**DOCUMENTAÇÃO DO SOFTWARE DO NEGOCIO DE VENDA - ESTOQUE**

**Versão 1.0**

**Desenvolvedor: Alisson Dill**

**2020**

**ÍNDICE DETALHADO**

1. **Introdução ao Documento...................................................................................... 03**
2. **Descrição Geral do Sistema..................................................................................... 03**
3. **Requisitos do Sistema............................................................................................. 04**
4. **Analise e Design...................................................................................................... 05**
5. **Banco de Dados...................................................................................................... 16**
6. **Conclusões e Considerações finais........................................................................... 22**
7. **Introdução ao Documento**

Objetivo do desafio é desenvolver um software simples que seja capaz de realizar uma venda e efetuar a entrada e saída de produtos no estoque, além de criar toda a estrutura básica para que essa entrada e saída aconteça.

* 1. **Tema**

Software do Negócio de Venda – Estoque

**1.2 Objetivo do Projeto**

É desenvolver um sistema capaz de efetuar uma venda com entrada e saída dos produtos no estoque.

**1.3 Delimitação do Problema**

Uma das grandes preocupações das redes de supermercado é a venda das mercadorias e seu estoque. O estoque é reflexo real da quantidade física dos produtos em uma filial.

**1.4 Método de Trabalho**

Será utilizado para a realização do desafio o Java, MySQL, Postman, Docker, Spring Boot, GitHub.

1. **Descrição Geral do Sistema**

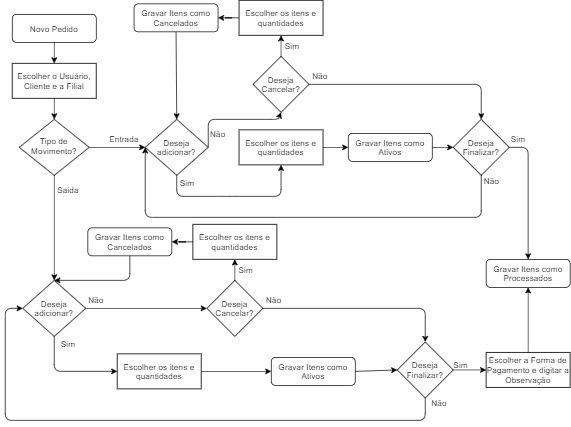
O sistema foi desenvolvido no framework Spring Boot, versão Java 8, apache maven 3.5.3 e Banco de dados MySQL.

* 1. **Descrição do Problema**

Um sistema capaz de realizar:

* Cadastrar um pedido
* Adicionar itens dos produtos
* Cancelar itens dos produtos
* Finalizar Pedidos  
  1. **Regras de Negócio**
* Ao escolher um produto, o usuário digita a quantidade desejada, que deve ser maior que zero e menor ou igual a quantidade do produto no estoque daquela filial.
* Cada item do pedido deve ter o produto, um status (ativo, cancelado ou processado). Todo novo item adicionado, possui o status de ativo e ao ser retirado do pedido, o status deve ser atualizado para cancelado. Não pode existir produtos repetidos no mesmo pedido, nos status de ativo ou processado. A cada item adicionado no pedido, a quantidade total dos itens e o valor total do pedido devem ser recalculados.
* Ao finalizar a compra, os itens do pedido são atualizados para o status de processado e a quantidade dos seus itens devem decrementar a quantidade dos estoques dos produtos da filial.
* Cada filial tem um controle de estoque individual onde cada produto tem sua representação por filial e contém quantidade total por produto. A quantidade total não pode ser negativa.
* A entrada e saída da mercadoria de uma filial é sempre feita por meio de pedidos de estoque do tipo entrada ou saída.
* Ao efetuar a entrada de um pedido de estoque a quantidade dos seus itens devem incrementar a quantidade dos estoques dos produtos da filial. Caso o estoque não exista para algum produto, o mesmo deverá ser criado.
* Ao efetuar a saída de um pedido de estoque a quantidade dos seus itens devem decrementar a quantidade dos estoques dos produtos da filial.

1. **Requisitos do Sistema**

Ao criar um novo pedido, deve informar o usuário, cliente, filial e tipo de movimento, Entrada ou Saída. O sistema vai encaminhar para tela de adicionar produtos, excluir produtos ou finalizar o pedido. 

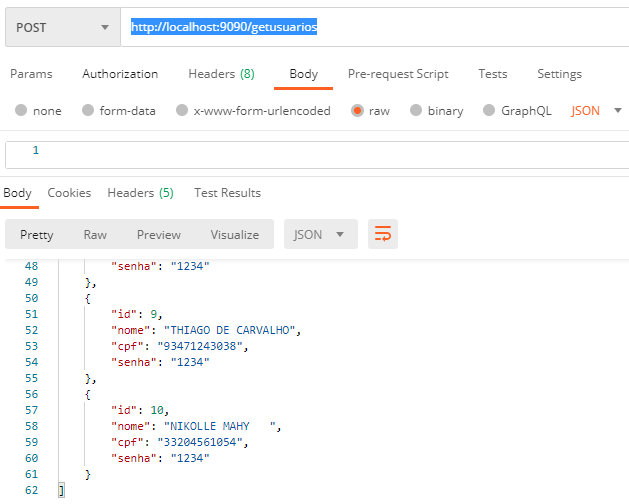
1. **Analise e Design**

Foram criadas 16 APIs REST método POST para facilitar a implantação da regra de negócio nas telas em Angular. Segue abaixo cada API:

* **API Listar todos os Usuários - /getusuarios**
* **API Adicionar Usuários - /addusuarios**
* **API Listar todos os Clientes - /getclientes**
* **API Adicionar Clientes - /addusuarios**
* **API Listar todos os Produtos - /getprodutos**
* **API Adicionar Produto - /addprodutos**
* **API Buscar somente um Produto pelo ID - /getprodutoid**
* **API Buscar somente um Produto pelo código de barra - /getprodutocod**
* **API Buscar somente um Produto pelo Nome - /getprodutocod**
* **API Listar todos os Pedidos - /getpedidos**
* **API Criar um Novo Pedido - /novopedido**
* **API Consultar o Pedido pelo ID - /getpedidoid**
* **API Adicionando Item no Pedido - /itempedido**
* **API Consulta Item do Pedido por ID - /listaritensid**
* **API Cancelar Lista do Item de Pedido - /listaritensid**
* **API Fechar Pedido - /fecharpedido**
  1. **Entrada e Saída de cada API**

API Listar todos os Usuários:

**POST** <http://localhost:9090/getusuarios>



API adicionar Usuários

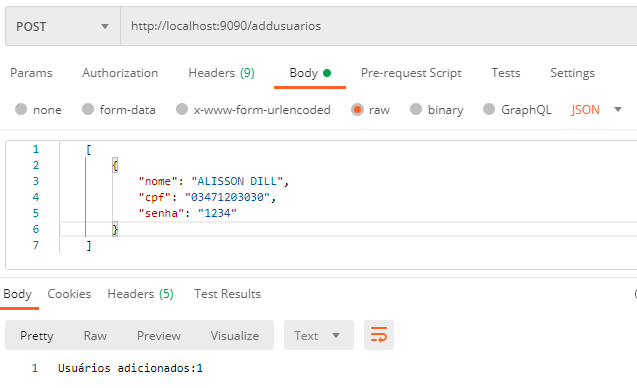
**POST** <http://localhost:9090/addusuarios>

**ENTRADA**   
[

{

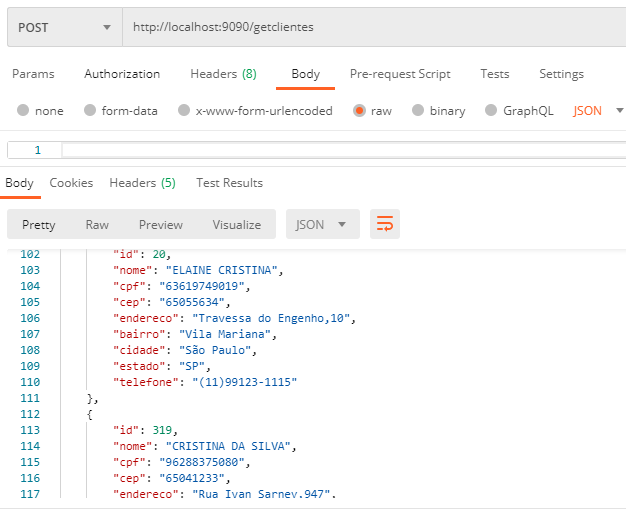
"nome": "ALISSON DILL",  
 "cpf": "03471203030",  
 "senha": "1234"   
 }

]



API Listar todos os Clientes:

**POST** <http://localhost:9090/getclientes>



API adicionar Clientes

**POST** <http://localhost:9090/addusuarios>

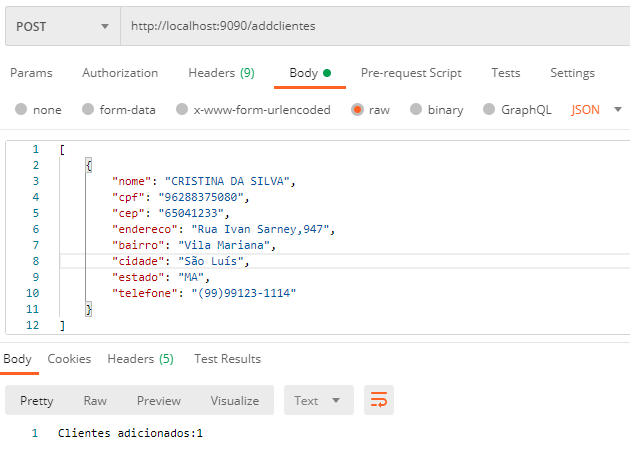
**ENTRADA**   
[

{

"nome": "CRISTINA DA SILVA",  
 "cpf": "96288375080",  
 "cep": "65041233",  
 "endereco": "Rua Ivan Sarney,947",  
 "bairro": "Vila Mariana",  
 "cidade": "São Luís",  
 "estado": "MA",  
 "telefone": "(99)99123-1114"

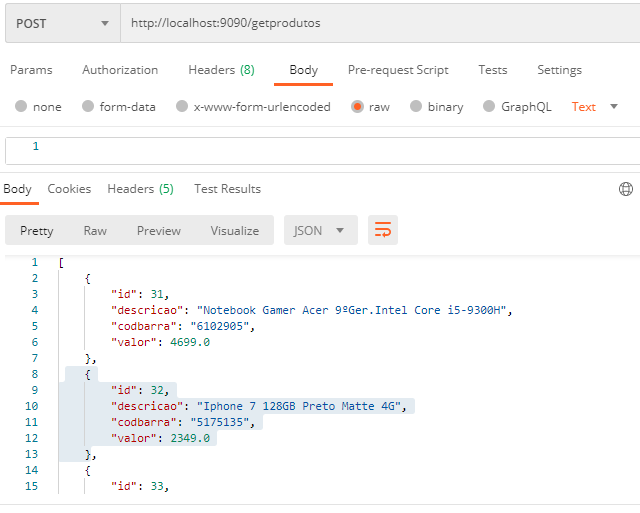
}

]



API Listar todos os Produtos:

**POST** <http://localhost:9090/getprodutos>



API adicionar Produto

**POST** <http://localhost:9090/addprodutos>

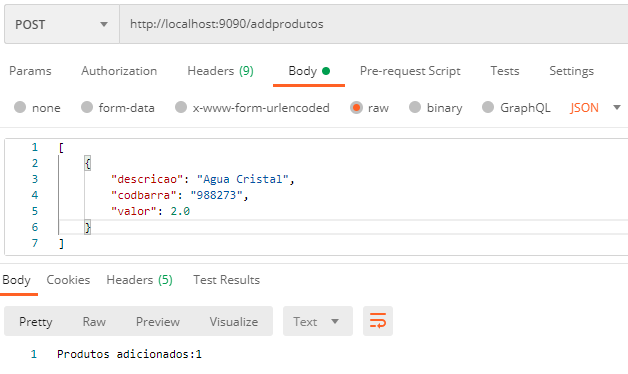
**ENTRADA**   
[

{

"descricao": "Agua Cristal",  
 "codbarra": "988273",  
 "valor": 2.0

}

]

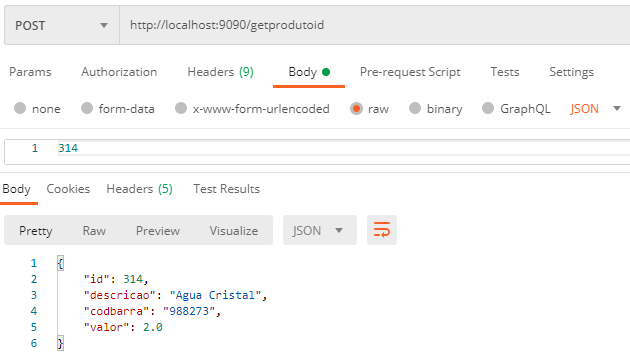


API Buscar somente um Produto pelo ID

**POST** <http://localhost:9090/getprodutoid>

**ENTRADA**

314

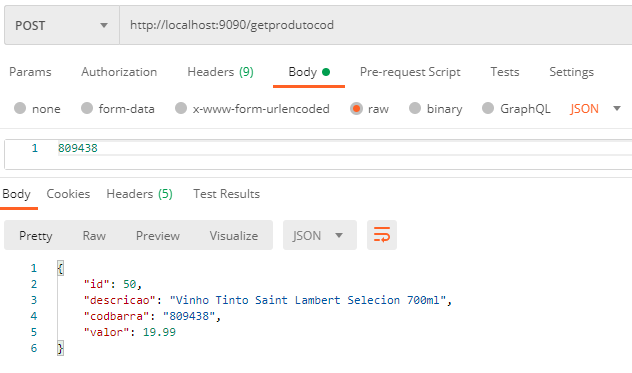


API Buscar somente um Produto pelo código de barra

**POST** <http://localhost:9090/getprodutocod>

**ENTRADA**

809438

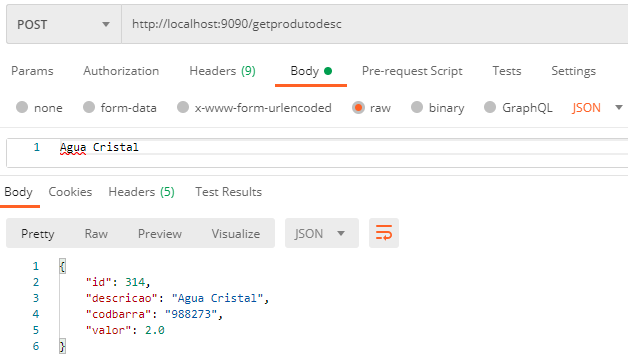


API Buscar somente um Produto pelo Nome

**POST** <http://localhost:9090/getprodutocod>

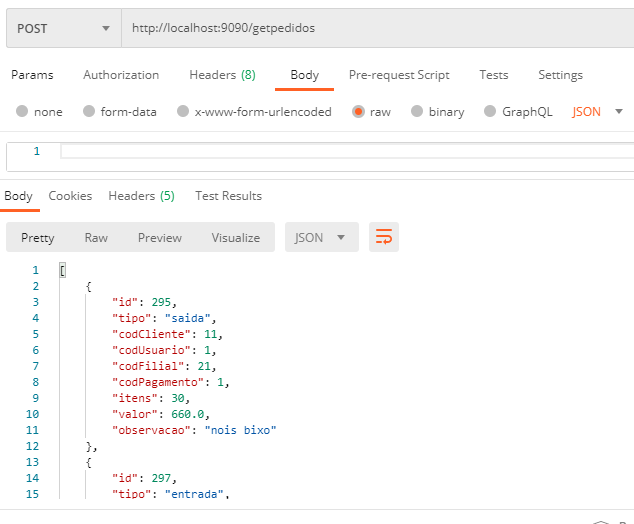
**ENTRADA**

Agua Cristal



API Listar todos os Pedidos

**POST** <http://localhost:9090/getpedidos>



API Criar um Novo Pedido

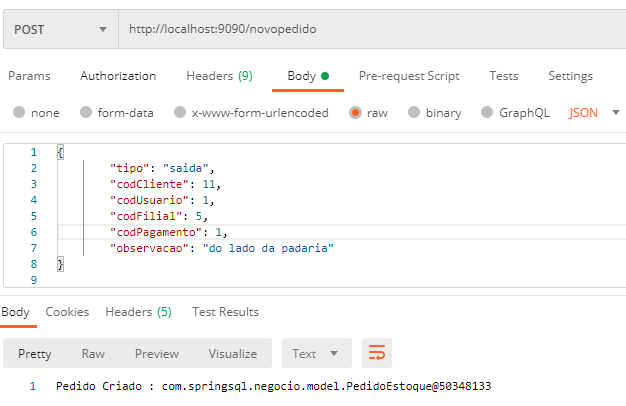
**POST** <http://localhost:9090/novopedido>

**ENTRADA**

{

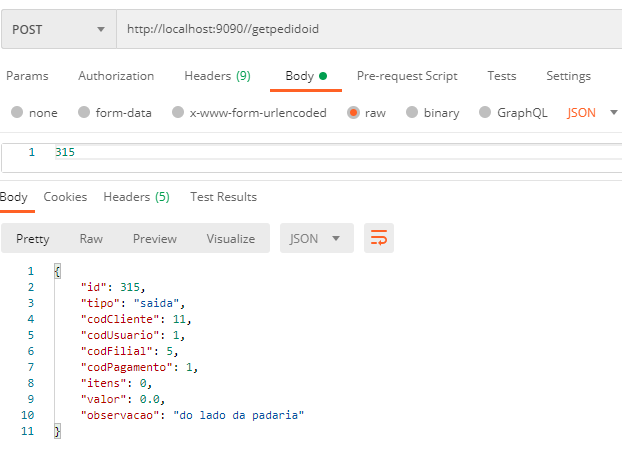
"tipo": "saida",  
 "codCliente": 11,  
 "codUsuario": 1,  
 "codFilial": 5,  
 "codPagamento": 1,  
 "observacao": "do lado da padaria"

}



API Consultar o Pedido pelo ID

**POST** <http://localhost:9090/getpedidoid>



API Adicionando Item no Pedido

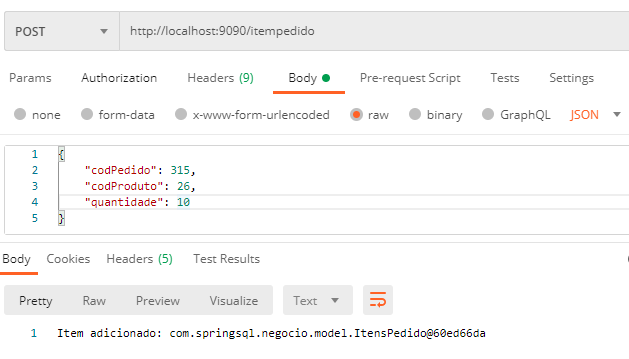
**POST** <http://localhost:9090/itempedido>

**ENTRADA**

{

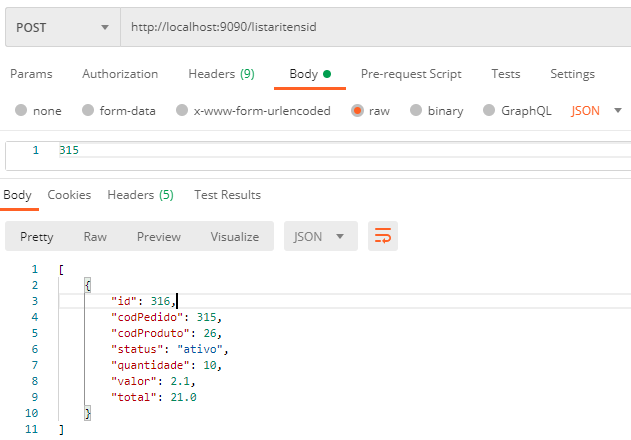
"codPedido": 315,  
 "codProduto": 26,  
 "quantidade": 10

}



API Consulta Item do Pedido por ID

**POST** <http://localhost:9090/listaritensid>



API Cancelar Lista do Item de Pedido

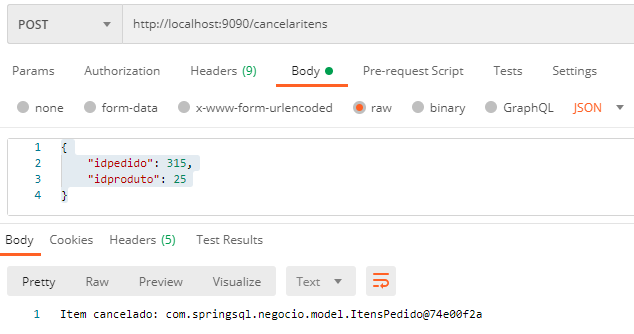
**POST** <http://localhost:9090/listaritensid>

**ENTRADA**

{

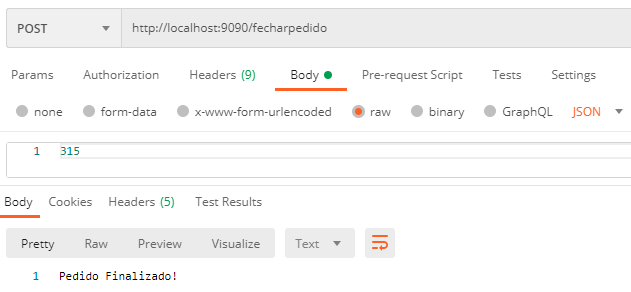
"idpedido": 315,  
"idproduto": 25

}



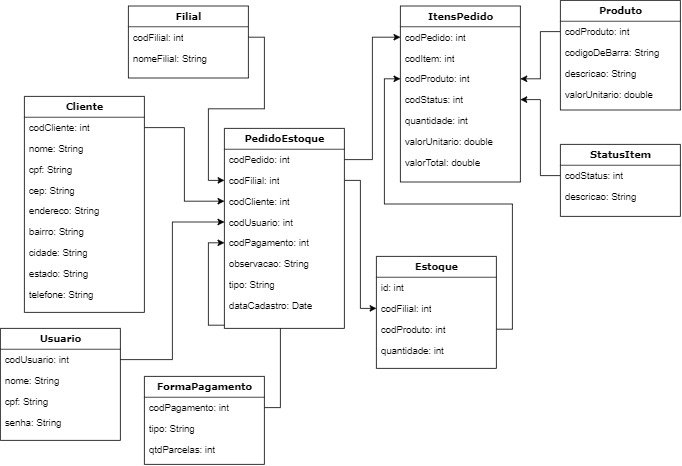
API Fechar Pedido

**POST** <http://localhost:9090/fecharpedido>



1. **Banco de Dados**

Foi utilizado o MySQL para a criação das tabelas cliente, estoque, forma\_pagamento, hibernate\_sequence, itens\_pedido, pedido\_estoque, produto, usuário, filial.



* 1. **Configuração do Banco – Passo a Passo**

Criando banco de dados

CREATE DATABASE negocio;

Usando o banco de dados

USE negocio;

Na pasta do projeto, vai ter um arquivo com ao código da criação das tabelas e dados. O nome do arquivo SQL/ tabelas\_negocio.sql. Copiar e colar para executar no MySQL antes de rodar o projeto.

Exemplos dos CREATE das tabelas:

SET SQL\_MODE = "NO\_AUTO\_VALUE\_ON\_ZERO";

SET AUTOCOMMIT = 0;

START TRANSACTION;

SET GLOBAL time\_zone = '+3:00';

DROP TABLE IF EXISTS `cliente`;

CREATE TABLE IF NOT EXISTS `cliente` (

`id` int(11) NOT NULL,

`nome` varchar(200) DEFAULT NULL,

`cpf` varchar(200) DEFAULT NULL,

`cep` varchar(200) DEFAULT NULL,

`endereco` varchar(200) DEFAULT NULL,

`bairro` varchar(200) DEFAULT NULL,

`cidade` varchar(200) DEFAULT NULL,

`estado` varchar(20) DEFAULT NULL,

`telefone` varchar(20) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `estoque`;

CREATE TABLE IF NOT EXISTS `estoque` (

`id` int(11) NOT NULL,

`quantidade` int(11) DEFAULT NULL,

`cod\_filial` int(11) NOT NULL,

`cod\_produto` int(11) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `filial`;

CREATE TABLE IF NOT EXISTS `filial` (

`id` int(11) NOT NULL,

`nome` varchar(200) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `forma\_pagamento`;

CREATE TABLE IF NOT EXISTS `forma\_pagamento` (

`id` int(11) NOT NULL,

`tipo` varchar(200) DEFAULT NULL,

`parcela` int(11) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `hibernate\_sequence`;

CREATE TABLE IF NOT EXISTS `hibernate\_sequence` (

`next\_val` bigint(20) DEFAULT NULL

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `itens\_pedido`;

CREATE TABLE IF NOT EXISTS `itens\_pedido` (

`id` int(11) NOT NULL,

`codPedido` int(11) DEFAULT NULL,

`codProduto` int(11) DEFAULT NULL,

`status` varchar(100) DEFAULT NULL,

`quantidade` int(11) DEFAULT NULL,

`valor` float DEFAULT NULL,

`total` float DEFAULT NULL,

`cod\_pedido` int(11) NOT NULL,

`cod\_produto` int(11) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `pedido\_estoque`;

CREATE TABLE IF NOT EXISTS `pedido\_estoque` (

`id` int(11) NOT NULL,

`tipo` varchar(100) DEFAULT NULL,

`codCliente` int(11) DEFAULT NULL,

`codUsuario` int(11) DEFAULT NULL,

`codFilial` int(11) DEFAULT NULL,

`codPagamento` int(11) DEFAULT NULL,

`Observacao` varchar(300) DEFAULT NULL,

`itens` int(11) DEFAULT NULL,

`valor` float DEFAULT NULL,

`cod\_cliente` int(11) NOT NULL,

`cod\_filial` int(11) NOT NULL,

`cod\_pagamento` int(11) NOT NULL,

`cod\_usuario` int(11) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `produto`;

CREATE TABLE IF NOT EXISTS `produto` (

`id` int(11) NOT NULL,

`descricao` varchar(200) DEFAULT NULL,

`codbarra` int(20) DEFAULT NULL,

`valor` float DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `usuario`;

CREATE TABLE IF NOT EXISTS `usuario` (

`id` int(11) NOT NULL,

`nome` varchar(200) DEFAULT NULL,

`cpf` varchar(200) DEFAULT NULL,

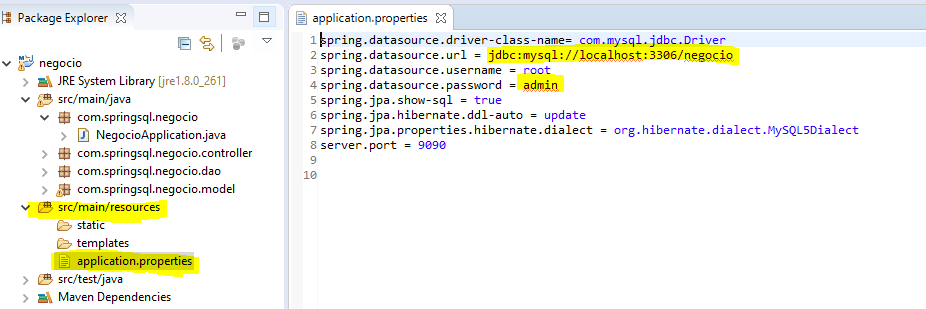
`senha` varchar(200) DEFAULT NULL,

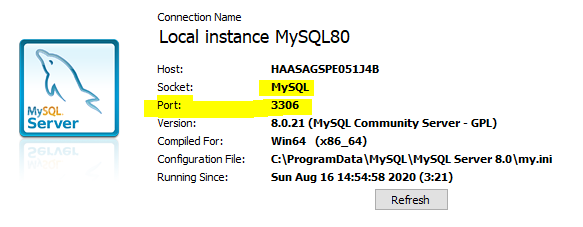
PRIMARY KEY (`id`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

* 1. **Configuração no Eclipse**

Verificar a porta do MySQL e a Senha.





* 1. **Consulta MySQL**
* Escrever uma consulta que retorne todos os produtos com quantidade maior ou igual a 100

use negocio;

SELECT produto.descricao, estoque.quantidade, produto.valor, filial.nome FROM estoque

INNER JOIN produto ON produto.id = estoque.cod\_produto

INNER JOIN filial ON filial.id = estoque.cod\_filial

WHERE estoque.quantidade > 100

* Escrever uma consulta que traga todos os produtos que têm estoque para a filial de código 60

use negocio;

SELECT produto.descricao, estoque.quantidade, produto.valor, filial.nome FROM estoque

INNER JOIN produto ON produto.id = estoque.cod\_produto

INNER JOIN filial ON filial.id = estoque.cod\_filial

WHERE estoque.cod\_filial = 60

* Escrever consulta que liste todos os campos para o domínio PedidoEstoque e ItensPedido filtrando apenas o produto de código 7993

use negocio;

SELECT pedido\_estoque.id,

pedido\_estoque.cod\_cliente,

pedido\_estoque.cod\_filial,

itens\_pedido.cod\_produto,

itens\_pedido.quantidade FROM itens\_pedido

INNER JOIN pedido\_estoque ON pedido\_estoque.id = itens\_pedido.cod\_pedido

WHERE itens\_pedido.cod\_produto = 7993

* Escrever uma consulta que liste os pedidos com suas respectivas formas de pagamento.

use negocio;

SELECT pedido\_estoque.id,

pedido\_estoque.cod\_cliente,

pedido\_estoque.cod\_filial,

forma\_pagamento.tipo FROM pedido\_estoque

inner join forma\_pagamento ON forma\_pagamento.id = pedido\_estoque.codPagamento

* Escrever uma consulta para sumarizar e bater os valores da capa do pedido com os valores dos ítens de pedido

use negocio;

SELECT itens\_pedido.cod\_pedido, SUM(itens\_pedido.total), pedido\_estoque.valor

FROM itens\_pedido

INNER JOIN pedido\_estoque ON pedido\_estoque.id = itens\_pedido.cod\_pedido

GROUP BY itens\_pedido.cod\_pedido

HAVING SUM(itens\_pedido.total) = pedido\_estoque.valor

* Escrever uma consulta para sumarizar o total dos itens por pedido e que filtre apenas os pedidos no qual a soma total da quantidade de ítens de pedido seja maior que 10

use negocio;

SELECT itens\_pedido.cod\_pedido, count(itens\_pedido.cod\_pedido), pedido\_estoque.valor

FROM itens\_pedido

INNER JOIN pedido\_estoque ON pedido\_estoque.id = itens\_pedido.cod\_pedido

GROUP BY itens\_pedido.cod\_pedido

HAVING count(itens\_pedido.cod\_pedido) > 10

1. **Conclusões e Considerações finais**

As funções básicas para o funcionamento do controle de estoque de APIs estão sendo executadas de acordo com o solicitado no desafio.

Foi disponibilizar uma aplicação conteinerizada utilizando o Docker.  


Queria fazer uma tela em Angular para chamar as APIs, mas não deu tempo.

O teste unitário não fiz. Os testes foram feitos através da API no Postman.