

---

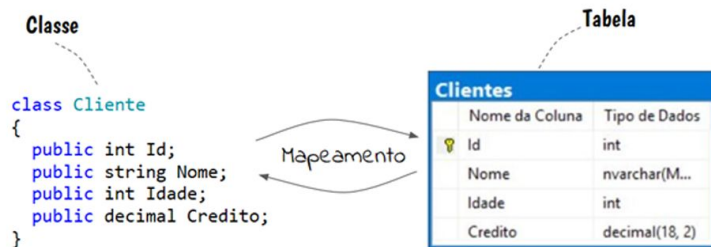
# Desenvolvimento WEB

-

Full Stack Completo: Java + React

---

# Relacionando Entidades no projeto do Back-end



# Relacionando Entidades

## 1) Implemente o CRUD de CategoriaProduto:

Backend



```
@Entity
@Table(name = "CategoriaProduto")
@SQLRestriction("habilitado = true")
@Builder
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class CategoriaProduto extends EntidadeAuditavel {

    @Column
    private String descricao;
```

```
@RestController
@RequestMapping("/api/categoriaproduto")
@CrossOrigin
public class CategoriaProdutoController {

    @Autowired
    private CategoriaProdutoService categoriaProdutoService;

    @PostMapping
    public ResponseEntity<CategoriaProduto> save(@RequestBody @Valid CategoriaProdutoRequest request) {

        CategoriaProduto categoriaProdutoNovo = request.build();
        CategoriaProduto categoriaProduto = categoriaProdutoService.save(categoriaProdutoNovo);
        return new ResponseEntity<CategoriaProduto>(categoriaProduto, HttpStatus.CREATED);
    }

    @GetMapping
    public List<CategoriaProduto> listarTodos() {

        return categoriaProdutoService.listarTodos();
    }

    @GetMapping("/{id}")
    public CategoriaProduto obterPorID(@PathVariable Long id) {

        return categoriaProdutoService.obterPorID(id);
    }

    @PutMapping("/{id}")
    public ResponseEntity<CategoriaProduto> update(@PathVariable("id") Long id, @RequestBody CategoriaProdutoRequest request) {

        categoriaProdutoService.update(id, request.build());
        return ResponseEntity.ok().build();
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> delete(@PathVariable Long id) {
```







# Relacionando Entidades

## 1) Implemente o CRUD de CategoriaProduto:

### Categoria de Produto

 Novo

Descrição	Ações
Teclado	 
Notebook	 
Mouse	 

### Categoria de Produto » Cadastro

Descrição \*

 Voltar

 Salvar

Frontend



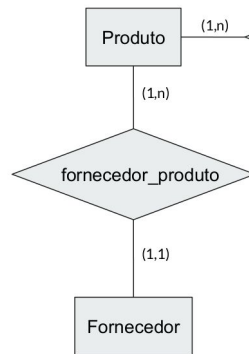
Navegador (Cliente)

# Relacionando Entidades

Vamos estudar a parte conceitual do relacionamento entre entidades no arquivo “Aula JPA - Prof. Francisco.pdf” observando os slides de 1 ao 18.

## JPA (JAVA PERSISTENCE API) - Mapeamento Objeto-Relacional (MOR) - Parte 3-4

Prof. Francisco do Nascimento



Muitos para Um

Sobre o relacionamento **Produto - Fornecedor**, temos:

- (a) cada produto tem um fornecedor e cada fornecedor tem um produto (UM PARA UM)
- (b) cada produto tem um fornecedor e cada fornecedor pode ter vários produtos (UM PARA MUITOS)
- (c) cada fornecedor pode ter vários produtos e cada produto pode ter vários fornecedores (MUITOS PARA MUITOS)

```
@Entity
public class Produto {
    private Fornecedor fornecedor;
}
```

(1) Valor único  
(2) Obrigatório

```
@Entity
public class Fornecedor {
    private List<Produto> produtos;
}
```

(1) Valor múltiplo  
(2) Opcional

# Relacionando Entidades :: 1 - N (Unidirecional)

Agora que entendemos a parte conceitual, vamos implementar um exemplo de um relacionamento do tipo “um para muitos” entre entidades no nosso projeto, iremos utilizar como exemplo o relacionando entre **Produto** e **CategoriaProduto**:

CategoriaProduto		
Atributo / Coluna	Tipo	Classe
id	Long	EntidadeNegocio
...		
versao	Long	EntidadeAuditavel
...		
descricao	String	CategoriaProduto



Produto		
Atributo / Coluna	Tipo	Classe
id	Long	EntidadeNegocio
...		
versao	Long	EntidadeAuditavel
...		
<b>categoria</b>	<b>CategoriaProduto</b>	
codigo	String	Produto
titulo	String	Produto
descricao	String	Produto
valorUnitario	Double	Produto
...		

# Relacionando Entidades :: 1 - N (Unidirecional)

2) Acrescente um novo atributo a classe `Produto` que irá representar o relacionamento de muitos para um entre `Produto` e `Categoria` de `Produto`:

```
...  
public class Produto extends EntidadeAuditavel {  
  
    @ManyToOne  
    private CategoriaProduto categoria;  
  
    @Column  
    private String codigo;  
  
    @Column  
    private String titulo;  
  
    @Column  
    private String descricao;  
  
    @Column  
    private Double valorUnitario;  
  
    ...  
}
```

# Relacionando Entidades :: 1 - N (Unidirecional)

3) Vamos refatorar o CRUD de Produto implementado em exercícios anteriores. Desta forma, acrescente o atributo abaixo na classe `ProdutoRequest`:

```
...
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class ProdutoRequest {

    private Long idCategoria;

    private String codigo;

    private String titulo;

    private String descricao;

    private Double valorUnitario;

    ...
}
```



# Relacionando Entidades :: 1 - N (Unidirecional)

4) Na classe `ProdutoController`, modifique o método `save` para consultar a categoria por ID e setar ela no produto a ser inserido:

```
...
public class ProdutoController {

    @Autowired
    private ProdutoService produtoService;

    @Autowired
    private CategoriaProdutoService categoriaProdutoService;

    @PostMapping
    public ResponseEntity<Produto> save(@RequestBody @Valid ProdutoRequest request) {

        Produto produtoNovo = request.build();
        produtoNovo.setCategoria(categoriaProdutoService.obterPorID(request.getIdCategoria()));
        Produto produto = produtoService.save(produtoNovo);
        return new ResponseEntity<Produto>(produto, HttpStatus.CREATED);
    }
    ...
}
```

# Relacionando Entidades :: 1 - N (Unidirecional)

5) Na classe `ProdutoController`, modifique o método `update` para consultar a categoria por ID e setar ela no produto a ser alterado:

```
...
public class ProdutoController {

    @Autowired
    private ProdutoService produtoService;

    @Autowired
    private CategoriaProdutoService categoriaProdutoService;

    @PutMapping("/{id}")
    public ResponseEntity<Produto> update(@PathVariable("id") Long id, @RequestBody ProdutoRequest request) {

        Produto produto = request.build();
        produto.setCategoria(categoriaProdutoService.obterPorID(request.getIdCategoria()));
        produtoService.update(id, produto);

        return ResponseEntity.ok().build();
    }
    ...
}
```

# Relacionando Entidades :: 1 - N (Unidirecional)

6) Na classe `ProdutoService`, modifique o método `update` setar a categoria alterada (linha em vermelho abaixo):

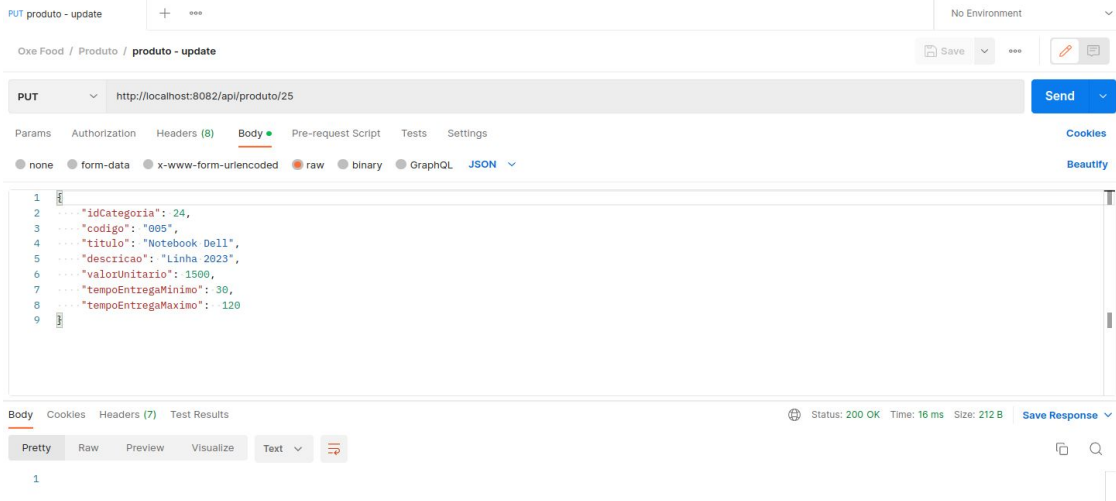
```
@Service
public class ProdutoService {
    ...
    @Transactional
    public void update(Long id, Produto produtoAlterado) {

        Produto produto = repository.findById(id).get();
        produto.setCategoria(produtoAlterado.getCategoria());
        produto.setCodigo(produtoAlterado.getCodigo());
        produto.setTitulo(produtoAlterado.getTitulo());
        produto.setDescricao(produtoAlterado.getDescricao());
        produto.setValorUnitario(produtoAlterado.getValorUnitario());
        produto.setTempoEntregaMinimo(produtoAlterado.getTempoEntregaMinimo());
        produto.setTempoEntregaMaximo(produtoAlterado.getTempoEntregaMaximo());

        produto.setVersao(produto.getVersao() + 1);
        repository.save(produto);
    }
    ...
}
```

# Relacionando Entidades :: 1 - N (Unidirecional)

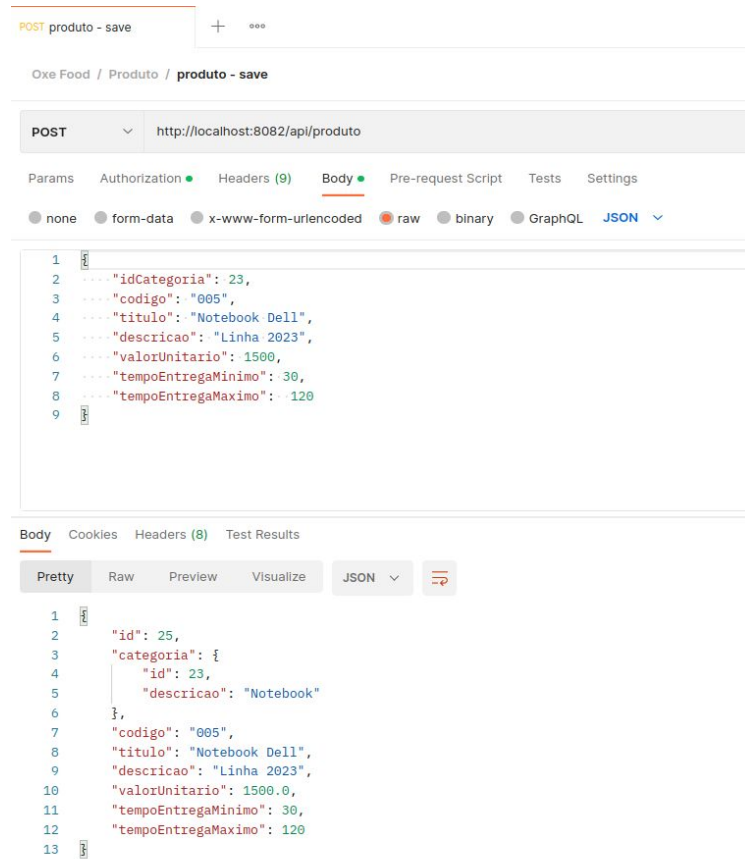
7) Teste a inclusão e a alteração do produto no Postman e verifique os valores no banco de dados



The image shows a Postman interface for a PUT request. The request is named "PUT produto - update" and is directed to the URL "http://localhost:8082/api/produto/25". The request body is in JSON format and contains the following data:

```
1 {
2   "idCategoria": 24,
3   "codigo": "005",
4   "titulo": "Notebook Dell",
5   "descricao": "Linha 2023",
6   "valorUnitario": 1500,
7   "tempoEntregaMinimo": 30,
8   "tempoEntregaMaximo": 120
9 }
```

The status bar at the bottom indicates a successful response with a status of 200 OK, a time of 16 ms, and a size of 212 B. The response is displayed in the "Body" tab in a pretty-printed JSON format.



The image shows a Postman interface for a POST request. The request is named "POST produto - save" and is directed to the URL "http://localhost:8082/api/produto". The request body is in JSON format and contains the following data:

```
1 {
2   "idCategoria": 23,
3   "codigo": "005",
4   "titulo": "Notebook Dell",
5   "descricao": "Linha 2023",
6   "valorUnitario": 1500,
7   "tempoEntregaMinimo": 30,
8   "tempoEntregaMaximo": 120
9 }
```

The response is displayed in the "Body" tab in a pretty-printed JSON format, showing the created product with an "id" of 25:

```
1 {
2   "id": 25,
3   "categoria": {
4     "id": 23,
5     "descricao": "Notebook"
6   },
7   "codigo": "005",
8   "titulo": "Notebook Dell",
9   "descricao": "Linha 2023",
10  "valorUnitario": 1500.0,
11  "tempoEntregaMinimo": 30,
12  "tempoEntregaMaximo": 120
13 }
```

# Dúvidas



# Exercícios



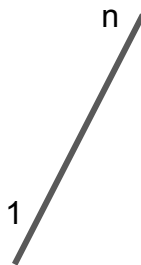
Implemente **apenas no projeto do back-end** (para o teste no Postman):

- (01) CRUD de `CategoriaProduto`
- (02) O relacionamento entre `Produto` -> `Categoria de Produto`

# Relacionando Entidades :: 1 - N (Bidirecional)

Agora vamos implementar um exemplo de um relacionamento de “um para muitos” bidirecional. Vamos utilizar no exemplo as entidades **Cliente** e **EnderecoCliente**, onde um cliente poderá ter uma lista de endereços relacionados a ele.

Cliente		
Atributo / Coluna	Tipo	Classe
id	Long	EntidadeNegocio
...		
versao	Long	EntidadeAuditavel
...		
<b>enderecos</b>	<b>List&lt;EnderecoCliente&gt;</b>	
nome	String	Cliente
dataNascimento	LocalDate	Cliente
...		



EnderecoCliente		
Atributo / Coluna	Tipo	Classe
id	Long	EntidadeNegocio
...		
versao	Long	EntidadeAuditavel
...		
<b>cliente</b>	<b>Cliente</b>	
rua	String	EnderecoCliente
numero	String	EnderecoCliente
bairro	String	EnderecoCliente
cep	String	EnderecoCliente
cidade	String	
estado	String	
complemento	String	

# Relacionando Entidades :: 1 - N (Bidirecional)

7) Dentro do pacote `../modelo/cliente` crie a classe `EnderecoCliente`, acrescente nesta classe um atributo para representar um relacionamento de 1 para n com o `Cliente`:

```
...
import com.fasterxml.jackson.annotation.JsonIgnore;
...
public class EnderecoCliente extends EntidadeAuditavel {

    @JsonIgnore
    @ManyToOne
    private Cliente cliente;

    @Column
    private String rua;

    @Column
    private String numero;

    @Column
    private String bairro;

    ...
}
```

```
...

@Column
private String cep;

@Column
private String cidade;

@Column
private String estado;

@Column
private String complemento;

}
```



# Relacionando Entidades :: 1 - N (Bidirecional)

8) Agora altere a classe `Cliente.java` acrescentando um atributo para armazenar uma lista de endereços do cliente:

```
import java.util.List;
...
public class Cliente extends EntidadeAuditavel {

    @OneToMany(mappedBy = "cliente", orphanRemoval = true, fetch = FetchType.EAGER)
    private List<EnderecoCliente> enderecos;

    @Column
    private String nome;

    @Column
    private LocalDate dataNascimento;

    @Column
    private String cpf;

    @Column
    private String foneCelular;

    @Column
    private String foneFixo;
}
```

# Relacionando Entidades :: 1 - N (Bidirecional)

9) Dentro do pacote `.../api/cliente` crie a classe `EnderecoClienteRequest.java`:

```
...
public class EnderecoClienteRequest {

    private String rua;

    private String numero;

    private String bairro;

    private String cep;

    private String cidade;

    private String estado;

    private String complemento;

    public EnderecoCliente build() {

        return EnderecoCliente.builder()
            .rua(rua)
            .numero(numero)
            .bairro(bairro)
            .cep(cep)
            .cidade(cidade)
            .estado(estado)
            .complemento(complemento)
            .build();

    }
}
```

# Relacionando Entidades :: 1 - N (Bidirecional)

10) No pacote `../modelo/cliente` crie a interface `EnderecoClienteRepository`:

```
package br.com.ifpe.oxefood.modelo.cliente;

import org.springframework.data.jpa.repository.JpaRepository;

public interface EnderecoClienteRepository extends JpaRepository<EnderecoCliente, Long> {

}
```

# Relacionando Entidades :: 1 - N (Bidirecional)

11) Na classe `ClienteController` implemente 3 métodos, um para adicionar um endereço a um cliente, outro para alterar um endereço de um cliente e outro para remover:

```
...
public class ClienteController {
...

    @PostMapping("/endereco/{clienteId}")
    public ResponseEntity<EnderecoCliente>adicionarEnderecoCliente(@PathVariable("clienteId") Long clienteId, @RequestBody @Valid EnderecoClienteRequest request) {

        EnderecoCliente endereco = clienteService.adicionarEnderecoCliente(clienteId, request.build());
        return new ResponseEntity<EnderecoCliente>(endereco, HttpStatus.CREATED);
    }

    @PutMapping("/endereco/{enderecoId}")
    public ResponseEntity<EnderecoCliente>atualizarEnderecoCliente(@PathVariable("enderecoId") Long enderecoId, @RequestBody EnderecoClienteRequest request) {

        EnderecoCliente endereco = clienteService.atualizarEnderecoCliente(enderecoId, request.build());
        return new ResponseEntity<EnderecoCliente>(endereco, HttpStatus.OK);
    }

    @DeleteMapping("/endereco/{enderecoId}")
    public ResponseEntity<Void>removerEnderecoCliente(@PathVariable("enderecoId") Long enderecoId) {

        clienteService.removerEnderecoCliente(enderecoId);
        return ResponseEntity.noContent().build();
    }
...
}
```

# Relacionando Entidades :: 1 - N (Bidirecional)

12.1) Na classe `ClienteService` implemente os métodos para incluir, alterar e remover o endereço do cliente:

```
...
public class ClienteService {
...

@Transactional
public EnderecoCliente adicionarEnderecoCliente(Long clienteId, EnderecoCliente endereco) {

    Cliente cliente = this.findById(clienteId);

    //Primeiro salva o EnderecoCliente:

    endereco.setCliente(cliente);
    endereco.setHabilitado(Boolean.TRUE);
    enderecoClienteRepository.save(endereco);

    //Depois acrescenta o endereço criado ao cliente e atualiza o cliente:

    List<EnderecoCliente> listaEnderecoCliente = cliente.getEnderecos();

    if (listaEnderecoCliente == null) {
        listaEnderecoCliente = new ArrayList<EnderecoCliente>();
    }

    listaEnderecoCliente.add(endereco);
    cliente.setEnderecos(listaEnderecoCliente);
    cliente.setVersao(cliente.getVersao() + 1);
    repository.save(cliente);

    return endereco;
}
```

# Relacionando Entidades :: 1 - N (Bidirecional)

12.2) Na classe `ClienteService` implemente os métodos para incluir, alterar e remover o endereço do cliente:

```
...
public class ClienteService {
    ...

    @Transactional
    public EnderecoCliente atualizarEnderecoCliente(Long id, EnderecoCliente enderecoAlterado) {

        EnderecoCliente endereco = enderecoClienteRepository.findById(id).get();
        endereco.setRua(enderecoAlterado.getRua());
        endereco.setNumero(enderecoAlterado.getNumero());
        endereco.setBairro(enderecoAlterado.getBairro());
        endereco.setCep(enderecoAlterado.getCep());
        endereco.setCidade(enderecoAlterado.getCidade());
        endereco.setEstado(enderecoAlterado.getEstado());
        endereco.setComplemento(enderecoAlterado.getComplemento());

        return enderecoClienteRepository.save(endereco);
    }
    ...
}
```

# Relacionando Entidades :: 1 - N (Bidirecional)

12.3) Na classe `ClienteService` implemente os métodos para incluir, alterar e remover o endereço do cliente:

```
...
public class ClienteService {
...

    @Transactional
    public void removerEnderecoCliente(Long id) {

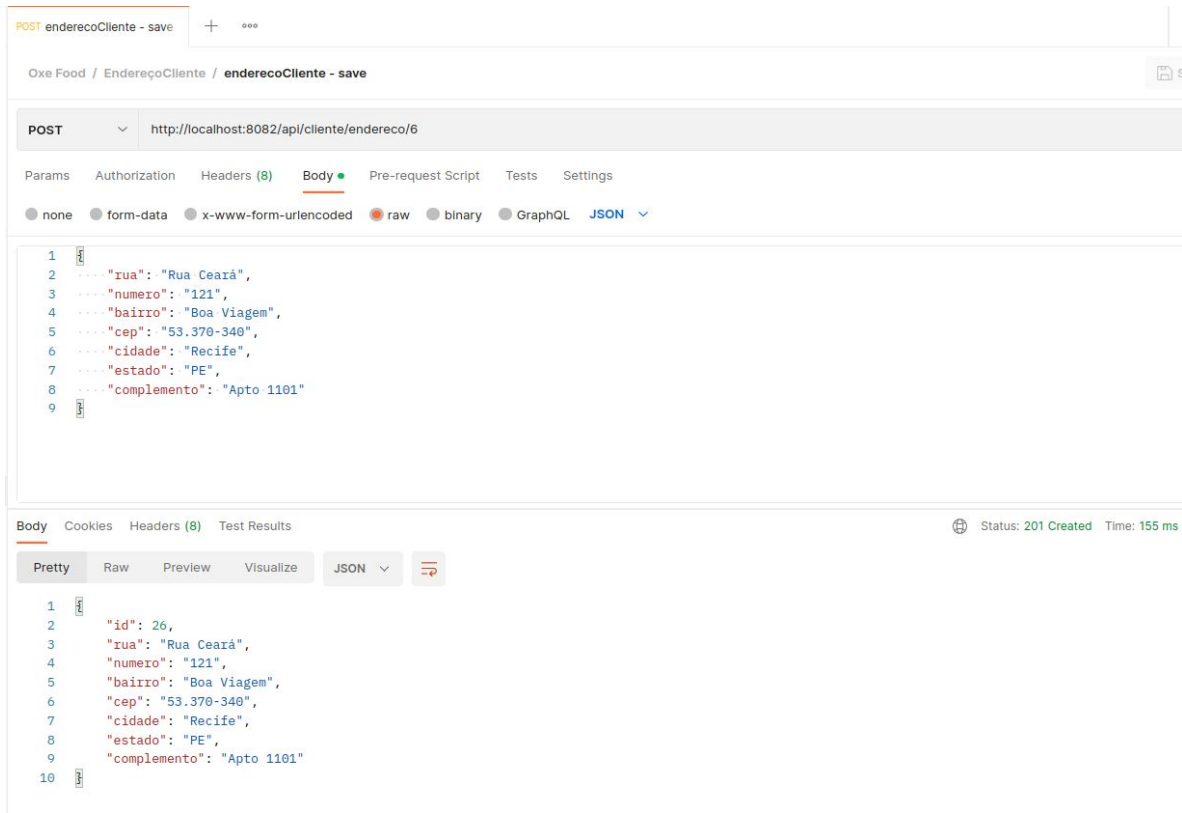
        EnderecoCliente endereco = enderecoClienteRepository.findById(id).get();
        endereco.setHabilitado(Boolean.FALSE);
        enderecoClienteRepository.save(endereco);

        Cliente cliente = this.obterPorID(endereco.getCliente().getId());
        cliente.getEnderecos().remove(endereco);
        cliente.setVersao(cliente.getVersao() + 1);
        repository.save(cliente);
    }

...
}
```

# Relacionando Entidades :: 1 - N (Bidirecional)

13.1) Teste a inclusão de um endereço de cliente no Postman:





# Relacionando Entidades :: 1 - N (Bidirecional)

13.2) Consulte o cliente no Postman para verificar a lista de endereços retornada:

The screenshot shows a Postman interface for a GET request to 'http://localhost:8082/apl/cliente/6'. The response status is 200 OK with a time of 19 ms. The response body is displayed in JSON format, showing a client with ID 6 and a list of two addresses.

**Query Params**

KEY	VALUE	DESCRIPTION
Key	Value	Description

**Body** | Cookies | Headers (8) | Test Results

Pretty | Raw | Preview | Visualize | JSON |

```
1  {
2    "id": 6,
3    "enderecos": [
4      {
5        "id": 26,
6        "rua": "Rua Ceará",
7        "numero": "121",
8        "bairro": "Boa Viagem",
9        "cep": "53.370-340",
10       "cidade": "Recife",
11       "estado": "PE",
12       "complemento": "Apto 1101"
13     },
14     {
15       "id": 27,
16       "rua": "Av. Canxagá",
17       "numero": "18",
18       "bairro": "Cordeiro",
19       "cep": "53.370-340",
20       "cidade": "Recife",
21       "estado": "PE",
22       "complemento": "Lote 02"
23     }
24   ],
25   "nome": "Roberto Alencar 123456",
```

# Dúvidas



# Exercícios



Implemente no projeto do back-end (para o teste no Postman):

- `EnderecoCliente -> Cliente`

The image features a white background with two large, solid green abstract shapes. One shape is a semi-circle on the left side, and the other is a more complex, organic shape on the right side. Centered between these shapes is the text "Obrigado !".

Obrigado !