Equipe:
Zanoni Campos Fernandes
Marcelo Miranda Cavalcanti
Pedro Lima Vergne
João Anísio Guimarães Nilo Dantas

PROJETO PARA AVALIAÇÃO A3
SISTEMAS DISTRIBUÍDOS E MOBILE

Professores:
Adailton de Jesus Cerqueira Jr., M. Sc.
Marivaldo Pereira dos Santos
Unidade Curricular - Sistemas Distribuídos e Mobile (2024/2)

Brasil
2024

RELATÓRIO DE PROJETO PARA AVALIAÇÃO A3

SISTEMAS DISTRIBUÍDOS E MOBILE

Zanoni Campos Fernandes - RA 12723131956
Pedro Lima Vergne - RA 12724217499
João Anísio Guimarães Nilo Dantas - RA 12723213518
Marcelo Miranda Cavalcanti - RA 1272313678[1]

1. INTRODUÇÃO

Trata-se de sistema desenvolvido na qualidade de trabalho prático no âmbito da avaliação A3 da Unidade Curricular (UC) Sistemas Distribuídos e Mobile, do 2º semestre do ano de 2024 (2024/2), da Universidade Salvador (UNIFACS), ministrada pelos professores Adailton de Jesus Cerqueira Jr. e Marivaldo Pereira Santos.

O sistema consiste parte em 02 (duas) APIs (*Application Programming Interface*), desenvolvidas em JavaScript (ECMAScript) que fazem o papel de *back-end* de uma rede de farmácias do tipo *drugstore*, sendo: i) 01 (uma) responsável por gerenciar (denominada "api_gerenciamento") o cadastro de clientes, de vendedores, de itens (produtos), de estoque e de vendas dessa cadeia de lojas; e ii) 01 (uma) responsável por gerar relatórios e munir a gestão de dados para análise de negócios (denominada "api_relatorio").

O tema para desenvolvimento do sistema - farmácia do tipo *drugstore* - surgiu naturalmente, em uma das primeiras reuniões da equipe, ainda na fase de modelagem para atendimento dos requisitos passados na descrição do trabalho. Em determinado momento, no âmbito de discussão sobre a modelagem dos objetos de software e do banco para persistência de dados, um dos integrantes deu um exemplo de como o sistema funcionaria utilizando do tema de farmácia *drugstore*, enquanto que o restante da equipe, além de ter compreendido o exemplo, aceitou o tema de forma natural, criando-se o entendimento comum de que se trabalharia com este dali em diante.

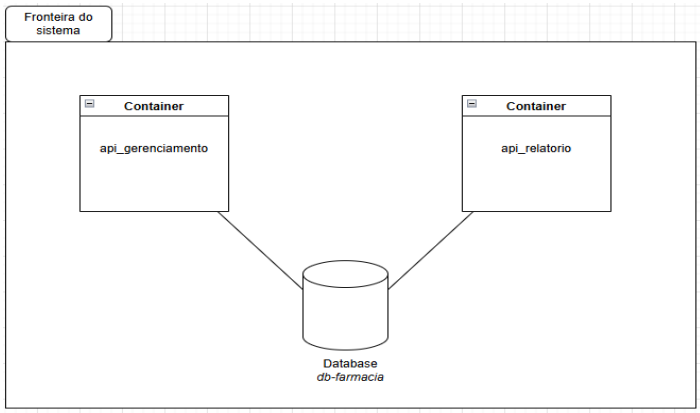
Além das APIs informadas, o sistema também conta com um banco de dados, do tipo relacional-estruturado (SQL), rodando em Oracle MySQL como SGBD (Sistema Gerenciador de Banco de Dados).

[1] Acadêmicos da Universidade Salvador (UNIFACS), campus Tancredo Neves (TCN), integrante da rede Ânima Educação. Relatório apresentado como requisito para atribuição de nota na Avaliação (A3) da Unidade Curricular (UC) Sistemas Distribuídos e Mobile, do semestre letivo 2024.2, da Universidade Salvador (UNIFACS), integrante da rede Ânima Educação. 2024. Professores: Adailton de Jesus Cerqueira Jr., M.Sc., e Marivaldo Pereira dos Santos

2. FUNDAMENTAÇÃO TEÓRICA

No conceito mais amplo, a arquitetura básica é do tipo "microsserviços"[2], com uma relativização, eis que ambas as APIs apresentadas acima utilizam do mesmo banco de dados. Enquanto que a "api_gerenciamento" manipula integralmente as tabelas, realizando o CRUD (*create*, *read*, *update* e *delete*) dos dados persistidos em banco de dados, a "api_relatorio" somente realiza a opção de leitura (*read*) dos dados guardados, para geração de relatórios estatísticos que apoiam a gerência e gestão nas tomadas de decisão do negócio, conforme descrito na requisição do sistema. Segue auxílio visual (figura 2.1) demonstrando a arquitetura do sistema - elaborado na ferramenta *online* draw.io[3]:

figura 2.1



Em menor granularidade, o sistema apresenta uma arquitetura de cliente-servidor, conforme classificado por Giocondo Gallotti[4], sendo cada API um cliente potencial do serviço de persistência de dados fornecido pelo banco de dados relacional. Ao mesmo tempo, as APIs fornecem serviços para *clients front-end* ou outras aplicações com acesso a elas – com uma observação: a camada de interface do usuário foi abstraída por não ser requisito do trabalho, permitindo assim eventual desenvolvimento posterior, caso seja necessário.

---

[2] "(...) devido à sua natureza de finalidade única, o tamanho do serviço nos microsserviços é muito menor que em outras arquiteturas distribuídas (...). Os microsserviços formam uma arquitetura distribuída: cada serviço roda em seu próprio processo, que originalmente implicava um computador físico, mas rapidamente evoluiu para máquinas virtuais e contêineres". NEAL, Richards, Mark, F. **Fundamentos de arquitetura de software: uma abordagem de engenharia**. págs. 94 e 95.

[3] Disponível em: https://github.com/facs-distribuido-mobile/A3/blob/main/relatorio/Arquitetura.drawio.pdf

[4] "Em uma arquitetura cliente-servidor, a funcionalidade do sistema está organizada em serviços - cada serviço é prestado por um servidor. Os clientes são os usuários desses serviços e acessam os servidores para fazer uso deles" (...) "É usado quando os dados em um banco de dados compartilhado precisam ser acessados a partir de uma série de locais. Como os servidores podem ser replicados, também pode ser usado quando a carga em um sistema é variável". GALLOTI, Giocondo Marino Antonio (org.). **Arquitetura de software**. página 22.

A comunicação com as APIs é feita por meio do protocolo HTTP e o *backend* utiliza o padrão REST (ainda que não RESTful), com recursos e dados acessíveis em URIs (*Uniform Resource Identifiers*), desenhado especificamente para trabalhar com requisições e respostas em JSON (*JavaScript Object Notation*).

Finalmente, tal como requisitado, cada um dos processos de software (APIs e banco de dados) foi colocado em um container Docker[5] para execução em separado, permitindo modularidade, flexibilidade e escalabilidade do sistema de forma distribuída.

3. ESPECIFICAÇÕES TÉCNICAS

Para construção e execução do sistema foram necessárias as seguintes tecnologias:

- **Docker (v20.18.1):** responsável por rodar os container específicos com a base de dados e cada uma das APIs;
- **Postman (v.11.21.0) ou Insomnia (v.2022.4.0)** – ou outra ferramenta de preferência: utilizados em substituição ao *front-end*, que foi abstraído, e necessários para teste e desenvolvimento das aplicações;
- **JavaScript (ECMAScript):** linguagem de programação interpretada de alto nível utilizada na construção das APIs;
- **Node.js (v.23.1.0):** software para ambiente de execução JavaScript que suporta o *back-end*, desvinculando a execução do script de um *client front-end*;
- **Git (v.2.47.1.windows.1):** software para controle de versionamento do sistema, utilizando em conjunto com o repositório em nuvem/central público hospedado no GitHub;
- **SQL:** linguagem de manipulação de banco de dados utilizada como fundamento do SGBD (Sistema Gerenciador de Banco de Dados);
- **MySQL:** SGBD (Sistema Gerenciador de Banco de Dados) desenvolvido pela Oracle e escolhido pela equipe como responsável por controlar a base de dados da aplicação;
- **NPM (Node Package Manager):** repositório de bibliotecas públicas e gerenciador de pacotes para JavaScript/Node;

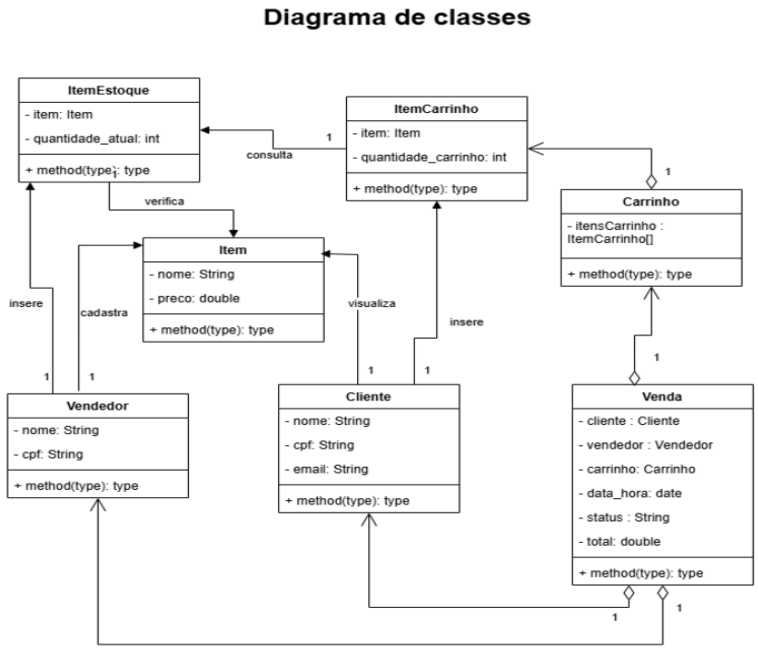
[5] "Docker Engine is an open source containerization technology for building and containerizing your applications. Docker Engine acts as a client-server application (...)". **DockerDocs**.

- **Nodemon (v^3.1.7):** biblioteca de suporte utilizada apenas durante o desenvolvimento, responsável por reinicializar a aplicação em Node.js cada vez que o repositório era modificado;
- **MySQL driver (v^2.18.1):** driver de suporte para utilização em aplicações Node.js que possuem MySQL como SGBD;
- **Express (v^4.21.1):** framework de suporte para criação de servidores de aplicações que trabalham com requisições HTTP e rodam em Node.js;
- **Consign (v^0.1.6):** biblioteca de suporte utilizada para carregar as rotas da aplicação de forma simplificada, melhorando a separação e a legibilidade do código;

4. PROJETO DE IMPLEMENTAÇÃO

No tocante ao projeto de implementação, o desenho básico dos objetos de *software* para a API de gerenciamento foi modelado de acordo com o seguinte diagrama UML de classes (figura 4.1) - elaborado na ferramenta *online* draw.io[6]:

figura 4.1



Oportuno observar que, inobstante a referida modelagem, a programação foi realizada inteiramente em JavaScript e não implementou orientação a objetos propriamente dita. Assim,

[6] Disponível em: https://github.com/facs-distribuido-mobile/A3/blob/main/relatorio/Diagrama%20de%20classe%20-%20Dist%20e%20Mobile.pdf

a modelagem das classes foi utilizada como referência para abertura de objetos em JavaScript (denominados "*models"* no projeto), de forma a guiar leitura e manipulação das informações guardadas na base de dados. São componentes básicos do sistema: i) Cliente; ii) Vendedor; iii) Itens; iv) ItemEstoque; v) Venda; vi) VendaDetalhe.

Sobre o funcionamento do sistema em si, temos que os cadastros de clientes, vendedores e itens são os mais fundamentais e operam de forma quase independente entre eles. Os clientes possuem os atributos de nome, CPF e e-mail. Os vendedores possuem os atributos de nome e CPF. Os itens possuem os atributos de nome e preço.

Na sequência, temos o cadastro de um item em estoque, que necessita da existência de item registrado previamente, sendo obrigatório identificar o item pelo id na hora de inserir uma quantidade deste item em estoque. Os itens em estoque possuem os atributos de id_item (chave estrangeira) e de quantidade atual em estoque.

Para realizar uma venda, o vendedor precisa inserir os itens (produtos) em uma listagem de itens, sendo que todo produto inserido no "carrinho" é verificado no estoque em termos de existência e quantidade atual em estoque compatível com o solicitado para venda. Após o encerramento do "carrinho", os demais dados de venda são inseridos (id do vendedor, id do cliente e status) sendo que os atributos restantes (total da venda e data e hora da venda) são automaticamente calculados pelo sistema - data e hora do cadastro da transação e o total da venda é calculado em um laço de repetição que analisa cada item do carrinho, multiplicando seu valor pela quantidade solicitada.

A venda é registrada em banco de dados, gerando um id de venda, atributo este que é utilizado no registro dos itens do "carrinho" em uma tabela específica do banco, denominada "vendas_detalhes", recebendo a chave estrangeira em questão que identifica unicamente a transação em que tal item foi vendido, a quantidade total solicitada daquele item e o subpreço.

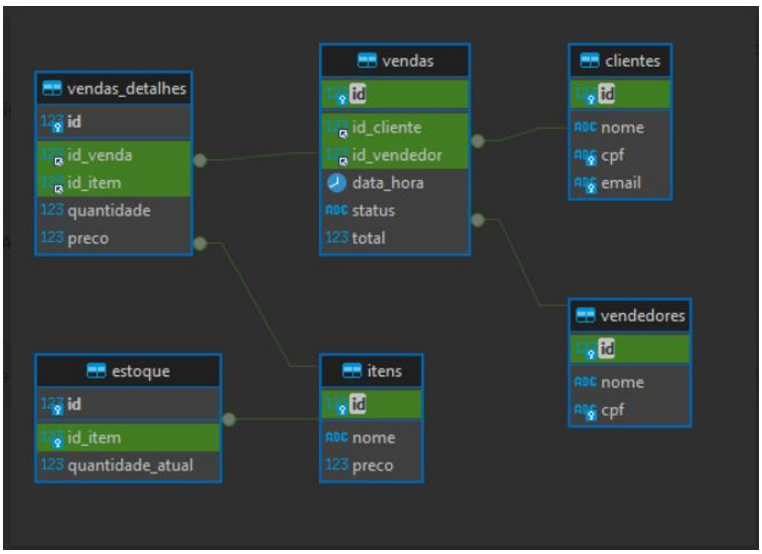
Para todos esses objetos é possível realizar o CRUD (*create, read, update e delete*), com algumas restrições de alteração para alguns desses objetos, como o item em estoque, por exemplo, que não pode ter seu id de item alterado por uma questão de consistência de dados. Ademais, caso algum objeto tenha seu cadastro excluído do banco de dados, e outra entrada na tabela use de seu atributo de identificação como chave estrangeira, optou-se pela operação CASCADE do SQL, de forma a modificar os registros e alterá-los para NULL.

Em relação à API de relatórios, ela somente realiza operações de leitura no banco de dados unificado, trazendo respostas também em JSON (*JavaScript Object Notation*) que demonstram aspectos relevantes para o negócio, como a posição do estoque (zerado e baixo),

dados sobre os produtos transacionados (com vendas finalizadas ou em outros estados) e sobre os clientes (consumo total, consumo médio, as compras feitas e os produtos comprados) - mais detalhes sobre sua utilização no item 5 abaixo (orientações de uso).

Tratando especificamente do banco de dados e sua modelagem, ela obedeceu ao seguinte diagrama de entidade-relacionamento (DER) exposto na figura a seguir (figura 4.2) - elaborado na ferramenta DBeaver[7]:

figura 4.2



Interessante destacar ainda que não foram utilizados algoritmos ou técnicas de *software* sofisticados, prezando-se pela simplicidade e legibilidade do código na solução dos problemas apresentados.

5. ORIENTAÇÕES DE USO

Para utilizar o sistema, basta clonar ("*git clone*") para sua máquina o repositório disponibilizado publicamente no GitHub[8] - procurar pela *tag* denominada "EntragaA3" - e, em seguida, abrir o diretório baixado em alguma IDE (*Integrated Development Environment*). Após, mudar para o diretório denominado "code" (cd /code) e - com o Docker instalado localmente na sua máquina - rodar o comando "*docker compose up -d*".

Aguardar o *download* de todos os arquivos e bibliotecas relacionadas ao projeto nos contêineres e, após o início de todos os três (*Container Started*) - farmacia-db, farmacia-api-

[7] Desenho disponível em: https://github.com/facs-distribuido-mobile/A3/blob/main/relatorio/Diagrama%20-%20entidade%20relacionamento.jpg
[8] https://github.com/facs-distribuido-mobile/A3/tree/main

gerenciamento e farmacia-api-relatorio - testar as aplicações. Em caso de dúvida, há um arquivo README.md dentro da pasta code, com as orientações de utilização do container Docker[9].

Finalmente, interessante pontuar que foi disponibilizada uma *collection* do Postman, de forma a facilitar o teste e avaliação do sistema[10].

6. CONSIDERAÇÕES FINAIS

O projeto para avaliação A3 da Unidade Curricular (UC) de Sistemas Distribuídos e Mobile, 2024/2, da Universidade Salvador (UNIFACS), campus CTN, ministrada pelos professores Adailton de Jesus Cerqueira Jr. e Marivaldo Pereira dos Santos foi bastante desafiador.

Foram enfrentados desafios não somente na parte de modelagem do sistema, como também em relação ao comportamento das aplicações em funcionamento, sobretudo em relação ao modelo (*model*) da venda, que deveria conter uma listagem de itens na transação, tal como ocorre de fato - lembrando que qualquer bom sistema deve resolver problemas do mundo real e não se limitar pelos desafios.

Além disso, houve discussões frutíferas sobre o comportamento do banco de dados em uma série de situações, inclusive sob o aspecto da consistência de dados, no momento em que alguma das entradas existentes em tabela viesse, por acaso, a sofrer uma alteração ou exclusão. Para resolver tal questão, foi utilizada uma funcionalidade nativa do SQL, o *cascade*.

Apesar dessas questões, e da grande quantidade de trabalho a ser entregue, a equipe trabalhou de forma muito sólida e comprometida. Os integrantes atuaram de forma colaborativa constantemente e, acredita-se, produziram um sistema robusto, com todas as funcionalidades pretendidas - e até mais em alguns aspectos.

No tocante ao tema da UC especificamente, concluiu-se pela importância e os benefícios de trabalhar com sistemas distribuídos e os protocolos adjacentes, notadamente sob os aspectos da escalabilidade e performance obtidas com tal arquitetura. De outro lado, nota-se a problemática relativa à estabilidade e velocidade da rede, o que pode comprometer o desempenho de aplicações desenhadas dessa forma.

Também se notou o lado positivo de trabalhar com tecnologias de “containerização”, como o Docker, abstraindo aspectos importantes de sistemas em camadas inferiores da

[9] Visite: https://github.com/facs-distribuido-mobile/A3/blob/main/code/README.md.
[10] Visite https://github.com/facs-distribuido-mobile/A3/blob/main/relatorio/TesteAplicacao.postman_collection.json

aplicação, sobretudo na virtualização, e ajudando o funcionamento desta de maneira agnóstica - além dos benefícios, novamente, de escalabilidade e flexibilidade.

7. BIBLIOGRAFIA

DOCKER. Docker Engine. Página de manual. Disponível em: <https://docs.docker.com/engine/>. Acesso em: 04 de dez. 2024.

GALLOTTI, Giocondo Marino Antonio (org.). **Arquitetura de software**. São Paulo: Pearson, 2016. *E-book*. Disponível em: https://plataforma.bvirtual.com.br. Acesso em: 04 dez. 2024.

NEAL, Richards, Mark, F. **Fundamentos da arquitetura de software: uma abordagem de engenharia**. Rio de Janeiro: Editora Alta Books, 2024. E-book. p.94. ISBN 9788550819754. Disponível em: https://app.minhabiblioteca.com.br/reader/books/9788550819754/. Acesso em: 04 dez. 2024.