|  |
| --- |
| sМинистерство 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:       Полежайковский 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](#_Toc212775251)

[Практическая работа №2 8](#_Toc212775252)

[Практическая работа №3 25](#_Toc212775253)

[Практическая работа №4 30](#_Toc212775254)

[Практическая работа №5 35](#_Toc212775255)

[Практическая работа №6 39](#_Toc212775256)

# 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:

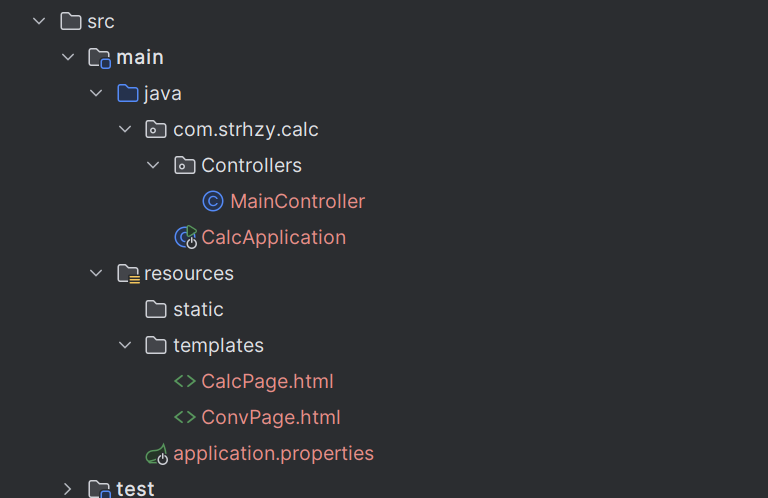


Рисунок - Структура проекта

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.

В моделях с помощью аннотаций @NotBlank, @Size добавляем валидацию, в контроллерах валидируем значения с помощью @Validated.

Также в контроллерах прописываем логику для фильтрации и сортировки.

В репозиториях прописываем логику для хранения данных, в сервисах логику для CRUD.

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.

# Практическая работа №3

Цель работы: изучить принципы работы с базой данных в spring boot с помощью spring boot JPA и hibernate, разработать веб приложение с возможностью взаимодействия с бд.

Ход работы:

Создаем проект и делаем в нем 5 моделей с валидацией, на каждую сервис, репозиторий и контроллер, как в предыдущей практической работе. В данном случае репозитории это интерфейсы которые наследуются от JpaRepository. В JpaRepository содержится основная логика для взаимодействия с БД: чтение, сохранение, удаление данных.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Пример репозитория

Для подключения бд прописываем в файле application.properties нужные данные.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 13 - application.properties

Код программы:

Пример контроллера(AuthorController):

package com.strhzy.dbproj.controllers;  
  
import com.strhzy.dbproj.models.Author;  
import com.strhzy.dbproj.services.AuthorService;  
import com.strhzy.dbproj.services.CountryService;  
import jakarta.validation.Valid;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.validation.BindingResult;  
import org.springframework.web.bind.annotation.\*;  
  
@Controller  
@RequestMapping("/authors")  
public class AuthorController {  
 private final AuthorService service;  
 private final CountryService countryService;  
  
 public AuthorController(AuthorService service, CountryService countryService) {  
 this.service = service;  
 this.countryService = countryService;  
 }  
  
 @GetMapping  
 public String list(Model model) {  
 model.addAttribute("authors", service.findAll());  
 return "authorlist";  
 }  
  
 @GetMapping("/add")  
 public String createForm(Model model) {  
 model.addAttribute("author", new Author());  
 model.addAttribute("countries", countryService.findAll());  
 return "authorform";  
 }  
  
 @PostMapping("/add")  
 public String create(@Valid @ModelAttribute("author") Author author, BindingResult br, Model model) {  
 if (br.hasErrors()) {  
 model.addAttribute("countries", countryService.findAll());  
 return "authorform";  
 }  
 service.save(author);  
 return "redirect:/authors";  
 }  
  
 @GetMapping("/edit/{id}")  
 public String editForm(@PathVariable Long id, Model model) {  
 service.findById(id).ifPresent(a -> model.addAttribute("author", a));  
 model.addAttribute("countries", countryService.findAll());  
 return "authorform";  
 }  
  
 @PostMapping("/edit/{id}")  
 public String edit(@PathVariable Long id, @Valid @ModelAttribute("author") Author author, BindingResult br, Model model) {  
 if (br.hasErrors()) {  
 model.addAttribute("countries", countryService.findAll());  
 return "authorform";  
 }  
 author.setId(id);  
 service.save(author);  
 return "redirect:/authors";  
 }  
  
 @PostMapping("/delete/{id}")  
 public String delete(@PathVariable Long id) {  
 service.deleteById(id);  
 return "redirect:/authors";  
 }  
}

Пример модели(Book):

package com.strhzy.dbproj.models;  
  
  
import jakarta.persistence.\*;  
import jakarta.validation.constraints.Min;  
import jakarta.validation.constraints.NotBlank;  
import lombok.\*;  
import org.hibernate.annotations.\*;  
  
@Entity  
@Data  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
public class Book {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private long id;  
  
 @NotBlank(message = "Название книги не может быть пустым")  
 private String title;  
  
 @Min(value = 1, message = "Количество страниц должно быть больше 0")  
 private Integer pages;  
 @ManyToOne  
 @JoinColumn(name = "author\_id")  
 @OnDelete(action = OnDeleteAction.*SET\_NULL*)  
 private Author author;  
 @ManyToOne  
 @JoinColumn(name = "genre\_id")  
 @OnDelete(action = OnDeleteAction.*SET\_NULL*)  
 private Genre genre;  
 @ManyToOne  
 @JoinColumn(name = "year\_id")  
 @OnDelete(action = OnDeleteAction.*SET\_NULL*)  
 private Year year;  
 @ManyToOne  
 @JoinColumn(name = "country\_id")  
 @OnDelete(action = OnDeleteAction.*SET\_NULL*)  
 private Country country;  
}

Пример сервиса(BookService):

package com.strhzy.dbproj.services;  
  
import com.strhzy.dbproj.models.Book;  
import com.strhzy.dbproj.repositories.BookRepository;  
import org.springframework.stereotype.Service;  
  
import java.util.Comparator;  
import java.util.List;  
import java.util.Optional;  
import java.util.stream.Collectors;  
  
@Service  
public class BookService {  
 private final BookRepository repository;  
  
 public BookService(BookRepository repository) {  
 this.repository = repository;  
 }  
  
 public List<Book> findAll() {  
 return repository.findAll();  
 }  
  
 public Optional<Book> findById(Long id) {  
 return repository.findById(id);  
 }  
  
 public Book save(Book book) {  
 return repository.save(book);  
 }  
  
 public void deleteById(Long id) {  
 repository.deleteById(id);  
 }  
  
 public List<Book> findAllFiltered(Long authorId, Long genreId, Long yearId, Long countryId, String sort) {  
 List<Book> all = repository.findAll();  
  
 List<Book> filtered = all.stream().filter(b -> {  
 if (authorId != null) {  
 if (b.getAuthor() == null || b.getAuthor().getId() != authorId) return false;  
 }  
 if (genreId != null) {  
 if (b.getGenre() == null || b.getGenre().getId() != genreId) return false;  
 }  
 if (yearId != null) {  
 if (b.getYear() == null || b.getYear().getId() != yearId) return false;  
 }  
 if (countryId != null) {  
 if (b.getCountry() == null || b.getCountry().getId() != countryId) return false;  
 }  
 return true;  
 }).collect(Collectors.*toList*());  
  
 Comparator<Book> comparator;  
 if ("title".equals(sort)) {  
 comparator = Comparator.*comparing*(Book::getTitle, Comparator.*nullsLast*(String::compareToIgnoreCase));  
 } else if ("pages".equals(sort)) {  
 comparator = Comparator.*comparing*(b -> b.getPages() == null ? 0 : b.getPages());  
 } else {  
 comparator = Comparator.*comparingLong*(Book::getId);  
 }  
  
 return filtered.stream().sorted(comparator).collect(Collectors.*toList*());  
 }  
}

Результат работы:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Страница с книгами

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Форма добавления книг

Вывод: в ходе практической работы были изучены принципы работы с базой данных в spring boot с помощью spring boot JPA и hibernate, разработано веб приложение с подключенной к ней базой данных postgresql.

# Практическая работа №4

Цель работы: реализовать авторизацию и регистрацию в проекте Java Spring Boot, реализовать разные роли с доступом к определенным страницам, хэширование пароля.

Ход работы:

Для основы берем предыдущую практическую работу. Добавляем в проект зависимость spring-boot-starter-security для работы с авторизацией и регистрацией.

Создаем модель user, с полем role для обозначения роли пользователя, всего их 3: администратор, менеджер, пользователь. Для нее создаем репозиторий и сервис для обработки логики авторизации и регистрации.

Создаем класс SecurityConfig для обработки логики доступа к страницам, обработки хэширования пароля.

Код программы:

1. SecurityConfig

package com.strhzy.dbproj;  
  
import com.strhzy.dbproj.services.UserService;  
import jakarta.servlet.http.Cookie;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.http.HttpMethod;  
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.web.SecurityFilterChain;  
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;  
  
@Configuration  
public class SecurityConfig {  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 @Bean  
 public DaoAuthenticationProvider authProvider(UserService userService, PasswordEncoder passwordEncoder) {  
 DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();  
 authProvider.setUserDetailsService(userService);  
 authProvider.setPasswordEncoder(passwordEncoder);  
 return authProvider;  
 }  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http  
 .formLogin(form -> form  
 .loginPage("/auth/login")  
 .successHandler(authenticationSuccessHandler())  
 .permitAll()  
 )  
 .logout(logout -> logout  
 .logoutUrl("/auth/logout")  
 .logoutSuccessUrl("/")  
 .deleteCookies("username", "role")  
 .invalidateHttpSession(true)  
 .permitAll()  
 )  
 .authorizeHttpRequests(auth -> auth  
 .requestMatchers("/", "/auth/login", "/auth/register", "/css/\*\*", "/js/\*\*").permitAll()  
 .requestMatchers(HttpMethod.*GET*,  
 "/books/\*\*",  
 "/authors/\*\*",  
 "/genres/\*\*",  
 "/countries/\*\*",  
 "/years/\*\*")  
 .hasAnyRole("USER", "MANAGER", "ADMIN")  
 .requestMatchers(HttpMethod.*GET*, "/books/add", "/books/edit/\*\*")  
 .hasAnyRole("MANAGER", "ADMIN")  
 .requestMatchers(HttpMethod.*GET*, "/admin").hasAnyRole("MANAGER", "ADMIN")  
 .requestMatchers(HttpMethod.*GET*, "/admin/users", "/admin/edit/\*\*").hasRole("ADMIN")  
 .requestMatchers(HttpMethod.*POST*, "/admin/edit/\*\*").hasRole("ADMIN")  
 .requestMatchers(HttpMethod.*GET*,  
 "/authors/add", "/authors/edit/\*\*",  
 "/genres/add", "/genres/edit/\*\*",  
 "/countries/add", "/countries/edit/\*\*",  
 "/years/add", "/years/edit/\*\*")  
 .hasRole("ADMIN")  
 .requestMatchers(HttpMethod.*POST*, "/books/\*\*")  
 .hasAnyRole("MANAGER", "ADMIN")  
 .requestMatchers(HttpMethod.*POST*,  
 "/authors/\*\*",  
 "/genres/\*\*",  
 "/countries/\*\*",  
 "/years/\*\*")  
 .hasRole("ADMIN")  
 .anyRequest().authenticated()  
 );  
 return http.build();  
 }  
  
 @Bean  
 public AuthenticationSuccessHandler authenticationSuccessHandler() {  
 return (request, response, authentication) -> {  
 String username = authentication.getName();  
 String role = authentication.getAuthorities().stream().findFirst().map(Object::toString).orElse("ROLE\_USER");  
 if (role.startsWith("ROLE\_")) role = role.substring(5);  
  
 Cookie cookie = new Cookie("username", username);  
 cookie.setPath("/");  
 cookie.setMaxAge(7 \* 24 \* 60 \* 60);  
 cookie.setHttpOnly(false);  
 response.addCookie(cookie);  
  
 Cookie roleCookie = new Cookie("role", role);  
 roleCookie.setPath("/");  
 roleCookie.setMaxAge(7 \* 24 \* 60 \* 60);  
 roleCookie.setHttpOnly(false);  
 response.addCookie(roleCookie);  
  
 response.sendRedirect("/");  
 };  
 }  
}

1. UserService

package com.strhzy.dbproj.services;  
  
import com.strhzy.dbproj.models.User;  
import com.strhzy.dbproj.repositories.UserRepository;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class UserService implements UserDetailsService {  
  
 private final UserRepository userRepository;  
 private final PasswordEncoder passwordEncoder;  
  
 public UserService(UserRepository userRepository, PasswordEncoder passwordEncoder) {  
 this.userRepository = userRepository;  
 this.passwordEncoder = passwordEncoder;  
 }  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 User user = userRepository.findByUsername(username)  
 .orElseThrow(() -> new UsernameNotFoundException("Пользователь не найден"));  
  
 String role = (user.getRole() == null || user.getRole().isBlank()) ? "USER" : user.getRole();  
  
 return org.springframework.security.core.userdetails.User  
 .*withUsername*(user.getUsername())  
 .password(user.getPassword())  
 .roles(role)  
 .build();  
 }  
  
 public void registerUser(String username, String password) {  
 if (userRepository.findByUsername(username).isPresent()) {  
 throw new RuntimeException("Пользователь уже существует");  
 }  
 User user = new User();  
 user.setUsername(username);  
 user.setPassword(passwordEncoder.encode(password));  
 userRepository.save(user);  
 }  
  
 public User authenticate(String username, String rawPassword) {  
 User user = userRepository.findByUsername(username)  
 .orElseThrow(() -> new RuntimeException("Пользователь не найден"));  
  
 if (!passwordEncoder.matches(rawPassword, user.getPassword())) {  
 throw new RuntimeException("Неверный пароль");  
 }  
 return user;  
 }  
  
 public User updateUserRole(String username, String newRole) {  
 User user = userRepository.findByUsername(username)  
 .orElseThrow(() -> new RuntimeException("Пользователь не найден"));  
 user.setRole(newRole);  
 return userRepository.save(user);  
 }  
  
 public User save(User user) {  
 return userRepository.save(user);  
 }  
  
 public Optional<User> findById(Long id) {  
 return userRepository.findById(id);  
 }  
  
 // Новый метод: получить всех пользователей  
 public List<User> findAll() {  
 return userRepository.findAll();  
 }  
}

1. User

package com.strhzy.dbproj.models;  
  
import jakarta.persistence.\*;  
import lombok.\*;  
  
@Data  
@Entity  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Table(name = "user\_details")  
public class User {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(unique = true)  
 private String username;  
  
 private String password;  
  
 private String role = "USER";  
}

Результат работы:

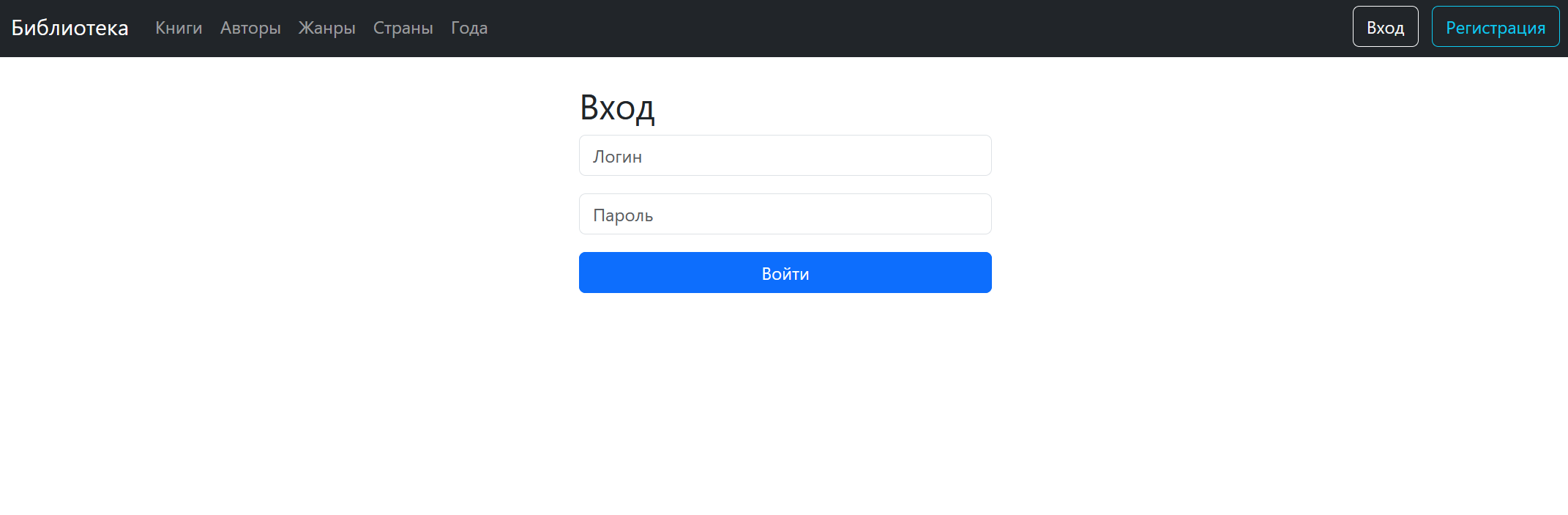


Рисунок - Страница входа

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Пример страницы ошибки

Вывод: в результате практической работы были реализованы авторизация и регистрация в проекте Java Spring Boot, реализованы разные роли с доступом к определенным страницам, хэширование пароля.

# Практическая работа №5

Цель работы: разработать crud-api на spring boot с подключением к базе данных postgre, реализовать валидацию данных для моделей и документацию OpenAPI swagger.

Ход работы:

API на spring строится по паттерну MVC, создаем 8 моделей по тематике «Магазин электроники»: категория, покупатель, производитель, заказ, сводная модель заказ-товар, товар, отзыв, остаток. В моделях реализуем валидацию через Jakarta.validation. Создаем репозитории и сервисы для работы с моделями.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Структура проекта

Далее необходимо создать контроллеры RestAPI. Для этого используется аннотация Restcontroller. Также в контроллере прописываются аннотации для описания операций и для названия контроллера в swagger и аннотации для валидации параметров.

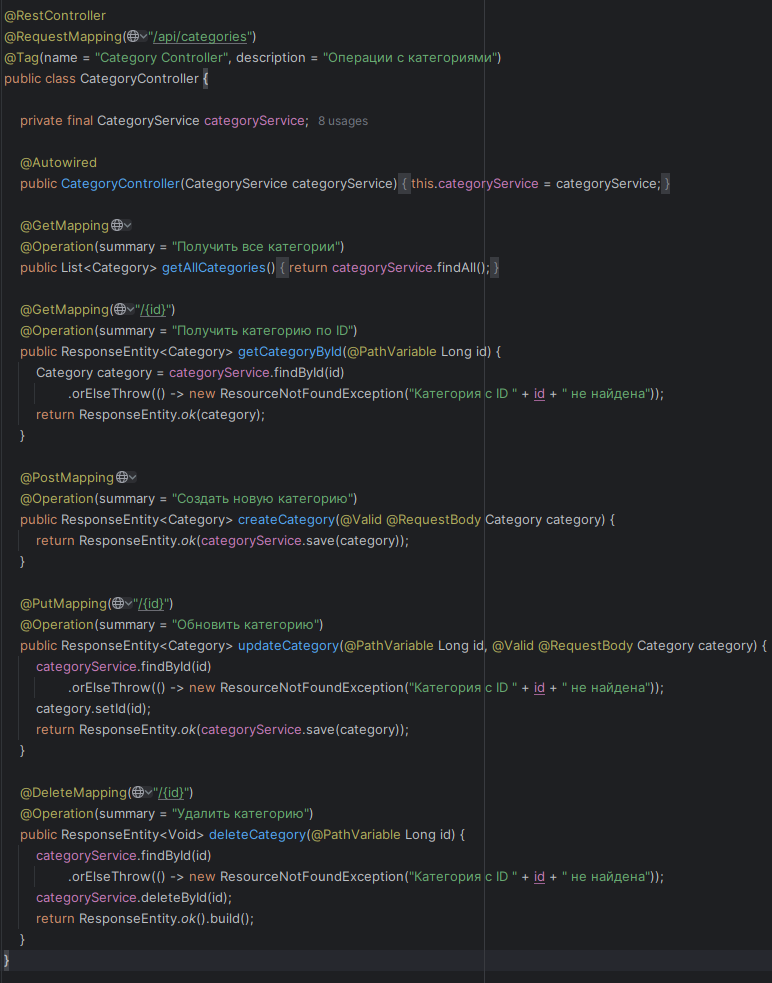


Рисунок - Пример контроллера

Для обработки ошибок создаем классы GlobalExceptionHandler и ResourceNotFoundException.

Для работы с swagger создаем файл SwaggerConfig. В нем прописываются параметры для swagger, такие как заголовок, версия и описание.

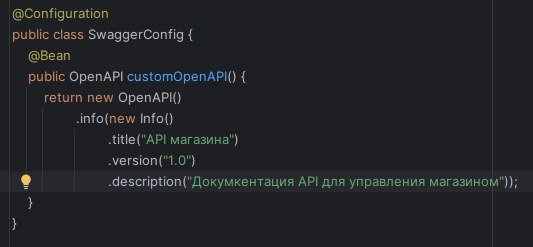


Рисунок – SwaggerConfig

Изображение выглядит как текст, снимок экрана, число, Параллельный

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Страница swagger

Вывод: в результате практической работы была разработана crud-api на spring boot с подключением к базе данных postgre, реализованы валидация данных для моделей и документация OpenAPI swagger.

# Практическая работа №6

Тема: Разработка веб-приложения для интернет магазина электроники.

Описание предметной области: Интернет-магазин электроники и техники.

Цель работы: разработать веб-приложение с подключением к api, реализовать авторизацию, регистрацию, разделение по ролям, CRUD операции от этих ролей.

Ход работы:

Для подключения к api создаем класс HttpService, в котором реализованы базовые методы для http запросов и методов для jwt токенов. API будет использоваться из практической работы №5.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Пример метода HttpService

Данный класс будет использоваться в репозиториях для работы с api.

Далее создаем модели, репозитории и сервисы для моделей.

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Модели проекта

Изображение выглядит как текст, снимок экрана, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Схема базы данных

Распределение по ролям состоит из следующих ролей:

* Администратор – CRUD пользователей,
* Менеджер – CRUD всех остальных таблиц,
* Покупатель – просмотр товаров, оформление заказов.

Код программы:

HttpService.java

public class HttpService {  
  
 private static final String *BASE\_URL* = "http://localhost:8090/api";  
 private static final HttpClient *client* = HttpClient.*newHttpClient*();  
 private static final ObjectMapper *mapper* = new ObjectMapper()  
 .configure(DeserializationFeature.*FAIL\_ON\_UNKNOWN\_PROPERTIES*, false)  
 .registerModule(new JavaTimeModule())  
 .configure(SerializationFeature.*WRITE\_DATES\_AS\_TIMESTAMPS*, false);  
  
 private static String *jwtToken*;  
  
 private HttpService() {}  
  
 private static String toJson(Object obj) {  
 try {  
 return *mapper*.writeValueAsString(obj);  
 } catch (JsonProcessingException e) {  
 throw new RuntimeException("Ошибка сериализации JSON", e);  
 }  
 }  
  
 private static <T> T fromJson(String json, Class<T> clazz) {  
 try {  
 return *mapper*.readValue(json, clazz);  
 } catch (JsonProcessingException e) {  
 throw new RuntimeException("Ошибка десериализации JSON", e);  
 }  
 }  
  
 private static String getSessionToken() {  
 try {  
 ServletRequestAttributes attrs =  
 (ServletRequestAttributes) RequestContextHolder.*getRequestAttributes*();  
 if (attrs == null) return null;  
 HttpServletRequest request = attrs.getRequest();  
 HttpSession session = request.getSession(false);  
 if (session == null) return null;  
 Object tokenObj = session.getAttribute("jwt");  
 if (tokenObj instanceof String token && !token.isBlank()) return token;  
 } catch (Exception ignored) {}  
 return null;  
 }  
  
 public static <T> T get(String path, Class<T> responseType) throws IOException, InterruptedException {  
 String token = *getSessionToken*();  
 if (token == null) token = *jwtToken*;  
  
 HttpRequest.Builder builder = HttpRequest.*newBuilder*()  
 .uri(URI.*create*(*BASE\_URL* + path));  
  
 if (token != null && !token.isBlank()) {  
 builder.header("Authorization", "Bearer " + token);  
 }  
  
 HttpRequest request = builder.GET().build();  
 HttpResponse<String> response = *client*.send(request, HttpResponse.BodyHandlers.*ofString*());  
 return *fromJson*(response.body(), responseType);  
 }  
  
 public static <T, R> R post(String path, T body, Class<R> responseType) throws IOException, InterruptedException {  
 String token = *getSessionToken*();  
 if (token == null) token = *jwtToken*;  
  
 String jsonBody = *toJson*(body);  
 HttpRequest.Builder builder = HttpRequest.*newBuilder*()  
 .uri(URI.*create*(*BASE\_URL* + path))  
 .header("Content-Type", "application/json");  
  
 if (token != null && !token.isBlank()) {  
 builder.header("Authorization", "Bearer " + token);  
 }  
  
 HttpRequest request = builder.POST(HttpRequest.BodyPublishers.*ofString*(jsonBody)).build();  
 HttpResponse<String> response = *client*.send(request, HttpResponse.BodyHandlers.*ofString*());  
 return *fromJson*(response.body(), responseType);  
 }  
  
 public static <T, R> R put(String path, T body, Class<R> responseType) throws IOException, InterruptedException {  
 String token = *getSessionToken*();  
 if (token == null) token = *jwtToken*;  
  
 String jsonBody = *toJson*(body);  
 HttpRequest.Builder builder = HttpRequest.*newBuilder*()  
 .uri(URI.*create*(*BASE\_URL* + path))  
 .header("Content-Type", "application/json");  
  
 if (token != null && !token.isBlank()) {  
 builder.header("Authorization", "Bearer " + token);  
 }  
  
 HttpRequest request = builder.PUT(HttpRequest.BodyPublishers.*ofString*(jsonBody)).build();  
 HttpResponse<String> response = *client*.send(request, HttpResponse.BodyHandlers.*ofString*());  
 return *fromJson*(response.body(), responseType);  
 }  
  
 public static <T, R> R patch(String path, T body, Class<R> responseType) throws IOException, InterruptedException {  
 String token = *getSessionToken*();  
 if (token == null) token = *jwtToken*;  
  
 String jsonBody = *toJson*(body);  
 HttpRequest.Builder builder = HttpRequest.*newBuilder*()  
 .uri(URI.*create*(*BASE\_URL* + path))  
 .header("Content-Type", "application/json");  
  
 if (token != null && !token.isBlank()) {  
 builder.header("Authorization", "Bearer " + token);  
 }  
  
 HttpRequest request = builder.method("PATCH", HttpRequest.BodyPublishers.*ofString*(jsonBody)).build();  
 HttpResponse<String> response = *client*.send(request, HttpResponse.BodyHandlers.*ofString*());  
 return *fromJson*(response.body(), responseType);  
 }  
  
 public static <R> R delete(String path, Class<R> responseType) throws IOException, InterruptedException {  
 String token = *getSessionToken*();  
 if (token == null) token = *jwtToken*;  
  
 HttpRequest.Builder builder = HttpRequest.*newBuilder*()  
 .uri(URI.*create*(*BASE\_URL* + path));  
  
 if (token != null && !token.isBlank()) {  
 builder.header("Authorization", "Bearer " + token);  
 }  
  
 HttpRequest request = builder.DELETE().build();  
 HttpResponse<String> response = *client*.send(request, HttpResponse.BodyHandlers.*ofString*());  
 return *fromJson*(response.body(), responseType);  
 }  
  
 public static void setToken(String token) {  
 *jwtToken* = token;  
 }  
  
 public static void clearToken() {  
 *jwtToken* = null;  
 }  
  
 public static String getToken() {  
 return *jwtToken*;  
 }  
  
 public static boolean isAuthenticated() {  
 String token = *getSessionToken*();  
 if (token == null) token = *jwtToken*;  
 return token != null && !token.isBlank();  
 }  
}

Пример контроллера:

@Controller  
public class MainController {  
  
 private void applySessionToken(HttpSession session) {  
 Object token = session.getAttribute("jwt");  
 if (token instanceof String) {  
 HttpService.*setToken*((String) token);  
 } else {  
 HttpService.*clearToken*();  
 }  
 }  
  
 @GetMapping("/")  
 public String index(Model model, HttpSession session) {  
 applySessionToken(session);  
 boolean isAuth = HttpService.*isAuthenticated*();  
 model.addAttribute("isAuthenticated", isAuth);  
 Object user = session.getAttribute("user");  
 if (user instanceof Customer customer) model.addAttribute("customer", customer);  
 model.addAttribute("title", "Главная");  
  
 List<Product> products = new ArrayList<>();  
 try {  
 Product[] arr = HttpService.*get*("/products?limit=6", Product[].class);  
 if (arr != null) products = Arrays.*asList*(arr);  
 } catch (Exception e) {  
 System.*err*.println("Failed to load featured products: " + e.getMessage());  
 }  
 model.addAttribute("products", products);  
 return "index";  
 }  
  
 @GetMapping("/products")  
 public String products(Model model, HttpSession session,  
 @RequestParam(required = false) Long categoryId,  
 @RequestParam(required = false) Long manufacturerId,  
 @RequestParam(required = false) String q,  
 @RequestParam(required = false, defaultValue = "") String sort) {  
 applySessionToken(session);  
 boolean isAuth = HttpService.*isAuthenticated*();  
 model.addAttribute("isAuthenticated", isAuth);  
 Object user = session.getAttribute("user");  
 if (user instanceof Customer customer) model.addAttribute("customer", customer);  
 model.addAttribute("title", "Каталог");  
  
 List<Product> products = new ArrayList<>();  
 List<Category> categories = new ArrayList<>();  
 List<Manufacturer> manufacturers = new ArrayList<>();  
  
 try {  
 List<String> params = new ArrayList<>();  
 if (categoryId != null) params.add("categoryId=" + categoryId);  
 if (manufacturerId != null) params.add("manufacturerId=" + manufacturerId);  
 if (q != null && !q.isBlank()) params.add("q=" + URLEncoder.*encode*(q, StandardCharsets.*UTF\_8*));  
 if (sort != null && !sort.isBlank()) params.add("sort=" + URLEncoder.*encode*(sort, StandardCharsets.*UTF\_8*));  
  
 String path = "/products";  
 if (!params.isEmpty()) path += "?" + String.*join*("&", params);  
  
 Product[] arr = HttpService.*get*(path, Product[].class);  
 if (arr != null) products = Arrays.*asList*(arr);  
 } catch (Exception e) {  
 System.*err*.println("Failed to load products: " + e.getMessage());  
 }  
  
 try {  
 Category[] cArr = HttpService.*get*("/categories", Category[].class);  
 if (cArr != null) categories = Arrays.*asList*(cArr);  
 } catch (Exception e) {  
 System.*err*.println("Failed to load categories: " + e.getMessage());  
 }  
  
 try {  
 Manufacturer[] mArr = HttpService.*get*("/manufacturers", Manufacturer[].class);  
 if (mArr != null) manufacturers = Arrays.*asList*(mArr);  
 } catch (Exception e) {  
 System.*err*.println("Failed to load manufacturers: " + e.getMessage());  
 }  
  
 model.addAttribute("products", products);  
 model.addAttribute("categories", categories);  
 model.addAttribute("manufacturers", manufacturers);  
 model.addAttribute("selectedCategoryId", categoryId);  
 model.addAttribute("selectedManufacturerId", manufacturerId);  
 model.addAttribute("q", q == null ? "" : q);  
 model.addAttribute("sort", sort == null ? "" : sort);  
  
 return "products";  
 }  
}

Пример репозитория

@Repository  
public class CategoryRepository {  
  
 private static final String *BASE\_PATH* = "/categories";  
  
 public List<Category> findAll() {  
 try {  
 return HttpService.*get*(*BASE\_PATH*, List.class);  
 } catch (IOException | InterruptedException e) {  
 throw new RuntimeException("Ошибка при получении списка категорий", e);  
 }  
 }  
  
 public Optional<Category> findById(Long id) {  
 try {  
 Category category = HttpService.*get*(*BASE\_PATH* + "/" + id, Category.class);  
 return Optional.*ofNullable*(category);  
 } catch (IOException | InterruptedException e) {  
 throw new RuntimeException("Ошибка при получении категории с ID: " + id, e);  
 }  
 }  
  
 public Category save(Category category) {  
 try {  
 if (category.getId() == null) {  
 return HttpService.*post*(*BASE\_PATH*, category, Category.class);  
 } else {  
 return HttpService.*put*(*BASE\_PATH* + "/" + category.getId(), category, Category.class);  
 }  
 } catch (IOException | InterruptedException e) {  
 throw new RuntimeException("Ошибка при сохранении категории", e);  
 }  
 }  
  
 public void deleteById(Long id) {  
 try {  
 HttpService.*delete*(*BASE\_PATH* + "/" + id, Void.class);  
 } catch (IOException | InterruptedException e) {  
 throw new RuntimeException("Ошибка при удалении категории с ID: " + id, e);  
 }  
 }  
}

Пример сервиса

@Service  
public class CategoryService {  
  
 @Autowired  
 private CategoryRepository categoryRepository;  
  
 public List<Category> findAll() {  
 return categoryRepository.findAll();  
 }  
  
 public Optional<Category> findById(Long id) {  
 return categoryRepository.findById(id);  
 }  
  
 public Category save(Category category) {  
 return categoryRepository.save(category);  
 }  
  
 public void deleteById(Long id) {  
 categoryRepository.deleteById(id);  
 }  
}

Результат работы программы:

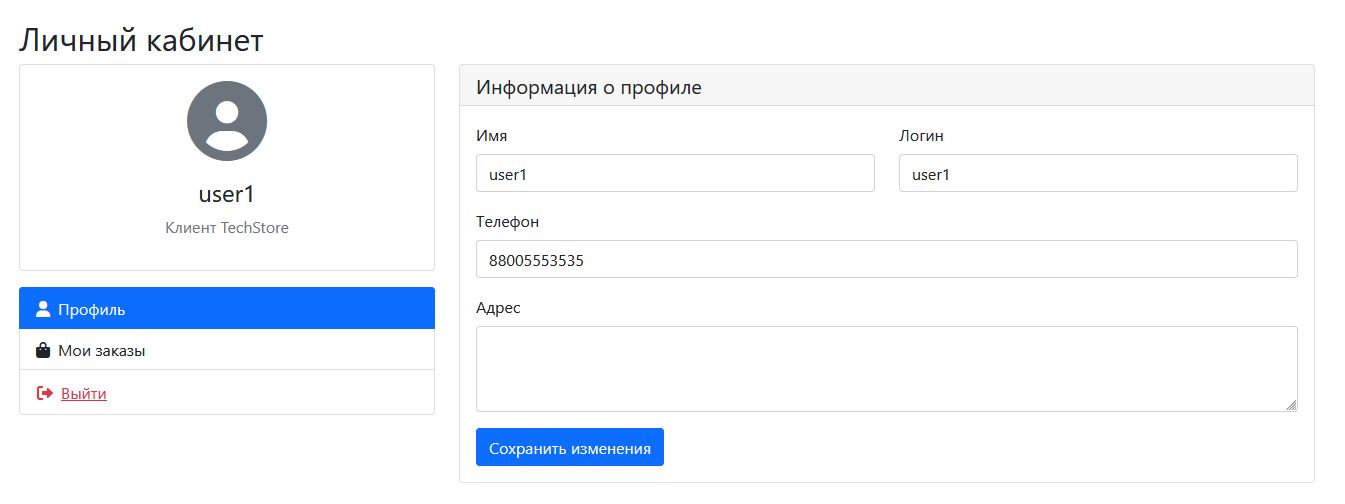


Рисунок - Страница личного кабинета

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Страница заказов

Изображение выглядит как текст, снимок экрана, число, диаграмма

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Страница товаров

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Страница корзины

Изображение выглядит как текст, снимок экрана, число, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Админ панель

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Главная страница панели менеджера

Изображение выглядит как текст, снимок экрана, число, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок - Панель товаров на странице менеджера

Вывод: в результате практической работы было разработано веб-приложение с подключением к api, реализованы авторизацию, регистрацию, разделение по ролям, CRUD операции от этих ролей.