|  |
| --- |
| Министерство науки и высшего образования Российской Федерации |
| федеральное государственное бюджетное образовательное учреждение высшего образования |

ОТЧЕТ

по учебной практике

|  |
| --- |
| УП.04.01  Внедрение и поддержка программного обеспечения |

|  |
| --- |
| Профессионального модуля ПМ.04  Сопровождение и обслуживание программного обеспечения компьютерных систем |

|  |
| --- |
| Специальность 09.02.07  Информационные системы и программирование |

Студент:       Кононыхин Евгений Евгеньевич

*фамилия, имя, отчество*

Группа: П50-7-22

Руководитель по практической подготовке от техникума:

Белова Дарья Романовка

*фамилия, имя, отчество*  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2025 года

Содержание

[Практическая работа №1 3](#_y6mrh3kpaue6)

Практическая работа №2 11

[Практическая работа №3 25](#_6lwn5nduzsp1)

[Практическая работа №4 31](#_65jla80dc90)

[Практическая работа №5 37](#_xkx4ndto4y7s)

[Итоговый проект 45](#_d2k9azqp6fad)

# Практическая работа №1

## Цель работы

Научиться создавать простые веб-страницы на Spring Boot и обрабатывать ввод пользователя.

## Задание

1. Создать простую страницу с калькулятором с операциями сложения, вычитания, умножения и деления и выводом результата
2. Добавить вторую страницу — конвертер валют с полем для ввода суммы в рублях, выпадающим списком для выбора валюты.

## Описание реализации

### 1. Конфигурация проекта (pom.xml)

Файл pom.xml содержит зависимости проекта:

<dependencies>  
 <!-- Spring Boot Web для создания веб-приложения -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
   
 <!-- Thymeleaf для шаблонов HTML -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
 </dependency>  
   
 <!-- DevTools для горячей перезагрузки -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-devtools</artifactId>  
 <scope>runtime</scope>  
 <optional>true</optional>  
 </dependency>  
</dependencies>

### 2. Главный класс приложения (Application.java)

package com.example;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Application {  
 public static void main(String[] args) {  
 SpringApplication.run(Application.class, args);  
 }  
}

Аннотация @SpringBootApplication объединяет: - @Configuration — класс является источником определений бинов - @EnableAutoConfiguration — автоматическая конфигурация Spring Boot - @ComponentScan — сканирование компонентов в текущем пакете

### 3. Контроллер калькулятора (CalculatorController.java)

package com.example.controller;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
  
@Controller  
public class CalculatorController {  
  
 @GetMapping("/calculator")  
 public String calculator() {  
 return "calculator";  
 }  
  
 @PostMapping("/calculator")  
 public String calculate(  
 @RequestParam("num1") double num1,  
 @RequestParam("num2") double num2,  
 @RequestParam("operation") String operation,  
 Model model) {  
   
 double result = 0;  
 String error = null;  
   
 switch (operation) {  
 case "add":  
 result = num1 + num2;  
 break;  
 case "subtract":  
 result = num1 - num2;  
 break;  
 case "multiply":  
 result = num1 \* num2;  
 break;  
 case "divide":  
 if (num2 == 0) {  
 error = "Ошибка: деление на ноль!";  
 } else {  
 result = num1 / num2;  
 }  
 break;  
 }  
   
 model.addAttribute("num1", num1);  
 model.addAttribute("num2", num2);  
 model.addAttribute("operation", operation);  
   
 if (error != null) {  
 model.addAttribute("error", error);  
 } else {  
 model.addAttribute("result", result);  
 }  
   
 return "calculator";  
 }  
}

Ключевые аннотации: - @Controller — указывает, что класс является контроллером Spring MVC - @GetMapping("/calculator") — обрабатывает GET-запросы на URL /calculator - @PostMapping("/calculator") — обрабатывает POST-запросы (отправка формы) - @RequestParam — извлекает параметры из запроса - Model — объект для передачи данных в шаблон

### 4. Контроллер конвертера валют (CurrencyController.java)

package com.example.controller;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
  
import java.util.LinkedHashMap;  
import java.util.Map;  
  
@Controller  
public class CurrencyController {  
  
 // Курсы валют (рубли к иностранной валюте)  
 private static final Map<String, Double> EXCHANGE\_RATES = new LinkedHashMap<>();  
 private static final Map<String, String> CURRENCY\_NAMES = new LinkedHashMap<>();  
   
 static {  
 EXCHANGE\_RATES.put("USD", 90.0); // 1 USD = 90 RUB  
 EXCHANGE\_RATES.put("EUR", 98.0); // 1 EUR = 98 RUB  
 EXCHANGE\_RATES.put("GBP", 114.0); // 1 GBP = 114 RUB  
 EXCHANGE\_RATES.put("CNY", 12.5); // 1 CNY = 12.5 RUB  
 EXCHANGE\_RATES.put("JPY", 0.6); // 1 JPY = 0.6 RUB  
   
 CURRENCY\_NAMES.put("USD", "Доллар США");  
 CURRENCY\_NAMES.put("EUR", "Евро");  
 CURRENCY\_NAMES.put("GBP", "Фунт стерлингов");  
 CURRENCY\_NAMES.put("CNY", "Китайский юань");  
 CURRENCY\_NAMES.put("JPY", "Японская иена");  
 }  
  
 @GetMapping("/converter")  
 public String converter(Model model) {  
 model.addAttribute("currencies", CURRENCY\_NAMES);  
 return "converter";  
 }  
  
 @PostMapping("/converter")  
 public String convert(  
 @RequestParam("amount") double amount,  
 @RequestParam("currency") String currency,  
 Model model) {  
   
 model.addAttribute("currencies", CURRENCY\_NAMES);  
 model.addAttribute("amount", amount);  
 model.addAttribute("selectedCurrency", currency);  
   
 if (EXCHANGE\_RATES.containsKey(currency)) {  
 double rate = EXCHANGE\_RATES.get(currency);  
 double result = amount / rate;  
 model.addAttribute("result", result);  
 model.addAttribute("currencyName", CURRENCY\_NAMES.get(currency));  
 model.addAttribute("rate", rate);  
 } else {  
 model.addAttribute("error", "Неизвестная валюта");  
 }  
   
 return "converter";  
 }  
}

Особенности реализации: - Используется LinkedHashMap для сохранения порядка валют - Статический блок инициализации для заполнения курсов - Формула конвертации: результат = сумма\_в\_рублях / курс

### 5. Шаблон калькулятора (calculator.html)

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org" lang="ru">  
<head>  
 <meta charset="UTF-8">  
 <title>Калькулятор</title>  
 <link rel="stylesheet" th:href="@{/css/style.css}">  
</head>  
<body>  
 <nav class="navbar">  
 <a th:href="@{/calculator}" class="nav-link active">Калькулятор</a>  
 <a th:href="@{/converter}" class="nav-link">Конвертер валют</a>  
 </nav>  
  
 <div class="container">  
 <div class="card">  
 <h1 class="title">🧮 Калькулятор</h1>  
   
 <form method="post" th:action="@{/calculator}" class="form">  
 <div class="input-group">  
 <label for="num1">Первое число</label>  
 <input type="number" id="num1" name="num1" step="any"   
 th:value="${num1}" required>  
 </div>  
   
 <div class="input-group">  
 <label for="operation">Операция</label>  
 <select id="operation" name="operation" required>  
 <option value="add">➕ Сложение</option>  
 <option value="subtract">➖ Вычитание</option>  
 <option value="multiply">✖️ Умножение</option>  
 <option value="divide">➗ Деление</option>  
 </select>  
 </div>  
   
 <div class="input-group">  
 <label for="num2">Второе число</label>  
 <input type="number" id="num2" name="num2" step="any"   
 th:value="${num2}" required>  
 </div>  
   
 <button type="submit" class="btn">Вычислить</button>  
 </form>  
   
 <!-- Вывод результата -->  
 <div th:if="${result != null}" class="result success">  
 <span class="result-label">Результат:</span>  
 <span class="result-value"   
 th:text="${#numbers.formatDecimal(result, 1, 4)}"></span>  
 </div>  
   
 <!-- Вывод ошибки -->  
 <div th:if="${error != null}" class="result error">  
 <span th:text="${error}"></span>  
 </div>  
 </div>  
 </div>  
</body>  
</html>

Thymeleaf-атрибуты: - th:href="@{/path}" — генерация URL - th:action="@{/calculator}" — URL для отправки формы - th:value="${variable}" — подстановка значения переменной - th:if="${condition}" — условный рендеринг - th:text="${text}" — вывод текста - th:selected="${condition}" — выбор элемента в списке - #numbers.formatDecimal() — форматирование чисел

### 6. Шаблон конвертера валют (converter.html)

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org" lang="ru">  
<head>  
 <meta charset="UTF-8">  
 <title>Конвертер валют</title>  
 <link rel="stylesheet" th:href="@{/css/style.css}">  
</head>  
<body>  
 <nav class="navbar">  
 <a th:href="@{/calculator}" class="nav-link">Калькулятор</a>  
 <a th:href="@{/converter}" class="nav-link active">Конвертер валют</a>  
 </nav>  
  
 <div class="container">  
 <div class="card">  
 <h1 class="title">💱 Конвертер валют</h1>  
   
 <form method="post" th:action="@{/converter}" class="form">  
 <div class="input-group">  
 <label for="amount">Сумма в рублях (₽)</label>  
 <input type="number" id="amount" name="amount"   
 step="0.01" min="0" th:value="${amount}" required>  
 </div>  
   
 <div class="input-group">  
 <label for="currency">Валюта для конвертации</label>  
 <select id="currency" name="currency" required>  
 <option th:each="entry : ${currencies}"   
 th:value="${entry.key}"   
 th:text="${entry.key + ' - ' + entry.value}"  
 th:selected="${entry.key == selectedCurrency}">  
 </option>  
 </select>  
 </div>  
   
 <button type="submit" class="btn">Конвертировать</button>  
 </form>  
   
 <!-- Результат конвертации -->  
 <div th:if="${result != null}" class="result success">  
 <div class="conversion-details">  
 <span th:text="${#numbers.formatDecimal(amount, 1, 2)}"></span> ₽ =   
 <span class="result-value"   
 th:text="${#numbers.formatDecimal(result, 1, 2)}"></span>  
 <span th:text="${selectedCurrency}"></span>  
 </div>  
 <div class="rate-info">  
 <small>Курс: 1 <span th:text="${selectedCurrency}"></span> =   
 <span th:text="${rate}"></span> ₽</small>  
 </div>  
 </div>  
 </div>  
 </div>  
</body>  
</html>

Особенности: - th:each="entry : ${currencies}" — итерация по Map - ${entry.key} и ${entry.value} — доступ к ключу и значению

## Скриншоты работы приложения

### Страница калькулятора

URL*:* <http://localhost:8080/calculator>

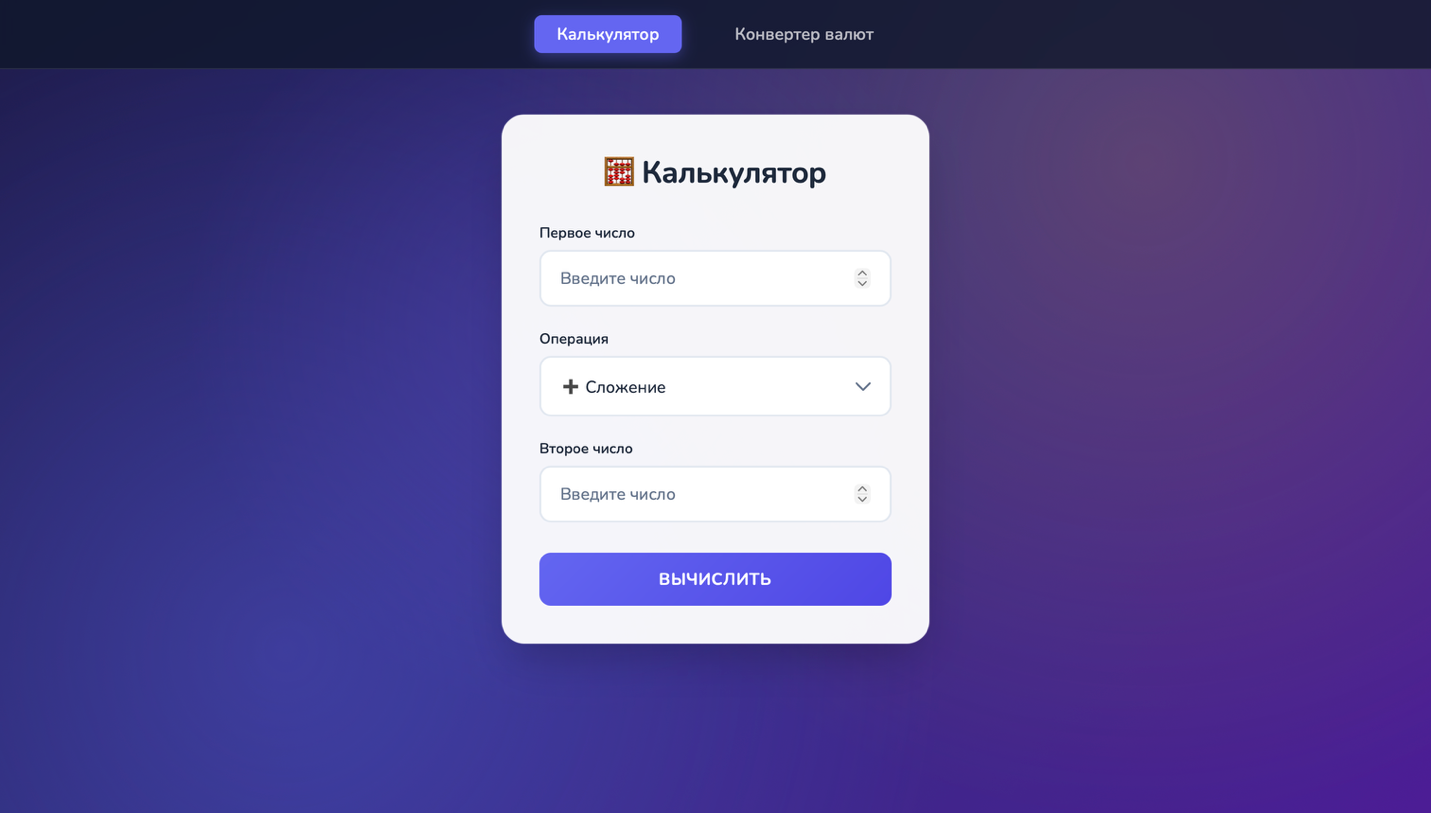


Рисунок 1 - Страница калькулятора

Функционал: - Ввод двух чисел - Выбор операции из выпадающего списка - Вывод результата вычисления - Обработка ошибки деления на ноль

### Страница конвертера валют

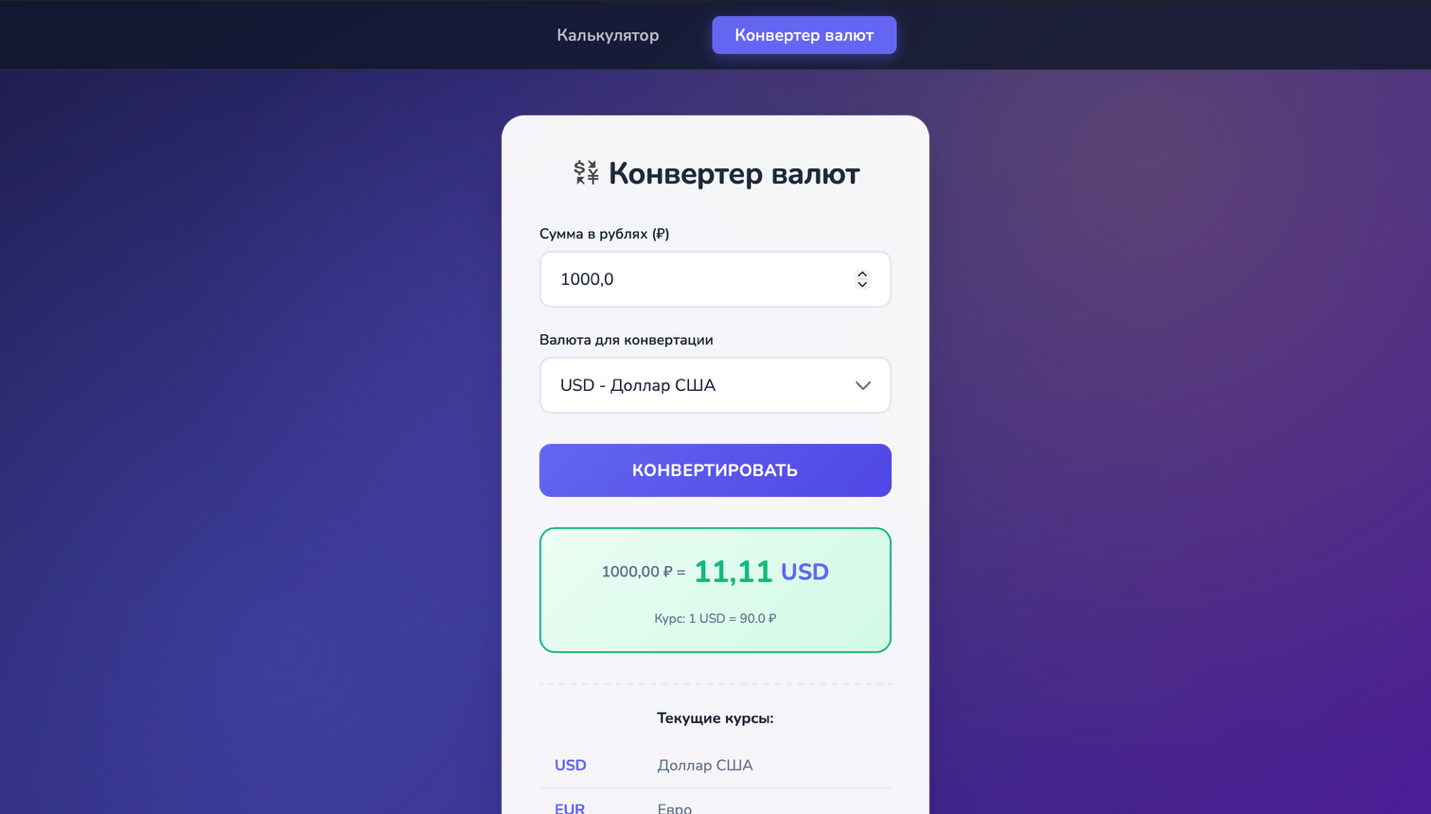
URL: http://localhost:8080/converter

Рисунок 2 - Страница конвертера валют

Функционал: - Ввод суммы в рублях - Выбор целевой валюты из списка (USD, EUR, GBP, CNY, JPY) - Вывод результата конвертации - Отображение текущего курса

Вывод: в ходе выполнения работы было создано веб-приложение на Spring Boot, содержащее 2 страницы: «Калькулятор» и «Конвертер валют». На странице «Калькулятор» реализован функционал базового калькулятора с операциями сложения, вычитания, умножения и деления двух чисел. Добавлена обработка ошибки деления на ноль. На странице «Конвертер валют» реализована конвертация рублей в иностранные валюты (USD, EUR, GBP, CNY, JPY) с использованием выпадающего списка для выбора валюты и отображением текущего курса.

Для обработки HTTP-запросов использованы аннотации:

- `@Controller` — для обозначения класса как контроллера Spring MVC;

- `@GetMapping` — для обработки GET-запросов (отображение страницы);

- `@PostMapping` — для обработки POST-запросов (отправка формы);

- `@RequestParam` — для извлечения параметров из запроса пользователя. Для генерации HTML-страниц использован шаблонизатор Thymeleaf с атрибутами `th:value`, `th:if`, `th:each`, `th:text` для динамического отображения данных.

# Практическая работа №2

## Цель работы

Отработать реализацию MVC-архитектуры и CRUD-операций в Spring Boot приложении для создания полнофункционального веб-приложения с управлением сущностями.

### Сущности

**Product (Товар):**

* id, name, description, price, quantity, category, deleted

**Category (Категория):**

* id, name, description, deleted

### CRUD-операции

| Операция | HTTP-метод | URL | Описание |
| --- | --- | --- | --- |
| Create | POST | /products | Создание товара |
| Read | GET | /products | Список товаров |
| Read | GET | /products/{id} | Просмотр товара |
| Update | POST | /products/{id} | Обновление товара |
| Delete | POST | /products/{id}/soft-delete | Логическое удаление |
| Delete | POST | /products/{id}/hard-delete | Физическое удаление |

### Фильтрация и сортировка

**Фильтрация товаров:**

* По названию (поиск по подстроке)
* По категории
* По диапазону цен (от/до)

**Сортировка:**

* По названию (asc/desc)
* По цене (asc/desc)
* По количеству (asc/desc)

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

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 3 – главная страница

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

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4 – страница товаров

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

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 5 – страница катерогии

## Вывод

В ходе выполнения работы было создано веб-приложение на Spring Boot с реализацией архитектуры MVC. Созданы 2 сущности: «Товар» и «Категория» с полными CRUD-операциями.

Реализована валидация полей с использованием аннотаций @NotBlank, @NotNull, @Size, @Min, @DecimalMin.

Добавлена фильтрация по нескольким параметрам (название, категория, цена) и сортировка (по названию, цене, количеству).

Реализовано множественное удаление в двух режимах: логическое (с возможностью восстановления из корзины) и физическое (безвозвратное).

Использованы аннотации: @Controller, @Service, @Repository, @Entity, @GetMapping, @PostMapping, @RequestParam, @PathVariable, @Valid.