

**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.  
Программист

Студент \_\_\_\_\_  
*подпись*

Суслин А.М.  
*фамилия, имя, отчество*

Группа П50-2-18

Руководитель по практической подготовке от техникума

\_\_\_\_\_  
*подпись*

Буканов И.Д.  
*фамилия, имя, отчество*  
«09» ноября 2021года

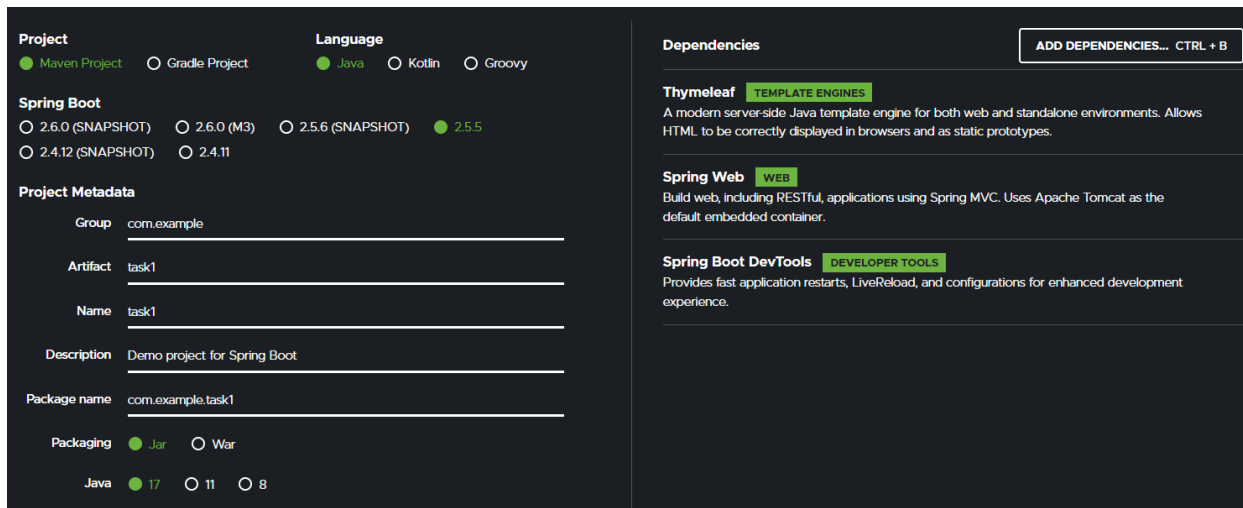
## ОГЛАВЛЕНИЕ

|                              |    |
|------------------------------|----|
| Практическая работа №1 ..... | 3  |
| Практическая работа №2 ..... | 8  |
| Практическая работа №3 ..... | 17 |
| Практическая работа №4 ..... | 22 |
| Практическая работа №5 ..... | 26 |
| Практическая работа №6 ..... | 31 |
| Индивидуальный проект. ....  | 37 |

## Практическая работа №1.

Цель работы: Необходимо сделать калькулятор с помощью Post и Get.

Для начала создадим шаблонный проект с нужными зависимостями на сайте [start.spring.io](https://start.spring.io)



The screenshot shows the Spring Boot project generator interface. On the left, under 'Project', 'Maven Project' is selected. Under 'Language', 'Java' is selected. Under 'Spring Boot', version '2.5.5' is selected. The 'Project Metadata' section includes fields for Group (com.example), Artifact (task1), Name (task1), Description (Demo project for Spring Boot), and Package name (com.example.task1). The 'Packaging' section shows 'Jar' selected and 'Java' version '17' selected. On the right, the 'Dependencies' section lists 'Thymeleaf' (TEMPLATE ENGINES), 'Spring Web' (WEB), and 'Spring Boot DevTools' (DEVELOPER TOOLS). A button 'ADD DEPENDENCIES... CTRL + B' is visible at the top right of the dependencies section.

Рисунок 1 – инициализация проекта.

- Thymeleaf – шаблонизатор, позволяющий вставлять java-код в html верстку;
- Spring web – создание веб приложений, с использованием spring mvc;
- DevTools – инструменты для разработки.

Далее сделаем файл с разметкой в каталоге `resources/templates`

```
1 <!DOCTYPE html>
2 <html lang="en" xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <meta charset="UTF-8">
5 <title>Title</title>
6 </head>
7 <body>
8 <form action="getcalculate" method="get">
9 <label>Get-калькулятор :</label>
10 <input placeholder="Первый аргумент" type="number" name="aop">
11 <input placeholder="Второй аргумент" type="number" name="bop">
12 <select name="operators">
13 <option value="add">Сложение</option>
14 <option value="dif">Вычитание</option>
15 <option value="mul">Умножение</option>
16 <option value="div">Деление</option>
17 </select>
18 <button type="submit">Вычислить</button>
19 </form>
20
21 <form action="postcalculate" method="post">
22 <label>Post-калькулятор:</label>
23 <input placeholder="Первый аргумент" type="number" name="aop">
24 <input placeholder="Второй аргумент" type="number" name="bop">
25 <select name="operators">
26 <option value="add">Сложение</option>
27 <option value="dif">Вычитание</option>
28 <option value="mul">Умножение</option>
29 <option value="div">Деление</option>
30 </select>
31 <button type="submit">Вычислить</button>
32 </form>
33 <p th:text="${result}"></p>
34 </body>
35 </html>
```

Рисунок 2 – разметка.

- `xmlns:th=http://www.thymeleaf.org` – подключение thymeleaf к разметке;
- `th:text="${result}"` – вставка текста из модели.

Вот так разметка выглядит в браузере

Рисунок 3 – разметка в браузере.

Далее создадим каталог `controllers` и добавим в него контроллер `MainController` и определим в нём метод `Index`, который будет отображать страницу с формами

```

@Controller
public class MainController {
    @GetMapping("/")
    public String Index(){
        return "index";
    }
}

```

Рисунок 4 – метод контроллера.

Далее определяем метод, который будет обрабатывать get-запрос первого калькулятора.

```

@GetMapping("/getcalculate")
public String GetCalculate(@RequestParam(required = false) Integer aop,
                           @RequestParam(required = false) Integer bop,
                           @RequestParam(required = false) String operators,
                           Model model){

    String op = "";
    Integer result = 0;
    switch (operators){
        case "add":
            op = "+";
            result = aop + bop;
            break;
        case "dif":
            op = "-";
            result = aop - bop;
            break;
        case "mul":
            op = "*";
            result = aop * bop;
            break;
        case "div":
            op = "/";
            result = aop / bop;
            break;
    }

    model.addAttribute("result", String.format("%d %s %d = %d", aop, op, bop, result));
    return "index";
}

```

Рисунок 5 – get-метод котроллера калькулятора.

- GetMapping – определение маршрута для get-запроса для выполнения этого метода контроллера;
- PostMapping – определение маршрута для post-запроса для выполнения этого метода контроллера;
- RequestParam – параметры, передаваемые в метод с запросом (required = false – означает необязательность указания параметра);

- Model model – модель, через которую мы передаем данные в представление.

И аналогичный метод для post-запроса

```
@PostMapping("postcalculate")
public String PostCalculate(@RequestParam(required = false) Integer aop,
                           @RequestParam(required = false) Integer bop,
                           @RequestParam(required = false) String operators,
                           Model model){

    String op = "";
    Integer result = 0;
    switch (operators){
        case "add":
            op = "+";
            result = aop + bop;
            break;
        case "dif":
            op = "-";
            result = aop - bop;
            break;
        case "mul":
            op = "*";
            result = aop * bop;
            break;
        case "div":
            op = "/";
            result = aop / bop;
            break;
    }

    model.addAttribute("result", String.format("%d %s %d = %d", aop, op, bop, result));

    return "index";
}
```

Рисунок 6 – post-метод контроллера калькулятора.

Результат:

Get-калькулятор : 20 3 Умножение Вычислить

Post-калькулятор: Первый аргумент Второй аргумент Сложение Вычислить

20 \* 3 = 60

Рисунок 7 – результат работы калькулятора.

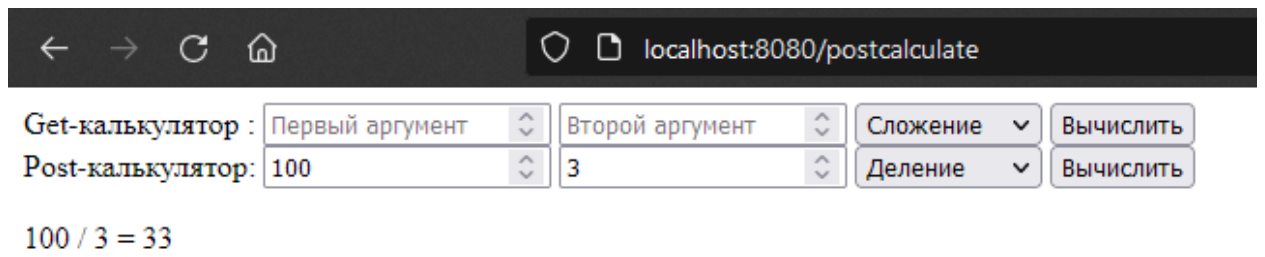


Рисунок 8 – результат работы калькулятора.

Итог: Программа успешно работает, а я научился работать с методами get и post.

## Практическая работа №2.

Тема: Знакомство с Bootstrap. Взаимодействие базы данных с сайтом. Создание сайта с табличным выводом и добавлением новых данных.

Цель работы: Необходимо сделать добавление и вывод данных для 5 таблиц. Данные из БД обязательно должны выводиться в <table>. К таблице, полям ввода, кнопкам должны быть применены стили.

Для быстрой верстки было решено изучить и в дальнейшем использовать Bootstrap5.

Для подключения приложения к бд, для начала необходимо в файле конфигурации прописать следующее:

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/task2db
spring.datasource.username=mysql
spring.datasource.password=mysql
```

Рисунок 9 – конфигурационный файл.

Первая строка необходима для автоматического обновления структуры бд по созданным моделям, вторая строка отвечает за адрес подключения к бд, а оставшиеся строки – это логин и пароль.

Также необходимо подключить две зависимости, отвечающие за работу с mysql.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
```

Рисунок 10 – зависимости.

Далее необходимо создать классы моделей, которые отражают структуру бд. Один из них:

```
@Entity
public class Author {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String birthday;

    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 getBirthday() {
        return birthday;
    }

    public void setBirthday(String birthday) {
        this.birthday = birthday;
    }

    public Author(String name, String birthday) {
        this.name = name;
        this.birthday = birthday;
    }

    public Author() {
    }
}

```

- @Entity – означает, что данный класс представляет сущность в бд;
- @Id – означает, что данное поле является первичным ключом;
- @GeneratedValue(strategy = GenerationType.IDENTITY) – означает, что данное поле будет автоматически заполняться и увеличиваться на 1;
- Для корректной работы необходимо также написать геттеры и сеттеры и конструкторы для класса.

Далее создаем интерфейс репозитория для модели, который будет отвечать за основные операции с данными.

```

package com.example.task2.repos;

import com.example.task2.models.Author;
import org.springframework.data.repository.CrudRepository;

public interface AuthorRepository extends CrudRepository<Author, Long> {
}

```

Рисунок 11 – репозиторий модели.

Далее делаем разметку страницы с использованием bootstrap5.

```

<!doctype html>
<html lang="ru" xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Практическая №2</title>
    <meta charset="utf-8">

```

```

    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOMmLASjC"
crossorigin="anonymous">
</head>
<body>
    <div class="container-fluid bg-dark text-white text-center p-1">
        <h3>Практическая №2</h3>
    </div>

    <!-- Авторы книг -->
    <div class="container border bg-light p-4 my-2">
        <table class="table caption-top table-light table-hover table-striped">
            <caption>Авторы книг</caption>
            <thead class="table-dark">
                <tr>
                    <th scope="col">Id</th>
                    <th scope="col">ФИО</th>
                    <th scope="col">Дата рождения</th>
                </tr>
            </thead>
            <tbody>
                <tr th:each="author : ${authors}">
                    <th scope="row" th:text="${author.id}"></th>
                    <td th:text="${author.name}"></td>
                    <td th:text="${author.birthday}"></td>
                </tr>
            </tbody>
        </table>

        <form action="authorAdd" method="post" class="row p-3">
            <h6 class="text-muted">Добавить автора</h6>
            <div class="col">
                <input type="text" name="authorName" class="form-control" placeholder="ФИО
Автора">
            </div>
            <div class="col-3">
                <input type="date" name="authorBirth" class="form-control">
            </div>
            <button type="submit" class="btn btn-warning col-2">Добавить</button>
        </form>
    </div>

    <!-- Книги -->
    <div class="container border bg-light p-4 my-2">
        <table class="table caption-top table-light table-hover table-striped">
            <caption>Книги</caption>
            <thead class="table-dark">
                <tr>
                    <th scope="col">Id</th>
                    <th scope="col">Наименование</th>
                    <th scope="col">Описание</th>
                    <th scope="col">Автор</th>
                </tr>
            </thead>
            <tbody>
                <tr th:each="book : ${books}">
                    <th scope="row" th:text="${book.id}"></th>
                    <td th:text="${book.name}"></td>
                    <td th:text="${book.description}"></td>
                    <td th:text="${book.author}"></td>
                </tr>
            </tbody>
        </table>

        <form action="bookAdd" method="post" class="row p-2">
            <h6 class="text-muted">Добавить книгу</h6>

```

```

        <div class="col-2">
            <input type="text" name="bookName" class="form-control"
placeholder="Наименование книги">
        </div>
        <div class="col-3">
            <select name="bookAuthors" class="form-select">
                <option selected>Автор книги</option>
                <option th:each="author : ${authors}" th:value="${author.name}"
th:text="${author.name}"></option>
            </select>
        </div>
        <div class="col">
            <input type="text" name="bookDescription" class="form-control"
placeholder="Описание"></textarea>
        </div>
        <button type="submit" class="btn btn-warning col-2">Добавить</button>
    </form>
</div>

<!-- ОТЗЫВЫ КНИГ -->
<div class="container border bg-light p-4 my-2">
    <table class="table caption-top table-light table-hover table-striped">
        <caption>Отзывы книг</caption>
        <thead class="table-dark">
            <tr>
                <th scope="col">ФИО</th>
                <th scope="col">Книга</th>
                <th scope="col">Комментарий</th>
                <th scope="col">Оценка</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="report : ${reportBooks}">
                <td th:text="${report.name}"></td>
                <td th:text="${report.bookName}"></td>
                <td th:text="${report.comm}"></td>
                <td th:text="${report.grade}"></td>
            </tr>
        </tbody>
    </table>

    <form action="reportAdd" method="post" class="row p-2">
        <h6 class="text-muted">Добавить отзыв</h6>
        <div class="col-2">
            <input type="text" name="reportName" class="form-control" placeholder="ФИО">
        </div>
        <div class="col-3">
            <select name="reportBooks" class="form-select">
                <option selected>Книга</option>
                <option th:each="book : ${books}" th:value="${book.name}"
th:text="${book.name}"></option>
            </select>
        </div>
        <div class="col">
            <input type="text" name="reportComm" class="form-control"
placeholder="Комментарий"></textarea>
        </div>
        <div class="col-2">
            <select name="reportGrade" class="form-select">
                <option selected>Оценка</option>
                <option value="1">1</option>
                <option value="2">2</option>
                <option value="3">3</option>
                <option value="4">4</option>
                <option value="5">5</option>
            </select>
        </div>
        <button type="submit" class="btn btn-warning col-2">Добавить</button>
    </form>
</div>

```

```

<!-- Студенты -->
<div class="container border bg-light p-4 my-2">
  <table class="table caption-top table-light table-hover table-striped">
    <caption>Студенты</caption>
    <thead class="table-dark">
      <tr>
        <th scope="col">Id</th>
        <th scope="col">ФИО</th>
        <th scope="col">Дата рождения</th>
        <th scope="col">Группа</th>
      </tr>
    </thead>
    <tbody>
      <tr th:each="student : ${students}">
        <th scope="row" th:text="${student.id}"></th>
        <td th:text="${student.name}"></td>
        <td th:text="${student.birthday}"></td>
        <td th:text="${student.group}"></td>
      </tr>
    </tbody>
  </table>

  <form action="studentAdd" method="post" class="row p-3">
    <h6 class="text-muted">Добавить студента</h6>
    <div class="col">
      <input type="text" name="studentName" class="form-control" placeholder="ФИО
Студента">
    </div>
    <div class="col-3">
      <input type="date" name="studentBirth" class="form-control">
    </div>
    <div class="col-2">
      <select name="studentGroups" class="form-select">
        <option selected>Группа</option>
        <option value="П50-1-18">П50-1-18</option>
        <option value="П50-2-18">П50-2-18</option>
        <option value="П50-1-19">П50-1-19</option>
        <option value="П50-2-19">П50-2-19</option>
        <option value="ИС50-1-18">ИС50-1-18</option>
      </select>
    </div>
    <button type="submit" class="btn btn-warning col-2">Добавить</button>
  </form>
</div>

<!-- Оценки студентов -->
<div class="container border bg-light p-4 my-2">
  <table class="table caption-top table-light table-hover table-striped">
    <caption>Оценки студентов</caption>
    <thead class="table-dark">
      <tr>
        <th scope="col">Студент</th>
        <th scope="col">Дисциплина</th>
        <th scope="col">Оценка</th>
      </tr>
    </thead>
    <tbody>
      <tr th:each="grade : ${gradeStudents}">
        <td th:text="${grade.student}"></td>
        <td th:text="${grade.discipline}"></td>
        <td th:text="${grade.grade}"></td>
      </tr>
    </tbody>
  </table>

  <form action="gradeAdd" method="post" class="row p-3">
    <h6 class="text-muted">Добавить оценку</h6>
    <div class="col">

```

```

        <select name="gradeStudents" class="form-select">
            <option>Студент</option>
            <option th:each="student : ${students}" th:value="${student.name}"
th:text="${student.name}"></option>
        </select>
    </div>
    <div class="col">
        <select name="gradeDiscipline" class="form-select">
            <option>Дисциплина</option>
            <option>Разработка программных модулей</option>
            <option>Разработка и защита баз данных</option>
            <option>Тестирование программных модулей</option>
            <option>Основы алгоритмизации и программирования</option>
            <option>Математическое моделирование</option>
        </select>
    </div>
    <div class="col-2">
        <select name="grades" class="form-select">
            <option selected>Оценка</option>
            <option value="1">1</option>
            <option value="2">2</option>
            <option value="3">3</option>
            <option value="4">4</option>
            <option value="5">5</option>
        </select>
    </div>
    <button type="submit" class="btn btn-warning col-2">Добавить</button>
</form>
</div>

<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
integrity="sha384-IQsoLX15PILPhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"
integrity="sha384-cVKIPhGWiC2Al4u+LWgxfKTRICfu0JtXr+EQDz/bglldoEyl4H0zUF0QKbrJ0EcQF"
crossorigin="anonymous"></script>
</body>
</html>

```

После создания разметки можем приступить к написанию контроллера.

```

@Controller
public class MainController {
    @Autowired
    private AuthorRepository authorRepository;
    @Autowired
    private BookRepository bookRepository;
    @Autowired
    private ReportBookRepository reportBookRepository;
    @Autowired
    private StudentRepository studentRepository;
    @Autowired
    private GradeStudentRepository gradeStudentRepository;

    @GetMapping("/")
    public String Index(Model model){
        Iterable<Author> authors = authorRepository.findAll();
        model.addAttribute("authors", authors);
        Iterable<Book> books = bookRepository.findAll();
        model.addAttribute("books", books);
        Iterable<ReportBook> reportBooks = reportBookRepository.findAll();
        model.addAttribute("reportBooks", reportBooks);
        Iterable<StudentData> students = studentRepository.findAll();
        model.addAttribute("students", students);
        Iterable<GradeStudent> gradeStudents = gradeStudentRepository.findAll();
        model.addAttribute("gradeStudents", gradeStudents);

        return "index";
    }
}

```

```

    }

    @PostMapping("authorAdd")
    public String AuthorAdd(@RequestParam String authorName,
                            @RequestParam String authorBirth){
        Author author = new Author(authorName, authorBirth);
        authorRepository.save(author);

        return "redirect:/";
    }

    @PostMapping("bookAdd")
    public String BookAdd(@RequestParam String bookName,
                          @RequestParam String bookAuthors,
                          @RequestParam String bookDescription){
        Book book = new Book(bookName, bookDescription, bookAuthors);
        bookRepository.save(book);

        return "redirect:/";
    }

    @PostMapping("reportAdd")
    public String ReportAdd(@RequestParam String reportName,
                           @RequestParam String reportBooks,
                           @RequestParam String reportComm,
                           @RequestParam String reportGrade){
        ReportBook reportBook = new ReportBook(reportName, reportBooks, reportComm,
reportGrade);
        reportBookRepository.save(reportBook);

        return "redirect:/";
    }

    @PostMapping("studentAdd")
    public String StudentAdd(@RequestParam String studentName,
                             @RequestParam String studentBirth,
                             @RequestParam String studentGroups){
        StudentData studentData = new StudentData(studentName, studentBirth,
studentGroups);
        studentRepository.save(studentData);

        return "redirect:/";
    }

    @PostMapping("gradeAdd")
    public String GradeAdd(@RequestParam String gradeStudents,
                           @RequestParam String gradeDiscipline,
                           @RequestParam String grades){
        GradeStudent gradeStudent = new GradeStudent(gradeStudents, gradeDiscipline,
grades);
        gradeStudentRepository.save(gradeStudent);

        return "redirect:/";
    }
}

```

- @Autowired – подключение репозитория к контроллеру;
- Iterable<Author> authors = authorRepository.findAll() – с помощью репозитория получаем список всех элементов данной модели;
- authorRepository.save(author) – с помощью репозитория добавляем в бд новый объект.

Результат работы программы:

Авторы книг

| Id | ФИО                     | Дата рождения |
|----|-------------------------|---------------|
| 1  | Антуан де Сент-Экзюпери | 1900-06-29    |
| 2  | Джоан Роулинг           | 1965-07-31    |

Добавить автора

Добавить

Книги

| Id | Наименование                      | Описание  | Автор                   |
|----|-----------------------------------|---|-------------------------|
| 1  | Маленький принц                   | Фантастическое приключение от французского писателя | Антуан де Сент-Экзюпери |
| 2  | Гарри Поттер и философский камень | Приключение юного мага                              | Джоан Роулинг           |

Добавить книгу

Автор книги

Добавить

Отзывы книг

| ФИО                  | Книга                             | Комментарий     | Оценка |
|----------------------|-----------------------------------|-----------------|--------|
| Иванов Иван          | Маленький принц                   | Отличная книга! | 5      |
| Дубов Алёша Иванович | Гарри Поттер и философский камень | Не реалистично  | 3      |
| Петров Илья          | Гарри Поттер и философский камень | Неплохо         | 4      |

Добавить отзыв

Книга

Оценка

Добавить

Студенты

| Id | ФИО                         | Дата рождения | Группа   |
|----|-----------------------------|---------------|----------|
| 1  | Суслин Александр Михайлович | 2003-02-08    | П50-2-18 |
| 2  | Максимов Андрей             | 2001-09-11    | П50-1-18 |

Добавить студента

Группа

Добавить

Оценки студентов

| Студент                     | Дисциплина                               | Оценка |
|-----------------------------|--|--------|
| Суслин Александр Михайлович | Разработка программных модулей           | 5      |
| Максимов Андрей             | Математическое моделирование             | 5      |
| Суслин Александр Михайлович | Математическое моделирование             | 2      |
| Максимов Андрей             | Основы алгоритмизации и программирования | 4      |

Добавить оценку

Студент

Дисциплина

Оценка

Добавить

Рисунок 12 – результаты работы программы.

Вывод: в ходе выполнения данной практической работы я изучил bootstrap, работу с бд в java spring, научился выводить данные из бд в таблицу.



### Практическая работа №3.

Тема: Создание поиска и вывод подробной информации.

Цель работы: создать функционал в веб приложении, который предусматривает наличие двух видов поиска: точечный и поиск по вхождению, а также сделать вывод отдельной записи с дополнительной информацией, получить новые знания о веб разработке на языке Java с помощью фреймворка Spring.

Код контроллера:

```
@GetMapping("/search-author")
public String SearchAuthor(){
    return "search-author";
}

@PostMapping("/search-author/result")
public String SearchAuthorResult(Model model,
                                @RequestParam String searchName){

    Iterable<Author> authors =
authorRepository.findByNameContainingIgnoreCase(searchName);
    model.addAttribute("authors", authors);

    return "/search-author";
}

@GetMapping("/author/{id}")
public String MoreAuthor(Model model,
                        @PathVariable(value = "id") long id){
    Optional<Author> author = authorRepository.findById(id);
    ArrayList<Author> res = new ArrayList<>();
    author.ifPresent(res::add);
    model.addAttribute("author", res);
    Iterable<Book> books = bookRepository.findByAuthor(res.get(0).getName());
    model.addAttribute("books", books);
    return "more-author";
}
```

- @PathVariable – аннотация, с помощью которой осуществляется получения переменной по ключу из строки запроса.

Для того, чтобы получить доступ к страницам поиска и детального отображения, необходимо создать методы с маршрутизацией, где будут возвращаться эти самые страницы. Далее необходимо создать методы, в которых мы будем осуществлять поиск по введенному пользователю значению, передаем в метод требуемый параметр для осуществления поиска, после чего передаем в переменную типа List найденные значения через репозиторий по двум методам, зависимо от типа поиска: findBy{поле, по которому ищем записи}(то, что ввел пользователь) данный метод используется для точечного поиска, а findBy{ поле, по которому ищем записи }ContainingIgnoreCase (то, что ввел пользователь) представляет собой поиск по вложенности с игнорированием верхнего и нижнего регистра.

Для вывода подробной информации об одной записи, необходимо написать методы, в строке которых будут передаваться id записей, после чего осуществляем выборку из всех записей по передаваемому id через метод

репозитория `findById`(значение `id`), осуществляем конвертацию полученной записи и передаем в модель через ключ.

Код репозитория:

```
public interface BookRepository extends CrudRepository<Book, Long> {  
    List<Book> findByNameContainingIgnoreCase(String name);  
    List<Book> findByAuthor(String name);  
}
```

Добавили 2 метода, которые зависят от типа поиска: `findBy{поле, по которому ищем записи}` (то, что ввел пользователь) данный метод используется для точечного поиска, а `findBy{поле, по которому ищем записи}ContainingIgnoreCase` (то, что ввел пользователь) представляет собой поиск по вложенности с игнорированием верхнего и нижнего регистра.

Верстка страницы поиска:

```
<!doctype html>  
<html lang="ru" xmlns:th="http://www.thymeleaf.org">  
<head>  
    <title>Учебная практика</title>  
    <meta charset="utf-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">  
    <link rel="stylesheet"  
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"  
integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTWfSpd3yD65VohhpuuCOMLASjC"  
crossorigin="anonymous">  
</head>  
<body>  
    <div class="container-fluid bg-dark text-white text-center p-1">  
        <h3>Поиск авторов книг</h3>  
    </div>  
  
    <div class="container my-3">  
        <form class="row" action="/search-author/result" method="post">  
            <input class="form-control col mx-1" type="text" name="searchName"  
placeholder="ФИО Автора">  
  
            <button class="btn btn-warning col-3 mx-1" type="submit">  
                Поиск  
                <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16"  
fill="currentColor" class="bi bi-search" viewBox="0 0 16 16">  
                    <path d="M11.742 10.344a6.5 6.5 0 1 0-1.397 1.398h-.001c.03.04.062.078.098.115l3.85 3.85a1 1 0 0 0 1.415-1.414l-3.85-3.85a1.007 1.007 0 0 0-.115-.115M12 6.5a5.5 5.5 0 1 1-11 0 5.5 5.5 0 0 1 11 0z"/>  
                </svg>  
            </button>  
  
        </form>  
  
        <div th:each="author : ${authors}" class="row border bg-light p-2 my-3">  
            <h3 class="col mx-1" th:text="${author.name}"></h3>  
            <div class="col-3 d-flex align-items-center justify-content-center">  
                <a th:href="'/author/'+${author.id}" class="px-4 text-center btn btn-warning text-decoration-none">  
                    Подробнее  
                </a>  
            </div>  
        </div>  
    </div>
```

```

<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
integrity="sha384-IQsoLX15PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"
integrity="sha384-cVKIPhGWiC2Al4u+LWgxfKTRlcfu0JTxR+EQDz/bglodoEyl4H0zUF0QKbrJ0EcQF"
crossorigin="anonymous"></script>
</body>
</html>

```

## Верстка страницы подробной информации:

```

<!doctype html>
<html lang="ru" xmlns:th=http://www.thymeleaf.org>
<head>
  <title>Учебная практика</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTWfSpd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
</head>
<body>
  <div class="container-fluid bg-dark text-white text-center p-1">
    <h3>Подробная информация об авторе</h3>
  </div>

  <div th:each="a : ${author}" class="container my-4 py-4 border bg-light text-center">
    <h2 th:text="${a.name}"></h2>
    <h4 th:text="'Дата рождения: '+${a.birthday}" class="my-4 text-muted"></h4>
    <hr>
    <h3>Написанные книги:</h3>
    <h4 th:each="book : ${books}" th:text="${book.name}" class="mt-3"></h4>
  </div>

  <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
integrity="sha384-IQsoLX15PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"
crossorigin="anonymous"></script>
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"
integrity="sha384-cVKIPhGWiC2Al4u+LWgxfKTRlcfu0JTxR+EQDz/bglodoEyl4H0zUF0QKbrJ0EcQF"
crossorigin="anonymous"></script>
</body>
</html>

```

- `th:href` – отвечает за ссылку.

## Результат работы программы:

| Поиск авторов книг |           |
|--------------------|-----------|
| Джоан              | Поиск Q   |
| Джоан Роулинг      | Подробнее |

| Поиск книг                        |           |
|-----------------------------------|-----------|
| Гарри                             | Поиск Q   |
| Гарри Поттер и философский камень | Подробнее |
| Гарри Поттер и тайная комната     | Подробнее |
| Гарри Поттер и узник азкабана     | Подробнее |

| Поиск отзывов книг                |                      |           |
|-----------------------------------|----------------------|-----------|
| Гарри                             | Поиск Q              |           |
| Гарри Поттер и философский камень | Дубов Алёша Иванович | Подробнее |
| Гарри Поттер и философский камень | Петров Илья          | Подробнее |

| Поиск студентов             |           |
|-----------------------------|-----------|
| Суслин                      | Поиск Q   |
| Суслин Александр Михайлович | Подробнее |

| Поиск оценок студентов      |                                  |           |
|-----------------------------|----------------------------------|-----------|
| ФИО Студента                | Поиск Q                          |           |
| Суслин Александр Михайлович | Разработка программных модулей   | Подробнее |
| Суслин Александр Михайлович | Математическое моделирование     | Подробнее |
| Суслин Александр Михайлович | Тестирование программных модулей | Подробнее |

| Подробная информация об авторе |  |
|--------------------------------|--|
| Антуан де Сент-Экзюпери        |  |
| Дата рождения: 1900-06-29      |  |
| Написанные книги:              |  |
| Маленький принц                |  |
| Планета людей                  |  |

| Подробная информация о книге  |  |
|---|--|
| Планета людей   |  |
| Автор: Антуан де Сент-Экзюпери  |  |
| Документально-публицистическая книга писателя Антуана де Сент-Экзюпери, опубликованная в 1939 году. Книга посвящена лётчику Анри Гийоме. Повествование ведётся от первого лица. |  |

| Подробная информация об отзыве                    |
|---|
| <div>Маленький принц</div> <div>Иванов Иван</div> |
| <div>Оценка: 5</div> <div>Отличная книга!</div>   |

| Подробная информация о студенте   |
|---|
| <div>Суслин Александр Михайлович</div> <div>Дата рождения: 2003-02-08</div> |
| <div>Группа: П50-2-18</div>   |

| Подробная информация об оценке студента  |
|--|
| <div>Суслин Александр Михайлович</div> <div>Математическое моделирование</div> |
| <div>Оценка: 2</div>   |

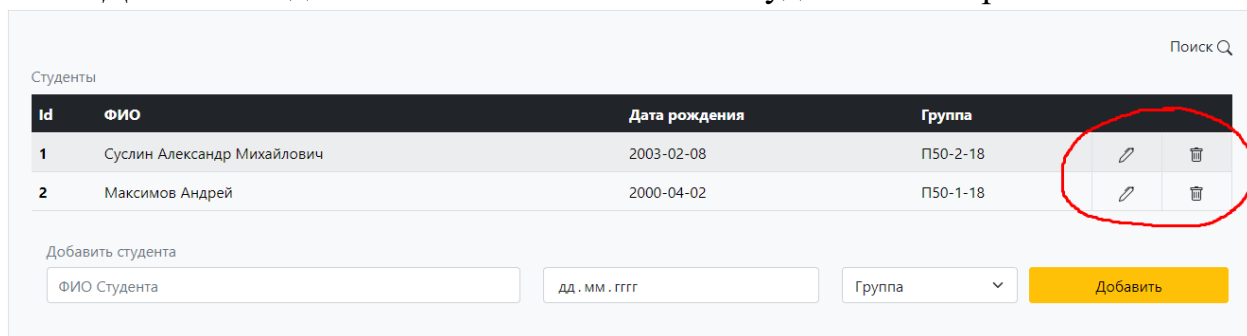
Рисунок 13 – результаты работы программы.

Вывод: в ходе выполнения данной практической работы я создал функционал в веб приложении, который предусматривает наличие двух видов поиска: точечный и поиск по вхождению, а также сделал вывод отдельной записи с дополнительной информацией, получил новые знания о веб разработке на языке Java с помощью фреймворка Spring.

## Практическая работа №4.

Тема: Создание функционала для редактирования и удаления выбранных записей.

Для начала добавим кнопки изменения и удаления выбранных записей.



| Id | ФИО                         | Дата рождения | Группа   |  |  |
|----|-----------------------------|---------------|----------|--|--|
| 1  | Суслин Александр Михайлович | 2003-02-08    | П50-2-18 |  |  |
| 2  | Максимов Андрей             | 2000-04-02    | П50-1-18 |  |  |

Добавить студента

Рисунок 14 – разметка для редактирования и удаления.

Далее создаем разметку для страницы изменения выбранной записи.

```
<!doctype html>
<html lang="ru" xmlns:th=http://www.thymeleaf.org>
<head>
  <title>Учебная практика</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" integrity="sha
384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTWfSpd3yD65VohhpuuC0mLASjC" crossorigin="anonymous">
</head>
<body>
  <div class="container-fluid bg-dark text-white text-center p-1">
    <h3>Редактирование записи о студенте</h3>
  </div>

  <div th:each="s : ${student}" class="container border bg-light p-3 my-4">
    <form method="post" class="row">
      <div class="col">
        <input th:value="${s.name}" type="text" name="studentName" class="form-
control" placeholder="ФИО Студента">
      </div>
      <div class="col-3">
        <input th:value="${s.birthday}" type="date" name="studentBirth" class="form-
control">
      </div>
      <div class="col-2">
        <select name="studentGroups" class="form-select">
          <option th:value="${s.group}" th:text="${s.group}" selected></option>
          <option value="П50-1-18">П50-1-18</option>
          <option value="П50-2-18">П50-2-18</option>
          <option value="П50-1-19">П50-1-19</option>
          <option value="П50-2-19">П50-2-19</option>
          <option value="ИС50-1-18">ИС50-1-18</option>
```

```

        </select>
    </div>

    <div class="text-center mt-4">
        <button class="btn btn-warning px-5" type="submit">Изменить</button>
    </div>
</form>
</div>

<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
integrity="sha384-IQsoLX15PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"
crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"
integrity="sha384-cVKIPhGWiC2Al4u+LWgxfKTRIcfu0JTxR+EQDz/bgldoEyl4N0zUf0QKbRj0EcQF"
crossorigin="anonymous"></script>
</body>
</html>

```

- `th:value="{s.name}"` – подставляет в поле для ввода нужное значение.

Далее создаем метод в контроллере для удаления выбранной записи.

```

@GetMapping("/student/delete/{id}")
public String DeleteStudent(@PathVariable(value = "id") long id){
    StudentData student = studentRepository.findById(id).orElseThrow();
    studentRepository.delete(student);
    return "redirect:/";
}

```

- `studentRepository.findById(id).orElseThrow()` – находим объект по `id` с помощью репозитория. Если такого объекта нету – выбрасывается исключение;
- `studentRepository.delete(student)` – удаляем запись из бд.

Далее создаем методы для обработки страницы редактирования записи.

```

@GetMapping("/student/edit/{id}")
public String EditStudent(@PathVariable(value = "id") long id,
    Model model){
    Optional<StudentData> student = studentRepository.findById(id);
    ArrayList<StudentData> res = new ArrayList<>();
    student.ifPresent(res::add);
    model.addAttribute("student", res);

    return "edit-student";
}

@PostMapping("/student/edit/{id}")
public String EditStudent(@PathVariable(value = "id") long id,
    @RequestParam String studentName,
    @RequestParam String studentBirth,
    @RequestParam String studentGroups){
    StudentData student = studentRepository.findById(id).orElseThrow();
    student.setName(studentName);
}

```

```

student.setBirthday(studentBirth);
student.setGroup(studentGroups);
studentRepository.save(student);

return "redirect:/";
}

```

- Обрабатываем get и post запросы к странице редактирования записей, ничего нового.

Результат работы программы:

Редактирование записи об авторе

Изменить

Редактирование записи о книге

Изменить

Редактирование записи об отзыве

Изменить

Редактирование записи о студенте

Изменить

Редактирование записи об оценке студента

Изменить

Студенты

Поиск

| Id | ФИО                         | Дата рождения | Группа   |  |  |
|----|-----------------------------|---------------|----------|--|--|
| 1  | Суслин Александр Михайлович | 2003-02-08    | П50-2-18 |  |  |
| 2  | Максимов Андрей             | 2000-04-02    | П50-1-18 |  |  |

Добавить студента

Добавить

Рисунок 15 – результаты работы программы.



Вывод: В ходе выполнения данной практической работы я научился создавать функционал для редактирования и удаления выбранных записей в бд и закрепил знания из предыдущих работ.

## Практическая работа №5.

Тема: Валидация данных.

Цель работы: ознакомиться с валидацией в Spring и научиться ее применять, получить новые знания о разработке веб приложения на языке Java с помощью фреймворка Spring.

Для начала подключаем нужные зависимости.

```
<dependency>
  <groupId>org.hibernate.validator</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>6.2.0.Final</version>
</dependency>
```

Далее добавляем к моделям аннотации валидации.

```
public class UserData {

    @Size(min = 2, max = 20, message = "Строка должна быть в диапазоне от 2-ух до 20-ти символов")
    @NotEmpty
    private String lastName;

    @Size(min = 2, max = 20, message = "Строка должна быть в диапазоне от 2-ух до 20-ти символов")
    @NotEmpty
    private String firstName;

    @Min(value = 18, message = "К сожалению, наши пользователи должны быть 18+")
    private Integer age;
```

- @NotEmpty – аннотация, которая проверяет на пустоту и null значение;
- @Size - аннотация, которая проверяет длину поле, можно задать максимальную и минимальную длину, а также сообщение в случае не прохождения валидации;
- @NotNull - аннотация, которая запрещает null значение поля;
- @Null – аннотация, которая допускает null значение поля;
- @Min – аннотация, которая устанавливает минимальное значение поля;
- @Max- аннотация, которая устанавливает максимальное значение поля;
- @Pattern – аннотация, которая задает формат поля.

Код метода контроллера, обрабатывающий и валидирующий форму:

```

@PostMapping("/registry")
public String RegistryUser(@Valid UserData userData, BindingResult bindingResult){
    if(bindingResult.hasErrors()){
        return "registry";
    }

    if(contactsRepo.findByPhone(userData.getPhone()) != null){
        ObjectError objectError = new ObjectError( objectName: "phone", defaultMessage: "Такой телефон уже зарегистрирован.");
        bindingResult.addError(objectError);
        return "registry";
    }

    if(userRepo.findByLogin(userData.getLogin()) != null){
        ObjectError objectError = new ObjectError( objectName: "login", defaultMessage: "Пользователь с таким логином уже существует.");
        bindingResult.addError(objectError);
        return "registry";
    }

    PersonalInfo personalInfo = new PersonalInfo(userData.getFirstName(), userData.getLastName(), userData.getBirthDay());
    personalInfoRepo.save(personalInfo);
    Contacts contacts = new Contacts(userData.getPhone(), userData.getEmail());
    contactsRepo.save(contacts);
    User user = new User(userData.getLogin(), userData.getPassword(), active: true, personalInfo, contacts);
    user.setPassword(new BCryptPasswordEncoder( strength: 8).encode(user.getPassword()));
    userRepo.save(user);

    return "redirect:/login";
}

```

В методах добавления и изменения записи изменили передачу модели, обращаясь непосредственно к модели. В методах добавления делаем кастомную валидацию, которая проверяет заголовки и имена на уникальное значение, в случае нахождения подобной записи, возвращается сообщение и не дает создать запись, в случае успешного прохождения валидации сохраняем запись. Аналогичные действия происходят и в методах изменения.

- @Valid – аннотация, связанная с валидацией модели.

Код разметки:

```

<!doctype html>
<html lang="ru" xmlns:th=http://www.thymeleaf.org>
<head>
    <title>Регистрация</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
</head>
<body>

    <div class="container-sm my-5 bg-light border px-3 py-5 d-flex flex-column
justify-content-center align-items-center">
        <div class="container-fluid d-flex justify-content-start">
            <a class="btn btn-outline-dark px-3" th:href="@{/}"
href="index.html">Главная</a>
        </div>
        

        <form method="post" th:action="@{/registry}" th:object="${userData}">
            <div th:if="${#fields.hasErrors('global')}" class="text-center">
                <h5 th:each="err : ${#fields.errors('global')}" th:text="${err}"
class="text-danger">message error</h5>
            </div>

```

```

        <div class="d-flex flex-column align-items-center">
            <div class="p-0 row container-sm mb-2">
                <div class="p-0 pe-1 col">
                    <input required minlength="2" maxlength="20" class="col form-control"
th:field="*{lastName}" name="lastName" type="text" placeholder="Фамилия">
                </div>
                <div class="p-0 ps-1 col">
                    <input required minlength="2" maxlength="20" class="col form-control"
th:field="*{firstName}" name="firstName" type="text" placeholder="Имя">
                </div>
            </div>

            <div class="p-0 row container-sm mb-2">
                <div class="p-0 pe-1 col">
                    <input required maxlength="40" class="col form-control"
th:field="*{email}" name="email" type="email" placeholder="Почта">
                </div>
                <div class="p-0 ps-1 col">
                    <input class="col form-control" th:field="*{phone}" name="phone"
type="text" required pattern="^((8|\+7)[\-\ ]?)?( \(?\d{3}\) )?[\-\ ]?)?[\d\-\ ]{7,10}$"
placeholder="Телефон">
                </div>
            </div>

            <div class="p-0 container-sm mb-2">
                <input required class="form-control" th:field="*{birthday}"
name="birthday" type="date" placeholder="Дата рождения">
            </div>

            <div class="p-0 container-sm mb-2">
                <input required minlength="4" maxlength="40" class="form-control"
th:field="*{login}" name="login" type="text" placeholder="Введите логин">
            </div>

            <div class="p-0 container-sm mb-2">
                <input required minlength="3" maxlength="40" class="form-control"
th:field="*{password}" name="password" type="password" placeholder="Введите пароль">
            </div>

            <div class="p-0 container-sm mb-2">
                <input required minlength="3" maxlength="40" class="form-control"
name="passwordRepeat" type="password" placeholder="Повторите пароль">
            </div>

            <input class="px-4 btn btn-warning mt-4" type="submit"
value="Зарегистрироваться">
        </div>
    </form>
    <a class="nav-link" href="/login">Войти</a>
</div>


<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
integrity="sha384-IQsoLX15PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"
integrity="sha384-cVKIPhGWiC2Al4u+LWgxfKTRlcfu0JTxR+EQDz/bglodoEyl4H0zUF0QKbrJ0EcQF"
crossorigin="anonymous"></script>
</body>
</html>

```

Теперь в верстке мы взаимодействуем непосредственно с моделью через `th:object`, а поля для работы с валидацией идентифицируем с помощью `th:field`. Также создали дополнительные контейнеры, в которых храним

системные и кастомные валидаторы, проверяем наличие ошибок с помощью `th:if` и вызываем их с помощью `th:errors`.

Результат работы:

 **DPRE**

Неверный логин и/или пароль.

Войти

Такой телефон уже зарегистрирован.

Зарегистрироваться

|                            |                                       |   |
|----------------------------|---------------------------------------|---|
| Имя                        | Фамилия                               | Имя пользователя                                      |
| <input type="text"/>       | <input type="text"/>                  | @ <input type="text"/>                                |
|                            |                                       | Пожалуйста, выберите имя пользователя.                |
| Город                      | Состояние                             | Почтовый индекс                                       |
| <input type="text"/>       | Выбирать... <input type="text"/>      | <input type="text"/>                                  |
| Укажите действующий город. | Пожалуйста, выберите допустимый штат. | Пожалуйста, предоставьте действующий почтовый индекс. |

Рисунок 16 – результаты работы программы.

Вывод: в ходе выполнения данной практической работы я изучил с валидацию в Spring и научился ее применять, получил новые знания о разработке веб приложения на языке Java с помощью фрейморка Spring.

## Практическая работа №6.

Тема: Связи таблиц.

Цель работы: ознакомиться со связями между таблицами в Spring и научиться их применять, получить новые знания о разработке веб приложения на языке Java с помощью фреймворка Spring.

Код контроллера, обрабатывающий таблицы со связями:

```
package com.example.dpre.controllers;

import com.example.dpre.data.AdvertData;
import com.example.dpre.models.*;
import com.example.dpre.repo.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import javax.validation.Valid;
import java.util.Optional;

@Controller
public class AdvertController {

    @Autowired
    private ObjectTypeRepo objectTypeRepo;
    @Autowired
    private HouseTypeRepo houseTypeRepo;
    @Autowired
    private UserRepo userRepo;
    @Autowired
    private AdvertRepo advertRepo;
    @Autowired
    private NearMetroRepo nearMetroRepo;
    @Autowired
    private AdvertConditionsRepo advertConditionsRepo;
    @Autowired
    private ApartmentInfoRepo apartmentInfoRepo;

    @GetMapping("/advertnew")
    public String AdvertNew(AdvertData advertData, Model model){
        model.addAttribute("objectTypes", objectTypeRepo.findAll());
        model.addAttribute("houseTypes", houseTypeRepo.findAll());
        model.addAttribute("userRepo", userRepo);

        return "advertnew";
    }

    @PostMapping("/advertnew")
    public String AdvertAdd(@Valid AdvertData advertData, BindingResult
bindingResult){
        if(bindingResult.hasErrors()){
            return "advertnew";
        }

        EstateObjectType estateObjectType =
objectTypeRepo.findByName(advertData.getEstateObjectType());
        Advert advert = new Advert(
            advertData.getAddress(),
            advertData.getDealType(),
            advertData.getRentType(),
            advertData.getTitle(),
            advertData.getDescription(),
```

```

userRepo.findById(Long.parseLong(advertData.getUserId())).orElseThrow(),
    estateObjectType);
advertRepo.save(advert);

NearMetro nearMetro = new NearMetro(
    advertData.getMetroName(),
    advertData.getTransportType(),
    advertData.getMetroMinutes(),
    advert);
nearMetroRepo.save(nearMetro);

AdvertConditions advertConditions = new AdvertConditions(
    advertData.getPrice(),
    advertData.getCommunal().equals("Включена"),
    advertData.getPrepayment(),
    advertData.getDeposit(),
    advert);
advertConditionsRepo.save(advertConditions);

HouseType houseType = houseTypeRepo.findByName(advertData.getHouseType());
ApartmentInfo apartmentInfo = new ApartmentInfo(
    advertData.getLodgingType(),
    advertData.getRoomQuantity(),
    advertData.getSquareAll(),
    advertData.getSquareLive(),
    advertData.getSquareKitchen(),
    advertData.getFloorNum(),
    advertData.getFloorAll(),
    advertData.getRepair(),
    advert,
    houseType);
apartmentInfoRepo.save(apartmentInfo);

return "redirect:/";
}

@GetMapping("/more-advert/{id}")
public String MoreAdvert(@PathVariable(name = "id") Long id,
    Model model){
    model.addAttribute("userRepo", userRepo);
    model.addAttribute("nearMetroRepo", nearMetroRepo);
    model.addAttribute("apartmentInfoRepo", apartmentInfoRepo);
    model.addAttribute("advertConditionsRepo", advertConditionsRepo);

    Optional<Advert> advert = advertRepo.findById(id);
    if(advert.isEmpty()){
        return "error404";
    }

    model.addAttribute("advert", advert.orElseThrow());

    return "more-advert";
}
}

```

Для всех 3 типов связей, мы создаем 2 метода: GET и POST. Метод GET получает данные всех записей и передает их в модель, после чего возвращает страницу, метод POST создает новую запись, включающую особенности одного из трех типов связей, после чего перенаправляет на ту же страницу

Один ко многим и многие к одному:

Advert.java



```

@Entity
public class Advert {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, length = 100)
    private String address;

    @Column(nullable = false, length = 50)
    private String dealType;

    @Column(nullable = false, length = 50)
    private String rentType;

    @Column(nullable = false, length = 50)
    private String title;

    @Column(nullable = false, length = 500)
    private String description;

    @ManyToOne(optional = false, cascade = CascadeType.ALL)
    @JoinColumn(name = "user_id")
    private User user;

    @ManyToOne(optional = false, cascade = CascadeType.ALL)
    @JoinColumn(name = "estateObjectType_id")
    private EstateObjectType estateObjectType;

    @OneToMany(mappedBy = "advert")
    private Collection<AdvertAttributs> advertAttributs;

    @OneToMany(mappedBy = "advert")
    private Collection<NearMetro> nearMetros;

    @OneToMany(mappedBy = "advert")
    private Collection<AdvertContacts> advertContacts;

    @OneToMany(mappedBy = "advert")
    private Collection<AdvertPhoto> advertPhotos;

    @OneToMany(mappedBy = "advert")
    private Collection<AdvertConditions> advertConditions;

    @OneToMany(mappedBy = "advert")
    private Collection<ApartmentInfo> apartmentInfos;
}

```

## User.java

```

@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, length = 50)
    private String login;

    @Column(nullable = false, length = 40)
    private String password;

    @Column(nullable = false)
    private Boolean active;

    @OneToOne(optional = false, cascade = CascadeType.ALL)
    @JoinColumn(name = "personalInfo_id")
    private PersonalInfo personalInfo;
}

```

```

@OneToOne(optional = false, cascade = CascadeType.ALL)
@JoinColumn(name = "contacts_id")
private Contacts contacts;

@OneToMany(mappedBy = "user", fetch = FetchType.EAGER)
private Collection<Advert> adverts;
}

```

В данной связи рассматривается пример между объявлениями и пользователями. Для модели, для которой мы можем связывать много записей пишем аннотацию `@OneToMany`, в обратном случае `@ManyToOne` в качестве параметров для `@OneToMany` передаем `mappedby` и `fetch`, после чего привязываем к коллекции объектов типа объявлений, для `@ManyToOne` передаем `cascade` и привязываем эту аннотацию к объекту типа пользователь.

- `FetchType.EAGER` — загружает коллекцию дочерних объектов сразу же, при загрузке родительских объектов;
- `FetchType.LAZY` — загружает коллекцию дочерних объектов, только непосредственно при обращении к объекту.

Многие ко многим.

Lesson.java

```

package com.example.secondProject.models;

import javax.persistence.*;
import java.util.List;

@Entity
public class Lesson {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String name;
    @ManyToOne
    @JoinTable (name = "student_lesson",
        joinColumns = @JoinColumn (name = "student_id"),
        inverseJoinColumns = @JoinColumn(name = "lesson_id"))
    private List<Student> students;

    public Lesson() {
    }

    public Lesson(String name, List<Student> students) {
        this.name = name;
        this.students = students;
    }

    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 List<Student> getStudents() {
        return students;
    }

    public void setStudents(List<Student> students) {
        this.students = students;
    }
}

```

## Student.java

```

package com.example.secondProject.models;

import javax.persistence.*;
import java.util.List;

@Entity
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String name;
    @ManyToMany
    @JoinTable (name = "student_lesson",
                joinColumns = @JoinColumn (name = "student_id"),
                inverseJoinColumns = @JoinColumn (name = "lesson_id"))
    private List<Lesson> lessons;

    public Student() {
    }

    public Student(String name, List<Lesson> lessons) {
        this.name = name;
        this.lessons = lessons;
    }

    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 List<Lesson> getLessons() {
        return lessons;
    }

    public void setLessons(List<Lesson> lessons) {
        this.lessons = lessons;
    }
}

```

В данной связи рассматривается пример между дисциплинами и студентами. В обеих моделях к каждому списку, состоящему из объектов данной модели, привязываем аннотацию `@ManyToMany`, которая указывает на тип связи, после чего указываем аннотацию `@JoinTable`, которая задает параметры смежной таблицы.

## Результат работы программы:

Поиск 🔍

Оценки студентов

| Студент                     | Дисциплина                               | Оценка |  |  |
|-----------------------------|--|--------|--|--|
| Суслин Александр Михайлович | Математическое моделирование             | 3      |  |  |
| Максимов Андрей             | Основы алгоритмизации и программирования | 4      |  |  |
| Суслин Александр Михайлович | Разработка программных модулей           | 5      |  |  |
| Суслин Александр Михайлович | Тестирование программных модулей         | 5      |  |  |

Добавить оценку

Суслин Александр Михайлович ▾

Математическое моделирование ▾

2 ▾

Добавить

Поиск 🔍

Отзывы книг

| ФИО                     | Книга                         | Комментарий       | Оценка |  |  |
|-------------------------|-------------------------------|-------------------|--------|--|--|
| Пупкин Алексей Петрович | Маленький принц               | Отличная книга!   | 5      |  |  |
| Дубов Андрей Иванович   | Гарри Поттер и тайная комната | Не реалистично... | 2      |  |  |

Добавить отзыв

ФИО

Книга ▾

Комментарий

Оценка ▾

Добавить

Поиск 🔍

Студенты

| Id | ФИО                         | Дата рождения | Группа   |  |  |
|----|-----------------------------|---------------|----------|--|--|
| 1  | Суслин Александр Михайлович | 2003-02-08    | П50-2-18 |  |  |
| 2  | Максимов Андрей             | 2000-04-02    | П50-1-18 |  |  |

Добавить студента

ФИО Студента

ДД . ММ . ГГГГ

Группа ▾

Добавить

Рисунок 17 – результаты работы программы.

**Вывод:** в ходе выполнения данной практической работы я изучил связи между таблицами в Spring и научился их применять, получил новые знания о разработке веб приложения на языке Java с помощью фреймворка Spring.

## Индивидуальный проект.

Цель: разработать веб приложение с использованием всех приобретенных навыков, полученных в ходе обучения на учебной практике.

Описание предметной области: Сервис поиска объявлений об аренде и продаже недвижимости.

Сервис является посредником между продавцами/арендаторами и покупателями/съемщиками различной недвижимости (пока только квартиры).

На сайте пользователь может найти нужное ему объявление по необходимым фильтрам и параметрам. А после регистрации и авторизации у пользователя будет возможность связаться с продавцом/арендатором или самому разместить объявление.

Логическая модель бд:

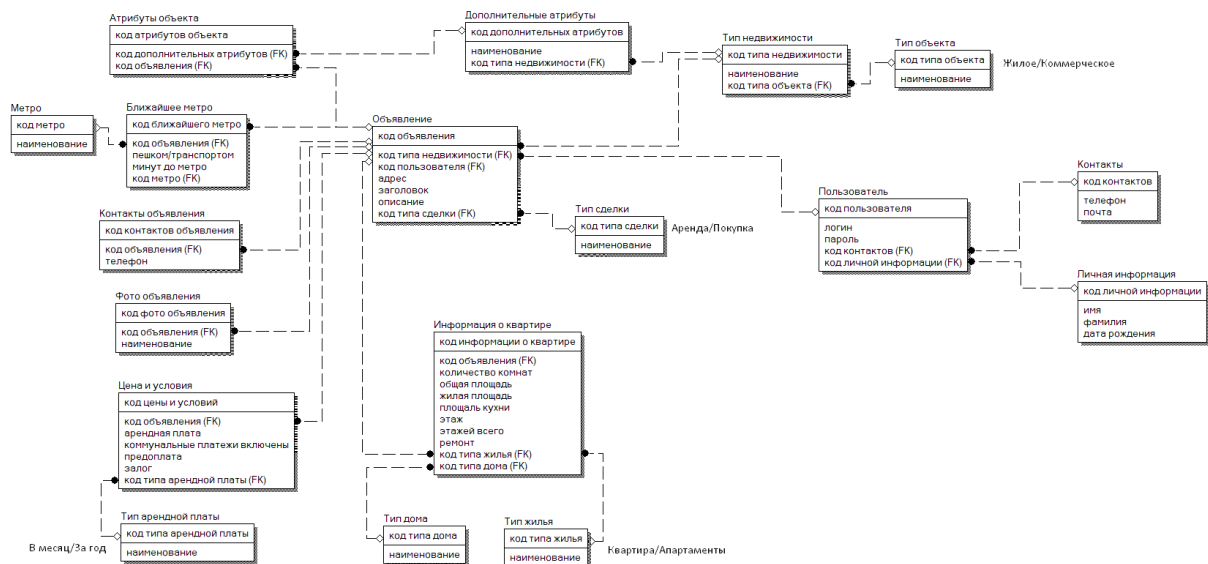


Рисунок 18 – даталогическая модель бд.

Физическая модель бд:

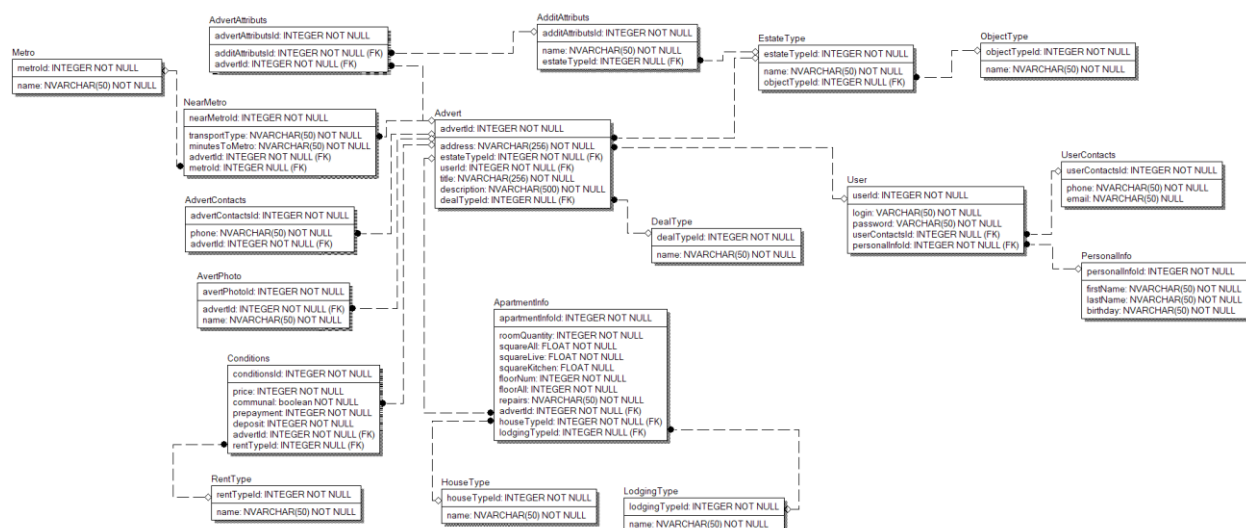


Рисунок 19 – физическая модель бд.

### Словарь данных:

Таблица 1 – словарь данных.

| Название таблицы  | Название столбца      | Тип данных   | Пустое значение | Примечание                  |
|-------------------|-----------------------|--------------|-----------------|-----------------------------|
| addit_attrIBUT    | id                    | bigint(20)   | Not null        | primary key, auto_increment |
| addit_attrIBUT    | name                  | varchar(50)  | Not null        |                             |
| addit_attrIBUT    | estate_object_type_id | bigint(20)   | Not null        | foreign key                 |
| advert            | id                    | bigint(20)   | Not null        | primary key, auto_increment |
| advert            | address               | varchar(100) | Not null        |                             |
| advert            | deal_type             | varchar(50)  | Not null        |                             |
| advert            | description           | varchar(500) | Not null        |                             |
| advert            | rent_type             | varchar(50)  | Not null        |                             |
| advert            | title                 | varchar(50)  | Not null        |                             |
| advert            | estate_object_type_id | bigint(20)   | Not null        | foreign key                 |
| advert            | user_id               | bigint(20)   | Not null        | foreign key                 |
| advert_attrIBUTS  | id                    | bigint(20)   | Not null        | primary key, auto_increment |
| advert_attrIBUTS  | addit_attrIBUT_id     | bigint(20)   | Not null        | foreign key                 |
| advert_attrIBUTS  | advert_id             | bigint(20)   | Not null        | foreign key                 |
| advert_conditionS | id                    | bigint(20)   | Not null        | primary key, auto_increment |

|                    |                |              |          |                             |
|--------------------|----------------|--------------|----------|-----------------------------|
| advert_conditions  | communal       | bit(1)       | Null     |                             |
| advert_conditions  | deposit        | bigint(20)   | Null     |                             |
| advert_conditions  | prepayment     | varchar(255) | Null     |                             |
| advert_conditions  | price          | bigint(20)   | Not null |                             |
| advert_conditions  | advert_id      | bigint(20)   | Not null | foreign key                 |
| advert_contacts    | id             | bigint(20)   | Not null | primary key, auto_increment |
| advert_contacts    | phone          | varchar(20)  | Not null |                             |
| advert_contacts    | advert_id      | bigint(20)   | Not null | foreign key                 |
| advert_photo       | id             | bigint(20)   | Not null | primary key, auto_increment |
| advert_photo       | name           | varchar(500) | Not null |                             |
| advert_photo       | advert_id      | bigint(20)   | Not null | foreign key                 |
| apartment_info     | id             | bigint(20)   | Not null | primary key, auto_increment |
| apartment_info     | floor_all      | int(11)      | Not null |                             |
| apartment_info     | floor_num      | int(11)      | Not null |                             |
| apartment_info     | lodging_type   | varchar(50)  | Not null |                             |
| apartment_info     | repair         | varchar(50)  | Not null |                             |
| apartment_info     | room_quantity  | varchar(50)  | Not null |                             |
| apartment_info     | square_all     | double       | Not null |                             |
| apartment_info     | square_kitchen | double       | Null     |                             |
| apartment_info     | square_live    | double       | Null     |                             |
| apartment_info     | advert_id      | bigint(20)   | Not null | foreign key                 |
| apartment_info     | house_type_id  | bigint(20)   | Not null | foreign key                 |
| contacts           | id             | bigint(20)   | Not null | primary key, auto_increment |
| contacts           | email          | varchar(50)  | Null     |                             |
| contacts           | phone          | varchar(20)  | Not null |                             |
| estate_object_type | id             | bigint(20)   | Not null | primary key, auto_increment |
| estate_object_type | estate_type    | varchar(50)  | Not null |                             |

|                    |                  |              |          |                             |
|--------------------|------------------|--------------|----------|-----------------------------|
| estate_object_type | name             | varchar(50)  | Not null |                             |
| house_type         | id               | bigint(20)   | Not null | primary key, auto_increment |
| house_type         | name             | varchar(50)  | Not null |                             |
| near_metro         | id               | bigint(20)   | Not null | primary key, auto_increment |
| near_metro         | minutes          | int(11)      | Not null |                             |
| near_metro         | name             | varchar(50)  | Not null |                             |
| near_metro         | transport_type   | varchar(50)  | Not null |                             |
| near_metro         | advert_id        | bigint(20)   | Not null | foreign key                 |
| personal_info      | id               | bigint(20)   | Not null | primary key, auto_increment |
| personal_info      | birthday         | varchar(10)  | Not null |                             |
| personal_info      | first_name       | varchar(40)  | Not null |                             |
| personal_info      | last_name        | varchar(40)  | Not null |                             |
| user               | id               | bigint(20)   | Not null | primary key, auto_increment |
| user               | active           | bit(1)       | Not null |                             |
| user               | login            | varchar(50)  | Not null |                             |
| user               | password         | varchar(256) | Not null |                             |
| user               | contacts_id      | bigint(20)   | Not null | unique                      |
| user               | personal_info_id | bigint(20)   | Not null | unique                      |

### Скрипт базы данных:

```

create table user
(
    id                bigint auto_increment
        primary key,
    active            bit                not null,
    login             varchar(50) charset utf8 not null,
    password          varchar(256) charset utf8 not null,
    contacts_id       bigint            not null,
    personal_info_id  bigint            not null,
    constraint UK_5lrsc4dgiqkqldf4lnhem7hbd
        unique (contacts_id),
    constraint UK_cw02ihailseic8ua74h49e1q2
        unique (personal_info_id),
    constraint FK1vnmpl9hxvlj8ure47h97lhco
        foreign key (personal_info_id) references personal_info (id),
    constraint FKjohv4n8sx93o8glmhm7jfnt10
        foreign key (contacts_id) references contacts (id)
);
create table near_metro
(

```



```

        id          bigint auto_increment
        primary key,
minutes          int          not null,
name             varchar(50)  charset utf8 not null,
transport_type   varchar(50)  charset utf8 not null,
advert_id        bigint       not null,
constraint FKa6ps8w38n4x4b4goyynfn3e44
        foreign key (advert_id) references advert (id)
);
create table personal_info
(
    id          bigint auto_increment
    primary key,
    birthday    varchar(10)  charset utf8 not null,
    first_name  varchar(40)  charset utf8 not null,
    last_name   varchar(40)  charset utf8 not null
);
create table house_type
(
    id          bigint auto_increment
    primary key,
    name         varchar(50)  charset utf8 not null
);
create table estate_object_type
(
    id          bigint auto_increment
    primary key,
    estate_type  varchar(50)  charset utf8 not null,
    name         varchar(50)  charset utf8 not null
);
create table advert_attributs
(
    id          bigint auto_increment
    primary key,
    addit_attribut_id bigint not null,
    advert_id    bigint not null,
    constraint FKm0l9l16qo6l3pbgvnw0fryalp
        foreign key (addit_attribut_id) references addit_attribut (id),
    constraint FKnljwo5ihgi4cl0h2a75fsmdyr
        foreign key (advert_id) references advert (id)
);
create table advert_contacts
(
    id          bigint auto_increment
    primary key,
    phone        varchar(20)  charset utf8 not null,
    advert_id    bigint       not null,
    constraint FK7ka7j30afv6b0s4k7wsngpjwh
        foreign key (advert_id) references advert (id)
);
create table apartment_info
(
    id          bigint auto_increment
    primary key,
    floor_all    int          not null,
    floor_num    int          not null,
    lodging_type varchar(50)  charset utf8 not null,
    repair        varchar(50)  charset utf8 not null,
    room_quantity varchar(50)  charset utf8 not null,
    square_all    double       not null,
    square_kitchen double      null,
    square_live   double      null,
    advert_id     bigint       not null,
    house_type_id bigint       not null,

```

```

        constraint FK38tkhjuszwxq045cobhsn3v87n
            foreign key (advert_id) references advert (id),
        constraint FKewb3du90y0gpds2ldn83j4ogp
            foreign key (house_type_id) references house_type (id)
    );
create table advert_conditions
(
    id          bigint auto_increment
        primary key,
    communal    bit                                null,
    deposit     bigint                             null,
    prepayment  varchar(255) charset utf8         null,
    price       bigint                             not null,
    advert_id   bigint                             not null,
    constraint FKc7konmgbc0rmnek3ngpv70nrh
        foreign key (advert_id) references advert (id)
);
create table advert_photo
(
    id          bigint auto_increment
        primary key,
    name        varchar(500) charset utf8         not null,
    advert_id   bigint                             not null,
    constraint FKbyrjppq0x9bdxdi9agr35lmyqo
        foreign key (advert_id) references advert (id)
);

```

Код программы.

AdvertController.java

```

package com.example.dpre.controllers;

import com.example.dpre.data.AdvertData;
import com.example.dpre.models.*;
import com.example.dpre.repo.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import javax.validation.Valid;
import java.util.Optional;

@Controller
public class AdvertController {

    @Autowired
    private ObjectTypeRepo objectTypeRepo;
    @Autowired
    private HouseTypeRepo houseTypeRepo;
    @Autowired
    private UserRepo userRepo;
    @Autowired
    private AdvertRepo advertRepo;
    @Autowired
    private NearMetroRepo nearMetroRepo;
    @Autowired

```

```

private AdvertConditionsRepo advertConditionsRepo;
@Autowired
private ApartmentInfoRepo apartmentInfoRepo;

@GetMapping("/advertnew")
public String AdvertNew(AdvertData advertData, Model model) {
    model.addAttribute("objectTypes", objectTypeRepo.findAll());
    model.addAttribute("houseTypes", houseTypeRepo.findAll());
    model.addAttribute("userRepo", userRepo);

    return "advertnew";
}

@PostMapping("/advertnew")
public String AdvertAdd(@Valid AdvertData advertData, BindingResult
bindingResult) {
    if(bindingResult.hasErrors()) {
        return "advertnew";
    }

    EstateObjectType estateObjectType =
objectTypeRepo.findByName(advertData.getEstateObjectType());
    Advert advert = new Advert(
        advertData.getAddress(),
        advertData.getDealType(),
        advertData.getRentType(),
        advertData.getTitle(),
        advertData.getDescription(),

userRepo.findById(Long.parseLong(advertData.getUserId())).orElseThrow(),
        estateObjectType);
    advertRepo.save(advert);

    NearMetro nearMetro = new NearMetro(
        advertData.getMetroName(),
        advertData.getTransportType(),
        advertData.getMetroMinutes(),
        advert);
    nearMetroRepo.save(nearMetro);

    AdvertConditions advertConditions = new AdvertConditions(
        advertData.getPrice(),
        advertData.getCommunal().equals("Включена"),
        advertData.getPrepayment(),
        advertData.getDeposit(),
        advert);
    advertConditionsRepo.save(advertConditions);

    HouseType houseType =
houseTypeRepo.findByName(advertData.getHouseType());
    ApartmentInfo apartmentInfo = new ApartmentInfo(
        advertData.getLodgingType(),
        advertData.getRoomQuantity(),
        advertData.getSquareAll(),
        advertData.getSquareLive(),
        advertData.getSquareKitchen(),
        advertData.getFloorNum(),
        advertData.getFloorAll(),
        advertData.getRepair(),
        advert,
        houseType);
    apartmentInfoRepo.save(apartmentInfo);

    return "redirect:/";
}

```

```

    }

    @GetMapping("/more-advert/{id}")
    public String MoreAdvert(@PathVariable(name = "id") Long id,
                             Model model){
        model.addAttribute("userRepo", userRepo);
        model.addAttribute("nearMetroRepo", nearMetroRepo);
        model.addAttribute("apartmentInfoRepo", apartmentInfoRepo);
        model.addAttribute("advertConditionsRepo", advertConditionsRepo);

        Optional<Advert> advert = advertRepo.findById(id);
        if(advert.isEmpty()){
            return "error404";
        }

        model.addAttribute("advert", advert.orElseThrow());

        return "more-advert";
    }
}

```

## LoginController.java

```

package com.example.dpre.controllers;

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

@Controller
public class LoginController {

    @GetMapping("/login")
    public String Login(){
        return "login";
    }

}

```

## MainController.java

```

package com.example.dpre.controllers;

import com.example.dpre.models.HouseType;
import com.example.dpre.repo.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class MainController {

    @Autowired
    private PersonalInfoRepo personalInfoRepo;
    @Autowired
    private ContactsRepo contactsRepo;
    @Autowired
    private UserRepo userRepo;
    @Autowired
    private AdvertRepo advertRepo;
    @Autowired
    private AdvertPhotoRepo advertPhotoRepo;
    @Autowired
    private AdvertConditionsRepo advertConditionsRepo;
}

```

```

    @Autowired
    private HouseTypeRepo houseTypeRepo;
    @Autowired
    private ApartmentInfoRepo apartmentInfoRepo;
    @Autowired
    private NearMetroRepo nearMetroRepo;
    @Autowired
    private ObjectTypeRepo objectTypeRepo;

    @GetMapping("/")
    public String Index(Model model){
        model.addAttribute("adverts", advertRepo.findAll());
        model.addAttribute("advertPhotoRepo", advertPhotoRepo);
        model.addAttribute("advertConditionsRepo", advertConditionsRepo);
        model.addAttribute("houseTypeRepo", houseTypeRepo);
        model.addAttribute("apartmentInfoRepo", apartmentInfoRepo);
        model.addAttribute("nearMetroRepo", nearMetroRepo);
        model.addAttribute("userRepo", userRepo);
        model.addAttribute("estateObjectTypes", objectTypeRepo.findAll());

        return "index";
    }
}

```

## RegistryController.java

```

package com.example.dpre.controllers;

import com.example.dpre.data.UserData;
import com.example.dpre.models.Contacts;
import com.example.dpre.models.PersonalInfo;
import com.example.dpre.models.User;
import com.example.dpre.repo.ContactsRepo;
import com.example.dpre.repo.PersonalInfoRepo;
import com.example.dpre.repo.UserRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.validation.ObjectError;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;

import javax.validation.Valid;

@Controller
public class RegistryController {

    @Autowired
    private PersonalInfoRepo personalInfoRepo;
    @Autowired
    private ContactsRepo contactsRepo;
    @Autowired
    private UserRepo userRepo;

    @GetMapping("/registry")
    public String Registry(UserData userData){
        return "registry";
    }

    @PostMapping("/registry")
    public String RegistryUser(@Valid UserData userData, BindingResult
bindingResult){

```

```

        if(bindingResult.hasErrors()){
            return "registry";
        }

        if(contactsRepo.findByPhone(userData.getPhone()) != null){
            ObjectError objectError = new ObjectError("phone", "Такой телефон
уже зарегистрирован.");
            bindingResult.addError(objectError);
            return "registry";
        }

        if(userRepo.findByLogin(userData.getLogin()) != null){
            ObjectError objectError = new ObjectError("login", "Пользователь
с таким логином уже существует.");
            bindingResult.addError(objectError);
            return "registry";
        }

        PersonalInfo personalInfo = new PersonalInfo(userData.getFirstName(),
userData.getLastName(), userData.getBirthDay());
        personalInfoRepo.save(personalInfo);
        Contacts contacts = new Contacts(userData.getPhone(),
userData.getEmail());
        contactsRepo.save(contacts);
        User user = new User(userData.getLogin(), userData.getPassword(),
true, personalInfo, contacts);
        user.setPassword(new
BCryptPasswordEncoder(8).encode(user.getPassword()));
        userRepo.save(user);

        return "redirect:/login";
    }
}

```

## SearchAdvertController.java

```

package com.example.dpre.controllers;

import com.example.dpre.data.AdvertData;
import com.example.dpre.models.Advert;
import com.example.dpre.models.AdvertConditions;
import com.example.dpre.models.ApartmentInfo;
import com.example.dpre.repo.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.ArrayList;
import java.util.List;

@Controller
public class SearchAdvertController {

    @Autowired
    private AdvertRepo advertRepo;
    @Autowired
    private UserRepo userRepo;
    @Autowired
    private ApartmentInfoRepo apartmentInfoRepo;
    @Autowired
    private PersonalInfoRepo personalInfoRepo;
    @Autowired

```

```

private ContactsRepo contactsRepo;
@Autowired
private AdvertPhotoRepo advertPhotoRepo;
@Autowired
private AdvertConditionsRepo advertConditionsRepo;
@Autowired
private HouseTypeRepo houseTypeRepo;
@Autowired
private NearMetroRepo nearMetroRepo;

@GetMapping("/search-advert")
public String SearchAdvert(@RequestParam(required = false) String
dealType,
                                @RequestParam(required = false) String
estateObjectType,
                                @RequestParam(required = false) String
roomQuantity,
                                @RequestParam(required = false) Long
priceFrom,
                                @RequestParam(required = false) Long priceTo,
                                Model model){
    model.addAttribute("personalInfoRepo", personalInfoRepo);
    model.addAttribute("contactsRepo", contactsRepo);
    model.addAttribute("advertPhotoRepo", advertPhotoRepo);
    model.addAttribute("advertConditionsRepo", advertConditionsRepo);
    model.addAttribute("houseTypeRepo", houseTypeRepo);
    model.addAttribute("userRepo", userRepo);
    model.addAttribute("nearMetroRepo", nearMetroRepo);
    model.addAttribute("apartmentInfoRepo", apartmentInfoRepo);

    /*
    List<Advert> adverts =
    advertRepo.findByDealTypeContainingAndEstateObjectType_NameContaining(
        dealType == null ? "" : dealType,
        estateObjectType == null ? "" : estateObjectType);
    model.addAttribute("adverts", adverts);*/

    List<Advert> adverts = new ArrayList<Advert>();
    for(Advert i: advertRepo.findAll()){
        ApartmentInfo apartmentInfo =
        apartmentInfoRepo.findByAdvert_Id(i.getId());
        AdvertConditions advertConditions =
        advertConditionsRepo.findByAdvert_Id(i.getId());

        if ((dealType == null || i.getDealType().contains(dealType)) &&
            (estateObjectType == null ||
            i.getEstateObjectType().getName().contains(estateObjectType)) &&
            (roomQuantity == null ||
            apartmentInfo.getRoomQuantity().contains(roomQuantity)) &&
            (priceFrom == null || advertConditions.getPrice() >=
            priceFrom) &&
            (priceTo == null || advertConditions.getPrice() <=
            priceTo)){
            adverts.add(i);
        }
    }

    model.addAttribute("adverts", adverts);

    return "search-advert";
}
}

```

Для регистрации и авторизации я добавил нужные зависимости и добавил файл, конфигурирующий безопасность приложения.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>
```

Рисунок 20 – зависимости для spring security.

### WebSecurityConfig.java

```
package com.example.dpre.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.EnableWebSecurity;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import javax.sql.DataSource;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private DataSource dataSource;
    @Autowired
    private PasswordEncoder passwordEncoder;

    @Bean
    public PasswordEncoder getPasswordEncoder() {
        return new BCryptPasswordEncoder(8);
    }
}
```



```

@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
        .antMatchers("/", "/registry", "/search-advert").permitAll()
        .anyRequest().authenticated()
        .and()
        .formLogin()
        .loginPage("/login")
        .permitAll()
        .and()
        .logout()
        .permitAll();
}

@Override
protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
    auth.jdbcAuthentication()
        .dataSource(dataSource)
        .passwordEncoder(passwordEncoder)
        .usersByUsernameQuery("select login, password, active from
user where login=?")
        .authoritiesByUsernameQuery("select login, password from user
where login=?");
}
}

```

В данном файле мы указываем какие страницы доступны неавторизованным пользователям, указываем какой шаблон является страницей авторизации, а также прописываем sql запрос к базе данных для корректной авторизации пользователя на сайте.

Результат работы программы:

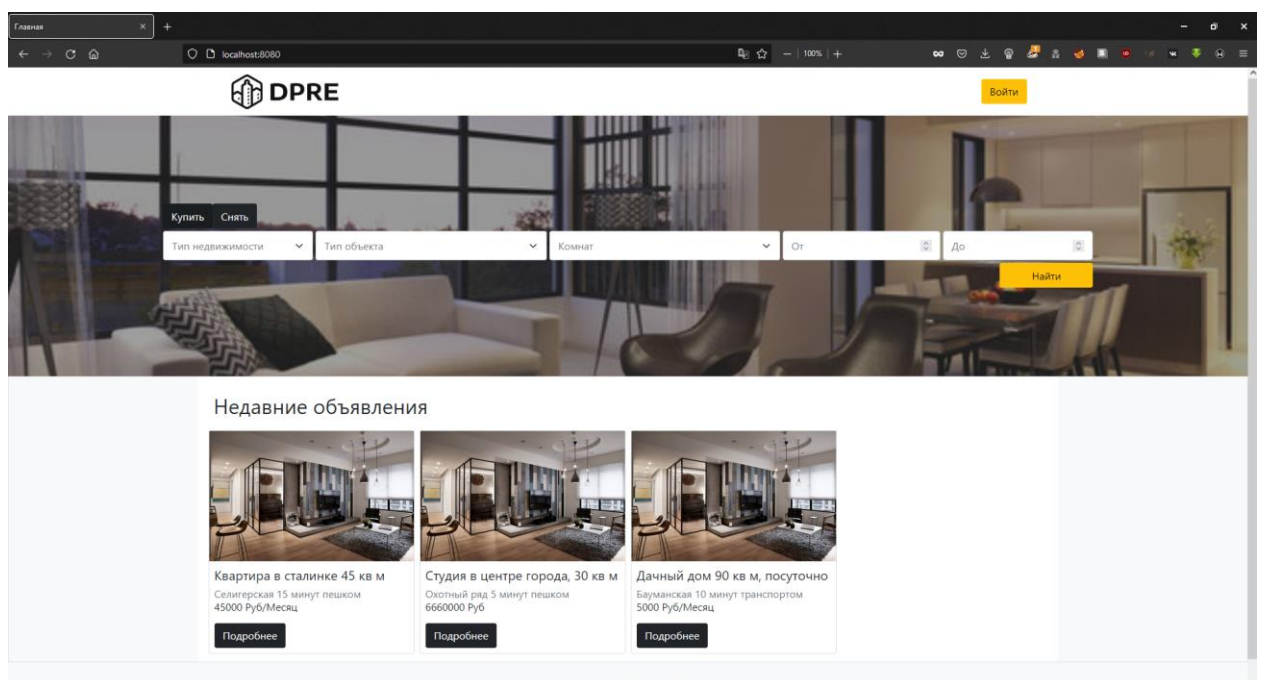


Рисунок 21 – результат работы программы (главная).

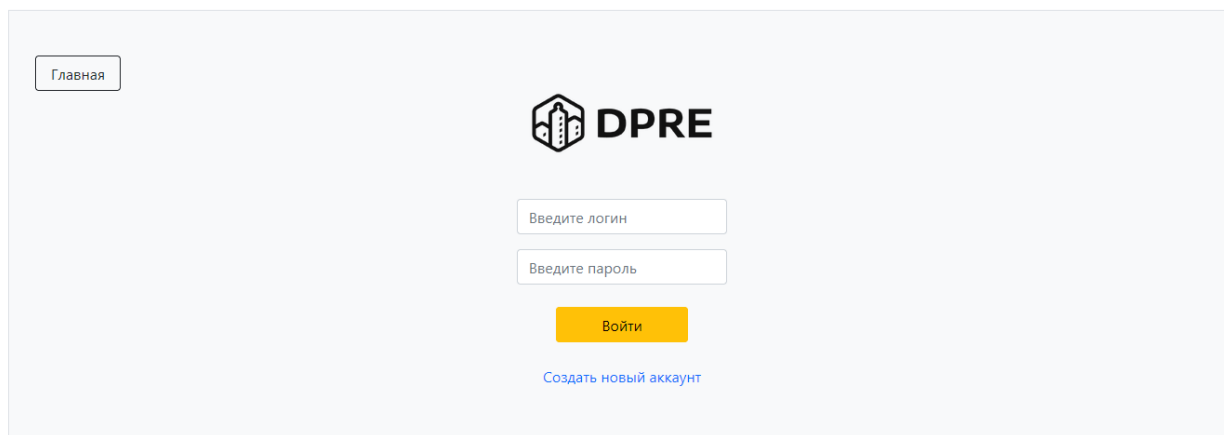


Рисунок 22 – результат работы программы (авторизация).

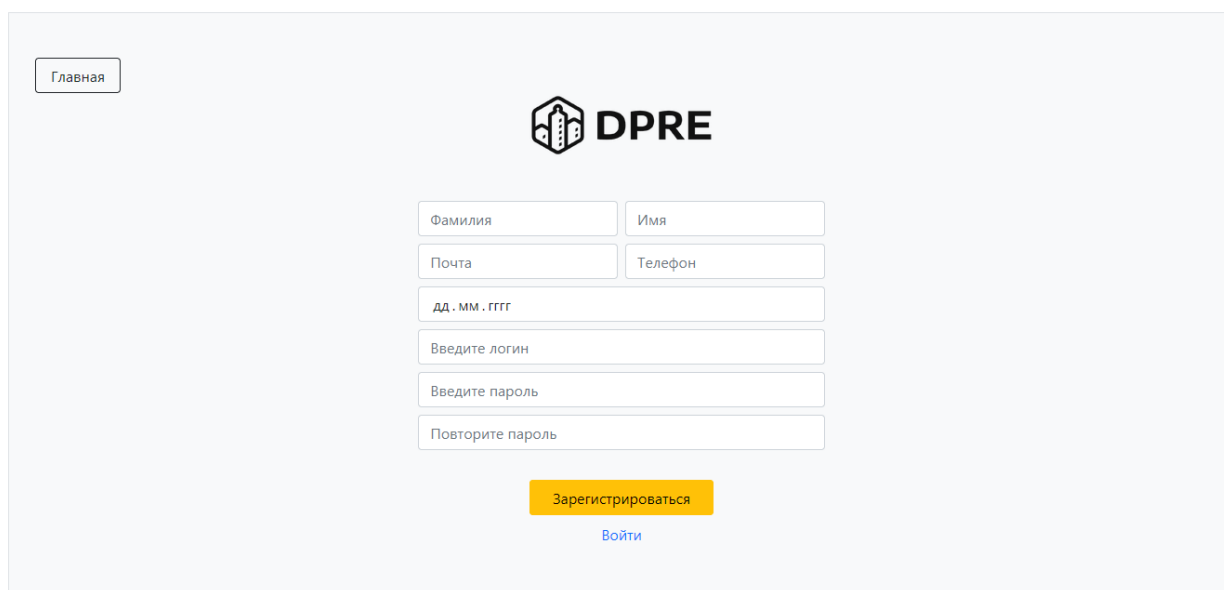


Рисунок 23 – результат работы программы (регистрация).

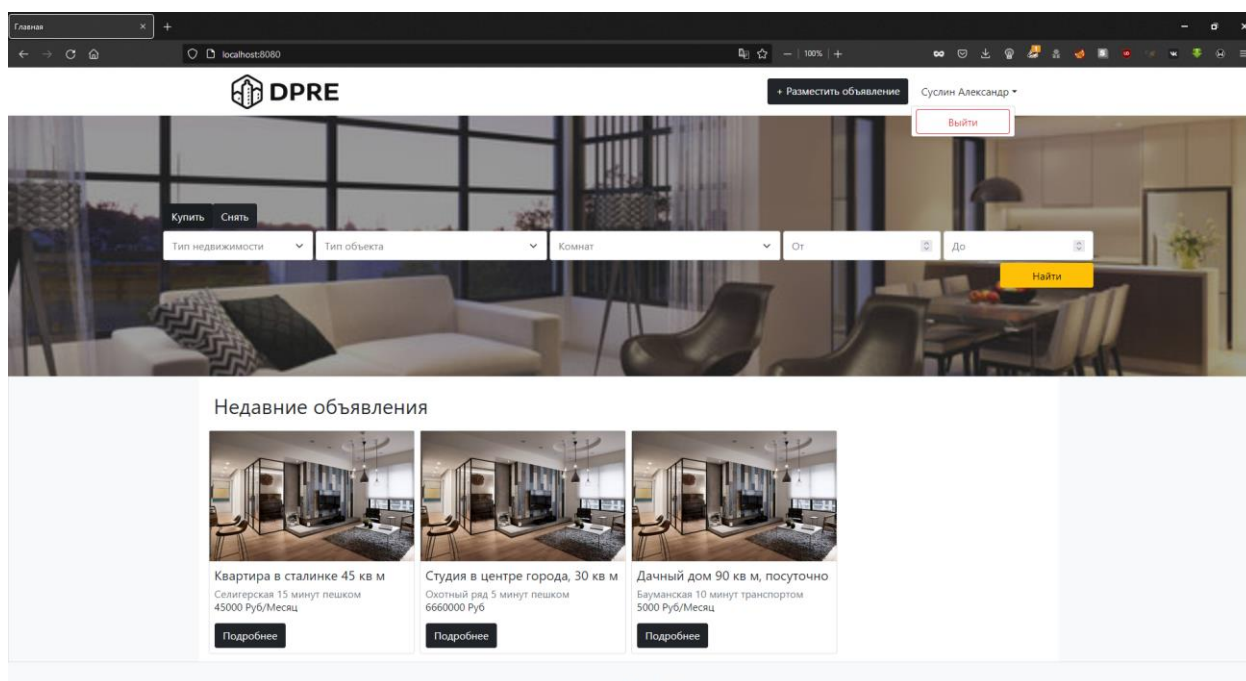


Рисунок 24 – результат работы программы (главная авторизован).

Результат работы программы (главная авторизован).

Рисунок 24 – результат работы программы (главная авторизован).

Рисунок 25 – результат работы программы (добавление объявления).

Рисунок 25 – результат работы программы (добавление объявления).

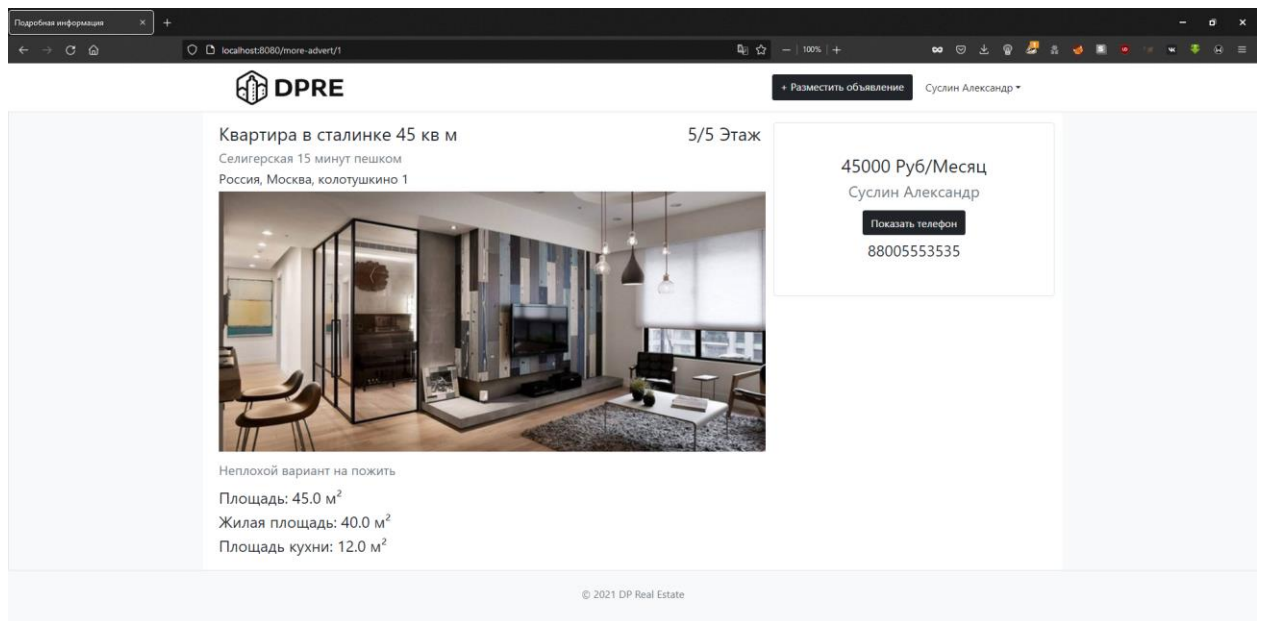


Рисунок 26 – результат работы программы (подробнее).

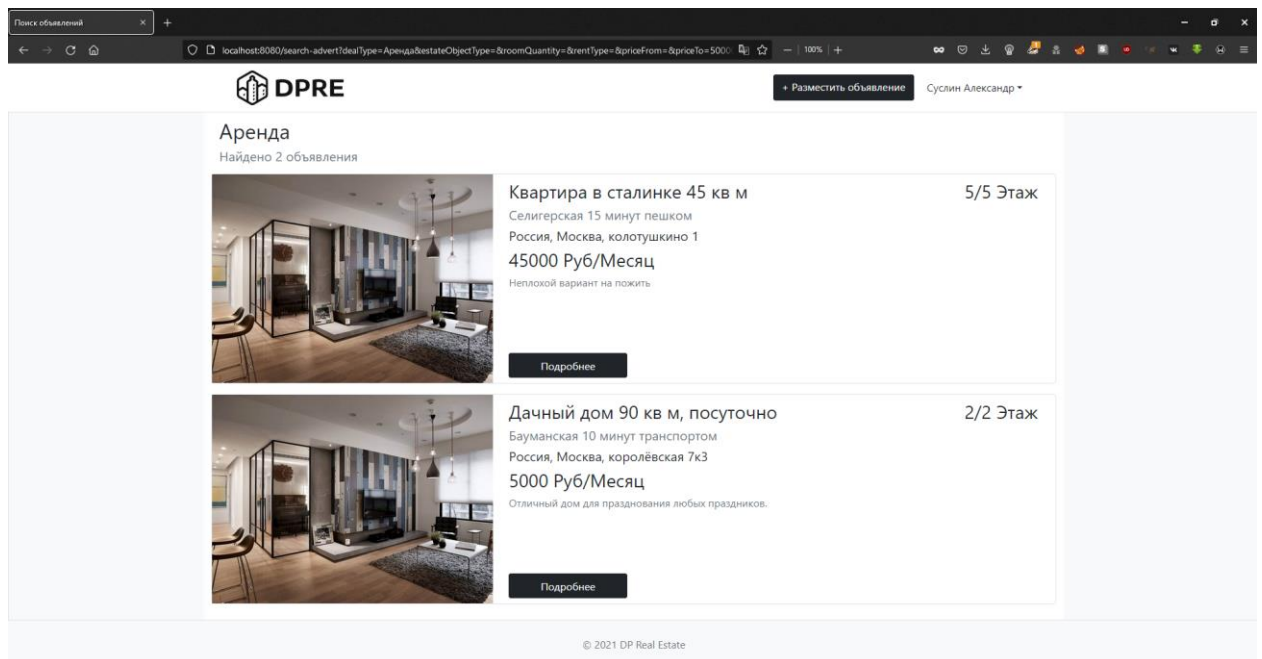


Рисунок 27 – результат работы программы (поиск объявлений).

Вывод: в ходе данной учебной практики я познакомился с фреймворком Java Spring, а также Spring Security, Spring MVC, Java Hibernate.