******

***Engenharia de Software***

***Segunda Avaliação***

*Integrantes*

*Murilo Barbosa 816577*

*Gustavo Castro 765954*

*Murilo da Silva Barbosa*

*Segunda Avaliação Final*

*Trabalho da instituição UNAERP*

*servindo como segunda avaliação final.*

*Teste de Software*

*Fabiano*

*Ribeirão Preto*

*Resumo*

A avaliação tem como objetivo desenvolver uma API Rest. com Spring boot, o tema sendo escolhido pelo aluno. Elaborar um relatório que será apresentado ao professor contendo oque foi desenvolvido, mostrando o desenvolvimento e os teste em TDD.

***Palavras-chaves:*** *Spring Boot, Rest. API, TDD.*

Sumário

[Introdução 4](#_Toc74912098)

[Desenvolvimento 4](#_Toc74912099)

[Conclusão 11](#_Toc74912100)

# Introdução

O programa desenvolvido contem programa desenvolvido em Java, sendo ele com o objetivo de desenvolver uma loja de livros, por isso foi desenvolvido uma API Rest. Que foi usando o Spring Boot para auxiliar na programação.

# Desenvolvimento

Foi desenvolvido um programa em Java que irá auxiliar uma loja de livros no gerenciamento da sua biblioteca, com o Auxilio do Spring Boot, o programa é um Rest. API. Foi utilizado no programa o método de teste TDD (Test Drive in Development) para comprovar o sucesso de sua aplicação.

Foi criado um model para a biblioteca chamada de Livros.java.

package apibiblioteca.biblioteca.Models;

import java.io.Serializable;

import java.math.BigDecimal;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

import javax.validation.constraints.NotNull;

@Entity

@Table(name= "bibliotecas")

public class Livros implements Serializable{

    private static final long serialVersionUID= 1L;

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private long id;

    @NotNull

    private String titulo;

    @NotNull

    private String autor;

    @NotNull

    private String descricao;

    @NotNull

    private String data;

    @NotNull

    private BigDecimal quantidade;

    @NotNull

    private BigDecimal valor;

    public Livros(long l, String string, String string2, String string3, String string4, int i, double d) {

    }

    public String getDescricao() {

        return descricao;

    }

    public void setDescricao(String descricao) {

        this.descricao = descricao;

    }

    public String getData() {

        return data;

    }

    public void setData(String data) {

        this.data = data;

    }

    public long getId() {

        return id;

    }

    public void setId(long id) {

        this.id = id;

    }

    public String getTitulo() {

        return titulo;

    }

    public void setTitulo(String titulo) {

        this.titulo = titulo;

    }

    public String getAutor() {

        return autor;

    }

    public void setAutor(String autor) {

        this.autor = autor;

    }

    public BigDecimal getQuantidade() {

        return quantidade;

    }

    public void setQuantidade(BigDecimal quantidade) {

        this.quantidade = quantidade;

    }

    public BigDecimal getValor() {

        return valor;

    }

    public void setValor(BigDecimal valor) {

        this.valor = valor;

    }

}

Foi criado um arquivo para as funções para execução da api.

package apibiblioteca.biblioteca.Resources;

import java.util.List;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.CrossOrigin;

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 apibiblioteca.biblioteca.Models.Livros;

import apibiblioteca.biblioteca.Repository.LivroRepository;

@CrossOrigin(origins = "\*")

@RestController

@RequestMapping(value = "/api")

public class LivrosResources {

    @Autowired

    LivroRepository livroRepository;

    @GetMapping("/livros")

    public List<Livros> listaLivros(){

        return livroRepository.findAll();

    }

    @GetMapping("/livro/{id}")

    public Livros listaLivro(@PathVariable(value = "id") long id){

        return livroRepository.findById(id);

    }

    @DeleteMapping("/livro")

    public void deletaLivro(@RequestBody @Valid Livros livros){

        livroRepository.delete(livros);

    }

    @PostMapping("/livro")

    public Livros salvarLivros(@RequestBody @Valid Livros livros){

        return livroRepository.save(livros);

    }

    @PutMapping("livros")

    public Livros atualizarLivros(@RequestBody @Valid Livros livros){

        return livroRepository.save(livros);

    }

}

Além disso, criou uma interface com a função de localização do campo id no banco de dados.

package apibiblioteca.biblioteca.Repository;

import java.util.Optional;

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

import apibiblioteca.biblioteca.Models.Livros;

public interface LivroRepository extends JpaRepository<Livros, Long>{

    Livros findById(long id);

    Optional<Livros> findByReserveName(String name);

}

A função main.

package apibiblioteca.biblioteca;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BibliotecaApplication {

    public static void main(String[] args) {

        SpringApplication.run(BibliotecaApplication.class, args);

    }

}

E a aplication.properties.

spring.jpa.properties.hibernate.jdbc.lob.non\_contextual\_creation = true

spring.jpa.hibernate.ddl-auto=update

spring.datasource.url=jdbc:mysql://localhost:3306/biblioteca

spring.datasource.username=root

spring.datasource.password=

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

spring.jpa.show-sql: true

server.port=8090

Classe controller.

package apibiblioteca.biblioteca.Controller;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import org.springframework.web.bind.annotation.RestController;

import apibiblioteca.biblioteca.Models.Livros;

@RestController

@RequestMapping("/bibliotecas")

public class BibliotecaController {

    @GetMapping

    @ResponseBody

    public String getAll(){

        return "Ok";

    }

    @PostMapping

    public ResponseEntity<Livros> save(Livros livros){

        return ResponseEntity.status(HttpStatus.OK).body(livros);

    }

}

BibliotecaControllerTest.

package apibiblioteca.biblioteca;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.junit.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.web.servlet.MockMvc;

import apibiblioteca.biblioteca.Models.Livros;

@SpringBootTest

@AutoConfigureMockMvc

public class BibliotecaControllerTest {

    @Autowired

    MockMvc mockMvc;

    @Autowired

    private ObjectMapper objectMapper;

    @Test

    public void bibliotecaTestGetAll() throws Exception{

        mockMvc.perform(get("/bibliotecas"))

            .andExpect(status().isOk());

    }

    @Test

    public void bibliotecaTestSave() throws Exception{

        Livros livros = new Livros(1L, "Harry Potter", "JK Rowling", "Livros de magia", "12/04/1997", 5, 65);

        mockMvc.perform(post("/bibliotecas"))

            .contentType("application/json")

            .content(objectMapper.writeValueAsString(livros))

            .andExpect(status().isOk());

    }

}

BibliotecaServiceTest.

package apibiblioteca.biblioteca;

import org.junit.Before;

import org.junit.Test;

import org.junit.jupiter.api.Assertions;

import org.junit.runner.RunWith;

import org.mockito.Mockito;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.context.TestConfiguration;

import org.springframework.boot.test.mock.mockito.MockBean;

import org.springframework.context.annotation.Bean;

import org.springframework.test.context.junit4.SpringRunner;

import apibiblioteca.biblioteca.Models.Livros;

import apibiblioteca.biblioteca.Repository.LivroRepository;

import apibiblioteca.biblioteca.Service.BibliotecaService;

@RunWith(SpringRunner.class)

public class BibliotecaServiceTest {

    @TestConfiguration

    static class BibliotecaServiceTestConfiguration{

        @Bean

        public BibliotecaService bService(){

            return new BibliotecaService();

        }

    }

    @Autowired

    BibliotecaService bibliotecaService;

    @MockBean

    LivroRepository livroRepository;

    @Test

    public void BibliotecaServiceDados(){

        String nome = "Harry Potter";

        String index = bibliotecaService.tituloRetorno(nome);

        Assertions.assertEquals(index, "Harry Potter");

    }

    @Before

    public void setup(){

        Livros livros =new Livros(1L, "Harry Potter", "JK Rowling",

         "Livro de magia", "12/12/1997", 5, 65.5);

         Mockito.when(livros.getTitulo().toString()).thenReturn(livros.getTitulo());

    }

}

# Conclusão

Considerando que a entrega é a apresentação do Spring boot funcionando. Ele funciona com sucesso. Porem houve alguns problemas no desenvolvimento como conexão e os testes em TDD também não funcionaram. O resultado foi da matéria foi o aprendizado que nos levou a construir não só a API mas também os teste que propuseram ao longo do semestre.