

# Criação de um Aplicação para cadastro de Usuários e Veículos com Spring + Hibernate

## Introdução

No mundo atual onde existem diversas tecnologias e sistemas diferentes no mercado, surgiu a necessidade da criação de API (*Application Programming Interface*) que é basicamente uma Interface para que um sistema se comunique com outro sistema compartilhando suas características.

Atualmente o processo de desenvolvimento de APIs pode ser feito de diversas maneiras, porém vale ressaltar que existem tecnologias capazes de ajudar na hora do desenvolvimento abstraindo códigos, assim o desenvolvedor terá mais tempo para se preocupar com outras coisas importantes como por exemplo as regras de negócios, arquitetura, documentação etc.

## Situação Proposta

Irei neste post demonstrar a implementação de forma simples e completa de uma API REST, tendo como objetivo a criação de três end-points (pontos de comunicação de acesso da aplicação), o primeiro end-point irá fazer o cadastro do usuário, o segundo end-point irá realizar o cadastro do veículo (aqui também iremos consumir a API da FIPE) e por último o terceiro end-point listará os veículos para o usuário especificado.

## Tecnologias utilizadas

**Spring MVC:** Nos ajuda no desenvolvimento de uma aplicação WEB, possuindo todas as funcionalidades necessárias, para receber requisições HTTP, analisar os dados recebidos, processá-los e depois preparar uma resposta.

**Spring Boot:** O Spring boot nos permite uma fácil configuração de tudo que é necessário para o projeto, adicionando dependências para o projeto, essas dependências podem ser do próprio Spring ou de terceiros que após a adição será configurado pelo Spring.

**Spring Cloud Feign:** Projeto incluído dentro do Spring Cloud, serve para integração e consumo de API de uma maneira simples, com pouco código já é possível realizar requisições a uma API.

**Spring Data JPA:** Foi criado para facilitar a nossa vida durante o desenvolvimento da camada de acesso ao banco de dados, o JPA nos ajuda nas implementações dos repositórios tendo como padrão a utilização do Hibernate.

## Arquitetura

O MVC (Model, View e Controller) é um padrão de arquitetura mais utilizado na criação de API Rest pela sua separação muito clara de camadas, ajudando na redução de acoplamento e aumento de coesão nas classes do projeto, por isso irei organizar as classes dessa maneira.

## Início do projeto e Implementação

### pom.xml

Após a criação do projeto o Maven (ferramenta de automação de compilação) irá carregar todas as dependências necessárias para o projeto, as dependências podem ser visualizadas no arquivo pom.xml, lembrando que podemos adicionar ou remover as dependências sempre que necessário.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
```

### application.properties

Na construção da aplicação irei utilizar o banco de dados Mysql, por motivos de familiaridade com o banco, por isso é preciso ajustar o arquivo application.properties

```
1 spring.jpa.hibernate.ddl-auto=create-drop
2
3 spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/Zup
4 spring.datasource.username=root
5 spring.datasource.password=123456
6 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
7
```

Com essas configurações consigo acessar o banco de dados e acrescentei uma configuração para recriar o banco quando a aplicação for inicializada.

## Model

Para representar as tabelas no banco criei as classes `Usuario` e `Veiculo` junto com os métodos necessários, para me ajudar no desenvolvimento utilizei as anotações:

**@Entity:** Utilizada para informar que a classe é uma entidade, ou seja, uma tabela no banco de dados.

**@Id:** Anotação que identifica o campo como chave primária no banco de dados

**@GeneratedValue:** Informa que o responsável pela geração dos valores da chave primária é o banco de dados, o strategy é um atributo que modifica a estratégia da geração, colocando a estratégia como `IDENTITY` o JPA sabe que os valores devem ser gerados a partir da coluna de autoincremento do banco.

**@OneToMany:** Representa a relação de tabelas.

**@JoinColumn:** Indica a ligação de tabelas através da chave.

**@Deprecated:** Apenas um padrão meu que indica depreciação, utilizo para avisar (junto com o comentário) que o método construtor vazio não deve ser utilizado pelo desenvolvedor apenas pelo hibernate.

```
10  @Entity
11  public class Usuario {
12
13      @Id
14      @GeneratedValue(strategy = GenerationType.IDENTITY)
15      private Long idUsuario;
16      private String nome;
17      private String email;
18      private String cpf;
19      private LocalDate dtNascimento;
20
21      @OneToMany
22      @JoinColumn(name = "idUsuario")
23      private List<Veiculo> veiculos;
24
25      /**
26       * constructor para uso exclusivo do Hibernate
27       */
28      @Deprecated
29      public Usuario() {
30      }
31
32      public Usuario(String nome, String email, String cpf, LocalDate dtNascimento) {
33          this.nome = nome;
34          this.email = email;
35          this.cpf = cpf;
36          this.dtNascimento = dtNascimento;
37      }
38
39      public Long getIdUsuario() {
40          return idUsuario;
41      }
```

```
43      public String getNome() {
44          return nome;
45      }
46
47      public String getEmail() {
48          return email;
49      }
50
51      public String getCpf() {
52          return cpf;
53      }
54
55      public LocalDate getDtNascimento() {
56          return dtNascimento;
57      }
58
59      public List<Veiculo> getVeiculos() {
60          return veiculos;
61      }
62  }
63
```

```
8  @Entity
9  public class Veiculo {
10
11      @Id
12      @GeneratedValue(strategy = GenerationType.IDENTITY)
13      private Long idVeiculo;
14      private String marca;
15      private String modelo;
16      private String ano;
17      private String valor;
18      private Long idUsuario;
19      private Integer anoModelo;
20
21      /**
22       * constructor para uso exclusivo do Hibernate
23       */
24      @Deprecated
25      public Veiculo() {
26      }
27
28      public Veiculo(String marca, String modelo, String ano, String valor,
29                      Long idUsuario, Integer anoModelo) {
30          this.marca = marca;
31          this.modelo = modelo;
32          this.ano = ano;
33          this.valor = valor;
34          this.idUsuario = idUsuario;
35          this.anoModelo = anoModelo;
36      }
37
38      public Long getIdVeiculo() {
39          return idVeiculo;
40      }
```

```

41
42     public String getMarca() {
43         return marca;
44     }
45
46     public String getModelo() {
47         return modelo;
48     }
49
50     public String getAno() {
51         return ano;
52     }
53
54     public String getValor() {
55         return valor;
56     }
57
58     public Long getIdUsuario() {
59         return idUsuario;
60     }
61
62     public Integer getAnoModelo() {
63         return anoModelo;
64     }
65 }

```

As classes DTO (Data Transfer Object) são muito importantes em uma criação de API, elas são responsáveis por manipular as respostas das requisições, apresentando apenas os campos que o desenvolvedor queira que o consumidor da API tenha acesso, como por exemplo um campo de senha não deve ser mostrado. Na minha aplicação criei as seguintes classes DTO:

UsuarioDTO - Para o retorno das requisições do Usuário.

```
5 public class UsuarioDTO {
6     private String nome;
7     private String email;
8     private String cpf;
9     private LocalDate dtNascimento;
10
11 @
12     public UsuarioDTO(Usuario usuario) {
13         this.nome = usuario.getNome();
14         this.email = usuario.getEmail();
15         this.cpf = usuario.getCpf();
16         this.dtNascimento = usuario.getDtNascimento();
17     }
18
19     public String getNome() { return nome; }
20
21     public String getEmail() { return email; }
22
23     public String getCpf() { return cpf; }
24
25     public LocalDate getDtNascimento() { return dtNascimento; }
26 }
27
28
29
30
31
32
33
34
```

VeiculoDTO - Para o retorno do Veículo.

```
3      public class VeiculoDTO {
4          private String marca;
5          private String modelo;
6          private String ano;
7          private String valor;
8          private Long idUsuario;
9
10
11  @   public VeiculoDTO(Veiculo veiculo) {
12      this.marca = veiculo.getMarca();
13      this.modelo = veiculo.getModelo();
14      this.ano = veiculo.getAno();
15      this.valor = veiculo.getValor();
16      this.idUsuario = veiculo.getIdUsuario();
17  }
18
19  public String getMarca() { return marca; }
22
23  public String getModelo() { return modelo; }
26
27  public String getAno() { return ano; }
30
31  public String getValor() { return valor; }
34
35  public Long getIdUsuario() { return idUsuario; }
38  }
```



UsuarioComVeiculoDTO - Para o retorno do Usuário com a lista de seus Veiculos cadastrados.

```
7 public class UsuarioComVeiculosDTO {
8
9     private String nome;
10    private String email;
11    private String cpf;
12    private LocalDate dtNascimento;
13    private List<VeiculoDetalhesDTO> veiculos;
14
15    @
16    public UsuarioComVeiculosDTO(Usuario usuario) {
17        this.nome = usuario.getNome();
18        this.email = usuario.getEmail();
19        this.cpf = usuario.getCpf();
20        this.dtNascimento = usuario.getDtNascimento();
21        this.veiculos = usuario.getVeiculos().stream()
22            .map(VeiculoDetalhesDTO::new)
23            .collect(Collectors.toList());
24    }
25
26    public String getNome() { return nome; }
27
28
29    public String getEmail() { return email; }
30
31
32    public String getCpf() { return cpf; }
33
34
35    public LocalDate getDtNascimento() { return dtNascimento; }
36
37
38    public List<VeiculoDetalhesDTO> getVeiculos() { return veiculos; }
39
40
41
42
43
44
45 }
```

VeiculoDetalhesDTO - para o retorno do Veículo, no método construtor também é onde acontece a chamada da lógica responsável pela atribuição dos campos “diaRodizio” e “rodizioAtivo”.

```
8 public class VeiculoDetalhesDTO {
9     private Long idVeiculo;
10    private String marca;
11    private String modelo;
12    private String ano;
13    private String valor;
14    private String diaRodizio;
15    private Boolean rodizioAtivo;
16
17    @
18    public VeiculoDetalhesDTO(Veiculo veiculo) {
19        this.idVeiculo = veiculo.getIdVeiculo();
20        this.marca = veiculo.getMarca();
21        this.modelo = veiculo.getModelo();
22        this.ano = veiculo.getAno();
23        this.valor = veiculo.getValor();
24
25        int ultimoDigitoDoAno = veiculo.getAnoModelo() % 10;
26        Rodizio rodizio = Rodizio.buscarDiaDeRodizio(ultimoDigitoDoAno);
27
28        this.diaRodizio = rodizio.getDiaDaSemanaFormatado();
29        this.rodizioAtivo = rodizio.rodizioAtivo();
30    }
31
32    public Long getIdVeiculo() { return idVeiculo; }
33
34    public String getMarca() { return marca; }
35
36    public String getModelo() { return modelo; }
37
38    public String getAno() { return ano; }
39
40    public String getValor() { return valor; }
41
42    public String getDiaRodizio() { return diaRodizio; }
43
44    public Boolean getRodizioAtivo() { return rodizioAtivo; }
45
46    }
47
48    }
```

Para receber as respostas feitas pelas requisições da API da FIPE, criei mais algumas classes, vale lembrar que embora elas tenham alguns atributos e métodos semelhantes, é uma boa prática criar uma para cada finalidade evitando assim problemas futuros, como por exemplo caso a estrutura da API mude terei que modificar apenas onde ocorreu essa mudança e não todo o meu código, as classes são:

BuscaAnoResponse – Responsável pelo conteúdo da requisição que nos retorna as informações do Ano do veículo.

```
3 public class BuscaAnoResponse {
4
5     private String nome;
6     private String codigo;
7
8     public BuscaAnoResponse(String nome, String codigo) {
9         this.nome = nome;
10        this.codigo = codigo;
11    }
12
13    public String getNome() { return nome; }
16
17    public String getCodigo() { return codigo; }
20 }
21
22
```

BuscaAnoResponse – Responsável pelo conteúdo da requisição que nos retorna as Marcas dos Veiculos.

```
3 public class BuscaMarcaResponse {
4
5     private String nome;
6     private String codigo;
7
8     public BuscaMarcaResponse(String nome, String codigo) {
9         this.nome = nome;
10        this.codigo = codigo;
11    }
12
13    public String getNome() { return nome; }
16
17    public String getCodigo() { return codigo; }
20 }
21
```

BuscaModeloResponse – Responsável pelo conteúdo dos Modelos dos Veículos.

```
2
3 public class BuscaModeloResponse {
4
5     private String nome;
6     private String codigo;
7
8     public BuscaModeloResponse(String nome, String codigo) {
9         this.nome = nome;
10        this.codigo = codigo;
11    }
12
13    public String getNome() { return nome; }
14
15
16
17    public String getCodigo() { return codigo; }
18
19 }
20
21 |
```

BuscaModeloResponseWrapper – Como a requisição de modelos retorna duas listas de objetos, foi necessário criar essa classe para pegar os dados necessários.

**@JsonProperty** – Essa anotação da biblioteca Jackson é utilizada para informar qual campo do Json queremos, nesse caso eu queria a lista de objetos “modelos”.

```
8 public class BuscaModeloResponseWrapper {
9
10    @JsonProperty("modelos")
11    private List<BuscaModeloResponse> modelos;
12
13    public void setModelos(List<BuscaModeloResponse> modelos) { this.modelos = modelos; }
14
15
16
17    public Optional<String> buscarCodigoDoModelo(String modelo) {
18        return modelos
19            .stream()
20            .filter(modeloResponse -> modeloResponse.getNome().equals(modelo))
21            .map(BuscaModeloResponse::getCodigo)
22            .findFirst();
23    }
24 }
25 |
```

BuscaModeloResponseWrapper – Classe criada para ter o resultado necessário da requisição que nos retorna um carro específico, também utilizei a anotação **@JsonProperty**.

```
7 public class BuscaValorResponse {
8
9     @JsonProperty("Valor")
10    private String valor;
11    @JsonProperty("AnoModelo")
12    private Integer anoModelo;
13
14    public void setValor(String valor) { this.valor = valor; }
17
18    public String getValor() { return valor; }
21
22    public Integer getAnoModelo() { return anoModelo; }
25
26    public void setAnoModelo(Integer ano) { this.anoModelo = ano; }
29 }
```

## Enum

Na realização do terceiro End-point vou precisar retornar alguns campos a mais, então criei um Enum que chamei de Rodizio, os métodos desse Enum irão retornar o dia da semana de acordo com o ano do carro, e também irá dizer se o dia da semana retornado é ou não o dia da semana atual.

```
9 public enum Rodizio {
10
11     SEGUNDA_FEIRA(Arrays.asList(0, 1), diaDaSemanaFormatado: "segunda-feira", DayOfWeek.MONDAY),
12     TERCA_FEIRA(Arrays.asList(2, 3), diaDaSemanaFormatado: "terça-feira", DayOfWeek.TUESDAY),
13     QUARTA_FEIRA(Arrays.asList(4, 5), diaDaSemanaFormatado: "quarta-feira", DayOfWeek.WEDNESDAY),
14     QUINTA_FEIRA(Arrays.asList(6, 7), diaDaSemanaFormatado: "quinta-feira", DayOfWeek.THURSDAY),
15     SEXTA_FEIRA(Arrays.asList(8, 9), diaDaSemanaFormatado: "sexta-feira", DayOfWeek.FRIDAY),
16     NAO_MAPEADO(new ArrayList<>(), diaDaSemanaFormatado: "não mapeado", diaDaSemana: null);
17
18     private final List<Integer> valores;
19     private final String diaDaSemanaFormatado;
20     private final DayOfWeek diaDaSemana;
21
22     Rodizio(List<Integer> valores, String diaDaSemanaFormatado, DayOfWeek diaDaSemana) {
23         this.valores = valores;
24         this.diaDaSemanaFormatado = diaDaSemanaFormatado;
25         this.diaDaSemana = diaDaSemana;
26     }
27
28     public List<Integer> getValores() { return this.valores; }
29
30     public String getDiaDaSemanaFormatado() { return this.diaDaSemanaFormatado; }
31
32     public DayOfWeek getDiaDaSemana() { return this.diaDaSemana; }
33
34
35     public static Rodizio buscarDiaDeRodizio(int ultimoDigito) {
36         for (Rodizio rodizio : Rodizio.values()) {
37             if (rodizio.getValores().contains(ultimoDigito)) {
38                 return rodizio;
39             }
40         }
41         return NAO_MAPEADO;
42     }
43
44     public boolean rodizioAtivo() {
45         DayOfWeek diaDaSemana = DayOfWeek.WEDNESDAY;
46         return (this.getDiaDaSemana().equals(diaDaSemana));
47     }
48 }
49
50
51 }
```

## Repository

Os repository são feitos para adicionar uma camada de abstração para acesso aos dados, para isso criei duas Interfaces “VeiculoRepository” e “UsuarioRepository”, ambas classes fazem um extends (herança) de **JpaRepository** passando a entidade e o tipo de dado do campo id. O **JpaRepository** implementa alguns métodos, como por exemplo o save().

**@Repository** – Anotação que utilizei para informar que as interfaces são Repository.

```
6  @Repository
7  public interface VeiculoRepository extends JpaRepository<Veiculo, Long> {
8  }
```

Nessa interface fiz as assinaturas de dois métodos boolean: `existsByEmail` e `existsByCpf`, irei utilizar eles na controller para saber se o email ou cpf já foram cadastrados.

```
10 @Repository
11 public interface UsuarioRepository extends JpaRepository<Usuario, Long> {
12
13     boolean existsByEmail(String email);
14
15     boolean existsByCpf(String cpf);
16 }
```

## Classes Form

Diferente das classes DTO que retornam dados, as classes de input manipulam os dados de entrada da aplicação, com a utilização dessa classe podemos definir apenas os campos necessários que o usuário irá enviar, como por exemplo o usuário não terá em momento algum acesso ao campo `idUsuario`, por isso criei duas classes para receber valores a classe “`UsuarioForm`” e “`VeiculoForm`”. É nessas classes que irei fazer as validações necessárias utilizando o Bean Validation, biblioteca que nos permite validar os valores na própria classe, na aplicação utilizei as restrições.

**@NotBlank** – Não permite valores nulos nem vazios.

**@Email** – Realiza as validações de um email, essa anotação verifica se o conteúdo do campo possui “@”, se existe caracteres antes e depois do “@” e outras técnicas para garantir se o email é válido.

**@Cpf** – Validação de um CPF real.

**@Past** – Valida se a data é anterior a atual.

**@JsonFormat** – O atributo `pattern` defini o padrão de como serializar o Objeto JSON.

```
11 public class UsuarioForm {
12
13     @NotBlank
14     private String nome;
15
16     @Email
17     private String email;
18
19     @NotBlank
20     @CPF
21     private String cpf;
22
23     @JsonFormat(pattern = "dd/MM/yyyy", shape = JsonFormat.Shape.STRING)
24     @Past
25     private LocalDate dtNascimento;
26
27     public UsuarioForm(String nome, String email, String cpf) {
28         this.nome = nome;
29         this.email = email;
30         this.cpf = cpf;
31     }
32
33     public void setDtNascimento(LocalDate dtNascimento) { this.dtNascimento = dtNascimento; }
34
35     public Usuario transformarEmUsuario() {
36         return new Usuario(nome, email, cpf, dtNascimento);
37     }
38
39
40     public String getEmail() { return email; }
41
42
43     public String getCpf() { return cpf; }
44
45
46 }
47
```



```

7   public class VeiculoForm {
8
9       @NotBlank
10      private String marca;
11      @NotBlank
12      private String modelo;
13      @NotBlank
14      private String ano;
15
16      public VeiculoForm(String marca, String modelo, String ano) {
17          this.marca = marca;
18          this.modelo = modelo;
19          this.ano = ano;
20      }
21
22      public String getMarca() { return marca; }
23
24      public String getModelo() { return modelo; }
25
26      public String getAno() { return ano; }
27
28      public Veiculo transformarEmVeiculo(String valor, Long idUsuario, Integer anoModelo){
29          return new Veiculo(marca, modelo, ano, valor, idUsuario, anoModelo);
30      }
31  }

```

## Tratamento de Exceções

Em criação de APIs é muito importante levar em consideração que alguns cenários podem acabar dando erro, por isso devemos tratar esses erros, fazendo o tratamento de exceções, o usuário final receberá uma mensagem amigável dizendo aonde foi o seu erro ao invés de uma mensagem enorme e de difícil entendimento.

Criei a classe “UsuarioNaoEncontradoException”, para quando houver uma busca por um usuário e ele não existir no banco de dados, o tratamento dessa exceção será uma resposta com status HTTP 404 (Not Found), consigo utilizar essa classe para exceções pois a classe herda as características da RuntimeException.

```

3   public class UsuarioNaoEncontradoException extends RuntimeException {
4
5       public UsuarioNaoEncontradoException(String mensagem) {
6           super(mensagem);
7       }
8   }
9

```

Para a visualização da mensagem tratada criei a classe “CampoComErro” (representação dos campos que estão com erros e uma mensagem) e “ApenasMensagemErro” (representação de uma mensagem de erro).

```

3      public class CampoComErro {
4
5          private String campo;
6          private String erro;
7
8          public CampoComErro(String campo, String erro) {
9              this.campo = campo;
10             this.erro = erro;
11         }
12
13         public String getCampo() { return campo; }
14
15
16
17         public String getErro() { return erro; }
18     }
19
20
21
22

```

```

3      public class ApenasMensagemErro {
4
5          private String erro;
6
7          public ApenasMensagemErro(String erro) {
8              this.erro = erro;
9          }
10
11
12
13         public String getErro() {
14             return erro;
15         }
16     }
17
18
19
20

```

Para fazer o tratamento das exception criei uma classe chamada `ApiExceptionHandler` com as seguintes anotações:

**@RestControllerAdvice** – Identificação da classe como controladora das exceções, com a estrutura Rest.

**@ResponseStatus** – Define o Status Code da Response.

**@ExceptionHandler** – O Spring irá rastrear essa anotação e saberá que esse método irá lidar com uma exceção nesse caso o `MethodArgumentNotValidException` e o `UsuarioNaoEncontradoException`.

```

12  @RestControllerAdvice
13  public class ApiExceptionHandler {
14
15      @ResponseStatus(HttpStatus.BAD_REQUEST)
16      @ExceptionHandler(MethodArgumentNotValidException.class)
17      @ public List<CampoComErro> handleMethodArgumentNotValidException(MethodArgumentNotValidException exception) {
18          return exception.getBindingResult().getFieldErrors().stream()
19              .map(fieldError -> new CampoComErro(fieldError.getField(), fieldError.getDefaultMessage()))
20              .collect(Collectors.toList());
21      }
22
23      @ResponseStatus(HttpStatus.NOT_FOUND)
24      @ExceptionHandler(UsuarioNaoEncontradoException.class)
25      @ public ApenasMensagemErro handleUsuarioNaoEncontradoException(UsuarioNaoEncontradoException exception){
26          String Mensagem = exception.getMessage();
27          return new ApenasMensagemErro(Mensagem);
28      }
29  }
30

```

## Client

Para conseguir o valor do veículo é necessário fazer no total 4 requisições na API da Fipe por isso utilizaremos o Spring Feign Cloud, com poucas linhas de código podemos utilizar o serviço.

Primeiramente criei a Interface FipeClient com as seguintes anotações:

**@FeignClient** – Define a interface para utilização do Feign Cloud, passando como atributo o nome e a URL da API que irei consumir.

**@GetMapping**- Essa anotação identifica quando é feito uma requisição GET que atende ao end-point especificado.

```

9  @FeignClient(name = "client", url = "https://parallelum.com.br/fipe/api/v1/carros")
10  public interface FipeClient {
11
12      @GetMapping("/marcas")
13      List<BuscaMarcaResponse> buscarMarcas();
14
15      @GetMapping("/marcas/{idMarca}/modelos")
16      BuscaModeloResponseWrapper buscarModelos(@PathVariable String idMarca);
17
18      @GetMapping("/marcas/{idMarca}/modelos/{idModelo}/anos")
19      List<BuscaAnoResponse> buscarAnos(@PathVariable String idMarca,
20                                       @PathVariable String idModelo);
21
22      @GetMapping("/marcas/{idMarca}/modelos/{idModelo}/anos/{ano}")
23      BuscaValorResponse buscarValor(@PathVariable String idMarca,
24                                     @PathVariable String idModelo,
25                                     @PathVariable String ano);
26  }
27

```

E em seguida devemos colocar a anotação **@EnableFeignClients** na classe main, para habilitar clientes Feign na aplicação.

```

7  @SpringBootApplication
8  @EnableFeignClients
9  ▶ public class CadastroVeiculoApplication {
10
11  ▶     public static void main(String[] args) { SpringApplication.run(CadastroVeiculoApplication.class, args); }
14
15  }
16

```

## Controllers

É na classe controller que faço o mapeamento das requisições, para a aplicação criei duas controllers a “UsuarioController” e “VeiculoController”.

**@RestController** – Essa anotação indica que a classe vai ser uma controladora, retorna o objeto e os dados do objeto diretamente na resposta HTTP como JSON (ou XML).

**@RequestMapping** – Nessa anotação devemos passar uma URL assim quando essa URL for chamada a controladora entrará em ação.

**@Autowired** – É essa anotação que faz a famosa Injeção de Dependência, ou seja, faz a instancia da classe anotada.

**@PathVariable** – Anotação que indica que a variável é passada na URL.

**@PostMapping** – Essa anotação identifica quando é feito uma requisição POST que atende ao end-point especificado.

**@Valid** – Faz a validação dos dados de entrada usando as restrições colocadas nos atributos das classes form.

**@RequestBody** – Faz a serialização do corpo recebido e transforma em um objeto Java.

```

15  @RestController
16  @RequestMapping("/usuarios")
17  public class UsuarioController {
18
19      @Autowired
20      private UsuarioRepository usuarioRepository;
21
22      @GetMapping("/{idUsuario}")
23      public ResponseEntity listarCarros(@PathVariable Long idUsuario) {
24          Optional<Usuario> possivelUsuario = usuarioRepository.findById(idUsuario);
25
26          if (!possivelUsuario.isPresent()){
27              throw new UsuarioNaoEncontradoException(String
28                  .format("Não foi encontrado um usuario com id %d, por favor utilize um usuário cadastrado", idUsuario));
29          }
30
31          UsuarioComVeiculosDTO usuarioComVeiculosDTO = new UsuarioComVeiculosDTO(possivelUsuario.get());
32
33          return ResponseEntity.ok().body(usuarioComVeiculosDTO);
34      }
35

```

```

35     @PostMapping
36     @ public ResponseEntity salvar(UriComponentsBuilder uriComponentsBuilder,
37                                     @Valid @RequestBody UsuarioForm usuarioForm) {
38         boolean existeEmail = usuarioRepository.existsByEmail(usuarioForm.getEmail());
39
40         if (existeEmail) {
41             return ResponseEntity.badRequest()
42                 .body(new ApenasMensagemErro("Email já cadastrado, por favor insira um novo."));
43         }
44         boolean existeCPF = usuarioRepository.existsByCpf(usuarioForm.getCpf());
45
46         if (existeCPF) {
47             return ResponseEntity.badRequest()
48                 .body(new ApenasMensagemErro("CPF já cadastrado, por favor insira um novo."));
49         }
50
51         Usuario usuario = usuarioForm.transformarEmUsuario();
52         Usuario usuarioSalvo = usuarioRepository.save(usuario);
53
54         URI uriRecurso = uriComponentsBuilder
55             .path("/usuarios/{idUsuario}")
56             .build(usuarioSalvo.getIdUsuario());
57         return ResponseEntity.created(uriRecurso).body(new UsuarioDTO(usuarioSalvo));
58     }
59 }

```

```

16 @RestController
17 @RequestMapping("/usuarios/{idUsuario}/veiculos")
18 public class VeiculoController {
19
20     @Autowired
21     private VeiculoRepository veiculoRepository;
22     @Autowired
23     private FipeClient fipeClient;
24     @Autowired
25     private UsuarioRepository usuarioRepository;
26
27     @PostMapping
28     public ResponseEntity salvar(UriComponentsBuilder uriComponentsBuilder,
29                                 @PathVariable Long idUsuario,
30                                 @RequestBody VeiculoForm veiculoForm) {
31         boolean existeUsuario = usuarioRepository.existsById(idUsuario);
32
33         if (!existeUsuario) {
34             throw new UsuarioNaoEncontradoException(String
35                 .format("Não foi encontrado um usuário com id %d, por favor utilize um usuário cadastrado", idUsuario));
36         }
37
38         List<BuscaMarcaResponse> buscaMarcaResponse = fipeClient.buscarMarcas();
39         Optional<BuscaMarcaResponse> possivelMarca = buscaMarcaResponse.stream()
40             .filter(elemento -> elemento.getNome().equals(veiculoForm.getMarca()))
41             .findFirst();
42
43         if (possivelMarca.isEmpty()) {
44             return ResponseEntity.badRequest()
45                 .body(new ApenasMensagemErro("Não existe uma marca com esse nome"));
46         }
47
48         BuscaMarcaResponse marca = possivelMarca.get();

```

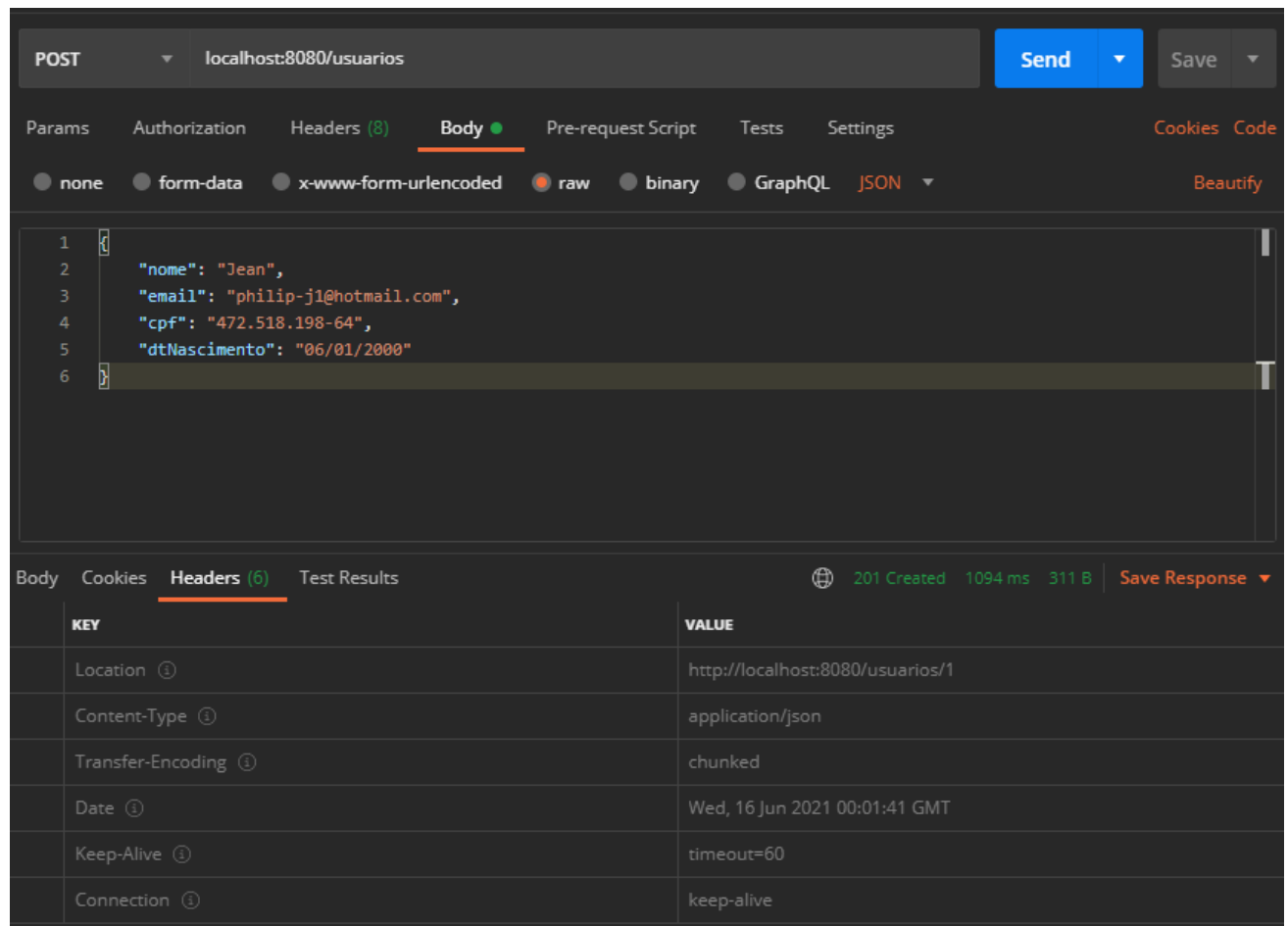
```

50 BuscaModeloResponseWrapper buscarModeloResponse = fipeclient.buscarModelos(marca.getCodigo());
51 Optional<String> possivelModelo = buscarModeloResponse.buscarCodigoDoModelo(veiculoForm.getModelo());
52
53 if (possivelModelo.isEmpty()) {
54     return ResponseEntity.badRequest().body(new ApenasMensagemErro(String
55         .format("Não existe um modelo com esse nome para a Marca: %s",
56             veiculoForm.getMarca())));
57 }
58
59 List<BuscaAnoResponse> buscaAnoResponse = fipeclient.buscarAnos(marca.getCodigo(), possivelModelo.get());
60 Optional<String> possivelAno = buscaAnoResponse.stream()
61     .filter(ano -> ano.getNome().equals(veiculoForm.getAno()))
62     .map(BuscaAnoResponse::getCodigo)
63     .findFirst();
64
65
66 if (possivelAno.isEmpty()) {
67     return ResponseEntity.badRequest().body(new ApenasMensagemErro(String
68         .format("Não existe um carro com esse ano para a Marca: %s e o Modelo: %s",
69             veiculoForm.getMarca(),
70             veiculoForm.getModelo())));
71 }
72
73 BuscaValorResponse carro = fipeclient.buscarValor(marca.getCodigo(), possivelModelo.get(), possivelAno.get());
74
75 Veiculo veiculo = veiculoForm.transformarEmVeiculo(carro.getValor(), idUsuario, carro.getAnoModelo());
76 Veiculo veiculoSalvo = veiculoRepository.save(veiculo);
77
78 URI uriRecurso = uriComponentsBuilder
79     .path("/{usuarios}/{idUsuario}/veiculos/{idVeiculo}")
80     .build(idUsuario, veiculoSalvo.getIdUsuario());
81 return ResponseEntity.created(uriRecurso).body(new VeiculoDTO(veiculoSalvo));
82 }
83 }

```

## Fazendo Requisições

### Sucesso – Cadastro de Usuário



POST localhost:8080/usuarios

Send Save

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

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **JSON** Beautify

```
1 {
2   "nome": "Jean",
3   "email": "philip-j1@hotmail.com",
4   "cpf": "472.518.198-64",
5   "dtNascimento": "06/01/2000"
6 }
```

Body Cookies **Headers (6)** Test Results

201 Created 1094 ms 311 B Save Response

KEY	VALUE
Location ⓘ	http://localhost:8080/usuarios/1
Content-Type ⓘ	application/json
Transfer-Encoding ⓘ	chunked
Date ⓘ	Wed, 16 Jun 2021 00:01:41 GMT
Keep-Alive ⓘ	timeout=60
Connection ⓘ	keep-alive

## Falha – Cpf Inválido

POST localhost:8080/usuarios

Send Save

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

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

```
1 {
2   "nome": "Jean",
3   "email": "philip-j1@hotmail.com",
4   "cpf": "472.518.198-64",
5   "dtNascimento": "06/01/2000"
6 }
```

Body Cookies Headers (4) Test Results 400 Bad Request 59 ms 244 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "campo": "cpf",
3   "erro": "número do registro de contribuinte individual brasileiro (CPF) inválido"
4 }
5
6
```

## Falha – Email Inválido

POST localhost:8080/usuarios

Send Save

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

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

```
1 {
2   "nome": "Jean",
3   "email": "phihotmail.com",
4   "cpf": "472.518.198-64",
5   "dtNascimento": "06/01/2000"
6 }
```

Body Cookies Headers (4) Test Results 400 Bad Request 30 ms 216 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "campo": "email",
3   "erro": "deve ser um endereço de e-mail bem formado"
4 }
5
6
```



## Falha – Campos Inválidos

POST localhost:8080/usuarios

Send Save

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

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

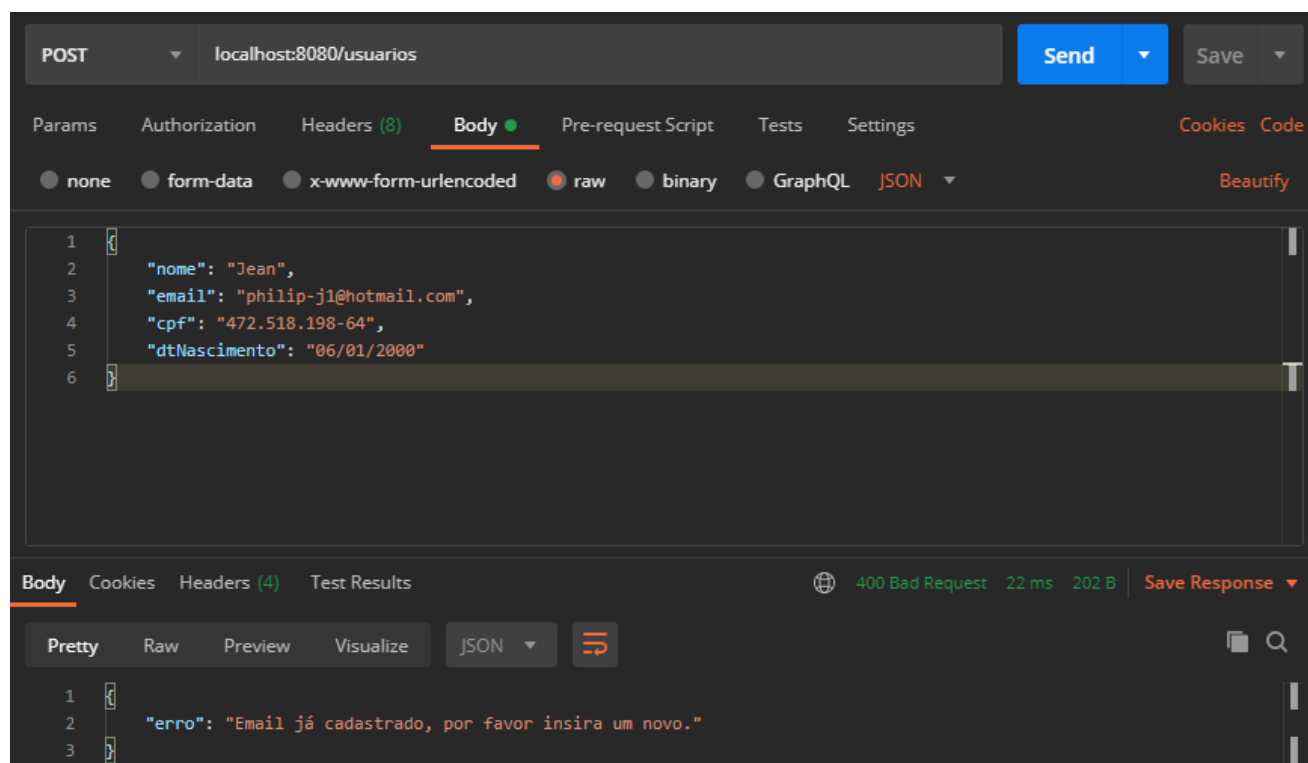
```
1 {
2   "nome": "",
3   "email": "",
4   "cpf": "",
5   "dtNascimento": ""
6 }
```

Body Cookies Headers (4) Test Results 400 Bad Request 17 ms 400 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "campo": "cpf",
4     "erro": "número do registro de contribuinte individual brasileiro (CPF) inválido"
5   },
6   {
7     "campo": "cpf",
8     "erro": "não deve estar em branco"
9   },
10  {
11    "campo": "email",
12    "erro": "não deve estar em branco"
13  }
14 }
```

## Falha – Email já cadastrado



POST localhost:8080/usuarios

Send Save

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

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

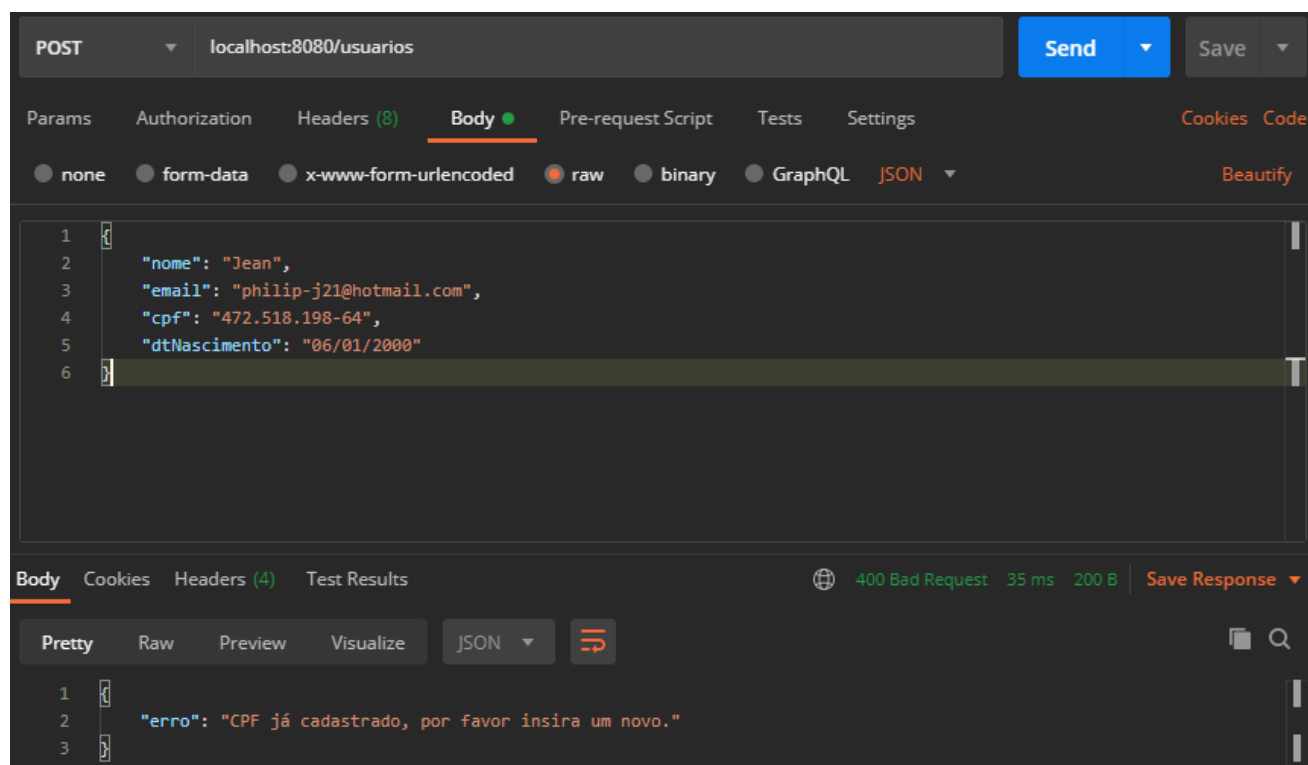
```
1 {
2   "nome": "Jean",
3   "email": "philip-j1@hotmail.com",
4   "cpf": "472.518.198-64",
5   "dtNascimento": "06/01/2000"
6 }
```

Body Cookies Headers (4) Test Results 400 Bad Request 22 ms 202 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "erro": "Email já cadastrado, por favor insira um novo."
3 }
```

## Falha – CPF já cadastrado



POST localhost:8080/usuarios

Send Save

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

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

```
1 {
2   "nome": "Jean",
3   "email": "philip-j21@hotmail.com",
4   "cpf": "472.518.198-64",
5   "dtNascimento": "06/01/2000"
6 }
```

Body Cookies Headers (4) Test Results 400 Bad Request 35 ms 200 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "erro": "CPF já cadastrado, por favor insira um novo."
3 }
```

## Sucesso – Cadastro de Veículo

The screenshot shows a Postman interface for a successful POST request. The URL is `localhost:8080/usuarios/1/veiculos`. The response status is 201 Created, with a time of 2.27 s and a size of 361 B. The Headers tab is selected, showing the following headers:

KEY	VALUE	DESCRIPTION
Location	<code>http://localhost:8080/usuarios/1/veiculos/1</code>	
Content-Type	<code>application/json</code>	
Transfer-Encoding	<code>chunked</code>	
Date	<code>Wed, 16 Jun 2021 00:08:21 GMT</code>	
Keep-Alive	<code>timeout=60</code>	
Connection	<code>keep-alive</code>	

## Falha – Usuário não cadastrado

The screenshot shows a Postman interface for a failed POST request. The URL is `localhost:8080/usuarios/1/veiculos`. The response status is 404 Not Found, with a time of 1057 ms and a size of 263 B. The Body tab is selected, showing the following JSON response:

```
{
  "erro": "Não foi encontrado um usuario com id 1, por favor utilize um usuário cadastrado"
}
```

## Falha – Marca inexistente

POST localhost:8080/usuarios/1/veiculos Send Save

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

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

```
1 {
2   "marca": "VW 1- Volkswagen",
3   "modelo": "AMAROK High.CD 2.0 16V TDI 4x4 Dies. Aut",
4   "ano": "2014 Diesel"
5 }
```

Body Cookies Headers (4) Test Results 400 Bad Request 605 ms 190 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "erro": "Não existe uma marca com esse nome"
3 }
```

## Falha – Modelo Inexistente

POST localhost:8080/usuarios/1/veiculos Send Save

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

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

```
1 {
2   "marca": "VW - Volkswagen",
3   "modelo": "1AMAROK High.CD 2.0 16V TDI 4x4 Dies. Aut",
4   "ano": "2014 Diesel"
5 }
```

Body Cookies Headers (4) Test Results 400 Bad Request 994 ms 220 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "erro": "Não existe um modelo com esse nome para a Marca: VW - Volkswagen"
3 }
```

## Falha – Veículo inexistente

POST localhost:8080/usuarios/1/veiculos

Send Save

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

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

```
1 {
2   "marca": "VW - Volkswagen",
3   "modelo": "AMAROK High.CD 2.0 16V TDI 4x4 Dies. Aut",
4   "ano": "20142 Diesel"
5 }
```

Body Cookies Headers (4) Test Results 400 Bad Request 1123 ms 271 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "erro": "Não existe um carro com esse ano para a Marca: VW - Volkswagen e o Modelo: AMAROK High.CD 2.0 16V TDI 4x4 Dies. Aut"
3 }
```

## Sucesso – Listagem de Veículos (“Hoje” sendo Segunda-Feira)

GET localhost:8080/usuarios/1/

Send Save

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

Query Params

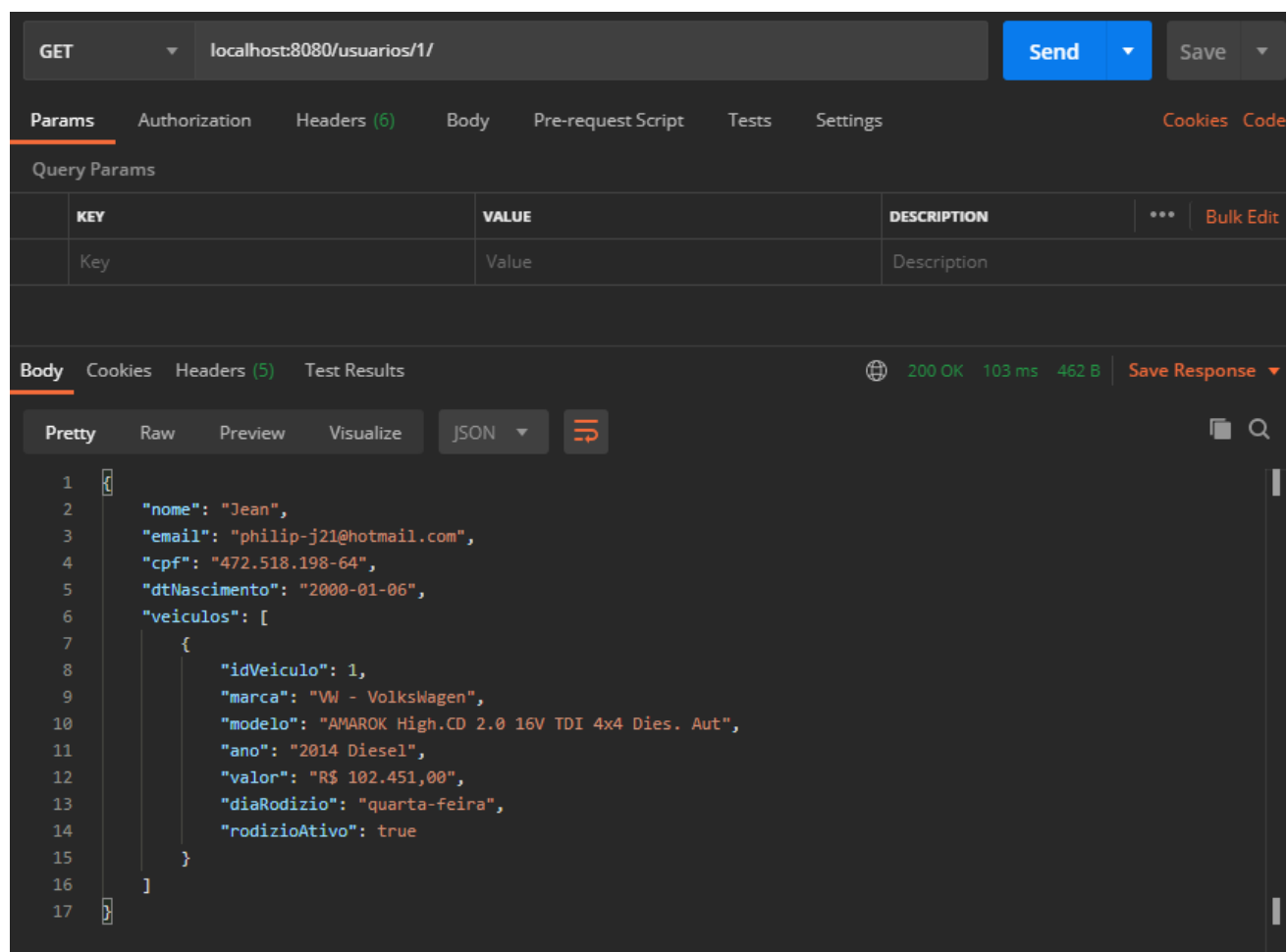
KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results 200 OK 215 ms 462 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "nome": "Jean",
3   "email": "philip-j1@hotmail.com",
4   "cpf": "472.518.198-64",
5   "dtNascimento": "2000-01-06",
6   "veiculos": [
7     {
8       "idVeiculo": 1,
9       "marca": "VW - Volkswagen",
10      "modelo": "AMAROK High.CD 2.0 16V TDI 4x4 Dies. Aut",
11      "ano": "2014 Diesel",
12      "valor": "R$ 102.451,00",
13      "diaRodizio": "quarta-feira",
14      "rodizioAtivo": false
15    }
16  ]
17 }
```

## Sucesso – Listagem de Veículos (“Hoje” sendo Quarta-Feira)



GET localhost:8080/usuarios/1/ Send Save

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

Query Params

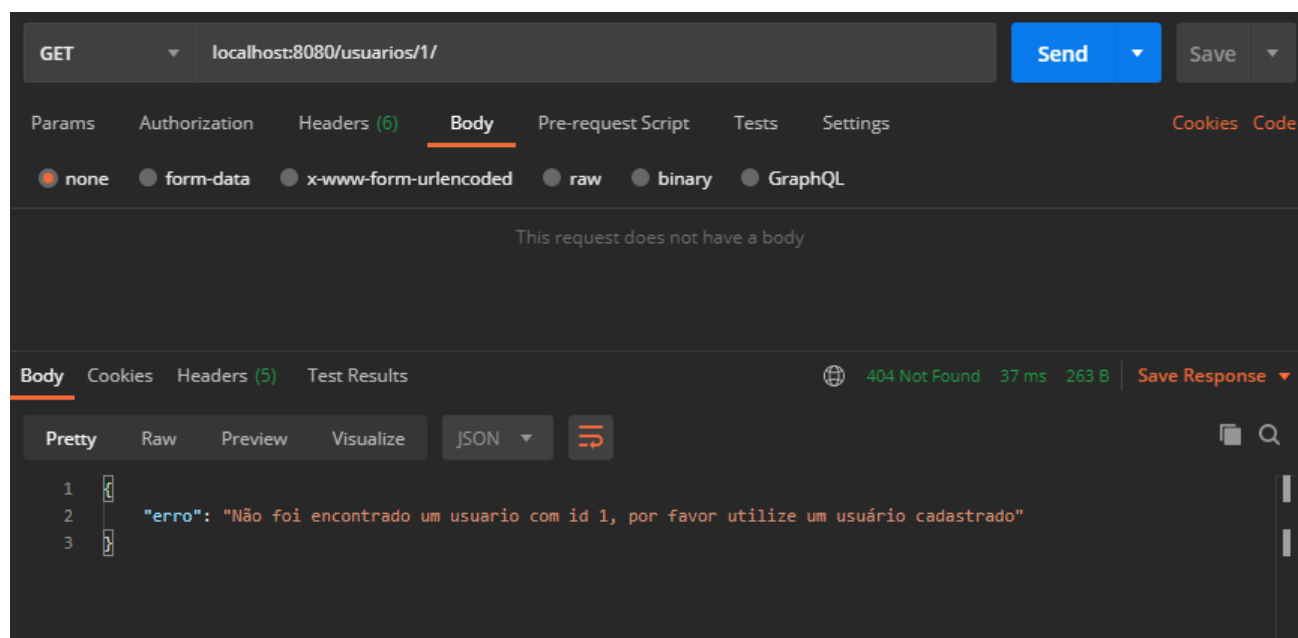
KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results 200 OK 103 ms 462 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "nome": "Jean",
3   "email": "philip-j21@hotmail.com",
4   "cpf": "472.518.198-64",
5   "dtNascimento": "2000-01-06",
6   "veiculos": [
7     {
8       "idVeiculo": 1,
9       "marca": "VW - Volkswagen",
10      "modelo": "AMAROK High.CD 2.0 16V TDI 4x4 Dies. Aut",
11      "ano": "2014 Diesel",
12      "valor": "R$ 102.451,00",
13      "diaRodizio": "quarta-feira",
14      "rodizioAtivo": true
15    }
16  ]
17 }
```

## Falha – Usuário não cadastrado



GET localhost:8080/usuarios/1/ Send Save

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

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

This request does not have a body

Body Cookies Headers (5) Test Results 404 Not Found 37 ms 263 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "erro": "Não foi encontrado um usuario com id 1, por favor utilize um usuário cadastrado"
3 }
```

## Considerações Finais

Na criação dessa API coloquei uma fração do que é programar utilizando todas essas tecnologias, o mundo da programação vem crescendo cada vez mais, tecnologias novas são criadas por isso é de grande importância saber fazer bom uso delas, acredito que com esse post abordei os principais meios de criação de uma API Rest de forma fácil, simples e sem dores de cabeça.

A API pode ser acessado em <https://github.com/jeanps22/api-cadastro-de-veiculos>