|  |
| --- |
| **МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ** |
| **федеральное государственное бюджетное образовательное учреждение высшего образования «Российский экономический университет имени Г.В. Плеханова»**  **Московский приборостроительный техникум** |

ОТЧЕТ

по учебной практике

УП.04.01

Внедрение и поддержка программного обеспечения .

Профессионального модуля ПМ.02.01

Разработка и администрирование баз данных .

Специальность 09.02.07

Информационные системы и программирование .

Студент Патрикеев Глеб Викторович.

*(фамилия, имя, отчество)*

Группа П50-3-20

Руководитель по практической подготовке от техникума

Пахомов Даниил Александрович.

*(фамилия, имя, отчество)*

«\_\_» \_\_\_\_\_\_ 2023 года

Практическая работа №1

Цель работы: разработать приложение с использованием Spring Framework, включающее 5 страниц: основную страницу, предоставляющую доступ к страницам «Калькулятор» и «Конвертер валют», а также страницы, отображающие результаты вычислений.

Для начала мы создаем контроллер, который будет обрабатывать запросы на главную страницу через HTTP метод GET и адрес корневой URL. В начале каждого контроллера мы применяем аннотацию @Controller, чтобы указать, что данный класс является контроллером.

Затем мы создаем метод с именем home() и помечаем его аннотацией @GetMapping("/"), что позволяет методу обрабатывать GET-запросы на корневой адрес.

Внутри этого метода мы определяем, какое представление должно быть возвращено для отображения на главной странице.

Аналогично, создаем методы calculator и currencyConverter.



Рисунок 1. Контроллер главной страницы.

Контроллер калькулятора начинает свою работу с теми же методами и аннотациями, но добавляет новый метод - calculate().

Аннотация @PostMapping("/Resault") указывает, что этот метод будет обрабатывать POST-запросы и определяет путь, по которому результат этого метода будет доступен.

Аннотация @RequestParam говорит о том, что метод ожидает параметры с именем «num1», «num2», «operator», из запроса, которое соответствует имени элемента в разметке страницы.

Объект Model представляет собой модель Spring, используемую для передачи данных между контроллером и представлением. Таким образом, значение, хранящееся в переменной «result», будет передано на представление под именем «result». Ниже представлена демонстрация того, как это значение будет использоваться.



Рисунок 2. Контроллер калькулятора. (1)

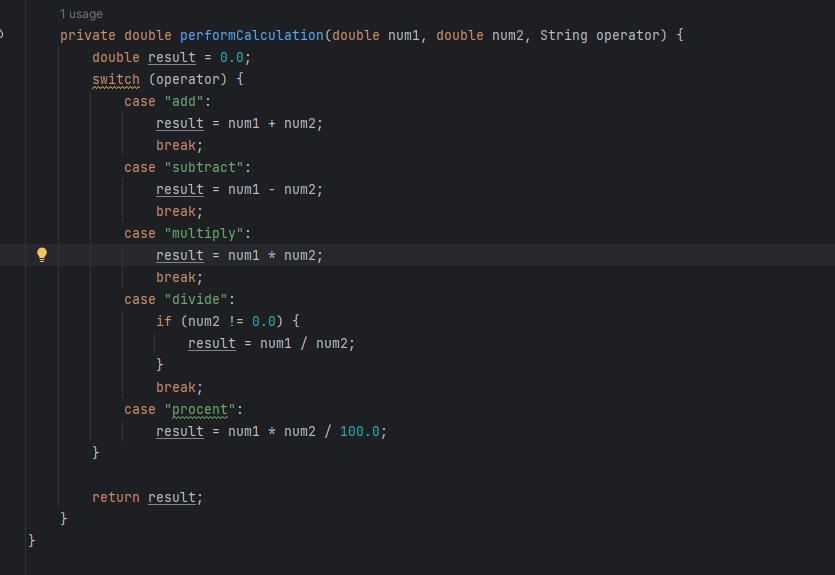


Рисунок 3. Контроллер калькулятора. (2)

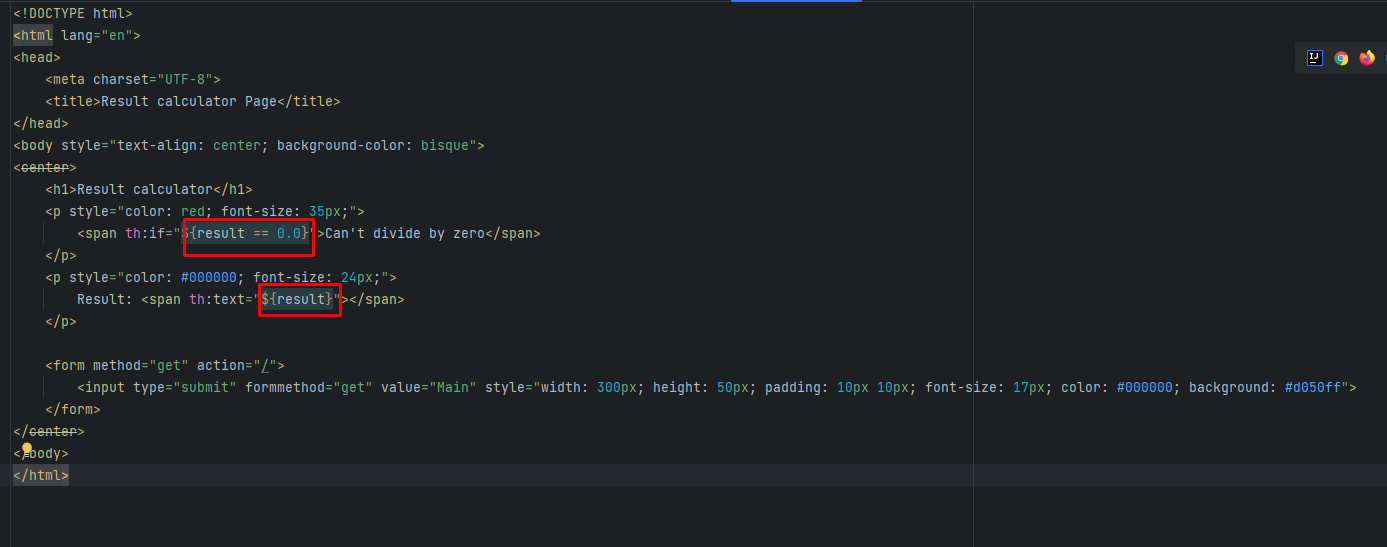


Рисунок 4. Использование result.

В контроллере конвертера валют мы задаем аналогичную логику, но расширяем список параметров, которые принимаем для обработки.

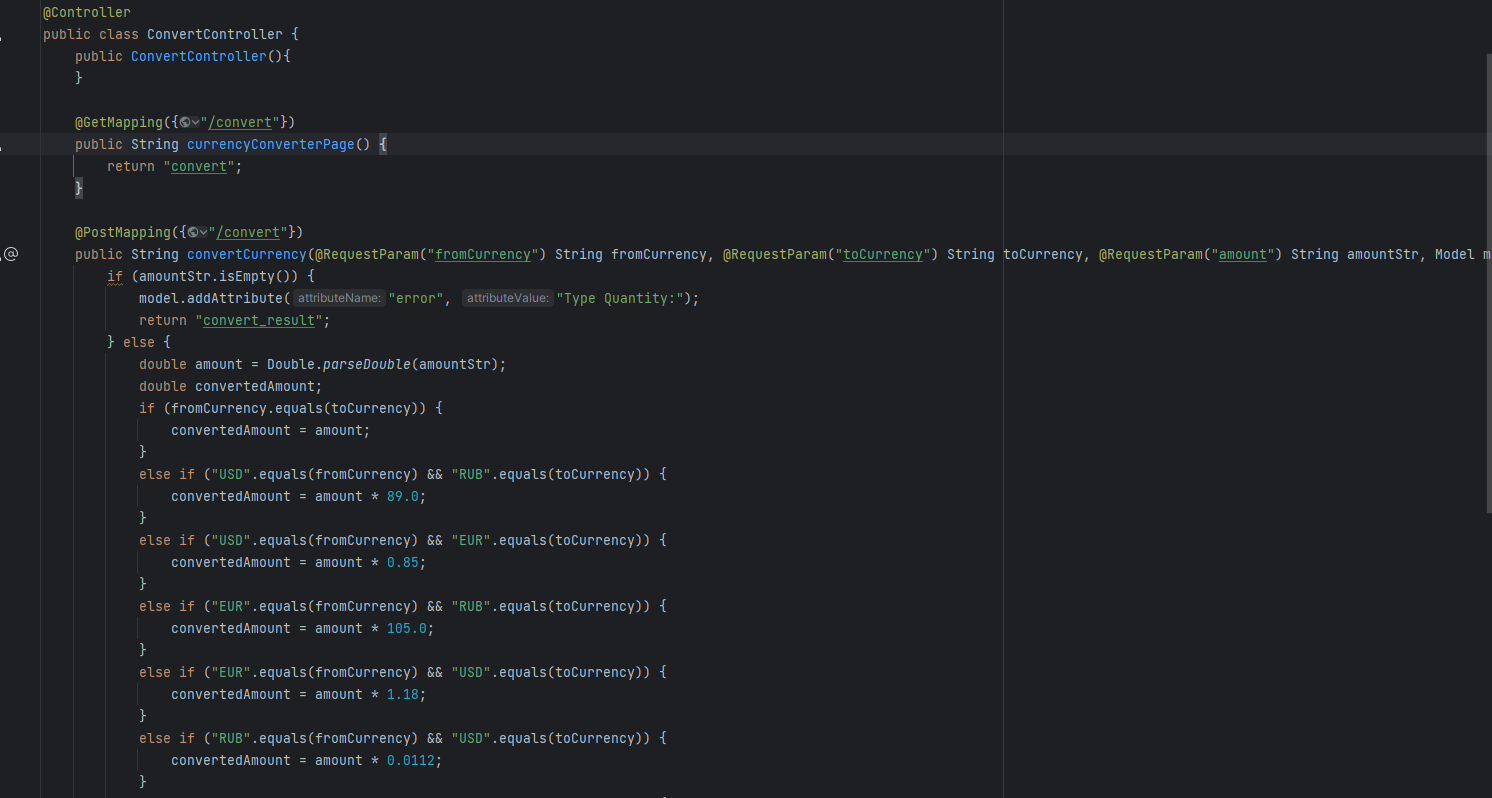
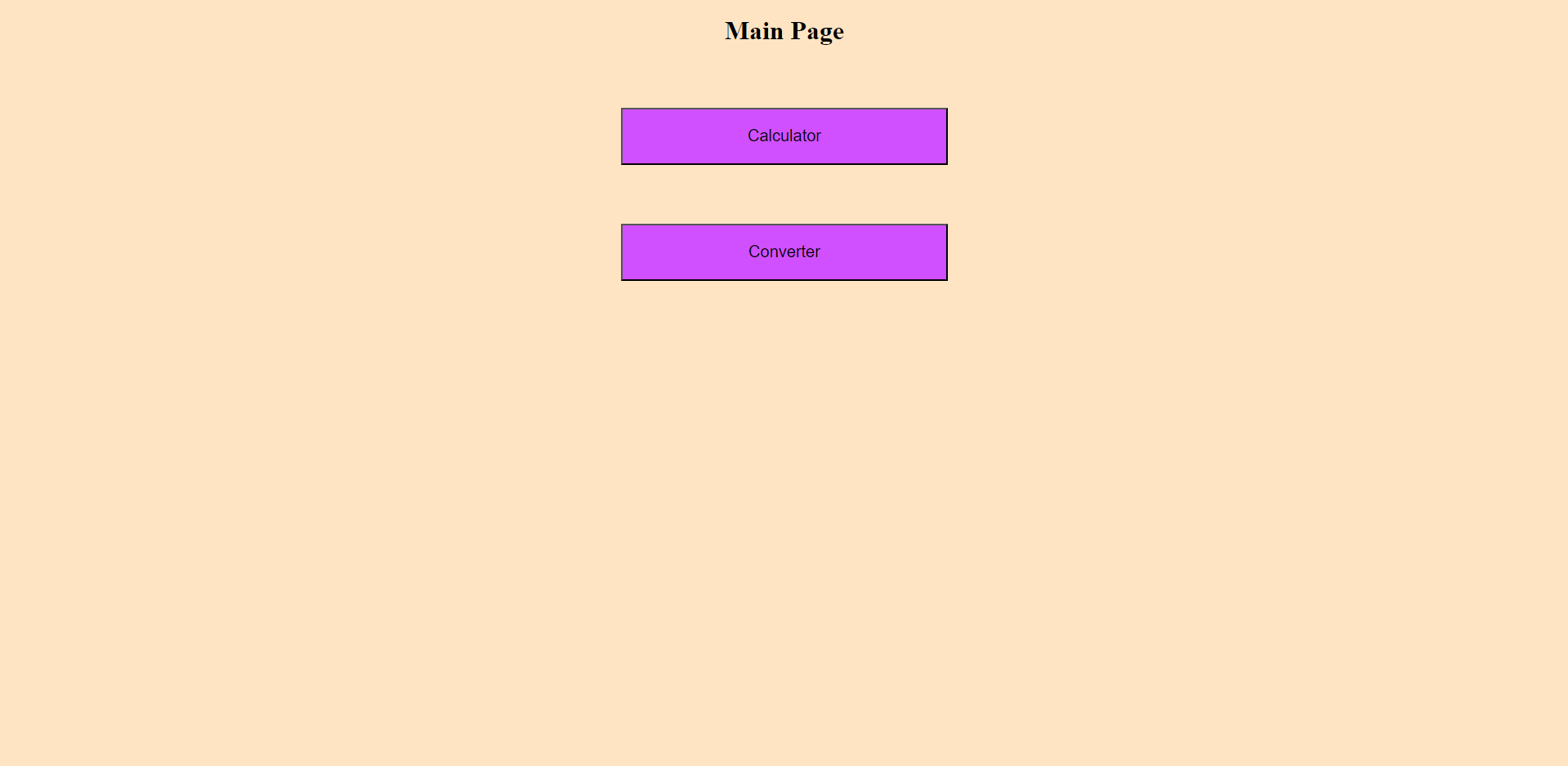
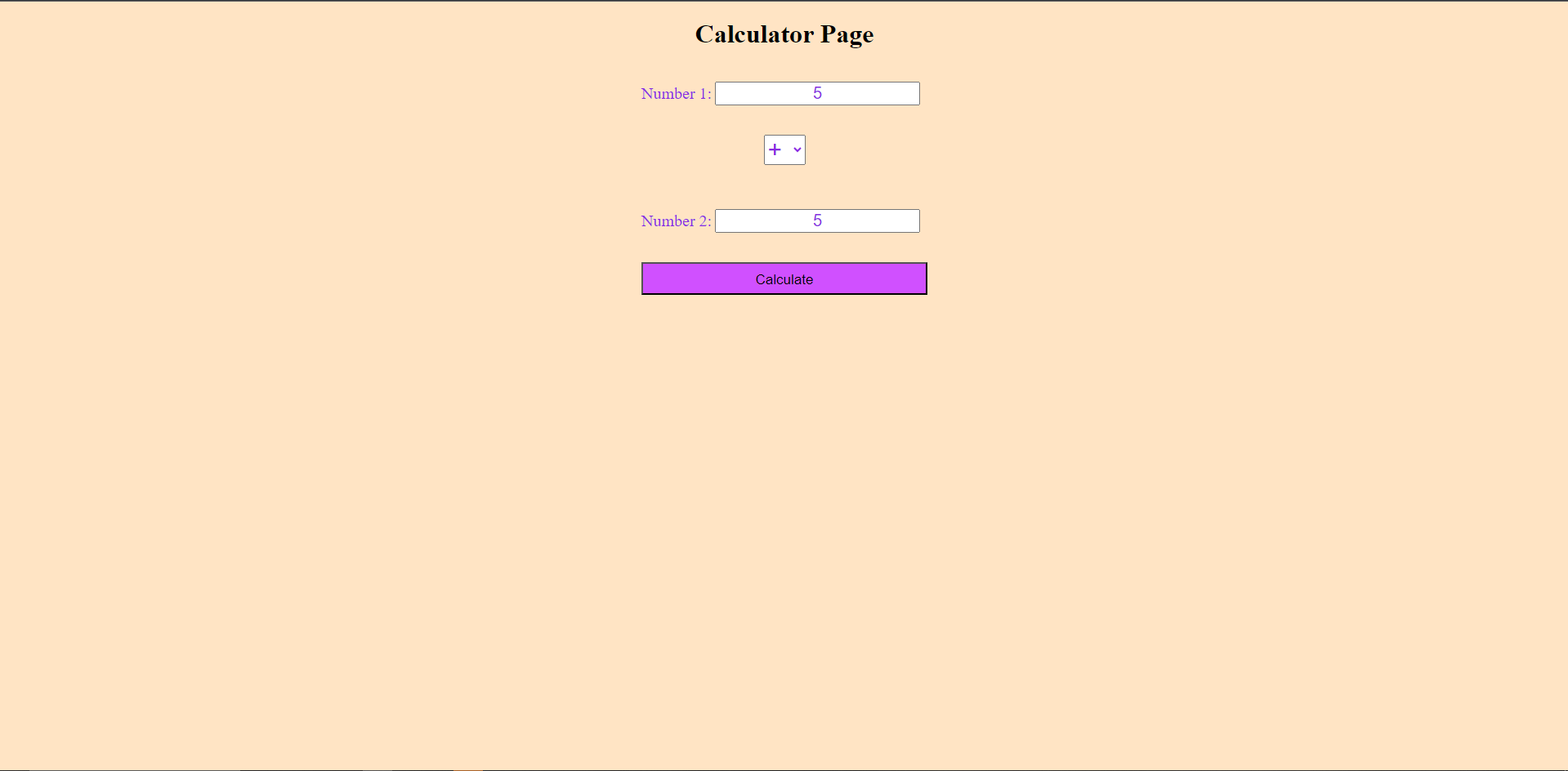
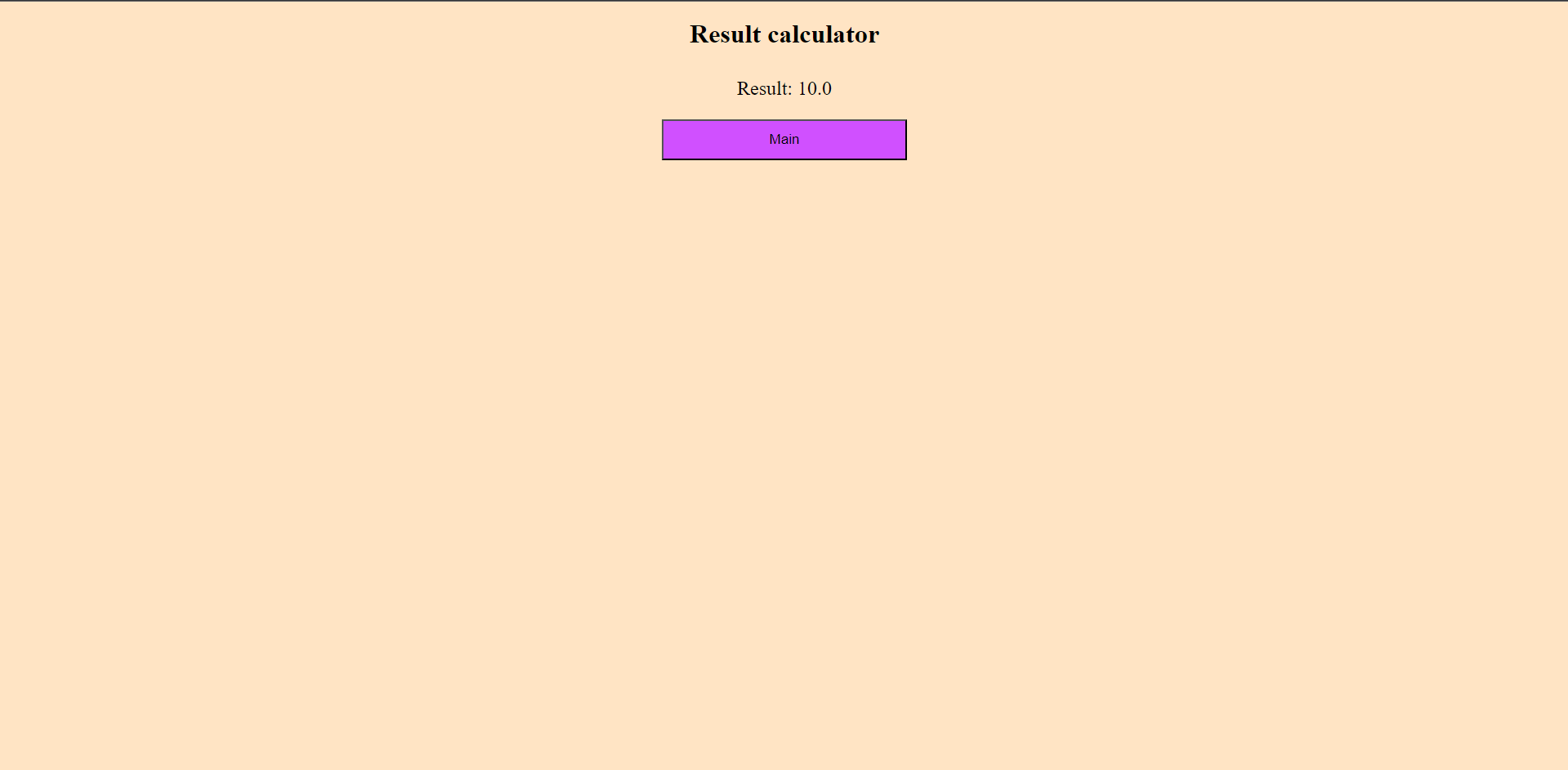


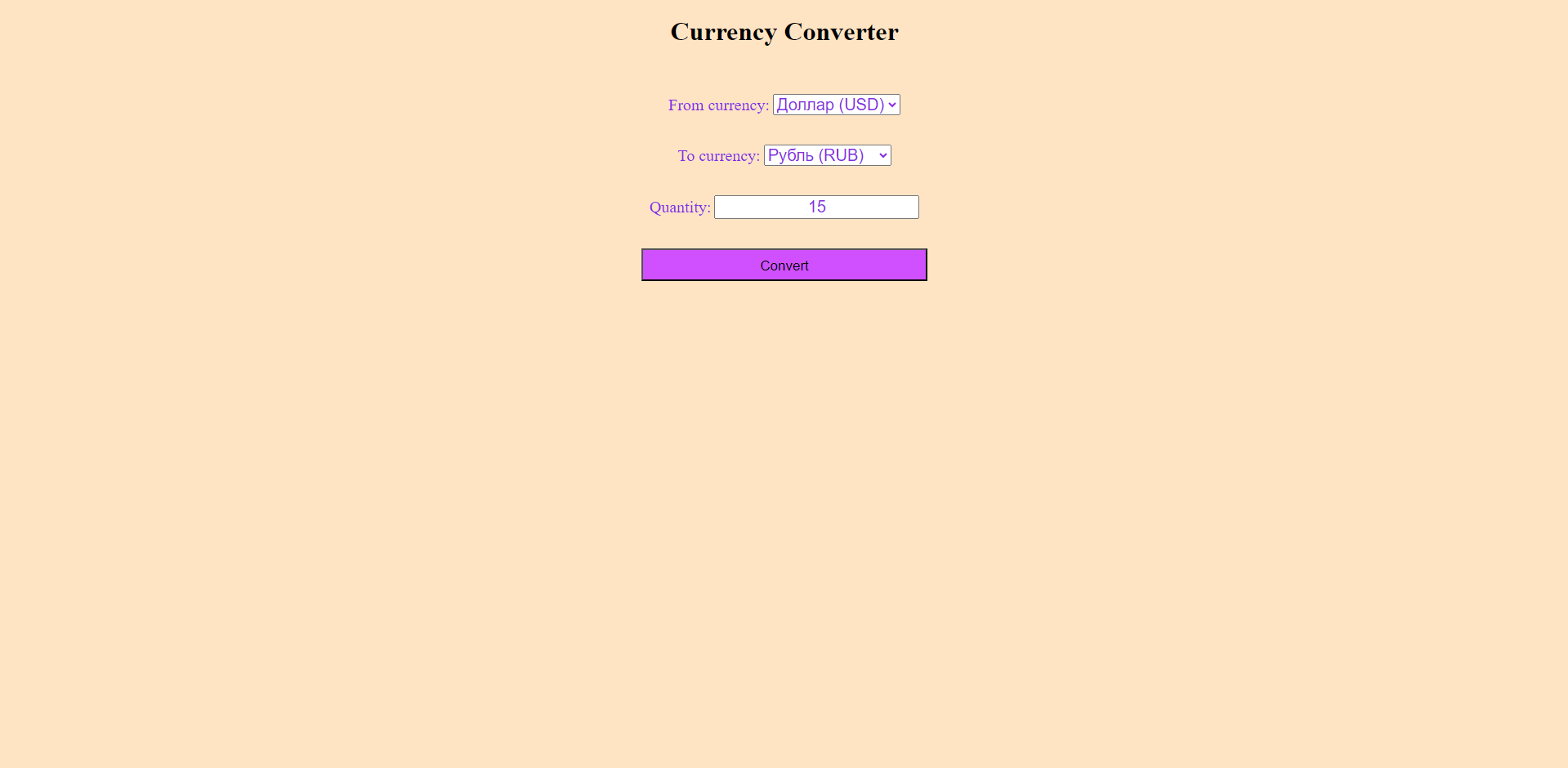
Рисунок 4. Контроллер конвертера валют.

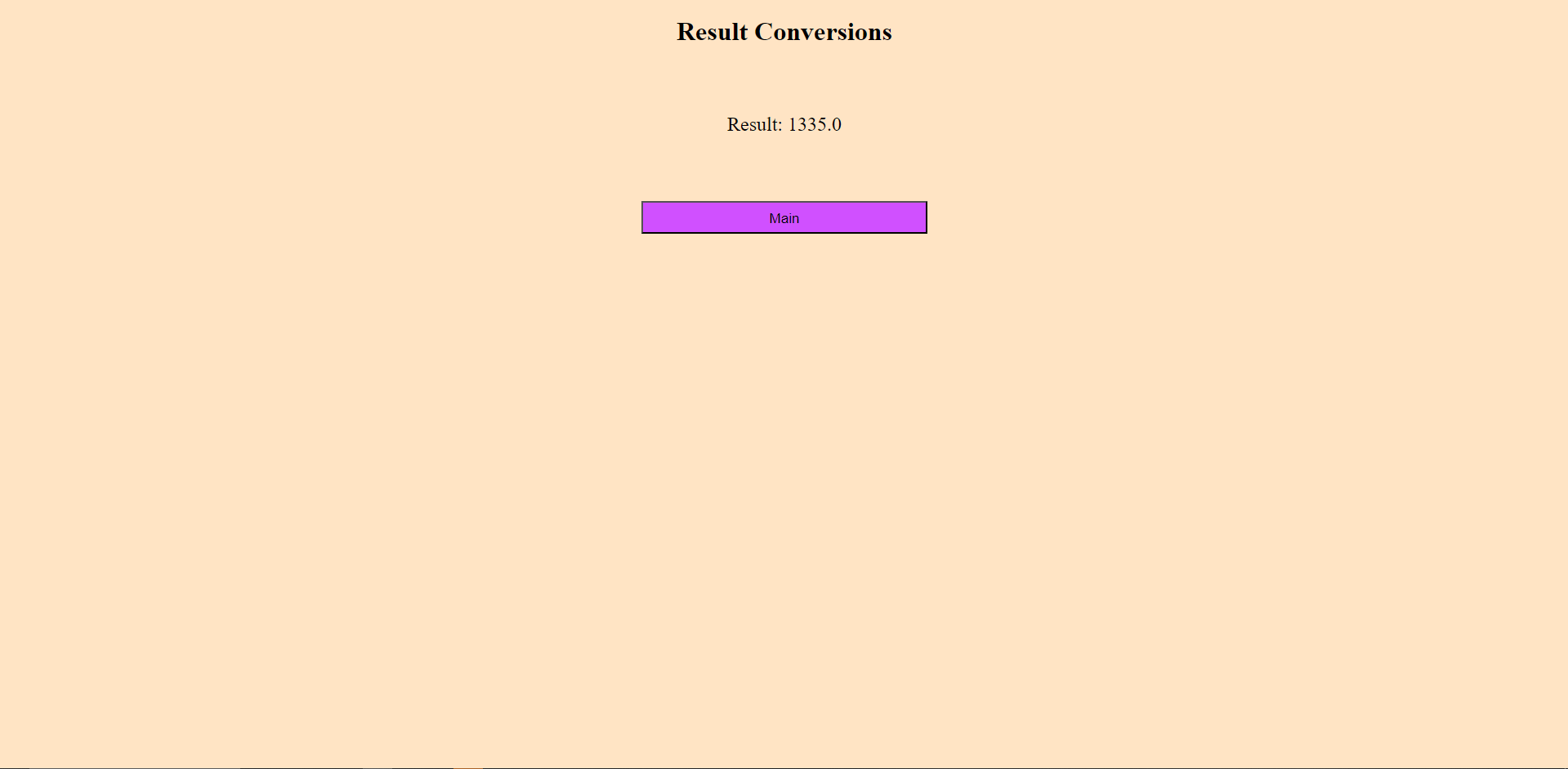
Результат работы:











Вывод: в результате данной работы было разработано приложение с использованием Spring Framework, включающее 5 страниц: основную страницу, предоставляющую доступ к страницам «Калькулятор» и «Конвертер валют», а также страницы, отображающие результаты вычислений.

Практическая работа №2

Цель работы: разработать приложение с помощью Spring Framework, в котором будет реализован паттерн DAO.

Для начала создаем модели с 4 полями.

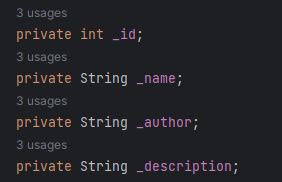


Рисунок 5. 4 поля.

Создаем конструкторы.

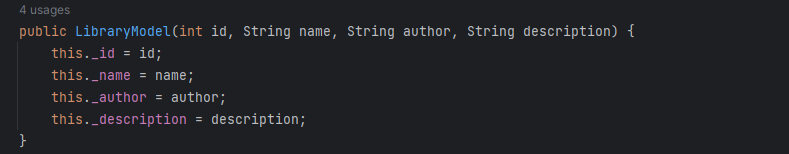


Рисунок 6. Конструктор.

Затем создаем геттеры и сеттеры, благодаря которым мы сможем получать и записывать данные модели.



Рисунок 7. Конструктор.

По такой же схеме создаем другие модели.

Затем мы определяем класс для работы с паттерном DAO, который обеспечивает функции создания, чтения, обновления и удаления данных.

В начале класса мы применяем аннотацию @Component, что сообщает Spring Framework о том, что класс bookDAO должен быть управляемым компонентом (bean) и подвергается управлению контейнером Spring. Это означает, что Spring будет создавать экземпляры этого класса и управлять их жизненным циклом.

Далее, мы создаем статическое поле для отслеживания количества элементов в списке. Список List<LibraryModel> используется для хранения объектов LibraryModel, и мы добавляем в него несколько данных.

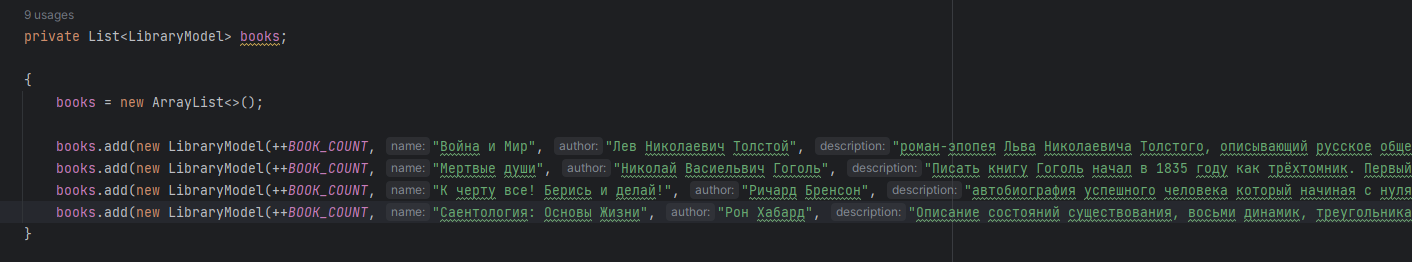


Рисунок 8. Список.

Метод indexBook() возвращает список всех книг, showBook(int id) возвращает книгу с указанным идентификатором, saveBook(LibraryModel book) добавляет объект в список.



Рисунок 9. Методы.

По той же логике прописываем класс DAO для всех моделей.

Аннотация @RequestMapping("/books") устанавливает базовый путь, который будет применяться ко всем методам внутри данного контроллера.

Аннотация @Autowired указывает на то, что зависимость bookDAO будет автоматически внедрена при создании контроллера.

В методе index мы извлекаем список всех элементов и помещаем его в модель, которая затем будет использоваться для представления этого списка пользователю.



Рисунок 9. Аннотации (1).

Метод show позволяет извлекать информацию о конкретном элементе, используя аннотацию @PathVariable, и затем добавляет эту информацию в модель.

NewBook просто открывает страницу для создания нового элемента, в то время как метод createBook(@ModelAttribute("book") LibraryModel book, Model model) получает данные из формы и сохраняет их в список.

EditBook также предоставляет доступ к странице для редактирования записи, передавая данные об элементе с конкретным идентификатором на эту страницу.

Метод updateBook(@ModelAttribute("book") LibraryModel book, @PathVariable("id") int id) обрабатывает PATCH-запросы, получает данные из формы и обновляет запись с указанным идентификатором.

deleteBook(@PathVariable("id") int id) удаляет запись с указанным идентификатором.

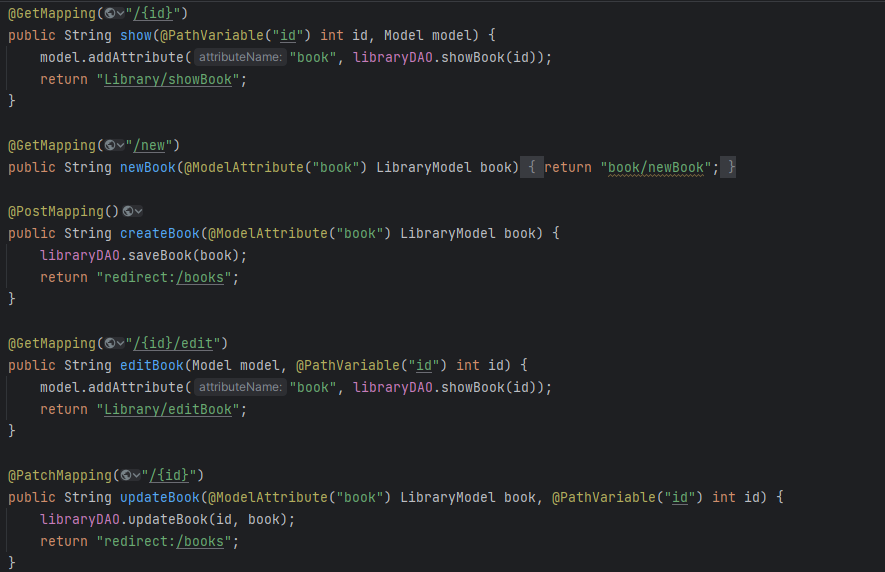
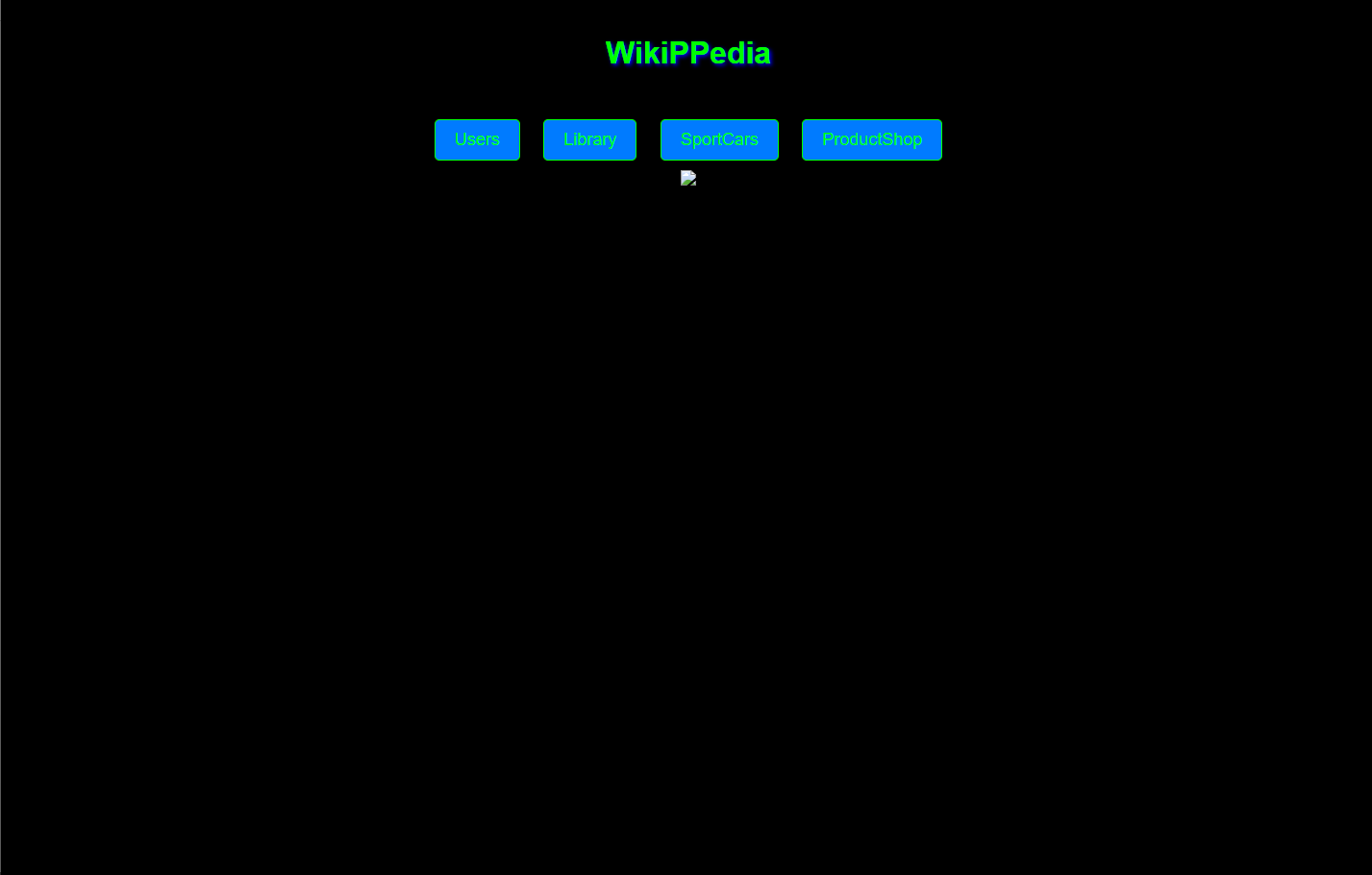
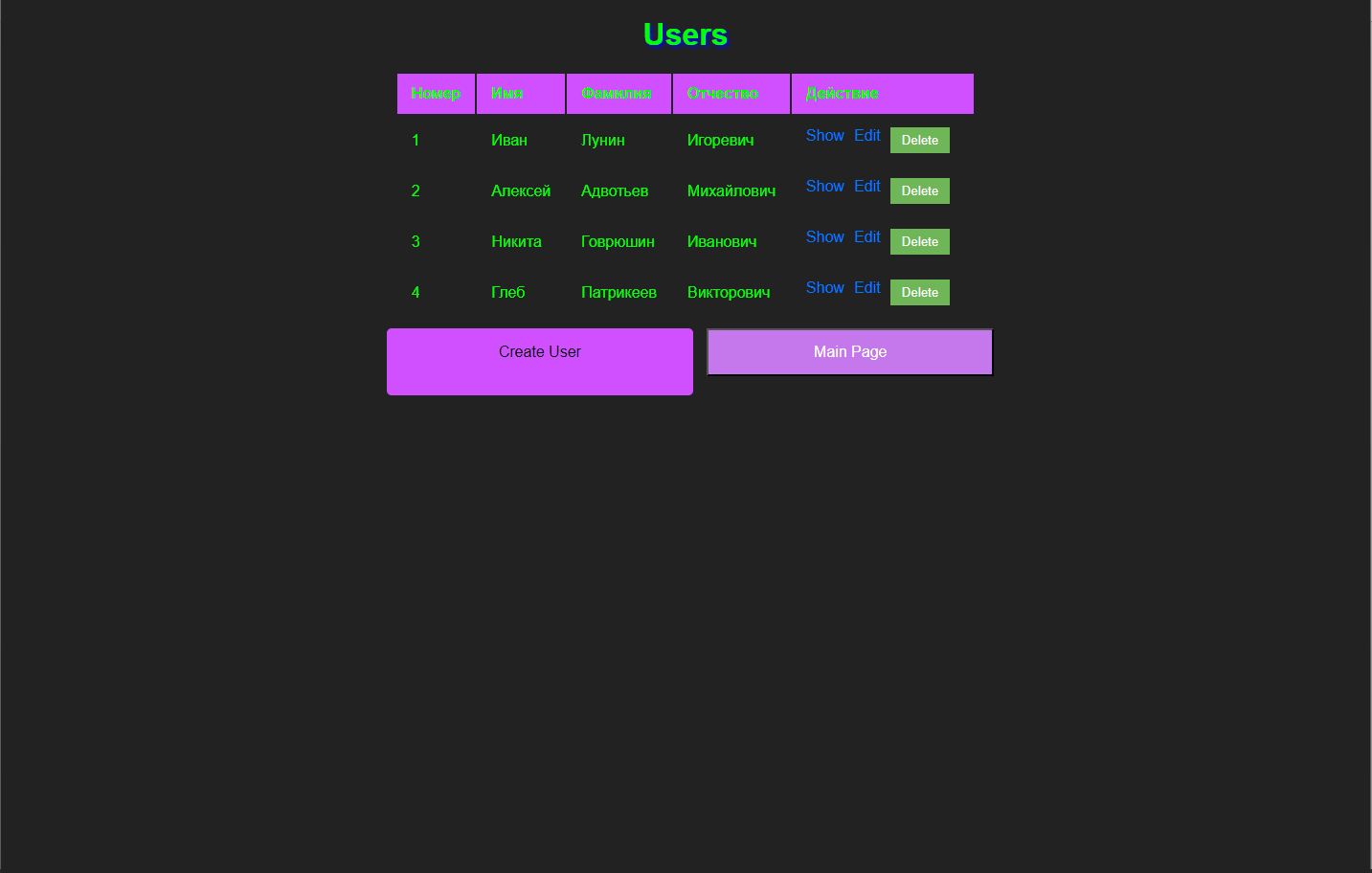


Рисунок 10. Аннотации (2).

Результат работы:

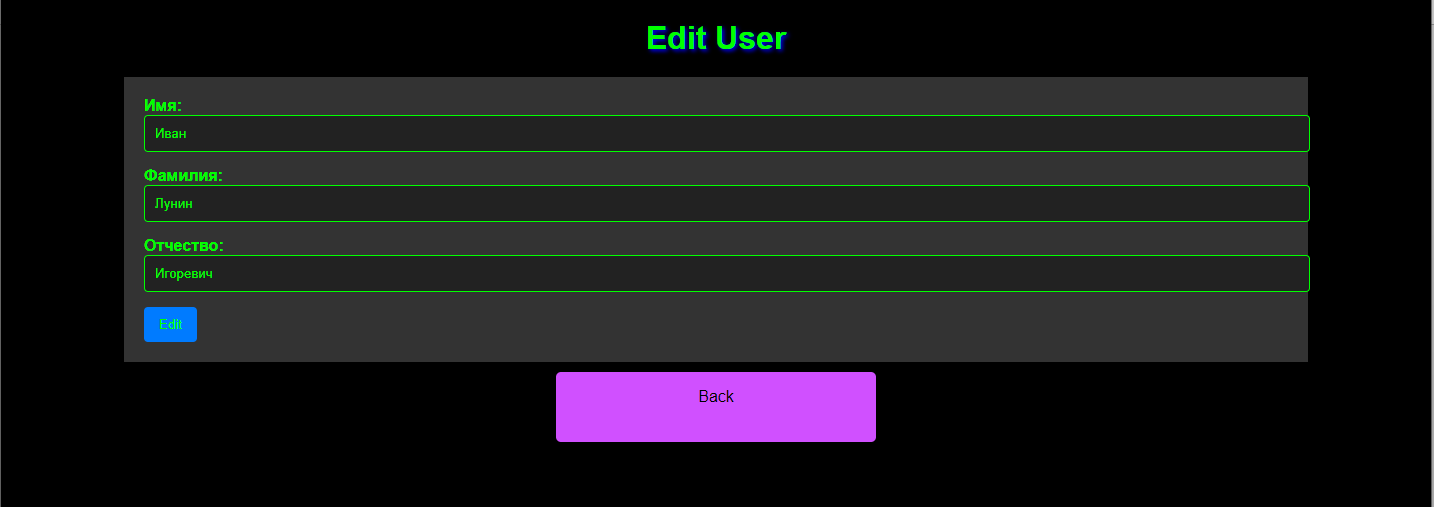




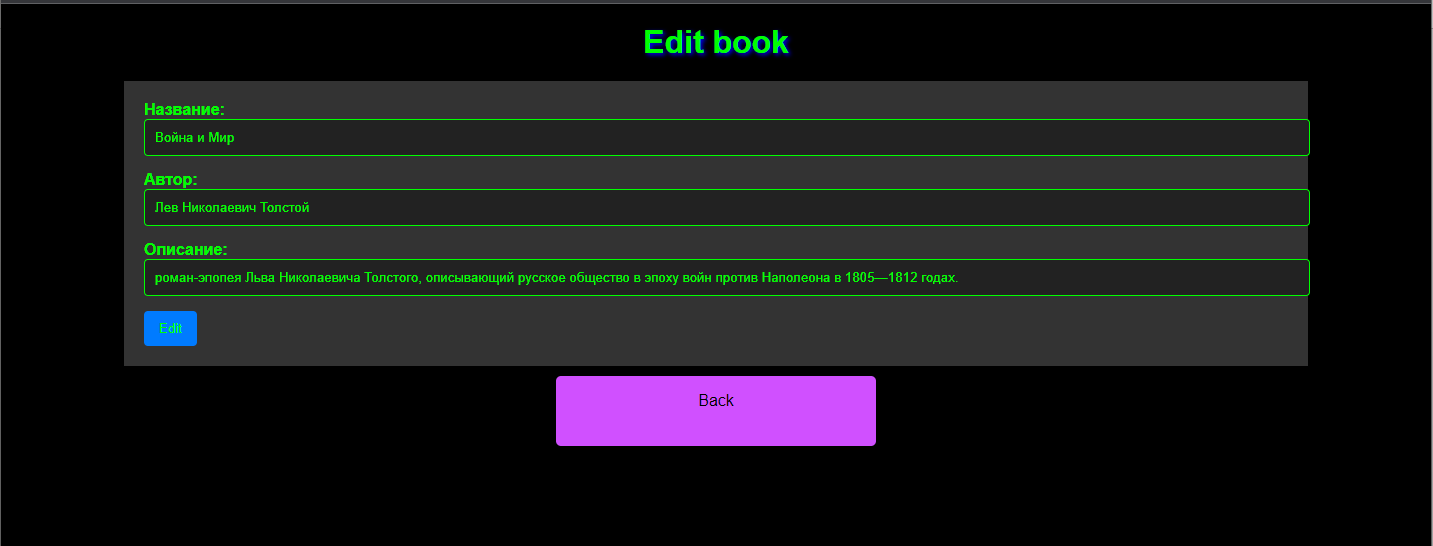


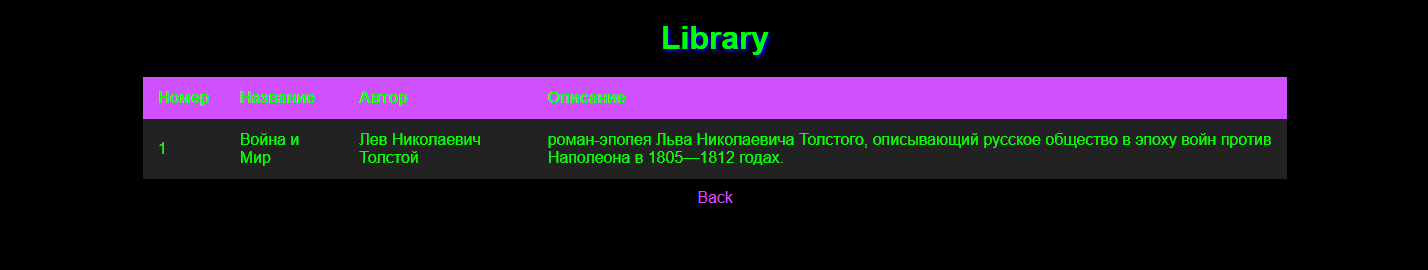


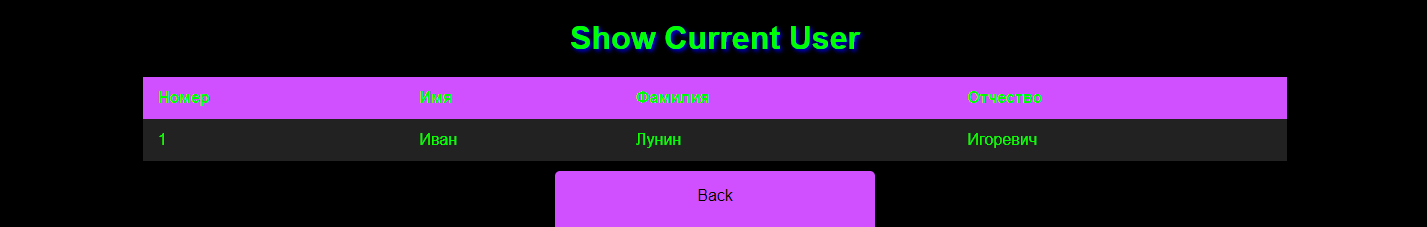












И т.д.

Вывод: в результате данной работы было разработано приложение с помощью Spring Framework, в котором реализован паттерн DAO.