Лабораторная работа №2. Spring MVC

Spring Boot MVC (Model-View-Controller) представляет собой архитектурный шаблон для разработки веб-приложений с использованием Spring Boot, который обеспечивает модульность, отделение ответственностей и упрощение разработки.

Вот краткое описание каждого компонента MVC в контексте Spring Boot:

Модель (Model):

- В Spring Boot модель представляет собой объекты данных или бизнес-объекты, которые отвечают за хранение данных и бизнес-логику приложения.
- Модель часто представлена классами Java, которые могут быть сущностями JPA (Java Persistence API) для взаимодействия с базой данных или просто объектами данных.
- Модель не содержит никакой логики отображения или представления данных.

Представление (View):

- Представление в Spring Boot MVC представляет собой пользовательский интерфейс, который отображает данные пользователю.
- Веб-страницы, HTML, шаблоны Thymeleaf или любые другие технологии отображения могут использоваться в качестве представлений.
- Представление отвечает за отображение данных из модели пользователю и за обработку пользовательского ввода.

Контроллер (Controller):

- Контроллеры в Spring Boot MVC отвечают за управление взаимодействием между моделью и представлением.
- Они обрабатывают запросы от клиентов, извлекают необходимые данные из модели, выполняют необходимую бизнес-логику и передают результаты представления для отображения пользователю.
- В Spring Boot контроллеры обычно аннотированы аннотацией @Controller или @RestController, которые указывают Spring, что они являются компонентами, обрабатывающими HTTP-запросы.

Spring Boot MVC обеспечивает гибкую и мощную среду для разработки веб-приложений, делая процесс разработки проще и более структурированным. Он также интегрируется с другими функциональными возможностями Spring Boot, такими как внедрение зависимостей, управление

транзакциями, обеспечение безопасности и тестирование, что делает его популярным выбором для разработчиков.

Цель работы: Получить практические навыки работы с Spring Framework.

Пример выполнения работы.

Начнем с определения темы нашего приложения. Определение темы поможет нам лучше понять, какие классы и функциональность нам понадобятся.

Создадим простое веб-приложение для управления задачами (to-do list). Это довольно популярная и полезная тема для практики, так как включает в себя множество основных концепций Spring MVC, таких как создание контроллеров, работа с моделью и представлением, управление запросами и маршрутизация и т. д.

Итак, наше приложение будет иметь следующие классы:

Task: Этот класс будет представлять собой модель задачи. Он будет содержать информацию о задаче, такую как название, описание, статус выполнения и т.д.

TaskController: Этот класс будет контроллером нашего приложения. Он будет обрабатывать HTTP-запросы от клиентов, взаимодействовать с сервисным слоем для выполнения операций с задачами и возвращать соответствующие представления.

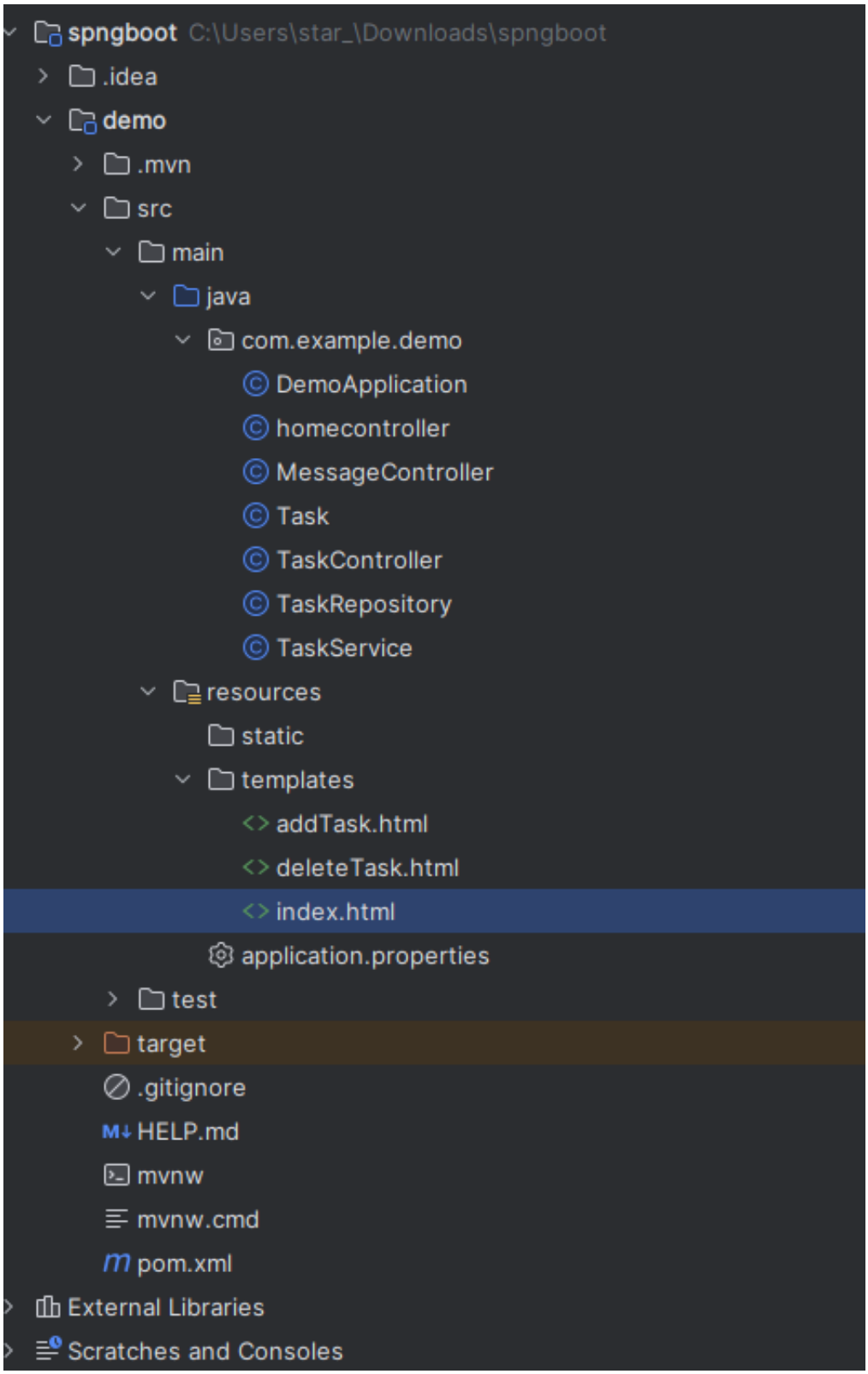
TaskService: Этот класс будет представлять собой сервисный слой нашего приложения. Он будет содержать логику для выполнения операций с задачами, таких как создание, обновление, удаление и получение задач.

TaskRepository: Этот класс будет представлять собой слой доступа к данным (Data Access Layer).

index.html: Это будет представление (View) для отображения списка задач, их создания, редактирования и удаления. Мы будем использовать шаблон Thymeleaf для создания этого представления.

Файлы с разметкой страницы (.html) должны находиться в проекте по адресу src/main/resources/templates/

Пример структуры приложения приведен на скриншоте.



Теперь рассмотрим процесс работы приложения:

Когда пользователь отправляет запрос на главную страницу приложения, контроллер TaskController обрабатывает этот запрос и возвращает представление index.html, передавая список задач.

Пользователь видит список задач на главной странице и может добавлять новые задачи, удалять существующие задачи и обновлять статус выполнения задач.

Для выполнения этих операций используются соответствующие методы в контроллере, которые взаимодействуют с сервисом TaskService и репозиторием TaskRepository.

После выполнения операции пользователь перенаправляется на главную страницу, где он видит обновленный список задач.

В итоге, пользователь может управлять списком задач через веб-интерфейс приложения, а операции с данными выполняются на сервере.

В данном проекте использовались следующие зависимости:

spring-boot-starter: Это агрегированная зависимость, которая включает в себя основные зависимости Spring Boot для создания приложения. Включает в себя всё необходимое для запуска приложения Spring Boot.

spring-boot-starter-web: Эта зависимость включает в себя все необходимые зависимости для разработки веб-приложения с использованием Spring MVC. Включает в себя Tomcat в качестве встроенного контейнера сервлетов.

spring-boot-starter-thymeleaf: Эта зависимость добавляет поддержку Thymeleaf в ваше приложение. Thymeleaf - это шаблонизатор HTML, который позволяет разработчикам создавать динамические веб-страницы с использованием шаблонов и выражений в HTML-коде.

Файл pom.xml находится в приложении.

Как уже было сказано Task.java хранит данные о задаче.

```
public class Task {
    private static Long nextId = 1L; // Статическая переменная для генерации
уникальных идентификаторов задач
    private Long id; // Уникальный идентификатор задачи
    private String title; // Название задачи
    private String description; // Описание задачи
    private boolean completed; // Статус задачи (выполнена или нет)

    // Конструкторы
    public Task() {
        this.id = generateId();
    }

    public Task(String title, String description) {
        this.id = generateId();
        this.title = title;
        this.description = description;
        this.completed = false; // По умолчанию задача не выполнена
    }

    // Генерация уникального идентификатора задачи
    private synchronized Long generateId() {
        return nextId++;
    }

    // Геттеры и сеттеры
    public Long getId() {
        return id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public boolean isCompleted() {
        return completed;
    }

    public void setCompleted(boolean completed) {
        this.completed = completed;
    }

    // Переопределение метода toString() для удобного вывода информации о
задаче
    @Override
    public String toString() {
        return "Task{" +
                "id=" + id +
                ", title='" + title + '\'' +
                ", description='" + description + '\'' +
```

```
                ", completed=" + completed +
                '}';
    }
}
```

private static Long nextId = 1L; - Статическая переменная, используемая для генерации уникальных идентификаторов задач. При каждом создании новой задачи она увеличивается на 1.

private Long id; - Уникальный идентификатор задачи.

private String title; - Название задачи.

private String description; - Описание задачи.

private boolean completed; - Переменная, отражающая статус задачи (выполнена или нет).

Конструкторы:

public Task(): Конструктор по умолчанию, который генерирует уникальный идентификатор для задачи.

public Task(String title, String description): Конструктор, который принимает название и описание задачи и также генерирует уникальный идентификатор.

private synchronized Long generateId(): Метод, используемый для генерации уникального идентификатора задачи. Он синхронизирован для обеспечения безопасности в многопоточной среде.

Геттеры и сеттеры для получения и установки значений полей объекта.

Класс TaskController обрабатывает запросы

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

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

@Controller
public class TaskController {

    private List<Task> taskList = new ArrayList<>();

    @GetMapping("/")
    public String index(Model model) {
        model.addAttribute("tasks", taskList);
        return "index";
```

```
    }

    @PostMapping("/addTask")
    public String addTask(@ModelAttribute Task task) {
        taskList.add(task);
        return "redirect:/";
    }
    @GetMapping("/deleteTask/{id}")
    public String deleteTask(@PathVariable Long id) {
        taskList.removeIf(task -> task.getId().equals(id));
        return "redirect:/";
    }
    @PostMapping("/updateTask/{id}")
    public String updateTask(@PathVariable Long id, @RequestParam boolean
completed) {
        for (Task task : taskList) {
            if (task.getId().equals(id)) {
                task.setCompleted(completed);
                break;
            }
        }
        return "redirect:/";
    }
}
```

@Controller: Аннотация, указывающая, что этот класс является контроллером веб-приложения.

private List<Task> taskList = new ArrayList<>();: Список задач, который будет управляться этим контроллером.

@GetMapping("/"): Метод обработки HTTP GET запросов к корневому URL. Он отображает список задач на странице index.

public String index(Model model): Метод, который принимает модель и добавляет список задач в неё перед отображением.

@PostMapping("/addTask"): Метод обработки HTTP POST запросов для добавления новой задачи.

public String addTask(@ModelAttribute Task task): Метод, который принимает объект задачи через атрибут модели и добавляет его в список задач.

@GetMapping("/deleteTask/{id}"): Метод обработки HTTP GET запросов для удаления задачи по её идентификатору

public String deleteTask(@PathVariable Long id): Метод, который принимает идентификатор задачи в качестве переменной пути и удаляет задачу с соответствующим идентификатором из списка задач.

@PostMapping("/updateTask/{id}"): Метод обработки HTTP POST запросов для обновления статуса задачи.

public String updateTask(@PathVariable Long id, @RequestParam boolean completed): Метод, который принимает идентификатор задачи и новый статус выполнения, а затем обновляет статус задачи с данным идентификатором.

Все методы возвращают строку "redirect:/", что означает перенаправление пользователя на главную страницу после выполнения операции.

TaskRepository – класс для доступа к данным

```
import org.springframework.stereotype.Repository;

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

@Repository
public class TaskRepository {

    private List<Task> taskList = new ArrayList<>();

    // Метод для добавления задачи в репозиторий
    public void addTask(Task task) {
        taskList.add(task);
    }

    // Метод для получения всех задач из репозитория
    public List<Task> getAllTasks() {
        return taskList;
    }

    // Метод для удаления задачи из репозитория по её идентификатору
    public void deleteTask(Long id) {
        taskList.removeIf(task -> task.getId().equals(id));
    }
}
```

Вот его основные особенности и функции:

@Repository: Аннотация, указывающая, что этот класс является компонентом репозитория, предназначенным для работы с данными.

private List<Task> taskList = new ArrayList<>();: Список задач, который будет храниться в репозитории.

public void addTask(Task task): Метод для добавления задачи в репозиторий.

public List<Task> getAllTasks(): Метод для получения всех задач из репозитория.

public void deleteTask(Long id): Метод для удаления задачи из репозитория по её идентификатору.

Все методы выполняют операции непосредственно с объектами в списке задач.

Этот класс отвечает за управление задачами в памяти приложения.

TaskService – создание и удаление задач

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class TaskService {

    private final TaskRepository taskRepository;

    @Autowired
    public TaskService(TaskRepository taskRepository) {
        this.taskRepository = taskRepository;
    }

    // Метод для добавления новой задачи
    public void addTask(Task task) {
        taskRepository.addTask(task);
    }

    // Метод для получения всех задач
    public List<Task> getAllTasks() {
        return taskRepository.getAllTasks();
    }

    // Метод для удаления задачи по её идентификатору
    public void deleteTask(Long id) {
        taskRepository.deleteTask(id);
    }
}
```

@Service: Аннотация, указывающая, что этот класс является компонентом сервиса, предназначенным для выполнения бизнес-логики приложения.

private final TaskRepository taskRepository;: Приватное поле, содержащее экземпляр TaskRepository, используемый для взаимодействия с данными о задачах.

@Autowired: Аннотация, указывающая Spring на автоматическое внедрение зависимостей, в данном случае TaskRepository, через конструктор.

public TaskService(TaskRepository taskRepository): Конструктор класса, принимающий TaskRepository в качестве зависимости.

public void addTask(Task task): Метод сервиса для добавления новой задачи. Он делегирует эту операцию объекту taskRepository.

public List<Task> getAllTasks(): Метод сервиса для получения всех задач. Он также делегирует эту операцию объекту taskRepository.

public void deleteTask(Long id): Метод сервиса для удаления задачи по её идентификатору. Он также использует объект taskRepository для выполнения этой операции.

Теперь рассмотрим интерфейс web-приложения.

Структура файла разметки страницы Index.html:

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
   <meta charset="UTF-8">
   <title>ToDo Tracker</title>
</head>
<body>
<h1>ToDo List</h1>
<ul>
   <!-- Здесь будут отображаться задачи -->
   <li th:each="task : ${tasks}">
      <span th:text="${task.title}"></span>
      <span th:text="${task.description}"></span>
      <span th:text="${task.completed ? 'Completed' : 'Pending'}"></span>
      <a th:href="@{/deleteTask/{id}(id=${task.id})}">Delete</a>
      <form th:action="@{/updateTask/{id}(id=${task.id})}" method="post" style="display:inline;">
         <input type="hidden" name="completed" value="true">
         <button type="submit">Mark as Completed</button>
      </form>
   </li>
</ul>
<h2>Add New Task</h2>
<form th:action="@{/addTask}" method="post">
   <label for="title">Title:</label>
   <input type="text" id="title" name="title">
   <label for="description">Description:</label>
   <input type="text" id="description" name="description">
   <button type="submit">Add Task</button>
</form>
</body>
</html>
```

Этот HTML-код представляет собой шаблон веб-страницы для отображения списка задач и добавления новых задач в веб-приложении ToDo Tracker. Вот его основные элементы:

<!DOCTYPE html>: Объявление типа документа.

<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">: Открывающий тег HTML с указанием языка и пространства имён для использования атрибутов Thymeleaf.

<head>: Секция заголовка документа.

<meta charset="UTF-8">: Указание кодировки символов.
<title>ToDo Tracker</title>: Заголовок страницы.
<body>: Основное содержимое страницы.

<h1>ToDo List</h1>: Заголовок, отображающий название списка задач.

: Начало списка задач.

<li th:each="task : ${tasks}">: Этот тег используется для каждой задачи в списке задач.

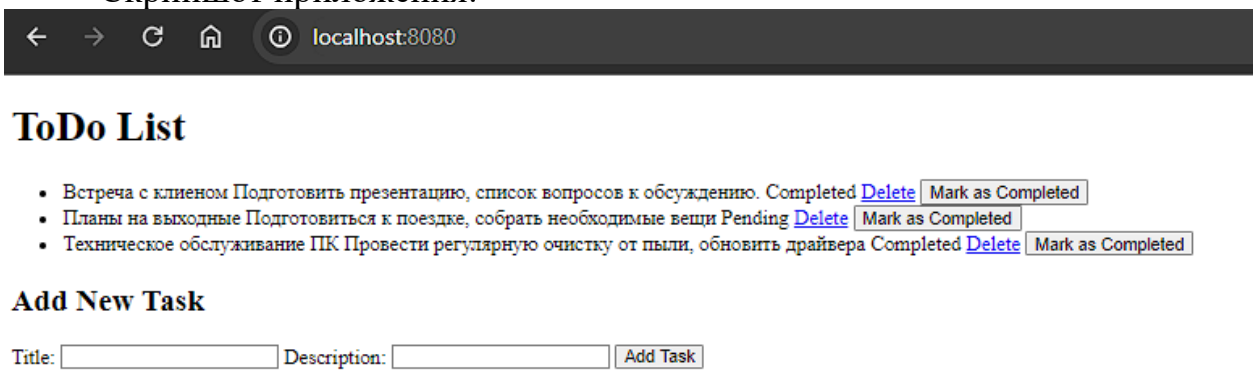
: Отображает название задачи.
: Отображает описание задачи.
: Отображает статус задачи (выполнена или ожидает выполнения).
<a th:href="@{/deleteTask/{id}(id=${task.id})}">Delete: Ссылка для удаления задачи.
<form th:action="@{/updateTask/{id}(id=${task.id})}" method="post" style="display:inline;">: Форма для обновления статуса задачи.
<input type="hidden" name="completed" value="true">: Скрытое поле, указывающее, что задача будет помечена как выполненная.
<button type="submit">Mark as Completed</button>: Кнопка для отправки формы и пометки задачи как выполненной.
<h2>Add New Task</h2>: Заголовок для добавления новой задачи.

<form th:action="@{/addTask}" method="post">: Форма для добавления новой задачи.

<label for="title">Title:</label>: Метка для поля ввода названия задачи.
<input type="text" id="title" name="title">: Поле ввода для названия задачи.
<label for="description">Description:</label>: Метка для поля ввода описания задачи.
<input type="text" id="description" name="description">: Поле ввода для описания задачи.
<button type="submit">Add Task</button>: Кнопка для отправки формы и добавления новой задачи.

Этот шаблон использует Thymeleaf для динамической генерации контента на основе данных, передаваемых из контроллера, таких как список задач и их атрибуты.

Скриншот приложения:



Задание:

1. Изучить теоретический материал;
2. Написать приложение следуя примеру;
3. Выполнить дополнительное задание;
4. Сделать отчет.

Дополнительное задание.

Расширьте функционал приложения.

Варианты заданий:

1. Разработать функционал для создания категорий задач (например, "Учеба", "Личные дела", "Работа").
2. Установка сроков выполнения задач. Добавить поля для установки сроков выполнения задач.
3. Добавьте уведомление с согласием на удаление заметки.
4. Добавьте возможность оставлять комментарии к заметкам.
5. Добавьте поиск по заметкам.
6. Улучшите внешний вид страницы index.html

Приложение 1

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>3.2.3</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>
    <properties>
       <java.version>21</java.version>
    </properties>
    <dependencies>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter</artifactId>
       </dependency>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-test</artifactId>
          <scope>test</scope>
       </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-thymeleaf</artifactId>
       </dependency>
    </dependencies>

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

</project>
```