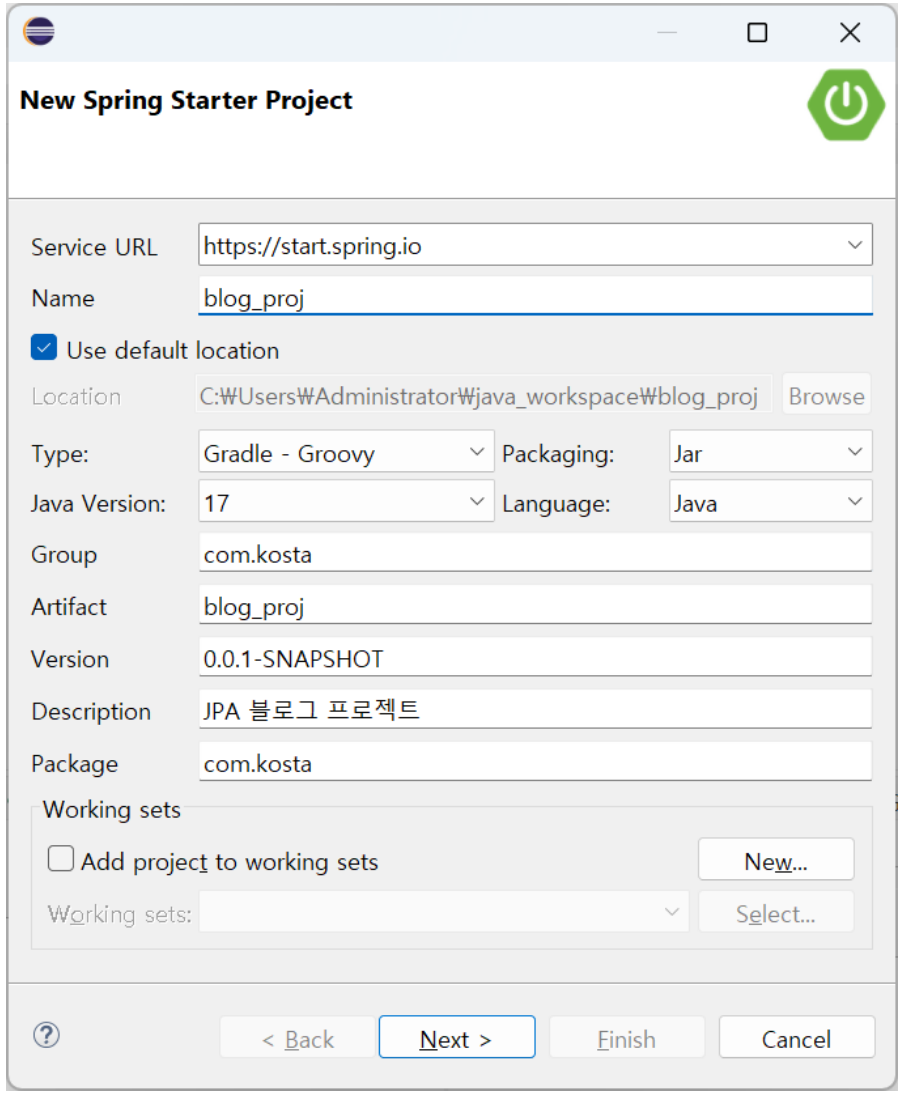

spring

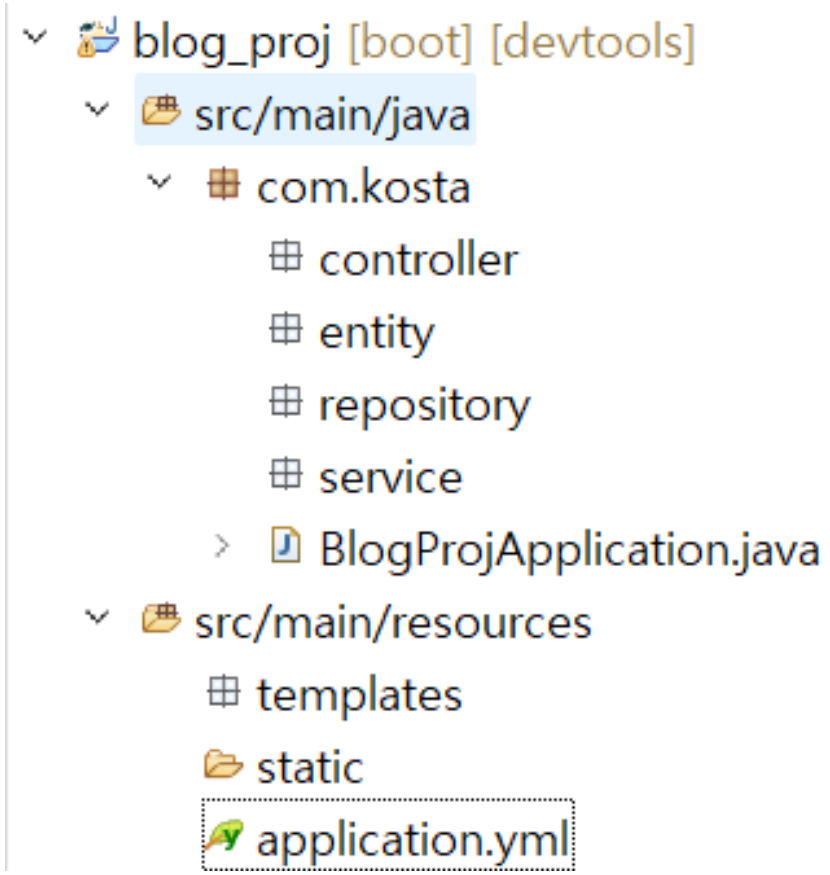
JPA 프로젝트 (블로그)



- Spring Web
- Spring DevTools
- Spring Data JPA
- MySQL Driver
- Lombok
- thymeleaf

JPA 프로젝트 (블로그)

- 프리젠테이션 계층 : 웹 클라이언트의 요청 및 응답을 처리 (@Controller)
- 서비스 계층 : 비즈니스 로직 처리 및 도메인 모델의 적합성을 검증 (@Service)
- 퍼시스턴트 계층 : 데이터 처리를 담당 (@Repository)
- 도메인 모델 : 데이터베이스의 엔티티, VO, DTO (@entity)



JPA 프로젝트 (블로그)

```yaml
spring:
    application:
        name: blog_proj
    datasource:
        driver-class-name: com.mysql.cj.jdbc.Driver
        url: jdbc:mysql://localhost:3306/blog_db
        username: root
        password: 1234
    jpa:
        database: mysql
        # 자동으로 테이블 생성과 같은 스크립트 실행 (실제는 false로 변경)
        generate-ddl: true
        show-sql: true
        open-in-view: false
    sql:
        init:
            mode: never
    mvc:
        hiddenmethod:
            filter:
                enabled: true
```

JPA 프로젝트 (블로그)

- 엔티티 구성 (테이블 구조)

컬럼명	자료형	null 허용	키	설명
id	BIGINT	X	기본키	게시물 일련변호
title	VARCHAR(255)	X		게시물 제목
content	VARCHAR(255)	X		게시물 내용

JPA 프로젝트 (블로그)

- 엔티티 구성(테이블구조)

```
package com.kosta.entity;

@Entity // 엔티티 지정
@RequiredArgsConstructor
@Data
public class Article {
	@Id // 기본키 지정
	@GeneratedValue(strategy = GenerationType.IDENTITY) // 자동 증가
	@Column(name="id", updatable = false) // update 시에 컬럼에 포함하지 않음
	private Long id;

	@Column(name="title", nullable = false)
	private String title;

	@Column(name="content", nullable = false)
	private String content;

	@Builder // 빌더 패턴으로 객체 생성
	public Article(Long id, String title, String content) {
		this.id = id;
		this.title = title;
		this.content = content;
	}
}
```

빌더 패턴

- 빌더 패턴 방식은 객체를 보다 유연하고 직관적으로 생성할 수 있도록 해주기 때문에 개발자들이 많이 애용하는 디자인 패턴이다.

- 빌더 패턴을 사용하면 어느 필드에 어떤 값이 들어가는지 명시적인 파악이 가능하다.

```
new Article("제목", "내용");


Article.builder()
    .title("제목")
    .content("내용")
    .build();
```

JPA 프로젝트 (블로그)

- 리포지터리 생성

```
package com.kosta.repository;

@Repository
public interface BlogRepository extends JpaRepository<Article, Long>{

}
```

JPA 프로젝트 (블로그)

- 테스트코드작성

```
package com.kosta.repository;

@DataJpaTest // JPA 관련 테스트를 위한 애너테이션
@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE) // 실제 데이터베이스 사용
public class BlogRepositoryTest {

	@Autowired
	private BlogRepository blogRepository;

	@Test
	void testSaveAndFindArticle() {
		// Given
		Article article = Article.builder().title("Test Title").content("Test Content").build();

		// When
		Article savedArticle = blogRepository.save(article);

		// Then
		assertThat(savedArticle).isNotNull();
		assertThat(savedArticle.getId()).isNotNull();
		assertThat(savedArticle.getTitle()).isEqualTo("Test Title");
		assertThat(savedArticle.getContent()).isEqualTo("Test Content");
	}
}
```

JPA 프로젝트 (블로그)

- 서비스생성

```java
package com.kosta.service;

public interface BlogService {
    public Article save(Article article);}
```

```java
package com.kosta.service;

@Service
@RequiredArgsConstructor // final 키워드나 @NotNull이 붙은 필드로 생성자 만듦
public class BlogServiceImpl implements BlogService {
    private final BlogRepository blogRepository;

    @Override
    public Article save(Article article) {
        return blogRepository.save(article);
    }
}
```

JPA 프로젝트 (블로그)

- 컨트롤러 생성

```
package com.kosta.controller;

@Controller
@RequiredArgsConstructor
public class BlogController {
    private final BlogService blogService;

    @GetMapping("/add")
    public String formArticle() {
        return "add";
    }

    @PostMapping("/add")
    public String addArticle(Article article) {
        Article savedArticle = blogService.save(article);
        return "redirect:/list";
    }
}
```

JPA 프로젝트 (블로그)

- src/main/resources/templates/add.html

```
<form action="/add" method="POST">
    <input name="title" placeholder="title" />
    <textarea name="content" placeholder="content"></textarea>
    <button>저장</button>
</form>
```

JPA 프로젝트 (블로그)

- 테스트코드작성

```
@Test
void articleListTest() {
    // Given
    Article article1 = Article.builder().title("Title 1").content("Content 1").build();
    Article article2 = Article.builder().title("Title 2").content("Content 2").build();
    blogRepository.save(article1);
    blogRepository.save(article2);

    // When
    List<Article> list = blogRepository.findAll();

    // Then
    assertThat(list).isNotNull();
    assertThat(list.size()).isGreaterThan(0); // 엔티티가 적어도 1개 이상 존재해야 함
    assertThat(list.stream().anyMatch(article -> article.getTitle().equals("Title 1"))).isTrue();
    assertThat(list.stream().anyMatch(article -> article.getTitle().equals("Title 2"))).isTrue();
}
```

JPA 프로젝트 (블로그)

- 서비스 작성

```
@Service
@RequiredArgsConstructor // final 키워드나 @NotNull이 붙은 필드로 생성자 만듦
public class BlogServiceImpl implements BlogService {
    private final BlogRepository blogRepository;
    @Override
    public Article save(Article article) {
        return blogRepository.save(article);
    }

    @Override
    public List<Article> findAll() {
        return blogRepository.findAll();
    }
}
```

JPA 프로젝트 (블로그)

- 컨트롤러 작성

```
@GetMapping("/list")
public String articleList(Model model) {
    List<Article> articleList = blogService.findAll();
    model.addAttribute("list", articleList);
    return "list";
}
```

JPA 프로젝트 (블로그)

- src/main/resources/templates/list.html

```
<p th:each="article : ${list}">
    <a th:href="@{'/detail/'+${article.id}}">
        [[${article.title}]]: [[${article.content}]]
    </a>
</p>
```

JPA 프로젝트 (블로그)

- 테스트코드작성

```
@Test
void findArticleById() {
    // Given
    Article article = Article.builder().title("Title 1").content("Content 1").build();
    Article savedArticle = blogRepository.save(article);

    // When
    Article foundArticle = blogRepository.findById(savedArticle.getId()).get();

    // Then
    assertThat(foundArticle).isNotNull();
    assertThat(foundArticle.getId()).isEqualTo(savedArticle.getId());
    assertThat(foundArticle.getTitle()).isEqualTo(savedArticle.getTitle());
    assertThat(foundArticle.getContent()).isEqualTo(savedArticle.getContent());
}
```

JPA 프로젝트 (블로그)

- 서비스 작성

```
@Override
public Article findById(Long id) throws Exception {
    Article article = blogRepository.findById(id)
        .orElseThrow(()-> new Exception("없는 아이디"));
    return article;
}
```

JPA 프로젝트 (블로그)

- 컨트롤러 작성

```
@GetMapping("/detail/{id}")
public String articleDetail(@PathVariable("id") Long id, Model model) {
    try {
        Article article = blogService.findById(id);
        model.addAttribute("article", article);
        return "detail";
    } catch (Exception e) {
        model.addAttribute("error", e.getMessage());
        return "error";
    }
}
```

JPA 프로젝트 (블로그)

- src/main/resources/templates/detail.html

```
[[${article.title}]]: [[${article.content}]]

<form th:action="@{'/delete/' + ${article.id}}" th:method="DELETE">
    <button>삭제</button>
</form>
```

JPA 프로젝트 (블로그)

- src/main/resources/templates/error.html

```
<p>[[ ${error ?: '에러발생' }]]</p>
```

JPA 프로젝트 (블로그)

- 테스트코드작성

```
@Test
void deleteArticleById() {
    // Given
    int originSize = blogRepository.findAll().size();
    Article article = Article.builder().title("Title 1").content("Content 1").build();
    Article savedArticle = blogRepository.save(article);

    // When
    blogRepository.deleteById(savedArticle.getId());
    int newSize = blogRepository.findAll().size();

    // Then
    assertThat(originSize).isEqualTo(newSize);
}
```

JPA 프로젝트 (블로그)

- 서비스 작성

```
@Override
public void deleteById(Long id) {
    blogRepository.deleteById(id);
}
```

JPA 프로젝트 (블로그)

- 컨트롤러 작성

```
@DeleteMapping("/delete/{id}")
public String deleteArticle(@PathVariable("id") Long id, Model model) {
    blogService.deleteById(id);
    return "redirect:/list";
}
```

JPA 프로젝트 (블로그)

- src/main/resources/templates/detail.html

```
[[${article.title}]]: [[${article.content}]]

<form th:action="@{'/delete/' + ${article.id}}" th:method="DELETE">
    <button>삭제</button>
</form>

<a th:href="@{'/modify/' + ${article.id}}">수정</a>
```

JPA 프로젝트 (블로그)

- 컨트롤러 작성

```
@GetMapping("/modify/{id}")
public String formArticleModify(@PathVariable("id") Long id, Model model) {
    try {
        Article article = blogService.findById(id);
        model.addAttribute("article", article);
        return "add";
    } catch (Exception e) {
        model.addAttribute("error", e.getMessage());
        return "error";
    }
}
```

JPA 프로젝트 (블로그)

- src/main/resources/templates/add.html

```
<form
    th:action="${article ne null ? '/modify' : '/add'}"
    th:method="${article ne null ? 'PATCH' : 'POST'}">
        <input type="hidden" name="id"
            th:if="${article ne null}" th:value="${article.id}" />
        <input name="title" placeholder="title"
            th:value="${article ne null ? article.title : ''}" />
        <textarea name="content" placeholder="content">
            [[ ${article ne null ? article.content : ''} ]]
        </textarea>

        <button>저장</button>
</form>
```

JPA 프로젝트 (블로그)

- 테스트코드작성

```
@Test
void updateArticle() {
    // Given
    Article article = Article.builder().title("Title 1").content("Content 1").build();
    Article savedArticle = blogRepository.save(article);
    // When
    Article foundArticle = blogRepository.findById(savedArticle.getId()).get();
    foundArticle.setTitle("변경");
    Article changedArticle = blogRepository.findById(savedArticle.getId()).get();


    // Then
    assertThat(foundArticle.getTitle()).isEqualTo(changedArticle.getTitle());
}
```

JPA 프로젝트 (블로그)

- 서비스 작성

```
@Override
public Article updateArticle(Article article) throws Exception {
    Article exArticle = blogRepository.findById(article.getId())
        .orElseThrow(() -> new Exception("없는 아이디"));

    exArticle.setTitle(article.getTitle());
    exArticle.setContent(article.getContent());

    Article updatedArticle = blogRepository.save(exArticle);
    return updatedArticle;
}
```

JPA 프로젝트 (블로그)

- 컨트롤러 작성

```
@PatchMapping("/modify")
public String modifyArticle(Article article, Model model) {
    try {
        Article updatedArticle = blogService.updateArticle(article);
        return "redirect:/list";
    } catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("error", e.getMessage());
        return "error";
    }
}
```