|  |
| --- |
| **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.04 Сопровождение и обслуживание программного обеспечения компьютерных систем .

Специальность 09.02.07 Информационные системы и программирование \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Студент Сорокин Дмитрий Максимвич.

(фамилия, имя, отчество)

Группа П50-6-20

Руководитель по практической подготовке от техникума

Серяк Даниил Владимирович.

(фамилия, имя, отчество)

«\_\_» \_\_\_\_\_\_ 2023 года

Оглавление

[Практическая работа №1 3](#_Toc154607777)

[Практическая работа №2 9](#_Toc154607778)

[Практическая работа №3 18](#_Toc154607779)

[Практическая работа №4 27](#_Toc154607780)

[Практическая работа №5 31](#_Toc154607781)

Практическая работа №1

Калькулятор и Конвертер валют

Цель: Необходимо создать приложение которое включает в себя страницы:

1) Главная страница , на которой находятся кнопки для перехода на страницу "Калькулятор" и "Конвертер валют"

2) Калькулятор, данная страница выполняет функционал базового калькулятора, но после получения результата, пользователя должно перекинуть на страницу с ответом. Необходимо использовать @PostMapping

3)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.

Основные требования по коду:

1)Использовать @GetMapping и @PostMapping

2)Использовать @RequestParam

Ход работы:

1. Создание фала c html разметкой для страниц

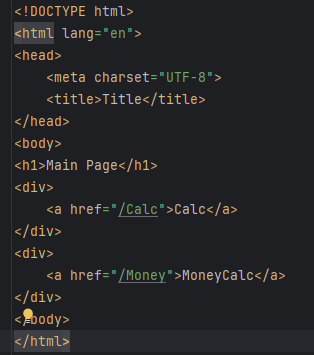


Рисунок 1 Главная страница

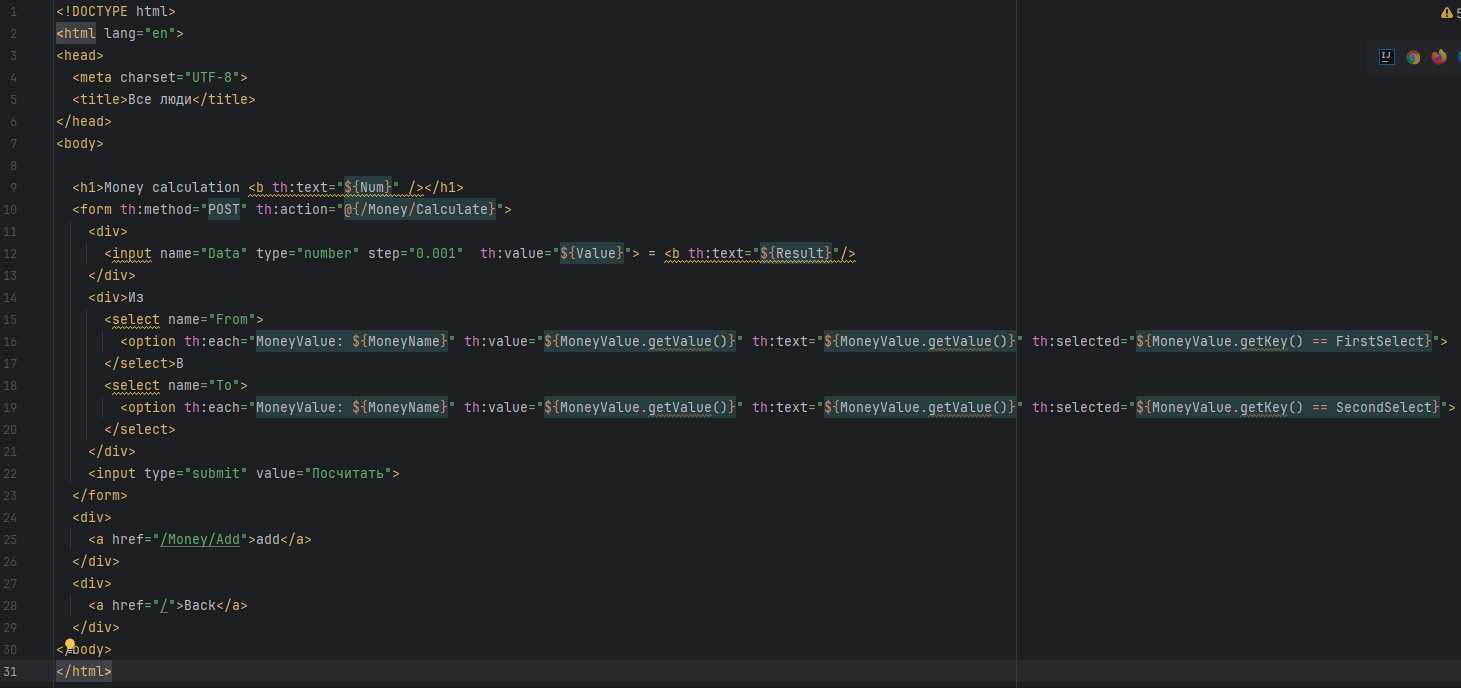


Рисунок 2 Страница перевода валют

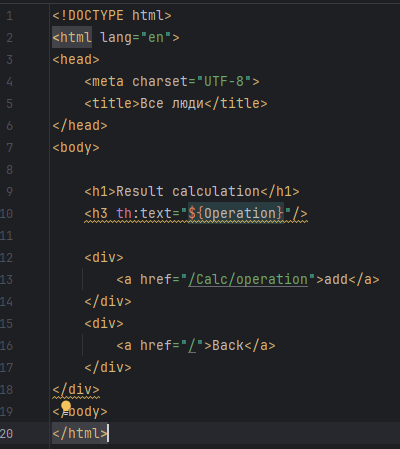


Рисунок 3 Страница с результатов вычислений

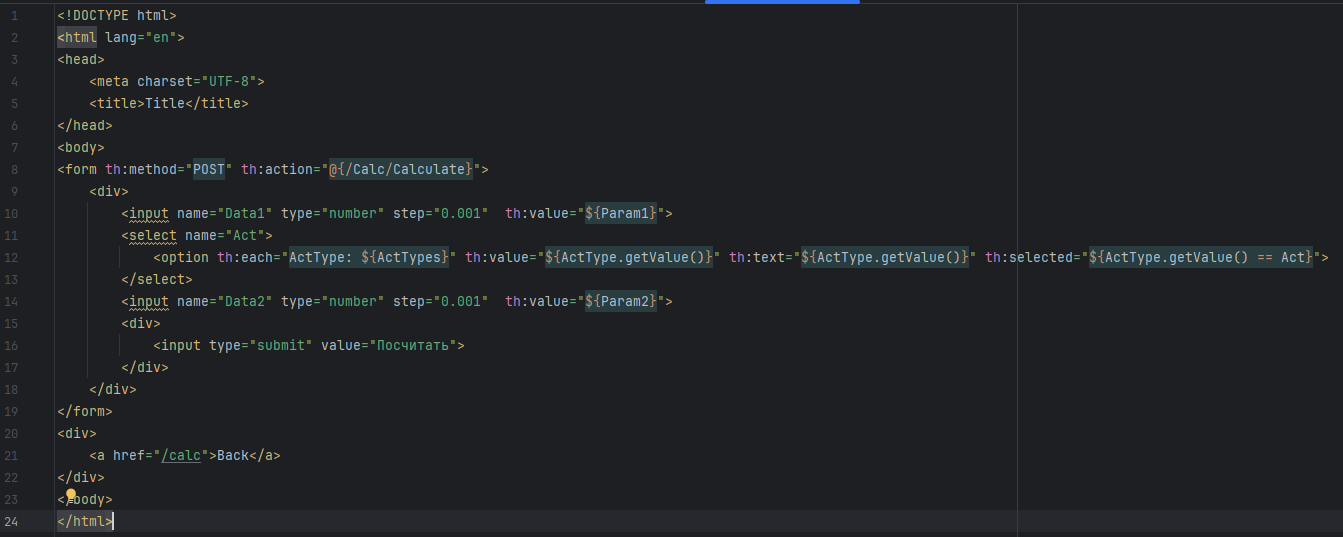


Рисунок 4 Страница указания параметров вычисления

1. Создание контроллеров управляющие переходами между страницами

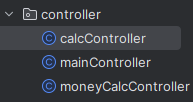


Рисунок 5 Контроллеры



Рисунок 6 Контроллер калькулятора

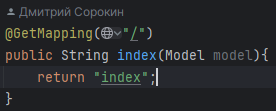


Рисунок 7 Контроллер начального окна

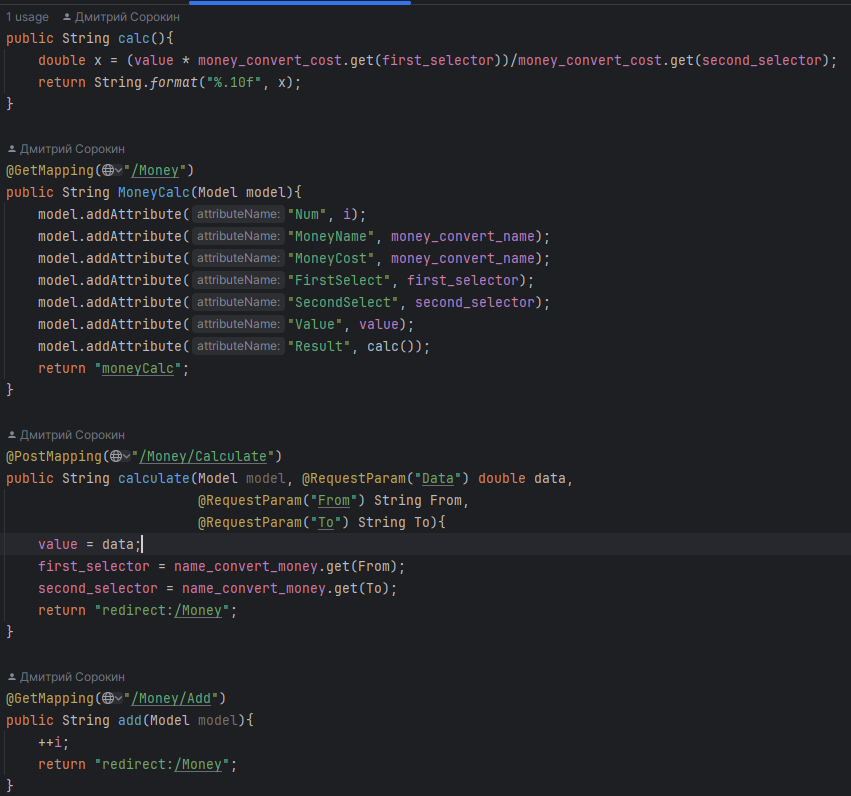


Рисунок 8 Контроллер конвертации валют

1. Результат работы

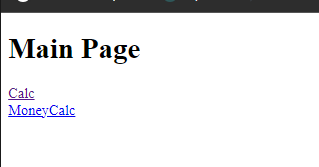


Рисунок 9 Главная страница

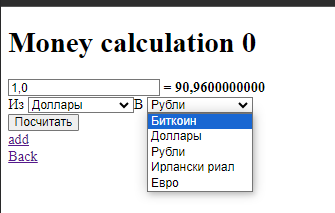


Рисунок 10 Перевод валют

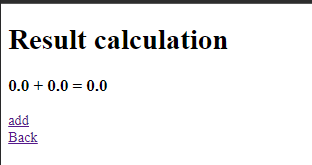


Рисунок 11 Страница вывод ответа калькулятора



Рисунок 12 Страница указания параметров вычисления

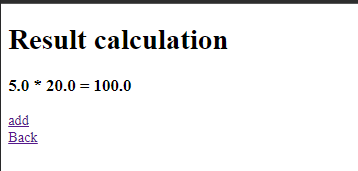


Рисунок 13 Вывод ответа

Вывод: В ходе практической работы №1 было создано приложение включающее в себя калькулятор и перевод валют.

Практическая работа №2

Работа с паттерном DAO

Цель: В данной практической необходимо реализовать приложение, в котором будет реализован паттерн DAO.

Требования:

1) На 3 необходимо создать 5 моделей по 4 поля в каждой, и реализовать базовый паттерн DAO(Повторить действия из примера)

2) На 4 необходимо сделать навигацию по своему сайту. Так же необходимо подтянуть стили на страницы.

Особо углубляться в frontend не обязательно, можете просто подключить Bootstrap

3) На 5 необходимо добавить универсальный класс для обработки CRUD-операций, который будет вызываться для каждой модели(чтобы не было много одинакового кода)

Ход работы:

1. Создание классов



Рисунок 14 Создание абстрактного класса для указания методов которые будут у наследников



Рисунок 15 Пример реализации одного из наследников абстрактного класса

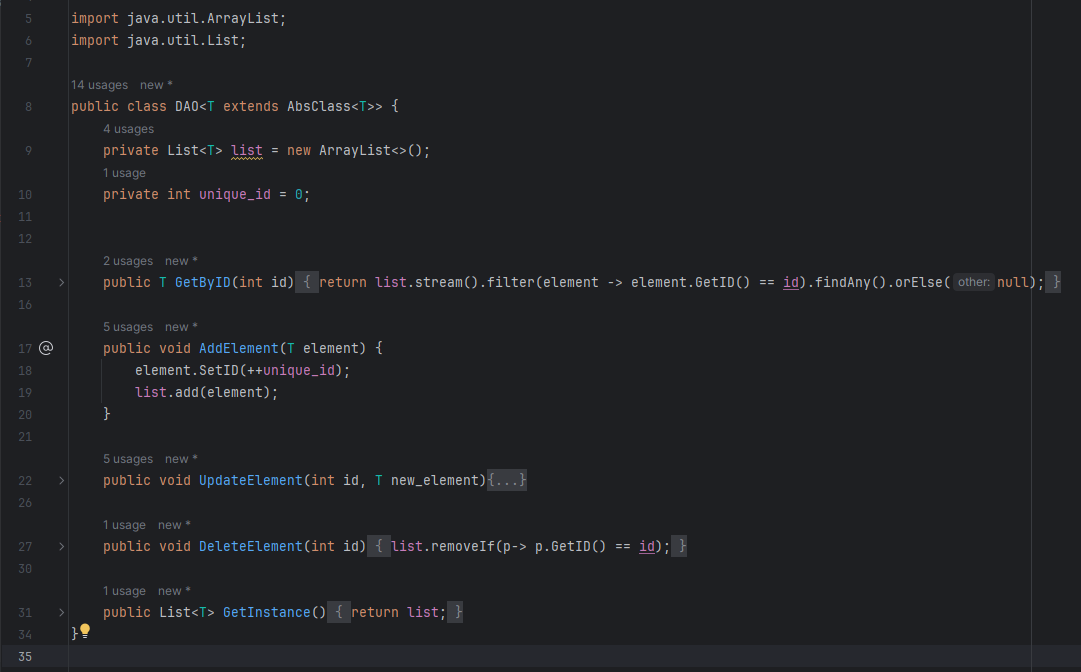


Рисунок 16 Создание шаблонного класса DAO который будет хранить и взаимодействовать с данными

1. Создание контроллеров

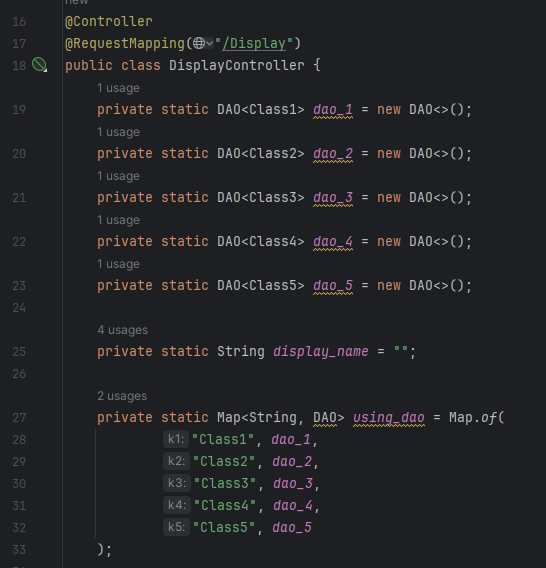


Рисунок 17 Создание переменных типа DAO для указания хранимых дынных

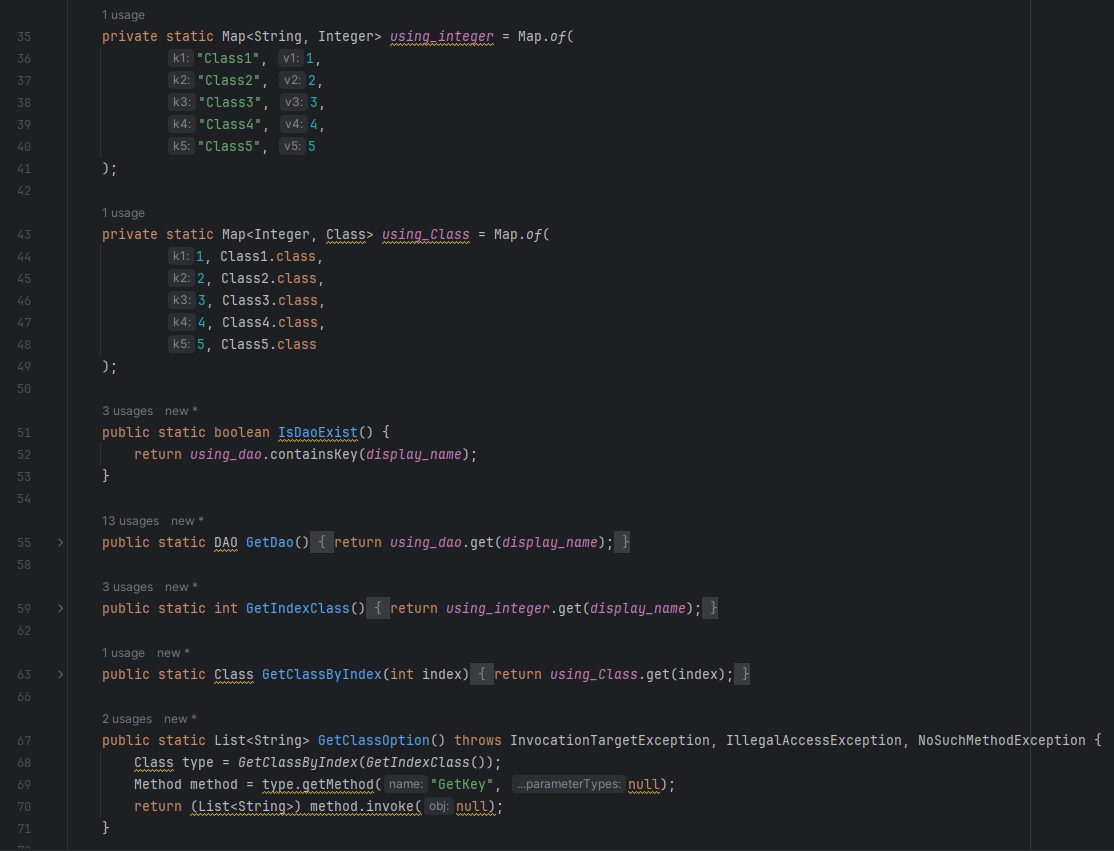


Рисунок 18 Создание статических полей и методов для быстрого взаимодействия с данными

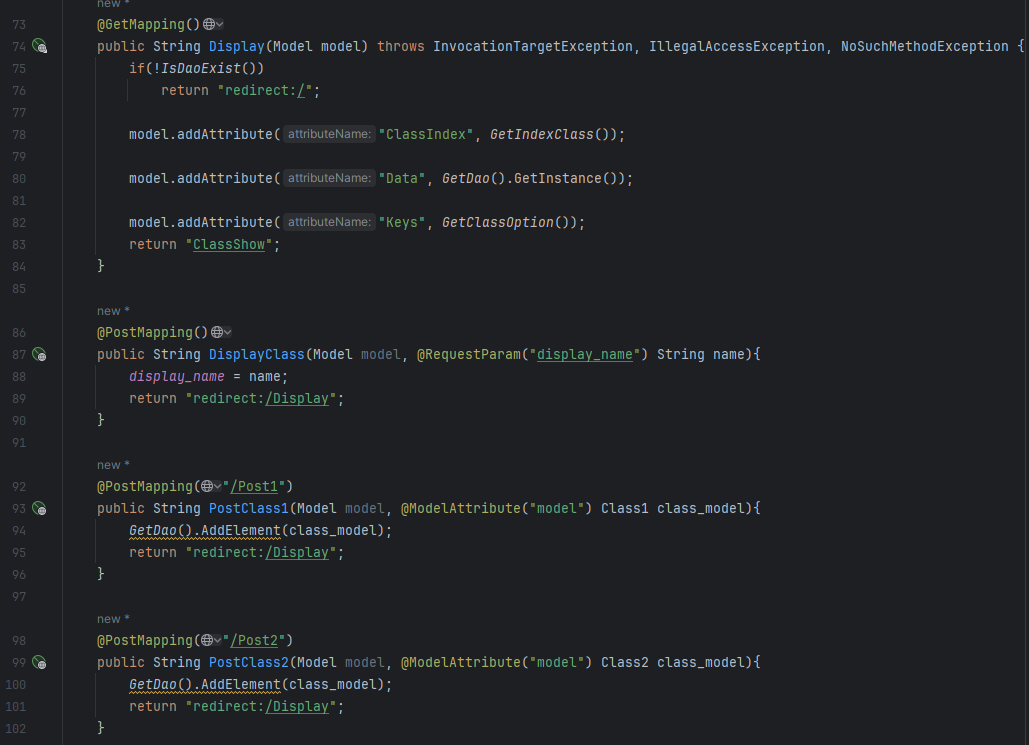


Рисунок 19 Создание запросов отображения данных и примеры добавления новых данных в поля DAO

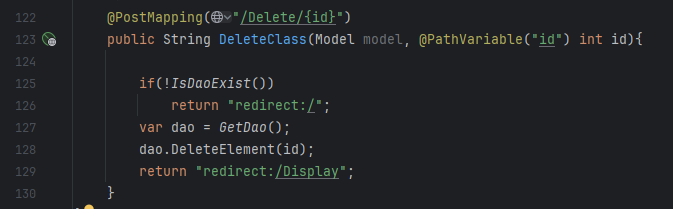


Рисунок 20 Создание запроса удаления записи из выбранного поля DAO

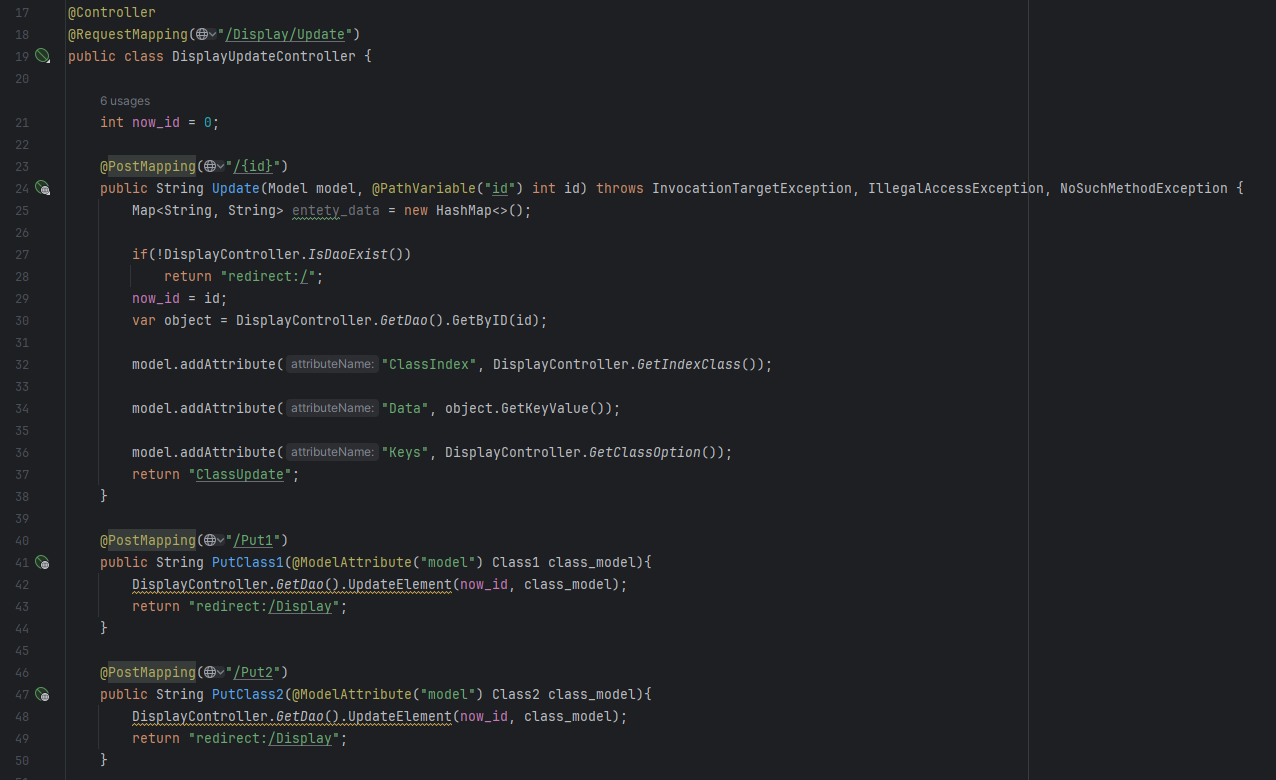


Рисунок 21 Создание контроллера изменения данных полей DAO

1. Создание разметки

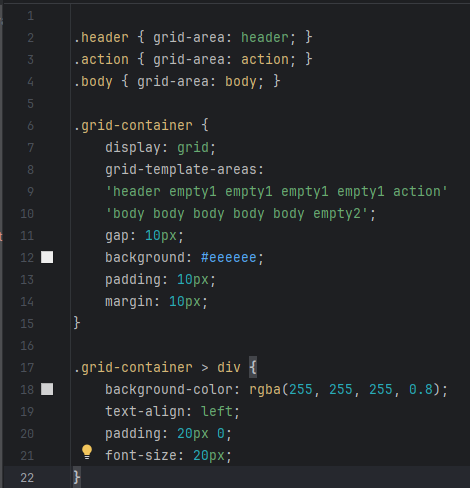


Рисунок 22 Создание стиля для отображения данных в виде ячеек

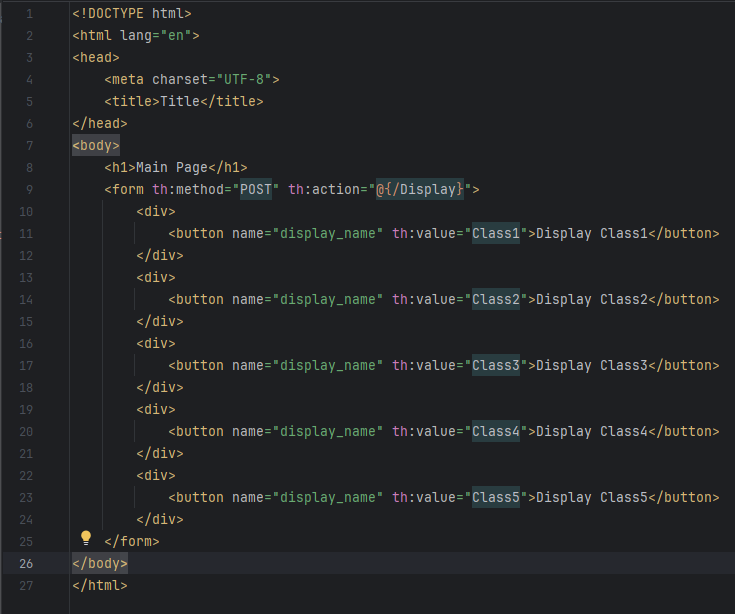


Рисунок 23 Разметка начальной страницы (Перехода на страницы с отображением информации)

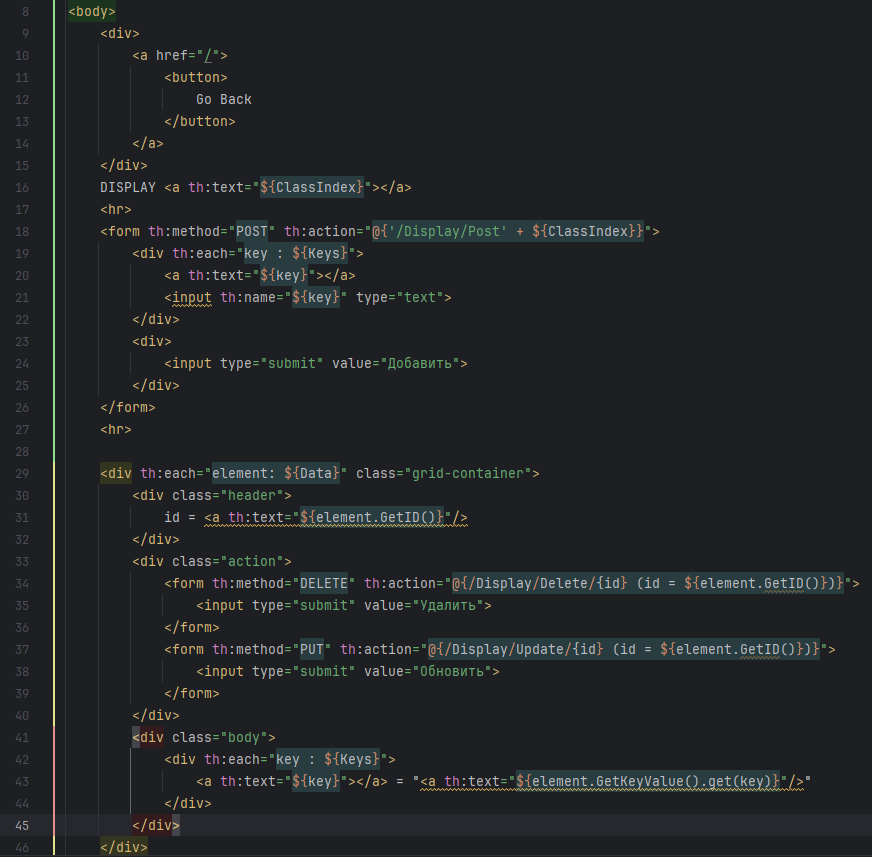


Рисунок 24 Создание разметки для отображения данных выбранного DAO (С выводом названия полей данного класса)

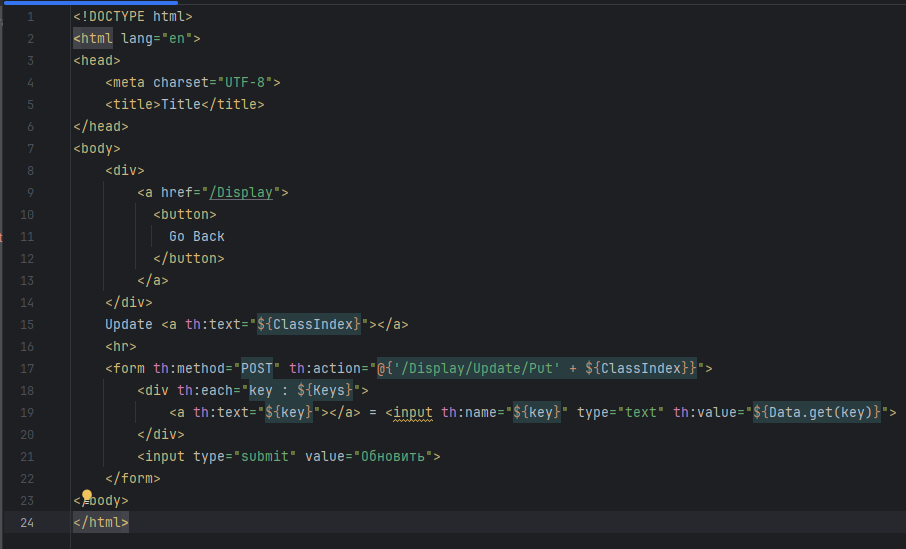


Рисунок 25 Разметка для вывода и редактирования данных одной записи

1. Результат

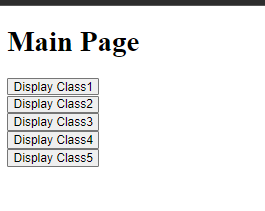


Рисунок 26 Начальная страница

Для каждого класса используется один файл отображения и окно изменения. Используется статический метод вывод названия всех полей через List, и абстрактный метод который выводит значения записи по ключам содержащихся в List. Создаваемые поля ввода генерируются в fpom что позволяет использовать @ModelAttribute для генерации нужного класса.

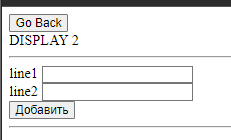


Рисунок 27 Страница отображения (пустая (Класс №2))

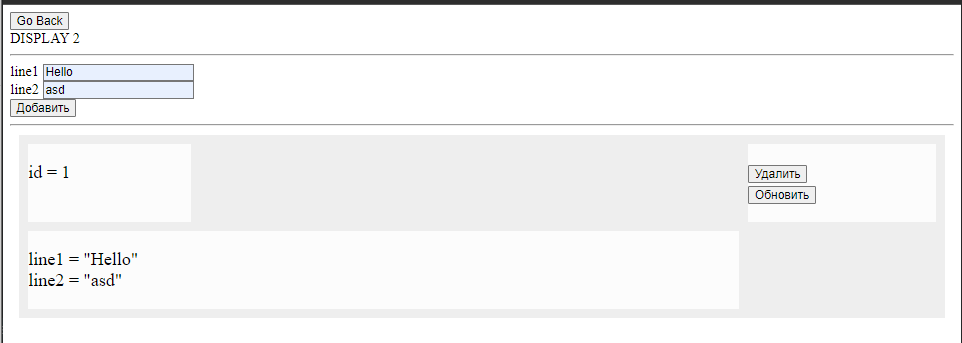


Рисунок 28 Страница отображения (не пустая (Класс №2))

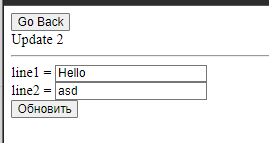


Рисунок 29 Переход на страницу изменения данных

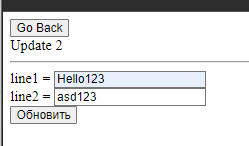


Рисунок 30 Изменение данных

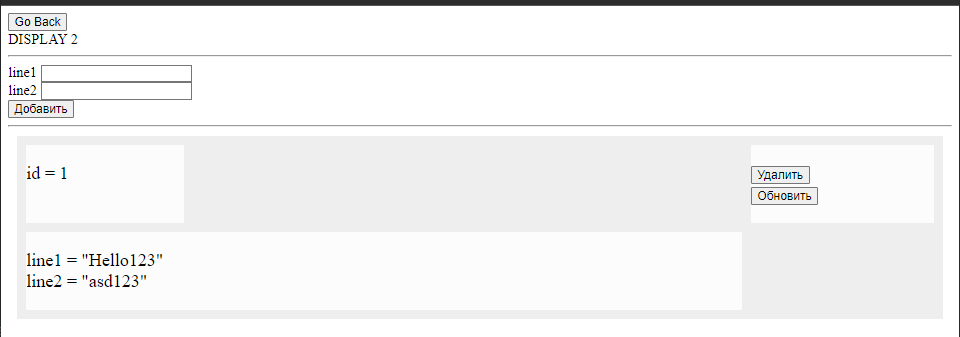


Рисунок 31 Переход обратно на страницу отображения

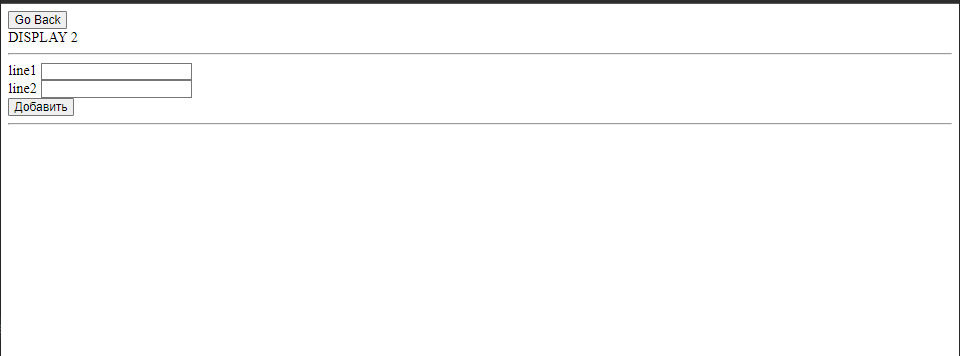


Рисунок 32 Удалить

Вывод: В ходе практической работы было создано приложения для хранения, изменения, удаления различных классов используя вспомогательный класс DAO.

Практическая работа №3

Работа с JPA и Validator

Цель: В данной практической работе необходимо подключить любую СУБД(MySQL, PostgreSQL, MS SQL) и изменить модели из предыдущей работы под новые технологии.

Требования:

1) На 3 необходимо просто переделать предыдущую работу с подключением выбранной СУБД, сделать валидацию на каждое поле с помощью аннотаций(пример @NotBlank или @Size)

2) На 4 добавить поиск определенной записи, можете выбрать любое поле, по которому будет происходить поиск

3) На 5 написать универсальные классы для CRUD операций

Ход работы:

1. Создание классов

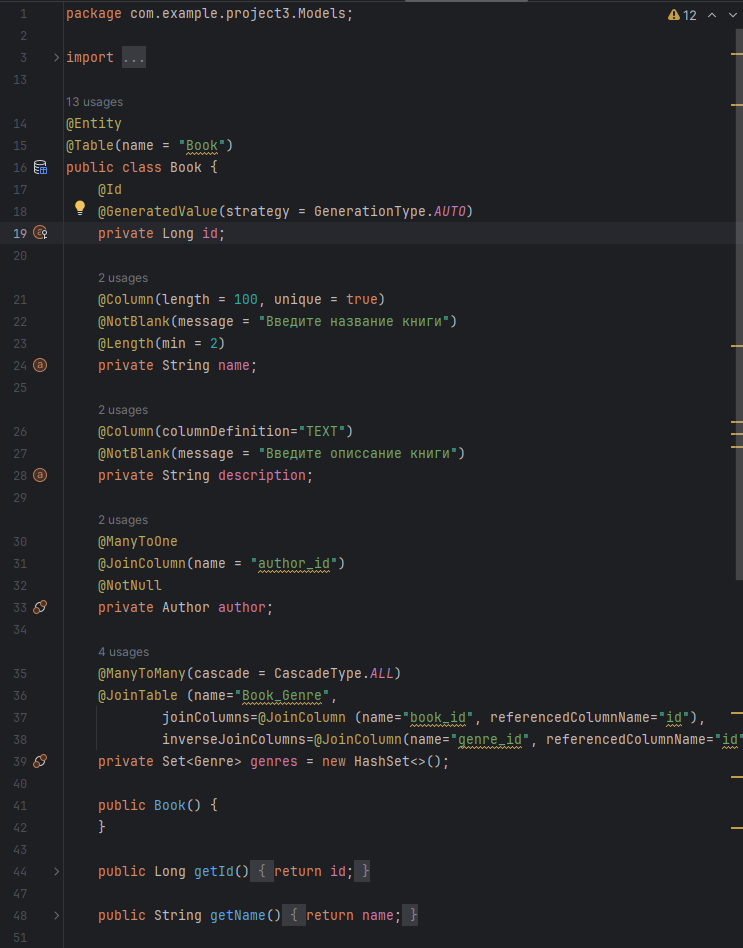


Рисунок 33 Пример создаваемого класса с использованием аннотаций

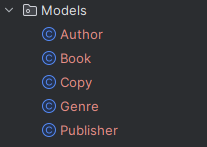


Рисунок 34 Созданные классы

1. Создание интерфейсы репозиториев

На основе этих будут создавать запросы к базе данных.

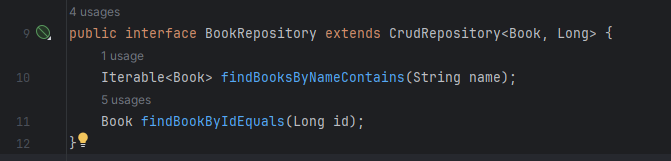


Рисунок 35 Пример создаваемого интерфейса репозитории



Рисунок 36 Созданные интерфейсы

1. Создание контроллеров

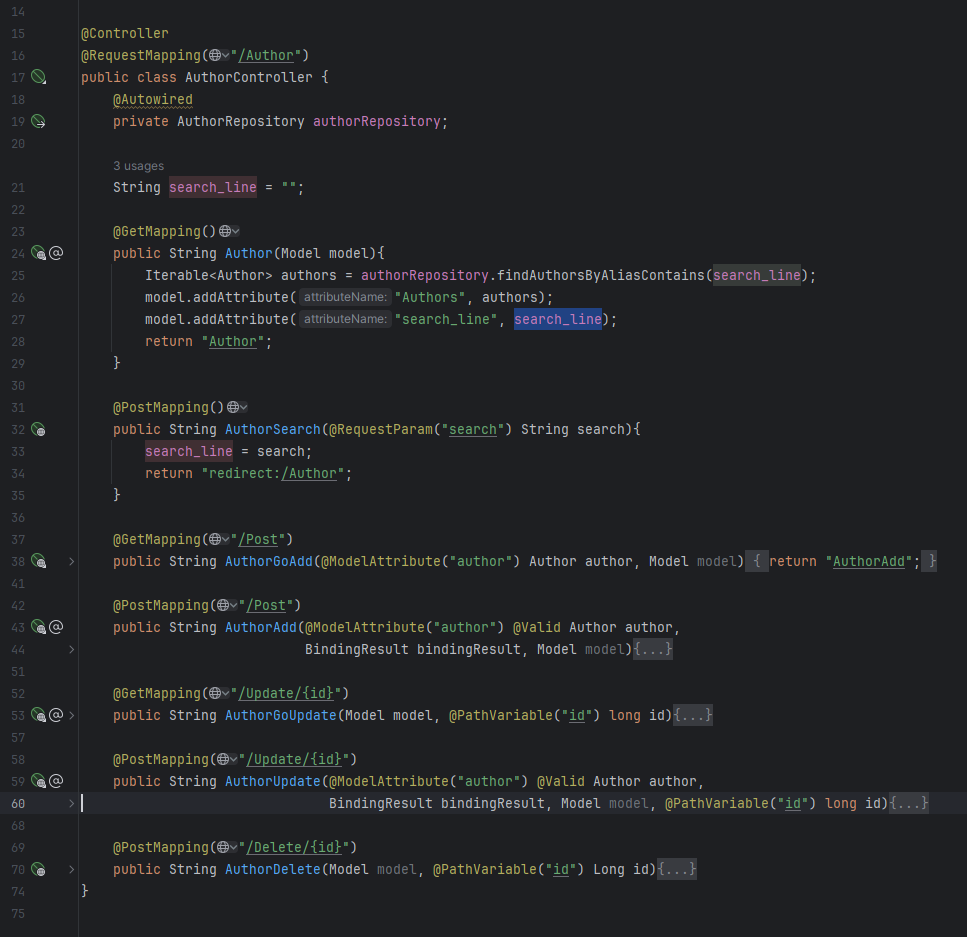


Рисунок 37 Пример созданных контроллеров



Рисунок 38 Созданные контроллеры

1. Создание базы данных

Для автоматического создания базы данных на основе написанных классов необходимо выбрать используемую бд. (Была выбранная PostgreSQL). Для возможности обратиться к базе данных ее нужно создать и указать в файле отображенном на рисунке 39 файле прописать где база данных находится (localhost) на каком порту и ее название. Далее для обращения нужно указать имя пользователя и его пароль. Далее указывается режим работы с базой данных (update – обновление базы данных при наличии новых элементов в классах) Потом указывается используемое СУБД.

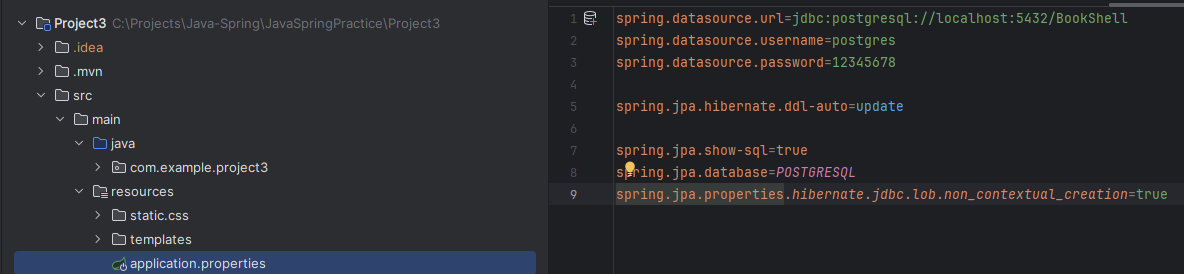


Рисунок 39 Файл с настройками для обращения к базе данных

После успешного запуска указанная база данных обновится и в нее будут добавлены таблицы с указанными в класса полями.

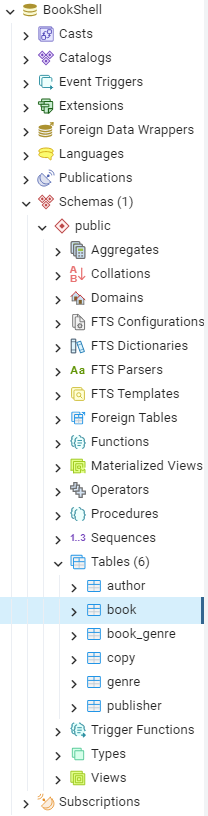


Рисунок 40 Созданная база данных

1. Результат работы программы

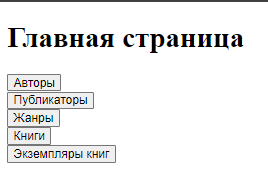


Рисунок 41 Главная страница

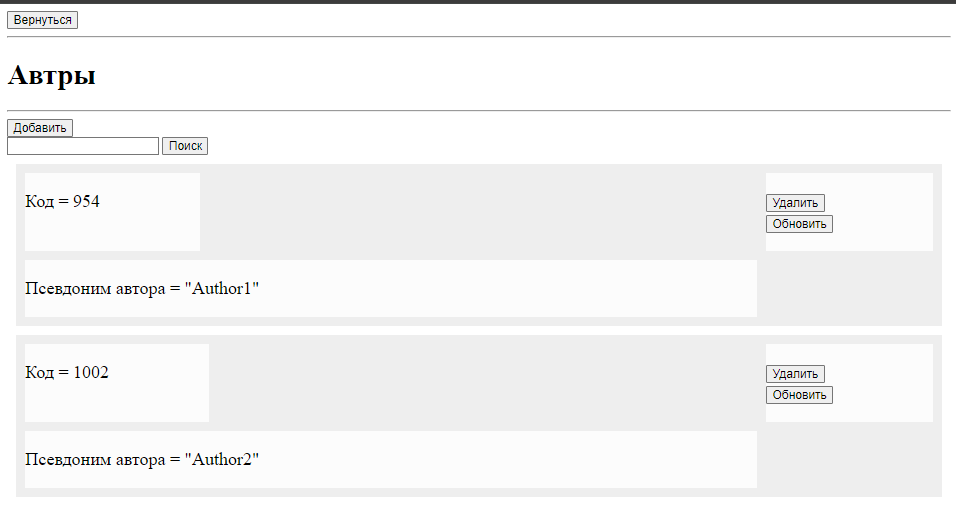


Рисунок 42 Страница авторов

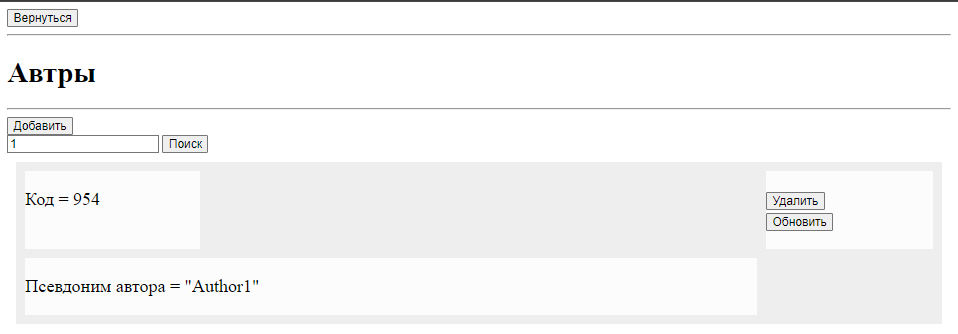


Рисунок 43 Пример поиска (авторы)

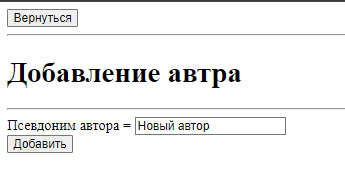


Рисунок 44 Добавление автора

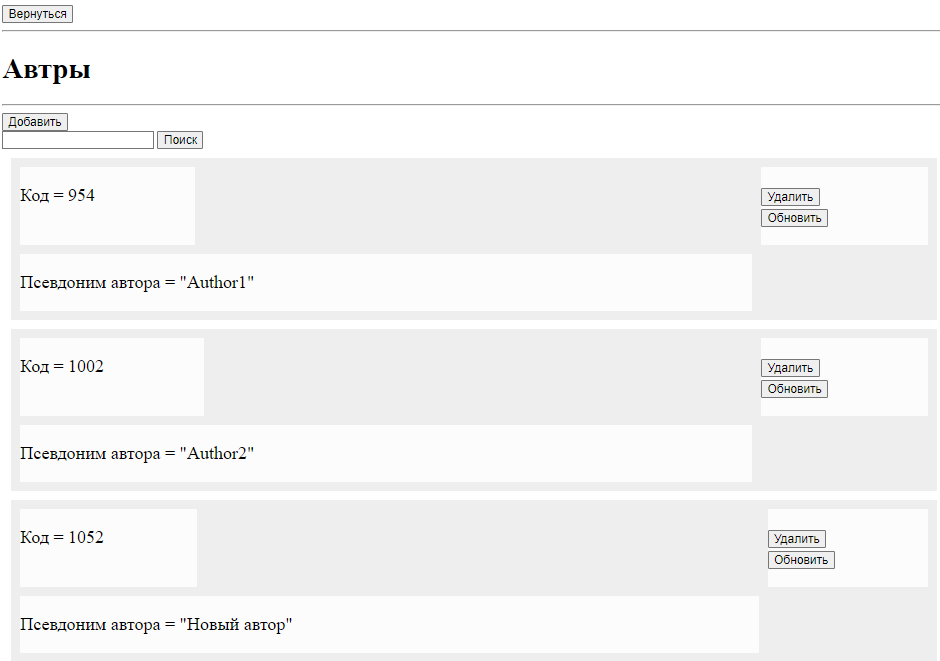


Рисунок 45 Отображение нового автора

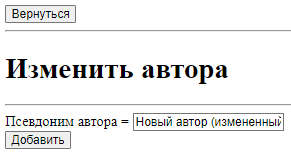


Рисунок 46 Изменение созданного автора

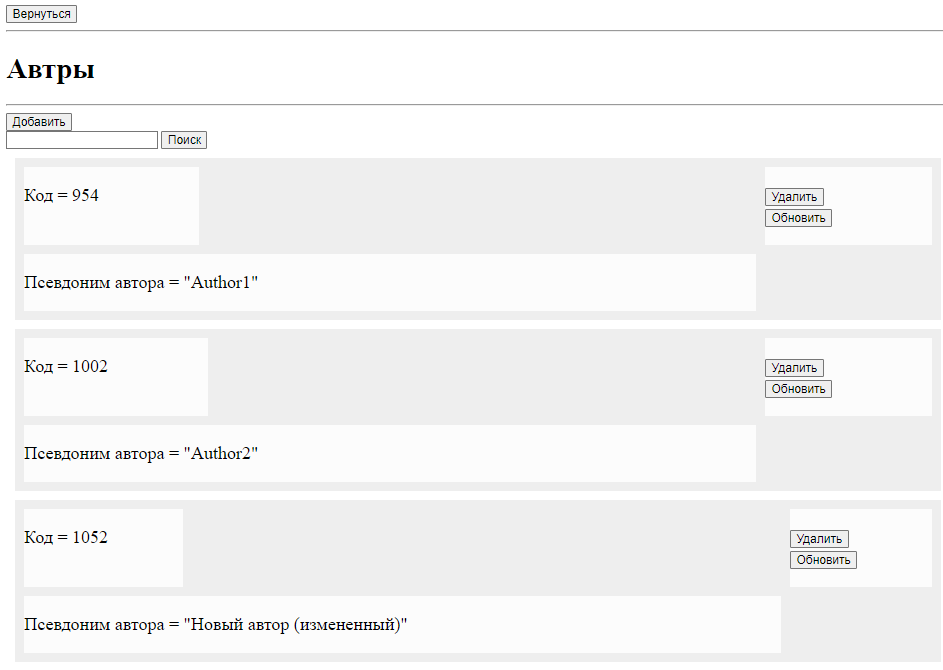


Рисунок 47 Отображение измененного автора

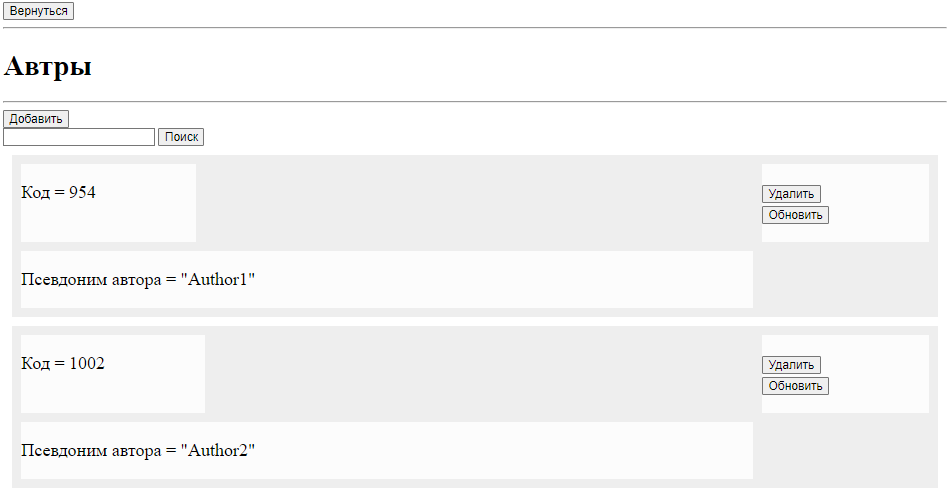


Рисунок 48 Пример удаления (Автор)

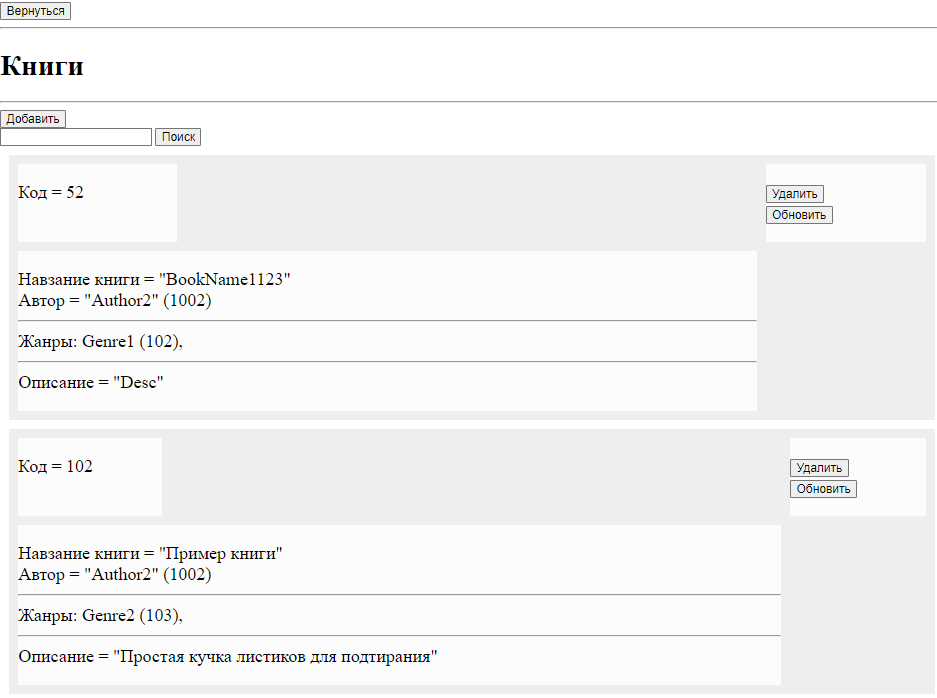


Рисунок 49 Пример отображения (книги)

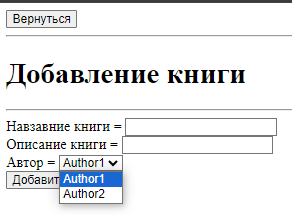


Рисунок 50 Пример добавления книги с использованием поля выбора

Вывод: В ходе практической работы №3 было созданного приложения и подключили к нему СУБД PostgreSQL и создана база данных и возможность обращения к ней с помощью приложения.

Практическая работа №4

Связи

Цель: Реализовать 3 типа связей, описать их и продемонстрировать их работу, для этого следует создать дополнительные модели и связать с уже созданными таблицами или между собой.

- Реализацию всех типов связей;

- Аннотации: @OneToOne, @ManyToOne, @OneToMany, @ManyToMany, @JoinTable, @Table;

- Описать как объединяем таблицы и какие необходимые параметры указываем для аннотаций;

- Продемонстрировать работу связей.

Ход работы:

1. Реализация связей

OneToOne – Связь «Один к одному». Одна запись может принадлежать только одной записи(Реализовывать данную связь сложно где либо применить потому что она мало где может пригодиться).

OneToMany – Связь «Один ко многим». Одна запись может принадлежать большому количеству других записей (Пример: Один автор может написать много книг (рисунок 51))

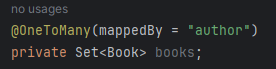


Рисунок 51 Пример OneToMany

ManyToOne – Связь «Много ко одному». Реверсивное представление OneToMany (Пример: Много книг имеют одного автора (рисунок 52))

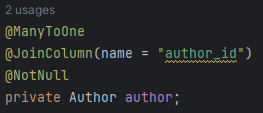


Рисунок 52 Пример ManyToOne

ManyToMany – Связь «Многие ко многим». Множество записей могут иметь множество других записей (Пример: Две футбольные команды пожимают друг другу руки)

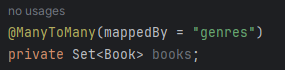


Рисунок 53 Пример ManyToMany

JoinTable – Связь строящаяся через дочернюю таблицу имею по одной ссылке на записи родительских таблиц (Теже команды футболистови рукопожатия (записи о рукопожатиях записываются в отдельную таблицу записи которой ссылается на одного члена первой командыи на одного члена второй команды)).

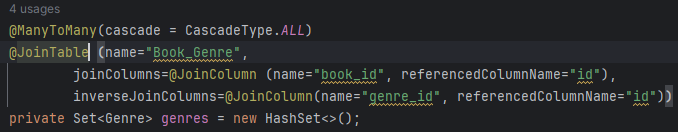


Рисунок 54 Пример JoinTable

1. Пример работы

OneToMany:

Данная связь нужна скорее для фильтрации и поиска по полям имеющих связь ManyToOne. Это может позволить найти все записи в которых присутствует данная запись (Можно найти все книги которые написал данный автор)



Рисунок 55 Возможности OneTMany

ManyToOne:

Связь обеспечивающее соединение нескольких записей к одной.



Рисунок 56 Пример ManyToOne в виде авторов книг

ManyToMany:

Связь обеспечивающее соединеие множества записей с множеством других записей.

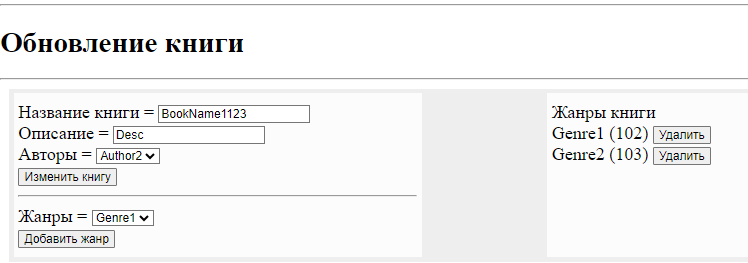


Рисунок 57 Пример ManyToMany в виде множества жанров множества книг

Вывод: В ходе практической работы №4 были реализованы связи между классами.

Практическая работа №5

Калькулятор и Конвертер валют

Цель: Добавить регистрацию и авторизацию в свой проект.

- Описать подключаемые зависимости;

- Описать файлы MvcConfig и WebSecurityConfig

- Описать создание контроллера для регистрации, модели и представления.

- В результате продемонстрировать регистрацию пользователя и его авторизацию.

Ход работы:

1. Подключение зависимостей

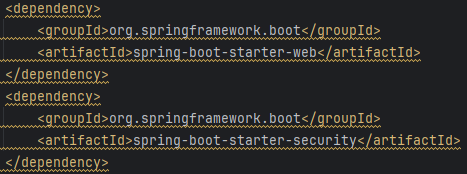


Рисунок 58 Добавленная зависимость для работы с безопасностью и авторизацией

1. Файлы MvcConfig и WebSecurityConfig

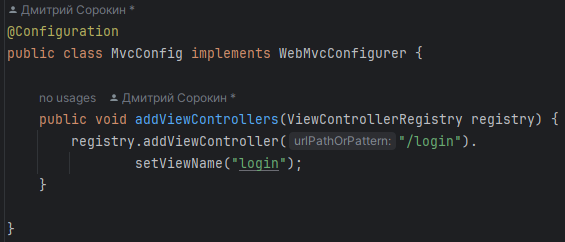


Рисунок 59 Файл MvcConfig

MvcConfig необходим для создания контроллера который будет обрабатывать авторизацию пользователя при переходе по пути /login.

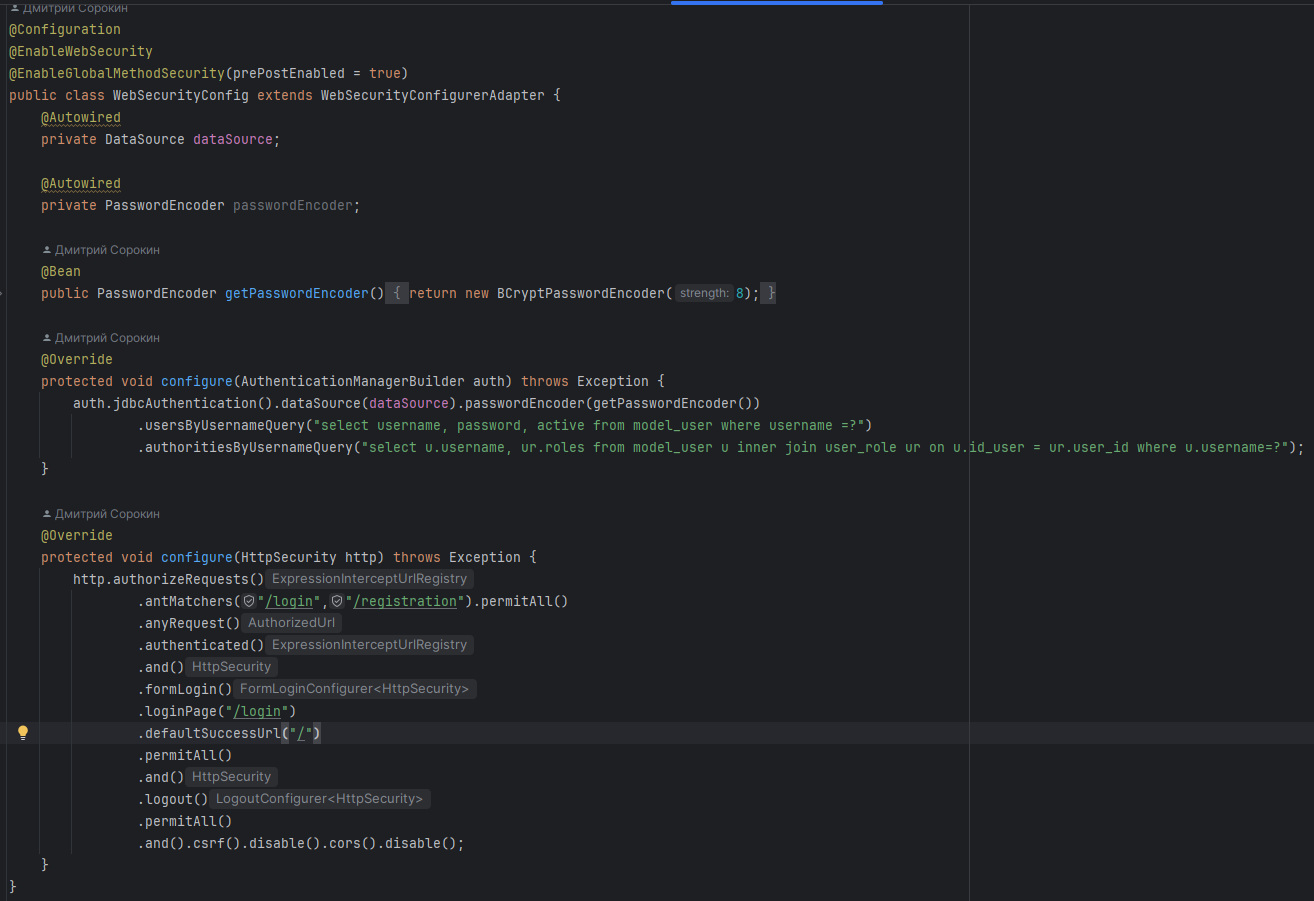


Рисунок 60 Файл WebSecurityConfig

WebSecurityConfig файл ответственный за проверку авторизовался ли пользователь. В случае утвердительного ответа он откроет html файл index.

1. Создание контроллера для регистрации, модели и представления

Создание моделей для авторизации и регистрации:

Класс пользователя должен сохраняться в базе данных поэтому необходимо как и с другими класса приложения использовать аннотацию @Entity и создать интерфейс репозитория наследующий класс CrudRepository.

Также необходимо для сохранения ролей добавить внутри класса пользователя перечисление (Set (множество)) ролей данного пользователя и указать что они будут сохраняться в таблице с полями user\_id и user\_role в виде строки.

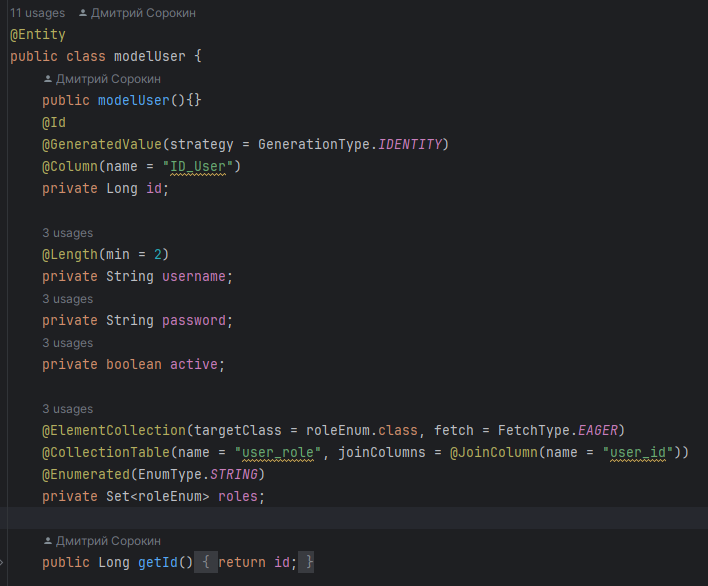


Рисунок 61 Класс пользователя

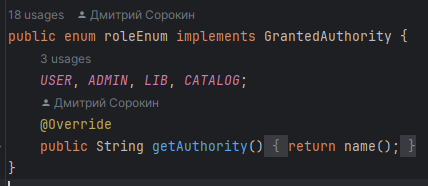


Рисунок 62 Enum класс ролей

Реализация контроллера регистрации:

Метод перехода на окно регистрации, метод регистрации которая перекинет на окно авторизации при успешной регистрации.

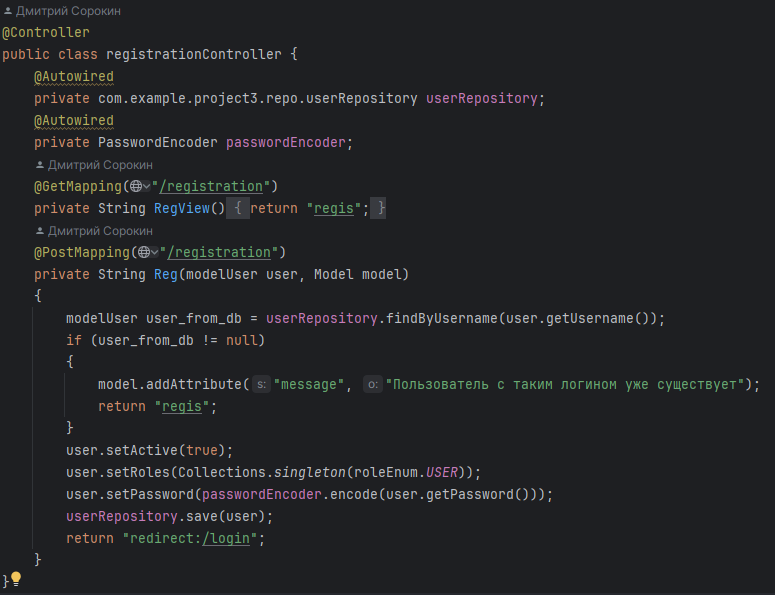


Рисунок 63 Контроллер регистрации

1. Результат работы

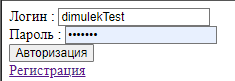


Рисунок 64 Попытка авторизироваться под не существующим аккаунтом

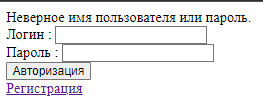


Рисунок 65 Вывод ошибки

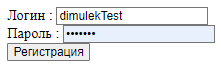


Рисунок 66 Регистрация

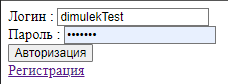


Рисунок 67 Успешная авторизация

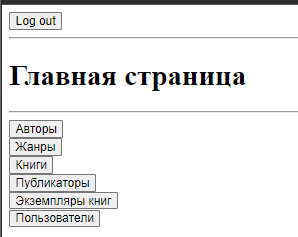


Рисунок 68 Открытие главной страницы

Вывод: В ходе практической работы №5 была реализована авторизация, регистрация пользователя с помощь spring-boot-starter-security и сохранение новых пользователей в базе данных.