

USJT - UC Gestão de Qualidade de Software

Turma: ADS1AN-BUC1-6272430

FitTrack - Projeto A3

RA 823159742 - Igor Cordeiro de Souza Pereira - 823159742@ulife.com.br

RA 823217461 - Lucca Palmieri Dittrich - 823217561@ulife.com.br

RA 823123930 - Eduardo Vieira de Jesus - 823123930@ulife.com.br

RA 82426451 - Eduardo Filipe Silva S. Santos - 82426451@ulife.com.br

São Paulo - SP

2024

Sumário

1. PLANEJAMENTO DE TESTES.....	3
1.1. Cronograma de atividades.....	3
1.2. Alocação de recursos.....	3
1.3. Marcos do projeto.....	4
2. DOCUMENTOS DE DESENVOLVIMENTO.....	4
2.1. Plano de Projeto.....	4
2.1.1. Planejamento do projeto.....	4
2.1.2. Escopo.....	4
2.1.3. Recursos.....	5
2.1.4. Estimativas de projeto.....	5
2.2. Documento de requisitos.....	5
Requisitos Funcionais.....	5
Requisitos Não Funcionais.....	6
2.3. Planejamento de testes.....	6
2.3.1. Plano de testes.....	6
2.3.1.1. Introdução.....	6
2.3.1.2. Escopo.....	7
2.3.1.3. Objetivos.....	7
2.3.1.4. Requisitos a serem testados.....	7
2.3.1.5. Estratégias, tipos de testes e ferramentas a serem utilizadas.....	8
2.3.1.6. Recursos a serem empregados.....	9
2.3.1.7. Cronograma das atividades.....	9
2.3.2. Casos de testes.....	11
2.3.3. Roteiro de testes.....	20
3. GESTÃO DE CONFIGURAÇÃO.....	25
Estrutura de Arquivos.....	25
Componentes Principais.....	26
4. REPOSITÓRIO DE GESTÃO DE CONFIGURAÇÃO.....	26
Controle de Versão.....	26
Padrões de Commits.....	26
Exemplos.....	27
Processo de Deploy.....	28
5. REFERÊNCIAS.....	28

1. PLANEJAMENTO DE TESTES

1.1. Cronograma de atividades

Fase	Atividade	Duração Estimada
Planejamento	Definição do escopo de testes	1 semana
Desenvolvimento	Criação de casos de teste	2 semanas
Execução	Testes unitários	1 semana
Execução	Testes de integração	1 semana
Execução	Testes de componentes	1 semana
Documentação	Relatório de resultados	1 semana

1.2. Alocação de recursos

- 1 Desenvolvedor Full-stack
- 1 QA Engineer
- 1 Testador
- Ambiente de desenvolvimento React
- Firebase(Cloud Firestore, Authentication)
- Ferramentas de teste(Jest, React Testing Library)

1.3. Marcos do projeto

1. MVP(Minimum Viable Product)

- Autenticação de usuários
- CRUD básico de treinos
- Interface responsiva

2. Versão 1.0

- Sistema completo de treinos
- Checkbox das séries
- Timer de descanso
- Gestão de exercícios

2. DOCUMENTOS DE DESENVOLVIMENTO

2.1. Plano de Projeto

2.1.1. Planejamento do projeto

O FitTrack é uma aplicação web desenvolvida em React para gerenciamento de treinos e exercícios físicos. O sistema permite que usuários criem, visualizem, editem e excluam seus treinos, além de controlar o tempo de descanso entre as séries e um aviso sonoro para começar a próxima série.

2.1.2. Escopo

Funcionalidades principais:

- Autenticação de usuários
- Gerenciamento de treinos
- Controle de séries e repetições
- Timer de descanso
- Interface responsiva

2.1.3. Recursos

Frontend:

- React
- React Hook Form
- Firebase SDK
- CSS

Backend:

- Firebase Authentication
- Cloud Firestore

2.1.4. Estimativas de projeto

Tempo de desenvolvimento: 3 meses

Equipe: 4 pessoas

Complexidade: Média

2.2. Documento de requisitos

Requisitos Funcionais

Autenticação

RF1.1: O sistema deve permitir login de usuários

RF1.2: O sistema deve permitir logout de usuários

Gerenciamento de Treinos

RF2.1: O sistema deve permitir Criação de novos treinos

RF2.2: O sistema deve permitir Edição de treinos existentes

RF2.3: O sistema deve permitir Exclusão de treinos existentes

RF2.4: O sistema deve permitir a Visualização de treinos

Gerenciamento de Exercícios

RF3.1: O sistema deve permitir a adição de exercícios ao treino, após dê logado no app.

RF3.2: O sistema deve permitir configurar séries e repetições dos exercícios.

RF3.3: O sistema deve permitir a Definição de tempo de descanso

RF3.4: O sistema deve permitir o Controle de séries realizadas

Requisitos Não Funcionais

Desempenho

RNF1.1: Tempo de resposta das funcionalidades deve ser no máximo de 3 segundos

RNF1.2: O Sistema deve Suportar múltiplos usuários simultâneos

Usabilidade

RNF2.1: O sistema deve ter a Interface responsiva

RNF2.2: O sistema deve apresentar uma interface que facilite a navegação do usuário, com elementos organizados de forma clara, uso consistente de ícones, e orientações visuais.

Segurança

RNF3.1: Autenticação segura via Firebase

RNF3.2: Dados protegidos por token de usuário

2.3. Planejamento de testes

2.3.1. Plano de testes

2.3.1.1. Introdução

O FitTrack é uma aplicação web desenvolvida em React com backend no Firebase, destinada ao gerenciamento de treinos e exercícios. Este documento

detalha o planejamento de testes para garantir a qualidade e confiabilidade do software.

2.3.1.2. Escopo

O plano de testes abrange os seguintes itens:

Interface do Usuário:

Testes de layout e responsividade em diferentes dispositivos.

Navegação entre páginas e modais.

Funcionalidades de Autenticação:

Teste de login e logout.

Validação de segurança no processo de autenticação.

Operações CRUD de Treinos e Exercícios:

Criação, edição, exclusão e visualização de treinos e exercícios.

Controle de séries realizadas e definição de tempo de descanso.

Timer de Descanso:

Funcionamento correto do timer e notificações sonoras.

Integração com Firebase:

Verificação de conexões e operações com Firebase.

Desempenho e Segurança:

Tempo de resposta máximo de 3 segundos.

Proteção de dados via token e validações de segurança.

2.3.1.3. Objetivos

- Garantir o funcionamento correto de todas as funcionalidades
- Validar a integração entre o frontend (React) e o backend (Firebase).
- Identificar e corrigir erros antes do deploy para produção.
- Assegurar uma experiência de usuário fluida e intuitiva, com feedback visual claro.
- Verificar a responsividade e usabilidade em dispositivos desktop, tablet e mobile.
- Certificar que o sistema atende aos requisitos de segurança e desempenho.

2.3.1.4. Requisitos a serem testados

Autenticação

- Login de usuário
- Logout de usuário

Gerenciamento de Treinos

- Criação de treinos
- Edição de treinos
- Exclusão de treinos
- Listagem de treinos

Gerenciamento de Exercícios

- Adição de exercícios
- Edição de exercícios
- Exclusão de exercícios
- Controle de séries
- Timer de descanso

Interface e Usabilidade

- Responsividade
- Navegação entre páginas

- Modais
- Feedback visual

2.3.1.5. Estratégias, tipos de testes e ferramentas a serem utilizadas

Testes Unitários

- Ferramenta: Jest + React Testing Library
- Foco em componentes isolados

Testes de Integração

- Ferramenta: Jest + React Testing Library
- Firebase Emulator
- Foco em fluxos completos

Testes End-to-End

- Ferramenta: Cypress
- Simulação de usuário real
- Fluxos completos

Testes de Responsividade

- Ferramenta: Chrome DevTools
- Dispositivos múltiplos
- Breakpoints principais

2.3.1.6. Recursos a serem empregados

Equipe

- 1 Desenvolvedor Full-stack
- 1 QA Engineer
- 1 Testador

Ambiente de Desenvolvimento

- Node.js
- React Development Tools
- Firebase Emulator Suite

Ferramentas de Teste

- Jest
- React Testing Library
- Cypress
- Firebase Testing Tools

2.3.1.7. Cronograma das atividades

Semana	Atividade	Duração
1	Definição do escopo de testes	2 dias
1	Desenvolvimento de testes unitários	3 dias
2	Testes de integração	3 dias
2	Testes E2E	2 dias
3	Testes de responsividade	2 dias
3	Correções e ajustes	3 dias

2.3.1.8. Definição dos marcos do projeto

Marco 1: Setup (Semana 1)

- Ambiente de testes configurado

- Primeiros testes unitários implementados

Marco 2: Cobertura Básica (Semana 2)

- Testes unitários completos
- Início dos testes de integração

Marco 3: Finalização (Semana 3)

- Todos os testes implementados
- Relatório de cobertura gerado

2.3.2. Casos de testes

Casos de Teste do Login

Caso de teste: 1

Resumo: Exibe tela de login corretamente

Prioridade: Média

Status: Aprovado

Pré-condição: Estar deslogado no sistema

Passos:

1. Verifica se campo de e-mail está presente
2. Verifica se campo de senha está presente
3. Verifica se botão de “Entrar” está presente
4. Verifica se botão de “Cadastro” está presente

Resultado esperado: Todos os campos e botões para login e cadastro devem aparecer

Caso de teste: 2

Resumo: Verificar se loga corretamente

Prioridade: Média

Status: Aprovado

Pré-condição: Estar deslogado no sistema

Passos:

1. Inserir login e senha válidos
2. Clicar no botão “Entrar”

Resultado esperado: Abrir página de treinos

Caso de teste: 3

Resumo: Exibe mensagem de erro ao falhar no login

Prioridade: Média

Status: Aprovado

Pré-condição: Estar deslogado no sistema

Passos:

5. Inserir login ou senha inválidos
6. Clicar no botão "Entrar"

Resultado esperado: Mensagem de erro exibida ao usuário indicando falha de autenticação

Caso de teste: 4

Resumo: Valida os campos do formulário antes do envio

Prioridade: Média

Status: Aprovado

Pré-condição: Estar deslogado no sistema

Passos:

1. Tentar submeter o formulário sem preencher os campos obrigatórios

Resultado esperado: Mostrar mensagens de erro de validação

Caso de teste: 5

Resumo: Abre o modal de registro ao clicar em "Cadastrar"

Prioridade: Baixa

Status: Aprovado

Pré-condição: Estar na tela de login

Passos:

1. Clicar no botão "Cadastrar"

Resultado esperado: Modal de registro é exibido

Caso de teste: 6

Resumo: Fecha o modal de registro ao clicar no botão de fechar

Prioridade: Baixa

Status: Aprovado

Pré-condição: Modal de registro deve estar aberto

Passos:

1. Clicar no botão de fechar no modal

Resultado esperado: Modal de registro é fechado

Código do teste de autenticação(completo no repositório em Referências `src/__tests__`)

```
describe("Login Component", () => {
  // limpa os mocks antes de cada teste
  beforeEach(() => {
    jest.clearAllMocks();
  });

  it("renders the login form correctly", () => {
    render(<Login onLogin={jest.fn()} handleDataRefresh={jest.fn()} />);

    // Verifica se todos os elementos necessários estão presentes
    expect(screen.getByLabelText(/e-mail/i)).toBeInTheDocument();
    expect(screen.getByLabelText(/senha/i)).toBeInTheDocument();
    expect(screen.getByRole("button", { name: /entrar/i })).toBeInTheDocument();
    expect(
      screen.getByRole("button", { name: /cadastrar/i })
    ).toBeInTheDocument();
  });

  it("handles successful login", async () => {
    const mockOnLogin = jest.fn();
    const mockHandleDataRefresh = jest.fn();

    render(
      <Login onLogin={mockOnLogin} handleDataRefresh={mockHandleDataRefresh} />
    );

    // Simula login bem-sucedido
    signInWithEmailAndPassword.mockResolvedValueOnce({
      user: { email: "valid@test.com" },
    });

    // Preenche os campos
    await userEvent.type(screen.getByLabelText(/e-mail/i), "valid@test.com");
    await userEvent.type(screen.getByLabelText(/senha/i), "password123");

    // Clica no botão de login
    await userEvent.click(screen.getByRole("button", { name: /entrar/i }));

    // Verifica se as funções foram chamadas
    await waitFor(() => {
      expect(signInWithEmailAndPassword).toHaveBeenCalledWith(
        expect.anything(),
        "valid@test.com",
        "password123"
      );
      expect(mockOnLogin).toHaveBeenCalled();
      expect(mockHandleDataRefresh).toHaveBeenCalled();
    });
  });

  it("displays an error message if login fails", async () => {
    render(<Login onLogin={jest.fn()} handleDataRefresh={jest.fn()} />);

    // Simula falha no login
    signInWithEmailAndPassword.mockRejectedValueOnce(
      new Error("auth/invalid-credentials")
    );

    // Preenche os campos
    await userEvent.type(screen.getByLabelText(/e-mail/i), "invalid@test.com");
    await userEvent.type(screen.getByLabelText(/senha/i), "wrongpassword");

    // Clica no botão de login
    await userEvent.click(screen.getByRole("button", { name: /entrar/i }));

    // Verifica se a mensagem de erro aparece
    await waitFor(() => {
      expect(screen.getByText(/error/i)).toBeInTheDocument();
    });
  });
});
```

▪
▪
▪

PASS src/pages/Login/__tests__/Login.test.jsx

Login Component

- ✓ renders the login form correctly (110 ms)
- ✓ handles successful login (555 ms)
- ✓ displays an error message if login fails (555 ms)
- ✓ validates form fields before submission (66 ms)
- ✓ opens the register modal when 'Cadastrar' is clicked (72 ms)
- ✓ closes the register modal when close button is clicked (155 ms)

Casos de Teste do Timer

Caso de teste: 7

Resumo: Renderiza corretamente com tempo inicial

Prioridade: Média

Status: Aprovado

Pré-condição: Timer configurado com um valor inicial

Passos:

1. Renderizar o componente Timer

Resultado esperado: Exibe o tempo inicial corretamente

Caso de teste: 8

Resumo: Inicia o timer quando a prop `shouldStart` muda para `true`

Prioridade: Média

Status: Aprovado

Pré-condição: Timer deve estar configurado

Passos:

1. Alterar a prop `shouldStart` para `true`

Resultado esperado: O timer começa a contar

Caso de teste: 9

Resumo: Mostra o botão de pausa quando o timer está rodando

Prioridade: Média

Status: Aprovado

Pré-condição: Timer está rodando

Passos:

1. Verificar o estado do botão quando o timer está ativo

Resultado esperado: Exibe o botão de pausa

Caso de teste: 10

Resumo: Mostra o botão de play quando o timer está pausado

Prioridade: Média

Status: Aprovado

Pré-condição: Timer está pausado

Passos:

1. Verificar o estado do botão quando o timer está inativo

Resultado esperado: Exibe o botão de play

Caso de teste: 11

Resumo: Chama `onExpire` e toca som quando o timer expira

Prioridade: Alta

Status: Aprovado

Pré-condição: Timer deve estar rodando com tempo restante

Passos:

1. Esperar o tempo acabar

Resultado esperado: Executa `onExpire` e toca o som configurado

Caso de teste: 12

Resumo: Reseta o timer ao alterar a prop `key`

Prioridade: Média

Status: Aprovado

Pré-condição: Timer deve estar configurado com uma prop `key`

Passos:

1. Alterar a prop `key`

Resultado esperado: O timer é reiniciado

Código do Timer(completo no repositório em Referências `src/__tests__`)

```
import React from "react";
import { render, screen, fireEvent } from "@testing-library/react";
import Timer from "../components/Timer";
import { useTimer } from "react-timer-hook";

// Mock para react-timer-hook
jest.mock("react-timer-hook", () => ({
  useTimer: jest.fn(),
}));

// Mock para o Audio
window.Audio = jest.fn().mockImplementation(() => ({
  play: jest.fn(),
}));

// Mock para as imagens
jest.mock("/playBtn.svg", () => "play-icon");
jest.mock("/pauseBtn.svg", () => "pause-icon");

describe("Timer Component", () => {
  const mockStart = jest.fn();
  const mockPause = jest.fn();
  const mockRestart = jest.fn();
  const mockOnExpire = jest.fn();
  const expiryTimestamp = new Date();
  expiryTimestamp.setSeconds(expiryTimestamp.getSeconds() + 30); // 30 segundos de timer

  beforeEach(() => {
    jest.clearAllMocks();
    // Configuração padrão do mock do useTimer
    useTimer.mockReturnValue({
      seconds: 30,
      minutes: 0,
      isRunning: false,
      start: mockStart,
      pause: mockPause,
      restart: mockRestart,
    });
  });

  it("renders correctly with initial time", () => {
    useTimer.mockReturnValue({
      seconds: 30,
      minutes: 0,
      isRunning: false,
      start: mockStart,
      pause: mockPause,
      restart: mockRestart,
    });

    render(
      <Timer
        expiryTimestamp={expiryTimestamp}
        autoStart={false}
        shouldStart={false}
        onExpire={mockOnExpire}
      />
    );

    // Verifica se o tempo está formatado corretamente (00:30)
    expect(screen.getByText("00")).toBeInTheDocument();
    expect(screen.getByText("30")).toBeInTheDocument();
  });

  it("starts timer when shouldStart prop changes to true", () => {
    const { rerender } = render(
      <Timer
        expiryTimestamp={expiryTimestamp}
        autoStart={false}
        shouldStart={false}
        onExpire={mockOnExpire}
      />
    );

    // Simula a mudança da prop shouldStart
    rerender(
      <Timer
        expiryTimestamp={expiryTimestamp}
        autoStart={false}
        shouldStart={true}
        onExpire={mockOnExpire}
      />
    );

    expect(mockStart).toHaveBeenCalled();
  });
});
```

-
-
-

```
PASS src/__tests__/Timer.test.jsx
Timer Component
  ✓ renders correctly with initial time (41 ms)
  ✓ starts timer when shouldStart prop changes to true (8 ms)
  ✓ shows pause button when timer is running (53 ms)
  ✓ shows play button when timer is not running (14 ms)
  ✓ calls onExpire and plays sound when timer expires (3 ms)
  ✓ resets timer when key prop changes (5 ms)
```

Casos de Teste do Workout

Caso de teste: 13

Resumo: Renderiza as informações do treino corretamente

Prioridade: Média

Status: Aprovado

Pré-condição: Treinos disponíveis no sistema

Passos:

1. Acessar a página de treino

Resultado esperado: Exibe todas as informações do treino corretamente

Caso de teste: 14

Resumo: Abre o modal de edição ao clicar no botão de adicionar

Prioridade: Média

Status: Aprovado

Pré-condição: Página de treino deve estar aberta

Passos:

1. Clicar no botão de adicionar treino

Resultado esperado: Modal de edição é exibido

Caso de teste: 15

Resumo: Fecha o modal ao clicar no botão cancelar

Prioridade: Média

Status: Aprovado

Pré-condição: Modal de edição deve estar aberto

Passos:

1. Clicar no botão cancelar no modal

Resultado esperado: Modal de edição é fechado

Caso de teste: 16

Resumo: Exclui treino ao clicar no botão excluir

Prioridade: Alta

Status: Aprovado

Pré-condição: Treino deve estar listado

Passos:

1. Clicar no botão de excluir treino

Resultado esperado: Treino é removido da lista

Caso de teste: 17

Resumo: Trata a atualização do treino corretamente

Prioridade: Alta

Status: Aprovado

Pré-condição: Treino deve estar listado

Passos:

1. Editar os detalhes de um treino

Resultado esperado: Atualização salva e exibida corretamente

Caso de teste: 18

Resumo: Trata erro ao excluir treino

Prioridade: Média

Status: Aprovado

Pré-condição: Ocorre um erro na API ou backend

Passos:

1. Tentar excluir um treino inválido

Resultado esperado: Mensagem de erro é exibida ao usuário

Código do teste do Treino(completo no repositório em Referências `src/__tests__`)

```
import { render, screen, fireEvent, waitFor } from "@testing-library/react";
import "testing-library/jest-dom";
import Workout from "../pages/WorkoutPage/Workout";
import { deleteDoc, doc } from "../_mocks_/firebase/firestore";

// Mock the components and modules
jest.mock("../components/AddButton", () => {
  return function DummyAddButton({ widthAndHeight }) {
    return <div data-testid="add-button">Add Button</div>;
  };
});

jest.mock("../components/Modal", () => {
  return function DummyModal({ children, show, handleModal }) {
    if (!show) return null;
    return (
      <div data-testid="modal" onClick={handleModal}>
        {children}
      </div>
    );
  };
});

// Mock Exercise component
jest.mock("../pages/WorkoutPage/Exercise", () => {
  return function DummyExercise(props) {
    return <div data-testid="exercise">{props.name}</div>;
  };
});

// Mock EditExercisesForm component
jest.mock("../pages/WorkoutPage/EditExercisesForm", () => {
  return function DummyEditExercisesForm({ workout, onSubmit, onCancel }) {
    return (
      <div data-testid="edit-form">
        <button onClick={() => onSubmit(...workout)}>Submit</button>
        <button onClick={onCancel}>Cancel</button>
      </div>
    );
  };
});

// Mock Firebase
jest.mock("../_mocks_/firebase/firestore", () => ({
  deleteDoc: jest.fn(),
  doc: jest.fn(),
}));

jest.mock("../_mocks_/firebase/config", () => ({
  db: {
    collection: jest.fn(),
    doc: jest.fn(),
  },
}));

describe("Workout Component", () => {
  const mockProps = {
    workoutName: "Push Day",
    id: "123",
    workoutNumber: 1,
    exercises: [
      {
        name: "Bench Press",
        sets: 3,
        reps: 12,
        weight: 100,
      },
    ],
    onWorkoutDataUpdate: jest.fn(),
    onWorkoutUpdate: jest.fn(),
    onWorkoutDelete: jest.fn(),
    onExerciseDelete: jest.fn(),
    user: { uid: "user123" },
  };

  let consoleSpy;

  beforeEach(() => {
    consoleSpy = jest.spyOn(console, "error").mockImplementation(() => {});
  });

  afterEach(() => {
    consoleSpy.mockRestore();
  });

  beforeEach(() => {
    jest.clearAllMocks();
    consoleSpy.mockClear();
  });

  it("renders workout information correctly", () => {
    render(<Workout {...mockProps} />);

    expect(
      screen.getByText(
        `Dis ${mockProps.workoutNumber}: ${mockProps.workoutName.toUpperCase()}`
      )
    ).toBeInTheDocument();
    expect(screen.getByText("adicionar novo exercicio")).toBeInTheDocument();
  });
});
```

-
-
-

```
PASS src/__tests__/Workout.test.jsx
Workout Component
  ✓ renders workout information correctly (42 ms)
  ✓ opens edit modal when add button is clicked (15 ms)
  ✓ closes modal when cancel is clicked (9 ms)
  ✓ deletes workout when delete button is clicked (119 ms)
  ✓ handles workout update correctly (8 ms)
  ✓ handles error when deleting workout (84 ms)
```

2.3.3. Roteiro de testes

Preparação do Ambiente

1. Configuração Inicial

```
npm i --save-dev jest @testing-library/react @testing-library/jest-dom
@testing-library/user-event jest-environment-jsdom babel-jest identity-obj-proxy
@babel/preset-env @babel/preset-react
```

2. Setup do Firebase Emulator

```
firebase init emulators
```

Execução dos Testes

1. Testes Unitários

```
npm run test:watch
```

2. Testes de Integração

```
npm run test:integration
```

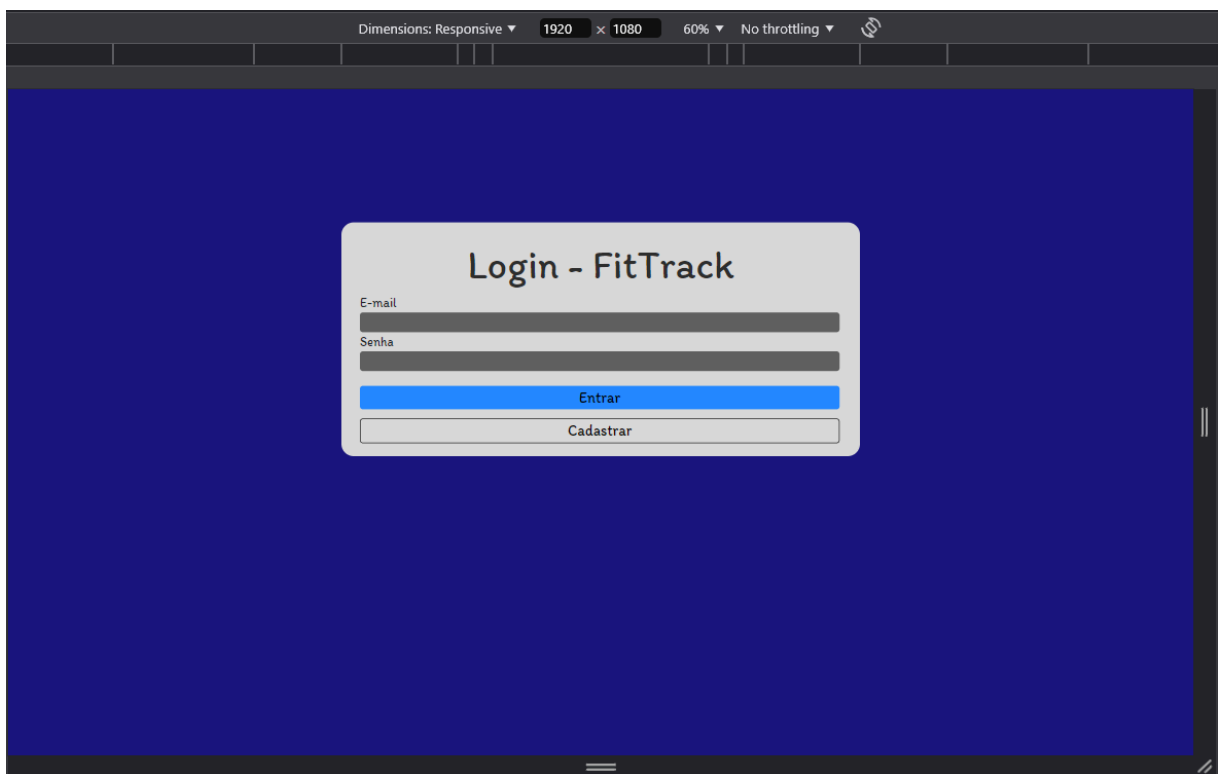
3. Testes E2E

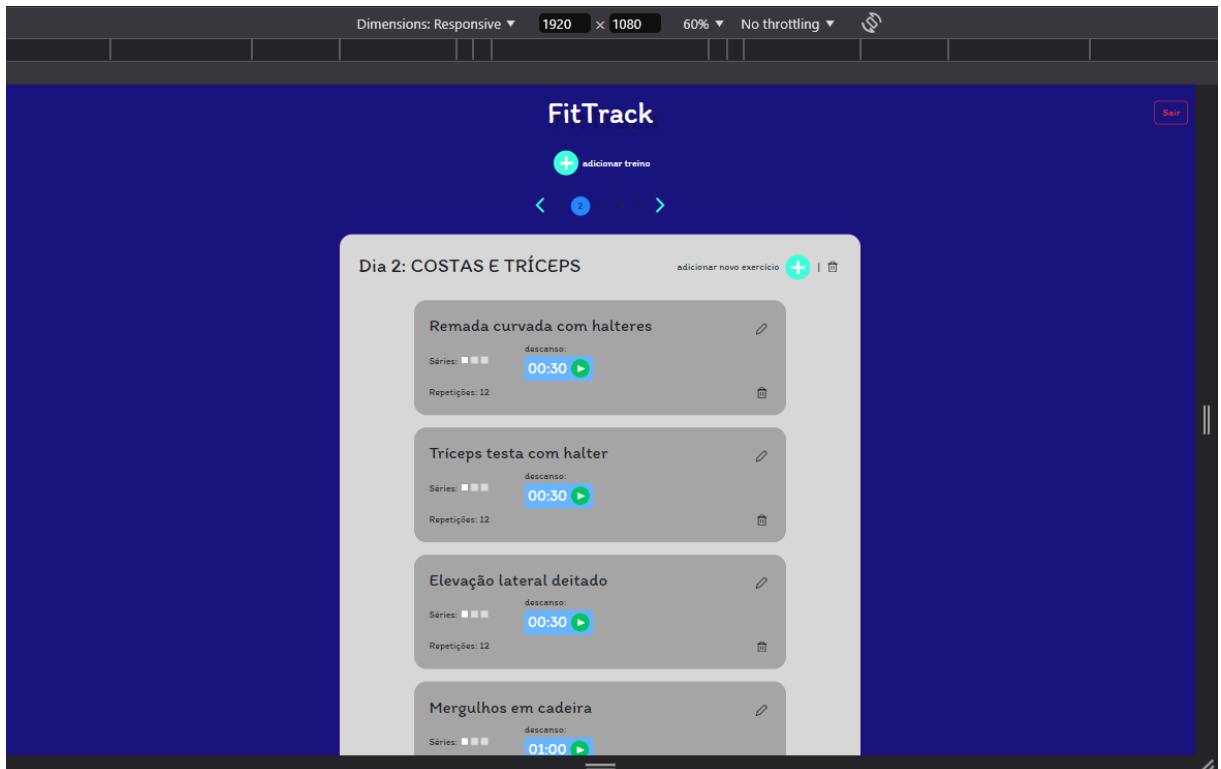
```
npm run test:e2e
```

Testes Manuais

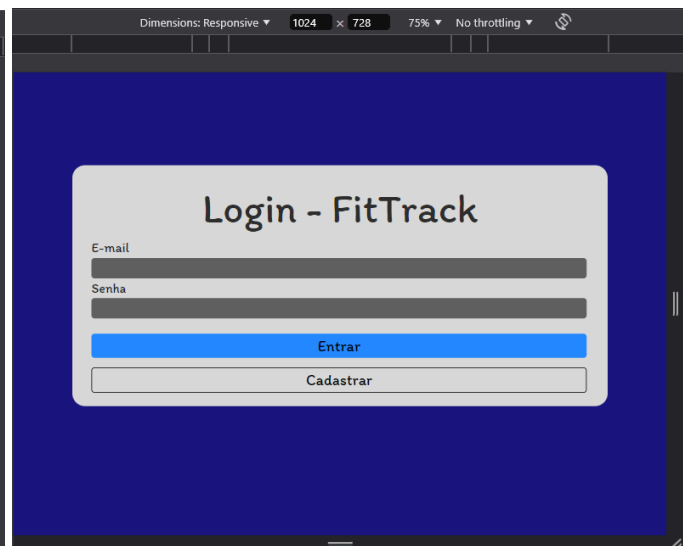
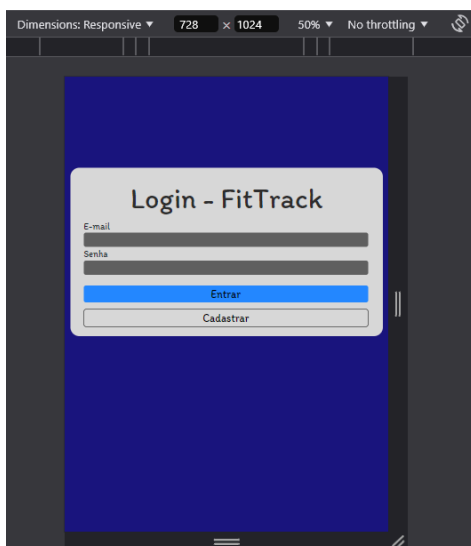
Responsividade

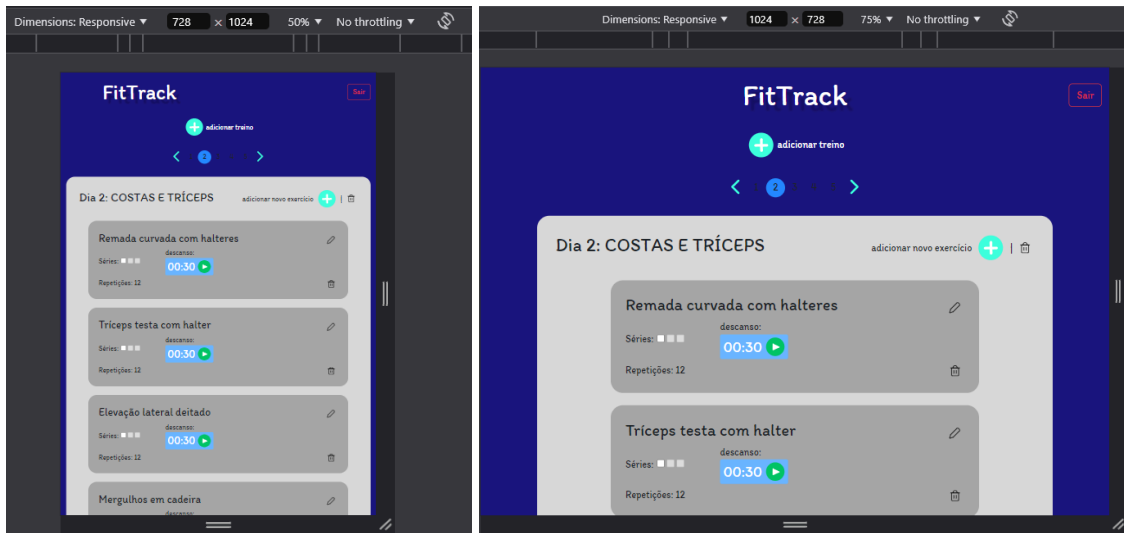
- Desktop (1920x1080)



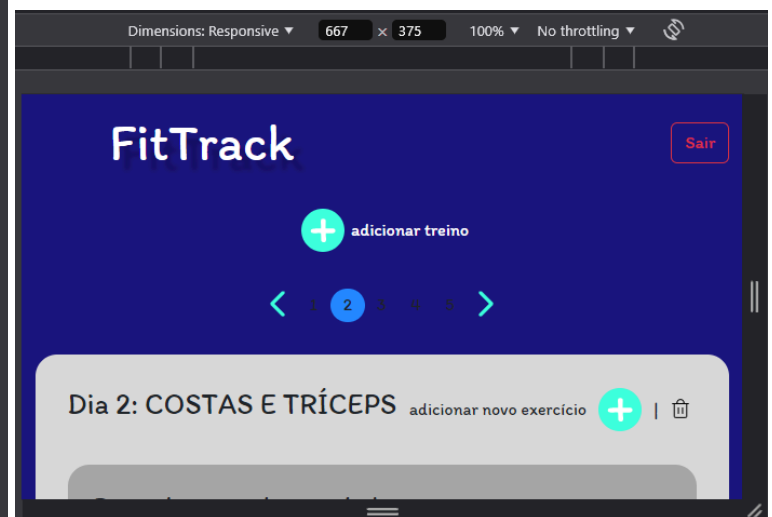
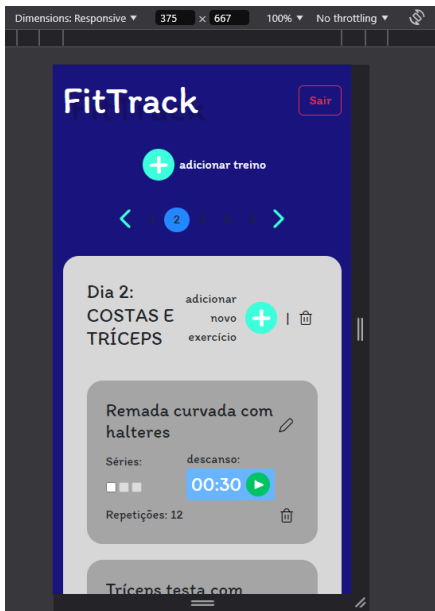
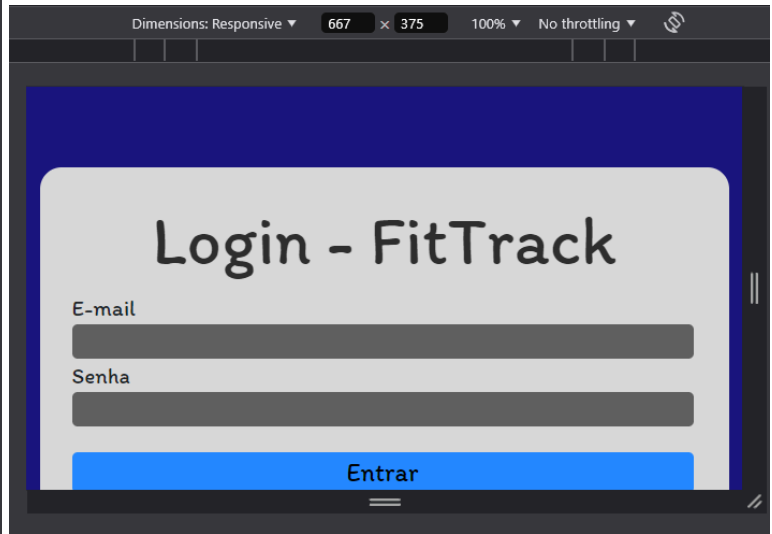
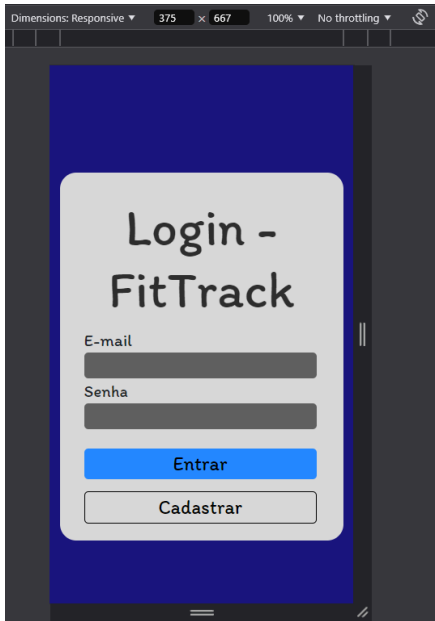


- Tablet (768x1024)






- Mobile (375x667)



Usabilidade

- Navegação intuitiva
- Feedback visual adequado
- Mensagens de erro claras

<p>E-mail</p> <input type="text"/> <p>Campo obrigatório</p> <p>Senha</p> <input type="password"/> <p>Campo obrigatório</p>	<p>E-mail</p> <input type="text" value="test@teste.com"/> <p>Senha</p> <input type="password" value="....."/> <p>Error: <i>Firebase: Error (auth/invalid-credential).</i></p>
--	---



×

Treino de...

Este campo é obrigatório

Nome do exercício

Este campo é obrigatório

Séries


Este campo é obrigatório. Máx: 10

Repetições

Este campo é obrigatório.

Tempo de descanso(s)

Este campo é obrigatório. Máx: 300s


 Adicionar mais exercícios

Enviar

3. GESTÃO DE CONFIGURAÇÃO

Estrutura de Arquivos

```
src/  
├── __mocks__/  
│   ├── firebase/  
│   │   └── auth.js  
│   ├── workout.js  
│   ├── timer.js  
│   ├── filemock.js  
│   └── firebaseConfig.js  
├── __tests__/  
│   ├── Login.test.jsx  
│   ├── Timer.test.jsx  
│   └── Workout.test.jsx  
├── assets/  
│   ├── *.svg  
│   └── *.wav  
├── components/  
│   ├── AddButton.jsx  
│   ├── CircularIndeterminate.jsx  
│   ├── Modal.jsx  
│   ├── Pagination.jsx  
│   ├── RemoveButton.jsx  
│   └── Timer.jsx  
├── pages/  
│   ├── Login/  
│   │   ├── RegisterForm.jsx  
│   │   └── Login.jsx  
│   └── WorkoutPage/  
│       ├── EditExercisesForm.jsx  
│       ├── Exercise.jsx  
│       ├── Workout.jsx  
│       └── WorkoutForm.jsx  
├── App.jsx  
├── AuthContext.jsx  
├── firebaseConfig.js  
├── setupTests.js  
├── index.css  
└── main.jsx
```

Componentes Principais

App.jsx

- Componente principal
- Gerencia estado global
- Controle de autenticação

Workout.jsx

- Gerencia treinos individuais
- Controle de exercícios
- Operações CRUD

Exercise.jsx

- Controle de séries
- Timer de descanso
- Status de exercício

4. REPOSITÓRIO DE GESTÃO DE CONFIGURAÇÃO

Controle de Versão

- Sistema: Git
- Repositório: GitHub
- Branches principais:
 - main: produção
 - develop: desenvolvimento
 - feature/*: novas funcionalidades

Padrões de Commits

- **feat** - Commits do tipo feat indicam que seu trecho de código está incluindo um novo recurso (se relaciona com o MINOR do versionamento semântico).
- **fix** - Commits do tipo fix indicam que seu trecho de código commitado está solucionando um problema (bug fix), (se relaciona com o PATCH do versionamento semântico).
- **docs** - Commits do tipo docs indicam que houveram mudanças na documentação, como por exemplo no Readme do seu repositório. (Não inclui alterações em código).

- **test** - Commits do tipo test são utilizados quando são realizadas alterações em testes, seja criando, alterando ou excluindo testes unitários. (Não inclui alterações em código)
- **build** - Commits do tipo build são utilizados quando são realizadas modificações em arquivos de build e dependências.
- **perf** - Commits do tipo perf servem para identificar quaisquer alterações de código que estejam relacionadas a performance.
- **style** - Commits do tipo style indicam que houveram alterações referentes a formatações do código, semicolons, trailing spaces, lint, etc (Não inclui alterações em código).
- **refactor** - Commits do tipo refactor referem-se a mudanças devido a refatorações que não alterem sua funcionalidade, como por exemplo, uma alteração no formato como é processada determinada parte da tela, mas que manteve a mesma funcionalidade, ou melhorias de performance devido a um code review.
- **chore** - Commits do tipo chore indicam atualizações de tarefas de build, configurações de administrador, pacotes. Como por exemplo adicionar um pacote no gitignore. (Não inclui alterações em código)
- **ci** - Commits do tipo ci indicam mudanças relacionadas a integração contínua (continuous integration).
- **raw** - Commits do tipo raw indicam mudanças relacionadas a arquivos de configurações, dados, features, parâmetros.
- **cleanup** - Commits do tipo cleanup são utilizados para remover código comentado, trechos desnecessários ou qualquer outra forma de limpeza do código-fonte, visando aprimorar sua legibilidade e manutenibilidade.
- **remove** - Commits do tipo remove indicam a exclusão de arquivos, diretórios ou funcionalidades obsoletas ou não utilizadas, reduzindo o tamanho e a complexidade do projeto e mantendo-o mais organizado.

Exemplos

git commit -m 'style: nova cor de fundo da página principal'

git merge feature-new-bg-color (*develop* <- *feature-new-bg-color*)

git merge develop (*main* <- *develop*)

Processo de Deploy

1. Desenvolvimento local
2. Testes automatizados
3. Code review
4. Merge para develop
5. Testes de integração
6. Deploy para produção

5. REFERÊNCIAS

Sites:

<https://github.com/iuricode/padroes-de-commits>

<https://jestjs.io/pt-BR/docs/getting-started>

<https://testing-library.com/docs/react-testing-library/intro/>

<https://docs.cypress.io/app/get-started/why-cypress>

Bibliografia:

[PRESSMAN, R. S. Software Engineering: A Practitioner's Approach. 7. ed. New York: McGraw-Hill, 2010.](#)

Aplicação:

<https://fittrack-project.vercel.app/>

Usuário para testes: *teste@teste.com* | senha: *senhaTeste*

Repositório:

<https://github.com/igorcsp/exercigor/tree/fittrack>