MANUAL DE ESCLARECIMENTO

Desafio BR Consultoria: API – CRUD Básico

Autor: Jefferson Queiroz da Costa

Telefone Celular: (87) 99951-1215

E-mail: jeffersonadven7@gmail.com

Git-Hub: https://github.com/Jackadven/DESAFIO-BRconsultoria.git

Sumário

Afogados da Ingazeira – PE

23 de novembro de 2022

PROJETO: Desenvolvimento de uma API capaz de realizar CRUD de vendas com uso e manipulação de tabelas (modalidade, loja, vendas) criadas no Postgress com NODEjs e Express. Utilizei do framework “Insomnia” para realizar o envio das requisições e assim realizar o teste de API Client.

INFRAESTRUTURA: A infraestrutura do projeto está dividida em quatro camadas, das quais possuem os seguintes arquivos.

1. Camada Base:
   - .env (arquivo com variáveis de ambiente para conexão com o DB);
   - packge.json (Dependências necessárias);
   - server.js (Defini a escuta do servidor na porta indicada no arquivo .env);
   - app.js (Aplicação do Express e possibilita trabalhar com request.body de requisições .json);
   - router.js (Definições das rotas e chamada das funções de validação, manipulação e recebimento de requests);

2. Middlewares: Nessa camada optei por fazer o tratamento de algumas exceptions através de middlewares. Essa camada é responsável por validar os dados necessários para o funcionamento da API:
   - lojaMiddlewares.js (funções para rotas da tabela loja);
   - vendasMiddlewares.js (funções para rotas da tabela vendas);

3. Controllers: Camada de funções assíncronas que recebem as requests e fazem a chamada das funções da camada models:
   - lojaController.js (funções de request assíncronas para tabela loja);
   - vendasController.js (funções de request assíncronas para tabela vendas);

4. Models: Camada responsável por fazer a conexão com o DB, e também possui as funções responsáveis por realizar as querys no DB:
   - conection.js (Uso da biblioteca pg e variáveis de ambiente para acessar o DB);
   - lojaModel.js (Funções de query para tabela loja);
   - vendasModel.js (Funções de query para tabela vendas);

BANCO DE DADOS: Foi utilizado o DB Postgres com as seguintes tabelas:

1. MODALIDA: Duas modalidades:

Data Output | Messages | Notifications

	modalidade [PK] integer	nome character varying (255)
1	1	DÉBITO
2	2	CRÉDITO

2. LOJA: id do tipo SERIAL (Autoincremento):

Data Output | Messages | Notifications

	id [PK] integer	nome_loja character varying (255)
1	2	3M
2	3	USP
3	12	CELPE
4	21	CASAS BAHIA
5	22	CBN
6	24	JQ ENGEHALL
7	27	BRconsultoria

3. VENDAS: id do tipo SERIAL (Autoincremento):

Data Output | Messages | Notifications

	id [PK] integer	n_cartao integer	valor_bruto numeric (10,2)	valor_liquido numeric (10,2)	data timestamp without time zone	parcelas integer	modalidade integer	bandeira character varying (255)	loja_id integer
1	5	1111	100.00	100.00	2022-11-16 01:19:37.047361	2	2	ELO	21
2	6	1111	100.00	100.00	2022-11-16 01:20:10.242202	2	2	ELO	21
3	7	2222	200.00	200.00	2022-11-16 01:21:40.4744	4	2	ELO	2
4	10	3333	300.00	300.00	2022-11-16 01:46:13.218257	6	2	ELO	2
5	13	5555	500.00	500.00	2022-11-17 18:57:49.650037	5	2	VISA	2
6	16	7777	500.00	500.00	2022-11-22 17:55:49.142744	6	2	VISA	2
7	20	1222	50000.00	50000.00	2022-11-22 18:19:36.284736	6	2	VISA	24
8	22	4222	50000.00	50000.00	2022-11-22 18:24:56.298332	6	2	VISA	24

VALIDAÇÃO DE DADOS: Foram utilizadas middlewares para verificar e garantir a entrada dos dados necessários para realização das operações, evitando quebrar o nosso backend. Segue o modelo de uma das middlewares utilizadas para os dados

da loja:

lojaMiddlewares.js

backend > src > middlewares > lojaMiddlewares.js > validateQueryLoja

```
const validateBody = (request,response,next) => {
    const {body} = request;
    if(body.nome_loja == undefined){
        return response.status(400).json({menssage: 'The field "nome_loja" is required'});
    }
    if(body.nome_loja == ''){
        return response.status(400).json({menssage: '"nome_loja" cannot be empty'});
    }
    next();
};

```

REQUEST (BODY, PARAMS AND QUERY): As funções de requisição obtiveram os dados via corpo da requisição (request.body), via parâmetros da URL (request.parms) e via query da própria requisição (request.query). A seguir temos os locais de aplicação:

1. request.body: Criação e atualização das tabelas:

lojaController.js

backend > src > controllers > lojaController.js > updateLoja

```
//importando as funções do models
const lojaModel = require('../models/lojaModel');


//MODELS FUNCTIONS OF TABLE LOJA//
//-----------------------------------------------------//
//CREATE//
const newLoja = async (request,response) =>{
    const newLoja = await lojaModel.newLoja(request.body);
    return response.status(201).json(newLoja);
};

```

2. request.parms: Deletar e consultar tabelas por meio de ID informado:

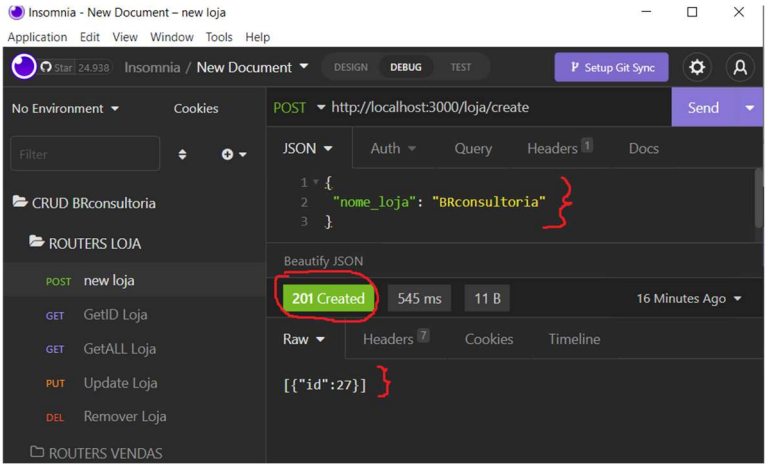
```

//GET FOR ID
const getLoja = async (request,response) =>{
    const {id} = request.params;
    const loja = await lojaModel.getLoja(id);
    return response.status(200).json(loja);
};

```

3. request.query: Consultas paginadas com query (take, skip):

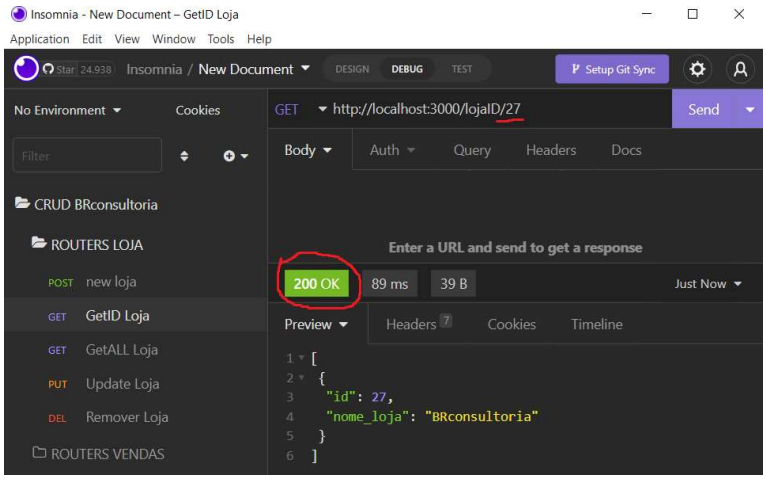
```
19
20  //GET ALL//
21  const getAll = async (request,response) =>{
22      const {take,skip} = request.query;
23      const loja = await lojaModel.getAll(take,skip);
24      return response.status(200).json(loja);
25  };
26
```

CONSULTAS E RESULTADOS: As consultas foram realizadas nas tabelas loja e vendas por meio do framework “Insomnia”, das quais foram testadas e obtiveram os seguintes resultados para cada tipo de solicitação:

1. LOJA:
    a. POST New Loja:
        . Envio de nome por meio do corpo da requisição;
        . Resultado: Código validando a operação e retorno do id da nova loja;



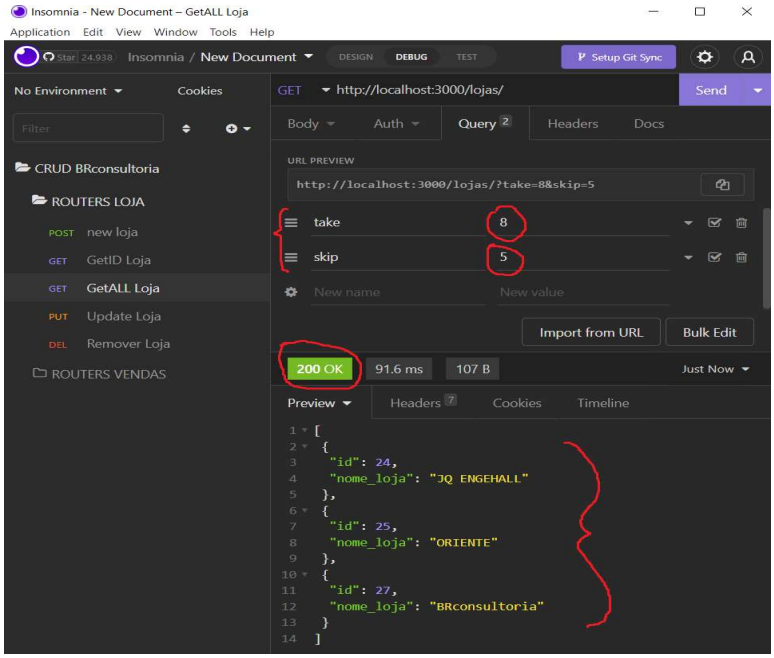
    b. GET GetID Loja:
        . Envio de parâmetro id via URL;
        . Resultado: Código validando a operação e retorno do objeto loja do id correspondente;



c. GET GetAll Loja:

. Envio da query (take,skip) para paginação;

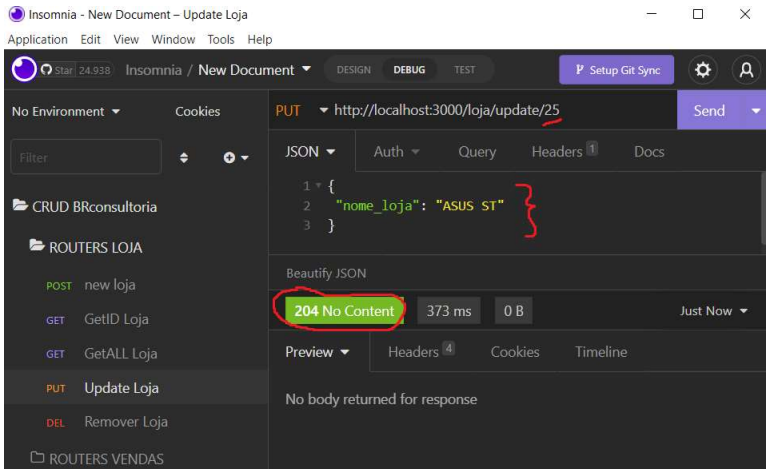
. Resultado: Código validando a operação e retorno dos objetos loja da paginação correspondente;



d. PUT Update Loja:

. Envio de nome por meio do corpo da requisição e parâmetro id via URL;

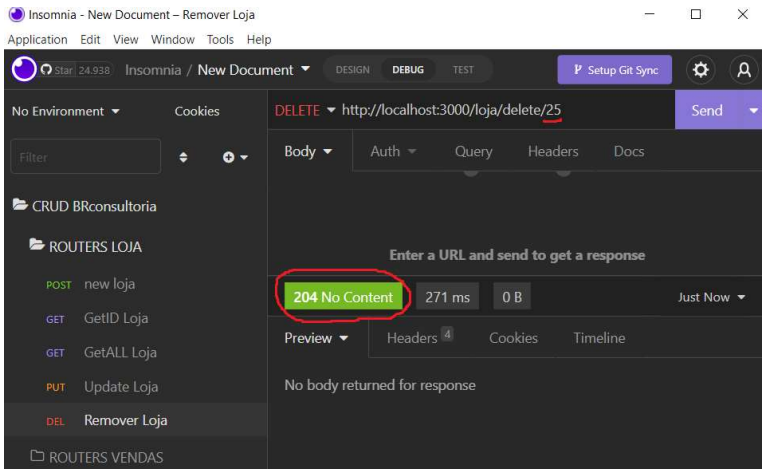
. Resultado: Código validando a operação;



e. DEL Remover Loja:

. Envio de parâmetro id via URL;

. Resultado: Código validando a operação;

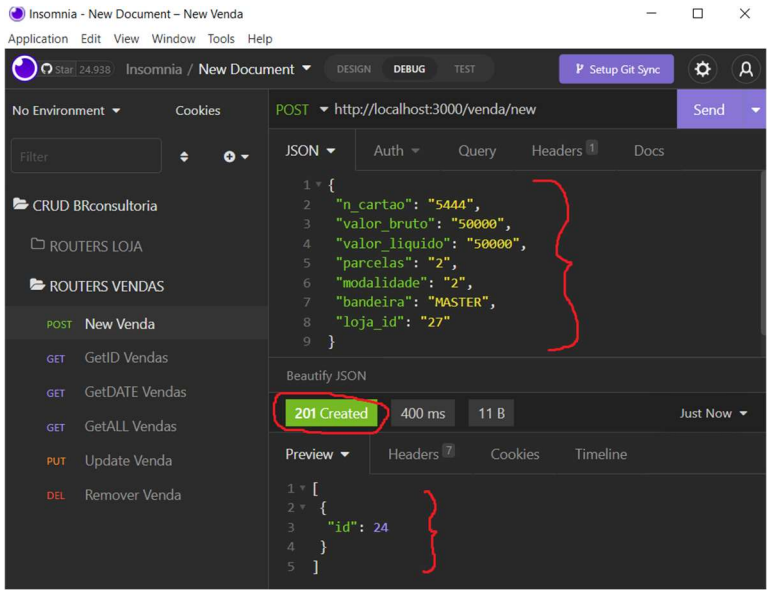


2. VENDAS:

a. POST New Venda:

. Envio de numero de cartão, valor bruto, valor líquido, parcelas, modalidade e bandeira da venda por meio do corpo da requisição;

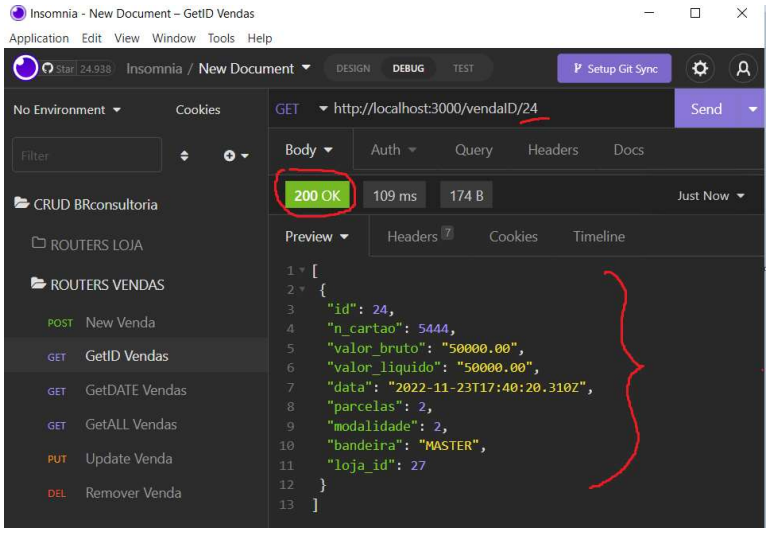
. Resultado: Código validando a operação e retorno do id da nova venda;



b. GET GetID Venda:

. Envio de parâmetro id da venda via URL;

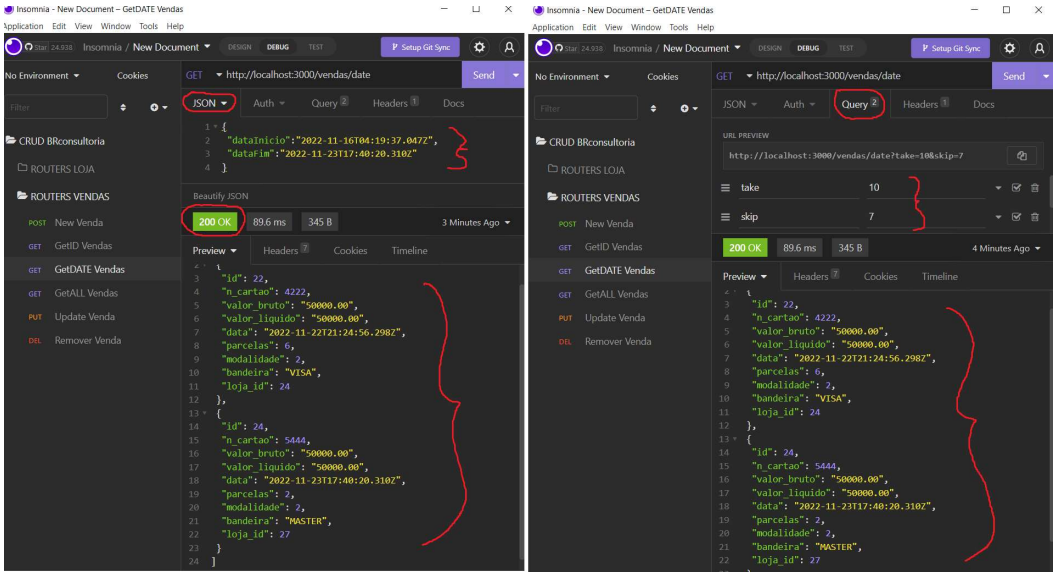
. Resultado: Código validando a operação e retorno do objeto venda para o id correspondente;



c. GET GetDATE Vendas:

. Envio da dataInicio e dataFim por meio do corpo da requisição, e query (take,skip) para paginação;

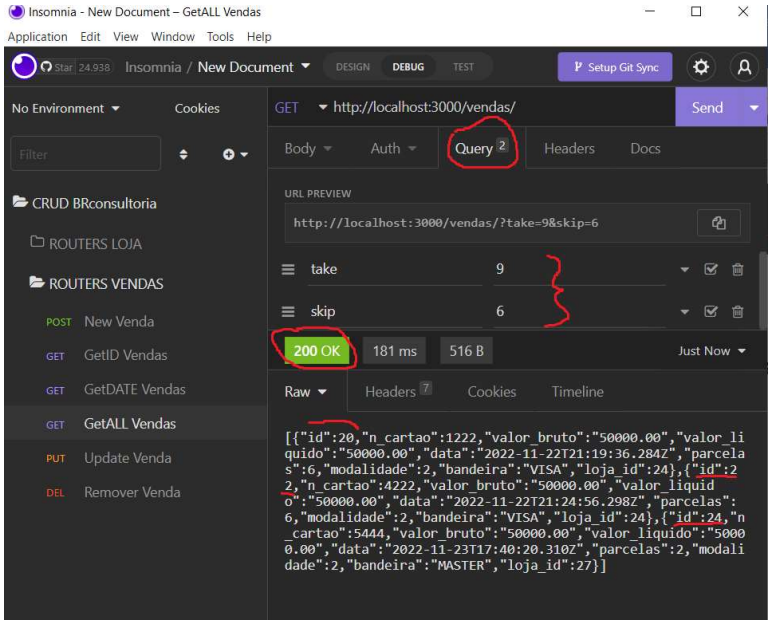
. Resultado: Código validando a operação e retorno dos objetos vendas da paginação correspondente;



d. GET GetALL Vendas:

. Envio da query (take,skip) para paginação;

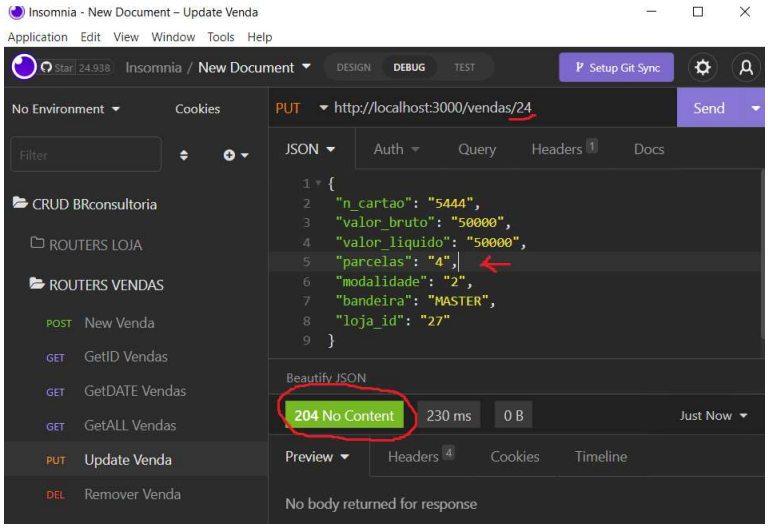
. Resultado: Código validando a operação e retorno dos objetos vendas da paginação correspondente;



e. PUT Update Venda:

. Envio de número do cartão, valor bruto, valor líquido, parcelas, modalidade e bandeira da venda por meio do corpo da requisição, e envio de parâmetro id via URL:

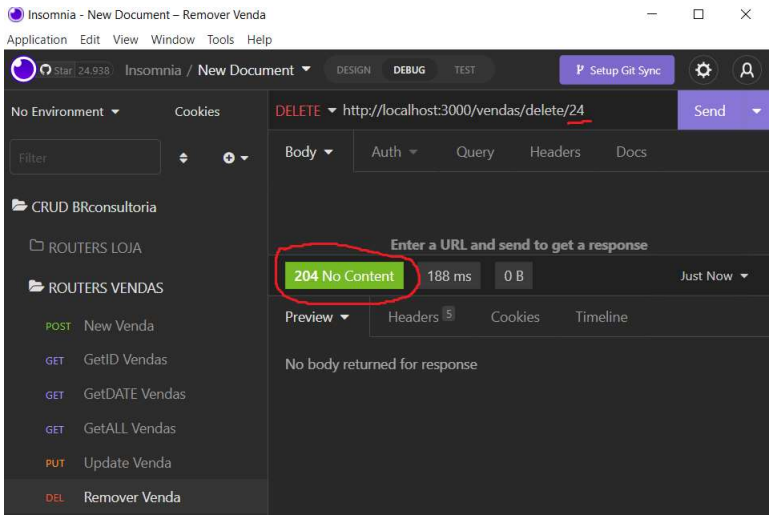
. Resultado: Código validando a operação;



f. DEL remover Venda:

. Envio de parâmetro id via URL;

. Resultado: Código validando a operação;



CONCLUSÃO: A API foi desenvolvida com sucesso, atendendo os requisitos necessário do desafio além de apresentar o bônus da opção de paginação bem como as requisições GET conforme solicitadas, tendo também a opção de ampliação das mesmas caso necessário.