Oficina SAEP 14-08-2024

1. UI/UX: Consistência Visual e Coerência - Teoria: A consistência visual é essencial para criar uma interface intuitiva e fácil de navegar. Elementos de interface, como botões e ícones, devem ter um design consistente em toda a aplicação para evitar confusão e melhorar a experiência do usuário. - Situação-problema: Um cliente está insatisfeito com a interface de um aplicativo que você desenvolveu. Ele menciona que os botões de ação têm estilos diferentes em diferentes telas, o que está causando confusão entre os usuários. Alguns botões estão em azul e outros em verde, sem um padrão claro. Pergunta: Como você ajustaria o design para garantir consistência visual em todo o aplicativo?

a) Usar um estilo de botão consistente em todas as telas, aplicando as mesmas cores e tamanhos.
b) Manter as cores diferentes dos botões para diversificar o design.
c) Criar uma paleta de cores aleatória para cada tela.
d) Manter os estilos diferentes para destacar a originalidade do design.

2. UI/UX: Hierarquia de Informação - Teoria: A hierarquia de informação ajuda a guiar o olhar do usuário para os elementos mais importantes em uma interface. Usando tamanhos de fonte, cores e espaçamentos adequados, você pode garantir que os usuários foquem nas informações prioritárias. - Situação-problema: Os usuários de uma plataforma de notícias relataram que estão tendo dificuldade para localizar informações importantes nas páginas de artigo, pois o título, subtítulo e o corpo do texto parecem ter o mesmo peso visual. - Pergunta: Qual estratégia você aplicaria para melhorar a hierarquia de informação na interface?

a) Usar o mesmo tamanho de fonte para todas as informações, criando consistência visual.
b) Aumentar o tamanho da fonte do título e subtítulo, além de usar cores que contrastem com o corpo do texto.
c) Diminuir o tamanho da fonte do título e aumentar o corpo do texto para melhorar a legibilidade.
d) Remover o subtítulo e deixar apenas o corpo do texto para não distrair o usuário.

3. UI/UX: Testes de Usabilidade - Teoria: Testes de usabilidade são fundamentais para identificar problemas na interação dos usuários com a interface. Durante esses testes, observamos como os usuários navegam pelo sistema, buscando pontos de fricção e áreas onde o fluxo de trabalho pode ser otimizado. - Situação-problema: Durante os testes de usabilidade de um aplicativo de e-commerce, vários usuários mencionaram que o processo de checkout é confuso, pois o botão "Finalizar Compra" está oculto em

uma aba diferente da esperada. - Pergunta: O que você faria para resolver o problema identificado nos testes de usabilidade?

a) Manter o botão "Finalizar Compra" na aba oculta, já que faz parte do design original.
b) Mover o botão "Finalizar Compra" para uma posição mais visível na página principal de checkout.
c) Retirar o botão "Finalizar Compra" e solicitar que os usuários concluam a compra por meio de um chat.
d) Adicionar mais abas para organizar o processo de checkout.

4. UI/UX: Microinterações - Teoria: As microinterações são pequenos detalhes de interação que melhoram a experiência do usuário, como animações ao clicar em botões ou alertas visuais ao cometer erros. Elas fazem a interface parecer mais fluida e responsiva. - Situação-problema: Em um sistema de envio de mensagens, os usuários estão relatando que não sabem se suas mensagens foram enviadas com sucesso, já que não há nenhuma indicação visual imediata. - Pergunta: Qual seria a solução ideal para melhorar a experiência de envio de mensagens?

a) Deixar o envio de mensagens sem qualquer feedback, pois as mensagens serão enviadas de qualquer forma.
b) Adicionar uma animação ou ícone que mostre que a mensagem foi enviada com sucesso.
c) Exigir que o usuário atualize a página para ver se a mensagem foi enviada.
d) Criar um botão adicional que os usuários possam apertar após enviar uma mensagem para verificar o status.

5. Testes de Back-end: Manipulação de Erros - Teoria: A manipulação de erros no back-end é crucial para garantir que o sistema não falhe em caso de exceções. É importante prever possíveis erros e fornecer mensagens claras, além de tratar as exceções de maneira controlada. - Situação-problema: Você está desenvolvendo uma API de login para um aplicativo, e durante os testes foi detectado que, quando o usuário tenta fazer login com um nome de usuário inexistente, a API retorna um erro genérico sem detalhes. - Pergunta: Qual seria a melhor prática para lidar com esse tipo de erro?

a) Retornar um erro genérico sem fornecer qualquer detalhe.
b) Retornar uma mensagem de erro clara que informe que o nome de usuário não foi encontrado.
c) Deixar o erro sem tratamento, pois o desenvolvedor pode ver o log do erro.
d) Retornar uma mensagem de erro interna do servidor ao usuário.

6. Testes de Back-end: Testes de Integração - Teoria: Os testes de integração garantem que diferentes módulos de um sistema funcionem juntos corretamente. Eles validam que as interações entre componentes, como APIs e banco de dados, estejam corretas e seguras. - Situação-problema: Você desenvolveu um serviço de pagamento que se comunica com um gateway externo. Durante os testes, foi identificado que, em certas condições, o sistema não consegue registrar pagamentos, mesmo quando a transação foi concluída com sucesso no gateway. - Pergunta: Que tipo de teste você realizaria para detectar e corrigir esse problema?

a) Testes unitários para verificar funções isoladas.
b) Testes de integração para verificar a comunicação entre o sistema e o gateway de pagamento.
c) Testes manuais, já que não há necessidade de automação para esse cenário.
d) Ignorar o problema e considerar uma falha no gateway.

7. Testes de Back-end: Performance - Teoria: Os testes de performance verificam se o sistema pode lidar com grandes quantidades de tráfego e processamento sem comprometer o desempenho. Estes testes garantem que o back-end tenha escalabilidade e responda dentro de limites aceitáveis. - Situação-problema: Durante um evento de vendas online, seu sistema de e-commerce apresentou lentidão e vários usuários relataram problemas ao concluir compras devido ao aumento no tráfego. - Pergunta: Qual seria a melhor abordagem para prevenir esse tipo de problema no futuro?

a) Executar testes de carga e stress para garantir que o sistema aguente grandes volumes de tráfego.
b) Reduzir o número de usuários permitidos no sistema durante eventos de vendas.
c) Remover funcionalidades para diminuir o uso do sistema.
d) Informar os usuários para acessarem o sistema em horários diferentes.

8. Testes de Back-end: Testes de Segurança - Teoria: Os testes de segurança visam identificar e corrigir vulnerabilidades no sistema que podem ser exploradas por atacantes. Estes testes incluem verificação de permissões, criptografia de dados e testes de penetração. - Situação-problema: Durante uma análise de segurança de um sistema de gestão de documentos, foi detectado que usuários não autorizados conseguem acessar documentos confidenciais através de uma falha no controle de permissões. - Pergunta: Que tipo de teste poderia ter prevenido essa vulnerabilidade?

a) Testes de penetração para simular ataques e verificar o controle de permissões.
b) Testes de usabilidade para verificar a experiência do usuário.
c) Testes unitários para verificar funções individuais.
d) Testes de carga para garantir a capacidade do sistema.

9. UI/UX: Design de Layout em F-forma - Teoria: Pesquisas sobre a leitura em telas sugerem que os usuários tendem a ler páginas da web em um padrão de "F", onde os olhos se concentram mais no topo e na esquerda da página. Um bom design deve organizar o conteúdo mais importante nessas áreas para melhorar a acessibilidade e a experiência do usuário. - Situação-problema: Um site de notícias está recebendo feedback negativo sobre a organização de seus artigos. Os usuários estão dizendo que não conseguem localizar facilmente as informações principais, e estão desistindo da leitura antes de chegarem ao final da página. - Pergunta: Como você ajustaria o layout do site para melhorar a experiência de leitura?

a) Organizar o conteúdo importante no topo e à esquerda, respeitando o padrão de leitura em F.
b) Centralizar todo o conteúdo no meio da página para focar a atenção do usuário.
c) Distribuir as informações uniformemente por toda a página, sem focar em áreas específicas.
d) Colocar o conteúdo principal no rodapé da página, forçando os usuários a rolarem até o final.

10. UI/UX: Espaçamento e Escaneabilidade - Teoria: O espaçamento adequado e a escaneabilidade são fundamentais em layouts modernos. Textos bem espaçados e organizados em blocos curtos com subtítulos permitem que os usuários localizem as informações mais importantes com rapidez, sem precisar ler o conteúdo integralmente. - Situação-problema: Você recebe um feedback de que os usuários de um blog corporativo estão achando os artigos "muito densos" e difíceis de ler. Eles mencionam que estão tendo dificuldade para encontrar rapidamente as informações que desejam. - Pergunta: O que você faria para melhorar a escaneabilidade dos artigos?

a) Adicionar subtítulos, listas e mais espaçamento entre os blocos de texto.
b) Diminuir o espaçamento e usar um único bloco de texto para compactar o conteúdo.
c) Reduzir o tamanho da fonte para caber mais texto na tela.
d) Colocar todas as informações importantes em um parágrafo no início.

11. UI/UX: Contraste e Legibilidade - Teoria: O contraste entre o texto e o fundo é crucial para a legibilidade. O uso de cores inadequadas pode dificultar a leitura, especialmente em dispositivos móveis, onde as condições de iluminação variam muito. Uma boa prática é usar um alto contraste entre texto e fundo para garantir a legibilidade em todas as situações. - Situação-problema: Usuários de um site de e-learning relataram que estão tendo dificuldade para ler os textos em dispositivos móveis, especialmente quando visualizam o site em ambientes externos com muita luz. - Pergunta: Como você justaria o design para melhorar a legibilidade?

a) Aumentar o contraste entre o texto e o fundo, usando cores mais escuras para o texto e um fundo claro.
b) Diminuir o contraste para tornar o design mais moderno e minimalista.
c) Usar cores mais vibrantes no fundo para destacar o texto.
d) Manter o contraste como está e sugerir que os usuários ajustem o brilho do dispositivo.

12. Banco de Dados: Normalização e Redundância de Dados - Teoria: A normalização no banco de dados visa reduzir a redundância e garantir que os dados sejam organizados de forma eficiente. Através de várias formas normais (1NF, 2NF, 3NF), é possível garantir que as informações sejam armazenadas sem duplicações desnecessárias e que cada tabela tenha um propósito claro. - Situação-problema: Você está gerenciando um banco de dados para uma loja online que, atualmente, armazena informações sobre clientes e pedidos em uma única tabela. Como resultado, os dados de clientes estão sendo repetidos em cada pedido, causando redundância. - Pergunta: Qual seria a melhor abordagem para eliminar essa redundância?

a) Criar uma tabela separada para os clientes e vincular os pedidos a essa tabela por meio de uma chave estrangeira.
b) Continuar armazenando todas as informações na mesma tabela para simplificar o design do banco de dados.
c) Remover todas as informações dos clientes e manter apenas os dados dos pedidos.
d) Criar uma cópia da tabela original para armazenar dados históricos e evitar redundância futura.

13. Banco de Dados: Operação de Leitura (Read) - Teoria: As operações de leitura (Read) permitem que os dados sejam recuperados do banco de dados. Consultas bem estruturadas são importantes para garantir que as informações corretas sejam exibidas rapidamente e de forma eficiente. O uso adequado de filtros, agregações e junções entre tabelas melhora o desempenho e a precisão das consultas. - Situação-problema: Em um banco de dados de uma empresa de transporte, você precisa consultar todos os motoristas que têm mais de 10 anos de experiência e cuja carteira de habilitação foi renovada nos últimos dois anos. Atualmente, a consulta retorna todos os motoristas, sem filtrar corretamente os anos de experiência e a renovação da habilitação. - Pergunta: Qual seria a melhor maneira de ajustar a consulta para obter os resultados corretos?

a) Usar uma cláusula WHERE para filtrar os motoristas com mais de 10 anos de experiência e que renovaram a habilitação nos últimos dois anos.
b) Adicionar um GROUP BY para agrupar todos os motoristas pela data de renovação.
c) Usar um ORDER BY para ordenar os motoristas por anos de experiência.
d) Utilizar uma cláusula JOIN para combinar os dados dos motoristas com uma tabela de pedidos.

14. Banco de Dados: Operação de Atualização (Update) - Teoria: A operação de atualização (Update) permite que registros existentes em uma tabela sejam modificados. É crucial garantir que as atualizações sejam feitas de forma correta e segura para evitar alterar registros incorretamente, especialmente em grandes bancos de dados. - Situação-problema: Um banco de dados de funcionários contém informações de salários, e você foi encarregado de aumentar em 10% o salário de todos os funcionários que estão há mais de 5 anos na empresa. No entanto, ao executar a consulta, os salários de todos os funcionários, independentemente do tempo de serviço, foram atualizados. - Pergunta: Qual ajuste na consulta evitaria que todos os salários fossem atualizados indevidamente?

a) Adicionar uma cláusula WHERE especificando que apenas funcionários com mais de 5 anos de serviço devem ser atualizados.
b) Usar uma cláusula ORDER BY para ordenar os funcionários pelo tempo de serviço antes de realizar a atualização.
c) Aplicar um SELECT * antes da atualização para garantir que todos os registros estejam corretos.
d) Utilizar um JOIN para combinar as tabelas de funcionários e salários antes de realizar a atualização.

15. Banco de Dados: Operação de Exclusão (Delete) - Teoria: A operação de exclusão (Delete) remove registros do banco de dados. É importante ter cautela ao usar DELETE, pois uma exclusão incorreta pode resultar em perda de dados críticos. Normalmente, é aconselhável fazer um backup dos dados antes de realizar grandes operações de exclusão. - Situação-problema: Você foi encarregado de remover todos os registros de clientes inativos que não fizeram nenhuma compra nos últimos 5 anos. No entanto, ao realizar a operação, todos os clientes, incluindo os ativos, foram excluídos da tabela. - Pergunta: O que você faria para garantir que apenas os clientes inativos sejam excluídos?

a) Adicionar uma cláusula WHERE para excluir apenas os clientes que não fizeram compras nos últimos 5 anos.
b) Utilizar uma cláusula ORDER BY para ordenar os clientes por data de última compra antes de excluí-los.
c) Aplicar um SELECT * antes de excluir para garantir que todos os registros estejam corretos.
d) Criar uma nova tabela para armazenar os clientes excluídos e evitar perder os dados.

16. API: Métodos HTTP em CRUD - Teoria: As APIs geralmente seguem as operações CRUD (Create, Read, Update, Delete) usando métodos HTTP. Os métodos comuns incluem POST (para criar), GET (para ler), PUT (para atualizar) e DELETE (para excluir). É fundamental utilizar o método correto para cada operação a fim de garantir que a API funcione de maneira padronizada. - Situação-problema: Você está desenvolvendo uma API para um serviço de gerenciamento de tarefas. O cliente relatou que a criação de

novas tarefas não está funcionando, pois os dados não são salvos no banco de dados. Após revisar o código, você percebe que o método HTTP utilizado para criar novas tarefas foi configurado incorretamente. - Pergunta: Qual método HTTP você deve usar para corrigir o problema e garantir que novas tarefas sejam criadas corretamente?

a) GET
b) POST
c) PUT
d) DELETE

17. API: Autenticação e Autorização - Teoria: Muitas APIs exigem autenticação (verificação de identidade do usuário) e autorização (permissões de acesso) para garantir a segurança dos dados e funcionalidades. Técnicas como OAuth, API Keys ou JWT (JSON Web Tokens) são usadas para controlar o acesso à API. - Situação-problema: Você desenvolveu uma API que permite a criação e edição de perfis de usuário. No entanto, alguns usuários estão reclamando que conseguem acessar dados e alterar perfis de outros usuários, o que viola as regras de acesso. - Pergunta: O que você deve implementar para garantir que os usuários possam apenas editar seus próprios perfis?

a) Implementar autenticação para verificar a identidade do usuário antes de permitir o acesso.
b) Implementar autorização, restringindo o acesso aos dados com base nas permissões do usuário.
c) Usar o método GET para garantir que os dados sejam apenas visualizados.
d) Implementar um sistema de logs para monitorar as alterações nos perfis.

18. API: Limitação de Taxa (Rate Limiting) - Teoria: A limitação de taxa (Rate Limiting) é uma prática usada para controlar o número de requisições que um cliente pode fazer a uma API em um determinado período. Isso ajuda a proteger a API de abusos e garantir que os recursos do servidor sejam usados de maneira eficiente. - Situação-problema: Uma API pública que você gerencia está sofrendo quedas de desempenho, e alguns clientes estão relatando que o serviço está indisponível. Após investigar, você descobre que um cliente está enviando milhares de requisições por minuto, sobrecarregando o servidor. - Pergunta: O que você pode fazer para evitar que isso ocorra no futuro?

a) Implementar limitação de taxa (Rate Limiting) para restringir o número de requisições por cliente em um determinado período de tempo.
b) Aumentar o número de servidores para suportar mais requisições.
c) Bloquear completamente o cliente que fez muitas requisições.
d) Diminuir a velocidade de resposta da API para todos os clientes.

19. API: Versionamento - Teoria: O versionamento de APIs permite que novas funcionalidades e alterações sejam introduzidas sem quebrar a compatibilidade com clientes que usam versões antigas. O uso de versões claras (como v1, v2) ajuda a manter o controle sobre diferentes iterações da API. - Situação-problema: Sua equipe está pronta para lançar uma nova versão de uma API que contém mudanças significativas na estrutura de dados e nos endpoints. No entanto, alguns clientes ainda dependem da versão anterior e não estão prontos para migrar imediatamente. - Pergunta: O que você deve fazer para garantir que ambos os grupos de clientes possam continuar usando a API sem interrupções?

a) Substituir a API antiga pela nova versão imediatamente.
b) Manter a versão antiga da API e introduzir a nova versão como uma opção separada (por exemplo, /api/v2/).
c) Remover a API antiga e informar os clientes para se adaptarem rapidamente.
d) Fazer mudanças na API antiga para que funcione com as novas funcionalidades.

20. Git: Branches e Merges - Teoria: O uso de branches no Git permite que desenvolvedores trabalhem em diferentes funcionalidades ou correções de bugs simultaneamente, sem interferir no código principal. Após o desenvolvimento, as alterações são mescladas (merged) de volta à branch principal (geralmente main ou master). A gestão correta de branches é fundamental para um fluxo de trabalho eficiente. - Situação-problema: Você está trabalhando em uma nova funcionalidade em uma branch separada chamada feature-login. Durante o desenvolvimento, a equipe fez algumas alterações importantes na branch main, e agora você precisa trazer essas mudanças para sua branch antes de continuar trabalhando na nova funcionalidade. - Pergunta: Qual é o procedimento correto para trazer as mudanças da branch main para a feature-login?

a) Usar o comando git pull main enquanto está na branch feature-login.
b) Usar o comando git merge main enquanto está na branch feature-login.
c) Usar o comando git rebase main enquanto está na branch main.
d) Usar o comando git checkout main e continuar trabalhando na feature-login.

21. Git: Controle de Versão com Tags - Teoria: No Git, as tags são usadas para marcar pontos específicos no histórico de commits, como lançamentos de software ou versões estáveis. Tags ajudam a rastrear versões específicas e são frequentemente utilizadas para gerenciar versões de software no desenvolvimento de projetos. - Situação-problema: Sua equipe acabou de lançar a versão 2.0 de um aplicativo, e você quer marcar esse commit com uma tag para facilitar a referência futura. Você precisa garantir que a tag esteja disponível para todos os membros da equipe. - Pergunta: Qual comando você deve usar para criar e compartilhar essa tag com a equipe?

a) git tag 2.0

b) git tag -a v2.0 -m "Versão 2.0"
c) git push origin --tags
d) git commit -m "Versão 2.0"

22. Versionamento Full-Stack: Controle de Versão de API - Teoria: No desenvolvimento full-stack, é comum versionar APIs para garantir que mudanças nos endpoints e funcionalidades não quebrem a compatibilidade com clientes existentes. O versionamento de API pode ser feito por meio de versões explícitas, como /api/v1/ ou /api/v2/, permitindo que diferentes clientes usem versões específicas. - Situação-problema: Sua equipe lançou recentemente uma nova versão da API com melhorias significativas, mas alguns clientes ainda estão utilizando a versão antiga. Agora, você precisa manter ambas as versões da API disponíveis. - Pergunta: Como você deve lidar com o versionamento da API para garantir compatibilidade com clientes antigos e novos?

a) Atualizar todos os clientes imediatamente para a nova versão da API.
b) Manter a versão antiga da API e introduzir a nova versão como uma rota separada, como /api/v2/.
c) Substituir a versão antiga pela nova imediatamente.
d) Fazer todas as mudanças na API antiga, sem introduzir uma nova versão.

23. Versionamento Full-Stack: Banco de Dados - Teoria: No desenvolvimento full-stack, além de versionar o código, também é importante versionar as alterações no banco de dados. Isso inclui mudanças em tabelas, campos e relações. Ferramentas de migração de banco de dados permitem aplicar alterações de forma controlada em diferentes ambientes (desenvolvimento, produção) sem afetar dados já existentes. Situação-problema: Você adicionou um novo campo data_nascimento à tabela de usuários no banco de dados, e agora precisa garantir que essa mudança seja aplicada em todos os ambientes (desenvolvimento, testes, produção), sem perder dados existentes. -Pergunta: Qual é a melhor prática para aplicar essa mudança no banco de dados?

a) Alterar manualmente o banco de dados em cada ambiente.
b) Usar uma ferramenta de migração de banco de dados para criar um script que aplique a alteração em todos os ambientes de forma controlada.
c) Fazer a alteração diretamente no banco de dados de produção e informar a equipe.
d) Excluir a tabela de usuários e recriá-la com o novo campo.

24. Front-end: Responsividade - Teoria: A responsividade no design de front-end garante que uma interface de usuário funcione corretamente em dispositivos com diferentes tamanhos de tela, como desktops, tablets e smartphones. Técnicas como media queries e layout flexível são usadas para adaptar o design de forma eficiente. - Situação-problema: Você desenvolveu uma página da web que funciona bem em telas grandes,

mas os usuários relataram problemas ao acessá-la em dispositivos móveis, pois o layout parece desorganizado e alguns elementos são cortados. - Pergunta: O que você deve fazer para corrigir o layout da página em dispositivos móveis?

a) Usar media queries para aplicar estilos específicos a diferentes tamanhos de tela.
b) Definir uma largura fixa para a página para garantir que ela tenha o mesmo tamanho em todos os dispositivos.
c) Aumentar o tamanho dos elementos para que fiquem visíveis em telas menores.
d) Reduzir a quantidade de conteúdo exibido em dispositivos móveis.

25. Front-end: Performance e Otimização - Teoria: A otimização de performance no front-end é essencial para garantir que a página carregue rapidamente. Técnicas como compactação de imagens, minificação de arquivos CSS/JS e carregamento assíncrono de scripts podem melhorar o tempo de carregamento da página e a experiência do usuário. - Situação-problema: Seu site está apresentando tempos de carregamento longos, especialmente para usuários com conexões de internet mais lentas. Após uma análise, você percebeu que as imagens grandes e arquivos de JavaScript estão contribuindo para o problema. - Pergunta: Quais práticas você deve adotar para otimizar o carregamento do site?

a) Compactar as imagens e usar o carregamento assíncrono para os arquivos JavaScript.
b) Aumentar o tamanho das imagens para que fiquem mais nítidas em conexões lentas.
c) Colocar todos os arquivos CSS e JavaScript inline no HTML.
d) Reduzir a quantidade de imagens usadas na página.

26. Front-end: Acessibilidade - Teoria: A acessibilidade no front-end garante que pessoas com diferentes tipos de deficiência possam navegar e interagir com o site. Isso inclui o uso de elementos semânticos, textos alternativos para imagens, navegação por teclado e outras práticas que tornam o site inclusivo para todos os usuários. - Situação-problema: Seu site passou por uma auditoria de acessibilidade, e foi constatado que usuários com deficiência visual estão tendo dificuldade em navegar pelos menus, especialmente porque os elementos não são bem identificados pelos leitores de tela. - Pergunta: O que você pode fazer para melhorar a acessibilidade dos menus?

a) Usar elementos HTML semânticos, como <nav>, e adicionar atributos aria-label para melhorar a navegação por leitores de tela.
b) Aumentar o tamanho do texto para facilitar a leitura dos menus.
c) Remover o menu para usuários que usam leitores de tela.
d) Mudar as cores do menu para melhorar o contraste visual.

27. Front-end: Single Page Application (SPA) - Teoria: Single Page Applications (SPAs) são sites ou aplicativos da web que carregam uma única página HTML e atualizam o conteúdo dinamicamente à medida que o usuário interage, sem recarregar a página inteira. SPAs oferecem uma experiência mais rápida e interativa, mas podem ter desafios como SEO (Search Engine Optimization) e gerenciamento de estado. - Situação-problema: Você está desenvolvendo um aplicativo web usando o modelo SPA. No entanto, alguns usuários estão relatando problemas com o funcionamento dos botões de voltar e avançar do navegador, pois o conteúdo da página não está sendo atualizado corretamente. - Pergunta: Qual seria a melhor solução para esse problema?

a) Usar a API de histórico do navegador (history.pushState()) para gerenciar as mudanças de estado da aplicação.
b) Adicionar um botão "Voltar" na página para simular a funcionalidade do navegador.
c) Carregar todas as páginas em novas abas para evitar problemas de navegação.
d) Desativar os botões de voltar e avançar do navegador para evitar confusão.

28. API: Paginação de Resultados - Teoria: Quando uma API retorna grandes quantidades de dados, é importante implementar a paginação para evitar sobrecarregar o cliente e o servidor. A paginação permite que os dados sejam divididos em blocos menores, tornando o consumo mais eficiente. - Situação-problema: Você está consumindo uma API que retorna uma lista de produtos, mas percebe que a quantidade de dados está sobrecarregando seu aplicativo, tornando o tempo de resposta muito lento. Ao verificar a documentação da API, você descobre que ela oferece suporte para paginação. - Pergunta: Como você deve ajustar sua requisição à API para utilizar a paginação e melhorar a performance?

a) Usar parâmetros de query como page e limit para solicitar um número menor de resultados por página.
b) Fazer múltiplas requisições à API até obter todos os resultados de uma vez.
c) Solicitar todos os resultados da API de uma vez e filtrar os dados localmente.
d) Alterar o método HTTP de GET para POST para reduzir a quantidade de dados retornados.

29. API: Tratamento de Erros - Teoria: O tratamento adequado de erros é essencial ao consumir APIs. As APIs devem retornar códigos de status HTTP apropriados (como 200 para sucesso, 404 para não encontrado, 500 para erro no servidor) e mensagens de erro claras, permitindo que o cliente trate essas respostas adequadamente. - Situação-problema: Você está consumindo uma API para obter informações de usuários, mas em alguns casos a API retorna um código de status 404 (não encontrado) quando o ID do usuário não existe. No entanto, seu aplicativo não está lidando bem com esses erros e está exibindo mensagens confusas aos usuários. - Pergunta: Como você deve ajustar seu código para tratar o erro 404 corretamente?

a) Exibir uma mensagem amigável informando que o usuário não foi encontrado.
b) Continuar tentando obter o recurso até que ele seja encontrado.
c) Encerrar o aplicativo imediatamente quando ocorrer um erro 404.
d) Ignorar o erro e retornar uma resposta genérica para o usuário.

30. API: Autenticação usando Tokens - Teoria: Muitas APIs requerem autenticação para acessar recursos protegidos. Uma técnica comum é o uso de tokens de autenticação, como JWT (JSON Web Tokens), que são passados no cabeçalho da requisição HTTP. O token verifica a identidade do cliente e permite acesso a recursos protegidos. - Situação-problema: Você está consumindo uma API que requer autenticação via token. No entanto, ao fazer requisições, está recebendo um código de erro 401 (não autorizado), o que indica que o token não está sendo passado corretamente. - Pergunta: Como você deve enviar o token para garantir que a API autentique suas requisições corretamente?

a) Incluir o token no corpo da requisição HTTP.
b) Passar o token no cabeçalho da requisição HTTP, geralmente usando o campo Authorization.
c) Enviar o token como um parâmetro de query na URL.
d) Enviar o token via método POST para todas as requisições.

31. API: Rate Limiting e Limitação de Uso - Teoria: Muitas APIs implementam rate limiting para controlar o número de requisições que um cliente pode fazer em um determinado período. Isso protege o servidor contra abusos e sobrecargas. O cliente deve lidar com essas limitações e ajustar o número de requisições conforme necessário. - Situação-problema: Você está desenvolvendo um aplicativo que consome uma API pública, mas começou a receber códigos de erro 429 (Too Many Requests), indicando que você excedeu o limite de requisições permitidas em um determinado período de tempo. - Pergunta: Como você deve lidar com esse erro e evitar exceder o limite de requisições no futuro?

a) Implementar um sistema de retries, onde a requisição é reenviada até ser bem-sucedida.
b) Monitorar o número de requisições e ajustar a frequência de acordo com o limite imposto pela API.
c) Aumentar o número de requisições por segundo para tentar obter mais dados em menos tempo.
d) Ignorar o erro 429 e continuar fazendo requisições normalmente.

32. Prototipagem: Baixa Fidelidade - Teoria: Prototipagem de baixa fidelidade é uma técnica que utiliza representações simples e rápidas de uma interface ou produto, como esboços em papel ou wireframes básicos. Essa abordagem permite testar conceitos de forma econômica e obter feedback inicial sem investir muito tempo em detalhes visuais. -

Situação-problema: Você está desenvolvendo um aplicativo móvel e deseja apresentar uma ideia inicial de interface para o cliente. No entanto, você tem pouco tempo e não quer se concentrar em detalhes visuais, como cores e tipografia. O foco deve estar nas funcionalidades e no fluxo de navegação. - Pergunta: Qual seria a melhor abordagem para criar o protótipo nesse estágio?

a) Criar um protótipo de baixa fidelidade, como esboços em papel ou wireframes simples, focando nas funcionalidades principais.
b) Desenvolver um protótipo de alta fidelidade com cores, tipografia e interações completas.
c) Criar diretamente a interface final no código.
d) Apresentar ao cliente apenas a ideia verbalmente, sem qualquer protótipo visual.

33. Prototipagem: Alta Fidelidade - Teoria: Prototipagem de alta fidelidade envolve criar representações detalhadas e quase finais de um produto, geralmente com foco em design visual, tipografia, cores, interações e animações. Esses protótipos permitem testes mais próximos da experiência final do usuário. - Situação-problema: Seu cliente quer visualizar como o aplicativo final se parecerá e funcionará. Ele já aprovou a estrutura e o fluxo de navegação, mas agora deseja ver um protótipo com cores, imagens, tipografia e interações quase finalizadas. - Pergunta: Qual seria a abordagem mais adequada neste estágio?

a) Desenvolver um protótipo de alta fidelidade com cores, fontes, imagens e interações detalhadas.
b) Manter um protótipo de baixa fidelidade, já que a estrutura foi aprovada.
c) Entregar diretamente o código do produto final.
d) Criar wireframes simples com anotações detalhadas.

34. Prototipagem: Escolha de Fidelidade para Testes de Usuário - Teoria: A escolha entre um protótipo de baixa ou alta fidelidade depende do estágio do projeto e dos objetivos do teste de usuário. Em estágios iniciais, a baixa fidelidade é ideal para validar ideias e funcionalidades rapidamente. Em estágios avançados, a alta fidelidade é usada para testar a experiência do usuário final, incluindo design visual e interações. - Situação-problema: Sua equipe está organizando testes de usabilidade com usuários. O foco principal é verificar se o fluxo de navegação é intuitivo e se as principais funcionalidades são compreendidas. O design visual não é uma prioridade neste momento. - Pergunta: Que tipo de protótipo seria mais apropriado para esses testes?

a) Prototipagem de baixa fidelidade, como wireframes clicáveis, focando no fluxo de navegação.
b) Prototipagem de alta fidelidade com todos os detalhes visuais incluídos.
c) Um documento escrito detalhando o fluxo de navegação.

d) Apresentação verbal do fluxo de navegação aos usuários.

35. Prototipagem: Quando Usar Baixa Fidelidade - Teoria: Prototipagem de baixa fidelidade é mais útil em estágios iniciais de um projeto, quando a equipe está explorando diferentes ideias e soluções, ou quando o objetivo é testar a funcionalidade básica sem investir em detalhes visuais. - Situação-problema: Você e sua equipe estão em uma fase inicial de um projeto e precisam apresentar rapidamente diferentes conceitos de interface para os stakeholders. No entanto, o design visual não foi definido e o foco está em testar a ideia e o fluxo de funcionalidades. - Pergunta: Qual abordagem seria a mais eficaz para apresentar esses conceitos?

a) Criar protótipos de baixa fidelidade, como wireframes ou esboços em papel, para apresentar as ideias de forma rápida e eficaz.
b) Criar um protótipo de alta fidelidade com todas as interações e design visual.
c) Desenvolver a interface completa em código para que os stakeholders possam ver a versão final.
d) Descrever verbalmente as funcionalidades e navegações da interface.

36. Prototipagem: Equilíbrio entre Baixa e Alta Fidelidade - Teoria: Em alguns projetos, é importante encontrar um equilíbrio entre protótipos de baixa e alta fidelidade. Isso pode significar criar protótipos funcionais (como wireframes clicáveis) que ainda não têm todos os detalhes visuais, mas já permitem testar interações e fluxos de forma realista. - Situação-problema: Seu projeto está em um estágio intermediário. A estrutura principal já foi aprovada e vocês estão começando a desenvolver o design visual. No entanto, ainda há algumas interações que precisam ser testadas com os usuários antes de finalizar o design. - Pergunta: O que seria mais apropriado para esta fase?

a) Criar um protótipo de média fidelidade, onde as interações e o fluxo estão funcionais, mas o design visual ainda não está totalmente finalizado.
b) Focar em um protótipo de baixa fidelidade e evitar qualquer design visual.
c) Criar um protótipo de alta fidelidade completo, incluindo todos os detalhes de design e interações.
d) Testar as interações diretamente no código do projeto.

37. UI/UX: Layout em Z-forma - Teoria: O layout em Z é comum em páginas com menos texto, como páginas de destino (landing pages). Ele segue o movimento natural dos olhos ao longo da página, começando da esquerda para a direita no topo, depois cruzando para o canto inferior esquerdo e terminando novamente no canto inferior direito. - Situação-problema: Você está criando uma página de destino para uma campanha publicitária e deseja que os usuários visualizem os pontos principais da campanha com rapidez e eficiência. No entanto, os primeiros testes indicam que os

usuários estão perdendo os elementos principais e abandonando a página rapidamente.
- Pergunta: Que tipo de layout seria mais adequado para essa página de destino?

a) Utilizar um layout em Z, posicionando as informações principais nos pontos superiores e inferiores da página.
b) Organizar todo o conteúdo em colunas verticais, sem hierarquia.
c) Colocar todo o texto em um bloco no centro da página.
d) Dividir o conteúdo igualmente entre várias caixas de texto pequenas.