Spring Boot, View Templates, Layout

524730-1
2024년 봄학기
3/27/2024
박경신

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 2004
  - EJB 문제점을 개선하기 위해 개발된 프레임워크.
  - 고가의 풀스택 자바EE 서버가 아닌 Tomcat같은 일반 서블릿 컨테이너에서 구동.
  - EJB보다 훨씬 간편한 방식으로 EJB가 제공하던 선언적 트랜잭션 및 보안 처리, 분산 환경 지원 등 주요 기능을 모두 사용할 수 있게 되었음

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.

Spring

- 엔터프라이즈용 Java 애플리케이션 개발을 지원하는 오픈소스 애플리케이션 프레임워크
- POJO (Plan 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(Bean) 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.

Spring

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

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 Boot

- Spring Boot Starter Dependency
  - Spring-boot-starter-web-service - SOAP 웹 서비스
  - Spring-boot-starter-web - RESTful 응용프로그램
  - Spring-boot-starter-test - 단위 테스트, 통합 테스트
  - Spring-boot-starter-jdbc - 기본적인 JDBC
  - Spring-boot-starter-security - 스프링 보안(인증, 권한)
  - Spring-boot-starter-data-jpa - 스프링 Data JPA (Hibernate)
  - Spring-boot-starter-cache - 캐시

pom.xml

- Maven 프로젝트를 생성하면 루트 디렉토리에 생성되는 파일로 Project Object Model (POM)를 담고 있음
- 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
  - "재료 명세서"("Bill of Materials")
  - 독립적인 방식으로 플러그인 및 종속성 버전을 정의함 .
  - dependencyManagement 및 pluginManagement 정의를 상속하기 위해 부모로 사용될 수 있음.
  - dependencyManagement만 활용하기 위해 가져올 수 있음.
  - spring-boot-dependencies에서는 spring boot에서 기본적으로 사용하고 있는 dependencies 들을 정의하고 있음.

Parent POM

- 개발 프로젝트는 여러 모듈을 생성할 가능성이 높음
- Parent POM은 다음을 유지하는 데 도움이 될 수 있음
  - 하위 모듈 일관성
  - 가벼운 하위 모듈
- Maven POM은
  - **단 하나의 상속된 부모**만 가질 수 있음
  - 하지만 **많은 POM dependency** 정의를 가져올 수 있음

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

- 상위 pom.xml에서 정의를 가져오려면 하위 pom.xml에서 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

```
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.2</version>
        <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Declare dependency on artifacts used

- 선언을 통해 parent-configured dependencies을 지정함
- 가져온 spring-boot-dependencies가 버전 선언을 처리함

 # Starter Web은 Spring MVC를 사용한 REST 서비스 개발에 사용

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

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

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

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 @Component 가 있는 클래스에 대한 다양한 패키지 검색을 포함하거나 제외하도록 컴포넌트 스캔을 구성하는 legacy Spring annotation 임.
    - 기본적으로 스캔은 annotation을 선언한 패키지로 시작하여 거기서부터 계속 진행됨.
  - **@SpringBootConfiguration** 은 legacy Spring annotation @Configuration 과 마찬가지로, 클래스가 구성 정보를 제공할 수 있음을 나타냄.
    - 예를 들어, 팩토리 @Bean 정의
  - **@EnableAutoConfiguration** 은 Spring이 클래스 경로, 애플리케이션에서 정의한 beans 및 property 설정을 기반으로 자동 구성을 수행할 수 있음.

@SpringBootApplication

```
@SpringBootApplication
// @ComponentScan
// @SpinghBootConfiguration
// @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!";
    }
}
```

@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 컨텐트가 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 RequestBody 로 JSON 형식의 요청 데이터를 전송했을 때, Java에서는 JSON 에서 Java Object로의 변환이 필요함
    - 서버에서 클라이언트로 다시 응답 데이터 ResponseBody 를 보낼 때도 Java Object 에서 JSON 또는 XML 같은 형식으로의 변환이 필요함
  - @RequestBody
    - 일반적인 GET/POST의 요청 파라미터라면 @RequestBody를 사용않함
    - 반면에 xml이나 json기반의 메시지를 사용한 요청의 경우에 유용함
  - @ResponseBody
    - @Controller 인 경우에 바디를 자바 객체로 받기 위해서는 @ResponseBody 를 반드시 명시해주어야 함

@RestController

- **@RestController**
  - **@RestController 는 @Controller 에 @ResponseBody 가 추가**
  - 리턴값에 자동으로 @ResponseBody가 붙게 되어 별도 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

```
@Controller
public class SpringBootHelloController {
    @GetMapping(value="/home")
    public String home(Model model) {
        model.addAttribute("data", "AJ24!!");
        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("user", dooly);
        List<Person> people = new ArrayList<Person>();
        people.add(dooly);
        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
    |    |                       `-- SpringBootHelloApplication.java
    |    |                       `-- SpringBootHelloController.java
    |    `-- resources
    `-- test
        |-- java₩kr₩ac₩dankook₩ace₩springboothello
            `-- SpringBootHelloApplicationTests.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-3.9.6-bin.zip** 다운로드해서 압축해제 후
  - **C:₩Program Files₩Maven₩apache-maven-3.9.6** 넣어줌
- VS Code 설정(파일->기본설정->설정)에서 **maven 검색**
  - **Java>Configuration>Maven:User Settings** 에 **C:₩Program Files₩Maven₩apache-maven-3.9.6₩conf₩settings.xml**
  - **Maven>Executable:Path** 에 **C:₩Program Files₩Maven₩apache-maven-3.9.6₩bin₩mvn.층**
- VS Code 사이드바로 돌아가서, Maven Project를 우클릭, **Run Maven Commands -> install** 하면 **jar로 빌드**

Execute Jar

- 빌드 성공 후 **target** 폴더를 확인해보면 **jar 파일이 생성됨**

```
[INFO] Installing c:\Users\park\source\javacode\aj24\spring-boot-hello-controller\target\spring-boot-hello-co
ntroller-0.0.1-SNAPSHOT.jar to C:\Users\park\.m2\repository\kr\ac\dankook\ace\spring-boot-hello-controller\0.
0.1-SNAPSHOT\spring-boot-hello-controller-0.0.1-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  6.411 s
[INFO] Finished at: 2024-03-23T21:44:52+09:00
[INFO] ------------------------------------------------------------------------
```

- 콘솔창에서 java -jar projectname-0.0.1-SNAPSHOT.jar 실행

```
C:\Users\park\source\javacode\aj24\spring-boot-hello-controller\target>java -jar spring-boot-hello-controller-0.0.1-SNAP
SHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.2)

2024-03-23T21:46:27.422+09:00  INFO 19012 --- [           main] a.s.SpringBootHelloControllerApplication : Starting Spri
ngBootHelloControllerApplication v0.0.1-SNAPSHOT using Java 21.0.2 with PID 19012 (C:\Users\park\source\javacode\aj24\sp
ring-boot-hello-controller\target\spring-boot-hello-controller-0.0.1-SNAPSHOT.jar started by park in C:\Users\park\sourc
e\javacode\aj24\spring-boot-hello-controller\target)
```

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

Thymeleaf

- 변수

```
Current time is <span th:text="${serverTime}" />
```

- 선택 변수

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

- 메시지

```
<span th:text="#{welcome.message}" />
```

- 링크

```
<a href="#" th:href="@{/sample/say}">Return to sample</a>
```

- Literals

```
Look at a <span th:text="'web application'">template file</span>
<p>This year is <span th:text="2024">2024</span></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>
```

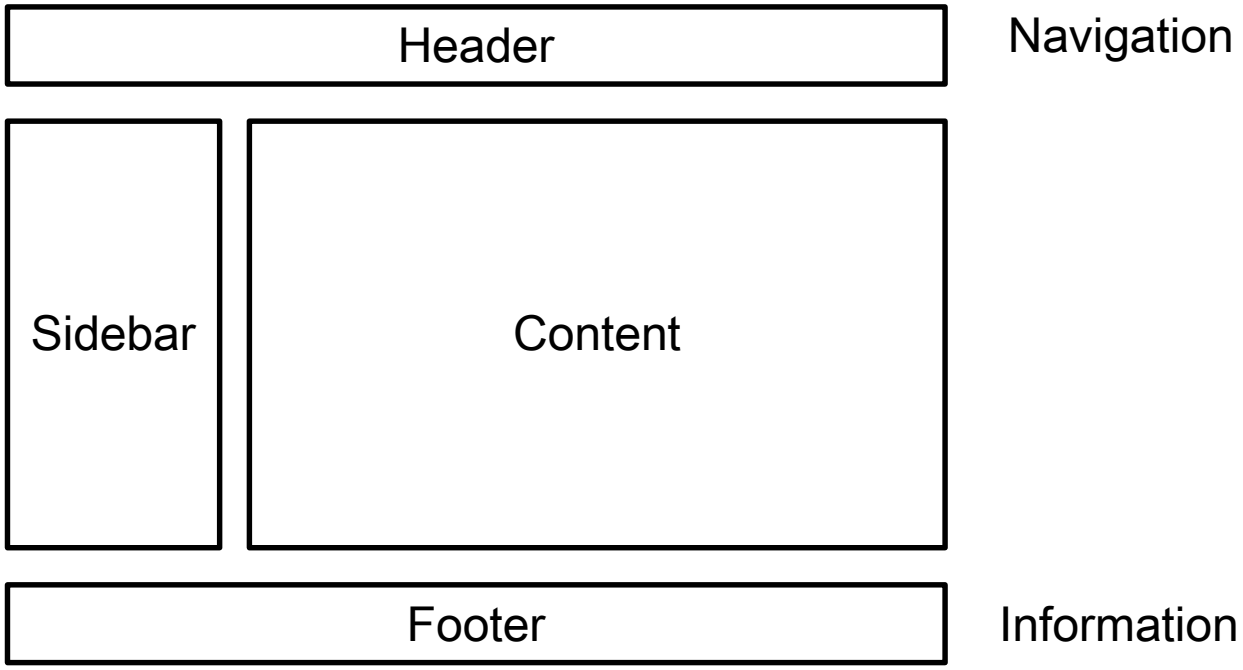
Thymeleaf

- 반복문 표현
  - 배열이나 컬랙션 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>AJ24 Header</title>
    <!-- Add your CSS stylesheets or links here -->
</head>
<body>
    <header>
        <!-- Header content goes here -->
        <h1>AJ24 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; 2024 AJ24 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 AJ24 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 – hello.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>Hello!</h1>
                    <p th:text="|Hello! ${person}!|" />
                </div>
            </div>
        </div>
    </div>
    <footer th:replace="~{fragments/footer :: footer}"></footer>
</body>
</html>
```

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 SpringBootViewlayoutController {
    @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
    }
}
```