AzShop – Documento de Decisões Arquiteturais

Daniel Maia

26 de fevereiro de 2026

1 Objetivo do Documento

Este documento visa registrar as principais **decisões arquiteturais** e **técnicas** do projeto **AzShop**, incluindo a escolha das tecnologias, versões, configurações de ambiente, banco de dados, estratégia de migrations, ORM/JPA, testes e convenções utilizadas. O objetivo é garantir rastreabilidade e clareza, evitando decisões informais e facilitando a reprodução do ambiente de desenvolvimento.

2 Visão Geral do Sistema

AzShop é um backend de e-commerce modular desenvolvido em Java com Spring Boot, focado em práticas avançadas de arquitetura, modelagem de domínio, JPA, segurança com JWT e boas práticas de engenharia de software.

2.1 Escopo (atual)

- API REST com Spring Web.
- Persistência com Spring Data JPA (Hibernate).
- Banco de dados PostgreSQL.
- Controle de schema com Flyway (migrations versionadas).
- Segurança com Spring Security (JWT planejado).

2.2 Fora de escopo (por enquanto)

- Fluxo completo de autenticação com JWT.
- Observabilidade (métricas, tracing, logs estruturados).
- CI/CD e deploy em cloud.
- Mensageria e processamento assíncrono.

3 Princípios e Metas de Arquitetura

- **Reprodutibilidade**: qualquer pessoa deve ser capaz de subir o banco e rodar a aplicação com passos claros.

- **Evolução controlada do schema**: mudanças no banco de dados devem ser explícitas e versionadas.
- **Separação de responsabilidades**: camadas de domínio, aplicação, infraestrutura e web devem ter limites claros.
- **Previsibilidade em produção**: evitar comportamentos inesperados, como alterações automáticas de schema pelo Hibernate.
- **Consistência**: decisões registradas de forma clara e repetível.

4 Stack Tecnológica e Versões

4.1 Tecnologias Principais

- **Java**: 21
- **Spring Boot**: 3.3.5
- **Build**: Maven
- **Web**: Spring MVC (starter-web)
- **Persistência**: Spring Data JPA + Hibernate
- **Banco**: PostgreSQL 15 (via Docker container)
- **Migrations**: Flyway
- **Segurança**: Spring Security (base)
- **Testes**: Spring Boot Test + JUnit + Mockito

4.2 Dependências (POM) – Referência

Abaixo está o POM do projeto, usado como referência.

Listing 1: pom.xml (referência do projeto)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                            https://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.3.5</version>
    <relativePath/>
  </parent>

  <groupId>com.daniel</groupId>
  <artifactId>azshop</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>azshop</name>
```

```
<description>
  Backend de e-commerce modular em Java com Spring Boot, desenvolvido
  para prtica avanada de arquitetura, modelagem de domnio, JPA,
  segurana com JWT e boas prticas de engenharia de software.
</description>

<properties>
  <java.version>21</java.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
  </dependency>

  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
  </dependency>

  <dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
  </dependency>

  <dependency>
```

```
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-database-postgresql</artifactId>
  </dependency>

  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.34</version>
    <optional>true</optional>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
  </dependency>
 </dependencies>

 <build>
  <plugins>
   <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
   </plugin>
  </plugins>
 </build>
</project>
```

5 Arquitetura Lógica (Visão por Camadas)

A arquitetura segue uma organização clássica por camadas, com planos de evoluir para um estilo mais explícito (como Clean Architecture) conforme o domínio cresce.

5.1 Camadas e Responsabilidades

- **Web (API)**: Controllers, DTOs, validação de entrada, mapeamentos HTTP.
- **Aplicação (Use Cases)**: Orquestra casos de uso, regras de fluxo e transações.
- **Domínio**: Entidades, value objects, invariantes, regras de negócio.
- **Infraestrutura**: Persistência (JPA), integrações e implementações técnicas.

6 Banco de Dados e Estratégia de Schema

6.1 Decisão Central

O schema do banco é controlado pelo **Flyway**, utilizando migrations SQL versionadas. O Hibernate/JPA é utilizado para:

- Mapear objetos para tabelas;
- Facilitar consultas e repositórios;
- Gerenciar o ciclo de vida das entidades e transações;
- Manter o domínio consistente com o banco controlado por migrations.

6.2 Por que usar Flyway e ainda usar ORM/JPA?

Flyway e ORM resolvem problemas distintos:

- **Flyway**: Como o banco evolui ao longo do tempo (controle de versão e reprodutibilidade)?
- **ORM/JPA**: Como trabalhamos com dados no código sem precisar escrever SQL diretamente?

  Na prática:

- Flyway gerencia a criação e alteração de tabelas, colunas, índices, e constraints.
- JPA/Hibernate lida com persistência e consultas de dados.

7 Configuração do Hibernate: `ddl-auto=validate`

A configuração `spring.jpa.hibernate.ddl-auto=validate` foi escolhida para:

- Impedir alterações automáticas do schema pelo Hibernate;
- Garantir que qualquer alteração estrutural no banco seja feita via migration;
- Forçar uma falha rápida caso as entidades não estejam consistentes com o banco.

8 Open Session in View (Open-In-View)

A configuração `spring.jpa.open-in-view=false` foi escolhida para:

- Impedir que a sessão do Hibernate fique aberta durante o processamento da resposta HTTP;
- Evitar disparos acidentais de queries na camada web (Lazy Loading indesejado);
- Garantir que o carregamento de dados aconteça na camada de serviço e dentro da transação.

9 Configuração Atual da Aplicação (`application.properties`)

Listing 2: application.properties

```
server.port=8080
```

```
spring.datasource.url=jdbc:postgresql://localhost:5433/azshop
spring.datasource.username=azshop
spring.datasource.password=azshop
spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.hibernate.ddl-auto=validate
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

spring.jpa.open-in-view=false
```

10 Ambiente Local com Docker Compose

O banco de dados é executado via Docker Compose para garantir reprodutibilidade e reduzir dependências manuais.

10.1 docker-compose.yml (estado atual)

Listing 3: docker-compose.yml

```
services:
  postgres:
    image: postgres:15-alpine
    container_name: azshop-db
    restart: unless-stopped
    environment:
      POSTGRES_DB: azshop
      POSTGRES_USER: azshop
      POSTGRES_PASSWORD: azshop
    ports:
      - "5433:5432"
    volumes:
      - azshop_postgres_data:/var/lib/postgresql/data

volumes:
  azshop_postgres_data:
```

11 Estratégia de Migrations (Flyway)

11.1 Estrutura sugerida no projeto

- `src/main/resources/db/migration`
- Arquivos no padrão `V{version}__descricao.sql`

12 Testes

12.1 Evolução planejada

- Testes de repositório (slice tests).
- Testes de controller com MockMvc.
- Integração com banco real (Docker Compose ou Testcontainers).

13 Decisões Registradas (Resumo)

ID	Decisão
AD-001	Spring Boot 3.3.5 e Java 21 como base do projeto.
AD-002	PostgreSQL como banco principal (Docker Compose).
AD-003	Controle de schema via Flyway (migrations SQL versionadas).
AD-004	`ddl-auto=validate` para impedir update automático de schema.
AD-005	`open-in-view=false` para evitar lazy loading acidental na camada web.
AD-006	Porta 5433 para evitar conflito com PostgreSQL local na 5432.

14 Como Rodar o Projeto (Passo a Passo)

14.1 1. Subir o Banco

```
docker compose up -d
```

14.2 2. Rodar a Aplicação

```
mvn spring-boot:run
```

15 Backlog Arquitetural (Próximas Decisões)

- Definir organização final dos pacotes (camadas / módulos).
- Definir padrão de DTOs e mapeamento (MapStruct ou manual).
- Definir estratégia de exceções e erros HTTP.
- Definir estratégia de autenticação JWT.
- Definir estratégia de logging estruturado e observabilidade.