ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

P.PORTO

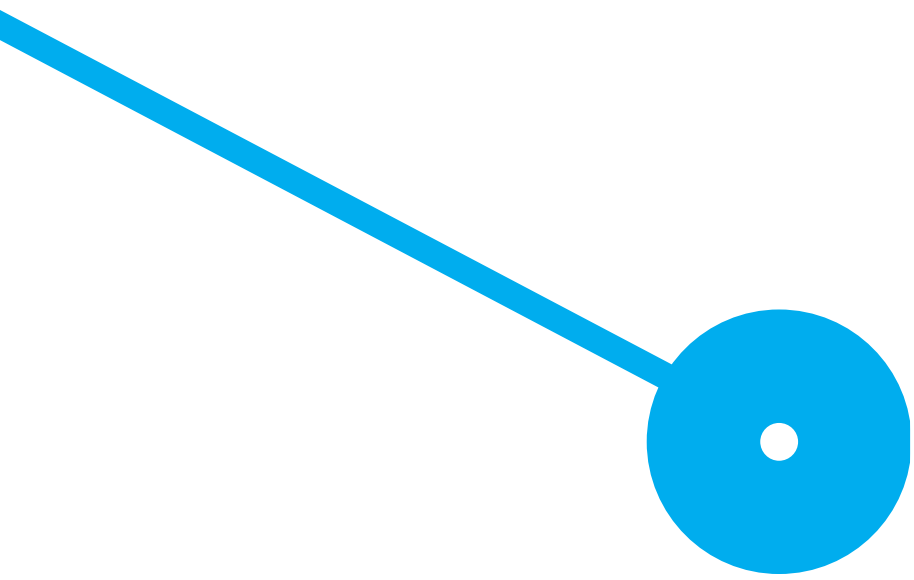
CTeSP CURSO TÉCNICO SUPERIOR PROFISSIONAL
Desenvolvimento para a Web e Dispositivos Móveis

FitLife

(8240558) Tiago Queirós

(8240549) Loureço Nóbrega

2025/2026



1. Índice

Conteúdo

2. Credenciais de Teste

Ao iniciar o backend pela primeira vez, são criados automaticamente dois utilizadores de teste para facilitar a avaliação:

ADMINISTRADOR	PERSONAL TRAINER
Email: admin@gym.com	Email: trainer@gym.com
Password: admin	Password: treinador
Acesso: /login	Acesso: /login

NOTA: Os utilizadores são criados automaticamente no ficheiro backend/index.js na ligação à base de dados MongoDB.

3. Introdução

3.1.Enquadramento do Tema

O presente projeto foi desenvolvido no âmbito da unidade curricular de Programação Web Avançada, do 2.º ano do CTeSP em Desenvolvimento para a Web e Dispositivos Móveis.

O tema proposto consiste na criação de uma plataforma web para Personal Trainers, denominada "FitLife", que serve de suporte à gestão e acompanhamento de treinos entre personal trainers e os seus clientes.

3.2.Objetivos do Projeto

OBJETIVOS TÉCNICOS:

- Desenvolver uma API RESTful em Node.js/Express
- Criar uma Single Page Application (SPA) em React
- Implementar autenticação segura com JWT
- Utilizar MongoDB como base de dados NoSQL
- Implementar comunicação real-time com Socket.IO
- Documentar a API com Swagger/OpenAPI

OBJETIVOS FUNCIONAIS:

- Sistema de registo e autenticação (username/password e QR Code)
- Três perfis de utilizador: Administrador, Personal Trainer e Cliente
- Gestão de planos de treino personalizados

- Sistema de consulta e registo de cumprimento de treinos
- Dashboard com estatísticas e gráficos
- Sistema de notificações automáticas
- Chat em tempo real entre trainer e cliente
- Suporte a tema escuro e tema claro

3.3.Requesitos Implementados

Suporte a tema escuro e tema claro (ThemeProvider com Context API)

Registo e autenticação de utilizadores

- Registo com validação de email e nome únicos
- Login com username/password
- Login por QR Code (QrcodeCreate/QrcodeRead)
- Perfis de utilizador com informações pessoais e histórico
- Perfil administrador com gestão de trainers e clientes

Gestão de planos de treino personalizados

- Planos semanais com exercícios, séries e repetições
- Configuração para 3, 4, 5, 6 ou 7 treinos por semana
- Limite de até 10 exercícios por sessão
- Anexar instruções ou links de vídeo para exercícios
- Filtros de pesquisa e ordenação por dia e cliente

Consulta de treinos e registo de cumprimento

- Vista estilo calendário para o cliente
- Registo diário de cumprimento/não cumprimento
- Indicação de motivo em caso de não cumprimento

- Upload de imagem como prova de cumprimento (Cloudinary)

Dashboards

- Gráficos com treinos concluídos por semana/mês (Recharts)

Notificações automáticas

- Toast ao trainer quando cliente falha treino (Socket.IO)

Comunicação entre trainers e clientes

- Chat em tempo real com Socket.IO
- Notificação toast para novas mensagens
- Alertas do trainer para treinos faltados

Gestão de Conteúdo

- Interface de administração segura
- Adicionar/remover personal trainers
- Personal trainer adiciona clientes

Swagger da API

Testes de Software (Jest + Vitest)

4. Mockups

4.1.Área de Autenticação

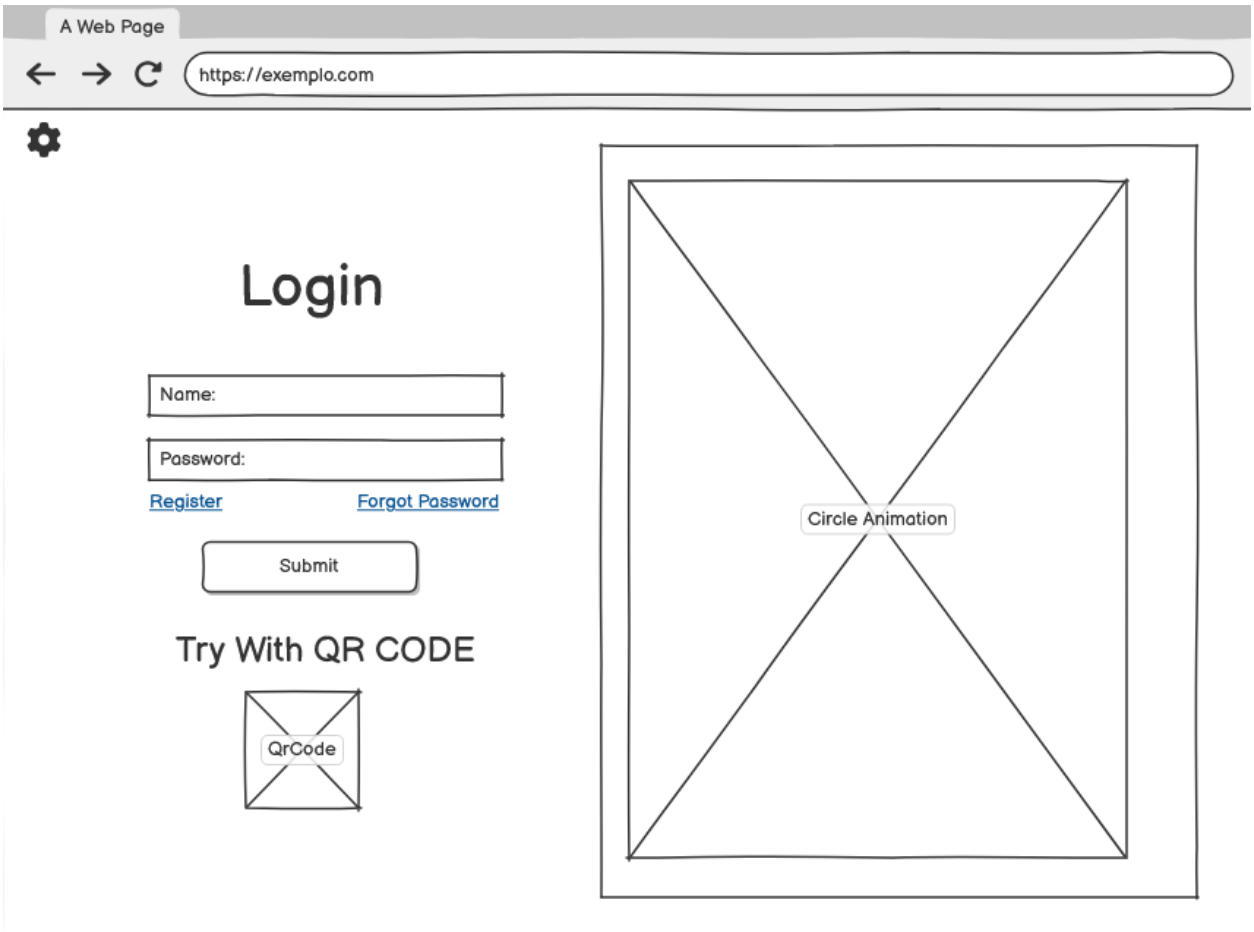


Figura 1-Area de Autenticação

4.2.Personal Trainer - Dashboard

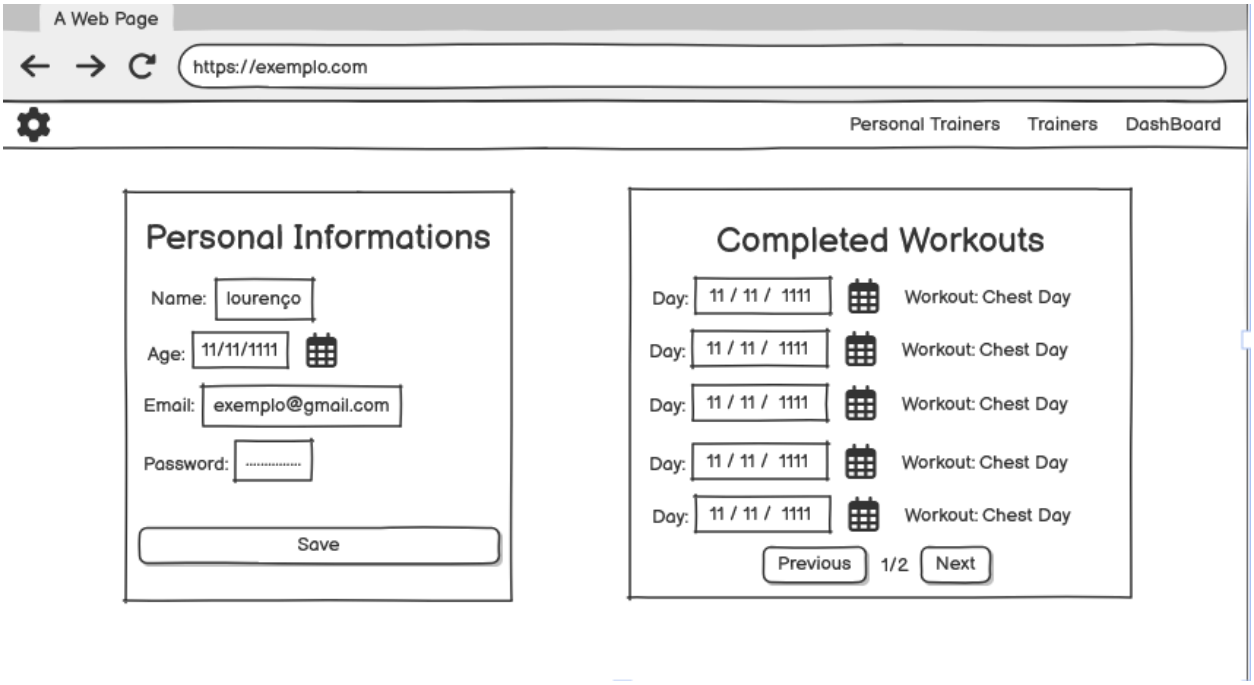


Figura 2-Personal Trainer(Dashboard)

4.3.Admin Page - Personal Trainer

Figura 3-Admin Page-Personal Trainer

4.4.Admin Page - Trainers

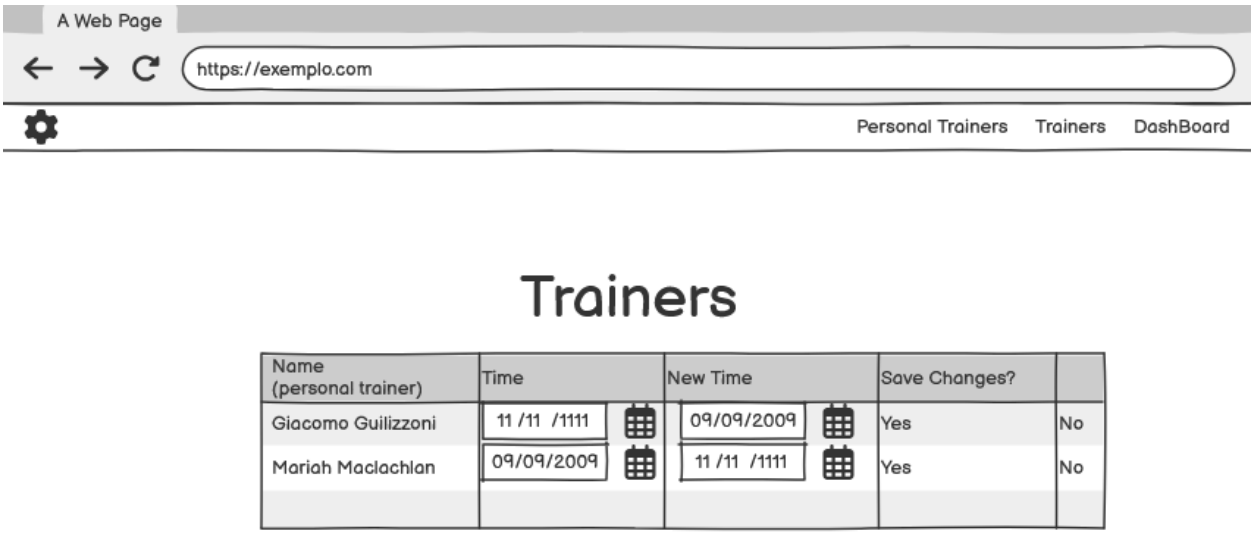


Figura 44- Admin Page-Trainers

4.5.Personal Trainers - Workout

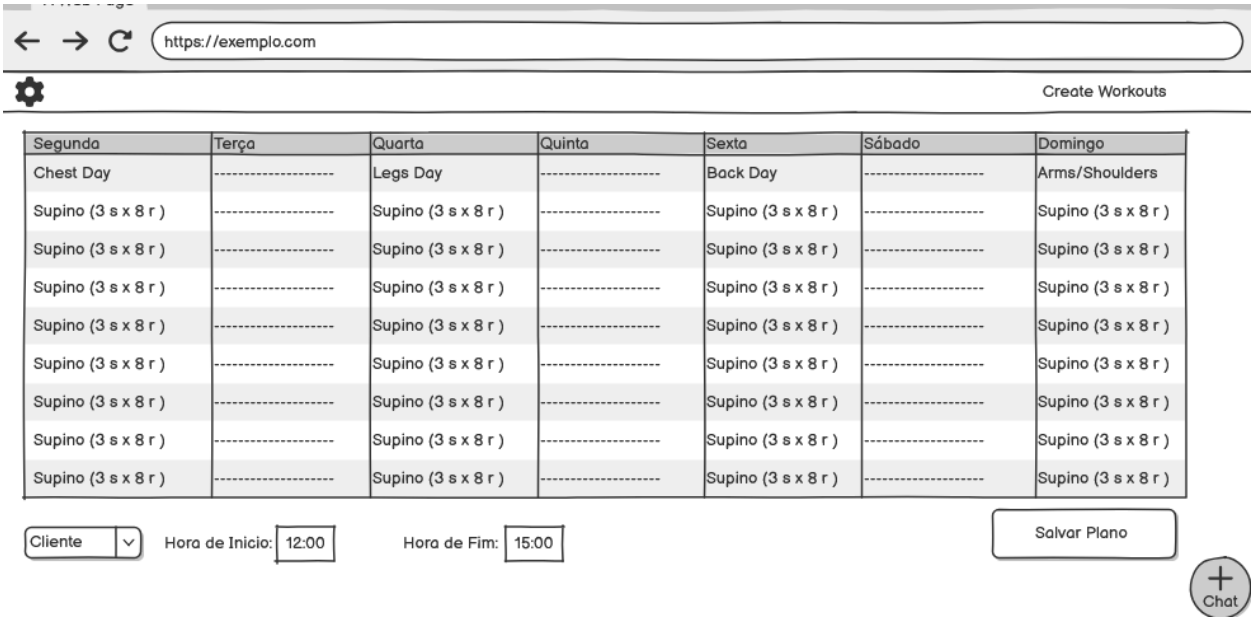


Figura 5-Personal Trainers – Workout

4.6.Clients

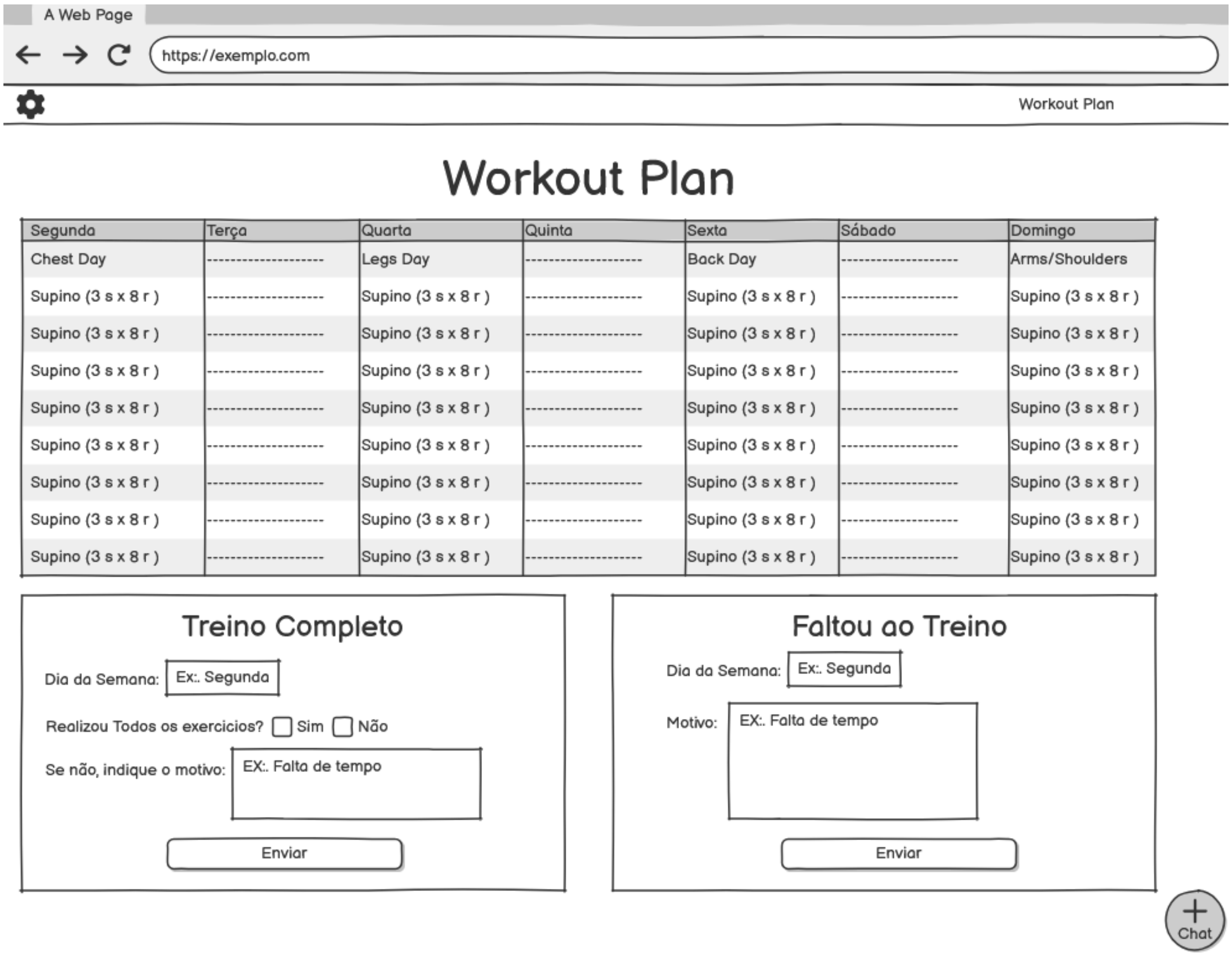


Figura 6-Clients

4.7.Dashboard

Figura 7-Dashboard

4.8.Real Time Chat

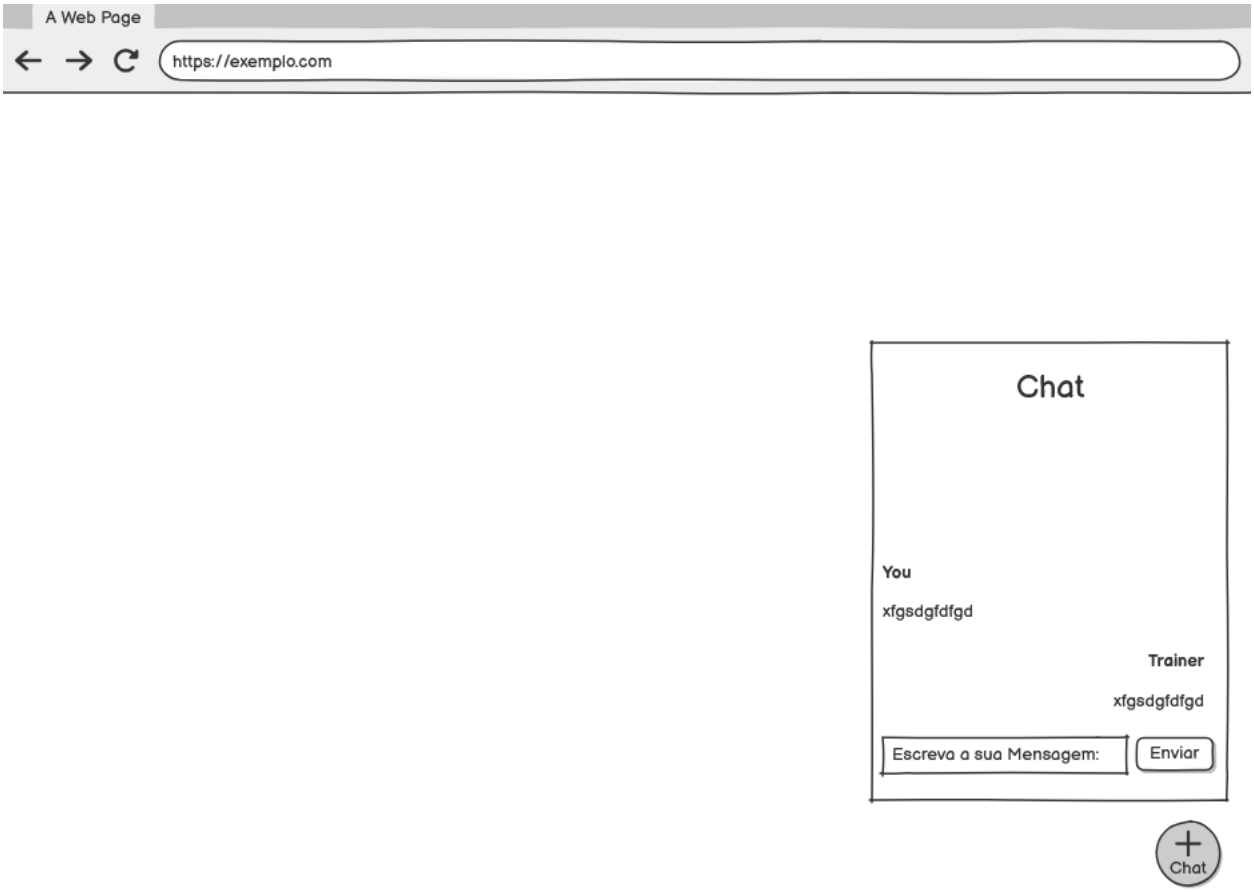


Figura 8-Real Time Chat

4.9.Admin Page

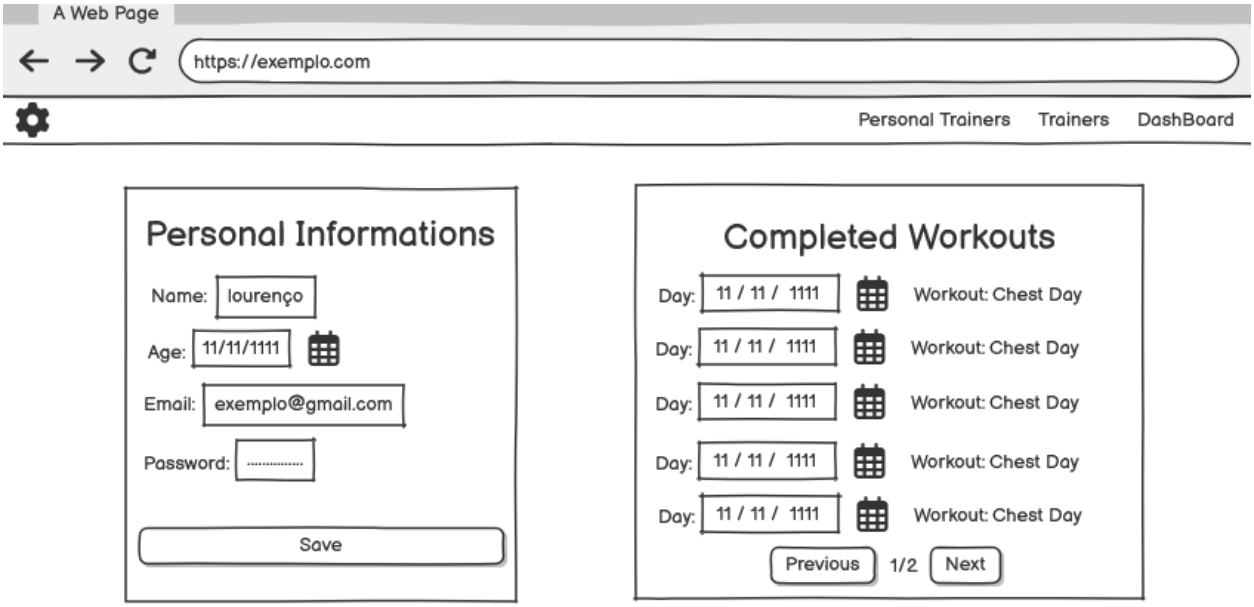


Figura 9-Admin Page

5. Implementação do Trabalho Prático

5.1.Dificuldades Encontradas

DIFICULDADE 1: Gestão de Roles e Permissões

Descrição:

Implementar um sistema flexível onde cada role (admin, trainer, user) tem permissões específicas em diferentes endpoints.

Solução:

Criação de um middleware RBAC (Role-Based Access Control) com scopes. Cada utilizador tem um array de scopes e o middleware verifica se possui o scope necessário para aceder ao endpoint.

DIFICULDADE 2: Comunicação Real-time

Descrição:

Implementar notificações instantâneas quando um cliente falta a um treino ou quando há uma nova mensagem de chat.

Solução:

Integração do Socket.IO no servidor Express, emitindo eventos específicos (workout-missed, new-message) que o frontend escuta e exibe como toasts.

DIFICULDADE 3: Upload de Imagens

Descrição:

Permitir upload de fotos de perfil e provas de treino de forma eficiente sem sobrecarregar o servidor.

Solução:

Utilização do Cloudinary como CDN, com Multer para processar os ficheiros em memória antes do upload. Transformações automáticas (resize 300x300 para perfil, 800x800 para provas).

DIFICULDADE 4: Associação Trainer-Cliente

Descrição:

Garantir que um cliente só pode estar associado a um trainer, e que a mudança requer aprovação do admin.

Solução:

Sistema de códigos de convite únicos gerados pelo trainer (formato: PT-NOME-XXXX), e workflow de DisassociationRequest com estados pending/approved/rejected.

DIFICULDADE 5: Validação de Formulários

Descrição:

Manter consistência entre validações do frontend e backend.

Solução:

React Hook Form no frontend com regras de validação, e validação server-side em todos os endpoints antes de persistir dados.

DIFICULDADE 6: Gestão de Estado entre Componentes

Descrição: Partilhar dados do utilizador autenticado entre vários componentes.

Solução: Context API com UsersProvider e ThemeProvider, evitando prop drilling.

5.2.Recursos utilizados

5.2.1. BACKEND - Bibliotecas/Packages:

- express (4.21.2) - Framework web
- mongoose (6.13.8) - ODM para MongoDB
- jsonwebtoken (8.5.1) - Autenticação JWT
- bcrypt (5.0.1) + bcryptjs (3.0.3) - Hash de passwords
- socket.io (4.8.1) - Comunicação real-time
- nodemailer (7.0.10) - Envio de emails
- cloudinary (2.8.0) - Upload de imagens CDN
- multer (2.0.2) - Processamento de ficheiros
- swagger-jsdoc (6.2.8) + swagger-ui-express (5.0.1) - Documentação API
- cors (2.8.5) - Cross-Origin Resource Sharing
- cookie-parser (1.4.6) - Parsing de cookies
- dotenv (17.2.3) - Variáveis de ambiente

5.2.2. BACKEND - DevDependencies:

- jest (30.2.0) - Framework de testes
- supertest (7.1.4) - Testes de API HTTP
- cross-env (10.1.0) - Variáveis de ambiente cross-platform
- nodemon (2.0.22) - Hot reload em desenvolvimento

5.2.3. FRONTEND - Bibliotecas/Packages:

- react (18.2.0) + react-dom (18.2.0) - Biblioteca UI
- react-router-dom (6.4.2) - Routing SPA
- react-hook-form (7.37.0) - Gestão de formulários
- socket.io-client (4.8.1) - Cliente Socket.IO
- recharts (3.5.1) - Gráficos e charts
- sweetalert2 (11.26.17) - Alertas e modais personalizados
- react-toastify (11.0.5) - Notificações toast
- swiper (12.0.3) - Carrossel/slider
- lucide-react (0.562.0) - Ícones
- classnames (2.3.2) - Gestão de classes CSS
- sass (1.55.0) - Pré-processador CSS
- bootstrap (5.3.8) + reactstrap (9.1.4) - Componentes UI
- react-qr-code (2.0.18) + @yudiel/react-qr-scanner (2.4.1) - QR Code
- lodash (4.17.21) – Utilitários

5.2.4. FRONTEND - DevDependencies:

- vite (7.1.5) - Build tool
- vitest (3.2.4) - Framework de testes
- @testing-library/react (16.3.1) - Testes de componentes
- @testing-library/jest-dom (6.9.1) - Matchers DOM
- tailwindcss (4.1.18) - Utility CSS
- jsdom (27.3.0) - Simulação de DOM

5.3. BACKEND (Node.js/Express)

5.3.1. Estrutura do Projeto

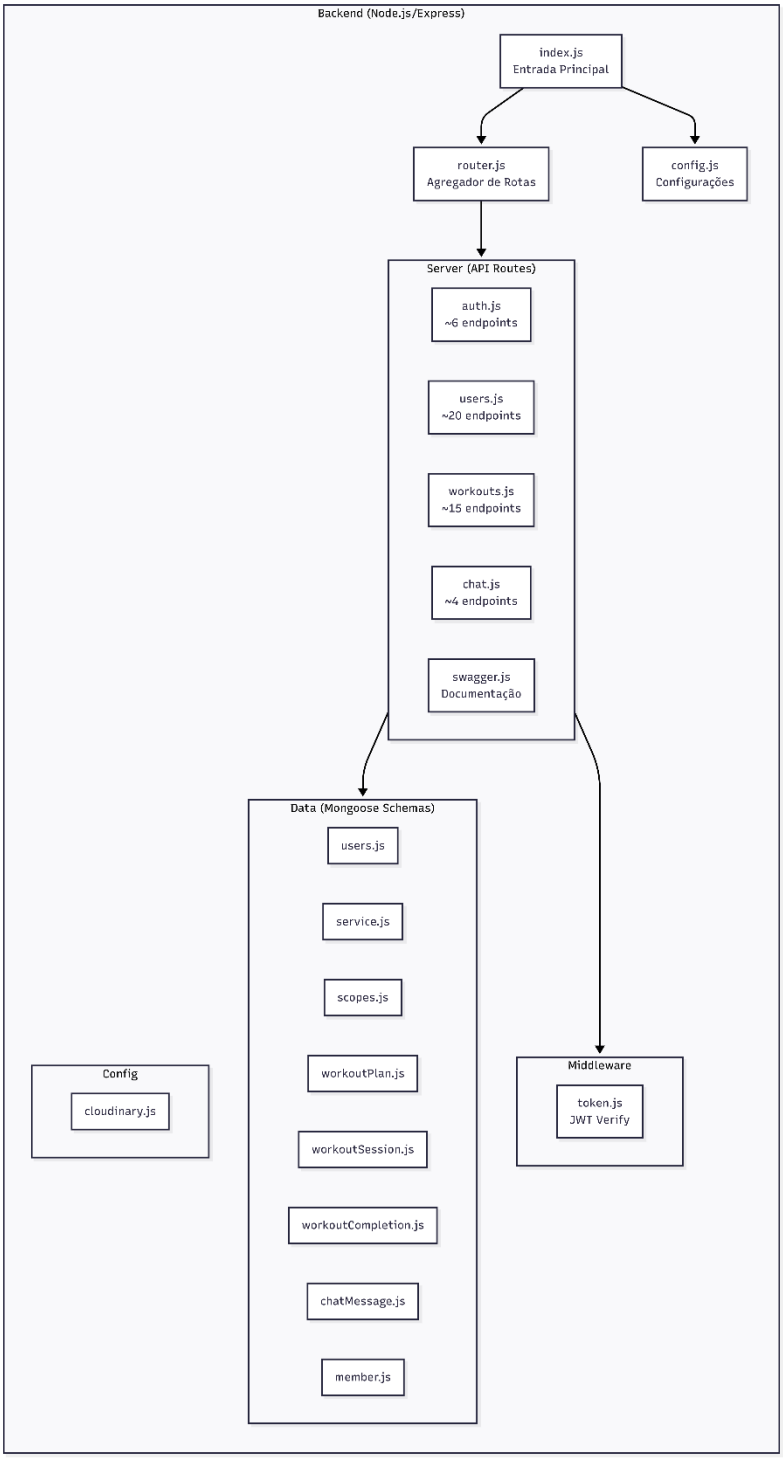


Figura 10-Estrutura do Projeto

5.3.2. Principais Decisões no backend

1. Autenticação JWT com Cookies:

- Token armazenado em cookie httpOnly para segurança
- Suporte a header x-access-token como fallback
- Expiração de 24 horas, extensível para 30 dias com "remember me"

2. Autorização RBAC:

- Scopes: admin, trainer, user
- Middleware authorize([scopes]) em cada rota
- Validação de scope no JWT decoded

3. Reset de Password:

- Token SHA256 com crypto.randomBytes
- Validade de 1 hora
- Email enviado via Nodemailer/Gmail
- Validação de password diferente da atual

4. Upload de Imagens:

- Multer com memoryStorage (não guarda em disco)
- Limite de 2-5MB dependendo do tipo
- Filtro apenas para imagens (mimetype)
- Upload stream para Cloudinary
- Transformações automáticas (crop, quality)

5. Socket.IO Integration:

- Instância io passada às rotas via router.init(io)
- Eventos emitidos: admin_notifications, new-message,
- workout_plan_updated, workout-missed
- CORS configurado para frontend

5.3.3. Endpoints Implementados (45 rotas):

AUTH (/api/auth)

- POST /register - Registo com código convite opcional
- POST /login - Login username/email + password
- POST /forgot-password - Envia email reset
- POST /reset-password/:token - Redefine password
- POST /logout - Limpa cookie
- GET /me - Retorna dados do user autenticado
- POST /login-qr

USERS (/api/users):

- GET /all-users - Lista paginada
- POST /create-user - Cria user (trainer only)
- GET /perfil - Ver próprio perfil
- PUT /perfil - Editar perfil
- PUT /:userId - Editar user (admin)
- DELETE /:userId - Eliminar user (admin)
- POST /perfil/upload-photo - Upload foto
- DELETE /perfil/delete-photo - Remover foto
- PUT /perfil/change-password - Alterar password
- GET /invite-code - Ver código convite
- POST /generate-invite-code - Gerar código (trainer)
- POST /associate-trainer - Associar via código
- POST /disassociation-request - Pedir desassociação
- GET /disassociation-requests/pending - Ver pendentes (admin)
- POST /disassociation-requests/:id/approve - Aprovar
- POST /disassociation-requests/:id/reject - Rejeitar
- GET /disassociation-request/status - Ver status próprio pedido
- GET /details/:id - Detalhes público de user

WORKOUTS (/api/workouts)

- POST /plans - Criar plano
- GET /plans/trainer - Listar do trainer
- GET /plans/my-plan - Plano do cliente

- PUT /plans/:planId - Atualizar plano
- PUT /plans/:planId/activate - Ativar plano
- PUT /plans/:planId/deactivate - Desativar plano
- GET /plans/history/:clientId - Histórico
- DELETE /plans/:planId - Eliminar plano
- POST /sessions - Criar sessão
- GET /sessions/:planId - Listar sessões
- DELETE /sessions/:sessionId - Eliminar sessão
- POST /completions - Registar cumprimento
- GET /completions/client/:clientId - Ver completions
- GET /stats/client/:clientId - Estatísticas
- GET /absences/:clientId - Treinos faltados

CHAT (/api/chat):

- POST /messages - Enviar mensagem
- GET /messages/:userId - Obter conversa
- PUT /messages/mark-read/:userId - Marcar lidas
- GET /contacts - Listar contactos

5.4.FRONTEND (React)

5.4.1. ESTRUTURA DO PROJETO

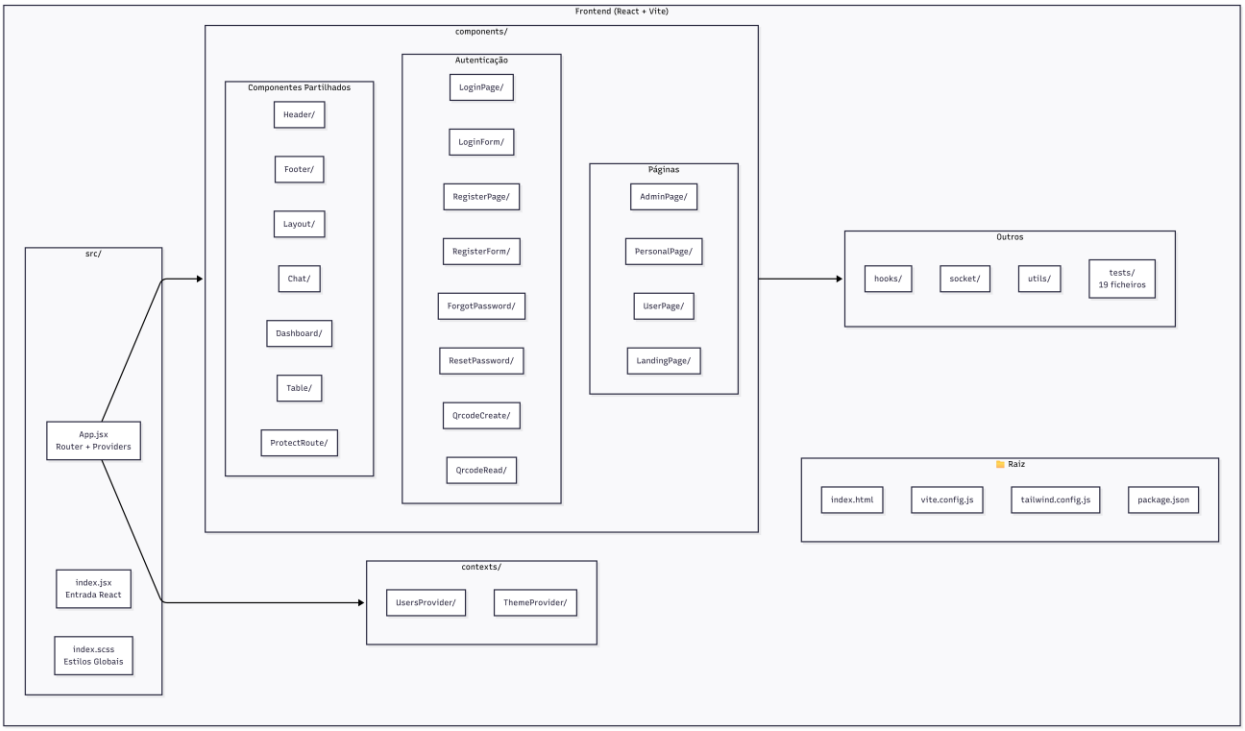


Figura 11-Estrutura do projeto-Frontend

5.4.2. Principais Rotas

- / - Landing Page (pública)
- /login - Página de Login (pública)
- /register - Página de Registo (pública)
- /forgotpassword - Recuperação de Password (pública)

- /reset-password/:token - Redefinir Password (pública)
- /admin - Área do Administrador (protegida - role: admin)
- /trainer - Área do Personal Trainer (protegida - role: trainer)
- /user   - Área do Cliente (protegida - role: user)

5.4.3. Gestão de Estado com Context API

UsersProvider:

- Lista global de utilizadores
- Contador de utilizadores (excluindo admins)
- Funções setUsers para atualização

ThemeProvider:

- Estado do tema (dark/light)
- Toggle de tema
- Persistência em localStorage

5.4.4. Protected Routes

O componente ProtectedRoute:

- Verifica se existe token válido
- Valida o role do utilizador
- Redireciona para / se não autorizado
- Mostra loading enquanto verifica

5.4.5. Intregação Socket.Io

socket.jsx

```
console.log('Socket connecting to:', socketBaseUrl);

const socket = io(socketBaseUrl, {
    withCredentials: true,
    transports: ['websocket', 'polling'],
    reconnection: true,
    reconnectionAttempts: 10,
    reconnectionDelay: 1000,
    reconnectionDelayMax: 5000,
    timeout: 20000,
    autoConnect: true,
});

let initialized = false;

export const initSocket = () => {
    if (initialized) return socket;

    socket.on("connect", () => {
        console.log("Socket connected:", socket.id);
    });

    socket.on("disconnect", (reason) => {
        console.log("Socket disconnected:", reason);
        if (reason === 'io server disconnect') {
            socket.connect();
        }
    });
```

Figura 12-Inicialização do Socket

O módulo socket.jsxconfigura a ligação WebSocket entre o frontend e o backend para comunicação em tempo real.

Funcionalidades:

- Conexão automática ao servidor com autoConnect: true
- Suporte a WebSocket com fallback para HTTP polling
- Reconexão automática até 10 tentativas em caso de falha

- Envio de cookies (JWT) com withCredentials: true
- Gestão de eventos de conexão/desconexão
- Reconexão automática se o servidor desligar a sessão

5.4.6. Design System

- SCSS Modules para isolamento de estilos
- CSS Variables para temas (dark/light)
- TailwindCSS para utility classes
- Responsivo com media queries
- Bootstrap + Reactstrap para componentes base

5.4.7. BASE DE DADOS (MongoDB)

DATABASE: gym (ou gym_test para testes)

CLUSTER: MongoDB Atlas (ou localhost:27017)

COLLECTIONS (7):

1. users

- Utilizadores do sistema (admin, trainer, user)
- Índice único: email, inviteCode (sparse)

2. workoutplans

- Planos de treino
- Referências: trainer → User, client → User

3. workoutsessions

- Sessões de treino (exercícios por dia)
- Referência: workoutPlan → WorkoutPlan
- Validação: máx 10 exercícios, janela 5h

4. workoutcompletions

- Registo de cumprimento
- Índice único composto: workoutSession + client + date
- Referências: workoutSession, client → User

5. chatmessages

- Mensagens de chat
- Índice: sender + receiver + createdAt
- Referências: sender, receiver → User

6. disassociationrequests

- Pedidos de desassociação trainer-cliente
- Estados: pending, approved, rejected
- Referências: user, trainer, resolvedBy → User

5.4.8. Modelo-Entidade-Relacionamento

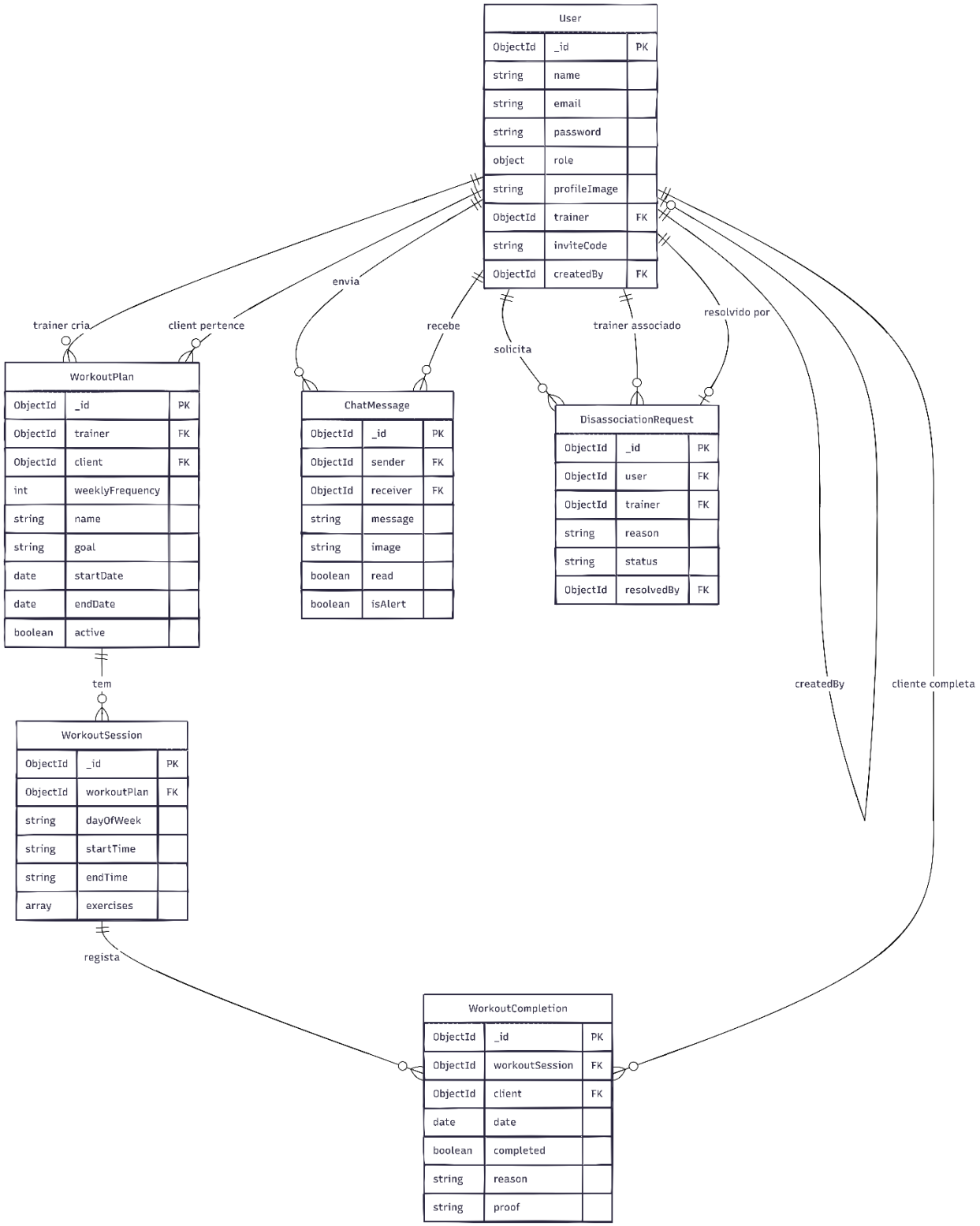


Figura 13-Modelo-Entidade-Relacionamento

6. Melhorias para o Futuro

6.1.Funcionalidades não Implementadas

1. Notificações Push

- Implementar Web Push Notifications
- Notificar clientes sobre novos treinos
- Lembretes automáticos para treinos agendados

2. App Mobile

- Desenvolver versão React Native
- Sincronização offline
- Integração com smartwatches

5. Análise Avançada

- Machine Learning para sugestões
- Previsão de progresso
- Recomendações automáticas

6. Integração com Wearables

- Sync com Fitbit, Garmin, Apple Watch
- Importação automática de dados
- Métricas de frequência cardíaca

8. Gamificação

- Sistema de pontos e badges
- Rankings entre clientes
- Desafios semanais

9. TypeScript

- Migrar todo o código para TypeScript
- Tipos definidos para todas as entidades
- Melhor manutenibilidade

10. PWA Completo

- Service Worker para offline
- Instalação no dispositivo
- Background sync

7. Deployment e Estrutura

7.1.Versão de Produção (Vercel + Render)

COMPONENTE	URL / SERVIÇO
Frontend	https://gym-pwa-three.vercel.app
Backend API	https://gym-5we7.onrender.com
Swagger Docs	https://gym-5we7.onrender.com/api-docs
MongoDB Atlas	cluster0.cahnque.mongodb.net
GitHub-Produção	https://github.com/lou-spec/gym
GitHub-Local	https://github.com/ArpaoCeleste/gym

7.2.Versão Local (Desenvolvimento)

COMPONENTE	URL
Frontend	http://localhost:5173
Backend API	http://localhost:3000
Swagger Docs	http://localhost:3000/api-docs

7.3.Configuração Render (Backend):

- Serviço: Web Service
- Build Command: npm install
- Start Command: npm start
- Environment: Node.js
- Variáveis de ambiente configuradas no Dashboard(.env)

CONFIGURAÇÃO VERCEL (Frontend):

- Framework Preset: Vite
- Build Command: npm run build
- Output Directory: build
- Proxy para API configurado em vite.config.js

7.4.Testes de Software

A garantia de qualidade do software foi assegurada através de diferentes tipos de testes, utilizando ferramentas modernas adequadas a cada camada da aplicação.

7.4.1. Ferramentas Utilizadas

- Backend: Jest e Supertest
- Frontend: Vitest e React Testing Library
- API Manual: Postman

7.4.2. Estratégia de Testes

1. Testes Unitários (Backend):

- Validação de serviços (Users, Workouts)
- Verificação de lógica de negócio isolada
- Mocking de dependências de base de dados

2. Testes de Integração (API):

- Teste de endpoints RESTful com Supertest
- Verificação de códigos de estado HTTP (200, 201, 400, 401, 500)
- Validação de payload de resposta JSON

3. Testes de Componentes (Frontend):

- Renderização correta de componentes React
- Simulação de eventos de utilizador (clicks, inputs)
- Verificação de estados condicionais (ex: loading, error)

4. Testes Manuais:

- Validação de fluxos de utilizador completos
- Teste de responsividade em diferentes viewports
- Verificação de integração com serviços externos (Cloudinary, MongoDB Atlas)

7.4.3. Exemplo de Teste

index.test.jsx

```
describe("LoginPage Integration Tests", () => {
    it("renders the login page correctly", () => {
        renderComponent();

        expect(screen.getByTestId("login-form-mock")).toBeInTheDocument();
        expect(screen.getByText(/Login with QR Code/i)).toBeInTheDocument();
    });

    it("toggles QR code reader", () => {
        renderComponent();

        expect(screen.queryByTestId("qr-reader-mock")).not.toBeInTheDocument();

        const qrButton = screen.getByText(/Login with QR Code/i);
        fireEvent.click(qrButton);

        expect(screen.getByTestId("qr-reader-mock")).toBeInTheDocument();
    });

    it("displays password recovery link", () => {
        renderComponent();

        const forgotLink = screen.getByRole("link", { name: /Esqueci a password/i });
        expect(forgotLink).toBeInTheDocument();
        expect(forgotLink).toHaveAttribute("href", "/forgotpassword");
    });

    it("displays register link", () => {
        renderComponent();

        const registerLink = screen.getByRole("link", { name: /Não tens conta/i });
        expect(registerLink).toBeInTheDocument();
        expect(registerLink).toHaveAttribute("href", "/register");
    });
});
```

Figura 14-Login Test

8. Conclusão

O projeto FitLife foi desenvolvido com sucesso, cumprindo todos os requisitos propostos no enunciado da unidade curricular de Programação Web Avançada.

8.1.Objetivos Alcançados

- API RESTful completa com ~50 endpoints funcionais
- Autenticação segura com JWT, bcrypt e cookies
- Sistema RBAC com 3 perfis (Admin, Trainer, User)
- Comunicação real-time com Socket.IO
- SPA React moderna com routing protegido
- Base de dados MongoDB com 7 collections
- Upload de imagens via Cloudinary
- Testes automatizados com Jest e Vitest
- Documentação Swagger/OpenAPI
- Suporte a tema escuro e claro
- Chat entre trainer e cliente
- Sistema de notificações automáticas
- Gestão completa de planos de treino
- Dashboard com gráficos e estatísticas

8.2.Métricas do Projeto

- Linhas de código Backend: ~4000+
- Linhas de código Frontend: ~6000+
- Endpoints API: ~50
- Componentes React: 20+
- Collections MongoDB: 7
- Ficheiros de testes: 24 (5 backend + 19 frontend)
- Eventos Socket.IO: 4

8.3.Aprendizagens

- Este projeto permitiu consolidar conhecimentos em:
- Arquitetura de aplicações web modernas
- Desenvolvimento de APIs RESTful
- Autenticação e autorização
- Comunicação real-time
- Gestão de estado em React
- Boas práticas de desenvolvimento
- Testes de software

9. Referências

9.1. Documentação Oficial

- Node.js - https://nodejs.org/en/docs/
- Express.js - https://expressjs.com/
- MongoDB/Mongoose - https://mongoosejs.com/docs/
- React - https://react.dev/

9.2.Bibliotecas Frontend

- React Router DOM - https://reactrouter.com/
- React Hook Form - https://react-hook-form.com/
- Recharts (gráficos) - https://recharts.org/
- SweetAlert2 - https://sweetalert2.github.io/
- React Toastify - https://fkhadra.github.io/react-toastify/
- Swiper (carrossel) - https://swiperjs.com/ & https://codepen.io/kristen17/pen/GRXgqaB
- Lucide React (ícones) - https://lucide.dev/
- React QR Code - https://www.npmjs.com/package/react-qr-code
- React QR Scanner - https://www.npmjs.com/package/@yudiel/react-qr-scanner
- Reactstrap/Bootstrap - https://reactstrap.github.io/
- Socket.io Client - https://socket.io/docs/v4/client-api/

9.3. Bibliotecas Backend

- Bcrypt - https://www.npmjs.com/package/bcrypt
- JSON Web Token - https://www.npmjs.com/package/jsonwebtoken
- Multer (upload ficheiros) - https://www.npmjs.com/package/multer
- Nodemailer - https://nodemailer.com/
- Cloudinary - https://cloudinary.com/documentation
- Socket.io - https://socket.io/docs/v4/
- Swagger (jsdoc + ui) - https://swagger.io/docs/
- CORS - https://www.npmjs.com/package/cors
- Cookie Parser - https://www.npmjs.com/package/cookie-parser

9.4. Ferramentas de Desenvolvimento

- Vite - https://vitejs.dev/
- Vitest (testes frontend) - https://vitest.dev/
- Jest (testes backend) - https://jestjs.io/
- SASS - https://sass-lang.com/
- Nodemon - https://nodemon.io/

9.5.Ferramentas de Apoio

- VS Code - Editor de código
- Postman - Testes de API
- Chrome DevTools - Debug e responsividade
- Git/GitHub - Controlo de versões
- Google Antigravity - Editor de código
- Balsamiq – Mockups - https://balsamiq.com/
- Mermaid – Diagramas - https://mermaid.js.org/