# Template de Arquitetura de Software - Aquila

Autores: Diogo Gibertoni, Murilo Paes Jeronymo, Pedro Lelis, Vinicius Carvalho, Cauê Felipe Knies Debus

## 1. Introdução e Objetivos

O Sistema Aquila é uma plataforma inteligente de monitoramento e comparação de preços que visa fornecer ao Consumidor Final informações transparentes e seguras, indo além dos grandes marketplaces. O sistema é projetado para verificar a segurança de sites e identificar promoções falsas ("metade do dobro") através de análise histórica de preços.

### 1.1. Visão Geral dos Requisitos

**Requisitos Funcionais** • RF01: Realizar web scraping em 8-12 sites de e-commerce selecionados • RF02: Comparar preços de produtos idênticos através de correspondência por nome/código • RF03: Avaliar a segurança e confiabilidade básica de sites de e-commerce • RF04: Detectar promoções falsas através de análise histórica de preços • RF05: Notificar usuários sobre oportunidades reais de economia via email • RF06: Manter histórico de preços por 60 dias para análise de tendências • RF07: Permitir configuração de alertas personalizados por produto • RF08: Fornecer dashboard para visualização de tendências

**Requisitos Não-Funcionais** • RNF01: Processar 500-1000 produtos por hora • RNF02: Tempo de resposta < 3 segundos para consultas web • RNF03: Disponibilidade de 98% • RNF04: Conformidade básica com LGPD • RNF05: Suporte a 8-12 sites diferentes • RNF06: Interface responsiva para desktop e mobile

### 1.2. Objetivos de Qualidade

| **Prioridade** | **Objetivo** | **Descrição** |
| --- | --- | --- |
| 1 | Confiabilidade | Garantir precisão nas análises de segurança e detecção de fraudes |
| 2 | Performance | Processar grande volume de dados em tempo hábil |
| 3 | Manutenibilidade | Facilitar a adição de novos sites e a evolução das regras de análise. |
| 4 | Manutenibilidade | Proteger dados dos usuários e garantir conformidade com LGPD (RNF04). |
| 5 | Segurança | Proteger dados dos usuários e respeitar privacidade |

### 1.3. Partes Interessadas

| **Função/Nome** | **Contato** | **Expectativas** |
| --- | --- | --- |
| Consumidor Final | [usuarios@aquila.com](mailto:usuarios@aquila.com) | Encontrar melhores preços com segurança |
| Administrador de Sistema | [admin@aquila.com](mailto:admin@aquila.com) | Sistema estável e de fácil manutenção |
| Parceiros E-commerce | [parceiros@aquila.com](mailto:parceiros@aquila.com) | Integração justa e transparente |
| Equipe de Desenvolvimento | [dev@aquila.com](mailto:dev@aquila.com) | Código limpo e documentado |
| Jurídico | [juridico@aquila.com](mailto:juridico@aquila.com) | Conformidade legal e ética |

## 2. Restrições Arquiteturais

### Restrições Técnicas

| **Categoria** | **Restrição** |
| --- | --- |
| Linguagem Backend | Node.js com Express.js |
| Linguagem Frontend | HTML5, CSS3, JavaScript Vanilla (MicroFrontEnd) |
| Banco de Dados | MongoDB Atlas (Produtos - NoSQL), Azure SQL Server (Preços - SQL) |
| Containerização | Docker, Docker Compose |
| Publicação | GitHub (código) e Docker Hub (imagens) |
| Versionamento | Git/GitHub |
| Arquitetura | Microsserviços + Serverless Functions + BFF |

### Restrições Organizacionais

| **Restrição** | **Descrição** |
| --- | --- |
| Equipe | Máximo de 5 desenvolvedores |
| Prazo | MVP em 3 meses |

### Restrições Legais

| **Restrição** | **Descrição** |
| --- | --- |
| LGPD | Conformidade total com Lei Geral de Proteção de Dados |
| Robots.txt | Respeitar políticas de scraping dos sites |
| Propriedade Intelectual | Não violar direitos autorais de conteúdo |

## 3. Contexto e Escopo

### Contexto Negocial

**Interfaces de Domínio Externo**

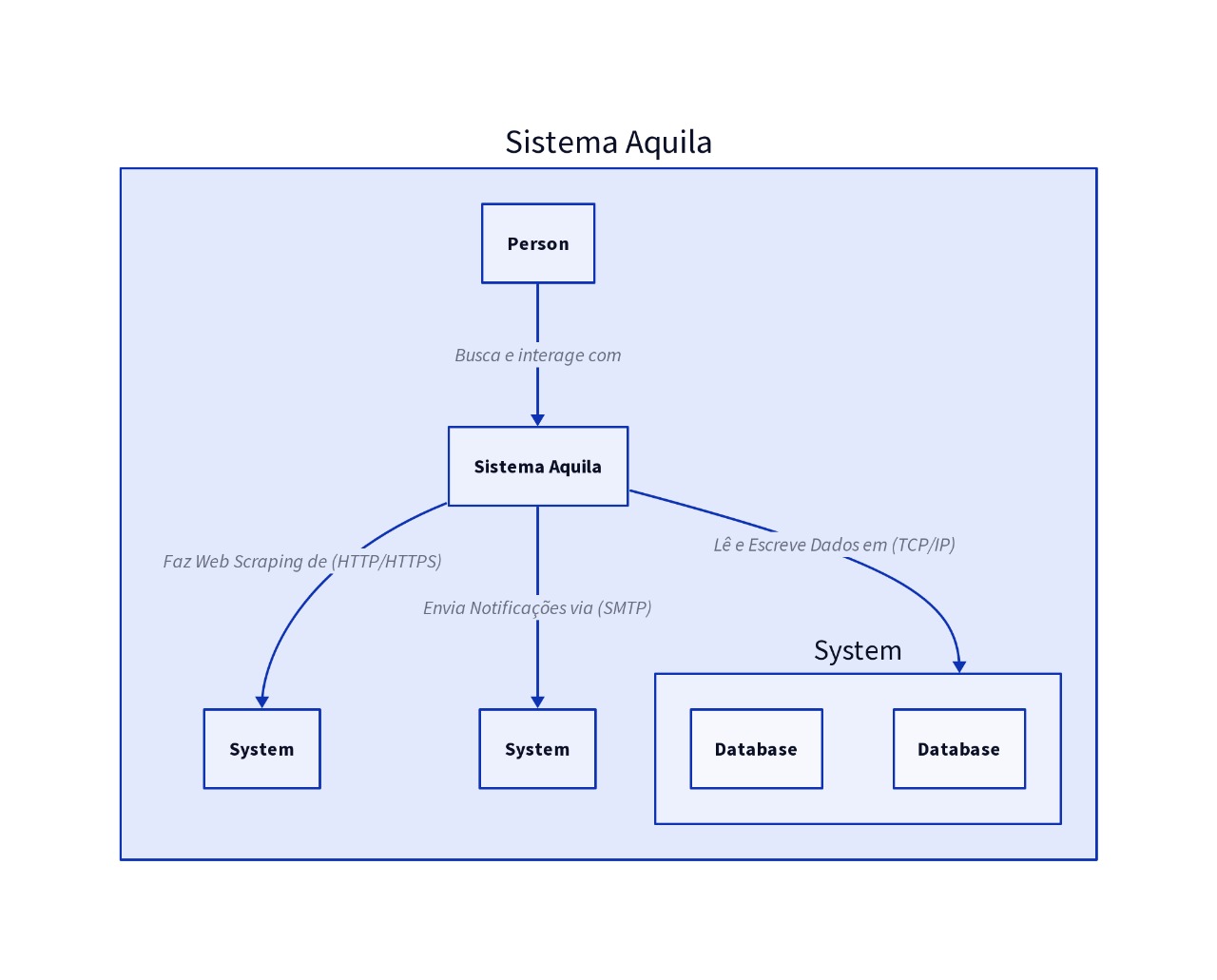
| **Sistema Externo** | **Descrição da Interface** |
| --- | --- |
| Sites E-commerce | Web scraping HTTP/HTTPS respeitando robots.txt |
| Serviços SSL | Verificação de certificados via bibliotecas |
| SMTP Server | Envio de emails via protocolo SMTP |
| APIs Públicas | Consulta a APIs quando disponíveis (ex: ML API) |

### Contexto Técnico

**Interfaces Técnicas**

| **Interface** | **Protocolo** | **Formato** |
| --- | --- | --- |
| MicroFrontEnd ↔ BFF | HTTPS | JSON/REST |
| BFF ↔ Microsserviços | HTTP/HTTPS | JSON/REST |
| Microsserviços ↔ Database | TCP/IP | SQL (Azure SQL), MongoDB Protocol |
| Functions ↔ Microsserviços | HTTP/HTTPS | JSON/REST |
| Backend ↔ Sites E-commerce | HTTP/HTTPS | HTML |
| Backend ↔ Email Service | SMTP | MIME |

## Diagrama C4 Model – Nivel 1:



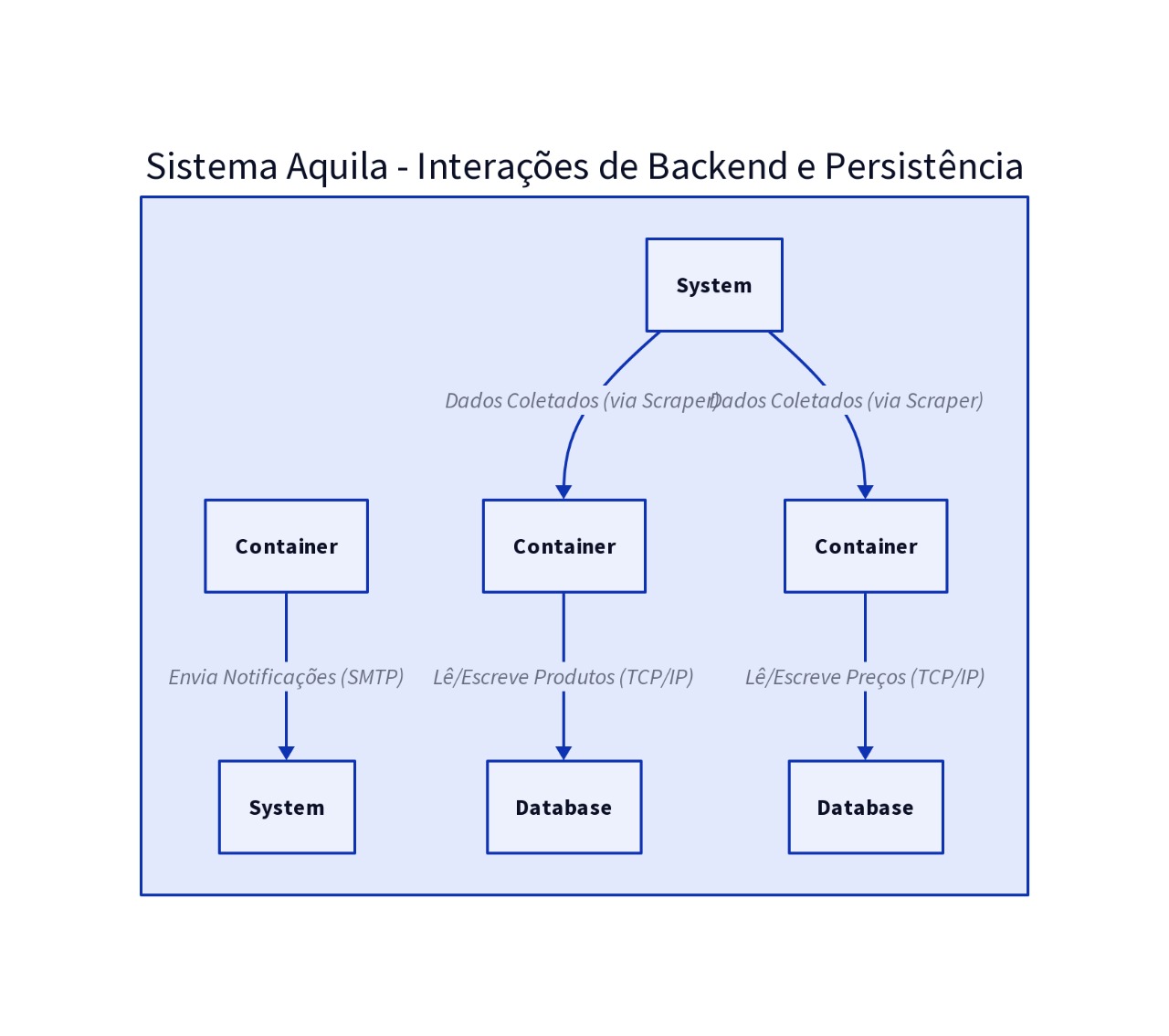
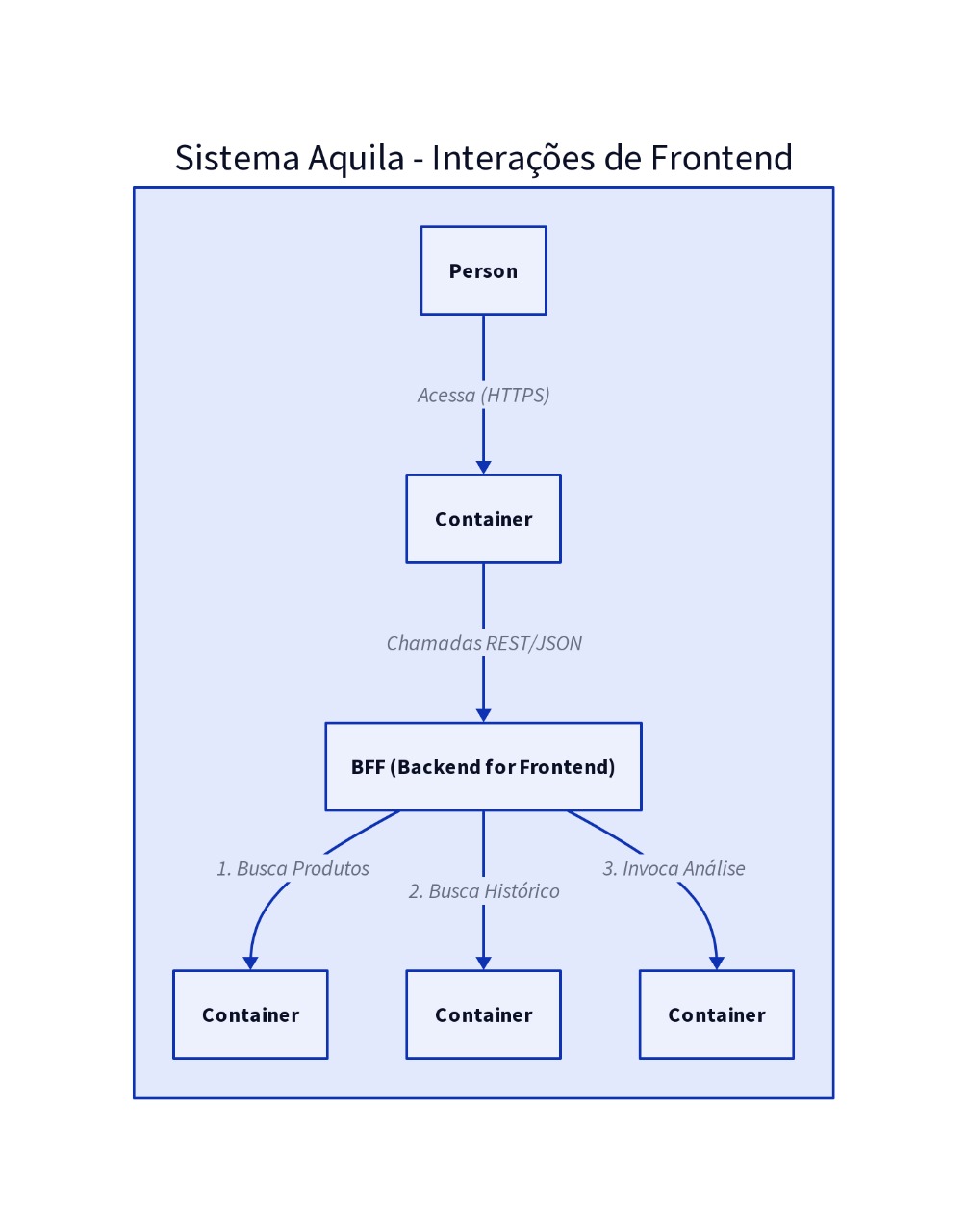
## 4. Estratégia de Solução

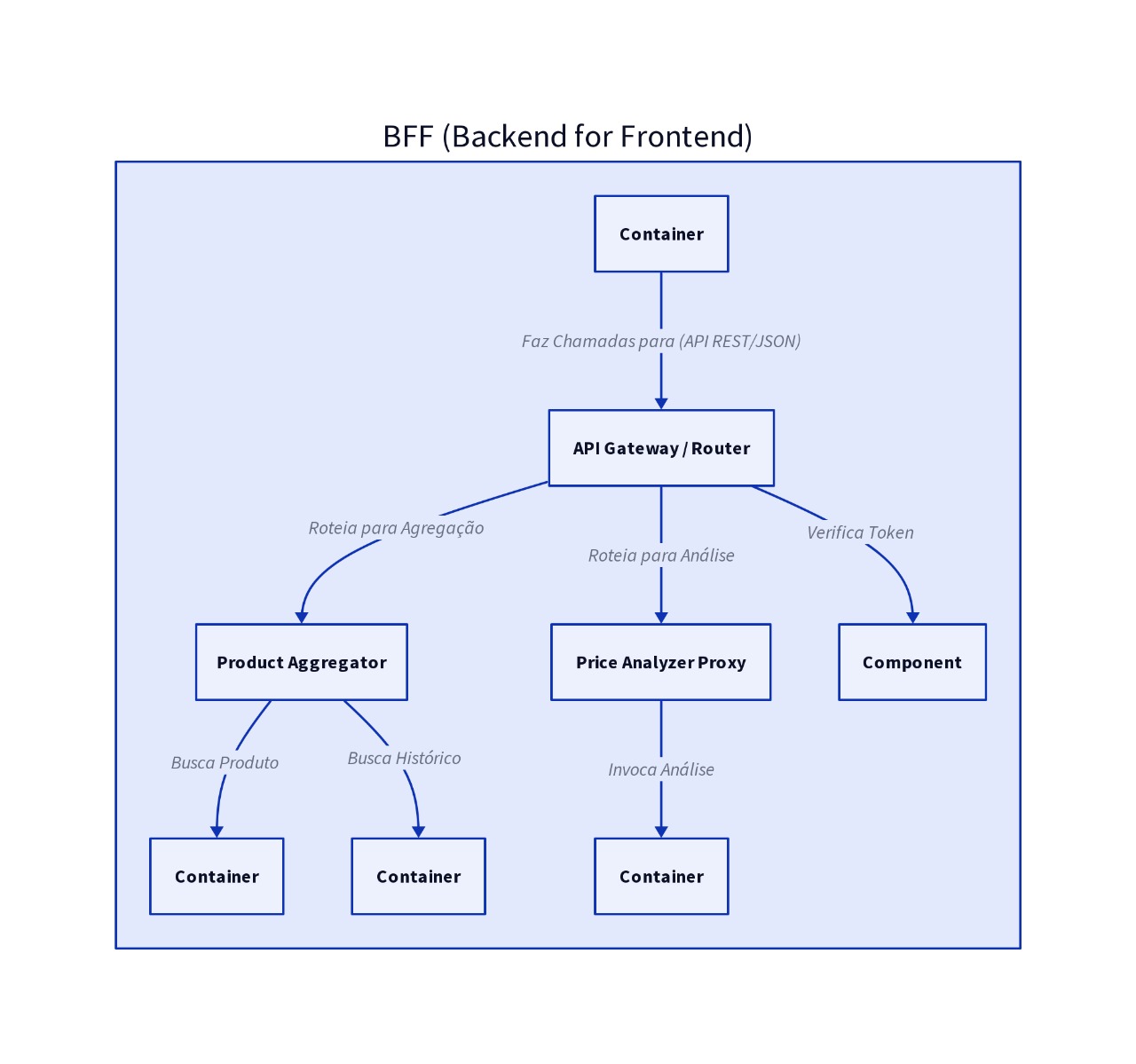
### Decisões Tecnológicas Fundamentais

| **Problema** | **Solução** |
| --- | --- |
| Escalabilidade e Acoplamento | Arquitetura de Microsserviços |
| Persistência de Dados | Polyglot Persistence (MongoDB + Azure SQL) |
| Agregação de Dados | BFF (Backend for Frontend) |
| Processamento Assíncrono | Serverless Functions |
| Web Scraping | Puppeteer (Headless Browser) |
| Autenticação | JWT (JSON Web Tokens) |
| Containerização | Docker + Docker Compose |

## 5. Visão de Blocos de Construção

O sistema é decomposto em contêineres (microsserviços, functions, BFF e MicroFrontEnd) que se comunicam via APIs REST/JSON. A arquitetura segue os princípios de Domain-Driven Design (DDD) com bounded contexts claros.





| Bloco de Construção | Tipo | Responsabilidade | Tecnologia |
| --- | --- | --- | --- |
| MicroFrontEnd | Aplicação Web | Camada de apresentação e interação com o usuário. | Node.js, HTML, CSS, JS |
| BFF | Aplicação Web | API Gateway e Agregador de dados para o frontend. | Node.js, Express.js |
| Product Service | Microsserviço | Gerencia o domínio de Produtos (CRUD). | Node.js, Express.js |
| Price Service | Microsserviço | Gerencia o domínio de Preços e Histórico. | Node.js, Express.js |
| Price Analyzer Function | Função | Detecção de promoções falsas (RF04). | Node.js |
| Event Processor Function | Função | Processamento de eventos de preço e notificação (RF05). | Node.js |
| MongoDB | Banco de Dados | Persistência de Produtos (NoSQL). | MongoDB Atlas |
| Azure SQL | Banco de Dados | Persistência de Preços (SQL). | Azure SQL Server |

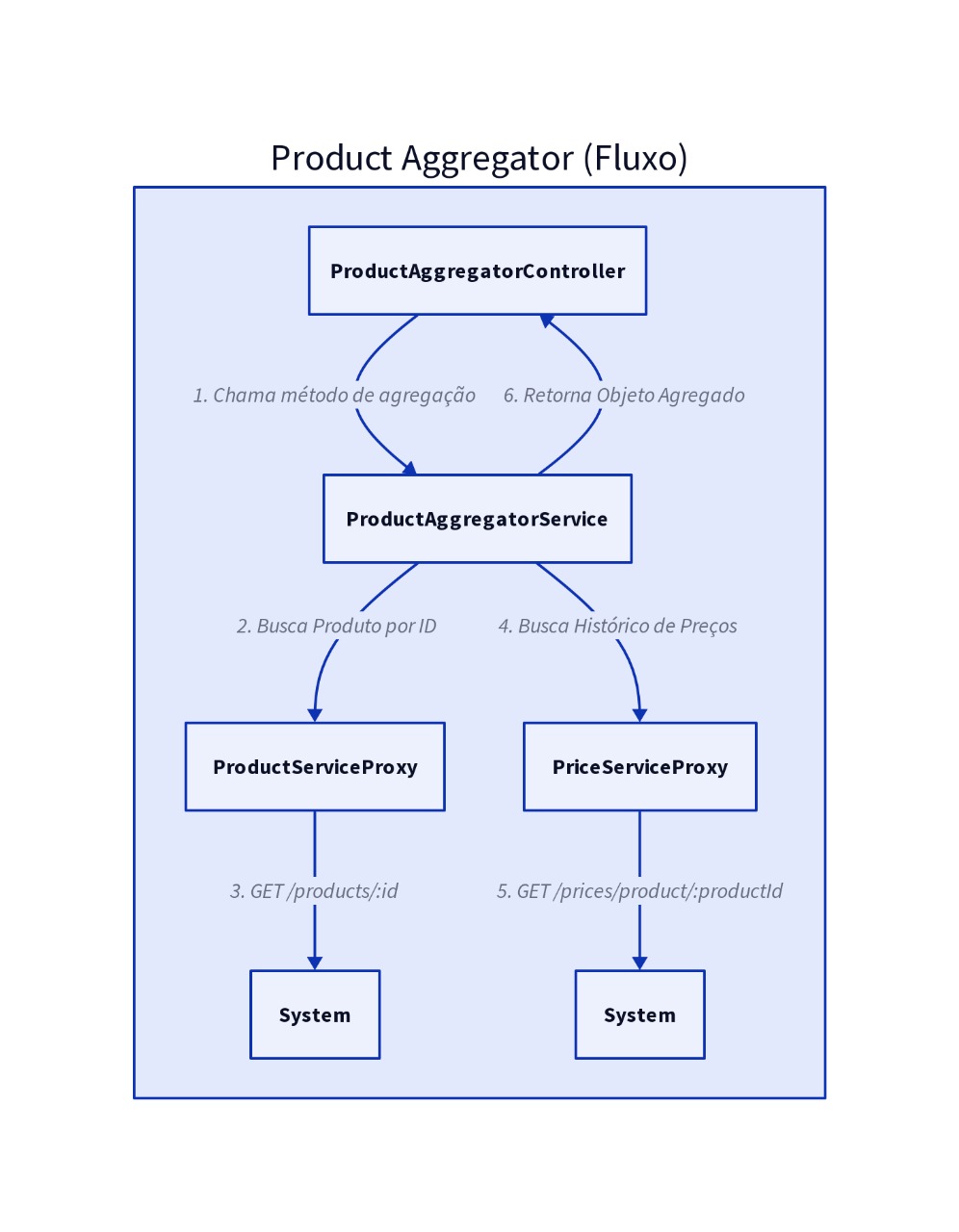
## 6. Visão de Tempo de Execução

***Cenário 1: Agregação Completa de Produto (BFF)***

**Propósito:** Atender à requisição do Consumidor Final para visualizar um produto e seu histórico de preços (GET /api/products/:id/complete).

1. **MicroFrontEnd** envia requisição ao **BFF** (via ProductAggregatorController).
2. O **BFF** (componente Product Aggregator) orquestra as seguintes chamadas em paralelo ou série: a. Chama o **Product Service** para obter dados básicos do produto. b. Chama o **Price Service** para obter o histórico de preços.
3. O **BFF** combina os dados e retorna a resposta agregada ao **MicroFrontEnd**.

*(Detalhes do fluxo de código no C4 Nível 4 - Product Aggregator)*



***Cenário 2: Processamento de Evento de Preço (Assíncrono*)**

**Propósito:** Inserir um novo preço no sistema e verificar se um alerta deve ser disparado.

1. Um novo preço é coletado (via Scraper, que pode ser um serviço externo ou interno) e um evento é gerado.
2. O **Event Processor Function** consome o evento.
3. O **Event Processor Function** persiste o novo preço no **Azure SQL** (via **Price Service**).
4. O **Event Processor Function** verifica as regras de alerta e, se necessário, envia uma mensagem ao **Serviço de Envio de E-mail** (RF05).

## 7. Visão de Implantação

### 7.1 Infraestrutura

A arquitetura de implantação utiliza **Docker** e **Docker Compose** para orquestração local e desenvolvimento, com imagens publicadas no **Docker Hub** e código-fonte no **GitHub** público.

### 7.2. Repositórios GitHub (Público)

| Componente | Repositório | Descrição |
| --- | --- | --- |
| MicroFrontEnd | https://github.com/[equipe]/aquila-frontend | Interface web responsiva com JavaScript Vanilla |
| BFF | https://github.com/[equipe]/aquila-bff | API Gateway e agregador de dados |
| Product Service | https://github.com/[equipe]/aquila-product-service | Microsserviço de gerenciamento de produtos |
| Price Service | https://github.com/[equipe]/aquila-price-service | Microsserviço de gerenciamento de preços |
| Price Analyzer Function | https://github.com/[equipe]/aquila-price-analyzer | Function para análise de promoções falsas |
| Event Processor Function | https://github.com/[equipe]/aquila-event-processor | Function para processamento de eventos |
| Documentação | https://github.com/[equipe]/aquila-docs | Diagramas C4, arc42, Software Architecture Canvas |

### 7.3. Imagens Docker Hub

| Componente | Imagem Docker Hub | Tag | Porta |
| --- | --- | --- | --- |
| MicroFrontEnd | diogogibertoni/aquila-frontend | latest, v1.0.0 | 80 |
| BFF | diogogibertoni/aquila-bff | latest, v1.0.0 | 3000 |
| Product Service | diogogibertoni/aquila-product-service | latest, v1.0.0 | 3001 |
| Price Service | [usuario]/aquila-price-service | latest, v1.0.0 | 3002 |
| Price Analyzer Function | [usuario]/aquila-price-analyzer | latest, v1.0.0 | 3003 |
| Event Processor Function | [usuario]/aquila-event-processor | latest, v1.0.0 | 3004 |

### 7.4. Docker Compose

version: '3.8'  
services:  
 frontend:  
 image: diogogibertoni/aquila-frontend:latest  
 ports:  
 - "80:80"  
 environment:  
 - BFF\_URL=http://bff:3000  
   
 bff:  
 image: diogogibertoni/aquila-bff:latest  
 ports:  
 - "3000:3000"  
 environment:  
 - PRODUCT\_SERVICE\_URL=http://product-service:3001  
 - PRICE\_SERVICE\_URL=http://price-service:3002  
 - JWT\_SECRET=${JWT\_SECRET}  
   
 product-service:  
 image: diogogibertoni/aquila-product-service:latest  
 ports:  
 - "3001:3001"  
 environment:  
 - MONGODB\_URI=${MONGODB\_URI}  
   
 price-service:  
 image: [usuario]/aquila-price-service:latest  
 ports:  
 - "3002:3002"  
 environment:  
 - AZURE\_SQL\_CONNECTION=${AZURE\_SQL\_CONNECTION}  
   
 price-analyzer:  
 image: [usuario]/aquila-price-analyzer:latest  
 ports:  
 - "3003:3003"  
   
 event-processor:  
 image: [usuario]/aquila-event-processor:latest  
 ports:  
 - "3004:3004"  
 environment:  
 - SMTP\_HOST=${SMTP\_HOST}  
 - SMTP\_USER=${SMTP\_USER}  
 - SMTP\_PASS=${SMTP\_PASS}

### 7.5. Requisitos de Hardware

| Componente | Desenvolvimento | Produção (por instância) |
| --- | --- | --- |
| **MicroFrontEnd** | 1 core, 512 MB RAM | 1 core, 512 MB RAM |
| **BFF** | 1 core, 512 MB RAM | 2 cores, 1 GB RAM |
| **Product Service** | 1 core, 512 MB RAM | 1 core, 1 GB RAM |
| **Price Service** | 1 core, 512 MB RAM | 2 cores, 1 GB RAM |
| **Functions** | Serverless (on-demand) | Serverless (escalabilidade automática) |
| **MongoDB Atlas** | Tier M0 (gratuito) | M10+ (produção) |
| **Azure SQL** | Basic tier | Standard tier |

### 7.6. Processo de Deploy

**1. Desenvolvimento Local:**

git clone https://github.com/[equipe]/aquila-[componente]  
cd aquila-[componente]  
npm install  
npm run dev

**2. Build e Push Docker:**

docker build -t [usuario]/aquila-[componente]:v1.0.0 .  
docker push [usuario]/aquila-[componente]:v1.0.0  
docker tag [usuario]/aquila-[componente]:v1.0.0 [usuario]/aquila-[componente]:latest  
docker push [usuario]/aquila-[componente]:latest

**3. Deploy com Docker Compose:**

docker-compose pull  
docker-compose up -d

## 8. Conceitos Transversais

### 8.1 Modelo de Domínio

O modelo de domínio segue os princípios de **Domain-Driven Design (DDD)** com bounded contexts claros para Produtos e Preços.

**Entidades Principais:**

Product Price  
├── id: UUID ├── id: UUID  
├── name: String ├── productId: UUID (FK)  
├── category: String ├── price: Decimal  
├── description: Text ├── store: String  
├── url: URL ├── url: URL  
├── imageUrl: URL ├── timestamp: DateTime  
├── createdAt: DateTime ├── isPromo: Boolean  
└── updatedAt: DateTime └── promoDetails: JSON  
  
User Alert  
├── id: UUID ├── id: UUID  
├── email: String ├── userId: UUID (FK)  
├── passwordHash: String ├── productId: UUID (FK)  
├── name: String ├── targetPrice: Decimal  
├── createdAt: DateTime ├── isActive: Boolean  
└── lastLogin: DateTime └── createdAt: DateTime

**Relacionamentos:** - Product 1 ↔ N Price (um produto tem múltiplos preços históricos) - User 1 ↔ N Alert (um usuário pode ter múltiplos alertas) - Product 1 ↔ N Alert (um produto pode ter múltiplos alertas de usuários diferentes)

### 8.2 Segurança

A segurança é implementada em múltiplas camadas seguindo as melhores práticas de segurança de aplicações web e microsserviços.

**Camadas de Segurança:**

|  |  |  |
| --- | --- | --- |
| Camada | Tecnologia | Implementação |
| **Autenticação** | JWT (JSON Web Tokens) | Tokens com expiração de 24h, refresh tokens com 7 dias. Algoritmo: HS256. |
| **Senhas** | Bcrypt | Hash com salt rounds = 10. Validação de força mínima: 8 caracteres, maiúsculas, números. |
| **Comunicação** | HTTPS/TLS 1.3 | Obrigatório em produção. Certificados Let’s Encrypt renovados automaticamente. |
| **Rate Limiting** | Express Rate Limit | 100 requests/minuto por IP. 10 requests/minuto para endpoints de autenticação. |
| **Input Validation** | Joi Schemas | Validação de todos os inputs no BFF e microsserviços. Sanitização de HTML/SQL. |
| **CORS** | Configuração Restritiva | Apenas origens whitelisted. Credentials: true para cookies seguros. |
| **Headers de Segurança** | Helmet.js | CSP, HSTS, X-Frame-Options, X-Content-Type-Options |
| **LGPD** | Criptografia + Logs | Dados sensíveis criptografados. Logs de acesso. Direito ao esquecimento implementado. |

## 9. Decisões de Arquitetura

### 9.1 ADR-001: Monolito vs Microsserviços

| **Aspecto** | **Decisão** |
| --- | --- |
| Status | Aceito |
| **Contexto** | Requisitos de alta performance (RNF01: 500-1000 produtos/hora) e escalabilidade independente exigem arquitetura mais granular. Equipe pequena (5 devs) precisa de clareza de responsabilidades. |
| **Decisão** | Adotar **Arquitetura de Microsserviços** com dois microsserviços (Product + Price), BFF, Functions e MicroFrontEnd. |
| Consequências | **(+)** Escalabilidade horizontal independente por serviço**(+)** Resiliência - falha isolada não derruba sistema**(+)** Tecnologia adequada a cada domínio (Polyglot)**(+)** Desenvolvimento paralelo de equipes**(-)** Maior complexidade de comunicação e debugging distribuído**(-)** Necessidade de orquestração (Docker Compose)**(-)** Overhead de rede entre serviços |

### 9.2 ADR-002: Estratégia de Persistência Poliglota

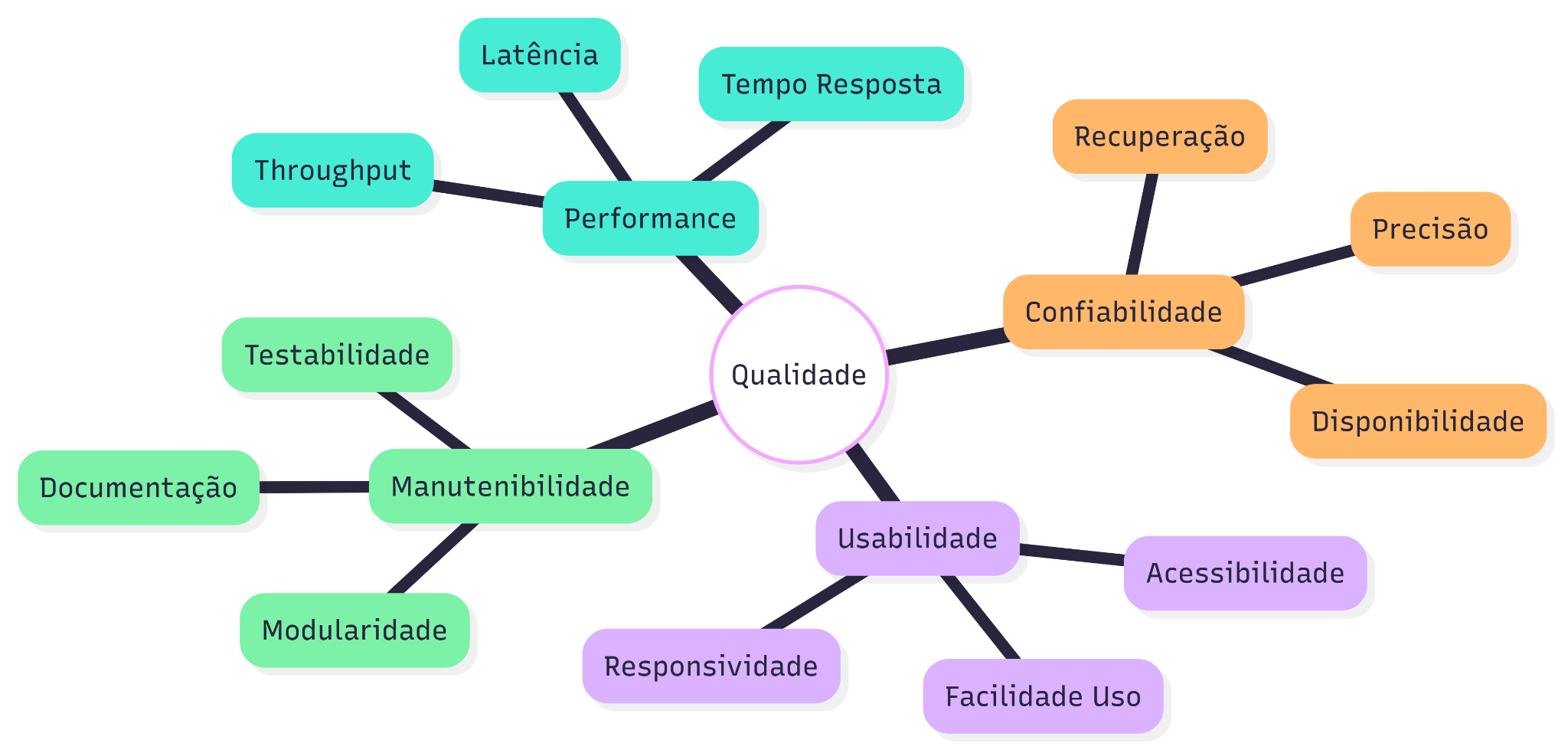
| **Aspecto** | **Decisão** |
| --- | --- |
| Status | Aceito |
| **Contexto** | Domínio de Produtos requer flexibilidade de schema (novos atributos, categorias dinâmicas). Domínio de Preços requer integridade transacional e consultas complexas de séries temporais. |
| **Decisão** | **MongoDB Atlas** para Produtos (NoSQL, schema flexível) + **Azure SQL Server** para Preços (SQL, integridade ACID). |
| **Consequências** | **(+)** MongoDB ideal para evolução rápida de schema de Produtos**(+)** Azure SQL garante integridade histórica de Preços**(+)** Cada banco otimizado para seu workload**(-)** Complexidade de gerenciar dois bancos**(-)** Impossibilidade de joins entre Produtos e Preços**(-)** Necessidade de coordenação de transações (Saga pattern) |

### 9.3 ADR-003 Estratégia de Scraping com Puppeteer

| **Aspecto** | **Decisão** |
| --- | --- |
| Status | Aceito |
| **Contexto** | Sites de e-commerce modernos utilizam SPAs (Single Page Applications) com conteúdo dinâmico renderizado via JavaScript. Scrapers simples (Axios + Cheerio) não conseguem extrair dados. |
| **Decisão** | Utilizar **Puppeteer** (Headless Chrome) para scraping. |
| Consequências | **(+)** Funciona com sites dinâmicos JavaScript**(+)** Pode interagir com elementos (cliques, scrolls)**(+)** Captura screenshots para debug**(-)** Consome mais recursos (memória, CPU)**(-)** Mais lento que scrapers simples**(-)** Pode ser detectado e bloqueado |

## 10. Requisitos de Qualidade

### 10.1 Árvore de Qualidade



### 10.2 Cenários de Qualidade

| **ID** | **Atributo** | **Cenário** | **Medida** |
| --- | --- | --- | --- |
| Q1 | Performance | Busca de produto | < 3 segundos |
| Q2 | Confiabilidade | Detecção de fraude | Precisão > 80% |
| Q3 | Manutenibilidade | Primeiro uso | < 5 min para criar alerta |
| Q4 | Disponibilidade | Sistema online | > 95% do tempo |
| Q5 | Manutenibilidade | Adicionar scraper | < 2 horas |

## 11. Riscos e Débitos Técnicos

### 11.1 Riscos

| **Risco** | **Probabilidade** | **Impacto** | **Mitigação** |
| --- | --- | --- | --- |
| Sites bloquearem scraping | Alta | Alto | User-agent rotation, delays |
| Mudança de estrutura HTML | Alta | Médio | Scrapers modulares |
| Limite de recursos gratuitos | Média | Alto | Otimização, cache |
| Atraso no cronograma | Média | Alto | Escopo reduzível |
| Latência entre Microsserviços | Média | Médio | Implementar *Health Checks* e *Circuit Breakers*. Otimizar o BFF para chamadas paralelas (Seção 6). Utilizar logs estruturados para rastreamento distribuído. |

### 11.2 Débitos Técnicos

| **Débito** | **Prioridade** | **Plano** |
| --- | --- | --- |
| Testes incompletos | Alta | Sprint final |
| Documentação API | Média | Swagger básico |
| Logs estruturados | Baixa | Pós-MVP |
| CI/CD completo | Baixa | Se houver tempo |
| Cache Redis | Baixa | Futura otimização |

## 12. Glossário

| **Termo** | **Definição** |
| --- | --- |
| **Aquila** | Nome do projeto: Plataforma inteligente de monitoramento e comparação de preços. |
| **BFF** | **Backend for Frontend**. Padrão de arquitetura que cria uma camada de agregação de dados e API Gateway otimizada para um tipo específico de frontend (MicroFrontEnd). |
| **Scraping** | Extração automatizada de dados de websites, geralmente através de análise do HTML. |
| **Promoção Falsa** | Aumento de preço seguido de um "desconto" que mantém o preço final próximo ou igual ao preço anterior, detectado pela análise histórica. |
| **JWT** | **JSON Web Token**. Padrão de autenticação stateless usado para transmitir informações de forma segura entre as partes como um objeto JSON. |
| **MicroFrontEnd** | Padrão de arquitetura que divide a interface de usuário em partes menores e independentes, permitindo o desenvolvimento, deploy e runtime autônomos. |
| **Polyglot Persistence** | Uso de múltiplas tecnologias de banco de dados (e.g., NoSQL e SQL) no mesmo sistema, onde cada uma é escolhida para atender melhor aos requisitos de um microsserviço específico. |
| **Puppeteer** | Biblioteca Node.js que fornece uma API de alto nível para controlar o Chrome/Chromium via protocolo DevTools. Usado para Scraping de sites dinâmicos. |
| **RNF** | **Requisito Não-Funcional**. Critério que julga a operação de um sistema, em vez de comportamentos específicos (e.g., Performance, Segurança). |
| **LGPD** | **Lei Geral de Proteção de Dados**. Legislação brasileira que regula as atividades de tratamento de dados pessoais. |
| **MVP** | **Minimum Viable Product**. Versão de um novo produto que permite à equipe coletar o máximo de aprendizado validado sobre os clientes com o mínimo de esforço. |
| **CRUD** | **Create, Read, Update, Delete**. As quatro operações básicas de persistência de dados. |
| **SPA** | **Single Page Application**. Aplicação web que carrega uma única página HTML e atualiza dinamicamente o conteúdo conforme o usuário interage. |
| **TLS** | **Transport Layer Security**. Protocolo criptográfico para fornecer comunicação segura pela rede, sucessor do SSL. |