

Spring Boot, View Templates, Layout

558280-1
2026년 봄학기
3/26/2026
박경신

Overview

- Web Framework
 - CGI -> Servlet -> Spring
- Spring
 - Bean/loC/DI/AOP
- Spring Boot
 - AutoConfig/Starter/JAR
- Spring MVC
 - Model View Controller
- Thymeleaf View Template
 - HTML 기반의 서버 사이드 템플릿 엔진
- Layout (Fragment)

From Servlet to Spring MVC

□ Servlet

- Client -> HttpServlet -> doGet/doPost -> HTML 생성
- Request/response 직접 처리 (복잡)
- HTML 직접 생성 (유지보수 어려움)
- 공통 로직 반복

□ Spring MVC

- Client -> DispatcherServlet -> Controller -> View로 HTML 생성
- 역할 분리 (Controller / View)

□ Spring Boot 는 실행과 설정을 단순화

□ Thymeleaf는 HTML 기반의 서버 사이드 템플릿 엔진

Web Framework

- CGI (Common Gateway Interface) (1993)
 - 초기 웹 프레임워크. 서버와 통신하여 동적 콘텐츠 결과를 반환
- JavaEE (Java Platform, Enterprise Edition)
 - 자바 버전 CGI라고 불리는 서블릿(Servlet)
 - 서버 개발에 필요한 기능을 모아둔 J2EE (1999)
 - JavaEE의 핵심은 EJB(Enterprise Java Beans) 분산처리, 트랜잭션, 보안 등을 지원하는 컴포넌트 모델을 제공
- Spring (Spring 1.0 2003)
 - EJB 문제점을 개선하기 위해 개발된 프레임워크.
 - 고가의 풀스택 자바 EE 서버가 아닌 Tomcat 같은 일반 서블릿 컨테이너에서 구동.
 - EJB 보다 훨씬 간편한 방식으로 EJB가 제공하던 선언적 트랜잭션 및 보안 처리, 분산 환경 지원 등 주요 기능을 모두 사용할 수 있게 되었음

Web Framework

□ Jakarta Persistence API (JPA)

- 과거 Java Persistence API였음
- JavaEE 커뮤니티 일부로 개발되었으며, 관계형 데이터 베이스에서 객체를 유지하기 위한 프레임워크 정의를 제공.
- EJB 표준을 대체함
- 2017 하반기 오라클은 JavaEE 프로젝트를 비영리 기관인 이클립스 재단으로 이관. JavaEE 상표권은 오라클이 갖고 있어서 이클립스 재단에서도 JavaEE 패키지 이름인 javax.* 를 사용할 수 없음. 이클립스 재단은 Jakarta EE 라고 하고 패키지는 jakarta.* 로 명명.

□ Spring Data

- 핵심 데이터 객체와 해당 기본 키를 중심으로 하는 데이터 액세스 프레임워크. DDD(Domain-Driven Design) 저장소 개념.

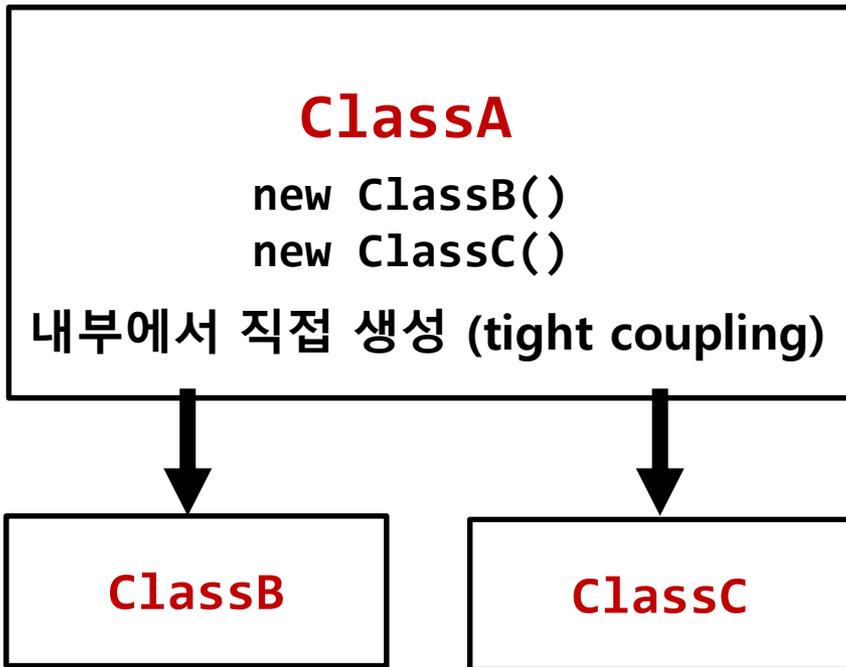
□ Spring Boot (2014)

- Spring 애플리케이션 개발을 위한 경량 프레임워크.
- Spring 6 기반 Spring Boot 3.0은 2022년 12월에 출시.

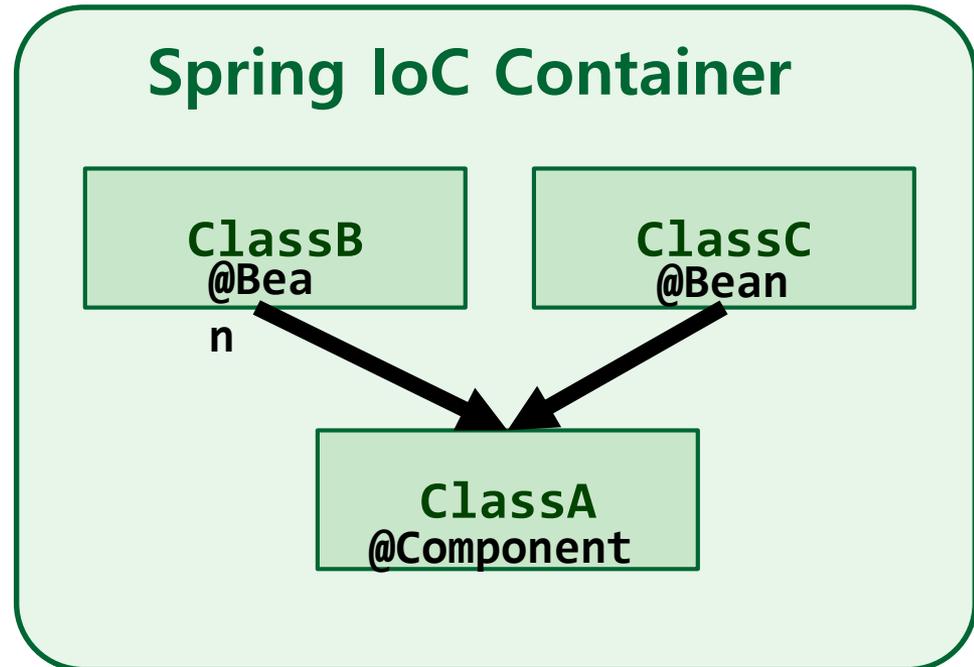
- 엔터프라이즈용 Java 애플리케이션 개발을 지원하는 오픈소스 애플리케이션 프레임워크
- POJO (Plain Old Java Object)
 - 순수 Java만을 통해서 생성한 객체를 의미.
- IoC (Inversion of Control) / DI (Dependency Injection)
 - IoC (제어 역전)는 메소드나 객체의 호출을 개발자가 결정하는 것이 아니라 외부에서 결정되는 것을 의미.
 - Spring IoC 컨테이너는 ApplicationContext 또는 BeanFactory 중 하나를 사용, Bean을 만들고, 의존성을 엮어주며 제공해주는 역할.
 - 스프링은 DI (의존성 주입)을 통해 객체 간의 관계를 설정. IoC가 일어날 때 스프링 내부에 있는 객체들(Beans) 간의 관계를 컨테이너가 자동적으로 연결.
- AOP (Aspect Oriented Programming)
 - 스프링에서는 시스템 보안, 로그, 트랙잭션 등 횡단 관심사를 모듈로 분리해서 제작 가능. 공통 관심 사항을 프로그래밍하여 이 코드를 여러 코드에 짜 넣음(weave). AspectJ로 이미 사용되어 왔음.

IoC/DI

DI 없이 (개발자가 직접 생성)



Spring DI 사용 (IoC 컨테이너)



컨테이너가 알아서 주입 - 코드 수정 최소화!

Spring

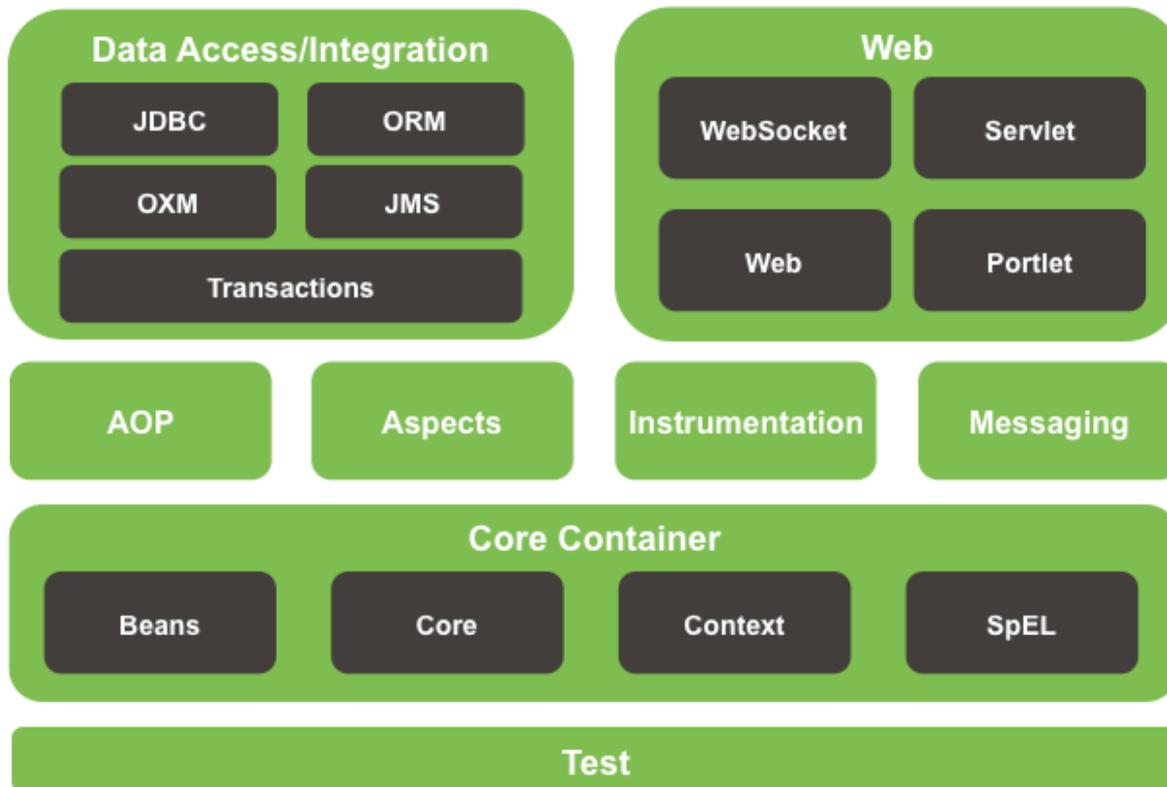
<https://docs.spring.io/spring-framework/reference/overview.html>

- 스프링 프레임워크는 다양한 기능이 존재함. 약 20개의 모듈로 구성되어 있음.
- 많이 사용되는 대표적 모듈
 - Spring JDBC (Java DataBase Connectivity)
 - Spring ORM (Object Relational Mapping)
 - Spring MVC (Model View Controller)
 - Spring AOP (Aspect Oriented Programming)
 - Spring Test
 - Spring Expression Language (SpEL)

Spring



Spring Framework Runtime



<https://docs.spring.io/spring-framework/docs/4.2.8.RELEASE/spring-framework-reference/html/overview.html>

Spring Boot



“Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can “just run”.”

- from <https://spring.io/projects/spring-boot>

Spring Boot

- 간단한 설정과 구성을 통해 스프링 애플리케이션 개발을 빠르게 시작할 수 있게 한 프레임워크
- 간결한 설정
 - 번거로운 XML 설정이 필요 없으며, 최소한의 설정으로 Spring을 사용할 수 있고, 기본 설정을 자동으로 처리함 (AutoConfiguration)
- 내장 서버
 - 스프링 부트는 내장된 서버(내장 Tomcat, Jetty, Undertow)를 제공하여 별도의 서버 설정 없이 애플리케이션을 실행할 수 있음.
 - 배포를 위해 War 파일을 생성해서 Tomcat에 배포할 필요 없으며, JAR 파일에는 모든 의존성 라이브러리가 포함되어 있어 외부 서버 없이도 애플리케이션을 실행할 수 있음.
 - 또한 스프링 부트는 독립적으로 실행 가능한 Jar 파일로 프로젝트를 빌드할 수 있어, 클라우드 서비스 및 도커와 같은 가상화 환경에 빠르게 배포할 수 있음.

Spring Boot

- 의존성(dependency) 관리 간소화
 - Spring boot starter 를 제공하여, Maven/Gradle 설정 시 자동으로 호환되는 버전을 관리해줌. 의존성 관리가 간편함.
 - 개발자는 버전 충돌이나 복잡한 의존성 설정 (호환성 체크 등)에 대해 걱정할 필요가 없음.
- 운영 편의성
 - 스프링 부트는 애플리케이션의 상태 모니터링, 로깅, 보안 설정 등 운영에 필요한 기능들을 제공하여, 애플리케이션의 운영과 관리가 편리해지고 안정성이 향상시킬 수 있음.
 - 모니터링 관리를 위한 Spring Actuator 제공
 - 서비스가 정상적으로 동작하는지 상태 모니터링 기능 제공

Spring vs Spring Boot

Spring Framework (Core)	Spring Boot
설정: XML 또는 Java Config 직접 작성 필요	설정: AutoConfiguration (거의 자동 처리)
서버: 외부 WAS (Tomcat emd) 별도 설치 후 WAR 파일 배포	서버: 내장 Tomcat (Embedded Server) — JAR 하나로 바로 실행
의존성: 라이브러리 버전 수동 관리 — 충돌 위험 가능	의존성: Starter로 버전 자동 관리 — 충돌 없음
배포: WAR 파일 → 서버에 배포	배포: 실행형 JAR java -jar app.jar 로 실행

Spring Boot = Spring Framework + Auto Configuration + Embedded Server + Starter Dependency. 기존 Spring 지식은 그대로 적용

Spring Boot Starter Dependency

- 스프링 부트에서 starter란 의존성과 설정을 자동화 해주는 모듈임
 - `spring-boot-starter-web` - 웹 응용프로그램 개발을 위한 핵심 기능
 - `spring-boot-starter-web-services` - SOAP 웹 서비스
 - `spring-boot-starter-test` - 단위 테스트, 통합 테스트
 - `spring-boot-starter-jdbc` - 기본적인 JDBC
 - `spring-boot-starter-hateoas` - HATEOAS 기능을 사용한 RESTful 서비스
 - `spring-boot-starter-security` - 스프링 보안(인증, 권한)
 - `spring-boot-starter-data-jpa` - 스프링 Data JPA (Hibernate)
 - `spring-boot-starter-data-web` - RESTful 응용프로그램
 - `spring-boot-starter-data-rest` - Spring Data REST를 사용한 REST 서비스
 - `spring-boot-starter-cache` - 캐시 지원

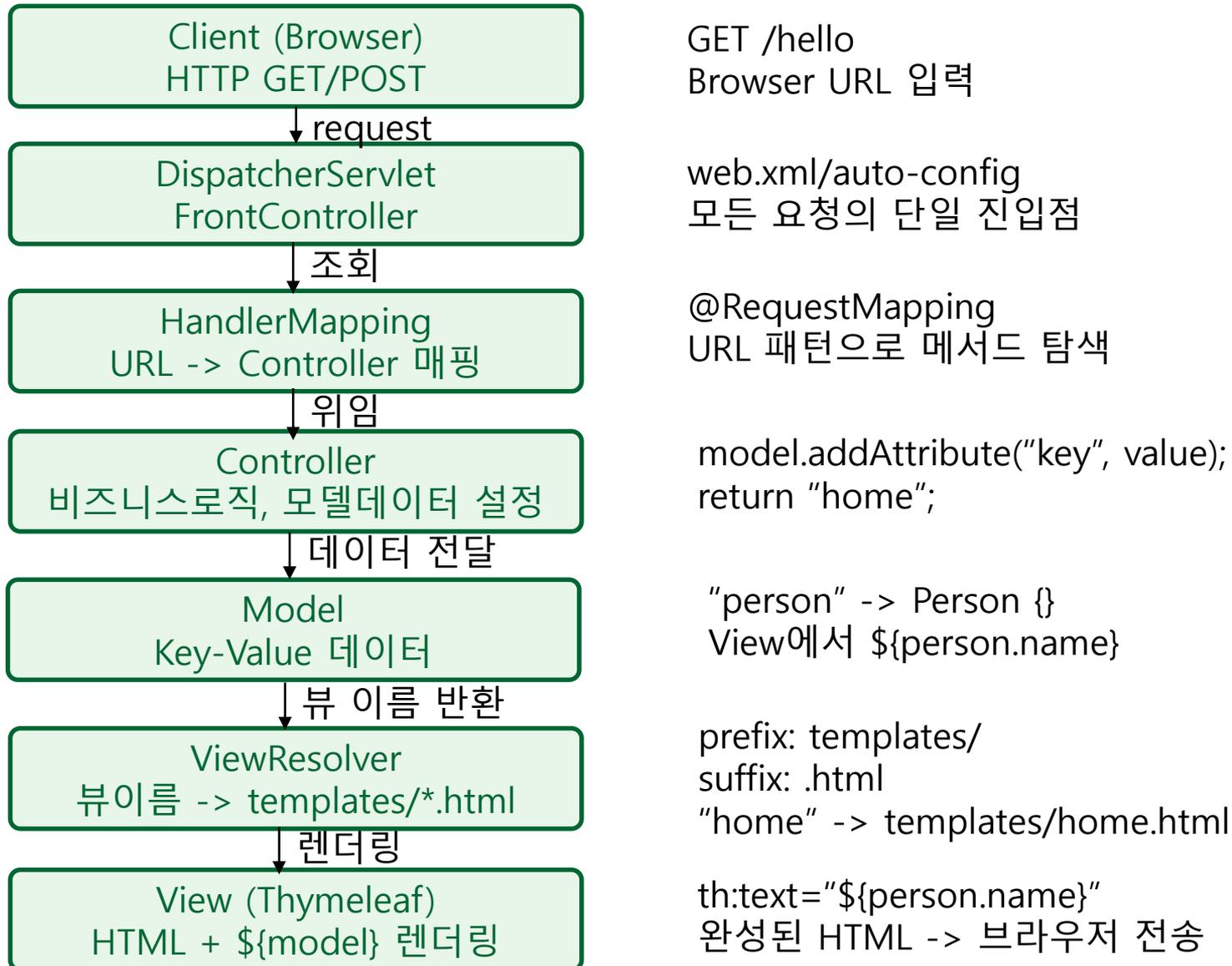
Spring Boot Starter Web

- [spring-boot-starter-web](#)을 사용하면, 기본적으로 임베디드 톰캣이 구성되어 웹 서버로 사용할 수 있으며, 스프링 MVC를 활용하여 컨트롤러를 정의하고 HTTP 요청을 처리할 수 있음
- 다음과 같은 주요 라이브러리를 포함함
 - [Spring MVC](#) - Spring Web MVC는 서블릿 API 위에 구축된 원래의 웹 프레임워크
 - [Tomcat](#) - 가장 널리 사용되는 Java 서블릿 컨테이너
 - [Jackson](#) - JSON을 Java 객체로 변환하거나 Java 객체를 JSON으로 변환하는데 사용되는 라이브러리

Spring MVC

- Spring MVC는 서블릿 API 위에 구축된 프레임워크로, 다음과 같은 구성 요소들이 있음
 - `DispatcherServlet` - 클라이언트 요청을 받아서 적절한 컨트롤러에게 라우팅함
 - `HandlerMapping` - 클라이언트 요청을 처리할 수 있는 컨트롤러를 찾는 역할. `RequestMappingHandlerMapping`이 가장 많이 사용됨
 - `Controller` - 클라이언트 요청을 처리하고 응답을 반환함
 - `ModelAndView` - 이 객체는 뷰 이름과 모델 객체를 담고 있으며, `DispatcherServlet`은 이 정보를 사용하여 응답을 생성함
 - `ViewResolver` - 컨트롤러가 반환한 뷰 이름을 실제 뷰로 변환하는 역할. `InternalResourceViewResolver`가 가장 흔히 사용됨
 - `View` - 최종적으로 클라이언트에게 반환될 응답을 렌더링함. 일반적으로 Thymeleaf 등 템플릿 엔진을 사용하여 HTML을 생성함

Spring Boot MVC Request Flow



pom.xml

- Maven 프로젝트를 생성하면 루트 디렉토리에 생성되는 파일로 Project Object Model (POM)를 설정함
- 주요 설정 정보
 - 프로젝트 정보 - 프로젝트 이름, 개발자 목록, 라이선스
 - 빌드 설정 정보 - 소스, 리소스, 라이프 사이클 등 실행할 플러그인 등
 - POM 연관 정보 - **의존 프로젝트(모듈)**, 상위 프로젝트, 하위 모듈 등

Spring Boot Maven Dependencies

□ spring-boot-starter-parent

- Spring Boot 기반 애플리케이션에 대한 기본 구성 및 종속성을 제공하는 특별한 상위 프로젝트임
- 플러그인(plugins) 및 종속성 (dependencies)을 정의하고 있음
- spring-boot-dependencies에서 상속받음

□ spring-boot-dependencies

- BOM (Bill of Materials), 즉 라이브러리 버전 목록 (재료 명세서)
- 독립적인 방식으로 플러그인 및 종속성 버전을 정의함
- **dependencyManagement** 및 **pluginManagement** 영역에 라이브러리들의 버전이 명시되어 있음
- 프로젝트 내부적으로 사용하는 parent가 있거나 명시적으로 Maven 선언을 원한다면 spring-boot-starter-parent를 parent로 사용하지 않고도 dependencyManagement만을 사용 가능함
- spring-boot-dependencies에서는 spring boot에서 기본적으로 사용하고 있는 dependencies 들을 정의하고 있음

Parent POM

- Parent POM은 여러 모듈에서 공통 설정을 공유하기 위한 “중앙 설정 파일”
- 대규모 프로젝트는 여러 모듈을 생성할 가능성이 높음
- POM은 Parent POM을 상속하여 미리 설정된 값이나 의존성들의 버전정보를 받아올 수 있음
- **Parent POM**은 다음을 유지하는 데 도움이 될 수 있음
 - 하위 모듈 일관성 (모든 모듈이 같은 라이브러리 버전 사용)
 - 가벼운 하위 모듈 (Child POM이 단순해짐)
- Maven POM은
 - **단 하나의 상속된 부모만** 가질 수 있음
 - **하지만 많은 POM dependency** 정의를 가져올 수 있음
- Parent POM 파일에 적힌 값들은 상속받은 Child POM에서 재정의(override) 가능함

Child POM - Declare pom inheritance in the child pom.xml

- 상위 pom.xml에서 정의를 가져오려면, 하위 pom.xml에서 아래와 같이 부모 POM 상속을 선언

Place this declaration in the child pom building the JAR archive

```
<parent>
```

```
  <groupId>(parent groupId)</groupId>
```

```
  <artifactId>(parent artifactId)</artifactId>
```

```
  <version>(parent version)</version>
```

```
</parent>
```

Starter Parent는 프로젝트에서 사용하는 다양한 라이브러리 간의 버전 충돌 문제를 방지하는 역할 (spring-boot-starter-parent-3.4.3.pom 파일에 정의됨)

```
<parent>
```

```
  <groupId>org.springframework.boot</groupId>
```

```
  <artifactId>spring-boot-starter-parent</artifactId>
```

```
  <version>3.4.3</version>
```

```
  <relativePath/> <!-- lookup parent from repository -->
```

```
</parent>
```

Declare dependency on artifacts used

- 선언을 통해 parent-configured dependencies을 지정함
- 가져온 **spring-boot-dependencies**가 버전 선언을 처리함

```
<dependencies>
```

```
# Starter Web은 Spring MVC를 사용한 REST 서비스 개발에 사용
```

```
  <dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-web</artifactId>
```

```
  </dependency>
```

```
# Starter Test는 JUnit, Hamcrest, Mockito 포함한 Spring 앱 테스트 기능 제공
```

```
  <dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-test</artifactId>
```

```
  </dependency>
```

```
</dependencies>
```

pom.xml

```
<!-- pom.xml -->
```

```
<parent>
```

```
  <groupId>org.springframework.boot</groupId>
```

```
  <artifactId>spring-boot-starter-parent</artifactId>
```

```
  <version>3.4.3</version>
```

```
  <relativePath/> <!-- lookup parent from repository -->
```

```
</parent>
```

```
<!-- spring-boot-starter-parent-3.4.3.pom -->
```

```
<parent>
```

```
  <groupId>org.springframework.boot</groupId>
```

```
  <artifactId>spring-boot-dependencies</artifactId>
```

```
  <version>3.4.3</version>
```

```
</parent>
```

pom.xml

```
<!-- spring-boot-dependencies-3.4.3.pom -->
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-dependencies</artifactId>
  <version>3.4.3</version>
  <packaging>pom</packaging>
  <name>spring-boot-dependencies</name>
  <description>Spring Boot Dependencies</description>
...
  <dependencyManagement>
    <dependencies>
...
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-test</artifactId>
        <version>3.4.3</version>
      </dependency>
...
    </dependencies>
  </dependencyManagement>
```

Simple Spring Boot Application - SpringBootHelloApplication

- **@SpringBootApplication**는 Spring 자동 구성을 트리거함
 - 응용프로그램 시작 시, Spring Boot는 **@Configuration** 클래스를 스캔함
 - Application-specific configuration classes
 - Spring Boot Class suffixed by AutoConfiguration

@SpringBootApplication

```
public class SpringBootHelloApplication {  
    public static void main(String[] args) {  
        // Initiate Spring application bootstrap by invoking SpringApplication.run()  
        // and passing an application class and an args passed into main()  
        SpringApplication.run(SpringBootHelloApplication.class, args);  
    }  
}
```

@SpringBootApplication

- **@SpringBootApplication** 는 다음을 포함함
 - **@ComponentScan** 는 legacy Spring annotation으로, ComponentScan이 있는 디렉터리부터 하위 디렉터리를 찾아서 @Service, @Repository, @Controller, @RestController, @Component 등이 붙어 있는 **Bean들을 찾아서 등록해줌**
 - @SpringBootApplication 즉, 메인 함수가 있는 패키지를 포함한 하위 디렉터리에 있는 Component를 Bean으로 등록해줌
 - **@SpringBootConfiguration** 은 Spring의 @Configuration 같은 기능. 클래스에 @Configuration이 있으면 스프링은 Java Configure 설정 클래스로 간주하여 Bean으로 등록함
 - 예를 들어, 팩토리 @Bean 정의
 - **@EnableAutoConfiguration** 은 Spring이 클래스 경로, 애플리케이션에서 정의한 Beans 및 property 설정을 기반으로 자동 설정(auto-configuration)을 수행할 수 있음

@SpringBootApplication

@SpringBootApplication

```
// @ComponentScan
// @SpringBootApplication
// @EnableAutoConfiguration
public class SpringBootHelloApplication {
    public static void main(String[] args) {
        ApplicationContext ctx =
SpringApplication.run(SpringBootHelloApplication.class, args);

        // print the spring boot loaded beans
String[] beans = ctx.getBeanDefinitionNames();
Arrays.sort(beans);
for (String bean: beans) {
            System.out.println(bean);
}
    }
}
```

@Controller

□ @Controller [Class]

- 해당 클래스가 스프링 부트가 인식해야 하는 Controller임을 알림
- @Controller는 전통적인 Spring MVC Controller

@Controller

```
public class SpringBootHelloController {  
    @RequestMapping(value="/helloworld", method=RequestMethod.GET)  
    @ResponseBody  
    public String helloworld() {  
        return "Hello World from Tomcat!";  
    }  
}
```

@Controller

□ Controller 역할

- 요청 수신 -> 비즈니스 로직 처리 -> 데이터 준비 -> 뷰 지정

@Controller

```
public class SpringBootHelloController {  
    @GetMapping(value="/home")  
    public String home(Model model) {  
        model.addAttribute("data", "Hello");  
        return "home"; // templates/home.html  
    }  
}
```

- 1.요청 처리 (Request Mapping)
- 2.비즈니스 로직 호출
- 3.데이터 (Model) 생성
- 4.View 이름 반환

@RequestMapping

□ @RequestMapping [Class|Method]

- **value** 는 맵핑할 주소를 명시
- **method** 는 해당 맵핑이 어떠한 요청 방식(GET, POST, PUT, DELETE, 등)으로 대응할지에 대한 설정
- **produces** 는 해당 맵핑의 응답 결과로 어떠한 콘텐츠가 반환될지에 대한 설정. MIME 타입을 사용.
- @RequestMapping이 Controller 클래스에 부여 되었을 경우 해당 컨트롤러가 가지는 맵핑 메서드(@GetMapping, @PostMapping, 등)의 전역적인 접두어로서 맵핑
- @RequestMapping이 컨트롤러 내부의 메서드에 부여되었을 경우 해당 요청에 대한 행동(Action)을 실행할 메서드를 지정하게 됨

@RequestBody, @ResponseBody

□ @RequestBody/@ResponseBody

- Request/Response 메시지의 본문(body)에 담기는 데이터 형식은 여러 가지 있지만 가장 대표적으로 **JSON** 이며, **XML** 도 사용됨
- @RequestBody, @ResponseBody 을 명시함으로써 MessageConverter를 통한 데이터 변환 과정을 거치게 됨
 - 클라이언트에서 서버에 RequestBody로 JSON 형식의 요청 데이터를 전송했을 때, Java에서는 **JSON 에서 Java Object로의 변환**이 필요함
 - 서버에서 클라이언트로 다시 응답 데이터 **ResponseBody**를 보낼 때도 **Java Object 에서 JSON 또는 XML** 같은 형식에서의 변환이 필요함
- @RequestBody
 - 일반적인 GET/POST의 요청 파라미터(URL/form)은 사용 않함
 - XML이나 **JSON**기반의 메시지를 사용한 요청의 경우에 유용함
- @ResponseBody
 - @Controller 인 경우에 응답 데이터를 **자바 객체로 받기 위해서는 @ResponseBody**를 반드시 명시해주어야 함

@RestController

□ @RestController

- @RestController 는 @Controller 에 @ResponseBody 가 추가
- 리턴값에 자동으로 @ResponseBody가 붙게 되어 별도로 어노테이션을 명시해주지 않아도 HTTP 응답 데이터 (ResponseBody) 에 자바 객체가 매핑되어 전달됨
- @RestController 주용도는 JSON 형태로 객체 데이터를 반환하는 것

@RestController

```
public class SpringBootHelloController {  
    @RequestMapping(value="/helloworld", method=RequestMethod.GET)  
    public String helloworld() {  
        return "Hello World from Tomcat!";  
    }  
}
```

Model & View

```
List<Person> people = new ArrayList<Person>();
```

@Controller

```
public class SpringBootHelloController {
```

```
    @GetMapping(value="/home")
```

```
    public String home(Model model) {
```

```
        model.addAttribute("data", "Problem Solving Programming!!");
```

```
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd  
HH:mm:ss");
```

```
        model.addAttribute("serverTime", dateFormat.format(new Date()));
```

```
        Person dooly = new Person("Dooly", 1000, 60.0, 126.4, Gender.MALE);
```

```
        model.addAttribute("person", dooly);
```

```
        people.add(person);
```

```
        people.add(new Person("Heedong", 3, 40.0, 56.4, Gender.MALE));
```

```
        model.addAttribute("people", people);
```

```
        return "home"; // templates/home.html 이란 view 이름 return
```

```
    }
```

```
}
```

Module Source Tree

```
|-- pom.xml
|-- src
|   |-- main
|       |-- java
|           |-- kr
|               |-- ac
|                   |-- dankook
|                       |-- ace
|                           |-- springboothello
|                               |-- SpringBootApplication.java
|                               |-- SpringHelloController.java
|               |-- resources
|-- test
|   |-- javaWkrWacWdankookWaceWspringboothello
|       |-- SpringBootApplicationTests.java
```

Spring Boot Folder

□ 스프링 부트 폴더 규약

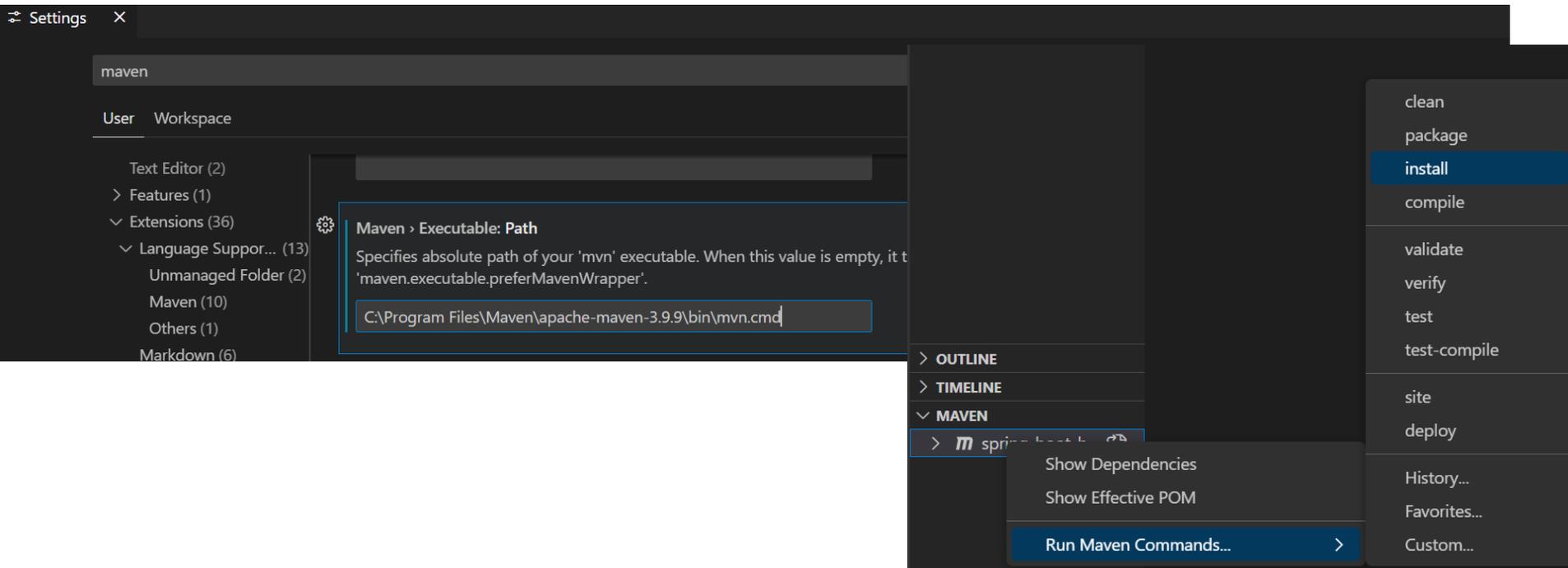
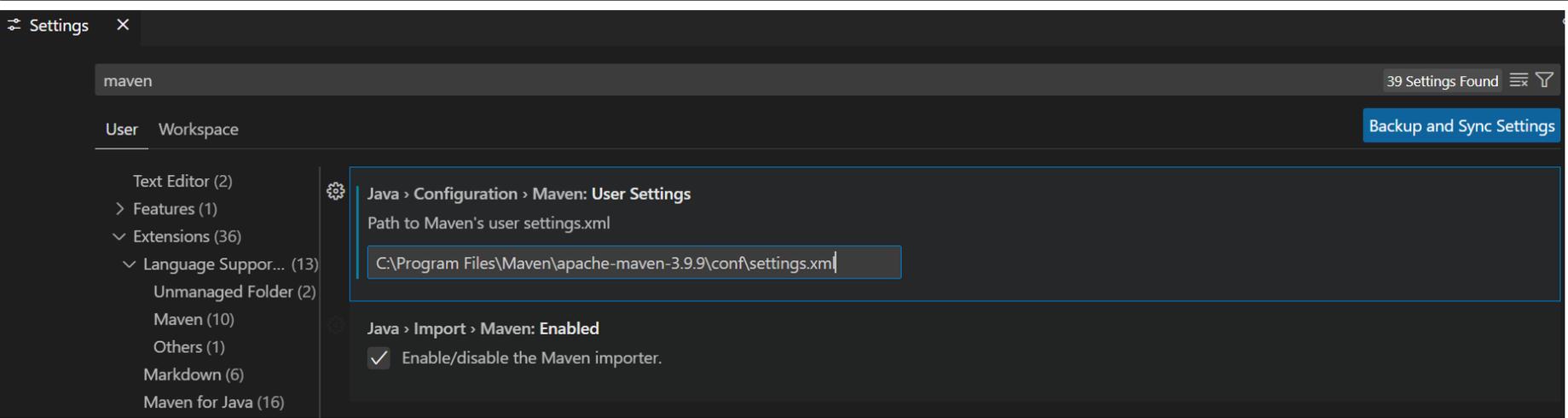
- 스프링 부트는 기본적으로 JAR 로 실행되므로 webapp 폴더를 만들지 않고, 웹 자원들을 사용하기 위한 다음 규약을 제공

웹 자원	경로
정적 HTML (html, css, image, js)	src/main/resources/static src/main/resources/static/css src/main/resources/static/image src/main/resources/static/js
웹페이지 대표 아이콘	src/main/resources/favicon.ico
템플릿	src/main/resources/templates .html - Thymeleaf .tpl - Groovy .ftl - Freemarker .vm - velocity .mustache - mustache

Spring Boot Maven Jar Build in VS Code

- VS Code에서 Maven 프로젝트 생성
 - 마켓플레이스(CTL+SHIFT+X)에서 **Maven for java** 패키지 설치
- Apache maven 설치
 - <https://maven.apache.org/download.cgi> 에서 **apache-maven-3.9.x-bin.zip** 다운로드해서 압축해제 후
 - **C:\Program Files\Maven\apache-maven-3.9.x** 넣어줌
- VS Code 설정(파일->기본설정->설정)에서 **maven 검색**
 - **Java>Configuration>Maven:User Settings** 에 **C:\Program Files\Maven\apache-maven-3.9.x\conf\settings.xml**
 - **Maven>Executable:Path** 에 **C:\Program Files\Maven\apache-maven-3.9.x\bin\mvn.cmd**
- VS Code 사이드바로 돌아가서, Maven Project를 우클릭,
Run Maven Commands -> install 하면 **jar로 빌드**

Spring Boot Maven Jar Build in VS Code



View Templates

□ View Templates

- <https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#web>
- 스프링 부트는 가능하다면 JSP를 피하고 Thymeleaf와 같은 템플릿 엔진(Thymeleaf, FreeMarker, Groovy, Mustache 지원) 사용을 권장
- 템플릿 엔진이란 서식(Html)과 데이터(Database)를 조합한 결과물을 만들어주는 도구

Thymeleaf

- pom.xml에 타임리프 의존성(dependency) 추가

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```
- 이후 별다른 설정 없이 html 파일에 `<html xmlns:th="http://www.thymeleaf.org">`를 명시하는 것으로 타임리프 사용 가능

Thymeleaf

□ Thymeleaf 특징

- *.html 확장자를 사용
- 별도의 레이아웃 프레임워크의 도움 없이 레이아웃 관리 가능
- Scriptlet이 아닌 HTML 문법으로 Java 데이터를 처리 가능

Thymeleaf

□ 기본 표현식

구분	유형	사용법
표현	변수	<code>\${...}</code>
	선택 변수	<code>*{...}</code> 하나의 변수 값이 아니라 객체의 속성을 출력하고자 할 때 <code>*{...}</code> 로 객체 이름을 명시한 뒤에 속성값들을 출력 또는 닷연산자 (.)를 사용해서 속성값 출력
	Message	<code>#{...}</code>
	URL Link	<code>@{/...}</code>
Literals	text	<code>" ' ... "</code>
	number	<code>" ... "</code>
	boolean	<code>\${...} == false</code>
	null	<code>\${...} == null</code>

Thymeleaf

□ 변수

Current time is ``

□ 선택 변수

```
<div th:object="{user}">
  <p>name: <span th:text="*{name}"></span></p>
  <p>age: <span th:text="*{age}"></span></p>
</div>
```

□ 메시지

``

□ 링크

`Return to sample`

Thymeleaf

□ Literals

- Text - 'spring boot', 'thymeleaf'
- Number - 0, 1, 2, 20, 3.14
- Boolean - true, false
- Null - null

□ Operators

- Arithmetic - `th:text = "${width} * ${height}"`
- Logical - `th:if = "${not finished}"`
 - and, or, not (!)
- Relational - `th:if = "${items.count} gt 10"`
 - gt (>), lt (<), ge (>=), le (<=), not (!), eq (==), neq/ne(!=)

Look at a `template file`

`text`

`<p>This number is 100</p>`

`<div th:if="${user.isAdmin()} == true">Admin</div>`

Thymeleaf

□ 조건문

■ if-unless

```
<td>  
  <span th:if="{person.gender} == 'M'" th:text="Male" />  
  <span th:unless="{person.gender} == 'M'" th:text="Female" />  
</td>
```

■ switch-case

```
<td th:switch="{person.gender}">  
  <span th:case="M" th:text="Male" />  
  <span th:case="F" th:text="Female" />  
</td>
```

Thymeleaf

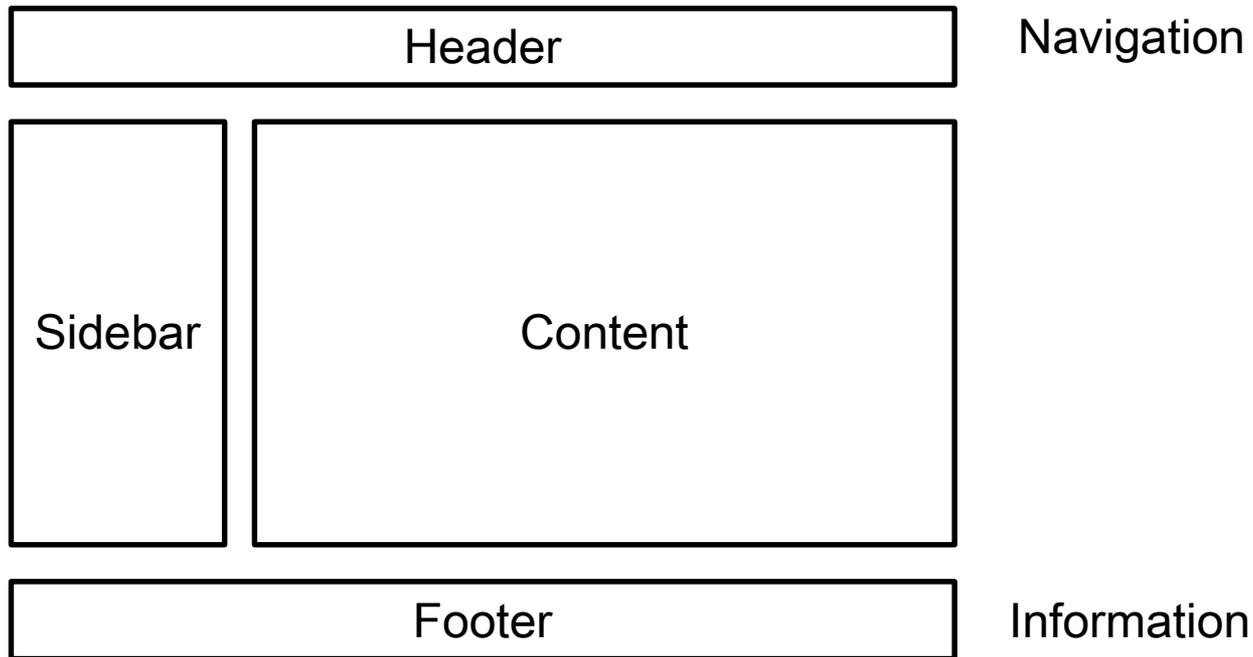
□ 반복문 표현

- 배열이나 컬렉션 같은 여러 요소를 가진 대상에 each 를 사용해서 요소들을 태그 안에서 표출

```
<tbody>
  <tr th:each= " person: ${people}">
    <td th:text="${person.name}" />
    <td th:text="${person.age}" />
    <td th:text="${person.gender}" />
  </tr>
</tbody>
```

Layout

- Layout
 - Header/sidebar/content/footer layout



Layout - Header

□ templates/fragments/header.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Problem Solving Programming Header</title>
  <!-- Add your CSS stylesheets or links here -->
</head>
<body>
  <header>
    <!-- Header content goes here -->
    <h1>Problem Solving Programming Header</h1>
    <!-- Add navigation links or other header elements -->
  </header>
</body>
</html>
```

Layout - Footer

□ templates/fragments/footer.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<body>
  <footer>
    <!-- Footer content goes here -->
    <p>&copy; Problem Solving Programming Website. All rights
reserved.</p>
    <!-- Add additional footer content or links -->
  </footer>
</body>
</html>
```

Layout - Sidebar

□ templates/fragments/sidebar.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
</head>
<body>
  <div th:fragment="sidebar">
    <!-- Sidebar content goes here -->
    <ul>
      <li><a href="#">Sidebar Link 1</a></li>
      <li><a href="#">Sidebar Link 2</a></li>
      <li><a href="#">Sidebar Link 3</a></li>
    </ul>
  </div>
</body>
</html>
```

Layout - Content

□ templates/fragments/content.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<body>
  <div th:fragment="content">
    <!-- Content area goes here -->
    <h2>Main Content Area</h2>
    <p>This is where your main content will be displayed.</p>
    <!-- Add your dynamic content using Thymeleaf syntax -->
  </div>
</body>
</html>
```

Layout – hello.html

□ templates/hello.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Main Template</title>
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
">
</head>
```


Layout – bye.html

□ templates/bye.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Main Template</title>
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
">
</head>
```

Layout – bye.html

```
<body>
  <header th:replace="~{fragments/header :: header}"></header>
  <div class="container">
    <div class="row">
      <div class="col-md-3">
        <aside th:replace="~{fragments/sidebar :: sidebar}"></aside>
      </div>
      <div class="col-md-9">
        <main th:replace="~{fragments/content :: content}"></main>
        <div>
          <h1>Bye!</h1>
          <p th:text="|Bye! ${name}!|" />
        </div>
      </div>
    </div>
  </div>
  <footer th:replace="~{fragments/footer :: footer}"></footer>
</body>
</html>
```

Controller

@Controller

```
public class SpringBootTestController {  
    @GetMapping("/viewlayout/hello")  
    public String hello(Model model) {  
        Person dooly = new Person("Dooly", 1000, 40.0, 126.4, Gender.MALE);  
        model.addAttribute("person", dooly);  
        return "hello"; // templates/hello.html  
    }  
  
    @GetMapping("/viewlayout/bye")  
    public String bye(Model model) {  
        model.addAttribute("name", "Dooly");  
        return "bye"; // templates/bye.html  
    }  
}
```