

**Sistemas distribuídos e mobile**

**Projeto A3**

Salvador

2023

**EQUIPE:**

CARLOS HENRIQUE BRAGA BOMFIM

12723211573

Vitor Reis dos Santos

12722124265

ISAC DANIEL PEREIRA DE ALMEIDA

12723116417

MATHEUS DO É SANTOS

1272229444

Projeto A3

Trabalho apresentado para fins avaliativos da Unidade Curricular – SISTEMAS DISTRIBUÍDOS E MOBILE- correspondente ao ano letivo 2023.2

Orientador: Adailton, Wellington

SALVADOR

2023

**SUMÁRIO**

[1 INTRODUÇÃO 7](#_Toc153478018)

[2 FUNDAMENTAÇÃO TEÓRICA 9](#_Toc153478019)

[2.4 API GETEWAY 9](#_Toc153478020)

[2.2 API REST 9](#_Toc153478021)

[2.3 SISTEMA ORIENTADO A SERVIÇOS 11](#_Toc153478022)

[2.4 CONTEINERIZAÇÃO E USO DE SOFTWARE ATRAVÉS DE CONTÊINERS 11](#_Toc153478023)

[2.5 CONTEINERIZAÇÃO E USO DE SOFTWARE ATRAVÉS DE CONTÊINERS 12](#_Toc153478024)

[2.6 ORQUESTRANDO CONTÊINERS 13](#_Toc153478025)

[2.7 KUBERNETS E SUA UTILIZAÇÃO COM DOCKER 13](#_Toc153478026)

[3 PROJETO DE IMPLEMENTAÇÃO 15](#_Toc153478027)

[3.1 DEFININDO A ARQUITETURA 15](#_Toc153478028)

[3.2 DESENVOLVENDO A API DE CLIENTES 16](#_Toc153478029)

[3.3 DESENVOLVENDO A API DE ESTOQUE E PRODUTOS 16](#_Toc153478030)

[3.4 DESENVOLVENDO A API DE VENDAS 17](#_Toc153478031)

[3.5 DESENVOLVENDO O API GATEWAY 18](#_Toc153478032)

[3.6 A UTILIZAÇÃO DO DOCKER E DOCKER COMPOSE 18](#_Toc153478033)

[3 CONSIDERAÇÕES FINAIS 20](#_Toc153478034)

[BIBLIOGRAFIA 21](#_Toc153478035)

# 1 INTRODUÇÃO

A arquitetura orientada a serviços (SOA) é um modelo de arquitetura de software que organiza um sistema em unidades independentes chamadas de serviços. Esses serviços são autônomos e podem ser desenvolvidos, implantados e mantidos de modo independente.

Nesse contexto, o trabalho foi iniciado tendo como principal objetivo a implementação de um sistema composto por serviços, objetivando desenvolver serviços integrados que se comunicassem compondo um sistema distribuído. Desse modo, os conteúdos utilizados para o desenvolvimento do projeto, foram: SOA (Arquitetura Orientada a Serviços, APIs, padrão REST, protocolo HTTP, conteinerização de processos e orquestração de containers. No primeiro momento, destaca-se a escolha do modelo de arquitetura utilizado na implementação do sistema, o SOA. Esse modelo de arquitetura (Arquitetura Orientada a Serviços) é um sistema composto de diversos serviços, onde cada serviço tem uma responsabilidade especifica no sistema, como por exemplo: na implementação do projeto, foi desenvolvido um serviço responsável pelo controle de clientes, outro serviço responsável pelo transacional (vendas) e outro responsável pelo estoque. Por fim, um importante ponto de destaque na implementação desse projeto é a comunicação entre os serviços.

O padrão de comunicação escolhido foi o padrão REST com o uso do protocolo HTTP, que são utilizados na comunicação entre APIs na web. Esse conjunto é muito utilizado no modelo de arquitetura cliente-servidor, e em no modelo n-tier, que também foi implementado em SOA na elaboração do projeto.

Portanto, a implementação do sistema utilizando a Arquitetura orientada a serviços (SOA) proporcionou uma abordagem modular e distribuída, no qual serviços autônomos foram elaborados para funções singulares como controle de clientes, transacionais e gestão de estoque. A escolha do padrão REST e do protocolo HTTP para comunicação entre esses serviços evidenciou eficácia, principalmente em ambientes web e modelos cliente-servidor. O emprego de conteinerização e orquestração de contêiners contribuiu para a escalabilidade e flexibilidade do sistema. Sincronicamente, esses elementos possibilitaram uma arquitetura coesa e integrada, facilitando o desenvolvimento, implantação e manutenção dos serviços, consolidando assim benefícios da abordagem SOA no cenário do projeto.

# 2 FUNDAMENTAÇÃO TEÓRICA

## 2.4 API GETEWAY

A necessidade constante de evolução da tecnologia e a crescente dependência das plataformas digitais levaram ao desenvolvimento de diversas ferramentas e arquiteturas para gerir e proteger as transações digitais. Dentre os diversos componentes insubstituíveis na infraestrutura digital, existe um específico que vai ser abordado nesse tópico, o API Gateway.

O API Gateway atua como ponto de entrada seguro, que gerencia solicitações da API com clientes e serviços do back-end. Sua principal funcionalidade é garantir que as solicitações da API sejam roteadas corretamente, autenticadas, autorizadas e processadas com eficiência.

A arquitetura do API Gateway é um servidor que atua como um front-end de API, recebendo solicitações e aplicando políticas de limitação e segurança, emitindo as requisições para o serviço do back-end e, posteriormente, retornando uma resposta ao requisitante. Ele atua como um proxy reverso para obter todas as chamadas da interface de programação de aplicativos (API), agrupar os diversos serviços necessários e retornar o resultado apropriado para a requisição. Seu principal campo de atuação está entre os aplicativos clientes e os serviços de back-end, operando como proxy reverso para responder todas as solicitações da API, e fornecendo um ponto de entrada centralizado para todos os clientes acessarem os serviços do back-end. Esse comportamento centralizado ajuda a gerenciar, monitorar e proteger o tráfego da API entre os clientes e os serviços do back-end.

## 2.2 API REST

API REST, ou API RESTful, é uma interface de programação de aplicações (API) que aplica conceitos do estilo de arquitetura de *software Representational State Transfer*(REST), criado por Roy Fielding, que descreve como tem sido utilizado para orientar projetos e o desenvolvimento arquitetural da Web moderna. (MAYER, 2018; LARANJEIRO; AGNELO; BERNARDINO, 2021 )

O padrão de arquitetura REST enfatiza a escalabilidade das interações dos componentes, generalidades das interfaces, implantação independente de componentes e componentes intermediários para reduzir a latência de integração, reforçar a segurança e encapsular sistemas legados. Um serviço REST(também conhecido como RESTful, WEB API ou REST API) depende do Uniform Resource Identifiers (URI) para identificação de seus recursos e de HTTP para troca de mensagens ( geralmente documentos JSON). O uso de HTTP inclui a presença de um verbo(GET, POST) que especifica o tipo de operação que deve ser executada sobre o recurso exposto. (MAYER ,2018; LARANJEIRO; AGNELO; BERNARDINO, 2021 )

Uma API REST é uma forma de permitir que duas aplicações se comuniquem entre si. Fornecendo um conjunto de rotinas e requisições que as aplicações podem usar para trocar dados. São comumente utilizadas para integrar aplicações web. Por exemplo, uma aplicação web de e-commerce pode usar uma API REST para se conectar a um servidor de banco de dados para acessar informações sobre produtos e pedidos.

As APIs REST oferecem flexibilidade ao permitir o desenvolvimento em diversas linguagens, suportando diferentes formatos de dados. Elas são baseadas em um conjunto de princípios de design que definem como os recursos da web devem ser representados e manipulados. Os princípios de design de APIs REST são:

1. Interface uniforme: As solicitações da API para o mesmo recurso devem ser iguais, não importando a origem da manipulação
2. Desacoplamento do cliente-servidor: Os aplicativos cliente e servidor devem ser independentes.
3. Sem estado definido: As APIs de REST não possuem um estado definido, ou seja, cada solicitação precisa incluir todas as informações necessárias para serem processadas.
4. Capacidade de armazenamento em cache: Os recursos devem ser armazenados em cache pelo cliente ou servidor. As respostas do servidor também precisam conter informações sobre as permissões de cache do recurso fornecido. Isso é necessário, pois o principal objetivo é melhorar o desempenho do cliente, além de aumentar a escalabilidade do servidor.
5. Arquitetura de sistema em camadas: As chamadas e respostas devem passar por diferentes camadas.
6. Código sob demanda: As Apis REST enviam recursos estáticos, mas em determinados cenários, as respostas também podem conter código executável. Nesse contexto, o código deve ser executado somente sob demanda.

Em suma, os princípios de design das APIs REST oferecem uma abordagem flexível e eficiente para o desenvolvimento de sistemas distribuídos. Ao proporcionar uniformidade na interface, o desacoplamento entre cliente e servidor, ausência de estado definido, capacidade de armazenamento em cache, arquitetura em camadas e a inclusão de código sob demanda, possibilitando uma base robusta para construção de serviços web escaláveis e interoperáveis.

## 2.3 SISTEMA ORIENTADO A SERVIÇOS

O Sistema Orientado a Serviços (SOA) é uma abordagem arquitetônica que organiza funcionalidades em serviços independentes e interoperáveis. Cada API (clientes, vendas, relatórios e estoque) é tratada como um serviço independente, com responsabilidades específicas relacionadas à sua função no sistema. A arquitetura SOA proporciona flexibilidade na escolha de tecnologias para implementar cada serviço/API, permitindo escalabilidade independente de acordo com as demandas específicas de cada função. Os serviços/APIs são projetados para serem interoperáveis, permitindo que eles se comuniquem eficientemente entre si para realizar operações mais complexas, se necessário. Ao adotar a abordagem SOA em seu projeto, você cria uma estrutura organizada, flexível e eficiente. Isso facilita a construção de um sistema robusto, preparado para evoluir conforme as necessidades do negócio, e permite uma fácil manutenção ao longo do tempo.

## 2.4 CONTEINERIZAÇÃO E USO DE SOFTWARE ATRAVÉS DE CONTÊINERS

Contêiner é o agrupamento de uma aplicação incorporado com suas dependências, que compartilham o kernel do sistema operacional do host, ou seja, da máquina (seja ela física ou virtual) onde a aplicação está sendo utilizada. (Carey,2021)

Contêineres são semelhantes às máquinas virtuais, mas são mais leves e altamente integrados com o sistema operacional da máquina hospedeira, uma vez que compartilham o mesmo kernel. Isso resulta em um desempenho superior devido ao gerenciamento unificado dos recursos.(Vitalino e Castro 2016)

Ao utilizar uma máquina virtual, ocorre a emulação de um novo sistema operacional e a alocação de recursos adicionais da máquina hospedeira. Isso é diferente ao utilizar contêineres, nos quais os recursos são compartilhados. A vantagem disso é a capacidade de executar um número maior de contêineres em comparação com as máquinas virtuais. (Vitalino e Castro(2016)

Outro aspecto notável na utilização de contêineres é a portabilidade. O ambiente no qual o contêiner foi criado não é relevante. A aplicação funcionará em qualquer sistema operacional, seja Linux, MacOS ou Windows. As preocupações com dependências são mitigadas, uma vez que todas estão encapsuladas dentro do próprio contêiner. (Vitalino e Castro 2016, Carey 2021)

## 2.5 CONTEINERIZAÇÃO E USO DE SOFTWARE ATRAVÉS DE CONTÊINERS

Fundada como *DotCloud* em 2008 por Solomon Hykes, em Paris, a entidade que posteriormente se tornaria o Docker começou como uma plataforma como serviço (PaaS). No entanto, em 2013, ela passou por uma transformação, direcionando seu foco para a democratização dos contêineres de software subjacentes nos quais sua plataforma operava. (Carey,2021)

O Docker decolou com os desenvolvedores de software porque encontrou uma nova forma de empacotar ferramentas necessárias para construir e lançar um contêiner. Dividindo em suas partes componentes, o Docker compreende *Dockerfile*, imagens de contêiner utilitários de execução Docker, Docker Hub, Docker *Engine*, Docker *Compose* e Docker Desktop.

A plataforma ganhou popularidade entre os desenvolvedores de software por ter introduzido uma abordagem inovadora para empacotar as ferramentas essenciais na construção e implantação de contêineres. Esses componentes incluem *Dockerfile*, imagens de contêiner, utilitários de execução Docker, Docker Hub, Docker *Engine*, Docker *Compose* e Docker Desktop. (Carey,2021)

## 2.6 ORQUESTRANDO CONTÊINERS

Impossível falar sobre orquestração de contêiners sem destacar os frameworks mais utilizados, como: *Kubernetes* e Docker. Sendo as principais plataformas de orquestração de contêiners da atualidade quando se trata de *cloud computing,* a utilização dessas ferramentas pode acontecer também em ambiente de servidor *On promisse* por ser uma ótima opção quando o objetivo é escalabilidade e resiliência.(ROSA; MOTA, 2021)

## 2.7 KUBERNETS E SUA UTILIZAÇÃO COM DOCKER

Kubernetes é uma plataforma open-source de orquestração de containers que permite aos desenvolvedores implantar, escalar e gerenciar aplicativos em um ambiente de container. Ele automatiza a implantação, o dimensionamento e a manutenção de aplicativos, tornando mais fácil e eficiente gerenciar aplicativos em ambientes complexos e distribuídos. (BERNSTEIN, 2014)

O Kubernetes adiciona conceitos como Pods, serviços e controladores para gerenciar e estruturar aplicativos em contêineres. Os Pods refletem a menor unidade de implantação no Kubernetes, reforçando um ou mais contêineres que são implantados simultaneamente em um mesmo nó. Os serviços desempenham a função de expor um conjunto de Pods como um único e consistente ponto de acesso de rede, permitindo que outros aplicativos acessem esses recursos de maneira simples. Enquanto isso, os controladores assumem a função de garantir a manutenção do estado desejado do sistema, gerenciando e ajustando os Pods e serviços conforme necessário. (BERNSTEIN, 2014)

Nesse contexto, O Kubernetes atua como uma plataforma de orquestração de contêineres amplamente utilizada para gerenciar aplicativos em contêineres Docker. Ele simplifica a implantação, escalabilidade e manutenção de aplicativos, integra-se perfeitamente com o Docker e oferece recursos avançados, como gerenciamento de estado, atualizações contínuas, monitoramento, segurança e comunicação entre contêineres. Essa combinação é amplamente usada para implementar aplicativos de forma eficiente e escalável em ambientes de produção. (BERNSTEIN, 2014)

# 3 PROJETO DE IMPLEMENTAÇÃO

## 3.1 DEFININDO A ARQUITETURA

Partindo da consolidação dos conceitos relacionados ao desenvolvimento do trabalho, foi iniciada a elaboração da arquitetura do sistema. A arquitetura seguiu o padrão de arquitetura orientado a serviços, composta por quatro serviços: clientes, estoque e vendas. Todos esses serviços foram desenvolvidos como APIs seguindo o padrão REST, onde a comunicação entre os serviços usa o protocolo HTTP e o formato JSON para estruturar o corpo das requisições e respostas nas interações entre as APIs.

**Figura 1:** Arquitetura

Diagrama

Descrição gerada automaticamente

**Fonte:** Autoria Própria

Além dos serviços de clientes, estoque, e vendas, a arquitetura é composta por uma quarta API, que implementa o *design pattern,* API Gateway. O API Gateway é um *design pattern* que tem o objetivo de desempenhar o papel de uma interface para as demais APIs (serviços) do sistema. O API Gateway roteia as chamadas originadas do cliente para as APIs e tem um papel importante relacionado a segurança no acesso às APIs. O API Gateway segue o princípio de transparência em um sistema distribuído, pois o cliente não sabe como o sistema é implementado, além do API Gateway em si. O cliente não sabe se há um ou mais serviços ou se foi implementado.

A escolha das tecnologias para o desenvolvimento das APIs foi realizada conforme a aptidão dos desenvolvedores para atuar nas linguagens e frameworks selecionados. Então, alguns serviços, como o API Gateway e a API de vendas foram desenvolvidas em Node.js com Express, e as APIs de clientes e estoque foram desenvolvidas em Java com Spring framework. Todos esses serviços têm um banco de dados próprio, nos quais foram implementados com o MySQL. Por fim, um ponto a de destaque é que as diferentes tecnologias escolhidas no desenvolvimento dos serviços destacam a heterogeneidade do sistema, que é uma característica marcante em sistemas distribuídos.

## 3.2 DESENVOLVENDO A API DE CLIENTES

O desenvolvimento da API de clientes foi iniciado a partir da definição da entidade Cliente, e após o desenvolvimento do CRUD de cliente. A API de clientes foi desenvolvida em Java com Spring e o banco de dados escolhido foi o MySQL. O padrão de arquitetura escolhido foi o MVC, que é um padrão muito utilizado no desenvolvimento de APIs REST com o Spring framework e Spring MVC.

## 3.3 DESENVOLVENDO A API DE ESTOQUE E PRODUTOS

A elaboração da API de produtos seguiu as mesmas etapas do desenvolvimento da API de clientes, com a diferença que esse serviço tem duas responsabilidades, e por isso duas entidades com um CRUD cada. Essa API também foi desenvolvida utilizando Java com o Spring framework e Spring MVC e o banco de dados utilizado foi o MySQL. Na modelagem das entidades, a primeira entidade é a entidade *Product*, que representa um *SKU*, o qual é o registro que caracteriza um único produto. A outra entidade é o *ProductInStock* que representada cada produto, ou seja, cada unidade que está no estoque.

**Figura 2:** Diagrama de classes da API de estoque.

Diagrama

Descrição gerada automaticamente

**Fonte:** Autoria Própria

## 3.4 DESENVOLVENDO A API DE VENDAS

Já a API de vendas foi desenvolvida utilizando Node.js com Express e o banco de dados MySQL, assim como nos outros serviços. O desenvolvimento da API de vendas também seguiu o padrão de arquitetura MVC, com duas entidades e uma única funcionalidade, sendo a funcionalidade de venda. Além disso, na modelagem desse serviço, temos duas entidades: a primeira entidade é a Sell, que é a entidade que representa a venda. Essa entidade consiste no ID do cliente que fez a compra, do SKU que foi vendido, e da quantidade de itens vendidos. A segunda entidade é a ProductSold, que representa o produto no estoque que foi vendido. Essa entidade contém o ID da venda e o ID do produto no estoque que foi vendido.

**Figura 3:** Diagrama de classes da API de vendas.

Diagrama

Descrição gerada automaticamente

**Fonte:** Autoria Própria

Nesse contexto, destaca-se o desenvolvimento da API de vendas e a utilização do API Gateway. Como a API de vendas, no algoritmo de cadastro da venda, precisa consultar se o usuário que está realizando a venda existe, e se o SKU existe, além de consultar se há produtos suficientes no estoque, a API de vendas fará isso através do API Gateway. Um ponto importante é que a API de vendas não se comunicará com API de estoque e a API de clientes diretamente, sendo assim ela precisará se comunicar com as outras APIs através do API Gateway. O API Gateway funciona como uma interface (ou um broker) não só para o cliente, mas também para a comunicação entre os serviços que constituem o SOA.

## 3.5 DESENVOLVENDO O API GATEWAY

A API Gateway foi desenvolvida em Node.js com Express, sem o uso de banco de dados. Não se fez necessário o uso de banco de dados no desenvolvimento do API Gateway, pois o papel dessa API é de apenas redirecionar as requisições para os respectivos serviços da SOA. Sendo assim, a implementação da API foi composta basicamente de *controllers*, sendo que cada *controller* representa um dos serviços no SOA e seus métodos cada endpoint dos serviços.

## 3.6 A UTILIZAÇÃO DO DOCKER E DOCKER COMPOSE

Por fim, tanto o Docker como o Docker Compose foram utilizados na execução dos serviços. O Docker foi utilizado para a execução de cada serviço em containers. A Conteinerização é um recurso que é muito utilizado na infraestrutura e orquestração de microsserviços, onde cada serviço é um container. A definição de container é um processo que é “isolado” do restante do sistema. Para os containers dos serviços, o Docker Compose foi utilizado. A vantagem da utilização de um orquestrador é a facilidade na configuração da infraestrutura para a execução dos containers em conjunto. Uma vantagem é que foi possível rodar mais de um container do MySQL, sendo cada container utilizado por um serviço.

**Figura 4 :**Configuração do Docker Compose

Texto

Descrição gerada automaticamente

.

**Fonte:** Autoria Própria

# 3 CONSIDERAÇÕES FINAIS

O desenvolvimento das APIs com as tecnologias utilizadas no projeto desempenhou um papel crucial na construção de um sistema integrado voltado para gestão eficiente dos processos relacionados a vendas. Por toda extensão desse processo, foram encontrados desafios substanciais que, quando superados, resultaram em uma solução efetiva, estruturada e em conformidade com as boas práticas de desenvolvimento de software.

No contexto das realizações mais significativas, a API de vendas foi elaborada com endpoints para gerir transações, abrangendo operações que incluem a criação, atualização e remoção de pedidos. Destaca-se também a integração efetiva com o banco de dados, possibilitando a retenção segura e recuperação precisa dos dados de vendas.

Portanto, visamos garantir a evolução contínua do projeto, recomendando práticas como monitoramento constante do desempenho das APIs, implementando testes para garantir a robustez do código, e a exploração de tecnologias adicionais para aprimorar a segurança e eficiência do sistema. Este trabalho retrata um marco importante em nossa jornada de desenvolvimento, avaliando nosso comprometimento com a entrega de soluções. O desenvolvimento do projeto criou um ambiente de confiança no desenvolvimento das APIs, fornecendo uma base sólida para futuras expansões e aprimoramentos.

# BIBLIOGRAFIA

AKITA F. Entendendo Como Containers Funcionam. **Akitaonrails**. 02 Mar. 2023. Disponível em:. <https://www.akitaonrails.com/2023/03/02/akitando-139-entendendo-como-containers-funcionam>. Acesso em: 19 out. 2023.

BERNSTEIN, David. Containers and Cloud: from lxc to docker to kubernetes. **Ieee Cloud Computing**, [S.L.], v. 1, n. 3, p. 81-84, set. 2014. Institute of Electrical and Electronics Engineers (IEEE). <http://dx.doi.org/10.1109/mcc.2014.51>.

BERNSTEIN, David. Containers and Cloud: from lxc to docker to kubernetes. Ieee Cloud Computing, [S.L.], v. 1, n. 3, p. 81-84, set. 2014. Institute of Electrical and Electronics Engineers (IEEE). <http://dx.doi.org/10.1109/mcc.2014.51>.

CAREY, Scott. What is Docker? The spark for the container revolution. InfoWorld, 2021. Disponível em:i. Acesso em: 18 jun. 2022.

INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY (IJCST) : API GATEWAY ARCHITECTURE EXPLAINED. [*S. l.*]: INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY (IJCST), 1900-2021. ISSN 27095088. Mensal. Disponível em:

https://www.ijcst.com.pk/index.php/IJCST/article/view/269/236. Acesso em: 12 dez. 2023.

LARANJEIRO, Nuno; AGNELO, Joao; BERNARDINO, Jorge. A Black Box Tool for Robustness Testing of REST Services. Ieee Access, [S.L.], v. 9, p. 24738-24754, 2021. Institute of Electrical and Electronics Engineers (IEEE). <http://dx.doi.org/10.1109/access.2021.3056505>.

MAYER, Geornuno Laranjeirog. RESTful APIs for the 5G ServiceBased Architecture. Journal Of Ict Standardization, [S.L.], v. 6, n. 1, p. 101-116, 2018. River Publishers. <http://dx.doi.org/10.13052/jicts2245-800x.617>.

ROSA, João Antônio Caetano; MOTA, José dos Reis. UTILIZAÇÃO E ORQUESTRAÇÃO DE CONTAINERS EM APLICAÇÕES WEB. Revista do Fórum Gerencial: Centro Universitário de Patos de Minas, Minas Gerais, v. 1, n. 2, p. 127-138, 2021. Semanal.

VITALINO, Jeferson Fernando Noronha; CASTRO, Marcus André Nunes. Descomplicando o Docker. Rio de Janeiro: Brasport, 2016