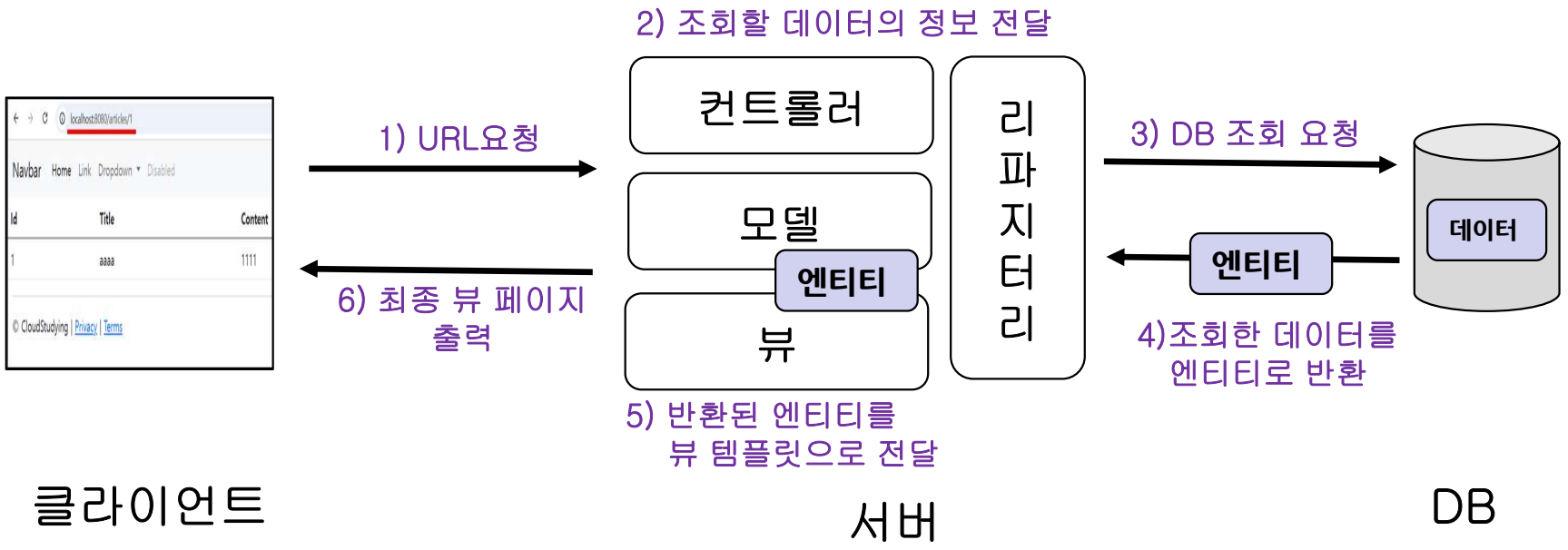
# 5장 게시글 읽기: Read

출처: 코딩 자율학습 스프링부트3 자바 백엔드 개발 입문, 홍팍, 길벗, 2023

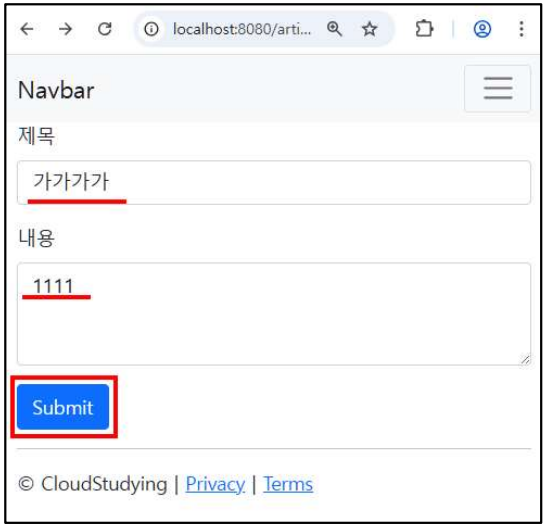
# 데이터 조회 과정

1) 사용자가 데이터를 조회 요청(웹 페이지에서 URL 요청을 보냄)
2) 서버의 컨트롤러가 요청을 받아 해당 URL에서 찾으려는 데이터 정보를 리파지토리에 전달
3) 리파지터리는 정보를 가지고 DB에 데이터 조회를 요청
4) DB는 해당 데이터를 찾아 이를 엔티티로 반환
5) 반환된 엔티티는 모델을 통해 뷰 템플릿으로 전달
6) 최종적으로 결과 뷰 페이지가 완성돼 사용자의 화면에 출력

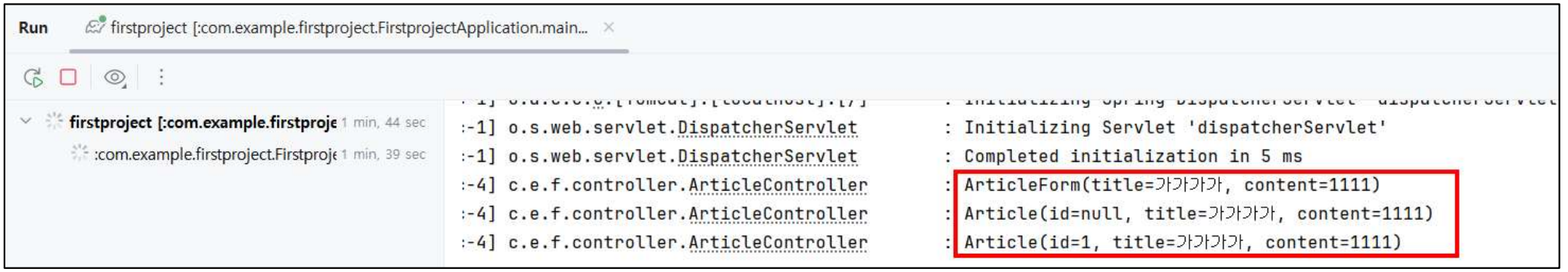


# 단일 데이터 조회하기

- **게시글 작성**
  - localhost:8080/articles/new 접속




  - 로그 확인



# 단일 데이터 조회하기

- **URL 요청 받기**
  - 게시글의 'id'를 사용해, 게시글을 조회할 때의 URL
    - ✓ localhost:8080/articles/id
      - 예: localhost:8080/articles/1,
        localhost:8080/articles/2,
        …
        Localhost:8080/articles/100,
        …

# 단일 데이터 조회하기

- **URL 요청을 받을 컨트롤러 만들기**
  - [com.example.firstproject - controller - ArticleController] 열기
  - @ Gettmaping("/articles/{id}") 어노테이션 작성
    - ✓ {변수명} : URL에서 변수 사용
  - URL요청을 받아 처리할 show(@PathVariable Long id) 메서드
    - ✓ @PathVariable
      - URL 요청으로 들어온 전달 값을 컨트롤러의 매개변수로 가져옴

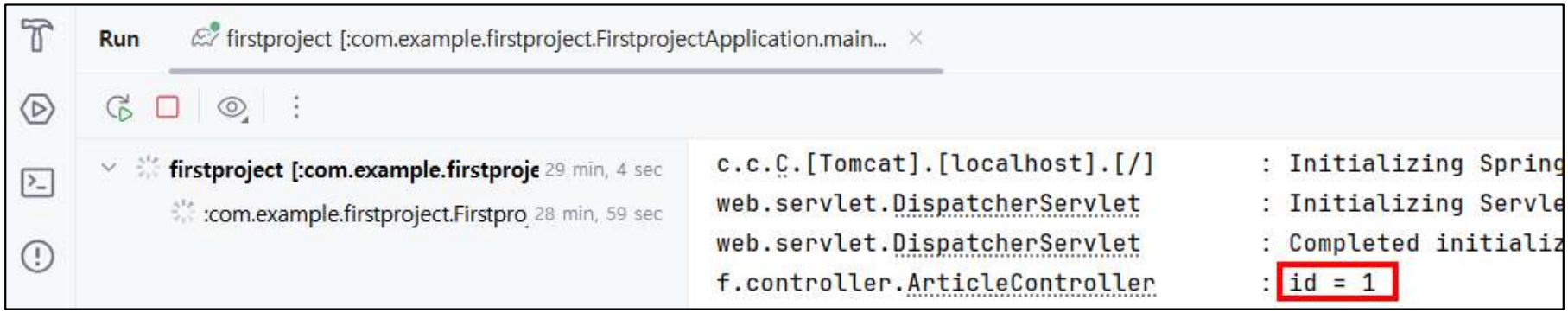
ArticleController.java

```
import org.springframework.web.bind.annotation.PathVariable;

                (중략)

public class ArticleController {

                (중략)

    @GetMapping("/articles/{id}")  //데이터 조회 요청 접수
    public String show(@PathVariable Long id){ // 매개변수로 id 받아오기
        log.info("id = " + id); //id를 잘 받았는지 확인하는 로그 찍기
        return "";
    }
}
```

# 단일 데이터 조회하기

- **Id를 잘 받았는지 점검**
  - 서버 재시작
  - localhost:8080/articles/1 접속


  - 로그 확인



# 데이터 조회해 출력하기

- **작업순서**
  1. Id를 조회해 DB에서 해당 데이터 가져오기
  2. 가져온 데이터를 모델에 등록하기
  3. 조회한 데이터를 사용자에게 보여 주기 위한 뷰 페이지 만들고 반환하기

ArticleController.java

```
@GetMapping("/articles/{id}")
public String show(@PathVariable Long id){
    log.info("id = " + id);
    // 1. id를 조회해 데이터 가져오기
    // 2. 모델에 데이터 등록하기
    // 3. 뷰 페이지 반환하기
    return "";
}
```

# 데이터 조회해 출력하기

- **Id를 조회해 데이터 가져오기**
  - DB에서 해당 id의 데이터를 가져오는 객체는 리파지토리
    - ✓ 3장에서 @Autowired로 주입 받은 articleRepository 객체 사용
  - Optional<T> findById(Long id)
    - ✓ JPA의 CrudRepository가 제공하는 메소드
    - ✓ 타입 매개변수 T: Article 엔티티

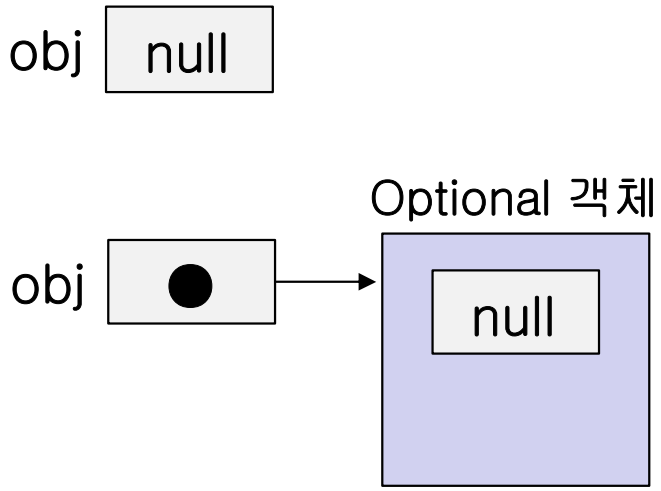
```
// 1. id를 조회해 데이터 가져오기
Optional<Article> articleEntity = articleRepository.findById(id);
```

```
// 1. id를 조회해 데이터 가져오기
Article articleEntity = articleRepository.findById(id).orElse(null);
```

# Optional<T> 클래스

- **간접적으로 null을 다루기 위한, Wrapper 클래스**

```
public final class Optional<T> {
      private final T value;      //모든 종류의 객체 저장, null
      …
}
```




  - null을 직접 다룰 때 문제점
    - ✓ NullPointerException 발생이 잦음
    - ✓ null을 체크하기 위해 if문 사용으로 코드가 지저분 해짐

```
Object  obj = getResult();     //리턴값은 null 또는 객체

 obj.toString();                    // null 이면, NullPointerException 발생

 if(obj != null) {
       obj.toString();
}
```

# Optional<T> 클래스

- **Optional<T> 사용 예**
  - Optional 객체 만들기
    - ✓ 값이 null 일 때, 비여 있는 Optional<T> 객체 사용

```
String str = null;   // 바람 직 하지 않음
Optional<String>  optVal = Optional.empty();  // 빈 객체로 초기화
```

    - ✓ 값이 null이 아닐 때

```
String  str = "hello";
Optional<String>  optVal1 = Optional.of(str);
```

    - ✓ 값이 null일 수도 아닐수도 있을 때

```
Optional<String>  optVal2 = Optional.ofNullable(str) ;
```

# Optional<T> 클래스

- **Optional<T> 사용 예**
  - Optional 객체의 값 가져오기: get(), orElse(), orElseGet(), orElseThrow()

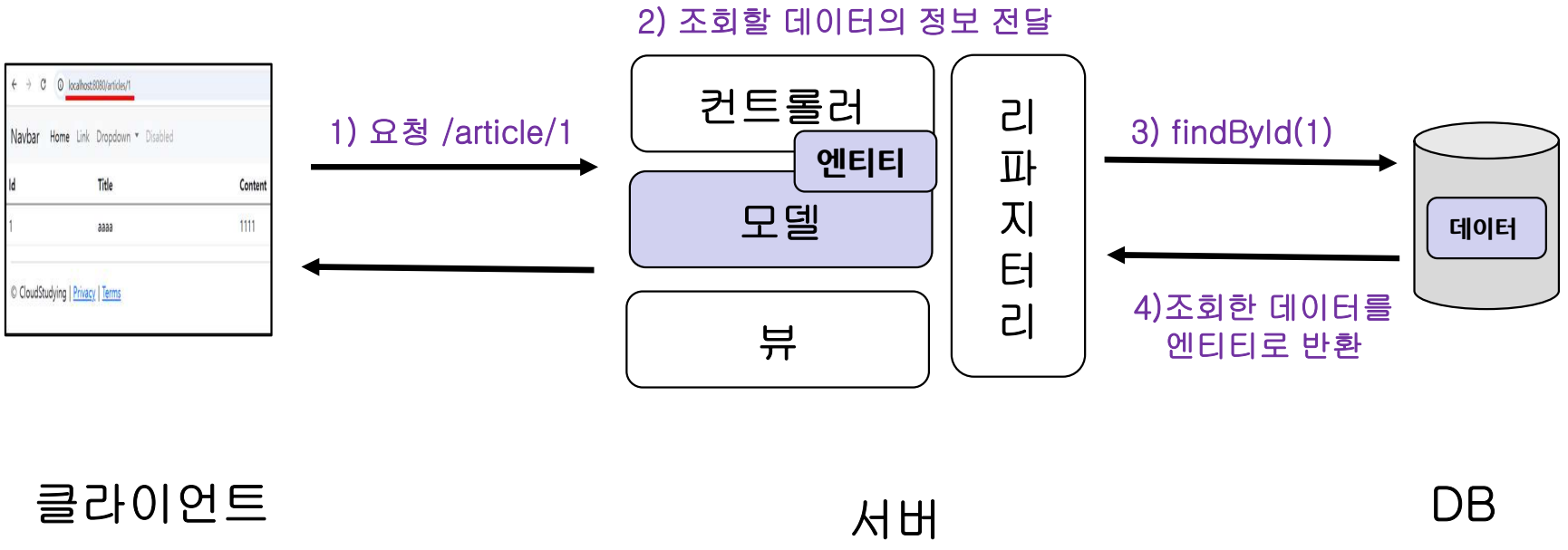
```
String str1 = optVal1.orElse("");  //optVal1에 저장된 값이 null일 때, ""반환
String str2 = optVal1.orElseGet(() -> new String());   //람다식 사용가능
```

  - ifPresent()
    - ✓ Optional 객체의 값이 null이면 false, 아니면, true 반환

```
if(Optional.ofNullable(str).isPresent()) {        // if(str!=null)
        System.out.println(str);
}
```

# 데이터 조회해 출력하기

- **모델에 데이터 등록하기**
  - articleEntity에 담긴 데이터를 모델에 등록하기



# 데이터 조회해 출력하기

- **추가 단계 : Article 엔티티에 기본 생성자 추가**
  - @NoArgsConstructor 추가

Article.java

```
import lombok.NoArgsConstructor;
      (중략)
@NoArgsConstructor
@AllArgsConstructor
@ToString
@Entity
public class Article {
    @Id
    @GeneratedValue
    private Long id;
    @Column
    private String title;
    @Column
    private String content;
}
```

```
public Article () {

}
```

# 데이터 조회해 출력하기

- **모델에 데이터 등록하기**
  - show() 메서드의 매개변수로 model 객체를 받아 옴
  - 모델에 addAtrribute() 메서드로 데이터 등록
    - ✓ 형식

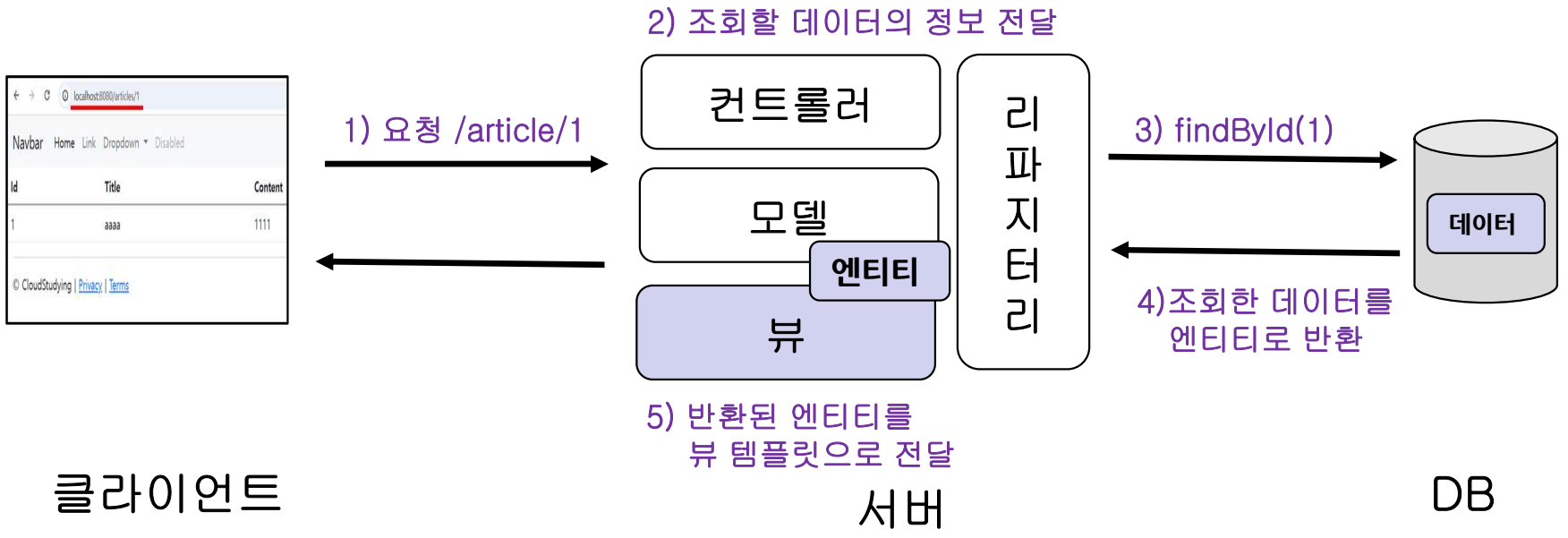
```
model.addAttribute("변수명", 변숫값);

model.addAttribute(String name, Object value);
```

ArticleController.java

```
@GetMapping("/articles/{id}")
public String show(@PathVariable Long id, Model model){
    log.info("id = " + id);
    // 1. id를 조회해 데이터 가져오기
    Article articleEntity = articleRepository.findById(id).orElse( other: null);
    // 2. 모델에 데이터 등록하기
    model.addAttribute( attributeName: "article", articleEntity);
    // 3. 뷰 페이지 반환하기
    return "";
}
```

# 데이터 조회해 출력하기

- **뷰 페이지 반환하기**




  - 컨트롤러에서 뷰 페이지 반환

```
@GetMapping("/articles/{id}")
public String show(@PathVariable Long id, Model model){
        (중략)

    // 3. 뷰 페이지 반환하기
    return "articles/show";
}
```

# 데이터 조회해 출력하기

- **뷰 페이지 반환하기**
  - [resources - templates - articles ]에서 마우스 오른쪽 버튼 클릭
  - 컨텍스트 메뉴에서 [New - File] 선택
    - ✓ 새 파일명: show.mustache
  - show.mustache 파일에 header와 footer 넣기

show.mustache

```
{{>layouts/header}}


{{>layouts/footer}}
```

# 데이터 조회해 출력하기

- **뷰 페이지 반환하기**
  - 헤더와 푸터 사이에 table을 삽입
    - ✓ getbotstrap.com(v5.0.2)에서 검색창에 table 검색
    - ✓ 검색된 페이지에서 기본 형식의 테이블 선택한 후, [Copy] 클릭
    - ✓ show.mustache 파일의 헤더와 푸터 사이에 붙여넣기

getbootstrap.com

## Overview

Due to the widespread use of `<table>` elements across third-party widgets like calendars and date pickers, Bootstrap's tables are **opt-in**. Add the base class `.table` to any `<table>`, then extend with our optional modifier classes or custom styles. All table styles are not inherited in Bootstrap, meaning any nested tables can be styled independent from the parent.

Using the most basic table markup, here's how `.table`-based tables look in Bootstrap.

| # | First | Last | Handle |
|---|---|---|---|
| 1 | Mark | Otto | @mdo |
| 2 | Jacob | Thornton | @fat |
| 3 | Larry the Bird | | @twitter |

Copy

```
<table class="table">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First</th>
      <th scope="col">Last</th>
      <th scope="col">Handle</th>
    </tr>
  </thead>
```

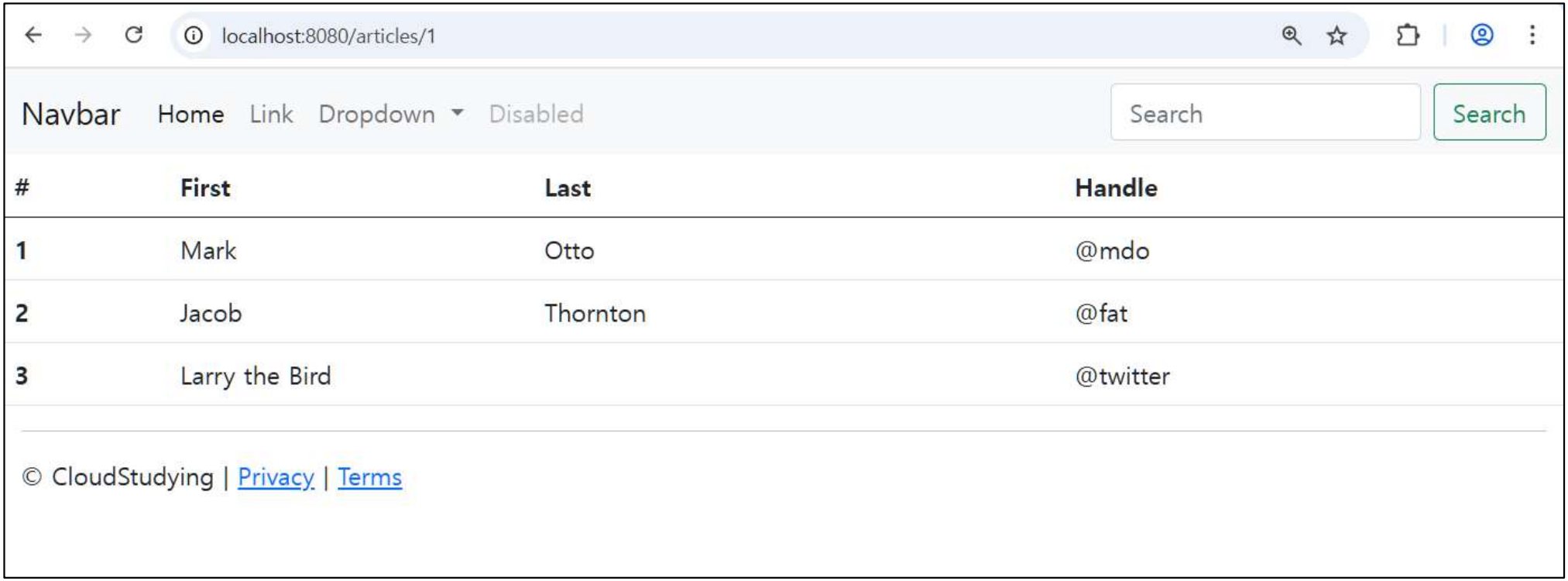
show.mustache

```
{{>layouts/header}}

<table class="table">
      (중략)
</table>

{{>layouts/footer}}
```

붙여넣기 (Ctrl + V)

# 데이터 조회해 출력하기

- **서버 재시작**
  - localhost:8080/articles/1
  - 표 확인



# 데이터 조회해 출력하기

- **뷰 페이지 반환하기**
  - show.mustache 편집
    - ✓ {{#변수명}} {{/변수명}}
      - 모델에서 가져 온 데이터를 템플릿에서 여러 줄에 걸쳐 사용할 때 범위 지정
      - 데이터를 어디부터 어디까지 사용할지 범위 지정
      - 예: {{#article}} {{/article}}

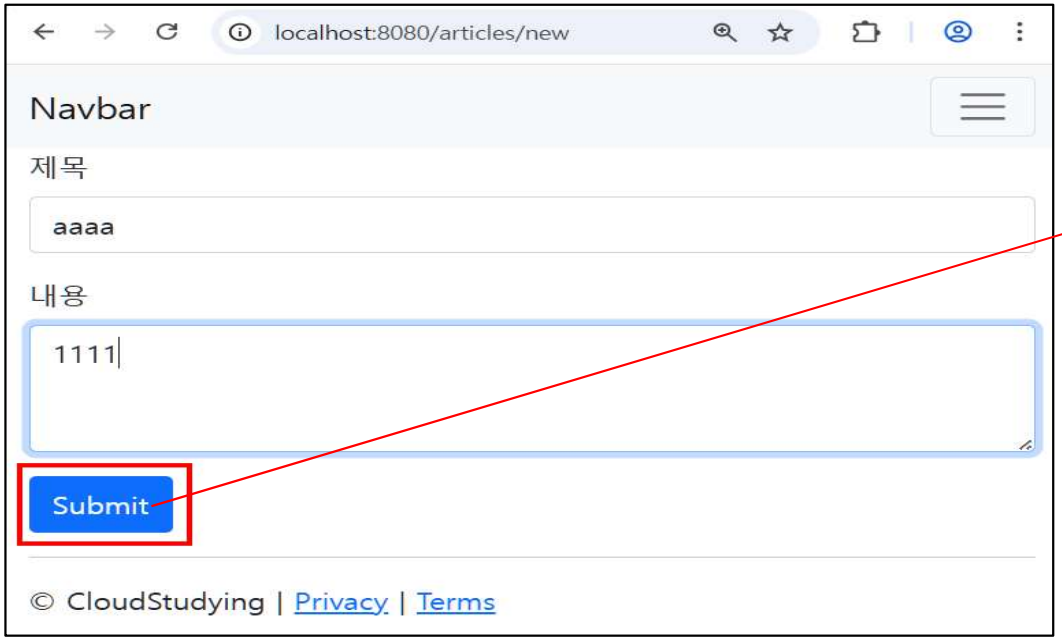
```
{{>layouts/header}}

<table class="table">
    <thead>
    <tr>
        <th scope="col">Id</th>
        <th scope="col">Title</th>
        <th scope="col">Content</th>
    </tr>
    </thead>
    <tbody>
    {{#article}}
    <tr>
        <td>{{id}}</td>
        <td>{{title}}</td>
        <td>{{content}}</td>
    </tr>
    {{/article}}
    </tbody>
</table>

{{>layouts/footer}}
```

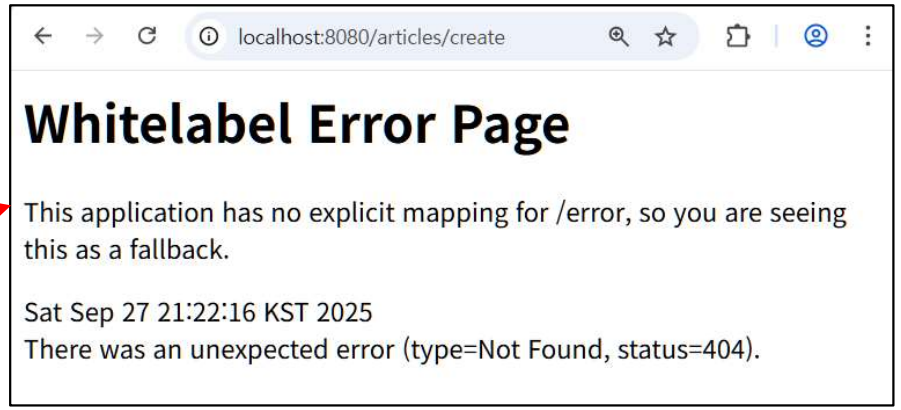
# 데이터 조회해 출력하기

- 서버 재실행
- localhost:8080/articles/1 접속

# 데이터 조회해 출력하기

- localhost:8080/articles/new 접속
  - 게시물 입력 후, [submit]



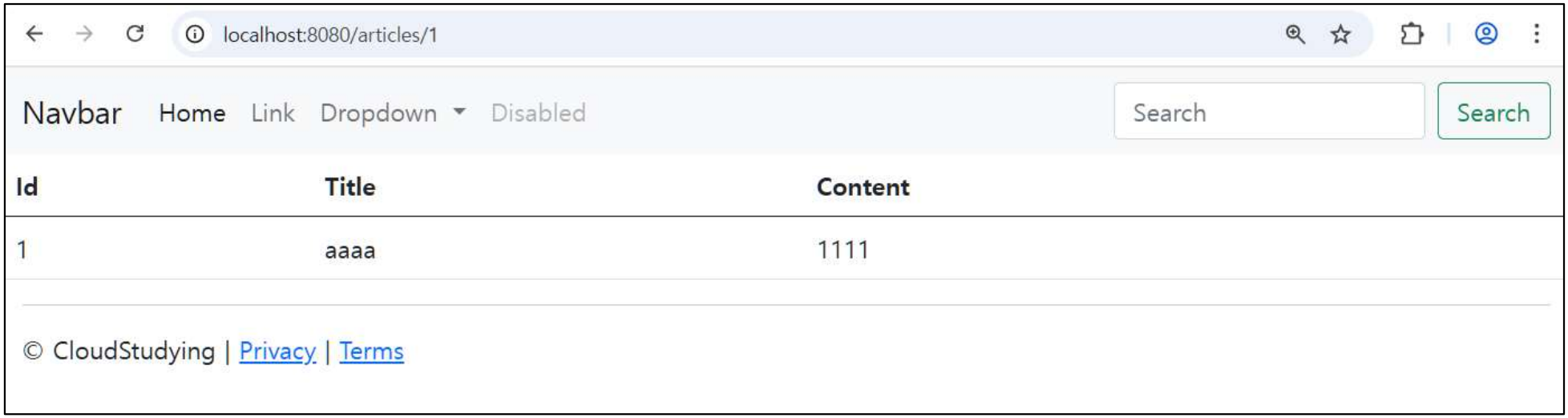




  - 로그 확인

```
.DispatcherServlet        : Completed initialization in 2 ms
r.ArticleController       : id = 1
r.ArticleController       : ArticleForm(title=aaaa, content=1111)
r.ArticleController       : Article(id=null, title=aaaa, content=1111)
r.ArticleController       : Article(id=1, title=aaaa, content=1111)
```

# 데이터 조회해 출력하기

- localhost:8080/articles/1 접속



# 데이터 목록 조회하기

- 데이터 목록을 조회할 때는 엔티티의 묶음인 리스트를 반환

# 데이터 목록 조회하기

- **URL 요청받기**
  - 데이터 목록 조회 URL: localhost:8080/articles
  - ArticleController에 index() 메서드 추가
  - index() 메서드 위에 @GetMapping("/articles") 선언

```
public Class ArticleController {
        (중략)
    @GetMapping("/articles/{id}")
    public String show(@PathVariable Long id, Model model){
        (중략)
    }

    @GetMapping("/articles")
    public String index() {
        return "";
    }

}
```

# 데이터 목록 조회하기

- **데이터 조회해 출력하기**
  - DB에서 모든 Article 데이터 가져오기
  - 가져온 Article 묶음을 모델에 등록하기
  - 사용자에게 보여 줄 뷰 페이지 설정하기

ArticleController.java

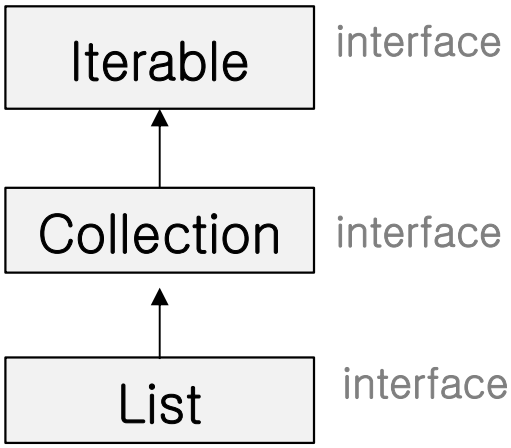
```
@GetMapping("/articles")
public String index() {
    //1. 모든 데이터 가져오기
    //2. 모델에 데이터 등록하기
    //3. 뷰 페이지 설정하기
    return "";
}
```

# 데이터 목록 조회하기

- **모든 데이터 가져오기**
  - CrudRepository의 findAll()
    - ✓ 테이블에 저장된 모든 엔티티들을 가져오기

```
Iterable<T>     findAll();
```

상속관계

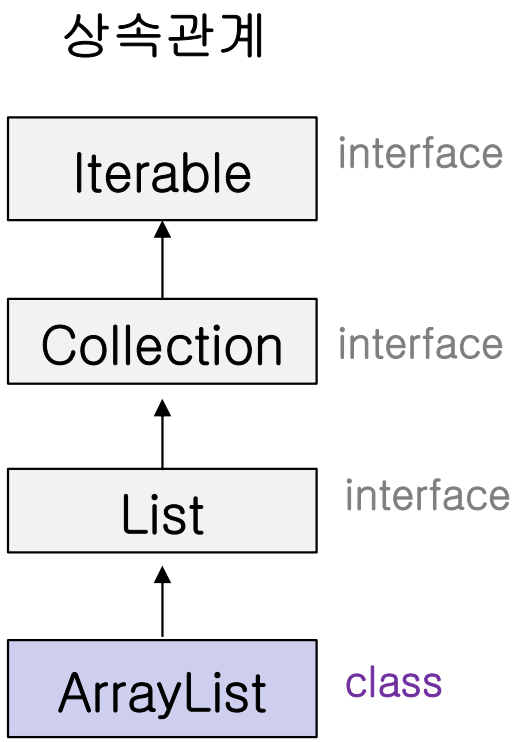



```
Iterable<Article> articleEntityList =  articleRepository.findAll();
```

```
List<Article> articleEntityList = (List<Article>) articleRepository.findAll();
```

# 데이터 목록 조회하기

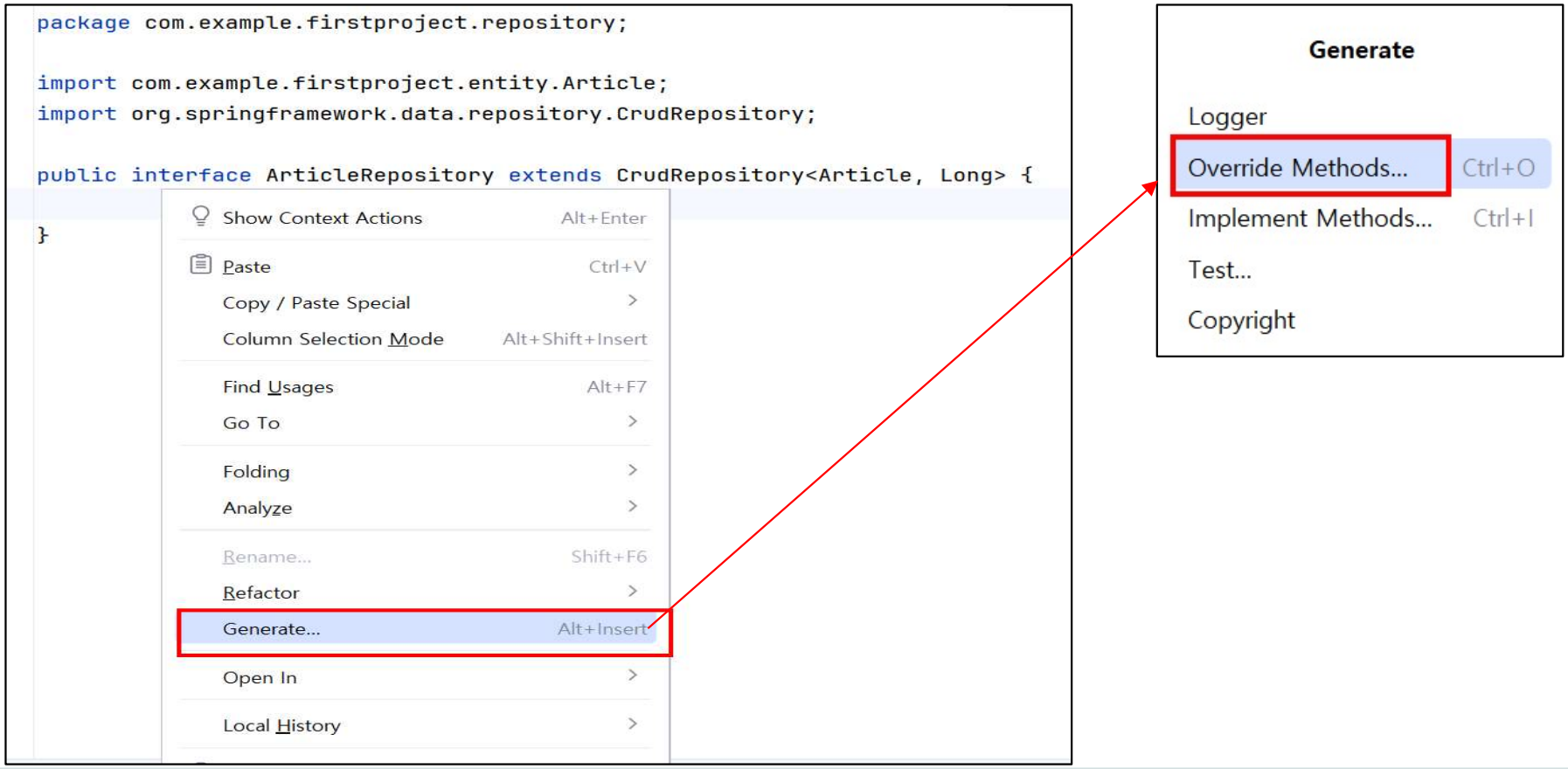
- **모든 데이터 가져오기**
  - CrudRepository의 findAll()이 ArrayList 타입을 반환하도록 ArticleRepository에서 Override 해서 사용




```
ArrayList<Article> articleEntityList = (List<Article>) articleRepository.findAll();
```

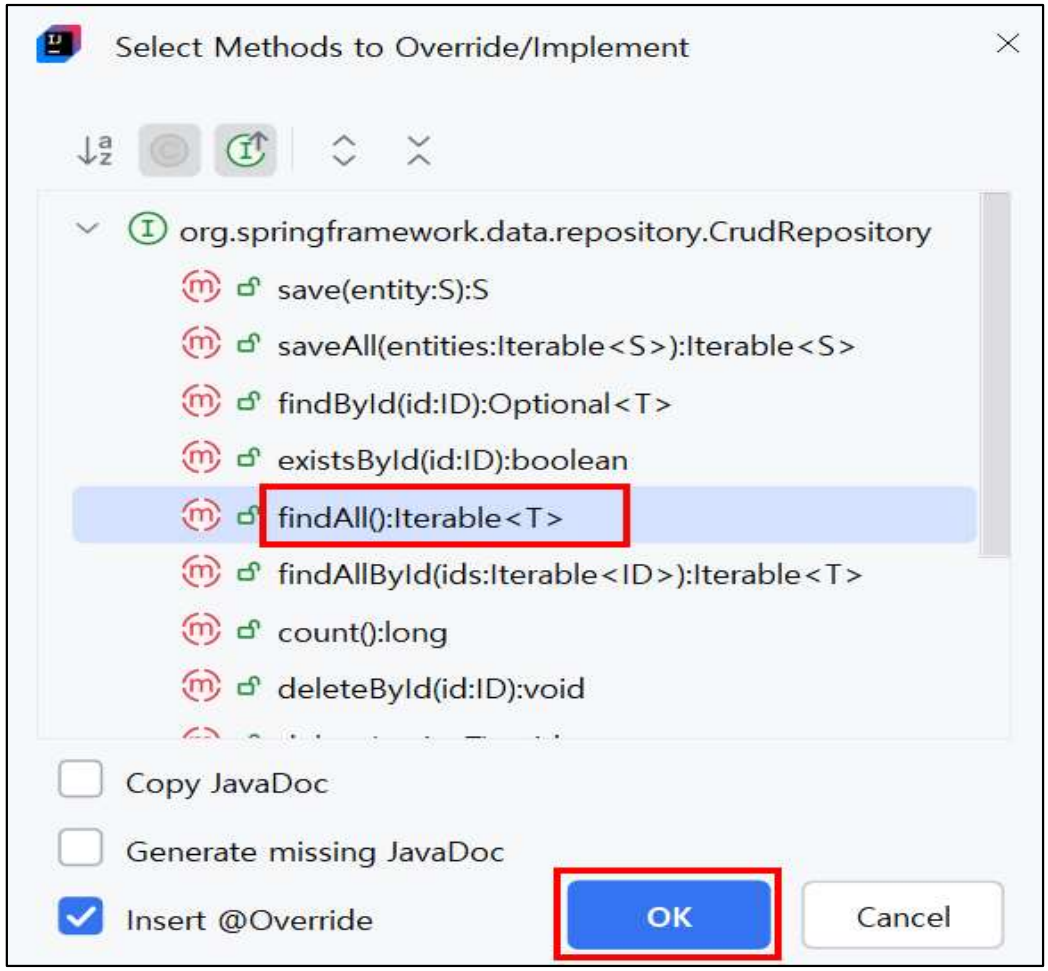
# 데이터 목록 조회하기

- **모든 데이터 가져오기**
  - [com.example.firstproject - repository - ArticleRepository] 열기
  - ArticleRepository 클래스 블록 안, 오버라이딩 메서드를 넣을 위치에서 마우스 오른쪽 버튼 클릭
  - 컨텍스트 메뉴에서 [Generate] 클릭
  - Generate 창에서 [Override Methods] 클릭



# 데이터 목록 조회하기

- **모든 데이터 가져오기**
  - 메서드 선택창에서 [findAll(): Iterable<T> 선택 후. [OK] 클릭



# 데이터 목록 조회하기

- **모든 데이터 가져오기**
  - findAll() 메서드 반환 타입을 ArrayList<Article>로 수정

```
package com.example.firstproject.repository;

import com.example.firstproject.entity.Article;
import org.springframework.data.repository.CrudRepository;

import java.util.ArrayList;

public interface ArticleRepository extends CrudRepository<Article, Long> {
    @Override
    ArrayList<Article> findAll();
}
```

# 데이터 목록 조회하기

- **모든 데이터 가져오기**
  - ArticleController의 index() 메서드 수정

ArticleController.java

```
@GetMapping("/articles")
public String index() {
    //1. 모든 데이터 가져오기
    ArrayList<Article> articleEntityList =  articleRepository.findAll();
    //2. 모델에 데이터 등록하기
    //3. 뷰 페이지 설정하기
    return "";
}
```

# 데이터 목록 조회하기

- **모델에 데이터 등록하기**
  - articleEntityList를 뷰 페이지로 전달

ArticleController.java

```
@GetMapping("/articles")
public String index(Model model) {
    //1. 모든 데이터 가져오기
    ArrayList<Article> articleEntityList =  articleRepository.findAll();
    //2. 모델에 데이터 등록하기
    model.addAttribute( attributeName: "articleList", articleEntityList);
    //3. 뷰 페이지 설정하기
    return "";
}
```

# 데이터 목록 조회하기

- **뷰 페이지 설정하기**
  - ArticleController의 index()에서 index.mustache를 뷰 페이지로 설정

ArticleController.java

```
@GetMapping("/articles")
public String index(Model model) {
    //1. 모든 데이터 가져오기
    ArrayList<Article> articleEntityList =  articleRepository.findAll();
    //2. 모델에 데이터 등록하기
    model.addAttribute( attributeName: "articleList", articleEntityList);
    //3. 뷰 페이지 설정하기
    return "articles/index";
}
```

# 데이터 목록 조회하기

- **뷰 페이지 설정하기**
  - [resources - templates - articles]에서 마우스 오른쪽 버튼 클릭
  - [New - File] 선택
    - ✓ 새 파일명: index.mustache
  - index.mustache 파일 편집

```
{{>layouts/header}}

<table class="table">
        (중략)
</table>

{{>layouts/footer}}
```

show.mustache 파일의
<table class="table">
...
</table>
를 복사한 후, 붙여넣기

# 데이터 목록 조회하기

- **뷰 페이지 설정하기**
  - index.mustache 파일에서
    {{#article}} … {{/article}}를 {{#articleList}} … {{/articleList}}로 수정

```
<table class="table">
    <thead>
    <tr>
        <th scope="col">Id</th>
        <th scope="col">Title</th>
        <th scope="col">Content</th>
    </tr>
    </thead>
    <tbody>
    {{#articleList}}
        <tr>
            <td>{{id}}</td>
            <td>{{title}}</td>
            <td>{{content}}</td>
        </tr>
    {{/articleList}}
    </tbody>
</table>
```

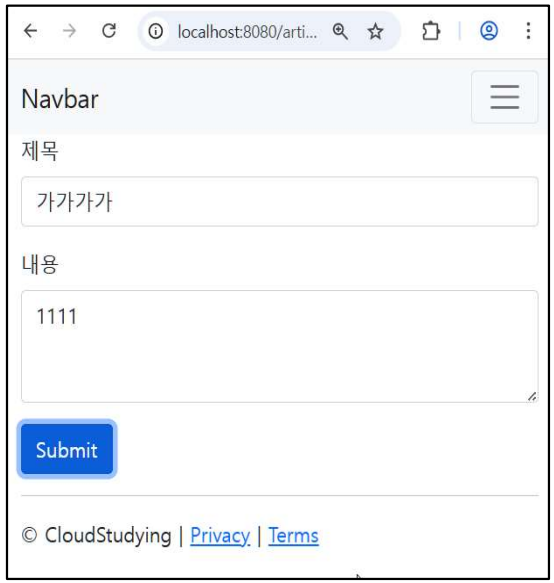
Mustache에서는 articleList가 데이터 묶음일 때, 내부 코드를 반복수행함

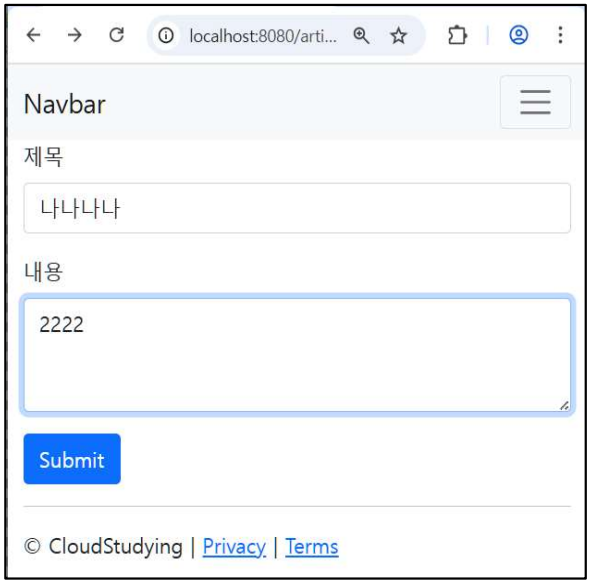
# 데이터 목록 조회하기

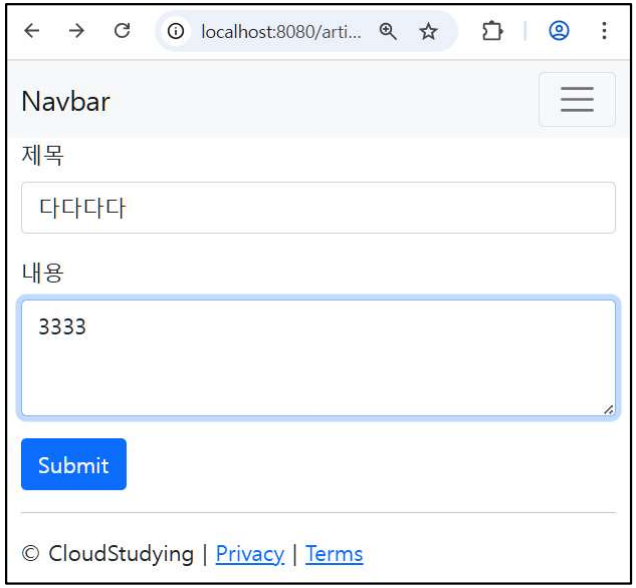
- 서버 재시작
- localhost:8080/articles 접속

# 데이터 목록 조회하기

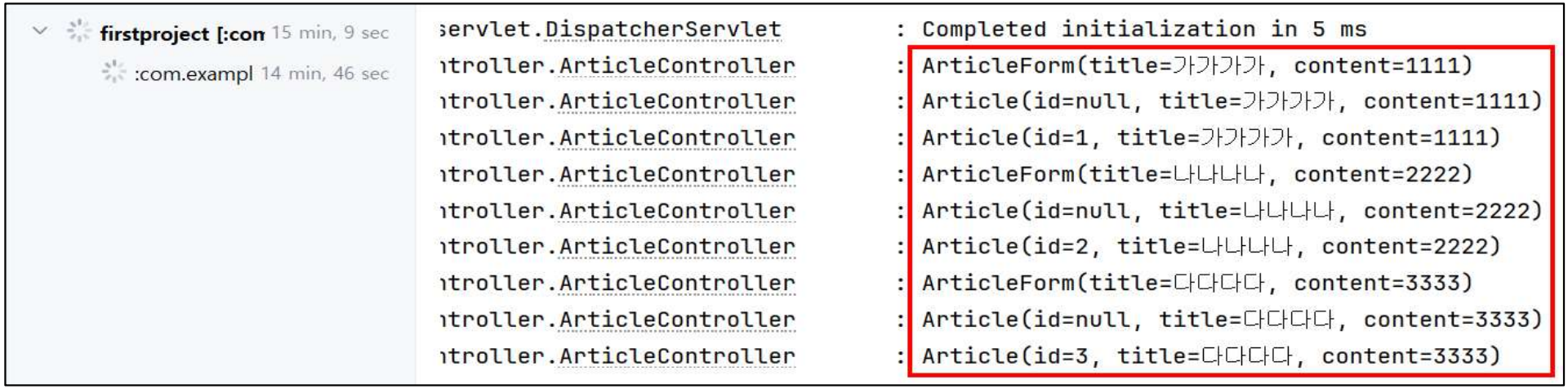
- localhost:8080/articles/new 접속 후, 데이터 3번 입력




- 로그 확인



# 데이터 목록 조회하기

- localhost:8080/articles 접속