

25/8/2025

TP2

Desenvolvimento de Serviços com SpringBoot

Professor(a): Flávio da Silva Neves

LINK GITHUB

https://github.com/faculdade-infnet/V-1-Spring_Boot/tree/main/TP2

1. VERSÃO DO JAVA

Versão Java

Escolhida a versão Java 21, por ser a versão LTS (Long-Term Support) mais recente, ou seja, ela recebe suporte estendido de longo prazo, com atualizações de segurança e correções por vários anos, garantindo estabilidade e confiabilidade para aplicações em produção.

Versão Spring Boot

Escolhida a versão Spring Boot 3.5.5, que é versão estável mais recente (GA – General Availability), tendo as últimas correções de bugs e melhoria, além de ser totalmente compatível com o Java 21 e com isso proporcionando um ambiente moderno e robusto para desenvolvimento de aplicações de grande porte.

2. INICIAÇÃO DO PROJETO

The screenshot shows the Spring Initializr web application interface. It is a dark-themed form for creating a new Spring project. The interface is divided into several sections:

- Project:** Includes radio buttons for **Gradle - Groovy**, **Gradle - Kotlin**, and **Maven** (which is selected).
- Language:** Includes radio buttons for **Java** (selected), **Kotlin**, and **Groovy**.
- Spring Boot:** Includes radio buttons for various versions: **4.0.0 (SNAPSHOT)**, **4.0.0 (M2)**, **3.5.6 (SNAPSHOT)**, **3.5.5** (selected), and **3.4.10 (SNAPSHOT)** / **3.4.9**.
- Project Metadata:** A series of input fields for:
 - Group:** com.infnet
 - Artifact:** TP2
 - Name:** TP2
 - Description:** TP2 - Desenvolvimento de Serviços com SpringBoot
 - Package name:** com.infnet.TP2
 - Packaging:** Radio buttons for **Jar** (selected) and **War**.
 - Java:** Radio buttons for **24**, **21** (selected), and **17**.
- Dependencies:** A section on the right with a button **ADD DEPENDENCIES... CTRL + B**. It lists several dependencies:
 - Lombok** (DEVELOPER TOOLS): Java annotation library which helps to reduce boilerplate code.
 - Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
 - Spring Boot DevTools** (DEVELOPER TOOLS): Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

3. GERENCIAMENTO DE DEPENDÊNCIAS:

Utilizei o arquivo pom.xml do Maven para declarar todas as dependências necessárias.

Os principais benefícios da minha escolha:

- Controle claro das bibliotecas usadas.
- Facilidade para adicionar/remover dependências.

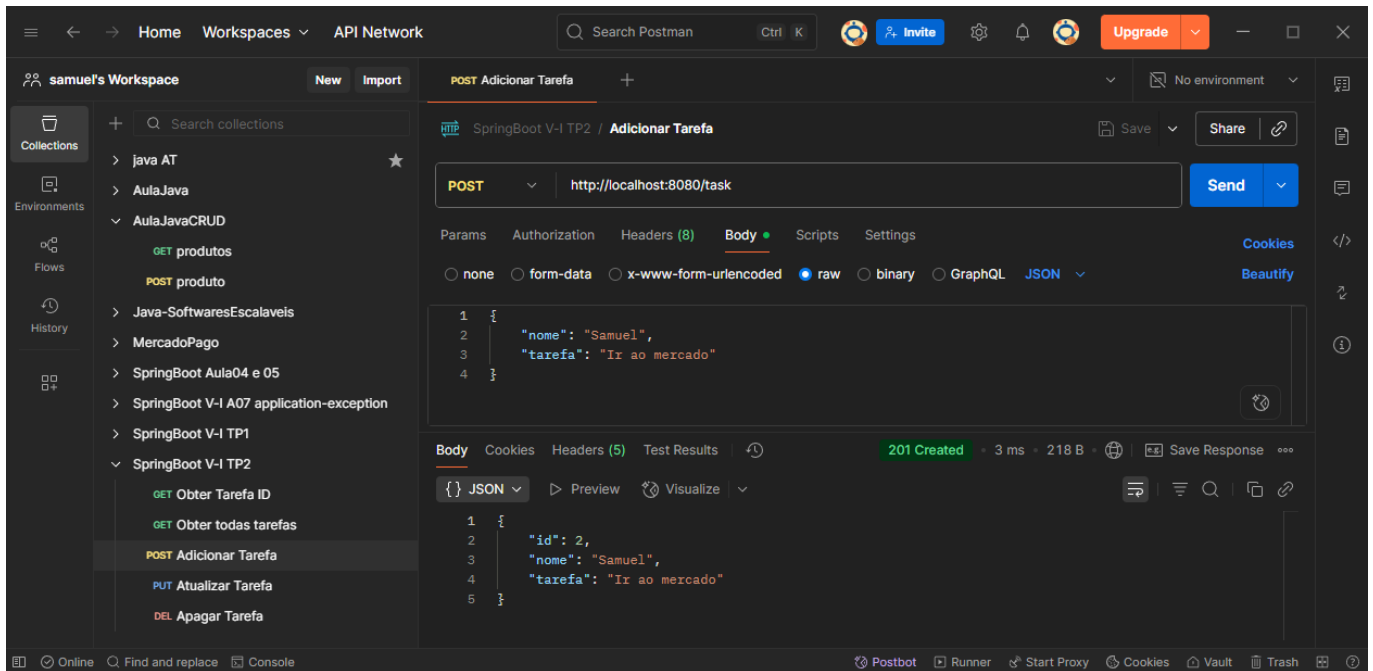
4. UTILIZAÇÃO DE AUTOCONFIGURAÇÃO:

Spring Boot configura automaticamente o contexto da aplicação para o Spring Web, via dependência spring-boot-starter-web, evitando necessidade de configuração manual de servidores, roteamento, e beans.

Utilizei em conjunto o `@SpringBootApplication` que inclui `@EnableAutoConfiguration`, no início onde da aplicação, resultando em um código mais limpo, menos repetição de código e foco no negócio.

5. VALIDAÇÃO DOS CÓDIGOS DE STATUS HTTP DE ACORDO COM AS ESPECIFICAÇÕES REST:

POST – Adicionar Tarefa



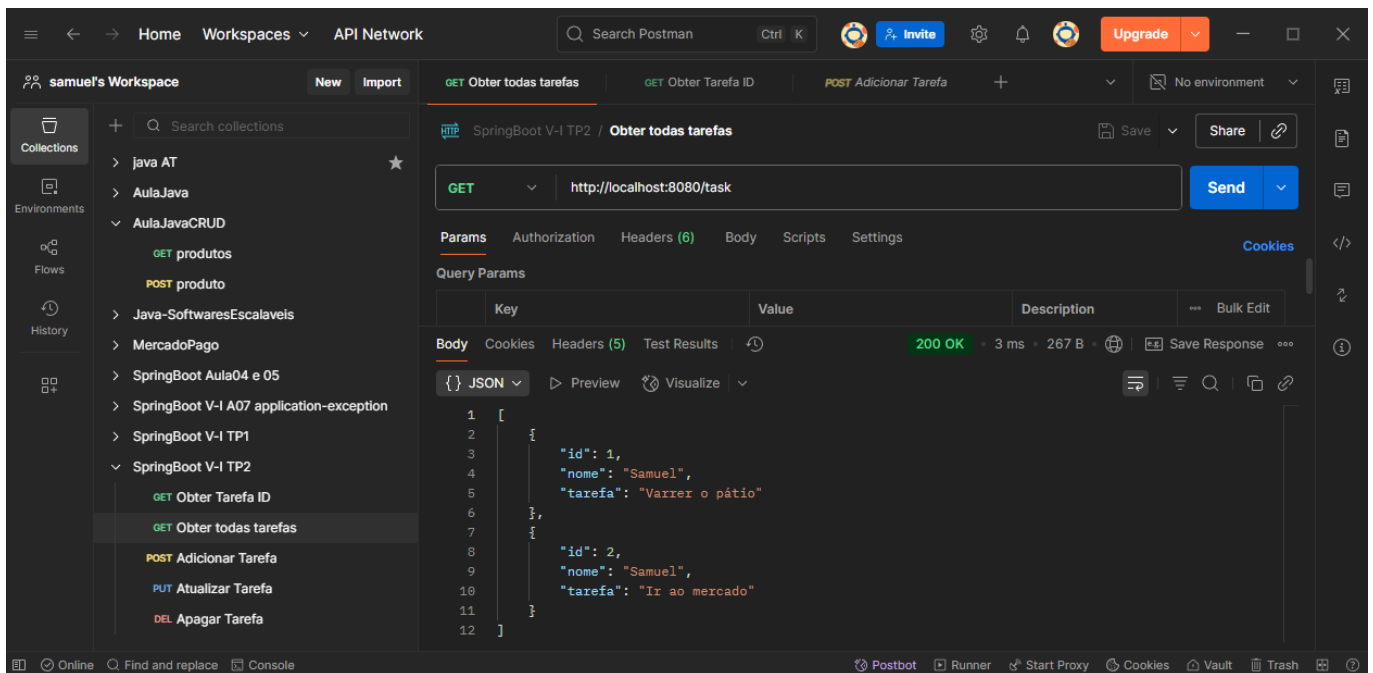
The screenshot shows the Postman interface with a workspace named 'samuel's Workspace'. The left sidebar displays a collection of API endpoints under 'SpringBoot V-I TP2'. The main panel shows a POST request to 'http://localhost:8080/task' with a JSON body:

```
{  "nome": "Samuel",  "tarefa": "Ir ao mercado"}
```

. The response is '201 Created' with a status of 201, a time of 3 ms, and a size of 218 B. The response body is shown in JSON format:

```
{  "id": 2,  "nome": "Samuel",  "tarefa": "Ir ao mercado"}
```

GET – Obter todas as tarefas



The screenshot shows the Postman interface with the same workspace. The main panel shows a GET request to 'http://localhost:8080/task'. The response is '200 OK' with a status of 200, a time of 3 ms, and a size of 267 B. The response body is shown in JSON format:

```
[  {    "id": 1,    "nome": "Samuel",    "tarefa": "Varrer o pátio"  },  {    "id": 2,    "nome": "Samuel",    "tarefa": "Ir ao mercado"  }]
```

GET – Obter tarefa por ID

The screenshot shows the Postman interface with a workspace named 'samuel's Workspace'. The left sidebar displays a collection of API requests under 'SpringBoot V-I TP2', including 'Obter Tarefa ID'. The main panel shows the 'Obter Tarefa ID' request, which is a GET request to 'http://localhost:8080/task/1'. The response is a 200 OK status with a 5 ms response time and a 215 B body. The response body is a JSON object:

```
1 {
2   "id": 1,
3   "nome": "Samuel",
4   "tarefa": "Varrer o pátio"
5 }
```

PUT – Atualizar Tarefa

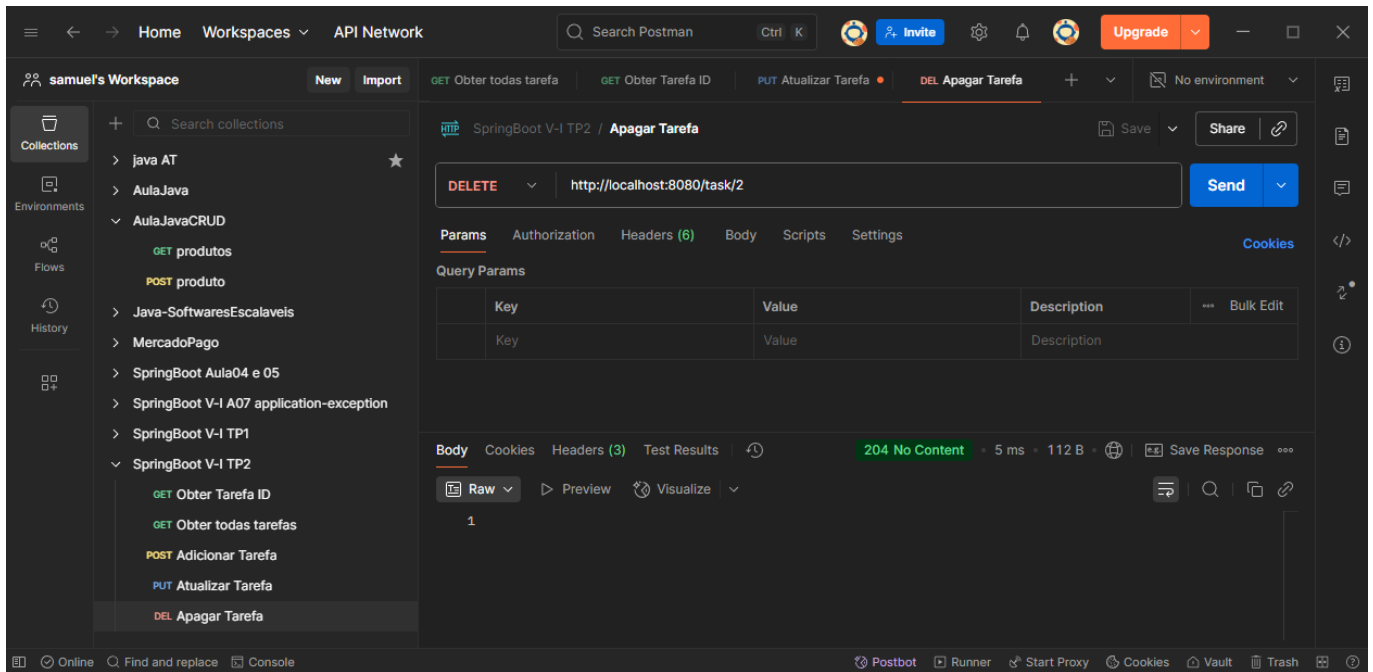
The screenshot shows the Postman interface with the same workspace. The left sidebar shows the 'PUT Atualizar Tarefa' request selected. The main panel shows the 'PUT Atualizar Tarefa' request, which is a PUT request to 'http://localhost:8080/task/1'. The request body is a JSON object:

```
1 {
2   "nome": "Samuel",
3   "tarefa": "Caminhar"
4 }
```

The response is a 200 OK status with a 4 ms response time and a 208 B body. The response body is a JSON object:

```
1 {
2   "id": 1,
3   "nome": "Samuel",
4   "tarefa": "Caminhar"
5 }
```

DELETE – Deletar tarefa



6. DESENVOLVIMENTO DE SERVIÇOS REST:

```
package com.infnet.TP2.api;

import com.infnet.TP2.facade.TaskFacade;
import com.infnet.TP2.models.TaskModels;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@Controller
@RequestMapping(value = "/task", produces = MediaType.APPLICATION_JSON_VALUE)
public class TaskAPI {

    // Ativa a injeção de dependência automática
    @Autowired
    private TaskFacade taskFacade;

    // Adiciona uma tarefa
    @PostMapping
    @ResponseBody
    public ResponseEntity<TaskModels> create(@RequestBody TaskModels taskModels) {
        TaskModels created = taskFacade.create(taskModels);
        return ResponseEntity.status(HttpStatus.CREATED).body(created); // 201
Created
    }

    // Obtém todas as tarefas
    @GetMapping
    @ResponseBody
    public ResponseEntity<List<TaskModels>> getAll() {
        List<TaskModels> tasks = taskFacade.getAll();
        return ResponseEntity.ok(tasks); // 200 OK
    }

    // Obtém uma tarefa por id
```

```

@GetMapping("/{taskId}")
@ResponseBody
public ResponseEntity<TaskModels> getById(@PathVariable Long taskId) {
    TaskModels task = taskFacade.getById(taskId);
    if (task == null) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).build(); // 404 Not
Found
    }
    return ResponseEntity.ok(task); // 200 OK
}

// Atualiza uma tarefa por id
@PutMapping("/{taskId}")
@ResponseBody
public ResponseEntity<TaskModels> update(@PathVariable("taskId") Long taskId,
@RequestBody TaskModels taskModels) {
    TaskModels existing = taskFacade.getById(taskId);
    if (existing == null) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).build(); // 404 Not
Found
    }
    taskModels.setId(taskId);
    TaskModels updated = taskFacade.update(taskModels, taskId);
    return ResponseEntity.ok(updated); // 200 OK
}

// Deleta uma tarefa por id
@DeleteMapping("/{taskId}")
@ResponseBody
public ResponseEntity<Void> delete(@PathVariable("taskId") Long taskId) {
    TaskModels existing = taskFacade.getById(taskId);
    if (existing == null) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).build(); // 404 Not
Found
    }
    taskFacade.delete(taskId);
    return ResponseEntity.noContent().build(); // 204 No Content
}
}

```