1 - Apresentação

Projeto:
A001

Título:
CRUD Comercial

Autor:
Diego dos Santos Oliveira

Início:
02/05/2024

Conceito:
Desenvolvimento de um sistema web para gestão de clientes, adaptável à utilização de comerciantes e prestadores de serviço das mais diversas áreas de atuação.

O objetivo deste projeto, é que ao final tenhamos desenvolvido um cadastro completo de clientes (criar, ler, atualizar, apagar), e a partir da ficha cadastral de cada cliente poder incluir uma venda e a emissão de um contrato de serviço, auto preenchido com os dados do cliente e da venda.

Existe a possibilidade de ampliação do sistema, mas manteremos o foco nestas funcionalidades, podendo haver no futuro novas etapas de projeto para ampliar este sistema.

Tecnologias Aplicadas:
Banco de Dados: MySQL
Backend: Java, Spring Framework
Frontend: Html, CSS, Javascript

2 - Desenvolvimento Backend

2.1 - Definição dos objetos:

Seguindo a concepção inicial, serão criados 3 objetos, a partir dos quais trabalharemos nosso banco de dados, backend e formulários e exibição de dados no frontend.

O primeiro objeto será “Cliente”. Cada cliente terá um cadastro único, a partir do qual poderá efetuar várias compras ou, pensando na nossa perspectiva, o cliente poderá participar de várias vendas.

O segundo objeto será “Venda”. Cada venda será o registro de uma única interação comercial com um “Cliente”, onde ficará registrado tudo que for acordado entre as partes.

O terceiro objeto será o “Contrato”, onde ficará registrado a efetivação da venda, incluindo alguns dados adicionais que só interessam a uma negociação finalizada.

2.1.1 - Cliente

Serão os seguintes dados necessários para o registro de um cliente:

1. idCliente;
2. nome;
3. cpf;
4. nacionalidade;
5. cep;
6. rua;
7. numero;
8. complemento;
9. bairro;
10. cidade;
11. estado;
12. telefone;
13. email.

2.1.2 - Venda

A partir de um cliente cadastrado, poderá registrar uma venda com os seguintes dados:

1. idVenda;
2. cliente;
3. serviço;
4. descrição;
5. valor.

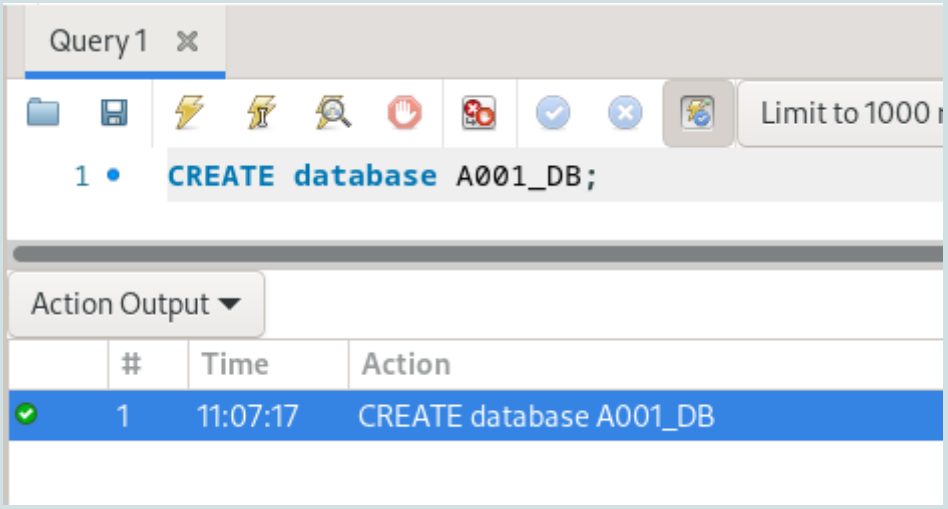
2.1.3 - Contrato

A partir do cadastro de uma venda poderá emitir um contrato com os seguintes dados:

1. idContrato;
2. venda;
3. dataInicio;
4. dataFinal;
5. formaPagamento.

2.2 - Banco de Dados

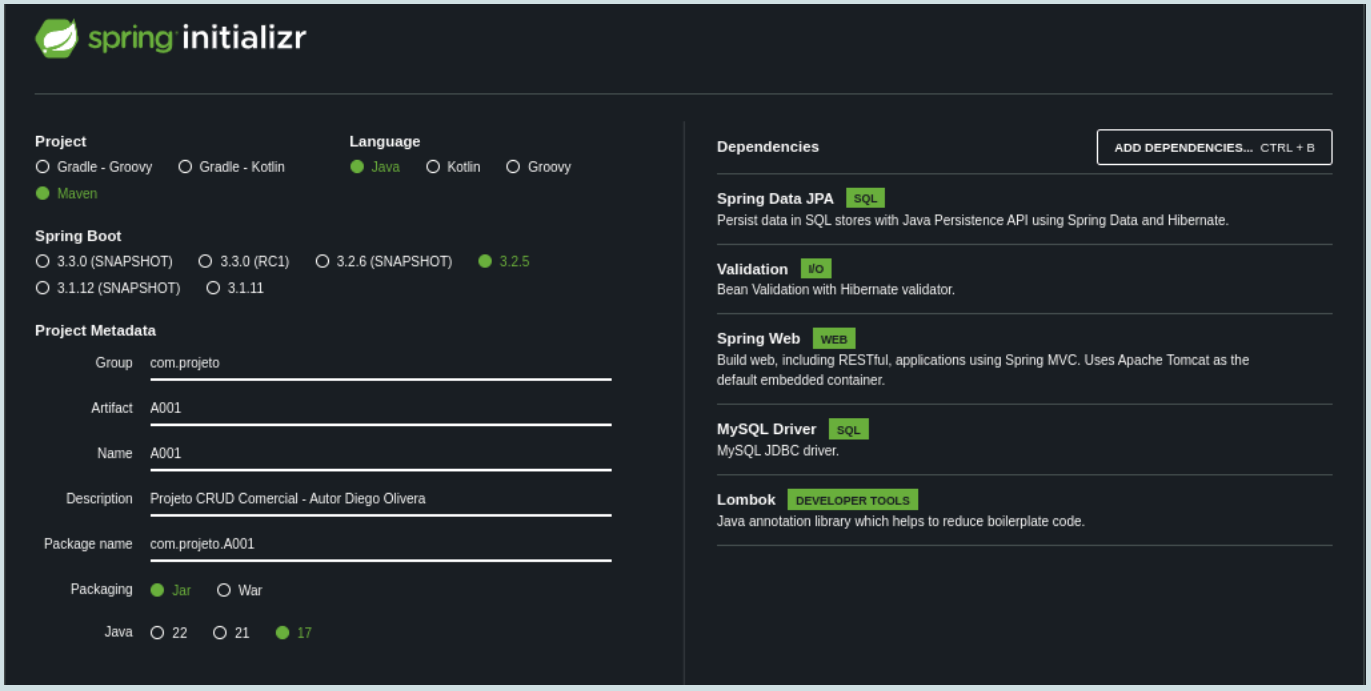
Utilizando o MySQL Workbench, foi criado o banco de dados “A001_DB” em Localhost.



As tabelas serão criadas pela dependência Spring Data, seguindo as anotações feitas nas classes do sistema.

2.3 - API

2.3.1 - Configuração



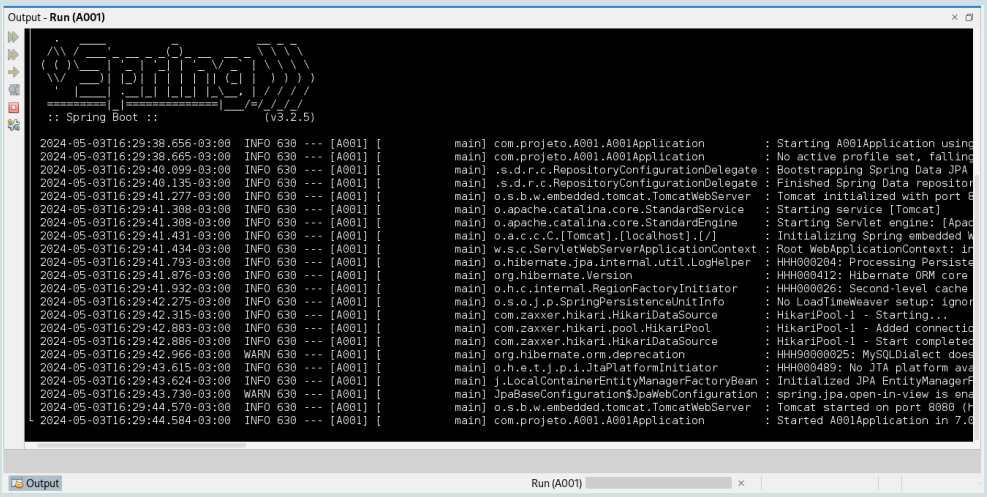
Inicialmente, utilizei a ferramenta “Spring Initializr” para configurar uma pasta de projeto com as dependências necessárias.

Abrindo o projeto no NetBeans, o primeiro passo será inserir as informações para a conexão com o banco de dados criado anteriormente.

Build:

```
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  16.614 s
Finished at: 2024-05-03T16:25:28-03:00
------------------------------------------------------------------------
```

Run:



Verificando que não há erros na configuração inicial, vamos adiante.

2.3.2 - Classes Model

Cliente

```java
package com.projeto.A001.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "Cliente")
public class Cliente {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer idCliente;
    private String nome;
    private String cpf;
    private String nacionalidade;
    private String cep;
    private String rua;
    private String numero;
    private String complemento;
    private String bairro;
    private String cidade;
    private String estado;
    private String telefone;
    private String email;
}
```

Venda

```java
package com.projeto.A001.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Table;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "Venda")
public class Venda {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer idVenda;
    @ManyToOne
    @JoinColumn(name="id_Cliente")
    private Cliente cliente;
    private String serviço;
    private String descrição;
    private double valor;
}
```

Contrato

```java
package com.projeto.A001.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.OneToOne;
import jakarta.persistence.Table;
import java.util.Date;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "Contrato")
public class Contrato {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer idContrato;
    @OneToOne
    @JoinColumn(name="id_Venda")
    private Venda venda;
    private Date dataInicio;
    private Date dataFinal;
    private String formaPagamento;
}
```

Após a criação das classes e executar novamente a Build do projeto, verifica-se a criação das tabelas no banco de dados, correspondentes para cada classe.

2.3.3 - Repository Interface

ClienteRepository

```java
package com.projeto.A001.repository;

import com.projeto.A001.model.Cliente;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ClienteRepository extends JpaRepository<Cliente, Integer>{

}
```

VendaRepository

```java
package com.projeto.A001.repository;

import com.projeto.A001.model.Venda;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface VendaRepository extends JpaRepository<Venda, Integer>{

}
```

ContratoRepository

```java
package com.projeto.A001.repository;

import com.projeto.A001.model.Contrato;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ContratoRepository extends JpaRepository<Contrato, Integer>{

}
```

*Poderá haver alterações ao longo do desenvolvimento.

2.3.4 - Classes Service

ClienteService

```java
@Service
public class ClienteService {
    @Autowired
    ClienteRepository clienteRepository;

    public Cliente criar(Cliente c){...5 lines }

    public List<Cliente> listarTodos(){...3 lines }

    public Cliente buscaPorId(Integer id){...3 lines }

    public void excluir(Integer id){...4 lines }

    public Cliente atualizar(Integer id, Cliente c){...20 lines }
}
```

VendaService

```java
@Service
public class VendaService {

    @Autowired
    VendaRepository vendaRepository;

    public Venda criar(Venda v){...5 lines }

    public List<Venda> listarVendas() {...3 lines }

    public List<Venda> listarVendasPorCliente(Integer idCliente) {...3 lines }

    public Venda buscaPorId(Integer id) {...3 lines }

    public void excluir(Integer id) {...4 lines }

    public Venda atualizar(Integer id, Venda v) {...11 lines }
}
```

ContratoService

```java
@Service
public class ContratoService {

    @Autowired
    ContratoRepository contratoRepository;

    public Contrato criar(Contrato c) {...5 lines }

    public List<Contrato> listarContratos() {...3 lines }

    public Contrato buscaPorId(Integer id) {...3 lines }

    public void excluir(Integer id) {...4 lines }

    public Contrato atualizar(Integer id, Contrato c) {...11 lines }
}
```

2.3.5 - Classes Controller

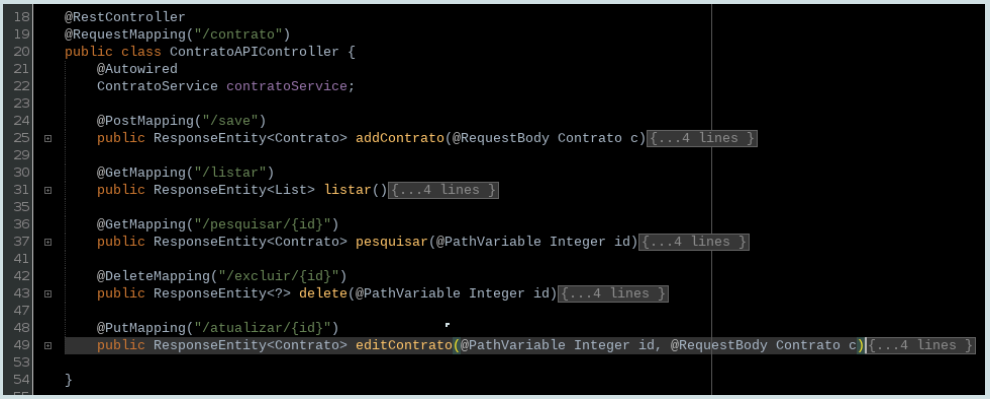
ClienteAPIController

```
@RestController
@RequestMapping("/cliente")
public class ClienteAPIController {
    @Autowired
    ClienteService clienteService;

    @PostMapping("/save")
    public ResponseEntity<Cliente> addCliente(@RequestBody Cliente c){...4 lines }

    @GetMapping("/listar")
    public ResponseEntity<List> listar(){...4 lines }

    @DeleteMapping("/excluir/{id}")
    public ResponseEntity<?> delete(@PathVariable Integer id){...4 lines }

    @GetMapping("/pesquisar/{id}")
    public ResponseEntity<Cliente> pesquisar(@PathVariable Integer id){...4 lines }

    @PutMapping("/atualizar/{id}")
    public ResponseEntity<Cliente> editCliente(@PathVariable Integer id, @RequestBody Cliente c){...4 lines }
}
```

VendaAPIController

```
@RestController
@RequestMapping("/venda")
public class VendaAPIController {
    @Autowired
    VendaService vendaService;

    @PostMapping("/save")
    public ResponseEntity<Venda> addVenda(@RequestBody Venda v){...4 lines }

    @GetMapping("/listar")
    public ResponseEntity<List> listar(){...4 lines }

    @GetMapping("/vendasporcliente/{idCliente}")
    public ResponseEntity<List> vendasPorCliente(@PathVariable Integer idCliente){...4 lines }

    @DeleteMapping("/excluir/{id}")
    public ResponseEntity<?> delete(@PathVariable Integer id){...4 lines }

    @GetMapping("pesquisar/{id}")
    public ResponseEntity<Venda> pesquisar(@PathVariable Integer id){...4 lines }

    @PutMapping("/atualizar/{id}")
    public ResponseEntity<Venda> editVenda(@PathVariable Integer id, @RequestBody Venda v){...4 lines }
}
```

ContratoAPIController

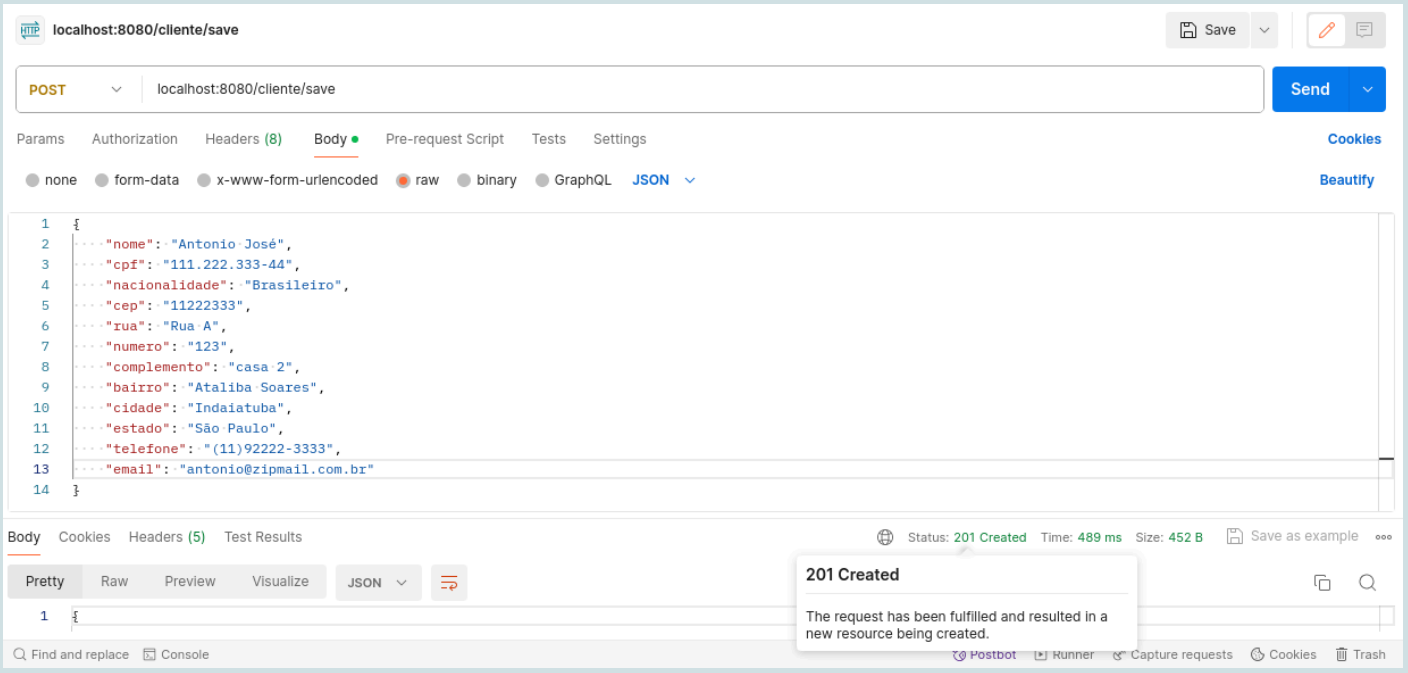


```
@RestController
@RequestMapping("/contrato")
public class ContratoAPIController {
    @Autowired
    ContratoService contratoService;

    @PostMapping("/save")
    public ResponseEntity<Contrato> addContrato(@RequestBody Contrato c){...4 lines }

    @GetMapping("/listar")
    public ResponseEntity<List> listar(){...4 lines }

    @GetMapping("/pesquisar/{id}")
    public ResponseEntity<Contrato> pesquisar(@PathVariable Integer id){...4 lines }

    @DeleteMapping("/excluir/{id}")
    public ResponseEntity<?> delete(@PathVariable Integer id){...4 lines }

    @PutMapping("/atualizar/{id}")
    public ResponseEntity<Contrato> editContrato(@PathVariable Integer id, @RequestBody Contrato c){...4 lines }

}
```

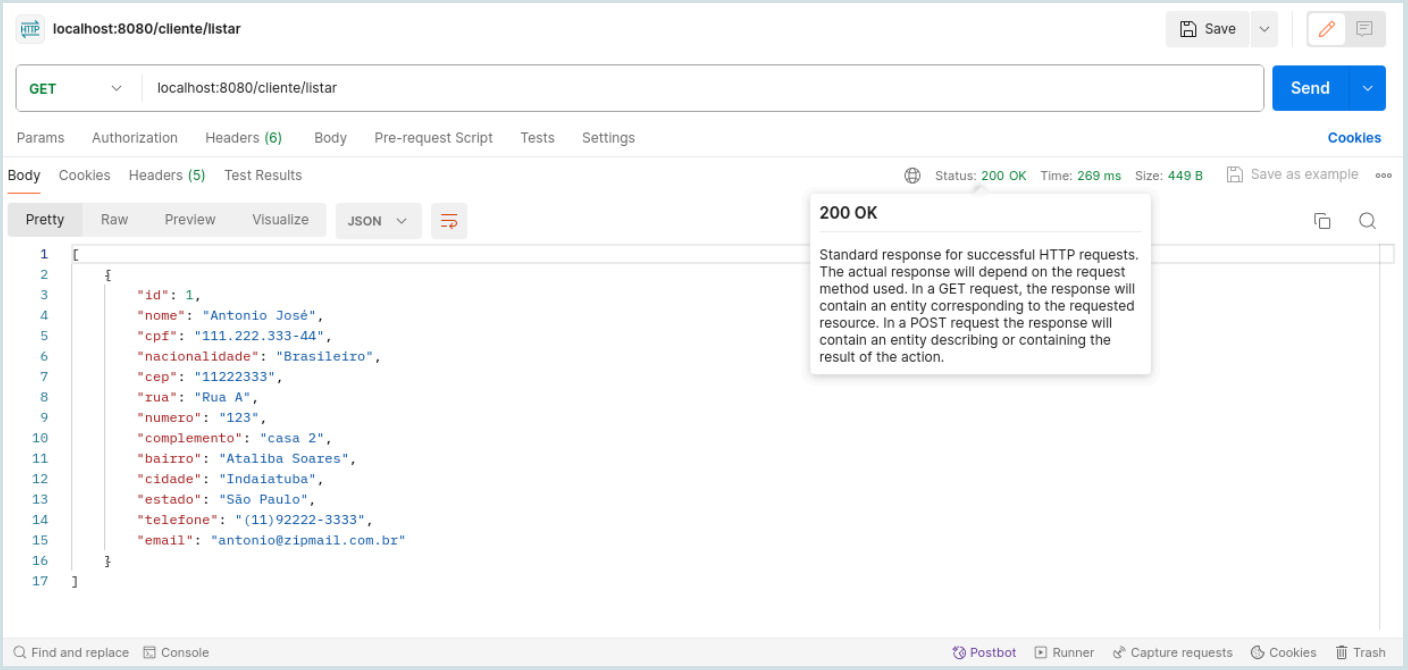
2.3.5 - Testando as requisições Http da API

- **Cliente**

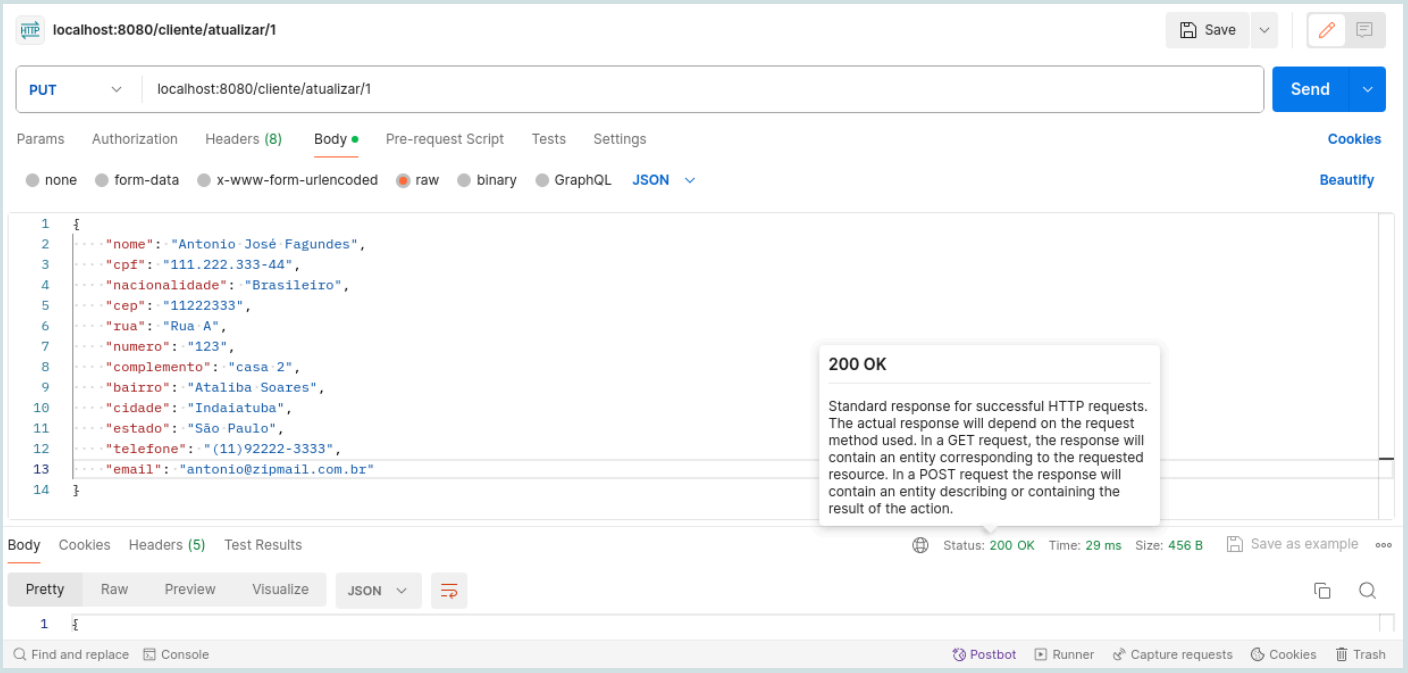
Create



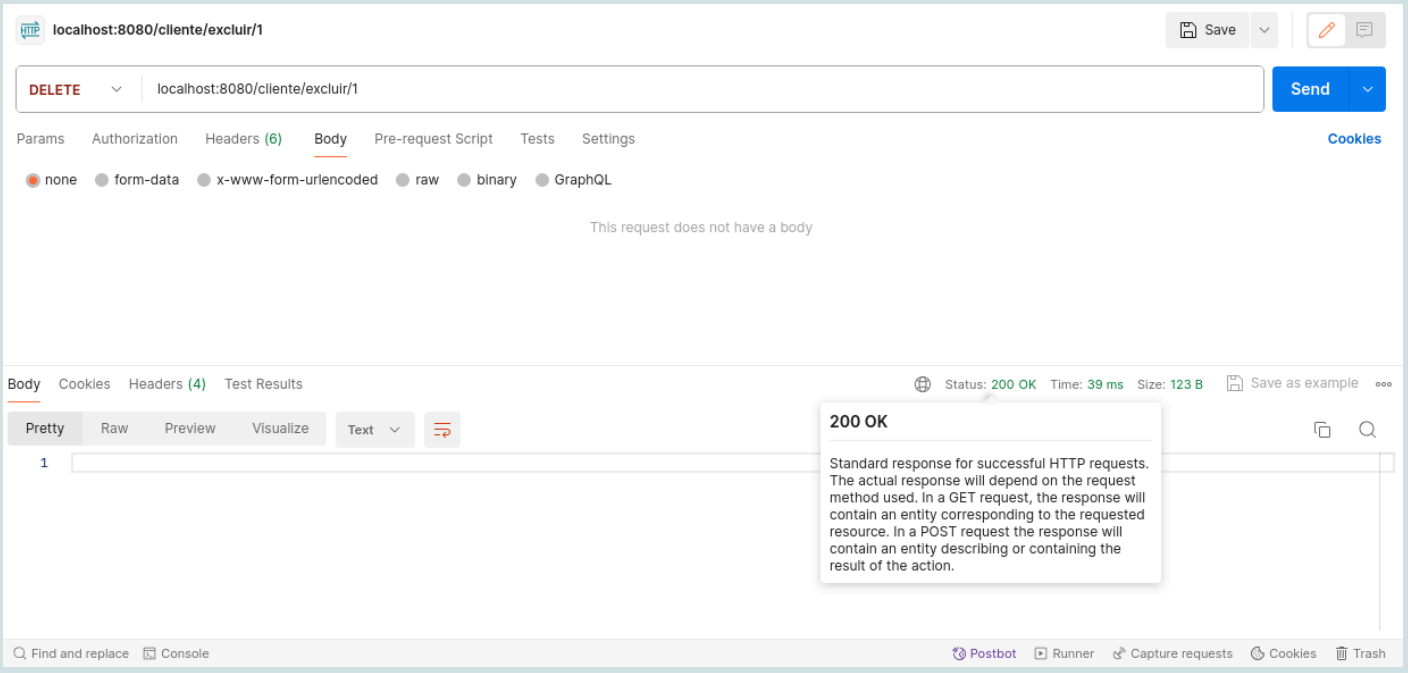
Read



Update



Delete

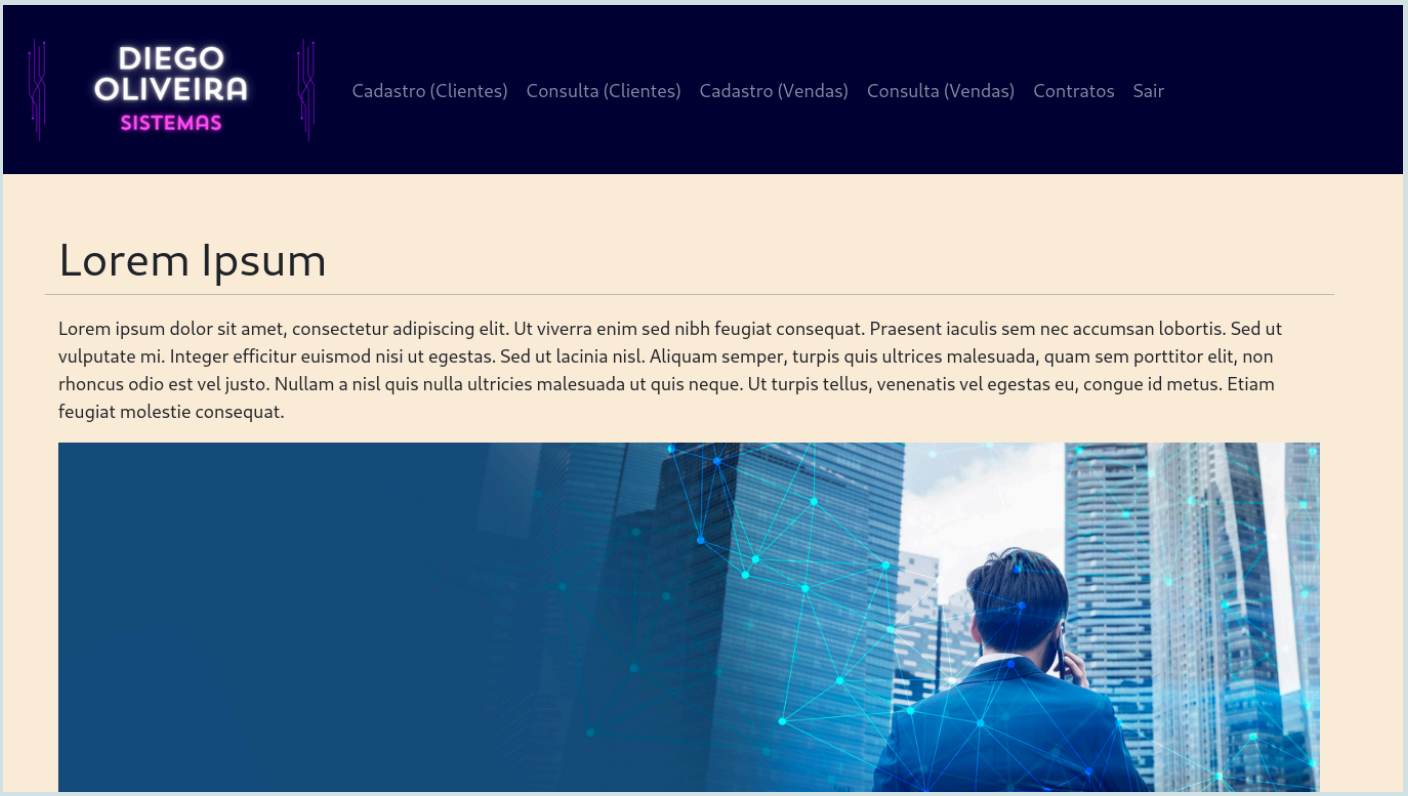


Também foi feito o teste de funcionamento das API's de Venda e Contrato, alcançando a resposta esperada em cada uma das requisições.

3 - Desenvolvimento Frontend

3.1 - Conceito Visual

Pensando em um design intuitivo e de fácil compreensão para um público o mais abrangente possível, cheguei ao seguinte design:



Logotipo da empresa e menu sobre um fundo escuro.
Área de trabalho em fundo claro.