consistência em múltiplas operações de banco de dados, além de validações moderadamente complexas para garantir que os dados sejam processados corretamente antes de serem persistidos.

Exemplos:

- Implementação de operações CRUD básicas com acesso ao banco de dados: Criar endpoints para criar, ler, atualizar e deletar registros de uma tabela simples, como "produtos" ou "clientes".

- Criação de um serviço que salva dados no banco de dados com validação básica: Implementar um endpoint que recebe dados, valida campos obrigatórios e salva no banco.
- Desenvolvimento de consultas ao banco de dados com filtros simples: Criar um endpoint que retorna registros filtrados por um único critério, como "status = ativo".
- Implementação de paginação básica em listagens: Desenvolver um serviço que retorna dados paginados com parâmetros de "page" e "limit".
- Criação de um endpoint que atualiza registros específicos: Implementar um serviço que permite atualizar os dados de um registro identificado por um ID.
- Implementação de um serviço que exclui registros no banco de dados: Criar um endpoint que deleta um registro específico, garantindo a integridade referencial.
- Desenvolvimento de lógica para evitar duplicação de dados: Implementar validação que impede a inserção de registros com campos únicos já existentes, como e-mail ou CPF.
- Criação de um serviço que realiza cálculos com dados do banco: Implementar um endpoint que soma valores de uma coluna para registros que atendem a um critério.
- Implementação de autenticação básica com verificação no banco de dados: Criar um serviço de login que verifica usuário e senha armazenados no banco ou LDAP
- Desenvolvimento de endpoints que retornam dados relacionados (JOIN simples): Implementar um serviço que retorna informações de duas tabelas relacionadas, como pedidos e clientes.
- Criação de um endpoint que importa dados de um arquivo para o banco: Implementar um serviço que lê um arquivo CSV pequeno e insere os dados em uma tabela.
- Implementação de transações simples no banco de dados: Garantir que múltiplas operações de banco ocorram de forma atômica, como inserir em duas tabelas.
- Desenvolvimento de um serviço que envia e-mails com base em dados do banco: Implementar um processo que envia e-mails para usuários cadastrados.
- Criação de um endpoint que realiza upload de arquivos e salva informações no banco: Implementar um serviço que permite o upload de imagens e registra os metadados.

- Implementação de lógica de negócios simples: Criar regras que determinam o comportamento do sistema com base em valores de entrada e dados do banco.
- Desenvolvimento de um serviço que gera relatórios a partir do banco de dados: Implementar um endpoint que retorna dados agregados para relatórios simples.
- Criação de um endpoint que busca registros por múltiplos critérios: Permitir que o cliente filtre dados por vários parâmetros opcionais.
- Implementação de cache básico de consultas ao banco de dados: Armazenar resultados de consultas frequentes para melhorar a performance.
- Desenvolvimento de um serviço que envia notificações push: Integrar com um serviço de notificações para enviar mensagens a dispositivos.
- Criação de endpoints que suportam múltiplos formatos de resposta: Permitir que o cliente escolha receber dados em JSON ou XML.
- Implementação de um serviço que gera PDFs a partir de dados: Criar um processo que gera documentos PDF com informações do banco.
- Desenvolvimento de um mecanismo de recuperação de senha: Implementar endpoints que permitem ao usuário solicitar e redefinir sua senha.
- Criação de um serviço que integra com uma API externa simples: Consumir dados de uma API pública e salvar no banco.
- Implementação de um endpoint que realiza operações em lote: Permitir a inserção ou atualização de múltiplos registros em uma única requisição.
- Desenvolvimento de um sistema de logging personalizado: Implementar registros de atividades específicas no banco de dados.
- Criação de um serviço que valida e normaliza dados de entrada: Implementar lógica para limpar e padronizar dados antes de salvar.
- Implementação de um serviço que gera códigos de autenticação temporários: Criar um endpoint que gera códigos para verificação de e-mail ou telefone.
- Desenvolvimento de lógica para cálculo de frete: Implementar um serviço que calcula o custo de envio baseado em peso e destino.
- Criação de um endpoint que realiza consultas geoespaciais simples: Implementar um serviço que retorna registros próximos a uma localização.
- Implementação de restrições de acesso por papel de usuário: Criar lógica que permite ou nega acesso a funcionalidades com base no papel do usuário.

- Desenvolvimento de um serviço que realiza backup simples do banco de dados: Implementar um processo que exporta dados para um arquivo de backup.
- Criação de um endpoint que processa pagamentos fictícios: Simular um processo de pagamento para fins de teste.
- Implementação de um serviço que atualiza registros com base em tempo: Criar um processo que expira registros após um período.
- Desenvolvimento de um mecanismo de notificações por e-mail com templates: Enviar e-mails usando templates com dados personalizados.
- Criação de um serviço que gera gráficos simples: Processar dados e retornar informações para geração de gráficos no frontend.
- Implementação de um sistema de preferências do usuário armazenado no banco: Permitir que usuários salvem e recuperem suas configurações.
- Desenvolvimento de endpoints para gerenciamento de sessões: Implementar criação e destruição de sessões de usuário.
- Criação de um serviço que realiza upload e armazenamento de arquivos: Integrar com um serviço de armazenamento, como AWS S3.
- Implementação de lógica de negócios para cálculo de descontos: Aplicar regras de negócio para oferecer descontos baseados em critérios.
- Desenvolvimento de um serviço que envia SMS: Integrar com uma API de SMS para enviar mensagens.
- Criação de um endpoint que retorna dados estatísticos: Calcular e retornar estatísticas como média, máximo e mínimo de valores.
- Implementação de lógica para verificar disponibilidade de recursos: Checar se determinado recurso está disponível antes de permitir sua reserva.
- Desenvolvimento de um mecanismo de auditoria simples: Registrar alterações em registros críticos no banco de dados.
- Criação de um serviço que processa filas simples: Implementar um sistema básico de enfileiramento para tarefas.
- Implementação de um endpoint que realiza autenticação via OAuth2 simplificado: Permitir login usando um provedor OAuth2 com escopo limitado.
- Desenvolvimento de lógica para envio de mensagens em horários programados: Agendar envios de e-mails ou notificações.
- Criação de um serviço que integra com serviços de mapas: Utilizar APIs como Google Maps para obter coordenadas.
- Implementação de um mecanismo de prevenção de spam: Limitar ações repetitivas em um curto período.
- Desenvolvimento de endpoints que suportam múltiplos idiomas: Adaptar respostas com base no idioma preferido do usuário.

- Criação de um serviço que realiza assinaturas digitais simples: Implementar assinatura de dados usando chaves privadas.

34 (trinta e quatro) Story Points

Tarefas de 34 Story Points no desenvolvimento de Backend envolvem um nível elevado de complexidade, exigindo a implementação de lógica de negócios mais elaborada e a interação com múltiplos sistemas ou tabelas no banco de dados. Essas tarefas frequentemente incluem a criação de operações CRUD completas, validações complexas e integração com serviços externos, como APIs que requerem autenticação avançada. Além disso, essas funcionalidades demandam planejamento detalhado para garantir que as regras de negócio sejam aplicadas corretamente e que o sistema seja escalável e seguro.

Um exemplo típico de tarefa de 34 Story Points é a implementação de um sistema de autenticação baseado em tokens JWT (JSON Web Tokens), onde o backend é responsável por gerar, validar e renovar tokens para permitir a autenticação segura de usuários. Isso envolve lógica para gerenciar o ciclo de vida dos tokens e garantir que apenas usuários autenticados possam acessar determinados recursos, além de manipulação de dados sensíveis.

Outro exemplo seria o desenvolvimento de um sistema de envio de e-mails com templates personalizáveis. Nesse caso, o backend deve ser capaz de gerar e-mails com conteúdo dinâmico, baseado em templates específicos, como e-mails transacionais (confirmações de pedidos, notificações de falhas) e enviá-los de forma programática, integrando-se com um serviço de envio de e-mails.

Além disso, tarefas dessa magnitude podem incluir a implementação de cache para melhorar a performance de consultas pesadas ao banco de dados. Isso envolve a configuração de um sistema como Redis ou Memcached, que armazena resultados de consultas frequentemente executadas, aliviando a carga sobre o banco de dados e acelerando o tempo de resposta para os usuários. Essas tarefas são fundamentais para garantir a escalabilidade do sistema, especialmente em cenários com grande volume de acessos.

Exemplos:

- Implementação de operações CRUD com lógica de negócios e validações complexas: Desenvolver endpoints para criar, ler, atualizar e deletar

registros, incluindo validações complexas e regras de negócio específicas (ex.: verificar limites de crédito antes de aprovar um pedido).

- Desenvolvimento de um sistema de autenticação com tokens JWT: Implementar autenticação segura usando tokens JWT, incluindo geração, validação e renovação de tokens.
- Criação de um serviço de envio de e-mails com templates personalizáveis: Implementar envio de e-mails transacionais utilizando templates que permitem personalização de conteúdo para diferentes usuários.
- Implementação de integração com uma API externa com autenticação OAuth2: Consumir uma API externa que requer autenticação OAuth2, incluindo obtenção de tokens e manejo de refresh tokens.
- Desenvolvimento de um sistema de upload de arquivos com armazenamento local e controle de tamanho: Permitir que usuários façam upload de arquivos, salvando-os no servidor, com validações de tipo e tamanho, e registrando metadados no banco de dados.
- Criação de um serviço de geração de relatórios em formatos diferentes (PDF, Excel): Implementar geração de relatórios que podem ser exportados em múltiplos formatos, incluindo formatação e agregação de dados.
- Implementação de cache para melhorar a performance de consultas pesadas: Configurar e utilizar um sistema de cache (como Redis) para armazenar resultados de consultas complexas e melhorar o desempenho.
- Desenvolvimento de um sistema de recuperação de senha com tokens temporários: Implementar funcionalidade que permite aos usuários solicitar a redefinição de senha, enviando um link com token seguro por e-mail.
- Criação de um serviço de notificações em tempo real utilizando WebSockets: Implementar comunicação em tempo real para enviar notificações aos clientes conectados, utilizando bibliotecas como Socket.IO.
- Implementação de um sistema de permissões baseado em roles simples: Desenvolver controle de acesso onde usuários têm papéis (roles) que definem quais recursos e ações eles podem acessar.
- Desenvolvimento de um mecanismo de auditoria para registro de ações do usuário: Implementar registro detalhado de operações críticas realizadas pelos usuários, como login, alteração de dados sensíveis, exclusão de registros, incluindo timestamp e identificação do usuário.
- Criação de um serviço que processa pagamentos utilizando uma API de pagamento simples: Integrar com um gateway de pagamento (como Stripe) para processar transações básicas, incluindo tratamento de respostas e registro de transações.

- Implementação de um sistema de agendamento de tarefas (cron jobs) para tarefas simples: Configurar tarefas automatizadas que executam funções específicas em horários pré-definidos, como envio de lembretes ou limpeza de dados temporários.
- Desenvolvimento de um serviço de importação de dados a partir de arquivos CSV: Permitir que usuários façam upload de arquivos CSV e importar os dados para o banco, incluindo validação e tratamento de erros.
- Criação de endpoints com filtros avançados e ordenação dinâmica: Implementar funcionalidades que permitem aos usuários filtrar resultados por múltiplos critérios e ordenar por diferentes campos, com paginação.
- Implementação de lógica de negócios para cálculo de preços com múltiplos fatores: Desenvolver cálculos que consideram impostos, descontos, promoções e outras variáveis para determinar o preço final de um produto ou serviço.
- Desenvolvimento de um sistema de notificações por e-mail baseadas em eventos: Implementar envio automático de e-mails quando certos eventos ocorrem, como cadastro, compra realizada ou alteração de senha.
- Criação de um serviço de integração com sistema de SMS: Integrar com um provedor de SMS para enviar mensagens, incluindo gerenciamento de templates e tratamento de respostas.
- Implementação de um mecanismo de rate limiting básico: Limitar o número de requisições que um usuário ou IP pode fazer em um determinado período, para prevenir abusos.
- Desenvolvimento de um sistema de log centralizado com diferentes níveis de severidade: Implementar logs estruturados que podem ser configurados para diferentes níveis (info, warn, error) e direcionados para arquivos ou sistemas de log.
- Criação de um serviço que realiza validação de documentos como CPF ou CNPJ: Implementar funções que validam a autenticidade de documentos brasileiros, incluindo algoritmos de verificação de dígitos.
- Implementação de um sistema de upload e manipulação básica de imagens: Permitir upload de imagens, com redimensionamento ou compressão, e armazenamento otimizado.
- Desenvolvimento de endpoints que suportam internacionalização (i18n): Adaptar as respostas do servidor para diferentes idiomas, com base nas preferências do usuário ou parâmetros da requisição.
- Criação de um serviço que integra com serviços de mapas para geocodificação: Utilizar APIs como Google Maps ou OpenStreetMap para converter endereços em coordenadas geográficas e vice-versa.

- Implementação de transações no banco de dados para garantir consistência: Utilizar transações para assegurar que um conjunto de operações no banco de dados seja executado integralmente ou revertido em caso de falha.
- Desenvolvimento de um sistema de mensagens internas entre usuários: Implementar funcionalidade que permite a troca de mensagens privadas dentro da aplicação, incluindo armazenamento e listagem de conversas.
- Criação de um serviço de geração de códigos promocionais ou cupons: Desenvolver funcionalidades para criar, validar e aplicar códigos de desconto em compras ou assinaturas.
- Implementação de um sistema de feedback de usuários: Permitir que usuários enviem feedback ou avaliações, incluindo armazenamento, moderação e exibição dos comentários.
- Desenvolvimento de um mecanismo de pesquisa simples nos dados: Implementar busca por palavras-chave em campos específicos, com suporte a operadores básicos.
- Criação de um serviço que exporta dados para formatos como CSV ou Excel: Permitir que usuários exportem listas ou relatórios em formatos amplamente utilizados.
- Implementação de um sistema de versionamento simples de registros: Manter histórico de alterações em registros importantes, permitindo consulta a versões anteriores.
- Desenvolvimento de um sistema de chat em tempo real básico: Implementar comunicação em tempo real entre usuários, para suporte ou interação simples, utilizando WebSockets.
- Criação de um serviço que integra com uma API de terceiros para autenticação social: Permitir login usando contas de redes sociais como Google ou Facebook, incluindo obtenção e validação de tokens.
- Implementação de um sistema de notificações push para aplicativos móveis: Integrar com serviços como Firebase Cloud Messaging para enviar notificações a dispositivos móveis.
- Desenvolvimento de um sistema de avaliação e classificação (ratings) de itens: Permitir que usuários avaliem produtos ou serviços, calculando médias e exibindo classificações.
- Criação de um serviço que gera relatórios estatísticos básicos: Processar dados para fornecer estatísticas como total de usuários, vendas por período etc.
- Implementação de um mecanismo de segurança para proteção contra ataques comuns: Implementar medidas como proteção contra SQL Injection, XSS e CSRF, incluindo validações e sanitização de entradas.

- Desenvolvimento de um sistema de monitoramento básico de saúde da aplicação: Implementar endpoints e alertas que monitoram recursos como uso de memória, CPU e disponibilidade de serviços.
- Criação de um serviço que suporta múltiplos formatos de autenticação: Permitir autenticação via JWT, sessões de cookies ou tokens de API, conforme necessário.
- Implementação de lógica de negócios para gestão de inventário com múltiplos depósitos: Controlar estoque considerando diferentes locais de armazenamento e transferências entre depósitos.
- Desenvolvimento de um sistema de upload de documentos com reconhecimento de texto básico (OCR simples): Permitir upload de documentos e extrair texto para indexação ou processamento básico.
- Criação de um serviço que realiza integração simples com sistemas legados: Consumir dados de um sistema antigo via arquivo ou serviço simples, para integração básica.
- Implementação de um mecanismo de fila simples para processamento assíncrono: Utilizar uma fila em memória ou banco de dados para gerenciar tarefas que precisam ser processadas em segundo plano.
- Desenvolvimento de um sistema de reservas com verificação de disponibilidade: Permitir que usuários façam reservas, verificando disponibilidade e prevenindo conflitos.
- Criação de um serviço de assinaturas com renovação automática: Implementar lógica para gerenciar assinaturas pagas, incluindo renovação automática e cancelamentos.
- Implementação de um sistema de backup automático dos dados críticos: Configurar processos que realizam backups periódicos de dados importantes e armazenam em local seguro.
- Desenvolvimento de um mecanismo de importação de dados de APIs públicas: Consumir dados de APIs públicas (como dados meteorológicos) e integrar ao sistema.
- Criação de um serviço que fornece dados para dashboards personalizados: Implementar endpoints que agregam e formatam dados conforme as necessidades de visualização dos usuários.
- Implementação de um sistema de notificações via e-mail para eventos específicos: Enviar e-mails automáticos em resposta a eventos como falhas no sistema ou ações administrativas.
- Desenvolvimento de um sistema de pesquisa avançada com filtros e ordenação: Implementar funcionalidades de busca que permitem aos usuários combinar múltiplos filtros e critérios de ordenação para encontrar registros específicos.

89 (oitenta e nove) Story Point

Representam o mais alto nível de complexidade, exigindo o desenvolvimento de lógica de negócios avançada e integração profunda com múltiplos sistemas ou serviços externos. Essas tarefas normalmente envolvem manipulação de grandes volumes de dados, integração com APIs complexas que exigem autenticação avançada, e a implementação de funcionalidades que afetam diversas partes da aplicação, exigindo coordenação significativa entre sistemas.

Um exemplo comum de tarefa de 89 Story Points seria a implementação de um sistema de autenticação multifator (MFA), onde o backend gerencia a lógica de envio de tokens via SMS, e-mail ou aplicativos autenticadores (como Google Authenticator). Esse tipo de tarefa envolve a integração de serviços de terceiros, geração e validação de tokens temporários, além de garantias de segurança robustas para proteger os usuários contra fraudes ou acesso indevido.

Outro exemplo seria a criação de um sistema de cache distribuído com Redis ou Memcached, implementando uma arquitetura de cache para armazenar resultados de consultas pesadas. Isso melhora drasticamente a performance em cenários de alto volume de acessos. A configuração de clusters de cache, invalidação de dados, e estratégias de sincronização entre servidores são componentes críticos desse tipo de tarefa.

As tarefas também podem incluir a integração com múltiplas APIs externas simultaneamente, com tratamento avançado de autenticação, erros e reconciliação de dados. Isso envolve consumir e combinar dados de diversas fontes, processando-os de forma eficiente e garantindo a consistência entre os sistemas. Alterações significativas na arquitetura do sistema, como a implementação de funcionalidades que impactam a escalabilidade ou a alta disponibilidade, também entram nessa categoria de esforço, exigindo planejamento detalhado e execução cuidadosa.

Exemplos:

- **Implementação de Autenticação Multifator (MFA):** Desenvolver um sistema que adiciona uma camada extra de segurança, permitindo que os usuários autenticuem via SMS, e-mail ou aplicativos autenticadores como Google Authenticator.

- Criação de um Serviço de Geração de Relatórios Avançados: Desenvolver um módulo que permite aos usuários gerar relatórios personalizados com filtros complexos, agrupamentos, cálculos agregados e exportação em múltiplos formatos (PDF, Excel).
- Implementação de Cache Distribuído com Redis: Configurar e integrar o Redis para armazenar em cache resultados de consultas pesadas, melhorando a performance e escalabilidade do sistema.
- Desenvolvimento de um Sistema de Notificações em Tempo Real: Implementar um serviço utilizando WebSockets ou tecnologias similares para enviar notificações em tempo real aos usuários, incluindo gerenciamento de canais e eventos.
- Criação de Integração com Múltiplas APIs Externas: Desenvolver conectores para integrar o sistema com vários serviços externos, como APIs de terceiros, sistemas legados ou parceiros, incluindo tratamento de autenticação, erros e reconciliação de dados.
- Implementação de um Sistema de Agendamento Complexo: Criar funcionalidades que permitem agendar tarefas ou eventos com múltiplas restrições, como disponibilidade de recursos, fusos horários e regras de recorrência.
- Desenvolvimento de Lógica de Negócios Complexa: Implementar regras de negócio que envolvem cálculos avançados, múltiplas condições e interações com diversas entidades, como cálculo de impostos complexos ou comissões baseadas em desempenho.
- Criação de um Mecanismo de Pesquisa Avançada: Integrar um motor de busca como Elasticsearch ou Solr para permitir pesquisas full-text, filtros avançados, autocompletar e relevância nos resultados.
- Implementação de um Sistema de Backup e Recuperação Automatizado: Desenvolver processos que realizam backups automáticos do banco de dados e arquivos críticos, incluindo scripts para restauração e monitoramento dos backups.
- Desenvolvimento de um Sistema de Logs Centralizado: Implementar uma solução que consolida logs de diferentes serviços e servidores, permitindo monitoramento e análise centralizada de eventos.
- Criação de um Sistema de Fila de Processamento Assíncrono: Utilizar ferramentas como RabbitMQ ou AWS SQS para gerenciar tarefas que precisam ser processadas de forma assíncrona, incluindo reprocessamento em caso de falhas.
- Implementação de um Sistema de Autorização Avançado (RBAC): Desenvolver um controle de acesso baseado em papéis e permissões,

permitindo configurar quem pode acessar quais recursos e ações no sistema.

- Desenvolvimento de um Módulo de Internacionalização (i18n) Completo: Adaptar o sistema para suportar múltiplos idiomas, incluindo tradução de conteúdo estático e dinâmico, formatação regional de datas, números e moedas.
- Criação de um Sistema de Monitoramento e Alertas: Integrar ferramentas de monitoramento (como Prometheus e Grafana) para rastrear métricas de desempenho, disponibilidade e erros, incluindo configuração de alertas.
- Implementação de Processamento de Arquivos em Lote: Desenvolver funcionalidades que permitem o upload e processamento de grandes volumes de dados ou arquivos, incluindo validações e tratamento de erros.
- Desenvolvimento de um Sistema de Auditoria Detalhada: Implementar registro detalhado de ações dos usuários e alterações no sistema, incluindo quem fez, o que fez e quando, para fins de segurança e conformidade.
- Criação de um Mecanismo de Cache Avançado para Melhorar Desempenho: Implementar estratégias de cache em diferentes camadas (aplicação, banco de dados, CDN) para otimizar a performance do sistema.
- Implementação de Integração com Serviços de Mensageria (e.g., SMS, WhatsApp): Desenvolver integração com APIs de mensageria para envio de notificações, incluindo gerenciamento de templates e respostas.
- Desenvolvimento de Funcionalidades para Suporte Offline: Implementar mecanismos que permitem o uso do sistema mesmo sem conexão com a internet, sincronizando dados quando a conexão for restabelecida.
- Criação de um Sistema de Gerenciamento de Conteúdo (CMS) Simples: Desenvolver funcionalidades que permitem a criação, edição e publicação de conteúdo dinâmico, como blogs, notícias ou páginas institucionais.
- Implementação de Funcionalidades de Segurança Avançadas: Adicionar medidas como proteção contra-ataques DDoS, implementação de CSP (Content Security Policy), HSTS (HTTP Strict Transport Security) e validação avançada de entradas.
- Desenvolvimento de um Sistema de Assinatura Digital: Implementar funcionalidades que permitem aos usuários assinar documentos eletronicamente, incluindo integração com certificados digitais e conformidade legal.
- Criação de um Serviço de Upload e Processamento de Imagens: Desenvolver funcionalidades que permitem o upload de imagens, processamento (redimensionamento, corte, compressão) e armazenamento otimizado.

- Implementação de um Sistema de Feedback e Avaliação: Permitir que os usuários forneçam feedback ou avaliações sobre produtos, serviços ou conteúdo, incluindo moderação e análise dos dados.
- Desenvolvimento de um Mecanismo de Notificações Personalizadas: Implementar funcionalidades que permitem aos usuários configurar preferências de notificação, como canal (e-mail, SMS) e tipo de evento.
- Criação de um Sistema de Gestão de Eventos: Desenvolver funcionalidades para criação, gerenciamento e inscrição em eventos, incluindo emissão de ingressos e integração com calendários.
- Implementação de um Sistema de Versionamento de APIs: Gerenciar diferentes versões da API, garantindo compatibilidade retroativa e permitindo atualizações sem interromper clientes existentes.
- Desenvolvimento de Integração com Sistemas de BI (Business Intelligence): Implementar conectores que permitem a extração e transformação de dados para sistemas de BI, facilitando a análise e geração de insights.
- Criação de um Sistema de Gestão de Inventário Avançado: Desenvolver funcionalidades que gerenciam estoque em tempo real, alertas de reposição, movimentações entre depósitos e relatórios detalhados.
- Implementação de Pagamentos com Suporte a Múltiplas Moedas e Métodos: Desenvolver integração com gateways de pagamento que suportam diversas moedas e métodos (cartão, boleto, PIX), incluindo conversão cambial.
- Desenvolvimento de um Sistema de Pesquisa Geoespacial: Permitir que os usuários realizem buscas baseadas em localização, como encontrar itens ou serviços próximos a uma coordenada geográfica.
- Criação de um Sistema de Controle de Versão de Documentos: Implementar funcionalidades que permitem manter histórico de alterações em documentos, comparações entre versões e restauração de versões anteriores.
- Implementação de um Sistema de Recomendações Simples: Desenvolver algoritmos que sugerem produtos ou conteúdo aos usuários com base em suas interações ou preferências.
- Desenvolvimento de um Mecanismo de Rate Limiting Avançado: Implementar controle de acesso que limita requisições com base em políticas definidas, prevenindo abusos e ataques de força bruta.
- Criação de um Sistema de Configuração Dinâmica de Recursos: Permitir que administradores alterem configurações do sistema em tempo de execução, sem necessidade de reiniciar serviços ou aplicações.

- Implementação de um Sistema de Gestão de Tarefas e Workflows: Desenvolver funcionalidades que permitem definir e acompanhar processos internos, com etapas, responsáveis e notificações.
- Desenvolvimento de Integração com Serviços de Logística e Rastreamento: Implementar conectores para serviços de entrega, permitindo cálculo de frete, geração de etiquetas e rastreamento de encomendas.
- Criação de um Sistema de Suporte a Multi-Tenancy Simples: Adaptar o sistema para suportar múltiplos clientes (tenants) isolados, permitindo que cada um tenha seus próprios dados e configurações.
- Implementação de Funcionalidades de Acessibilidade Avançadas: Garantir que o sistema esteja em conformidade com normas de acessibilidade (WCAG), incluindo suporte a leitores de tela, navegação por teclado e contraste adequado.
- Desenvolvimento de um Sistema de Gestão de Preferências Complexas: Permitir que usuários personalizem extensivamente sua experiência, incluindo layouts, notificações, temas e funcionalidades.
- Criação de um Sistema de Importação e Exportação de Dados em Massa: Desenvolver funcionalidades que permitem a importação e exportação de grandes volumes de dados, com validação e tratamento de erros.
- Implementação de um Sistema de Gerenciamento de Sessões Seguras: Melhorar a segurança das sessões de usuários, incluindo detecção de hijacking, logout em múltiplos dispositivos e monitoramento de atividades suspeitas.
- Desenvolvimento de Funcionalidades para Conformidade com LGPD/GDPR: Implementar mecanismos que permitem aos usuários controlar seus dados pessoais, incluindo consentimento, anonimização e solicitações de exclusão.
- Criação de um Sistema de Análise de Dados Estatísticos: Desenvolver módulos que coletam e analisam dados de uso, fornecendo estatísticas como comportamento de usuários, tendências e desempenho do sistema.
- Implementação de Funcionalidades de Chat ou Mensagens Internas: Permitir comunicação direta entre usuários dentro do sistema, incluindo histórico de mensagens, notificações e gerenciamento de conversas.
- Desenvolvimento de um Sistema de Reconhecimento de Padrões Simples: Implementar algoritmos que detectam padrões ou anomalias nos dados, auxiliando em tomadas de decisão ou alertas.
- Criação de um Módulo de E-commerce Básico: Desenvolver funcionalidades que permitem a venda de produtos ou serviços, incluindo carrinho de compras, processamento de pedidos e integração com pagamentos.

- Implementação de um Sistema de Testes Automatizados: Desenvolver uma suíte de testes automatizados abrangendo testes unitários, de integração e funcionais, garantindo a qualidade e estabilidade do sistema.
- Desenvolvimento de um Sistema de Entrega Contínua (CD): Configurar pipelines que automatizam o deploy do sistema em ambientes de homologação e produção, incluindo etapas de aprovação e rollback.

Integração de APIs e Serviços Externos

A Integração de APIs e Serviços Externos é um macro assunto que envolve a conexão entre uma aplicação interna e sistemas, serviços ou plataformas de terceiros, com o objetivo de consumir, processar ou enviar dados. As APIs (Application Programming Interfaces) permitem que diferentes sistemas se comuniquem de forma padronizada, possibilitando que funcionalidades e dados sejam compartilhados de maneira segura e eficiente. Este macro assunto abrange desde chamadas simples a APIs públicas até integrações mais complexas que requerem autenticação e autorização avançadas.

No desenvolvimento de software, a integração com serviços externos é crucial para estender as capacidades de uma aplicação. Isso pode incluir integrações com provedores de pagamento, plataformas de autenticação (como OAuth), serviços de mensagens SMS ou e-mail, ou até mesmo APIs de dados externos para fornecer informações em tempo real, como previsões meteorológicas ou dados financeiros. A lógica de negócios no lado da aplicação deve ser capaz de consumir essas APIs, processar os dados recebidos e enviar respostas adequadas ao usuário ou a outros sistemas.

A complexidade das tarefas de integração com APIs e serviços externos pode variar bastante. As tarefas simples podem envolver chamadas GET para recuperar dados públicos sem autenticação, enquanto tarefas mais complexas podem exigir o uso de OAuth 2.0, manipulação de tokens JWT, ou a implementação de fluxos completos de autenticação e autorização. Além disso, a integração com múltiplas APIs simultaneamente, o tratamento de grandes volumes de dados e a garantia de alta disponibilidade e escalabilidade do serviço são aspectos essenciais que exigem planejamento cuidadoso e desenvolvimento avançado.

Ao estimar Story Points para tarefas nesse macro assunto, deve-se levar em conta o grau de complexidade da autenticação, a profundidade da integração (se é uma simples leitura de dados ou envolve várias chamadas de APIs com coordenação entre elas), e o impacto da tarefa na arquitetura geral da aplicação. A

implementação de serviços que dependem de comunicação constante com sistemas externos deve ser feita de maneira segura e otimizada, garantindo a integridade e a confiabilidade dos dados transferidos.

1 (um) Story Point

Tarefas de 1 Story Point em Integração de APIs e Serviços Externos são caracterizadas por sua simplicidade e baixa complexidade. Essas tarefas geralmente envolvem interações básicas com APIs ou serviços externos que não requerem autenticação complexa, manipulação avançada de dados ou configurações elaboradas. O objetivo dessas tarefas é realizar integrações rápidas e diretas, que podem ser concluídas em um curto período.

Essas tarefas podem incluir a criação de uma chamada API GET simples para obter dados públicos, como a previsão do tempo ou informações de notícias, sem a necessidade de autenticação. Outro exemplo comum é a configuração de uma chave de API em um arquivo de configuração, permitindo que a aplicação utilize o serviço externo posteriormente. Essas tarefas são importantes para garantir que a integração básica entre a aplicação e os serviços externos funcione corretamente, mas sem exigir um grande esforço técnico.

Outras atividades típicas de 1 Story Point incluem a verificação de conectividade com um serviço externo, realizando uma requisição simples para garantir que o serviço está acessível e funcionando como esperado. Além disso, a adição de parâmetros simples de query ou a configuração básica de CORS para permitir chamadas a um serviço externo são tarefas que se enquadram nesse nível de complexidade. O foco dessas tarefas é garantir a conectividade básica e a comunicação entre sistemas, sem a necessidade de lógica de negócios avançada.

Exemplos:

- Criação de uma chamada API GET simples para obter dados públicos: Consumir uma API pública que não requer autenticação para exibir informações básicas, como a previsão do tempo ou notícias.
- Configuração de uma chave de API em arquivo de configuração: Inserir uma chave de API fornecida por um serviço externo em um arquivo de configuração para uso posterior.
- Teste de conectividade com um serviço externo: Realizar uma requisição simples para verificar se o serviço externo está acessível e retornando respostas.

- Atualização de uma biblioteca ou SDK para a versão mais recente: Atualizar uma dependência que facilita a integração com um serviço externo para garantir compatibilidade.
- Implementação de um ping a um endpoint externo: Enviar uma requisição para verificar se um endpoint está online e responsivo.
- Adição de um endpoint externo ao arquivo de ambiente: Configurar a URL de um serviço externo em variáveis de ambiente para uso na aplicação.
- Consumo de um feed RSS ou JSON estático: Obter dados de um feed público e exibi-los em uma parte da aplicação.
- Criação de uma função utilitária para chamadas API básicas: Escrever uma função simples que possa ser reutilizada para fazer requisições GET a um serviço externo.
- Configuração de headers básicos em uma requisição: Adicionar cabeçalhos padrão a uma requisição HTTP para um serviço que os exige.
- Implementação de tratamento básico de erros em chamadas API: Adicionar lógica simples para lidar com falhas de conexão ou respostas inválidas de um serviço externo.
- Verificação do status de um serviço externo: Implementar uma chamada que verifica o status ou a saúde de um serviço através de um endpoint dedicado.
- Consumo de dados estáticos de um arquivo hospedado externamente: Carregar dados de um arquivo JSON ou CSV disponível em um servidor externo.
- Adição de parâmetros de query simples a uma requisição: Modificar a URL de uma requisição para incluir parâmetros como ?lang=pt.
- Configuração de CORS para permitir requisições a um serviço externo: Ajustar as configurações da aplicação para permitir chamadas a APIs que estão em domínios diferentes.
- Criação de documentação básica para uma integração existente: Escrever instruções simples sobre como a aplicação interage com um serviço externo específico.
- Teste de uma API usando uma ferramenta como Postman ou Insomnia: Realizar testes manuais para entender as respostas de um serviço antes de implementar na aplicação.
- Atualização de endpoints de API devido a mudanças menores: Alterar URLs de API na aplicação para refletir atualizações feitas pelo provedor do serviço.
- Implementação de um timeout padrão para requisições externas: Definir um tempo limite para evitar que a aplicação fique esperando indefinidamente por uma resposta.

- Adição de logs básicos para monitorar chamadas API: Incluir mensagens de log simples para registrar quando uma requisição é feita e qual foi a resposta.
- Configuração de uma dependência ou pacote necessário para integração: Instalar e configurar uma biblioteca necessária para se comunicar com um serviço externo.
- Implementação de uma chamada API para obter dados de localização: Consumir uma API pública que retorna informações geográficas baseadas em IP.
- Adição de um botão que faz uma requisição simples a um serviço externo: Criar um botão que, ao ser clicado, faz uma chamada a uma API e exibe uma resposta estática.
- Atualização de credenciais expiradas de acesso a um serviço: Substituir chaves ou tokens que perderam a validade por novas credenciais fornecidas.
- Implementação de uma requisição POST simples sem payload complexo: Enviar dados básicos a um serviço externo que não requer estruturação complexa de dados.
- Verificação de respostas de erro padrão de uma API: Implementar lógica para reconhecer e lidar com códigos de status HTTP comuns, como 404 ou 500.
- Criação de um mock simples para simular respostas de um serviço externo: Configurar uma resposta estática para facilitar testes sem depender do serviço real.
- Adição de suporte a HTTPS em chamadas a serviços externos: Garantir que as requisições utilizem o protocolo seguro quando necessário.
- Implementação de callbacks ou promessas para chamadas assíncronas simples: Ajustar a chamada para lidar com a natureza assíncrona da comunicação com a API.
- Atualização de endpoints devido a mudanças de versão menor na API: Modificar as URLs para corresponder a uma atualização da API de v1 para v1.1, por exemplo.
- Criação de um teste unitário básico para uma função de chamada API: Escrever um teste simples que verifica se a função retorna os dados esperados.
- Configuração de headers de autenticação simples: Adicionar um header de autorização com uma chave de API que não requer processos de autenticação complexos.

- Implementação de um spinner de carregamento durante uma chamada API: Exibir um indicador de carregamento enquanto aguarda a resposta de uma requisição.
- Adição de suporte a JSON ou XML em chamadas API: Configurar a aplicação para interpretar e utilizar respostas em diferentes formatos de dados.
- Verificação de compatibilidade de versão de uma biblioteca de integração: Certificar-se de que a versão da biblioteca utilizada é compatível com a API do serviço externo.
- Criação de uma variável para armazenar o endpoint base da API: Centralizar a URL base da API em uma variável para facilitar futuras alterações.
- Implementação de cache básico para resultados de chamadas API: Armazenar temporariamente a resposta de uma API para evitar requisições redundantes em curto prazo.
- Adição de suporte a parâmetros de caminho na URL: Modificar a chamada API para incluir identificadores específicos na URL, como /users/123.
- Configuração de um proxy simples para chamadas API: Ajustar as configurações de rede para permitir que a aplicação acesse serviços externos através de um proxy.
- Implementação de uma função para parsear respostas da API: Escrever uma função que converte a resposta da API em um formato utilizável pela aplicação.
- Atualização de endpoints para ambiente de teste ou produção: Alterar as URLs para apontar para o ambiente correto, conforme necessário.
- Criação de uma mensagem de erro amigável para falhas em chamadas API: Exibir uma mensagem clara ao usuário quando a requisição não puder ser concluída.
- Implementação de retries simples em caso de falha na requisição: Tentar novamente uma chamada API uma vez em caso de falha temporária.
- Adição de suporte a compressão em chamadas API: Configurar headers para aceitar respostas comprimidas, melhorando a performance.
- Configuração de SSL/TLS para comunicação segura com a API: Garantir que as chamadas utilizem protocolos seguros quando exigido pelo serviço externo.
- Implementação de uma função para formatar dados recebidos de uma API: Ajustar os dados para o formato esperado pela aplicação após recebê-los.
- Verificação de limites de taxa (rate limiting) simples: Reconhecer quando a API retorna um código indicando que o limite de requisições foi atingido.

- Atualização de documentação com informações de endpoints e parâmetros usados: Manter registros atualizados das chamadas API implementadas.
- Criação de um exemplo de chamada API para equipe de desenvolvimento: Fornecer um modelo simples de como realizar chamadas a um serviço externo.
- Implementação de suporte a diferentes métodos HTTP básicos: Além do GET, permitir que a aplicação faça chamadas usando POST, PUT ou DELETE quando apropriado.
- Adição de comentários no código explicando integrações simples: Documentar partes do código que realizam chamadas a serviços externos para facilitar manutenção.

5 (cinco) Story Points

Tarefas de 5 Story Points em Integração de APIs e Serviços Externos envolvem um nível de complexidade moderado, normalmente exigindo interações com APIs que requerem algum tipo de autenticação básica ou manipulação de dados recebidos. Essas tarefas demandam mais esforço do que as atividades simples, mas ainda são gerenciáveis em um curto período, com um grau intermediário de configuração e implementação.

Um exemplo típico seria a implementação de autenticação básica (Basic Auth) em chamadas de API, onde o desenvolvedor configura a aplicação para enviar credenciais simples, como nome de usuário e senha, no cabeçalho de requisições. Outras atividades podem incluir o consumo de uma API que requer um token de acesso estático, que deve ser inserido no cabeçalho da requisição ou como parâmetro de query string para realizar chamadas autenticadas.

Essas tarefas também podem envolver a manipulação de dados JSON recebidos de uma API, transformando-os em um formato adequado para exibição ou processamento dentro da aplicação. Além disso, há a necessidade de lidar com tratamento de erros mais detalhado, implementando lógica que lida com respostas específicas da API, como diferentes códigos de erro (404, 500), e tomando as ações corretivas necessárias ou exibindo mensagens adequadas para o usuário.

Exemplos:

- Implementação de autenticação básica (Basic Auth) em chamadas API: Configurar as requisições para incluir credenciais básicas (usuário e senha)

no header de autorização ao consumir uma API que requer autenticação simples.

- Consumo de uma API com autenticação por token estático: Ajustar as chamadas para incluir um token de acesso fornecido, inserindo-o nos headers ou como parâmetro de query conforme exigido pela API.
- Manipulação de dados JSON recebidos de uma API: Processar a resposta de uma API para extrair informações específicas, transformando os dados em um formato adequado para uso na aplicação.
- Implementação de chamadas API com parâmetros dinâmicos: Fazer requisições que utilizam parâmetros fornecidos pelo usuário ou por outra parte da aplicação, como filtros ou critérios de busca.
- Tratamento de erros e exceções específicos de uma API: Implementar lógica para lidar com diferentes códigos de erro retornados pela API, exibindo mensagens adequadas ao usuário ou tomando ações corretivas.
- Integração com uma API que requer chave de API dinâmica: Configurar o sistema para obter e utilizar uma chave de API que pode mudar periodicamente, armazenando-a de forma segura.
- Criação de uma função para paginar resultados de uma API: Implementar lógica que consome dados paginados, permitindo navegar entre páginas de resultados retornados pela API.
- Envio de dados em formato JSON ou XML para uma API: Estruturar e serializar dados em um formato específico exigido pela API ao fazer requisições POST ou PUT.
- Implementação de chamadas API usando métodos HTTP diferentes (PUT, DELETE): Realizar operações de atualização ou exclusão de recursos em uma API, além das requisições GET e POST.
- Configuração de headers personalizados em requisições: Adicionar headers específicos exigidos pela API, como Content-Type, Accept-Language ou outros necessários para comunicação correta.
- Consumo de uma API que retorna dados em formato não convencional: Ajustar a aplicação para interpretar respostas em formatos como YAML, Protobuf ou outros, realizando a conversão necessária.
- Implementação de consultas com filtros complexos em uma API: Construir requisições que utilizam múltiplos parâmetros de filtro, ordenação ou busca avançada conforme suportado pela API.
- Integração com uma API que utiliza autenticação OAuth 1.0 simples: Configurar a aplicação para autenticar-se usando OAuth 1.0, incluindo a assinatura de requisições conforme necessário.

- Criação de uma função para atualizar recursos via API: Implementar chamadas que permitem atualizar dados existentes em um serviço externo, utilizando métodos como PUT ou PATCH.
- Implementação de retry com exponencial backoff para chamadas API: Ajustar a lógica de requisições para tentar novamente em caso de falhas transitórias, aumentando o intervalo entre tentativas.
- Configuração de tempo limite (timeout) personalizado para requisições específicas: Definir tempos limites diferentes para certas chamadas que podem demorar mais, evitando bloqueios na aplicação.
- Implementação de upload de arquivos para uma API externa: Enviar arquivos a um serviço externo, configurando a requisição para lidar com multipart/form-data e outros requisitos.
- Criação de um cache simples para respostas de API frequentes: Armazenar em cache respostas de APIs que não mudam com frequência para melhorar a performance e reduzir o número de chamadas.
- Manipulação de autenticação com tokens de sessão simples: Gerenciar tokens de sessão fornecidos pela API, armazenando-os temporariamente para uso em múltiplas requisições.
- Integração com um serviço de geocodificação básico: Consumir uma API que converte endereços em coordenadas geográficas ou vice-versa, processando os resultados para uso na aplicação.
- Implementação de internacionalização em chamadas API: Configurar requisições para enviar cabeçalhos ou parâmetros que definem o idioma desejado nas respostas da API.
- Tratamento de dados de resposta com estrutura complexa: Navegar em objetos JSON aninhados para extrair informações relevantes, lidando com possíveis inconsistências nos dados.
- Configuração de um cliente API para reutilização de conexões: Ajustar a aplicação para manter conexões persistentes quando possível, melhorando a eficiência das chamadas.
- Integração com um serviço de envio de e-mails simples via API: Configurar chamadas para enviar e-mails através de um serviço externo, fornecendo os parâmetros necessários.
- Implementação de lógica para lidar com rate limiting básico: Detectar quando a API retorna respostas indicando limite de requisições atingido e implementar espera antes de novas tentativas.
- Criação de testes unitários para funções que consomem APIs externas: Escrever testes que simulam respostas de APIs para verificar o comportamento das funções de integração.

- Atualização de dados locais com base em respostas de API: Implementar lógica que atualiza o estado da aplicação ou banco de dados local com informações obtidas de um serviço externo.
- Implementação de logs detalhados para chamadas API: Registrar informações detalhadas sobre as requisições e respostas para facilitar o diagnóstico de problemas.
- Integração com uma API que requer autenticação via token Bearer: Configurar as requisições para incluir um token Bearer no header de autorização, obtendo-o previamente se necessário.
- Criação de uma função para deletar recursos via API: Implementar chamadas que permitem excluir recursos em um serviço externo utilizando o método DELETE.
- Implementação de suporte a múltiplos ambientes em chamadas API: Ajustar a aplicação para utilizar diferentes endpoints ou configurações conforme o ambiente (desenvolvimento, homologação, produção).
- Configuração de serialização e desserialização customizada de dados: Ajustar como os dados são convertidos entre objetos da aplicação e formatos para envio ou recebimento de uma API.
- Integração com um serviço de autenticação simples (login via API): Implementar chamadas que permitem autenticar usuários através de um serviço externo, gerenciando as sessões.
- Implementação de chamadas assíncronas com tratamento de promessas: Utilizar `async/await` ou promessas para lidar com chamadas API assíncronas, garantindo o fluxo correto da aplicação.
- Criação de uma função para lidar com redirecionamentos em chamadas API: Ajustar as requisições para seguir ou não redirecionamentos conforme necessário pela API.
- Integração com uma API que utiliza websockets simples: Configurar a aplicação para se conectar a um websocket e receber dados em tempo real, exibindo informações atualizadas.
- Implementação de lógica para atualizar tokens expirados: Detectar quando um token de autenticação expirou e obter um novo automaticamente antes de refazer a requisição.
- Configuração de políticas de CORS específicas para chamadas API: Ajustar o servidor ou cliente para lidar com requisitos de CORS ao consumir APIs que exigem configurações especiais.
- Criação de uma função para tratar diferentes tipos de respostas (JSON, XML, texto): Implementar lógica que identifica o tipo de conteúdo da resposta e processa adequadamente.

- Implementação de compressão de dados enviados para a API: Configurar as requisições para comprimir payloads grandes, melhorando a eficiência da comunicação.
- Integração com um serviço de notificações push básico: Configurar a aplicação para enviar notificações push através de uma API externa, fornecendo os parâmetros necessários.
- Implementação de lógica para lidar com diferentes fusos horários em dados recebidos: Ajustar datas e horas recebidas de uma API para o fuso horário correto utilizado na aplicação.
- Configuração de autenticação via API Key em query string: Incluir a chave de API como parâmetro na URL, conforme exigido por alguns serviços externos.
- Criação de um mecanismo simples de reconexão para websockets: Implementar lógica que tenta reconectar automaticamente quando a conexão websocket é perdida.
- Implementação de suporte a versões diferentes de uma API: Ajustar as chamadas para utilizar diferentes versões da API conforme necessário, mantendo compatibilidade.
- Integração com um serviço de mensagens SMS básico: Consumir uma API que permite enviar mensagens SMS, fornecendo os parâmetros necessários e tratando as respostas.
- Implementação de lógica para detectar e tratar respostas em cache: Reconhecer quando uma resposta veio do cache e decidir se deve ser atualizada ou utilizada como está.
- Configuração de políticas de retry para diferentes tipos de erros: Definir quais erros devem acionar novas tentativas de requisição e quais devem ser tratados imediatamente.
- Criação de um wrapper simples para chamadas API repetitivas: Escrever funções ou classes que encapsulam chamadas comuns, reduzindo repetição de código.
- Implementação de lógica para monitorar o desempenho de chamadas API: Coletar métricas simples como tempo de resposta e taxa de sucesso para análise posterior.

13 (treze) Story Points

Integração de APIs e Serviços Externos envolvem um nível moderado de complexidade. Essas tarefas incluem interações com APIs que requerem autenticação e autorização mais elaboradas, manipulação de dados complexos,

implementação de lógica de negócios associada às integrações e tratamento abrangente de erros e exceções.

Tarefas de 13 Story Points em Integração de APIs e Serviços Externos envolvem uma complexidade moderada, exigindo interações mais avançadas com APIs e a implementação de lógica de negócios associada à integração. Essas tarefas frequentemente requerem autenticação e autorização mais elaboradas, manipulação de dados complexos e o tratamento abrangente de erros e exceções. Esse nível de tarefa já implica na necessidade de configurar fluxos de autenticação ou manipular respostas complexas de serviços externos.

Um exemplo comum é a implementação de autenticação OAuth 2.0 para consumir APIs, onde a aplicação precisa obter e renovar tokens de acesso. Esse processo envolve o gerenciamento de sessões e tokens, além de lidar com falhas na autenticação e possíveis erros de renovação dos tokens. Outro exemplo seria a integração com APIs baseadas em GraphQL, que exige a construção de queries e mutations para obter e manipular dados, juntamente com o tratamento de respostas e erros específicos desse tipo de API.

Tarefas de 13 Story Points podem incluir o desenvolvimento de um cliente API personalizado, que encapsula chamadas, gerencia sessões e tokens de acesso, e inclui lógica para retry em caso de falhas. Esse tipo de tarefa envolve a criação de uma solução que facilita a comunicação com a API, lidando com a complexidade da autenticação e da manipulação de dados retornados.

Exemplos:

- Implementação de autenticação OAuth 2.0 para consumo de APIs: Configurar a aplicação para autenticar-se usando o fluxo OAuth 2.0, incluindo obtenção e renovação de tokens de acesso e atualização.
- Integração com APIs que utilizam GraphQL: Consumir uma API baseada em GraphQL, construindo queries e mutations para obter e manipular dados, além de tratar respostas e erros específicos do GraphQL.
- Criação de um cliente API personalizado com gerenciamento de sessão: Desenvolver uma classe ou módulo que encapsula chamadas API, gerencia sessões, tokens e inclui lógica para manipular erros e retries.
- Implementação de sincronização bidirecional de dados com um serviço externo: Desenvolver lógica que mantém dados consistentes entre a aplicação e um serviço externo, incluindo detecção de alterações e resolução de conflitos.

- Integração com um serviço de pagamento (ex: Stripe, PayPal): Implementar chamadas para processar pagamentos, lidar com tokens de cartão, gerenciar webhooks para confirmação de transações e tratar possíveis erros.
- Implementação de um sistema de webhooks para receber notificações de um serviço externo: Configurar endpoints que recebem chamadas de um serviço externo, validar e processar os dados recebidos de forma segura.
- Consumo de APIs com requisitos de segurança avançados (ex: assinatura de requisições): Implementar lógica que inclui assinatura criptográfica das requisições, seguindo protocolos específicos exigidos pela API.
- Integração com serviços de terceiros que requerem SDKs específicos: Utilizar e configurar SDKs fornecidos por serviços externos, entendendo sua documentação e ajustando para atender às necessidades da aplicação.
- Implementação de lógica para lidar com rate limiting avançado: Gerenciar limites de requisições impostos pela API, incluindo implementação de filas ou esperas assíncronas para respeitar as políticas do serviço externo.
- Criação de um sistema de cache avançado para dados de APIs: Desenvolver um mecanismo de cache que armazena respostas de APIs, incluindo lógica para invalidação de cache baseada em tempo ou eventos específicos.
- Integração com serviços de autenticação federada (ex: SAML, OpenID Connect): Configurar a aplicação para autenticar usuários através de provedores de identidade externos, implementando os fluxos necessários.
- Implementação de chamadas API em paralelo com coordenação de resultados: Realizar múltiplas requisições simultaneamente para otimizar o tempo de resposta, consolidando e tratando os dados ao final.
- Integração com serviços de armazenamento em nuvem (ex: AWS S3, Google Cloud Storage): Implementar chamadas para upload, download e gerenciamento de arquivos em serviços de armazenamento externo.
- Criação de um mecanismo de fallback para serviços externos: Implementar lógica que utiliza serviços alternativos ou dados locais em caso de falha ou indisponibilidade do serviço externo principal.
- Implementação de autenticação com tokens JWT (JSON Web Tokens): Configurar a aplicação para utilizar tokens JWT, incluindo sua obtenção, validação e renovação conforme necessário.
- Integração com APIs que exigem versionamento e negociação de conteúdo: Ajustar as requisições para especificar versões da API ou formatos de conteúdo suportados, garantindo compatibilidade.
- Desenvolvimento de lógica para processamento em lote de dados via API: Implementar chamadas que enviam ou recebem grandes volumes de dados em lote, incluindo paginação e tratamento de performance.

- Implementação de lógica de retry com políticas avançadas: Configurar tentativas adicionais em caso de falhas, utilizando políticas como circuit breaker ou retry com limites e condições específicas.
- Integração com serviços de mensageria (ex: RabbitMQ, Kafka) via API: Configurar a aplicação para enviar e receber mensagens através de sistemas de mensageria externos, tratando confirmações e erros.
- Implementação de segurança adicional em chamadas API (ex: uso de certificados): Configurar a aplicação para utilizar certificados digitais ou outros mecanismos de segurança ao se comunicar com serviços externos.
- Criação de testes de integração para validação das chamadas API: Desenvolver testes que verificam o comportamento da aplicação em relação às APIs consumidas, incluindo casos de sucesso e falha.
- Integração com serviços de análise ou monitoramento (ex: Google Analytics, Sentry): Implementar chamadas que enviam dados de uso ou erros para plataformas externas de análise e monitoramento.
- Implementação de lógica para atualização automática de dados a partir de APIs: Configurar a aplicação para obter novos dados periodicamente ou em resposta a eventos, mantendo as informações atualizadas.
- Integração com APIs que utilizam protocolos não convencionais (ex: SOAP): Consumir serviços que utilizam protocolos como SOAP, incluindo manipulação de XML e configuração de envelopes de mensagens.
- Desenvolvimento de um sistema de logging centralizado para chamadas API: Implementar um mecanismo que registra detalhes das requisições e respostas de APIs em um local central para análise posterior.
- Implementação de lógica para lidar com diferentes fusos horários e localizações geográficas: Ajustar chamadas e tratamento de dados para considerar diferenças regionais, como formatos de data e moeda.
- Integração com serviços de mídia (ex: streaming de vídeo, processamento de imagens): Consumir APIs que permitem manipulação de mídia, incluindo upload, transformação e streaming de conteúdo.
- Criação de um sistema de notificações em tempo real via websockets ou SSE: Implementar comunicação em tempo real com serviços externos para receber atualizações imediatas.
- Implementação de autenticação via Single Sign-On (SSO): Configurar a aplicação para permitir que usuários façam login utilizando credenciais de um sistema centralizado ou corporativo.
- Integração com APIs que exigem aprovação ou revisão (ex: APIs de redes sociais): Implementar lógica para lidar com restrições impostas por serviços que exigem aprovação prévia de aplicativos ou chamadas.

- Desenvolvimento de lógica para manipulação de erros complexos e exceções personalizadas: Tratar cenários de erro específicos, incluindo análise de códigos de erro e mensagens detalhadas fornecidas pela API.
- Implementação de suporte a multi-tenancy em integrações com serviços externos: Ajustar chamadas API para lidar com múltiplos clientes ou contextos dentro da mesma aplicação.
- Integração com serviços de inteligência artificial ou machine learning: Consumir APIs que fornecem funcionalidades como reconhecimento de imagem, processamento de linguagem natural, entre outros.
- Criação de um sistema de tokenização e anonimização de dados ao consumir APIs: Implementar lógica que protege dados sensíveis ao interagir com serviços externos, atendendo a requisitos de privacidade.
- Implementação de processos assíncronos com filas de mensagens para chamadas API demoradas: Utilizar filas para gerenciar requisições que podem levar muito tempo, evitando bloqueios na aplicação.
- Integração com serviços de localização avançada (ex: cálculo de rotas, geofencing): Consumir APIs que fornecem funcionalidades geoespaciais complexas, incluindo tratamento de dados retornados.
- Desenvolvimento de lógica para manipulação de dados em tempo real: Implementar integrações que exigem processamento imediato de dados recebidos de serviços externos, com baixa latência.
- Implementação de mecanismos de segurança para prevenção de ataques (ex: throttling): Configurar limites e monitoramento para evitar abusos nas chamadas a serviços externos, protegendo a aplicação.
- Integração com APIs que requerem autenticação mútua (mutual TLS): Configurar certificados tanto no cliente quanto no servidor para estabelecer comunicação segura com a API.
- Criação de mecanismos para auditoria e conformidade em chamadas API: Implementar registro detalhado das interações com serviços externos, atendendo a requisitos legais ou de conformidade.
- Implementação de lógica para transformação de dados complexos: Ajustar formatos de dados entre a aplicação e o serviço externo, incluindo mapeamentos, conversões e validações.
- Integração com serviços de terceiros que exigem testes e certificações: Preparar a aplicação para cumprir requisitos de certificação ou homologação necessários para utilizar certas APIs.
- Desenvolvimento de um sistema de atualização em tempo real de múltiplos serviços externos: Coordenar chamadas a diferentes APIs para manter os dados sincronizados e atualizados.

- Implementação de lógica para gerenciamento de transações distribuídas: Garantir consistência de dados ao realizar operações que envolvem múltiplos serviços ou recursos externos.
- Integração com serviços de identidade digital ou certificação eletrônica: Consumir APIs que permitem assinatura digital de documentos, verificação de identidade ou autenticação forte.
- Criação de um sistema de monitoramento de performance de serviços externos: Implementar ferramentas que acompanham a disponibilidade e desempenho das APIs consumidas, alertando sobre problemas.
- Implementação de processos de batch ou processamento em massa via APIs: Desenvolver lógica que lida com grandes volumes de dados, incluindo chunking, paralelismo e controle de fluxo.
- Integração com serviços que utilizam protocolos de comunicação em tempo real (ex: MQTT): Configurar a aplicação para se comunicar com serviços que utilizam protocolos específicos para IoT ou outras aplicações.
- Desenvolvimento de lógica para reconciliação de dados entre sistemas: Comparar e ajustar dados entre a aplicação e o serviço externo para garantir consistência e integridade.
- Implementação de funcionalidades para lidar com autenticação multifator em serviços externos: Adaptar a aplicação para suportar etapas adicionais de autenticação exigidas por certos serviços.
- Integração com APIs que fornecem serviços de blockchain ou contratos inteligentes: Consumir serviços que permitem interação com redes blockchain, incluindo envio de transações e consulta de dados.
- Criação de mecanismos para lidar com atualização de esquemas ou contratos de APIs: Adaptar a aplicação para mudanças nas definições das APIs, garantindo compatibilidade contínua.

34 (trinta e quatro) Story Points

Tarefas de 34 Story Points em Integração de APIs e Serviços Externos envolvem um nível significativo de complexidade e esforço, exigindo a implementação de lógica avançada e interações com APIs que demandam autenticação e autorização complexas, bem como manipulação de grandes volumes de dados. Essas tarefas incluem integrações críticas para a aplicação, que exigem um planejamento cuidadoso e a capacidade de tratar erros e exceções de maneira abrangente, garantindo que a comunicação com os serviços externos seja segura e eficiente.

Um exemplo típico de tarefa de 34 Story Points seria a implementação do fluxo completo de autorização OAuth 2.0, que envolve redirecionar o usuário para uma página de login, obter tokens de acesso, armazená-los de forma segura e renová-los quando necessário. Esse tipo de tarefa exige a compreensão e aplicação correta de um processo de autenticação robusto, além do tratamento seguro dos tokens gerados.

Outra tarefa seria a integração com múltiplas APIs de serviços externos simultaneamente, onde o desenvolvedor deve coordenar chamadas a diferentes APIs, combinar os dados recebidos e processá-los de forma que a aplicação integre todas as informações de maneira coesa. Esse tipo de tarefa inclui manipulação de dados em larga escala e o gerenciamento de fluxos de resposta para garantir que os dados sejam processados corretamente.

Além disso, tarefas de 34 Story Points podem incluir a implementação de um sistema de pagamento completo, integrando com vários métodos de pagamento (cartões de crédito, débito, boletos) e incluindo o tratamento de erros, confirmações de transações e reembolsos. Tarefas como a sincronização de dados em tempo real entre a aplicação e um serviço externo usando WebSockets ou APIs em tempo real, garantindo a consistência dos dados, também são exemplos dessa complexidade.

Exemplos:

- Implementação de autenticação OAuth 2.0 com fluxo de autorização completo: Configurar a aplicação para utilizar o fluxo completo de autorização do OAuth 2.0, incluindo redirecionamento para páginas de login, obtenção de tokens de acesso e atualização, e armazenamento seguro desses tokens.
- Integração com múltiplas APIs de serviços externos simultaneamente: Desenvolver lógica que coordena chamadas a várias APIs diferentes, combinando e processando os dados recebidos para fornecer funcionalidades integradas na aplicação.
- Implementação de um sistema de pagamento completo com suporte a diferentes métodos: Integrar com um ou mais gateways de pagamento para processar transações usando cartões de crédito, débito, boletos e outros métodos, incluindo tratamento de erros, confirmações e reembolsos.
- Criação de um sistema de sincronização de dados em tempo real com um serviço externo: Implementar comunicação bidirecional em tempo real

com um serviço externo, utilizando tecnologias como WebSockets ou APIs em tempo real, garantindo consistência dos dados.

- Desenvolvimento de um middleware personalizado para manipulação de requisições API: Criar um middleware que intercepta e processa requisições e respostas de APIs, implementando lógica de negócio, transformação de dados e gerenciamento de erros complexos.
- Integração com um serviço de autenticação multifator (MFA): Configurar a aplicação para suportar autenticação multifator através de um serviço externo, incluindo gerenciamento de tokens temporários, códigos via SMS ou aplicativos autenticadores.
- Implementação de um sistema de webhooks complexo para múltiplos eventos: Configurar endpoints que recebem diversos tipos de eventos de um serviço externo, processando cada um de acordo com regras de negócio específicas e garantindo segurança e confiabilidade.
- Integração com APIs que requerem conformidade com padrões de segurança avançados (ex: PCI DSS): Implementar integrações que atendem a requisitos rigorosos de segurança e conformidade, incluindo criptografia, mascaramento de dados sensíveis e auditorias.
- Desenvolvimento de lógica para manipular grandes volumes de dados através de APIs: Implementar mecanismos eficientes para lidar com transferência e processamento de grandes quantidades de dados, incluindo paginação avançada, compressão e otimização de performance.
- Implementação de um sistema de notificação push complexo utilizando serviços externos: Integrar com serviços como Firebase Cloud Messaging ou Apple Push Notification Service, gerenciando tokens de dispositivos, tópicos de assinatura e envio de mensagens personalizadas.
- Criação de um mecanismo de fallback e tolerância a falhas robusto: Desenvolver lógica que permite à aplicação continuar operando mesmo quando um ou mais serviços externos estão indisponíveis, incluindo uso de caches, serviços alternativos ou modos degradados.
- Integração com APIs que utilizam protocolos de comunicação complexos (ex: AMQP, STOMP): Configurar a aplicação para se comunicar com serviços externos utilizando protocolos avançados, incluindo tratamento de mensagens, assinaturas e confirmação de entrega.
- Desenvolvimento de um sistema de autenticação single sign-on corporativo (ex: LDAP, Active Directory): Implementar integração com sistemas de autenticação corporativos, permitindo que usuários utilizem suas credenciais existentes para acessar a aplicação.
- Implementação de um mecanismo de rate limiting inteligente para chamadas a APIs: Desenvolver lógica que monitora e controla o número de

requisições feitas a serviços externos, ajustando dinamicamente conforme políticas e evitando bloqueios ou penalidades.

- Integração com serviços de processamento de pagamentos internacionais: Adaptar a aplicação para processar pagamentos em múltiplas moedas, lidar com taxas de câmbio, impostos internacionais e regulamentações específicas de cada país.
- Desenvolvimento de lógica para reconciliação financeira com sistemas externos: Implementar processos que conciliam transações financeiras entre a aplicação e serviços externos, identificando discrepâncias e gerando relatórios detalhados.
- Implementação de um sistema de monitoramento e alerta para integrações com APIs: Configurar ferramentas que monitoram a saúde das integrações, enviando alertas em tempo real em caso de falhas, latências elevadas ou comportamentos anômalos.
- Criação de um módulo de integração com serviços de inteligência artificial avançados: Consumir APIs que fornecem funcionalidades como aprendizado de máquina, análise preditiva ou processamento de linguagem natural, incluindo tratamento de dados complexos.
- Implementação de segurança avançada com criptografia ponta a ponta nas comunicações: Configurar a aplicação para criptografar dados em trânsito e em repouso ao interagir com serviços externos, atendendo a requisitos de segurança elevados.
- Integração com sistemas legados que requerem protocolos ou formatos de dados específicos: Desenvolver adaptadores ou tradutores que permitem a comunicação com sistemas antigos, incluindo manipulação de EDI, XML complexo ou protocolos proprietários.
- Desenvolvimento de um sistema de atualização automática de dados com agendamento: Implementar processos que realizam chamadas a APIs em horários programados, incluindo gerenciamento de agendamentos, concorrência e tratamento de exceções.
- Implementação de lógica para gestão de sessões e autenticação distribuída: Gerenciar sessões de usuário e autenticação em um ambiente distribuído, coordenando informações entre diferentes serviços e mantendo a segurança.
- Integração com serviços de mensageria corporativos (ex: IBM MQ, TIBCO): Configurar a aplicação para enviar e receber mensagens através de sistemas de mensageria utilizados em ambientes corporativos, garantindo confiabilidade e desempenho.
- Criação de um sistema de auditoria detalhada das interações com APIs: Implementar registro abrangente de todas as chamadas a serviços

externos, incluindo dados enviados, recebidos e contexto, atendendo a requisitos de conformidade.

- Implementação de suporte a transações distribuídas com garantia de consistência: Desenvolver lógica que coordena operações que envolvem múltiplos serviços externos, garantindo que todas sejam concluídas com sucesso ou revertidas em caso de falha.
- Desenvolvimento de um módulo de integração com serviços de logística e rastreamento avançados: Consumir APIs que fornecem informações detalhadas de logística, permitindo rastreamento em tempo real, estimativas de entrega e notificações.
- Implementação de mecanismos de autenticação e autorização complexos com RBAC ou ABAC: Configurar a aplicação para utilizar modelos avançados de controle de acesso ao interagir com serviços externos que exigem tais modelos.
- Integração com APIs que fornecem serviços de realidade aumentada ou virtual: Consumir serviços que permitem incorporar funcionalidades de AR/VR na aplicação, incluindo manipulação de modelos 3D e dados sensoriais.
- Desenvolvimento de lógica para gestão de estado e sincronização em aplicações offline: Implementar mecanismos que permitem à aplicação funcionar sem conexão, sincronizando dados com serviços externos quando a conectividade é restabelecida.
- Implementação de um sistema de cache distribuído para dados de APIs: Configurar caches que funcionam em ambientes distribuídos, garantindo consistência e desempenho ao acessar dados de serviços externos.
- Integração com serviços de compliance e verificação regulatória: Consumir APIs que verificam conformidade com leis e regulamentações, como verificações de KYC (Know Your Customer) ou AML (Anti-Money Laundering).
- Desenvolvimento de um sistema de transformação e mapeamento de dados complexo: Implementar lógica que converte dados entre diferentes esquemas ou formatos, incluindo validações, enriquecimento e limpeza de dados.
- Implementação de mecanismos de balanceamento de carga para chamadas a serviços externos: Distribuir requisições entre múltiplas instâncias de um serviço externo, otimizando desempenho e disponibilidade.
- Integração com serviços de gerenciamento de identidades e acessos (IAM) avançados: Configurar a aplicação para utilizar serviços externos de IAM, incluindo provisionamento de usuários, grupos e políticas de acesso.

- Criação de um sistema de fila e processamento assíncrono para integrações demoradas: Implementar filas e workers para gerenciar operações com serviços externos que podem levar muito tempo, evitando bloqueios na aplicação principal.
- Implementação de lógica para manipulação de eventos e streams de dados: Consumir e processar fluxos contínuos de dados provenientes de serviços externos, aplicando transformações e análises em tempo real.
- Desenvolvimento de um módulo de integração com serviços de reconhecimento de voz ou imagem: Consumir APIs que permitem transcrição de áudio, reconhecimento facial ou análise de imagens, incluindo tratamento de dados binários e resultados complexos.
- Integração com serviços de blockchain para registro ou verificação de dados: Desenvolver interações com APIs que permitem registrar informações em blockchain ou verificar autenticidade de dados.
- Implementação de mecanismos de segurança contra-ataques específicos (ex: MITM, replay attacks): Adicionar proteções adicionais nas comunicações com serviços externos para prevenir ataques sofisticados.
- Criação de um sistema de configuração dinâmica de integrações: Permitir que endpoints, credenciais e parâmetros de APIs externas sejam configurados em tempo de execução, facilitando adaptações sem necessidade de deploys.
- Desenvolvimento de lógica para lidar com concorrência e consistência em operações distribuídas: Garantir que operações que envolvem múltiplos serviços externos mantenham consistência mesmo em ambientes concorrentes.
- Implementação de um sistema de logging e monitoramento distribuído para integrações: Centralizar logs e métricas de interações com serviços externos, permitindo análise e diagnóstico abrangentes.
- Integração com serviços de orquestração e automação (ex: Zapier, IFTTT): Configurar a aplicação para interagir com plataformas que permitem automatizar fluxos de trabalho envolvendo múltiplos serviços.
- Desenvolvimento de um módulo de integração com serviços de localização indoor: Consumir APIs que fornecem informações de localização dentro de edifícios, exigindo tratamento de dados específicos e cálculos avançados.
- Implementação de lógica para garantir alta disponibilidade nas integrações: Configurar redundâncias e mecanismos de failover ao interagir com serviços externos, garantindo que a aplicação continue operando mesmo em casos de falha.

- Integração com serviços que exigem processos de homologação ou certificação: Preparar a aplicação para cumprir requisitos e passar por processos de homologação exigidos por certos provedores de serviços.
- Desenvolvimento de um sistema de migração de dados entre serviços externos: Implementar processos que transferem dados de um serviço externo para outro, incluindo transformação, validação e verificação de integridade.
- Implementação de suporte a protocolos de comunicação em tempo real avançados (ex: WebRTC): Configurar a aplicação para utilizar protocolos que permitem comunicação em tempo real com baixa latência, incluindo ajustes de rede e segurança.
- Integração com serviços de inteligência artificial personalizados: Consumir APIs que permitem treinamento e implantação de modelos de IA customizados, incluindo gerenciamento de datasets e recursos computacionais.
- Criação de um sistema de testes automatizados para integrações complexas: Desenvolver uma suíte de testes que verifica o funcionamento das integrações com serviços externos, incluindo mocks, stubs e ambientes de teste dedicados.

Design e Arquitetura de Banco de Dados

O Design e Arquitetura de Banco de Dados é um macro assunto essencial que se concentra na criação, estruturação e otimização de sistemas de gerenciamento de banco de dados (SGBD). Este processo envolve o desenvolvimento de esquemas, a definição de relações entre tabelas, a configuração de índices e a criação de políticas de segurança, sempre com o objetivo de garantir desempenho, escalabilidade e integridade dos dados armazenados. A arquitetura de banco de dados é a base sobre a qual a aplicação opera, e um design bem planejado garante que as operações de leitura e escrita sejam eficientes, seguras e escaláveis.

Interagindo diretamente com o Backend, o design e a arquitetura de banco de dados afetam todas as funcionalidades que dependem de dados, desde operações CRUD básicas até integrações complexas com APIs externas. O desempenho da aplicação é diretamente influenciado pelo design do banco, e uma arquitetura inadequada pode levar a problemas de performance, escalabilidade e manutenção. Tarefas como a normalização de dados, a criação de índices para otimizar consultas, ou a configuração de particionamento de

dados (sharding) para distribuir grandes volumes de dados são comuns nesse contexto.

Outro aspecto central desse macro assunto é a segurança de dados, que interage com o tópico de Segurança. Implementar políticas de controle de acesso, criptografia de dados sensíveis e auditoria de operações é crucial para garantir a conformidade com legislações como a LGPD e o GDPR, além de proteger o banco de dados contra acessos indevidos. Além disso, a definição de estratégias de backup e recuperação assegura que os dados possam ser restaurados em caso de falhas, e a implementação de replicação e failover garante alta disponibilidade e resiliência.

Tarefas neste macro assunto podem variar desde a criação de tabelas simples até a implementação de arquiteturas complexas, como a configuração de ETL (Extração, Transformação e Carga) para processamento de grandes volumes de dados. Um design eficaz de banco de dados facilita a escalabilidade horizontal e vertical, garantindo que a aplicação seja capaz de lidar com o crescimento de dados e usuários sem comprometer o desempenho.

1 (um) Story Point

Tarefas de 1 Story Point no Design e Arquitetura de Banco de Dados são caracterizadas por sua simplicidade e impacto mínimo na estrutura geral do banco de dados. Essas tarefas geralmente envolvem ajustes menores ou a criação de elementos básicos que não afetam significativamente a arquitetura ou o desempenho do sistema. O foco está em pequenas otimizações ou adições que podem ser implementadas rapidamente e com baixo risco.

Um exemplo típico de tarefa de 1 Story Point é a criação de uma tabela simples com poucos campos, sem a necessidade de definições complexas de relacionamentos ou restrições. Outras tarefas podem incluir a adição de um índice em uma coluna existente para melhorar a performance de consultas, ou a modificação do tipo de dados de um campo, desde que essa alteração não tenha impacto significativo em outras funcionalidades ou dependências.

Além disso, tarefas como a remoção de colunas não utilizadas, a atualização de valores padrão em campos existentes, ou a criação de uma view simples para facilitar consultas a uma única tabela também se enquadram nesse nível de complexidade. Outro exemplo seria a criação de um usuário de banco de dados com permissões básicas de leitura ou escrita, facilitando o acesso controlado sem a necessidade de gerenciamento avançado de permissões.

Exemplos:

- Criação de uma tabela simples com poucos campos: Definir uma nova tabela no banco de dados com campos básicos, sem restrições ou relacionamentos complexos.
- Adição de um índice em uma coluna existente: Criar um índice em uma coluna para melhorar a performance de consultas simples.
- Modificação do tipo de dados de um campo sem impacto: Alterar o tipo de dados de uma coluna que não afeta outros componentes ou funcionalidades.
- Adição de uma coluna opcional em uma tabela: Incluir uma nova coluna em uma tabela existente, sem requisitos de preenchimento obrigatório ou impacto em chaves.
- Criação de uma view simples para facilitar consultas: Definir uma view que agrega dados de uma única tabela para simplificar o acesso a informações.
- Atualização de valores padrão em campos existentes: Modificar o valor padrão de uma coluna para novos registros, sem alterar dados já inseridos.
- Remoção de uma coluna não utilizada: Excluir uma coluna que não está em uso e não possui dependências no sistema.
- Criação de um usuário de banco de dados com permissões básicas: Adicionar um novo usuário com acesso limitado para operações simples de leitura ou escrita.
- Adição de comentários ou descrições em tabelas e colunas: Documentar elementos do banco de dados para melhorar a compreensão e manutenção futura.
- Atualização de scripts de migração com pequenas mudanças: Ajustar scripts de migração para refletir alterações simples no esquema do banco de dados.
- Configuração de restrições NOT NULL em uma coluna: Definir que uma coluna não pode aceitar valores nulos, garantindo a integridade dos dados.
- Renomeação de uma coluna ou tabela: Alterar o nome de uma coluna ou tabela para refletir melhor seu propósito, sem afetar outras partes do sistema.
- Criação de uma sequência para geração automática de IDs: Implementar uma sequência que automatiza a atribuição de identificadores únicos em uma tabela.
- Adição de uma constraint CHECK simples: Estabelecer uma restrição que verifica se os valores inseridos em uma coluna atendem a uma condição específica.

- Atualização de permissões de um usuário existente: Modificar as permissões de um usuário do banco de dados para restringir ou expandir seu acesso.
- Criação de um sinônimo para uma tabela ou view: Definir um nome alternativo para facilitar o acesso a um objeto do banco de dados.
- Remoção de dados de teste ou temporários: Excluir registros que foram inseridos para testes e não são mais necessários.
- Atualização de estatísticas do banco de dados: Executar comandos que atualizam as estatísticas para otimizar o desempenho de consultas.
- Criação de um índice único em uma coluna: Garantir que os valores em uma coluna sejam únicos, adicionando um índice com essa restrição.
- Adição de colunas calculadas em uma tabela: Incluir colunas que realizam cálculos simples com base em outras colunas da mesma tabela.
- Implementação de um default constraint em uma coluna: Definir um valor padrão para uma coluna caso nenhum valor seja fornecido durante a inserção.
- Criação de um esquema simples: Definir um novo esquema para organizar objetos do banco de dados.
- Atualização de comentários ou documentação do banco de dados: Revisar e melhorar a documentação existente para refletir mudanças recentes.
- Reorganização de índices fragmentados: Executar comandos para reorganizar índices que estão fragmentados, melhorando a performance.
- Criação de um link de banco de dados: Configurar uma conexão simples para permitir acesso a outro banco de dados.
- Adição de uma chave primária em uma tabela existente: Definir uma coluna ou conjunto de colunas como chave primária, desde que não haja conflitos com os dados existentes.
- Criação de uma função escalar simples: Escrever uma função que realiza cálculos ou transformações simples e retorna um único valor.
- Implementação de restrições de integridade referencial básica: Estabelecer chaves estrangeiras simples entre tabelas para garantir a integridade dos dados.
- Remoção de índices não utilizados: Identificar e excluir índices que não estão sendo utilizados, liberando espaço no banco de dados.
- Atualização de collation de uma coluna ou tabela: Alterar o collation para ajustar a forma como os dados de texto são comparados e ordenados.
- Criação de um script para backup simples do banco de dados: Escrever um script básico que realiza o backup dos dados para fins de segurança.

- Adição de um campo de timestamp para controle de alterações: Incluir uma coluna que registra a data e hora de inserção ou atualização de registros.
- Implementação de permissões em nível de coluna: Restringir o acesso a colunas específicas para determinados usuários ou funções.
- Atualização de triggers para refletir mudanças simples: Modificar triggers existentes para ajustá-los a alterações em tabelas ou colunas.
- Criação de uma tabela temporária para operações intermediárias: Definir uma tabela que será usada temporariamente durante uma operação ou processamento.
- Adição de uma função agregada personalizada: Criar uma função que realiza agregações simples, como soma ou média, em um conjunto de registros.
- Implementação de políticas de retenção de dados simples: Definir regras básicas sobre quanto tempo certos dados devem ser mantidos antes de serem arquivados ou excluídos.
- Atualização de dados inconsistentes ou incorretos: Executar scripts que corrigem valores em registros que estão incorretos ou desatualizados.
- Criação de sinônimos para facilitar o acesso a objetos em outros esquemas: Definir nomes alternativos para objetos que residem em esquemas diferentes.
- Adição de restrições de chave estrangeira simples: Estabelecer relacionamentos básicos entre tabelas, garantindo que os valores correspondentes existam.
- Implementação de um mecanismo simples de auditoria: Criar tabelas ou registros que mantêm um histórico básico de alterações em dados críticos.
- Remoção de dados duplicados em uma tabela: Identificar e eliminar registros duplicados para manter a integridade dos dados.
- Criação de uma tabela de referência com valores estáticos: Definir uma tabela que contém valores fixos utilizados como referência em outras tabelas.
- Adição de um campo booleano para controle de status: Incluir uma coluna que indica um estado simples, como ativo/inativo ou verdadeiro/falso.
- Atualização de procedimentos armazenados com pequenas alterações: Modificar stored procedures para refletir mudanças menores nas regras de negócio ou no esquema.
- Implementação de views materializadas simples: Criar uma view materializada que armazena resultados de uma consulta para melhorar a performance.

- Criação de uma política de segurança básica: Definir regras simples que controlam o acesso a determinados objetos do banco de dados.
- Adição de colunas para suporte a internacionalização: Incluir colunas que permitem armazenar dados em diferentes idiomas ou formatos regionais.
- Atualização de tabelas para suportar logs de atividades: Modificar tabelas para incluir campos que registram ações dos usuários ou eventos do sistema.
- Implementação de funções de validação simples: Criar funções que verificam a conformidade de dados com regras básicas antes de serem inseridos ou atualizados.
- Criação de sinônimos para facilitar migrações futuras: Definir nomes alternativos que ajudam na transição entre diferentes versões ou estruturas do banco de dados.

5 (cinco) Story Points

Tarefas de 5 Story Points no Design e Arquitetura de Banco de Dados são um pouco mais complexas do que as de 1 Story Point, exigindo modificações em estruturas existentes ou a criação de novos elementos com relacionamentos simples. O foco dessas tarefas é melhorar a integridade, a eficiência e a organização dos dados, mas sem envolver mudanças profundas na arquitetura do banco de dados.

Um exemplo seria a criação de uma tabela com chave estrangeira simples, estabelecendo uma relação básica entre duas tabelas para garantir a integridade referencial. Outro exemplo seria a adição de índices em múltiplas colunas para otimizar consultas que usam combinações frequentes de filtros, melhorando a performance de busca e análise de dados. Também pode-se incluir a criação de tabelas de junção para implementar relacionamentos muitos-para-muitos, permitindo que duas tabelas se conectem por meio de uma terceira tabela intermediária.

Tarefas desse nível podem incluir a modificação de stored procedures simples para refletir mudanças recentes no esquema do banco de dados, adição de triggers simples para auditoria ou a implementação de restrições de unicidade em colunas para garantir que não haja valores duplicados em campos críticos.

Exemplos:

- Criação de uma tabela com chave estrangeira simples: Definir uma nova tabela que inclui uma chave estrangeira para relacionar-se com outra tabela existente, garantindo a integridade referencial básica.
- Adição de índices em múltiplas colunas para otimizar consultas: Criar índices em várias colunas que são frequentemente usadas em cláusulas WHERE, melhorando o desempenho de consultas específicas.
- Modificação de colunas para aceitar valores nulos ou não nulos: Alterar a definição de colunas para permitir ou não valores nulos, considerando o impacto nos dados existentes e nas operações.
- Criação de um relacionamento muitos-para-muitos com tabela de junção: Implementar uma tabela intermediária para representar um relacionamento muitos-para-muitos entre duas tabelas existentes.
- Atualização de stored procedures simples: Modificar procedimentos armazenados para refletir mudanças recentes no esquema do banco de dados ou melhorar a lógica existente.
- Implementação de restrições de unicidade em colunas: Adicionar constraints UNIQUE a colunas para garantir que não haja valores duplicados, assegurando a integridade dos dados.
- Criação de triggers simples para auditoria: Implementar um trigger que registra alterações em uma tabela específica, como inserções ou atualizações, em uma tabela de log.
- Otimização de consultas com subconsultas simples: Reescrever consultas que utilizam subconsultas para melhorar a performance, evitando scans desnecessários.
- Adição de campos calculados em views ou tabelas: Criar colunas que calculam valores com base em outras colunas, facilitando a obtenção de informações derivadas.
- Implementação de partições simples em tabelas grandes: Dividir uma tabela grande em partições baseadas em uma coluna, como data, para melhorar o desempenho de consultas e manutenção.
- Criação de índices compostos para melhorar consultas complexas: Definir índices que envolvem múltiplas colunas para otimizar consultas que filtram por mais de um critério.
- Atualização de funções de banco de dados com lógica adicional: Modificar funções existentes para incluir novas regras de negócio ou cálculos mais complexos.
- Implementação de políticas de segurança em nível de esquema: Configurar permissões específicas para usuários ou roles em determinados esquemas do banco de dados.

- Criação de visões materializadas com refresh automático: Definir views materializadas que são atualizadas periodicamente para refletir mudanças nos dados subjacentes.
- Adição de colunas com compressão ou criptografia: Incluir colunas que armazenam dados de forma comprimida ou criptografada para otimizar espaço ou segurança.
- Implementação de replicação simples de dados entre bancos: Configurar a replicação básica de tabelas ou esquemas para outro banco de dados, visando redundância ou distribuição de carga.
- Criação de procedimentos armazenados para operações comuns: Escrever stored procedures que encapsulam operações frequentes, como inserções ou atualizações em múltiplas tabelas.
- Atualização de triggers para manipulação de dados complexos: Modificar triggers para executar lógica adicional, como validações ou cálculos ao ocorrerem eventos específicos.
- Implementação de gerenciamento de transações simples: Utilizar comandos de BEGIN, COMMIT e ROLLBACK para garantir a consistência de operações que envolvem múltiplas etapas.
- Criação de índices full-text para pesquisa de texto: Implementar índices que permitem buscas eficientes em colunas de texto, melhorando funcionalidades de pesquisa.
- Adição de constraints CHECK complexas: Definir restrições que aplicam regras mais elaboradas aos valores inseridos nas colunas.
- Otimização de consultas com joins complexos: Reescrever consultas que envolvem múltiplas tabelas para melhorar a eficiência dos joins e reduzir o tempo de execução.
- Implementação de logs de auditoria básicos: Configurar o banco de dados para registrar atividades de usuários ou alterações em dados críticos.
- Criação de funções agregadas personalizadas: Desenvolver funções que realizam agregações específicas não fornecidas nativamente pelo banco de dados.
- Atualização de esquemas para suportar multitenancy simples: Modificar a estrutura do banco de dados para suportar múltiplos clientes ou organizações de forma isolada.
- Implementação de tabelas temporárias para processos complexos: Utilizar tabelas temporárias para armazenar dados intermediários durante operações elaboradas.
- Criação de sinônimos ou aliases para facilitar migrações: Definir nomes alternativos para objetos que estão sendo substituídos ou migrados, evitando impacto imediato no código existente.

- Adição de suporte a tipos de dados avançados (ex: JSON, XML): Modificar tabelas para incluir colunas que armazenam dados semiestruturados, permitindo maior flexibilidade.
- Implementação de lógica de versionamento de dados: Adicionar colunas ou tabelas que permitem rastrear mudanças nos registros ao longo do tempo.
- Criação de tabelas de log para monitoramento: Estabelecer tabelas que registram eventos específicos, como falhas de login ou tentativas de acesso não autorizadas.
- Otimização de índices fragmentados: Executar operações de rebuild ou reorganização de índices que estão altamente fragmentados, melhorando a performance.
- Implementação de políticas de arquivamento de dados: Definir procedimentos para mover dados antigos para tabelas de histórico ou bancos de dados separados.
- Criação de funções para validação de CPF/CNPJ: Desenvolver funções que validam formatos e dígitos verificadores de documentos brasileiros.
- Atualização de configurações de sessão para otimização: Ajustar parâmetros como tamanho de página ou memória alocada para sessões, melhorando o desempenho.
- Implementação de colunas de controle de concorrência (ex: timestamp): Adicionar colunas que ajudam a gerenciar concorrência otimista em atualizações de registros.
- Criação de procedimentos para limpeza de dados obsoletos: Desenvolver scripts que removem periodicamente dados que não são mais necessários.
- Adição de triggers para sincronização básica entre tabelas: Implementar triggers que replicam mudanças de uma tabela para outra, mantendo dados consistentes.
- Implementação de funções para manipulação de datas complexas: Criar funções que calculam períodos, somam intervalos ou formatam datas de forma específica.
- Criação de índices filtrados: Definir índices que incluem apenas registros que atendem a uma determinada condição, reduzindo o tamanho e melhorando a performance.
- Atualização de scripts de migração para ambientes diferentes: Adaptar scripts para serem executados em diferentes ambientes (desenvolvimento, teste, produção) com ajustes necessários.
- Implementação de tabelas particionadas por intervalo: Configurar tabelas que são particionadas com base em intervalos de valores, facilitando a gestão de dados grandes.

- Criação de funções de agregação com janela (window functions): Utilizar funções que permitem cálculos avançados sobre conjuntos de registros relacionados.
- Adição de colunas para suporte a auditoria (ex: usuário que alterou): Incluir colunas que registram quem realizou a última alteração em um registro.
- Implementação de chaves estrangeiras com ações ON DELETE/UPDATE: Definir chaves estrangeiras que especificam ações como CASCADE ou SET NULL ao excluir ou atualizar registros.
- Criação de políticas de segurança com roles e grupos: Estabelecer roles que agrupam permissões e facilitam a gestão de acesso para múltiplos usuários.
- Atualização de estatísticas de uso do banco de dados: Executar comandos que atualizam as estatísticas utilizadas pelo otimizador de consultas.
- Implementação de criptografia transparente de dados (TDE) básica: Configurar o banco de dados para criptografar dados em repouso, aumentando a segurança.
- Criação de procedimentos para gerenciamento de identidades: Desenvolver scripts que gerenciam sequências ou identidades utilizadas para gerar chaves primárias.
- Adição de colunas para suporte a sistemas externos (ex: GUIDs): Incluir colunas que armazenam identificadores globais únicos para integração com outros sistemas.
- Implementação de políticas de restrição de acesso por horário: Configurar regras que permitem ou negam acesso ao banco de dados em determinados períodos.
- Criação de funções para manipulação de strings avançadas: Desenvolver funções que realizam operações complexas com strings, como expressões regulares ou substituições.

13 (treze) Story Point

Tarefas de 13 Story Points no contexto de Design e Arquitetura de Banco de Dados envolvem uma complexidade intermediária, exigindo modificações significativas na estrutura existente ou a criação de novos componentes no banco de dados. Essas tarefas requerem planejamento cuidadoso, já que podem impactar o desempenho geral do sistema e incluir a implementação de lógica de negócios mais complexa diretamente no banco de dados.

Um exemplo típico de uma tarefa desse nível é a criação de um modelo relacional completo para um novo módulo da aplicação. Isso envolve desenvolver o

esquema do banco de dados, definindo tabelas, chaves primárias e estrangeiras, além de relacionamentos entre entidades e criação de índices para otimizar as consultas. Esse processo é fundamental para garantir a consistência dos dados e o bom funcionamento do módulo na aplicação.

Outro exemplo é a normalização de dados até a terceira forma normal, onde o esquema existente é revisado e ajustado para eliminar redundâncias e dependências parciais, melhorando a integridade dos dados. Essa tarefa exige uma análise cuidadosa do design do banco de dados e a reestruturação de tabelas para garantir eficiência e escalabilidade.

Essas tarefas também podem incluir o desenvolvimento de procedimentos armazenados complexos que implementam lógica de negócios diretamente no banco de dados. Isso pode envolver o uso de controle de fluxo, manipulação de cursores e tratamento de exceções para gerenciar operações críticas, como cálculos financeiros ou atualizações em massa de dados. Além disso, triggers complexos podem ser criados para garantir a integridade dos dados, como a manutenção automática de saldos e o histórico de transações.

Exemplos:

- Criação de um modelo relacional completo para um novo módulo: Desenvolver o esquema de banco de dados para um novo módulo da aplicação, incluindo tabelas, chaves primárias, estrangeiras, índices e relacionamentos entre as entidades.
- Implementação de normalização até a terceira forma normal: Revisar e ajustar o esquema existente para eliminar redundâncias e dependências parciais, assegurando que o banco de dados esteja normalizado até a terceira forma normal.
- Desenvolvimento de procedimentos armazenados complexos: Escrever stored procedures que implementam lógica de negócios avançada, incluindo controle de fluxo, manipulação de cursor e tratamento de exceções.
- Criação de triggers complexos para manutenção de integridade: Implementar triggers que garantem a consistência dos dados em situações complexas, como atualização de saldos, histórico de transações ou cálculos financeiros.
- Implementação de replicação de dados entre servidores: Configurar replicação de dados entre diferentes servidores ou instâncias de banco de dados, garantindo alta disponibilidade e tolerância a falhas.

- Otimização de consultas com desempenho insatisfatório: Analisar e otimizar consultas que apresentam problemas de performance, utilizando técnicas como reescrita de queries, criação de índices específicos ou ajustes de planos de execução.
- Criação de tabelas particionadas por intervalo e hash: Implementar particionamento de tabelas grandes utilizando métodos combinados, como intervalo e hash, para melhorar a performance e facilitar a manutenção.
- Implementação de políticas de segurança avançadas com criptografia de dados: Configurar o banco de dados para criptografar dados sensíveis, tanto em trânsito quanto em repouso, incluindo gerenciamento de chaves e certificados.
- Desenvolvimento de funções definidas pelo usuário para cálculos complexos: Criar funções que realizam cálculos avançados, como algoritmos matemáticos, processamento estatístico ou manipulação de dados geoespaciais.
- Implementação de mecanismos de auditoria completos: Configurar o banco de dados para registrar detalhadamente as operações realizadas, incluindo quem realizou, quando e quais dados foram afetados, atendendo a requisitos de compliance.
- Criação de um data warehouse básico: Desenvolver um esquema dimensional para suporte a atividades de Business Intelligence, incluindo fatos, dimensões e hierarquias.
- Implementação de mecanismos de controle de concorrência: Configurar níveis de isolamento de transações apropriados, implementar bloqueios otimizados e resolver problemas de deadlocks.
- Desenvolvimento de processos de ETL simples: Criar scripts ou procedimentos que extraem dados de fontes diversas, transformam e carregam em tabelas específicas para análise ou integração.
- Criação de índices columnstore para otimização de consultas analíticas: Implementar índices que melhoram o desempenho de consultas que envolvem grandes volumes de dados, típicas em análises e relatórios.
- Implementação de funções para manipulação de dados geoespaciais: Configurar o banco de dados para armazenar e processar dados de localização, incluindo cálculos de distância, área e interseção.
- Desenvolvimento de soluções para gerenciamento de dados hierárquicos: Implementar modelos que permitem representar e consultar estruturas hierárquicas, como árvores ou grafos, utilizando técnicas como adjacência, caminho enumerado ou modelos aninhados.

- Criação de políticas de acesso baseadas em linhas (Row-Level Security): Configurar o banco de dados para restringir o acesso a linhas específicas em tabelas, com base em atributos do usuário ou regras definidas.
- Implementação de suporte a multi-tenancy avançado: Ajustar o esquema e as configurações do banco de dados para suportar múltiplos clientes ou organizações, garantindo isolamento e segurança dos dados.
- Desenvolvimento de procedimentos para manutenção automatizada: Criar scripts que executam tarefas de manutenção, como reindexação, atualização de estatísticas, limpeza de logs e monitoramento de espaço em disco.
- Criação de mecanismos de versionamento de esquema: Implementar controles que permitem gerenciar diferentes versões do esquema do banco de dados, facilitando atualizações e migrações.
- Implementação de triggers para sincronização com sistemas externos: Desenvolver triggers que, ao ocorrerem determinadas operações, enviam dados para outros sistemas ou iniciam processos externos.
- Desenvolvimento de modelos de dados para suporte a internacionalização completa: Ajustar o esquema para armazenar dados em múltiplos idiomas, formatos regionais e considerar fusos horários.
- Criação de procedimentos para importação e exportação de dados em massa: Implementar processos que permitem carregar e extrair grandes volumes de dados, incluindo validação e tratamento de erros.
- Implementação de lógica de negócio complexa no banco de dados: Mover partes da lógica da aplicação para o banco de dados, utilizando stored procedures, funções e triggers para melhorar a performance ou garantir integridade.
- Desenvolvimento de soluções para armazenamento e consulta de dados semiestruturados: Configurar o banco de dados para trabalhar eficientemente com dados JSON, XML ou outros formatos, incluindo indexação e consultas específicas.
- Implementação de mecanismos de recuperação de desastres: Configurar estratégias de backup e restauração, incluindo backups incrementais, logs de transações e planos de recuperação em caso de falhas graves.
- Criação de um catálogo de dados: Desenvolver um sistema que documenta e classifica os objetos do banco de dados, facilitando a descoberta e compreensão dos dados disponíveis.
- Otimização de banco de dados para alta concorrência: Ajustar configurações e estruturas para suportar muitos usuários simultâneos, minimizando contenções e melhorando a escalabilidade.

- Implementação de lógica para suporte a auditoria regulatória: Configurar o banco de dados para cumprir requisitos legais específicos, como LGPD ou GDPR, incluindo consentimento, anonimização e direito ao esquecimento.
- Desenvolvimento de soluções para controle de acesso granular: Implementar mecanismos que permitem controlar permissões em nível de coluna ou até mesmo de célula, garantindo segurança detalhada.
- Criação de procedimentos para migração de dados entre ambientes: Desenvolver scripts e processos que permitem migrar dados entre diferentes ambientes (desenvolvimento, teste, produção), mantendo consistência e integridade.
- Implementação de algoritmos de compressão de dados personalizados: Configurar o banco de dados para utilizar técnicas de compressão específicas, reduzindo o espaço em disco e melhorando a performance de I/O.
- Desenvolvimento de soluções para detecção e prevenção de fraudes: Implementar mecanismos que analisam padrões nos dados para identificar atividades suspeitas ou anômalas.
- Criação de modelos preditivos dentro do banco de dados: Utilizar funcionalidades avançadas que permitem executar algoritmos de machine learning diretamente no banco de dados.
- Implementação de suporte a dados temporais ou históricos: Configurar o banco de dados para armazenar versões históricas dos dados, permitindo consultas sobre o estado dos dados em momentos passados.
- Desenvolvimento de processos para anonimização e mascaramento de dados: Implementar técnicas que protegem dados sensíveis em ambientes de teste ou desenvolvimento, sem comprometer a usabilidade.
- Criação de índices espaciais para dados geográficos complexos: Implementar estruturas que permitem consultas eficientes sobre dados de localização complexos.
- Implementação de sistemas de notificações baseados em eventos do banco de dados: Configurar mecanismos que disparam notificações ou mensagens para outros sistemas ou serviços quando certos eventos ocorrem.
- Desenvolvimento de soluções para armazenamento de grandes objetos (BLOBs): Configurar o banco de dados para armazenar e gerenciar eficientemente arquivos grandes, como imagens, vídeos ou documentos.
- Criação de planos de particionamento e distribuição de dados em clusters: Projetar como os dados serão distribuídos em um ambiente de cluster ou sharding, garantindo balanceamento e performance.

- Implementação de funções para criptografia personalizada de campos: Desenvolver funções que criptografam e descriptografam dados em nível de campo, atendendo a requisitos específicos de segurança.
- Desenvolvimento de processos de carga incremental de dados: Criar mecanismos que permitem atualizar dados de forma incremental, identificando e aplicando apenas as mudanças ocorridas.
- Criação de soluções para gerenciamento de metadados: Implementar sistemas que armazenam e gerenciam informações sobre os dados, como origem, qualidade, transformações e linhagem.
- Implementação de estratégias de caching no banco de dados: Configurar mecanismos que armazenam resultados de consultas frequentes, melhorando o tempo de resposta.
- Desenvolvimento de procedimentos para detecção e resolução de conflitos em dados replicados: Implementar lógica que identifica discrepâncias entre réplicas e resolve conflitos de acordo com regras definidas.
- Criação de modelos de dados para suporte a analítica em tempo real: Ajustar o esquema e as estruturas para permitir consultas analíticas com baixa latência em dados em constante atualização.
- Implementação de técnicas de balanceamento de carga em servidores de banco de dados: Configurar o ambiente para distribuir solicitações entre múltiplos servidores, melhorando a disponibilidade e desempenho.
- Desenvolvimento de soluções para integração com bancos de dados NoSQL: Configurar mecanismos que permitem a interação e sincronização de dados entre bancos de dados relacionais e NoSQL.
- Criação de sistemas de alerta para monitoramento de performance: Implementar processos que monitoram métricas críticas do banco de dados e enviam alertas em caso de anomalias.
- Implementação de controles de qualidade de dados: Desenvolver mecanismos que validam e asseguram a qualidade dos dados, identificando inconsistências, outliers e erros.

34 (trinta e quatro) Story Points

Tarefas de 34 Story Points em Design e Arquitetura de Banco de Dados envolvem um nível elevado de complexidade, exigindo modificações substanciais nas estruturas existentes e a criação de novos componentes com um impacto significativo na performance e na arquitetura do banco de dados. Essas tarefas normalmente exigem planejamento detalhado e a implementação de soluções que afetam diretamente o desempenho, a escalabilidade e a segurança dos

dados. O esforço necessário para essas atividades é consideravelmente maior do que para tarefas de 13 Story Points, uma vez que são incluídas otimizações e ajustes críticos.

Um exemplo de tarefa de 34 Story Points é a criação de um data warehouse completo com processos ETL. Isso envolve o design e a implementação de um data warehouse, incluindo a modelagem dimensional, a criação de tabelas de fatos e dimensões, além do desenvolvimento de processos ETL (Extração, Transformação e Carga) para reunir dados de múltiplas fontes, transformá-los e carregá-los de forma estruturada. Esse processo exige integração com diversas fontes de dados e manipulação de grandes volumes, com foco na performance e na consistência dos dados.

Outra tarefa comum nesse nível de complexidade é a implementação de sharding para escalabilidade horizontal. O sharding divide o banco de dados em várias partes, distribuindo os dados entre múltiplos servidores para melhorar o desempenho e permitir que o sistema escale com o aumento da demanda. Além disso, é necessário implementar lógica de roteamento de consultas, garantindo que os dados corretos sejam acessados rapidamente e de forma eficiente.

Também podemos incluir como exemplo a migração de um banco de dados legado para uma nova tecnologia. Esta tarefa exige um planejamento detalhado, execução da migração, reescrita de consultas e procedimentos armazenados, além de garantir a integridade dos dados no novo sistema. Esse tipo de tarefa pode incluir a transformação de dados, ajustes na arquitetura e testes para garantir que o sistema esteja funcionando corretamente após a migração.

Exemplos:

- Criação de um data warehouse completo com processos ETL: Projetar e implementar um data warehouse, incluindo a modelagem dimensional, criação de tabelas de fatos e dimensões, e desenvolver processos ETL para extração, transformação e carregamento de dados de múltiplas fontes.
- Implementação de sharding para escalabilidade horizontal: Reestruturar o banco de dados para suportar sharding, distribuindo dados entre múltiplos servidores para melhorar a escalabilidade e desempenho, incluindo a implementação de lógica de roteamento de consultas.
- Migração de um banco de dados legado para uma nova tecnologia: Planejar e executar a migração de um banco de dados existente para um novo sistema de gerenciamento de banco de dados, incluindo transformação de dados, reescrita de consultas e procedimentos, e garantia de integridade dos dados.

- Projeto e implementação de alta disponibilidade com failover automático: Configurar um ambiente de banco de dados com alta disponibilidade, implementando replicação síncrona, clusters de failover e testando os mecanismos de recuperação em caso de falhas.
- Implementação de criptografia de dados avançada e conformidade regulatória: Configurar criptografia em nível de coluna, tabela e banco de dados, incluindo gerenciamento de chaves, para atender a requisitos de conformidade como LGPD ou GDPR, e realizar auditorias de segurança.
- Otimização de performance para um banco de dados de alto volume: Analisar e otimizar um banco de dados que sofre com problemas de desempenho devido a alto volume de transações, incluindo reestruturação de índices, particionamento de tabelas e ajuste de configurações do SGBD.
- Desenvolvimento de um sistema de replicação e sincronização bidirecional: Implementar replicação bidirecional entre bancos de dados geograficamente distribuídos, garantindo a consistência dos dados e resolvendo conflitos de sincronização.
- Implementação de um sistema de monitoramento e alertas para o banco de dados: Configurar ferramentas de monitoramento que coletam métricas de desempenho e uso de recursos, enviando alertas proativos em caso de anomalias ou problemas críticos.
- Projeto de uma arquitetura de banco de dados para suporte a multi-regiões: Planejar e implementar uma arquitetura que suporta múltiplas regiões geográficas, incluindo replicação de dados, latência otimizada e conformidade com regulamentações locais.
- Desenvolvimento de um sistema de controle de versão de esquemas: Implementar ferramentas e processos que permitem controlar versões do esquema do banco de dados, facilitando a colaboração entre equipes e o gerenciamento de mudanças estruturais.
- Criação de um banco de dados em cluster com balanceamento de carga: Configurar um cluster de bancos de dados com balanceamento de carga para distribuir as consultas entre várias instâncias, melhorando o desempenho e a disponibilidade.
- Implementação de uma solução de backup e recuperação de desastres avançada: Desenvolver um plano completo de backup, incluindo backups incrementais, diferenciais, e configurar um ambiente de recuperação de desastres com testes regulares de restauração.
- Desenvolvimento de um sistema de cache distribuído para dados do banco: Implementar um mecanismo de cache distribuído que reduz a carga no banco de dados, incluindo estratégias de invalidação e atualização do cache em resposta a mudanças nos dados.

- Implementação de um banco de dados NoSQL integrado ao sistema existente: Projetar e integrar um banco de dados NoSQL para armazenar tipos específicos de dados, como documentos ou dados de séries temporais, e ajustar a aplicação para interagir com múltiplos tipos de bancos de dados.
- Projeto e implementação de segurança avançada com Row-Level Security: Configurar políticas de segurança que restringem o acesso a linhas específicas nas tabelas com base em atributos do usuário, garantindo que cada usuário veja apenas os dados permitidos.
- Desenvolvimento de uma solução de particionamento de dados complexa: Implementar particionamento de tabelas com estratégias combinadas (por intervalo, lista, hash) para otimizar consultas e gerenciamento de dados em tabelas muito grandes.
- Implementação de processamento de dados em tempo real no banco: Configurar o banco de dados para processar fluxos de dados em tempo real, utilizando ferramentas e tecnologias como Apache Kafka e Streaming SQL.
- Desenvolvimento de funções e procedimentos para cálculos financeiros complexos: Implementar lógica de negócios complexa diretamente no banco de dados, incluindo cálculos financeiros avançados, provisionamento e reconciliação de dados.
- Otimização de consultas complexas com planos de execução personalizados: Ajustar consultas SQL complexas que não podem ser otimizadas automaticamente pelo otimizador do SGBD, criando planos de execução personalizados e índices especializados.
- Implementação de controles de acesso e auditoria para conformidade PCI DSS: Configurar o banco de dados para atender aos requisitos de segurança do padrão PCI DSS, incluindo registro detalhado de atividades, controles de acesso rigorosos e criptografia de dados sensíveis.
- Desenvolvimento de uma solução de data masking para dados sensíveis: Implementar técnicas de mascaramento de dados em ambientes de desenvolvimento e teste, garantindo que dados confidenciais não sejam expostos.
- Implementação de um sistema de replicação heterogênea entre diferentes SGBDs: Configurar replicação de dados entre bancos de dados de diferentes fornecedores (por exemplo, Oracle e PostgreSQL), incluindo transformação de dados e resolução de incompatibilidades.
- Projeto e implementação de uma base de dados temporal: Configurar o banco de dados para armazenar e consultar dados com validades temporais, permitindo análises históricas e previsões.

- Desenvolvimento de um sistema de gerenciamento de metadados corporativo: Implementar um repositório central de metadados que documenta o esquema do banco de dados, linhagem de dados, definições e outras informações relevantes para a governança de dados.
- Implementação de lógica de negócios distribuída entre banco de dados e aplicação: Decidir e implementar quais partes da lógica de negócios devem ser executadas no banco de dados (procedures, triggers) e quais na aplicação, garantindo performance e manutenção.
- Desenvolvimento de uma solução para armazenamento e consulta de big data: Integrar o banco de dados com soluções de big data, como Hadoop ou Spark, para armazenar e processar grandes volumes de dados não estruturados ou semiestruturados.
- Implementação de um sistema de versionamento de dados: Configurar o banco de dados para manter versões históricas dos registros, permitindo rastreamento de mudanças e auditoria completa.
- Projeto de um banco de dados geoespacial com dados complexos: Implementar funcionalidades geoespaciais avançadas, incluindo armazenamento de geometria complexa, indexação espacial e realização de consultas geográficas sofisticadas.
- Desenvolvimento de um sistema de autorização baseado em atributos (ABAC): Implementar controles de acesso avançados que utilizam atributos dos usuários e dos dados para determinar permissões, diretamente no nível do banco de dados.
- Implementação de um sistema de análise preditiva dentro do banco de dados: Utilizar capacidades de machine learning e estatística avançada do SGBD para realizar análises preditivas diretamente no banco de dados.
- Desenvolvimento de um mecanismo de sincronização de dados em ambientes offline: Implementar soluções que permitem que dados sejam sincronizados entre o banco de dados central e bases locais ou dispositivos móveis que operam offline.
- Projeto e implementação de um banco de dados poliglota: Integrar diferentes tipos de bancos de dados (relacionais, NoSQL, grafos) em uma única aplicação, escolhendo o tipo adequado para cada caso de uso.
- Implementação de um sistema de monitoramento de integridade de dados: Configurar processos que verificam regularmente a integridade dos dados, identificando e corrigindo inconsistências ou corrupção.
- Desenvolvimento de soluções para gerenciamento de dados hierárquicos e grafos: Implementar modelos de dados e consultas eficientes para representar e navegar em estruturas de grafos complexas.

- Otimização do desempenho de consultas em bancos de dados massivamente paralelos: Ajustar configurações e estruturas para aproveitar o processamento paralelo em SGBDs que suportam essa funcionalidade.
- Implementação de estratégias de arquivamento de dados históricos: Desenvolver processos que movem dados antigos para sistemas de arquivamento, mantendo a possibilidade de acesso quando necessário e liberando recursos no banco principal.
- Desenvolvimento de uma solução de armazenamento e consulta de dados JSON/XML complexos: Implementar técnicas para trabalhar eficientemente com dados JSON ou XML complexos, incluindo indexação, validação e consultas avançadas.
- Implementação de técnicas de compressão e compactação de dados avançadas: Configurar o banco de dados para utilizar compressão de dados avançada, reduzindo significativamente o uso de armazenamento sem degradar a performance.
- Projeto de uma arquitetura de banco de dados orientada a microserviços: Dividir o banco de dados monolítico em bancos de dados separados para cada microserviço, definindo estratégias de comunicação e consistência entre eles.
- Desenvolvimento de processos de ETL complexos com transformação avançada: Implementar processos de ETL que envolvem regras de transformação complexas, limpeza de dados e integração de múltiplas fontes heterogêneas.
- Implementação de controles de concorrência e isolamento em transações complexas: Configurar níveis de isolamento apropriados e implementar mecanismos para evitar problemas como deadlocks, garantindo a consistência dos dados.
- Desenvolvimento de um sistema de auditoria completo com trilha de auditoria: Implementar um sistema que registra todas as operações realizadas no banco de dados, incluindo quem fez, o que fez, quando e como, atendendo a requisitos de conformidade rigorosos.
- Implementação de um sistema de gestão de chaves para criptografia: Configurar um sistema de gerenciamento de chaves (KMS) para controlar a criptografia de dados no banco, incluindo rotação de chaves e acesso seguro.
- Desenvolvimento de um mecanismo de indexação personalizada: Implementar índices personalizados para tipos de dados ou consultas específicas que não são suportados pelos índices padrão do SGBD.

- Projeto e implementação de uma estratégia de escalabilidade vertical e horizontal: Planejar como o banco de dados irá escalar tanto verticalmente (melhor hardware) quanto horizontalmente (mais servidores), incluindo estratégias para distribuição de carga e replicação.
- Implementação de soluções para compliance com LGPD/GDPR: Configurar o banco de dados para atender aos requisitos legais de proteção de dados, incluindo direito ao esquecimento, portabilidade de dados e consentimento.
- Desenvolvimento de um sistema de monitoramento de performance com ajuste automático: Implementar ferramentas que monitoram o desempenho do banco de dados e ajustam automaticamente parâmetros de configuração para otimização.
- Implementação de lógica para suporte a transações distribuídas: Configurar o banco de dados para suportar transações que abrangem múltiplos bancos de dados ou sistemas, garantindo atomicidade e consistência.
- Projeto e implementação de um sistema de controle de acesso baseado em papéis (RBAC) avançado: Configurar um sistema de controle de acesso complexo, definindo papéis, permissões e hierarquias de acesso no nível do banco de dados.
- Desenvolvimento de uma solução de migração de dados com zero downtime: Planejar e executar a migração de dados ou esquema do banco de dados sem interromper a disponibilidade do sistema para os usuários finais.
- Implementação de um sistema de processamento de eventos complexos (CEP): Configurar o banco de dados para detectar e reagir a padrões de eventos complexos em tempo real, utilizando funcionalidades específicas ou integrando com sistemas CEP externos.

Testes e Garantia de Qualidade

O Testes e Garantia de Qualidade é um macro assunto essencial no desenvolvimento de software, responsável por garantir que a aplicação funcione conforme o esperado, sem erros e com a melhor experiência possível para o usuário. Esse processo envolve a criação de casos de teste, execução de testes manuais e automatizados, e a validação de funcionalidades e fluxos críticos da aplicação, visando minimizar falhas e garantir que as entregas sejam confiáveis e estáveis.

No contexto de desenvolvimento, os testes de qualidade estão interligados com todos os outros macros assuntos, como Frontend, Backend e Integração de APIs. Por exemplo, os testes de interface no Frontend verificam se os componentes visuais estão funcionando corretamente em diferentes dispositivos e navegadores. Já os testes no Backend garantem que a lógica de negócios esteja correta e que as respostas das APIs estejam de acordo com os requisitos da aplicação.

O processo de garantia de qualidade também inclui testes de performance, segurança e usabilidade, com o objetivo de assegurar que a aplicação atenda aos padrões de mercado. Os testes automatizados, como os testes unitários, de integração e de regressão, são usados para verificar se as alterações no código não introduzem novos problemas e para garantir que funcionalidades críticas continuem operando conforme o esperado. Testes de stress e carga, por sua vez, simulam grandes volumes de usuários e interações para identificar possíveis gargalos de performance.

O esforço de Story Points para as tarefas de Testes e Garantia de Qualidade varia conforme a complexidade dos testes. Tarefas simples, como a criação de testes unitários básicos, são classificadas com 1 Story Point, enquanto atividades mais complexas, como testes de integração entre módulos críticos ou execução de testes de desempenho sob alta carga, podem exigir até 13 Story Points, devido à necessidade de planejamento, automação e análises detalhadas de resultados.

Em resumo, Testes e Garantia de Qualidade é um macro assunto transversal a todo o ciclo de desenvolvimento, garantindo que as entregas sejam consistentes, de alta qualidade, e atendam aos padrões esperados em termos de funcionalidade, performance e segurança.

1 (um) Story Point

Tarefas de 1 Story Point em Testes e Garantia de Qualidade são atividades de baixa complexidade e de curta duração, focadas em testes simples, manuais ou automáticos, que validam funcionalidades básicas da aplicação. Essas tarefas não envolvem a criação de lógica complexa ou a preparação extensa de ambientes, sendo adequadas para verificações pontuais que garantem a integridade do sistema sem exigir um grande esforço técnico.

Um exemplo de tarefa de 1 Story Point seria a escrita de um teste unitário simples para uma função existente. Neste caso, o teste visa garantir que uma função retorna o valor esperado para uma entrada específica, validando comportamentos básicos. Outra atividade comum é a execução de um conjunto

de testes manuais pré-definidos, seguindo um roteiro simples para verificar funcionalidades básicas, como o comportamento de uma página web ou a exibição de mensagens de erro.

Além disso, tarefas de 1 Story Point podem incluir a verificação de layout e elementos de interface, conferindo se os elementos estão posicionados corretamente e funcionando como esperado. Outras atividades incluem a validação de mensagens de erro exibidas ao usuário, conferindo se elas aparecem de forma apropriada quando uma ação incorreta é realizada. Essas atividades são essenciais para garantir que a qualidade geral do software seja mantida, sem exigir grande esforço de implementação ou de análise.

Exemplos:

- Escrita de um teste unitário simples para uma função existente: Criar um teste que verifica se uma função retorna o resultado esperado para uma entrada específica.
- Atualização de um teste unitário com novos casos de entrada: Adicionar casos de teste a um conjunto existente para cobrir cenários adicionais simples.
- Execução de um conjunto de testes manuais pré-definidos: Realizar testes manuais seguindo um roteiro simples para verificar funcionalidades básicas.
- Revisão de casos de teste escritos por outros: Ler e fornecer feedback sobre casos de teste simples criados por colegas, garantindo clareza e cobertura.
- Correção de pequenos bugs encontrados durante testes: Ajustar o código para resolver problemas menores identificados durante a execução de testes.
- Configuração de dados de teste simples no banco de dados: Inserir registros de teste necessários para a execução de casos específicos.
- Atualização de documentação de teste com informações recentes: Modificar documentos para refletir mudanças pequenas nas funcionalidades testadas.
- Execução de testes de regressão automatizados: Rodar testes automatizados existentes para garantir que novas alterações não afetaram funcionalidades anteriores.
- Validação de mensagens de erro exibidas ao usuário: Verificar se as mensagens de erro são apresentadas corretamente em situações simples.

- Verificação de layout e elementos de interface: Confirmar que elementos da interface estão posicionados corretamente e funcionam conforme esperado.
- Criação de um script de teste automatizado básico: Desenvolver um script simples que automatiza a interação com uma funcionalidade específica.
- Reportar bugs com informações básicas: Registrar defeitos encontrados com descrição, passos para reproduzir e evidências simples.
- Atualização de ambientes de teste com a versão mais recente do software: Instalar a última versão para garantir que os testes sejam executados no ambiente atualizado.
- Verificação de conformidade com padrões de codificação em pequenos trechos: Revisar código para assegurar aderência a padrões em partes específicas.
- Execução de testes de desempenho simples: Medir tempos de resposta para funcionalidades básicas sem carga pesada.
- Participação em reuniões de planejamento de testes: Contribuir com insights simples sobre o escopo e priorização de testes.
- Criação de checklists para testes manuais: Elaborar listas de verificação que auxiliam na execução de testes manuais simples.
- Validação de links e navegação em páginas web: Testar se os links levam às páginas corretas e se a navegação é funcional.
- Testes de compatibilidade em um único navegador ou dispositivo: Verificar se a aplicação funciona corretamente em um navegador ou dispositivo específico.
- Atualização de scripts de teste automatizado devido a pequenas mudanças: Ajustar scripts existentes para refletir alterações mínimas na interface ou fluxo.
- Configuração de propriedades de um teste automatizado: Ajustar parâmetros como tempos de espera ou variáveis de ambiente para um teste específico.
- Execução de testes de segurança básicos: Verificar se inputs simples são tratados corretamente para evitar vulnerabilidades comuns.
- Validação de formulários com dados válidos e inválidos: Testar se o sistema aceita dados corretos e rejeita dados incorretos em campos de formulário.
- Revisão de logs de teste para identificar erros: Analisar logs gerados durante testes para identificar e reportar problemas simples.
- Criação de casos de teste para requisitos triviais: Desenvolver casos de teste que cobrem requisitos simples e bem definidos.
- Atualização de planilhas de rastreamento de testes: Registrar o status dos testes executados, marcando-os como passados ou falhados.

- Participação em sessões de testes exploratórios rápidos: Realizar testes sem roteiro pré-definido para descobrir comportamentos inesperados em áreas simples.
- Verificação de traduções e internacionalização em uma língua: Conferir se as traduções estão corretas em um idioma suportado pela aplicação.
- Teste de campos de entrada para limites conhecidos: Inserir valores nos limites mínimos e máximos permitidos para verificar o comportamento.
- Validação de uploads e downloads de arquivos simples: Testar se o sistema permite carregar e baixar arquivos conforme esperado.
- Execução de testes unitários após pequenas alterações no código: Rodar testes existentes para garantir que mudanças não introduziram novos erros.
- Verificação de conformidade com padrões de acessibilidade básicos: Testar se a aplicação atende a requisitos simples de acessibilidade.
- Configuração de ambiente de teste local: Preparar o ambiente local para executar testes, instalando dependências necessárias.
- Reportar resultados de testes para a equipe: Comunicar de forma breve os resultados dos testes realizados, destacando pontos relevantes.
- Criação de cenários de teste para validação de regras de negócio simples: Elaborar cenários que confirmem o funcionamento correto de regras básicas.
- Atualização de testes automatizados com pequenos ajustes no fluxo: Adaptar scripts para refletir alterações menores na sequência de ações da aplicação.
- Verificação de conteúdos estáticos em páginas web: Conferir textos, imagens e outros conteúdos estáticos para garantir que estão corretos.
- Teste de notificações e alertas simples: Validar se mensagens de notificação são exibidas nos momentos apropriados.
- Validação de emails enviados pela aplicação: Testar se emails são disparados corretamente e contêm as informações esperadas.
- Revisão de políticas de qualidade e procedimentos: Ler documentos de qualidade para entender e aderir aos processos estabelecidos.
- Execução de testes funcionais em APIs com chamadas simples: Realizar requisições básicas para APIs e verificar as respostas obtidas.
- Atualização de suites de teste com novos casos triviais: Adicionar casos simples a conjuntos de testes existentes para melhorar a cobertura.
- Teste de funcionalidade de pesquisa com termos simples: Inserir termos de busca comuns e verificar se os resultados são adequados.
- Validação de funcionalidades após correção de bugs: Testar áreas afetadas por correções recentes para garantir que o problema foi resolvido.

- Criação de dados de teste para cenários específicos: Preparar dados necessários para executar casos de teste simples.
- Verificação de formatos de data, hora e números: Conferir se os formatos exibidos estão corretos e seguem os padrões esperados.
- Teste de funcionalidade de login e logout: Validar se o usuário consegue acessar e sair da aplicação corretamente.
- Atualização de testes devido a mudanças em dependências externas: Ajustar casos de teste para refletir alterações em APIs ou serviços utilizados.
- Validação de perfis de usuário com permissões básicas: Testar se usuários com diferentes níveis de acesso veem as funcionalidades corretas.
- Execução de testes em diferentes resoluções de tela: Verificar se a aplicação se adapta adequadamente a diferentes tamanhos de tela.
- Verificação de redirecionamentos simples: Testar se URLs redirecionam para as páginas corretas quando necessário.
- Revisão de scripts de teste para melhorias simples: Analisar e melhorar scripts existentes para torná-los mais eficientes ou claros.

5 (cinco) Story Points

Tarefas de 5 Story Points no contexto de Testes e Garantia de Qualidade envolvem atividades de teste mais elaboradas que exigem uma compreensão mais profunda da aplicação e suas funcionalidades. Embora ainda sejam de complexidade moderada, essas tarefas podem incluir automação de testes simples e a execução de testes que requerem uma preparação adicional, além de maior atenção aos detalhes para garantir que os cenários de teste cubram adequadamente as regras de negócio e as funcionalidades da aplicação.

Um exemplo comum seria o desenvolvimento de testes unitários para funções com lógica condicional. Esse tipo de tarefa exige a criação de casos de teste que cobrem diferentes caminhos de execução em funções que contêm condicionais, laços ou tratamentos de erro. A complexidade reside na necessidade de garantir que todos os possíveis resultados sejam validados adequadamente.

Tarefas de 5 Story Points podem incluir a automação de testes funcionais básicos, onde são desenvolvidos scripts que replicam interações do usuário com a interface da aplicação. Isso pode envolver testes de navegação, envio de formulários ou validação de mensagens exibidas ao usuário. Além disso, também pode incluir a execução de testes de integração simples, que verificam a interação entre dois ou mais componentes ou módulos do sistema, assegurando que funcionam corretamente em conjunto.

Exemplos:

- Desenvolvimento de testes unitários para funções com lógica condicional: Escrever testes que cobrem diferentes caminhos de execução em funções que possuem condicionais, loops ou tratamentos de erro.
- Automação de testes funcionais básicos: Criar scripts automatizados que replicam interações do usuário com a aplicação, validando funcionalidades principais.
- Execução de testes de integração simples: Testar a interação entre dois ou mais componentes ou módulos do sistema para verificar se funcionam corretamente em conjunto.
- Criação de casos de teste para cenários de borda: Desenvolver casos que testam os limites das funcionalidades, como valores máximos e mínimos, inputs vazios ou formatos inesperados.
- Implementação de testes de API para endpoints RESTful: Escrever testes que enviam requisições HTTP a APIs e verificam as respostas, incluindo códigos de status e conteúdo retornado.
- Atualização de suites de teste automatizado com novos casos complexos: Adicionar casos que cobrem novas funcionalidades ou cenários não testados anteriormente.
- Execução de testes de desempenho com carga moderada: Testar como a aplicação se comporta sob uma carga maior de usuários ou transações, identificando possíveis gargalos.
- Configuração de ambientes de teste automatizado: Preparar ambientes que permitem a execução de testes automatizados, incluindo instalação de ferramentas e configurações necessárias.
- Análise de resultados de testes automatizados e identificação de falhas: Revisar logs e relatórios de testes para identificar falhas, determinar causas e propor correções.
- Revisão e atualização de planos de teste: Atualizar documentos que descrevem o escopo, abordagem, recursos e cronograma das atividades de teste.
- Desenvolvimento de testes para validação de regras de negócio complexas: Escrever casos que confirmam se as regras de negócio estão sendo aplicadas corretamente em situações diversas.

- Execução de testes de compatibilidade em múltiplos navegadores ou dispositivos: Verificar se a aplicação funciona corretamente em diferentes ambientes, como vários navegadores ou sistemas operacionais.
- Automatização de testes de regressão para funcionalidades críticas: Criar scripts que verificam automaticamente se funcionalidades importantes continuam operando após alterações no código.
- Implementação de testes de segurança básicos: Testar vulnerabilidades comuns como injeção de SQL, cross-site scripting (XSS) ou falhas de autenticação.
- Criação de dados de teste complexos: Preparar conjuntos de dados que permitem testar cenários específicos ou funcionalidades avançadas da aplicação.
- Execução de testes exploratórios focados em áreas críticas: Realizar testes sem roteiro pré-definido, explorando funcionalidades complexas para descobrir defeitos não identificados.
- Desenvolvimento de mocks ou stubs para isolamento de testes: Criar componentes simulados que permitem testar módulos específicos sem dependências externas.
- Atualização de scripts de teste automatizado devido a mudanças significativas: Adaptar scripts existentes para refletir alterações importantes na interface ou lógica da aplicação.
- Configuração de ferramentas de gerenciamento de defeitos: Implementar e configurar sistemas que permitem registrar, rastrear e gerenciar bugs e falhas encontradas.
- Participação em revisões de código focadas na qualidade: Analisar o código desenvolvido por outros com foco em padrões de qualidade, possíveis erros e aderência a boas práticas.
- Implementação de testes de usabilidade básicos: Realizar testes que avaliam a facilidade de uso da aplicação, identificando áreas que podem confundir os usuários.
- Execução de testes em ambientes com configurações específicas: Testar a aplicação em ambientes com configurações particulares, como diferentes versões de dependências ou configurações de rede.
- Desenvolvimento de scripts para automação de testes de API: Criar scripts que automatizam o envio de requisições e verificação de respostas em APIs complexas.
- Análise de cobertura de testes e identificação de lacunas: Utilizar ferramentas que medem a cobertura dos testes e identificar partes do código que não estão sendo testadas.

- Execução de testes de recuperação de falhas simples: Verificar como a aplicação se comporta em situações de erro, como perda de conexão ou falta de recursos.
- Criação de casos de teste para fluxo de processos complexos: Desenvolver casos que cobrem sequências de ações que o usuário pode realizar, incluindo variações e desvios.
- Implementação de testes de banco de dados básicos: Escrever testes que verificam operações de CRUD (Create, Read, Update, Delete) e integridade dos dados.
- Participação em reuniões de análise de riscos: Contribuir para a identificação e priorização de riscos relacionados à qualidade e definir estratégias de mitigação.
- Configuração de pipelines de integração contínua para execução de testes: Ajustar ferramentas de CI/CD para que os testes sejam executados automaticamente em cada build.
- Desenvolvimento de testes para verificar conformidade com requisitos legais ou regulatórios: Escrever casos que confirmam se a aplicação atende a normas específicas.
- Execução de testes de internacionalização e localização: Verificar se a aplicação funciona corretamente em diferentes idiomas e formatos regionais.
- Implementação de testes A/B básicos: Configurar e executar testes que comparam duas versões da aplicação para determinar qual oferece melhor desempenho ou usabilidade.
- Análise de logs do sistema para identificar problemas ocultos: Revisar logs detalhados em busca de erros ou comportamentos anômalos não evidentes durante os testes.
- Desenvolvimento de testes de carga simples: Configurar e executar testes que simulam um número crescente de usuários para avaliar o comportamento da aplicação.
- Criação de cenários de teste para permissões e roles de usuário: Testar se diferentes tipos de usuários têm acesso correto às funcionalidades conforme suas permissões.
- Implementação de testes de acessibilidade avançados: Utilizar ferramentas e técnicas para verificar se a aplicação atende a padrões de acessibilidade mais rigorosos.
- Execução de testes de integração com sistemas externos: Testar como a aplicação interage com APIs ou serviços de terceiros, incluindo manejo de erros e tempo de resposta.

- Desenvolvimento de scripts para testes de interface com verificação visual: Automatizar testes que verificam não apenas a funcionalidade, mas também a aparência dos elementos de interface.
- Participação em sessões de pareamento para testes: Trabalhar em conjunto com desenvolvedores ou outros testadores para criar e executar casos de teste.
- Configuração de ambientes de teste com dados em massa: Preparar ambientes que contêm grandes volumes de dados para testar performance e escalabilidade.
- Implementação de testes de segurança focados em autenticação e autorização: Verificar se os mecanismos de login, senha, tokens e permissões estão funcionando e são seguros.
- Criação de relatórios de teste detalhados: Documentar os resultados dos testes com informações abrangentes, incluindo métricas, gráficos e análises.
- Atualização de documentação de teste para refletir mudanças significativas: Revisar e atualizar planos, casos e scripts de teste após alterações importantes na aplicação.
- Execução de testes de instalação e configuração: Testar se a aplicação pode ser instalada e configurada corretamente em diferentes ambientes.
- Desenvolvimento de testes para processos assíncronos ou em background: Escrever casos que verificam funcionalidades que ocorrem em segundo plano ou de forma não imediata.
- Análise de requisitos para identificar casos de teste adicionais: Revisar requisitos e especificações para descobrir cenários de teste que não foram previamente considerados.
- Implementação de testes para verificar integridade de dados após migração: Testar se dados migrados entre sistemas ou versões mantêm a integridade e consistência.
- Execução de testes de stress básicos: Colocar a aplicação sob carga extrema para identificar pontos de falha ou limitações.
- Desenvolvimento de scripts para automação de testes em dispositivos móveis: Criar scripts que automatizam testes em aplicações móveis ou responsivas.
- Participação em retrospectivas focadas na qualidade: Contribuir com feedback sobre o processo de teste e sugestões para melhorias futuras.
- Configuração de testes parametrizados ou data-driven: Escrever testes que executam o mesmo cenário com diferentes conjuntos de dados para aumentar a cobertura.

- Implementação de testes de usabilidade com usuários reais: Organizar e conduzir testes onde usuários reais interagem com a aplicação e fornecem feedback.
- Desenvolvimento de planos de teste para novas funcionalidades: Elaborar documentos que detalham a abordagem, escopo e recursos necessários para testar novas features.
- Execução de testes de compatibilidade com versões antigas de software: Verificar se a aplicação mantém compatibilidade com versões anteriores de navegadores, sistemas operacionais ou dependências.
- Análise de desempenho e identificação de gargalos: Utilizar ferramentas de profiling para identificar partes do código que impactam negativamente a performance.
- Implementação de testes de regressão visual: Utilizar ferramentas que comparam imagens da interface ao longo do tempo para detectar mudanças não intencionais.
- Criação de ambientes de teste isolados usando contêineres ou máquinas virtuais: Configurar ambientes independentes que não interferem com outros sistemas ou testes.

13 (treze) Story Point

Tarefas de 13 Story Points em Testes e Garantia de Qualidade envolvem uma complexidade moderada a alta, exigindo um esforço significativo em termos de planejamento e execução. Essas tarefas incluem a criação de testes automatizados que cobrem cenários complexos, testes de integração entre múltiplos componentes ou a implementação de testes que demandam uma compreensão aprofundada da arquitetura do sistema. Tarefas desse nível podem requerer coordenação com outras equipes e uma análise detalhada das funcionalidades e do comportamento da aplicação.

Um exemplo típico seria o desenvolvimento de testes de integração abrangentes para módulos críticos. Esses testes verificam a interação entre diferentes componentes ou serviços, garantindo que funcionem corretamente em conjunto. A complexidade reside na necessidade de cobrir fluxos completos da aplicação, onde múltiplos sistemas interagem.

Outro exemplo seria a implementação de testes automatizados avançados para interfaces gráficas, que envolvem a criação de scripts que simulam interações complexas do usuário, como navegação, manipulação de dados e validação de resultados em diferentes estados da aplicação. Essas tarefas exigem uma boa

compreensão das interações de interface e das diferentes situações que o usuário pode encontrar.

Além disso, tarefas de 13 Story Points podem incluir testes de carga e desempenho, simulando comportamentos reais de usuários para avaliar como o sistema se comporta sob condições de alta demanda. Isso exige a configuração de ambientes de teste que replicam a infraestrutura de produção e a análise detalhada dos resultados para identificar gargalos de performance e otimizar o sistema.

Exemplos:

- Desenvolvimento de testes de integração abrangentes para módulos críticos: Escrever testes que verificam a interação entre múltiplos componentes ou serviços, cobrindo fluxos complexos e garantindo que funcionem corretamente em conjunto.
- Implementação de testes automatizados de interface gráfica avançados: Criar scripts que automatizam interações complexas na interface do usuário, incluindo navegação, manipulação de dados e validação de resultados em diferentes estados da aplicação.
- Execução de testes de carga e desempenho com simulação de usuários reais: Configurar e executar testes que simulam comportamentos reais dos usuários, avaliando como a aplicação se comporta sob condições de alta carga.
- Desenvolvimento de um framework de testes personalizado: Criar ou estender frameworks existentes para atender às necessidades específicas do projeto, facilitando a criação e manutenção de testes.
- Implementação de testes de segurança avançados, incluindo testes de penetração: Realizar análises de segurança detalhadas, identificando vulnerabilidades e explorando potenciais falhas para melhorar a robustez do sistema.
- Criação de casos de teste para cenários de falhas complexas: Desenvolver casos que testam como o sistema se comporta em situações de erro não triviais, como perda de conexão com serviços externos, falhas de hardware ou corrupção de dados.
- Automação de testes para fluxos de negócios críticos e complexos: Criar scripts que automatizam a validação de processos de negócios essenciais que envolvem múltiplos passos e decisões condicionais.
- Execução de testes de compatibilidade em múltiplos ambientes e plataformas: Testar a aplicação em diversas combinações de sistemas

operacionais, navegadores, dispositivos e configurações para garantir ampla compatibilidade.

- Configuração e manutenção de ambientes de teste complexos: Preparar ambientes de teste que replicam a infraestrutura de produção, incluindo bancos de dados, servidores de aplicação e serviços externos.
- Análise de cobertura de código e melhoria de testes para aumentar a cobertura: Utilizar ferramentas para identificar partes do código não cobertas pelos testes e desenvolver novos casos para aumentar a cobertura geral.
- Desenvolvimento de testes de integração contínua com feedback imediato: Configurar pipelines que executam testes automaticamente a cada mudança no código, fornecendo feedback rápido sobre a qualidade das alterações.
- Implementação de testes de recuperação de desastres: Testar a capacidade do sistema de se recuperar de falhas graves, incluindo restauração de backups, failover e recuperação de dados.
- Criação de testes de usabilidade com análise detalhada: Conduzir testes de usabilidade com usuários reais, coletando dados qualitativos e quantitativos, e analisando os resultados para identificar áreas de melhoria.
- Execução de testes de desempenho em cenários de pico de uso: Simular condições extremas de uso, como eventos promocionais ou campanhas de marketing, para avaliar se a aplicação suporta o aumento repentino de carga.
- Desenvolvimento de testes para validar integrações com sistemas de terceiros complexos: Escrever testes que verificam a correta interação com APIs e serviços externos que possuem comportamentos complexos ou não determinísticos.
- Implementação de testes para processos assíncronos e filas de mensagens: Criar casos que verificam o funcionamento correto de processos que ocorrem em segundo plano ou que dependem de sistemas de mensageria.
- Automação de testes para aplicações móveis ou multiplataforma: Desenvolver scripts que automatizam testes em dispositivos móveis ou aplicações que rodam em múltiplas plataformas, considerando as particularidades de cada ambiente.
- Análise e otimização de desempenho com profiling detalhado: Utilizar ferramentas avançadas de profiling para identificar gargalos de performance no código e propor otimizações.

- Desenvolvimento de testes de regressão visual avançados: Implementar ferramentas que detectam mudanças visuais sutis na interface, garantindo que atualizações não afetem negativamente o design ou a usabilidade.
- Configuração de testes de segurança contínuos: Integrar ferramentas de análise de vulnerabilidades ao processo de desenvolvimento, executando verificações regulares e automatizadas.
- Criação de testes para validar algoritmos complexos ou cálculos críticos: Escrever casos que confirmam a precisão e corretude de algoritmos que realizam cálculos matemáticos, financeiros ou lógicos complexos.
- Execução de testes de conformidade com padrões e regulamentações específicos: Verificar se a aplicação atende a normas técnicas, legais ou de mercado, como PCI DSS, HIPAA ou ISO 27001.
- Desenvolvimento de testes para verificar a escalabilidade horizontal da aplicação: Testar se o sistema pode ser expandido adicionando novos recursos de hardware ou instâncias, mantendo ou melhorando o desempenho.
- Implementação de testes para sistemas distribuídos ou microserviços: Escrever casos que verificam a interação entre múltiplos serviços independentes, incluindo comunicação, sincronização e tolerância a falhas.
- Automação de testes para pipelines de dados e processos ETL: Criar scripts que validam a integridade e consistência de dados através de processos de extração, transformação e carregamento.
- Análise de logs e métricas em ambientes de produção para identificar problemas: Monitorar e analisar dados reais de uso para detectar comportamentos anômalos ou tendências preocupantes.
- Desenvolvimento de testes para garantir consistência em bases de dados distribuídas: Verificar se transações e operações em bancos de dados distribuídos mantêm a consistência e integridade dos dados.
- Implementação de testes de acessibilidade avançados conforme WCAG 2.1: Garantir que a aplicação atenda a critérios rigorosos de acessibilidade, beneficiando usuários com diferentes necessidades.
- Execução de testes de stress prolongados (soak testing): Testar a aplicação sob carga normal ou alta por um período estendido para identificar problemas que surgem ao longo do tempo, como vazamentos de memória.
- Desenvolvimento de cenários de teste para recuperação e continuidade de negócios: Planejar e executar testes que verificam se a aplicação pode continuar operando durante e após situações de crise.

- Implementação de testes para validar internacionalização completa: Testar a aplicação em múltiplos idiomas, culturas e fusos horários, incluindo scripts de escrita complexos e direções de texto diferentes.
- Automação de testes para sistemas com inteligência artificial ou machine learning: Criar testes que validam modelos preditivos, garantindo que funcionem corretamente e permaneçam precisos após atualizações.
- Análise de segurança com testes de fuzzing: Utilizar técnicas de fuzzing para enviar entradas inesperadas ou malformadas, identificando possíveis falhas ou vulnerabilidades.
- Desenvolvimento de testes para garantir a conformidade com requisitos de desempenho em contratos de SLA: Verificar se a aplicação atende aos níveis de serviço acordados, como tempos de resposta ou disponibilidade.
- Implementação de testes para verificar a correta implementação de padrões de design: Validar se o código segue padrões arquiteturais estabelecidos, como MVC, MVVM ou padrões específicos da organização.
- Criação de ambientes de teste automatizados usando infraestrutura como código: Utilizar ferramentas como Terraform ou Ansible para provisionar ambientes de teste de forma automática e reproduzível.
- Execução de testes de integração contínua em múltiplos branches e versões: Configurar sistemas que executam testes em diferentes versões do código simultaneamente, garantindo qualidade em todas as linhas de desenvolvimento.
- Desenvolvimento de testes para sistemas em tempo real ou com requisitos de baixa latência: Escrever casos que verificam se a aplicação responde dentro de limites de tempo estritos.
- Implementação de testes para validação de transações distribuídas: Testar operações que envolvem múltiplos sistemas ou bancos de dados, garantindo atomicidade e consistência.
- Análise de riscos e priorização de testes com base em impacto e probabilidade: Utilizar técnicas de análise de risco para determinar quais áreas devem ser testadas com mais rigor.
- Automação de testes para validação de compliance com LGPD/GDPR: Criar scripts que verificam se a aplicação lida corretamente com dados pessoais, incluindo consentimento, anonimização e direito ao esquecimento.
- Desenvolvimento de testes para aplicações que utilizam blockchain ou tecnologias distribuídas: Verificar se a aplicação interage corretamente com redes blockchain, incluindo validação de transações e contratos inteligentes.

- Implementação de testes de falhas em redes e infraestrutura: Simular problemas de rede, como latência, perda de pacotes ou desconexões, e verificar como a aplicação se comporta.
- Criação de testes de segurança para aplicações móveis, incluindo análise de código estático: Utilizar ferramentas e técnicas específicas para testar a segurança de aplicações em dispositivos móveis.
- Execução de testes de usabilidade com análise heurística: Avaliar a aplicação com base em princípios de usabilidade reconhecidos, identificando problemas de design ou experiência do usuário.
- Desenvolvimento de testes para validar políticas de segurança e permissões complexas: Escrever casos que verificam se regras de acesso e autorização são aplicadas corretamente em cenários diversos.
- Implementação de testes para verificar a resiliência da aplicação: Testar como o sistema se recupera de falhas internas, como exceções não tratadas ou recursos indisponíveis.
- Análise de impacto de mudanças e atualização de suites de teste: Avaliar como alterações no código ou requisitos afetam os testes existentes e atualizar ou criar casos conforme necessário.
- Automação de testes para aplicações com interfaces de linha de comando (CLI): Desenvolver scripts que interagem com aplicações baseadas em CLI, verificando a funcionalidade e saída.
- Execução de testes de compatibilidade com dispositivos IoT ou hardware específico: Testar a integração e funcionamento da aplicação com dispositivos físicos, sensores ou atuadores.
- Desenvolvimento de testes para verificar a escalabilidade vertical da aplicação: Testar se o sistema aproveita eficientemente recursos adicionais, como mais CPU ou memória, em um único servidor.
- Implementação de testes de segurança para verificar compliance com OWASP Top 10: Focar em identificar e corrigir as vulnerabilidades mais críticas conforme definido pela OWASP.
- Criação de testes para validar processos de atualização e migração de dados: Testar se a aplicação e os dados são atualizados corretamente entre versões, sem perda ou corrupção de informações.
- Automação de testes para validar integração contínua e entrega contínua (CI/CD): Garantir que todo o pipeline de desenvolvimento e deploy esteja funcionando corretamente, incluindo testes em cada etapa.
- Execução de testes de conformidade para padrões específicos da indústria: Verificar se a aplicação atende a normas como HL7 para saúde, SWIFT para finanças ou outros relevantes.

- Desenvolvimento de testes para sistemas com requisitos de alta disponibilidade: Testar cenários de failover, balanceamento de carga e recuperação automática para garantir disponibilidade contínua.
- Implementação de testes para validar o uso correto de serviços em nuvem: Verificar se a aplicação interage corretamente com serviços cloud, incluindo gerenciamento de recursos e conformidade com práticas recomendadas.
- Análise de dados de uso para identificar áreas de risco ou melhoria: Utilizar analytics para entender como os usuários interagem com a aplicação e identificar pontos que necessitam de atenção.
- Criação de planos de teste abrangentes para novos projetos ou módulos: Desenvolver documentos detalhados que cobrem todos os aspectos do teste, incluindo estratégia, recursos, cronograma e métricas.

Implementação de Funcionalidades de Segurança

A Implementação de Funcionalidades de Segurança aborda um conjunto específico de tarefas focadas em proteger a integridade e confidencialidade dos sistemas e dados. Ao contrário de outros macros assuntos que podem incluir aspectos de segurança de maneira mais ampla, este macro assunto é reservado para cenários onde são necessárias medidas de segurança especializadas e robustas, que não foram contempladas nas atividades regulares de desenvolvimento.

Neste contexto, considera-se que diversas medidas de segurança essenciais, como a validação de entradas, proteção contra injeção de código, e criptografia básica de dados, já foram aplicadas em outros macros assuntos, como Backend, APIs ou Frontend. Portanto, o foco aqui é em ações de segurança avançadas e específicas, destinadas a casos em que o sistema requer proteção adicional ou respostas a ameaças mais complexas.

As tarefas desta categoria variam em complexidade, indo de medidas de segurança moderadas, como autenticação multifator (2FA) e logs de auditoria, até a implementação de sistemas de criptografia avançada, proteção contra ataques distribuídos (DDoS), ou gestão de identidades com Single Sign-On (SSO). Além disso, inclui também configurações de segurança para APIs e microserviços, que demandam integração e análise mais detalhadas dos componentes de um sistema.

Devido à natureza crítica destas implementações, este macro assunto é utilizado em situações específicas onde a segurança exige soluções personalizadas e

medidas preventivas que vão além das práticas padrão já implementadas. Aqui, o planejamento e a integração cuidadosa com a arquitetura do sistema são essenciais para garantir a robustez das soluções propostas.

1 (um) Story Point

As tarefas de 1 Story Point em Implementação de Funcionalidades de Segurança são caracterizadas por sua simplicidade e baixo impacto na arquitetura geral do sistema. Essas atividades geralmente incluem ajustes menores ou a implementação de funcionalidades básicas de segurança que podem ser concluídas rapidamente, sem exigir um planejamento extenso ou mudanças significativas na estrutura existente.

Essas tarefas frequentemente envolvem ações como a atualização de pacotes para corrigir vulnerabilidades conhecidas, configuração de políticas de segurança simples, como limites para tentativas de login ou implementação de protocolos básicos como HTTPS em ambientes de desenvolvimento. O foco está em ajustes que melhoram a segurança sem afetar diretamente o desempenho ou a escalabilidade do sistema.

Essas atividades podem abranger verificações e configurações rápidas, como revisar permissões em arquivos ou ajustar headers HTTP para evitar ataques comuns. Embora sejam simples, essas tarefas são essenciais para manter boas práticas de segurança e garantir que a aplicação esteja protegida contra ameaças triviais.

Exemplos:

- Atualização de pacotes ou dependências para versões seguras: Atualizar bibliotecas ou frameworks para versões que corrigem vulnerabilidades conhecidas, garantindo que o sistema esteja protegido contra ameaças identificadas.
- Configuração de políticas básicas de senhas: Definir requisitos mínimos para senhas de usuários, como tamanho mínimo ou inclusão de caracteres especiais, para melhorar a segurança das autenticações.
- Implementação de bloqueio de conta após tentativas de login falhas: Configurar o sistema para bloquear temporariamente uma conta após um número específico de tentativas de login sem sucesso, prevenindo ataques de força bruta simples.

- Adição de validação de entrada básica em formulários: Implementar validações que impedem a inserção de dados inválidos ou potencialmente maliciosos em campos de entrada do usuário.
- Configuração de HTTPS no ambiente de desenvolvimento: Habilitar o protocolo HTTPS no ambiente local para garantir que as comunicações sejam criptografadas durante o desenvolvimento.
- Atualização de certificados SSL próximos da expiração: Renovar ou substituir certificados SSL que estão prestes a expirar, assegurando que as conexões seguras permaneçam válidas.
- Remoção de informações sensíveis do código-fonte: Eliminar chaves de API, senhas ou outros dados confidenciais que foram acidentalmente incluídos no código.
- Adição de headers de segurança básicos nas respostas HTTP: Incluir headers como X-Content-Type-Options ou X-Frame-Options para fortalecer a segurança contra ataques comuns.
- Configuração de regras simples de firewall de aplicação: Definir regras que bloqueiam tráfego indesejado ou não autorizado, protegendo a aplicação de acessos indevidos.
- Verificação de permissões em arquivos e diretórios: Ajustar as permissões do sistema de arquivos para garantir que somente usuários autorizados possam acessar ou modificar certos arquivos.
- Implementação de logout em todas as sessões ativas: Adicionar funcionalidade que permite ao usuário encerrar todas as sessões abertas, aumentando o controle sobre o acesso à conta.
- Atualização de políticas de privacidade e termos de uso: Revisar e atualizar documentos legais para refletir mudanças simples nas práticas de segurança ou coleta de dados.
- Configuração de mensagens de erro genéricas: Ajustar mensagens de erro para não revelar informações sensíveis sobre o sistema ou a infraestrutura.
- Implementação de restrições de tamanho de upload de arquivos: Definir limites para o tamanho de arquivos que podem ser enviados, prevenindo possíveis ataques de negação de serviço.
- Verificação de inputs para evitar injeção de SQL básica: Implementar medidas simples para sanitizar entradas do usuário, prevenindo tentativas de injeção de comandos SQL.
- Remoção de contas ou usuários inativos: Excluir contas que não são usadas há muito tempo, reduzindo a superfície de ataque potencial.
- Atualização de chaves ou tokens de acesso comprometidos: Substituir chaves de API ou tokens que possam ter sido expostos ou comprometidos.

- Configuração de timeout de sessão padrão: Definir um tempo limite após o qual sessões inativas são encerradas automaticamente.
- Adição de CAPTCHA em formulários públicos: Implementar um CAPTCHA básico em formulários para prevenir spam e atividades automatizadas maliciosas.
- Verificação de que logs não armazenam dados sensíveis: Revisar configurações de logging para garantir que informações confidenciais não sejam registradas em logs.

5 (cinco) Story Points

As tarefas de 5 Story Points em Implementação de Funcionalidades de Segurança abrangem configurações ou implementações que exigem um esforço moderado e que impactam áreas específicas do sistema, sem envolver uma reestruturação completa da segurança. Essas atividades podem incluir a adição de camadas extras de proteção ou a integração de funcionalidades que complementam as medidas de segurança existentes, exigindo um pouco mais de planejamento e tempo em comparação com tarefas muito simples.

Este tipo de tarefa pode incluir a implementação de autenticação de dois fatores (2FA), que requer integração com provedores de envio de códigos via e-mail ou SMS, ou a configuração de sistemas de logs para monitorar atividades críticas, como tentativas de login e alterações de configuração. Outras atividades envolvem a proteção contra-ataques de força bruta, configurando bloqueios temporários após falhas repetidas de login, e a validação de entradas para impedir injeções de SQL ou ataques Cross-Site Scripting (XSS).

As tarefas vão além de simples ajustes e exigem interação com múltiplos componentes do sistema, como banco de dados, APIs ou mecanismos de autenticação. No entanto, o impacto é localizado, sendo normalmente limitado a uma funcionalidade ou aspecto específico da segurança da aplicação.

Exemplos:

- Implementação de autenticação de dois fatores (2FA) simples: Adicionar uma camada extra de segurança ao processo de login, solicitando um código enviado por e-mail ou SMS após a inserção da senha.
- Configuração de proteção contra-ataques de força bruta: Implementar mecanismos que detectam e bloqueiam tentativas repetidas de login, como aumento progressivo do tempo de bloqueio após cada falha.

- Implementação de armazenamento seguro de senhas com hashing e salting: Modificar o sistema para armazenar senhas utilizando algoritmos de hash seguros e adicionando um salt único para cada usuário.
- Criação de um sistema básico de autorização por roles: Definir níveis de acesso baseados em papéis, permitindo que diferentes tipos de usuários tenham permissões distintas dentro da aplicação.
- Configuração de logs de segurança para atividades críticas: Registrar eventos importantes, como logins, falhas de autenticação e alterações de configuração, em logs seguros para auditoria.
- Implementação de validação de input para prevenção de injeção de SQL: Adicionar sanitização e validação de entradas do usuário para evitar que comandos maliciosos sejam executados no banco de dados.
- Configuração de CSP (Content Security Policy) básica: Definir políticas de segurança de conteúdo para evitar ataques de Cross-Site Scripting (XSS) e injeção de código.
- Implementação de proteção contra Cross-Site Request Forgery (CSRF): Adicionar tokens CSRF nos formulários e requisições para garantir que as ações sejam originadas de fontes confiáveis.
- Criação de um mecanismo de recuperação de senha seguro: Implementar um processo que permite ao usuário redefinir sua senha de forma segura, utilizando links temporários enviados por e-mail.
- Configuração de autenticação baseada em tokens JWT (JSON Web Tokens): Implementar autenticação utilizando tokens JWT, garantindo a segurança das sessões e facilidade de gerenciamento.
- Implementação de políticas de CORS (Cross-Origin Resource Sharing) adequadas: Configurar o servidor para controlar quais origens têm permissão para acessar recursos, prevenindo acesso não autorizado.
- Criação de um sistema de bloqueio de IP após tentativas maliciosas: Monitorar e bloquear endereços IP que demonstram comportamentos suspeitos, como múltiplas tentativas de acesso indevidas.
- Configuração de verificação de integridade de arquivos: Implementar mecanismos que verificam se os arquivos críticos do sistema não foram alterados ou corrompidos.
- Implementação de políticas de segurança para uploads de arquivos: Verificar extensões, tipos MIME e conteúdo de arquivos enviados pelos usuários para prevenir a inclusão de código malicioso.
- Configuração de alertas de segurança básicos: Configurar o sistema para enviar notificações aos administradores em caso de eventos suspeitos ou falhas de segurança.

- Implementação de logout automático após período de inatividade: Configurar o sistema para encerrar automaticamente a sessão do usuário após determinado tempo sem atividade.
- Adição de suporte a autenticação OAuth 2.0 com provedores externos: Permitir que os usuários façam login utilizando contas de serviços como Google ou Facebook, integrando de forma segura com a aplicação.
- Implementação de mascaramento de dados sensíveis na interface: Garantir que informações como números de cartão ou documentos sejam parcialmente ocultadas quando exibidas ao usuário.
- Configuração de ambiente seguro para desenvolvimento e teste: Estabelecer ambientes que simulam as configurações de segurança de produção, permitindo testes mais realistas.
- Implementação de verificação de segurança em dependências: Utilizar ferramentas que analisam bibliotecas e pacotes utilizados pelo sistema, identificando vulnerabilidades conhecidas.
- Criação de um processo de revisão de código focado em segurança: Estabelecer práticas onde o código é revisado com foco em identificar potenciais falhas de segurança antes do deploy.
- Implementação de criptografia de dados em trânsito: Configurar a aplicação para utilizar protocolos seguros (como HTTPS) para todas as comunicações, incluindo APIs e serviços internos.
- Configuração de gerenciamento de sessão segura: Atribuir identificadores de sessão seguros, impedir fixação de sessão e assegurar que sessões sejam invalidadas corretamente no logout.
- Implementação de validação de tamanho e formato de inputs: Garantir que entradas do usuário atendam a critérios específicos, prevenindo estouro de buffer e outros ataques baseados em inputs maliciosos.
- Criação de políticas de acesso baseadas em endereço IP ou geolocalização: Restringir o acesso a determinadas partes da aplicação com base na localização ou endereço IP do usuário.
- Configuração de segurança em serviços web (SOAP/WSDL): Implementar medidas de segurança específicas para serviços SOAP, incluindo assinaturas digitais e criptografia de mensagens.
- Implementação de controles para evitar Clickjacking: Utilizar headers como X-Frame-Options para impedir que a aplicação seja carregada em iframes de outros sites.
- Configuração de backups automatizados com segurança: Estabelecer procedimentos para backups regulares, garantindo que os dados estejam seguros e acessíveis apenas por pessoas autorizadas.

- Implementação de armazenamento seguro para chaves e credenciais: Utilizar cofres de segredo ou serviços de gerenciamento de chaves para armazenar informações sensíveis utilizadas pela aplicação.
- Adição de mecanismos de detecção de ataques de injeção de comandos: Implementar validações que previnem a execução de comandos do sistema operacional através de entradas do usuário.
- Configuração de políticas de segurança para APIs REST: Definir controles de acesso, limitação de taxa e autenticação adequada para endpoints de API.
- Implementação de proteção contra-ataques DoS/DDoS simples: Configurar mecanismos que detectam e limitam requisições excessivas de uma única fonte, prevenindo sobrecarga do sistema.
- Criação de procedimentos para atualização de patches de segurança: Estabelecer um cronograma e processo para aplicar atualizações de segurança regularmente nos sistemas e softwares utilizados.
- Configuração de verificação de certificados SSL/TLS: Assegurar que a aplicação verifica a validade e autenticidade dos certificados dos serviços com os quais se comunica.
- Implementação de política de retenção e descarte seguro de dados: Definir e aplicar regras para quanto tempo os dados são mantidos e como são eliminados de forma segura quando não mais necessários.
- Adição de suporte a logging estruturado para segurança: Registrar eventos de segurança em formato estruturado, facilitando a análise e detecção de padrões suspeitos.
- Configuração de ambiente seguro para execução de código não confiável: Utilizar sandboxing ou outras técnicas para isolar a execução de código potencialmente inseguro.
- Implementação de métodos para prevenir exposição de informações sensíveis em URLs: Evitar que dados confidenciais sejam passados como parâmetros de query string ou fragmentos de URL.
- Criação de páginas de erro personalizadas que não revelem detalhes do sistema: Desenvolver páginas de erro que informam o usuário sem fornecer informações técnicas que possam ser exploradas.
- Configuração de limitação de taxa (rate limiting) em APIs e serviços: Implementar controles que limitam o número de requisições permitidas em determinado período, prevenindo abusos.
- Implementação de mecanismos para prevenção de ataques de replay: Utilizar tokens únicos ou timestamps para assegurar que requisições não possam ser capturadas e repetidas.

- Adição de suporte a autenticação multifator para áreas críticas: Exigir múltiplos fatores de autenticação para acesso a funcionalidades sensíveis ou administrativas.
- Configuração de políticas de segurança para cookies: Definir atributos como HttpOnly, Secure e SameSite nos cookies utilizados pela aplicação para aumentar a segurança.
- Implementação de métodos para detecção de usuários bots ou automatizados: Utilizar técnicas como análise de comportamento ou desafios que distinguem humanos de bots.
- Criação de processos para gestão de vulnerabilidades: Estabelecer procedimentos para identificar, avaliar e remediar vulnerabilidades descobertas no sistema.
- Configuração de controles de acesso baseados em tempo: Restringir o acesso a determinadas funcionalidades ou dados durante horários específicos.
- Implementação de logs de auditoria para ações de usuários: Registrar atividades importantes realizadas pelos usuários, como alterações de dados ou configurações.
- Adição de verificações de integridade em arquivos transferidos: Utilizar checksums ou assinaturas digitais para garantir que arquivos enviados ou recebidos não foram corrompidos ou alterados.
- Configuração de monitoramento básico de segurança: Implementar ferramentas que monitoram o sistema em busca de atividades suspeitas ou anômalas.
- Implementação de procedimentos para gestão de incidentes de segurança: Definir passos a serem seguidos em caso de detecção de uma falha ou ataque, incluindo comunicação e mitigação.
- Criação de políticas de senha fortes com expiração e histórico: Exigir que os usuários criem senhas complexas, que expirem após certo período, e evitar a reutilização de senhas antigas.
- Configuração de métodos de criptografia para dados em repouso: Garantir que dados armazenados, especialmente informações sensíveis, estejam criptografados no banco de dados ou sistemas de arquivos.
- Implementação de mecanismos para validação de tokens de segurança: Verificar a validade, expiração e integridade de tokens utilizados na autenticação ou autorização.
- Adição de processos para garantir a conformidade com normas de segurança: Ajustar práticas e configurações para atender a requisitos de padrões como OWASP, PCI DSS ou outros relevantes.

13 (treze) Story Points

As tarefas de 13 Story Points na Implementação de Funcionalidades de Segurança envolvem um nível moderado de complexidade e exigem planejamento cuidadoso, pois afetam de forma significativa a segurança da aplicação. Essas atividades frequentemente incluem a integração de funcionalidades que requerem interações com múltiplos componentes do sistema e um impacto mais profundo na arquitetura de segurança.

Nesse nível, são implementadas medidas como a autenticação multifator (MFA) com aplicativos autenticadores, que envolve a configuração de tokens temporários e a integração com sistemas de autenticação externa. Além disso, a criptografia de dados sensíveis em repouso e em trânsito também é uma tarefa típica, exigindo a configuração de protocolos de segurança como TLS para comunicações e criptografia avançada no armazenamento.

Essas tarefas também podem incluir a implementação de proteção contra ataques distribuídos (DDoS) ou a configuração de um sistema de detecção de intrusões (IDS/IPS). Outra característica comum dessas tarefas é a necessidade de monitoramento contínuo e a coleta de métricas de segurança, como o desenvolvimento de um sistema de auditoria e monitoramento para identificar atividades suspeitas em tempo real.

As atividades de 13 Story Points demandam mais do que simples ajustes de segurança, envolvendo a modificação de processos críticos e a integração com serviços de segurança avançados, como autenticação baseada em certificados ou gerenciamento de chaves de criptografia.

Exemplos:

- Implementação de autenticação multifator com aplicativos autenticadores: Adicionar suporte para autenticação multifator utilizando aplicativos como Google Authenticator ou Authy, incluindo geração e verificação de códigos temporários (TOTP).
- Configuração de Single Sign-On (SSO) com protocolos SAML ou OpenID Connect: Integrar a aplicação com um provedor de identidade que utiliza SAML ou OpenID Connect, permitindo que usuários se autenticem usando suas credenciais corporativas.

- Implementação de criptografia de dados sensíveis em repouso e em trânsito: Configurar a criptografia de dados sensíveis tanto no banco de dados quanto durante a transmissão, utilizando técnicas como TLS para comunicações e criptografia de campo ou coluna no armazenamento.
- Desenvolvimento de um sistema de gerenciamento de chaves de criptografia: Implementar um sistema seguro para geração, armazenamento e rotação de chaves de criptografia, garantindo que apenas componentes autorizados possam acessar as chaves.
- Implementação de um sistema de detecção e prevenção de intrusões (IDS/IPS) básico: Integrar ferramentas que monitoram o tráfego e as atividades do sistema em busca de comportamentos suspeitos, alertando administradores ou bloqueando atividades maliciosas.
- Configuração de políticas avançadas de firewall de aplicação web (WAF): Implementar e ajustar regras de WAF para proteger a aplicação contra ataques como SQL Injection, Cross-Site Scripting (XSS) e outros tipos de ameaças.
- Implementação de autenticação baseada em certificados digitais: Configurar o sistema para aceitar e validar certificados digitais como método de autenticação, aumentando a segurança para usuários ou sistemas específicos.
- Desenvolvimento de um sistema de auditoria e monitoramento de segurança abrangente: Criar funcionalidades que registram detalhadamente eventos de segurança, acessos e atividades, permitindo análise e detecção de incidentes.
- Implementação de segurança em APIs com OAuth 2.0 e scopes de acesso: Configurar a autorização em APIs utilizando OAuth 2.0, definindo scopes para controlar o nível de acesso concedido a diferentes clientes ou aplicações.
- Configuração de ambientes seguros para desenvolvimento, teste e produção, incluindo isolamento de redes: Garantir que cada ambiente tenha configurações de segurança adequadas, isolando redes e controlando acesso para prevenir vazamentos ou ataques.
- Implementação de proteção contra ataques de Cross-Site Scripting (XSS) avançados: Adicionar medidas para detectar e neutralizar tentativas de injeção de scripts em entradas do usuário, utilizando técnicas de codificação e validação.
- Configuração de políticas de segurança para servidores e infraestrutura: Ajustar configurações de servidores web, bancos de dados e outros componentes para seguir práticas recomendadas de segurança, como desabilitar serviços não utilizados e aplicar patches.

- Implementação de mecanismos de segurança para microserviços: Adicionar autenticação e autorização entre microserviços, utilizando tokens, certificados ou outros métodos para garantir comunicações seguras.
- Desenvolvimento de um sistema de gerenciamento de identidades e acessos (IAM) personalizado: Implementar funcionalidades que permitem gerenciar usuários, roles, permissões e políticas de acesso de forma granular.
- Configuração de segurança para comunicação entre serviços utilizando TLS mútuo (mTLS): Implementar TLS mútuo para autenticação de ambas as partes na comunicação entre serviços, garantindo segurança nas transações internas.
- Implementação de mecanismos de prevenção contra-ataques de injeção de código: Adicionar validações e controles que impedem a execução de código malicioso através de entradas do usuário ou outros vetores.
- Desenvolvimento de políticas de segurança para gerenciamento de dispositivos móveis (MDM): Estabelecer regras e configurações que asseguram a segurança quando a aplicação é acessada por dispositivos móveis, incluindo controle de acesso e proteção de dados.
- Implementação de monitoramento de segurança em tempo real: Integrar a aplicação com sistemas de SIEM (Security Information and Event Management) para coleta e análise de eventos de segurança em tempo real.
- Configuração de segurança para serviços em nuvem: Ajustar configurações de segurança específicas para ambientes de nuvem, como controle de acesso IAM, segurança de storage e políticas de rede.
- Implementação de proteção contra-ataques de força bruta distribuídos: Desenvolver mecanismos que detectam e bloqueiam tentativas de login maliciosas provenientes de múltiplos endereços IP ou fontes.
- Desenvolvimento de processos de resposta a incidentes de segurança: Criar planos e procedimentos detalhados para identificação, contenção, erradicação e recuperação de incidentes de segurança.
- Implementação de segurança para APIs GraphQL: Adicionar controles específicos para proteger APIs GraphQL, incluindo limitação de profundidade de queries e validação de inputs.
- Configuração de criptografia ponta a ponta (E2EE) para comunicações: Implementar técnicas que garantem que somente os participantes da comunicação possam ler as mensagens, protegendo dados durante todo o percurso.

- Implementação de autenticação biométrica: Integrar métodos de autenticação que utilizam características biométricas, como impressão digital ou reconhecimento facial, para aumentar a segurança de acesso.
- Desenvolvimento de um sistema de detecção de fraudes: Implementar funcionalidades que analisam padrões de uso e comportamento para identificar atividades fraudulentas ou suspeitas.
- Configuração de políticas de segurança para APIs públicas e privadas: Definir controles de acesso, limitação de taxa e requisitos de autenticação diferentes para APIs destinadas a uso interno e externo.
- Implementação de controle de acesso baseado em atributos (ABAC): Desenvolver um sistema de autorização que utiliza atributos de usuários e recursos para determinar permissões de acesso.
- Desenvolvimento de funcionalidades para conformidade com GDPR/LGPD: Implementar processos que permitem aos usuários exercer direitos como acesso, retificação, exclusão de dados e gestão de consentimento.
- Implementação de segurança para sistemas de mensageria e filas: Garantir que mensagens em sistemas como RabbitMQ ou Kafka estejam protegidas contra acesso não autorizado e integridade comprometida.
- Configuração de políticas de segurança para armazenamento de dados sensíveis: Implementar controles rigorosos para acesso a dados como informações financeiras, de saúde ou pessoais, incluindo criptografia e monitoramento de acesso.
- Desenvolvimento de mecanismos de proteção contra-ataques de negação de serviço (DoS/DDoS) avançados: Implementar técnicas para identificar e mitigar ataques que tentam sobrecarregar o sistema, como balanceamento de carga ou serviços de mitigação.
- Implementação de segurança para aplicações serverless ou funções em nuvem: Configurar permissões e ambientes seguros para funções que executam em plataformas serverless, prevenindo acesso indevido ou escalonamento de privilégios.
- Configuração de políticas de segurança de dados em repouso e em trânsito em bancos de dados: Garantir que os dados armazenados e as comunicações com o banco de dados sejam criptografados e protegidos contra acesso não autorizado.
- Desenvolvimento de sistemas de autenticação delegada (delegated authentication): Implementar mecanismos que permitem que uma aplicação confie na autenticação realizada por outro sistema ou serviço confiável.

- Implementação de sistemas de logging seguro e centralizado: Configurar sistemas que coletam e armazenam logs de forma segura, protegendo contra alteração ou acesso não autorizado e facilitando auditorias.
- Configuração de políticas de segurança de rede, incluindo VPNs e sub-redes seguras: Projetar e implementar a arquitetura de rede que isola componentes críticos e protege contra ameaças externas.
- Implementação de mecanismos de proteção contra-ataques Man-in-the-Middle (MitM): Adicionar técnicas que garantem a integridade e confidencialidade das comunicações, como verificação de certificados e uso de canais seguros.
- Desenvolvimento de procedimentos de segurança para deploy e integração contínua (CI/CD): Estabelecer práticas seguras no pipeline de desenvolvimento, incluindo verificação de código, controle de acesso e gestão de segredos.
- Configuração de autenticação e autorização para APIs REST e SOAP: Implementar padrões como OAuth, SAML ou WS-Security para proteger serviços web e controlar o acesso.
- Implementação de mecanismos de proteção contra-ataques de Cross-Site Request Forgery (CSRF) em múltiplos frameworks: Garantir que diferentes partes da aplicação, possivelmente em diferentes tecnologias, estejam protegidas contra CSRF.
- Desenvolvimento de políticas de segurança para dispositivos IoT: Estabelecer regras e controles para proteger dispositivos conectados, incluindo autenticação segura, criptografia e atualizações de firmware.
- Configuração de mecanismos de detecção de anomalias baseados em IA: Integrar soluções que utilizam inteligência artificial para identificar comportamentos anômalos ou suspeitos no sistema.
- Implementação de segurança para aplicações móveis, incluindo armazenamento seguro e proteção contra engenharia reversa: Adicionar camadas de segurança específicas para aplicativos móveis, protegendo dados locais e dificultando a análise maliciosa.
- Desenvolvimento de sistemas de autenticação baseada em contexto: Implementar mecanismos que ajustam o nível de segurança com base no contexto do acesso, como localização, dispositivo ou horário.
- Configuração de segurança para integração com serviços de terceiros: Estabelecer controles para interações com APIs e serviços externos, incluindo validação de certificados, gerenciamento de chaves e monitoramento de tráfego.

- Implementação de controles de acesso baseados em políticas (PBAC): Utilizar um sistema centralizado de políticas para determinar permissões, permitindo maior flexibilidade e controle.
- Desenvolvimento de mecanismos para prevenção de perda de dados (DLP): Implementar soluções que monitoram e controlam o fluxo de dados sensíveis, prevenindo vazamentos acidentais ou maliciosos.
- Configuração de segurança para ambientes de contêineres e orquestração: Ajustar configurações de segurança em plataformas como Docker e Kubernetes, incluindo isolamento de recursos, políticas de rede e gerenciamento de segredos.
- Implementação de autenticação sem senha (passwordless authentication): Adicionar métodos de autenticação que não dependem de senhas, como links mágicos, tokens de uso único ou autenticação biométrica.
- Desenvolvimento de processos para gestão e resposta a vulnerabilidades descobertas: Estabelecer procedimentos para lidar com vulnerabilidades identificadas, incluindo avaliação de risco, planejamento de correção e comunicação com partes interessadas.

Otimização de Performance e Escalabilidade

A Otimização de Performance e Escalabilidade foca em garantir que a aplicação funcione de forma eficiente e consiga crescer sem perder qualidade no desempenho. Este macro assunto engloba uma série de atividades que vão desde ajustes simples para melhorar a responsividade de sistemas até mudanças mais profundas na arquitetura para suportar um maior volume de usuários e operações.

As otimizações de performance podem incluir melhorias no tempo de resposta, na eficiência do uso de recursos como memória e CPU, além de ajustes em processos intensivos de I/O. Isso pode envolver a implementação de cache em diferentes camadas, otimização de algoritmos críticos e ajustes em protocolos de comunicação. Em relação à escalabilidade, o foco é garantir que o sistema possa crescer de maneira sustentável, seja por meio de escalabilidade horizontal (adição de mais servidores) ou vertical (aumento da capacidade de processamento de um servidor).

Além disso, a Otimização de Performance e Escalabilidade não se limita ao backend, mas também abrange o frontend, bancos de dados, APIs, entre outros. O objetivo é garantir que a aplicação seja capaz de lidar com cargas elevadas e picos inesperados de tráfego sem comprometer a experiência do usuário.

Este macro assunto será utilizado em situações em que se busca melhorar significativamente a capacidade do sistema de atender mais requisições ou reduzir o tempo de resposta, sem a necessidade de refatorações completas de outras áreas. Enquanto alguns ajustes de performance podem ser considerados em outros macros assuntos, este tópico é dedicado às intervenções mais profundas e às estratégias avançadas de escalabilidade.

1 (um) Story Point

As tarefas de 1 Story Point em Otimização de Performance e Escalabilidade são caracterizadas por ajustes menores que, apesar de simples, resultam em melhorias pontuais no desempenho ou na capacidade de escalabilidade do sistema. Essas atividades não exigem mudanças estruturais significativas na arquitetura ou no código e podem ser implementadas rapidamente, com baixo risco de impacto negativo em outras áreas do sistema.

Tarefas desse tipo normalmente envolvem a otimização de pequenas partes da aplicação, como a identificação e remoção de código desnecessário, ajustes em consultas SQL simples para melhorar o tempo de resposta, ou a aplicação de técnicas de minificação em arquivos CSS e JavaScript, resultando em menor consumo de banda e tempos de carregamento mais rápidos. Além disso, podem incluir a configuração de cache de navegador para recursos estáticos, compressão de respostas HTTP com GZIP, ou ajustes de parâmetros de paginação em consultas ou interfaces que lidam com grandes volumes de dados.

Essas otimizações são direcionadas para melhorar a eficiência do sistema de forma incremental, garantindo que mesmo com ajustes simples, o desempenho geral seja beneficiado sem exigir grandes esforços de desenvolvimento.

Exemplos:

- Identificação e remoção de código morto ou não utilizado: Revisar o código para encontrar e eliminar trechos que não são mais necessários, reduzindo o tamanho e melhorando a legibilidade.
- Otimização de consultas SQL simples: Ajustar consultas que estão lentas devido a sintaxes ineficientes ou falta de critérios de filtragem adequados.
- Adição de índices em colunas frequentemente utilizadas em consultas: Criar índices em colunas que são usadas em cláusulas WHERE para acelerar o acesso aos dados.

- Minificação de arquivos CSS e JavaScript: Reduzir o tamanho de arquivos estáticos removendo espaços em branco e comentários, melhorando o tempo de carregamento.
- Configuração de cache de navegador para recursos estáticos: Ajustar headers HTTP para permitir que navegadores armazenem em cache arquivos estáticos, reduzindo requisições ao servidor.
- Compressão de respostas HTTP com GZIP: Configurar o servidor web para comprimir respostas, diminuindo o tempo de transferência de dados.
- Otimização de imagens para web: Reduzir o tamanho de arquivos de imagem sem perda significativa de qualidade, melhorando o tempo de carregamento das páginas.
- Remoção de plugins ou bibliotecas desnecessárias: Eliminar dependências que não estão sendo usadas, reduzindo o tamanho da aplicação e possíveis impactos na performance.
- Ajuste de parâmetros de paginação em listas: Modificar o número de itens exibidos por página para equilibrar a quantidade de dados carregados e a usabilidade.
- Atualização de versões de bibliotecas com melhorias de performance: Atualizar dependências para versões mais recentes que incluem otimizações.
- Implementação de lazy loading para imagens ou componentes: Carregar recursos somente quando necessários, diminuindo o tempo inicial de carregamento.
- Revisão de loops e estruturas de repetição ineficientes: Otimizar loops que podem ser substituídos por alternativas mais rápidas ou eliminados.
- Configuração de timeouts apropriados para requisições HTTP: Definir tempos limite que evitam que requisições fiquem pendentes indefinidamente, liberando recursos.
- Utilização de tipos de dados apropriados em variáveis: Escolher tipos de dados que consomem menos memória ou são mais rápidos para processar.
- Remoção de logs excessivos em produção: Reduzir a quantidade de logging desnecessário que pode impactar a performance.
- Implementação de cache simples em funções ou métodos: Armazenar resultados de operações custosas que não mudam frequentemente para reutilização.
- Ajuste de configurações do garbage collector: Modificar parâmetros básicos que podem melhorar a gestão de memória em tempo de execução.

- Utilização de variáveis locais em vez de globais quando possível: Melhorar o acesso a dados e reduzir conflitos de escopo.
- Remoção de chamadas síncronas desnecessárias: Substituir chamadas bloqueantes por alternativas assíncronas quando apropriado.
- Simplificação de expressões regulares complexas: Otimizar regex que podem estar consumindo recursos excessivamente.
- Organização e limpeza de arquivos e diretórios de recursos: Facilitar o acesso e gerenciamento de arquivos, melhorando a eficiência do sistema.
- Configuração de DNS para reduzir latência: Ajustar registros DNS para otimizar a resolução de nomes de domínio.
- Utilização de CDN para recursos estáticos simples: Implementar o uso de redes de distribuição de conteúdo para acelerar a entrega de arquivos estáticos.
- Ajuste de configurações básicas do servidor web: Modificar parâmetros como número de threads ou tamanho de buffers para melhorar o desempenho.
- Implementação de pré-conexão ou pré-busca em páginas web: Utilizar técnicas que antecipam requisições futuras, melhorando a experiência do usuário.
- Otimização de tamanho de fontes e arquivos de estilo: Reduzir o peso de arquivos CSS removendo estilos não utilizados.
- Remoção de comentários e espaços em arquivos de produção: Limpar arquivos para diminuir o tamanho sem afetar a funcionalidade.
- Configuração de Keep-Alive em conexões HTTP: Manter conexões abertas para múltiplas requisições, reduzindo o overhead de estabelecimento de conexões.
- Utilização de escopos adequados em linguagens de programação: Garantir que variáveis e funções estejam no escopo correto para eficiência.
- Implementação de debounce ou throttle em eventos de interface: Controlar a frequência de execução de funções ligadas a eventos frequentes como scroll ou resize.
- Revisão de consultas ao banco de dados para evitar N+1: Ajustar consultas que podem estar causando múltiplas chamadas desnecessárias ao banco de dados.
- Utilização de pools de conexões para acesso a recursos externos: Reutilizar conexões para melhorar a eficiência na comunicação com bancos de dados ou serviços.
- Otimização de carregamento de fontes web: Utilizar formatos eficientes e técnicas como font-display para melhorar o carregamento.

- Implementação de headers para controle de cache no servidor: Configurar políticas de cache adequadas para recursos dinâmicos e estáticos.
- Utilização de variáveis imutáveis quando possível: Aproveitar benefícios de desempenho e previsibilidade em linguagens que suportam imutabilidade.
- Revisão de código para identificar e remover duplicações: Eliminar código redundante que pode estar afetando a performance.
- Configuração de prioridades em carregamento de scripts: Definir a ordem e modo de carregamento de scripts para otimizar o tempo de renderização.
- Ajuste de tamanhos de buffer em operações de I/O: Modificar tamanhos de buffer para operações de leitura e escrita, otimizando a transferência de dados.
- Utilização de algoritmos de ordenação ou busca mais eficientes para pequenos conjuntos de dados: Escolher algoritmos apropriados ao contexto.
- Implementação de paged queries no banco de dados: Limitar a quantidade de dados retornados em consultas para melhorar o desempenho.
- Configuração de compressão de arquivos no sistema de arquivos: Utilizar compressão onde apropriado para economizar espaço e melhorar a velocidade de acesso.
- Utilização de variables de ambiente para configurações: Evitar leituras desnecessárias de arquivos de configuração em tempo de execução.
- Remoção de código de depuração em builds de produção: Garantir que código utilizado apenas para desenvolvimento não impacte a performance em produção.
- Otimização de rotas em aplicações web: Simplificar rotas e middlewares para acelerar o processamento de requisições.
- Implementação de short-circuit evaluation em condições: Escrever condições de forma que a avaliação possa ser interrompida mais cedo.
- Utilização de métodos nativos da linguagem ou biblioteca padrão: Preferir funções otimizadas já disponíveis em vez de implementar do zero.
- Revisão de políticas de serialização e desserialização: Otimizar processos que convertem dados para formatos como JSON ou XML.
- Configuração de limites para uso de recursos em aplicações: Definir limites de memória ou CPU para evitar que uma parte do sistema afete o todo.

- Implementação de parâmetros opcionais com valores padrão: Evitar sobrecarga de processamento verificando a existência de parâmetros.
- Utilização de operações em lote quando possível: Agrupar operações para reduzir o número de chamadas ou transações necessárias.
- Revisão de políticas de conexão a bancos de dados: Ajustar tempos de espera e reutilização de conexões para melhorar o desempenho.
- Implementação de checagens de estado antes de operações custosas: Verificar condições simples que podem evitar a execução de código pesado desnecessariamente.
- Ajuste de níveis de log para produção: Configurar níveis de logging para evitar processamento excessivo e consumo de espaço.
- Utilização de ferramentas de análise de performance básicas: Aplicar ferramentas simples para identificar rapidamente possíveis gargalos.
- Revisão de dependências para identificar versões mais leves: Substituir bibliotecas por alternativas mais eficientes quando possível.

5 (cinco) Story Points

As tarefas de 5 Story Points em Otimização de Performance e Escalabilidade envolvem ajustes mais significativos no código ou nas configurações que trazem melhorias perceptíveis no desempenho ou na escalabilidade do sistema. Estas atividades requerem um nível de esforço moderado, pois envolvem mudanças que otimizam aspectos críticos da aplicação sem a necessidade de grandes reformulações na arquitetura.

Essas tarefas podem incluir a implementação de cache em camadas de aplicação para armazenar resultados de operações custosas e, assim, reduzir o tempo de resposta em solicitações subsequentes. Outras atividades típicas neste nível envolvem a otimização de consultas SQL complexas, ajustes no balanceamento de carga simples entre servidores para distribuir melhor o tráfego, ou a implementação de paralelismo em processos de backend para aproveitar melhor os recursos de hardware.

Estas atividades são direcionadas para otimizar áreas específicas que impactam diretamente a performance da aplicação, sem envolver mudanças radicais, mas que trazem ganhos substanciais de eficiência.

Exemplos:

- Implementação de cache em camadas de aplicação: Adicionar um sistema de cache na aplicação para armazenar resultados de operações custosas, reduzindo o tempo de resposta em solicitações subsequentes.
- Otimização de consultas complexas no banco de dados: Reescrever consultas SQL que estão consumindo muitos recursos, utilizando joins eficientes, subconsultas otimizadas e evitando scans completos de tabelas.
- Configuração de um CDN para recursos estáticos: Implementar uma rede de distribuição de conteúdo para entregar arquivos estáticos de forma mais rápida aos usuários, reduzindo a latência e a carga no servidor principal.
- Implementação de paralelismo ou multithreading em processos: Modificar partes do código para executar operações em paralelo, aproveitando melhor os recursos de hardware disponíveis.
- Ajuste de parâmetros avançados do servidor web ou de aplicação: Modificar configurações como tamanho de thread pools, limites de conexão e parâmetros de memória para otimizar o desempenho.
- Utilização de lazy loading para módulos ou componentes: Carregar módulos ou componentes da aplicação somente quando necessários, reduzindo o tempo de carregamento inicial.
- Implementação de balanceamento de carga simples: Configurar um balanceador de carga básico para distribuir o tráfego entre múltiplos servidores, aumentando a capacidade de atendimento.
- Otimização de algoritmos e estruturas de dados: Substituir algoritmos ineficientes por alternativas mais rápidas ou utilizar estruturas de dados mais adequadas para o contexto.
- Configuração de compressão avançada para arquivos estáticos: Utilizar métodos de compressão como Brotli para reduzir ainda mais o tamanho de arquivos servidos aos usuários.
- Implementação de pré-processamento ou compilação antecipada: Utilizar técnicas como AOT (Ahead-of-Time) compilation para melhorar o desempenho em tempo de execução.
- Análise e redução de chamadas a APIs externas: Identificar e eliminar chamadas redundantes a serviços externos, implementando cache ou combinando requisições.
- Implementação de paginação ou scroll infinito eficiente: Melhorar a forma como grandes listas de dados são carregadas e exibidas, reduzindo o consumo de recursos e melhorando a experiência do usuário.

- Otimização de sessões e gerenciamento de estado: Ajustar como o estado do usuário é armazenado e recuperado, utilizando métodos mais eficientes como tokens ou armazenamento em memória.
- Configuração de pool de conexões ao banco de dados: Implementar um pool de conexões para reutilizar conexões existentes, reduzindo a sobrecarga de criação de novas conexões.
- Implementação de processamento assíncrono para tarefas demoradas: Mover operações que consomem tempo para processos assíncronos ou filas de mensagens, liberando recursos para atender outras requisições.
- Utilização de técnicas de memoization em funções puras: Armazenar resultados de funções determinísticas para evitar recomputação desnecessária.
- Configuração de cache em nível de banco de dados: Utilizar mecanismos de cache oferecidos pelo banco de dados para acelerar consultas frequentes.
- Otimização de serialização e desserialização de objetos: Escolher formatos mais eficientes ou ajustar configurações para reduzir o tempo gasto em conversão de dados.
- Implementação de pré-carregamento de dados essenciais: Carregar antecipadamente dados críticos para o funcionamento da aplicação, melhorando a responsividade.
- Análise e otimização de uso de memória: Identificar pontos de consumo excessivo de memória e implementar soluções para reduzir o footprint da aplicação.
- Implementação de técnicas de pooling para objetos pesados: Reutilizar objetos que são custosos para criar ou destruir, como conexões de rede ou objetos gráficos.
- Configuração de políticas de retenção de logs: Ajustar o tempo de retenção e nível de detalhe dos logs para equilibrar entre necessidade de informação e consumo de recursos.
- Utilização de técnicas de compressão em armazenamento de dados: Comprimir dados armazenados para reduzir o espaço em disco e melhorar a velocidade de acesso.
- Implementação de cache em camadas de serviço ou repositório: Armazenar resultados de consultas ou operações em níveis intermediários para reduzir a carga em sistemas de backend.
- Otimização de consultas para evitar deadlocks e contenção: Ajustar transações e bloqueios no banco de dados para melhorar a concorrência e o desempenho.

- Configuração de monitoramento básico de performance: Implementar ferramentas que coletam métricas simples de desempenho, permitindo identificar gargalos.
- Implementação de algoritmos de balanceamento de carga simples: Escolher e configurar um algoritmo adequado para distribuir requisições entre servidores de forma eficiente.
- Otimização de tarefas agendadas ou cron jobs: Ajustar a frequência e horários de execução para evitar impacto no desempenho durante picos de uso.
- Configuração de parâmetros de garbage collection avançados: Ajustar configurações do coletor de lixo para melhorar a gestão de memória em linguagens que suportam GC.
- Implementação de chunking em uploads e downloads de arquivos: Dividir transferências de arquivos grandes em partes menores para melhorar a eficiência e tolerância a falhas.
- Utilização de serviços de cache distribuído como Redis ou Memcached: Implementar cache em memória distribuída para acelerar o acesso a dados compartilhados.
- Otimização de processamento de arquivos em lote: Melhorar a eficiência de tarefas que processam grandes volumes de dados, utilizando técnicas como processamento em streaming.
- Configuração de pipelines de build otimizados: Ajustar o processo de compilação e empacotamento da aplicação para reduzir o tempo de deploy.
- Implementação de compressão de imagens em tempo real: Utilizar técnicas que reduzem o tamanho de imagens servidas sem perda perceptível de qualidade.
- Análise e otimização de código em linguagens interpretadas: Identificar e melhorar trechos de código que estão causando lentidão em linguagens como Python ou JavaScript.
- Configuração de políticas de prefetching em aplicações web: Antecipar e carregar recursos que provavelmente serão necessários em seguida, melhorando a fluidez.
- Implementação de técnicas de virtualização ou containerização eficientes: Ajustar configurações de VMs ou contêineres para melhor uso de recursos.
- Otimização de tarefas de renderização no frontend: Melhorar o desempenho de renderização no navegador, evitando repaints ou reflows desnecessários.

- Utilização de data binding eficiente em frameworks frontend: Ajustar como os dados são vinculados à interface para reduzir o consumo de recursos.
- Configuração de prioridades em filas de mensagens: Ajustar prioridades para garantir que tarefas críticas sejam processadas primeiro.
- Implementação de sharding simples em bancos de dados NoSQL: Dividir dados entre diferentes nós para melhorar a escalabilidade horizontal.
- Análise e otimização de latência em comunicação entre serviços: Identificar atrasos na comunicação e implementar soluções como RPCs mais rápidos ou protocolos binários.
- Configuração de políticas de reconexão para serviços instáveis: Ajustar como a aplicação lida com falhas temporárias em serviços externos, melhorando a resiliência.
- Utilização de algoritmos de compressão eficientes em transmissão de dados: Escolher algoritmos que equilibram bem compressão e velocidade para otimizar transferências.
- Implementação de técnicas de edge computing simples: Processar dados mais próximos da fonte para reduzir latência e carga nos servidores centrais.
- Configuração de balanceamento de carga DNS simples: Utilizar técnicas de balanceamento de carga baseadas em DNS para distribuir tráfego.
- Otimização de parâmetros de protocolo de rede: Ajustar configurações como tamanho de janela TCP ou buffers UDP para melhorar a performance de rede.
- Implementação de técnicas de redução de uso de banda em aplicações móveis: Otimizar a quantidade de dados transferidos para melhorar a experiência em redes móveis.
- Configuração de mecanismos de fallback em caso de falha de serviços: Implementar soluções alternativas que mantêm a funcionalidade básica em caso de falhas.
- Utilização de técnicas de preempção em sistemas multitarefa: Ajustar como tarefas são interrompidas e retomadas para melhorar o uso de CPU.
- Implementação de pooling de threads para operações assíncronas: Gerenciar eficientemente threads para evitar overhead de criação e destruição frequente.
- Análise e otimização de operações de entrada e saída (I/O): Melhorar a eficiência em operações de leitura e escrita em disco ou rede.

- Configuração de ambientes de staging para testes de performance: Criar ambientes que permitem testar otimizações antes de aplicar em produção.

13 (treze) Story Points

As tarefas de 13 Story Points em Otimização de Performance e Escalabilidade envolvem modificações significativas no código ou na arquitetura da aplicação, com foco em melhorar substancialmente o desempenho e a capacidade de escalabilidade. Essas atividades demandam um planejamento mais detalhado e podem exigir ajustes em múltiplos componentes do sistema, como banco de dados, infraestrutura de rede ou serviços de backend.

Neste nível, as otimizações frequentemente incluem a implementação de mecanismos de cache distribuído, configuração de balanceamento de carga avançado, e a otimização de consultas complexas no banco de dados, com possíveis alterações no esquema de dados para melhorar a eficiência. Além disso, tarefas como o uso de técnicas de sharding para distribuir dados entre múltiplos nós ou a implementação de auto scaling em infraestrutura de nuvem são comuns nesse escopo, garantindo que o sistema consiga atender picos de demanda sem perda de desempenho.

Exigem um impacto direto na performance do sistema, tornando-o mais preparado para lidar com grandes volumes de tráfego ou operações intensivas.

Exemplos:

- Implementação de cache distribuído em múltiplos servidores usando Redis ou Memcached: Configurar um sistema de cache distribuído, incluindo setup de clusters, estratégias de invalidação e ajuste fino para melhorar a performance.
- Otimização de consultas complexas no banco de dados com alterações no esquema: Reestruturar tabelas, criar índices avançados como índices compostos ou full-text, e ajustar relacionamentos para otimizar consultas pesadas.
- Implementação de balanceamento de carga avançado com Nginx ou HAProxy: Configurar balanceadores de carga com algoritmos personalizados, health checks e failover para distribuir eficientemente o tráfego.

- Refatoração de código crítico para melhorar a performance: Reescrever partes do sistema utilizando algoritmos mais eficientes ou linguagens de programação de alto desempenho.
- Implementação de sharding em bancos de dados relacionais: Planejar e executar a divisão de dados em diferentes shards, ajustando a aplicação para lidar com a nova arquitetura.
- Configuração de escalabilidade horizontal com auto scaling na nuvem: Implementar políticas de auto scaling em AWS, Azure ou GCP para ajustar automaticamente os recursos com base na demanda.
- Implementação de CDNs avançadas com cache hierárquico: Configurar CDNs com regras complexas de cache e invalidação, integrando com a aplicação para atualização dinâmica de conteúdo.
- Desenvolvimento de pré-processamento e pré-carregamento de dados: Utilizar filas e workers para processar dados antecipadamente, reduzindo a latência em operações críticas.
- Otimização de pipelines de dados e processos ETL: Paralelizar processos, implementar frameworks de processamento distribuído como Apache Spark, melhorando a eficiência em grandes volumes de dados.
- Migração para arquitetura de microserviços: Dividir uma aplicação monolítica em microserviços, incluindo definição de APIs, comunicação interserviços e refatoração de código.
- Implementação de caching de página completa (full-page caching): Configurar cache de páginas inteiras, incluindo lógica para invalidação baseada em eventos ou atualizações.
- Otimização de código para dispositivos móveis ou recursos limitados: Adaptar a aplicação para funcionar eficientemente em dispositivos com restrições de hardware ou conectividade.
- Implementação de filas de mensagens para processamento assíncrono: Utilizar sistemas como RabbitMQ ou Kafka, alterando a aplicação para suportar processamento em background.
- Configuração de replicação de banco de dados para melhorar leituras: Implementar réplicas de leitura, ajustando a aplicação para distribuir consultas e melhorar o desempenho.
- Implementação de paralelismo e concorrência avançados: Utilizar multi-threading, async/await ou programação reativa para melhorar a eficiência em operações intensivas.
- Otimização de algoritmos críticos através de perfilamento: Analisar o desempenho do código, identificar gargalos e implementar algoritmos mais eficientes.

- Implementação de cache coeso e distribuído: Configurar cache que mantém consistência entre múltiplos nós, incluindo estratégias de invalidação e sincronização.
- Configuração de monitoramento de performance avançado: Utilizar ferramentas como Prometheus e Grafana para coletar métricas detalhadas e configurar alertas proativos.
- Otimização de renderização no frontend com virtual DOM: Implementar técnicas que reduzem o número de atualizações no DOM, melhorando a performance em aplicações web.
- Criação de índices avançados no banco de dados: Implementar índices columnstore, índices hash ou views materializadas para otimizar consultas específicas.
- Configuração de sistemas de caching em nível de aplicação: Utilizar ferramentas como Varnish ou Squid para melhorar a entrega de conteúdo dinâmico e estático.
- Planejamento e implementação de escalabilidade vertical e horizontal: Analisar a capacidade atual, prever necessidades futuras e ajustar a arquitetura para suportar o crescimento.
- Otimização de protocolos de comunicação com gRPC ou Protobuf: Migrar APIs para protocolos binários mais eficientes, reduzindo latência e uso de banda.
- Implementação de arquitetura CQRS ou Event Sourcing: Separar comandos de consultas ou registrar eventos para melhorar a escalabilidade em sistemas com alto volume de operações.
- Configuração de replicação e failover para alta disponibilidade: Implementar mecanismos que garantem continuidade do serviço em caso de falhas, incluindo replicação síncrona ou assíncrona.
- Utilização de edge computing ou funções serverless: Processar dados mais próximos dos usuários ou utilizar computação sem servidor para melhorar a performance e reduzir custos.
- Otimização de processamento de dados em tempo real: Utilizar frameworks como Apache Kafka Streams ou Flink para processar fluxos de dados com baixa latência.
- Implementação de compressão avançada e otimização de payloads: Utilizar algoritmos como Brotli, ajustar formatos de dados para reduzir o tamanho das mensagens.
- Otimização de pipelines de CI/CD: Automatizar testes, builds e deploys para reduzir o tempo de entrega e melhorar a confiabilidade.

- Implementação de containerização e orquestração com Kubernetes: Migrar aplicações para contêineres, configurar clusters e implementar políticas de escalabilidade e resiliência.
- Ajuste fino de garbage collection e gestão de memória: Configurar parâmetros avançados do GC, utilizar pools de memória ou linguagens sem GC quando apropriado.
- Implementação de arquiteturas serverless para cargas variáveis: Utilizar serviços como AWS Lambda ou Azure Functions para escalar automaticamente com base na demanda.
- Configuração de rate limiting avançado: Implementar políticas que controlam o uso de recursos, protegendo contra abuso e melhorando a qualidade do serviço.
- Otimização de assets e código no frontend: Aplicar técnicas como code splitting, tree shaking e minificação agressiva para reduzir o tamanho dos bundles.
- Implementação de desnormalização de dados para performance: Ajustar o esquema do banco de dados para evitar joins custosos, mantendo a consistência através da aplicação.
- Configuração de cache em bancos NoSQL com TTL e políticas de expiração: Ajustar estratégias de armazenamento para dados que podem ser descartados após certo tempo.
- Implementação de padrões como Circuit Breaker e Bulkhead: Proteger a aplicação contra falhas em serviços externos e isolar componentes para melhorar a resiliência.
- Otimização para altos volumes de transações: Ajustar filas, conexões e recursos para suportar picos de uso sem degradação significativa.
- Configuração de balanceamento de carga de camada 7: Utilizar informações de aplicação para roteamento avançado, persistência de sessão e inspeção de tráfego.
- Implementação de backpressure em sistemas reativos: Controlar o fluxo de dados em sistemas assíncronos para evitar sobrecarga e garantir estabilidade.
- Redução do tempo de inicialização da aplicação: Otimizar carregamento de módulos, lazy loading e reduzir dependências para melhorar o startup time.
- Implementação de warmup de caches e serviços: Preencher caches antecipadamente e iniciar serviços críticos para reduzir latência em períodos de alto acesso.
- Otimização de operações I/O intensivas: Utilizar técnicas de buffering, pipelines ou protocolos mais eficientes para melhorar o throughput.

- Configuração de sistemas de logs centralizados: Implementar soluções como ELK Stack para coletar e analisar logs, auxiliando na identificação de problemas de performance.
- Implementação de compactação e armazenamento eficiente em big data: Utilizar formatos como Parquet ou ORC para armazenar grandes volumes de dados de forma otimizada.
- Otimização de consultas analíticas complexas: Implementar data warehouses, OLAP cubes ou tecnologias columnar para acelerar consultas de BI.
- Configuração de filas de mensagens de alto throughput: Ajustar particionamento, replicação e consumo paralelo em sistemas como Kafka para melhorar a taxa de processamento.
- Implementação de cache multinível (L1, L2): Configurar diferentes níveis de cache para balancear entre velocidade e capacidade de armazenamento.
- Otimização para processadores multinúcleo e arquiteturas NUMA: Ajustar a aplicação para aproveitar eficientemente recursos de hardware modernos.
- Implementação de prefetching e predição: Utilizar algoritmos que antecipam operações futuras para melhorar a responsividade.
- Configuração de API Gateway: Centralizar o gerenciamento de APIs, incluindo autenticação, rate limiting e caching para melhorar a escalabilidade.
- Implementação de data lake e otimização de ingestão de dados: Projetar arquiteturas que suportam armazenamento e processamento de grandes volumes de dados não estruturados.
- Otimização de latência de rede: Utilizar técnicas como compactação, protocolos rápidos e edge computing para reduzir o tempo de resposta.
- Configuração de replicação geográfica: Distribuir dados e serviços em múltiplas regiões para melhorar a performance global e disponibilidade.
- Implementação de streaming adaptativo em mídia: Ajustar a qualidade de streaming em tempo real com base nas condições da rede para melhorar a experiência do usuário.
- Otimização para IoT e dispositivos embarcados: Adaptar a aplicação para funcionar eficientemente em dispositivos com recursos extremamente limitados.
- Configuração de gerenciamento elástico de recursos: Implementar auto scaling baseado em métricas customizadas para ajustar recursos automaticamente.

- Implementação de padrões de design orientados a performance: Aplicar padrões como batch processing ou bulk operations para melhorar a eficiência.
- Utilização de tecnologias emergentes como WebAssembly: Migrar partes da aplicação para WebAssembly para obter ganhos de performance significativos no frontend.
- Preparação para picos inesperados de carga: Implementar estratégias de overflow para utilizar recursos em nuvem pública durante picos de demanda.
- Implementação de arquiteturas tolerantes a falhas: Projetar sistemas que mantêm a performance mesmo quando componentes individuais falham.
- Otimização com bancos de dados in-memory: Utilizar bancos de dados como Redis ou VoltDB para acelerar o acesso a dados críticos.

Implementação de Funcionalidades de DevOps e CI/CD

A Implementação de Funcionalidades de DevOps e CI/CD abrange uma série de atividades destinadas a automatizar, monitorar e aprimorar os processos de desenvolvimento, integração contínua (CI) e entrega contínua (CD) de software. Embora a PPSA espere que a CONTRATADA já tenha suas práticas e rotinas básicas de DevOps em operação, este macro assunto visa abordar solicitações que exigem esforços adicionais para implementar soluções personalizadas ou expandir os processos existentes, a fim de otimizar a entrega e a escalabilidade das aplicações.

Essas solicitações podem incluir desde ajustes simples em pipelines de CI/CD até a implementação de soluções mais complexas, como deploys automatizados com rollback, validações multi-stage, ou integração com sistemas de monitoramento e segurança avançada. O objetivo é garantir que o sistema mantenha alta disponibilidade e eficiência, especialmente em cenários que demandam respostas rápidas a mudanças, como deploys frequentes ou escalabilidade dinâmica.

Esse macro assunto será utilizado em situações específicas, como quando há necessidade de melhorar processos críticos de deploy, integração de novos ambientes ou automatização de auditorias de segurança, que vão além das práticas DevOps comuns já estabelecidas.

1 (um) Story Point

As tarefas de 1 Story Point em Implementação de Funcionalidades de DevOps e CI/CD são atividades de baixo esforço que se concentram em ajustes menores ou automatizações simples nos pipelines de integração e entrega contínua. Essas tarefas não requerem mudanças significativas na infraestrutura ou nos processos principais de CI/CD já estabelecidos pela CONTRATADA, e seu impacto é restrito a pequenas melhorias ou correções pontuais.

As atividades neste nível podem incluir ajustes em scripts de build para refletir pequenas mudanças no código, configuração de variáveis de ambiente necessárias para o funcionamento correto dos pipelines ou a integração de testes unitários simples no processo de integração contínua. Outros exemplos incluem a atualização de plugins ou ferramentas de CI/CD para versões mais recentes e a implementação de notificações automáticas sobre o status dos builds.

Essas tarefas têm como objetivo melhorar a eficiência operacional de forma incremental, sem a necessidade de reestruturar processos ou implementar mudanças complexas na automação existente.

Exemplos:

- Atualização de scripts de build com pequenas mudanças: Ajustar scripts existentes para refletir alterações simples no código ou nas dependências.
- Configuração de variáveis de ambiente em pipelines de CI/CD: Adicionar ou atualizar variáveis de ambiente necessárias para builds ou deploys.
- Automatização de um teste unitário simples no pipeline: Integrar um teste unitário básico ao processo de integração contínua.
- Correção de erros em scripts de deploy: Ajustar scripts de implantação para corrigir pequenos erros ou incompatibilidades.
- Adição de notificações de build em ferramentas de chat: Configurar notificações simples para alertar a equipe sobre o status dos builds em plataformas como Slack ou Microsoft Teams.
- Atualização de imagens de contêiner com novas versões de dependências: Rebuild de imagens Docker com versões atualizadas de pacotes ou bibliotecas.
- Configuração de chaves SSH para acesso a repositórios: Adicionar chaves públicas a servidores ou serviços para permitir acesso seguro.
- Criação de um job agendado simples no Jenkins: Configurar um job que executa uma tarefa básica em intervalos definidos.
- Implementação de linting automático no pipeline: Adicionar uma etapa que verifica o código em busca de problemas de estilo ou erros simples.

- Configuração de um ambiente de teste local: Preparar um ambiente básico para testes locais, incluindo instalação de dependências necessárias.
- Atualização de documentação de processos de CI/CD: Revisar e melhorar documentos que descrevem os processos atuais de integração e entrega contínua.
- Adição de tags em imagens Docker para identificação: Marcar imagens com tags apropriadas para facilitar o rastreamento e uso.
- Configuração de logs básicos em servidores: Ajustar configurações para garantir que logs importantes sejam armazenados e acessíveis.
- Atualização de plugins em ferramentas de CI/CD: Atualizar plugins ou extensões para versões mais recentes que incluem correções ou melhorias.
- Implementação de testes de fumaça simples no pipeline: Adicionar testes básicos que verificam se a aplicação está rodando após o deploy.
- Configuração de parâmetros de timeout em pipelines: Definir limites de tempo para etapas do pipeline, evitando bloqueios ou esperas indesejadas.
- Adição de um repositório privado no sistema de versionamento: Configurar acesso a um repositório privado para dependências ou código interno.
- Criação de aliases ou scripts de atalho para comandos frequentes: Facilitar a execução de comandos comuns com scripts simples ou aliases.
- Implementação de limpeza automática de arquivos temporários: Configurar scripts que removem arquivos temporários após builds ou testes.
- Atualização de certificados SSL próximos da expiração: Renovar certificados em servidores ou serviços para garantir conexões seguras.
- Configuração de parâmetros básicos de segurança em servidores: Ajustar configurações simples para melhorar a segurança, como desabilitar portas não utilizadas.
- Adição de metadados em commits ou builds: Incluir informações adicionais que facilitam o rastreamento de mudanças.
- Criação de um script simples para backup de arquivos: Automatizar o backup de arquivos importantes em um local seguro.
- Configuração de mensagens de commit padrão: Definir um template para mensagens de commit, garantindo consistência.
- Implementação de versão automática em builds: Ajustar o pipeline para incrementar automaticamente o número de versão em cada build.

- Adição de licenças ou avisos legais em arquivos de código: Incluir cabeçalhos padrão que atendem a requisitos legais ou organizacionais.
- Configuração de notificações por e-mail em caso de falhas: Enviar alertas básicos quando builds ou deploys falharem.
- Atualização de configurações de proxy em servidores ou ferramentas: Ajustar configurações para permitir acesso à internet ou outros recursos através de um proxy.
- Implementação de testes de compatibilidade simples: Adicionar testes que verificam a compatibilidade com versões específicas de linguagens ou frameworks.
- Configuração de acesso a banco de dados de teste: Preparar o ambiente para que testes possam acessar um banco de dados dedicado.
- Criação de um arquivo README com instruções básicas: Documentar passos simples para setup ou contribuições no projeto.
- Atualização de scripts para suporte a novas plataformas: Ajustar scripts para rodar em diferentes sistemas operacionais ou ambientes.
- Implementação de verificação de assinatura em commits: Configurar o repositório para aceitar apenas commits assinados, melhorando a segurança.
- Configuração de um servidor web simples para testes: Instalar e configurar um servidor web básico para servir a aplicação em desenvolvimento.
- Adição de comentários em scripts complexos: Melhorar a legibilidade e manutenção adicionando comentários explicativos.
- Implementação de políticas básicas de branch em VCS: Definir regras simples para criação e nomeação de branches no sistema de controle de versão.
- Configuração de um webhook para integração com outros serviços: Ajustar o repositório ou ferramenta para enviar notificações a serviços externos.
- Atualização de dependências em arquivos de configuração: Atualizar versões de pacotes ou bibliotecas em arquivos como package.json ou pom.xml.
- Implementação de uma página de status simples: Criar uma página que exibe informações básicas sobre o estado do sistema ou aplicação.
- Configuração de políticas de retenção de logs: Definir por quanto tempo os logs serão mantidos antes de serem excluídos.
- Adição de um arquivo .gitignore ou equivalente: Especificar quais arquivos ou pastas não devem ser rastreados pelo sistema de controle de versão.

- Atualização de URLs ou endpoints em scripts de deploy: Ajustar scripts para refletir mudanças em URLs ou endpoints de serviços utilizados.
- Implementação de suporte a múltiplos ambientes em scripts: Adicionar parâmetros ou configurações que permitem que scripts rodem em diferentes ambientes (dev, test, prod).
- Configuração de permissões básicas em repositórios: Ajustar quem pode ler, escrever ou administrar um repositório ou projeto.
- Criação de um script para limpeza de caches: Automatizar a remoção de caches que podem causar inconsistências durante o desenvolvimento ou testes.
- Adição de labels ou tags em issues ou tickets: Organizar tarefas ou bugs utilizando labels para facilitar o gerenciamento.
- Implementação de um template de pull request: Criar um modelo padrão para novas pull requests, garantindo que informações importantes sejam incluídas.
- Configuração de builds paralelos simples: Ajustar o pipeline para executar etapas que não dependem entre si ao mesmo tempo, reduzindo o tempo total.
- Atualização de informações de licença em arquivos ou pacotes: Garantir que a informação de licença esteja correta e atualizada nos artefatos gerados.
- Adição de passos de validação simples antes do commit: Implementar hooks que executam validações básicas, como verificação de formato de código ou presença de testes.
- Configuração de um serviço simples de monitoramento: Integrar a aplicação com um serviço que verifica se ela está online, sem ajustes complexos.
- Implementação de suporte a múltiplas versões de linguagens ou frameworks: Ajustar scripts para permitir builds com diferentes versões de dependências.
- Adição de testes de integração simples ao pipeline: Incluir testes que verificam a interação entre componentes básicos do sistema.
- Configuração de variáveis secretas em ambientes de teste: Adicionar chaves ou tokens necessários para testes, mantendo-os seguros.
- Atualização de scripts de inicialização da aplicação: Ajustar scripts que iniciam a aplicação para refletir pequenas mudanças na configuração.
- Implementação de suporte a novos formatos de logs: Ajustar o sistema para gerar logs em um formato específico, como JSON, para facilitar a análise.

- Criação de aliases para comandos frequentes no terminal: Facilitar o uso de comandos complexos ou longos através de atalhos.
- Configuração de timezones ou localizações em servidores: Ajustar configurações de fuso horário ou regionalização que podem afetar a aplicação.
- Adição de scripts de migração simples para banco de dados: Automatizar alterações básicas no esquema do banco de dados durante o deploy.
- Atualização de tokens de acesso expirados ou comprometidos: Renovar tokens utilizados em serviços ou APIs para manter a funcionalidade e segurança.

5 (cinco) Story Points

As tarefas de 5 Story Points em Implementação de Funcionalidades de DevOps e CI/CD exigem um esforço moderado e podem incluir a configuração mais detalhada de pipelines, ajustes em processos críticos de deploy ou integração com ferramentas que aprimoram o ciclo de vida de desenvolvimento, testes e entrega contínua. Embora não demandem uma reestruturação profunda, essas atividades têm um impacto significativo em áreas específicas do fluxo DevOps, contribuindo para maior eficiência e automação.

Entre as atividades típicas deste nível estão a criação de pipelines de integração contínua (CI) para múltiplos ambientes (desenvolvimento, teste, produção), a automatização de testes de integração para verificar a interação entre componentes da aplicação, e a implementação de deploys automatizados para ambientes de teste, permitindo a entrega contínua de software com menos intervenção manual. Outros exemplos incluem o ajuste de configurações de Docker Compose para coordenar múltiplos serviços em ambientes locais, ou a implementação de versionamento automático para imagens Docker, garantindo a rastreabilidade das versões de aplicação.

Essas tarefas são voltadas para a automação e aprimoramento dos fluxos de trabalho já existentes, com foco na eficiência e na redução de erros operacionais, sem exigir mudanças extensivas na infraestrutura de DevOps da organização.

Exemplos:

- Configuração de pipelines de integração contínua (CI) para múltiplos ambientes: Criar pipelines que se adaptam automaticamente a diferentes ambientes (desenvolvimento, teste, produção).

- Automatização de testes de integração no pipeline: Integrar testes de integração no pipeline, verificando a interação entre múltiplos componentes da aplicação.
- Configuração de deploy automatizado para ambiente de teste: Implementar um processo automatizado que realiza o deploy da aplicação em um ambiente de teste após a conclusão de um build bem-sucedido.
- Ajuste de configurações de Docker Compose para múltiplos serviços: Modificar o arquivo Docker Compose para coordenar o funcionamento de diversos serviços em ambientes locais ou de teste.
- Implementação de versionamento automático em imagens Docker: Configurar o pipeline para gerar versões incrementais automaticamente ao construir novas imagens Docker.
- Configuração de pipelines com paralelismo: Ajustar o pipeline para que múltiplas tarefas independentes sejam executadas em paralelo, reduzindo o tempo total de execução.
- Implementação de hooks em sistemas de versionamento: Adicionar hooks que automatizam ações específicas, como verificação de formato de código, antes de permitir commits ou pushes.
- Criação de infraestrutura como código (IaC) simples: Usar ferramentas como Terraform ou AWS CloudFormation para provisionar infraestrutura básica automaticamente.
- Configuração de deploy contínuo (CD) em ambiente de homologação: Ajustar o pipeline para realizar deploys automáticos no ambiente de homologação após a aprovação de uma build.
- Implementação de pipeline com rollback automático: Configurar o pipeline para reverter automaticamente para uma versão anterior da aplicação caso um deploy falhe.
- Configuração de cache em pipelines para otimizar builds: Implementar cache para acelerar o tempo de builds, armazenando dependências ou artefatos reutilizáveis.
- Automatização do deploy de banco de dados: Criar scripts que realizam automaticamente migrações e atualizações no banco de dados durante o processo de deploy.
- Adição de testes de performance no pipeline: Configurar o pipeline para executar testes de performance automaticamente após o build da aplicação.
- Implementação de monitoramento básico de aplicações em produção: Integrar a aplicação com ferramentas de monitoramento que verificam disponibilidade e coletam métricas simples.

- Configuração de deploy canário em ambiente de produção: Implementar um processo de deploy canário que realiza a implantação gradual de novas versões para uma pequena parcela de usuários.
- Automatização de validações de segurança básicas: Adicionar uma etapa no pipeline que realiza verificações automáticas de segurança, como análise de vulnerabilidades em dependências.
- Implementação de integração com ferramentas de análise de qualidade de código: Configurar o pipeline para integrar com ferramentas como SonarQube, que avaliam a qualidade do código automaticamente.
- Criação de artefatos versionados para distribuição: Ajustar o pipeline para gerar artefatos versionados (ex.: binários ou pacotes) que podem ser distribuídos de maneira controlada.
- Configuração de logs centralizados para múltiplos serviços: Implementar uma solução de logs centralizados que consolida os logs de diferentes serviços em um único local para facilitar o monitoramento e análise.
- Implementação de backup automatizado de dados durante o deploy: Configurar o pipeline para realizar backup de bancos de dados ou arquivos antes de um deploy, garantindo segurança em caso de falhas.
- Integração de testes automatizados de segurança no pipeline: Adicionar uma etapa que executa ferramentas de análise de segurança, como scanners de vulnerabilidades, durante o build.
- Automatização de auditorias de compliance simples: Configurar o pipeline para gerar relatórios automáticos de conformidade com padrões, como GDPR ou LGPD.
- Configuração de pipelines para suportar blue-green deployment: Implementar uma estratégia de deploy onde duas versões da aplicação coexistem, facilitando a transição entre elas sem tempo de inatividade.
- Ajuste de configurações de containerização para ambientes de staging: Adaptar as configurações de contêineres para que rodem de forma otimizada em ambientes de teste e homologação.
- Configuração de um pipeline com multi-stage builds: Implementar um build de múltiplas etapas para melhorar a eficiência e segurança, separando etapas de build, teste e deploy.
- Automatização de deploy rollback em caso de falha de saúde da aplicação: Implementar uma lógica que monitora o estado da aplicação após o deploy e reverte automaticamente em caso de falha.
- Adição de integração com ferramentas de gerenciamento de tickets no pipeline: Configurar o pipeline para atualizar automaticamente o status de tickets no JIRA ou Trello com base no progresso do build ou deploy.

- Implementação de notificações detalhadas em ferramentas de colaboração: Configurar o pipeline para enviar notificações mais detalhadas, incluindo logs de falha ou sucesso, em ferramentas como Slack ou Teams.
- Configuração de integração com serviços de autenticação externos (ex.: OAuth, LDAP) no pipeline: Ajustar o processo de deploy para lidar com serviços de autenticação de terceiros.
- Implementação de migrações automáticas de banco de dados no pipeline: Configurar o pipeline para aplicar migrações de esquema automaticamente antes de um deploy.
- Integração com ferramentas de análise de dependências: Adicionar uma etapa no pipeline que analisa as dependências do projeto, verificando por vulnerabilidades conhecidas.
- Configuração de monitoramento em tempo real de pipelines de CI/CD: Implementar um sistema que monitora a execução dos pipelines e fornece métricas em tempo real para acompanhamento da equipe.
- Criação de scripts de inicialização e parada de múltiplos serviços: Automatizar a inicialização e parada de vários serviços dependentes com scripts de controle centralizados.
- Implementação de sistemas de validação de conformidade de código: Configurar o pipeline para verificar se o código segue padrões de conformidade, como PSR-12 no PHP ou PEP-8 no Python.
- Ajuste de pipelines para suportar múltiplos workflows: Configurar pipelines que suportam diferentes fluxos de trabalho de desenvolvimento, como branch principal, desenvolvimento e hotfix.
- Automatização de builds paralelos para diferentes versões da aplicação: Configurar o pipeline para construir e testar múltiplas versões da aplicação em paralelo.
- Configuração de artefatos reprodutíveis no pipeline: Ajustar o pipeline para garantir que builds possam ser reproduzidos exatamente da mesma forma em diferentes momentos.
- Implementação de alertas proativos para falhas em pipelines: Configurar alertas que avisam a equipe quando certos thresholds de falha são atingidos, permitindo correções antes de problemas maiores surgirem.

13 (treze) Story Points

As tarefas de 13 Story Points em Implementação de Funcionalidades de DevOps e CI/CD envolvem um esforço mais elevado e modificações que impactam significativamente a automação, a segurança e o monitoramento dos processos

de CI/CD. Essas atividades demandam um planejamento cuidadoso e afetam múltiplos componentes do sistema, com a implementação de estratégias avançadas de deploy e validações complexas.

Tarefas deste nível incluem a criação de pipelines completos de CI/CD que gerenciam deploys para múltiplos ambientes (desenvolvimento, homologação, produção), utilizando técnicas como deploys blue-green, que garantem zero downtime. Além disso, podem incluir a configuração de pipelines com validações multi-stage, que realizam testes de integração, segurança e performance antes de qualquer deploy em produção. A integração com clusters Kubernetes para automatizar o deploy de microserviços também é comum nesse escopo, garantindo escalabilidade automática.

Tarefas típicas incluem a automatização de rollback em caso de falhas detectadas após o deploy e a implementação de pipelines que suportam múltiplos repositórios de código e múltiplas versões de uma aplicação, otimizando a colaboração e a eficiência do time de desenvolvimento. Essas tarefas são essenciais para garantir uma entrega contínua segura e eficiente, especialmente em ambientes que exigem alta disponibilidade e resiliência.

Exemplos:

- Implementação de um pipeline completo de CI/CD com múltiplos ambientes (desenvolvimento, homologação, produção): Criar um pipeline automatizado que gerencie deploys para diferentes ambientes com validações e testes automáticos.
- Configuração de deploys com zero downtime usando blue-green deployment: Implementar uma estratégia onde duas versões da aplicação coexistem, permitindo uma transição sem interrupções no serviço.
- Automatização de testes de carga e stress no pipeline: Configurar o pipeline para realizar testes automatizados de carga e stress antes de um deploy em produção, garantindo que a aplicação suporte altos volumes de tráfego.
- Implementação de rollback automatizado com monitoramento de health checks: Integrar a aplicação com um sistema de monitoramento que realiza rollback automaticamente em caso de falhas detectadas em health checks pós-deploy.
- Configuração de pipelines para integração com Kubernetes: Criar um pipeline que automatiza o deploy de aplicações em clusters Kubernetes, incluindo validações e ajustes de escalabilidade.

- Implementação de pipelines com múltiplos estágios de validação: Configurar pipelines que passam por várias camadas de validação, como testes de integração, segurança e performance, antes do deploy.
- Criação de um sistema de deploy contínuo com gates manuais: Implementar um processo de deploy que integra automação contínua com checkpoints manuais para aprovação em ambientes críticos.
- Automatização de auditorias de conformidade com frameworks de segurança: Configurar o pipeline para realizar verificações automáticas de conformidade com padrões de segurança, como OWASP Top 10, antes do deploy.
- Configuração de deploys com feature toggles: Implementar uma solução de feature toggles para que novas funcionalidades possam ser ativadas ou desativadas dinamicamente após o deploy.
- Implementação de pipelines que integram com múltiplos repositórios de código: Configurar pipelines que utilizam múltiplos repositórios de diferentes componentes, integrando e testando-os em conjunto.
- Automatização de gerenciamento de infraestrutura com Terraform ou Ansible: Configurar o pipeline para provisionar e gerenciar infraestrutura automaticamente utilizando ferramentas de IaC (Infrastructure as Code).
- Configuração de testes de segurança automatizados com integração de ferramentas como OWASP ZAP ou Snyk: Implementar a execução automática de testes de segurança e análise de vulnerabilidades no pipeline.
- Implementação de deploy com escalabilidade automática configurada via CI/CD: Criar um processo em que a aplicação é implantada com recursos escaláveis automaticamente com base no volume de tráfego.
- Automatização de pipeline com deploy em ambientes multicloud: Configurar pipelines que realizam deploys simultâneos ou alternados em diferentes provedores de nuvem, como AWS, Azure ou Google Cloud.
- Integração de pipelines com sistemas de gerenciamento de tickets para automatização de rastreamento: Configurar o pipeline para que, automaticamente, crie ou atualize tickets em sistemas como Jira ou Trello ao longo do processo de build e deploy.
- Implementação de monitoramento de logs centralizado com ELK Stack (Elasticsearch, Logstash, Kibana): Configurar a aplicação e o pipeline para enviar logs centralizados para análise em tempo real.
- Configuração de pipelines com deploys canários: Implementar deploys canários que entregam a aplicação para uma pequena porcentagem de usuários inicialmente, monitorando a estabilidade antes de expandir para o público total.

- Automação de validação de compatibilidade de dependências em múltiplos ambientes: Implementar uma verificação automática de compatibilidade de bibliotecas e dependências em diferentes ambientes de execução.
- Configuração de pipelines que suportam diferentes versões paralelas da aplicação: Configurar pipelines para que diferentes versões da aplicação possam ser testadas e mantidas em paralelo.
- Automação de builds de múltiplos módulos de uma aplicação monolítica: Implementar um pipeline que executa builds e testes de diferentes módulos de um sistema monolítico em paralelo para otimizar o processo.
- Implementação de integrações entre pipelines de CI/CD e sistemas de observabilidade: Integrar o pipeline com sistemas de monitoramento e alertas (ex.: Prometheus, Grafana), permitindo a análise contínua da performance da aplicação.
- Criação de um ambiente de staging replicando produção para testes finais antes do deploy: Automatizar a criação e destruição de ambientes de staging que replicam fielmente o ambiente de produção.
- Automação de pipelines para múltiplos microserviços: Criar pipelines que integram, testam e fazem o deploy de microserviços de forma automatizada e em paralelo.
- Configuração de autenticação e autorização em pipelines de CI/CD: Implementar controles de acesso no pipeline, garantindo que somente usuários autorizados possam realizar certas ações como aprovações e deploys.
- Implementação de versionamento semântico automático em artefatos: Configurar o pipeline para gerar versões automáticas seguindo o padrão de versionamento semântico (semver), baseado nas mudanças feitas no código.
- Integração de pipelines com gerenciamento de dependências e repositórios privados: Configurar o pipeline para buscar e validar dependências automaticamente em repositórios privados ou gerenciados.
- Automação de snapshots e restauração de ambientes: Configurar pipelines que tiram snapshots automáticos de ambientes antes de deploys e permitem restaurações rápidas em caso de falha.
- Implementação de políticas de escalabilidade dinâmica baseadas em métricas: Configurar pipelines que ajustam automaticamente a quantidade de recursos (CPU, memória) com base em métricas de performance.

- Integração de pipelines com testes em navegadores reais para validação de frontend: Configurar o pipeline para realizar testes de interface utilizando navegadores reais, como via Selenium ou Puppeteer.
- Automatização de builds e deploys com múltiplas arquiteturas (x86, ARM): Configurar o pipeline para suportar diferentes arquiteturas de hardware, garantindo compatibilidade para diferentes tipos de dispositivos.

Design e Experiência do Usuário (UX/UI)

O macro assunto Design e Experiência do Usuário (UX/UI) trata de atividades específicas relacionadas à usabilidade e à estética da interface das aplicações. Embora muitos aspectos de design e usabilidade já sejam considerados durante o desenvolvimento inicial, este macro assunto será utilizado para lidar com tarefas que envolvem adequações visuais específicas, alinhamento com as diretrizes de marca da PPSA, e ajustes na experiência do usuário sob a ótica da comunicação visual.

As atividades podem variar desde a implementação de alterações de cores e fontes, respeitando as diretrizes de identidade visual, até ajustes mais profundos na interface que garantam a consistência de navegação e a fluidez entre dispositivos móveis e desktops. Este macro assunto também aborda tarefas que visam melhorar a acessibilidade, como ajustes de contraste e espaçamento, além de otimizações voltadas para a experiência do usuário, como a adição de microinterações ou melhorias em elementos de navegação.

Dado que a PPSA já possui suas rotinas de design organizadas em outros macros assuntos, este tópico será aplicado principalmente em casos que exigem personalizações ou melhorias específicas, garantindo que a interface esteja em conformidade com os padrões visuais e de experiência definidos pela marca.

1 (um) Story Point

As tarefas de 1 Story Point em Design e Experiência do Usuário (UX/UI) são caracterizadas por ajustes visuais e de usabilidade de baixa complexidade, que podem ser implementados rapidamente sem impacto significativo na arquitetura da aplicação. Essas atividades têm como objetivo melhorar a interface ou a experiência do usuário de maneira pontual, seguindo as diretrizes de marca estabelecidas pela PPSA.

Os ajustes incluem modificações simples na interface, como a troca de ícones, ajustes de cores em conformidade com as diretrizes de identidade visual,

pequenas correções de alinhamento de texto ou imagens, e mudanças sutis no layout que visam melhorar a legibilidade ou acessibilidade. Além disso, essas tarefas podem abranger ajustes em botões, links ou componentes para otimizar a navegação, aplicando hover states ou melhorando a responsividade em diferentes dispositivos.

Esses ajustes incrementais visam garantir que a aplicação mantenha consistência visual e funcional, promovendo uma experiência mais fluida e acessível para os usuários sem a necessidade de grandes reformulações no design ou na estrutura da interface.

Exemplos:

- Ajuste de espaçamento entre elementos na página: Modificar o padding ou margin entre componentes para melhorar a disposição visual.
- Troca de ícones em botões ou menus: Substituir ícones por novos, atualizando o design sem alterar a funcionalidade.
- Ajuste de cores em botões e links: Mudar cores para atender diretrizes de branding ou melhorar a acessibilidade.
- Correção de alinhamento de textos e imagens: Ajustar o alinhamento para garantir consistência e legibilidade em diferentes dispositivos.
- Ajuste de fontes e tamanhos de texto: Modificar a tipografia para melhorar a hierarquia visual ou a legibilidade, alterando o tamanho ou estilo da fonte.
- Adição de tooltip simples: Incluir mensagens curtas que aparecem quando o usuário passa o mouse sobre um ícone ou botão.
- Implementação de um botão de "Voltar ao topo": Adicionar um botão fixo que permite ao usuário voltar ao topo da página facilmente.
- Correção de pequenas inconsistências no layout responsivo: Ajustar quebras de layout em telas menores, como dispositivos móveis.
- Troca de imagens de fundo ou banners: Atualizar imagens estáticas de fundo sem alterar a estrutura da página.
- Ajuste de contraste entre texto e fundo: Modificar cores para garantir acessibilidade, aumentando o contraste entre textos e o background.
- Aplicação de hover states em botões e links: Adicionar efeitos simples quando o usuário passa o mouse sobre botões ou links.
- Implementação de um ícone de carregamento (loading): Adicionar um ícone que indica que uma operação está em andamento.
- Adição de um rótulo de "Novo" em produtos ou seções: Inserir etiquetas visuais para destacar novos itens ou funcionalidades.

- Ajuste de responsividade para imagens: Garantir que as imagens redimensionem corretamente em diferentes tamanhos de tela.
- Adição de bordas ou sombras em cards ou componentes: Aplicar efeitos visuais simples como bordas ou sombras para dar destaque a elementos.
- Troca de ícones por versões SVG mais leves: Substituir ícones antigos por versões mais leves e otimizadas em SVG.
- Modificação de estilo de botões para diferentes estados (ativo, desabilitado): Ajustar a aparência de botões com base em seu estado atual (ex.: ativo, desabilitado).
- Aplicação de gradientes simples como fundo: Modificar o fundo de seções ou componentes para incluir gradientes de cores.
- Correção de largura e altura de componentes para consistência: Ajustar dimensões de botões, inputs ou outros elementos para garantir uniformidade visual.
- Adição de animações simples ao rolar a página: Implementar animações discretas, como fade-in, para elementos que aparecem ao rolar a página.
- Adição de indicadores de campo obrigatório em formulários: Colocar asteriscos ou outras marcas para identificar campos que são obrigatórios no preenchimento.
- Ajuste de labels em formulários para melhorar a clareza: Modificar textos de rótulos em campos de formulário para torná-los mais claros e descritivos.
- Aplicação de efeitos de sombra ou destaque em botões ao serem pressionados: Adicionar um efeito visual quando o botão é clicado para dar feedback ao usuário.
- Remoção de bordas ou estilos visuais ultrapassados: Simplificar o design ao remover bordas ou efeitos antigos que não são mais necessários.
- Adição de uma barra de progresso visual simples: Implementar uma barra que indica o progresso em formulários ou processos.
- Adição de placeholders descritivos em campos de entrada: Incluir exemplos de texto ou dicas dentro de campos de formulário.
- Correção de erros de espaçamento em listas ou tabelas: Ajustar o espaçamento entre itens de lista ou células de tabelas para melhor visualização.
- Implementação de ícones de rede social com links: Adicionar ícones de redes sociais na página com links para as respectivas plataformas.
- Ajuste de estilo de listas (bullets, numeração): Modificar o estilo de lista para atender ao design desejado, como mudar de pontos para quadrados.
- Adição de microinterações em botões: Implementar pequenas animações ou mudanças visuais ao clicar em botões.

- Ajuste de estilo para melhorar a legibilidade de longos blocos de texto: Modificar o espaçamento entre linhas e o alinhamento para facilitar a leitura de textos longos.
- Aplicação de estilos diferentes para links visitados: Modificar a cor ou estilo de links após terem sido visitados pelo usuário.
- Adição de efeitos de transição em menus suspensos: Implementar transições suaves quando menus dropdown são abertos ou fechados.
- Adição de rótulos descritivos em ícones sem texto: Incluir textos alternativos ou rótulos que descrevem a função dos ícones.
- Correção de erros menores de formatação em dispositivos móveis: Ajustar pequenos problemas de layout que surgem em smartphones ou tablets.
- Modificação de estilo de botões desabilitados: Alterar o visual de botões que estão desabilitados para torná-los visualmente diferentes dos ativos.
- Aplicação de animações simples ao clicar em botões: Adicionar uma leve animação ao pressionar botões, como um efeito de "empurrar" para baixo.
- Remoção de elementos que poluem a interface visualmente: Eliminar ícones ou imagens que não adicionam valor à interface ou que estão desatualizados.
- Adição de ícones de feedback em formulários (ex.: check de sucesso ou aviso de erro): Implementar ícones que indicam visualmente o sucesso ou erro ao preencher um campo de formulário.
- Ajuste de bordas arredondadas em botões ou inputs: Modificar o estilo para aplicar cantos arredondados em botões e campos de entrada.
- Aplicação de melhorias na responsividade de tabelas: Garantir que tabelas sejam exibidas corretamente em dispositivos móveis, com rolagem horizontal se necessário.
- Implementação de animações de feedback em campos de formulário: Adicionar pequenos efeitos visuais quando um campo de formulário é preenchido corretamente ou incorretamente.

5 (cinco) Story Points

Ajustes mais significativos que impactam tanto o visual quanto a usabilidade da interface. Essas tarefas podem incluir melhorias que exigem um planejamento mais cuidadoso e uma integração de elementos que melhoram a experiência do usuário em múltiplos dispositivos ou seções da aplicação.

As tarefas de 5 Story Points em Design e Experiência do Usuário (UX/UI) envolvem ajustes mais significativos que afetam tanto o visual quanto a usabilidade da interface. Embora não sejam extremamente complexas, essas atividades exigem

um planejamento mais cuidadoso e uma integração de elementos que melhorem a experiência do usuário em diferentes dispositivos ou em múltiplas seções da aplicação.

As tarefas podem incluir a criação de wireframes de média fidelidade para novos fluxos de usuário, revisão de fluxos de navegação entre dispositivos móveis e desktop para garantir uma experiência consistente, ou ajustes na hierarquia visual para melhorar a escaneabilidade das páginas. Além disso, elas podem abranger a implementação de feedback visual ao submeter formulários, como mensagens ou ícones que indicam sucesso ou erro, e o desenvolvimento de layouts responsivos para novas seções da aplicação.

Essas atividades focam em garantir que a interface ofereça uma experiência otimizada e intuitiva, ajustando detalhes que melhoram tanto a estética quanto a funcionalidade, especialmente em cenários que envolvem diferentes plataformas e dispositivos.

Exemplos:

- Criação de wireframes de média fidelidade para novos fluxos de usuário: Desenvolver esboços detalhados de novos fluxos ou funcionalidades que serão implementadas com foco na experiência de uso.
- Revisão de fluxos de navegação entre dispositivos móveis e desktop: Avaliar e ajustar a navegação para garantir que a experiência seja consistente e eficiente tanto em dispositivos móveis quanto em desktops.
- Implementação de feedback visual ao submeter um formulário: Adicionar mensagens ou indicadores visuais (como ícones de erro ou sucesso) que informam ao usuário sobre o status do envio de formulários.
- Criação de um layout responsivo para uma nova seção da aplicação: Desenvolver layouts que se adaptam de forma fluida a diferentes resoluções de tela, garantindo uma experiência otimizada em dispositivos móveis e desktops.
- Ajuste de hierarquia visual para melhorar a escaneabilidade: Reorganizar elementos na página para otimizar a hierarquia visual, facilitando a compreensão e navegação do usuário.
- Otimização de páginas para acessibilidade seguindo diretrizes WCAG: Implementar melhorias visuais e estruturais para garantir que as páginas sigam boas práticas de acessibilidade, como contraste de cores e navegação por teclado.

- Adição de suporte a temas claros e escuros (dark/light mode): Implementar a funcionalidade que permite ao usuário alternar entre modos de tema claro e escuro, sem prejudicar a usabilidade ou estética da aplicação.
- Criação de um sistema de breadcrumbs para melhorar a navegação: Desenvolver um componente de breadcrumbs que ajude os usuários a entender sua localização dentro da aplicação e facilite a navegação entre seções.
- Implementação de validação visual de campos em formulários: Adicionar validações visuais que indicam quando um campo foi preenchido corretamente ou quando há erros, ajudando o usuário a preencher o formulário de forma adequada.
- Criação de um guia de estilo visual para a interface: Documentar padrões visuais e guidelines para garantir que a interface mantenha consistência ao longo da aplicação.
- Ajuste de tabelas para garantir ordenação e filtragem de dados: Melhorar a interface das tabelas, permitindo que o usuário filtre e ordene informações de forma eficiente.
- Implementação de um mecanismo de busca interna com filtros básicos: Adicionar um campo de busca que permite ao usuário filtrar resultados em tempo real, melhorando a descoberta de conteúdo dentro da aplicação.
- Adição de microinterações para melhorar a experiência de navegação: Implementar animações ou transições sutis que proporcionam feedback visual ao usuário, como animações ao passar o mouse sobre botões ou links.
- Revisão de componentes visuais para garantir consistência no design: Avaliar e ajustar diferentes componentes da interface (botões, inputs etc.) para garantir consistência visual e de usabilidade.
- Otimização de imagens e gráficos para melhorar o tempo de carregamento: Implementar técnicas de compressão e formatos otimizados para garantir que as imagens carreguem rapidamente sem perder qualidade visual.
- Implementação de um sistema de tags interativas para facilitar a filtragem de conteúdo: Adicionar tags clicáveis que permitem ao usuário filtrar e descobrir conteúdos relacionados de maneira simples.
- Adição de uma seção de perguntas frequentes (FAQ) com navegação interativa: Implementar uma página de FAQ com uma navegação por tópicos que facilita o acesso às informações mais relevantes.

- Ajuste de formulários para garantir compatibilidade com dispositivos móveis: Melhorar a experiência de uso de formulários em telas pequenas, adaptando o layout e os campos para uma melhor usabilidade.

13 (treze) Story Points

As tarefas de 13 Story Points em Design e Experiência do Usuário (UX/UI) envolvem um nível de complexidade intermediário, onde as alterações exigem um planejamento mais detalhado e afetam diretamente a usabilidade e a navegação da aplicação. Essas atividades geralmente incluem ajustes ou redesenhos que impactam múltiplas seções da interface, além da necessidade de testes mais extensivos para garantir uma experiência fluida e consistente.

Esse tipo de tarefa pode incluir o desenvolvimento de protótipos interativos de alta fidelidade, revisões completas de navegação em processos críticos como checkout ou registro, ou a criação de layouts responsivos e dinâmicos para seções complexas da aplicação. Além disso, podem abranger a implementação de melhorias significativas na acessibilidade, seguindo diretrizes como as normas WCAG, ou o desenvolvimento de sistemas de navegação avançados, com menus suspensos e interações personalizadas para dispositivos móveis e desktop.

Essas atividades são cruciais para aprimorar a experiência do usuário em ambientes mais complexos e garantir que a interface esteja otimizada para diferentes plataformas, com foco tanto em estética quanto em funcionalidade.

Exemplos:

- Criação de um protótipo interativo de alta fidelidade para uma funcionalidade complexa: Desenvolver um protótipo interativo que simula a funcionalidade completa de uma parte importante da aplicação, permitindo testes de navegação e interações.
- Revisão completa de navegação e fluxo de usuários em um processo crítico: Analisar e melhorar o fluxo de usuários em processos críticos como checkout ou registro, garantindo uma navegação mais eficiente e com menos fricção.
- Criação de um layout responsivo para páginas multi-seções com conteúdo dinâmico: Desenvolver um layout flexível que se adapta a dispositivos móveis e desktops, garantindo a boa apresentação de conteúdo dinâmico, como blogs ou painéis de dados.

- Implementação de testes de usabilidade com usuários reais para ajustar a interface: Conduzir testes de usabilidade com usuários reais, coletar feedback e ajustar a interface com base nas observações para melhorar a experiência de uso.
- Desenvolvimento de um guia de estilo visual abrangente para todo o sistema: Criar diretrizes visuais que cubram toda a aplicação, incluindo regras para tipografia, paleta de cores, espaçamentos e componentes reutilizáveis.
- Criação de um componente de navegação avançada com submenus e interações personalizadas: Desenvolver uma barra de navegação com múltiplos níveis e interações dinâmicas, como menus suspensos e breadcrumbs, melhorando a navegabilidade.
- Redesenho de uma página crítica com foco em melhorar a taxa de conversão: Reestruturar páginas importantes, como a de checkout ou produto, utilizando princípios de design voltados para maximizar a conversão e reduzir o abandono.
- Implementação de um sistema de design com componentes reutilizáveis: Desenvolver uma biblioteca de componentes visuais que possam ser reutilizados em diferentes partes do sistema, garantindo consistência e rapidez nas implementações.
- Criação de um painel de controle interativo com visualização de dados e filtros dinâmicos: Desenvolver um painel (dashboard) que exibe gráficos e tabelas interativos, permitindo que o usuário aplique filtros e veja dados em tempo real.
- Análise e melhoria da acessibilidade de uma aplicação inteira com base nas diretrizes WCAG: Realizar uma auditoria da aplicação para garantir que ela segue as diretrizes de acessibilidade, ajustando cores, navegação por teclado e leitura de tela.
- Desenvolvimento de um fluxo de onboarding com tutoriais interativos: Criar um fluxo de onboarding que guia novos usuários pelas principais funcionalidades da aplicação, usando pop-ups e guias interativos.
- Otimização da interface para melhorar o desempenho em dispositivos móveis: Reestruturar componentes da interface e otimizar o carregamento de páginas para melhorar a performance e usabilidade em dispositivos móveis.
- Criação de um sistema de feedback do usuário com respostas automáticas e interações dinâmicas: Implementar um sistema onde os usuários podem enviar feedback e receber respostas automáticas ou em tempo real, melhorando a comunicação e experiência do usuário.

34 (trinta e quatro) Story Points

As tarefas de 34 Story Points em Design e Experiência do Usuário (UX/UI) envolvem atividades complexas que afetam profundamente a experiência do usuário e exigem a criação ou modificação de elementos significativos na interface. Essas tarefas exigem planejamento detalhado e frequentemente afetam várias áreas da aplicação, integrando múltiplos componentes para criar uma experiência coesa e otimizada, tanto em termos de usabilidade quanto de estética.

Neste nível de complexidade, as atividades podem incluir o redesenho completo de páginas ou de processos críticos, como um fluxo de checkout, onde todas as etapas precisam ser repensadas para melhorar a experiência do usuário e as taxas de conversão. Também podem abranger o desenvolvimento de sistemas de design complexos, garantindo consistência visual e eficiência de desenvolvimento em larga escala. Além disso, soluções robustas de acessibilidade que garantam a conformidade com as diretrizes WCAG, e a implementação de sistemas de navegação avançados, como menus dinâmicos ou breadcrumbs interativos, fazem parte deste escopo.

Essas tarefas têm impacto direto na usabilidade, performance e na forma como os usuários interagem com a aplicação, exigindo uma abordagem cuidadosa para garantir uma experiência de uso fluida e intuitiva.

Exemplos:

- Redesenho completo de uma aplicação ou sistema para otimização da experiência do usuário: Reestruturar o design de toda a aplicação com base em princípios de usabilidade, acessibilidade e estética, focando em melhorar o fluxo e a interação do usuário.
- Desenvolvimento de um sistema completo de design (design system) para padronização: Criar um sistema de design abrangente, incluindo tipografia, cores, componentes e padrões de interface, para garantir consistência visual e eficiência no desenvolvimento.
- Criação de uma jornada do usuário personalizada baseada em comportamento e preferências: Desenvolver uma experiência personalizada que se adapta ao comportamento do usuário, como recomendações de produtos, sugestões de navegação ou conteúdo dinâmico.

- Implementação de um painel de controle (dashboard) altamente interativo com análises avançadas: Desenvolver um dashboard que permite a visualização de dados complexos, com interações avançadas como filtros, gráficos dinâmicos e atualização em tempo real.
- Revisão e redesign de processos críticos com base em extensivos testes de usabilidade: Conduzir múltiplas rodadas de testes de usabilidade com usuários reais, analisar os resultados e aplicar melhorias significativas em processos importantes como checkout, registro ou configurações.
- Desenvolvimento de uma solução de acessibilidade completa para toda a aplicação: Implementar melhorias extensivas na acessibilidade, incluindo navegação por teclado, suporte a leitores de tela, e ajustes em cores e contrastes de acordo com as diretrizes WCAG.
- Implementação de uma experiência de navegação multi-dispositivo otimizada: Criar uma experiência fluida que se adapta perfeitamente entre desktop, tablets e dispositivos móveis, garantindo que o usuário tenha uma interface coerente e otimizada independentemente do dispositivo.
- Criação de um sistema de interação contextual (tooltips dinâmicos e tutoriais) para toda a aplicação: Desenvolver um sistema que orienta o usuário em tempo real através de dicas e tutoriais interativos, baseados no comportamento e nas ações do usuário dentro da aplicação.
- Otimização e redesign completo de páginas de conversão com foco em performance: Redesenhar páginas de alta conversão (como landing pages ou formulários de cadastro) com foco em otimizar o desempenho e reduzir o tempo de carregamento, sem sacrificar a experiência do usuário.
- Criação de um fluxo de login e autenticação multifatorial com UX otimizada: Desenvolver uma solução de login integrada com autenticação multifatorial (MFA), mantendo o foco em minimizar fricções para o usuário enquanto garante segurança.
- Desenvolvimento de um sistema de busca avançada com personalização baseada em AI: Implementar um sistema de busca que utiliza algoritmos de aprendizado para sugerir resultados personalizados e prever o que o usuário pode estar procurando.
- Otimização completa de uma aplicação com foco em SEO, performance e usabilidade: Reestruturar a aplicação para melhorar o ranqueamento em motores de busca, reduzir o tempo de carregamento e otimizar a experiência de uso para dispositivos móveis e desktops.
- Implementação de um fluxo de gamificação que impacta a navegação e experiência do usuário: Criar um sistema de gamificação que incentiva o

uso contínuo da aplicação, incluindo elementos como progressão, recompensas, desafios e pontuações.

- Redesenho e criação de uma aplicação com suporte a múltiplos idiomas e regiões: Desenvolver a interface para suportar múltiplos idiomas, garantindo que o design se adapte corretamente às diferentes localizações, preferências culturais e formatos regionais.
- Desenvolvimento de um sistema de pesquisa avançada com resultados instantâneos e filtros dinâmicos: Implementar um mecanismo de busca que oferece sugestões em tempo real e permite ao usuário refinar os resultados com filtros avançados e personalizáveis.

Documentação e Manuais de Usuário

macro assunto Documentação e Manuais de Usuário foca na necessidade de criar e manter uma documentação clara e organizada sobre o desenvolvimento de sistemas e suas funcionalidades. Esta documentação tem como objetivo garantir que a PPSA compreenda detalhadamente o trabalho realizado e possa facilmente repassar essas informações a outros profissionais que venham a assumir o projeto ou dar continuidade às atividades de desenvolvimento.

Além disso, esse macro assunto inclui a criação de manuais de usuário final, fornecendo instruções claras e acessíveis sobre o uso dos sistemas desenvolvidos. Esses manuais são essenciais para garantir que os usuários finais possam aproveitar todas as funcionalidades oferecidas pelos sistemas, de forma eficiente e intuitiva.

As atividades associadas à documentação podem incluir a descrição de processos técnicos complexos, fluxos de trabalho, integração de sistemas, bem como as melhores práticas para manutenção e evolução das soluções

Este macro assunto será aplicado sempre que houver uma necessidade específica de documentação aprofundada ou personalização de manuais, indo além das tarefas de documentação de rotina que fazem parte de outros macros assuntos.

1 (um) Story Point

As tarefas de 1 Story Point em Documentação e Manuais de Usuário envolvem a criação ou atualização de conteúdo técnico simples e direto. Essas atividades são de baixa complexidade e demandam pouco tempo e esforço, focando em ajustes pontuais ou documentações que não exigem pesquisa extensa ou análise profunda.

Essas tarefas podem incluir a criação de guias rápidos de instalação para ambientes de desenvolvimento, a adição de seções curtas em manuais técnicos que explicam novas funcionalidades ou ajustes em variáveis de ambiente, e a inserção de exemplos de uso básico de APIs ou bibliotecas. Também podem envolver atualizações menores em manuais existentes, como a correção de pequenos erros ou a inclusão de exemplos práticos para melhorar o entendimento dos usuários.

O objetivo dessas atividades é garantir que a documentação permaneça clara e acessível, cobrindo aspectos simples que facilitam a manutenção e o entendimento do sistema sem a necessidade de grandes reformulações.

Exemplos:

- Criação de um guia de instalação para ambiente de desenvolvimento: Desenvolver um guia simples que descreva os passos básicos para a instalação do ambiente de desenvolvimento, incluindo as dependências e ferramentas necessárias, sem entrar em detalhes avançados de configuração.
- Atualização de uma seção de troubleshooting técnico: Revisar e adicionar novas informações a uma seção de "Solução de Problemas" (troubleshooting), cobrindo problemas conhecidos na execução de scripts ou configuração de ambientes de desenvolvimento.
- Adição de instruções para configuração de uma variável de ambiente: Incluir uma breve explicação no manual técnico sobre como configurar uma nova variável de ambiente necessária para o sistema funcionar corretamente em ambiente de testes ou desenvolvimento.
- Criação de uma tabela com status de APIs: Elaborar uma tabela simples com a lista de APIs utilizadas pelo sistema, descrevendo brevemente o propósito de cada API e o status atual de operação (ativo/inativo), sem detalhamento técnico aprofundado.
- Atualização de instruções de deploy em ambiente de staging: Atualizar o manual técnico com pequenas modificações no processo de deploy em ambiente de homologação (staging), refletindo ajustes simples em um script ou comando utilizado.
- Inserção de exemplos de requisições de API: Adicionar exemplos de requisições e respostas de APIs RESTful ao manual técnico, utilizando dados estáticos e explicações simples sobre os endpoints principais.
- Criação de um guia rápido para uso do controle de versão (Git): Escrever um guia prático com os comandos básicos para uso do Git no

desenvolvimento, focando em instruções como "commit", "push", e "pull", sem abordar fluxos avançados de Git.

- Ajuste em documentação de nomenclatura de branches: Atualizar uma seção do manual que descreve a nomenclatura padronizada de branches para o repositório Git, refletindo pequenas mudanças no padrão de nomeação adotado pela equipe de desenvolvimento.
- Implementação de um exemplo de script de automação simples: Inserir um exemplo básico de um script de automação de tarefas rotineiras (como backup ou limpeza de cache) na documentação técnica, com explicação sobre o seu uso.
- Atualização do procedimento de rollback: Revisar e atualizar as instruções para execução de rollback (reversão de versão) em caso de falha no deploy, sem necessidade de incluir novos fluxos ou ferramentas.
- Documentação de dependências de bibliotecas: Criar uma seção no manual técnico listando as principais bibliotecas utilizadas no projeto, com uma breve descrição de cada uma e sua função no sistema.
- Atualização do manual sobre padrões de commits: Adicionar uma seção que descreva as boas práticas para mensagens de commits no controle de versão, com exemplos de mensagens claras e objetivas.
- Criação de um guia para execução de testes unitários: Desenvolver um guia simples que explica como rodar testes unitários no projeto, incluindo comandos básicos e onde encontrar os relatórios de execução.
- Inserção de exemplo de configuração de arquivo .env: Adicionar um exemplo básico de configuração de um arquivo .env no manual, explicando as principais variáveis de ambiente necessárias para rodar o sistema.
- Criação de um passo a passo para abrir tickets de bug: Escrever um guia prático para a equipe de desenvolvimento sobre como abrir tickets de bugs no sistema de controle de tarefas, especificando o formato ideal de descrição e os dados necessários.
- Atualização de instruções para migrações de banco de dados: Revisar e incluir novos comandos ou práticas recomendadas no manual técnico para realizar migrações de banco de dados de forma segura.
- Criação de exemplos de uso de containers Docker: Inserir no manual técnico exemplos básicos de comandos Docker para criar e gerenciar containers, focado em desenvolvedores que utilizam a ferramenta para isolar ambientes.
- Adição de instruções de integração contínua (CI): Atualizar o manual com instruções sobre como configurar pipelines de integração contínua,

incluindo comandos e regras básicas para verificar a integração de novas funcionalidades.

- Inserção de boas práticas de refatoração de código: Adicionar uma seção ao manual técnico sobre boas práticas de refatoração, explicando quando e como realizar melhorias em código legado sem introduzir novos bugs.
- Criação de um checklist para revisão de código: Desenvolver um checklist simples com os itens principais que devem ser verificados durante uma revisão de código (code review), incluindo padrões de estilo, cobertura de testes e clareza de lógica.
- Adição de uma seção no manual sobre um novo módulo desenvolvido: Documentar brevemente a funcionalidade de um novo módulo ou componente desenvolvido, explicando seu propósito e como ele se integra ao restante do sistema.
- Atualização do manual com exemplos de uso de uma nova API desenvolvida: Incluir exemplos de requisições e respostas para uma API que foi recentemente desenvolvida, detalhando os principais endpoints e parâmetros aceitos.
- Criação de um guia rápido de uso para uma nova funcionalidade: Desenvolver um tutorial curto e direto, com linguagem acessível, explicando como utilizar uma nova funcionalidade do sistema. O guia incluirá instruções passo a passo e capturas de tela, cobrindo apenas as ações mais básicas e frequentes que o usuário final precisará realizar.

5 (cinco) Story Points

As tarefas de 5 Story Points no macro assunto Documentação e Manuais de Usuário envolvem atividades que, embora ainda relativamente simples, requerem um pouco mais de esforço e tempo em comparação às de 1 Story Point. Aqui, a complexidade começa a aumentar, demandando um nível moderado de atenção e, ocasionalmente, alguma pesquisa complementar ou consultas rápidas para garantir a precisão das informações.

Essas tarefas podem incluir a criação ou atualização de documentações mais detalhadas, como a elaboração de manuais com seções explicativas que cobrem múltiplas funcionalidades de um sistema ou a criação de tutoriais práticos que detalham cenários de uso específicos. Também podem envolver a revisão de conteúdos mais extensos em manuais de usuário, como atualizações em fluxos de trabalho, ajustes mais amplos em descrições técnicas, ou ainda a documentação de integrações de ferramentas com algum nível de complexidade. A inclusão de instruções técnicas sobre configurações que envolvem múltiplas

etapas ou a adição de exemplos de uso mais complexos de APIs também pode se enquadrar nesse nível.

Essas atividades exigem um pouco mais de planejamento e clareza, visando garantir que os usuários possam executar tarefas com base nas informações fornecidas de forma autônoma e sem maiores dificuldades. Embora ainda não envolvam grandes reformulações ou um esforço extensivo, elas demandam mais atenção aos detalhes e uma compreensão mais sólida das funcionalidades abordadas.

Exemplos:

- Criação de um guia completo de integração contínua (CI/CD): Documentar todos os passos necessários para configurar e manter pipelines de integração e entrega contínua, cobrindo desde a criação do pipeline até testes automáticos e deploys.
- Elaboração de um manual detalhado sobre migração de dados: Criar um documento que explica os processos de migração de dados entre diferentes versões do sistema ou entre bancos de dados, detalhando comandos, precauções e estratégias para evitar perdas de dados.
- Desenvolvimento de um manual para a configuração de ambientes de desenvolvimento complexos: Criar uma documentação que descreve a configuração de ambientes de desenvolvimento com múltiplos serviços e dependências, como Docker, Kubernetes ou micros serviços.
- Documentação de um novo processo de deploy multi-ambiente: Criar um guia que cobre o fluxo de deploy para múltiplos ambientes (desenvolvimento, homologação, produção), explicando as diferenças entre cada um e as etapas específicas de cada deploy.
- Criação de um guia de boas práticas de segurança no desenvolvimento: Elaborar um manual que descreva as melhores práticas de segurança para o desenvolvimento de software, como o uso correto de autenticação, criptografia, e prevenção de vulnerabilidades.
- Criação de um guia passo a passo sobre o uso de uma nova funcionalidade complexa: Elaborar um tutorial detalhado com instruções para o uso de uma nova funcionalidade que envolva múltiplos passos, integrando diferentes partes do sistema, com capturas de tela ou exemplos práticos.
- Criação de um manual explicativo para um sistema de permissões complexo: Desenvolver uma documentação técnica detalhada que explique como o novo sistema de permissões foi implementado, incluindo

como configurar diferentes níveis de acesso, controle por papéis e auditoria de permissões.

- Desenvolvimento de um manual completo sobre o uso de uma nova funcionalidade complexa: Criar um manual detalhado e estruturado, com múltiplas seções, que explica ao usuário final como utilizar todas as funcionalidades de uma nova área ou módulo do sistema. O documento incluirá instruções passo a passo, exemplos de uso prático, capturas de tela e seções de troubleshooting para ajudar o usuário em casos de dúvidas ou erros.

Fluxo de utilização do roteiro de métricas

Ao receber uma nova demanda de desenvolvimento, a CONTRATADA deverá realizar um processo estruturado para estimar o esforço necessário, seguindo um fluxo de decisão claro e objetivo. Primeiramente, a demanda deve ser analisada detalhadamente, com o objetivo de identificar todos os componentes e funcionalidades envolvidas no projeto. Nesta fase inicial, é essencial que a CONTRATADA compreenda completamente as necessidades do cliente, avaliando quais áreas do sistema serão afetadas e quais funcionalidades precisarão ser desenvolvidas ou integradas.

Com essa análise inicial realizada, a demanda deverá ser quebrada em macros assuntos relevantes para o desenvolvimento, conforme os tópicos estabelecidos, como por exemplo: Desenvolvimento de Backend, Integração de APIs e Serviços Externos, Design UX/UI, Testes, entre outros. Cada uma dessas categorias representa um grande conjunto de atividades e responsabilidades dentro do escopo do projeto. É importante que a CONTRATADA avalie quais macros assuntos se aplicam àquela demanda. Caso algum macro assunto não seja relevante para o desenvolvimento solicitado, ele deverá ser descartado.

Após a identificação dos macros assuntos aplicáveis, a CONTRATADA deverá tomar uma decisão sobre a relevância de cada um. Para cada macro assunto, a equipe de desenvolvimento precisará verificar se as atividades associadas fazem parte do trabalho necessário para entregar a demanda. Apenas os macros assuntos que forem considerados realmente necessários devem ser incluídos na estimativa de esforço. Qualquer macro assunto que não se aplique àquela demanda deve ser ignorado, mantendo o foco apenas nos aspectos diretamente ligados ao desenvolvimento solicitado.

A CONTRATADA deverá escolher um único grau de esforço para cada um deles. Esse esforço será medido em Story Points, de acordo com a escala pré-definida. A escolha do esforço deverá refletir a complexidade, o tempo necessário para conclusão e os riscos associados àquela parte do trabalho.

Após definir o grau de esforço para cada macro assunto envolvido na demanda, a CONTRATADA deverá somar os Story Points de todos os macros assuntos aplicáveis. Essa soma representará o total de Story Points estimados para a realização da demanda, fornecendo uma visão clara do esforço necessário para concluir o projeto. Essa etapa final do processo visa garantir que a equipe tenha uma estimativa clara e objetiva do trabalho a ser feito, permitindo um planejamento eficaz e previsível.

Casos de uso do Roteiro de Métricas

(Todas as informações nos casos abaixo são altamente hipotéticas, assim como suas ferramentas e necessidades, sendo de uso exclusivo para o entendimento do roteiro de métricas. Deve-se considerar que o escopo das demandas, assim como sua avaliação, não inclui **todos** os detalhes técnicos que, na prática, serão levados em conta para a avaliação do esforço final.)

Criação de uma nova aplicação web para a PPSA dentro da intranet

A proposta é desenvolver uma aplicação web para uso interno, onde os funcionários da PPSA possam fazer login utilizando seu usuário e senha do Active Directory (AD). Após a autenticação, o sistema exibirá para o usuário suas últimas folhas de pagamento, extraídas diretamente do ERP SAP. O usuário terá a opção de visualizar e baixar essas folhas de pagamento em formato PDF.

O design da página deverá seguir a identidade visual da PPSA, incluindo cores e fontes padronizadas. Além disso, a aplicação precisará contar com um menu interativo para facilitar a navegação.

É essencial que o desenvolvimento da aplicação inclua testes rigorosos para garantir a funcionalidade e evitar a entrega de um produto com problemas. Dada a urgência, a demanda será dividida em duas partes, e cada etapa deverá passar por uma fase de testes antes de avançar para a próxima.

Avaliação assuntos:

- **Desenvolvimento de Funcionalidades de Frontend → Sim**

A interface da aplicação (login, exibição de folhas de pagamento, visualização e download em PDF) será implementada no frontend. Além disso, é mencionado o uso de um menu interativo e a aplicação da identidade visual da PPSA, o que envolve desenvolvimento frontend.

- **Desenvolvimento de Funcionalidades de Backend → Sim**

A autenticação dos funcionários via Active Directory (AD) e a integração para extração das folhas de pagamento do ERP SAP são funcionalidades

típicas do backend. O backend também será responsável por lidar com as requisições, realizar autenticação e comunicação com o ERP.

- **Integração de APIs e Serviços Externos → Sim**

Haverá a necessidade de integrar o sistema ao AD para autenticação e ao ERP SAP para obter as folhas de pagamento. Isso envolve comunicação via APIs e serviços externos, essencial para o funcionamento da aplicação.

- **Design e Experiência do Usuário (UX/UI) → Sim**

A interface deve seguir a identidade visual da PPSA, respeitando cores e fontes padronizadas, o que exige atenção ao design e à experiência do usuário. A implementação de um menu interativo para facilitar a navegação também faz parte do escopo de UX/UI.

- **Design e Arquitetura de Banco de Dados → Não**

Não há menção de criação ou modificação de banco de dados nesta demanda. As informações de folhas de pagamento serão extraídas diretamente do ERP SAP e não armazenadas localmente, dispensando a necessidade de design ou arquitetura de banco de dados.

- **Implementação de Funcionalidades de Segurança → Não**

O macro assunto "Segurança" geralmente cobre medidas avançadas como autenticação multifator ou proteção contra-ataques complexos. Como a autenticação será gerida via Active Directory (AD) e as práticas comuns de segurança podem ser incluídas no backend, esse macro assunto não se aplica aqui. As medidas básicas de segurança (como criptografia de dados e autenticação) já estão cobertas pelo backend e integração de APIs.

- **Testes e Garantia de Qualidade → Sim**

O desenvolvimento será dividido em duas etapas, e cada uma deve passar por testes rigorosos para garantir a funcionalidade correta e evitar a entrega de um produto com problemas. A exigência de testes detalhados indica a necessidade de práticas formais de qualidade.

- **Implementação de Funcionalidades de DevOps e CI/CD → Não**
Não há menção de automação de pipelines, integração contínua ou práticas DevOps específicas nesta demanda. O foco está no desenvolvimento e na funcionalidade final, não em processos de deploy automatizados ou CI/CD.
- **Otimização de Performance e Escalabilidade → Não**
O escopo da demanda não menciona preocupação com performance ou escalabilidade. O foco está na funcionalidade, e não foram identificados requisitos de otimização ou escalabilidade que exijam atenção especial neste momento.
- **Documentação e Manuais de Usuário → Não**
Não há menção da criação de documentação técnica ou manuais de usuário como parte do escopo desta demanda.

Estimativa dos esforços:

- **Desenvolvimento de Funcionalidades de Frontend → 5 Story Points**
Justificativa: A criação de uma interface de login, exibição de folhas de pagamento e o design alinhado à identidade visual da PPSA são tarefas de baixa complexidade, sem grandes impactos estruturais, como descrito anteriormente.
- **Desenvolvimento de Funcionalidades de Backend → 13 Story Points**
Justificativa: A integração com o Active Directory para autenticação e a extração de dados do ERP SAP envolvem lógica de backend mais complexa, incluindo autenticação, comunicação com APIs e manipulação de dados. Essas tarefas exigem um esforço intermediário, pois envolvem a criação de módulos que lidam com a segurança e fluxo de dados.
- **Integração de APIs e Serviços Externos → 13 Story Points**
Justificativa: A integração com o Active Directory para autenticação e o ERP SAP para obter folhas de pagamento é uma tarefa de complexidade

intermediária. Envolve a criação de rotas para comunicação entre sistemas e o tratamento adequado das respostas dessas APIs externas.

- Design e Experiência do Usuário (UX/UI) → **5 Story Points**

Justificativa: Seguir a identidade visual da PPSA (cores e fontes) e implementar um menu interativo são ajustes visuais simples e de baixa complexidade, sem exigir uma reformulação completa de usabilidade ou design.

- Testes e Garantia de Qualidade → **1 Story Point**

Justificativa: Os testes necessários são muito simples, limitando-se à verificação do login e à validação da funcionalidade de visualização e download das folhas de pagamento. Isso não envolve múltiplas camadas de validação ou testes complexos, sendo um teste funcional básico, focado em assegurar que o login funcione corretamente e que o download dos documentos ocorra conforme esperado.

Total para demanda: $5+13+13+5+1 = 37$ **Story Points**

Criação de uma nova funcionalidade na aplicação web para a PPSA dentro da intranet.

A PPSA solicitou uma adição à aplicação web previamente desenvolvida para consulta das folhas de pagamento dos funcionários. Agora, a aplicação deverá incluir uma nova funcionalidade que permitirá ao funcionário verificar se possui direito a férias no ano corrente. A nova página será integrada ao sistema existente, utilizando as funcionalidades já disponíveis, como a autenticação via Active Directory (AD) e a interface baseada no design da folha de pagamento.

A consulta das informações de férias será feita diretamente no sistema SAP, que já realiza o cálculo e retorna apenas duas opções: "SIM" ou "NÃO". Não haverá necessidade de desenvolver novas telas de login ou criar outra conexão com o ERP, visto que o sistema existente já faz essas integrações. No entanto, será necessário configurar a comunicação com o SAP para que a aplicação possa consultar a informação específica de direito a férias. A resposta será exibida na

interface de forma simples, e a estrutura visual já estabelecida será reaproveitada, sem a necessidade de mudanças significativas no design.

Além disso, o desenvolvimento desta funcionalidade não exige a implementação de downloads ou ajustes avançados de segurança, uma vez que a arquitetura básica da aplicação já contempla esses requisitos. É necessário testar aplicação antes da sua entrega.

Avaliação assuntos:

- **Desenvolvimento de Funcionalidades de Frontend → Sim**

Será necessário realizar ajustes na interface do usuário para exibir as informações de férias, aproveitando o design já existente da página de folha de pagamento. Essa atividade envolve alterações simples na exibição de dados.

- **Desenvolvimento de Funcionalidades de Backend → Sim**

A aplicação precisará consultar o sistema SAP para obter as informações sobre o direito a férias, sendo necessário implementar essa lógica no backend. Isso envolve integração com o sistema existente e comunicação com o SAP.

- **Integração de APIs e Serviços Externos → Sim**

A nova funcionalidade depende da integração com o SAP, que fornecerá a resposta sobre o direito a férias do funcionário. Essa integração será feita via API, utilizando serviços externos já existentes.

- **Design e Experiência do Usuário (UX/UI) → Sim**

Pequenos ajustes visuais serão realizados na interface existente para acomodar a exibição das informações de férias. O design reaproveitará a estrutura atual, sem a necessidade de reformulações significativas.

- **Design e Arquitetura de Banco de Dados → Não**

Não será necessário desenvolver ou modificar banco de dados, uma vez que as informações serão consultadas diretamente do SAP e não haverá armazenamento local.

- **Implementação de Funcionalidades de Segurança → Não**

As funcionalidades de segurança já implementadas, como a autenticação via Active Directory, serão mantidas. Não há necessidade de desenvolver novas medidas de segurança específicas para essa funcionalidade.

- **Testes e Garantia de Qualidade → Sim**

Testes serão necessários para validar a integração com o SAP e a correta exibição das informações de férias na interface do usuário. Os testes incluirão verificações funcionais para garantir a precisão da consulta e resposta.

- **Implementação de Funcionalidades de DevOps e CI/CD → Não**

A demanda não menciona ajustes nos pipelines de integração contínua ou deploy contínuo. Não há necessidade de implementar automações adicionais nesse sentido.

- **Otimização de Performance e Escalabilidade → Não**

O escopo da demanda não menciona preocupação com performance ou escalabilidade. O foco está na funcionalidade, e não foram identificados requisitos de otimização ou escalabilidade que exijam atenção especial neste momento.

- **Documentação e Manuais de Usuário → Não**

Não foi solicitado desenvolvimento de documentação técnica adicional ou manuais de usuário para essa funcionalidade.

Estimativa dos esforços:

- **Desenvolvimento de Funcionalidades de Frontend → 2,5 Story Points**
Será necessário realizar ajustes na interface do usuário para exibir as informações de férias, aproveitando o design já existente da página de folha de pagamento. Essa atividade envolve alterações simples na exibição de dados, sem grandes impactos visuais ou estruturais.
Reaproveitamento de Código: Como o design da folha de pagamento será reaproveitado, incluindo a estrutura visual, o esforço original de 5 Story Points é reduzido em 50%, resultando em 2,5 Story Points.
- **Desenvolvimento de Funcionalidades de Backend → 6,5 Story Points**
A aplicação precisará consultar o sistema SAP para obter as informações sobre o direito a férias, sendo necessário implementar essa lógica no backend. Isso inclui a comunicação com a API do SAP, que exige um esforço moderado de integração e validação.
Reaproveitamento de Código: A lógica de backend para autenticação e outras integrações já está implementada, e a nova integração reutiliza boa parte da infraestrutura existente, resultando na redução de 50% no esforço original de 13 Story Points, ficando em 6,5.
- **Integração de APIs e Serviços Externos → 6,5 Story Points**
A nova funcionalidade depende da integração com o SAP, que fornecerá a resposta sobre o direito a férias do funcionário. Essa integração será feita via API, utilizando serviços externos já existentes.
Reaproveitamento de Código: Como a integração com o ERP já foi realizada anteriormente, o esforço para configurar a integração com o SAP segue o mesmo padrão, sendo reduzido em 50% do valor original de 13 Story Points para 6,5.
- **Design e Experiência do Usuário (UX/UI) → 0,5 Story Point**
Pequenos ajustes visuais serão realizados na interface existente para acomodar a exibição das informações de férias. O design reaproveitará a estrutura atual, sem a necessidade de reformulações significativas.
Reaproveitamento de Código: Como o design será praticamente o mesmo da página de folha de pagamento, o esforço de 1 Story Point é reduzido para 0,5 devido ao reaproveitamento.
- **Testes e Garantia de Qualidade → 1 Story Point**

Testes serão necessários para validar a integração com o SAP e a correta exibição das informações de férias na interface do usuário. Os testes incluirão verificações funcionais para garantir a precisão da consulta e resposta. Não há reaproveitamento aqui, então o esforço continua sendo 1 Story Point.

Total para demanda: $2,5 + 6,5 + 6,5 + 0,5 + 1 = 17$ Story Points

Criação de uma nova página no site da PPSA para divulgação de evento

A PPSA solicitou a criação de uma página nova em seu site institucional para a divulgação de um evento especial. Esta página deve incluir um carrossel de imagens com os melhores momentos da PPSA no último ano, com transições suaves de "fade". Além disso, a página contará com dois vídeos interativos, que serão exibidos através de pop-ups ao clicar em textos específicos. Os vídeos serão carregados diretamente do YouTube.

Outro elemento importante da página é a integração de um painel interativo de Power BI, que apresentará dados da produção estimada da PPSA para os próximos 10 anos. Esse painel será desenvolvido do zero, com base em dados fornecidos em uma planilha Excel pela PPSA. O painel permitirá ao usuário filtrar as informações por contrato específico (CPP) ou visualizar todos os dados ao mesmo tempo. A identidade visual da PPSA deve ser mantida, incluindo o design da página e do painel Power BI.

Essa demanda não aproveita nenhum código existente, sendo desenvolvida totalmente do zero.

Avaliação assuntos:

- **Desenvolvimento de Funcionalidades de Frontend → Sim**

O desenvolvimento do carrossel de imagens, dos pop-ups interativos para os vídeos e a implementação do painel Power BI são funcionalidades diretamente ligadas ao frontend. Também será necessário garantir que o design siga o padrão visual da PPSA.

- **Desenvolvimento de Funcionalidades de Backend → Não**

Não há funcionalidades de backend envolvidas, já que os dados do painel virão de uma planilha Excel e não há processamento ou manipulação de dados no servidor.

- **Integração de APIs e Serviços Externos → Sim**

A integração com o YouTube, para exibir vídeos nos pop-ups, e com o Power BI, para o painel de dados, são elementos essenciais dessa demanda. Essas integrações garantem a funcionalidade interativa da página.

- **Design e Experiência do Usuário (UX/UI) → Sim**

O design da página e a experiência do usuário precisam seguir as diretrizes de identidade visual da PPSA, assegurando uma navegação fluida e interativa, especialmente no uso do carrossel de imagens e nos pop-ups.

- **Design e Arquitetura de Banco de Dados → Não**

Não será necessária a criação ou manipulação de bases de dados, pois os dados serão fornecidos diretamente por meio de planilhas e visualizados no Power BI.

- **Implementação de Funcionalidades de Segurança → Não**

Não há necessidade de medidas de segurança avançadas, como autenticação ou criptografia, já que a demanda se concentra em funcionalidades de frontend e exibição de dados estáticos.

- **Testes e Garantia de Qualidade → Sim**

Serão necessários testes básicos para garantir que o carrossel, os pop-ups e o painel Power BI funcionem corretamente. Esses testes validarão a interação da página e a funcionalidade dos elementos principais.

- **Implementação de Funcionalidades de DevOps e CI/CD → Não**

Não foi mencionada a necessidade de automação de deploys ou pipelines de CI/CD para esta demanda.

- **Otimização de Performance e Escalabilidade → Não**

Não há menção de requisitos específicos relacionados à escalabilidade ou performance, além da necessidade de garantir o funcionamento básico da página.

- **Documentação e Manuais de Usuário → Sim**

Será necessária a criação de documentação detalhada, tanto para descrever o desenvolvimento e as integrações realizadas, quanto para fornecer um manual de manutenção futura da página.

Estimativa dos esforços:

- **Desenvolvimento de Funcionalidades de Frontend → 13 Story Points**

A criação do carrossel de imagens, pop-ups para vídeos e integração do painel Power BI exigem um esforço considerável em frontend, além de garantir que o design siga a identidade visual da PPSA.

- **Integração de APIs e Serviços Externos → 1 Story Points**

A integração com o YouTube para vídeos e com o Power BI para o painel interativo envolve a configuração de APIs e serviços externos para que esses elementos funcionem corretamente de baixa complexidade.

- **Design e Experiência do Usuário (UX/UI) → 5 Story Points**

Ajustes visuais e de usabilidade seguindo as diretrizes da PPSA, com foco na apresentação do carrossel, nos pop-ups e no painel Power BI, garantindo uma navegação intuitiva.

- **Testes e Garantia de Qualidade → 1 Story Point**

Testes simples para garantir que as funcionalidades interativas (carrossel, pop-ups, painel) estão funcionando corretamente nos principais navegadores e dispositivos.

- **Documentação e Manuais de Usuário → 5 Story Points**

Criação de documentação técnica detalhada, incluindo descrições das integrações, além de um manual para manutenção futura da página e seus componentes.

Total estimado: $13 + 1 + 5 + 1 + 5 = 25$ **Story Points**

Criação de 3 novas páginas no site da PPSA para eventos

Após a entrega da primeira página de divulgação de evento, a comunicação da PPSA solicitou a criação de mais três páginas no mesmo formato, com carrossel de imagens, pop-ups interativos para vídeos do YouTube e um painel Power BI exibindo dados de produção.

As alterações entre as páginas estão restritas às imagens do carrossel, os textos, os links dos vídeos do YouTube e os arquivos Excel para o painel Power BI. Todo o restante da estrutura será reaproveitado do projeto anterior.

Avaliação assuntos:

- **Desenvolvimento de Funcionalidades de Frontend → Sim**

O desenvolvimento do carrossel de imagens, dos pop-ups interativos para os vídeos e a implementação do painel Power BI são funcionalidades diretamente ligadas ao frontend. Também será necessário garantir que o design siga o padrão visual da PPSA.

- **Desenvolvimento de Funcionalidades de Backend → Não**

Não há funcionalidades de backend envolvidas, já que os dados do painel virão de uma planilha Excel e não há processamento ou manipulação de dados no servidor.

- **Integração de APIs e Serviços Externos → Sim**

A integração com o YouTube, para exibir vídeos nos pop-ups, e com o Power BI, para o painel de dados, são elementos essenciais dessa demanda. Essas integrações garantem a funcionalidade interativa da página.

- **Design e Experiência do Usuário (UX/UI) → Sim**

O design da página e a experiência do usuário precisam seguir as diretrizes de identidade visual da PPSA, assegurando uma navegação fluida e interativa, especialmente no uso do carrossel de imagens e nos pop-ups.

- **Design e Arquitetura de Banco de Dados → Não**

Não será necessária a criação ou manipulação de bases de dados, pois os dados serão fornecidos diretamente por meio de planilhas e visualizados no Power BI.

- **Implementação de Funcionalidades de Segurança → Não**
Não há necessidade de medidas de segurança avançadas, como autenticação ou criptografia, já que a demanda se concentra em funcionalidades de frontend e exibição de dados estáticos.
- **Testes e Garantia de Qualidade → Sim**
Serão necessários testes básicos para garantir que o carrossel, os pop-ups e o painel Power BI funcionem corretamente. Esses testes validarão a interação da página e a funcionalidade dos elementos principais.
- **Implementação de Funcionalidades de DevOps e CI/CD → Não**
Não foi mencionada a necessidade de automação de deploys ou pipelines de CI/CD para esta demanda.
- **Otimização de Performance e Escalabilidade → Não**
Não há menção de requisitos específicos relacionados à escalabilidade ou performance, além da necessidade de garantir o funcionamento básico da página.
- **Documentação e Manuais de Usuário → Sim**
Será necessária a criação de documentação detalhada, tanto para descrever o desenvolvimento e as integrações realizadas, quanto para fornecer um manual de manutenção futura da página.

Estimativa dos esforços:

- **Desenvolvimento de Funcionalidades de Frontend → 13 Story Points**
A criação do carrossel de imagens, pop-ups para vídeos e integração do painel Power BI exigem um esforço considerável em frontend, além de garantir que o design siga a identidade visual da PPSA. **(Reaproveitamento de código aplicado)**
- **Integração de APIs e Serviços Externos → 1 Story Points**
A integração com o YouTube para vídeos e com o Power BI para o painel interativo envolve a configuração de APIs e serviços externos para que esses elementos funcionem corretamente de baixa complexidade. **(Reaproveitamento de código aplicado)**

- Design e Experiência do Usuário (UX/UI) → 5 Story Points
Ajustes visuais e de usabilidade seguindo as diretrizes da PPSA, com foco na apresentação do carrossel, nos pop-ups e no painel Power BI, garantindo uma navegação intuitiva. **(Reaproveitamento de código aplicado)**
- Testes e Garantia de Qualidade → 1 Story Point
Testes simples para garantir que as funcionalidades interativas (carrossel, pop-ups, painel) estão funcionando corretamente nos principais navegadores e dispositivos.
- Documentação e Manuais de Usuário → 5 Story Points
Criação de documentação técnica detalhada, incluindo descrições das integrações, além de um manual para manutenção futura da página e seus componentes. **(Reaproveitamento de código aplicado) – A mesma documentação será usada, apenas modificando a origem dos dados.**

Total estimado (com reaproveitamento de código): **$(6.5 + 0.5 + 2.5) + 1 + (2.5) = 13$ Story Points**

Aplicando o conceito de repetição para mais 2 páginas:
 $13 + (13 \times 0.10 \times 2) = 15.6$ Story Points