UNIFACS - UNIVERSIDADE SALVADOR
SISTEMAS DE INFORMAÇÃO

GLENDA SOUZA FERNANDES DOS SANTOS
MARCUS VINICIUS LAMEU LIMA
BRUNO DE MENEZES SALES
DIEGO AMARO FERREIRA

RELATÓRIO DA APLICAÇÃO PARA A REDE DE LOJAS DE ELETRÔNICOS

Salvador/BA
2024

GLENDA SOUZA FERNANDES DOS SANTOS
MARCUS LAMEU LIMA
BRUNO DE MENEZES SALES
DIEGO AMARO FERREIRA

RELATÓRIO DA APLICAÇÃO PARA A REDE DE LOJAS DE ELETRÔNICOS

Trabalho apresentado à disciplina de Sistemas Distribuídos integrado ao curso Sistema de Informação pela Universidade de Salvador - Campus Tancredo Neves, como requisito parcial para a obtenção de nota da A3.

Orientador: Profº Adailton Júnior e Profº Marivaldo Santos.

Salvador

2024

SUMÁRIO

INTRODUÇÃO

A proposta deste trabalho consiste em criar uma aplicação que simule a captação de dados de venda de uma rede de loja de eletrônicos aplicando os conhecimentos desenvolvidos durante a Unidade Curricular vigente.

Concluímos que a melhor forma de atender aos requisitos do projeto e maximizar os benefícios seria construir uma API utilizando Spring Boot, tendo o MySQL como sistema gerenciador de banco de dados, conteinerizada com Docker e sendo consumida através do software Postman. Optamos por uma arquitetura de microserviços devido à maior independência e estabilidade entre os segmentos da aplicação, tolerância a falhas, fácil integração e atualização do código [3, 8.].

Iniciamos com a construção de entidades básicas para o funcionamento da aplicação, alinhada à construção do banco de dados. Decidimos dividir a aplicação em duas APIs, que não estão inter-relacionadas entre si e se comunicam somente com o banco de dados. Temos a API REST, que faz o gerenciamento das classes de cliente, produto, venda, vendedor e outras classes utilizadas para correlacionar essas classes principais. Esta é a única API que pode fazer inserção no banco de dados. Implementamos nela o Spring Security e a autenticação JWT [5] para garantir mais segurança na inserção dos dados e na confirmação de dados no login. A segunda API é destinada à geração de relatórios estatísticos da aplicação; ela realiza apenas consultas no banco de dados, sem autorização para realizar inserções.

Consideramos que a construção da aplicação desta forma atende a todos os requisitos solicitados no projeto e também implementa inovações que aprimoram a aplicação, como a implementação do Spring Security e a autenticação JWT, visando maior segurança. A escolha de utilizar o Spring Boot permite focar mais no desenvolvimento das funcionalidades da aplicação, enquanto a utilização do Docker possibilita a conteinerização, que, entre outras vantagens, traz portabilidade à aplicação, além da rapidez na criação e execução dos contêineres [7]. Nos tópicos subsequentes, apresentaremos um detalhamento das versões de software e tecnologias empregadas na implementação do sistema. Além disso, discutiremos o progresso atual do desenvolvimento, os impactos observados até o momento e as conclusões preliminares obtidas a partir do projeto.

FUNDAMENTAÇÃO TEÓRICA

A aplicação consiste em um sistema de serviço que gerencia os clientes, vendedores, compras, produtos, supervisores e relatórios estatísticos utilizando os seguintes softwares e linguagens para estruturação e composição da arquitetura da aplicação:

Para o armazenamento em banco de dados, foi utilizado o *MySQL* (Sistema Gerenciador de Banco de Dados) na versão 8.0. Tal escolha considerou a sua estrutura baseada na linguagem *SQL*, que o caracteriza como um banco de dados relacional. Essa característica traz diversos benefícios para a organização e estruturação dos dados na forma de tabelas bem definidas. O *MySQL* também apresenta um desempenho eficaz em estruturas cliente/servidor, sendo um servidor *multi-threads* [1]. Ressaltando as características de portabilidade, o fato de ser um sistema em constante aprimoramento e atualização contribui para sua eficiência e adaptabilidade [2].

A linguagem de programação utilizada foi o *Java* (versão 21), por meio do framework *Spring Boot*, considerando sua melhor adequação ao projeto e ao escopo proposto. Optamos por uma arquitetura baseada em microserviços devido à sua facilidade de atualização, permitindo, a longo prazo, o crescimento e o desenvolvimento do projeto de maneira mais simplificada. Além disso, a tolerância a falhas contribui para a confiabilidade da aplicação [3]. Atrelada ao uso do *Spring Boot* que proporciona um modelo de autoconfiguração [4], permite um maior foco no desenvolvimento das funcionalidades da aplicação, trazendo autonomia para a implementação, operação e escalonamento do sistema [8], bem como a implementação de mecanismos de segurança, como o Spring Security e a autenticação JWT [5].

Para a interface de comunicação que auxilia na inserção e deleção de dados, bem como no controle de login, utilizamos o software *Postman* [6]. Este é amplamente empregado para testar e analisar *APIs*, possuindo diversas funcionalidades. Entre as funcionalidades utilizadas em nosso projeto estão as requisições *HTTP (GET, POST, UPDATE e DELETE*).

Toda a aplicação foi conteinerizada utilizando o software *Docker*. A conteinerização permite que a aplicação seja executada em qualquer ambiente, garantindo portabilidade e consistência entre as diferentes plataformas [7].

PLANO DE IMPLANTAÇÃO

Para a construção do software decidimos dividir a aplicação em duas API 's, uma ficaria responsável pela criação e gerenciamento dos clientes, vendedores, produtos e etc., chamamos de API principal, e a outra serve para a geração de relatórios estatísticos. Na figura 1 e 2, ilustramos a arquitetura da API principal:

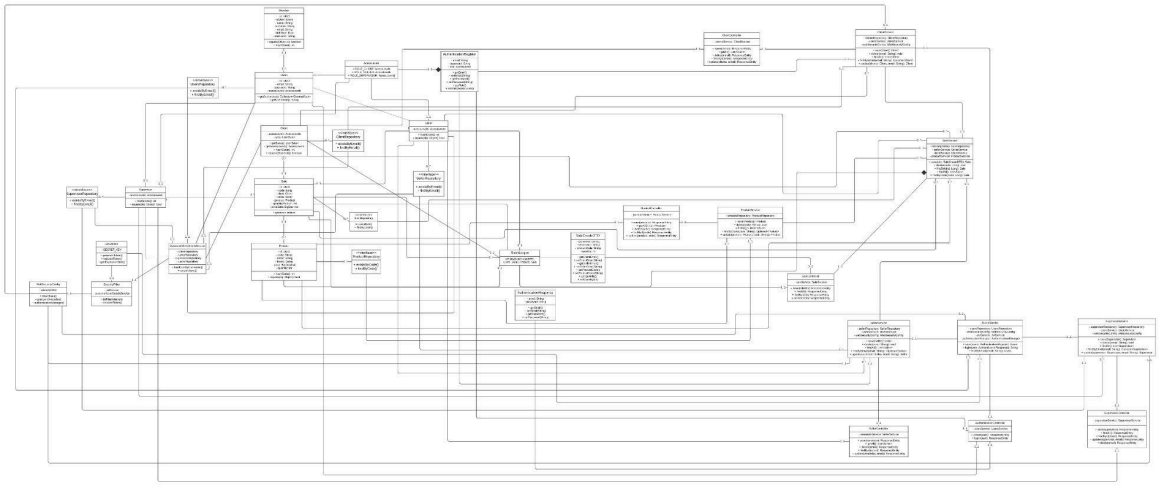

Figura 1: Diagrama UML da aplicação principal

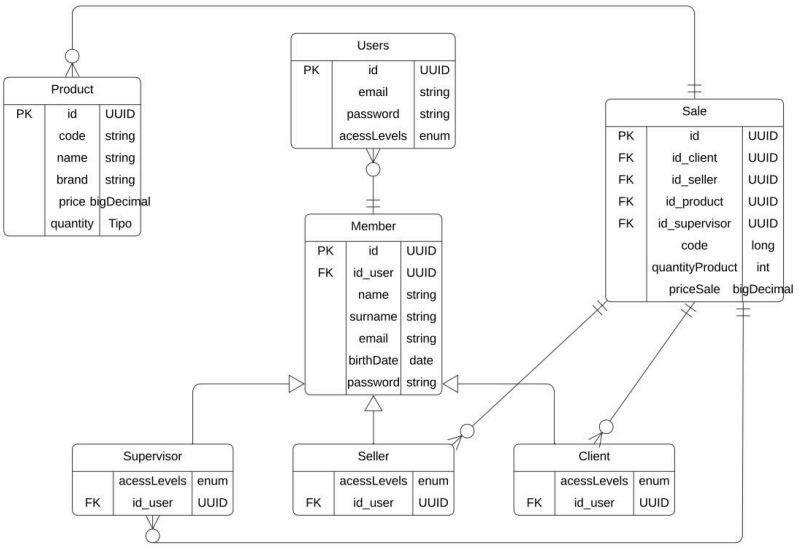

Figura 2: Diagrama de Relacionamento e Entidades

Passo a Passo para iniciar a aplicação

Antes de iniciar a aplicação, você deve conferir se a sua máquina está equipada com o Java em sua versão 21, o mySQL versão 8.0 e o docker instaladas, senão abaixo estão alguns links úteis:
JAVA 21.
Download MySQL.
Download Docker.

Após fazer as configurações necessárias no seu sistema, faça a clonagem do repositório git do projeto para a sua máquina, se for optar por usar o VSCode como IDE para visualizar o projeto, instale as extensões abaixo:

Extension Pack for Spring Boot: `vmware.vscode-boot-dev-pack`
Extension Pack for Java: `vscjava.vscode-java-pack`
Docker: `ms-azuretools.vscode-docker`

Acima temos o nome das extensões e ao lado o seu identificador, depois de instalar as extensões e clonar o repositório, certifique-se de que o software Docker está ativo. Antes de iniciar a aplicação, é necessário criar um banco de dados no MySQL chamado db_venda na porta 3306, no localhost. Configure seus usuário e senha no arquivo `src\main\resources\application.properties` A seguir, torne os scripts shell executáveis com o comando `chmod +x` e execute a aplicação com o comando `./start.sh`.
Esse script quando acionado tem a função de fazer o build do projeto e subir os containers.
Quando o projeto é iniciado, alguns cadastros iniciais, como produtos, vendedores e clientes, são realizados automaticamente. No entanto, utilizando o Postman, é possível adicionar vendas adicionais acessando o endpoint: `http://localhost:8080/sale/register`. Isso permite a geração de relatórios mais completos com base nos dados das vendas adicionadas.
A documentação completa dos Endpoints da aplicação pode ser visualizada aqui.

Regras de Negócio

Autenticação

Regra:
• Apenas usuários com os papéis SUPERVISOR ou SELLER podem acessar a rota POST /auth.
• Qualquer usuário pode acessar a rota POST /auth/login para efetuar login.

Validação:
• Verificar se o usuário possui os papéis necessários (SUPERVISOR ou SELLER) antes de permitir o acesso à rota /auth.
• Acesso à rota /auth/login é permitido sem autenticação.

Gestão de Clientes

Regra:
• Apenas usuários com o papel SELLER podem criar novos clientes usando a rota POST /client.
• Apenas usuários com os papéis SUPERVISOR ou SELLER podem acessar ou modificar informações de clientes em qualquer rota que comece com /client/**.

Validação:
• Verificar se o usuário autenticado possui o papel SELLER antes de permitir a criação de clientes.

• Verificar se o usuário autenticado possui os papéis SUPERVISOR ou SELLER antes de permitir o acesso às rotas relacionadas a clientes.

Gestão de Vendedores

Regra:
• Apenas usuários com o papel SUPERVISOR podem acessar qualquer rota relacionada a vendedores (/seller/**).

Validação:
• Verificar se o usuário autenticado possui o papel SUPERVISOR antes de permitir o acesso às funcionalidades relacionadas a vendedores.

Gestão de Vendas

Regra:
• Apenas usuários com o papel SUPERVISOR podem excluir vendas pela rota DELETE /sale/{id}.
• Usuários com os papéis SUPERVISOR ou SELLER podem acessar ou realizar outras operações relacionadas às vendas em qualquer rota que comece com /sale/**.

Validação:
• Verificar o papel do usuário antes de permitir a exclusão de vendas.
• Garantir que o usuário autenticado tenha os papéis necessários (SUPERVISOR ou SELLER) para outras operações relacionadas às vendas.

Gestão de Produtos

Regra:
• Usuários com os papéis CLIENT, SUPERVISOR ou SELLER podem consultar a lista de produtos na rota GET /product.
• Apenas usuários com os papéis SUPERVISOR ou SELLER podem acessar ou modificar informações de produtos em qualquer rota que comece com /product/**.

Validação:
• Garantir que o usuário autenticado possua os papéis necessários para consultar ou manipular informações relacionadas aos produtos.

Requisições gerais

Regra:

• Qualquer requisição que não corresponda às rotas especificadas no código deve ser feita por usuários autenticados.

Validação:
• Verificar se o usuário está autenticado antes de processar requisições sem regras explícitas.

Segurança

Regra:
• A aplicação utiliza autenticação baseada em JWT, desabilitando sessões e autenticação de formulário ou HTTP Basic.
• As requisições são processadas no modo "stateless" (sem armazenamento de estado no servidor).
• Um filtro de segurança (SecurityFilter) é usado para validar o token JWT em cada requisição.

Validação:
• Verificar a validade do token JWT em todas as requisições protegidas.
• Rejeitar requisições com tokens inválidos ou ausentes.

CONSIDERAÇÕES FINAIS

Em suma, a proposta deste trabalho foi devidamente concretizada ao construir uma aplicação que simula a captação de dados de vendas de uma rede de lojas de eletrônicos, aplicando os conhecimentos adquiridos durante a Unidade Curricular vigente. A utilização do Spring Boot, MySQL, Docker e Postman, aliada à arquitetura de microserviços, mostrou-se uma abordagem eficaz para atender aos requisitos do projeto, proporcionando benefícios significativos como a independência, estabilidade, tolerância a falhas, e facilidade de integração e atualização do código.

Dividindo a aplicação em duas APIs interdependentes, uma dedicada à inserção e gerenciamento de dados com segurança através de Spring Security e autenticação JWT, e outra focada na geração de relatórios estatísticos, conseguimos criar uma solução robusta e segura. A adoção do Docker para conteinerização assegurou a portabilidade e a rapidez na criação e execução dos contêineres, facilitando o desenvolvimento e a manutenção da aplicação.

Os resultados obtidos até o momento demonstram que esta abordagem não só atende às expectativas iniciais, mas também estabelece uma base sólida para futuras expansões e

aprimoramentos, como a implementação de um *Frontend* e implementação de um *gateway* para as requisições do *Frontend*. Continuaremos monitorando o progresso do desenvolvimento, avaliando o impacto das tecnologias empregadas e refinando a aplicação conforme necessário para garantir seu sucesso a longo prazo.

BIBLIOGRAFIA

O que é o MySQL? Disponível em: <https://www.oracle.com/br/mysql/what-is-mysql/>.

SOUZA, Elaine Calasans; DE OLIVEIRA, Marcus Rogério. Comparativo entre os bancos de dados mysql e mongodb: quando o mongodb é indicado para o desenvolvimento de uma aplicaçao. **Revista Interface Tecnológica**, v. 16, n. 2, p. 38-48, 2019.

DIO. Todo o poder dos Microsserviços com Spring Boot e Spring Cloud em Arquiteturas Java. Disponível em: <https://www.dio.me/articles/todo-o-poder-dos-microservicos-com-spring-boot-e-spring-cloud-em-arquiteturas-java>.

O que é Java Spring Boot? | IBM. Disponível em: <https://www.ibm.com/br-pt/topics/java-spring-boot>.

IBM Sterling Order Management System. Disponível em: <https://www.ibm.com/docs/pt-br/order-management?topic=users-jwt-authentication>.

O que é o Postman? Disponível em: <https://enotas.com.br/blog/postman/>.

O que é o Docker? | IBM. Disponível em: <https://www.ibm.com/br-pt/topics/docker>.

O que são microsserviços? | AWS. Disponível em: <https://aws.amazon.com/pt/microservices/>.

Classe UML: Lucidchart. Disponível em: <https://lucid.app/lucidchart/2aee018a-ea0b-4cd1-900b-6e3df459e98b/edit?viewport_loc=22

99%2C1153%2C2994%2C1452%2CHWEp-vi-RSFO&invitationId=inv_5340d0c2-0e61-43fa-821a-203702215dc2>.

DER do banco de dados: Lucidchart. Disponível em: <https://lucid.app/lucidchart/d3e97e41-9c53-4f9a-8ee7-bc1cabf313d1/edit?viewport_loc=31%2C49%2C2032%2C986%2C0_0&invitationId=inv_b2b924fc-6803-4097-8ee7-45e880cda65c>.