Escrito por Pablo Bizarro

Em 03/01/2021

Olá, tudo bem?

Hoje iremos aprender um pouco mais sobre spring boot juntos e na prática!

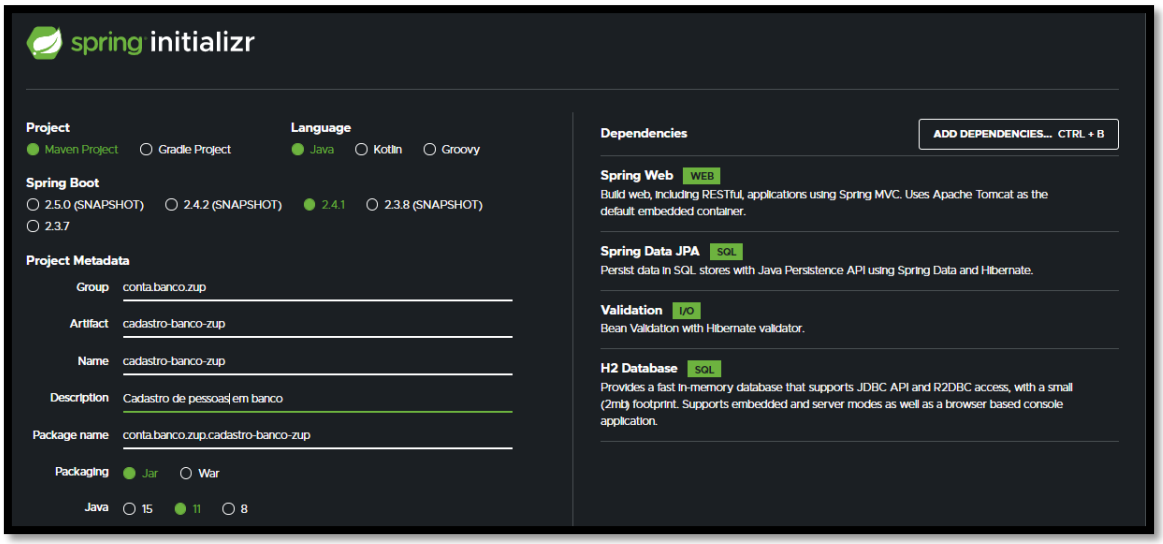
Faremos uma API REST que suportará processos de abertura de novas contas em banco. Utilizaremos como informações obrigatórias os seguintes atributos:

Nome
- E-mail
- CPF
- Data de Nascimento

Após checarmos a integridade desses dados devemos gravá-los no banco de dados e retornarmos o status adequado. Vamos começar?

Criação do projeto utilizando o spring initializr

A parte inicial de um projeto (configurar estrutura de pastas, incluir dependências etc.) pode ser um pouco moroso de executar manualmente quando o desenvolvedor já quer botar a mão na massa e começar a codar. Para facilitar a nossa vida foi criado o Spring Initializr (https://start.spring.io/), uma aplicação web que consegue gerar a estrutura de um projeto Spring Boot toda para você com a inserção de apenas algumas informações chave.



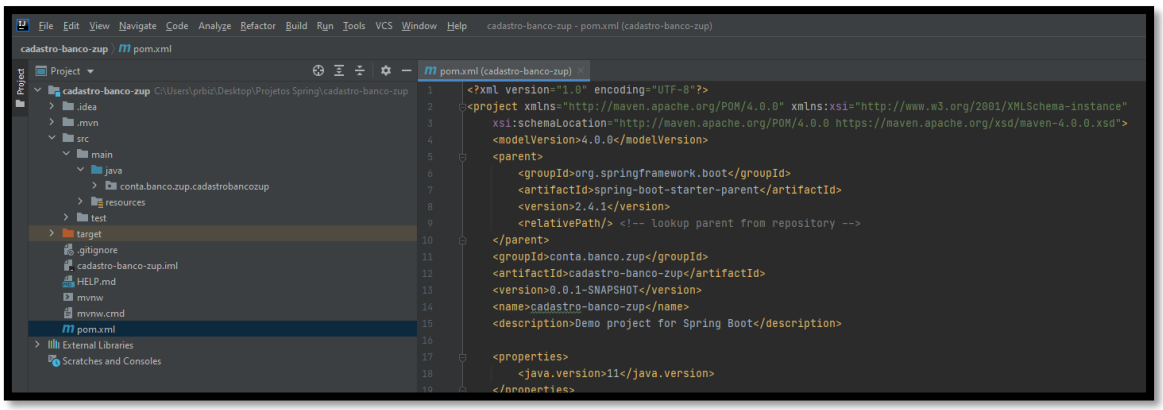
Dados inseridos para a criação do nosso projeto

Nesse caso utilizaremos as configurações default do spring initializr, build: Maven; Linguagem: Java 11 e Spring boot 2.4.1, é necessário também adicionar as dependências das tecnologias que usaremos, essas serão:

- “Spring Web” para construirmos a API RESTful
- “Spring Data JPA” para que possamos utilizar o Hibernate e Spring Data nas interações com o banco de dados

- “Validation” para utilizarmos o Hibernate nas validações dos atributos recebidos pela API
- “H2 Database” para suportar o banco de dados H2 que será implementado

Após, gerar a estrutura, abra o projeto com sua IDE de preferência, no meu caso utilizarei no resto do projeto a IntelliJ Community Edition.



Arquivo pom contendo as dependências inseridas no initializr

Criação da classe “Pessoa”

Essa será nossa principal entidade nessa API de cadastro.

Utilizaremos a anotação @Entity para indicar que essa classe também é uma entidade e que o JPA deverá estabelecer uma ligação dessa classe com uma tabela no banco de dados.

```
@Entity
public class Pessoa {
```

No atributo id utilizaremos a anotação @Id para indicar que essa será a chave primária dessa entidade no banco de dados e a @GeneratedValue que fará com que o valor desse atributo seja gerado e gerenciado pela própria aplicação.

```
@Id
@GeneratedValue
private Long id;
```

Nos demais atributos utilizamos a anotação @Column para especificar alguns detalhes sobre essas colunas no banco, em todas indicaremos que “nullable=false” para que não sejam aceitas entradas em branco. Nas colunas email e cpf também adicionaremos “unique=true” para que não sejam aceitos dados duplicados.

```java
@Column (nullable = false, unique = true)
private String cpf;

@Column(nullable = false)
private String nome;

@Column(nullable = false, unique = true)
private String email;

@Column(nullable = false)
private String dataNascimento;
```

Após essas configurações criamos os getters e setters para os atributos da classe.

```java
package conta.banco.zup.cadastrobancozup;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Pessoa {

    @Id
    @GeneratedValue
    private Long id;

    @Column (nullable = false, unique = true)
    private String cpf;

    @Column(nullable = false)
    private String nome;

    @Column(nullable = false, unique = true)
    private String email;

    @Column(nullable = false)
    private String dataNascimento;

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getCpf() { return cpf; }
    public void setCpf(String cpf) { this.cpf = cpf; }
    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public String getDataNascimento() { return dataNascimento; }
    public void setDataNascimento(String dataNascimento) { this.dataNascimento = dataNascimento; }
}
```

Visão geral da classe “Pessoa”

Criação do repositório “PessoaRepository”

Agora criaremos a interface “PessoaRepository” herdando a interface JPARepository que já nos traz todos os métodos básicos que precisamos para fazer um CRUD. Devemos incluir no extends

a classe do repositório e o tipo da chave primária, no nosso caso passaremos <Pessoa, Long> pois nossa chave primária da entidade Pessoa é a Id, lembra?

```java
package conta.banco.zup.cadastrobancozup;

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

public interface PessoaRepository extends JpaRepository<Pessoa, Long> {
}
```

Criação da classe “PessoaRequest”

Dando prosseguimento, construiremos a classe “PessoaRequest” para enviar as requisições de criação de pessoas ao back-end. Os atributos são os mesmos definidos na entidade Pessoa anteriormente (menos o Id), aqui utilizei a anotação @NotEmpty para que não sejam aceitas entradas vazias e defini mensagens específicas para cada atributo. Adicionei a anotação @Size no campo cpf, limitando a entrada a 11 caracteres.

```java
package conta.banco.zup.cadastrobancozup;

import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.Size;

public class PessoaRequest {

    @NotEmpty(message = "O campo cpf nao pode estar vazio")
    @Size(max = 11, message = "O campo cpf deve ter no maximo 11 caracteres")
    private String cpf;

    @NotEmpty(message = "O campo nome nao pode estar vazio")
    private String nome;

    @NotEmpty(message = "O campo email nao pode estar vazio")
    private String email;

    @NotEmpty(message = "O campo dataNascimento nao pode estar vazio")
    private String dataNascimento;

    public String getCpf() { return cpf; }

    public void setCpf(String cpf) { this.cpf = cpf; }

    public String getNome() { return nome; }

    public void setNome(String nome) { this.nome = nome; }

    public String getEmail() { return email; }

    public void setEmail(String email) { this.email = email; }

    public String getDataNascimento() { return dataNascimento; }

    public void setDataNascimento(String dataNascimento) { this.dataNascimento = dataNascimento; }
}
```

Visão geral da classe “PessoaRequest”

Criação do serviço “PessoaService”

Hora de criarmos o serviço que guardará os dados recebidos via requisição POST no nosso repositório de pessoas. Criaremos a classe pessoaService e adicionaremos a anotação @Service para definir que esta pertence à camada de serviço e que é gerenciável pelo Spring.

```
@Service
public class PessoaService {
```

Aqui também instanciaremos um repositório pessoaRepository da classe homônima que construímos anteriormente. A anotação @Autowired serve para indicar que aqui deverá ser realizada a injeção de dependência do Spring.

```
@Autowired
private PessoaRepository pessoaRepository;
```

Após isso, escreveremos o método “createNewPessoa” que irá receber um objeto “pessoaRequest” e o adicionará ao Repository. Note que instanciamos uma nova Pessoa e passamos todos os atributos via sets. Ao final utilizamos try e catch para tratar exceções DataIntegrityViolationException que no caso do nosso projeto são as de CPF e Email já existentes em outro cadastro. Com o retorno desse serviço definiremos no Controller a resposta a ser enviada no HTTP.

```
@Service
public class PessoaService {

    @Autowired
    private PessoaRepository pessoaRepository;

    public String createNewPessoa(PessoaRequest pessoaRequest) {

        Pessoa pessoa = new Pessoa();
        pessoa.setNome(pessoaRequest.getNome());
        pessoa.setCpf(pessoaRequest.getCpf());
        pessoa.setEmail(pessoaRequest.getEmail());
        pessoa.setDataNascimento(pessoaRequest.getDataNascimento());

        try {
            pessoa = pessoaRepository.save(pessoa);
        } catch (DataIntegrityViolationException e) {
            return "duplicado";
        }

        return "sucesso";
    }
}
```

Visão geral do serviço “PessoaService”

Criação do controller “PessoaController”

Agora criaremos o rest controller que irá tratar as chamadas para o endpoint da API que estamos desenvolvendo, se chamará “PessoaController”.

Primeiramente criamos a classe “PessoaController” com a anotação @RestController (utilizada para simplificar a criação de web services Restful controlando requisições e respostas) e definimos na anotação @RequestMapping o caminho para a chamada desse método (no caso utilizaremos “/api/pessoas”). Logo após instanciamos o serviço pessoaService que será utilizado mais para frente.

```
@RestController
@RequestMapping("/api/pessoas")
public class PessoaController {

    @Autowired
    private PessoaService pessoaService;
```

Agora escreveremos o método que irá tratar as requisições POST que recebermos, utilizamos a anotação @PostMapping para que essas requisições sejam direcionadas ao chegar no endpoint. Utilizaremos aqui o tipo genérico ResponseEntity do spring e a anotação @RequestBody para conseguirmos receber os atributos que vem no body do POST). Destacamos também a anotação @Valid que fará as validações previamente setadas na classe PessoaRequest como @Size e @NotEmpty.

```
@PostMapping
public ResponseEntity createNewPessoa(@Valid @RequestBody PessoaRequest pessoaRequest) {
```

Nesse método também chamamos o serviço pessoaService para que os dados cadastrais recebidos sejam salvos em uma entidade Pessoa. A resposta da chamada desse serviço é guardada na variável mensagem e esse retorno definirá qual a resposta HTTP será evniada. No caso de sucesso retornará 201 e no caso de falha retornará 422 Unprocessable Entity e uma mensagem identificando o erro.

```java
@RestController
@RequestMapping("/api/pessoas")
public class PessoaController {

    @Autowired
    private PessoaService pessoaService;

    @PostMapping
    public ResponseEntity consomeHttp(@Valid @RequestBody PessoaRequest pessoaRequest) {

        String mensagem = pessoaService.createNewPessoa(pessoaRequest);

        if(mensagem == "sucesso"){
            return new ResponseEntity<Void>(HttpStatus.CREATED);
        } else {
            return ResponseEntity.unprocessableEntity().body( t "Ja existe um cadastro com o mesmo CPF ou EMAIL");
        }

    }


}
```

Visão geral do controller ”PessoaController”

Hora do teste!

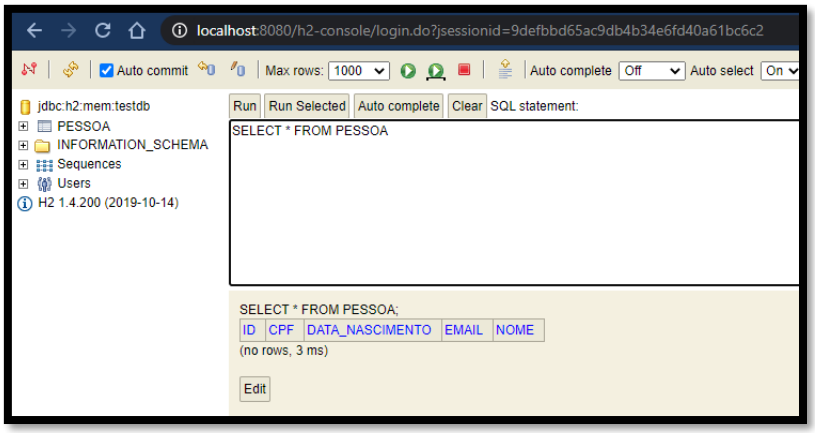
Para que possamos visualizar nosso banco de dados H2 via interface web devemos ir no caminho resources > application.properties e adicionar as seguintes configurações:

```
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:testdb
spring.data.jpa.repositories.bootstrap-mode=default
spring.jpa.hibernate.ddl-auto=create-drop
```

Agora basta salvar o projeto, ir via terminal na pasta do mesmo e executar o comando que iniciará a API REST:

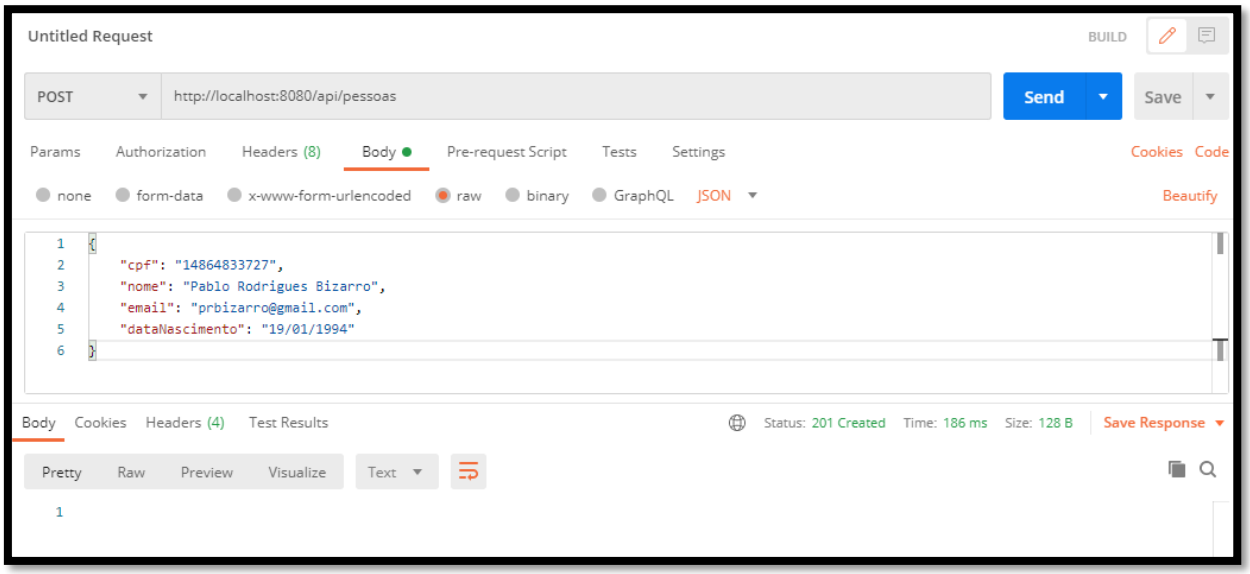
```
03/01/2021  11:52    <DIR>          .
03/01/2021  11:52    <DIR>          ..
02/01/2021  21:22               395 .gitignore
03/01/2021  18:46    <DIR>          .idea
02/01/2021  21:22    <DIR>          .mvn
03/01/2021  11:09            10.628 cadastro-banco-zup.iml
02/01/2021  21:22             1.428 HELP.md
02/01/2021  21:22            10.070 mvnw
02/01/2021  21:22             6.608 mvnw.cmd
03/01/2021  11:03             1.858 pom.xml
02/01/2021  21:22    <DIR>          src
03/01/2021  11:52    <DIR>          target
               6 arquivo(s)         30.987 bytes
               6 pasta(s)   157.676.019.712 bytes disponíveis

C:\Users\prbiz\Desktop\Projetos Spring\cadastro-banco-zup>mvn spring-boot:run
```

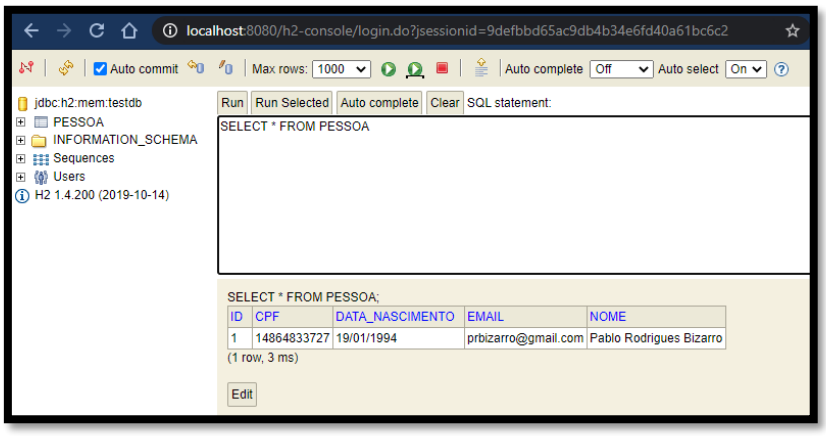
Para abrir a interface do banco de dados H2 da aplicação digitamos no navegador o caminho localhost:8080/h2-console/ e entramos com o usuário padrão “sa”. Aqui já podemos ver a tabela feita para nossa classe Pessoa apenas esperando os cadastros!



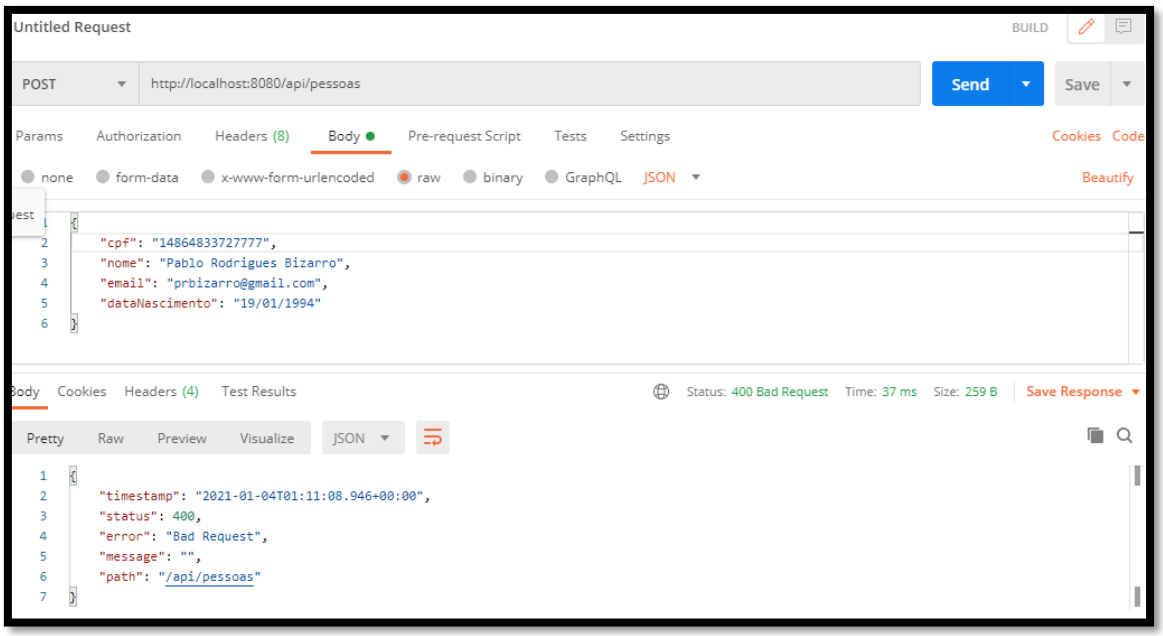
Agora, com a ajuda do software Postman faremos algumas requisições HTTP POST enviando JSONs para testarmos as funcionalidades que programamos ao longo do projeto.

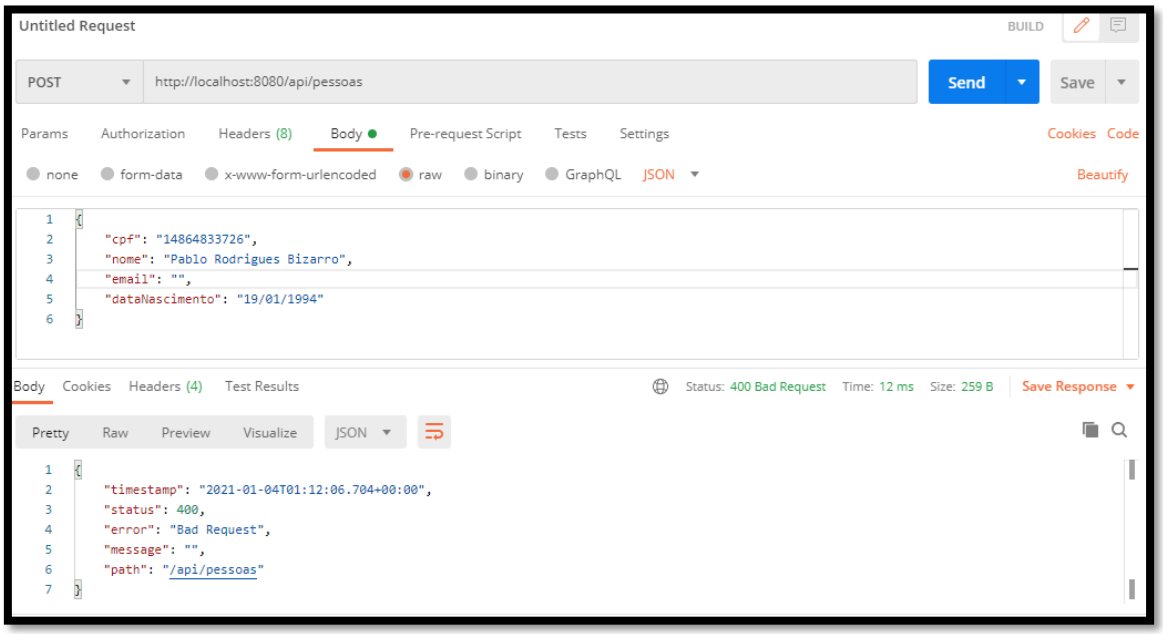
- **Criação com sucesso**



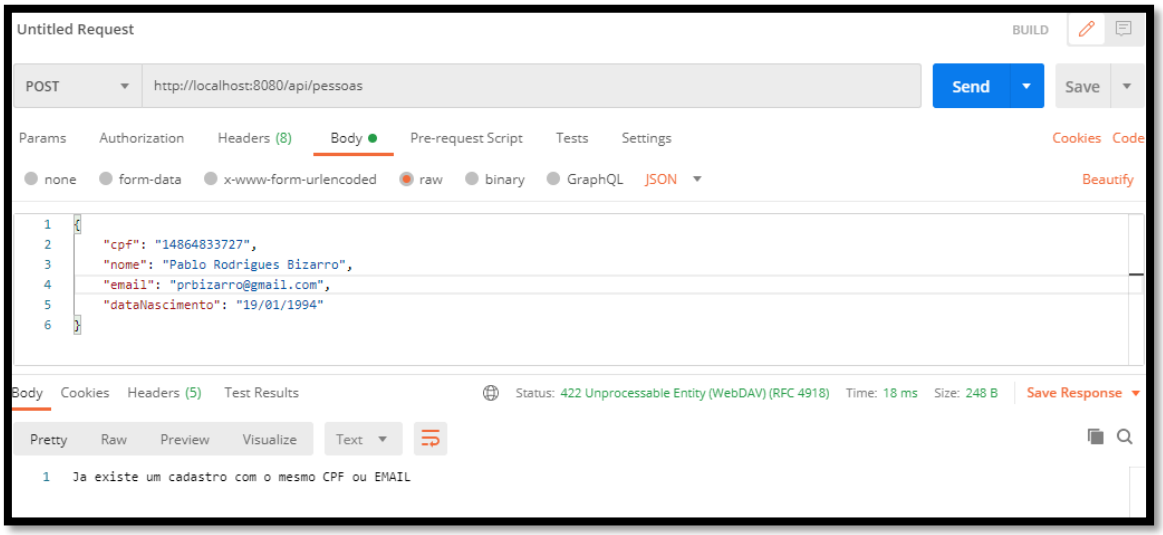


- **Criação com dados inválidos (checados pela anotação @Valid)**





- **Criação com dados duplicados (checados pela anotação @Column no BD)**



Obrigado por ler até aqui!

Mesmo tendo pouco contato anterior com o Spring acho que aprendi bastante realizando esse desafio 😊

Feliz 2021 e abraços,

Pablo Bizarro

prbizarro@gmail.com