Location404 - Jogo de Geolocalização Multijogador em Tempo Real

Ryan Gabriel Mazzei Bromati

Location404

Jogo de Geolocalização Multijogador em Tempo Real

Ryan Gabriel Mazzei Bromati

Engenharia de Software

2024-2025

1 Resumo

Este documento apresenta a especificação técnica do Location404, um jogo de geolocalização multijogador em tempo real inspirado no conceito do GeoGuessr. O projeto adota uma arquitetura de microsserviços com C# .NET 9+ e Vue 3, garantindo escalabilidade, alta disponibilidade e separação clara de responsabilidades. O Location404 proporcionará uma experiência imersiva, permitindo que os usuários testem seus conhecimentos geográficos em diversos cenários e compitam com outros jogadores, sustentado por uma infraestrutura robusta de serviços distribuídos.

1. Introdução

1.1. Contexto

O mercado de jogos geográficos online tem crescido significativamente nos últimos anos, com plataformas como GeoGuessr ganhando popularidade tanto para entretenimento quanto para fins educacionais. Nesse contexto, o Location404 surge como uma alternativa moderna que busca explorar as mais recentes tecnologias de desenvolvimento web e arquiteturas distribuídas para oferecer uma experiência superior ao usuário.

1.2. Justificativa

O projeto Location404 é relevante para o campo da engenharia de software pois implementa conceitos avançados de arquitetura distribuída em um cenário prático e atrativo. A adoção de microsserviços em um jogo de geolocalização permite explorar desafios reais de desenvolvimento de software, como latência em comunicações distribuídas, persistência de dados, escalabilidade em momentos de pico de uso e segurança em um ambiente altamente interativo. Além disso, o projeto demonstra a aplicação prática de padrões arquiteturais modernos e tecnologias de ponta, incluindo conceitos de DevOps, observabilidade e resiliência de sistemas.

1.3. Objetivos

Objetivo Principal: Desenvolver uma plataforma de jogo de geolocalização completa, baseada em microsserviços, que ofereça desempenho superior, alta disponibilidade e uma experiência de usuário envolvente.

Objetivos Secundários:

- Implementar uma arquitetura de microsserviços escalável e resiliente.
- Criar uma interface de usuário responsiva e intuitiva com Vue 3.
- Desenvolver um sistema de autenticação seguro e eficiente.
- Implementar um sistema preciso de geolocalização e cálculo de pontuação.
- Criar uma infraestrutura de dados que permita expansão futura do jogo.
- Desenvolver um sistema básico de competição e ranking entre jogadores.
- Garantir observabilidade completa do sistema com monitoramento e logging.
- Implementar práticas de DevOps com CI/CD automatizado.

2. Descrição do Projeto

2.1. Tema do Projeto

O Location404 é uma plataforma de jogo online que desafia os usuários a identificar localizações geográficas com base em imagens de satélite, fotos panorâmicas de ruas e outros recursos visuais. Utilizando dados geográficos precisos, o jogo transporta os jogadores para diversos locais ao redor do mundo, testando seu conhecimento em geografia, culturas, paisagens urbanas e naturais.

O projeto adota uma arquitetura de microsserviços para garantir escalabilidade e manutenibilidade, utilizando C# .NET 9+ para o backend e Vue 3 para o frontend. Cada componente do sistema é isolado em seu próprio serviço, comunicando-se através de interfaces bem definidas, facilitando a evolução independente e a resiliência do sistema.

3.2 2.2. Problemas a Resolver

1. **Latência e Responsividade**: Garantir tempos de resposta rápidos mesmo com processamento distribuído entre microsserviços.
2. **Integridade de Dados**: Manter a consistência dos dados do jogo e perfis de usuário entre serviços distribuídos.
3. **Segurança**: Proteger dados sensíveis dos usuários e evitar trapaças no sistema de jogo.
4. **Disponibilidade**: Garantir que o sistema permaneça operacional mesmo quando partes específicas estejam em manutenção.
5. **Experiência do Usuário**: Oferecer uma interface fluida e intuitiva apesar da complexidade da infraestrutura.
6. **Integração de Dados Geográficos**: Incorporar fontes de dados geográficos precisos e atualizados.
7. **Resiliência**: Implementar tolerância a falhas e recuperação automática de serviços.
8. **Observabilidade**: Garantir visibilidade completa do comportamento do sistema em produção.

3.3 2.3. Limitações

- O projeto não abordará a criação ou captura de imagens geográficas próprias, utilizando integrações com APIs de terceiros (Google Street View API).
- O sistema será inicialmente otimizado apenas para plataforma web (desktop e móvel), sem aplicativos nativos para dispositivos móveis.
- A cobertura geográfica inicial poderá ser limitada com base na disponibilidade de dados de qualidade das APIs utilizadas.
- O projeto não incluirá múltiplos modos de jogo na versão inicial, focando apenas no modo padrão de identificação de localização.
- O sistema não implementará um sistema de troféus e conquistas nesta versão.
- Não haverá suporte inicial para criação de desafios personalizados pelos usuários.
- O sistema não incluirá funcionalidades de streaming/transmissão ao vivo das partidas.
- Chat em tempo real entre jogadores não será implementado na versão inicial.

4 3. Especificação Técnica

4.1 3.1. Requisitos de Software

4.1.1 3.1.1. Lista de Requisitos

Requisitos Funcionais (RF):

1. **RF01** - O sistema deve permitir cadastro de novos usuários com email e senha ou Google/Facebook.
2. **RF02** - O sistema deve autenticar usuários registrados com sessões seguras.
3. **RF03** - O sistema deve permitir que usuários iniciem uma nova partida solo ou multijogador.
4. **RF04** - O sistema deve apresentar localizações aleatórias ou temáticas para os jogadores.
5. **RF05** - O sistema deve calcular a pontuação baseada na proximidade da resposta do jogador à localização real usando algoritmos geoespaciais.
6. **RF06** - O sistema deve armazenar histórico completo de partidas dos usuários.
7. **RF07** - O sistema deve permitir a criação de rankings globais e por período.
8. **RF08** - O sistema deve permitir que usuários adicionem amigos e os desafiem para partidas.
9. **RF09** - O sistema deve permitir que usuários visualizem e editem seu perfil.

10. **RF10** - O sistema deve permitir que usuários recuperem senhas via email.
11. **RF11** - O sistema deve exibir estatísticas detalhadas de desempenho do jogador.
12. **RF12** - O sistema deve implementar sistema de níveis baseado na experiência do jogador.

Requisitos Não-Funcionais (RNF):

1. **RNF01** - O sistema deve responder a interações do usuário em menos de 500ms para operações críticas de jogo.
2. **RNF02** - O sistema deve suportar pelo menos 1.000 usuários simultâneos sem degradação de performance.
3. **RNF03** - O sistema deve garantir disponibilidade de 99.5% durante horários de pico.
4. **RNF04** - O sistema deve proteger dados sensíveis de usuários com criptografia AES-256.
5. **RNF05** - O sistema deve ser compatível com os principais navegadores (Chrome 90+, Firefox 88+, Safari 14+, Edge 90+).
6. **RNF06** - O sistema deve ser responsivo para dispositivos móveis e tablets.
7. **RNF07** - O sistema deve resistir a ataques comuns (XSS, CSRF, injeção SQL, DDoS básico).
8. **RNF08** - As APIs devem ser RESTful com documentação OpenAPI/Scalar completa.
9. **RNF09** - O sistema deve implementar versionamento de API para backward compatibility.
10. **RNF10** - O sistema deve implementar observabilidade com logs estruturados, métricas e traces distribuídos.
11. **RNF11** - O sistema deve garantir integridade transacional entre microsserviços usando padrão Saga.
12. **RNF12** - O sistema deve implementar rate limiting adaptativo para prevenir abusos.
13. **RNF13** - O sistema deve usar cache distribuído para otimizar requisições recorrentes com TTL apropriado.
14. **RNF14** - O sistema deve possuir cobertura de testes automatizados de pelo menos 80%.
15. **RNF15** - O sistema deve suportar deployment sem downtime (blue-green deployment).
16. **RNF16** - O sistema deve implementar monitoramento de saúde em tempo real de todos os serviços com alertas automáticos.
17. **RNF17** - O sistema deve suportar crescimento horizontal automático baseado em demanda (auto-scaling).
18. **RNF18** - O sistema deve implementar backup automático diário dos dados críticos.
19. **RNF19** - O sistema deve ter tempo de recuperação (RTO) máximo de 30 minutos.
20. **RNF20** - O sistema deve consumir máximo de 80% dos recursos do servidor em operação normal.

4.1.2 3.1.2. Representação dos Requisitos

4.1.2.1 Diagrama de Casos de Uso - UC01: Jogar Partida Casos de Uso Principais:

- **UC01**: Jogar Partida
- **UC02**: Autenticar Usuário
- **UC03**: Gerenciar Perfil
- **UC04**: Calcular Pontuação
- **UC05**: Visualizar Rankings
- **UC06**: Gerenciar Amizades
- **UC07**: Visualizar Estatísticas

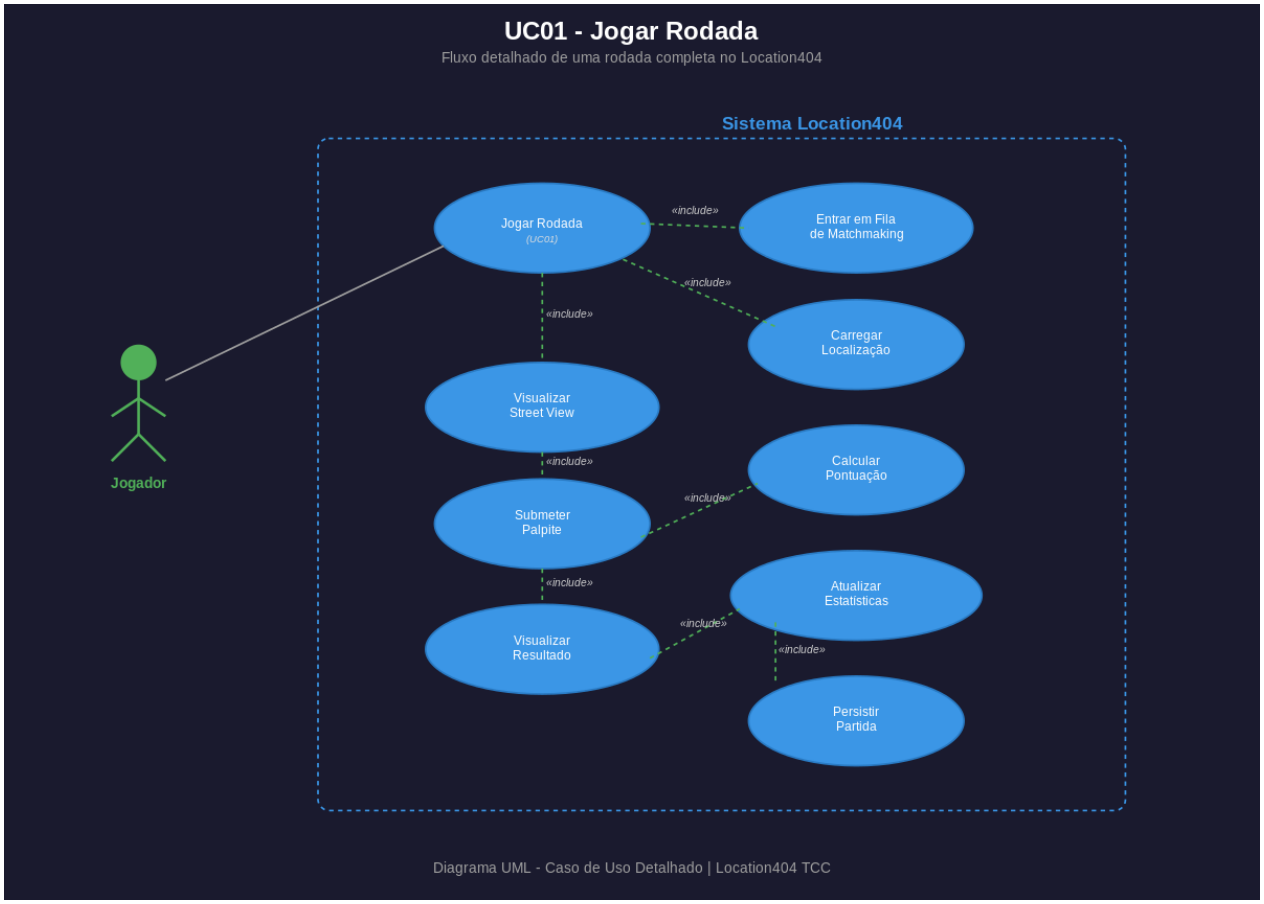

Figure 1: Diagrama de Casos de Uso - UC01

4.2 3.2. Considerações de Design

4.2.1 3.2.1. Visão Inicial da Arquitetura

O Location404 seguirá uma arquitetura de microsserviços baseada em Domain-Driven Design (DDD), com os seguintes componentes principais:

1. **Traefik**: Proxy reverso e load balancer para roteamento de requisições com SSL termination automático.
2. **location404-auth**: Gerenciamento de identidade, autenticação e autorização com JWT e refresh tokens.
3. **location404-game**: Lógica central de jogabilidade em tempo real via SignalR, matchmaking, cálculo de pontuações e mecânicas de jogo.
4. **location404-data**: Serviço de persistência e análise responsável por gerenciar localizações (+100 seeds), processar eventos de partidas via RabbitMQ, persistir histórico completo, calcular estatísticas de jogadores e manter sistema de ranking global.
5. **Location404.Shared.Observability**: SDK interno compartilhado para instrumentação padronizada de todos os serviços, incluindo métricas (Prometheus), tracing distribuído (Jaeger) e logs estruturados (Loki).

Componentes de Infraestrutura:

- **Dragonfly**: Cache distribuído Redis-compatible para fila de matchmaking e estado de partidas.
- **PostgreSQL**: Banco de dados principal (2 instâncias: auth + data).
- **RabbitMQ**: Message broker para comunicação assíncrona entre serviços.
- **Grafana LGTM Stack + Pyroscope**: Loki (logs), Grafana (visualização), Tempo (traces), Prometheus (métricas), Pyroscope (profiling) - implementado com OpenTelemetry.

4.2.2 3.2.2. Padrões de Arquitetura

- **Microsserviços**: Arquitetura principal, permitindo desenvolvimento, implantação e escalabilidade independentes.
- **API Gateway Pattern**: Traefik como ponto único de entrada para todas as requisições.
- **CQRS (Command Query Responsibility Segregation)**: Para separar operações de leitura e escrita em serviços críticos.
- **Event-Driven Architecture**: Para comunicação assíncrona entre serviços usando RabbitMQ.
- **Repository Pattern**: Para abstração da camada de persistência.
- **Clean Architecture**: Para organização interna de cada microsserviço.
- **Circuit Breaker Pattern**: Para resiliência entre chamadas de serviços.
- **Saga Pattern**: Para transações distribuídas complexas.
- **Outbox Pattern**: Para garantir entrega de eventos.
- **BFF (Backend for Frontend)**: Para otimizar APIs específicas para o cliente web.

4.2.3 3.2.3. Modelos C4

Modelo C4: Contexto do Sistema

Modelo C4: Contêineres

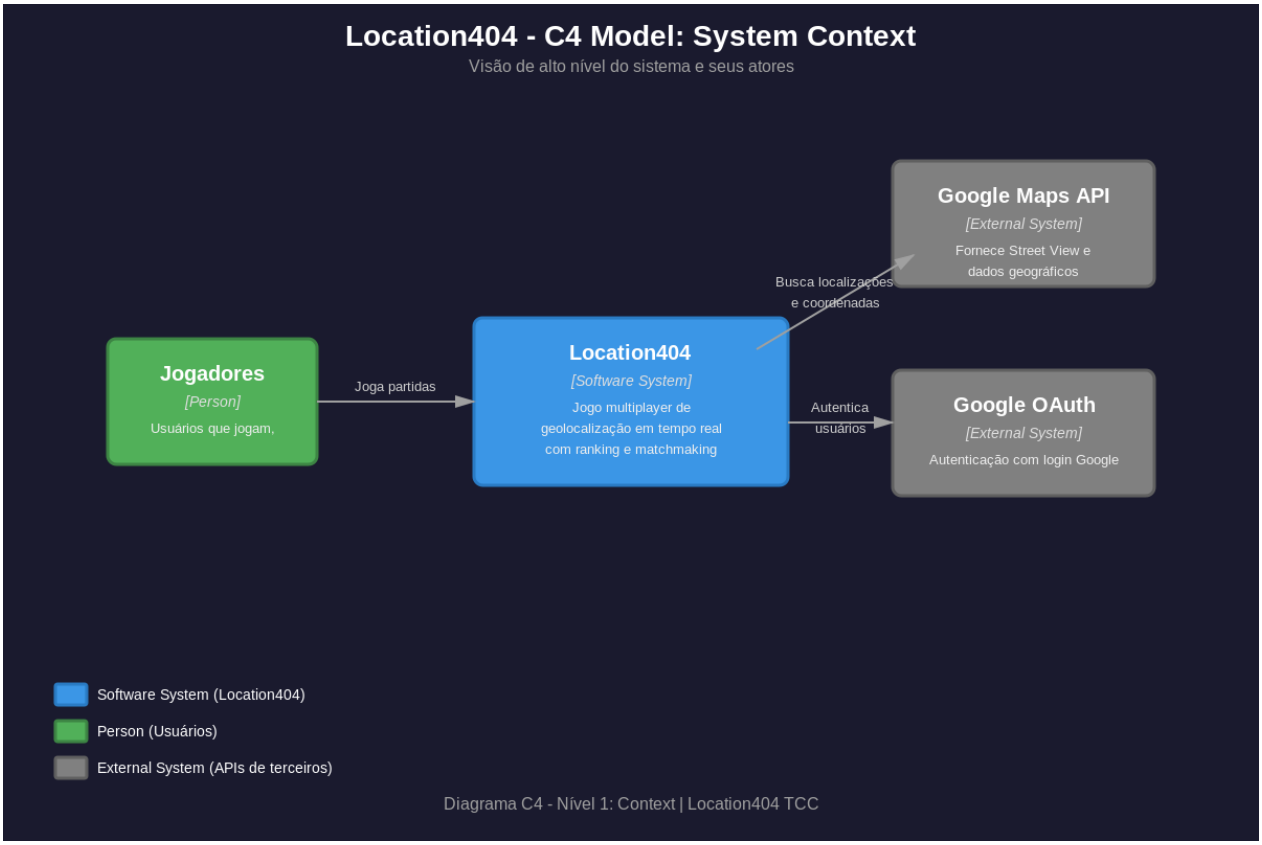

Figure 2: Diagrama C4 - Level 1: System Context

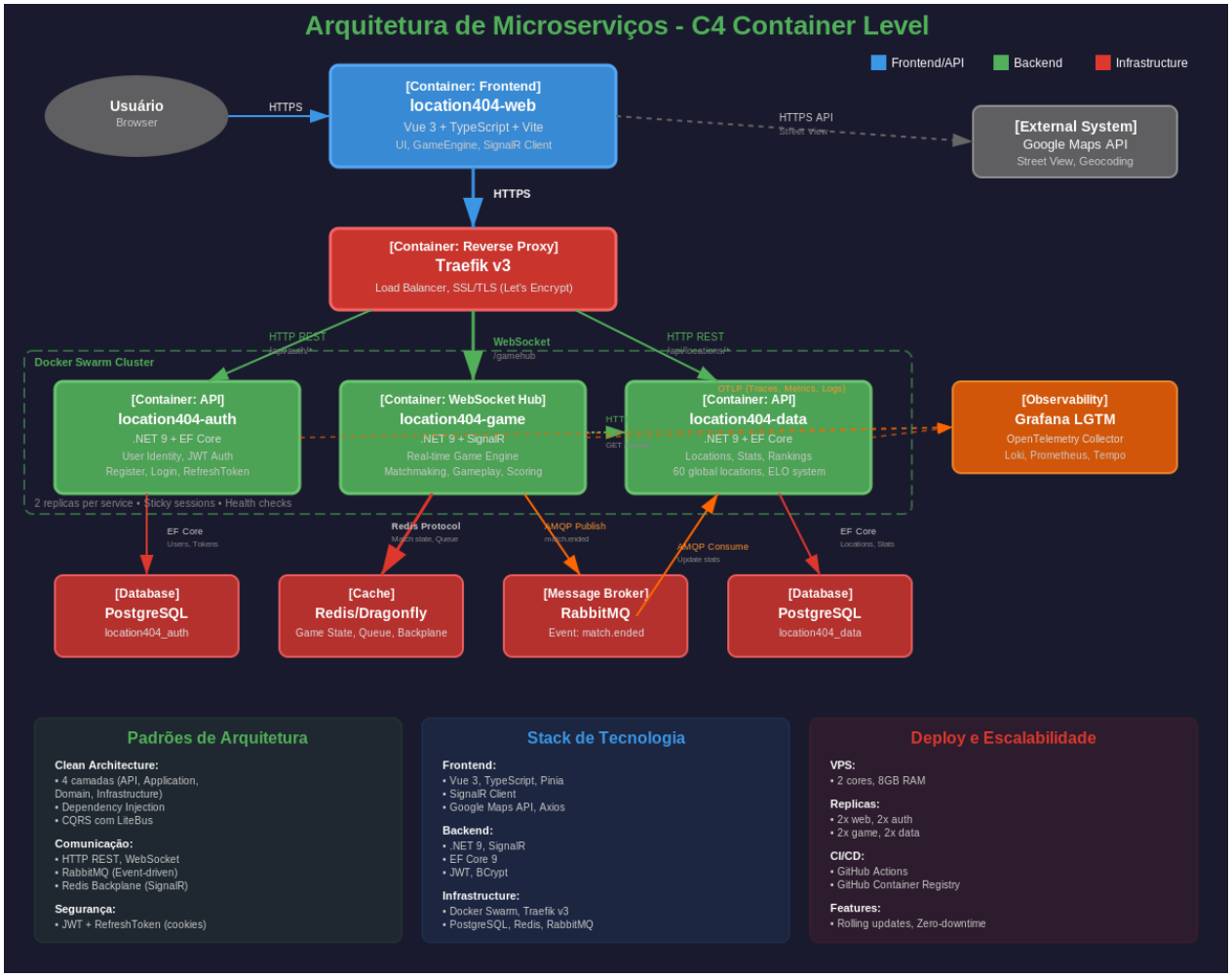

Figure 3: Diagrama C4 - Container Level

4.3 3.3. Stack Tecnológica

4.3.1 3.3.1. Linguagens de Programação

- **C# 12 (.NET 9+)**: Escolhido para o desenvolvimento do backend devido à sua robustez, desempenho excepcional e excelente suporte a aplicações empresariais. O .NET 9 traz recursos avançados de performance (AOT compilation), produtividade e observabilidade para microsserviços.
- **TypeScript 5.4+**: Para o desenvolvimento frontend com Vue 3, proporcionando segurança de tipo e recursos avançados de linguagem que melhoram a qualidade do código e a experiência de desenvolvimento.
- **SQL**: Para consultas otimizadas em bancos de dados relacionais.
- **JavaScript**: Para scripts de automação e algumas funcionalidades específicas do frontend.

4.3.2 3.3.2. Frameworks e Bibliotecas

Backend (.NET 9+):

- **ASP.NET Core 9+**: Framework base para desenvolvimento de APIs RESTful com suporte nativo a OpenTelemetry.
- **Entity Framework Core 9+**: ORM para acesso a dados com otimizações de performance.
- **LiteBus**: Para implementação do padrão mediator e CQRS (command/query bus pattern).
- **Polly**: Para implementação de políticas de resiliência (Circuit Breaker, Retry, Timeout).
- **SignalR**: Para comunicação em tempo real entre jogadores.
- **Serilog**: Para logging estruturado com sinks para múltiplos destinos.
- **OpenAPI/Scalar**: Para documentação automática de APIs (Scalar.AspNetCore para UI interativa).
- **StackExchange.Redis**: Cliente .NET para Dragonfly (Redis-compatible).
- **RabbitMQ.Client**: Para integração com message broker.
- **BackgroundService**: Para processamento de jobs em background (consumer RabbitMQ).

Testes:

- **xUnit**: Framework de testes unitários.
- **Moq**: Para criação de mocks.

Frontend (Vue 3):

- **Vue 3**: Framework principal com Composition API.
- **Vue Router 4**: Para roteamento SPA.
- **Pinia**: Para gerenciamento de estado global com persistência (pinia-plugin-persistedstate).
- **Google Maps JavaScript API**: Para visualização de mapas interativos e Street View.
- **TailwindCSS**: Para estilização utilitária e responsiva.
- **Vite**: Build tool otimizado para desenvolvimento.
- **Axios**: Para requisições HTTP com interceptors.
- **@microsoft/signalr**: Cliente SignalR para comunicação real-time.
- **vue-sonner**: Sistema de notificações toast.

Testes Frontend:

- **Vitest**: Para testes unitários rápidos com cobertura via v8.
- **@testing-library/vue**: Para testes de componentes.

Infraestrutura e DevOps:

- **Docker Swarm**: Orquestração de containers em produção com 2 réplicas por serviço.
- **Traefik v3**: Proxy reverso, load balancing e SSL automático com Let's Encrypt.
- **Dragonfly**: Cache distribuído Redis-compatible para matchmaking e estado de jogo.
- **PostgreSQL 16+**: Banco de dados principal (2 instâncias: auth + data).
- **RabbitMQ 3.12+**: Para mensageria assíncrona entre serviços (match.ended events).
- **Grafana LGTM Stack + Pyroscope**: Loki (logs), Grafana (dashboards), Tempo (traces), Prometheus (métricas), Pyroscope (profiling).
- **OpenTelemetry**: Instrumentação padronizada via Location404.Shared.Observability (NuGet).
- **GitHub Actions**: CI/CD automatizado com build e push para GitHub Container Registry.
- **Dokploy**: Platform de deploy e gerenciamento de stacks Docker.

4.3.3 3.3.3. Ambiente de Hospedagem

O sistema está hospedado em uma **VPS** com as seguintes especificações:

- **CPU**: 2 cores
- **RAM**: 8 GB
- **Storage**: 100 GB SSD
- **Domínio**: location404.com
- **OS**: Ubuntu 22.04 LTS

Esta configuração permite controle total sobre o ambiente de execução, rede, segurança e escalabilidade, facilitando a implantação de microsserviços com Docker Swarm, configurações personalizadas de proxy reverso com Traefik e otimizações específicas de desempenho.

Configuração Atual: O ambiente roda **2 réplicas por serviço** (web, auth, game, data) com load balancing via Traefik. Limitação: Docker Swarm não possui auto-scaling horizontal nativo (HPA). Um roadmap futuro prevê migração para Kubernetes (K3s) para implementar Horizontal Pod Autoscaler baseado em CPU/memória.

4.4 3.4. Considerações de Segurança

O Location404 implementará múltiplas camadas de segurança:

4.4.1 3.4.1. Autenticação e Autorização

- **JWT (JSON Web Tokens)** com tempo de expiração de 15 minutos
- **Refresh tokens** com rotação automática e tempo de vida de 7 dias
- **Rate limiting** específico para tentativas de login
- **OAuth 2.0/OpenID Connect** com PKCE (planejado para versão futura)
- **Integração com Google OAuth** para login social (planejado para versão futura)
- **Confirmação de email** para novos cadastros (planejado para versão futura)

4.4.2 3.4.2. Proteção de Dados

- **Criptografia em repouso**: AES-256 para dados sensíveis no banco de dados
- **Hashing de senhas**: bcrypt com salt randômico e custo adaptável

4.4.3 3.4.3. Segurança da API

- **CORS configurado** com whitelist de domínios específicos
- **Proteção CSRF** com tokens anti-forgery
- **Sanitização** de dados de entrada para prevenir XSS
- **Proteção contra SQL Injection** via Entity Framework parameterizado
- **Rate limiting adaptativo** baseado em comportamento do usuário

4.4.4 3.4.4. Segurança da Infraestrutura

- **Firewall de aplicação web (WAF)** configurado no Traefik
- **Escaneamento automático** de vulnerabilidades com dependabot
- **Secrets management** com Docker Secrets ou vault external
- **Network policies** restritivas entre serviços

4.4.5 3.4.5. Monitoramento e Detecção

- **Logging estruturado** de todos os eventos de segurança
- **Detecção de anomalias** em tempo real com alertas automáticos
- **Métricas de segurança** no dashboard de monitoramento
- **Audit logs** para trilha de auditoria completa

4.4.6 3.4.6. Segurança Específica do Jogo

- **Validação server-side** de todas as ações de jogo
- **Obfuscação de dados**: APIs não expõem respostas corretas antes do fim da rodada
- **Limitação de tentativas** por partida

5 4. Arquitetura Detalhada

5.1 4.1. Fluxo de Dados

1. **Autenticação**: Cliente → Traefik → location404-auth → JWT retornado
2. **Matchmaking**: Cliente → SignalR → location404-game → Dragonfly (fila de matchmaking)
3. **Iniciar Rodada**: location404-game → location404-data (buscar localização aleatória) → Cliente
4. **Submeter Palpite**: Cliente → SignalR → location404-game → Cálculo (Haversine) → RabbitMQ
5. **Persistir Partida**: location404-game → RabbitMQ → location404-data (salvar match + atualizar stats)
6. **Observabilidade**: Todos os serviços → Location404.Shared.Observability → Prometheus, Loki, Tempo

5.2 4.2. Estratégia de Cache

- **Dragonfly**: Fila de matchmaking (SortedSet), estado de partidas ativas (TTL: 2h), palpites temporários (TTL: 5min)

5.3 4.3. Estratégia de Banco de Dados

- **PostgreSQL (location404-auth)**: Usuários, senhas hash (BCrypt), refresh tokens
- **PostgreSQL (location404-data)**: +100 localizações seed, histórico de partidas, rodadas, estatísticas de jogadores
- **Dragonfly**: Cache distribuído, matchmaking queue, estado temporário de jogo

6 5. Próximos Passos

6.1 5.1. Status de Desenvolvimento (Novembro 2024)

Componentes Implementados:

- location404-auth: Autenticação JWT + RefreshToken
- location404-game: Engine real-time com SignalR + matchmaking
- location404-data: API de localizações (+100 seeds) + ranking global
- location404-web: Interface Vue 3 + TypeScript + Pinia
- Location404.Shared.Observability: SDK OpenTelemetry (NuGet publicado)
- Docker Swarm em produção com Traefik v3 + SSL automático
- Grafana LGTM Stack + Pyroscope (Loki, Grafana, Tempo, Prometheus, Pyroscope)
- RabbitMQ para eventos assíncronos (match.ended)
- CI/CD com GitHub Actions

Testes Implementados:

- +80% cobertura backend no 25% no frontend)
- SonarCloud: Grade A, 0 code smells
- Health checks em todos os serviços

Próximos Passos (Dezembro 2024):

- Testes E2E com Playwright ou Cypress
- Rate limiting no Traefik
- Preparação para Demo Day e apresentação final

Melhorias Futuras Planejadas:

- **Google OAuth**: Integração com Google OAuth 2.0/OpenID Connect para login social
- **Confirmação de Email**: Sistema de verificação de email para novos cadastros
- **Recuperação de Senha**: Fluxo de reset de senha via email
- **Sistema de Amizades**: Adicionar amigos e desafiar para partidas
- **Sistema de Níveis**: Experiência (XP) e progressão de níveis
- **Migração para Kubernetes (K3s)**: Auto-scaling horizontal com HPA

6.2 5.2. Métricas de Sucesso

- **Performance**: 95% das requisições < 500ms
- **Disponibilidade**: 99.5% de uptime
- **Usuários**: Suporte para 1000+ usuários simultâneos
- **Cobertura de Testes**: 80%+ de code coverage
- **Segurança**: Zero vulnerabilidades críticas

7 6. Referências

1. Microsoft. (2024). *.NET 9 Documentation*. Disponível em: https://docs.microsoft.com/en-us/dotnet/
2. Vue.js Team. (2024). *Vue 3 Guide*. Disponível em: https://vuejs.org/guide/
3. OWASP Foundation. (2024). *Web Security Testing Guide*. Disponível em: https://owasp.org/www-project-web-security-testing-guide/
4. Google Developers. (2024). *Maps Platform API Documentation*. Disponível em: https://developers.google.com/maps/documentation
5. OpenStreetMap Foundation. (2024). *API Documentation*. Disponível em: https://wiki.openstreetmap.org/wik
6. Traefik Labs. (2024). *Traefik v3.0 Documentation*. Disponível em: https://doc.traefik.io/traefik/
7. DragonflyDB. (2024). *Dragonfly Documentation*. Disponível em: https://www.dragonflydb.io/docs
8. PostgreSQL Global Development Group. (2024). *PostgreSQL Documentation*. Disponível em: https://www.postgresql.org/docs/
9. Newman, S. (2021). *Building Microservices: Designing Fine-Grained Systems*. 2nd Edition. O'Reilly Media.

8 7. Apêndices

8.1 Apêndice A: Glossário de Termos

- **Microsserviços**: Estilo arquitetural onde o aplicativo é composto de pequenos serviços autônomos que se comunicam via APIs.
- **API Gateway**: Ponto único de entrada que roteia requisições para microsserviços apropriados.
- **Circuit Breaker**: Padrão que previne falhas em cascata parando chamadas para serviços que estão falhando.
- **CQRS**: Command Query Responsibility Segregation - separação entre operações de leitura e escrita.
- **Event-Driven Architecture**: Paradigma onde componentes se comunicam através de eventos assíncronos.
- **JWT**: JSON Web Token - padrão para criação de tokens de acesso seguros.
- **OAuth 2.0**: Framework de autorização que permite acesso limitado a recursos do usuário.
- **Rate Limiting**: Técnica para controlar o número de requisições por período de tempo.

8.2 Apêndice B: Estimativa de Recursos

8.2.1 Infraestrutura de Produção

Componente	Especificação	Justificativa
Servidor Principal (VPS)	2 cores, 8 GB RAM, 100 GB SSD	Hospedagem de todos os microsserviços com Docker Swarm
PostgreSQL	2 instâncias independentes (auth + data)	Separação de contextos e isolamento de dados
Dragonfly	Instância única, Redis-compatible	Cache distribuído para matchmaking e rankings
Observabilidade	Prometheus + Grafana + Loki + Tempo + Pyroscope	Stack completa de monitoramento e tracing

Componente	Especificação	Justificativa
Backup	Backup diário automático, retenção 30 dias	Proteção de dados críticos

8.2.2 Custos de Infraestrutura

- **VPS**: R$ 399 (12 meses)
- **Domínio**: R$ 39 (anual)
- **APIs Externas**: Gratuito (Google Maps API - uso dentro do free tier)
- **Total**: R$ 438/ano

8.3 Apêndice C: Plano de Testes

8.3.1 Testes Unitários

- **Backend (.NET)**: 80% de cobertura - serviços de domínio, cálculos de pontuação, validações
- **Frontend (Vue)**: 25% de cobertura - componentes críticos e composables

8.3.2 Testes de Integração

- APIs entre microsserviços
- Integração com PostgreSQL
- Cache Dragonfly
- Message queue (RabbitMQ)

8.3.3 Testes End-to-End

- Fluxo completo de jogo
- Autenticação e autorização
- Cenários de falha
- Performance em carga

8.3.4 Testes de Segurança

- Penetration testing automatizado
- Verificação de vulnerabilidades
- Teste de autenticação
- Validação de rate limiting