# Desafio Backend

O desafio consiste na implementação de uma API para o Sistema de Vendas Bagarote. Essa API deve ser desenvolvida em JAVA utilizando o Framework Spring Boot (Web, JPA, Security, OAuth2, Validation), Maven, Swagger para documentar a API e banco de dados PostgreSQL.

Todos os endpoints devem estar com suas respectivas rotas protegidas e exigindo autenticação. Para essa autenticação será utilizado o protocolo OAuth2 através de um access token JWT, que deve ser eviado no header da requisição AUTHORIZATION (Bearer Token).

A autenticação e Autorização do acesso já está implementada e se dá pelos seguintes dados:

- URL: https://auth.bagarote.com.br/auth/realms/bagarote-desafio/protocol/openid-connect/token

- client_id: desafio-front

- username: {{user}}

- password: {{pass}}

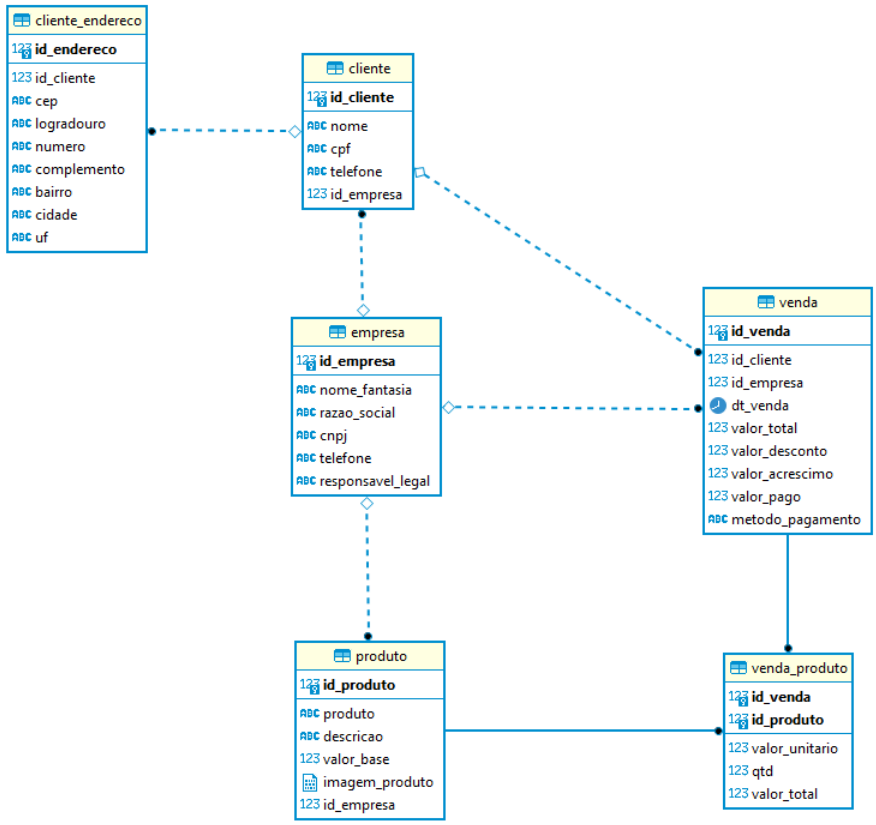
Existem 2 usuários disponíveis, são eles:

| username | password | authorities |
|---|---|---|
| administrador | administrador@2022 | ADMINISTRADOR |
| vendedor | vendedor@2022 | VENDEDOR |

A role ADMINISTRADOR dará acesso ilimitado a todos os endpoints da aplicação, já a role VENDEDOR permitirá o acesso leitura de todos os dados e apenas será permitido criação de uma nova venda.

Esse sistema suporta o cadastro de múltiplas empresas e todos os seus respectivos dados devem ser individualizados.

Diagrama do banco de dados:




*todas as tabelas devem implementar sequences para geração de IDs

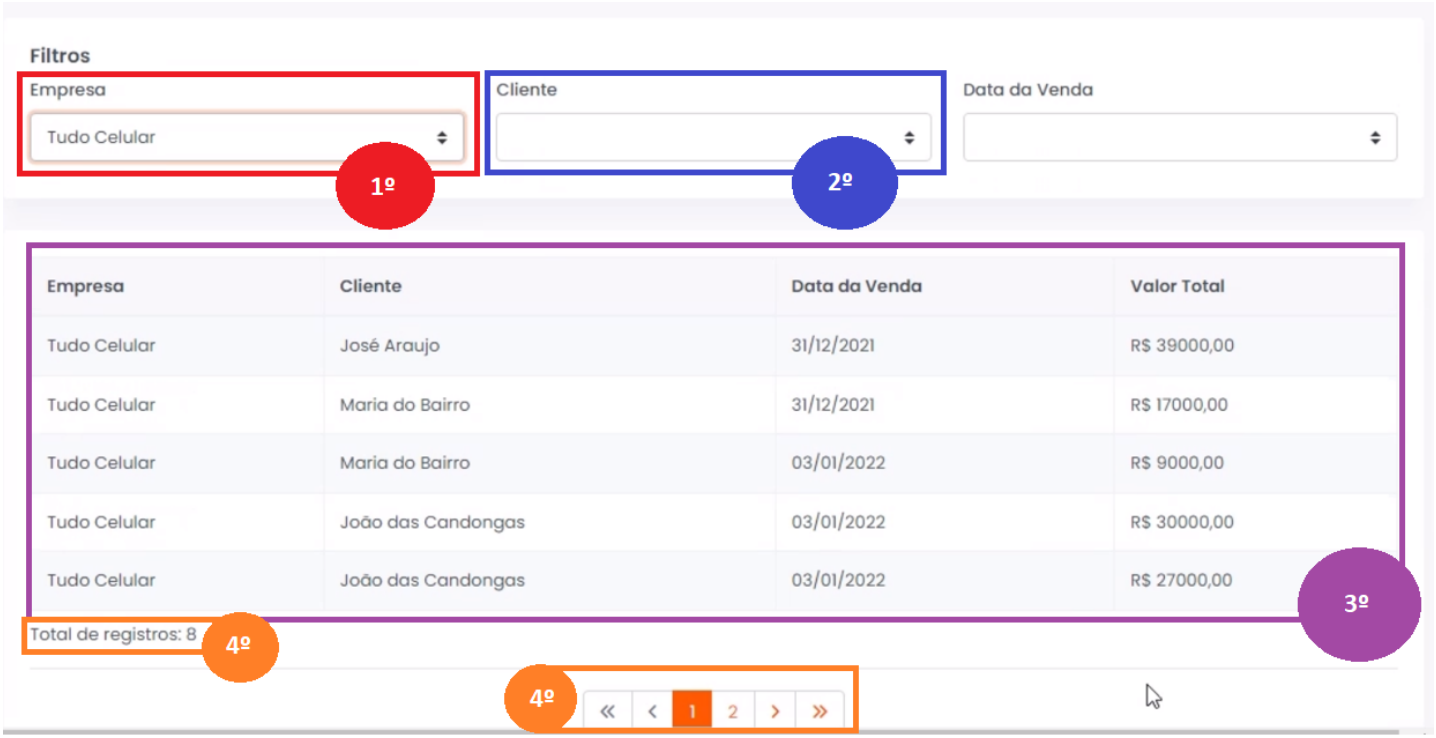
O Principal objetivo da API é gerenciar as Vendas das empresas cadastradas no banco, para isso devemos garantir que existam empresas previamente cadastradas e essas empresas possuam: clientes e produtos.

Toda a comunicação entre os clientes da API (front, mobile, etc) devem ser feitos via DTOs, ou seja, a API não deve expor o modelo de dados do domínio nas requisições http.

Todas as validações e respectivas exceptions devem ser expostas ao cliente da requisição, informando o código HTTP e a mensagem com o erro de negócio e/ou erro da api. Por exemplo: 404 (recurso não é encontrado), 401 (usuário não autenticado), 403 (acesso negado), etc.

O front já possui 3 telas projetadas para essa aplicação de forma que o desenvolvimento da API deve servir de apoio as funcionalidades dessas telas:

- **Tela 1 – Listagem de Vendas:**




Na tela acima o backend deverá fornecer endpoints para acesso aos os seguintes dados:

**1 º** - Dados de todas as Empresas cadastradas no sistema, a apresentação deve ser simplificada. Modelo do JSON:

```
[
{
    “idEmpresa”: 1,
    “nomeFantasia”: “Nome Fantasia da Empresa”
    }
]
```

**2 º** - Dados de todos os clientes da empresa filtrada no passo **1 º** , a apresentação deve ser simplificada. Modelo do JSON:

```
[
{
    “idCliente”: 1,
    “nome”: “Nome do Cliente”
    }
]
```

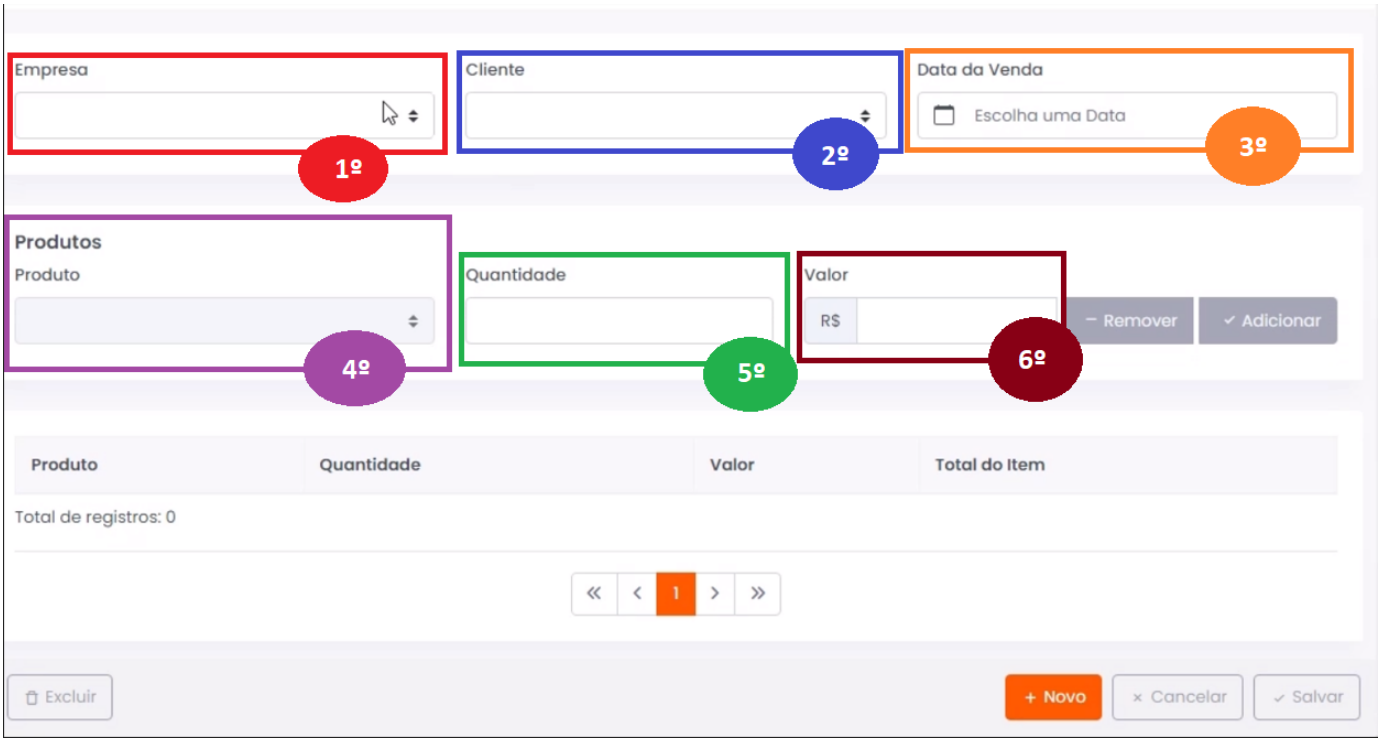
**3 º** - Listagem de todas as Vendas filtradas nos passos **1 º** e **2 º**, para exibir as vendas deve-se ao menos selecionar uma empresa, o resultado do filtro deverá ser paginado conforme a própria implementação do Spring da interface Page<T> (modelo do json da paginação no passo **4º**), a apresentação deve ser simplificada **(content)** . Modelo do JSON:

```
[
{
        “idVenda”: 1,
        “idEmpresa: 1,
        “nomeFantasia”: “Nome da Empresa”,
        “idCliente”: 1,
        “nomeCliente”: “Nome do Cliente",
        “dataVenda”: “yyyy-MM-dd HH:mm:ss”,
        “metodoPagamento”: [PENDENTE, DINHEIRO, PIX, CREDITO, DEBITO],
        “valorDesconto”: 0.00,
        “valorAcrescimo”: 0.00,
        “valorTotal”: 0.00,
                “valorPago”: 0.00
        }
]
```

**4º** - O resultado da listagem das vendas devem seguir a implementação da interface Page<T> do pacote ***org.springframework.data.domain***. Modelo do JSON:

```
{ "content": [],
  "pageable": {"sort": {"sorted": true,"unsorted": false,"empty": false },
     "pageSize": 5,  "pageNumber": 0,   "offset": 0, "paged": true, "unpaged": false
  },
  "totalPages": 1,  "totalElements": 3,  "last": true, "number": 0,"size": 5,
  "numberOfElements": 3, "sort": {"sorted": true,"unsorted": false, "empty": false  },
  "first": true,
  "empty": false}
```

- **Tela 2 – Nova Venda:**




**1 º** - Dados de todas as Empresas cadastradas no sistema, a apresentação deve ser simplificada. Modelo do JSON:

[

{

“idEmpresa”: 1,

“nomeFantasia”: “Nome Fantasia da Empresa”

}

]

**2 º** - Dados de todos os clientes da empresa filtrada no passo **1 º** , a apresentação deve ser simplificada. Modelo do JSON:

[

{

“idCliente”: 1,

“nome”: “Nome do Cliente”

}

]

**3 º** - Data da venda com pattern “yyyy-MM-dd HH:mm:ss”.

**4 º** - Dados de todos os produtos da empresa filtrada no passo **1 º** , a apresentação deve ser simplificada. Modelo do JSON:

```
[ { “idProduto”: 1,
“produto”: “Nome do Produto” } ]
```

**5 º** - Quantidade do produto selecionado para ser adicionado na venda. A quantidade deve ser maior que zero.

**6 º** - Ao selecionar o produto no passo 4 o backend deve retornar um produto by idProduto com os detalhes desse produto, incluindo o seu preço cadastrado. Modelo do JSON:

```
{ "idProduto": 1,
"produto": "nome do Produto",
"descricao": "descricao do produto",
"valorBase": 0.0,
"imagemProduto": "imagem do produto em formato BASE64",
"idEmpresa": 1 }
```

Ao submeter o formulário finalizando a venda o backend deve receber o seguinte JSON:

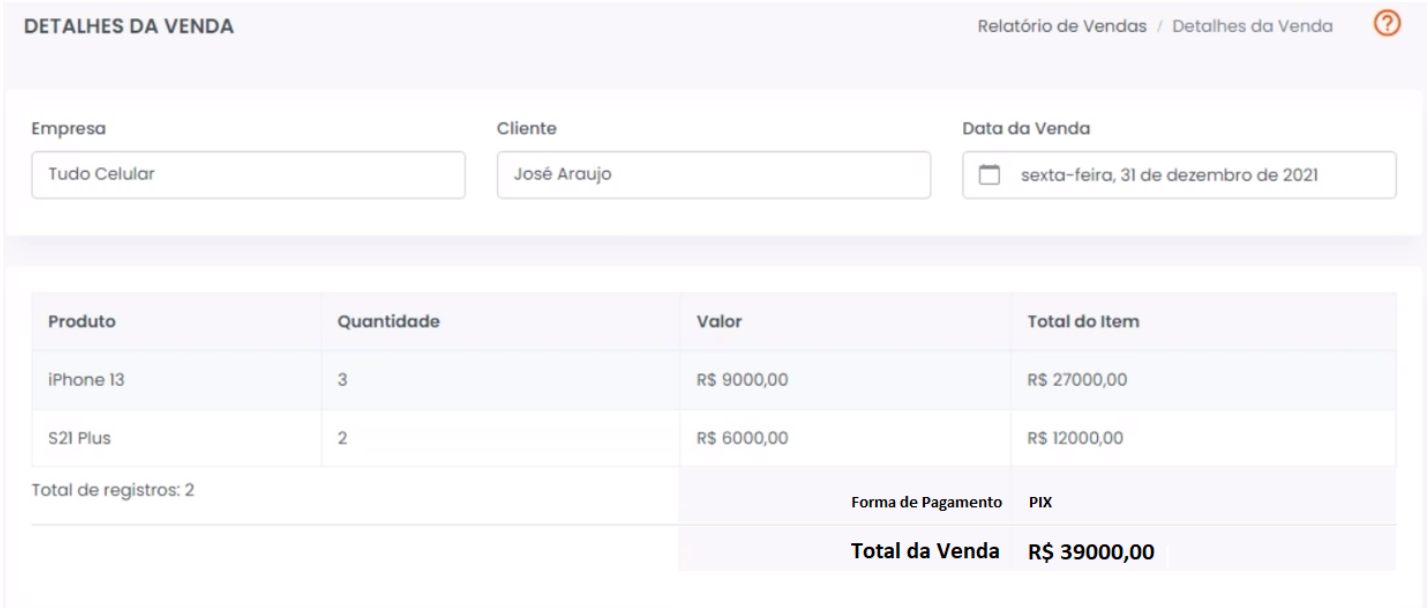
```
{ "idEmpresa": 1,
"idCliente": 1,
"dataVenda": "2022-01-06 13:19:52",
"metodoPagamento": "PENDENTE",
"produtos": [{ "idProduto": 1,
"quantidade": 10 }] }
```

**Regras:**

- Todos os campos acima devem ser explicitados como obrigatórios;
- O backend deve validar se o cliente informado na venda, assim como os produtos nela incluídos, pertencem realmente à empresa informada;
- Pattern dataVenda: “yyyy-MM-dd HH:mm:ss”;
- Deve existir ao menos 1 produto para que a venda seja realizada;
- Caso algum produto/cliente não esteja em conformidade com o banco de dados a venda não pode ser concluída;
- O método de pagamento deve aceitar apenas os valores: “**PENDENTE”, “DINHEIRO”, “CREDITO”, “DEBITO”, “PIX”**
- A quantidade de cada produto na venda deve ser maior que zero, caso algum produto não respeite essa regra a venda não pode ser concluída;
- Apesar do front da API realizar os cálculos de valorTotal de Produto e ValorTotal da Venda, os cálculos lá apresentados são apenas informativos, por tanto, o backend deverá realizar todos os cálculos da venda e de seus respectivos produtos;

- **Tela 3 – Detalhar Venda:**

Na Tela 1, onde são expostas todas as vendas de uma determinada empresa, será possível detalhar uma venda específica pelo idVenda:




O endpoint deverá receber 2 parâmetros que será o idEmpresa e idVenda e o backend deverá ser capaz de avaliar se a venda solicitada realmente pertence à empresa em questão. Modelo do JSON:


{ "idVenda": 0,
"idEmpresa": 0,
"nomeFantasia": "string",
"idCliente": 0,
"nomeCliente": "string",
"dataVenda": "2022-01-06 13:53:10",
"metodoPagamento": "PENDENTE",
"valorDesconto": 0,
"valorAcrescimo": 0,
"valorTotal": 0,
"valorPago": 0,
"produtos": [ { "idVenda": 0, "idProduto": 0, "produto": "string", "valorUnitario": 0, "quantidade": 0, "valorTotal": 0
} ]}

# Dados do Desafio:

A API já possui parte de seu código desenvolvido. Sem garantias de qualidade e/ou funcionamento.

Código Fonte da API: https://github.com/marlon-bagarote/desafio-pdv.git

## Desenvolvimento Obrigatório:

- Desenvolvimento de todos os endpoints do fluxo de Venda (Tela 1, 2 e 3);

- Todos os endpoints devem possuir validação de request de dados, como: obrigatoriedade de campos, tamanhos de strings, validação de CNPJ/CPF quando necessário e tipos de dados e todos os cálculos que envolvem a venda, toda validação de regra de negócio ou acesso indevido deve ser comunicado ao cliente da requisição;

- Todas as rotas dos enpoints devem estar protegidas;

- A camada Controller/resource da API não deve possuir acesso a camada de dados, os dados devem ser acessados por uma camada de serviço e assim serem expostos aos controllers;

- Como já mencionados os enpoints não devem expor o modelo de dados para os requests e/ou responses, para isso devem ser implementados DTOs que auxiliarão nesta comunicação;

- Os endpoints do fluxo de Vendas devem estar documentados e expostos utilizando Swagger e as principais regras/validações obrigatórias dos modelos de dados devem estar também explícitos no documento e todos os recursos devem estar acessíveis por ele;

- O scrpit de geração do banco de dados está disponível na pasta “resources” do projeto.

## Desenvolvimento Bônus:

- O desenvolvimento dos demais endpoints para os modelos de dados de Empresa, Cliente e Produto, lembrando que todo cadastro se inicia pela Empresa e também observar as authorities;

- Toda a comunicação desses endpoints devem seguir o mesmo padrão de desenvolvimento estabelecido no enpoint de vendas;

- Tornar a evolução do bancos versionável através de um database-migration;

- Considerando que a API pode se tornar uma Plataforma de Ecommerce contemplando múltiplas empresas no modelo de marktplace, descreva os principais gargalos que a modelagem atual possui e as possíveis estratégias de evolução para garantir a disponibilidade e escalabilidade da API.

***Regras:**

Todas as restrições dos cadastros dos recursos bônus devem implementar as regras/limitações mínimas expostas nas tabelas do banco de dados.

****Qualquer dúvida por favor nos comunique pelo e-mail: marlon@karyon.com.br**