AzShop – Architecture Document

Daniel Maia

26 de fevereiro de 2026

1 Objetivo do documento

Este documento registra, de forma objetiva e rastreável, as principais decisões arquiteturais e técnicas do projeto **AzShop**, incluindo stack, versões, configuração de ambiente, banco de dados, estratégia de migrations, ORM/JPA, testes e convenções. A intenção é reduzir ambiguidade, evitar decisões "na cabeça" e permitir que qualquer pessoa consiga reproduzir o ambiente e entender o *porquê* das escolhas.

2 Visão geral do sistema

AzShop é um backend de e-commerce modular desenvolvido em Java com Spring Boot. O projeto tem foco em prática avançada de arquitetura, modelagem de domínio, JPA, segurança com JWT e boas práticas de engenharia de software.

2.1 Escopo (neste momento)

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

2.2 Fora de escopo (por enquanto)

- JWT final e fluxo completo de autenticação/autorização.
- Observabilidade (metrics, tracing, logs estruturados).
- CI/CD e deploy em cloud.
- Mensageria e processamento assíncrono.

3 Princípios e metas de arquitetura

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

- **Evolução controlada do schema**: mudanças de banco devem ser explícitas e versionadas.
- **Separação de responsabilidades**: domínio, aplicação, infraestrutura e web com limites claros.
- **Previsibilidade em produção**: evitar comportamentos "mágicos" (ex.: Hibernate alterando schema automaticamente).
- **Consistência**: decisões registradas e repetíveis.

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 (container)
- **Migrations**: Flyway
- **Segurança**: Spring Security (base)
- **Testes**: Spring Boot Test + JUnit + Mockito

4.2 Dependências (POM) – referência

Abaixo está o POM utilizado como referência do estado atual do projeto (resumo fiel do que está em uso).

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 atual segue uma organização clássica por camadas, com intenção de evoluir para um estilo mais explícito de arquitetura (ex.: Clean Architecture) conforme o domínio crescer.

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 centrais.

- **Infraestrutura**: persistência (JPA), integrações, implementações técnicas.

6 Banco de dados e estratégia de schema

6.1 Decisão central

O schema do banco é controlado por **Flyway** através de **migrations SQL versionadas**. O Hibernate/JPA é utilizado para:

- mapear objetos → tabelas;
- facilitar queries e repositórios;
- gerenciar o ciclo de vida das entidades e transações;
- manter o domínio em código consistente com a estrutura já definida por migrations.

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

Flyway e ORM resolvem problemas diferentes:

- **Flyway**: "Como o banco muda ao longo do tempo?" (evolução controlada, auditável, reproduzível).
- **ORM/JPA**: "Como eu trabalho com dados no código sem escrever SQL o tempo todo?" (modelagem e persistência).

  Na prática:

- Flyway cria/alterar tabelas, colunas, índices, constraints (DDL).
- JPA/Hibernate persistem/consultam dados com entidades e repositórios (DML), respeitando o schema existente.

6.3 Configuração do Hibernate: ddl-auto=validate

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

- impedir que o Hibernate modifique schema automaticamente;
- falhar rápido se as entidades não batem com o banco;
- forçar que qualquer alteração estrutural seja feita via migration.

  Observação importante: `update` é evitado porque pode:

- gerar alterações implícitas e difíceis de rastrear;
- criar divergência entre ambientes;
- introduzir comportamento inesperado em produção.

7 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 queries "sem querer" disparadas na camada web (Lazy Loading fora de lugar);

- forçar que o carregamento de dados aconteça dentro da camada de serviço e dentro da transação.

7.1 Impacto prático

Com Open-In-View desligado, se um controller tentar acessar uma relação lazy depois que a transação acabou, você vai ver erro de lazy initialization. Isso é intencional: significa "você deveria ter buscado isso antes, no service, com o fetch certo".

8 Configuração atual da aplicação (application.properties)

Listing 2: application.properties (estado atual)

```
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
```

9 Ambiente local com Docker Compose

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

9.1 docker-compose.yml (estado atual)

Listing 3: docker-compose.yml (estado atual)

```
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:
```

9.2 Decisão: por que porta 5433?

A aplicação usa `localhost:5433` para evitar conflito com instalações locais existentes do PostgreSQL na porta padrão 5432.

10 Estratégia de migrations (Flyway)

10.1 Estrutura sugerida no projeto

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

  Exemplos:

- `V1__create_tables.sql`
- `V2__add_indexes.sql`
- `V3__create_users.sql`

10.2 Regras práticas

- Uma migration aplicada em um ambiente **não deve ser editada**. Se precisar mudar, crie outra migration.
- Alterações estruturais sempre via migration (nada de "o Hibernate ajusta").
- Se for necessário dado inicial, usar migrations específicas para seed controlado.

11 Testes

11.1 Estado atual

- **Framework**: Spring Boot Starter Test.
- **Objetivo imediato**: garantir que o contexto sobe e que os componentes principais inicializam.

11.2 Decisão (registrada)

O banco é executado via Docker Compose localmente. Isso garante que o ambiente de desenvolvimento se aproxime do comportamento real do PostgreSQL (diferenças em relação a H2 são frequentes).

11.3 Evolução planejada

- Testes de repositório (slice tests).
- Testes de controller com MockMvc.
- Integração com banco real (Docker Compose ou Testcontainers, a depender da maturidade do projeto).

12 Segurança

12.1 Estado atual

- Base de segurança ativada com `spring-boot-starter-security`.
- JWT planejado para implementação.

12.2 Direção

A intenção é evoluir para:

- autenticação stateless com JWT;
- controle de acesso por roles/perfis;
- endpoints públicos vs protegidos;
- boas práticas (password hashing, refresh token, blacklist se necessário).

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
```

14.3 3. Verificar

- Aplicação: `http://localhost:8080`
- Banco: `localhost:5433` (PostgreSQL dentro do container)

15 Padrão de evolução deste documento

- Sempre que uma decisão técnica importante for tomada, ela deve aparecer na seção de decisões (AD-xxx).
- Se o projeto crescer, as decisões podem ser extraídas para ADRs individuais (um arquivo por decisão).

16 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 (Problem Details / RFC 7807).
- Definir estratégia de autenticação JWT (access/refresh, rotação, expiração).
- Definir estratégia de testes de integração (Compose vs Testcontainers).
- Definir logging estruturado e observabilidade.