PONTIFÍCIA UNIVERSIDADE CATOLICA DE MINAS GERAIS

ROTEIRO 01

CAMADAS DE CONTROLLER E REPOSITORY

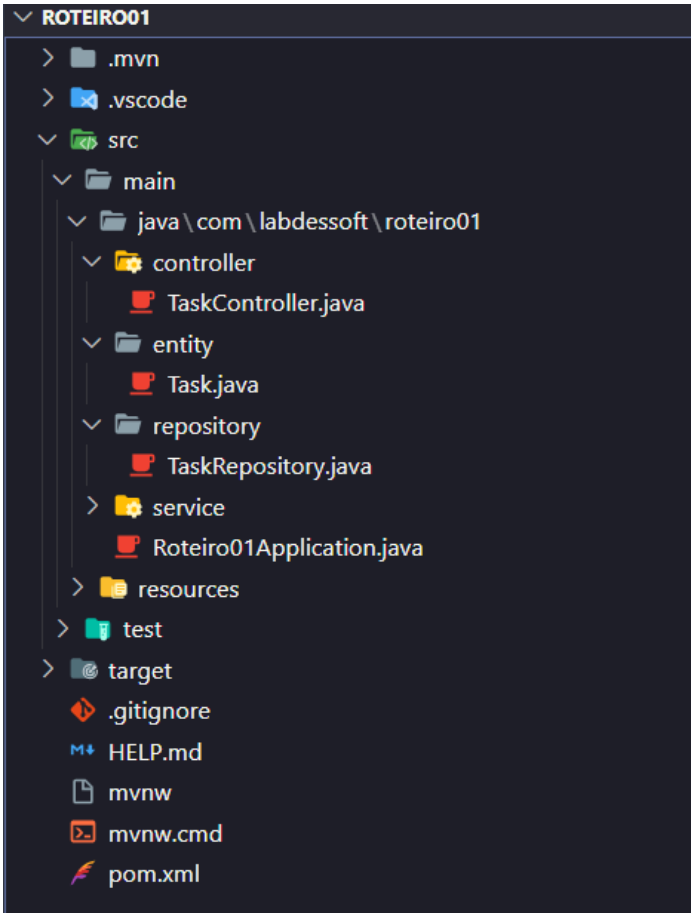
ENTREGA 2

Júlia Borges Araújo Silva

Belo Horizonte
2024

Estrutura de pastas:

Para a elaboração do Roteiro 01, foi realizada uma organização de pastas seguindo uma arquitetura que adota o padrão Model-View-Controller (MVC). A estrutura de pastas adotada assemelha-se ao seguinte:



Códigos fonte:

Os códigos fontes criados são os seguintes:

1. **Em controller/TaskController.java:**

```
package com.labdessoft.roteiro01.controller;

import io.swagger.v3.oas.annotations.Operation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
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;
```

```java
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.labdessoft.roteiro01.entity.Task;
import com.labdessoft.roteiro01.repository.TaskRepository;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@RestController
public class TaskController {
    @Autowired
    TaskRepository taskRepository;

    @SuppressWarnings("null")
    @GetMapping("/task")
    @Operation(summary = "Lista todas as tarefas da lista")
    public ResponseEntity<List<Task>> listAll() {
        try {
            List<Task> taskList = new ArrayList<Task>();
            taskRepository.findAll().forEach(taskList::add);
            if (taskList.isEmpty()) {
                return new
ResponseEntity<>(HttpStatus.NO_CONTENT);
            }
            return new ResponseEntity<>(taskList, HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>(null,
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    @GetMapping("/task/{id}")
    @Operation(summary = "Procura uma tarefa através do ID")
    public ResponseEntity<Task> findTaskById(@PathVariable("id")
long id) {
        Optional<Task> taskData = taskRepository.findById(id);

        if (taskData.isPresent()) {
            return new ResponseEntity<>(taskData.get(),
HttpStatus.OK);
        } else {
```

```
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @SuppressWarnings("null")
    @PostMapping("/task")
    @Operation(summary = "Inclui uma tarefa na lista")
    public ResponseEntity<Task> addTask(@RequestBody Task task) {
        try {
            Task newTask = taskRepository.save(new
Task(task.getDescription()));

            return new ResponseEntity<>(newTask,
HttpStatus.CREATED);
        } catch (Exception e) {
            return new ResponseEntity<>(null,
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    @PutMapping("/task/{id}")
    @Operation(summary = "Atualiza uma tarefa na lista através do
ID")
    public ResponseEntity<Task> updateTask(@PathVariable("id")
long id, @RequestBody Task task) {
        Optional<Task> taskData = taskRepository.findById(id);

        if (taskData.isPresent()) {
            Task newTask = taskData.get();
            newTask.setDescription(task.getDescription());
            newTask.setCompleted(task.getCompleted());
            return new
ResponseEntity<>(taskRepository.save(newTask), HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @DeleteMapping("/task/{id}")
    @Operation(summary = "Deleta uma tarefa pelo ID")
    public ResponseEntity<Void> deleteTask(@PathVariable("id")
long id) {
        try {
```

```
            if (taskRepository.existsById(id)) {
                taskRepository.deleteById(id);
                return new
ResponseEntity<>(HttpStatus.NO_CONTENT);
} else {
                return new ResponseEntity<>(HttpStatus.ACCEPTED);
            }
        } catch (Exception e) {
            return new
ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

}
```

```
package com.labdessoft.roteiro01.entity;

import io.swagger.v3.oas.annotations.media.Schema;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.validation.constraints.Size;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Entity
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Schema(description = "Todos os detalhes sobre uma tarefa")
public class Task {
@Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Schema(name = "Descrição da tarefa deve possuir pelo menos
10 caracteres")
    @Size(min = 10, message = "Descrição da tarefa deve ter no
mínimo 10 caracteres")
```

2. **Em entity/Task.java:**

```
private String description; private
    Boolean completed;

    public Task(String description){
        this.description = description;
        }

@Override
public String toString() {
return "Task [id=" + id + ", description=" + description
+ ", completed=" +
completed + "]";
        }
}
```

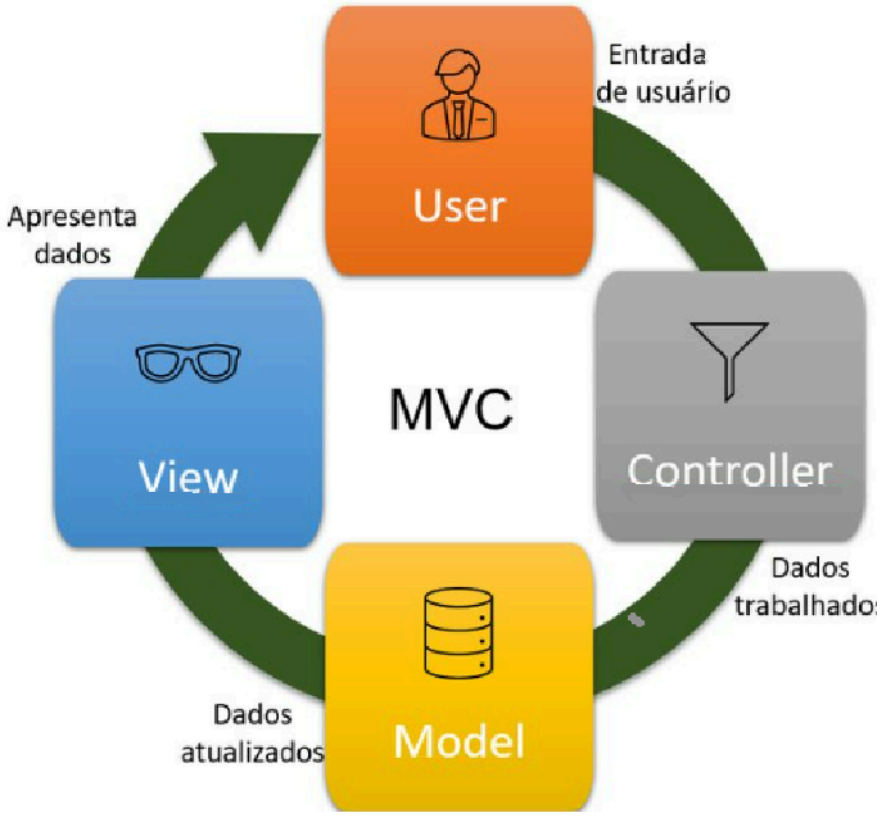
3. **Em repository/TaskRepository.java:**

```
package com.labdessoft.roteiro01.repository;

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

import com.labdessoft.roteiro01.entity.Task;

public interface TaskRepository extends JpaRepository<Task, Long>
{

}
```

4. **Em Roteiro01Application.java:**

```java
package com.labdessoft.roteiro01;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Roteiro01Application {

    public static void main(String[] args) {
        SpringApplication.run(Roteiro01Application.class, args);
    }

}
```

Desenho de como as camadas da solução estão funcionando:



Vantagens e Desvantagens do Modelo de Implementação:

Vantagens:

Separação: A clara separação entre lógica de negócios, interface do usuário e controle de acesso a dados facilita a manutenção e a escalabilidade.

Desenvolvimento Modular: Cada componente pode ser desenvolvido, testado e depurado independentemente, aumentando a eficiência do desenvolvimento.

Facilidade de Teste: A separação facilita a realização de testes unitários e de integração em cada camada.

Reuso de Código: A camada de Model e Repository pode ser reutilizada por diferentes Views e Controllers, promovendo o reuso de código.

Desvantagens:

Complexidade: Para projetos menores, a estrutura MVC pode introduzir uma complexidade desnecessária, tornando o desenvolvimento mais lento.

Acoplamento: Apesar da separação de concerns, pode haver um acoplamento indireto entre as camadas, especialmente se não forem bem projetadas.