UNIVERSIDADE ESTADUAL DA PARAÍBAS
CENTRO DE CIÊNCIAS EXATAS E SOCIAIS APLICADAS
CAMPUS VII - PATOS
Bacharelado em Ciência da Computação
Disciplina: Gerenciamento de Projetos
Professor: Rodrigo Alves Costa
Turno: Noturno
Alunos: Esly Caetano da Silva Ferreira,
Luanny Kelly de Almeida Leitão

Arquitetura do projeto

É uma combinação de padrões comuns em aplicações web modernas com React e Supabase. Aqui está uma visão geral:

1. **Arquitetura Geral: Client-Server**
   - **Frontend (cliente)**: Aplicação React + TypeScript que cuida da interface do usuário e interação com a API.
   - **Backend (servidor)**: API RESTful usando Express + TypeScript que lida com lógica de autenticação, regras de negócio e acesso ao banco (PostgreSQL via Prisma).
   - **Banco de Dados**: Supabase e PostgreSQL, com autenticação gerenciada por Supabase Auth.

2. **Frontend: Arquitetura baseada em Componentes**
   - **Framework**: React com TypeScript.
   - **Estrutura**:
     - **Pasta `components/`**: Estamos organizando de forma modular e reutilizável.
     - **`pages/`**: Uso de rotas com `react-router-dom`.
     - **`hooks/`** e **`contexts/`:** Uma estrutura preparada para compartilhamento de estado e lógica reutilizável.
     - **`services/`**: Boa prática para centralizar chamadas à API/backend.
     - **Estado**: Com `useState`, `useEffect` e potencial para `Context API` ou `Redux` caso o projeto cresça.
   - **Padrão**: Não seguimos um padrão rígido como Redux ou Context API para estado global (ainda), mas usamos uma abordagem simples baseada em componentes com estado local, adequada para o escopo atual.

   - **Composição de Componentes**

   - **Service Layer** → separa lógica de chamadas externas do resto do app

3. **Backend: Arquitetura RESTful com Camadas**
    - **Framework**: Express.js com TypeScript.
    - **Estrutura**:
        - **`controllers/`** → lógica dos endpoints (padrão Controller).
        - **`routes/`** → define as rotas e mapeia para os controllers.
        - **`models/`** (provavelmente relacionado ao Prisma ou schemas).
        - **`services/`** → lógica de negócio, intermediando entre controllers e banco.
        - **`middlewares/`** → validação, autenticação, etc.
        - **`utils/`** e **`types/`** → funções auxiliares e tipagem.
        - **Configuração**: `supabase.ts e prisma` centraliza as conexões.

    - **Padrão**:
      Controller-Service-Repository (quase um MVC simplificado):

        - Controller: trata requisições/respostas
        - Service: lógica da aplicação
        - Prisma funciona como Repository implícito

      Singleton (implícito): conexões com Supabase e Prisma geralmente seguem padrão singleton

      Middleware: padrão para interceptar e processar requisições antes de chegar ao controller
    - **Integração**: O backend atua como um proxy para o Supabase, manipulando autenticação e inserções nas tabelas do Prisma.

4. **Autenticação com Supabase:**

o Supabase Auth, com integração via backend:

- Backend como proxy seguro para manipular usuários/autenticação
- Frontend escuta eventos com **`onAuthStateChange`** para atualizar estado

Fluxo de dados simplificado
[Usuário] ⇄ [React App] ⇄ [Express API] ⇄ [Supabase/PostgreSQL]

5. **Banco de Dados: Supabase (Auth) + Postgresql (Prisma)**
    - **Tabelas**:
        - **auth.users** (gerenciada pelo Supabase): Armazena id, email, user_metadata (com full_name, avatar_url, etc.).
        - Prisma:
    - **Autenticação**: Supabase Auth gerência login com email/senha e OAuth (Google), fornecendo tokens JWT.

6. **Padrões e Princípios**
    - **Separação de Responsabilidades**: Frontend cuida da UI, backend gerência tabelas e integração com Supabase.
    - **REST**: Comunicação entre frontend e backend segue o padrão REST.
    - **Event-Driven**: Usamos `onAuthStateChange` para reagir a mudanças de autenticação no frontend.
    - **Minimalismo**: A arquitetura é simples e direta, sem camadas complexas, adequada para o escopo atual.

7. **Escalabilidade**
    - Adicionar mais endpoints no backend.
    - Introduzir um estado global no frontend (Context API ou Redux).
    - Separar o backend em microserviços (se necessário, mas improvável no momento).

Resumo da Arquitetura

- **Tipo**: Client-Server com React no frontend e uma API RESTful no backend.
- **Tecnologias**: React, TypeScript, Express, Supabase (Auth) e PostgreSQL(Prisma).