# **스프링 부트와 Thymeleaf를 이용한 웹 ToDo 애플리케이션 개발 가이드 (2단계 - Gradle 버전)**

## **1. 시작하며**

이 가이드는 Java 콘솔 기반 ToDo 애플리케이션 개발 경험을 바탕으로, 스프링 부트(Spring Boot)와 Thymeleaf를 사용하여 웹 기반 ToDo 애플리케이션을 구축하는 두 번째 학습 단계를 안내합니다. 첫 번째 학습 단계에서 제공된 피드백을 적극적으로 반영하여, 사용자 정의 메소드는 사용 전에 정의하고, 코드 내 주요 요소(Java 기본 메소드, 스프링 어노테이션 등)에 대한 주석 설명을 추가하여 학습 효과를 높이고자 합니다.

**학습 목표:**

* 스프링 부트 프로젝트 설정 및 기본 구조 이해
* MVC(Model-View-Controller) 패턴 적용 (Controller, Service)
* Thymeleaf 템플릿 엔진을 사용한 동적 웹 페이지 생성
* 웹 요청 처리 (GET, POST) 및 폼 데이터 처리
* 기존 ToDo 애플리케이션 로직(Task 관리)을 웹 환경에 통합

**핵심 기술:**

* **스프링 부트 (Spring Boot):** Java 기반 웹 애플리케이션을 빠르고 쉽게 개발할 수 있도록 지원하는 프레임워크입니다. 자동 설정, 내장 서버 등의 기능으로 개발 생산성을 크게 향상시킵니다.
* **MVC 패턴 (Model-View-Controller):** 애플리케이션 로직을 모델(데이터), 뷰(화면), 컨트롤러(흐름 제어) 세 가지 역할로 분리하는 디자인 패턴입니다. 코드의 구조화 및 유지보수 용이성을 높입니다.
* **Thymeleaf:** 서버 측 Java 템플릿 엔진으로, HTML 구조를 유지하면서 동적인 데이터를 표현할 수 있게 해줍니다. 스프링 부트와의 통합이 용이합니다.

**사용자 피드백 반영:**

1. **메소드 정의 우선:** 가이드 내에서 사용될 사용자 정의 메소드는 호출되기 전에 반드시 먼저 정의하고 설명합니다.
2. **인라인 주석 추가:** Java 표준 라이브러리 메소드, 스프링 프레임워크의 주요 어노테이션 및 클래스, 특정 키워드가 처음 사용될 때 해당 코드 라인에 주석으로 간략한 설명을 추가합니다.
3. **개념-코드 연결 강화:** 코드 블록 전후에 관련 핵심 개념(예: 자동 설정, 의존성 주입, MVC)에 대한 설명을 명확하게 제공합니다.

이제 스프링 부트와 Thymeleaf를 사용하여 여러분의 ToDo 애플리케이션을 웹으로 확장해 봅시다.

## **2. 개발 환경 설정 및 프로젝트 생성**

스프링 부트 프로젝트를 시작하는 가장 쉬운 방법은 Spring Initializr (start.spring.io)를 사용하는 것입니다.1 이 웹 서비스를 통해 필요한 의존성을 선택하고 프로젝트 기본 구조를 생성할 수 있습니다.1

**단계:**

1. **Spring Initializr 접속:** 웹 브라우저에서 <https://start.spring.io> 로 이동합니다.1
2. **프로젝트 설정:**
   * **Project:** Gradle Project 선택 (이 가이드에서는 Gradle을 기준으로 설명합니다).1
   * **Language:** Java 선택.1
   * **Spring Boot:** 기본으로 제안되는 최신 안정화 버전(GA)을 선택합니다 (특정 버전 요구사항이 없다면 기본값을 사용).1
   * **Project Metadata:**
     + Group: 보통 회사의 도메인명을 역순으로 사용합니다 (예: com.example).1
     + Artifact: 프로젝트 이름입니다 (예: webtodoapp).1
     + Name: 프로젝트 이름 (Artifact와 동일하게 설정).1
     + Description: 프로젝트 설명 (예: Spring Boot Web ToDo Application).1
     + Package name: Group과 Artifact를 조합하여 생성됩니다 (예: com.example.webtodoapp). 이 패키지명 아래에 기본 애플리케이션 클래스가 생성됩니다.1
     + Packaging: Jar 선택 (스프링 부트는 내장 서버를 포함한 실행 가능한 Jar 파일 생성을 지원합니다).1
     + Java: 설치된 Java 버전 선택 (Spring Boot 3.x 버전은 Java 17 이상이 필요합니다).1
3. **의존성 (Dependencies) 추가:**
   * ADD DEPENDENCIES... 버튼 클릭.1
   * Spring Web: Spring MVC를 사용하여 웹 애플리케이션(RESTful 포함)을 구축하기 위한 기본 의존성입니다. 내장 Tomcat 서버를 포함합니다.1
   * Thymeleaf: HTML, XML 등을 처리하는 서버 측 Java 템플릿 엔진입니다. Spring MVC 뷰 레이어로 사용됩니다.1 spring-boot-starter-thymeleaf 의존성을 추가하면 spring-boot-starter-web도 자동으로 포함될 수 있습니다.6
   * Spring Boot DevTools: 개발 편의성을 위한 도구입니다. 코드 변경 시 자동 재시작, 정적 리소스 및 템플릿 캐시 비활성화, LiveReload 기능 등을 제공하여 개발 속도를 높여줍니다.2
4. **프로젝트 생성 및 다운로드:**
   * GENERATE 버튼 클릭.1
   * 설정된 내용으로 구성된 .zip 파일이 다운로드됩니다.1
5. **프로젝트 압축 해제 및 IDE로 가져오기:**
   * 다운로드한 .zip 파일의 압축을 해제합니다.1
   * 사용하는 IDE(IntelliJ IDEA, Eclipse with Buildship 등)에서 해당 프로젝트를 Gradle Project로 가져옵니다(Import). IDE가 build.gradle 파일을 분석하여 필요한 라이브러리들을 다운로드합니다.

**build.gradle 확인 (Gradle):**

프로젝트 루트 디렉토리에 있는 build.gradle 파일을 열어보면 선택한 의존성들이 추가된 것을 확인할 수 있습니다. 주요 의존성은 다음과 같습니다.3

Gradle

plugins {  
 id 'java' // Java 플러그인 적용  
 id 'org.springframework.boot' version '3.x.x' // 스프링 부트 플러그인 적용 (버전은 실제 생성된 버전에 맞게) [11, 12]  
 id 'io.spring.dependency-management' version '1.x.x' // 스프링 의존성 관리 플러그인 적용 (버전은 실제 생성된 버전에 맞게) [13]  
}  
  
group = 'com.example' // 프로젝트 그룹 ID  
version = '0.0.1-SNAPSHOT' // 프로젝트 버전  
  
java {  
 sourceCompatibility = '17' // 사용할 Java 버전 지정 (Spring Boot 3.x는 17 이상)  
}  
  
repositories {  
 mavenCentral() // 의존성을 다운로드할 저장소 지정 (메이븐 중앙 저장소)  
}  
  
dependencies {  
 // Spring Web 스타터: 웹 애플리케이션 개발 (MVC, 내장 Tomcat 등) [1, 3, 4]  
 implementation 'org.springframework.boot:spring-boot-starter-web'  
  
 // Thymeleaf 스타터: Thymeleaf 템플릿 엔진 통합 [5]  
 implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'  
  
 // Spring Boot DevTools: 개발 편의 도구 (자동 재시작, LiveReload 등) [7, 8, 9, 10]  
 // developmentOnly: 개발 시에만 사용하고, 최종 빌드 결과물(jar)에는 포함되지 않도록 함 [7, 10, 14]  
 developmentOnly 'org.springframework.boot:spring-boot-devtools'  
  
 // Spring Boot 테스트 스타터: 테스트 코드 작성 지원 (JUnit 5, Mockito 등) [1]  
 testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}  
  
tasks.named('test') {  
 useJUnitPlatform() // JUnit 5 플랫폼 사용 설정  
}

이제 기본적인 스프링 부트 웹 애플리케이션 개발 환경이 준비되었습니다.

## **3. 스프링 부트 프로젝트 구조 이해**

Spring Initializr로 생성된 프로젝트는 표준 Gradle/Maven 디렉토리 구조를 따릅니다.1 이 구조를 이해하는 것은 코드를 체계적으로 관리하는 데 중요합니다.

**주요 디렉토리 및 파일:**

* **src/main/java**: 애플리케이션의 Java 소스 코드가 위치하는 곳입니다.1
  + com/example/webtodoapp (설정한 패키지명): 기본 패키지 경로입니다.1
    - WebtodoappApplication.java: 스프링 부트 애플리케이션의 시작점(main 메소드 포함)입니다. @SpringBootApplication 어노테이션이 붙어 있습니다.1 이 클래스는 루트 패키지에 위치하는 것이 좋습니다.16 그래야 컴포넌트 스캔(Component Scan)이 하위 패키지들을 자동으로 검색할 수 있습니다.16
* **src/main/resources**: 리소스 파일(설정 파일, 정적 파일, 템플릿 등)이 위치하는 곳입니다.1
  + static: CSS, JavaScript, 이미지 등 정적 리소스 파일들을 위치시키는 디렉토리입니다.1 스프링 부트는 기본적으로 이 경로의 파일들을 웹 요청에 대해 제공합니다.2
  + templates: Thymeleaf와 같은 템플릿 엔진이 사용할 템플릿 파일(예: .html)을 위치시키는 디렉토리입니다.1
  + application.properties (또는 application.yml): 애플리케이션의 설정을 정의하는 파일입니다.1 서버 포트, 데이터베이스 연결 정보, 로깅 설정 등을 지정할 수 있습니다.
* **src/test/java**: 단위 테스트(Unit Test) 및 통합 테스트(Integration Test) 코드가 위치하는 곳입니다.1
* **build.gradle**: 프로젝트의 의존성 관리 및 빌드 설정을 담고 있는 파일입니다 (Gradle).17
* **gradlew / gradlew.bat**: Gradle Wrapper 실행 스크립트입니다. 로컬에 Gradle을 설치하지 않고도 프로젝트 빌드를 가능하게 합니다.
* **settings.gradle**: 프로젝트 이름 등 빌드 관련 설정을 정의합니다.

**권장 패키지 구조 (계층별 구조):**

애플리케이션 규모가 커짐에 따라 코드를 체계적으로 구성하는 것이 중요합니다. 일반적인 방법 중 하나는 기능적 계층(Layer)에 따라 패키지를 분리하는 것입니다.15

com  
 +- example  
 +- webtodoapp  
 +- WebtodoappApplication.java // 메인 애플리케이션 클래스 (루트 패키지)  
|  
 +- controller // 웹 요청 처리 (HTTP 요청 -> 응답 또는 뷰)  
| +- TodoController.java  
|  
 +- service // 비즈니스 로직 처리  
| +- TodoService.java (인터페이스)  
| +- TodoServiceImpl.java (구현체)  
|  
 +- model // 데이터 구조 정의 (Entity, DTO 등)  
| +- Task.java  
| +- DeadlineTask.java  
|  
 +- repository // 데이터 영속성 처리 (DB 연동 - 추후 단계)  
| // (이번 단계에서는 사용하지 않음)  
|  
 +- config // 애플리케이션 설정 관련 클래스 (선택 사항)  
|  
 +- dto // 데이터 전송 객체 (선택 사항)  
 +- TaskForm.java

* **controller**: 사용자의 HTTP 요청을 받아 적절한 서비스 메소드를 호출하고, 그 결과를 뷰(View)나 데이터(JSON 등) 형태로 반환하는 역할을 합니다.2 @Controller 또는 @RestController 어노테이션을 사용합니다.15
* **service**: 핵심 비즈니스 로직을 구현합니다.15 여러 Repository를 조합하거나 트랜잭션 관리 등을 수행할 수 있습니다. @Service 어노테이션을 사용합니다.15 인터페이스와 구현체로 분리하는 것이 일반적입니다.17
* **model / domain / entity**: 애플리케이션의 데이터 구조를 정의하는 클래스(Java Beans, POJOs)가 위치합니다.17 데이터베이스 테이블과 매핑되는 Entity 클래스나, 계층 간 데이터 전달을 위한 DTO(Data Transfer Object) 등이 포함될 수 있습니다.20
* **repository / dao**: 데이터베이스와의 상호작용을 담당합니다.17 데이터 조회, 저장, 수정, 삭제(CRUD) 로직을 구현합니다. (이번 단계에서는 메모리 기반으로 Service에서 처리하므로 생략합니다.) @Repository 어노테이션을 사용합니다.15
* **dto**: 계층 간(특히 Controller-View, Controller-Service) 데이터 전송을 위해 사용되는 객체입니다.20 필요한 데이터만 담아 전달함으로써 관심사를 분리하고 Entity 노출을 방지할 수 있습니다.
* **config**: 애플리케이션의 설정을 위한 Java 기반 설정 클래스들이 위치합니다 (예: Security 설정).15

이러한 구조는 코드의 역할 분담을 명확히 하고, 유지보수 및 테스트를 용이하게 만듭니다. 프로젝트 규모나 팀 컨벤션에 따라 기능별(Feature-based) 구조를 사용할 수도 있습니다.15

## **4. 핵심 컴포넌트 구현: Model, Service, Controller**

이제 웹 ToDo 애플리케이션의 핵심 구성 요소인 Model, Service, Controller를 구현합니다. 이전 콘솔 앱의 로직을 재사용하고 웹 환경에 맞게 조정합니다.

### **4.1 모델 (Model) 정의**

데이터를 표현하는 클래스입니다. 이전 단계에서 사용한 Task와 DeadlineTask 클래스를 model 패키지 아래에 생성하거나 복사합니다. 웹 환경에서는 각 할 일 항목을 식별하기 위한 고유 ID가 필요하므로, id 필드를 추가하는 것이 좋습니다.

**Task.java 수정:**

Java

package com.example.webtodoapp.model;  
  
import java.util.Objects;  
import java.util.concurrent.atomic.AtomicLong; // 원자적 ID 생성을 위해 import  
  
public class Task {  
 // 원자적 ID 생성을 위한 정적 카운터  
 // private static final AtomicLong counter = new AtomicLong(); // ID 자동 생성을 위해 추가  
  
 private long id; // 할 일 고유 ID 추가  
 private String description; // 할 일 설명  
 private boolean completed; // 완료 여부  
  
 // 기본 생성자 (필요한 경우)  
 public Task() {}  
  
 // 생성자: ID 자동 할당 제거 (Service에서 관리하도록 변경)  
 public Task(String description) {  
 // this.id = counter.incrementAndGet(); // ID 자동 생성 로직 제거  
 this.setDescription(description);  
 this.completed = false;  
 }  
  
 // ID 필드 추가에 따른 생성자 수정 (Service에서 ID를 받아 설정)  
 public Task(long id, String description) {  
 this.id = id;  
 this.setDescription(description);  
 this.completed = false;  
 }  
  
 // --- Getters ---  
 public long getId() { // ID Getter 추가  
 return id;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 // boolean 타입의 getter는 'is'로 시작하는 것이 관례  
 public boolean isCompleted() {  
 return completed;  
 }  
  
 // --- Setters ---  
 public void setId(long id) { // ID Setter 추가  
 this.id = id;  
 }  
  
 public void setDescription(String description) {  
 // 간단한 유효성 검사 예시  
 // String.trim(): 문자열 앞뒤 공백 제거  
 // String.isEmpty(): 문자열 길이가 0인지 확인  
 if (description!= null &&!description.trim().isEmpty()) {  
 this.description = description;  
 } else {  
 // 실제 웹 애플리케이션에서는 예외 처리나 더 정교한 로직 필요  
 System.err.println("오류: 할 일 설명은 비워둘 수 없습니다.");  
 // throw new IllegalArgumentException("할 일 설명은 비워둘 수 없습니다."); // 혹은 예외 발생  
 }  
 }  
  
 public void setCompleted(boolean completed) {  
 this.completed = completed;  
 }  
  
 // --- 객체 표현 및 비교 ---  
  
 @Override  
 public String toString() {  
 // 객체를 System.out.println() 등으로 출력할 때 보여질 문자열 형식 정의  
 // 삼항 연산자 (condition? value\_if\_true : value\_if\_false) 사용  
 return "Task [id=" + id + ", description=" + description + ", completed=" + (completed? "Y" : "N") + "]";  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 // 객체 동등성 비교 로직 정의 (주로 ID 기반으로 비교)  
 if (this == o) return true; // 동일 객체 참조 확인  
 // instanceof: 객체가 특정 클래스(또는 하위 클래스)의 인스턴스인지 확인  
 // getClass(): 객체의 런타임 클래스를 반환  
 if (o == null |  
| getClass()!= o.getClass()) return false; // null 또는 다른 클래스 타입 확인  
 Task task = (Task) o; // 비교 대상 객체를 Task 타입으로 형변환  
 // Objects.equals(): 두 객체가 null-safe하게 동일한지 비교  
 return id == task.id && completed == task.completed && Objects.equals(description, task.description);  
 }  
  
 @Override  
 public int hashCode() {  
 // 객체의 해시 코드 생성 로직 정의 (equals에서 사용된 필드 기반)  
 // Objects.hash(): 주어진 값들을 기반으로 해시 코드 생성  
 return Objects.hash(id, description, completed);  
 }  
}

**DeadlineTask.java 수정:**

Java

package com.example.webtodoapp.model;  
  
import java.util.Objects;  
  
// extends Task: Task 클래스의 모든 필드와 메소드를 상속받음  
public class DeadlineTask extends Task {  
 private String deadline; // 마감일 필드  
  
 // 기본 생성자 (필요한 경우)  
 public DeadlineTask() {  
 super(); // 부모 클래스(Task)의 기본 생성자 호출  
 }  
  
 // 생성자: 부모 클래스 생성자 호출 및 deadline 필드 초기화  
 public DeadlineTask(String description, String deadline) {  
 // super(description): 부모 클래스(Task)의 생성자 호출하여 description 등 초기화  
 super(description); // ID 자동 생성 로직 제거됨  
 this.deadline = deadline;  
 }  
  
 // ID 필드 추가에 따른 생성자 수정  
 public DeadlineTask(long id, String description, String deadline) {  
 super(id, description); // 부모 클래스의 ID 받는 생성자 호출  
 this.deadline = deadline;  
 }  
  
 // --- Getter ---  
 public String getDeadline() {  
 return deadline;  
 }  
  
 // --- Setter ---  
 public void setDeadline(String deadline) {  
 this.deadline = deadline;  
 }  
  
 // --- 객체 표현 및 비교 (Override) ---  
  
 @Override  
 public String toString() {  
 // super.toString(): 부모 클래스(Task)의 toString() 메소드 결과 활용  
 // 마감일 정보 추가  
 return super.toString().replace("]", ", deadline=" + deadline + "]");  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null |  
| getClass()!= o.getClass()) return false;  
 // super.equals(o): 부모 클래스(Task)의 필드(id, description, completed) 비교  
 if (!super.equals(o)) return false;  
 DeadlineTask that = (DeadlineTask) o;  
 // Objects.equals(): deadline 필드 비교 (null-safe)  
 return Objects.equals(deadline, that.deadline);  
 }  
  
 @Override  
 public int hashCode() {  
 // Objects.hash(): 부모 클래스 해시코드와 deadline 필드를 조합하여 해시코드 생성  
 return Objects.hash(super.hashCode(), deadline);  
 }  
}

**주요 변경 사항:**

* id 필드 (long 타입) 및 관련 getter/setter 추가.
* 생성자에서 ID 자동 생성 로직 제거 (Service 계층에서 ID 관리 예정).
* ID를 포함하는 생성자 추가.
* equals() 및 hashCode() 메소드에 id 필드 반영.
* JavaDoc 스타일 주석 및 인라인 주석 추가 (예: extends, super(), Objects.equals(), Objects.hash(), instanceof, trim(), isEmpty()).

### **4.2 서비스 (Service) 계층 구현**

비즈니스 로직을 담당하는 부분입니다.18 할 일 목록 관리(추가, 조회, 완료 처리 등) 로직을 구현합니다. 이번 단계에서는 데이터베이스 대신 메모리 내 List를 사용하여 데이터를 관리합니다.

**1. TodoService 인터페이스 정의 (service 패키지):**

서비스 계층의 기능을 정의하는 인터페이스를 먼저 만듭니다. 이는 구현체와의 결합도를 낮추고 테스트 용이성을 높입니다.

Java

package com.example.webtodoapp.service;  
  
import com.example.webtodoapp.model.Task;  
import java.util.List;  
import java.util.Optional; // 값이 없을 수도 있음을 명시적으로 나타내는 컨테이너 객체  
  
public interface TodoService {  
  
 /\*\*  
 \* 새로운 할 일을 추가합니다.  
 \* @param task 추가할 Task 객체 (ID는 서비스 내부에서 할당될 수 있음)  
 \* @return 추가된 Task 객체 (ID 포함)  
 \*/  
 Task addTask(Task task);  
  
 /\*\*  
 \* 모든 할 일 목록을 조회합니다.  
 \* @return Task 객체 리스트  
 \*/  
 List<Task> findAllTasks();  
  
 /\*\*  
 \* 특정 ID의 할 일을 조회합니다.  
 \* @param id 조회할 할 일의 ID  
 \* @return 해당 ID의 Task 객체를 담은 Optional (없으면 Optional.empty())  
 \*/  
 Optional<Task> findTaskById(long id);  
  
 /\*\*  
 \* 특정 ID의 할 일을 완료 상태로 변경합니다.  
 \* @param id 완료 처리할 할 일의 ID  
 \* @return 완료 처리된 Task 객체를 담은 Optional (해당 ID가 없으면 Optional.empty())  
 \*/  
 Optional<Task> markTaskComplete(long id);  
  
 /\*\*  
 \* 특정 ID의 할 일을 삭제합니다. (추후 확장 기능)  
 \* @param id 삭제할 할 일의 ID  
 \* @return 삭제 성공 여부  
 \*/  
 boolean deleteTask(long id); // 예시: 삭제 기능 인터페이스  
}

**2. TodoServiceImpl 구현체 작성 (service 패키지):**

TodoService 인터페이스를 구현하는 클래스를 작성합니다.

Java

package com.example.webtodoapp.service;  
  
import com.example.webtodoapp.model.Task;  
import org.springframework.stereotype.Service; // 이 클래스가 서비스 계층의 컴포넌트임을 나타냄 [22, 24, 25, 26, 27]  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
import java.util.concurrent.ConcurrentHashMap; // 동시성 이슈를 고려한 Map 구현체  
import java.util.concurrent.atomic.AtomicLong; // 원자적 ID 생성을 위해 사용  
import java.util.Map;  
import java.util.stream.Collectors; // Stream API의 Collector 사용  
  
// @Service: 스프링 컨테이너가 이 클래스의 인스턴스를 빈(Bean)으로 관리하도록 지정합니다. [22, 24, 25, 26, 27]  
// 이는 컴포넌트 스캔(Component Scan)의 대상이 되며, 다른 컴포넌트에서 의존성 주입(Dependency Injection)을 통해 사용할 수 있게 됩니다.  
@Service  
public class TodoServiceImpl implements TodoService {  
  
 // 메모리 기반 데이터 저장소 (동시성 고려)  
 // ConcurrentHashMap: 여러 스레드에서 동시에 접근해도 안전하도록 설계된 Map  
 private final Map<Long, Task> taskStore = new ConcurrentHashMap<>();  
 // 원자적 ID 생성을 위한 카운터  
 private final AtomicLong idCounter = new AtomicLong();  
  
 /\*\*  
 \* 새 할 일 추가 (ID 할당 및 저장)  
 \*/  
 @Override  
 public Task addTask(Task task) {  
 // idCounter.incrementAndGet(): 현재 값을 1 증가시키고 그 결과를 반환 (원자적 연산)  
 long newId = idCounter.incrementAndGet();  
 task.setId(newId); // 새로운 ID 설정  
 // taskStore.put(key, value): Map에 키-값 쌍을 저장  
 taskStore.put(newId, task);  
 return task;  
 }  
  
 /\*\*  
 \* 모든 할 일 목록 조회  
 \*/  
 @Override  
 public List<Task> findAllTasks() {  
 // taskStore.values(): Map의 모든 값(Task 객체)들을 Collection 형태로 반환  
 // new ArrayList<>(...): Collection을 사용하여 새로운 ArrayList 생성 (수정 가능한 리스트 반환 목적)  
 // 또는 Stream API 사용:  
 // return taskStore.values().stream().collect(Collectors.toList());  
 return new ArrayList<>(taskStore.values());  
 }  
  
 /\*\*  
 \* ID로 할 일 조회  
 \*/  
 @Override  
 public Optional<Task> findTaskById(long id) {  
 // taskStore.get(key): Map에서 주어진 키에 해당하는 값을 반환 (없으면 null)  
 // Optional.ofNullable(value): 주어진 값이 null이 아니면 값을 포함하는 Optional을, null이면 비어있는 Optional을 반환  
 return Optional.ofNullable(taskStore.get(id));  
 }  
  
 /\*\*  
 \* 할 일 완료 처리  
 \*/  
 @Override  
 public Optional<Task> markTaskComplete(long id) {  
 // ID로 Task 조회  
 Optional<Task> taskOptional = findTaskById(id);  
  
 // taskOptional.ifPresent(lambda): Optional 객체가 값을 포함하고 있을 경우에만 주어진 람다식 실행  
 taskOptional.ifPresent(task -> {  
 task.setCompleted(true); // 완료 상태로 변경  
 taskStore.put(id, task); // 변경된 Task 객체로 Map 업데이트  
 });  
  
 return taskOptional; // 변경되었거나 찾지 못한 Optional 반환  
 }  
  
 /\*\*  
 \* 할 일 삭제 (구현 예시)  
 \*/  
 @Override  
 public boolean deleteTask(long id) {  
 // taskStore.remove(key): Map에서 주어진 키에 해당하는 항목을 삭제하고, 삭제된 값을 반환 (없으면 null)  
 // 삭제 성공 여부 반환 (삭제된 객체가 null이 아니면 true)  
 return taskStore.remove(id)!= null;  
 }  
}

**핵심 개념:**

* **@Service**: 이 클래스가 비즈니스 로직을 처리하는 서비스 컴포넌트임을 스프링에게 알립니다.22 스프링 컨테이너는 이 클래스의 인스턴스(빈)를 생성하고 관리합니다.
* **의존성 주입 (Dependency Injection - DI):** Controller가 Service를 직접 생성하는 대신, 스프링 컨테이너가 생성된 Service 빈(Bean)을 Controller에 주입해줍니다.34 이는 클래스 간의 결합도를 낮추고 유연성과 테스트 용이성을 높입니다. (Controller 구현 시 자세히 설명)
* **In-Memory Store:** ConcurrentHashMap과 AtomicLong을 사용하여 간단한 메모리 기반 데이터 저장소를 구현했습니다. 여러 사용자가 동시에 접근하는 웹 환경을 고려하여 동시성(Concurrency)에 안전한 자료구조를 사용했습니다. 실제 애플리케이션에서는 데이터베이스를 사용하는 것이 일반적입니다.
* **Optional<T>**: 값이 존재할 수도, 존재하지 않을 수도 있는 경우를 명확하게 표현하기 위해 사용됩니다. null을 직접 반환하는 것보다 NullPointerException(NPE)을 방지하고 코드의 가독성을 높이는 데 도움이 됩니다.

### **4.3 컨트롤러 (Controller) 계층 구현**

사용자의 웹 요청(HTTP Request)을 받아 처리하고, 적절한 서비스 로직을 호출한 뒤, 그 결과를 사용자에게 보여줄 뷰(View) 또는 데이터를 반환하는 역할을 합니다.2

**TodoController.java 생성 (controller 패키지):**

Java

package com.example.webtodoapp.controller;  
  
import com.example.webtodoapp.service.TodoService; // 사용할 서비스 인터페이스 import  
import org.springframework.beans.factory.annotation.Autowired; // 의존성 주입을 위해 사용 (생성자 주입 시 생략 가능) [34, 35, 37, 38]  
import org.springframework.stereotype.Controller; // 이 클래스가 웹 컨트롤러임을 나타냄 [2, 21, 22, 23, 41, 42, 43, 44, 45, 46]  
import org.springframework.web.bind.annotation.GetMapping; // HTTP GET 요청을 특정 메소드에 매핑 [3, 47, 48, 49, 50]  
  
// @Controller: 이 클래스가 Spring MVC 컨트롤러임을 나타냅니다. [2, 21, 22, 23, 41, 42, 43, 44, 45, 46]  
// 스프링은 이 어노테이션이 붙은 클래스에서 @RequestMapping (또는 @GetMapping, @PostMapping 등)을 찾아 요청을 처리할 메소드를 결정합니다.  
// 주로 HTML 뷰를 반환하는 웹 애플리케이션에 사용됩니다.  
// REST API처럼 데이터(JSON/XML) 자체를 반환하려면 @RestController를 사용합니다. @RestController는 @Controller와 @ResponseBody를 합친 것입니다. [23, 41, 42, 43, 44, 45, 46]  
@Controller  
public class TodoController {  
  
 // 서비스 계층 의존성 선언 (final 키워드로 불변성 확보 권장)  
 private final TodoService todoService;  
  
 // --- 의존성 주입 (Dependency Injection) ---  
 // @Autowired: 스프링 컨테이너에게 해당 타입(TodoService)의 빈(Bean)을 찾아 자동으로 주입하도록 지시합니다. [34, 35, 37, 38]  
 // 생성자가 하나만 있는 경우, 스프링 4.3 버전부터는 @Autowired를 생략해도 자동으로 주입됩니다 (권장 방식). [34, 37]  
 // 이를 '생성자 주입(Constructor Injection)'이라고 하며, 필드 주입(Field Injection)이나 세터 주입(Setter Injection)보다 권장됩니다. [34, 36, 38]  
 // 이유: 의존성 불변성 확보, 순환 참조 방지, 테스트 용이성 등. [34, 38]  
 // @Autowired // 생성자가 하나이므로 생략 가능 [34, 37]  
 public TodoController(TodoService todoService) {  
 this.todoService = todoService;  
 }  
  
 // --- 요청 핸들러 메소드 ---  
 // @GetMapping("/"): HTTP GET 요청 중 루트 경로("/")를 이 메소드와 매핑합니다. [3, 48, 49, 50]  
 // 사용자가 웹사이트의 기본 주소로 접속했을 때 이 메소드가 호출됩니다.  
 @GetMapping("/")  
 public String showHomePage() {  
 // "index" 라는 이름의 뷰(View)를 반환하도록 지정합니다. [2, 39, 42, 43, 44, 45, 46]  
 // 스프링 부트는 Thymeleaf 설정에 따라 src/main/resources/templates/ 디렉토리에서  
 // "index.html" 파일을 찾아 렌더링합니다. [1, 51]  
 return "index"; // 논리적 뷰 이름 반환  
 }  
  
 // 여기에 할 일 목록 조회, 추가, 완료 처리 등을 위한 핸들러 메소드들을 추가할 예정입니다.  
}

**핵심 개념:**

* **@Controller**: 클래스가 웹 요청을 처리하는 컨트롤러임을 명시합니다.2 주로 뷰 이름을 반환하여 HTML 페이지를 렌더링하는 데 사용됩니다. 만약 JSON이나 XML 같은 데이터를 직접 반환하는 REST API를 만든다면 @RestController를 사용합니다.23 @RestController는 @Controller와 @ResponseBody(메소드 반환 값을 HTTP 응답 본문에 직접 쓰도록 지시)를 합친 편리한 어노테이션입니다.23
* **생성자 주입 (Constructor Injection)**: @Autowired를 사용하여 (또는 생략하여) 스프링 컨테이너가 관리하는 TodoService 빈을 TodoController 생성 시 주입받습니다.34 이는 의존성을 명확하게 하고 테스트에 유리한 방식입니다.34
* **@GetMapping**: 특정 URL 경로로 들어오는 HTTP GET 요청을 처리할 메소드를 지정합니다.3 /는 웹 애플리케이션의 루트 경로를 의미합니다.
* **뷰 이름 반환**: 컨트롤러 메소드가 문자열("index")을 반환하면, 스프링 부트는 설정된 뷰 리졸버(View Resolver, Thymeleaf의 경우 자동 설정됨)를 통해 해당 이름의 템플릿 파일을 찾아(templates/index.html) 렌더링하여 사용자에게 보여줍니다.2

이제 애플리케이션의 기본 뼈대가 준비되었습니다. 다음 단계에서는 컨트롤러에서 서비스 계층을 호출하여 데이터를 가져오고, Thymeleaf를 사용하여 웹 페이지에 동적으로 표시하는 방법을 알아봅니다.

## **5. 데이터 표시: Thymeleaf를 이용한 동적 웹 페이지**

컨트롤러에서 가져온 할 일 목록 데이터를 Thymeleaf 템플릿 엔진을 사용하여 HTML 페이지에 동적으로 표시해 보겠습니다.52

### **5.1 컨트롤러 수정: 데이터 모델에 추가**

TodoController에 할 일 목록을 조회하여 Model 객체에 담아 뷰로 전달하는 로직을 추가합니다.48

Java

package com.example.webtodoapp.controller;  
  
import com.example.webtodoapp.model.Task; // Task 모델 import  
import com.example.webtodoapp.service.TodoService;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model; // 뷰에 데이터를 전달하기 위한 Model 인터페이스 import [48]  
import org.springframework.web.bind.annotation.GetMapping;  
  
import java.util.List; // List 인터페이스 import  
  
@Controller  
public class TodoController {  
  
 private final TodoService todoService;  
  
 public TodoController(TodoService todoService) {  
 this.todoService = todoService;  
 }  
  
 // 루트 경로("/") GET 요청 처리 메소드 수정  
 @GetMapping("/")  
 public String showTasks(Model model) { // Model 객체를 파라미터로 받음 [48]  
 // 1. 서비스 계층을 통해 모든 할 일 목록 조회  
 List<Task> tasks = todoService.findAllTasks();  
  
 // 2. Model 객체에 조회된 할 일 목록 추가  
 // model.addAttribute(attributeName, attributeValue): [48]  
 // - attributeName: 뷰(템플릿)에서 사용할 데이터의 이름 (문자열)  
 // - attributeValue: 뷰로 전달할 실제 데이터 객체 (여기서는 List<Task>)  
 // 템플릿에서는 "tasks"라는 이름으로 List<Task>에 접근할 수 있게 됩니다.  
 model.addAttribute("tasks", tasks);  
  
 // 3. 뷰 이름("index") 반환 -> templates/index.html 렌더링  
 return "index";  
 }  
}

**핵심 개념:**

* **org.springframework.ui.Model**: 컨트롤러에서 뷰(템플릿)로 데이터를 전달하는 역할을 하는 인터페이스입니다.48 addAttribute(String name, Object value) 메소드를 사용하여 이름-값 쌍으로 데이터를 추가하면, 템플릿 엔진(Thymeleaf)에서 해당 이름으로 데이터에 접근할 수 있습니다. 스프링 MVC가 메소드 호출 시 Model 객체를 자동으로 생성하여 파라미터로 넘겨줍니다.

### **5.2 Thymeleaf 템플릿 생성 (index.html)**

src/main/resources/templates 디렉토리 아래에 index.html 파일을 생성하고 다음 내용을 작성합니다.1

HTML

<!DOCTYPE **html**>  
<html xmlns:th="http://www.thymeleaf.org"> <head>  
 <meta charset="UTF-8">  
 <title>Web ToDo App</title>  
 <style>  
 body { font-family: sans-serif; }  
 table { border-collapse: collapse; width: 100%; }  
 th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }  
 th { background-color: #f2f2f2; }  
 .completed { text-decoration: line-through; color: grey; }  
 </style>  
</head>  
<body>  
  
 <h1>My ToDo List</h1>  
  
 <table>  
 <thead>  
 <tr>  
 <th>ID</th>  
 <th>Description</th>  
 <th>Deadline</th>  
 <th>Status</th>  
 <th>Actions</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each="task : ${tasks}">  
 <td th:text="${task.id}">1</td>  
 <td th:text="${task.description}" th:classappend="${task.completed}? 'completed' : ''">Task Description</td>  
 <td>  
 <span th:if="${#objects.isInstance(task, 'com.example.webtodoapp.model.DeadlineTask')}"  
 th:text="${task.deadline}">YYYY-MM-DD</span>  
 <span th:unless="${#objects.isInstance(task, 'com.example.webtodoapp.model.DeadlineTask')}">-</span>  
 </td>  
 <td th:text="${task.completed}? 'Completed' : 'Pending'">Pending</td>  
 <td>  
 </td>  
 </tr>  
 <tr th:if="${tasks.isEmpty()}">  
 <td colspan="5">No tasks yet!</td>  
 </tr>  
 </tbody>  
 </table>  
  
 <h2>Add New Task</h2>  
 </body>  
</html>

**핵심 Thymeleaf 속성:**

* **xmlns:th="http://www.thymeleaf.org"**: Thymeleaf 사용을 위한 네임스페이스 선언입니다.48
* **th:each="task : ${tasks}"**: Model에 "tasks"라는 이름으로 추가된 리스트(List<Task>)를 순회합니다.52 각 반복에서 리스트의 요소는 task 변수에 할당됩니다.
* **th:text="${...}"**: 태그의 텍스트 내용을 ${...} 안의 표현식 결과로 설정합니다.52 ${task.id}는 task 객체의 getId() 메소드를 호출합니다.
* **th:if="${condition}" / th:unless="${condition}"**: 조건(condition)의 참/거짓 여부에 따라 해당 HTML 요소를 렌더링할지 결정합니다.52
* **th:classappend="${condition}? 'class-name' : ''"**: 조건(condition)이 참이면 지정된 CSS 클래스(class-name)를 기존 클래스에 추가합니다.53
* **${tasks.isEmpty()}**: Model에서 전달된 tasks 리스트가 비어있는지 확인합니다 (List.isEmpty() 호출).
* **#objects.isInstance(obj, 'className')**: Thymeleaf 유틸리티 객체(objects)를 사용하여 obj가 특정 클래스(className 문자열로 지정)의 인스턴스인지 확인합니다.52 instanceof 연산자와 유사합니다.

### **5.3 실행 및 확인**

1. WebtodoappApplication.java 파일을 찾아 마우스 오른쪽 버튼을 클릭하고 Run 'WebtodoappApplication.main()' (또는 유사한 메뉴)를 선택하여 스프링 부트 애플리케이션을 실행합니다.2
2. 콘솔 로그에 Tomcat이 8080 포트(기본값)에서 시작되었다는 메시지가 나타나는지 확인합니다.1
3. 웹 브라우저를 열고 http://localhost:8080 주소로 접속합니다.
4. 아직 할 일을 추가하지 않았으므로 "No tasks yet!" 메시지가 표시된 빈 테이블이 나타나야 합니다.

이제 컨트롤러에서 서비스 계층을 통해 데이터를 가져와 Thymeleaf 템플릿을 통해 동적으로 웹 페이지에 표시하는 기본 흐름이 완성되었습니다. 다음 단계에서는 사용자가 웹 페이지에서 직접 할 일을 추가할 수 있도록 폼(Form) 처리 기능을 구현합니다.

## **6. 데이터 추가: Thymeleaf 폼 처리**

사용자가 웹 인터페이스를 통해 새로운 할 일을 입력하고 제출할 수 있도록 HTML 폼과 이를 처리하는 컨트롤러 로직을 구현합니다.48

### **6.1 할 일 추가 폼 생성 (index.html)**

index.html 파일 하단에 새로운 할 일을 입력받는 HTML <form> 태그를 추가합니다. Thymeleaf의 폼 관련 속성들을 사용하여 폼 데이터와 모델 객체를 연결합니다.48

HTML

<form th:action="@{/addTask}" th:object="${newTask}" method="post"> <div>  
 <label for="desc">Description:</label>  
 <input type="text" id="desc" th:field="\*{description}" placeholder="Enter task description" required />  
 </div>  
 <div>  
 <label for="deadline">Deadline (Optional, YYYY-MM-DD):</label>  
 <input type="text" id="deadline" th:field="\*{deadline}" placeholder="YYYY-MM-DD" />  
 </div>  
 <div>  
 <button type="submit">Add Task</button>  
 </div>  
</form>

**핵심 Thymeleaf 폼 속성:**

* **th:action="@{/url}"**: 폼이 제출될 때 요청을 보낼 URL을 지정합니다.48 @ 심볼은 컨텍스트 경로를 포함한 URL을 생성합니다.
* **th:object="${objectName}"**: 폼 데이터를 담을 모델 객체를 지정합니다.48 컨트롤러에서 model.addAttribute("objectName",...) 형태로 전달된 객체여야 합니다.
* **th:field="\*{fieldName}"**: <input>, <select>, <textarea> 등의 폼 필드를 th:object로 지정된 객체의 특정 필드(fieldName)와 양방향으로 바인딩합니다.48 해당 필드의 getter/setter가 필요합니다. id, name, value 속성을 자동으로 생성/관리해줍니다. \*{...} 구문은 선택된 객체(selection)에 대한 표현식으로, th:object 내에서 필드에 접근할 때 사용합니다.

### **6.2 폼 바인딩 객체 준비 (Controller 및 DTO)**

폼 데이터를 받기 위한 객체를 준비해야 합니다. 기존 Task 모델을 직접 사용할 수도 있지만, 폼 전용 데이터를 처리하거나 유효성 검사 등을 용이하게 하기 위해 별도의 DTO(Data Transfer Object)를 사용하는 것이 좋습니다.20

**1. TaskForm DTO 생성 (dto 패키지):**

Java

package com.example.webtodoapp.dto;  
  
// 폼 데이터를 전달하기 위한 간단한 DTO 클래스  
public class TaskForm {  
 private String description;  
 private String deadline; // 폼에서는 문자열로 받고, 서비스에서 필요시 파싱  
  
 // --- Getters and Setters ---  
 // Thymeleaf의 th:field가 객체의 필드 값을 읽고 쓰기 위해 필요합니다.  
 public String getDescription() {  
 return description;  
 }  
  
 public void setDescription(String description) {  
 this.description = description;  
 }  
  
 public String getDeadline() {  
 return deadline;  
 }  
  
 public void setDeadline(String deadline) {  
 this.deadline = deadline;  
 }  
}

**2. 컨트롤러 수정: 폼 객체 모델에 추가:**

할 일 목록을 보여주는 showTasks 메소드에서 빈 TaskForm 객체를 생성하여 모델에 추가합니다. 이 객체가 index.html의 th:object="${newTask}"와 연결됩니다.

Java

// TodoController.java 상단에 import 추가  
import com.example.webtodoapp.dto.TaskForm;  
  
// TodoController.java 내 showTasks 메소드 수정  
@GetMapping("/")  
public String showTasks(Model model) {  
 List<Task> tasks = todoService.findAllTasks();  
 model.addAttribute("tasks", tasks);  
  
 // --- 폼 바인딩을 위한 빈 TaskForm 객체 추가 ---  
 // "newTask"라는 이름으로 모델에 추가 -> index.html의 th:object="${newTask}"와 연결됨  
 model.addAttribute("newTask", new TaskForm()); // [48]  
  
 return "index";  
}

**핵심 개념:**

* **Form Backing Object:** HTML 폼의 필드들과 매핑되는 서버 측 객체입니다.48 Thymeleaf의 th:object와 th:field는 이 객체를 사용하여 폼 데이터를 채우거나(GET 요청 시) 폼에서 제출된 데이터를 받아옵니다(POST 요청 시). DTO를 사용하는 것이 일반적입니다.

### **6.3 폼 제출 처리 (@PostMapping)**

사용자가 폼을 작성하고 "Add Task" 버튼을 누르면, th:action에 지정된 /addTask URL로 POST 요청이 전송됩니다. 이 요청을 처리할 컨트롤러 메소드를 구현합니다.21

**사용자 피드백 반영:** @PostMapping, @ModelAttribute 어노테이션과 관련된 메소드를 먼저 정의하고 설명합니다.

Java

// TodoController.java 상단에 import 추가  
import org.springframework.web.bind.annotation.PostMapping; // HTTP POST 요청 매핑 [21, 47, 48, 49, 50]  
import org.springframework.web.bind.annotation.ModelAttribute; // 폼 데이터를 객체에 바인딩 [21, 48]  
import com.example.webtodoapp.model.DeadlineTask; // DeadlineTask 모델 import  
  
// TodoController.java 내부에 메소드 추가  
/\*\*  
 \* 할 일 추가 폼 제출을 처리하는 메소드.  
 \* index.html의 <form th:action="@{/addTask}" method="post"> 에서 호출됩니다.  
 \*  
 \* @param taskForm 폼 데이터가 바인딩된 TaskForm 객체. @ModelAttribute("newTask")를 통해 모델의 "newTask" 객체와 연결됩니다.  
 \* @return 할 일 추가 후 리다이렉트할 URL 경로 (루트 경로 "/")  
 \*/  
// @PostMapping("/addTask"): "/addTask" 경로로 들어오는 HTTP POST 요청을 이 메소드에 매핑합니다. [21, 47, 48, 49, 50]  
@PostMapping("/addTask")  
public String addTask(@ModelAttribute("newTask") TaskForm taskForm) { // [21, 48]  
 // @ModelAttribute("newTask"): HTTP 요청 파라미터(폼 데이터)를 TaskForm 타입의 객체(taskForm)에 자동으로 바인딩합니다.  
 // 이때 사용되는 모델 어트리뷰트 이름("newTask")은 GET 요청 시 모델에 추가했던 이름과 일치해야 합니다.  
 // 스프링은 요청 파라미터 이름과 객체의 필드 이름(setter 기준)을 비교하여 값을 채워줍니다.  
  
 // 1. TaskForm DTO를 실제 Task/DeadlineTask 엔티티로 변환  
 Task taskToAdd;  
 // String.trim(): 문자열 앞뒤 공백 제거  
 // String.isEmpty(): 문자열 길이가 0인지 확인 (null 체크도 필요)  
 if (taskForm.getDeadline()!= null &&!taskForm.getDeadline().trim().isEmpty()) {  
 // 마감일이 입력되었으면 DeadlineTask 생성  
 // 실제 애플리케이션에서는 날짜 형식 검증 등이 필요  
 taskToAdd = new DeadlineTask(taskForm.getDescription(), taskForm.getDeadline());  
 } else {  
 // 마감일이 없으면 일반 Task 생성  
 taskToAdd = new Task(taskForm.getDescription());  
 }  
  
 // 2. 서비스 계층을 통해 할 일 추가 로직 호출  
 todoService.addTask(taskToAdd);  
  
 // 3. 작업 완료 후 할 일 목록 페이지로 리다이렉트  
 // "redirect:/": 클라이언트(브라우저)에게 지정된 URL("/")로 다시 요청을 보내도록 지시합니다.  
 // 이는 POST 요청 처리 후 새로고침 시 폼이 다시 제출되는 것을 방지하는 일반적인 패턴(Post-Redirect-Get, PRG)입니다.  
 return "redirect:/";  
}

**핵심 개념:**

* **@PostMapping("/url")**: 특정 URL로 들어오는 HTTP POST 요청을 처리하는 메소드를 지정합니다.21 HTML 폼의 method="post"와 th:action 속성과 연관됩니다.
* **@ModelAttribute("attributeName")**: 메소드 파라미터에 사용될 때, HTTP 요청 파라미터들을 해당 파라미터 객체의 필드에 자동으로 바인딩(Data Binding)하는 역할을 합니다.21 모델 어트리뷰트 이름(attributeName)은 폼을 보여줄 때 사용한 이름과 일치해야 합니다. 스프링이 내부적으로 TaskForm 객체를 생성하고, 요청 파라미터 중 description 값을 taskForm.setDescription()에, deadline 값을 taskForm.setDeadline()에 전달하여 객체를 채웁니다.
* **리다이렉트 (redirect:/url)**: 컨트롤러 메소드가 redirect: 접두사와 함께 URL 문자열을 반환하면, 스프링 MVC는 HTTP 응답 상태 코드를 302 Found로 설정하고 Location 헤더에 지정된 URL을 담아 클라이언트에게 보냅니다. 브라우저는 이 응답을 받고 해당 URL로 새로운 GET 요청을 자동으로 보냅니다. 이는 폼 중복 제출을 방지하는 데 효과적입니다.

### **6.4 실행 및 테스트**

1. 애플리케이션을 재시작합니다 (DevTools 사용 시 자동으로 재시작될 수 있음).7
2. http://localhost:8080으로 접속합니다.
3. 페이지 하단에 새로 추가된 "Add New Task" 폼이 보이는지 확인합니다.
4. Description 필드에 할 일 내용을 입력합니다.
5. (선택) Deadline 필드에 날짜(예: 2024-12-31)를 입력합니다.
6. "Add Task" 버튼을 클릭합니다.
7. 페이지가 새로고침되면서(리다이렉트) 할 일 목록 테이블에 방금 추가한 항목이 나타나는지 확인합니다. 마감일을 입력했다면 DeadlineTask로, 입력하지 않았다면 일반 Task로 추가되어야 합니다.

이제 사용자는 웹 페이지를 통해 새로운 할 일을 등록할 수 있습니다. 다음 단계에서는 기존 할 일의 상태를 변경하는 기능(완료 처리)을 구현합니다.

## **7. 데이터 수정: 할 일 완료 처리**

목록에 있는 할 일을 "완료" 상태로 변경하는 기능을 추가합니다. 각 할 일 항목 옆에 버튼을 추가하고, 이 버튼을 클릭하면 해당 할 일의 상태가 업데이트되도록 구현합니다.

### **7.1 뷰에 완료 처리 컨트롤 추가 (index.html)**

index.html의 할 일 목록 테이블 각 행(<tr>)의 마지막 <td> 안에 "Mark Complete" 버튼을 추가합니다. 이 버튼은 특정 할 일의 ID를 포함하는 URL로 POST 요청을 보내도록 설정합니다.

HTML

<td>  
 <form th:if="${!task.completed}"  
 th:action="@{/tasks/complete/{id}(id=${task.id})}" method="post">  
 <button type="submit">Mark Complete</button>  
 </form>  
 <span th:if="${task.completed}">Completed</span>  
</td>

**핵심 개념:**

* **경로 변수 (Path Variable) in Thymeleaf URL:** th:action="@{/path/{varName}(varName=${value})}" 구문을 사용하여 URL 경로의 일부를 동적으로 설정할 수 있습니다.54 {id} 부분이 플레이스홀더 역할을 하며, (id=${task.id}) 부분이 이 플레이스홀더를 실제 task 객체의 id 값으로 대체합니다. 이렇게 생성된 URL은 예를 들어 /tasks/complete/1, /tasks/complete/2 와 같은 형태가 됩니다.
* **데이터 변경을 위한 POST:** 상태를 변경하는 작업은 HTTP GET 보다는 POST (또는 PUT, DELETE) 메소드를 사용하는 것이 REST 원칙에 더 부합합니다. 여기서는 간단하게 POST를 사용합니다.

### **7.2 컨트롤러 핸들러 생성 (@PostMapping 및 @PathVariable)**

/tasks/complete/{id} 형태의 URL로 들어오는 POST 요청을 처리할 컨트롤러 메소드를 추가합니다. URL 경로에서 ID 값을 추출하기 위해 @PathVariable 어노테이션을 사용합니다.47

**사용자 피드백 반영:** @PostMapping, @PathVariable 어노테이션과 관련된 메소드를 먼저 정의하고 설명합니다.

Java

// TodoController.java 상단에 import 추가  
import org.springframework.web.bind.annotation.PathVariable; // URL 경로에서 변수 값을 추출 [47, 48, 49, 50]  
  
// TodoController.java 내부에 메소드 추가  
/\*\*  
 \* 특정 ID의 할 일을 완료 상태로 변경하는 요청을 처리합니다.  
 \* index.html의 'Mark Complete' 버튼 폼에서 호출됩니다.  
 \*  
 \* @param id URL 경로에서 추출된 완료할 할 일의 ID (@PathVariable("id")를 통해 바인딩됨)  
 \* @return 작업 완료 후 리다이렉트할 URL 경로 (루트 경로 "/")  
 \*/  
// @PostMapping("/tasks/complete/{id}"): "/tasks/complete/" 다음에 오는 경로 부분을 ID로 인식하여 POST 요청을 매핑합니다. [21, 47, 48, 49, 50]  
// {id}는 경로 변수(Path Variable) 플레이스홀더입니다. [47, 48, 49, 50]  
@PostMapping("/tasks/complete/{id}")  
public String markTaskComplete(@PathVariable("id") long id) { // [47, 48, 49, 50]  
 // @PathVariable("id"): URL 경로의 {id} 부분에 해당하는 값을 메소드 파라미터 'id'(long 타입)에 바인딩합니다.  
 // 만약 파라미터 이름과 경로 변수 이름이 같다면 ("id"로 동일) @PathVariable 뒤의 괄호와 이름 생략 가능 (@PathVariable long id)  
  
 // 1. 서비스 계층의 메소드를 호출하여 해당 ID의 할 일을 완료 처리  
 todoService.markTaskComplete(id);  
 // Optional<Task> 반환 값을 사용하지 않았지만, 실제로는 찾지 못한 경우 등을 처리할 수 있음  
  
 // 2. 작업 완료 후 할 일 목록 페이지로 리다이렉트  
 return "redirect:/";  
}

**핵심 개념:**

* **@PathVariable**: @RequestMapping (또는 @GetMapping, @PostMapping 등)의 URL 경로 템플릿({...}으로 표시된 부분)에서 값을 추출하여 메소드 파라미터에 바인딩하는 어노테이션입니다.47 RESTful API에서 리소스를 식별하는 데 자주 사용됩니다.

### **7.3 실행 및 테스트**

1. 애플리케이션을 재시작합니다.
2. http://localhost:8080으로 접속하여 이전에 추가했던 할 일들이나 새로운 할 일을 추가합니다.
3. "Pending" 상태인 할 일 옆의 "Mark Complete" 버튼을 클릭합니다.
4. 페이지가 리다이렉트되면서 해당 할 일의 상태가 "Completed"로 변경되고, 설명 텍스트에 취소선이 그어지며(CSS 적용 시), "Mark Complete" 버튼 대신 "Completed" 텍스트가 표시되는지 확인합니다.

이제 기본적인 할 일 조회, 추가, 완료 기능을 갖춘 웹 기반 ToDo 애플리케이션이 완성되었습니다.

## **8. 주요 스프링 어노테이션 요약**

이번 가이드에서 사용된 주요 스프링 및 스프링 부트 어노테이션을 요약하면 다음과 같습니다.

| **어노테이션** | **패키지** | **목적** | **사용 예시 (컨텍스트)** |
| --- | --- | --- | --- |
| @SpringBootApplication | org.springframework.boot.autoconfigure | 메인 애플리케이션 클래스 지정, 자동 설정(auto-configuration) 및 컴포넌트 스캔(component scan) 활성화.1 | 메인 클래스 (WebtodoappApplication.java) |
| @Controller | org.springframework.stereotype | 클래스를 Spring MVC 웹 컨트롤러로 지정. 주로 뷰 이름을 반환하여 웹 페이지 렌더링.2 | TodoController 클래스 |
| @Service | org.springframework.stereotype | 클래스를 비즈니스 로직을 처리하는 서비스 컴포넌트로 지정.18 | TodoServiceImpl 클래스 |
| @Autowired | org.springframework.beans.factory.annotation | (단일 생성자 시 생략 가능) 의존성 주입(Dependency Injection)을 수행할 대상(생성자, 필드, 세터) 지정.34 | TodoController 생성자 (암묵적 사용) |
| @GetMapping | org.springframework.web.bind.annotation | HTTP GET 요청을 특정 URL 경로와 핸들러 메소드에 매핑.3 | showTasks() 메소드 (/ 경로) |
| @PostMapping | org.springframework.web.bind.annotation | HTTP POST 요청을 특정 URL 경로와 핸들러 메소드에 매핑.21 | addTask(), markTaskComplete() 메소드 (/addTask, /tasks/complete/{id} 경로) |
| @ModelAttribute | org.springframework.web.bind.annotation | 폼 데이터(요청 파라미터)를 객체에 바인딩하거나, 모델에 객체를 추가할 때 사용.21 | addTask() 파라미터, showTasks()에서 newTask 모델 추가 |
| @PathVariable | org.springframework.web.bind.annotation | URL 경로의 일부(경로 변수) 값을 메소드 파라미터에 바인딩.47 | markTaskComplete() 파라미터 id |

이 표는 학습 중 또는 복습 시 각 어노테이션의 역할과 사용법을 빠르게 파악하는 데 도움이 될 것입니다. 스프링 프레임워크는 어노테이션 기반 설정을 광범위하게 사용하므로, 주요 어노테이션의 의미를 이해하는 것이 중요합니다.

## **9. 결론 및 다음 단계**

축하합니다! 스프링 부트와 Thymeleaf를 사용하여 기본적인 CRUD(Create, Read, Update - 여기서는 완료 처리) 기능을 갖춘 웹 기반 ToDo 애플리케이션을 성공적으로 구축했습니다.

**이번 학습 단계에서 다룬 핵심 내용:**

* Spring Initializr를 이용한 스프링 부트 프로젝트 생성 및 의존성 관리 (web, thymeleaf, devtools).
* 표준 프로젝트 구조 및 계층별(Controller, Service, Model) 코드 구성 방식 이해.
* MVC 패턴의 기본 적용: @Controller, @Service 어노테이션 사용 및 역할 분담.
* 의존성 주입(생성자 주입)을 통한 컴포넌트 간의 느슨한 결합.
* @GetMapping을 이용한 데이터 조회 요청 처리 및 Model을 통한 뷰로 데이터 전달.
* Thymeleaf 기본 사용법: th:each, th:text, th:if, th:action, th:object, th:field 등을 이용한 동적 HTML 생성 및 폼 처리.
* @PostMapping, @ModelAttribute, @PathVariable을 이용한 폼 제출 및 데이터 수정 요청 처리.
* Post-Redirect-Get (PRG) 패턴을 이용한 폼 중복 제출 방지.

**사용자 피드백 반영 결과:**

* 가이드 내에서 사용자 정의 메소드(Service, Controller 메소드)는 사용(호출)되기 전에 먼저 정의하고 설명하는 순서로 구성했습니다.
* 코드 예제 내에서 Java 표준 라이브러리 메소드(예: String.isEmpty(), Objects.equals()), 스프링의 주요 어노테이션(@Controller, @Service, @GetMapping 등), Thymeleaf 속성(th:each, th:field 등)이 처음 사용될 때 인라인 주석으로 간략한 설명을 추가했습니다.
* 코드 블록 전후에 관련 개념(MVC, DI, Thymeleaf, 어노테이션 등)에 대한 설명을 제공하여 코드와 개념의 연결성을 높였습니다.

**향후 학습 방향 및 애플리케이션 개선 아이디어:**

현재 애플리케이션은 기본적인 기능을 갖추었지만, 실제 운영 환경에서 사용하기에는 여러 가지 개선점이 있습니다. 다음 단계로 다음과 같은 내용들을 학습하고 적용해 볼 수 있습니다.

1. **데이터 영속성 (Persistence):** 현재 메모리 기반 데이터 저장소는 애플리케이션 재시작 시 데이터가 사라집니다. Spring Data JPA와 H2 (인메모리 DB) 또는 MySQL/PostgreSQL 같은 실제 데이터베이스를 연동하여 데이터를 영구적으로 저장하는 방법을 학습합니다.2 @Entity, @Repository 인터페이스 등이 사용됩니다.16
2. **입력값 검증 (Validation):** 사용자가 폼에 유효하지 않은 데이터(예: 빈 설명)를 입력하는 것을 방지하기 위해 서버 측 유효성 검증(Bean Validation API, @Valid 어노테이션 등)을 추가합니다.53 Thymeleaf를 사용하여 오류 메시지를 사용자에게 표시할 수도 있습니다.53
3. **예외 처리 (Error Handling):** 특정 ID의 할 일을 찾지 못하는 경우 등 예외 상황 발생 시 사용자에게 친절한 오류 페이지를 보여주거나 적절한 응답을 반환하도록 전역 예외 처리(@ControllerAdvice) 등을 구현합니다.
4. **REST API 개발:** 현재는 서버에서 HTML을 직접 렌더링하지만, 프론트엔드(React, Vue 등)와 백엔드를 분리하기 위해 @RestController를 사용하여 JSON 형식의 데이터를 주고받는 RESTful API를 개발해 볼 수 있습니다.2
5. **보안 (Security):** Spring Security를 사용하여 사용자 인증(로그인) 및 인가(권한 관리) 기능을 추가하여 허가된 사용자만 ToDo 목록에 접근하고 수정할 수 있도록 합니다.68
6. **테스팅 (Testing):** JUnit, Mockito 등을 사용하여 컨트롤러, 서비스 로직에 대한 단위 테스트(Unit Test) 및 통합 테스트(Integration Test) 코드를 작성하여 코드의 안정성을 확보합니다.2
7. **스타일링 (Styling):** CSS 프레임워크(Bootstrap 등)를 적용하거나 직접 CSS를 작성하여 애플리케이션의 디자인을 개선합니다.

이 가이드가 스프링 부트 웹 개발 학습 여정에 든든한 발판이 되기를 바랍니다. 꾸준히 학습하고 직접 코드를 작성하며 다양한 기능을 추가해 보세요!

#### 참고 자료

1. Getting Started | Building an Application with Spring Boot, 4월 21, 2025에 액세스, <https://spring.io/guides/gs/spring-boot/>
2. Getting Started | Serving Web Content with Spring MVC, 4월 21, 2025에 액세스, <https://spring.io/guides/gs/serving-web-content/>
3. Spring Boot – Starter Web - GeeksforGeeks, 4월 21, 2025에 액세스, <https://www.geeksforgeeks.org/spring-boot-starter-web/>
4. org.springframework.boot:spring-boot-starter-web - Maven Central - Sonatype, 4월 21, 2025에 액세스, <https://central.sonatype.com/artifact/org.springframework.boot/spring-boot-starter-web>
5. Spring Boot Thymeleaf - Tutorialspoint, 4월 21, 2025에 액세스, <https://www.tutorialspoint.com/spring_boot/spring_boot_thymeleaf.htm>
6. spring-boot-starter-web vs spring-boot-starter-thymeleaf - Stack Overflow, 4월 21, 2025에 액세스, <https://stackoverflow.com/questions/31253440/spring-boot-starter-web-vs-spring-boot-starter-thymeleaf>
7. Developer Tools :: Spring Boot, 4월 21, 2025에 액세스, <https://docs.spring.io/spring-boot/reference/using/devtools.html>
8. 20. Developer tools - Spring, 4월 21, 2025에 액세스, <https://docs.spring.io/spring-boot/docs/1.5.16.RELEASE/reference/html/using-boot-devtools.html>
9. Spring boot Dev tools | Coding Shuttle, 4월 21, 2025에 액세스, <https://www.codingshuttle.com/spring-boot-hand-book/spring-boot-dev-tools>
10. Spring Boot – DevTools | GeeksforGeeks, 4월 21, 2025에 액세스, <https://www.geeksforgeeks.org/spring-boot-devtools/>
11. 64. Spring Boot Gradle plugin, 4월 21, 2025에 액세스, <https://docs.spring.io/spring-boot/docs/1.4.5.RELEASE/reference/html/build-tool-plugins-gradle-plugin.html>
12. Building Spring Boot Web Applications Sample - Gradle User Manual, 4월 21, 2025에 액세스, <https://docs.gradle.org/current/samples/sample_building_spring_boot_web_applications.html>
13. Gradle Spring Boot plugin - Help/Discuss, 4월 21, 2025에 액세스, <https://discuss.gradle.org/t/gradle-spring-boot-plugin/26407>
14. Gradle Spring Boot Devtools: developmentOnly and runtimeClasspath - Stack Overflow, 4월 21, 2025에 액세스, <https://stackoverflow.com/questions/59252470/gradle-spring-boot-devtools-developmentonly-and-runtimeclasspath>
15. Spring Boot folder structure best practices - Symflower, 4월 21, 2025에 액세스, <https://symflower.com/en/company/blog/2024/spring-boot-folder-structure/>
16. Structuring Your Code :: Spring Boot, 4월 21, 2025에 액세스, <https://docs.spring.io/spring-boot/reference/using/structuring-your-code.html>
17. Spring Boot – Code Structure | GeeksforGeeks, 4월 21, 2025에 액세스, <https://www.geeksforgeeks.org/spring-boot-code-structure/>
18. arsy786/springboot-best-practices: This repo serves as a guide for Spring Boot best practices! - GitHub, 4월 21, 2025에 액세스, <https://github.com/arsy786/springboot-best-practices>
19. What is the recommended project structure for spring boot rest projects? - Stack Overflow, 4월 21, 2025에 액세스, <https://stackoverflow.com/questions/40902280/what-is-the-recommended-project-structure-for-spring-boot-rest-projects>
20. Spring Boot MVC + Persistence - DTO, model, entity relationship and file structure, 4월 21, 2025에 액세스, <https://stackoverflow.com/questions/78622024/spring-boot-mvc-persistence-dto-model-entity-relationship-and-file-structu>
21. SPRING MVC TUTORIAL - Cogent University, 4월 21, 2025에 액세스, <https://www.cogentuniversity.com/post/spring-mvc-tutorial>
22. Difference Between @Controller and @Service Annotation in Spring | GeeksforGeeks, 4월 21, 2025에 액세스, <https://www.geeksforgeeks.org/difference-between-controller-and-service-annotation-in-spring/>
23. Difference Between @Controller and @RestController Annotation in Spring, 4월 21, 2025에 액세스, <https://www.geeksforgeeks.org/difference-between-controller-and-restcontroller-annotation-in-spring/>
24. Spring Boot @Service Annotation with Example | GeeksforGeeks, 4월 21, 2025에 액세스, <https://www.geeksforgeeks.org/spring-service-annotation-with-example/>
25. Spring @Service Annotation | DigitalOcean, 4월 21, 2025에 액세스, <https://www.digitalocean.com/community/tutorials/spring-service-annotation>
26. @Component vs @Repository and @Service in Spring | Baeldung, 4월 21, 2025에 액세스, <https://www.baeldung.com/spring-component-repository-service>
27. Can you please explain the functional ity of the annotation @Service and @Transactional? : r/SpringBoot - Reddit, 4월 21, 2025에 액세스, <https://www.reddit.com/r/SpringBoot/comments/10ektxj/can_you_please_explain_the_functional_ity_of_the/>
28. TmaneChouaib/spring-boot-mvc-app-demo: This Repository showcases the implementation of a basic Spring Boot application using the MVC architectural pattern - GitHub, 4월 21, 2025에 액세스, <https://github.com/TmaneChouaib/spring-boot-mvc-app-demo>
29. Spring Boot @Repository Annotation with Example | GeeksforGeeks, 4월 21, 2025에 액세스, <https://www.geeksforgeeks.org/spring-repository-annotation-with-example/>
30. Spring @Repository Annotation - DigitalOcean, 4월 21, 2025에 액세스, <https://www.digitalocean.com/community/tutorials/spring-repository-annotation>
31. Spring Data Annotations - Baeldung, 4월 21, 2025에 액세스, <https://www.baeldung.com/spring-data-annotations>
32. Repository (Spring Framework 6.2.6 API), 4월 21, 2025에 액세스, <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/stereotype/Repository.html>
33. Well Structured Projects : r/SpringBoot - Reddit, 4월 21, 2025에 액세스, <https://www.reddit.com/r/SpringBoot/comments/1drhemj/well_structured_projects/>
34. Spring Beans and Dependency Injection :: Spring Boot, 4월 21, 2025에 액세스, <https://docs.spring.io/spring-boot/reference/using/spring-beans-and-dependency-injection.html>
35. Guide to Spring @Autowired | Baeldung, 4월 21, 2025에 액세스, <https://www.baeldung.com/spring-autowire>
36. Spring Dependency Injection: @Autowired vs Constructor Injection | GeeksforGeeks, 4월 21, 2025에 액세스, <https://www.geeksforgeeks.org/spring-dependency-injection-autowired-vs-constructor-injection/>
37. Using @Autowired :: Spring Framework, 4월 21, 2025에 액세스, <https://docs.spring.io/spring-framework/reference/core/beans/annotation-config/autowired.html>
38. Constructor Dependency Injection in Spring | Baeldung, 4월 21, 2025에 액세스, <https://www.baeldung.com/constructor-injection-in-spring>
39. Spring MVC Tutorial - Baeldung, 4월 21, 2025에 액세스, <https://www.baeldung.com/spring-mvc-tutorial>
40. Spring MVC Tutorial | GeeksforGeeks, 4월 21, 2025에 액세스, <https://www.geeksforgeeks.org/spring-mvc/>
41. What is the difference between @Controller vs @RestController in Spring Boot? - Symflower, 4월 21, 2025에 액세스, <https://symflower.com/en/company/blog/2024/controller-restcontroller-spring-boot/>
42. Difference between spring @Controller and @RestController annotation - Stack Overflow, 4월 21, 2025에 액세스, <https://stackoverflow.com/questions/25242321/difference-between-spring-controller-and-restcontroller-annotation>
43. Difference between @Controller vs @RestController in Spring - DEV Community, 4월 21, 2025에 액세스, <https://dev.to/somadevtoo/difference-between-controller-vs-restcontroller-in-spring-4emk>
44. RestController vs. Controller: What's the difference? | TheServerSide, 4월 21, 2025에 액세스, <https://www.theserverside.com/video/RestController-vs-Controller-Whats-the-difference>
45. Difference Between Controller and RestController - Shiksha, 4월 21, 2025에 액세스, <https://www.shiksha.com/online-courses/articles/difference-between-controller-and-restcontroller-blogId-159819>
46. Should there always be a @RestController : r/SpringBoot - Reddit, 4월 21, 2025에 액세스, <https://www.reddit.com/r/SpringBoot/comments/1etkh4b/should_there_always_be_a_restcontroller/>
47. Mapping Requests :: Spring Framework, 4월 21, 2025에 액세스, <https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-controller/ann-requestmapping.html>
48. Getting Started | Handling Form Submission - Spring, 4월 21, 2025에 액세스, <https://spring.io/guides/gs/handling-form-submission>
49. Spring – @PostMapping and @GetMapping Annotation | GeeksforGeeks, 4월 21, 2025에 액세스, <https://www.geeksforgeeks.org/spring-postmapping-and-getmapping-annotation/>
50. GetMapping and PostMapping annotations - Stack Overflow, 4월 21, 2025에 액세스, <https://stackoverflow.com/questions/40680370/getmapping-and-postmapping-annotations>
51. Spring Boot MVC Tutorial | Spring Boot Web Application Course - YouTube, 4월 21, 2025에 액세스, <https://www.youtube.com/watch?v=Ku3gsv7_bCc>
52. Tutorial: Using Thymeleaf, 4월 21, 2025에 액세스, <https://www.thymeleaf.org/doc/tutorials/3.1/usingthymeleaf.html>
53. Tutorial: Thymeleaf + Spring, 4월 21, 2025에 액세스, <https://www.thymeleaf.org/doc/tutorials/2.1/thymeleafspring.html>
54. Tutorial: Thymeleaf + Spring, 4월 21, 2025에 액세스, <https://www.thymeleaf.org/doc/tutorials/3.1/thymeleafspring.html>
55. 12 Thymeleaf Looping or Iteration (th:each) with Spring Boot - YouTube, 4월 21, 2025에 액세스, <https://www.youtube.com/watch?v=c0_ix_l8UEA>
56. Spring Web MVC crash course | TheServerSide, 4월 21, 2025에 액세스, <https://www.theserverside.com/video/Spring-Web-MVC-crash-course>
57. Thymeleaf th:each cannot find object - spring boot - Stack Overflow, 4월 21, 2025에 액세스, <https://stackoverflow.com/questions/54646947/thymeleaf-theach-cannot-find-object>
58. Spring Boot Thymeleaf Form Handling Tutorial - CodeJava.net, 4월 21, 2025에 액세스, <https://www.codejava.net/frameworks/spring-boot/spring-boot-thymeleaf-form-handling-tutorial>
59. Spring Boot – Thymeleaf with Example | GeeksforGeeks, 4월 21, 2025에 액세스, <https://www.geeksforgeeks.org/spring-boot-thymeleaf-with-example/>
60. How to create a simple form in Java Spring with Thymeleaf - Stack Overflow, 4월 21, 2025에 액세스, <https://stackoverflow.com/questions/54260747/how-to-create-a-simple-form-in-java-spring-with-thymeleaf>
61. Thymeleaf Tutorial: #19 - Form Handling - Submit Form and Display User Registration Form Data - YouTube, 4월 21, 2025에 액세스, <https://www.youtube.com/watch?v=nrXktUFQlqI>
62. Thymeleaf Tutorial: #18 - Form Handling - Design User Registration Form - YouTube, 4월 21, 2025에 액세스, <https://www.youtube.com/watch?v=0WPpFJsGGnc>
63. How to handle forms in Thymeleaf and Spring in one to many relationships?, 4월 21, 2025에 액세스, <https://stackoverflow.com/questions/65053147/how-to-handle-forms-in-thymeleaf-and-spring-in-one-to-many-relationships>
64. Introduction to Spring Data JPA - Baeldung, 4월 21, 2025에 액세스, <https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa>
65. 1. Working with Spring Data Repositories, 4월 21, 2025에 액세스, <https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html>
66. Spring Boot Tutorial | GeeksforGeeks, 4월 21, 2025에 액세스, <https://www.geeksforgeeks.org/spring-boot/>
67. When to use @Controller and when @RestController annotation in RESTApi based on Spring - Stack Overflow, 4월 21, 2025에 액세스, <https://stackoverflow.com/questions/72211755/when-to-use-controller-and-when-restcontroller-annotation-in-restapi-based-on>
68. Spring Boot Tutorial – Bootstrap a Simple Application - Baeldung, 4월 21, 2025에 액세스, <https://www.baeldung.com/spring-boot-start>
69. Real World Spring Boot Web Application Example with tons of ready to use features - GitHub, 4월 21, 2025에 액세스, <https://github.com/gtiwari333/spring-boot-web-application-sample>