

16. Фреймворк Spring. Создание веб-приложения.

Теоретическое введение.

Spring Framework — это один из самых популярных фреймворков для разработки корпоративных приложений на языке Java. Он предоставляет мощные инструменты для упрощения разработки, управления зависимостями, обработки данных и интеграции с различными технологиями. Spring активно используется для создания веб-приложений, микросервисов и других типов программного обеспечения.

Основные принципы Spring

Инверсия управления (IoC)

Spring использует контейнер Inversion of Control для управления зависимостями между компонентами приложения. Вместо того чтобы создавать объекты и управлять их зависимостями вручную, Spring автоматически настраивает и связывает компоненты.

Внедрение зависимостей (Dependency Injection, DI)

DI позволяет передавать зависимости через конструкторы, сеттеры или поля, снижая связность компонентов и упрощая их тестирование.

Модульность и расширяемость

Spring — это набор модулей (Core, Data Access, Web, Security и др.), что позволяет использовать только те компоненты, которые нужны для проекта.

Легковесность

Несмотря на широкий функционал, Spring остается относительно легковесным, с минимальными затратами на инициализацию и управление ресурсами.

Ключевые модули Spring Framework

Spring Core

Основной модуль, включающий ядро фреймворка, контейнер IoC и DI.

Он обеспечивает управление жизненным циклом объектов и их зависимостей.

Spring AOP (Aspect-Oriented Programming)

Модуль, позволяющий реализовывать аспекты, такие как логирование, транзакции и безопасность, не влияя на основной код.

Spring Data

Модуль для упрощения работы с базами данных, включая ORM (например, Hibernate) и интеграцию с NoSQL-хранилищами.

Spring Web

Компоненты для создания веб-приложений и RESTful API.

Включает поддержку MVC (Model-View-Controller).

Spring Security

Фреймворк для управления безопасностью приложения, включая аутентификацию, авторизацию и защиту от атак (CSRF, SQL-инъекции).

Spring Boot

Дополнение к Spring Framework, которое позволяет быстро создавать приложения с минимальной настройкой. Spring Boot предоставляет встроенный сервер (Tomcat/Jetty), автоконфигурацию и удобный способ управления зависимостями.

Начало работы в Spring

Перед началом работы с фреймворком необходимо создать сам Spring – проект. Некоторые IDE позволяют это сделать автоматически, в ином случае можно воспользоваться сервисом [Spring Initializr](#) (Рисунок 16.1). Это официальный сервис, в котором можно настроить необходимые параметры своего будущего Spring – приложения и скачать базовый проект с уже готовой структурой для дальнейшей разработки.

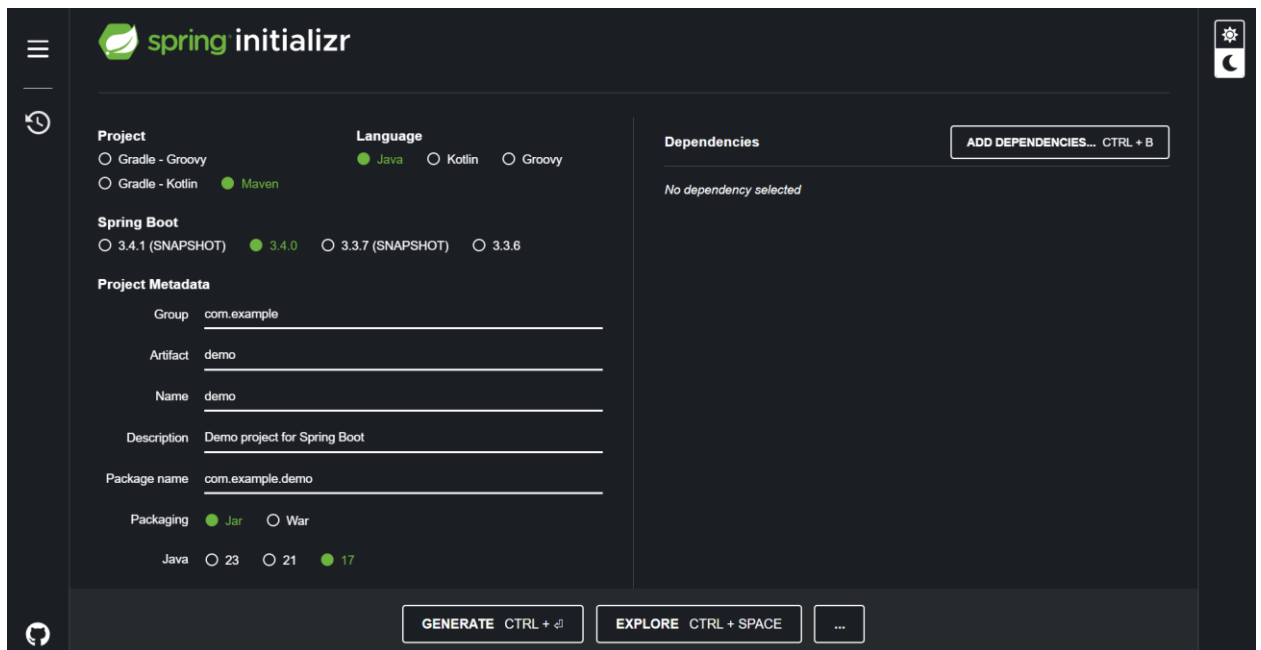


Рисунок 16.1 – Интерфейс Spring Initializr

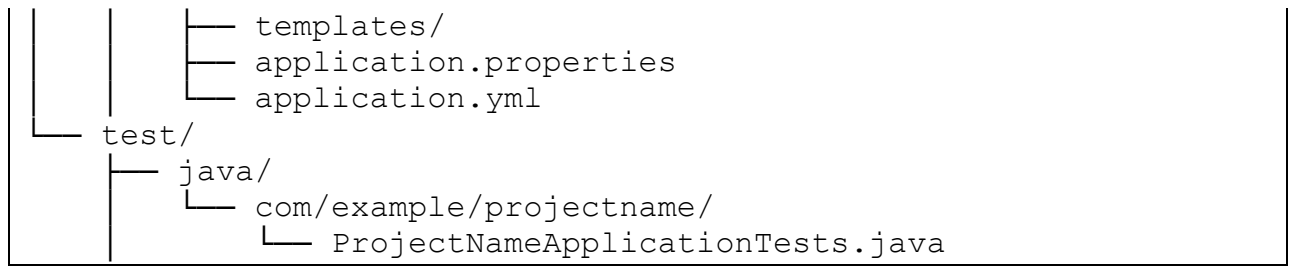
В сервисе можно указать такие параметры, как: название проекта, название пакета, описание, версию Java и Spring Boot. Также можно выбрать желаемый сборщик проекта и необходимые зависимости. После настройки необходимо нажать на кнопку «Generate», чтобы скачать архив с вашим проектом.

Структура проекта

Как и в любом другом фреймворке, в Spring имеется определенная структура проекта, которая помогает разграничить логику кода. Ниже представлена общая структура файлов в Spring – проекте (Листинг 16.1):

Листинг 16.1 – Структура Spring – проекта

```
src/
├── main/
│   ├── java/
│   │   └── com/example/projectname/
│   │       ├── config/
│   │       ├── controller/
│   │       ├── dto/
│   │       ├── exception/
│   │       ├── model/
│   │       ├── repository/
│   │       ├── service/
│   │       ├── util/
│   │       └── ProjectNameApplication.java
│   └── resources/
│       └── static/
```



Главный пакет проекта - `src/main/java/com/example/projectname/`. Здесь размещаются все основные компоненты приложения.

config/

Хранит файлы конфигурации, например настройки Spring Security, CORS, или другие специфические конфигурации.

controller/

Содержит REST или MVC-контроллеры, которые обрабатывают HTTP-запросы и возвращают ответы.

dto/ (Data Transfer Objects)

Классы для передачи данных между слоями приложения (например, для запросов или ответов).

exception/

Классы для обработки ошибок и исключений (например, `GlobalExceptionHandler`).

model/

Содержит классы-сущности (Entity), которые представляют данные в базе данных.

repository/

Интерфейсы, расширяющие `JpaRepository` или `CrudRepository`, для взаимодействия с базой данных.

service/

Содержит бизнес-логику приложения. Здесь размещаются классы, которые обрабатывают данные между контроллерами и репозиториями.

util/

Утилитарные классы или вспомогательные методы, которые можно использовать в разных частях приложения.

ProjectNameApplication.java

Основной класс запуска Spring Boot приложения с аннотацией @SpringBootApplication.

Пример реализации

Подробнее рассмотрим структуру проекта и кода на примере небольшого веб-приложения с аутентификацией пользователя.

Данный пример представляет собой минимальный набор элементов для запуска Spring – приложения.

На рисунке 16.2 представлена иерархия файлов проекта.

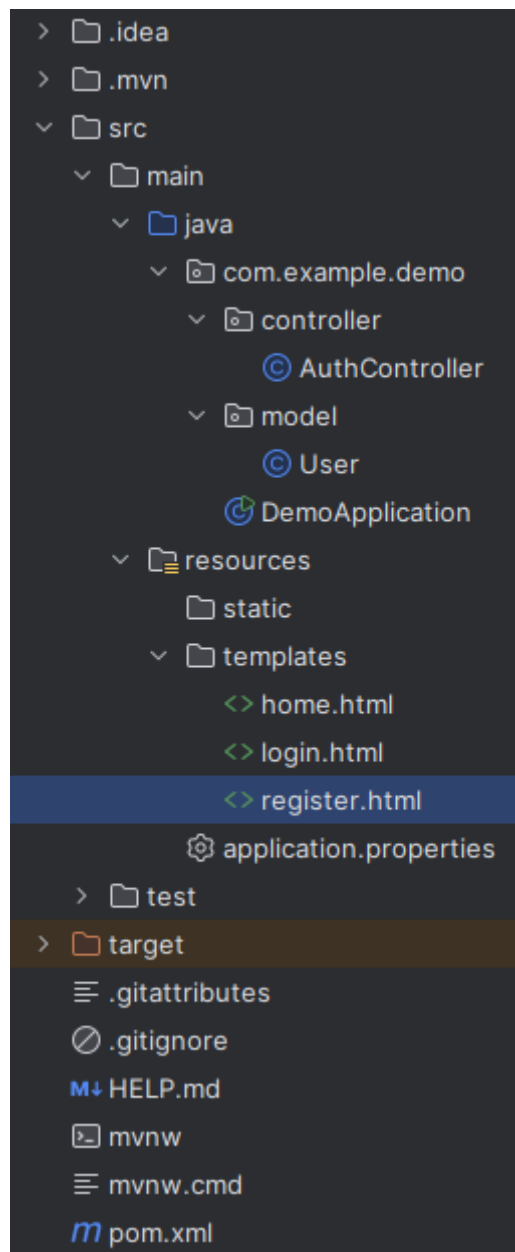


Рисунок 16.2 – Иерархия файлов проекта

Рассмотрим каждый из файлов по порядку.

Файл `pom.xml` является файлом зависимостей проекта, если вы используете сборщик Maven. В нем указаны все необходимые зависимости инструменты, используемые в проекте. Ниже представлено содержимое файла `pom.xml`.

Листинг 16.2 – `pom.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.4.0</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.example</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>demo</name>
  <description>Demo project for Spring Boot</description>
  <url/>
  <licenses>
    <license/>
  </licenses>
  <developers>
    <developer/>
  </developers>
  <scm>
    <connection/>
    <developerConnection/>
    <tag/>
    <url/>
  </scm>
  <properties>
    <java.version>23</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

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

```

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

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>

```

Файл `User` в пакете `model` описывает собой класс сущности, использующийся для хранения данных о пользователе. С помощью подобных классов реализовать структуру базы данных в Spring – проекте. В нашем случае данный класс просто хранит в себе данные о пользователе, конструктор, геттеры и сеттеры.

Листинг 16.3 – `User.java`

```

package com.example.demo.model;

public class User {
    private String username;
    private String password;

    // Конструкторы
    public User() {}

    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }

    // Геттеры и сеттеры
    public String getUsername() {
        return username;
    }

```

```

    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}

```

В файле `AuthController` хранятся методы обработки веб-страниц и данных форм. Контроллеры – одна из главных составляющих Spring. С их помощью реализуется вся логика веб-приложения. Ниже представлен контроллер, который отвечает за отображение HTML – страниц `home`, `login` и `register`, а также за обработку данных форм регистрации и авторизации.

Листинг 16.4 – `AuthController.java`

```

package com.example.demo.controller;
// Определение пакета, к которому относится данный контроллер.
// Это помогает организовать код и структурировать его.

import com.example.demo.model.User;
// Импорт класса User, который используется для работы с данными
// пользователя.

import org.springframework.stereotype.Controller;
// Импорт аннотации @Controller. Она говорит Spring, что данный
// класс является контроллером.

import org.springframework.ui.Model;
// Импорт для работы с объектом Model, который позволяет
// передавать данные между контроллером и представлением.

import org.springframework.web.bind.annotation.*;
// Импорты для работы с аннотациями, такими как @GetMapping,
// @PostMapping и другими, чтобы обрабатывать HTTP-запросы.

@Controller
// Аннотация, которая сообщает Spring, что данный класс является
// контроллером.
public class AuthController {
// Определение класса-контроллера AuthController, который будет
// обрабатывать запросы авторизации и регистрации пользователей.

```



```
private User registeredUser;
// Переменная для хранения информации о зарегистрированном
пользователе. По умолчанию равна null.

private boolean isAuthenticated = false;
// Переменная-флаг, показывающая, авторизован ли
пользователь в данный момент.

@GetMapping("/")
// Аннотация для обработки GET-запроса по корневому URL
("/").
public String showRegisterPage() {
    // Метод, который будет вызываться при GET-запросе по
корневому URL.
    return "register";
    // Возвращает название представления (HTML-шаблона)
"register", которое отображается пользователю.
}

@PostMapping("/register")
// Аннотация для обработки POST-запроса по URL "/register".
public String register(@RequestParam String username,
@RequestParam String password, Model model) {
    // Метод обработки данных из формы регистрации.
    if (username.isEmpty() || password.isEmpty()) {
        // Проверяем, не пустые ли значения имени
пользователя и пароля.
        model.addAttribute("error", "Имя пользователя и
пароль не должны быть пустыми!");
        // Если пустые, отправляем сообщение об ошибке в
представление.
        return "register";
        // Возвращаемся на страницу регистрации.
    }
    registeredUser = new User(username, password);
    // Создаём нового пользователя с полученными данными и
сохраняем в переменную registeredUser.
    return "redirect:/login";
    // Перенаправляем пользователя на страницу входа
("/login").
}

@GetMapping("/login")
// Аннотация для обработки GET-запроса по URL "/login".
public String showLoginPage() {
    return "login";
    // Возвращает представление (HTML-шаблон) "login".
}

@PostMapping("/login")
// Аннотация для обработки POST-запроса по URL "/login".
public String login(@RequestParam String username,
@RequestParam String password, Model model) {
```

```

        // Метод обработки данных из формы входа.
        if (registeredUser == null ||
!registeredUser.getUsername().equals(username) ||
        !registeredUser.getPassword().equals(password)) {
            // Проверяем, зарегистрирован ли пользователь и
            совпадают ли данные с формой.
            model.addAttribute("error", "Неверное имя
пользователя или пароль!");
            // Если данные некорректны, отправляем сообщение об
            ошибке.

            return "login";
            // Возвращаемся на страницу входа.
        }
        isAuthenticated = true;
        // Если данные корректны, устанавливаем флаг авторизации
        в true.
        return "redirect:/home";
        // Перенаправляем пользователя на домашнюю страницу
        после успешного входа.
    }

    @GetMapping("/home")
    // Аннотация для обработки GET-запроса по URL "/home".
    public String showHomePage(Model model) {
        if (!isAuthenticated) {
            // Проверяем, авторизован ли пользователь. Если нет:
            return "redirect:/login";
            // Перенаправляем его на страницу входа.
        }
        model.addAttribute("username",
registeredUser.getUsername());
        // Если пользователь авторизован, передаём имя
        пользователя в представление.
        return "home";
        // Возвращаем представление (HTML-шаблон) "home".
    }
}

```

Класс `DemoApplication` является главной точкой входа. С его помощью происходит запуск всего веб-приложения.

Листинг 16.5 – `DemoApplication.java`

```

package com.example.demo;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {

```

```
SpringApplication.run(DemoApplication.class, args);  
}  
  
}
```

В папке `templates` хранятся `html` страницы, которые обрабатывает контроллер. Обработка, в том числе, происходит с помощью обработчика шаблонов `Thymeleaf`. Он предназначен для динамического создания `HTML`-страниц и позволяет легко связывать данные из контроллера с представлением.

Основные особенности Thymeleaf

Шаблонный движок: `Thymeleaf` используется для генерации `HTML`, динамически заменяя переменные данными из сервера.

Поддержка стандартного HTML:

Шаблоны `Thymeleaf` пишутся на обычном `HTML`, с добавлением специфичных атрибутов для динамической обработки.

Например: `th:text`, `th:href`, `th:if`.

Простота интеграции с Spring:

`Thymeleaf` полностью совместим с `Spring Boot` и `Spring MVC`.

Он автоматически обрабатывает данные из `Model` и привязывает их к `HTML`-элементам.

HTML-валидность при разработке:

В отличие от некоторых других движков (например, `JSP`), `Thymeleaf` позволяет разработчикам видеть корректный `HTML` и в браузере при отладке без необходимости обработки сервером.

Условные операторы и итерации:

Поддерживает условные выражения и итерации прямо в `HTML`-шаблонах.

Интерактивность и динамические данные:

Позволяет динамически вставлять переменные, создавать ссылки, условно показывать или скрывать блоки и т.д.

Ниже представлены листинги html – страниц. Желтым цветом выделены поля, которые обрабатываются с помощью контроллеров и шаблонизатора.

Листинг 16.6 – register.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Register</title>
</head>
<body>
<h1>Регистрация</h1>
<div th:if="${error}" style="color: red;">[[${error}]]</div>
<form action="#" th:action="@{/register}" method="post">
    <label>Имя пользователя: <input type="text" name="username"
/></label><br/>
    <label>Пароль: <input type="password" name="password"
/></label><br/>
    <input type="submit" value="Зарегистрироваться" />
</form>
</body>
</html>
```

Листинг 16.7 – login.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Login</title>
</head>
<body>
<h1>Вход</h1>
<div th:if="${error}" style="color: red;">[[${error}]]</div>
<form action="#" th:action="@{/login}" method="post">
    <label>Имя пользователя: <input type="text" name="username"
/></label><br/>
    <label>Пароль: <input type="password" name="password"
/></label><br/>
    <input type="submit" value="Войти" />
</form>
</body>
</html>
```

Листинг 16.8 – home.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Home</title>
</head>
<body>
<h1>Добро пожаловать, <span
```

```
th:text="{username}">Пользователь</span></h1>  
</body>  
</html>
```

После сборки и запуска проекта, перейдя по ссылке <http://localhost:8080>, пользователь попадает на страницу регистрации, где может ввести данные для входа. Далее происходит переход на страницу авторизации, где пользователь подтверждает свои данные. После успешного входа пользователь попадает на домашнюю страницу, на которой выводится имя пользователя (Рисунок 16.3, Рисунок 16.4, Рисунок 16.5).

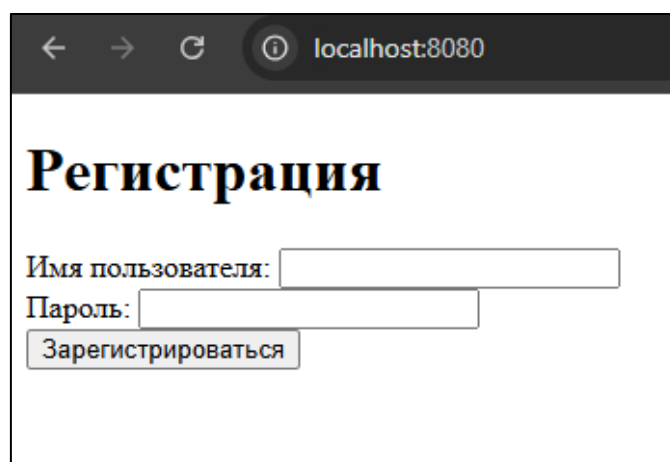


Рисунок 16.3 – Страница регистрации

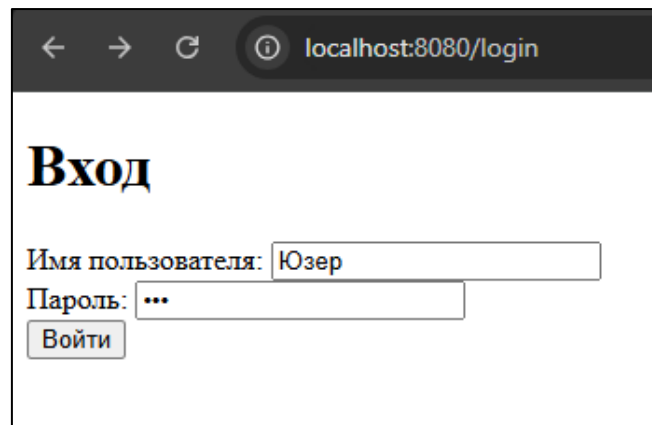


Рисунок 16.4 – Страница авторизации

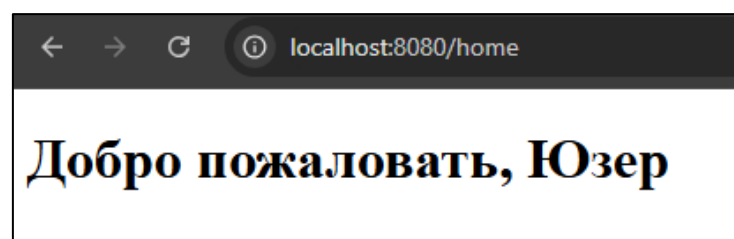


Рисунок 16.5 – Домашняя страница

Практическое задание

В данной практической работе вам необходимо разработать Spring – приложение с регистрацией и аутентификацией пользователя. Успешно вошедший в систему пользователь должен попасть на домашнюю страницу, на которой будет располагаться реализация задания по вариантам.

Задания по вариантам:

Вариант 1.

Реализовать обработчик для формы обратной связи.

```
<h2>Обратная связь</h2>

    <form th:action="@{/home}" method="post">
        <input                type="hidden"                name="formType"
value="feedback" />
        <label>Имя:</label>
        <input type="text" name="name" required /><br/>
        <label>Email:</label>
        <input type="email" name="email" required /><br/>
        <label>Сообщение:</label>
        <textarea name="message" required></textarea><br/>
        <button type="submit">Отправить</button>
    </form>

    <p th:if="${feedback}" th:text="${feedback}"></p>
```

Вариант 2.

Реализовать обработчик для формы поиска по ключевому слову.

```
<h2>Поиск</h2>

    <form th:action="@{/home}" method="post">
        <input type="hidden" name="formType" value="search"
/>

        <label>Ключевое слово:</label>
        <input type="text" name="query" required />
        <button type="submit">Поиск</button>
    </form>
```

```

        <h3 th:if="{query}">Результаты поиска для: <span
th:text="{query}"></span></h3>
        <ul th:if="{results != null}" th:each="result :
{results}">
            <li th:text="{result}"></li>
        </ul>

```

Вариант 3.

Реализовать обработчик для формы голосования.

```

<h2>Голосование</h2>
    <form th:action="@{/home}" method="post">
        <input type="hidden" name="formType" value="vote" />
        <p>Ваш любимый язык программирования:</p>
        <label><input type="radio" name="option"
value="Java" required /> Java</label><br/>
        <label><input type="radio" name="option"
value="Python" /> Python</label><br/>
        <label><input type="radio" name="option" value="C++"
/> C++</label><br/>
        <button type="submit">Голосовать</button>
    </form>
    <p th:if="{voteResult}" th:text="{voteResult}"></p>

```

Вариант 4.

Реализовать обработчик для формы конвертера валют.

```

<h2>Конвертер валют</h2>
    <form th:action="@{/home}" method="post">
        <input type="hidden" name="formType" value="convert"
/>
        <label>Сумма:</label>
        <input type="number" name="amount" step="0.01"
required /><br/>
        <label>Из валюты:</label>
        <select name="fromCurrency">

```

```

        <option value="USD">USD</option>
        <option value="EUR">EUR</option>
    </select><br/>
    <label>В валюту:</label>
    <select name="toCurrency">
        <option value="RUB">RUB</option>
        <option value="USD">USD</option>
    </select><br/>
    <button type="submit">Конвертировать</button>
</form>

<p
                                th:if="${conversionResult}"
th:text="${conversionResult}"></p>

```

Вариант 5.

Реализовать обработчик для формы отзывов.

```

<h2>Оставить отзыв</h2>
    <form th:action="@{/home}" method="post">
        <input type="hidden" name="formType" value="review"
/>

        <textarea name="review" required></textarea><br/>
        <button type="submit">Отправить</button>
    </form>
    <ul th:if="${reviews != null}" th:each="review :
${reviews}">
        <li th:text="${review}"></li>
    </ul>

```