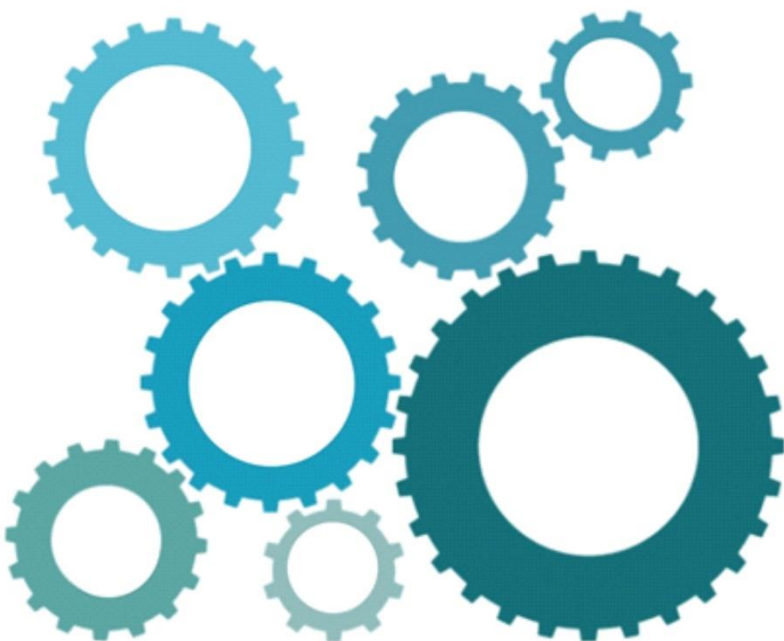







## Métodos e Técnicas de Suporte ao Desenvolvimento de Software

**DMW**  
*Microservices*



David Afonso - 8180011  
Michele Freitas - 8180014

## Índice

Enquadramento teórico .....	1
Engenharia de Software .....	1
DDD - Domain Driven Design.....	1
Bounded Context .....	2
Aplicação Monolítica e Microserviços .....	3
Objetivos .....	6
Recursos .....	6
Caso de Estudo - Plataforma de “Loja Online” .....	7
O Conceito .....	7
O desenho conceptual .....	7
Análise do Domínio .....	7
Definição dos Bounded Contexts.....	8
Arquitetura técnica .....	9
Microserviços .....	9
Criação da Arquitetura Final.....	14
UI Inicial .....	16
UI Final.....	17
Implementação - Tecnologias.....	20
 RabbitMQ .....	20
 Kubernete .....	21
 GitHub .....	21
Considerações finais .....	22
Bibliografia.....	23



## Enquadramento teórico

### Engenharia de Software

A engenharia de software engloba não só os aspetos tecnológicos da programação de aplicações mas também os vários aspetos sociais com que está relacionada, a conceptualização do funcionamento dos sistemas, os princípios das bases de dados e das redes de computadores, os princípios de gestão, a gestão da equipa, os testes e os ambientes de produção. A engenharia de software é um importante suporte para o desenvolvimento do crescimento da economia mundial eletrónica em que estamos inseridos.

Os processos de desenvolvimento dividem-se em dois tipos funcionais: os clássicos - que visam responder apenas aos requisitos identificados, e os ágeis - que têm a preocupação de avaliar constantemente o que está a ser feito e verificar se está correto.

A tendência tem sido de recorrer aos processos ágeis e incorporar o DevOps que não é mais que a combinação de filosofias culturais, práticas e ferramentas que aumentam a capacidade de uma empresa de distribuir aplicativos e serviços em alta velocidade: otimizando e aperfeiçoando produtos num ritmo mais rápido do que o das empresas que usam processos tradicionais de desenvolvimento de software e gerenciamento de infraestrutura. Essa velocidade permite que as empresas atendam melhor aos seus clientes e compitam de modo mais eficaz no mercado.

### DDD - Domain Driven Design

O DDD induz à implementação de um cenário de melhoria contínua, podendo ser uma ferramenta extremamente útil para se desenvolver software de qualidade e que atenda bem as necessidades do cliente.

As técnicas de DDD, que ensinam justamente boas práticas de como modelar seu domínio, além de tornar eficiente a interação entre os vários papéis de pessoas que fazem parte do processo de desenvolvimento de software. DDD pode ser muito útil no trabalho em equipa no desenvolvimento de um sistema complexo.

Domain Driven Design significa Projeto Orientado ao Domínio. Ele veio do título do livro escrito por Eric Evans, dono da Domain Language, uma empresa especializada

em formação e consultoria para desenvolvimento de software. O livro de Evans, boas práticas de programação é um grande catálogo de Padrões, baseados em experiências do autor ao longo de mais de 20 anos desenvolvendo software utilizando técnicas de Orientação a Objetos.

DDD pode ser visto como o regresso da orientação a objetos. Quando se fala em Orientação a Objetos pensa-se logo em classes, heranças, polimorfismo, encapsulamento. Mas a essência da Orientação a Objetos também tem outros elementos como:

- Alinhamento do código com o negócio: o contato dos desenvolvedores com os especialistas do domínio é algo essencial quando se faz DDD;
- Favorecer reutilização: os blocos de construção, facilitam aproveitar um mesmo conceito de domínio ou um mesmo código em vários lugares;
- Mínimo de acoplamento: Com um modelo bem feito, organizado, as várias partes de um sistema interagem sem que haja muita dependência entre módulos ou classes de objetos de conceitos distintos;
- Independência da Tecnologia: DDD não se foca em tecnologia, mas sim em entender as regras de negócio e como elas devem estar refletidas no código e no modelo do domínio.

Para ter um software que atenda perfeitamente a um determinado domínio, é necessário que se estabeleça, em primeiro lugar, uma Linguagem comum, com termos bem definidos, que fazem parte do domínio do negócio e que são usados por todas as pessoas que fazem parte do processo de desenvolvimento de software. Nessa linguagem estão termos que fazem parte das conversas diárias entre especialistas de negócio e equipes de desenvolvimento. Todos devem usar os mesmos termos tanto na linguagem falada quanto no código.

### Bounded Context

Usando uma abordagem DDD ajudará a estruturar microsserviços, para que todos os serviços de formulários um ajuste natural para um requisito comercial funcional. Ele pode ajudar a evitar a armadilha de permitir que os limites organizacionais ou opções de tecnologia ditam seu design.

O DDD é uma abordagem para o desenvolvimento de software para necessidades complexas, conectando a implementação a um modelo em evolução.

Antes de escrever qualquer código, tem um panorama geral do sistema a ser criado. O DDD começa com a Modelagem do domínio de negócios e a criação de um modelo de domínio, que é um modelo abstrato do domínio empresarial. Seleciona e organiza os dados de conhecimento do domínio e fornece uma linguagem comum para programadores e especialistas de domínio.

### Aplicação Monolítica e Microsserviços

A arquitetura monolítica é a arquitetura de sistema operacional mais comum e antiga, no qual cada componente do S.O. está contido no núcleo do sistema. Ela é uma aplicação formada por vários módulos que são compilados separadamente e depois linkados, formando assim um grande sistema onde os módulos podem interagir.

Ela aumenta a complexidade e o tamanho do código, agrupando tarefas similares em camadas. Dividido em camadas com níveis sobrepostos. Camadas inferiores oferecem funções para camadas superiores. Cada camada comunica-se exclusivamente com as camadas acima e abaixo apenas.

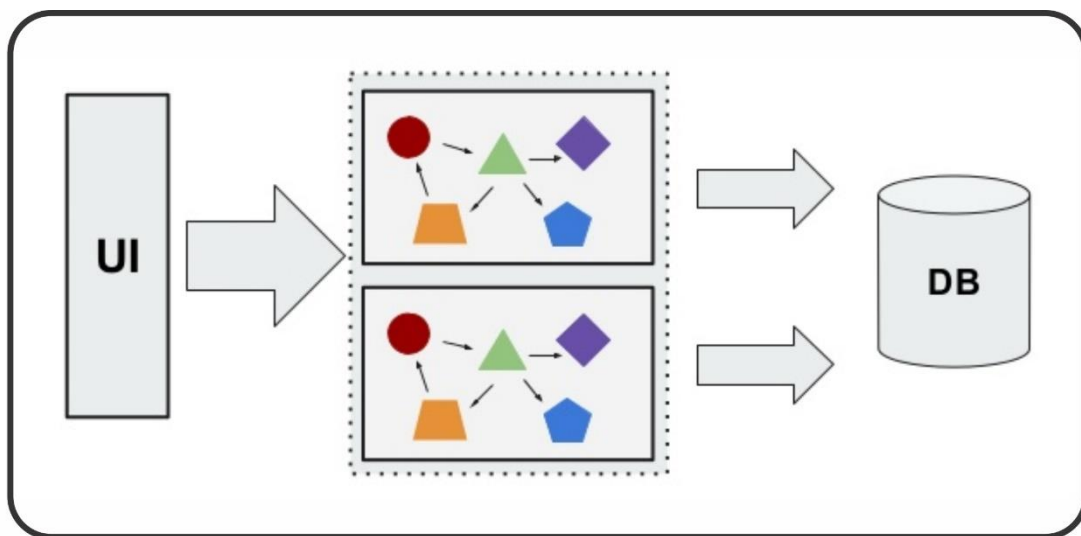
Ela descreve uma única aplicação de software em camadas no qual a interface de usuário e código de acesso aos dados são combinados em um único programa a partir de uma única plataforma. A utilização de Microsserviços é uma abordagem para desenvolver um único aplicativo como um conjunto de pequenos serviços, cada um executando em seu próprio processo e comunicando-se com mecanismos leves, geralmente uma API de recurso HTTP.

Como tudo em desenvolvimento de software, existem vantagens e desvantagens nos sistemas monolíticos.

Por outro lado, temos um sistema cujo deploy é fácil de ser feito, já que o banco de dados facilmente evoluirá junto para todas as funcionalidades e há apenas um ponto onde o deploy precisa ser feito. Além disso, não há duplicidade de código e classes necessárias entre os diferentes módulos, já que todas elas fazem parte da mesma unidade.

Um dos principais pontos negativos é que você tem um grande ponto único de falha, como mostra a figura 1 abaixo, que significa que se houver algum erro no cadastro de funcionários que deixe o sistema fora do ar, isso vai levar junto todo o sistema, incluindo funcionalidades que não possuem nenhuma relação com essa funcionalidade.

Outro ponto negativo é a base de código, que se torna muito extensa, podendo deixar novos membros do projeto menos produtivos por algum tempo, já que a complexidade do código é bem maior.



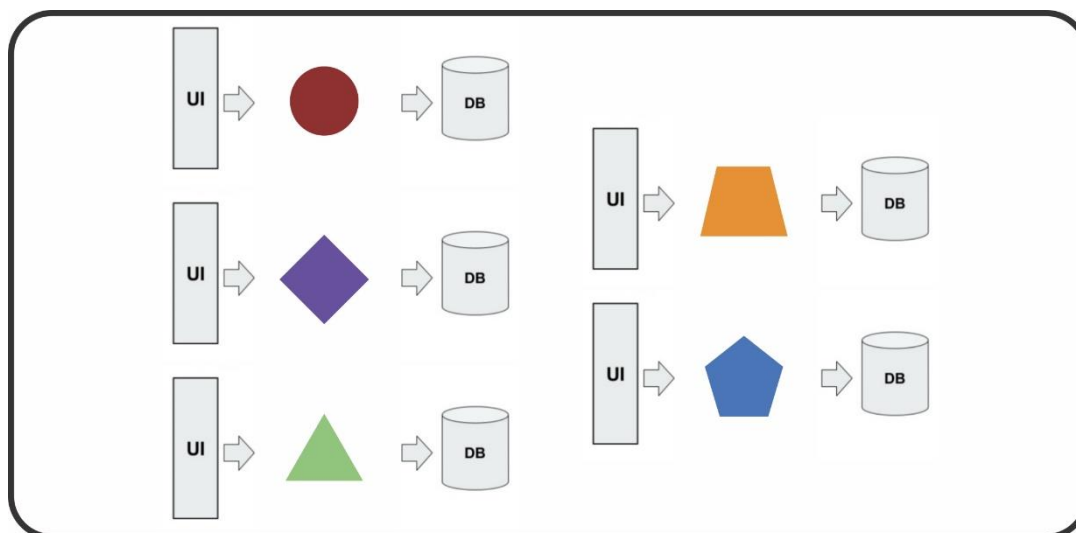
*Figura 1 - Exemplo desenho arquitetura monolítica*

Esses serviços são criados com base nos recursos de negócios e implementados de maneira independente por um mecanismo de implantação totalmente automatizado.

Há um mínimo de gerenciamento centralizado desses serviços, que pode ser escrito em diferentes linguagens de programação e usar diferentes tecnologias de armazenamento de dados.

Microserviço é um estilo arquitetônico, o que significa que é uma especialização de tipos de elementos e relações, juntamente com restrições e como elas podem ser usadas.

O principal benefício dos microserviços é a agilidade, reduzindo muito o tempo de produção por exemplo numa aplicação muito grande. Permitindo dividir o aplicativo num conjunto de componentes desacoplados, como mostra a figura 2, que fornecem serviços definidos (meios definidos com uma interface ou API conhecida).



*Figura 2 - Exemplo desenho arquitetura microserviços*

Com isso os componentes comunicam entre si através de um qualquer protocolo escolhido, geralmente REST, mas não necessariamente. Podendo os componentes usar linguagens e tecnologias que quiserem, sendo desenvolvidos, lançados e implantados de forma independente de modo que a orquestração de toda a aplicação seja reduzida ao mínimo.



## Objetivos

- Conceber a arquitetura técnica de um projeto segundo o paradigma microsserviços;
- Desenvolver e implementar uma solução tecnológica de acordo com as práticas e padrões associados a aplicações orientadas a microsserviços;
- Argumentar decisões de arquitetura e justificar opções de implementação no contexto de desenvolvimento de arquitetura baseadas em microsserviços.

## Recursos

- Minikube
- Kubernetes
- GIT Server (qualquer)
- Google Cloud Platform (ou alternativa semelhante disponível publicamente)
- Opções para o desenvolvimento dos microsserviços (Vert.x, SeedStack, Spring Boot, JHypster, etc)

## Caso de Estudo

### Plataforma de “Loja Online”

#### O Conceito

Para esse projeto com intenção de atingir os objetivos propostos, desenvolvemos uma loja online.

Atualmente, a internet é um canal de vendas tão consolidado quanto as lojas físicas. Ao fazer compras online, os consumidores também desfrutam da comodidade e da economia de tempo. Grande parte das pessoas que compram pela web já conhecem o produto que estão comprando. Por isso, não sentem a necessidade de ter um contato físico com o artigo.

Ao comprar num comércio eletrônico, o cliente não precisa enfrentar trânsito para ir até à loja física, nem se preocupar com estacionamento.

O processo é fácil e rápido. Inicia-se com a vontade de um cliente em comprar um produto. Acendendo à plataforma da loja, navega pela lista de produtos e seleciona quais itens deseja comprar. Após escolher os produtos pretendidos, clica no botão “Checkout to Cart”, sendo então redirecionado para a página do carrinho para finalizar suas compras.

A compra despoleta a confirmação dos produtos pretendidos, detalhando a identificação da compra, o valor de cada item, a data da compra e o valor total.

Após a compra o cliente finaliza no botão “Go to Order” e já tem o sucesso do seu pedido.

#### O desenho conceptual

##### Análise do Domínio

Tendo como ponto de partida a aplicação de uma metodologia de desenvolvimento de software orientada ao Domínio segundo o DDD - Domain driven Design, o primeiro passo é a definição do Domínio. Assim sendo, como o âmbito principal da plataforma é a disponibilização de uma lista de produtos que podem ser comprados, definimos que o domínio principal é a compra de produtos.

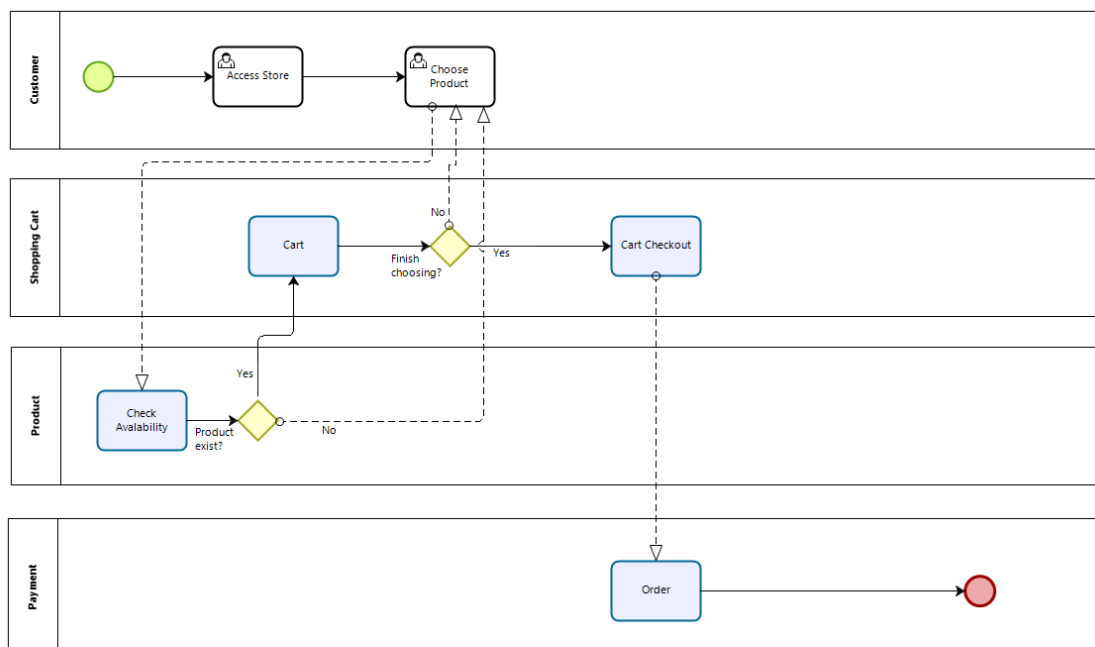
O passo seguinte passa pela definição dos bounded contexts.

### Definição dos Bounded Contexts

Definimos que a plataforma de “Loja Online” permite:

- Acesso a página principal da loja;
- Consulta da lista de Produtos disponíveis;
- Possibilidade de escolha de produtos;
- Aquisição dos produtos;
- Finalização da compra.

Utilizamos o Bizagi para fazer o BPMN da arquitetura inicial do projeto. Nele definimos os diagramas do passo a passo do ciclo do nosso projeto, desde a fase do cliente aceder à loja online, acessar a lista de produtos e escolher todos os produtos desejados. Após concluir a seleção e clicar no botão checkout que redireciona para a página do carrinho de compra, o cliente confere os itens selecionados e finaliza a compra no botão “go to order”.



Powered by  
**bizagi**  
Modeler

Figura 5– Idealização inicial dos processos da loja

Após análise da ideia inicial definimos que os bounded contexts serão:

- **Produto** – visualização e seleção;
- **Carrinho de Compra** – confirmação do que se pretende comprar;
- **Pagamento** – Pagamento e conclusão da compra.

### Arquitetura técnica

A idealização técnica da loja está suportada numa solução tecnológica que permite o seu desenvolvimento continuado e sua integração sem tempos de indisponibilidade. Esta solução tecnológica é o MINIKUBE que se trata de uma plataforma de Kubernetes mais acessível, que está limitado à utilização de um cluster. No entanto, permite testar localmente o deployment dos programas sem grande dificuldade.

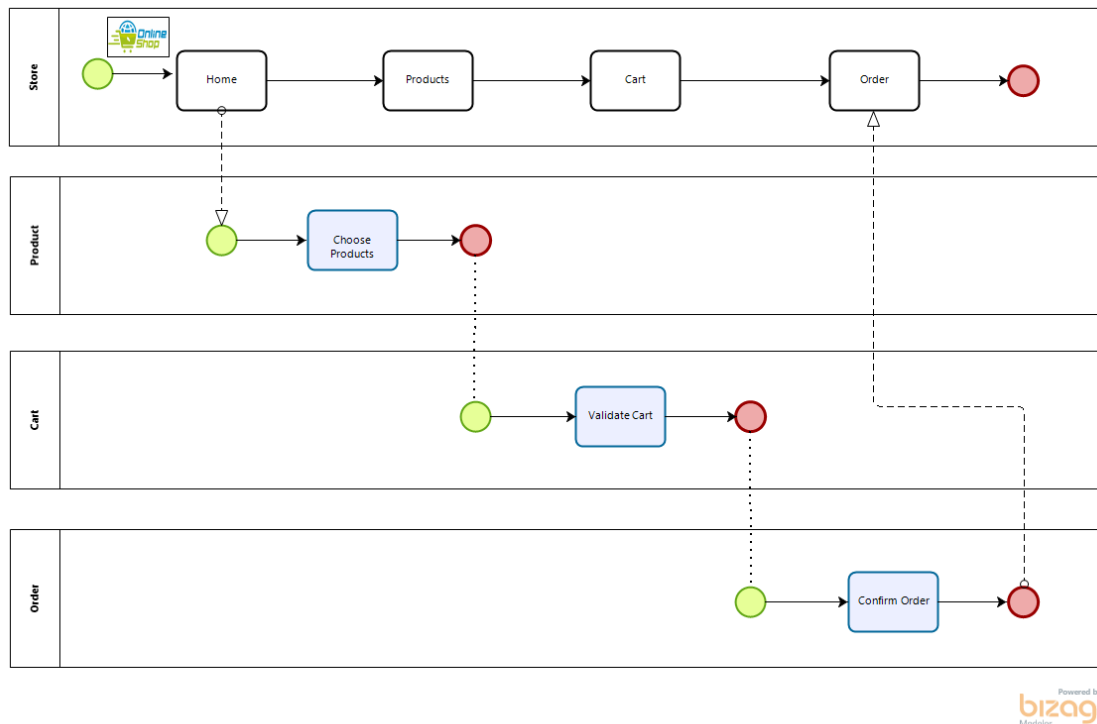
O Kubernetes é um orquestrador de containers. No Kubernetes a unidade básica são os pods, onde conseguimos guardar os nossos containers, assim sendo, poderemos ter um ou mais containers dentro de um pod. O mais interessante é que, dentro dos nossos pods poderemos ter, além dos containers, várias configurações, para que se comportem da maneira que desejamos.

Após definição dos bounded contexts, pretendemos criar as aplicações e usando a metodologia dos microsserviços, isolar ao máximo as suas funcionalidades.

Este isolamento deverá ser idealmente levado ao extremo, de modo a que a comunicação seja feita usando apenas os dados estritamente necessários. Também deverá ser dada importância à utilização de comunicação assíncrona para que haja a menor perda de tempo entre operações.

### Microsserviços

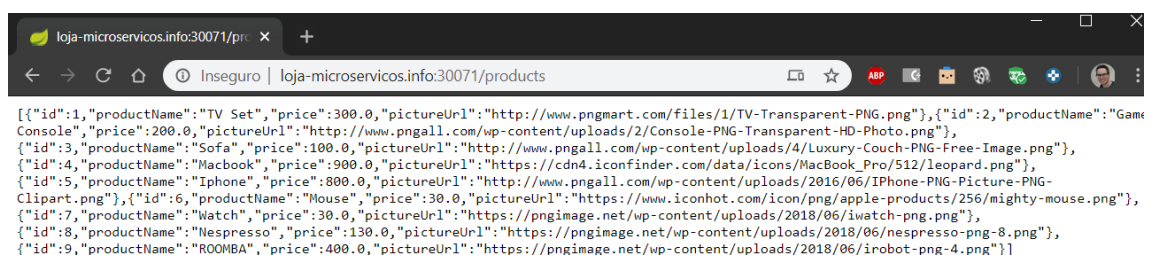
Inicialmente definimos três microsserviços diferenciados de acordo com o esquema seguinte:



- **Product:** O seu papel é disponibilizar a lista de produtos para o UI e receber a lista de produtos selecionados para publicar na queue de produtos selecionados.

As funcionalidades deste programa foram divididas em 2, criando assim 2 microserviços:

- **Productlist** que disponibiliza ao UI a lista de produtos via API REST no endereço <http://loja-microservicos.info:30071/products>;



- **Productselect**, que recebe a lista de produtos selecionados no UI da loja e os envia para a queue `productselectionqueue` do RabbitMQ usando a funcionalidade publish.

Inseguro | loja-microservicos.info:30072/#/queues/%2F/productselection.queue

RabbitMQ 3.8.2 Erlang 22.2.6

Refreshed 2020-02-17 21:34:50 Refresh every 5 seconds

Virtual host All Cluster rabbit@rabbitmq User guest Log out

OverviewConnectionsChannelsExchangesQueuesAdmin

## Queue productselection.queue

Overview

Queued messages last minute ?

Ready	0
Unacked	0
Total	0

Message rates last minute ?

Publish	0.00/s
Deliver (manual ack)	0.00/s
Deliver (auto ack)	0.00/s
Consumer ack	0.00/s
Redelivered	0.00/s
Get (auto ack)	0.00/s
Get (empty)	0.00/s

Details

Features	State	Consumers	Messages	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Policy	idle	1	?	0	0	0	1	0	0

Inseguro | loja-microservicos.info:30072/#/queues/%2F/productselection.queue

RabbitMQ 3.8.2 Erlang 22.2.6

Refreshed 2020-02-17 21:35:04 Refresh every 5 seconds

Virtual host All Cluster rabbit@rabbitmq User guest Log out

OverviewConnectionsChannelsExchangesQueuesAdmin

## Queue productselection.queue

Overview

Queued messages last minute ?

Ready	0
Unacked	0
Total	0

Message rates last minute ?

Publish	0.00/s
Deliver (manual ack)	0.00/s
Deliver (auto ack)	0.00/s
Consumer ack	0.00/s
Redelivered	0.00/s
Get (auto ack)	0.00/s
Get (empty)	0.00/s

Details

Features	State	Consumers	Operator policy	Effective policy definition	Consumer utilisation	Messages	Message body bytes	Process memory	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Policy	idle	1			0%	?	0B	10kB	0	0	0	0	0	0

Bindings

Publish message

Inseguro | loja-microservicos.info:30072/#/queues

RabbitMQ 3.8.2 Erlang 22.2.6

Refreshed 2020-02-17 21:32:05 Refresh every 5 seconds

Virtual host All Cluster rabbit@rabbitmq User guest Log out

OverviewConnectionsChannelsExchangesQueuesAdmin

## Queues

All queues (1)

Pagination

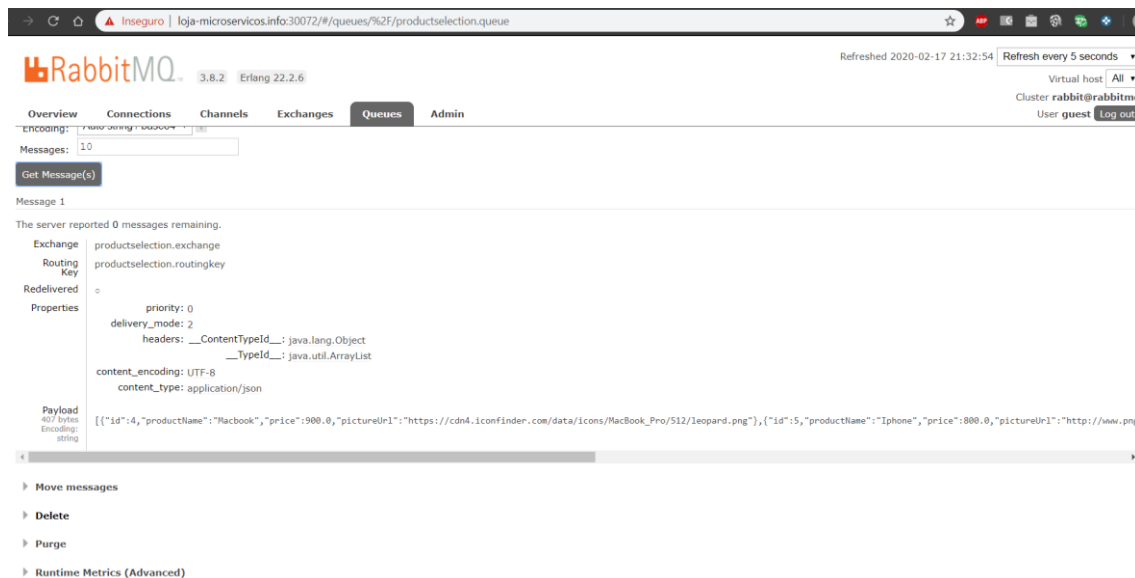
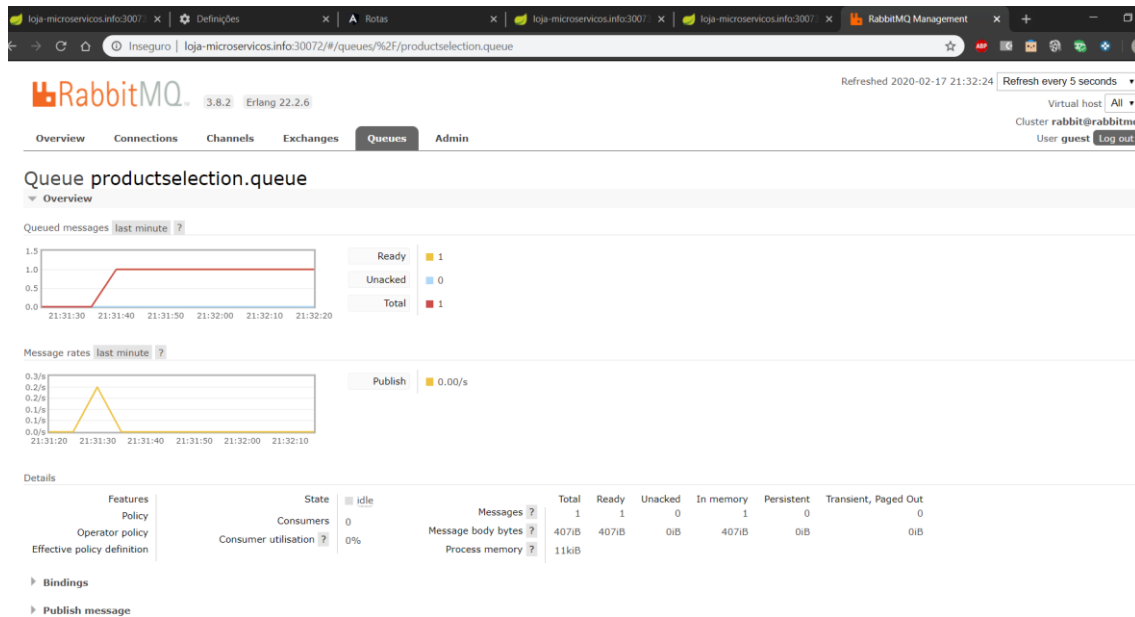
Page 1 of 1 - Filter: Regexp

Displaying 1 item, page size up to: 1

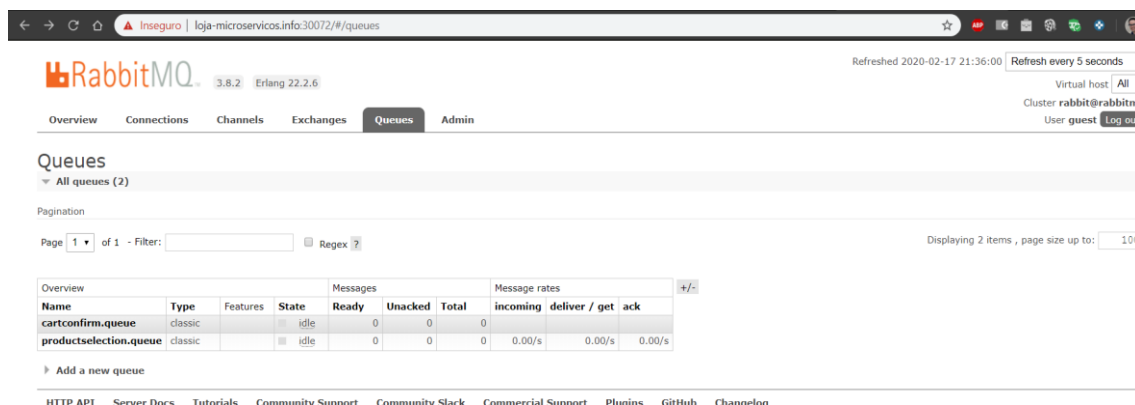
Overview				Messages			Message rates			
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
productselection.queue	classic		idle	1	0	1	0.00/s			

Add a new queue

[HTTP API](#)
[Server Docs](#)
[Tutorials](#)
[Community Support](#)
[Community Slack](#)
[Commercial Support](#)
[Plugins](#)
[GitHub](#)
[Changelog](#)

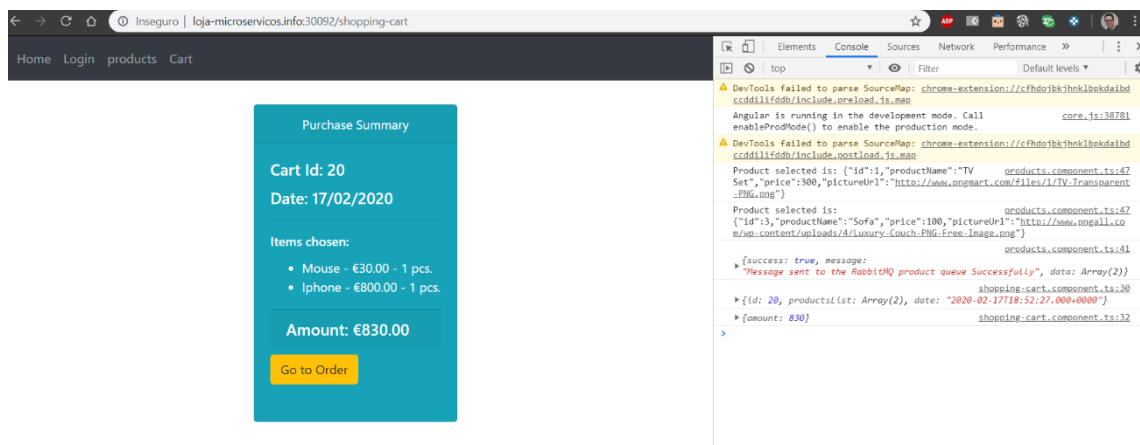


- **Shopping Cart:** Vai buscar a lista de produtos selecionados, disponibiliza o valor total da compra e permite confirmar o carrinho.



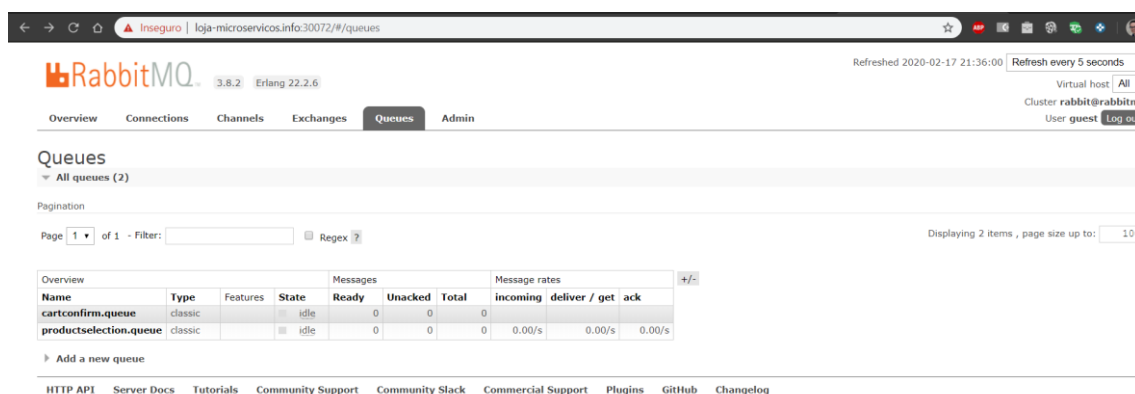
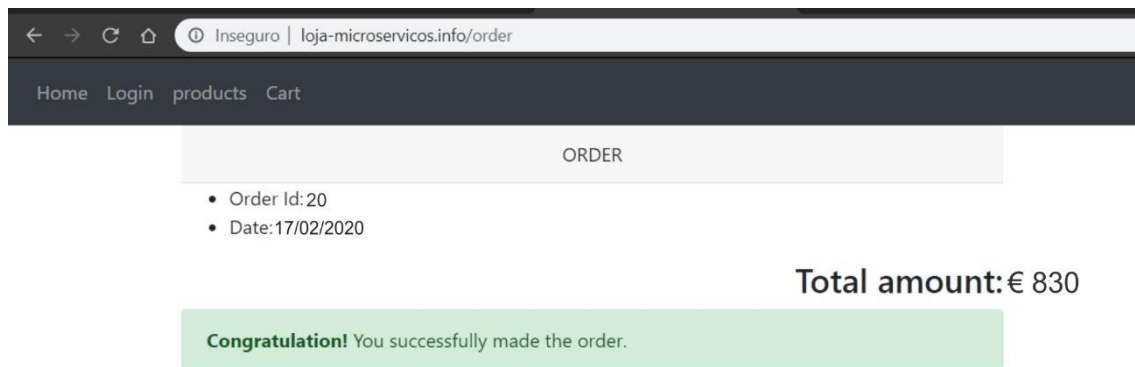
As suas funcionalidades foram igualmente divididas em 2, criando outros 2 microserviços:

- **Cartloader**, Fazendo subscribe da queue `productselectionqueue` do RabbitMQ aguarda que surja uma mensagem com uma lista de produtos a adicionar ao carrinho. Através da API REST em `http://loja-microservicos.info:30073/lastcart` disponibiliza ao UI a lista de produtos e seu preço.



- **Cartconfirm**, Recebe o id do carrinho de compras, a quantia e a data de confirmação e envia estes dados para a queue `cartconfirmqueue` do RabbitMQ através da funcionalidade publish.
- **Order**: Faz subscribe da queue `cartconfirmqueue` do RabbitMQ e disponibiliza estes dados ao UI através do API REST em `http://loja-microservicos.info:30074/lastorder`.





Todos os serviços são independentes e cada um possui um banco de dados separado, seja em pods independentes MySQL ou bases de dados H2 do próprio aplicativo Springboot.

## Criação da Arquitetura Final

A arquitetura final, como apresentado na figura abaixo, abordamos todas as tecnologias envolvidas no projeto na criação dos microserviços, desde o backend que até ao front end com o angular, a base de dados MySQL e o RabbitMQ como servidor de mensagens para garantir a comunicação assíncrona entre os microserviços.

A gestão dos pods, serviços Também utilizamos o kubernetes para gerenciamento dos nossos containers.

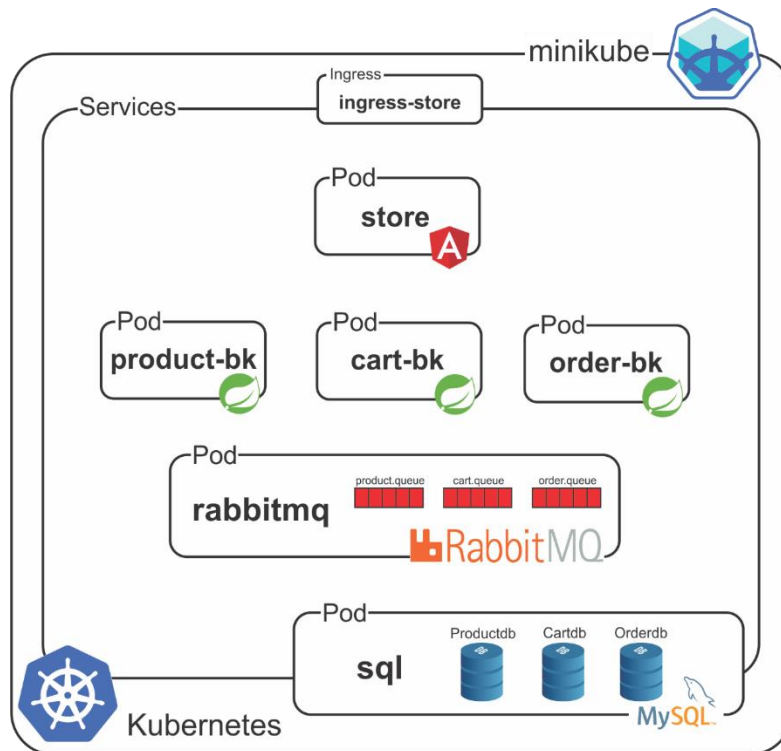


Figura 3 - Arquitetura anterior do projeto

Inicialmente havíamos criado o pod que são serviços geridos pelo Kuberne com o minikube, para correr nosso programa verificando a resposta do banco de dados, sendo que cada microsserviço possui a sua base de dados correspondente (Productdb, Cartdb e Orderdb). Depois notamos que para ficarem independentes cada pod deveríamos criar pods separados para cada base de dados. Então decidimos criar um pod para cada serviço e ele cuidar da sua base de dados correspondente, como por exemplo o pod order tratar da base de dados dtOrder, e assim com os outros serviços. Assim os serviços são executados no minikube e cada um liga ao seu pod equivalente.

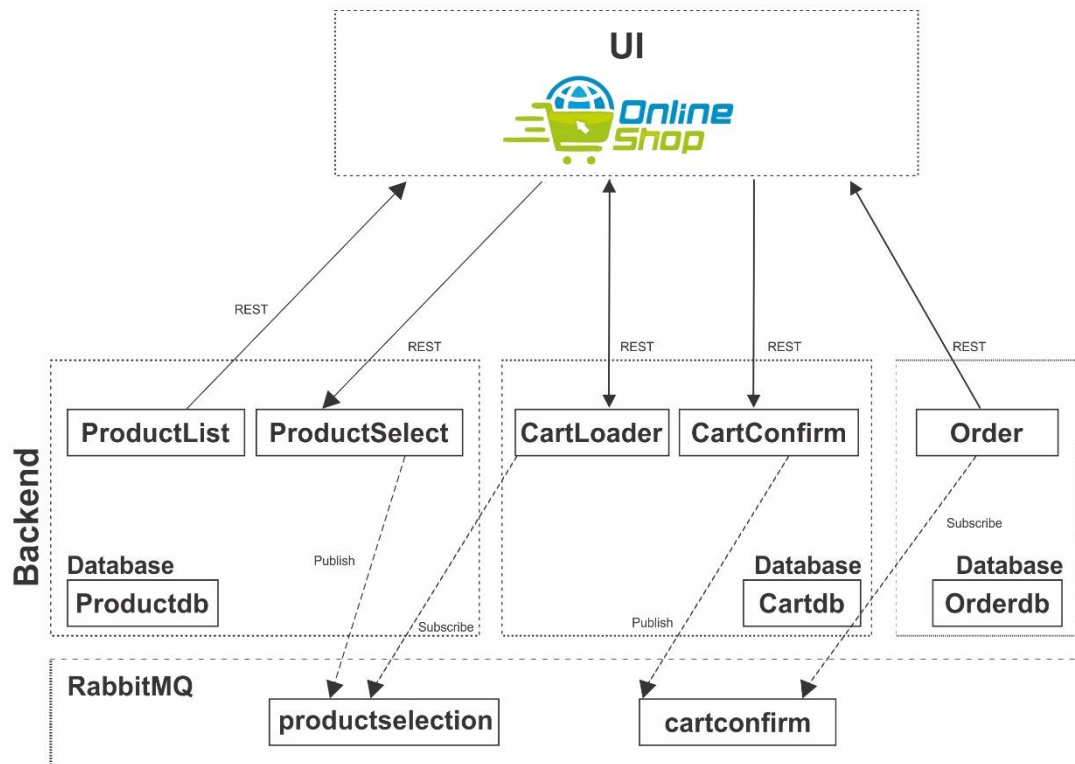


Figura 4 - Arquitetura final

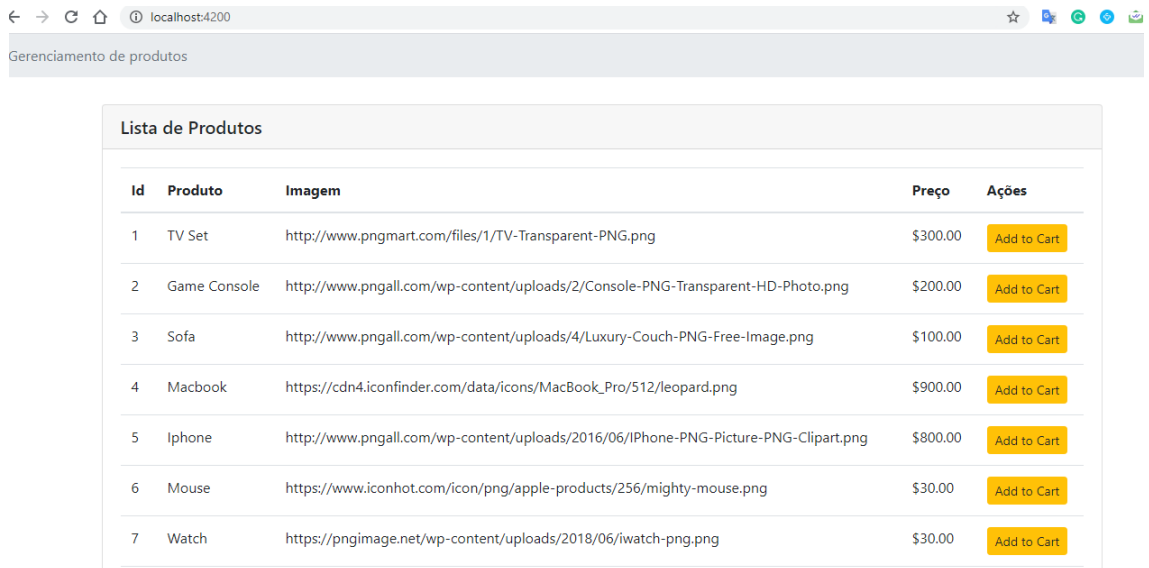
A loja também estava em um único pod, mas criamos para cada serviço um pod separado para ficarem também independentes.

O objetivo dessa nova arquitetura foi partir ainda mais os serviços em serviços menores ainda e melhorar a orquestração e manipulação dos dados.

## UI Inicial

Para consumir a API do spring boot criamos um projeto com o angular CLI, e fizemos CRUD para nos ajudar de forma simples e básica a manipular os produtos da loja, também criaremos uma API REST.

Depois adicionamos os módulos, interface para exibir a lista dos nossos produtos e o serviço responsável pelas requisições. Usamos o HTTPClient para criar o método para obter todos os produtos como mostrado na figura 9, o método onOnit() foi disparado chamando o método getProducts() exibindo a listagem de produtos do nosso serviço ProductService.



Lista de Produtos				
Id	Produto	Imagem	Preço	Ações
1	TV Set	http://www.pngmart.com/files/1/TV-Transparent-PNG.png	\$300.00	<a href="#">Add to Cart</a>
2	Game Console	http://www.pngall.com/wp-content/uploads/2/Console-PNG-Transparent-HD-Photo.png	\$200.00	<a href="#">Add to Cart</a>
3	Sofa	http://www.pngall.com/wp-content/uploads/4/Luxury-Couch-PNG-Free-Image.png	\$100.00	<a href="#">Add to Cart</a>
4	Macbook	https://cdn4.iconfinder.com/data/icons/MacBook_Pro/512/leopard.png	\$900.00	<a href="#">Add to Cart</a>
5	Iphone	http://www.pngall.com/wp-content/uploads/2016/06/IPhone-PNG-Picture-PNG-Clipart.png	\$800.00	<a href="#">Add to Cart</a>
6	Mouse	https://www.iconhot.com/icon/png/apple-products/256/mighty-mouse.png	\$30.00	<a href="#">Add to Cart</a>
7	Watch	https://pngimage.net/wp-content/uploads/2018/06/iwatch-png.png	\$30.00	<a href="#">Add to Cart</a>

**Figura 9 – Tela da UI inicial da aplicação**

### UI Final

Após o entendimento como consumir nossa API, fizemos as adaptações de acordo com as URLs dos nossos microserviços definidas no spring boot e foi possível chamar as informações desenvolvidas no backend para o frontend.

Melhoramos o layout da nossa interface de apresentação e adicionamos rotas ao projeto do angular.

Temos a página home que exhibe a apresentação da loja Online Shop, com informações para acessar a aba products, como mostra a figura 10.

Na parte do products temos uma lista de produtos disponíveis na loja, figura 11, com nome, valor e preço. Com um botão add to cart, para o cliente adicionar as compras desejadas.

No final das compras ao clicar em checkout to Cart, o cliente é redirecionado para a aba Cart, onde apresentamos o Purchase Summary com os itens comprados, os valores de cada item, a data, o id identificador da compra e o Total.

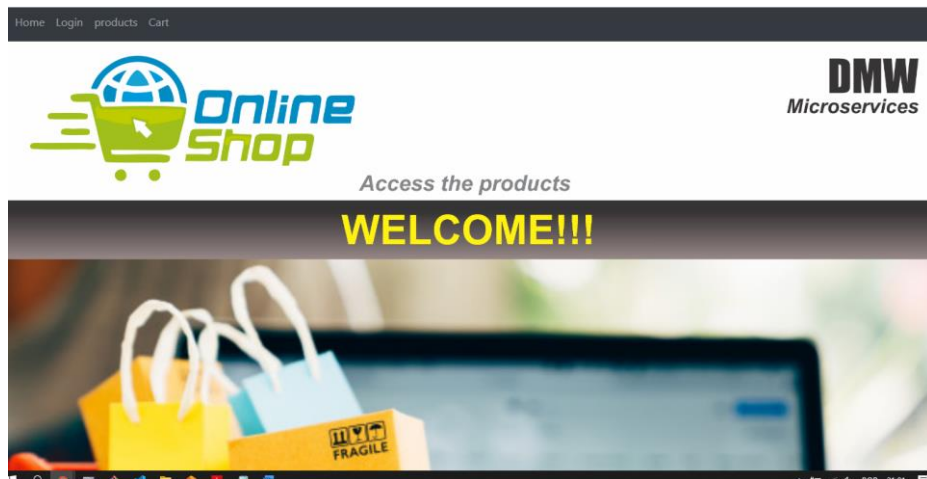


Figura 50 - UI Final Home

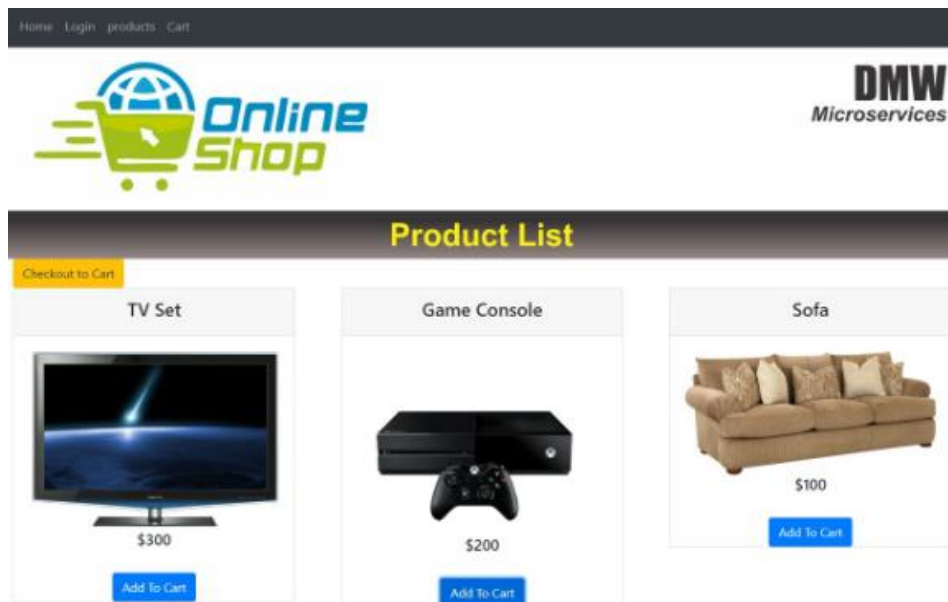


Figura 11 – UI Final Products

A figura 12 exemplifica o funcionamento do "product" para "cart". O microserviço order também tem um listener consumer que ao ser alertado de dado na queue acima, pega o dado e persiste na base de dados.

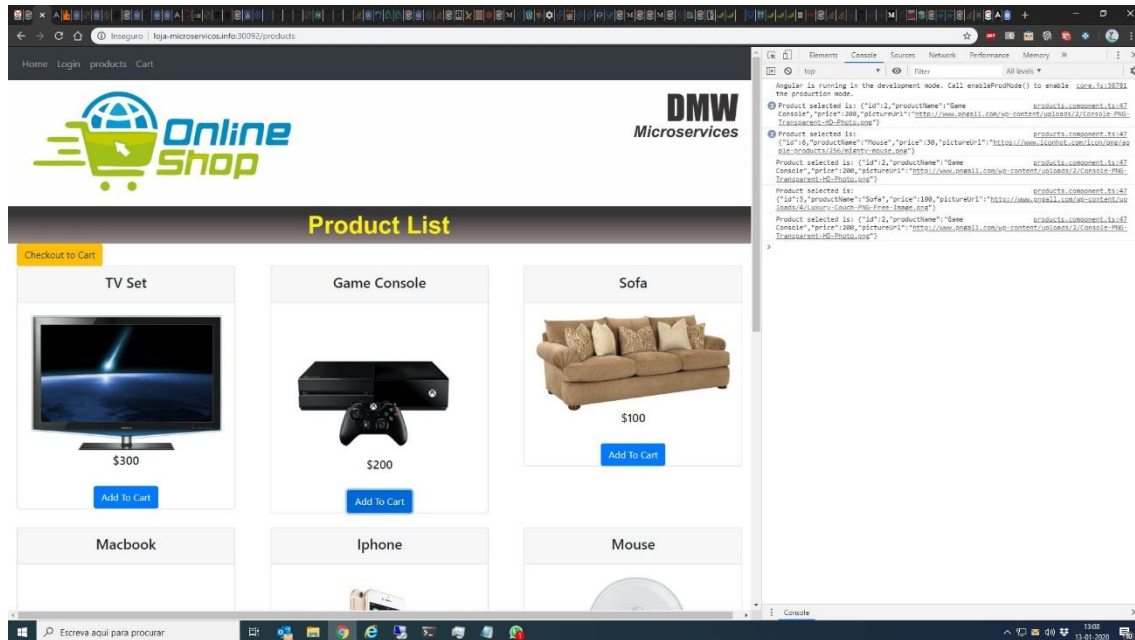


Figura 12 - UI Final Cart

Na figura 13, demonstramos um exemplo do funcionamento da nossa aplicação com o rabbitMQ acabando de vir da lista de produtos depois de fazer checkout. No console é possível ver os produtos que foram selecionados na tela de produtos. Ao clicar em checkout, o array de produtos é enviado para product microservice (backend) e de lá enviado pelo rabbitmq para a productqueue.rabbitmq.queue = product.queue

Então o listener consumer do rabbitmq no cart service que está escutando a queue acima, vai buscar os dados assim que for colocada lá e no mesmo botão checkout também houve o roteamento para a página cart e está por estar configurada pra ir buscar os dados, alimenta-os logo que é carregada.

O mesmo botão também roteia a página na user interface para order e lá também está configurada para buscar os dados no backend e carregá-los nos seus campos.

O botão Go to Order envia o valor ou outra informação que decidir enviar, como por exemplo o nome do usuário (não deu tempo de implementar) para o backend (microservice cart) e então é enviado para o microservice order por meio do rabbitmq para cartqueue.rabbitmq.queue = cart.queue.

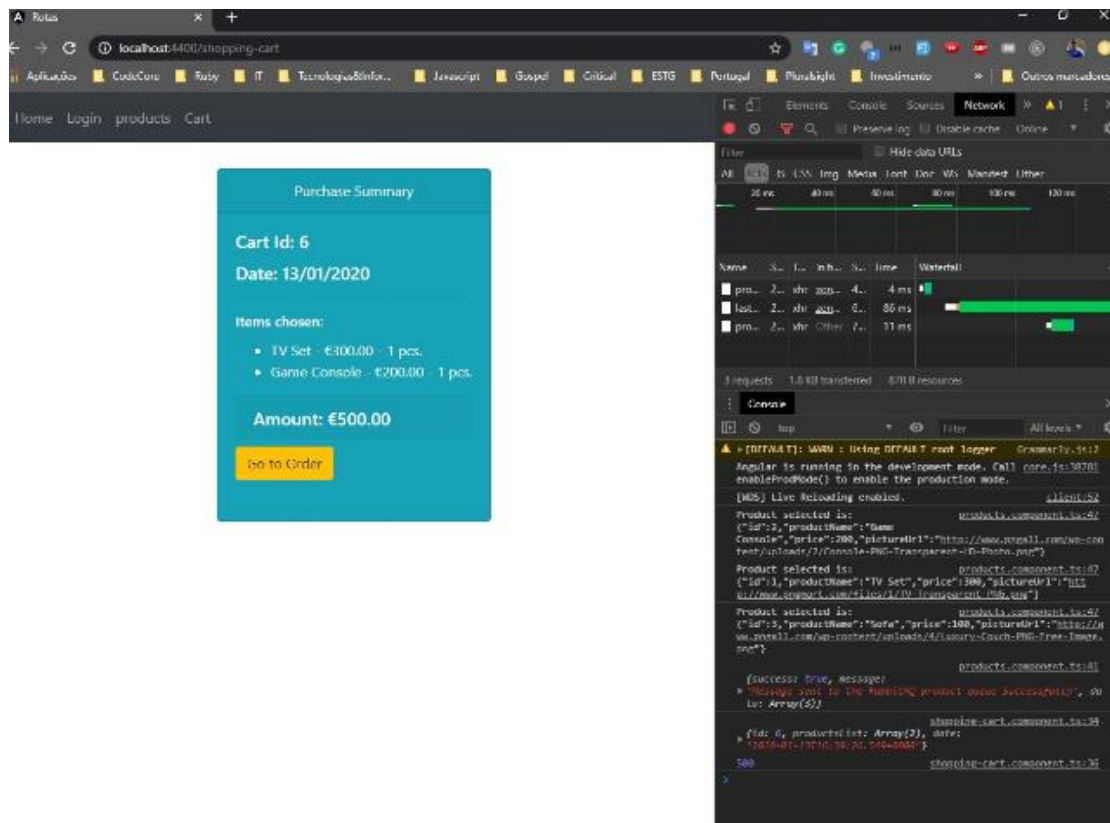


Figura 13 - UI Final Go to Order

## Implementação - Tecnologias



É um message broker, implementado para suportar as mensagens no protocolo denominado Advanced Message Queuing Protocol (AMQP).

O RabbitMq possibilita lidar com o tráfego de mensagens de forma rápida e confiável, além de ser compatível com diversas linguagens de programação, possuir interface de administração nativa e ser multiplataforma.

Escolhemos o RabbitMQ para fazer o gerenciamento das nossas filas pois é open source, escrito em Erlang (confiável), roda em diversos sistemas operacionais e compatível com a linguagem utilizada no projeto.

Ele tem um painel de gerenciamento para controlar as filas, também apresenta um bom desempenho e com possibilidade de configurações para atender alta escalabilidade e por ser o centro de um sistema de microserviços responsivos.



O Kubernetes é um orquestrador de containers. No Kubernetes temos os pods, onde conseguimos guardar os nossos containers, assim sendo, poderemos ter um ou mais containers dentro de um pod. O mais interessante é que, dentro dos nossos pods poderemos ter, além dos containers, várias configurações, para que se comportem da maneira que desejamos.

Usamos o minikube, o que nos permite executar o Kubernetes localmente. Alguns exemplos nas figuras abaixo 15 e 16.

```
FROM openjdk:11-jdk-alpine
VOLUME /tmp
ARG JAR_FILE
ADD ${JAR_FILE} app.jar
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/spring-boot-amqp-consumer-javainuse-0.0.1-SNAPSHOT.jar"]
```

*Figura 15 - Dockerfile\_minikube*

```
from openjdk:11
copy ./target/spring-boot-amqp-consumer-javainuse-0.0.1-SNAPSHOT.jar spring-boot-amqp-consumer-javainuse-0.0.1-SNAPSHOT.jar
CMD ["java","-jar","spring-boot-amqp-consumer-javainuse-0.0.1-SNAPSHOT.jar"]
```

*Figura 16 - Dockerfile\_minikube*



Escolhemos o GitHub por ser um site e serviço que fornece GIT gratuito que é um sistema de controle de versão distribuído e um sistema de gerenciamento de código fonte.

Para disponibilizar nosso projeto em um repositório para facilitar o acesso e fazer commit das modificações ao longo do trabalho e para o grupo colaborar e acompanhar criamos repositórios para ter um histórico do que foi feito e poder até reverter em caso de erros.

<https://github.com/dtafonso/mtsds-microservices-store>

*Figura 17 - Alguns dos problemas e solução que tivemos durante o trabalho*



## Considerações finais

No desenvolvimento desse projeto foi possível adquirir novas competências no paradigma de desenvolvimento, mas também novas experiências aprendidas na utilização de ferramentas, frameworks, enfim, tecnologias que colaboram muito na implementação da arquitetura com microserviços.

Especialmente sobre o projeto realizado, tivemos diversos desafios, muito trabalho e superação nos obstáculos que surgiram para obter sucesso nos deveres que foram cumpridos.

Descobrimos que para a equipe trabalhar corretamente é preciso todos estarem conectados na forma como estão a desenvolver e alinhar todas as ferramentas.

Apesar de o desenvolvimento com microserviços ser mais complexo, entendemos que para a evolução na criação de aplicações e para manutenção é essencial pois são independentes e muito mais rápidas.

## Bibliografia

- <https://www.rabbitmq.com/getstarted.html>
- <https://microservices.io/patterns/>
- <https://spring.io/guides/gs/messaging-rabbitmq/>
- <https://microservices.io/patterns/microservices.html>
- <https://www.youtube.com/watch?v=MPHyrZWTAis>
- <https://spring.io/guides/gs/messaging-rabbitmq/>
- <https://angular.io/tutorial>
- <https://github.com/DickChesterwood/k8s-fleetman/tree/master/k8s-fleetman-webapp-angular>
- <https://microservices.io/patterns/ui/client-side-ui-composition.html>
- <https://gorillalogic.com/blog/build-and-deploy-a-spring-boot-app-on-kubernetes-minikube/>
- <https://dzone.com/articles/quick-guide-to-microservices-with-kubernetes-spring>
- <https://www.javainuse.com/spring/spring-boot-rabbitmq-hello-world>
- <https://microservices.io/patterns/data/database-per-service.html>
- <https://www.javainuse.com/spring/sprsec>
- <https://microservices.io/patterns/security/access-token.html>
- <https://dzone.com/articles/quick-guide-to-microservices-with-kubernetes-spring>