

Questionário Frontend (Professor)

Questionário por Arquivo (Frontend)

Formato: 3 perguntas e respostas por arquivo, estilo professor avaliando React, estado, arquitetura e integração. Arquivos *_old excluídos.

main.tsx

Q1: Qual a responsabilidade principal de `main.tsx` em uma aplicação React com Vite?

A: Inicializar o root React, montar o componente raiz (`<App />`) dentro do elemento DOM e aplicar providers globais (se existirem).

Q2: Por que centralizar o ponto de entrada em um único arquivo ajuda na manutenção?

A: Facilita adicionar wrappers (contextos, tema, roteamento) sem alterar múltiplos arquivos e padroniza o bootstrap da aplicação.

Q3: Que tipo de lógica NÃO deve ser colocada em `main.tsx`?

A: Lógica de negócios ou estados específicos de features; deve limitar-se a composição inicial e configurações globais.

App.tsx

Q1: Qual o papel de `App.tsx` no fluxo geral?

A: Funciona como container de alto nível orquestrando layout principal e compondo seções (ex.: calendário, janela, sidebar).

Q2: Como `App.tsx` poderia integrar roteamento futuramente sem quebrar a estrutura atual?

A: Envolvendo o conteúdo em `` e mapeando rotas para páginas mantendo componentes existentes como filhos.

Q3: Por que separar `App.tsx` de componentes de feature reduz acoplamento?

A: Evita dependências diretas entre inicialização e lógica de cada feature, permitindo evolução independente.

components/sidebar/usuario.tsx

Q1: Que tipo de informação este componente deve exibir e por que é isolado?

A: Dados do usuário (nome, email, avatar), isolado para facilitar atualização e cache/local state sem re-renderizar toda a sidebar.

Q2: Como poderia consumir contexto global de autenticação corretamente?

A: Usando `useContext(AuthContext)` para acessar usuário e status sem prop drilling.

Q3: Qual cuidado de performance aplicar ao exibir dados de usuário?

A: Memorizar subcomponentes ou evitar recomputações caras (ex.: formatação repetida) com `React.memo`.

components/sidebar/sidebar.tsx

Q1: Qual a função principal da `sidebar`?

A: Navegação e agrupamento visual de módulos (calendário, refeições, estatísticas).

Q2: Como tornar a sidebar responsiva sem reescrever lógica?

A: Aplicando classes condicionais (ex.: `is-collapsed`) e media queries nos estilos existentes.

Q3: Que padrão aplicar para destacar item ativo?

A: Gerenciar estado ativo via prop ou contexto e aplicar classe `active` para estilos diferenciados.

components/calendario/calendario.tsx

Q1: Como este componente integra controller, utils e UI?

A: Gera dias usando `CalendarioController`, consulta dados via `nutryoFetch` e injeta props em

`Dia`, controlando seleção com estado local.

Q2: Por que usar `useEffect` para selecionar o dia inicial?

A: Garante execução após o estado inicial dos dias existir, evitando seleção antes do render map.

Q3: Qual vantagem de delegar lógica de cálculo de dias ao `CalendarioController`?

A: Mantém o componente enxuto e permite reutilizar lógica em outras partes (ex.: estatísticas) sem duplicação.

components/calendario/dia.tsx

Q1: Qual responsabilidade única de `dia.tsx`?

A: Renderizar uma célula de dia aplicando classes visuais (selecionado, anotado, tipo de mês).

Q2: Por que é útil receber `temAnotacao` como prop?

A: Mantém a lógica de detecção fora do componente simples, favorecendo reutilização e teste da condição em nível superior.

Q3: Como evitar re-renderizações desnecessárias de muitos dias?

A: Usar `React.memo` para comparar props primitivas e não recalcular quando inalteradas.

components/janela/janela.tsx

Q1: Qual papel da janela no fluxo de refeições?

A: Exibir abas/conteúdo dinâmico ligado ao dia selecionado (refeições, detalhes, possivelmente metas).

Q2: Que padrão aplicar para trocar abas sem aumentar complexidade?

A: Controlar aba ativa com estado único e renderização condicional ou mapa de componentes.

Q3: Como conectar a janela ao calendário sem acoplamento forte?

A: Receber data selecionada como prop ou via contexto e reagir a mudanças sem importar diretamente controller.

components/auth/registro.tsx

Q1: Qual fluxo esperado de registro em React?

A: Inputs controlados -> validação -> chamada async (controller ou service) -> feedback de sucesso/erro.

Q2: Onde validar campos para melhor UX?

A: Em tempo real (onChange) para formato básico e no submit para regras completas.

Q3: Como evitar duplicar lógica de validação entre registro e login?

A: Extrair funções para util ou hook compartilhado (`useAuthForm`).

components/auth/login.tsx

Q1: Qual diferença principal entre login e registro em termos de estado?

A: Login usa menos campos (email/senha) e foca em autenticação, registro envolve criação de entidade usuário.

Q2: Como tratar erros de credenciais?

A: Exibir mensagem clara e limpar apenas o campo sensível (senha) mantendo email para retry.

Q3: O que usar para persistir sessão?

A: Tokens armazenados em `httpOnly cookies` ou `localStorage` (menos seguro) combinados com renovação de sessão.

components/auth/auth.tsx

Q1: Qual função de um wrapper auth?

A: Alternar entre telas de login/registro e possivelmente proteger rotas após autenticação.

Q2: Como organizar estados de modo escalável?

A: Usar um estado discriminado (ex.: `mode: 'login' | 'register'`) e componente condicional.

Q3: Como integrar feedback de backend (ex.: erro 401)?

A: Canalizar resposta via controller e mapear para mensagens amigáveis no UI.

components/janela/aba.tsx

Q1: Qual a responsabilidade típica de uma aba?

A: Representar cabeçalho/seleção de conteúdo, disparando evento para trocar a view ativa.

Q2: Melhor forma de sinalizar aba ativa?

A: Classe CSS derivada de prop `active` ou comparação por id.

Q3: Como torná-la acessível?

A: Usar roles apropriados (`tab`) e navegação por teclado (`onKeyDown`).

components/janela/refeicoes/refeicoes.tsx

Q1: Diferença entre lista de refeições e componente refeição individual?

A: Lista coordena coleção (map, add), o individual gerencia estado interno e eventos de seus alimentos.

Q2: Onde manter estado de coleção vs. item?

A: Coleção no componente de lista; item recebe props e gerencia subtarefas (ex.: edição campo).

Q3: Como preparar para virtualização se lista crescer?

A: Abstrair render em função e introduzir biblioteca como `react-window` sem refatorar lógica de negócios.

components/janela/refeicoes/refeicao.tsx

Q1: Por que normalizar alimentos ao carregar?

A: Garante estrutura previsível (ids sequenciais, tipos consistentes) simplificando update/remoção e minimizando bugs.

Q2: Como melhorar performance ao recalcular macros de muitos alimentos?

A: Memorizar cálculos por dependência (peso, tipo) ou mover para util pura reutilizável.

Q3: Qual cuidado ao manipular ids locais e ids do backend?

A: Separar `id` (ordem local) de `_id` (identificador persistente) evitando colisões ou updates errados.

components/janela/alimentos/listaAlimentos.tsx

Q1: Qual propósito deste componente?

A: Exibir resultados de busca e permitir seleção rápida de alimento para preencher dados nutricionais.

Q2: Como prevenir re-renderizações ao digitar?

A: Debounce na busca e renderização condicionada apenas quando resultados mudam.

Q3: Como lidar com muitos resultados?

A: Paginação ou limite de resultados + scroll virtual.

components/janela/alimentos/alimento.tsx

Q1: Qual abordagem de estado usada para edição de peso e macros?

A: Estado interno controlado (`pesoLocal`) recalculando macros derivados; updates propagados ao parent via callback.

Q2: Por que recalcular macros em tempo real no componente?

A: Fornece feedback imediato ao usuário, reduz round trips e melhora experiência.

Q3: Como tornar este cálculo mais testável?

A: Extrair função pura `calcularMacros(base, peso)` em arquivo util e cobrir com testes unitários.

utils/nutryoFetch.ts

Q1: Qual papel deste util no fluxo?

A: Abstrair chamadas ao backend para obter refeições/dias, centralizando endpoints e parsing.

Q2: Vantagem de centralizar fetch em um módulo?

A: Reuso, cache potencial e padronização de tratamento de erros.

Q3: Como evoluir para lidar com estados de carregamento?

A: Retornar Promises embrulhadas em hooks (`useNutryoData`) com loading/error/data.

utils/diaObjeto.ts

Q1: Por que manter representação local do dia?

A: Permite edição offline/otimista antes de persistir, reduzindo latência percebida.

Q2: Como decidir entre postar ou editar?

A: Verificando existência do dia em `diasSalvos` e disparando PUT ou POST conforme referência.

Q3: Risco de estado global estático?

A: Conflitos em múltiplos componentes; mitigação com contexto ou classe instanciável/hook isolado.

utils/connection.ts

Q1: Função esperada de um módulo de conexão?

A: Configurar base URL, headers (auth), e reutilizar instância para requisições uniformes.

Q2: Como adicionar retry exponencial?

A: Encapsular fetch em função que tenta novamente com delays crescentes até limite.

Q3: Como proteger credenciais?

A: Evitar expor tokens em logs e usar variáveis de ambiente + cookies seguros.

utils/buscarAlimento.ts

Q1: Qual responsabilidade desta função/util?

A: Consultar backend ou fonte local por alimentos filtrados por texto para autocomplete.

Q2: Como otimizar buscas sucessivas?

A: Implementar cache LRU por termo ou prefixo.

Q3: Melhor estratégia para evitar chamadas redundantes?

A: Debounce e cancelar requisição anterior em digitação rápida.

controllers/auth/authController.ts

Q1: Por que separar controller de componentes?

A: Isola lógica de autenticação, facilita testes e troca de implementação.

Q2: Como tratar fluxo de expiração de sessão?

A: Controller gerencia refresh token e invalida sessão em falhas consecutivas.

Q3: Benefício de retornar objetos padronizados (status, data, error)?

A: Simplifica consumo e exibição consistente de feedback.

controllers/auth/loginController.ts

Q1: Diferença em relação ao `registerController`?

A: Foca apenas em validação de credenciais e emissão de sessão, sem criação de perfil.

Q2: Como lidar com lockout por tentativas?

A: Implementar contagem no backend e informar estado ao frontend para UX adequada.

Q3: Por que normalizar resposta do backend?

A: Evita checks de campos variáveis nos componentes, aumenta robustez.

controllers/auth/registerController.ts

Q1: Qual verificação extra comparada ao login?

A: Checagem de força de senha, unicidade de email e possivelmente confirmação de senha.

Q2: Onde colocar sanitização de entrada?

A: Antes de enviar ao backend (trim, remover caracteres inválidos) + validação servidor.

Q3: Como preparar para internacionalização?

A: Evitar strings fixas de erro; retornar códigos e mapear para traduções.

controllers/refeicoes/refeicoesController.ts

Q1: Principal função do controller de refeições?

A: Coordenar criação, atualização e recuperação de refeições com suas listas de alimentos.

Q2: Como suportar edição otimista?

A: Aplicar atualização local imediata e enviar PATCH; reverter se falhar.

Q3: Qual benefício de expor métodos puros?

A: Facilita testes unitários sem precisar de ambiente React.

controllers/calendario/calendarioController.ts

Q1: Qual lógica ele abstrai do componente?

A: Geração de dias, controle de mês/ano e data selecionada; centraliza cálculos relativos ao calendário.

Q2: Como facilitar testes desta lógica?

A: Garantir funções puras que recebem mês/ano e retornam estrutura previsível.

Q3: Vantagem de não misturar com estado React?

A: Reusabilidade em outros contextos (Node scripts, SSR) e menor dependência de ciclo de render.

Fim do questionário.