

**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
Практическая работа №2	9
Практическая работа №3	18
Практическая работа №4	27
Практическая работа №5	31
Практическая работа №6	36
Список иллюстраций	41

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 файла с html разметкой для страниц

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<h1>Main Page</h1>
<div>
  <a href="/Calc">Calc</a>
</div>
<div>
  <a href="/Money">MoneyCalc</a>
</div>
</body>
</html>
```

Рисунок 1 Главная страница

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Все люди</title>
</head>
<body>

<h1>Money calculation <b th:text="{Num}" /></h1>
<form th:method="POST" th:action="{@/Money/Calculate}">
  <div>
    <input name="Data" type="number" step="0.001" th:value="{Value}" * <b th:text="{Result}" />
  </div>
  <div>#<div>
    <select name="From">
      <option th:each="Moneyvalue: ${MoneyName}" th:value="{Moneyvalue.getValue()}" th:text="{Moneyvalue.getValue()}" th:selected="{Moneyvalue.getKey() == FirstSelect}">
    </select>#<div>
    <select name="to">
      <option th:each="Moneyvalue: ${MoneyName}" th:value="{Moneyvalue.getValue()}" th:text="{Moneyvalue.getValue()}" th:selected="{Moneyvalue.getKey() == SecondSelect}">
    </select>
    <input type="submit" value="Почислать">
  </form>
  <div>
    <a href="{@/Money/add}">add</a>
  </div>
  <div>
    <a href="{@/}">Back</a>
  </div>
</body>
</html>
```

Рисунок 2 Страница перевода валют

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Все люди</title>
6 </head>
7 <body>
8
9   <h1>Result calculation</h1>
10  <h3 th:text="{Operation}" />
11
12  <div>
13    <a href="{@/Calc/operation}">add</a>
14  </div>
15  <div>
16    <a href="{@/}">Back</a>
17  </div>
18 </div>
19 </body>
20 </html>
```

Рисунок 3 Страница с результатов вычислений

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <title>Title</title>
6 </head>
7 <body>
8 <form th:method="POST" th:action="@{/Calc/Calculate}">
9 <div>
10 <input name="Data1" type="number" step="0.001" th:value="${Param1}">
11 <select name="Act">
12 <option th:each="ActType: ${ActTypes}" th:value="${ActType.getValue()}" th:text="${ActType.getValue()}" th:selected="${ActType.getValue() == Act}">
13 </select>
14 <input name="Data2" type="number" step="0.001" th:value="${Param2}">
15 <div>
16 <input type="submit" value="Посчитать">
17 </div>
18 </div>
19 </form>
20 <div>
21 <a href="/calc">Back</a>
22 </div>
23 </body>
24 </html>

```

Рисунок 4 Страница указания параметров вычисления

2) Создание контроллеров управляющие переходами между страницами

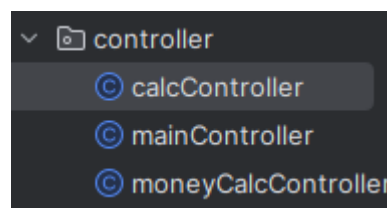


Рисунок 5 Контроллеры

```

35 4 Дмитрий Сорокин
36 @GetMapping("/{Calc}")
37 public String calc(Model model){
38     model.addAttribute( attributeName: "Param1", param1);
39     model.addAttribute( attributeName: "Param2", param2);
40     model.addAttribute( attributeName: "Act", act);
41     model.addAttribute( attributeName: "Operation", calc());
42     return "calc";
43 }
44 4 Дмитрий Сорокин
45 @GetMapping("/{Calc/operation}")
46 public String operation(Model model){
47     model.addAttribute( attributeName: "ActTypes", act_convert_char);
48     model.addAttribute( attributeName: "Param1", param1);
49     model.addAttribute( attributeName: "Param2", param2);
50     model.addAttribute( attributeName: "Act", act);
51     return "operation";
52 }
53 4 Дмитрий Сорокин
54 @PostMapping("/{Calc/Calculate}")
55 public String calculate(Model model, @RequestParam("Data1") double data1,
56                             @RequestParam("Act") String Act,
57                             @RequestParam("Data2") double data2){
58     param1 = data1;
59     param2 = data2;
60     act = char_convert_act.get(Act);
61     return "redirect:/Calc";
62 }

```

Рисунок 6 Контроллер калькулятора

```

Дмитрий Сорокин
@GetMapping("/{")
public String index(Model model){
    return "index";
}

```

Рисунок 7 Контроллер начального окна

```

1 usage Дмитрий Сорокин
public String calc(){
    double x = (value * money_convert_cost.get(first_selector))/money_convert_cost.get(second_selector);
    return String.format("%.10f", x);
}

Дмитрий Сорокин
@GetMapping("/{Money}")
public String MoneyCalc(Model model){
    model.addAttribute(attributeName: "Num", i);
    model.addAttribute(attributeName: "MoneyName", money_convert_name);
    model.addAttribute(attributeName: "MoneyCost", money_convert_name);
    model.addAttribute(attributeName: "FirstSelect", first_selector);
    model.addAttribute(attributeName: "SecondSelect", second_selector);
    model.addAttribute(attributeName: "Value", value);
    model.addAttribute(attributeName: "Result", calc());
    return "moneyCalc";
}

Дмитрий Сорокин
@PostMapping("/{Money/Calculate}")
public String calculate(Model model, @RequestParam("Data") double data,
    @RequestParam("From") String From,
    @RequestParam("To") String To){
    value = data;
    first_selector = name_convert_money.get(From);
    second_selector = name_convert_money.get(To);
    return "redirect:/Money";
}

Дмитрий Сорокин
@GetMapping("/{Money/Add}")
public String add(Model model){
    ++i;
    return "redirect:/Money";
}

```

Рисунок 8 Контроллер конвертации валют

3) Результат работы



Рисунок 9 Главная страница

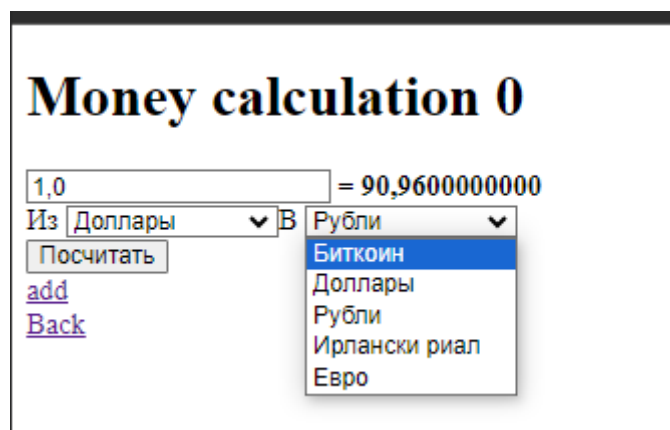


Рисунок 10 Перевод валют

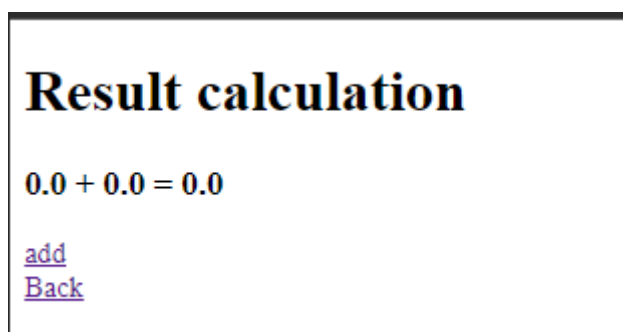
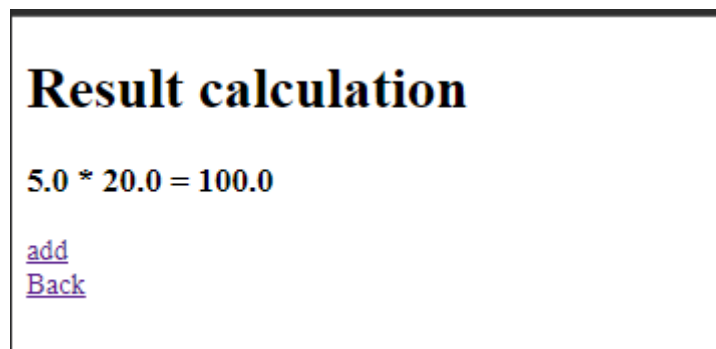


Рисунок 11 Страница вывод ответа калькулятора



A web form with a horizontal border. It contains two input fields: the first contains the number '5' and the second contains '20'. Between them is a dropdown menu showing a '*' symbol. Below the first input field is a button labeled 'Посчитать' (Calculate). Below the button is a blue underlined link labeled 'Back'.

Рисунок 12 Страница указания параметров вычисления



A page titled 'Result calculation' in a large, bold, black serif font. Below the title, the calculation '5.0 * 20.0 = 100.0' is displayed. At the bottom left, there are two blue underlined links: 'add' and 'Back'.

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



```
<> ClassUpdate.html  mainController.java  DisplayController.java
1 package com.example.project2.Models;
2
3 import java.util.HashMap;
4 import java.util.List;
5 import java.util.Map;
6
7 7 usages 5 inheritors new *
8 public abstract class AbsClass<T>{
9     2 usages
10     private int id = -1;
11     5 usages new *
12     public AbsClass(){
13
14     1 usage new *
15     public void SetID(int id){
16         this.id = id;
17     }
18
19     2 usages new *
20     public int GetID(){return id;}
21
22     1 usage 5 implementations new *
23     public abstract void UpdateData(T new_element);
24     1 usage 5 implementations new *
25     public abstract Map<String, String> GetKeyValue();
26 }
```

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

```

6 usages new *
9 public class Class1 extends AbsClass<Class1> {
10     3 usages
11     public String line1;
12
13     no usages new *
14     public Class1(String line1) { this.line1 = line1; }
15
16     1 usage new *
17     @Override
18     public void UpdateData(Class1 new_element) { this.line1 = new_element.GetLine1(); }
19
20     2 usages new *
21     public String GetLine1() { return line1; }
22
23     1 usage new *
24     @Override
25     public Map<String, String> GetKeyValue(){
26         return Map.of(
27             k1: "line1", GetLine1()
28         );
29     }
30
31     no usages new *
32     public static List<String> GetKey(){
33         return List.of(
34             e1: "line1"
35         );
36     }
37 }
38

```

Рисунок 15 Пример реализации одного из наследников абстрактного класса

```

5 import java.util.ArrayList;
6 import java.util.List;
7
8 14 usages new *
9 public class DAO<T> extends AbsClass<T>> {
10     4 usages
11     private List<T> list = new ArrayList<>();
12     1 usage
13     private int unique_id = 0;
14
15     2 usages new *
16     public T GetByID(int id) { return list.stream().filter(element -> element.GetID() == id).findAny().orElse( other: null); }
17
18     5 usages new *
19     public void AddElement(T element) {
20         element.SetID(++unique_id);
21         list.add(element);
22     }
23
24     5 usages new *
25     public void UpdateElement(int id, T new_element){...}
26
27     1 usage new *
28     public void DeleteElement(int id) { list.removeIf(p-> p.GetID() == id); }
29
30     1 usage new *
31     public List<T> GetInstance() { return list; }
32 }
33
34
35

```

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

2) Создание контроллеров

```
16 @Controller
17 @RequestMapping("/Display")
18 public class DisplayController {
19     1 usage
20     private static DAO<Class1> dao_1 = new DAO<>();
21     1 usage
22     private static DAO<Class2> dao_2 = new DAO<>();
23     1 usage
24     private static DAO<Class3> dao_3 = new DAO<>();
25     1 usage
26     private static DAO<Class4> dao_4 = new DAO<>();
27     1 usage
28     private static DAO<Class5> dao_5 = new DAO<>();
29
30     4 usages
31     private static String display_name = "";
32
33     2 usages
34     private static Map<String, DAO> using_dao = Map.of(
35         k1: "Class1", dao_1,
36         k2: "Class2", dao_2,
37         k3: "Class3", dao_3,
38         k4: "Class4", dao_4,
39         k5: "Class5", dao_5
40     );
41 }
```

Рисунок 17 Создание переменных типа DAO для указания хранимых данных

```
35 1 usage
36 private static Map<String, Integer> using_integer = Map.of(
37     k1: "Class1", v1: 1,
38     k2: "Class2", v2: 2,
39     k3: "Class3", v3: 3,
40     k4: "Class4", v4: 4,
41     k5: "Class5", v5: 5
42 );
43
44 1 usage
45 private static Map<Integer, Class> using_class = Map.of(
46     k1: 1, Class1.class,
47     k2: 2, Class2.class,
48     k3: 3, Class3.class,
49     k4: 4, Class4.class,
50     k5: 5, Class5.class
51 );
52
53 3 usages new *
54 public static boolean isDaoExist() {
55     return using_dao.containsKey(display_name);
56 }
57
58 13 usages new *
59 > public static DAO GetDao() { return using_dao.get(display_name); }
60
61 3 usages new *
62 > public static int GetIndexClass() { return using_integer.get(display_name); }
63
64 1 usage new *
65 > public static Class GetClassByIndex(int index) { return using_class.get(index); }
66
67 2 usages new *
68 public static List<String> GetClassOption() throws InvocationTargetException, IllegalAccessException, NoSuchMethodException {
69     Class type = GetClassByIndex(GetIndexClass());
70     Method method = type.getMethod("getKey", ...parameterTypes: null);
71     return (List<String>) method.invoke(obj: null);
72 }
```

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

```

73 new *
74 @GetMapping()
75 public String Display(Model model) throws InvocationTargetException, IllegalAccessException, NoSuchMethodException {
76     if(!IsDaoExist())
77         return "redirect:/";
78
79     model.addAttribute("ClassIndex", GetIndexClass());
80
81     model.addAttribute("Data", GetDao().GetInstance());
82
83     model.addAttribute("Keys", GetClassOption());
84     return "ClassShow";
85 }
86
87 new *
88 @PostMapping()
89 public String DisplayClass(Model model, @RequestParam("display_name") String name){
90     display_name = name;
91     return "redirect:/Display";
92 }
93
94 new *
95 @PostMapping("/Post1")
96 public String PostClass1(Model model, @ModelAttribute("model") Class1 class_model){
97     GetDao().AddElement(class_model);
98     return "redirect:/Display";
99 }
100
101 new *
102 @PostMapping("/Post2")
103 public String PostClass2(Model model, @ModelAttribute("model") Class2 class_model){
104     GetDao().AddElement(class_model);
105     return "redirect:/Display";
106 }

```

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

```

122 @PostMapping("/Delete/{id}")
123 public String DeleteClass(Model model, @PathVariable("id") int id){
124
125     if(!IsDaoExist())
126         return "redirect:/";
127     var dao = GetDao();
128     dao.DeleteElement(id);
129     return "redirect:/Display";
130 }

```

Рисунок 20 Создание запроса удаления записи из выбранного поля DAO

```

17 @Controller
18 @RequestMapping(@"/Display/Update")
19 public class DisplayUpdateController {
20
21     6 usages
22     int now_id = 0;
23
24     @PostMapping(@"/{id}")
25     public String Update(Model model, @PathVariable("id") int id) throws InvocationTargetException, IllegalAccessException, NoSuchMethodException {
26         Map<String, String> entity_data = new HashMap<>();
27
28         if(!DisplayController.IsDaoExist())
29             return "redirect:/";
30         now_id = id;
31         var object = DisplayController.GetDao().GetByID(id);
32
33         model.addAttribute("attributeName: "ClassIndex", DisplayController.GetIndexClass());
34         model.addAttribute("attributeName: "Data", object.GetKeyValue());
35
36         model.addAttribute("attributeName: "Keys", DisplayController.GetClassOption());
37         return "ClassUpdate";
38     }
39
40     @PostMapping(@"/Put1")
41     public String PutClass1(@ModelAttribute("model") Class1 class_model){
42         DisplayController.GetDao().UpdateElement(now_id, class_model);
43         return "redirect:/Display";
44     }
45
46     @PostMapping(@"/Put2")
47     public String PutClass2(@ModelAttribute("model") Class2 class_model){
48         DisplayController.GetDao().UpdateElement(now_id, class_model);
49         return "redirect:/Display";
50     }

```

Рисунок 21 Создание контроллера изменения данных полей DAO

3) Создание разметки

```

1
2 .header { grid-area: header; }
3 .action { grid-area: action; }
4 .body { grid-area: body; }
5
6 .grid-container {
7     display: grid;
8     grid-template-areas:
9         'header empty1 empty1 empty1 empty1 action'
10        'body body body body body empty2';
11     gap: 10px;
12     background: #eeeeee;
13     padding: 10px;
14     margin: 10px;
15 }
16
17 .grid-container > div {
18     background-color: rgba(255, 255, 255, 0.8);
19     text-align: left;
20     padding: 20px 0;
21     font-size: 20px;
22 }

```

Рисунок 22 Создание стиля для отображения данных в виде ячеек

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Title</title>
6  </head>
7  <body>
8      <h1>Main Page</h1>
9      <form th:method="POST" th:action="@{/Display}">
10         <div>
11             <button name="display_name" th:value="Class1">Display Class1</button>
12         </div>
13         <div>
14             <button name="display_name" th:value="Class2">Display Class2</button>
15         </div>
16         <div>
17             <button name="display_name" th:value="Class3">Display Class3</button>
18         </div>
19         <div>
20             <button name="display_name" th:value="Class4">Display Class4</button>
21         </div>
22         <div>
23             <button name="display_name" th:value="Class5">Display Class5</button>
24         </div>
25     </form>
26 </body>
27 </html>

```

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

```

8  <body>
9      <div>
10         <a href="/">
11             <button>
12                 Go Back
13             </button>
14         </a>
15     </div>
16     DISPLAY <a th:text="${ClassIndex}"></a>
17     <hr>
18     <form th:method="POST" th:action="@{/Display/Post' + ${ClassIndex}}">
19         <div th:each="key : ${Keys}">
20             <a th:text="${key}"></a>
21             <input th:name="${key}" type="text">
22         </div>
23         <div>
24             <input type="submit" value="Добавить">
25         </div>
26     </form>
27     <hr>
28
29     <div th:each="element: ${Data}" class="grid-container">
30         <div class="header">
31             id = <a th:text="${element.GetID()}"></a>
32         </div>
33         <div class="action">
34             <form th:method="DELETE" th:action="@{/Display/Delete/{id} (id = ${element.GetID()}}">
35                 <input type="submit" value="Удалить">
36             </form>
37             <form th:method="PUT" th:action="@{/Display/Update/{id} (id = ${element.GetID()}}">
38                 <input type="submit" value="Обновить">
39             </form>
40         </div>
41         <div class="body">
42             <div th:each="key : ${Keys}">
43                 <a th:text="${key}"></a> = "<a th:text="${element.GetKeyValue().get(key)}"/>"
44             </div>
45         </div>
46     </div>

```

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

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Title</title>
6  </head>
7  <body>
8      <div>
9          <a href="/Display">
10             <button>
11                 Go Back
12             </button>
13          </a>
14      </div>
15      Update <a th:text="${ClassIndex}"></a>
16      <hr>
17      <form th:method="POST" th:action="@{'/Display/Update/Put' + ${ClassIndex}}">
18          <div th:each="key : ${Keys}">
19              <a th:text="${key}"></a> = <input th:name="${key}" type="text" th:value="${Data.get(key)}">
20          </div>
21          <input type="submit" value="Обновить">
22      </form>
23  </body>
24  </html>

```

Рисунок 25 Разметка для вывода и редактирования данных одной записи

4) Результат

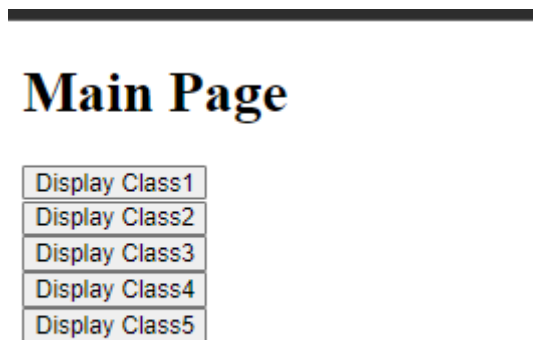


Рисунок 26 Начальная страница

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 from что позволяет использовать @ModelAttribute для генерации нужного класса.

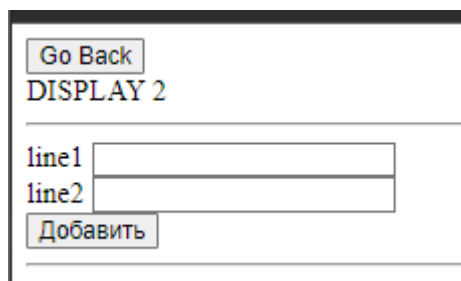


Рисунок 27 Страница отображения (пустая (Класс №2))



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

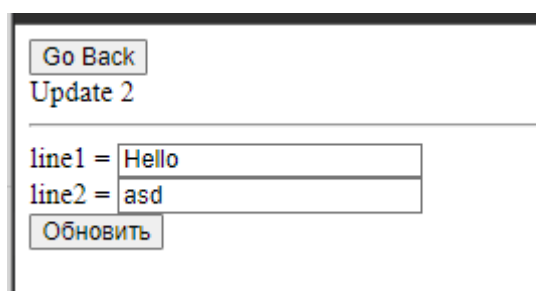


Рисунок 29 Переход на страницу изменения данных

Go Back

Update 2

line1 = Hello123

line2 = asd123

Обновить

Рисунок 30 Изменение данных

Go Back

DISPLAY 2

line1

line2

Добавить

id = 1	line1 = "Hello123" line2 = "asd123"	Удалить Обновить
--------	--	---------------------

Рисунок 31 Переход обратно на страницу отображения

Go Back

DISPLAY 2

line1

line2

Добавить

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

```
1 package com.example.project3.Models;
2
3 > import ...
4
13 13 usages
14 @Entity
15 @Table(name = "Book")
16 public class Book {
17     @Id
18     @GeneratedValue(strategy = GenerationType.AUTO)
19     private Long id;
20
21     2 usages
22     @Column(length = 100, unique = true)
23     @NotBlank(message = "Введите название книги")
24     @Length(min = 2)
25     private String name;
26
27     2 usages
28     @Column(columnDefinition="TEXT")
29     @NotBlank(message = "Введите описание книги")
30     private String description;
31
32     2 usages
33     @ManyToOne
34     @JoinColumn(name = "author_id")
35     @NotNull
36     private Author author;
37
38     4 usages
39     @ManyToMany(cascade = CascadeType.ALL)
40     @JoinTable (name="Book_Genre",
41         joinColumns=@JoinColumn (name="book_id", referencedColumnName="id"),
42         inverseJoinColumns=@JoinColumn(name="genre_id", referencedColumnName="id")
43     )
44     private Set<Genre> genres = new HashSet<>();
45
46     public Book() {
47     }
48
49     public Long getId() { return id; }
50
51     public String getName() { return name; }
```

Рисунок 33 Пример создаваемого класса с использованием аннотаций

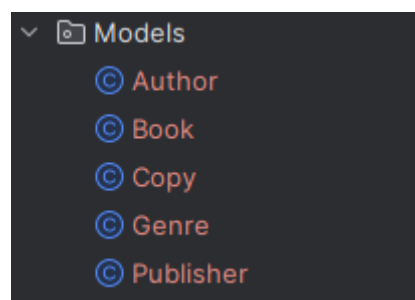


Рисунок 34 Созданные классы

2) Создание интерфейсы репозитория

На основе этих будут создавать запросы к базе данных.

```
4 usages
9 public interface BookRepository extends CrudRepository<Book, Long> {
10     1 usage
    Iterable<Book> findBooksByNameContains(String name);
11     5 usages
    Book findBookByIdEquals(Long id);
12 }
```

Рисунок 35 Пример создаваемого интерфейса репозитории

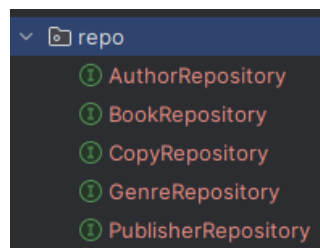


Рисунок 36 Созданные интерфейсы

3) Создание контроллеров

```
14
15 @Controller
16 @RequestMapping("/author")
17 public class AuthorController {
18     @Autowired
19     private AuthorRepository authorRepository;
20
21     3 usages
22     String search_line = "";
23
24     @GetMapping("/")
25     public String Author(Model model){
26         Iterable<Author> authors = authorRepository.findAuthorsByAliasContains(search_line);
27         model.addAttribute("authors", authors);
28         model.addAttribute("search_line", search_line);
29         return "Author";
30     }
31
32     @PostMapping("/")
33     public String AuthorSearch(@RequestParam("search") String search){
34         search_line = search;
35         return "redirect:/Author";
36     }
37
38     @GetMapping("/{id}")
39     public String AuthorGetAdd(@ModelAttribute("author") Author author, Model model){ return "AuthorAdd"; }
40
41     @PostMapping("/{id}")
42     public String AuthorAdd(@ModelAttribute("author") @Valid Author author,
43                             BindingResult bindingResult, Model model){ ... }
44
45     @GetMapping("/{id}/update")
46     public String AuthorGetUpdate(Model model, @PathVariable("id") Long id){ ... }
47
48     @PostMapping("/{id}/update")
49     public String AuthorUpdate(@ModelAttribute("author") @Valid Author author,
50                               BindingResult bindingResult, Model model, @PathVariable("id") Long id){ ... }
51
52     @PostMapping("/{id}/delete")
53     public String AuthorDelete(Model model, @PathVariable("id") Long id){ ... }
54 }
```

Рисунок 37 Пример созданных контроллеров



Рисунок 38 Созданные контроллеры

4) Создание базы данных

Для автоматического создания базы данных на основе написанных классов необходимо выбрать используемую бд. (Была выбрана 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.

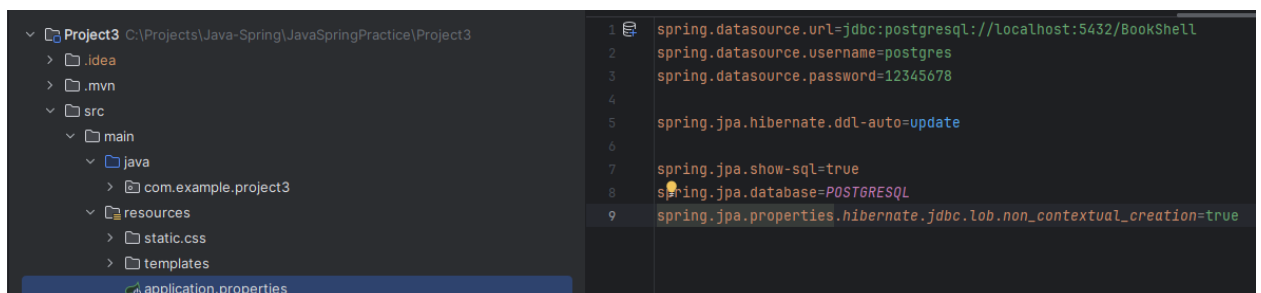


Рисунок 39 Файл с настройками для обращения к базе данных

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

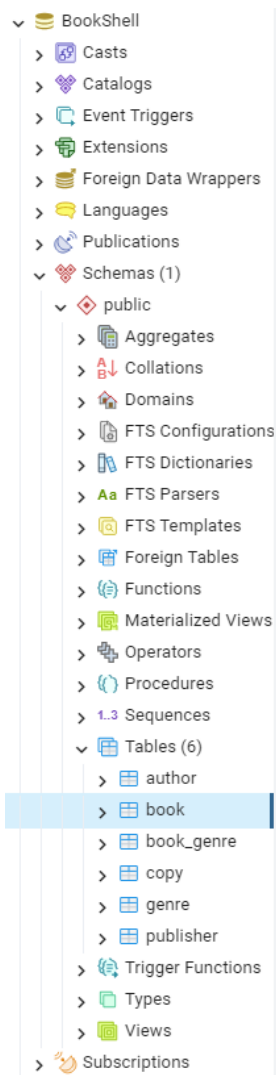


Рисунок 40 Созданная база данных

5) Результат работы программы

Главная страница

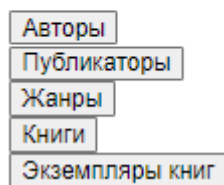


Рисунок 41 Главная страница

Вернуться

Автры

Добавить

Поиск

Код = 954

Удалить

Обновить

Псевдоним автора = "Author1"

Код = 1002

Удалить

Обновить

Псевдоним автора = "Author2"

Рисунок 42 Страница авторов

Вернуться

Автры

Добавить

1

Поиск

Код = 954

Удалить

Обновить

Псевдоним автора = "Author1"

Рисунок 43 Пример поиска (авторы)

Вернуться

Добавление автра

Псевдоним автора =

Новый автор

Добавить

Рисунок 44 Добавление автора

[Вернуться](#)

Автры

[Добавить](#)

[Поиск](#)

Код = 954

[Удалить](#)

[Обновить](#)

Псевдоним автора = "Author1"

Код = 1002

[Удалить](#)

[Обновить](#)

Псевдоним автора = "Author2"

Код = 1052

[Удалить](#)

[Обновить](#)

Псевдоним автора = "Новый автор"

Рисунок 45 Отображение нового автора

[Вернуться](#)

Изменить автора

Псевдоним автора =

[Добавить](#)

Рисунок 46 Изменение созданного автора

[Вернуться](#)

Автры

[Добавить](#)

[Поиск](#)

Код = 954

[Удалить](#)

[Обновить](#)

Псевдоним автора = "Author1"

Код = 1002

[Удалить](#)

[Обновить](#)

Псевдоним автора = "Author2"

Код = 1052

[Удалить](#)

[Обновить](#)

Псевдоним автора = "Новый автор (измененный)"

Рисунок 47 Отображение измененного автора

[Вернуться](#)

Автры

[Добавить](#)

[Поиск](#)

Код = 954

[Удалить](#)

[Обновить](#)

Псевдоним автора = "Author1"

Код = 1002

[Удалить](#)

[Обновить](#)

Псевдоним автора = "Author2"

Рисунок 48 Пример удаления (Автор)

[Вернуться](#)

Книги

[Добавить](#)

[Поиск](#)

Код = 52

[Удалить](#)

[Обновить](#)

Название книги = "BookName1123"

Автор = "Author2" (1002)

Жанры: Genre1 (102),

Описание = "Desc"

Код = 102

[Удалить](#)

[Обновить](#)

Название книги = "Пример книги"

Автор = "Author2" (1002)

Жанры: Genre2 (103),

Описание = "Простая кучка листиков для подтирания"

Рисунок 49 Пример отображения (книги)

[Вернуться](#)

Добавление книги

Название книги =

Описание книги =

Автор = Author1 ▼

[Добавить](#) Author1
Author2

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))

```
no usages
@OneToMany(mappedBy = "author")
private Set<Book> books;
```

Рисунок 51 Пример OneToMany

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

```
2 usages
@ManyToOne
@JoinColumn(name = "author_id")
@NotNull
private Author author;
```

Рисунок 52 Пример ManyToOne

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

```
no usages
@ManyToMany(mappedBy = "genres")
private Set<Book> books;
```

Рисунок 53 Пример ManyToMany

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

```
4 usages
@ManyToMany(cascade = CascadeType.ALL)
@JoinTable(name="Book_Genre",
    joinColumns=@JoinColumn(name="book_id", referencedColumnName="id"),
    inverseJoinColumns=@JoinColumn(name="genre_id", referencedColumnName="id"))
private Set<Genre> genres = new HashSet<>();
```

Рисунок 54 Пример JoinTable

2. 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)

```
▼ authors = {ArrayList@12267} size = 2
  ▼ 0 = {Author@12271}
    > id = {Long@12273} 954
    > alias = "Author1"
    > books = {PersistentSet@12275} size = 0
  ▼ 1 = {Author@12272}
    > id = {Long@12283} 1002
    > alias = "Author2"
    ▼ books = {PersistentSet@12285} size = 2
      > 0 = {Book@12288}
      > 1 = {Book@12289}
```

Рисунок 55 Возможности OneToMany

ManyToOne:

Связь обеспечивающее соединение нескольких записей к одной.

Книги

Код = 52

Название книги = "BookName1123"
Автор = "Author2" (1002)

Рисунок 56 Пример ManyToOne в виде авторов книг

ManyToMany:

Связь обеспечивающее соединение множества записей с множеством других записей.

Обновление книги

Название книги =

Описание =

Авторы = ▼

Жанры = ▼

Жанры книги

Genre1 (102)

Genre2 (103)

Рисунок 57 Пример ManyToMany в виде множества жанров множества книг

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

Практическая работа №5

Авторизация и Регистрация

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

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

Рисунок 58 Добавленная зависимость для работы с безопасностью и авторизацией

2. Файлы MvcConfig и WebSecurityConfig

```
Дмитрий Сорокин *
@Configuration
public class MvcConfig implements WebMvcConfigurer {

    no usages  Дмитрий Сорокин *
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController( urlPathOrPattern: "/login").
            setViewName("login");
    }
}
```

Рисунок 59 Файл MvcConfig

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

```

Дмитрий Сорокин
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private DataSource dataSource;

    @Autowired
    private PasswordEncoder passwordEncoder;

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

    Дмитрий Сорокин
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.jdbcAuthentication().dataSource(dataSource).passwordEncoder(getPasswordEncoder())
            .usersByUsernameQuery("select username, password, active from model_user where username=?")
            .authoritiesByUsernameQuery("select u.username, ur.roles from model_user u inner join user_role ur on u.id_user = ur.user_id where u.username=?");
    }

    Дмитрий Сорокин
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests() ExpressionInterceptUriRegistry
            .antMatchers("@*/login*", "@*/registration*").permitAll()
            .anyRequest() AuthorizedUri
            .authenticated() ExpressionInterceptUriRegistry
            .and() HttpSecurity
            .formLogin() FormLoginConfigurer<HttpSecurity>
            .loginPage("/login")
            .defaultSuccessUrl("/")
            .permitAll()
            .and() HttpSecurity
            .logout() LogoutConfigurer<HttpSecurity>
            .permitAll()
            .and().csrf().disable().cors().disable();
    }
}

```

Рисунок 60 Файл WebSecurityConfig

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

3. 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.

```
11 usages  Дмитрий Сорокин
@Entity
public class modelUser {
    Дмитрий Сорокин
    public modelUser(){
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID_User")
    private Long id;

    3 usages
    @Length(min = 2)
    private String username;
    3 usages
    private String password;
    3 usages
    private boolean active;

    3 usages
    @ElementCollection(targetClass = roleEnum.class, fetch = FetchType.EAGER)
    @CollectionTable(name = "user_role", joinColumns = @JoinColumn(name = "user_id"))
    @Enumerated(EnumType.STRING)
    private Set<roleEnum> roles;

    Дмитрий Сорокин
    public Long getId(){ return id; }
```

Рисунок 61 Класс пользователя

```
18 usages  Дмитрий Сорокин
public enum roleEnum implements GrantedAuthority {
    3 usages
    USER, ADMIN, LIB, CATALOG;
    Дмитрий Сорокин
    @Override
    public String getAuthority(){ return name(); }
}
```

Рисунок 62 Enum класс ролей

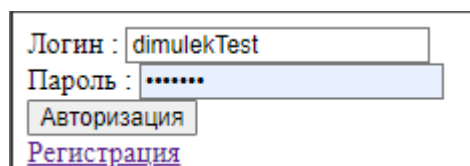
Реализация контроллера регистрации:

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

```
Дмитрий Сорокин
@Controller
public class registrationController {
    @Autowired
    private com.example.project3.repo.userRepository userRepository;
    @Autowired
    private PasswordEncoder passwordEncoder;
    Дмитрий Сорокин
    @GetMapping("/registration")
    private String RegView() { return "regis"; }
    Дмитрий Сорокин
    @PostMapping("/registration")
    private String Reg(modelUser user, Model model)
    {
        modelUser user_from_db = userRepository.findByUsername(user.getUsername());
        if (user_from_db != null)
        {
            model.addAttribute("message", "Пользователь с таким логином уже существует");
            return "regis";
        }
        user.setActive(true);
        user.setRoles(Collections.singleton(roleEnum.USER));
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        userRepository.save(user);
        return "redirect:/login";
    }
}
```

Рисунок 63 Контроллер регистрации

4. Результат работы



Логин : dimulekTest
Пароль : *****
Авторизация
Регистрация

Рисунок 64 Попытка авторизоваться под не существующим аккаунтом

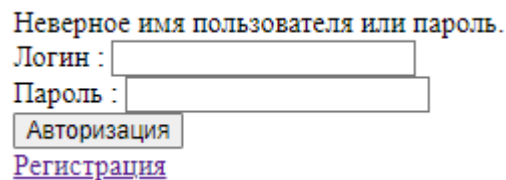


Рисунок 65 Вывод ошибки

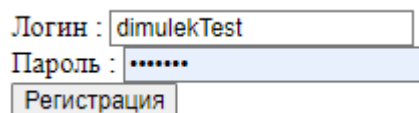


Рисунок 66 Регистрация

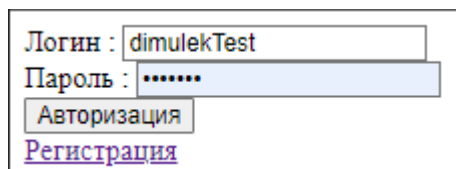


Рисунок 67 Успешная авторизация

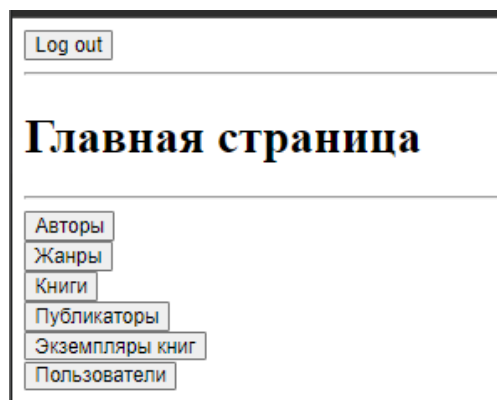


Рисунок 68 Открытие главной страницы

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

Практическая работа №6

Разделение прав доступа

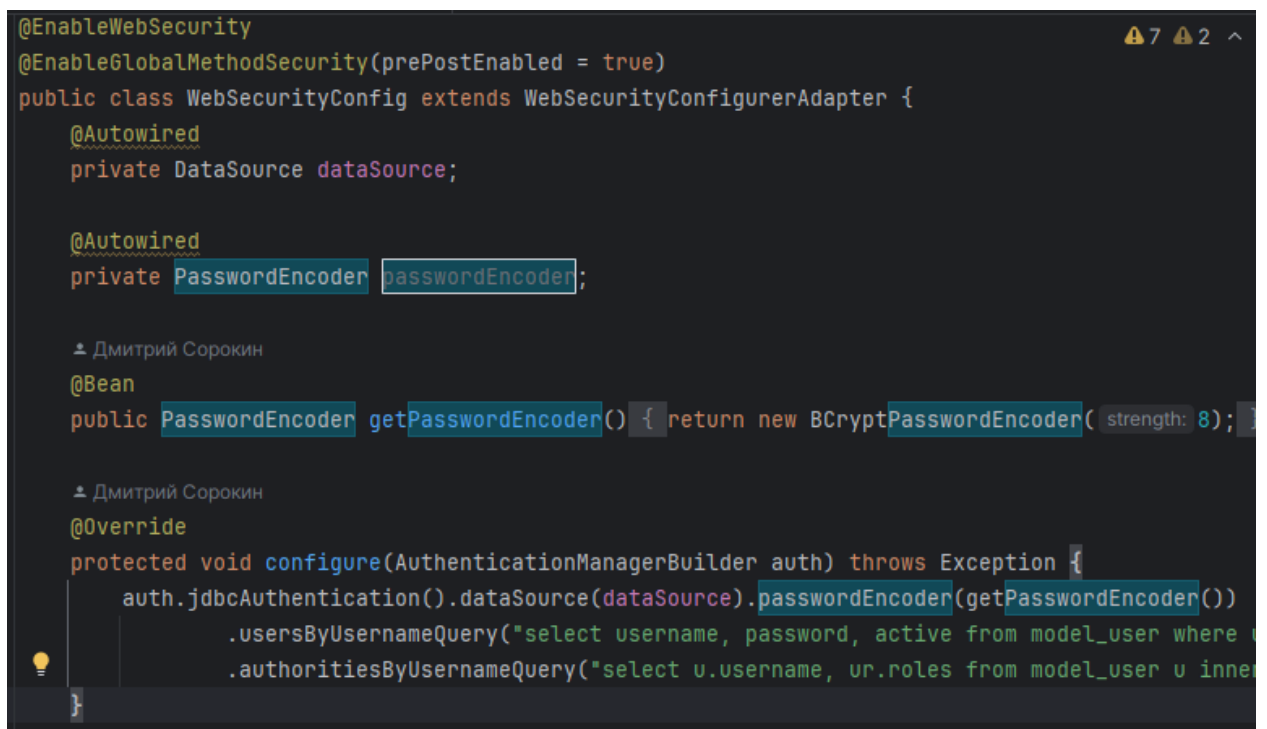
Цель: Реализовать механизм шифрование пароля пользователя.
Добавить разграничение прав доступа для пользователей.

- Реализация механизма шифрования;
- Добавить новую роль;
- Создать контроллер для редактирования прав доступа пользователей и задать доступ только новой роли;

Ход работы:

1. Шифрование

Для проверки введенного пользователем пароля на соответствие с зашифрованным паролем находящимся в базе данных необходимо использовать шифрование как при авторизации так и при регистрации.



```
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private DataSource dataSource;

    @Autowired
    private PasswordEncoder passwordEncoder;

    // Дмитрий Сорокин
    @Bean
    public PasswordEncoder getPasswordEncoder() { return new BCryptPasswordEncoder( strength: 8); }

    // Дмитрий Сорокин
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.jdbcAuthentication().dataSource(dataSource).passwordEncoder(getPasswordEncoder())
            .usersByUsernameQuery("select username, password, active from model_user where ")
            .authoritiesByUsernameQuery("select u.username, ur.roles from model_user u inner");
    }
}
```

Рисунок 69 Шифрование в авторизации

```

Дмитрий Сорокин
@Controller
public class registrationController {
    @Autowired
    private com.example.project3.repo.userRepository userRepository;
    @Autowired
    private PasswordEncoder passwordEncoder;
    Дмитрий Сорокин
    @GetMapping("/registration")
    private String RegView() { return "regis"; }
    Дмитрий Сорокин
    @PostMapping("/registration")
    private String Reg(modelUser user, Model model)
    {
        modelUser user_from_db = userRepository.findByUsername(user.ge
        if (user_from_db != null)
        {
            model.addAttribute(s: "message", o: "Пользователь с таким
            return "regis";
        }
        user.setActive(true);
        user.setRoles(Collections.singleton(roleEnum.USER));
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        userRepository.save(user);
        return "redirect:/login";
    }
}

```

Рисунок 70 Шифрование при регистрации

2. Добавление новых ролей

USER – новый пользователь системы (не имеет доступа ни к чему)

ADMIN – администратор системы (имеет доступ ко всему)

LIB – Библиотекарь (Имеет доступ к редактированию публикаторов и экземпляров книг (просмотр существующих книг))

CATALOG – Каталогизатор (Имеет доступ к редактированию авторов, жанров, книг и жанров книг).

```

18 usages  Дмитрий Сорокин
public enum roleEnum implements GrantedAuthority {
    3 usages
    USER, ADMIN, LIB, CATALOG;
    Дмитрий Сорокин
    @Override
    public String getAuthority() { return name(); }
}

```

Рисунок 71 Добавленные роли

3. Создание контроллера редактирующий права доступа

Для проверки необходимо использовать строковое представление ролей.

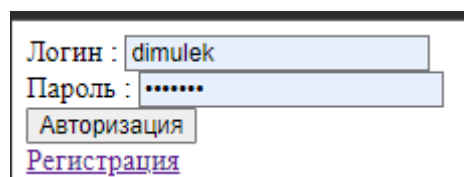
```

Дмитрий Сорокин
@Controller
@RequestMapping("/Author")
@PreAuthorize("hasAnyAuthority('ADMIN') or hasAnyAuthority('CATALOG')")
public class AuthorController {

```

Рисунок 72 Указание какие роли имеют доступ к данному контроллеру

4. Результат работы



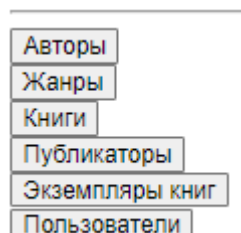
Логин : dimulek

Пароль :

Авторизация

[Регистрация](#)

Рисунок 73 Авторизация как администратор



Авторы

Жанры

Книги

Публикаторы

Экземпляры книг

Пользователи

Рисунок 74 Доступные вкладки для администратора

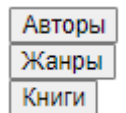


Рисунок 75 Доступные вкладки каталогизатора

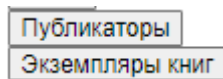


Рисунок 76 Доступные вкладки библиотекаря

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Wed Dec 27 22:33:17 MSK 2023

There was an unexpected error (type=Forbidden, status=403).

Forbidden

org.springframework.security.access.AccessDeniedException: Доступ запрещен

Рисунок 77 Результат попытки перейти во вкладку не доступную для авторизованного пользователя

[Вернуться](#)

Пользователи

Код = 1	Обновить
Логин = "dimulek"	
Код = 2	Обновить
Логин = "dimulek1"	
Код = 3	Обновить

Рисунок 78 Окно просмотра существующих пользователей

[Вернуться](#)

Изменение ролей пользователя

<div>Код = 2 Логин = dimulek1</div> <hr/> <div>Роли = Администратор ▼ Добавить роль</div>		<div>Роли пользователя</div> <div>Пользователь Удалить</div> <div>Каталогизатор Удалить</div>
--	--	---

Рисунок 79 Окно изменение ролей пользователя

[Вернуться](#)

Изменение ролей пользователя

<div>Код = 2 Логин = dimulek1</div> <hr/> <div>Роли = Администратор ▼ Добавить роль</div>		<div>Роли пользователя</div> <div>Каталогизатор Удалить</div>
--	--	---

Рисунок 80 Удаление роли пользователя

[Вернуться](#)

Изменение ролей пользователя

<div>Код = 2 Логин = dimulek1</div> <hr/> <div>Роли = Администратор ▼ Добавить роль</div>		<div>Роли пользователя</div> <div>Каталогизатор Удалить</div> <div>Библиотекарь Удалить</div>
--	--	---

Рисунок 81 Добавление роли библиотекаря

Вывод: В ходе практической работы №6 был реализовано разграничение по ролям и возможность изменения ролей пользователей.

Список иллюстраций

Рисунок 1 Главная страница	3
Рисунок 2 Страница перевода валют	4
Рисунок 3 Страница с результатов вычислений	4
Рисунок 4 Страница указания параметров вычисления	5
Рисунок 5 Контроллеры	5
Рисунок 6 Контроллер калькулятора	5
Рисунок 7 Контроллер начального окна	6
Рисунок 8 Контроллер конвертации валют	6
Рисунок 9 Главная страница	7
Рисунок 10 Перевод валют	7
Рисунок 11 Страница вывод ответа калькулятора.....	7
Рисунок 12 Страница указания параметров вычисления	8
Рисунок 13 Вывод ответа	8
Рисунок 14 Создание абстрактного класса для указания методов которые будут у наследников	9
Рисунок 15 Пример реализации одного из наследников абстрактного класса	10
Рисунок 16 Создание шаблонного класса DAO который будет хранить и взаимодействовать с данными.....	10
Рисунок 17 Создание переменных типа DAO для указания хранимых данных.....	11
Рисунок 18 Создание статических полей и методов для быстрого взаимодействия с данными.....	11
Рисунок 19 Создание запросов отображения данных и примеры добавления новых данных в поля DAO	12

Рисунок 20 Создание запроса удаления записи из выбранного поля DAO	12
Рисунок 21 Создание контроллера изменения данных полей DAO	13
Рисунок 22 Создание стиля для отображения данных в виде ячеек	13
Рисунок 23 Разметка начальной страницы (Перехода на страницы с отображением информации).....	14
Рисунок 24 Создание разметки для отображения данных выбранного DAO (С выводом названия полей данного класса)	14
Рисунок 25 Разметка для вывода и редактирования данных одной записи	15
Рисунок 26 Начальная страница	15
Рисунок 27 Страница отображения (пустая (Класс №2))	16
Рисунок 28 Страница отображения (не пустая (Класс №2)).....	16
Рисунок 29 Переход на страницу изменения данных	16
Рисунок 30 Изменение данных	17
Рисунок 31 Переход обратно на страницу отображения	17
Рисунок 32 Удалить	17
Рисунок 33 Пример создаваемого класса с использованием аннотаций	19
Рисунок 34 Созданные классы.....	19
Рисунок 35 Пример создаваемого интерфейса репозитории	20
Рисунок 36 Созданные интерфейсы	20
Рисунок 37 Пример созданных контроллеров	20
Рисунок 38 Созданные контроллеры	21
Рисунок 39 Файл с настройками для обращения к базе данных	21

Рисунок 40 Созданная база данных.....	22
Рисунок 41 Главная страница	22
Рисунок 42 Страница авторов.....	23
Рисунок 43 Пример поиска (авторы).....	23
Рисунок 44 Добавление автора	23
Рисунок 45 Отображение нового автора	24
Рисунок 46 Изменение созданного автора.....	24
Рисунок 47 Отображение измененного автора	25
Рисунок 48 Пример удаления (Автор)	25
Рисунок 49 Пример отображения (книги).....	26
Рисунок 50 Пример добавления книги с использованием поля выбора	26
Рисунок 51 Пример OneToMany.....	28
Рисунок 52 Пример ManyToOne.....	28
Рисунок 53 Пример ManyToMany	28
Рисунок 54 Пример JoinTable	29
Рисунок 55 Возможности OneToMany	29
Рисунок 56 Пример ManyToOne в виде авторов книг.....	30
Рисунок 57 Пример ManyToMany в виде множества жанров множества книг	30
Рисунок 58 Добавленная зависимость для работы с безопасностью и авторизацией	31
Рисунок 59 Файл MvcConfig.....	31
Рисунок 60 Файл WebSecurityConfig.....	32
Рисунок 61 Класс пользователя.....	33
Рисунок 62 Enum класс ролей.....	33

Рисунок 63 Контроллер регистрации	34
Рисунок 64 Попытка авторизоваться под не существующим аккаунтом.....	34
Рисунок 65 Вывод ошибки.....	35
Рисунок 66 Регистрация.....	35
Рисунок 67 Успешная авторизация	35
Рисунок 68 Открытие главной страницы.....	35
Рисунок 69 Шифрование в авторизации	36
Рисунок 70 Шифрование при регистрации.....	37
Рисунок 71 Добавленные роли	38
Рисунок 72 Указание какие роли имеют доступ к данному контроллеру	38
Рисунок 73 Авторизация как администратор	38
Рисунок 74 Доступные вкладки для администратора	38
Рисунок 75 Доступные вкладки каталогизатора	39
Рисунок 76 Доступные вкладки библиотекаря	39
Рисунок 77 Результат попытки перейти во вкладку не доступную для авторизованного пользователя	39
Рисунок 78 Окно просмотра существующий пользователей.....	39
Рисунок 79 Окно изменение ролей пользователя.....	40
Рисунок 80 Удаление роли пользователя.....	40
Рисунок 81 Добавление роли библиотекаря	40