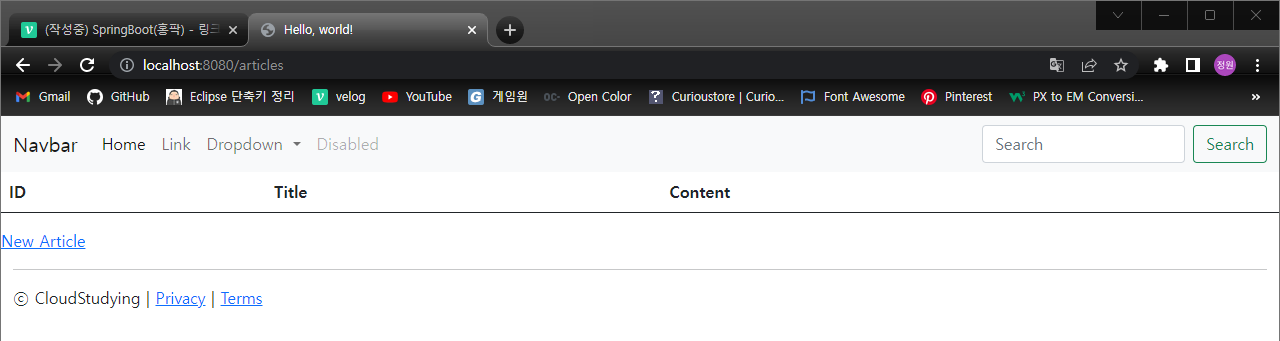
링크와 리다이렉트,수정 폼 만들기



**새 글작성 링크**

index.mustache에 링크추가

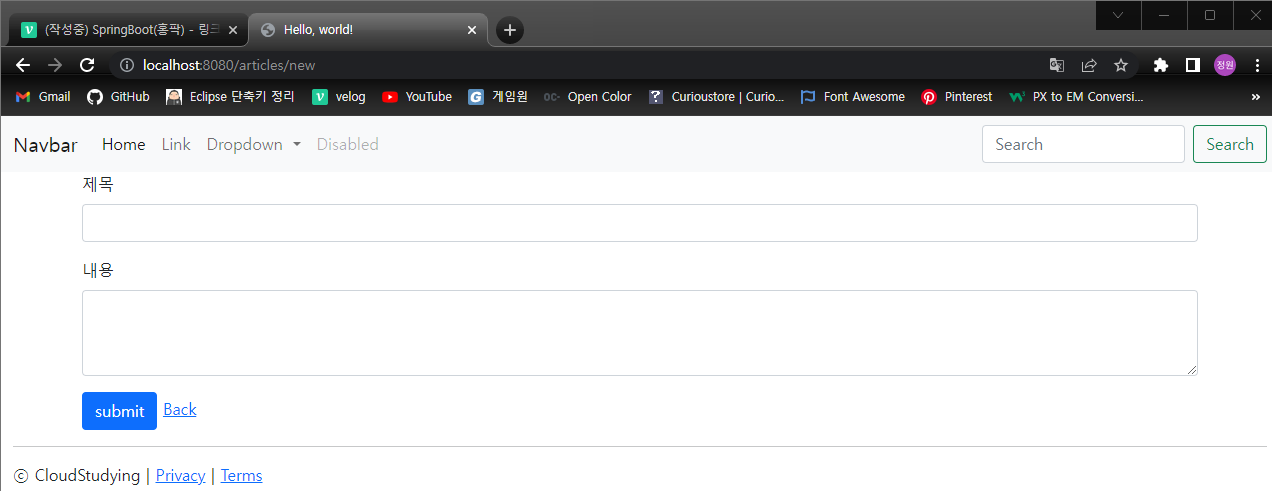
<a href="/articles/new">New Article</a>



**뒤로가기 링크**

new.mustache에 링크추가

<a href="/articles">Back</a>



**리다이렉트**

  
articles/create 에서 글을 작성한 후에 redirect를 진행.

Article.java 에 @Getter 추가.

@Getter

@AllArgsConstructor

@NoArgsConstructor // 디폴트 생성자 추가!

@ToString

@Entity // 1.DB가 해당 객체를 인식 가능하게 됨.

public class Article {

...

}

ArticleController.java 메소드에 return값 추가.  
saved.getId() 를 이용해 id값에 따른 상세페이지로 이동할수 있도록 한다.

@PostMapping("/articles/create")

public String createArticle(ArticleForm form) {

log.info(form.toString());

// System.out.println(form.toString()); -> 로깅기능으로 대체!!

// 1. DTO -> Entity 변환

Article article = form.toEntity();

log.info(article.toString());

// System.out.println(article);

// 2. Repository 에게 Entity를 DB안에 저장하게 함

Article saved = articleRepository.save(article);

log.info(saved.toString());

// System.out.println(saved.toString());

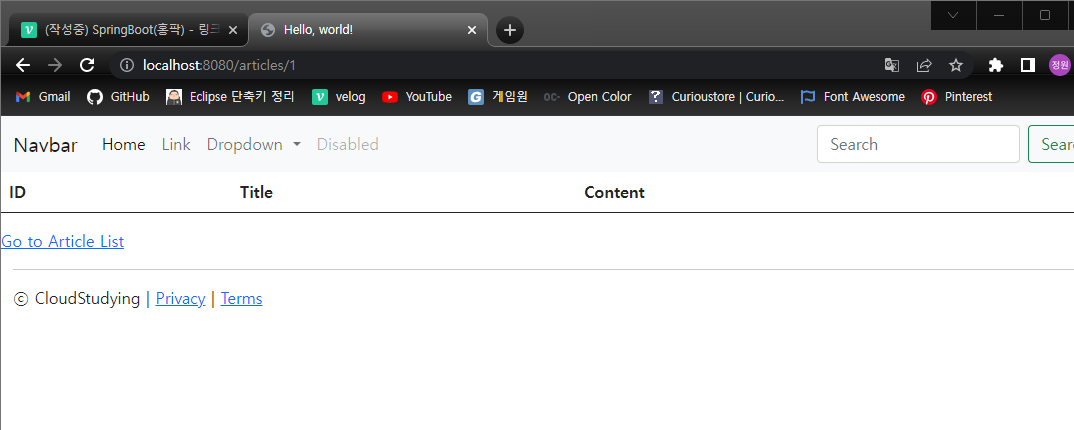
return "redirect:/articles/" + saved.getId();

}

**상세보기에서 목록으로 가기**

show.mustache에 링크추가

<a href="/articles">Go to Article List</a>



**목록에서 클릭시 상세보기로 가기**

index.mustache에 title을 a태그로 감싸고 href에 id값 전달

{{#articleList}}

<tr>

<th>{{id}}</th>

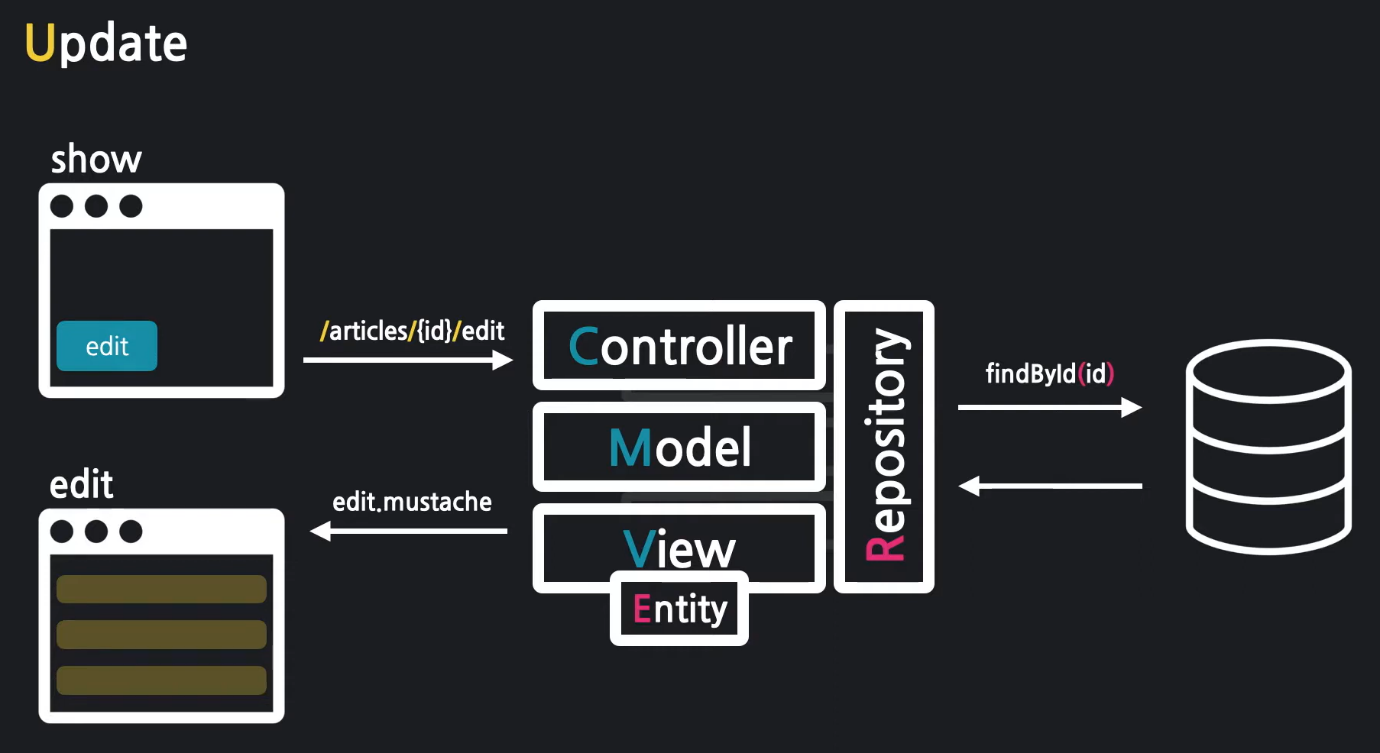
<td><a href="/articles/{{id}}">{{title}}</a></td>

<td>{{content}}</td>

</tr>

{{/articleList}}

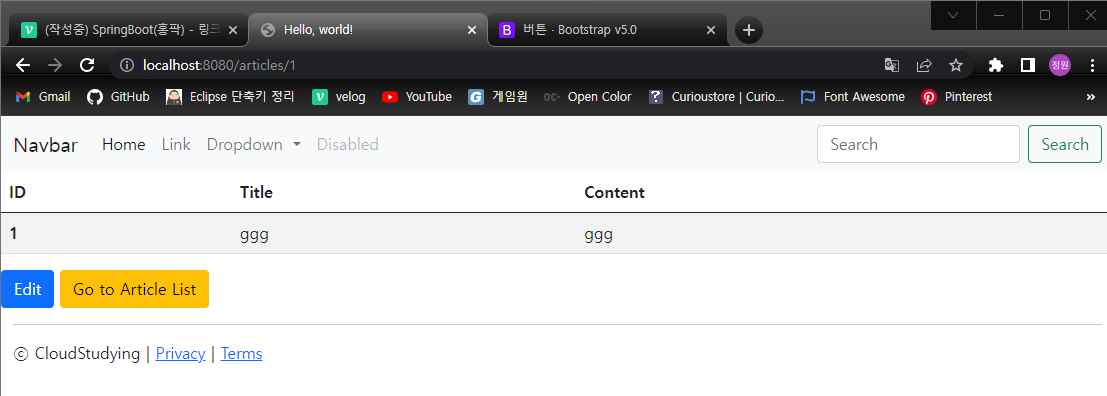
**수정 폼 만들기**



**수정 링크 걸기**

show.mustache에 링크 만들기  
article.id로 id값을 가져와서 해당 id값을 수정하도록한다.

<a href="/articles/{{article.id}}/edit"></a>



**요청 받기**

ArticleController에 메소드 작성.

@GetMapping("/articles/{id}/edit")

public String edit(@PathVariable Long id, Model model) {

// 수정할 데이터 가져오기

Article articleEntity = articleRepository.findById(id).orElse(null);

// 모델에 데이터 등록

model.addAttribute("article",articleEntity);

return "articles/edit";

}

**뷰 페이지 작성**

edit.mustache 작성

<form class="container" action="" method="post">

<div class="mb-3">

<label class="form-label">제목</label>

<input type="text" class="form-control" name="title" value="{{article.title}}">

</div>

<div class="mb-3">

<label class="form-label">내용</label>

<textarea class="form-control" rows="3" name="content">{{article.content}}</textarea>

</div>

<button type="submit" class="btn btn-primary">Submit</button>

<a href="/articles/{{article.id}}" class="btn btn-success">Back</a>

</form>

article.title, article.content, article.id ... article이 반복되기 때문에  
#article을 선언해서 반복되는 코드를 없애자.

{{>layouts/header}}

{{#article}}

<form class="container" action="" method="post">

<div class="mb-3">

<label class="form-label">제목</label>

<input type="text" class="form-control" name="title" value="{{title}}">

</div>

<div class="mb-3">

<label class="form-label">내용</label>

<textarea class="form-control" rows="3" name="content">{{content}}</textarea>

</div>

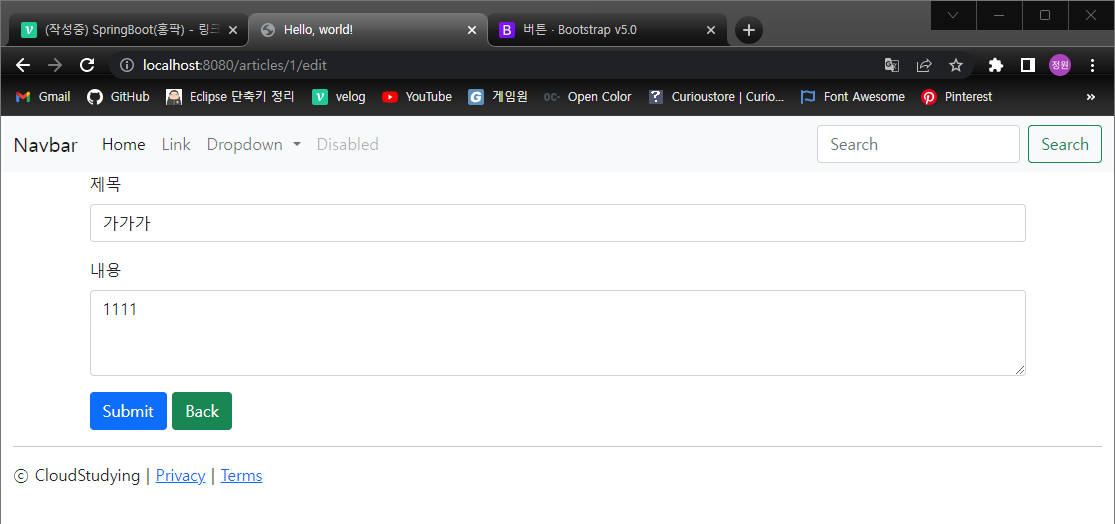
<button type="submit" class="btn btn-primary">Submit</button>

<a href="/articles/{{id}}">Back</a>

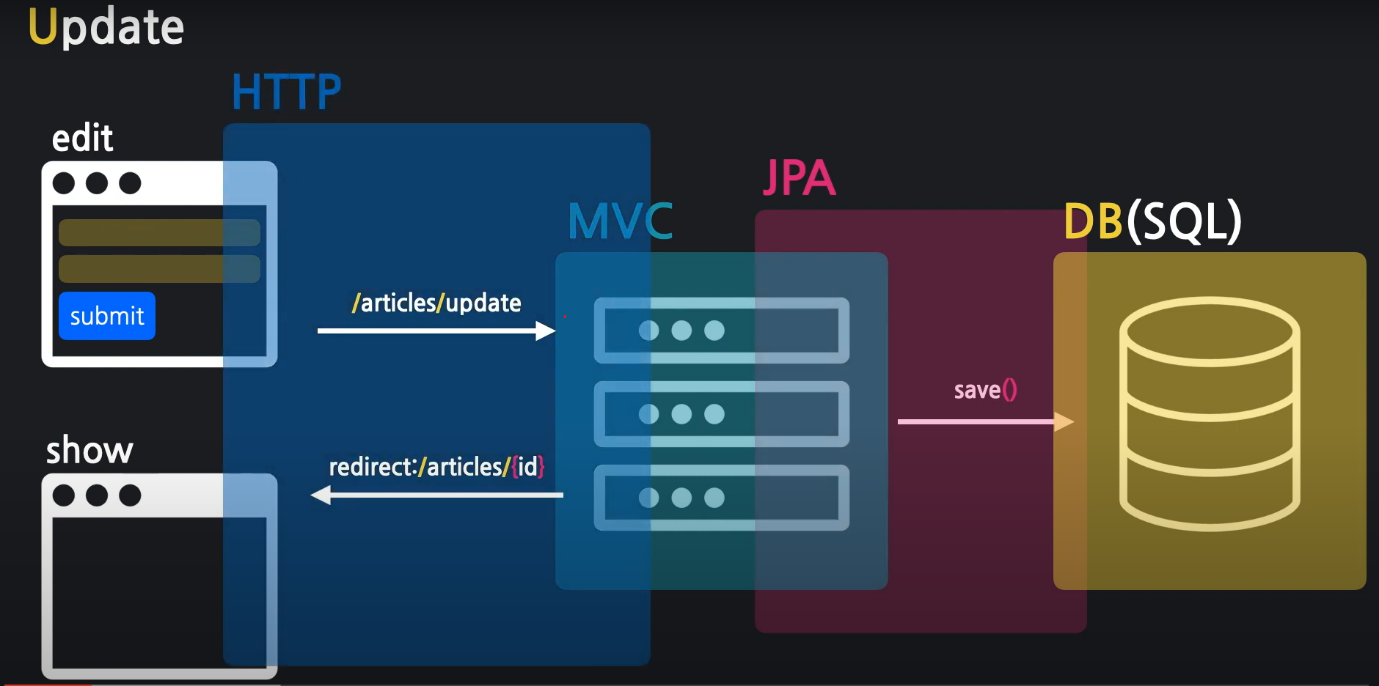
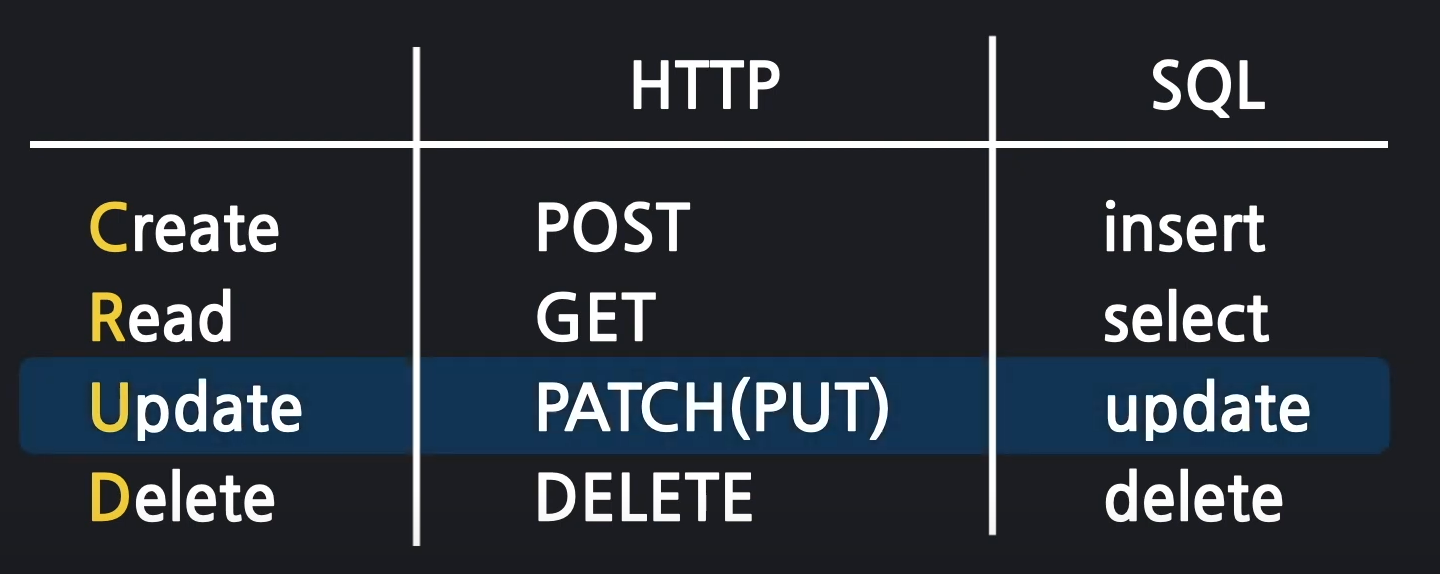
</form>

{{/article}}

{{>layouts/footer}}

edit 버튼 클릭시 데이터 끌고와서 잘 뿌려준당 ✨👍👌  


**데이터 수정하기**

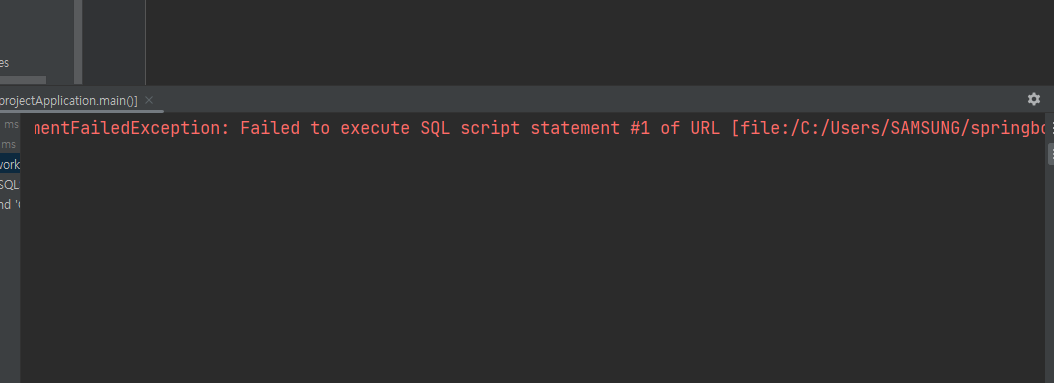
**더미 데이터 생성**

resources에 data.sql 생성

INSERT INTO article(id, title, content) VALUES(1, '가가가가', '1111');  
INSERT INTO article(id, title, content) VALUES(2, '나나나나', '2222');  
INSERT INTO article(id, title, content) VALUES(3, '다다다다', '3333');

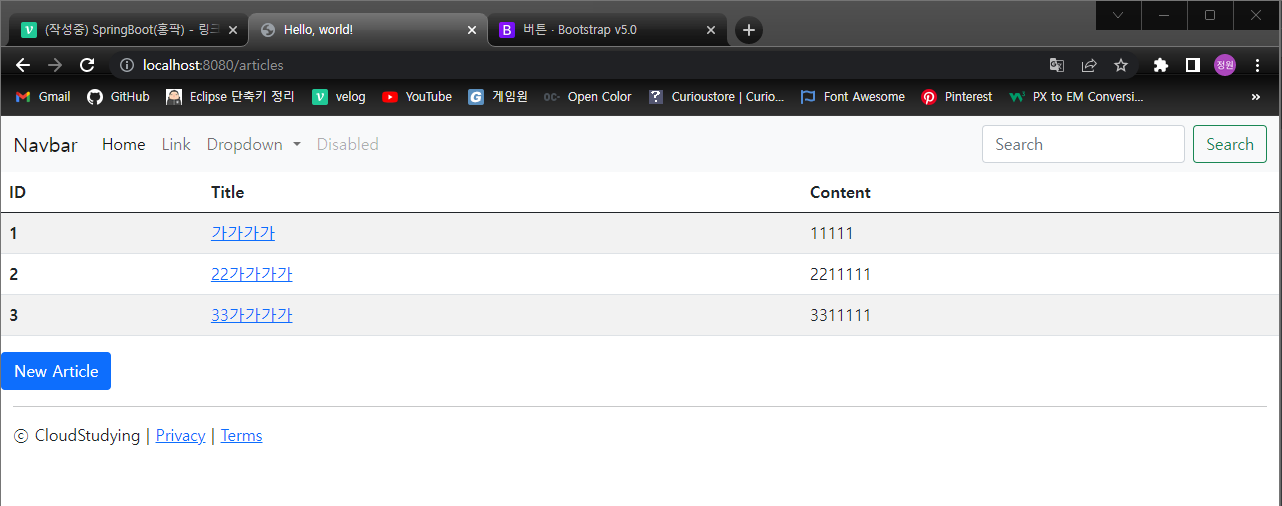
이렇게 하면 서버 껏다 키면 자동으로 테이블에 데이터 저장된다.

**failed to execute sql script statement...에러**

  
application.properties 에 spring.jpa.defer-datasource-initialization=true 추가하기.  
spring.sql.init.mode=always

spring.sql.init.data-locations=classpath:data.sql

spring.jpa.defer-datasource-initialization=true

data.sql에 넣은 데이터 잘 나온다.  


**edit.mustache form action 추가**

action과 method, 몇번 id를 수정하는지 알려주기 위해  
id도 함께 전달.

{#article}}

<form class="container" action="/articles/update" method="post">

<input name="id" type="hidden" value="{{id}}" />

...

**form의 method**

HTTP에서 update는 Patch이지만  
form은 get*과 post만 지원하기때문에 일단은  
method="post"로 진행한다.*

**수정폼받기**

ArticleController.java에 메소드 추가.  
update할떄는 @PatchMapping을 사용하면 되지만  
일단은 @PostMapping 사용.  
edit.mustache에서 id도 함께 전달 되기 때문에  
DTO를 수정해주자.

// @PatchMapping

@PostMapping("/articles/update")

public String update(ArticleForm form) {

return "";

}

**DTO 변경**

id 변수 추가하고 toEntity()에도 null이었던 곳에 id를 전달해준다.

@AllArgsConstructor

@ToString

public class ArticleForm {

private Long id; //id 필드 추가

private String title;

private String content;

public Article toEntity() {

return new Article(id, title, content);

}

}

**update 처리 흐름**

@PostMapping("/articles/update")

public String update(ArticleForm form) {

log.info(form.toString());

// 1: DTO를 Entity로 변환한다.

Article articleEntity= form.toEntity();

log.info(articleEntity.toString());

// 2: Entity를 DB로 저장한다.

// 2-1: DB에서 기존 데이터를 가져온다.

Article target = articleRepository.findById(articleEntity.getId()).orElse(null);

// 2-2: 기존 데이터에 값을 갱신한다.

if(target != null) {

articleRepository.save(articleEntity); // Entity가 DB로 갱신.

}

// 3: 수정 결과 페이지로 리다이렉트.

return "redirect:/articles/" + articleEntity.getId();

}

👌 UPDATE 완료  
