Relatório Técnico
Arquitetura de Frontend e Backend

Disciplina: Engenharia de Software

Professor: César Olavo

Equipe: Hiel Saraiva, Roberta Alanis e Charles Lima

Perguntas:

1. **Onde ficou a maior parte da lógica em cada arquitetura?**

- **MVC:**

A lógica de negócio do nosso projeto, obedecendo a arquitetura MVC, está distribuída principalmente no Controller (todoController.js). É lá que ficam as validações, como verificar se o título está vazio antes de criar um todo, e a orquestração das operações, decidindo quando carregar dados, quando criar um novo todo ou quando removê-lo.

O Model (todoModel.js) atua como uma camada de acesso a dados, responsável apenas por fazer a comunicação com o banco de dados (Supabase) e executar as operações CRUD.

A View (todoView.jsx) é responsável pela apresentação e captura das interações do usuário.

- **MVP:**

Na arquitetura MVP, a maior parte da lógica do sistema está concentrada no Presenter (todoPresenter.js). É o Presenter que assume o papel de coordenar o fluxo da aplicação, contendo as regras de negócio e de apresentação, como validações (por exemplo, impedir a criação de tarefas com título vazio), decisões de quando buscar os dados no backend e quando atualizar a interface.

O Model (todoModel.js) continua com uma responsabilidade semelhante à do MVC, atuando exclusivamente como camada de acesso aos dados, realizando a comunicação com o Supabase e executando as operações de leitura, criação e remoção de tarefas.

A principal diferença está na View (TodoView.jsx), que no MVP é uma view passiva: ela não contém lógica de negócio nem toma decisões. Sua função é apenas exibir os dados recebidos do Presenter e notificar eventos de interação do usuário (cliques, digitação), delegando completamente ao Presenter a responsabilidade de decidir o que fazer a partir dessas interações.

- **MVVM:**

Dentro da arquitetura MVVM, a lógica se concentra no ViewModel (todoViewModel.js), pois ela expõe fluxos de dados para manter o estado da View e manipula o Model. Apesar de ser similar ao MVP, o ViewModel ao expor fluxos de dados não precisa guardar referências da View. Isso faz com que várias Views recebam dados de uma ViewModel.

O Model (todoModel.js) mantêm o mesmo comportamento como nas outras arquiteturas, abstrair o banco de dados.

A View (todoView.jsx) liga as ações expostas pelo ViewModel e a informa das ações tomadas pelo cliente.

2. **Qual arquitetura foi mais simples de integrar com o backend reativo?**

   A arquitetura MVVM é considerada mais simples de integrar com um backend reativo em comparação com MVC e MVP por causa da forma como ela lida com a atualização automática de dados na interface, sem exigir que o desenvolvedor escreva código explícito para sincronizar mudanças no modelo com a visualização.

   No padrão MVVM, o ViewModel atua como uma camada intermediária que expõe dados e comandos de forma que a View possa se ligar a eles diretamente por meio de "data binding", ou seja, a View atualiza automaticamente quando os dados do ViewModel mudam e vice-versa. Esse mecanismo reduz a quantidade de código manual necessário para refletir mudanças de estado no frontend quando o backend envia atualizações, porque a maioria das frameworks modernas (React, Angular, Vue, etc.) já provêm recursos reativos para isso.

3. **O que mudou no frontend ao trocar REST por reativo?**

   A principal mudança foi a introdução de reatividade em tempo real através do Supabase Realtime. No frontend com REST tradicional nós apenas fazíamos requisições e atualizávamos a tela quando o usuário disparava uma ação. Agora, com a abordagem reativa, a interface se atualiza automaticamente quando os dados mudam no backend, mesmo que a mudança tenha vindo de outro cliente.

   No nosso código, isso aparece claramente no App.jsx, nós nos inscrevemos no canal Realtime do Supabase (supabase.channel('public:todos')) e sempre que ocorre uma mudança na tabela todos (inserção, atualização ou deleção), o callback recarrega os dados automaticamente (controller.loadTodos(setTodos)).

   Uma vantagem desta implementação é que nós controlamos a alternância entre REST e reativo diretamente no Supabase, ativando ou desativando a reatividade da tabela todos. Isso significa que não precisamos alterar o código para mudar entre os dois modos, pois o comportamento muda conforme a configuração do Supabase, mantendo a mesma interface e lógica de negócio.

4. **Qual arquitetura você escolheria para um sistema maior? Por quê?**

   Para um sistema maior, a arquitetura escolhida seria o MVVM. Ela ajuda bastante a manter o código mais organizado, já que separa bem a interface da lógica. A View fica focada só em mostrar os dados, enquanto o ViewModel concentra o estado da aplicação e as regras, evitando que a lógica fique espalhada pelos componentes.

   Outro ponto importante é que o MVVM lida melhor com aplicações que têm muitos estados e atualizações em tempo real, como no uso de um backend reativo. Quando algo muda nos dados, o ViewModel atualiza o estado e a interface acompanha automaticamente, sem precisar de muito controle manual.

   Além disso, essa separação entre View e lógica facilita testes, manutenção e futuras mudanças no sistema. Em projetos maiores, onde o código cresce com o tempo, isso ajuda a manter tudo mais organizado e reduz a chance de erros.