|  |
| --- |
| Министерство науки и высшего образования Российской Федерации |
| федеральное государственное бюджетное образовательное учреждение высшего образования |

ОТЧЕТ

по учебной практике

|  |
| --- |
| УП.04.01  Внедрение и поддержка программного обеспечения |

|  |
| --- |
| Профессионального модуля ПМ.04  Сопровождение и обслуживание программного обеспечения компьютерных систем |

|  |
| --- |
| Специальность 09.02.07  Информационные системы и программирование |

Студент:       Полжайковский Алексей Дмитриевич

*фамилия, имя, отчество*

Группа: П50-3-22

Руководитель по практической подготовке от техникума:

Белова Дарья Романовна

*фамилия, имя, отчество*  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2025 года

Содержание

[Практическая работа №1 3](#_Toc211617232)

[Практическая работа №2 8](#_Toc211617233)

# Практическая работа №1

Цель работы: создать приложение, включающее страницы «Калькулятор» и «Конвертер валют». Страница «Калькулятор» должна содержать функционал базового калькулятора. «Конвертер валют» – страница, на которой находится 2 выпадающих списка: в первом находится валюта, из которой надо перевести деньги, а во втором в какую валюту надо перевести.

Ход работы:

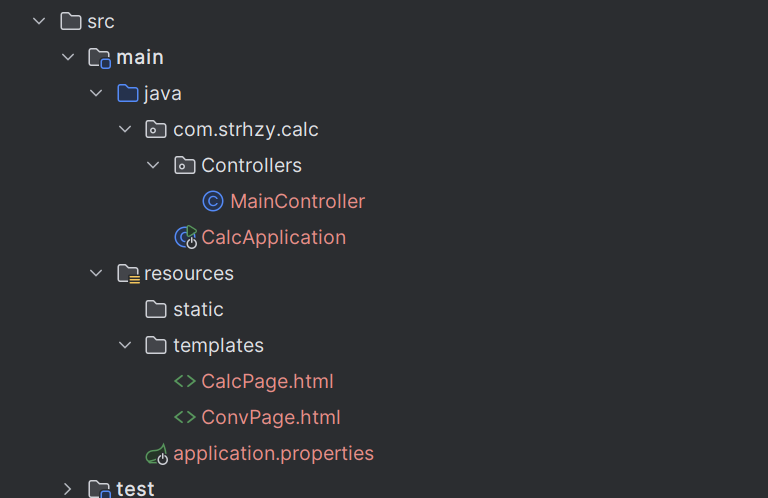


Рисунок - Структура проекта

Главным классом приложения, содержащим метод main, который запускает Spring Boot приложение является файл **CalcApplication**.

Для обработки HTTP запросов пользователей был создан **MainController**, здесь в зависимости от логики контроллер возвращает соответствующие шаблоны.

**Static** – папка для хранения статических ресурсов, в представленной структуре там хранятся стили для некоторых шаблонов.

**Templates** – папка с HTML-шаблонами, которые использует Thymeleaf для рендеринга страниц.

Аннотации, используемые в работе:

**@Controller** – указывает на класс контроллер.

**@RequestMapping** – говорит о том, что все методы в этом контроллере будут обрабатывать запросы, начинающиеся с определенного пути, в коде, представленном ниже, это /home.

**@GetMapping** – используется для указания, что метод контроллера будет обрабатывать GET-запросы.

**@PostMapping** – используется для указания, что метод контроллера будет обрабатывать POST-запросы.

**@RequestParam** – используется для привязки параметров запроса к методам контроллера. @RequestParam(name = «paramName», required = false, defaultValue = «defaultValue»), где name – имя параметра, required – указатель, является ли параметр обязательным, defaultValue – значение по умолчанию.

**Model** – это специальный объект, который используется для передачи данных из контроллера в HTML-шаблон.

Код программы:

1. MainController.java:

package com.strhzy.calc.Controllers;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
  
@Controller  
public class MainController {  
 @GetMapping("/")  
 public String showCalculator() {  
 return "CalcPage";  
 }  
  
 @PostMapping("/calculate")  
 public String calculate(@RequestParam double num1, @RequestParam double num2, @RequestParam String operation, Model model) {  
 double result = 0;  
 switch (operation) {  
 case "+":  
 result = num1 + num2;  
 break;  
 case "-":  
 result = num1 - num2;  
 break;  
 case "\*":  
 result = num1 \* num2;  
 break;  
 case "/":  
 if (num2 == 0) {  
 model.addAttribute("error", "Деление на ноль невозможно");  
 return "CalcPage";  
 }  
 result = num1 / num2;  
 break;  
 }  
  
 model.addAttribute("result", result);  
  
 return "CalcPage";  
 }  
  
 @GetMapping("/conv")  
 public String showConverter() {  
 return "ConvPage";  
 }  
  
 @PostMapping("/convert")  
 public String convert(@RequestParam double num, @RequestParam String currency, Model model) {  
 double result = switch (currency) {  
 case "usd" -> num / 90;  
 case "eur" -> num / 100;  
 default -> 0;  
 };  
 model.addAttribute("result", result);  
 return "ConvPage";  
 }  
}

**Метод showCalculator()** обрабатывает GET-запрос на путь /home и отображает страницу с калькулятором main.html.

**Метод calculate()** обрабатывает POST-запрос с данными для выполнения арифметической операции. Принимает три параметра через @RequestParam: два числа (num1 и num2) и операцию (operation). В зависимости от операции вычисляет результат и передает его в модель, чтобы отобразить на странице result.html.

**Метод converter()** обрабатывает GET-запрос на путь /home/converter для конвертации валют. Принимает три параметра: валюты для конвертации (val1 и val2) и сумму для конвертации (amount). Вычисляет результат конвертации на основе предустановленных курсов валют и отображает его на странице converter.html, через метод **getConversionRate()**, который возвращает курс обмена между валютами, которые указаны в запросе.

**Метод showCalculator()** обрабатывает GET-запрос на путь / и отображает страницу с калькулятором CalcPage.html.

**Метод calculate()** обрабатывает POST-запрос с данными для выполнения арифметической операции. Принимает три параметра через @RequestParam: два числа (num1 и num2) и операцию (operation) и выводит на этой странице CalcPage.html..

**Метод showConverter()** обрабатывает GET-запрос на путь /conv и отображает страницу с калькулятором ConvPage.html.

**Метод convert()** обрабатывает GET-запрос на путь /home/converter для конвертации валют. Принимает два параметра: валюта для конвертации (currency) и сумму для конвертации (num). Вычисляет результат конвертации на основе предустановленных курсов валют и отображает его на странице ConvPage.html.

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - CalcPage.html

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - CalcPage.html с результатом

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - ConvPage.html

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - ConvPage.html с результатом

Вывод: в ходе выполнения работы было создано приложение, содержащее 2 страницы «Калькулятор» и «Конвертер валют», реализован функционал базового калькулятора, а также конвертора валют. Использованы аннотации @GetMapping, @PostMapping и @RequestParam.

# Практическая работа №2

Цель: разработать веб-приложение на spring с использованием паттерна MVC. В приложении должны быть созданы: 2 модели, с валидацией полей, CRUD операции для моделей, фильтрацию и сортировку.

Ход работы:

Создаем проект spring boot и добавляем зависимости как в предыдущей практической работе.

В проекте создаем 4 папки по паттерну MVC: Models, Controllers, Repositories, Services.

Создаем базовые модели Book и Author, для них создаем репозитории, сервисы и контроллеры.

Код программы:

1. AuthorController

package com.example.mvc.Controllers;  
  
import java.util.\*;  
import java.util.concurrent.atomic.AtomicLong;  
import java.util.stream.Collectors;  
  
import com.example.mvc.Models.Author;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.validation.BindingResult;  
import org.springframework.validation.annotation.Validated;  
import org.springframework.web.bind.annotation.\*;  
  
import jakarta.validation.Valid;  
  
@Controller  
@RequestMapping("/authors")  
@Validated  
public class AuthorController {  
 private final List<Author> authors = new ArrayList<>();  
 private final AtomicLong idCounter = new AtomicLong(1);  
  
 @GetMapping("/")  
 public String getAllAuthors(  
 @RequestParam(required = false) String q,  
 @RequestParam(required = false) String filter,  
 @RequestParam(required = false) String sort,  
 Model model) {  
 List<Author> result = new ArrayList<>(authors);  
  
 if (q != null && !q.trim().isEmpty()) {  
 String qq = q.trim().toLowerCase();  
 result = result.stream().filter(a -> {  
 if (a.getName() != null && a.getName().toLowerCase().contains(qq)) return true;  
 if (a.getSurname() != null && a.getSurname().toLowerCase().contains(qq)) return true;  
 if (a.getLastname() != null && a.getLastname().toLowerCase().contains(qq)) return true;  
 // если запрос — число, сравним по id  
 try {  
 Long id = Long.*parseLong*(qq);  
 if (Objects.*equals*(a.getId(), id)) return true;  
 } catch (NumberFormatException ignored) {  
 }  
 return false;  
 }).collect(Collectors.*toList*());  
 } else if (filter != null && !filter.trim().isEmpty()) {  
 Map<String, String> filters = Arrays.*stream*(filter.split(","))  
 .map(String::trim)  
 .map(s -> s.split(":", 2))  
 .filter(arr -> arr.length == 2)  
 .collect(Collectors.*toMap*(arr -> arr[0].trim(), arr -> arr[1].trim()));  
  
 result = result.stream().filter(a -> {  
 for (Map.Entry<String, String> e : filters.entrySet()) {  
 String key = e.getKey();  
 String val = e.getValue();  
 boolean match;  
 switch (key) {  
 case "name":  
 match = a.getName() != null && a.getName().toLowerCase().contains(val.toLowerCase());  
 break;  
 case "surname":  
 match = a.getSurname() != null && a.getSurname().toLowerCase().contains(val.toLowerCase());  
 break;  
 case "lastname":  
 match = a.getLastname() != null && a.getLastname().toLowerCase().contains(val.toLowerCase());  
 break;  
 case "id":  
 try {  
 Long id = Long.*parseLong*(val);  
 match = Objects.*equals*(a.getId(), id);  
 } catch (NumberFormatException ex) {  
 match = false;  
 }  
 break;  
 default:  
 match = true;  
 }  
 if (!match) return false;  
 }  
 return true;  
 }).collect(Collectors.*toList*());  
 }  
  
 if (sort != null && !sort.trim().isEmpty()) {  
 Comparator<Author> comparator = null;  
 String[] parts = sort.split(",");  
 for (String part : parts) {  
 String[] p = part.trim().split(":", 2);  
 String field = p[0].trim();  
 String dir = p.length > 1 ? p[1].trim().toLowerCase() : "asc";  
 Comparator<Author> c = null;  
 switch (field) {  
 case "name":  
 c = Comparator.*comparing*(a -> Optional.*ofNullable*(a.getName()).orElse(""), String.*CASE\_INSENSITIVE\_ORDER*);  
 break;  
 case "surname":  
 c = Comparator.*comparing*(a -> Optional.*ofNullable*(a.getSurname()).orElse(""), String.*CASE\_INSENSITIVE\_ORDER*);  
 break;  
 case "lastname":  
 c = Comparator.*comparing*(a -> Optional.*ofNullable*(a.getLastname()).orElse(""), String.*CASE\_INSENSITIVE\_ORDER*);  
 break;  
 case "id":  
 c = Comparator.*comparing*(a -> Optional.*ofNullable*(a.getId()).orElse(-1L));  
 break;  
 default:  
 // skip  
 }  
 if (c != null) {  
 if ("desc".equals(dir)) c = c.reversed();  
 comparator = comparator == null ? c : comparator.thenComparing(c);  
 }  
 }  
 if (comparator != null) result.sort(comparator);  
 }  
  
 model.addAttribute("authors", result);  
 return "authorList";  
 }  
  
 @GetMapping("/add")  
 public String addForm(Model model) {  
 model.addAttribute("author", new Author());  
 return "AddAuthor";  
 }  
  
 @PostMapping("/add")  
 public String addAuthorForm(@Valid @ModelAttribute("author") Author author, BindingResult bindingResult, Model model) {  
 if (bindingResult.hasErrors()) {  
 model.addAttribute("author", author);  
 return "AddAuthor";  
 }  
 author.setId(idCounter.getAndIncrement());  
 authors.add(author);  
 return "redirect:/authors/";  
 }  
  
 @PostMapping(value = "/add", consumes = "application/json", produces = "application/json")  
 @ResponseBody  
 public Author addAuthorJson(@Valid @RequestBody Author author) {  
 author.setId(idCounter.getAndIncrement());  
 authors.add(author);  
 return author;  
 }  
  
 @GetMapping("/edit/{id}")  
 public String editForm(@PathVariable Long id, Model model) {  
 Author author = authors.stream()  
 .filter(a -> a.getId().equals(id))  
 .findFirst()  
 .orElseThrow(() -> new NoSuchElementException("Author not found"));  
 model.addAttribute("author", author);  
 return "EditAuthor";  
 }  
  
 @PostMapping("/edit/{id}")  
 public String editAuthorForm(@PathVariable Long id, @Valid @ModelAttribute("author") Author updatedAuthor, BindingResult bindingResult, Model model) {  
 if (bindingResult.hasErrors()) {  
 model.addAttribute("author", updatedAuthor);  
 return "EditAuthor";  
 }  
 Author author = authors.stream()  
 .filter(a -> a.getId().equals(id))  
 .findFirst()  
 .orElseThrow(() -> new NoSuchElementException("Author not found"));  
 author.setName(updatedAuthor.getName());  
 author.setSurname(updatedAuthor.getSurname());  
 author.setLastname(updatedAuthor.getLastname());  
 return "redirect:/authors/";  
 }  
  
 @PutMapping("/edit/{id}")  
 @ResponseBody  
 public Author editAuthor(@PathVariable Long id, @Valid @RequestBody Author updatedAuthor) {  
 Author author = authors.stream()  
 .filter(a -> a.getId().equals(id))  
 .findFirst()  
 .orElseThrow(() -> new NoSuchElementException("Author not found"));  
 author.setName(updatedAuthor.getName());  
 author.setSurname(updatedAuthor.getSurname());  
 author.setLastname(updatedAuthor.getLastname());  
 return author;  
 }  
  
 @DeleteMapping("/delete/{id}")  
 @ResponseBody  
 public void deleteAuthor(@PathVariable Long id) {  
 authors.removeIf(a -> a.getId().equals(id));  
 }  
  
 @PostMapping("/delete/{id}")  
 public String deleteAuthorForm(@PathVariable Long id) {  
 authors.removeIf(a -> a.getId().equals(id));  
 return "redirect:/authors/";  
 }  
}

1. BookController

package com.example.mvc.Controllers;  
  
import java.util.\*;  
import java.util.concurrent.atomic.AtomicLong;  
import java.util.stream.Collectors;  
  
import com.example.mvc.Models.\*;  
  
import jakarta.validation.Valid;  
import org.springframework.stereotype.\*;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.validation.BindingResult;  
import org.springframework.validation.annotation.Validated;  
  
@Controller  
@RequestMapping("/books")  
@Validated  
public class BookController {  
 private final List<Book> books = new ArrayList<>();  
 private final AtomicLong idCounter = new AtomicLong(1);  
  
 @GetMapping("/")  
 public String getAllBooks(  
 @RequestParam(required = false) String q,  
 @RequestParam(required = false) String filter,  
 @RequestParam(required = false) String sort,  
 Model model) {  
 List<Book> result = new ArrayList<>(books);  
  
 if (q != null && !q.trim().isEmpty()) {  
 String qq = q.trim().toLowerCase();  
 result = result.stream().filter(b -> {  
 if (b.getTitle() != null && b.getTitle().toLowerCase().contains(qq)) return true;  
 try {  
 Long aid = Long.*parseLong*(qq);  
 if (Objects.*equals*(b.getAuthor\_id(), aid)) return true;  
 if (Objects.*equals*(b.getId(), aid)) return true;  
 } catch (NumberFormatException ignored) {  
 }  
 try {  
 Integer pc = Integer.*parseInt*(qq);  
 if (Objects.*equals*(b.getPageCount(), pc)) return true;  
 } catch (NumberFormatException ignored) {  
 }  
 return false;  
 }).collect(Collectors.*toList*());  
 } else if (filter != null && !filter.trim().isEmpty()) {  
 Map<String, String> filters = Arrays.*stream*(filter.split(","))  
 .map(String::trim)  
 .map(s -> s.split(":", 2))  
 .filter(arr -> arr.length == 2)  
 .collect(Collectors.*toMap*(arr -> arr[0].trim(), arr -> arr[1].trim()));  
  
 result = result.stream().filter(b -> {  
 for (Map.Entry<String, String> e : filters.entrySet()) {  
 String key = e.getKey();  
 String val = e.getValue();  
 boolean match;  
 switch (key) {  
 case "title":  
 match = b.getTitle() != null && b.getTitle().toLowerCase().contains(val.toLowerCase());  
 break;  
 case "author\_id":  
 try {  
 Long aid = Long.*parseLong*(val);  
 match = Objects.*equals*(b.getAuthor\_id(), aid);  
 } catch (NumberFormatException ex) {  
 match = false;  
 }  
 break;  
 case "pageCount":  
 try {  
 Integer pc = Integer.*parseInt*(val);  
 match = Objects.*equals*(b.getPageCount(), pc);  
 } catch (NumberFormatException ex) {  
 match = false;  
 }  
 break;  
 default:  
 // игнорируем неизвестные поля  
 match = true;  
 }  
 if (!match) return false;  
 }  
 return true;  
 }).collect(Collectors.*toList*());  
 }  
  
 if (sort != null && !sort.trim().isEmpty()) {  
 Comparator<Book> comparator = null;  
 String[] parts = sort.split(",");  
 for (String part : parts) {  
 String[] p = part.trim().split(":", 2);  
 String field = p[0].trim();  
 String dir = p.length > 1 ? p[1].trim().toLowerCase() : "asc";  
 Comparator<Book> c = null;  
 switch (field) {  
 case "title":  
 c = Comparator.*comparing*(b -> Optional.*ofNullable*(b.getTitle()).orElse(""), String.*CASE\_INSENSITIVE\_ORDER*);  
 break;  
 case "author\_id":  
 c = Comparator.*comparing*(b -> Optional.*ofNullable*(b.getAuthor\_id()).orElse(-1L));  
 break;  
 case "pageCount":  
 c = Comparator.*comparing*(b -> Optional.*ofNullable*(b.getPageCount()).orElse(-1));  
 break;  
 case "id":  
 c = Comparator.*comparing*(b -> Optional.*ofNullable*(b.getId()).orElse(-1L));  
 break;  
 default:  
 }  
 if (c != null) {  
 if ("desc".equals(dir)) c = c.reversed();  
 comparator = comparator == null ? c : comparator.thenComparing(c);  
 }  
 }  
 if (comparator != null) {  
 result.sort(comparator);  
 }  
 }  
  
 model.addAttribute("books", result);  
 return "bookList";  
 }  
  
 @GetMapping("/add")  
 public String addForm(Model model) {  
 model.addAttribute("book", new Book());  
 return "AddBook";  
 }  
  
 @PostMapping(value = "/add")  
 public String addBookForm(@Valid @ModelAttribute("book") Book book, BindingResult bindingResult, Model model) {  
 if (bindingResult.hasErrors()) {  
 model.addAttribute("book", book);  
 return "AddBook";  
 }  
 book.setId(idCounter.getAndIncrement());  
 books.add(book);  
 return "redirect:/books/";  
 }  
  
 @PostMapping(value = "/add", consumes = "application/json", produces = "application/json")  
 @ResponseBody  
 public Book addBookJson(@Valid @RequestBody Book book) {  
 book.setId(idCounter.getAndIncrement());  
 books.add(book);  
 return book;  
 }  
  
 @GetMapping("/edit/{id}")  
 public String editForm(@PathVariable Long id, Model model) {  
 Book book = books.stream()  
 .filter(b -> b.getId().equals(id))  
 .findFirst()  
 .orElseThrow(() -> new NoSuchElementException("Book not found"));  
 model.addAttribute("book", book);  
 return "EditBook";  
 }  
  
 @PostMapping("/edit/{id}")  
 public String editBookForm(@PathVariable Long id, @Valid @ModelAttribute("book") Book updatedBook, BindingResult bindingResult, Model model) {  
 if (bindingResult.hasErrors()) {  
 model.addAttribute("book", updatedBook);  
 return "EditBook";  
 }  
 Book book = books.stream()  
 .filter(b -> b.getId().equals(id))  
 .findFirst()  
 .orElseThrow(() -> new NoSuchElementException("Book not found"));  
 book.setTitle(updatedBook.getTitle());  
 book.setAuthor\_id(updatedBook.getAuthor\_id());  
 book.setPageCount(updatedBook.getPageCount());  
 return "redirect:/books/";  
 }  
  
 @PutMapping("/edit/{id}")  
 @ResponseBody  
 public Book editBook(@PathVariable Long id, @Valid @RequestBody Book updatedBook) {  
 Book book = books.stream()  
 .filter(b -> b.getId().equals(id))  
 .findFirst()  
 .orElseThrow(() -> new NoSuchElementException("Book not found"));  
 book.setTitle(updatedBook.getTitle());  
 book.setAuthor\_id(updatedBook.getAuthor\_id());  
 book.setPageCount(updatedBook.getPageCount());  
 return book;  
 }  
  
 @DeleteMapping("/delete/{id}")  
 @ResponseBody  
 public void deleteBook(@PathVariable Long id) {  
 books.removeIf(b -> b.getId().equals(id));  
 }  
  
 @PostMapping("/delete/{id}")  
 public String deleteBookForm(@PathVariable Long id) {  
 books.removeIf(b -> b.getId().equals(id));  
 return "redirect:/books/";  
 }  
}

1. HomeController

package com.example.mvc.Controllers;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
  
@Controller  
public class HomeController {  
 @GetMapping("/")  
 public String index() {  
 return "index";  
 }  
}

1. Author

package com.example.mvc.Models;  
  
import jakarta.validation.constraints.\*;  
  
public class Author {  
 private Long id;  
  
 @NotBlank(message = "Имя не может быть пустым")  
 @Size(max = 100, message = "Имя не должно превышать 100 символов")  
 private String name;  
  
 @NotBlank(message = "Фамилия не может быть пустой")  
 @Size(max = 100, message = "Фамилия не должна превышать 100 символов")  
 private String surname;  
  
 @Size(max = 100, message = "Отчество не должно превышать 100 символов")  
 private String lastname;  
  
 public Author() {  
 }  
  
 public Author(Long id, String name, String surname, String lastname) {  
 this.id = id;  
 this.name = name;  
 this.surname = surname;  
 this.lastname = lastname;  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
  
 public String getLastname() {  
 return lastname;  
 }  
  
 public void setLastname(String lastname) {  
 this.lastname = lastname;  
 }  
}

1. Book

package com.example.mvc.Models;  
  
import jakarta.validation.constraints.\*;  
  
public class Book {  
 private Long id;  
  
 @NotBlank(message = "Название не может быть пустым")  
 @Size(max = 255, message = "Название не должно превышать 255 символов")  
 private String title;  
  
 @NotNull(message = "Требуется указать id автора")  
 private Long author\_id;  
  
 @NotNull(message = "Требуется указать количество страниц")  
 @Min(value = 1, message = "Количество страниц должно быть не меньше 1")  
 private Integer pageCount;  
  
 public Book() {  
 }  
  
 public Book(Long id, String title, Long author\_id, Integer pageCount) {  
 this.id = id;  
 this.title = title;  
 this.author\_id = author\_id;  
 this.pageCount = pageCount;  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public String getTitle() {  
 return title;  
 }  
  
 public void setTitle(String title) {  
 this.title = title;  
 }  
  
 public Long getAuthor\_id() {  
 return author\_id;  
 }  
  
 public void setAuthor\_id(Long author\_id) {  
 this.author\_id = author\_id;  
 }  
  
 public Integer getPageCount() {  
 return pageCount;  
 }  
  
 public void setPageCount(Integer pageCount) {  
 this.pageCount = pageCount;  
 }  
}

1. AuthorRepo

package com.example.mvc.Repositories;  
  
import java.util.\*;  
import com.example.mvc.Models.\*;  
  
public class AuthorRepo {  
 private final Map<Long, Author> authorStorage = new HashMap<>();  
 private long idCounter = 1;  
  
 public Author save(Author author){  
 if (author.getId() == null){  
 author.setId(idCounter++);  
 }  
 authorStorage.put(author.getId(), author);  
 return author;  
 }  
 public Optional<Author> findById(Long id){  
 return Optional.*ofNullable*(authorStorage.get(id));  
 }  
 public List<Author> findAll(){  
 return new ArrayList<>(authorStorage.values());  
 }  
 public void deleteById(Long id){  
 authorStorage.remove(id);  
 }  
 public void deleteManyById(List<Long> ids){  
 ids.forEach(authorStorage::remove);  
 }  
}

1. BookRepo

package com.example.mvc.Repositories;  
  
import java.util.\*;  
import com.example.mvc.Models.\*;  
  
public class BookRepo {  
 private final Map<Long, Book> bookstorage = new HashMap<>();  
 private long idCounter = 1;  
  
 public Book save(Book book){  
 if (book.getId() == null){  
 book.setId(idCounter++);  
 }  
 bookstorage.put(book.getId(), book);  
 return book;  
 }  
 public Optional<Book> findById(Long id){  
 return Optional.*ofNullable*(bookstorage.get(id));  
 }  
 public List<Book> findAll(){  
 return new ArrayList<>(bookstorage.values());  
 }  
 public void deleteById(Long id){  
 bookstorage.remove(id);  
 }  
 public void deleteManyById(List<Long> ids){  
 ids.forEach(bookstorage::remove);  
 }  
}

1. AuthorService

package com.example.mvc.Services;  
  
import com.example.mvc.Models.\*;  
import com.example.mvc.Repositories.\*;  
  
public class AuthorService {  
 private final AuthorRepo authorRepo = new AuthorRepo();  
 public Author create(Author author){  
 return authorRepo.save(author);  
 }  
  
 public Author update(Long id, Author authorupd){  
 Author author = authorRepo.findById(id).orElseThrow();  
 author.setName(authorupd.getName());  
 author.setSurname(authorupd.getSurname());  
 author.setLastname(authorupd.getLastname());  
 return authorRepo.save(author);  
 }  
  
 public void hardDelete(Long id){  
 authorRepo.deleteById(id);  
 }  
}

1. BookService

package com.example.mvc.Services;  
  
import com.example.mvc.Models.\*;  
import com.example.mvc.Repositories.\*;  
  
public class BookService {  
 private final BookRepo bookRepo = new BookRepo();  
 public Book create(Book book){  
 return bookRepo.save(book);  
 }  
  
 public Book update(Long id, Book bookupd){  
 Book book = bookRepo.findById(id).orElseThrow();  
 book.setTitle(bookupd.getTitle());  
 book.setAuthor\_id(bookupd.getAuthor\_id());  
 return bookRepo.save(book);  
 }  
  
 public void hardDelete(Long id){  
 bookRepo.deleteById(id);  
 }  
}

AuthorController

Метод getAllAuthors()

Обрабатывает GET-запрос на путь /authors/. Метод отображает список всех авторов. Позволяет выполнять поиск по имени, фамилии или id автора, фильтрацию по параметрам и сортировку результатов.

Метод addForm()

Возвращает HTML-страницу формы для добавления нового автора.

Метод addAuthorForm()

Обрабатывает POST-запрос с формы добавления автора. Проверяет корректность данных. Если данные верны — сохраняет автора и перенаправляет на список авторов.

Метод addAuthorJson()

Позволяет добавлять автора через JSON-запрос. Возвращает добавленного автора.

Метод editForm()

Возвращает форму редактирования данных автора по его ID.

Метод editAuthorForm()

Обрабатывает POST-запрос с формы редактирования автора. Проверяет введённые данные и сохраняет изменения.

Метод editAuthor()

Обрабатывает PUT-запрос (через JSON). Обновляет данные автора по ID и возвращает обновлённого автора.

Метод deleteAuthor()

Обрабатывает DELETE-запрос, удаляет автора по ID без возвращения страницы.

Метод deleteAuthorForm()

Обрабатывает POST-запрос с формы, удаляет автора и перенаправляет на список авторов.

BookController

Метод getAllBooks()

Обрабатывает GET-запрос на путь /books/. Отображает список всех книг с возможностью поиска, фильтрации и сортировки по различным полям.

Метод addForm()

Возвращает форму для добавления новой книги.

Метод addBookForm()

Обрабатывает POST-запрос с формы добавления книги. Проверяет данные и сохраняет книгу, если ошибок нет.

Метод addBookJson()

Позволяет добавить книгу через JSON-запрос и возвращает добавленную запись.

Метод editForm()

Возвращает форму для редактирования данных книги по ID.

Метод editBookForm()

Обрабатывает POST-запрос с формы редактирования книги. Обновляет данные книги после проверки.

Метод editBook()

Обрабатывает PUT-запрос (JSON). Обновляет данные книги по ID и возвращает обновлённую запись.

Метод deleteBook()

Обрабатывает DELETE-запрос, удаляет книгу по её ID.

Метод deleteBookForm()

Обрабатывает POST-запрос с формы удаления книги и перенаправляет пользователя на список книг.

HomeController

Метод index()

Обрабатывает GET-запрос на путь /. Возвращает главную страницу приложения (index.html).

AuthorService

Метод create()

Создаёт нового автора через вызов метода save() в репозитории.

Метод update()

Обновляет данные существующего автора по ID и сохраняет изменения в репозитории.

Метод hardDelete()

Удаляет автора по ID без возможности восстановления.

BookService

Метод create()

Создаёт новую книгу, используя метод save() из репозитория.

Метод update()

Обновляет данные книги по ID и сохраняет изменения в репозитории.

Метод hardDelete()

Удаляет книгу по ID без возможности восстановления.

AuthorRepo

Метод save()

Сохраняет автора в хранилище. Если ID отсутствует — присваивает новый.

Метод findById()

Находит автора по его ID, возвращает Optional<Author>.

Метод findAll()

Возвращает список всех авторов в хранилище.

Метод deleteById()

Удаляет автора по ID.

Метод deleteManyById()

Удаляет сразу несколько авторов по списку ID.

BookRepo

Метод save()

Сохраняет книгу в хранилище. Если ID не задан, присваивает новый.

Метод findById()

Находит книгу по ID, возвращает Optional<Book>.

Метод findAll()

Возвращает список всех книг в хранилище.

Метод deleteById()

Удаляет книгу по её ID.

Метод deleteManyById()

Удаляет несколько книг по списку ID.

Изображение выглядит как текст, линия, Шрифт, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Страница с кнгиами

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Страница добавления книг

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Окно редактирования книги

Изображение выглядит как текст, линия, Шрифт, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Страница с авторами

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Страница добавления автора

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Страница редактирования автора

Вывод: в ходе практической работы были изучены принципы работы паттерна MVC в spring boot, создано простое приложение с CRUD операциями для моделей Author и Book.