

## 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.

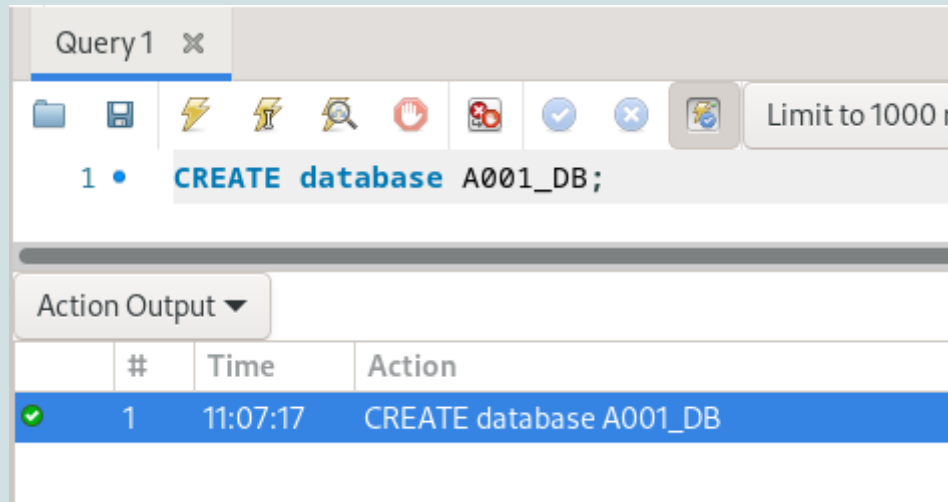
### **2.1.3 - Contrato**

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**

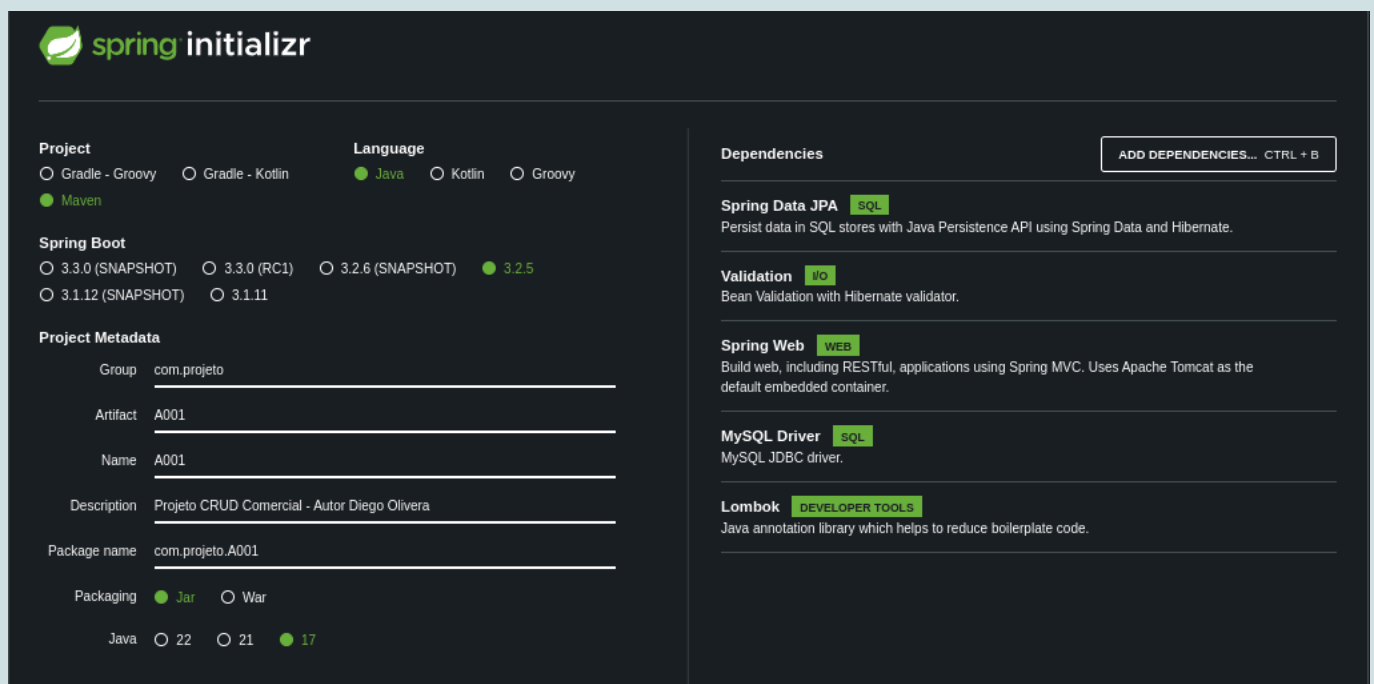
Utilizando o MySQL Workbench, foi criado o banco de dados “A001\_DB” em Localhost.



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:

```

Output - Run (A001)
Spring Boot (v3.2.5)
2024-05-03T16:29:38.656-03:00 INFO 630 --- [A001] [main] com.projeto.A001.A001Application : Starting A001Application using
2024-05-03T16:29:38.665-03:00 INFO 630 --- [A001] [main] com.projeto.A001.A001Application : No active profile set, falling
2024-05-03T16:29:40.138-03:00 INFO 630 --- [A001] [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA
2024-05-03T16:29:41.277-03:00 INFO 630 --- [A001] [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository
2024-05-03T16:29:41.308-03:00 INFO 630 --- [A001] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080
2024-05-03T16:29:41.308-03:00 INFO 630 --- [A001] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-05-03T16:29:41.308-03:00 INFO 630 --- [A001] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache.Catalina.core.StandardEngine]
2024-05-03T16:29:41.431-03:00 INFO 630 --- [A001] [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-05-03T16:29:41.434-03:00 INFO 630 --- [A001] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed
2024-05-03T16:29:41.793-03:00 INFO 630 --- [A001] [main] o.hibernate.jpa.internal.util.LogHelper : Processing PersistenceUnitInfo
2024-05-03T16:29:41.876-03:00 INFO 630 --- [A001] [main] org.hibernate.Version : Hibernate ORM core
2024-05-03T16:29:41.932-03:00 INFO 630 --- [A001] [main] o.h.c.internal.RegionFactoryInitiator : Second-level cache
2024-05-03T16:29:42.275-03:00 INFO 630 --- [A001] [main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignored
2024-05-03T16:29:42.315-03:00 INFO 630 --- [A001] [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2024-05-03T16:29:42.883-03:00 INFO 630 --- [A001] [main] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection
2024-05-03T16:29:42.886-03:00 INFO 630 --- [A001] [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed
2024-05-03T16:29:42.966-03:00 WARN 630 --- [A001] [main] org.hibernate.orm.deprecation : MySQLDialect does
2024-05-03T16:29:43.615-03:00 INFO 630 --- [A001] [main] o.h.e.t.j.p.i.JtaPlatformInitiator : No JTA platform available
2024-05-03T16:29:43.624-03:00 INFO 630 --- [A001] [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory
2024-05-03T16:29:43.739-03:00 WARN 630 --- [A001] [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled
2024-05-03T16:29:44.579-03:00 INFO 630 --- [A001] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http)
2024-05-03T16:29:44.584-03:00 INFO 630 --- [A001] [main] com.projeto.A001.A001Application : Started A001Application in 7.6s

```

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

### 2.3.2 - Classes Model

Cliente

```

1 package com.projeto.A001.model;
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue;
5 import jakarta.persistence.GenerationType;
6 import jakarta.persistence.Id;
7 import jakarta.persistence.Table;
8 import lombok.AllArgsConstructor;
9 import lombok.Data;
10 import lombok.NoArgsConstructor;
11 @Entity
12 @Data
13 @NoArgsConstructor
14 @AllArgsConstructor
15 @Table(name = "Cliente")
16 public class Cliente {
17     @Id
18     @GeneratedValue(strategy = GenerationType.AUTO)
19     private Integer idCliente;
20     private String nome;
21     private String cpf;
22     private String nacionalidade;
23     private String cep;
24     private String rua;
25     private String numero;
26     private String complemento;
27     private String bairro;
28     private String cidade;
29     private String estado;
30     private String telefone;
31     private String email;
32 }

```

Venda

```
1 package com.projeto.A001.model;
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue;
5 import jakarta.persistence.GenerationType;
6 import jakarta.persistence.Id;
7 import jakarta.persistence.JoinColumn;
8 import jakarta.persistence.ManyToOne;
9 import jakarta.persistence.Table;
10 import lombok.AllArgsConstructor;
11 import lombok.Data;
12 import lombok.NoArgsConstructor;
13 @Entity
14 @Data
15 @NoArgsConstructor
16 @AllArgsConstructor
17 @Table(name = "Venda")
18 public class Venda {
19     @Id
20     @GeneratedValue(strategy = GenerationType.AUTO)
21     private Integer idVenda;
22     @ManyToOne
23     @JoinColumn(name="id_Cliente")
24     private Cliente cliente;
25     private String serviço;
26     private String descrição;
27     private double valor;
28 }
```

## Contrato

```

1  package com.projeto.A001.model;
2
3  import jakarta.persistence.Entity;
4  import jakarta.persistence.GeneratedValue;
5  import jakarta.persistence.GenerationType;
6  import jakarta.persistence.Id;
7  import jakarta.persistence.JoinColumn;
8  import jakarta.persistence.OneToOne;
9  import jakarta.persistence.Table;
10 import java.util.Date;
11 import lombok.AllArgsConstructor;
12 import lombok.Data;
13 import lombok.NoArgsConstructor;
14 @Entity
15 @Data
16 @NoArgsConstructor
17 @AllArgsConstructor
18 @Table(name = "Contrato")
19 public class Contrato {
20     @Id
21     @GeneratedValue(strategy = GenerationType.AUTO)
22     private Integer idContrato;
23     @OneToOne
24     @JoinColumn(name="id_Venda")
25     private Venda venda;
26     private Date dataInicio;
27     private Date dataFinal;
28     private String formaPagamento;
29 }

```

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

```

1  package com.projeto.A001.repository;
2
3  import com.projeto.A001.model.Cliente;
4  import org.springframework.data.jpa.repository.JpaRepository;
5  import org.springframework.stereotype.Repository;
6
7  @Repository
8  public interface ClienteRepository extends JpaRepository<Cliente, Integer>{
9
10 }

```

#### VendaRepository

```

1  package com.projeto.A001.repository;
2
3  import com.projeto.A001.model.Venda;
4  import org.springframework.data.jpa.repository.JpaRepository;
5  import org.springframework.stereotype.Repository;
6
7  @Repository
8  public interface VendaRepository extends JpaRepository<Venda, Integer>{
9
10 }

```

## ContratoRepository

```

1 package com.projeto.A001.repository;
2
3 import com.projeto.A001.model.Contrato;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 @Repository
8 public interface ContratoRepository extends JpaRepository<Contrato, Integer>{
9
10 }

```

\*Poderá haver alterações ao longo do desenvolvimento.

## 2.3.4 - Classes Service

## ClienteService

```

9 @Service
10 public class ClienteService {
11     @Autowired
12     ClienteRepository clienteRepository;
13
14     public Cliente criar(Cliente c){...5 lines }
15
16     public List<Cliente> listarTodos(){...3 lines }
17
18     public Cliente buscaPorId(Integer id){...3 lines }
19
20     public void excluir(Integer id){...4 lines }
21
22     public Cliente atualizar(Integer id, Cliente c){...20 lines }
23 }

```

## VendaService

```

9 @Service
10 public class VendaService {
11
12     @Autowired
13     VendaRepository vendaRepository;
14
15     public Venda criar(Venda v){...5 lines }
16
17     public List<Venda> listarVendas() {...3 lines }
18
19     public List<Venda> listarVendasPorCliente(Integer idCliente) {...3 lines }
20
21     public Venda buscaPorId(Integer id) {...3 lines }
22
23     public void excluir(Integer id) {...4 lines }
24
25     public Venda atualizar(Integer id, Venda v) {...11 lines }
26 }

```

## ContratoService

```

9 @Service
10 public class ContratoService {
11
12     @Autowired
13     ContratoRepository contratoRepository;
14
15     public Contrato criar(Contrato c) {...5 lines }
16
17     public List<Contrato> listarContratos() {...3 lines }
18
19     public Contrato buscaPorId(Integer id) {...3 lines }
20
21     public void excluir(Integer id) {...4 lines }
22
23     public Contrato atualizar(Integer id, Contrato c) {...11 lines }
24 }

```

### 2.3.5 - Classes Controller

#### ClienteAPIController

```

18 @RestController
19 @RequestMapping("/cliente")
20 public class ClienteAPIController {
21     @Autowired
22     ClienteService clienteService;
23
24     @PostMapping("/save")
25     public ResponseEntity<Cliente> addCliente(@RequestBody Cliente c) {...4 lines }
26
27     @GetMapping("/listar")
28     public ResponseEntity<List> listar() {...4 lines }
29
30     @DeleteMapping("/excluir/{id}")
31     public ResponseEntity<?> delete(@PathVariable Integer id) {...4 lines }
32
33     @GetMapping("/pesquisar/{id}")
34     public ResponseEntity<Cliente> pesquisar(@PathVariable Integer id) {...4 lines }
35
36     @PutMapping("/atualizar/{id}")
37     public ResponseEntity<Cliente> editCliente(@PathVariable Integer id, @RequestBody Cliente c) {...4 lines }
38 }

```

#### VendaAPIController

```

18 @RestController
19 @RequestMapping("/venda")
20 public class VendaAPIController {
21     @Autowired
22     VendaService vendaService;
23
24     @PostMapping("/save")
25     public ResponseEntity<Venda> addVenda(@RequestBody Venda v) {...4 lines }
26
27     @GetMapping("/listar")
28     public ResponseEntity<List> listar() {...4 lines }
29
30     @GetMapping("/vendaspercliente/{idCliente}")
31     public ResponseEntity<List> vendasPorCliente(@PathVariable Integer idCliente) {...4 lines }
32
33     @DeleteMapping("/excluir/{id}")
34     public ResponseEntity<?> delete(@PathVariable Integer id) {...4 lines }
35
36     @GetMapping("/pesquisar/{id}")
37     public ResponseEntity<Venda> pesquisar(@PathVariable Integer id) {...4 lines }
38
39     @PutMapping("/atualizar/{id}")
40     public ResponseEntity<Venda> editVenda(@PathVariable Integer id, @RequestBody Venda v) {...4 lines }
41 }

```

#### ContratoAPIController

```

18 @RestController
19 @RequestMapping("/contrato")
20 public class ContratoAPIController {
21     @Autowired
22     ContratoService contratoService;
23
24     @PostMapping("/save")
25     public ResponseEntity<Contrato> addContrato(@RequestBody Contrato c) {...4 lines }
26
27     @GetMapping("/listar")
28     public ResponseEntity<List> listar() {...4 lines }
29
30     @GetMapping("/pesquisar/{id}")
31     public ResponseEntity<Contrato> pesquisar(@PathVariable Integer id) {...4 lines }
32
33     @DeleteMapping("/excluir/{id}")
34     public ResponseEntity<?> delete(@PathVariable Integer id) {...4 lines }
35
36     @PutMapping("/atualizar/{id}")
37     public ResponseEntity<Contrato> editContrato(@PathVariable Integer id, @RequestBody Contrato c) {...4 lines }
38 }

```



## 2.3.5 - Testando as requisições Http da API

### - Cliente

#### Create

The screenshot shows the Postman interface for a POST request to `localhost:8080/cliente/save`. The request body is a JSON object representing a client record. The response status is **201 Created**, with a time of 489 ms and a size of 452 B. A tooltip explains that the request was fulfilled and resulted in a new resource being created.

**Request:**

```
POST localhost:8080/cliente/save
```

**Body (JSON):**

```
{
  "nome": "Antonio José",
  "cpf": "111.222.333-44",
  "nacionalidade": "Brasileiro",
  "cep": "11222333",
  "rua": "Rua A",
  "numero": "123",
  "complemento": "casa 2",
  "bairro": "Ataliba Soares",
  "cidade": "Indaiatuba",
  "estado": "São Paulo",
  "telefone": "(11)92222-3333",
  "email": "antonio@zipmail.com.br"
}
```

**Response:** Status: 201 Created, Time: 489 ms, Size: 452 B

**201 Created**  
The request has been fulfilled and resulted in a new resource being created.

#### Read

The screenshot shows the Postman interface for a GET request to `localhost:8080/cliente/listar`. The response status is **200 OK**, with a time of 269 ms and a size of 449 B. A tooltip explains that this is a standard response for successful HTTP requests, where the actual response depends on the method used (GET or POST).

**Request:**

```
GET localhost:8080/cliente/listar
```

**Response:** Status: 200 OK, Time: 269 ms, Size: 449 B

**200 OK**  
Standard response for successful HTTP requests. The actual response will depend on the request method used. In a GET request, the response will contain an entity corresponding to the requested resource. In a POST request the response will contain an entity describing or containing the result of the action.

**Body (JSON):**

```
{
  "id": 1,
  "nome": "Antonio José",
  "cpf": "111.222.333-44",
  "nacionalidade": "Brasileiro",
  "cep": "11222333",
  "rua": "Rua A",
  "numero": "123",
  "complemento": "casa 2",
  "bairro": "Ataliba Soares",
  "cidade": "Indaiatuba",
  "estado": "São Paulo",
  "telefone": "(11)92222-3333",
  "email": "antonio@zipmail.com.br"
}
```

## Update

localhost:8080/cliente/atualizar/1

PUT localhost:8080/cliente/atualizar/1

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

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

```

1 {
2   "nome": "Antonio José Fagundes",
3   "cpf": "111.222.333-44",
4   "nacionalidade": "Brasileiro",
5   "cep": "11222333",
6   "rua": "Rua A",
7   "numero": "123",
8   "complemento": "casa 2",
9   "bairro": "Ataliba Soares",
10  "cidade": "Indaiatuba",
11  "estado": "São Paulo",
12  "telefone": "(11)92222-3333",
13  "email": "antonio@zipmail.com.br"
14 }

```

200 OK

Standard response for successful HTTP requests. The actual response will depend on the request method used. In a GET request, the response will contain an entity corresponding to the requested resource. In a POST request the response will contain an entity describing or containing the result of the action.

Status: 200 OK Time: 29 ms Size: 456 B Save as example

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

1 {

Find and replace Console Postbot Runner Capture requests Cookies Trash

## Delete

localhost:8080/cliente/excluir/1

DELETE localhost:8080/cliente/excluir/1

Params Authorization Headers (6) **Body** Pre-request Script Tests Settings

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

This request does not have a body

Status: 200 OK Time: 39 ms Size: 123 B Save as example

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize Text

1

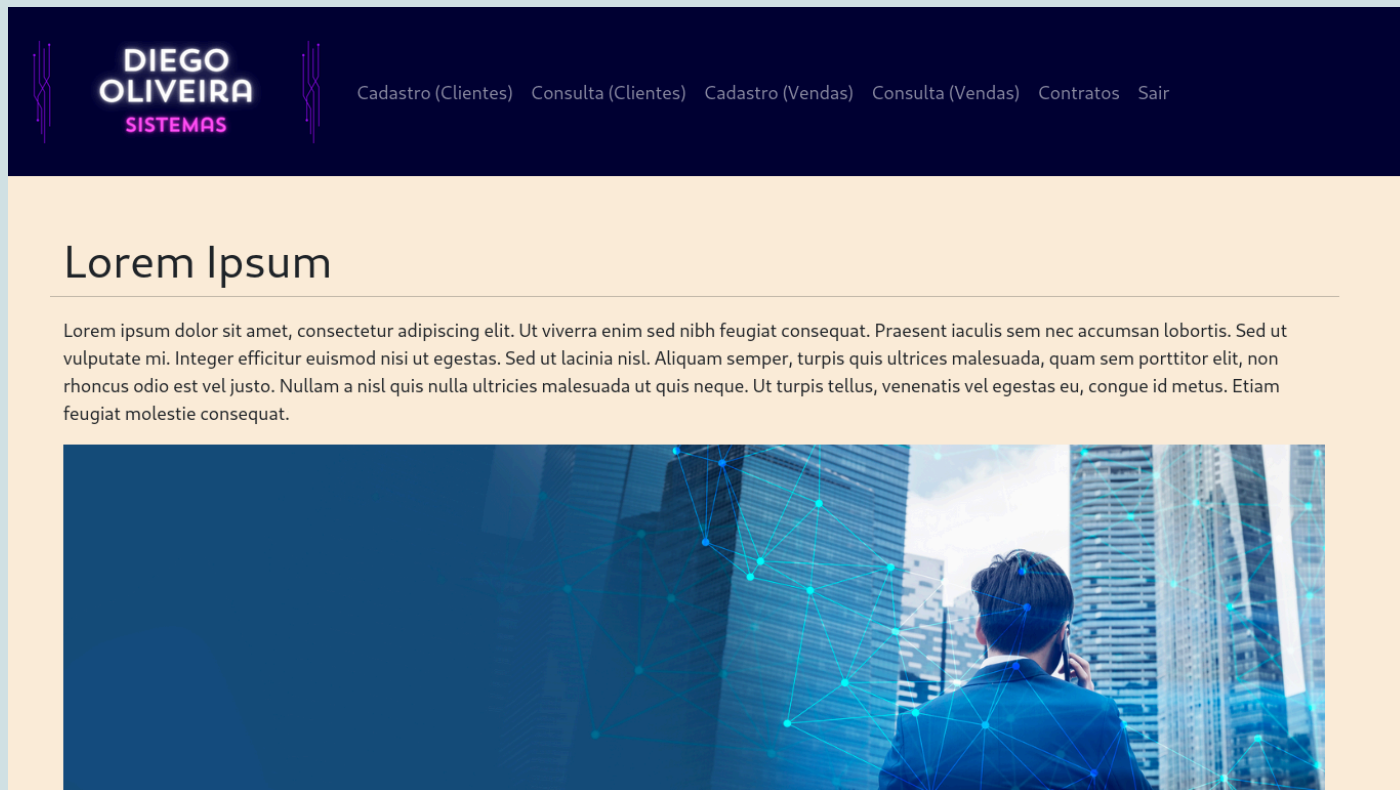
Find and replace Console Postbot Runner Capture requests Cookies Trash

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.