RELATÓRIO TÉCNICO DO SISTEMA JAVA EAST

Autores:
Estevão Costa Oliveira – RM 368353
Everton Nobrega Barbosa – RM 368760
Luis Fernando Silva do Nascimento – RM 367853
Lucas Novaes Vieira – RM 367795
Irving Lucas da Silva Melo – RM 367546

Data de entrega: 20/01/2026

Sumário

---

1 INTRODUÇÃO

Este relatório técnico tem como objetivo documentar o desenvolvimento do sistema **JavaEats**, apresentando sua proposta, arquitetura, endpoints da API e instruções para configuração e execução. O documento segue os padrões acadêmicos estabelecidos pela Associação Brasileira de Normas Técnicas (ABNT), visando clareza, organização e padronização das informações apresentadas.

---

2 DESCRIÇÃO DETALHADA DO PROJETO

Um grupo de restaurantes decidiu contratar estudantes para construir um sistema de gestão compartilhada para seus estabelecimentos. Essa decisão foi motivada pelo alto custo de sistemas individuais, o que levou os restaurantes a se unirem para desenvolver uma solução única e centralizada.

O sistema permite que os clientes escolham restaurantes com base na comida oferecida, e não na qualidade ou complexidade do sistema de gestão utilizado, promovendo uma experiência mais justa e padronizada entre os estabelecimentos participantes.

---

3 OBJETIVO DO PROJETO

O objetivo do projeto é desenvolver um sistema robusto que permita aos restaurantes gerenciar suas operações de forma eficiente, enquanto os clientes poderão consultar informações, realizar pedidos online e registrar avaliações sobre os estabelecimentos e serviços oferecidos.

---

4 ARQUITETURA DO SISTEMA

O sistema **JavaEats** adota os princípios da **Clean Architecture**, visando a separação de responsabilidades, facilidade de manutenção e independência de frameworks e tecnologias externas.

A arquitetura está organizada em camadas bem definidas, garantindo que as regras de negócio não dependam de detalhes de infraestrutura.

4.1 Camada Core

A camada **Core** representa o núcleo da aplicação e é responsável por concentrar as regras de negócio. Nesta camada encontram-se:

- Domínios, que representam as entidades centrais do sistema;
- Casos de uso (Use Cases), responsáveis por encapsular a lógica de negócio e orquestrar as operações do sistema.

Essa camada não possui dependências diretas com frameworks, banco de dados ou mecanismos de entrega.

4.2 Camada de Infraestrutura

A camada de **Infraestrutura** é responsável por lidar com os detalhes técnicos da aplicação, incluindo:

- Controllers, responsáveis por expor os endpoints da API REST;
- Persistência de dados utilizando MongoDB, por meio do Spring Data;
- Configurações do projeto, incluindo inicialização da aplicação, uso do Spring Actuator e demais componentes técnicos.

Essa camada depende da camada Core, respeitando o princípio de inversão de dependência.

---

5 STACK TECNOLÓGICA

A seguir são apresentadas as principais tecnologias utilizadas no desenvolvimento do sistema:

- Linguagem de programação: Java 21
- Framework: Spring Boot
- Persistência de dados: Spring Data
- Banco de dados: MongoDB
- Empacotamento da aplicação: JAR (Maven)
- Orquestração de serviços: Docker Compose
- Observabilidade: Spring Actuator

---

6 OBSERVABILIDADE E MONITORAMENTO

O sistema utiliza o **Spring Actuator** para expor informações operacionais da aplicação, incluindo o endpoint de verificação de saúde (*health check*), permitindo o monitoramento do estado da aplicação e facilitando diagnósticos em ambientes de execução.

---

7 ENDPOINTS DA API

A Tabela 1 apresenta os principais endpoints da API do sistema **JavaEats**, incluindo o método HTTP, o endpoint e uma breve descrição de sua funcionalidade.

Tabela 1 – Endpoints da API JavaEats

Método HTTP	Endpoint	Descrição
POST	/addresses	Realiza o cadastro de um novo endereço no sistema.
DELETE	/addresses/{addressId}	Remove um endereço previamente cadastrado.
POST	/users	Realiza o cadastro de um novo usuário na plataforma.
GET	/users/{userId}	Retorna os dados de um usuário específico.
PUT	/users/{userId}	Atualiza as informações de um usuário existente.
DELETE	/users/{userId}	Remove um usuário do sistema.
POST	/restaurants	Realiza o cadastro de um novo restaurante.
GET	/restaurants	Retorna a lista de restaurantes cadastrados.
GET	/restaurants/{restaurantId}	Retorna os dados detalhados de um restaurante específico.
PUT	/restaurants/{restaurantId}	Atualiza as informações de um restaurante.
DELETE	/restaurants/{restaurantId}	Remove um restaurante do sistema.
POST	/restaurants/{restaurantId}/menus	Cria um novo cardápio associado a um restaurante.
GET	/restaurants/{restaurantId}/menus	Lista os cardápios de um restaurante.
PUT	/restaurants/{restaurantId}/menus/{menuId}	Atualiza um cardápio existente.

DELETE	/restaurants/{restaurantId}/menus/{menuId}	Remove um cardápio de um restaurante.
POST	/user-addresses	Cria a relação entre um usuário e um endereço.
GET	/user-addresses	Lista todas as relações entre usuários e endereços.
PUT	/user-addresses/{userAddressId}	Atualiza uma relação entre usuário e endereço.
DELETE	/user-addresses/{userAddressId}	Remove a relação entre usuário e endereço.

Endereço do Swagger:
http://localhost:8081/swagger-ui/index.html

---

8 CONFIGURAÇÃO E EXECUÇÃO DO SISTEMA

8.1 Pré-requisitos

Para execução do sistema é necessário:

- Docker Engine instalado e em execução;
- Docker Compose disponível;
- Acesso ao terminal;
- Estar posicionado na raiz do projeto, onde se encontra o arquivo `docker-compose.yml`.

8.2 Execução

Após atender aos pré-requisitos, o sistema pode ser iniciado por meio do seguinte comando:

```
docker compose up -d
```

Esse comando realiza a inicialização dos containers necessários para o funcionamento da aplicação, incluindo o banco de dados MongoDB e o backend desenvolvido em Spring Boot.