Conexão Treinamento - Documentação Completa

Agência Experimental de Engenharia de Software - PUCRS

Porto Alegre, 2025

SUMÁRIO

1 - INTRODUÇÃO

O presente documento tem como finalidade apresentar e consolidar informações essenciais do projeto Conexão Treinamento, abrangendo tanto seus aspectos técnicos quanto não técnicos. Aqui estão descritos o contexto do produto, proposta de valor, componentes tecnológicos etc.

O objetivo é oferecer uma visão clara e integrada do projeto, promovendo o entendimento dos stakeholders e foi elaborado para servir a diferentes públicos, como: desenvolvedores, equipe de DevOps/Infraestrutura, usuários finais e operadores de negócio.

O escopo deste documento inclui: guia de instalação e configuração do projeto, arquitetura do software, documentação da API e manual do usuário.

2 - VISÃO GERAL DO PROJETO

O Conexão Treinamento é uma plataforma web desenvolvida para solucionar os desafios de gestão enfrentados por academias e centros de treinamento do cliente. Tradicionalmente, as academias lidam com processos manuais que acabam gerando descentralização de informações e falta de integração entre dados de alunos, agendamentos e acompanhamentos de treinos. O sistema desenvolvido oferece uma solução completa e integrada que centraliza todas as operações em uma única plataforma, desde o cadastro detalhado de alunos com histórico médico, anamnese até gerenciamento de sessões, planos de treino personalizados e avaliações físicas periódicas.

2.1 - MOTIVAÇÃO E FUNCIONALIDADES

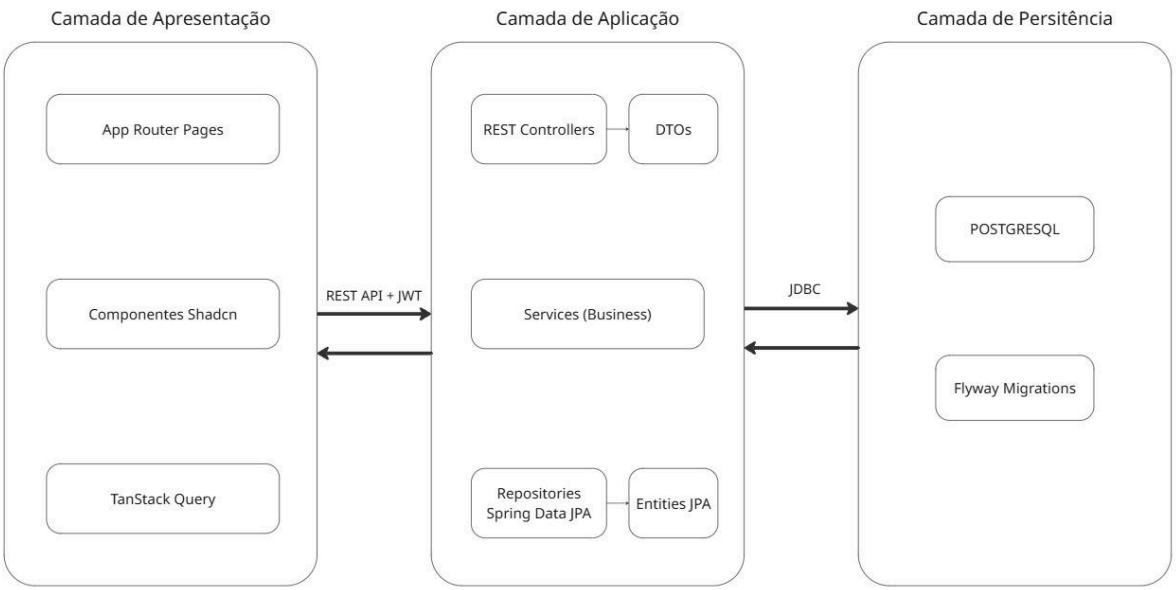
A motivação principal é auxiliar todo o processo de gestão da academia, proporcionando aos treinadores e administradores ferramentas eficientes para oferecer um atendimento de qualidade, baseado em dados concretos sobre a evolução de cada aluno, ao mesmo tempo que otimiza processos administrativos e melhora a experiência geral dos usuários.

O sistema oferece um conjunto completo de funcionalidades organizadas em módulos integrados: (1) módulo de gestão de usuários, responsável pelo cadastro e controle de administradores, treinadores e alunos, com autenticação segura e diferentes níveis de permissão; (2) módulo de gestão de alunos, que inclui cadastro completo com dados pessoais, endereço, contatos de emergência e um questionário de anamnese detalhado com histórico médico, uso de medicações e comprometimentos físicos; (3) módulo de avaliações físicas, que possibilita o registro periódico de medidas como peso, altura, IMC, circunferências, dobras cutâneas e diâmetros para acompanhamento da evolução; (4) módulo de treinos, que oferece uma biblioteca de exercícios e permite a criação de planos personalizados atribuídos aos alunos com duração definida, mantendo o histórico; (5) módulo de agendamentos, que gerencia sessões de treino com controle de capacidade, permitindo inscrições em horários disponíveis, registro de presença e gestão da disponibilidade dos treinadores por dia da semana; (6) módulo de eventos, que possibilita a organização de atividades especiais como workshops, competições e avaliações coletivas, com sistema de inscrições e controle de participação; e (7) módulo de relatórios, que gera análises e estatísticas sobre o desempenho dos treinadores, a distribuição etária dos alunos e outras métricas relevantes para a tomada de decisões estratégicas.

3 - ARQUITETURA

O Conexão Treinamento adota uma arquitetura de três camadas no padrão cliente-servidor, com separação de responsabilidades entre frontend, backend e camada de persistência. A comunicação entre camadas é realizada através de uma API REST totalmente documentada com OpenAPI. Esta abordagem permite que frontend e backend evoluam de maneira independente, possibilitando inclusive a criação de múltiplos clientes (web, mobile e desktop) que consumam a mesma API.

Abaixo o diagrama conceitual da aplicação:



4 - BACKEND

O projeto do backend foi construído utilizando as seguintes tecnologias principais:

1. Spring Boot: framework Java para desenvolvimento
2. Java 21: versão LTS mais recente da linguagem (outubro de 2025)
3. PostgreSQL 16: banco de dados relacional
4. Spring Data JPA: abstração de persistência com Hibernate como ORM
5. Flyway: versionamento e migração de schemas de banco de dados

4.1 - ARQUITETURA EM CAMADAS

A primeira camada é a camada de apresentação (controllers), responsável por expor a API e gerenciar requisições HTTP. Os controllers recebem as requisições, delegam processamento para a camada de serviço e retornam respostas padronizadas. Toda a documentação é gerada automaticamente

através de anotações OpenAPI, disponibilizando uma interface interativa com o Swagger UI.

A segunda camada é a camada de negócio (services). Esta camada é responsável por conter toda a lógica de negócio da aplicação, orquestrando operações mais complexas que envolvem múltiplas entidades, aplicando regras de validação, controlando transações e coordenando a comunicação entre diferentes partes do sistema. Esta camada chama a camada de acesso aos dados para aplicar regras e validações necessárias.

A terceira camada é a camada de acesso a dados (repositories) que é a responsável por abstrair toda a interação com o banco de dados utilizando Spring Data JPA. As interfaces de repositório estendem JPARepository e JpaSpecificationExecutor, permitindo operações CRUD automáticas e queries dinâmicas. Para buscas customizadas, utiliza-se o padrão Specification que implementa queries via JPA Criteria API, possibilitando construção dinâmica de consultas complexas com filtros combinados.

A quarta e última camada é a camada de persistência (entities) que representa o modelo de domínio com entidades JPA que mapeiam as tabelas do banco de dados. As entidades definem relacionamentos entre si e incluem auditoria automática (timestamps de criação e atualização). O projeto adota o padrão de Soft Delete, onde registros não são removidos fisicamente, mas marcados com timestamp na coluna “deleted_at”.

4.2 - PRINCÍPIOS APLICADOS

Alguns princípios aplicados no projeto são:

1. Separação de Responsabilidades: cada camada possui um propósito específico e uma única responsabilidade
2. Inversão de Dependência: camadas superiores dependem de abstrações (interfaces)

3. Data Transfer Objects (DTOs): separação entre modelo de domínio e API externa, nunca devolvendo a entidade do banco de dados diretamente para o cliente
4. Validação em múltiplas camadas: validação básica na camada de apresentação e regras complexas na camada de negócio.

4.3 - SEGURANÇA

A autenticação é implementada utilizando JWT (JSON Web Tokens) seguindo o padrão OAuth2. O fluxo inicia com login via credenciais, gerando um token JWT assinado que deve ser enviado no header Authorization, no formato "Bearer <token>" nas chamadas seguintes. A maioria dos endpoints requer o uso do header de autorização, portanto é essencial o login antes do uso da API.

O controle de acesso é baseado em roles (ROLE_ADMIN e ROLE_TRAINER), permitindo restrições granulares por endpoint. Apenas o endpoint /auth/login é público, os demais exigem autenticação válida.

4.4 - TESTES

O backend possui testes organizados em duas categorias, sendo elas: testes unitários e testes de integração. Todos os testes são escritos com JUnit.

Os testes unitários focam em testar componentes de maneira isolada com uso extensivo de mocks com Mockito. A estrutura espelha a organização do código principal.

Já os testes de integração, utilizam TestContainers para provisionar um container Docker com PostgreSQL real durante a execução, garantindo que a integração completa entre todas as camadas funcione corretamente. Eles tem como características um banco de dados PostgreSQL 16 real, perfil de teste isolado (@ActiveProfiles("test")) e validação end-to-end, ou seja, requisição HTTP chamando controller, service, repository e banco de dados.

4.5 - ESPECIFICAÇÃO OPENAPI

O projeto mantém uma especificação OpenAPI atualizada exposta pelo backend em */v3/api-docs* e com visualização exposta em */swagger-ui.html*, que atua como "fonte da verdade" do contrato da API. Essa especificação descreve endpoints, modelos, códigos de resposta e requisitos de autenticação e é consumida pelo frontend para gerar automaticamente um cliente tipado. Com isso, garantimos alinhamento entre equipes, tipagem end-to-end, documentação navegável e detecção precoce de quebras de contrato.

Para atualizar a especificação (arquivo *oplenapi.yml*) existem 3 métodos:

1. Via curl:

```
curl -s http://localhost:8080/v3/api-docs | python3 -c "import json, sys, yaml;
print(yaml.dump(json.load(sys.stdin), default_flow_style=False,
sort_keys=False, allow_unicode=True, width=120))" > API/openapi.yml
```

2. Baixar JSON e converter manualmente:

```
curl -s http://localhost:8080/v3/api-docs -o openapi.json
python3 -c "import json, yaml; f=open('openapi.json'); data=json.load(f);
f.close(); yf=open('API/openapi.yml','w'); yaml.dump(data, yf,
default_flow_style=False, sort_keys=False, allow_unicode=True); yf.close()"
rm openapi.json
```

3. Via Maven plugin

```
./mvnw springdoc:generate
```

A especificação é gerada automaticamente pelo SpringDoc a partir de anotações nos controllers, mantenha o arquivo *openapi.yml* sempre atualizado e sincronizado com a API em produção. Use a especificação para documentação externa, geração de clientes e testes de contrato. Lembre-se que todos os endpoints exigem autenticação JWT, exceto endpoint de login */auth/login*.

5 - FRONTEND

O projeto frontend do Conexão Treinamento é uma aplicação React que utiliza Next.js 15, React 19 e TypeScript. Ele fornece uma interface moderna e responsiva para administradores e treinadores interagirem com os recursos do sistema (agendamentos, planos, avaliações, eventos etc.)

É uma Single Page Application (SPA) com App Router, cliente de API tipado gerado automaticamente e cache/estado do servidor com TanStack Query. Além das tecnologias citadas anteriormente, o projeto utiliza Tailwind CSS + shadcn/ui para design system consistente, acessível e facilmente customizável.

5.1 - ESTRUTURA E ARQUITETURA

O código fonte do frontend fica localizado no diretório */web* e está organizado da seguinte maneira:

1. ***app/***: contém as rotas da aplicação, seguindo a convenção do App Router do Next.js. Cada pasta corresponde a um segmento de URL.
2. ***components/***: onde ficam localizados os componentes React reutilizáveis, organizados por funcionalidade
3. ***components/ui/***: componentes de UI genéricos de baixo nível, padrão do shadcn/ui
4. ***lib/***: contém o cliente de API, configurações, utilitários
5. ***hooks/***: hooks customizados do React
6. ***style/***: estilos globais

5.2 - ROTEAMENTO E LAYOUT

Para fazer o roteamento, o projeto utiliza o App Router e organiza as rotas: *administrators, students, trainers, schedule, events, plans, reports, dashboard* e *profile*.

O arquivo *layout.tsx* define provedores globais e guarda de autenticação para proteger rotas e gerenciar redirecionamentos. Abaixo, trecho do arquivo de layout:

```
return (
  <html lang="pt-BR" suppressHydrationWarning>
    <body className={inter.className}>
      <Providers>
        <AuthGuard>
          {children}
        </AuthGuard>
      </Providers>
    </body>
  </html>
)
```

5.3 - AUTENTICAÇÃO E AUTORIZAÇÃO

A aplicação exige autenticação para ser utilizada, então após o login, o token JWT retornado pela API é armazenado no *localStorage* do navegador e enviado automaticamente no header Authorization em cada requisição via cliente.

O componente <AuthGuard /> é responsável por verificar se o token existe e se não está expirado. Caso contrário, os dados serão limpos e o usuário redirecionado para o login.

```
  if (typeof window !== "undefined") {
    const storedToken = localStorage.getItem("token");
    if (!storedToken || isTokenExpired(storedToken)) {
      localStorage.removeItem("userRole");
      localStorage.removeItem("userName");
      localStorage.removeItem("token");
      router.push("/");
      setChecking(false);
      return;
    }
```

5.4 - COMUNICAÇÃO COM A API (CLIENTE GERADO)

O cliente é gerado a partir da especificação OpenAPI do backend, a configuração do gerador aponta para *v3/api-docs* do backend em desenvolvimento. Os hooks gerados com integração ao TanStack Query ficam em **lib/api-client/@tanstack/react-query.gen.ts**. O projeto também expõe wrappers amigáveis (por exemplo: useLoginMutation) para padronizar opções e side effects.

Para gerar o cliente OpenAPI é necessário que o backend esteja rodando e acessível e que o NPM e o pnpm estejam instalados. Após, é preciso iniciar o backend (localmente ou via Docker) e, no diretório **web/**, executar:

```
pnpm install
pnpm run generate-api-client
```

O comando lê a especificação e escreve o cliente tipado e hooks em **lib/api-client/.**

5.5 - GERENCIAMENTO DE ESTADO DO SERVIDOR

O projeto utiliza o TanStack Query (React Query) para gerenciar o estado do servidor. Para busca de dados, o hook *useQuery* é usado e é responsável por gerenciar cache, revalidação em segundo plano e estado de carregamento/erro. Já para criar, atualizar e excluir dados é utilizado o hook *useMutation*. Ele fornece uma maneira simples de lidar com efeitos colaterais das mutações, como a invalidação de queries em cache.

Exemplo de wrapper de login:

```
export const useLoginMutation = (
  options?: UseMutationOptions<LoginResponse, DefaultError,
LoginVariables>
) => {
  const base = loginMutation({ client: apiClient })

  return useMutation({
    ...base,
```

```
    ...options,
  })
}
```

Uso prático (exemplo ilustrativo):

```
import { useLoginMutation } from "@/lib/auth/hooks/auth-mutations";

export function LoginForm() {
  const { mutate, isPending, error } = useLoginMutation({
    onSuccess: (data) => {
      localStorage.setItem("token", data.token);
      // navegar, carregar perfil, etc.
    },
  });

  // chame mutate({ body: { email, password } })
  return null;
}
```

5.6 - FORMULÁRIOS E VALIDAÇÃO

É utilizado o React Hook Form para gerenciar estado e performance dos formulários da aplicação, além de Zod para definir schemas e mensagens de validação consistentes e o shadcn/ui para definir UX consistente (componentes de input, select, toast etc).

Erros de rede ou autorização são capturados pelos hooks e o usuário receberá feedbacks em formatos de toast ou mensagem. Quando há expiração de sessão, o componente <AuthGuard /> valida e redireciona o usuário de volta para a página de login.

5.7 - CONFIGURAÇÃO, AMBIENTE E EXECUÇÃO

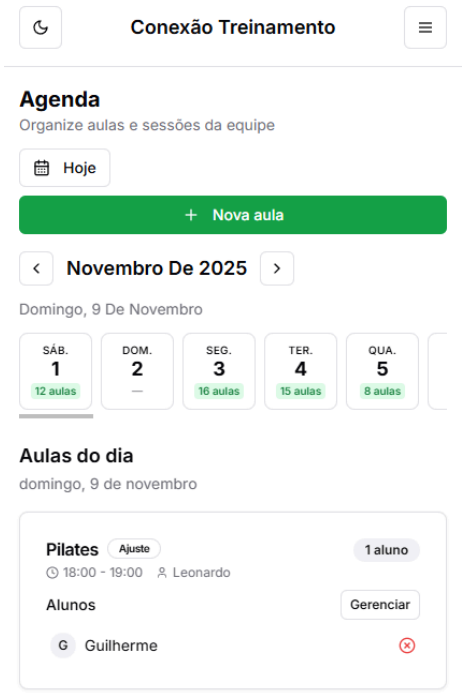
O projeto do frontend possui apenas *NEXT_PUBLIC_API_URL* como variável de ambiente e o seu valor deve ser configurado para apontar para a URL base da API.

Já para a execução, existem alguns comandos úteis para desenvolvimento, sendo eles:

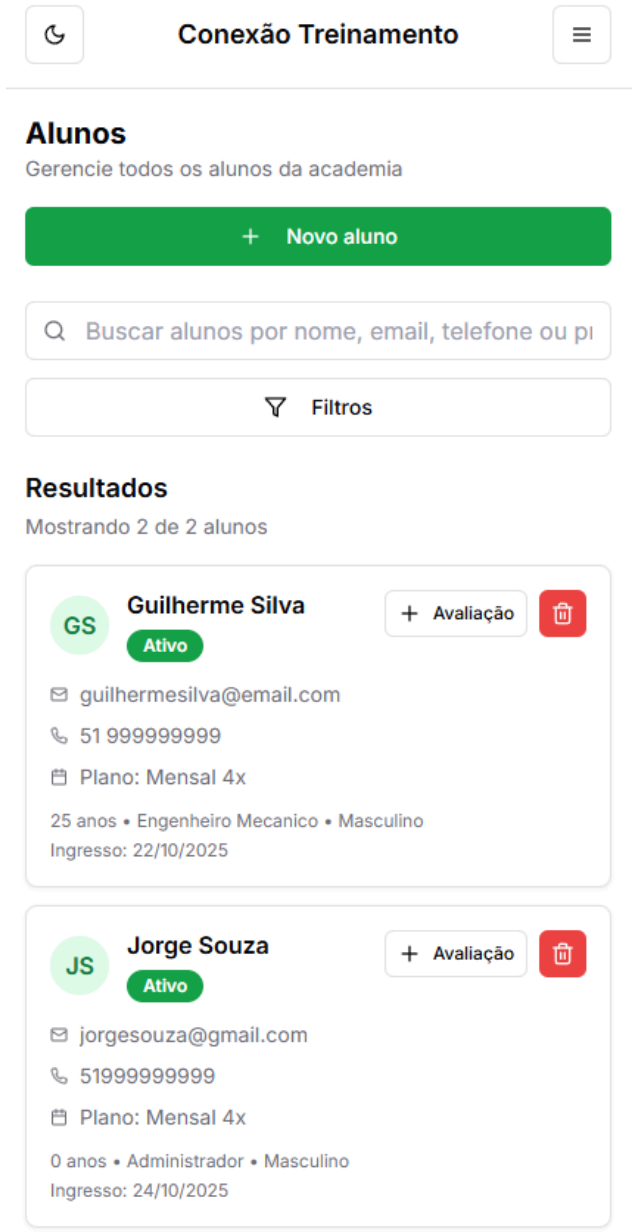
1. pnpm install: instalar dependências
2. pnpm dev: executar o projeto localmente
3. pnpm build: gerar build de produção
4. pnpm start: sobe servidor de produção servindo o conteúdo gerado por pnpm buld
5. pnpm run generate-api-client: lê a especificação do backend e gera cliente tipado

5.8 - PRINCIPAIS TELAS DA APLICAÇÃO

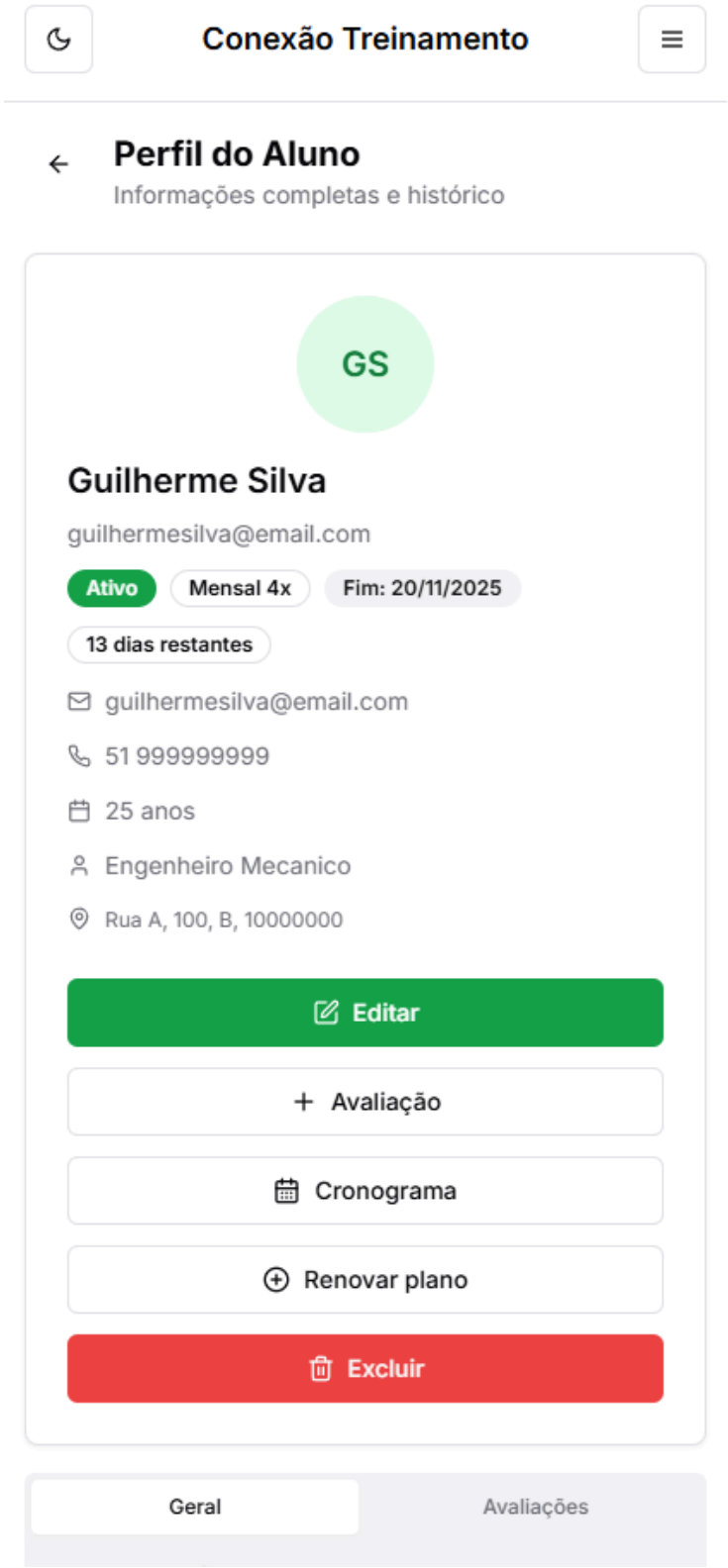
Agenda de aulas:



Lista de alunos:

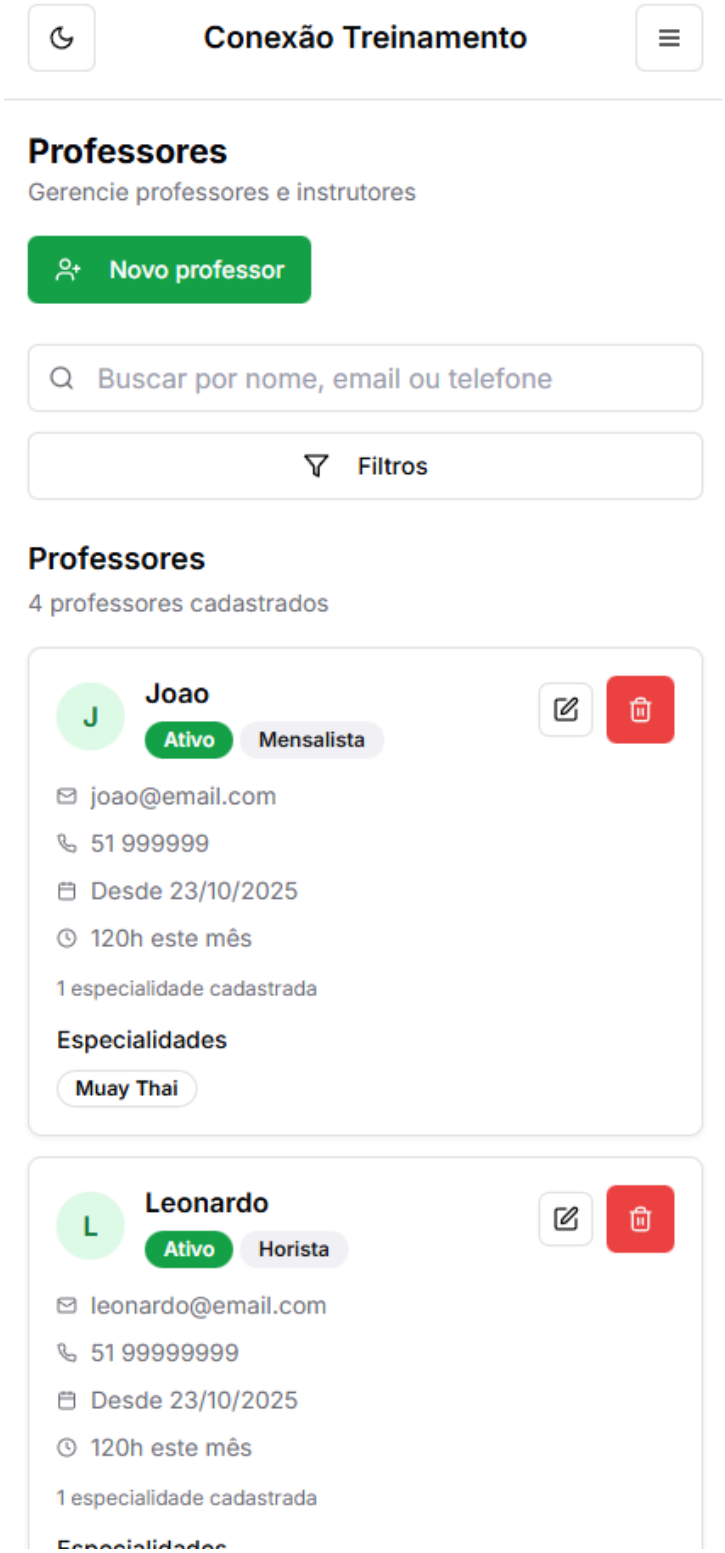


Perfil do aluno:

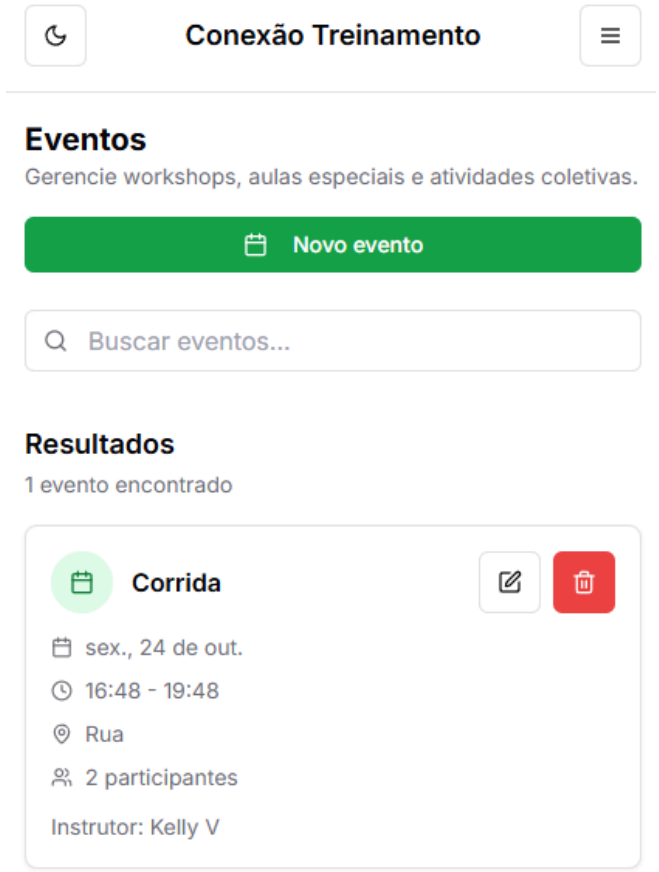


Avaliação Física:

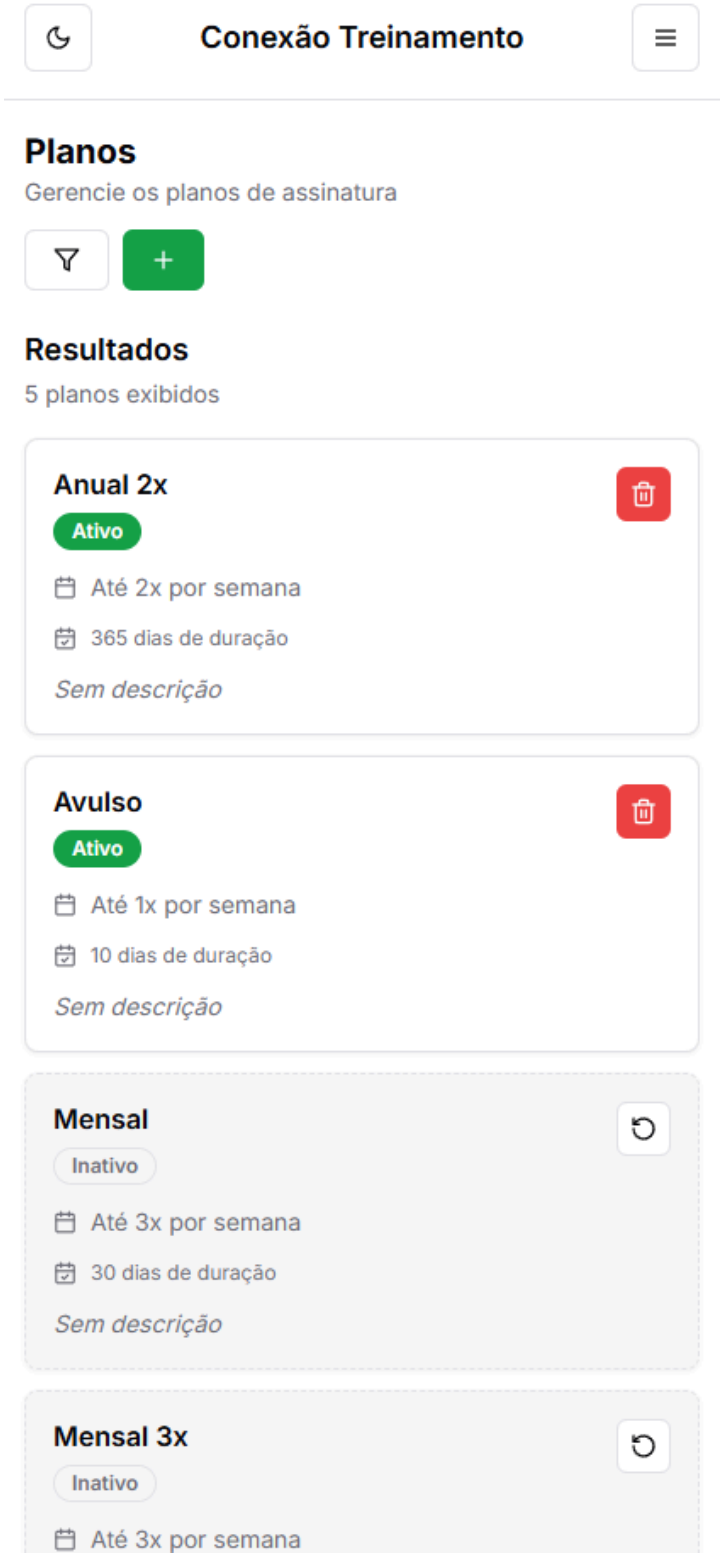
Lista de Professores:



Listagem de Eventos:



Lista de planos:



Tela de relatórios:

Performance dos professores

Detalhamento de horas trabalhadas e aulas ministradas por professor

J

Joao
Mensalista

Horas
0h

Aulas
0

Alunos
0

Especialidades
Muay Thai

KV

Kelly V
Horista

Horas
1h

Aulas
1

Alunos
2

Especialidades
—

L

Leonardo
Horista

Horas
1h

Aulas
1

6 - BANCO DE DADOS

O projeto utiliza PostgreSQL como banco de dados relacional, versionado através de migrações utilizando o Flyway. O modelo de dados cober autenticação e perfis de usuários, avaliações físicas, exercícios, planos e agendamentos, além de eventos. O desenho privilegia integridade referencial, normalização adequada e índices para consultas frequentes.

Foram habilitadas as extensões **pg_trm** e **unaccent.**

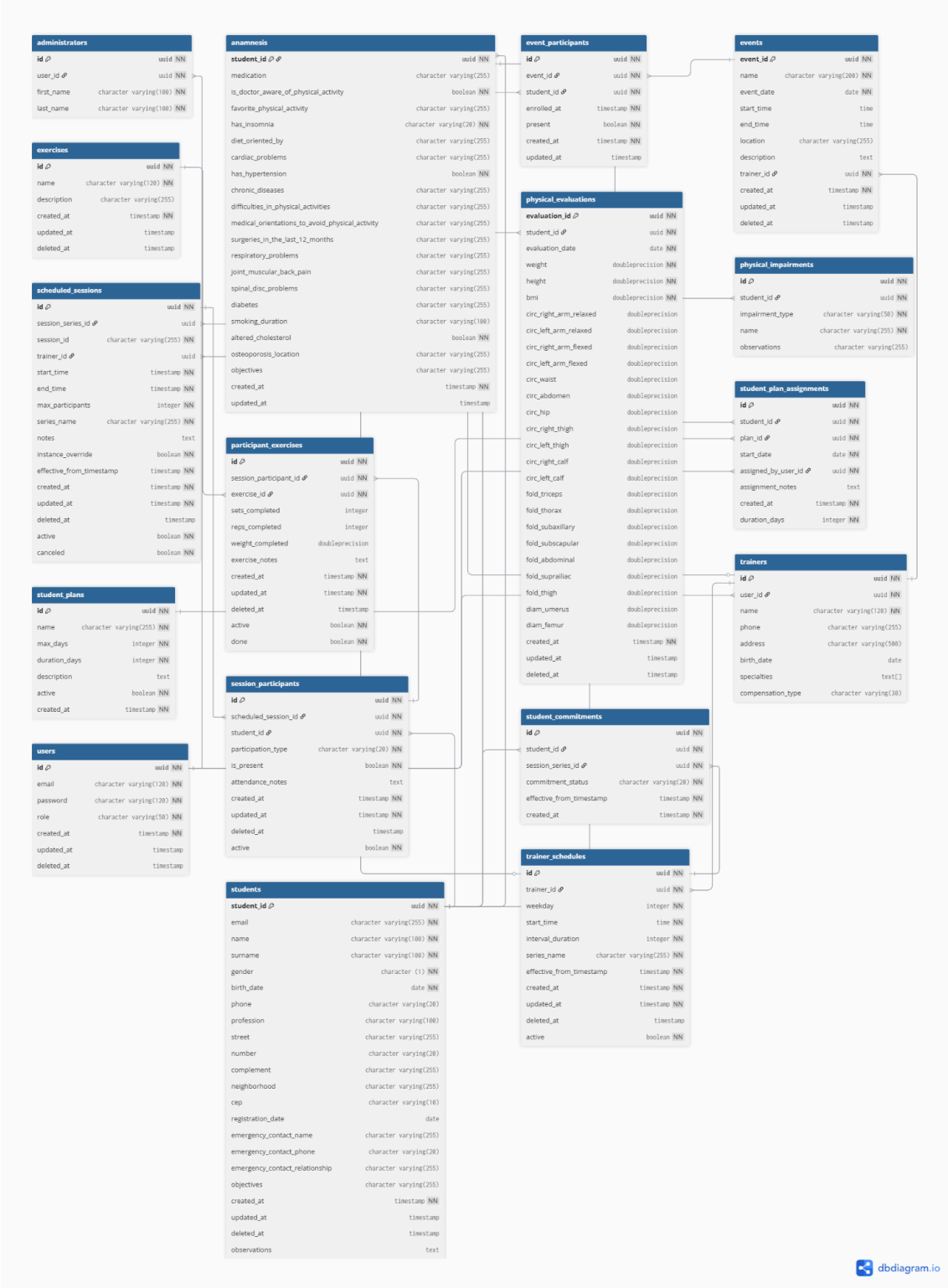
6.1 - MODELAGEM

A modelagem abrange as entidades do domínio (usuários, estudantes e treinadores), avaliação física, exercícios, planos, disponibilidade de treinadores e sessões agendadas, além de participantes, compromissos e eventos.

Algumas convenções foram adotadas ao definir a modelagem dos dados, como: utilização de UUID como chaves primárias, colunas de auditoria frequentes

(created_at, updated_at e deleted_at) e nomes descritivos e consistentes para FKs e constraints;

Diagrama da modelagem:



6.2 - INTEGRIDADE E RELACIONAMENTOS

O projeto utiliza integridade referencial com chaves estrangeiras e deleções em cascata onde fazem sentido (por exemplo, remover um aluno irá remover sua anamnese).

A aplicação possui alguns relacionamentos chave, como:

1. Users → Administrators/Trainers
2. Students → Anamnesis/Physical Evaluations/Physical Impairments
3. Trainer Schedules → Scheduled Sessions → Session Participants → Participant Exercises
4. Student Plans → Student Plan Assignments
5. Events → Events Participants

6.3 - INDÍCES E PERFORMANCE

Alguns índices foram criados para colunas de busca e ordenação frequentes. Os índices principais são:

1. Students
    a. IDX_STUDENTS_EMAIL: busca por email
    b. IDX_STUDENTS_NAME_SURNAME: busca por nome e sobrenome
    c. IDX_STUDENTS_CREATED_AT: ordenação por ordem de criação
    d. IDX_STUDENTS_DELETED_AT: filtros de soft delete
2. Administrators:
    a. IDX_ADMINISTRATORS_USER_ID: junções com users
    b. IDX_ADMINISTRATORS_NAME: busca por nome/sobrenome
3. Anamnesis:
    a. IDX_STUDENT_ID: junções e filtros por aluno

7 - GUIA DE INSTALAÇÃO E CONFIGURAÇÃO

O projeto pode ser executado via Docker (recomendado para onboarding mais rápido) ou localmente executando os serviços separadamente.

7.1 - USANDO DOCKER

Para executar o projeto com o Docker é necessário possuir Docker e Docker Compose instalados. Para subir todos os serviços, o usuário deve entrar no diretório raíz do projeto e executar:

```
docker compose up -d
```

Este comando irá subir PostgreSQL, Backend e Frontend. O banco de dados executará em localhost:5432, o backend será executado em http://localhost:8080 e o frontend em http://localhost:3000.

7.2 - EXECUTANDO SEM DOCKER

É necessário ter Java 21, Maven, Node, PNPM e PostgreSQL instalados.
É necessário configurar e criar um banco de dados local, ouvindo na porta 5432. Já para executar o backend basta executar o seguinte comando na pasta **backend/** para subir a API em *http://localhost:8080*:

```
# (opcional) exporte variáveis caso precise customizar
# DB_URL, POSTGRES_USER, POSTGRES_PASSWORD, JWT_SECRET
./mvnw spring-boot:run
```

Para rodar o frontend é preciso executar o comando abaixo para que fique disponível em *http://localhost:3000*:

```
pnpm install
# aponte para a API local (padrão já é http://localhost:8080)
setx NEXT_PUBLIC_API_URL "http://localhost:8080"
pnpm dev
```