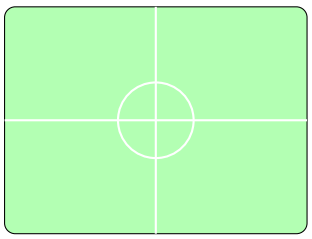
Universidade Federal da Bahia

Instituto de Computação

engenharia de software I

Documentação Técnica
Soccer Quiz

Sistema de Quiz de Futebol
Arquitetura: Monólito Modular com DDD e BFF Pattern



Desenvolvido por:

Equipe Miracle Generation

Dezembro de 2025

Conteúdo

1 Introdução

1.1 Objetivo do Documento

Este documento apresenta a documentação técnica completa do sistema **Soccer Quiz**, uma aplicação web desenvolvida para testar conhecimentos sobre futebol através de quizzes interativos. O documento detalha a arquitetura do sistema, a aplicação dos princípios SOLID, o fluxo de dados, os 14 requisitos funcionais implementados e instruções para execução do projeto.

1.2 Escopo do Sistema

O Soccer Quiz é uma aplicação de quiz de futebol que permite:

- Cadastro, autenticação e gerenciamento de usuários
- Criação e gerenciamento de quizzes, perguntas, respostas e times
- Disputa de quizzes com temporizador
- Visualização de ranking global e identificação do jogador mais rápido
- Sistema de convites e notificações por e-mail

1.3 Tecnologias Utilizadas

Camada	**Tecnologias**
Frontend	React 18, TypeScript, Tailwind CSS, Vite (PWA)
Backend (BFF)	Hono (Edge Functions), Supabase Functions
Autenticação	Supabase Auth (OAuth2/JWT)
Persistência	KV Store (Key-Value)
E-mail	EmailJS (SMTP transacional)
UI Components	Radix UI, Lucide Icons

Tabela 1: Stack Tecnológica do Projeto

2 Arquitetura do Sistema

2.1 Estilo Arquitetural: Monólito Modular com DDD

O sistema adota uma arquitetura de **Monólito Modular** combinada com **Domain-Driven Design (DDD)** e o padrão **Backend for Frontend (BFF)**. Esta escolha foi motivada por:

- **Simplicidade Operacional**: Facilita deployment e manutenção para equipes reduzidas
- **Modularização Interna**: Mantém separação clara de responsabilidades
- **Evolução Gradual**: Permite migração futura para microserviços se necessário
- **Baixa Latência**: Reduz complexidade de rede entre módulos

2.2 Padrão BFF (Backend for Frontend)

O **Operador** atua como um BFF, sendo uma camada intermediária entre o Frontend (PWA) e os serviços de dados. Suas responsabilidades incluem:

- Agregação de dados de múltiplas fontes
- Formatação de respostas otimizadas para o frontend
- Orquestração de chamadas entre módulos
- Implementação de regras de negócio

2.3 Diagrama de Arquitetura

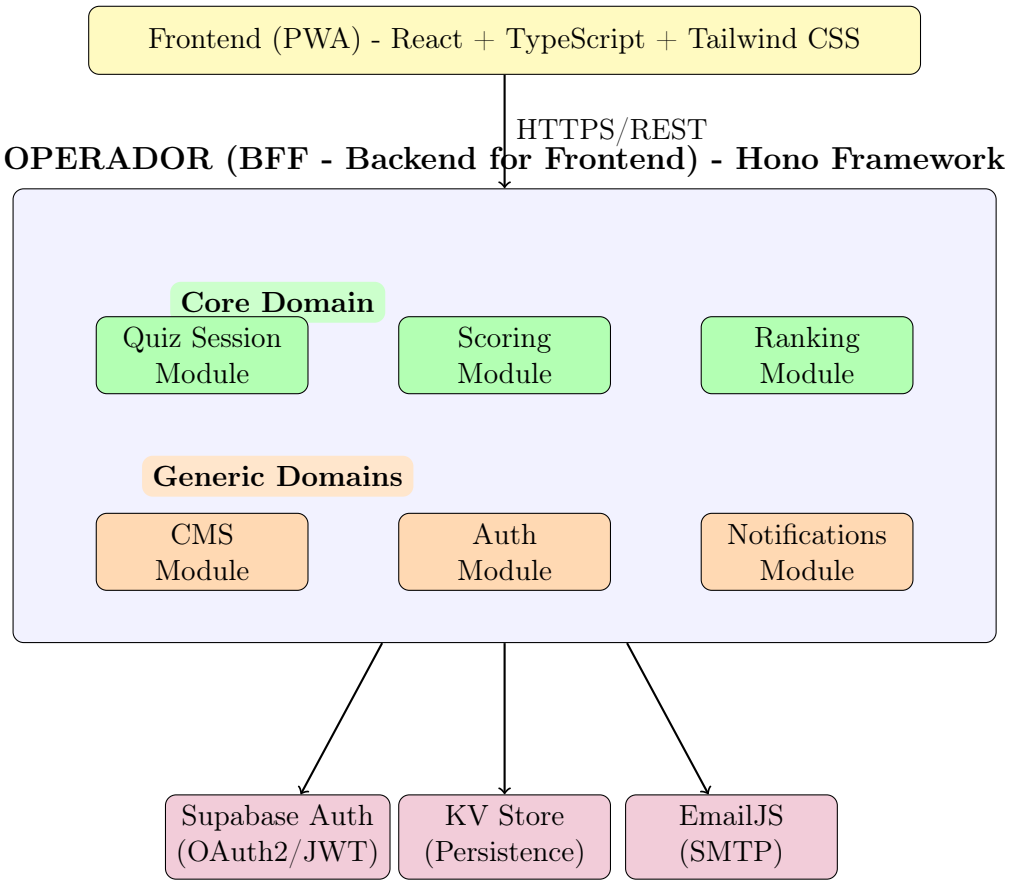


Figura 1: Diagrama de Arquitetura - Monólito Modular com BFF

2.4 Características Arquiteturais

Característica	Implementação
Segurança	Autenticação JWT via Supabase Auth, RBAC (Role-Based Access Control)
Disponibilidade	Edge Functions com auto-scaling, Health Check endpoints
Elasticidade	Cache de ranking com TTL de 30 segundos
Observabilidade	Endpoints /health e /metrics, logs estruturados

Tabela 2: Características Arquiteturais Implementadas

2.5 Estrutura de Diretórios

```
Soccer-Quiz/
|-- src/
|   |-- App.tsx                 # Componente principal (Router)
|   |-- components/
|   |   |-- Login.tsx           # REQ 09 - Logar no Sistema
|   |   |-- SignUp.tsx          # REQ 01 - Manter Usuario
|   |   |-- ResetPassword.tsx   # REQ 02 - Recuperar Senha
|   |   |-- Home.tsx            # Dashboard, REQ 10, REQ 11
|   |   |-- Quiz.tsx            # REQ 06 - Disputar Quiz
|   |   |-- Ranking.tsx         # REQ 08, REQ 14, REQ 13
|   |   +-- AdminPanel.tsx      # REQ 03, 04, 05, REQ 17
|   |-- contexts/
|   |   +-- AuthContext.tsx     # Gerenciamento de estado global
|   |-- hooks/
|   |   |-- useQuiz.ts          # Logica de sessao de quiz
|   |   +-- useAdminQuestions.ts
|   |-- services/               # Camada de servicos (API)
|   |   |-- authService.ts
|   |   |-- quizService.ts
|   |   |-- invitesService.ts   # REQ 11 - Convidar Usuario
|   |   +-- notificationsService.ts # REQ 17
+-- supabase/functions/make-server-4d5764ce/
    |-- index.ts                # Router BFF (API First)
    |-- auth-module.tsx         # SRP: Autenticacao
    |-- cms-module.tsx          # SRP: Gestao de Conteudo
    |-- quiz-session-module.tsx # SRP: Sessao de Quiz
    |-- scoring-module.tsx      # SRP: Pontuacao
    |-- ranking-module.tsx      # SRP: Ranking
    |-- kv_store.tsx            # Camada de persistencia
    +-- types.tsx               # Tipos compartilhados (ISP)
```

Listing 1: Estrutura de Diretórios do Projeto

3 Aplicação dos Princípios SOLID

O projeto Soccer Quiz foi desenvolvido seguindo os princípios SOLID de design orientado a objetos. A seguir, detalhamos onde cada princípio foi aplicado no código.

3.1 S - Single Responsibility Principle (SRP)

"Uma classe deve ter apenas uma razão para mudar."

3.1.1 Aplicação no Backend

Cada módulo do backend tem uma única responsabilidade bem definida:

```
// auth-module.tsx - APENAS autenticacao e identidade
export class AuthModule {
  async signUp(email, password, name, role) { /* ... */ }
  async getAuthenticatedUser(authHeader) { /* ... */ }
  async getUserProfile(userId) { /* ... */ }
}

// scoring-module.tsx - APENAS calculo de pontuacao
export class ScoringModule {
  async finishQuiz(userId) { /* ... */ }
  async updateUserScore(userId, score) { /* ... */ }
  async getUserStats(userId) { /* ... */ }
}

// ranking-module.tsx - APENAS gerenciamento de ranking
export class RankingModule {
  async getGlobalRanking() { /* ... */ }
  async invalidateCache() { /* ... */ }
  async getTopPlayers(limit) { /* ... */ }
}

// cms-module.tsx - APENAS gestao de conteudo
export class CMSModule {
  async createQuestion(data) { /* ... */ }
  async createQuiz(data) { /* ... */ }
  async createTeam(data) { /* ... */ }
}
```

Listing 2: SRP - Módulos com Responsabilidade Única

3.1.2 Aplicação no Frontend

Os serviços são separados por domínio:

```
// authService.ts - Apenas operacoes de autenticacao
export const authService = {
  signIn, signUp, signOut, resetPassword, getUserProfile
};

// quizService.ts - Apenas operacoes de sessao de quiz
export const quizService = {
  startQuiz, getCurrentQuestion, answerQuestion, finishQuiz
};

// invitesService.ts - Apenas operacoes de convite
export const invitesService = {
  sendInvite, listInvites
};
```

Listing 3: SRP - Services Layer no Frontend

3.2 O - Open/Closed Principle (OCP)

"Entidades devem estar abertas para extensão, mas fechadas para modificação."

3.2.1 Aplicação: Sistema de Módulos Extensível

O router principal (index.ts) permite adicionar novos módulos sem modificar o código existente:

```
// index.ts - Novos modulos podem ser adicionados sem modificar existentes
import { AuthModule } from "./auth-module.tsx";
import { CMSModule } from "./cms-module.tsx";
import { QuizSessionModule } from "./quiz-session-module.tsx";
// Para adicionar novo modulo (ex: PaymentModule):
// import { PaymentModule } from "./payment-module.tsx";

const authModule = new AuthModule();
const cmsModule = new CMSModule();
const quizSessionModule = new QuizSessionModule();
// const paymentModule = new PaymentModule(); // Extensao sem modificacao

// Cada modulo registra suas proprias rotas
app.post("/auth/signup", (c) => authModule.handleSignUp(c));
app.post("/quiz/start", (c) => quizSessionModule.handleStart(c));
// app.post("/payment/process", (c) => paymentModule.handle(c)); // Nova rota
```

Listing 4: OCP - Router Extensível

3.2.2 Aplicação: Componentes React Extensíveis

Os componentes de UI são extensíveis via props:

```
// Home.tsx - Extensivel via callbacks sem modificacao interna
interface HomeProps {
  onStartQuiz: (quizId: string) => void;
  onRanking: () => void;
  onAdmin: () => void;
  // Novas funcionalidades podem ser adicionadas via props
  // onPayment?: () => void;
}

export function Home({ onStartQuiz, onRanking, onAdmin }: HomeProps) {
  // Componente fechado para modificacao, aberto para extensao
}
```

Listing 5: OCP - Componentes Extensíveis

3.3 L - Liskov Substitution Principle (LSP)

"Subtipos devem ser substituíveis por seus tipos base."

3.3.1 Aplicação: Interfaces Consistentes

Os módulos implementam contratos consistentes que podem ser substituídos:

```
// types.tsx - Interfaces que garantem substituibilidade
export interface Question {
  id: string;
  question: string;
  options: string[];
  correctAnswer: string;
  team: string;
}

// Qualquer implementacao que siga a interface pode ser usada
// O QuizSessionModule aceita qualquer Question valida
async startSession(userId: string, quizId: string): Promise<
    SessionResult> {
  const questions: Question[] = await this.loadQuestions(quizId);
  // questions podem vir de diferentes fontes (KV, SQL, API externa)
  // desde que sigam a interface Question
}
```

Listing 6: LSP - Contratos Substituíveis

3.3.2 Aplicação: Services Intercambiáveis

Os serviços seguem contratos que permitem substituição:

```
// api.ts - Funcoes utilitarias que funcionam com qualquer endpoint
export const getHeaders = (token: string) => ({
  'Authorization': `Bearer ${token}`,
  'Content-Type': 'application/json'
});

export const handleResponse = async (response: Response) => {
  const data = await response.json();
  if (!response.ok) throw new Error(data.error);
  return data;
};

// Qualquer service pode usar estas funcoes - LSP garantido
// authService, quizService, rankingService - todos substituiveis
```

Listing 7: LSP - Services Intercambiáveis

3.4 I - Interface Segregation Principle (ISP)

"Clientes não devem ser forçados a depender de interfaces que não utilizam."

3.4.1 Aplicação: Interfaces Segregadas por Domínio

O arquivo types.tsx define interfaces específicas para cada contexto:

```
// types.tsx - Interfaces especificas, nao interfaces "gordas"

// Interface para autenticacao - apenas o necessario
export interface User {
```

```
  id: string;
  email: string;
  name: string;
  role: 'player' | 'admin';
}

// Interface para perfil - dados adicionais quando necessario
export interface UserProfile {
  id: string;
  email: string;
  name: string;
  role: string;
  totalScore: number;    // Apenas para contexto de jogo
  gamesPlayed: number;   // Apenas para contexto de jogo
}

// Interface para ranking - apenas dados de exibicao
export interface RankingEntry {
  position: number;
  userId: string;
  name: string;
  totalScore: number;
  gamesPlayed: number;
  average: string;
}

// Interface para sessao - apenas dados de gameplay
export interface QuizSession {
  sessionId: string;
  userId: string;
  quizId: string;
  currentQuestion: number;
  score: number;
  status: 'active' | 'completed';
}
```

Listing 8: ISP - Interfaces Segregadas

3.4.2 Aplicação: Props Específicas nos Componentes

```
// Quiz.tsx - Recebe apenas o que precisa
interface QuizProps {
  quizId: string;
  onBack: () => void;
}

// Ranking.tsx - Props diferentes para contexto diferente
interface RankingProps {
  onBack: () => void;
}

// AdminPanel.tsx - Props especificas para admin
interface AdminPanelProps {
  onBack: () => void;
}

// Cada componente depende apenas das props que usa
```

Listing 9: ISP - Props Específicas

3.5 D - Dependency Inversion Principle (DIP)

"Dependa de abstrações, não de implementações concretas."

3.5.1 Aplicação: Context API como Abstração

O AuthContext abstrai os detalhes de implementação da autenticação:

```
// AuthContext.tsx - Abstrai implementacao de autenticacao
interface AuthContextType {
  user: User | null;
  accessToken: string | null;
  loading: boolean;
  signIn: (email: string, password: string) => Promise<void>;
  signUp: (email: string, password: string, name: string) => Promise<
   void>;
  signOut: () => Promise<void>;
}

// Componentes dependem da ABSTRACAO (interface), nao da implementacao
export function Login() {
  const { signIn } = useAuth(); // Depende da abstracao
  // Nao importa se signIn usa Supabase, Firebase, ou Auth0
  // O componente depende apenas do contrato
}
```

Listing 10: DIP - Context como Abstração

3.5.2 Aplicação: Injeção de Dependência nos Módulos

```
// auth-module.tsx - Dependencias injetadas via construtor
export class AuthModule {
  private supabase;

  constructor() {
    // Dependencia injetada via variaveis de ambiente
    // Pode ser substituida por mock em testes
    this.supabase = createClient(
      Deno.env.get('SUPABASE_URL') || '',
      Deno.env.get('SUPABASE_SERVICE_ROLE_KEY') || ''
    );
  }
}

// kv_store.tsx - Abstracao da camada de persistencia
// Modulos dependem de kv.get/kv.set, nao da implementacao
import * as kv from "./kv_store.tsx";

// QuizSessionModule depende da abstracao kv, nao do Deno.Kv diretamente
await kv.set(sessionId, JSON.stringify(sessionData));
const data = await kv.get(sessionId);
```

Listing 11: DIP - Injeção de Dependência

3.6 Resumo da Aplicação SOLID

Princípio	Sigla	Onde foi aplicado
Single Responsibility	S	Módulos separados: AuthModule, CMSModule, ScoringModule, RankingModule
Open/Closed	O	Router extensível, componentes com props callbacks
Liskov Substitution	L	Interfaces consistentes em types.tsx, services intercambiáveis
Interface Segregation	I	Interfaces específicas: User, UserProfile, RankingEntry, QuizSession
Dependency Inversion	D	AuthContext, kv_store abstraction, injeção via ambiente

Tabela 3: Resumo da Aplicação dos Princípios SOLID

4 Requisitos Funcionais Implementados

O sistema implementa todos os 14 requisitos funcionais especificados. A seguir, detalhamos cada requisito com sua justificativa arquitetural e implementação.

4.1 Requisitos de Autenticação e Usuário

4.1.1 REQ 01 - Manter Usuário (CRUD)

Definição: O sistema deve permitir a manutenção de usuários (CRUD).

Justificativa Arquitetural: É o pré-requisito funcional para o REQ 09 (Logar). Sem criar o registro no banco de dados, o componente de Autenticação não tem o que validar. Necessário para vincular pontuações a uma entidade persistente.

Implementação:

```
// SignUp.tsx - Criacao de usuario
const handleSubmit = async (e: React.FormEvent) => {
  await signUp(email, password, name, role);
};

// Home.tsx - Atualizacao de perfil
const handleUpdateProfile = async () => {
  await updateProfile({ name: nameDraft });
};

// Home.tsx - Exclusao de conta
const handleDeleteAccount = async () => {
  await deleteProfile();
};

// auth-module.tsx - Backend CRUD
app.post("/auth/signup", ...);     // Create
app.get("/user/profile", ...);     // Read
app.put("/user/profile", ...);     // Update
app.delete("/user/profile", ...);  // Delete
```

Listing 12: REQ 01 - CRUD de Usuário

4.1.2 REQ 02 - Recuperar Senha

Definição: O sistema deve permitir que o usuário possa recuperar a senha de acesso.

Justificativa Arquitetural: Valida a implementação de fluxo de e-mail transacional e reset de token, demonstrando domínio sobre "Canais de Comunicação" na arquitetura.

Implementação:

```
// Login.tsx - Solicitar recuperacao
const handleResetPassword = async () => {
  await resetPassword(email);
  setResetMessage('Enviamos um link de recuperacao para seu email.');
};

// ResetPassword.tsx - Definir nova senha
const handleSubmit = async (e: React.FormEvent) => {
  await updatePassword(password);
};

// authService.ts - Integracao com Supabase
async resetPassword(email: string) {
  const redirectTo = `${window.location.origin}/#type=recovery`;
  await supabase.auth.resetPasswordForEmail(email, { redirectTo });
}
```

Listing 13: REQ 02 - Recuperação de Senha

4.1.3 REQ 09 - Logar no Sistema

Definição: O sistema deve permitir que o usuário efetue log-in.

Justificativa Arquitetural: Valida o Componente de Auth. A arquitetura define Segurança como característica principal. Sem autenticação, o Operador não consegue distinguir entre Jogador e Administrador.

Implementação:

```
// Login.tsx
const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  try {
    await signIn(email, password);
  } catch (err: any) {
    setError(err.message || 'Erro ao fazer login');
  }
};

// AuthContext.tsx - Gerenciamento de sessao JWT
const signIn = async (email: string, password: string) => {
  const data = await authService.signIn(email, password);
  if (data.session?.access_token) {
    setAccessToken(data.session.access_token);
    await fetchUserProfile(data.session.access_token);
  }
};
```

Listing 14: REQ 09 - Login

4.1.4 REQ 10 - Sair do Sistema

Definição: O sistema deve permitir que o usuário efetue log-out.

Justificativa Arquitetural: Encerra o ciclo de vida do token JWT no lado do cliente (PWA), garantindo a segurança básica exigida no REQ 09.

Implementação:

```
// Home.tsx - Botao de logout
<button onClick={handleLogout} title="Sair">
  <LogOut className="w-6 h-6 text-white" />
</button>

const handleLogout = async () => {
  await signOut();
};

// AuthContext.tsx
const signOut = async () => {
  await authService.signOut();
  setAccessToken(null);
  setUser(null);
};
```

Listing 15: REQ 10 - Logout

4.2 Requisitos de Gestão de Conteúdo (CMS)

4.2.1 REQ 03 - Cadastrar Time

Definição: O sistema deve permitir o cadastro de times pelo usuário administrador.

Justificativa Arquitetural: No modelo de dados, "Time" é a entidade categorizadora das "Perguntas". Sem times, o filtro de quiz por time (REQ 06) não pode ser implementado.

Implementação:

```
// AdminPanel.tsx - Formulario de time
const handleCreateTeam = async (e: React.FormEvent) => {
  await teamService.create(accessToken, {
    name: teamName,
    description: teamDescription
  });
  await loadTeams();
};

// cms-module.tsx - Backend
async createTeam(name: string, description: string, createdBy: string):
   Promise<Team> {
  const teamId = `team:${Date.now()}`;
  const team: Team = { id: teamId, name, description, createdBy,
   createdAt: new Date().toISOString() };
  await kv.set(teamId, JSON.stringify(team));
  return team;
}
```

Listing 16: REQ 03 - Cadastrar Time

4.2.2 REQ 04 - Cadastrar Pergunta

Definição: O sistema deve permitir o cadastro de perguntas pelo usuário administrador.

Justificativa Arquitetural: Valida o CMS como componente responsável pela "Gestão de conteúdo". Para que o Operador funcione, ele precisa consumir dados escritos pelo CMS.

Implementação:

```
// AdminPanel.tsx
const handleSubmit = async (e: React.FormEvent) => {
  const options = [option1, option2, option3, option4];

  if (!options.includes(correctAnswer)) {
    setError('A resposta correta deve ser uma das opcoes');
    return;
  }

  await createQuestion({
    question, options, correctAnswer,
    team: 'general', quizId: selectedQuizId
  });
};

// cms-module.tsx
async createQuestion(data: CreateQuestionDTO): Promise<Question> {
  const questionId = `question:${Date.now()}`;
  const question: Question = {
    id: questionId,
    question: data.question,
    options: data.options,
    correctAnswer: data.correctAnswer,
    team: data.team,
    createdAt: new Date().toISOString()
  };
  await kv.set(questionId, JSON.stringify(question));
  // Associa ao quiz
  await this.addQuestionToQuiz(data.quizId, questionId);
  return question;
}
```

Listing 17: REQ 04 - Cadastrar Pergunta

4.2.3 REQ 05 - Cadastrar Resposta

Definição: O sistema deve permitir o cadastro de respostas dado uma questão definida.

Justificativa Arquitetural: Uma "Pergunta" sem "Respostas" associadas é um objeto incompleto. Este requisito fecha o grafo de objetos necessário para o Operador montar uma partida válida.

Implementação:

```
// AdminPanel.tsx - Respostas sao cadastradas junto com a pergunta
<input placeholder="Opcao 1" value={option1} onChange={(e) => setOption1
    (e.target.value)} />
<input placeholder="Opcao 2" value={option2} onChange={(e) => setOption2
    (e.target.value)} />
<input placeholder="Opcao 3" value={option3} onChange={(e) => setOption3
    (e.target.value)} />
```

```
<input placeholder="Opcao 4" value={option4} onChange={(e) => setOption4
    (e.target.value)} />

<select value={correctAnswer} onChange={(e) => setCorrectAnswer(e.target
    .value)}>
  <option value="">Selecione a resposta correta</option>
  <option value={option1}>{option1}</option>
  <option value={option2}>{option2}</option>
  <option value={option3}>{option3}</option>
  <option value={option4}>{option4}</option>
</select>

// Modelo de dados - Question inclui opcoes e resposta correta
interface Question {
  id: string;
  question: string;
  options: string[];         // REQ 05 - Respostas
  correctAnswer: string;     // REQ 05 - Resposta correta
  team: string;
}
```

Listing 18: REQ 05 - Cadastrar Resposta (Integrado à Pergunta)

4.3 Requisitos de Gameplay (Core Domain)

4.3.1 REQ 06 - Disputar Quiz

Definição: O sistema deve permitir a disputa do Quiz pelo usuário (responder perguntas de um time ou gerais).

Justificativa Arquitetural: Este é o Core Domain do sistema. A ADR-002 (Arquitetura DDD) instrui o foco no domínio principal. Testa Disponibilidade e Elasticidade, pois é onde ocorre maior volume de acessos.

Implementação:

```
// Quiz.tsx - Interface de jogo
export function Quiz({ quizId, onBack }: QuizProps) {
  const { state, handleAnswerSelect } = useQuiz(quizId, handleFinish);

  return (
    <div>
      <h2>{state.currentQuestion.question}</h2>
      {state.currentQuestion.options.map((option) => (
        <button onClick={() => handleAnswerSelect(option)}>
          {option}
        </button>
      ))}
    </div>
  );
}

// quiz-session-module.tsx - Logica de sessao
async startSession(userId: string, quizId: string, team?: string) {
  const quiz = await kv.get(quizId);
  const questions = await kv.mget(quiz.questionIds);

  // Randomizacao (Fisher-Yates)
```

```
  const shuffledQuestions = this.shuffleArray(parsedQuestions);

  const session: QuizSession = {
    sessionId: `session:${userId}:${Date.now()}`,
    questions: shuffledQuestions,
    currentQuestion: 0,
    score: 0,
    status: 'active'
  };

  await kv.set(session.sessionId, JSON.stringify(session));
  return session;
}
```

Listing 19: REQ 06 - Disputar Quiz

4.3.2 REQ 07 - Encerrar Quiz

Definição: O sistema deve permitir o encerramento do quiz e o cálculo da pontuação.

Justificativa Arquitetural: Valida os módulos críticos do Operador: "Pontuações" e "Fechamento de Quiz". Garante consistência na transição de "jogando" para "finalizado".

Implementação:

```
// useQuiz.ts - Finalizacao no frontend
if (!data.hasMoreQuestions) {
  const finishData = await quizService.finishQuiz(accessToken);
  await refreshProfile();
  alert(`Quiz finalizado!\nPontuacao: ${finishData.finalScore}`);
  onFinish();
}

// scoring-module.tsx - Calculo de pontuacao
async finishQuiz(userId: string): Promise<FinishResult> {
  const session = await this.getActiveSession(userId);

  session.status = 'completed';
  session.completedAt = new Date().toISOString();

  // Atualiza perfil do usuario
  await this.updateUserScore(userId, session.score);

  // Invalida cache do ranking
  await rankingModule.invalidateCache();

  return {
    finalScore: session.score,
    correctAnswers: session.answers.filter(a => a.correct).length,
    totalQuestions: session.questions.length
  };
}
```

Listing 20: REQ 07 - Encerrar Quiz

4.4 Requisitos de Ranking e Gamificação

4.4.1 REQ 08 - Visualizar Ranking

Definição: O sistema deve permitir a visualização do ranking do Quiz pelo usuário.

Justificativa Arquitetural: Fecha o ciclo de experiência do usuário no Frontend. Valida a estratégia de Elasticidade e uso de Cache mencionada na arquitetura.

Implementação:

```
// Ranking.tsx
const fetchRanking = async () => {
  const response = await fetch(`${API_BASE_URL}/ranking`, {
    headers: { 'Authorization': `Bearer ${accessToken}` }
  });
  const data = await response.json();
  setRanking(data.ranking);
};

// ranking-module.tsx - Cache de 30 segundos
async getGlobalRanking(): Promise<RankingEntry[]> {
  const cached = await this.getRankingFromCache();
  if (cached) return cached; // Cache hit - performance

  const ranking = await this.calculateRanking();
  await this.cacheRanking(ranking);
  return ranking;
}
```

Listing 21: REQ 08 - Visualizar Ranking

4.4.2 REQ 13 - Identificar Jogador Mais Rápido

Definição: Identificar o jogador que finalizou com maior pontuação em menor tempo.

Justificativa Arquitetural: Adiciona lógica de negócio no backend (cálculo de delta time entre start_quiz e end_quiz), enriquecendo a regra de negócio.

Implementação:

```
// Ranking.tsx - Exibicao do mais rapido
const fetchFastest = async (quizId: string) => {
  const data = await quizService.getFastest(accessToken, quizId);
  setFastest(data.fastest);
};

// scoring-module.tsx - Registro do mais rapido
private async trackFastestPlayer(session: QuizSession): Promise<void> {
  const durationMs = new Date(session.completedAt).getTime()
                   - new Date(session.startedAt).getTime();

  const entry: FastestEntry = {
    userId: session.userId,
    name: profile.name,
    quizId: session.quizId,
    durationMs,
    score: session.score
  };

  const currentFastest = await kv.get(`fastest:${session.quizId}`);
```

```
  if (!currentFastest || durationMs < currentFastest.durationMs) {
    await kv.set(`fastest:${session.quizId}`, JSON.stringify(entry));
  }
}
```

Listing 22: REQ 13 - Jogador Mais Rápido

4.4.3 REQ 14 - Criar Ranking Geral de Jogadores

Definição: O sistema deve disponibilizar um ranking geral de jogadores acumulado.

Justificativa Arquitetural: Valida capacidade do banco de dados de realizar consultas de agregação (SUM, AVG) históricas.

Implementação:

```
// ranking-module.tsx
private async calculateRanking(): Promise<RankingEntry[]> {
  // Busca TODOS os perfis (ranking acumulado historico)
  const allProfiles = await kv.getByPrefix('user_profile:');

  const profiles = allProfiles
    .map(item => JSON.parse(item))
    .filter(profile => profile.gamesPlayed > 0);

  // Ordenacao por pontuacao total (acumulada)
  profiles.sort((a, b) => b.totalScore - a.totalScore);

  return profiles.map((profile, index) => ({
    position: index + 1,
    name: profile.name,
    totalScore: profile.totalScore,      // Soma historica
    gamesPlayed: profile.gamesPlayed,    // Total de partidas
    average: (profile.totalScore / profile.gamesPlayed).toFixed(1)
  }));
}
```

Listing 23: REQ 14 - Ranking Geral Acumulado

4.5 Requisitos de Comunicação e Engajamento

4.5.1 REQ 11 - Convidar Usuário

Definição: O sistema deve permitir convidar outros usuários por e-mail dentro da solução.

Justificativa Arquitetural: Valida integração com serviços de SMTP (envio de e-mail), demonstrando competência técnica comum em aplicações reais.

Implementação:

```
// Home.tsx - Formulario de convite
const handleSendInvite = async () => {
  await emailjs.send('service_id', 'template_id', {
    to_email: inviteEmail,
    from_name: user?.name,
    message: 'Venha jogar o Quiz de Futebol!'
  });
  setInviteMessage('Convite enviado com sucesso!');
};
```

```
10
11 // Inicializacao do EmailJS
12 import emailjs from '@emailjs/browser';
13 emailjs.init('3zNcnv-oQk4PUhkQN'); // Public Key
```

Listing 24: REQ 11 - Convidar Usuário por E-mail

4.5.2 REQ 17 - Notificar Usuário sobre Novo Quiz

Definição: O sistema deve notificar os usuários quando um quiz for ser iniciado.

Justificativa Arquitetural: Introduz conceito de eventos ou disparos assíncronos na arquitetura. Justifica comunicação proativa do servidor para o cliente.

Implementação:

```
1  // Home.tsx - Carregamento de notificacoes
2  useEffect(() => {
3    const loadNotifications = async () => {
4      const data = await notificationsService.list(accessToken);
5      setNotifications(data.notifications || []);
6    };
7    loadNotifications();
8  }, [accessToken]);
9
10 // cms-module.tsx - Broadcast de notificacao ao criar quiz
11 async createQuiz(name: string, ...): Promise<Quiz> {
12   const quiz = await this.saveQuiz(quizData);
13
14   // REQ 17 - Notifica usuarios sobre novo quiz
15   await this.broadcastNotification({
16     type: 'new_quiz',
17     title: 'Novo Quiz Disponivel!',
18     message: `O quiz "${name}" foi criado. Venha jogar!`
19   });
20
21   return quiz;
22 }
```

Listing 25: REQ 17 - Notificações

4.6 Matriz de Rastreabilidade

REQ	Nome	Componente Frontend	Módulo Backend
REQ 01	Manter Usuário	SignUp.tsx, Home.tsx	AuthModule
REQ 02	Recuperar Senha	Login.tsx, ResetPassword.tsx	AuthModule (Supabase)
REQ 03	Cadastrar Time	AdminPanel.tsx	CMSModule
REQ 04	Cadastrar Pergunta	AdminPanel.tsx	CMSModule
REQ 05	Cadastrar Resposta	AdminPanel.tsx	CMSModule
REQ 06	Disputar Quiz	Quiz.tsx, useQuiz.ts	QuizSessionModule
REQ 07	Encerrar Quiz	useQuiz.ts	ScoringModule
REQ 08	Visualizar Ranking	Ranking.tsx	RankingModule

REQ	Nome	Componente Frontend	Módulo Backend
REQ 09	Logar no Sistema	Login.tsx	AuthModule
REQ 10	Sair do Sistema	Home.tsx	AuthModule
REQ 11	Convidar Usuário	Home.tsx	EmailJS (externo)
REQ 13	Jogador Mais Rápido	Ranking.tsx	ScoringModule
REQ 14	Ranking Geral	Ranking.tsx	RankingModule
REQ 17	Notificar Novo Quiz	Home.tsx	CMSModule

Tabela 4: Matriz de Rastreabilidade - Requisitos x Componentes

5 Fluxo de Dados

5.1 Fluxo de Autenticação (REQ 01, 09, 10)

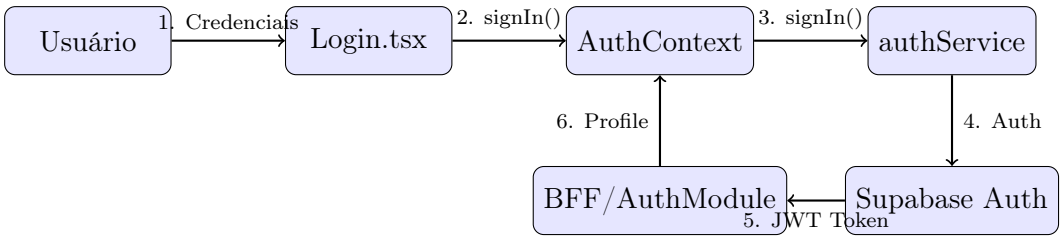


Figura 2: Fluxo de Autenticação

5.2 Fluxo de Quiz (REQ 06, 07)

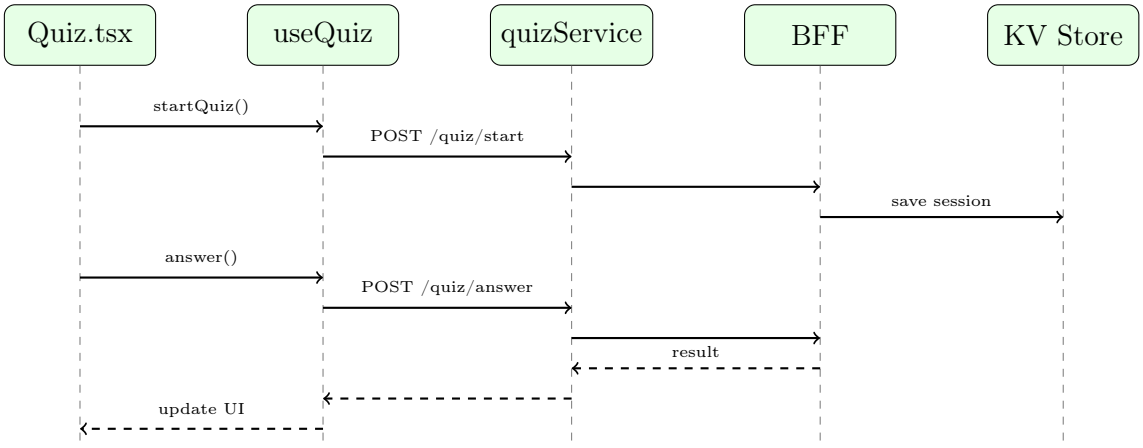


Figura 3: Fluxo de Disputa de Quiz

6 Como Rodar o Projeto

6.1 Pré-requisitos

- **Node.js** versão 18 ou superior
- **npm** ou **yarn**
- **Git**
- Conta no **Supabase** (para backend)

6.2 Instalação

```
# 1. Clonar o repositorio
git clone https://github.com/PauloFilho1/Soccer_Quiz_Miracle_Generation.
    git
cd Soccer-Quiz

# 2. Instalar dependencias
npm install

# 3. Configurar variaveis de ambiente
# Criar arquivo .env.local com:
VITE_SUPABASE_URL=sua_url_do_supabase
VITE_SUPABASE_ANON_KEY=sua_chave_anonima
```

Listing 26: Passos de Instalação

6.3 Executando o Projeto

```
# Modo desenvolvimento (hot-reload)
npm run dev
# Acesse: http://localhost:5173

# Build para producao
npm run build
# Arquivos gerados em: dist/
```

Listing 27: Comandos de Execução

6.4 Checklist de Validação dos Requisitos

1. **REQ 01, 09**: Cadastre uma conta e faça login
2. **REQ 02**: Teste "Esqueci minha senha" na tela de login
3. **REQ 03, 04, 05**: Como admin, crie times, quizzes e perguntas
4. **REQ 06, 07**: Jogue um quiz completo
5. **REQ 08, 14**: Verifique o ranking após jogar
6. **REQ 10**: Faça logout

7. **REQ 11**: Convide um amigo por e-mail
8. **REQ 13**: Verifique o "Jogador Mais Rápido" no ranking
9. **REQ 17**: Crie um quiz e verifique as notificações

7 Conclusão

O Soccer Quiz demonstra a aplicação prática de conceitos avançados de Engenharia de Software:

- **Arquitetura**: Monólito Modular com DDD e padrão BFF
- **Princípios SOLID**: Aplicados em todas as camadas do sistema
- **14 Requisitos Funcionais**: Todos implementados e rastreáveis
- **Características de Qualidade**: Segurança (JWT), Elasticidade (Cache), Observabilidade (Health/Metrics)

A estrutura modular e a aplicação consistente dos princípios SOLID garantem que o sistema seja manutenível, extensível e pronto para evoluir conforme novas necessidades surjam.

8 Referências

1. Martin, Robert C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.
2. Evans, Eric. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2003.
3. React Documentation. Disponível em: https://react.dev
4. Supabase Documentation. Disponível em: https://supabase.com/docs
5. Hono Documentation. Disponível em: https://hono.dev
6. Martin, Robert C. *SOLID Principles of Object-Oriented Design*. Disponível em: https://blog.cleancoder.com