

## Aula 6 - Criação de API Rest

Tempo estimado para esta prática: 25 min

IDE utilizada: Eclipse JavaEE

Banco de Dados: Mysql Workbench

Ferramenta para teste: Postman

Nessa aula iremos fazer um **Controller de Cadastro**.

1. Criar um projeto no inicializr

<https://start.spring.io/>

com as configurações a seguir:



<b>Project</b> <input type="radio"/> Gradle - Groovy <input type="radio"/> Gradle - Kotlin <input checked="" type="radio"/> <b>Maven</b>	<b>Language</b> <input checked="" type="radio"/> <b>Java</b> <input type="radio"/> Kotlin <input type="radio"/> Groovy
<b>Spring Boot</b> <input type="radio"/> 3.3.1 (SNAPSHOT) <input checked="" type="radio"/> <b>3.3.0</b> <input type="radio"/> 3.2.7 (SNAPSHOT) <input type="radio"/> 3.2.6	
<b>Project Metadata</b>	
Group	<input type="text" value="com.example"/>
Artifact	<input type="text" value="demorest"/>
Name	<input type="text" value="demorest"/>
Description	<input type="text" value="Demo project for Spring Boot"/>
Package name	<input type="text" value="com.example.demorest"/>
Packaging	<input checked="" type="radio"/> <b>Jar</b> <input type="radio"/> War

**Dependencies** ADD DEPENDENCIES... CTRL + B

**Spring Boot DevTools** DEVELOPER TOOLS  
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

**Spring Web** WEB  
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**MySQL Driver** SQL  
MySQL JDBC driver.

**Spring Data JPA** SQL  
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

2. criar 3 classes (já estão disponíveis no material):

- ContatoController
- Contato
- ContatoRepository

Nessa prática conheceremos também o Lombok. Para saber um pouco mais sobre o Lombok, pode acessar o site oficial do projeto: <https://projectlombok.org/>

3. Temos que colocar então uma dependência no pom:

<dependency>

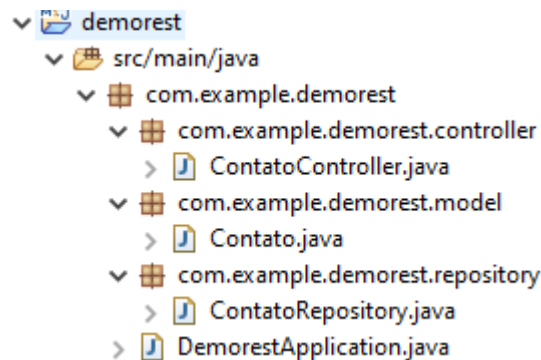
<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

A seguir temos a estrutura de pastas do nosso projeto



```
▼ demorest
  ▼ src/main/java
    ▼ com.example.demorest
      ▼ com.example.demorest.controller
        > ContatoController.java
      ▼ com.example.demorest.model
        > Contato.java
      ▼ com.example.demorest.repository
        > ContatoRepository.java
        > DemorestApplication.java
```

Em seguida temos as classes do nosso projeto

ContatoController.java

```
package com.example.demorest.controller;
import java.util.List;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.example.demorest.model.Contato;
import com.example.demorest.repository.ContatoRepository;

@RestController
@RequestMapping({ "/contatos" })
public class ContatoController {
    private ContatoRepository repository;

    ContatoController(ContatoRepository contatoRepository) {
        this.repository = contatoRepository;
    }
    @GetMapping
    public List<?> findAll() {
        return repository.findAll();
    }
    @GetMapping(path = {("/{id}" })
    public ResponseEntity<?> findById(@PathVariable long id) {
```

```

        return repository.findById(id).map(record ->
ResponseEntity.ok().body(record))
            .orElse(ResponseEntity.notFound().build());
    }
    @PostMapping
    public Contato create(@RequestBody Contato contato) {
        return repository.save(contato);
    }
    @PutMapping(value =("/{id}")
    public ResponseEntity<?> update(@PathVariable("id") long id, @RequestBody
Contato contato) {
        return repository.findById(id).map(record -> {
            record.setName(contato.getName());
            record.setEmail(contato.getEmail());
            record.setFone(contato.getFone());
            Contato updated = repository.save(record);
            return ResponseEntity.ok().body(updated);
        }).orElse(ResponseEntity.notFound().build());
    }
    @DeleteMapping(path = {("/{id}" })
    public ResponseEntity<?> delete(@PathVariable long id) {
        return repository.findById(id).map(record -> {
            repository.deleteById(id);
            return ResponseEntity.ok().build();
        }).orElse(ResponseEntity.notFound().build());
    }
}

```

## Contato.java

```

package com.exemplo.demorest.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@AllArgsConstructor
@NoArgsConstructor
@Data
@Entity
public class Contato {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)

```

```
    private Long id;

    private String nome;
    private String email;
    private String fone;
}
```

#### ContatoRepository.java

```
package com.example.demorest.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.example.demorest.model.Contato;
@Repository
public interface ContatoRepository extends JpaRepository <Contato, Long>{
}
```

Nosso application.properties :

spring.application.name=demorest

#DATASOURCE

spring.datasource.url=  
jdbc:mysql://localhost:3306/bd\_rest?createDatabaseIfNotExist=true

spring.datasource.username= root

spring.datasource.password= root

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

#JPA

spring.jpa.hibernate.ddl-auto= update

spring.jpa.show-sql= true

spring.jpa.open-in-view= true

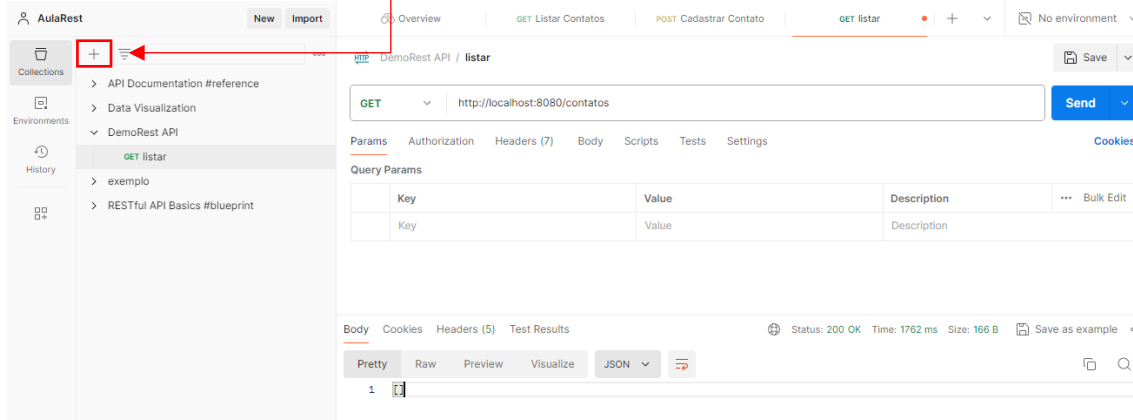
#### 4. Compilar a aplicação

### Testes e documentação com postman

- Entrar em <https://www.postman.com/>
- Criar uma conta.

## 1. Criar collection

Para criar uma Collection, clicar no +

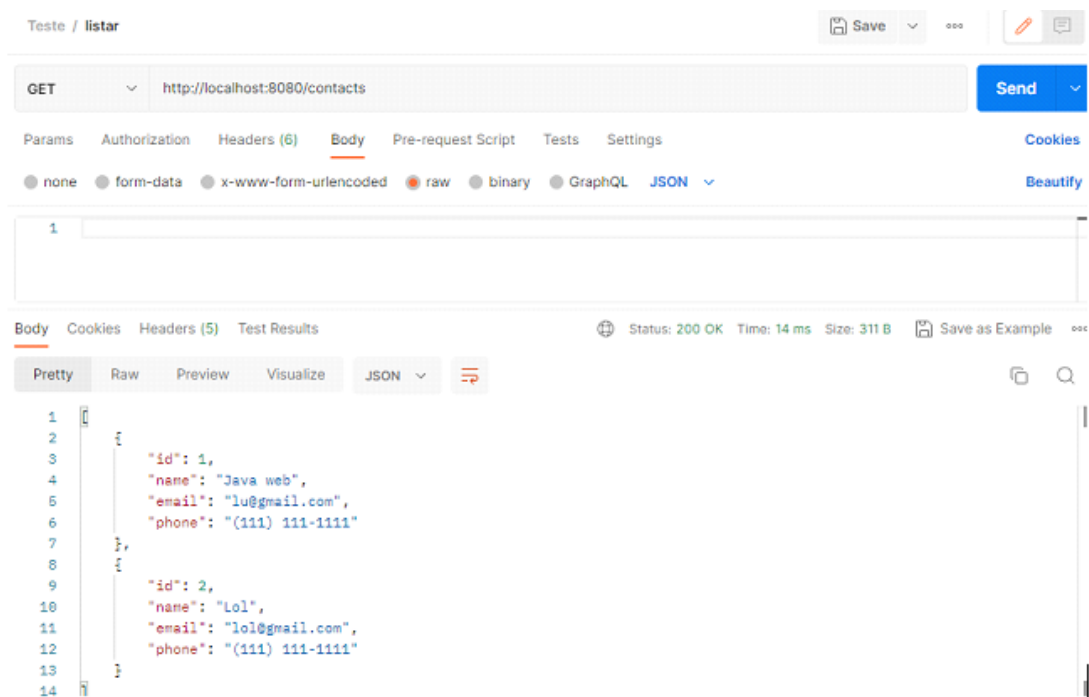


## 2. Criar Listar

GET

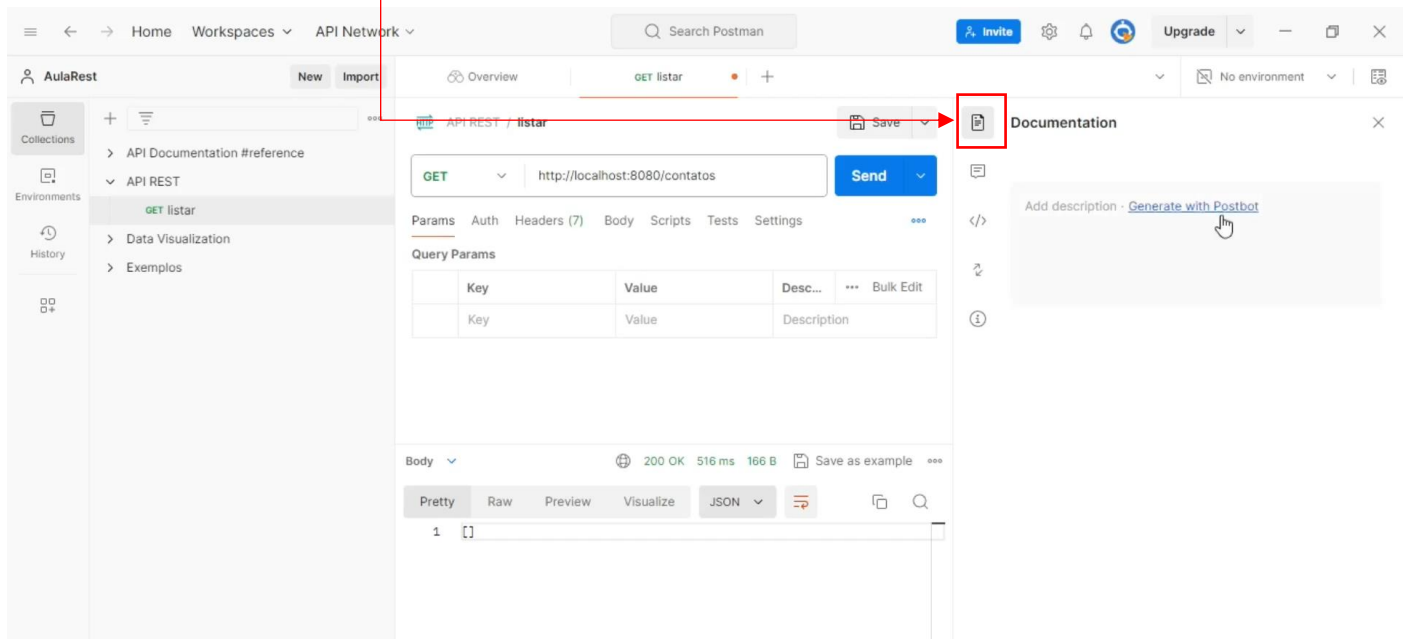
Obs: se não tiver nada no banco, não lista nada!

<http://localhost:8080/contacts>



Vamos agora já documentar.

Clicar em documentation



### 3. Cadastrar

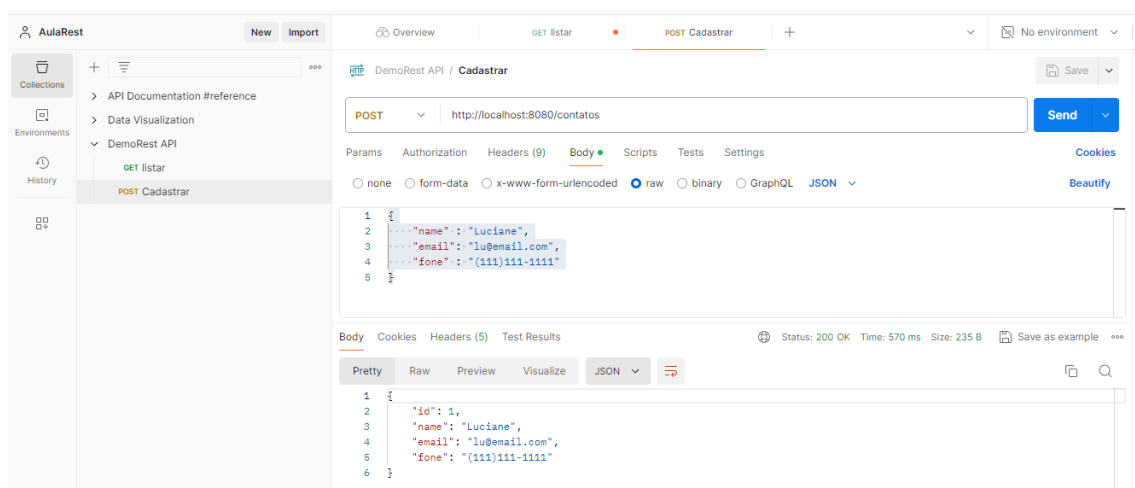
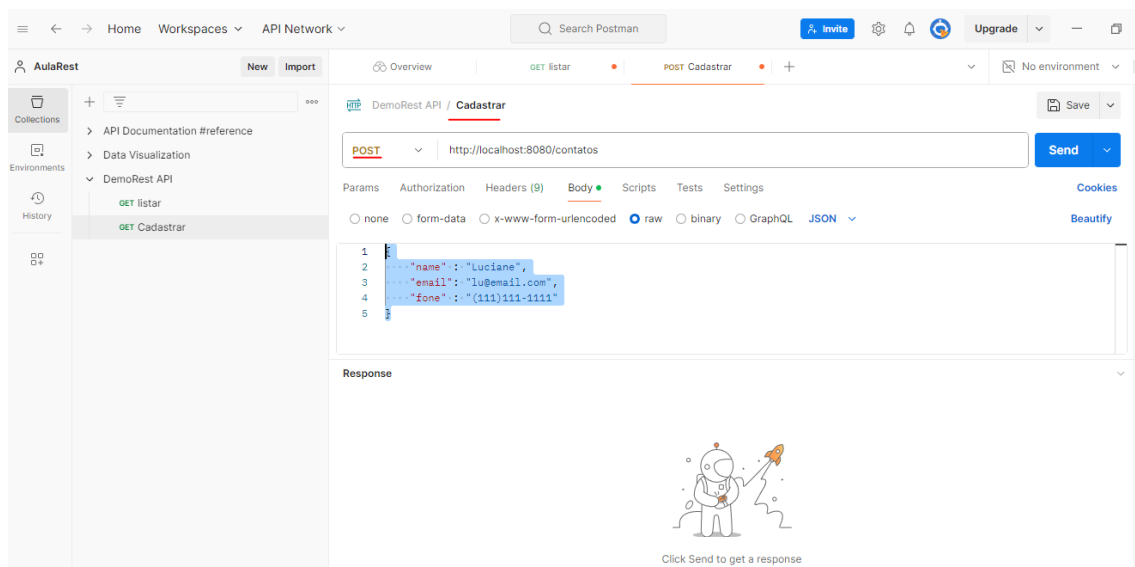
Botão direito do mouse em cima do Get listar> duplicate

POST

http://localhost:8080/contacts

```
{  
  "name" : "Luciane",  
  "email": "lu@email.com",  
  "fone" : "(111)111-1111"  
}
```

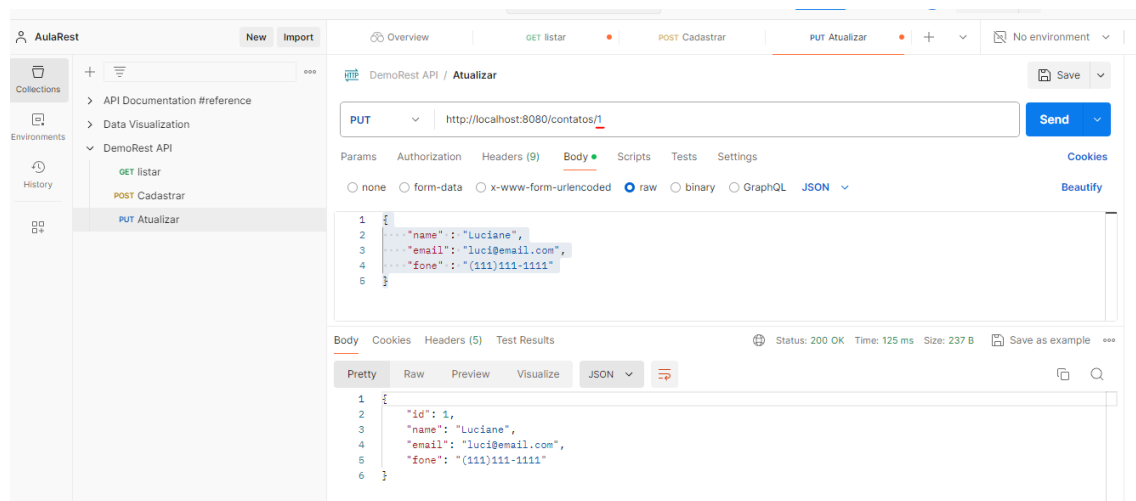
Para o POST configurar como abaixo:



## 4. Atualizar

PUT

http://localhost:8080/contacts/1



## 5. Deletar

### DELETE

http://localhost:8080/contacts/1

The screenshot displays the Postman API client interface. The top navigation bar includes a search bar, an 'Invite' button, and an 'Upgrade' button. The left sidebar shows a collection named 'AulaRest' with a sub-collection 'DemoRest API' containing several endpoints: 'GET listar', 'POST Cadastrar', 'PUT Atualizar', and 'DEL Deletar'. The main panel shows the configuration for the 'DEL Deletar' endpoint. The request method is set to 'DELETE' and the URL is 'http://localhost:8080/contatos/1'. Below the URL bar, there are tabs for 'Params', 'Authorization', 'Headers (9)', 'Body', 'Scripts', 'Tests', and 'Settings'. The 'Params' tab is active, showing a table with columns 'Key', 'Value', and 'Description'. The table contains one row with 'Key' and 'Value' fields. Below the table, there is a 'Bulk Edit' button. At the bottom, the 'Body' tab is selected, showing a status of '200 OK', a time of '84 ms', and a size of '123 B'. The response body is displayed in a 'Pretty' format.

Overview GET listar POST Cadastrar PUT Atualizar DEL Deletar No environment

DemoRest API / Deletar

DELETE http://localhost:8080/contatos/1 Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (4) Test Results Status: 200 OK Time: 84 ms Size: 123 B Save as example

Pretty Raw Preview Visualize Text

1