

# **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. Análise e Design.....	05
5. Banco de Dados.....	16
6. Conclusões e Considerações finais.....	22

## **1. 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.

### **1.1. 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.

## **2. 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.

### **2.1. Descrição do Problema**

Um sistema capaz de realizar:

- Cadastrar um pedido
- Adicionar itens dos produtos
- Cancelar itens dos produtos
- Finalizar Pedidos

### **2.2 Regras de Negócio**

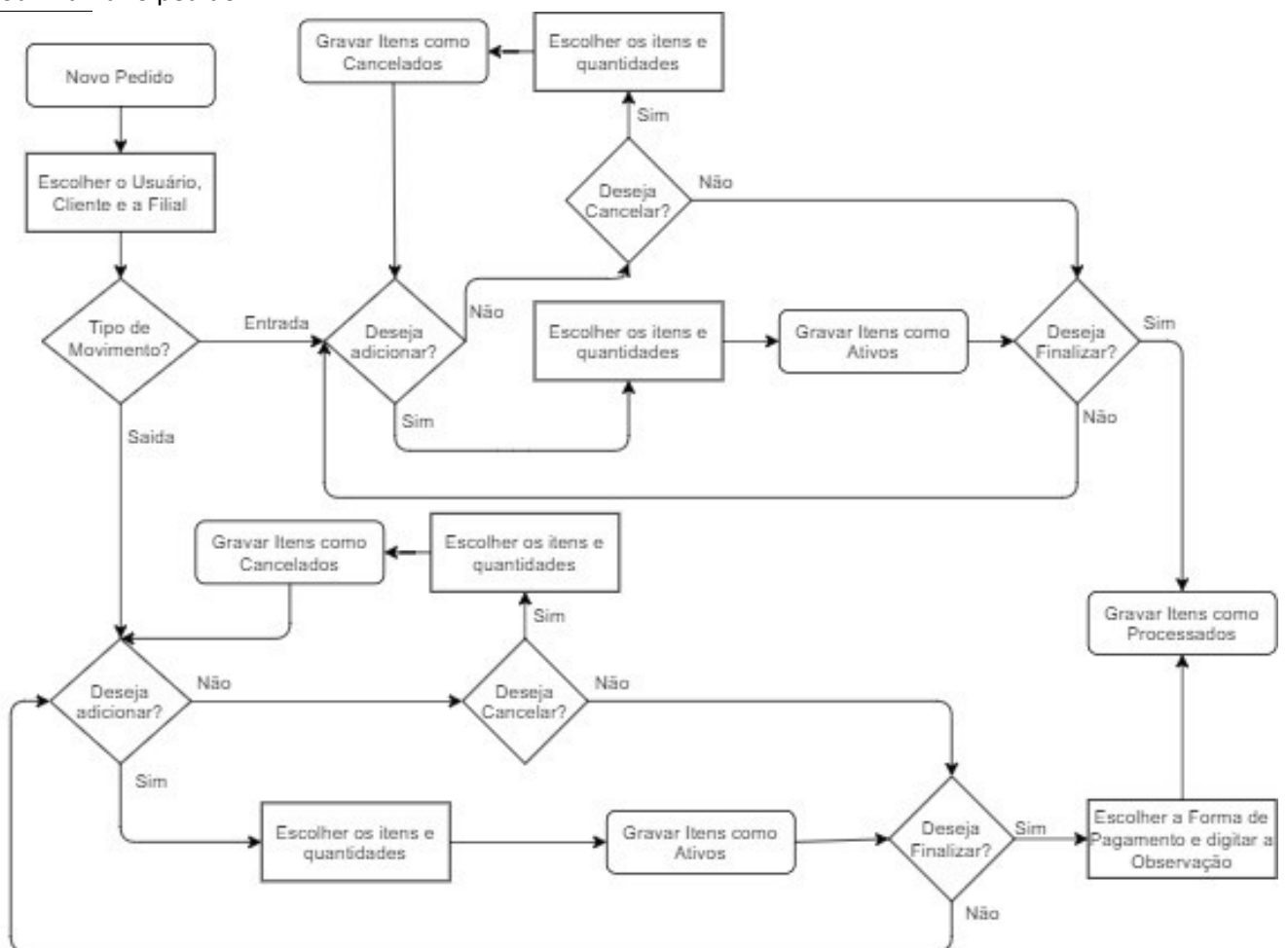
- Ao escolher um produto, o usuário digita a quantidade desejada, que deve ser maior que zero e menor ou igual a quantidade do produto no estoque daquela filial.
- Cada item do pedido deve ter o produto, um status (ativo, cancelado ou processado). Todo novo item adicionado, possui o status de ativo e 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.

### 3. 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.



## 4. 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

### 4.1. Entrada e Saída de cada API

API Listar todos os Usuários:

<b>POST</b> <a href="http://localhost:9090/getusuarios">http://localhost:9090/getusuarios</a>
---


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

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

1

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼ 

```
48     "senha": "1234"
49   },
50   {
51     "id": 9,
52     "nome": "THIAGO DE CARVALHO",
53     "cpf": "93471243038",
54     "senha": "1234"
55   },
56   {
57     "id": 10,
58     "nome": "NIKOLLE MAHY ",
59     "cpf": "33204561054",
60     "senha": "1234"
61   }
62 ]
```

#### API adicionar Usuários

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

##### **ENTRADA**

```
[
  {
    "nome": "ALISSON DILL",
    "cpf": "03471203030",
    "senha": "1234"
  }
]
```

POST http://localhost:9090/addusuarios

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded **raw** binary GraphQL JSON

```
1 [
2   {
3     "nome": "ALISSON DILL",
4     "cpf": "03471203030",
5     "senha": "1234"
6   }
7 ]
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text

1 Usuários adicionados:1

API Listar todos os Clientes:

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

POST http://localhost:9090/getclientes

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded **raw** binary GraphQL JSON

1

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
102 "id": 20,
103 "nome": "ELAINE CRISTINA",
104 "cpf": "63619749019",
105 "cep": "65055634",
106 "endereco": "Travessa do Engenho,10",
107 "bairro": "Vila Mariana",
108 "cidade": "São Paulo",
109 "estado": "SP",
110 "telefone": "(11)99123-1115"
111 },
112 {
113   "id": 319,
114   "nome": "CRISTINA DA SILVA",
115   "cpf": "96288375080",
116   "cep": "65041233",
117   "endereco": "Rua Ivan Sarnev.947".
```

## 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"
  }
]
```

POST <http://localhost:9090/addclientes>

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 [
2   {
3     "nome": "CRISTINA DA SILVA",
4     "cpf": "96288375080",
5     "cep": "65041233",
6     "endereco": "Rua Ivan Sarney,947",
7     "bairro": "Vila Mariana",
8     "cidade": "São Luís",
9     "estado": "MA",
10    "telefone": "(99)99123-1114"
11  }
12 ]
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text

```
1 Clientes adicionados:1
```



API Listar todos os Produtos:

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


POST ▼ http://localhost:9090/getprodutos

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **Text** ▼

1

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼ 

```
1  [
2    {
3      "id": 31,
4      "descricao": "Notebook Gamer Acer 9ªGer.Intel Core i5-9300H",
5      "codbarra": "6102905",
6      "valor": 4699.0
7    },
8    {
9      "id": 32,
10     "descricao": "Iphone 7 128GB Preto Matte 4G",
11     "codbarra": "5175135",
12     "valor": 2349.0
13   },
14   {
15     "id": 33,
```

API adicionar Produto

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

**ENTRADA**

```
[
  {
    "descricao": "Agua Cristal",
    "codbarra": "988273",
    "valor": 2.0
  }
]
```

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

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 [
2   {
3     "descricao": "Agua Cristal",
4     "codbarra": "988273",
5     "valor": 2.0
6   }
7 ]
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text

```
1 Produtos adicionados:1
```

API Buscar somente um Produto pelo ID

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

**ENTRADA**

314

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

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 314
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize **JSON**

```
1 {
2   "id": 314,
3   "descricao": "Agua Cristal",
4   "codbarra": "988273",
5   "valor": 2.0
6 }
```

API Buscar somente um Produto pelo código de barra

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

**ENTRADA**

809438

POST http://localhost:9090/getprodutocod

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **JSON** ▼

1 809438

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼

```
1 {
2   "id": 50,
3   "descricao": "Vinho Tinto Saint Lambert Selecion 700ml",
4   "codbarra": "809438",
5   "valor": 19.99
6 }
```

API Buscar somente um Produto pelo Nome

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

**ENTRADA**

Agua Cristal

POST http://localhost:9090/getprodutodesc

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **JSON** ▼

1 Agua Cristal

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼

```
1 {
2   "id": 314,
3   "descricao": "Agua Cristal",
4   "codbarra": "988273",
5   "valor": 2.0
6 }
```

## API Listar todos os Pedidos

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


POST ▼ http://localhost:9090/getpedidos

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

1

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize **JSON** ▼ 

```
1 [
2   {
3     "id": 295,
4     "tipo": "saida",
5     "codCliente": 11,
6     "codUsuario": 1,
7     "codFilial": 21,
8     "codPagamento": 1,
9     "itens": 30,
10    "valor": 660.0,
11    "observacao": "nois bixo"
12  },
13  {
14    "id": 297,
15    "tipo": "entrada",
```

## 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"
}
```


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

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {
2   "tipo": "saida",
3   "codCliente": 11,
4   "codUsuario": 1,
5   "codFilial": 5,
6   "codPagamento": 1,
7   "observacao": "do lado da padaria"
8 }
9
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text 

```
1 Pedido Criado : com.springsql.negocio.model.PedidoEstoque@50348133
```

## API Consultar o Pedido pelo ID

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


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

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 315
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize **JSON** 

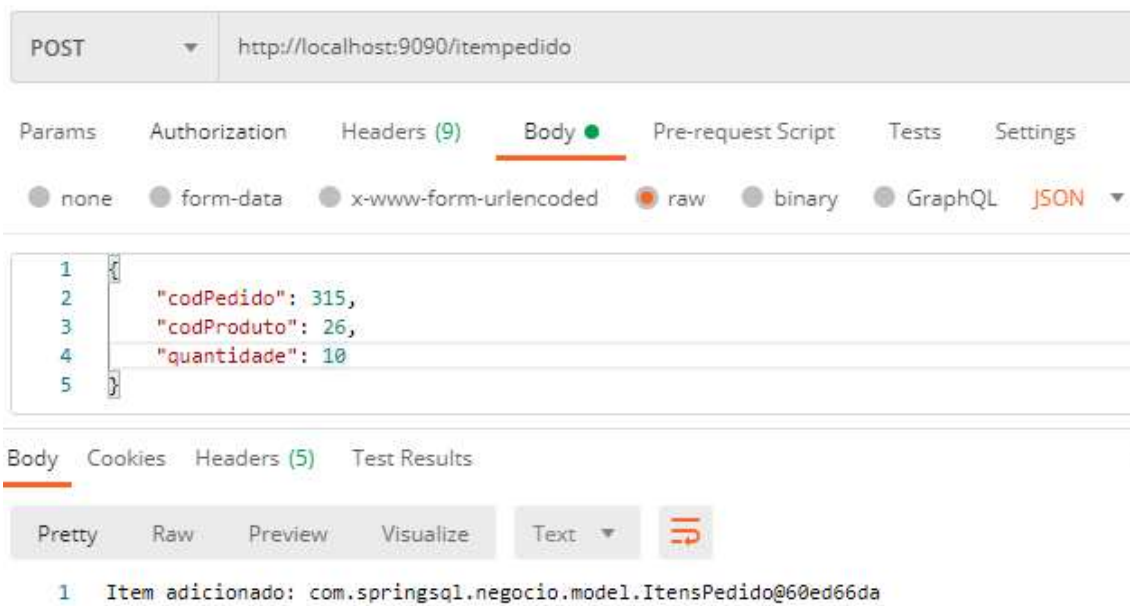
```
1 {
2   "id": 315,
3   "tipo": "saida",
4   "codCliente": 11,
5   "codUsuario": 1,
6   "codFilial": 5,
7   "codPagamento": 1,
8   "itens": 0,
9   "valor": 0.0,
10  "observacao": "do lado da padaria"
11 }
```

## 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>

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

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

1 315

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize **JSON** ▼

```
1 [
2   {
3     "id": 316,
4     "codPedido": 315,
5     "codProduto": 26,
6     "status": "ativo",
7     "quantidade": 10,
8     "valor": 2.1,
9     "total": 21.0
10  }
11 ]
```

#### API Cancelar Lista do Item de Pedido

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

#### ENTRADA

```
{
  "idpedido": 315,
  "idproduto": 25
}
```

POST <http://localhost:9090/cancelaritens>

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

```
1 {
2   "idpedido": 315,
3   "idproduto": 25
4 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize **Text** ▼

1 Item cancelado: com.springsql.negocio.model.ItensPedido@74e00f2a

## API Fechar Pedido

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

POST

http://localhost:9090/fecharpedido

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

● none

● form-data

● x-www-form-urlencoded

● raw

● binary

● GraphQL

JSON ▼

1

315

Body

Cookies

Headers (5)

Test Results

Pretty

Raw

Preview

Visualize

Text ▼

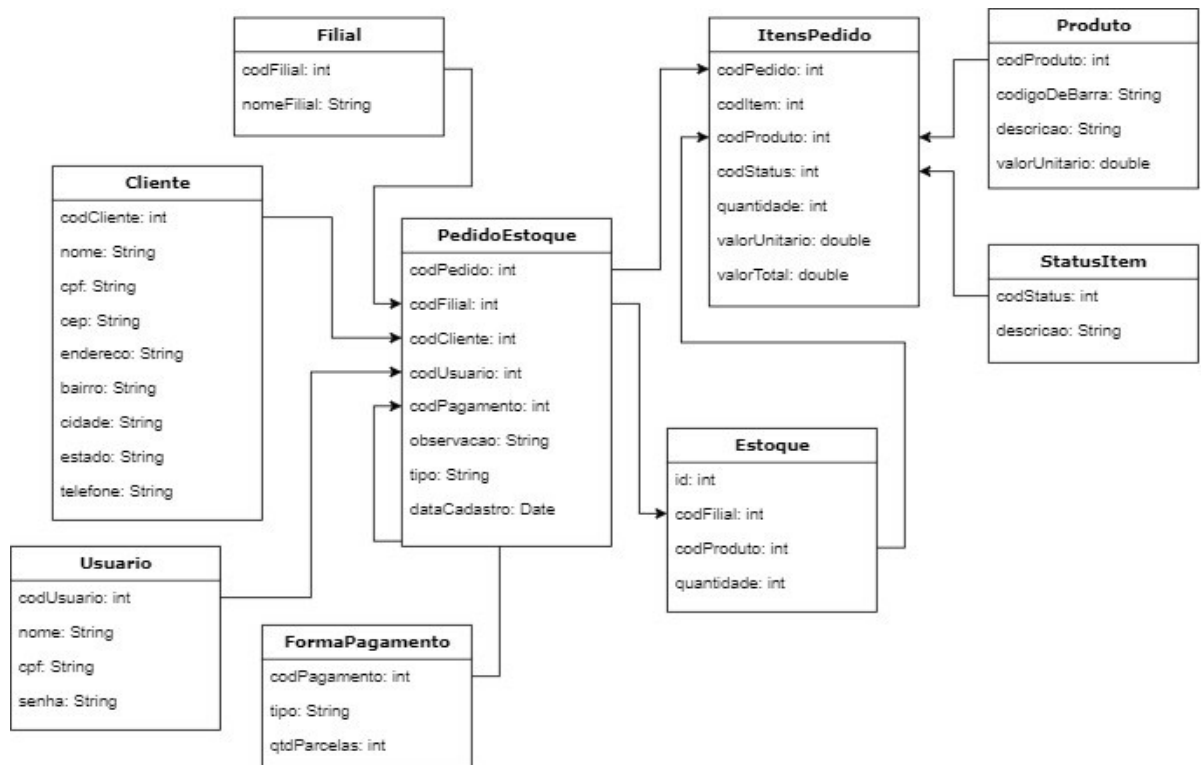
≡

1

Pedido Finalizado!

## 5. 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.





## 5.1. 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 o 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,

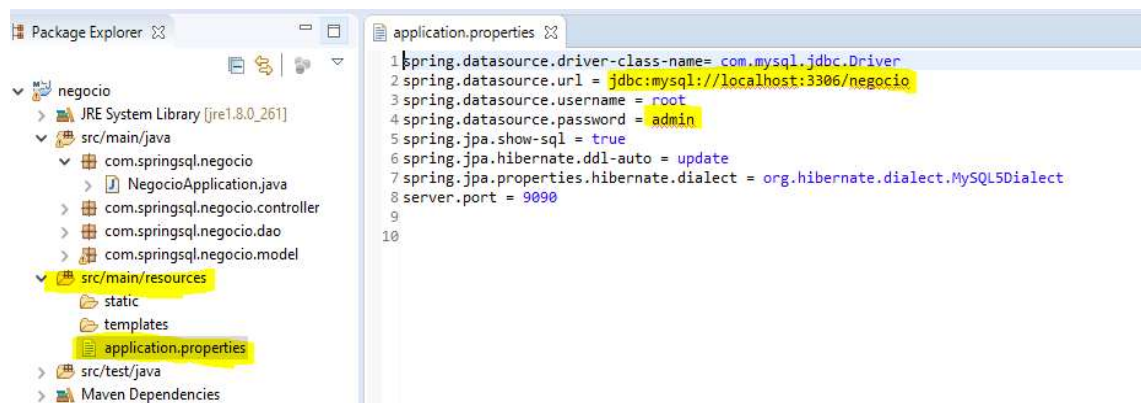
PRIMARY KEY (`id`)

) ENGINE=MyISAM DEFAULT CHARSET=latin1;

```

## 5.2. Configuração no Eclipse

Verificar a porta do MySQL e a Senha.



### 5.3 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 itens 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 itens de pedido seja maior que 10

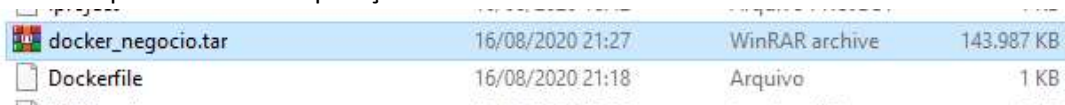
```
use negocio;
SELECT itens_pedido.cod_pedido, count(itens_pedido.cod_pedido), pedido_estoque.valor
```

```
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
```

## 6. 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 containerizada utilizando o Docker.



docker_negocio.tar	16/08/2020 21:27	WinRAR archive	143.987 KB
Dockerfile	16/08/2020 21:18	Arquivo	1 KB

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.