análise de REQUISITOS .com.br

ROTEIRO 01

ESPECIFICAÇÃO DE CASOS DE USO

DOCUMENTO 1-0001

ARTHUR CAPANEMA BRETAS

ÚLTIMA ATUALIZAÇÃO: 15/03/2024

HISTÓRICO DE REVISÕES DO DOCUMENTO

DATA	VERSÃO	DESCRIÇÃO DA ALTERAÇÃO	AUTOR
04/03/2024	1	CRIAÇÃO DESTE DOCUMENTO	Arthur Capanema Bretas
10/03/2024	2	INCLUSÃO DO CASOS DE USO 01	Arthur Capanema Bretas
10/03/2024	3	INCLUSÃO DO CASOS DE USO 02	Arthur Capanema Bretas
31/03/2024	4	INCLUSÃO DO CASOS DE USO 03	Arthur Capanema Bretas
31/03/2024	5	INCLUSÃO DO CASOS DE USO 04	Arthur Capanema Bretas
31/03/2023	6	CONCLUSÃO DO DOCUMENTO	Arthur Capanema Bretas

IDENTIFICAÇÃO DOS ENVOLVIDOS

PAPEL	NOME	EMAIL
ANALISTA DE REQUISITOS	Arthur Capanema Bretas	arthurcbretas@gmail.com
PRODUCT OWNER	Cristiano de Macedo Neto	
STAKEHOLDER	Cristiano de Macedo Neto	
PATROCINADOR	Cristiano de Macedo Neto	

DESCRIÇÃO DO CASO 1

O caso de uso CDU001 – Listar Tarefas na Lista de Tarefas é responsável por listar as tarefas na lista de tarefas.

DOCUMENTOS RELACIONADOS CASO 1

- Diagrama_Caso_Uso_1

DIAGRAMA DO CASO 1

Diagrama_Caso_Uso_1

ATORES CASO 1

Usuário: Pessoa que usará o sistema de gerenciamento de lista de tarefas.

PRÉ-CONDIÇÕES CASO 1

Não se aplica.

FLUXO PRINCIPAL DO CASO 1

FB01 – Listar Tarefas na Lista de Tarefas	
ID	**Passo**
1	O **sistema** busca no banco de dados todas as tarefas existentes;
2	O **sistema** exibe todas essas tarefas cadastradas no banco de dados.
3	O caso de uso é finalizado.

PÓS-CONDIÇÃO OU RESULTADO ESPERADO 1

O sistema deve persistir os dados informados.

DESCRIÇÃO DO CASO 2

O caso de uso CDU002 – Adicionar Tarefas na Lista de Tarefas é responsável por incluir uma tarefa na lista de tarefas

DOCUMENTOS RELACIONADOS CASO 2

- Diagrama_Caso_Uso_2

DIAGRAMA DO CASO 2

Diagrama_Caso_Uso_2

ATORES CASO 2

Usuário: Pessoa que usará o sistema de gerenciamento de lista de tarefas.

PRÉ-CONDIÇÕES CASO 2

Para o caso 2 acontecer o caso 1 deve ter sido completo, pois o sistema deve ser capaz de listar as tarefas que serão adicionadas.

FLUXO PRINCIPAL DO CASO 2

FB02 – Concluir Tarefa na Lista de Tarefas	
ID	**Passo**
1	O **sistema** exibe todas as tarefas cadastradas no banco de dados;
2	O **usuário** informa o nome e a descrição da tarefa desejada e confirma a operação.
3	O **sistema** verifique se o nome ou a descrição da tarefa NÂO foi preenchida e exibe a mensagem de erro “Favor preencher a tarefa!” e retornar para o passo 2.
4	O **sistema** exibe a tarefa cadastrada no topo da lista.
5	O caso de uso finaliza.

PÓS-CONDIÇÃO OU RESULTADO ESPERADO 2

O sistema deve persistir os dados informados.

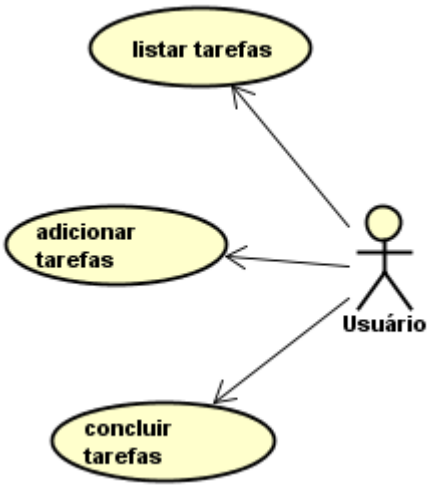
DESCRIÇÃO DO CASO 3

O caso de uso CDU003 – Concluir Tarefa na Lista de Tarefas é responsável por marcar uma tarefa na lista de tarefas como concluída

DOCUMENTOS RELACIONADOS CASO 3

- Diagrama_Caso_Uso_3

DIAGRAMA DO CASO 3



Diagrama_Caso_Uso_3

ATORES CASO 3

Usuário: Pessoa que usará o sistema de gerenciamento de lista de tarefas.

PRÉ-CONDIÇÕES CASO 3

Os Caso de uso 1 e 2 devem ter sido completos para que alguma tarefa listada seja então concluída.

FLUXO PRINCIPAL DO CASO 3

FB03xz – Concluir Tarefa na Lista de Tarefas	
ID	**Passo**
1	O **sistema** exibe todas as tarefas cadastradas no banco de dados;
2	O **usuário** seleciona a tarefa a ser concluída, acionando a operação (X)
3	O **sistema** exibe uma mensagem de confirmação “Confirma a realização da tarefa? [Sim] [Não]”.
4	Caso o usuário selecione [**Não**] o sistema retorna ao passo 2 do fluxo atual
5	O **sistema** move a tarefa confirmada para a lista de tarefas concluídas.
6	O caso de uso finaliza.

PÓS-CONDIÇÃO OU RESULTADO ESPERADO 3

O sistema deve persistir os dados informados.

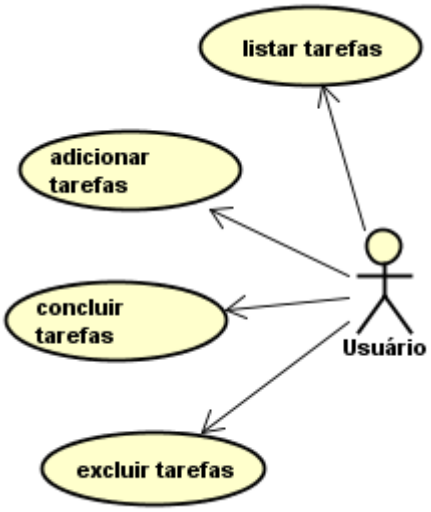
DESCRIÇÃO DO CASO 4

O caso de uso CDU004 – Concluir Tarefa na Lista de Tarefas é responsável por marcar uma tarefa na lista de tarefas como concluída

DOCUMENTOS RELACIONADOS CASO 4

- Diagrama_Caso_Uso_4

DIAGRAMA DO CASO 4



Diagrama_Caso_Uso_4

ATORES CASO 4

Usuário: Pessoa que usará o sistema de gerenciamento de lista de tarefas.

PRÉ-CONDIÇÕES CASO 4

Os casos de uso 1 e 2 devem ter sido completos para que uma tarefa listada seja, então, excluída.

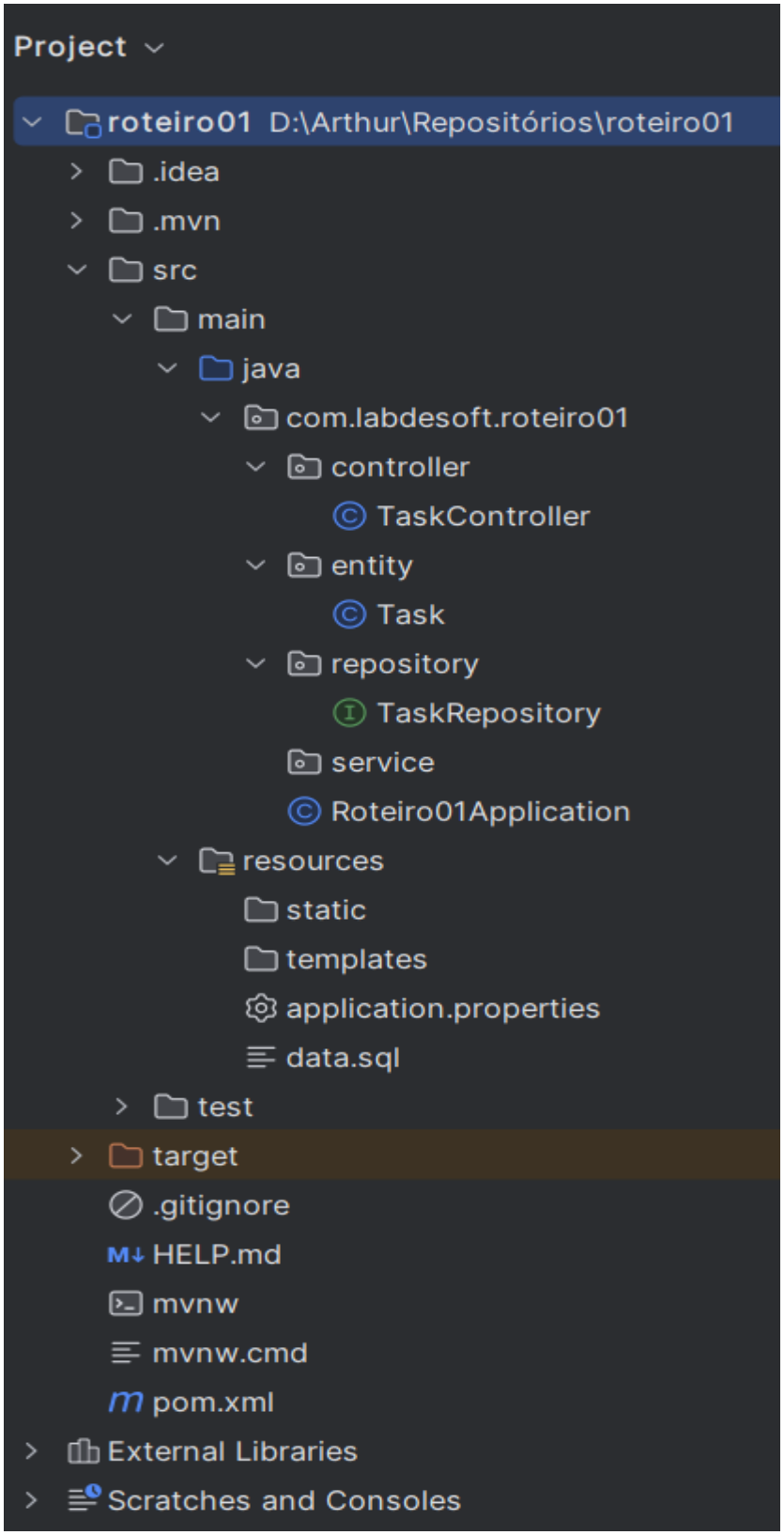
FLUXO PRINCIPAL DO CASO 4

FB04 – Excluir Tarefa na Lista de Tarefas	
ID	**Passo**
1	O **sistema** exibe todas as tarefas cadastradas no banco de dados;
2	O **usuário** seleciona a tarefa a ser excluída, acionando a operação.
3	O **sistema** exibe uma mensagem de confirmação “Confirma a exclusão da tarefa? [**Sim**] [**Não**]”.
4	Caso o usuário selecione [**Não**] o sistema retorna ao passo 3 do fluxo atual
5	O **sistema** exclui a tarefa do banco de dados.
6	O caso de uso finaliza.

PÓS-CONDIÇÃO OU RESULTADO ESPERADO 4

O sistema deve persistir os dados informados.

ESTRUTURA DAS PASTAS



Organização das Pastas I

CÓDIGO FONTE

1)Controller

```
package com.labdesoft.roteiro01.controller;

import com.labdesoft.roteiro01.entity.Task;
import com.labdesoft.roteiro01.repository.TaskRepository;
import io.swagger.v3.oas.annotations.Operation;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;


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

@RestController
public class TaskController {

    @Autowired
    TaskRepository taskRepository;

    @GetMapping("/task")
    @Operation(summary = "Lista todas as tarefas da lista")
    public ResponseEntity<List<Task>> listAll() {
        try {
            List<Task> taskList = new ArrayList<>();
            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);
        }
    }

    @PostMapping("/task")
    @Operation(summary = "Adiciona uma nova tarefa")
    public ResponseEntity<Task> addTask(@Valid @RequestBody Task task)
{
        try {
            Task newTask = taskRepository.save(task);
            return new ResponseEntity<>(newTask, HttpStatus.CREATED);
        } catch (Exception e) {
            return new ResponseEntity<>(null,
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    @PatchMapping("/task/{id}/complete")
    @Operation(summary = "Marca uma tarefa como completa")
```

```
    public ResponseEntity<Task> markTaskAsCompleted(@PathVariable Long
id) {
        try {
            Optional<Task> optionalTask = taskRepository.findById(id);
            if (optionalTask.isPresent()) {
                Task existingTask = optionalTask.get();
                existingTask.setCompleted(true);
                Task updatedTask = taskRepository.save(existingTask);
                return ResponseEntity.ok(updatedTask);
            } else {
                return ResponseEntity.notFound().build();
            }
        } catch (Exception e) {
            return new ResponseEntity<>(null,
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    @DeleteMapping("/task/{id}")
    @Operation(summary = "Exclui uma tarefa")
    public ResponseEntity<HttpStatus> deleteTask(@PathVariable Long id)
{
        try {
            taskRepository.deleteById(id);
            return new ResponseEntity<>(HttpStatus.NO_CONTENT);
        } catch (Exception e) {
            return new
ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
}
```

2)Entity

```
package com.labdesoft.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 possuir pelo
menos 10 caracteres")
            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 + "]";
    }

    //Getters e Setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Boolean getCompleted() {
        return completed;
    }

    public void setCompleted(Boolean completed) {
        this.completed = completed;
    }
}
```

3)Repository

```
package com.labdesoft.roteiro01.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import com.labdesoft.roteiro01.entity.Task;
public interface TaskRepository extends JpaRepository<Task, Long> {
}
```

4)Service

```
package com.labdesoft.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);
    }

}
```

Vantagens do Modelo MVC:

- **Organização Estruturada:** A divisão clara de responsabilidades entre as camadas torna nosso código mais organizado e fácil de entender.
- **Flexibilidade e Manutenção:** A separação das camadas nos permite fazer modificações em partes específicas da aplicação sem afetar as outras, facilitando a manutenção e a evolução do sistema ao longo do tempo.
- **Reutilização de Componentes:** Componentes como Controllers e Models podem ser reutilizados em diferentes partes da aplicação, o que aumenta a produtividade e reduz o tempo de desenvolvimento.
- **Testabilidade:** A estrutura MVC facilita a escrita de testes automatizados para validar o comportamento esperado de cada componente, o que contribui para a qualidade e a robustez do software.

Desvantagens do Modelo MVC:

- **Complexidade Inicial:** A introdução da arquitetura MVC pode ser desafiadora para desenvolvedores menos experientes, pois exige o entendimento de conceitos e padrões específicos.
- **Possível Sobrecarga de Abstração:** Em alguns casos, a divisão das camadas pode resultar em uma sobrecarga de abstração, tornando o código mais complexo e difícil de dar manutenção.
- **Curva de Aprendizado:** Para equipes novas ou inexperientes, pode haver uma curva de aprendizado para dominar os conceitos e as melhores práticas da arquitetura MVC.
- **Possível Acoplamento Indesejado:** Se não implementado corretamente, o modelo MVC pode levar a um acoplamento excessivo entre as camadas, o que pode dificultar a manutenção e a evolução do sistema no futuro.