7장 게시글 수정하기: Update

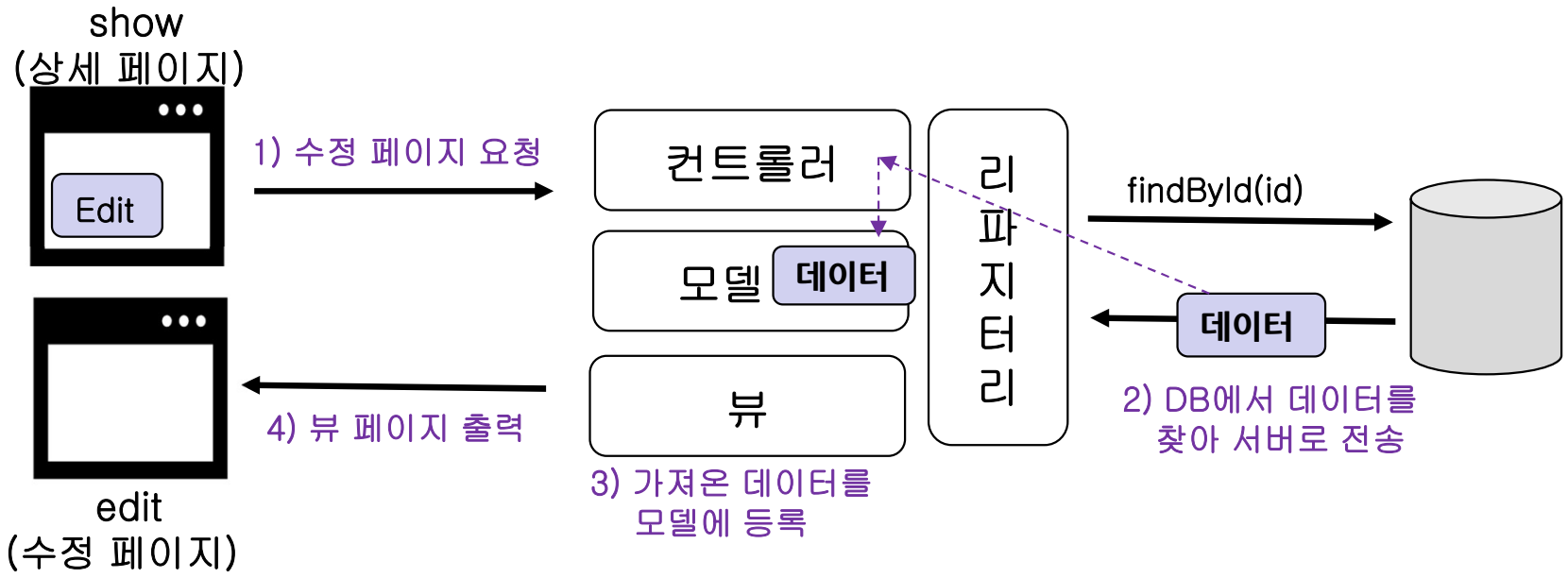
출처: 코딩 자율학습 스프링부트3 자바 백엔드 개발 입문, 홍팍, 길벗, 2023

데이터 수정 과정

- **〈수정 페이지〉 만들고 기존 데이터 불러오기**
- **수정 폼의 데이터로 DB를 갱신하기**
  - 데이터를 수정해 DB에 반영한 후 결과를 볼 수 있게 〈상세 페이지〉로 리다이렉트하기

〈수정 페이지〉 만들고 기존 데이터 불러오기

1) 〈상세 페이지〉에서 [Edit] 버튼 클릭
2) 요청을 받은 컨트롤러는 해당 글의 id로 DB에서 데이터를 찾아 가져옴
3) 컨트롤러는 가져온 데이터를 뷰에서 사용할 수 있도록 모델에 등록
4) 모델에 등록된 데이터를 〈수정 페이지〉에서 보여 줌



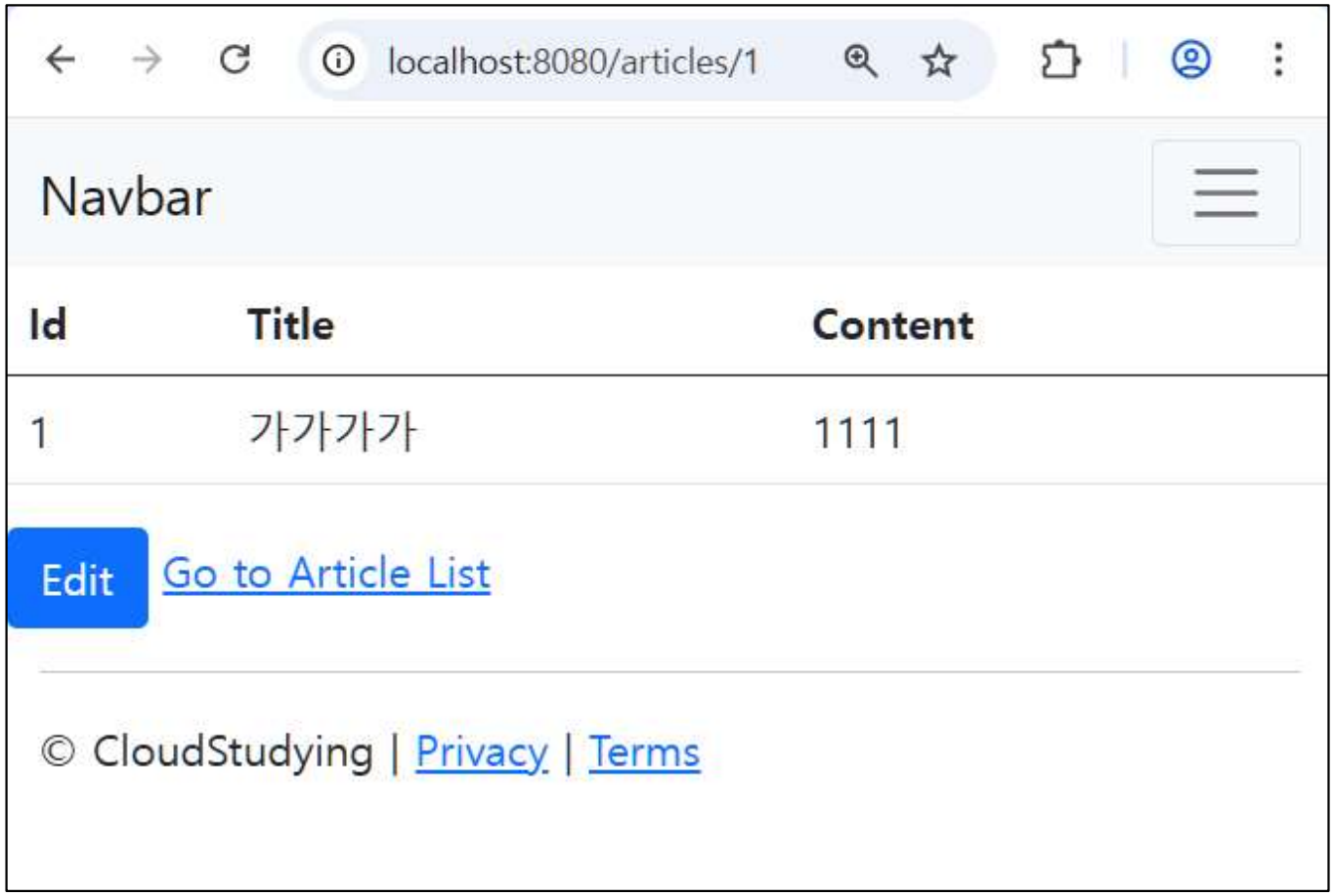
〈수정 페이지〉 만들기

- **〈상세 페이지〉에 Edit 버튼 만들기**
  - show.mustache 파일 편집
    - ✓ "/articles/{{article.id}}/edit"로 연결하는 링크 추가
      - 1번 글을 수정할 경우 URL : "/articles/1/eidt"
      - 2번 글을 수정할 경우 URL: "/articles/2/edit", …

```
    <tbody>
    {{#article}}
    <tr>
        <td>{{id}}</td>
        <td>{{title}}</td>
        <td>{{content}}</td>
    </tr>
    {{/article}}
    </tbody>
</table>

<a href="/articles/{{article.id}}/edit">Edit</a>
<a href="/articles">Go to Article List</a>
```

〈수정 페이지〉 만들기

- 서버 실행
- localhost:8080/articles/new 에 접속해 게시글 작성 후, [Submit]
- 〈상세 페이지〉에서 [Edit] 버튼이 잘 나오는지 확인
  - localhost:8080/articles/1



Edit 요청을 받아 데이터 가져오기

- **edit() 메서드 기본 틀 만들기**
  - ArticleController.java에 edit() 메서드 추가
    - ✓ 반환 값: articles/edit → 뷰 파일: templates/articles/eidt.mustache
    - ✓ 컨트롤러에서 URL 변수를 사용할 때는 {}를 사용

ArticleController.java

```
@GetMapping("/articles/{id}/edit")
public String edit(){
    return "articles/edit";
}
```

Edit 요청을 받아 데이터 가져오기

- **수정할 데이터 가져오기**
  - DB에 있는 기존 데이터 불러오기
    - ✓ articleRepository의 findById(id) 메서드로 데이터를 가져옴
    - ✓ id는 @GetMapping("/articles/{id}/edit")의 URL 주소에 있는 id를 받아 옴
      → @PathVariable 추가

ArticleController.java

```
@GetMapping("/articles/{id}/edit")
public String edit(@PathVariable Long id){
    //수정할 데이터 가져오기
    Article articleEntity = articleRepository.findById(id).orElse( other: null);
    return "articles/edit";
}
```

id를 매개변수로 받아오기

DB에서 수정할 데이터 가져오기

Edit 요청을 받아 데이터 가져오기

- **모델에 데이터 등록하기**
  - DB에서 가져온 데이터를 뷰 페이지에서 사용할 수 있도록 모델에 등록

```
@GetMapping("/articles/{id}/edit")
public String edit(@PathVariable Long id, Model model){
    //수정할 데이터 가져오기
    Article articleEntity = articleRepository.findById(id).orElse( other: null);
    //모델에 데이터 등록하기
    model.addAttribute( attributeName: "article", articleEntity);
    return "articles/edit";
}
```

model 객체 받아 오기

articleEntity를 article로 등록

수정 폼 만들기

- **[src - main - resources - templates - articles]에서 마우스 오른쪽 버튼 클릭**
- **컨텍스트 메뉴에서 [New - File] 선택**
  - 새 파일명: edit.mustache
- **edit.mustache 파일 편집**
  - new.mustache 복사해서 edit.mustache 파일로 붙여넣기
  - 폼의 action 속성 값과 Back 링크를 수정

수정 폼 만들기

edit.mustache

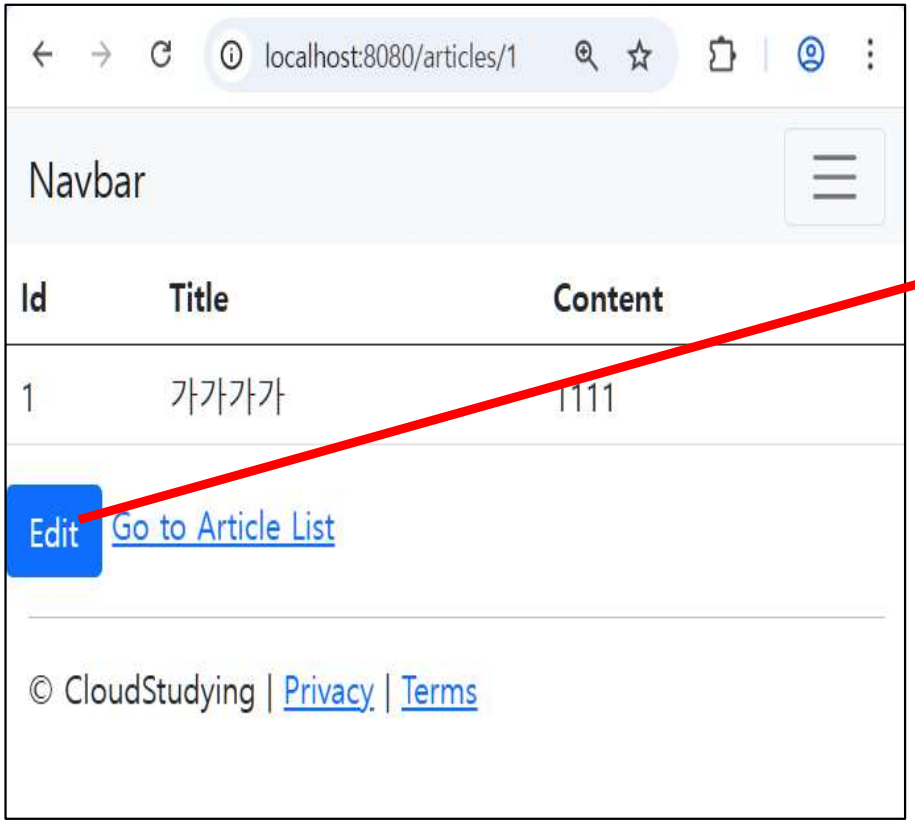
```
{{>layouts/header}}

<form class="container" action="" method="post">
    <div class="mb-3">
        <label class="form-label">제목</label>
        <input type="text" class="form-control" name="title">
    </div>
    <div class="mb-3">
        <label class="form-label">내용</label>
        <textarea class="form-control" rows="3" name="content"></textarea>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
    <a href="/articles/{{article.id}}">Back</a>
</form>

{{>layouts/footer}}
```

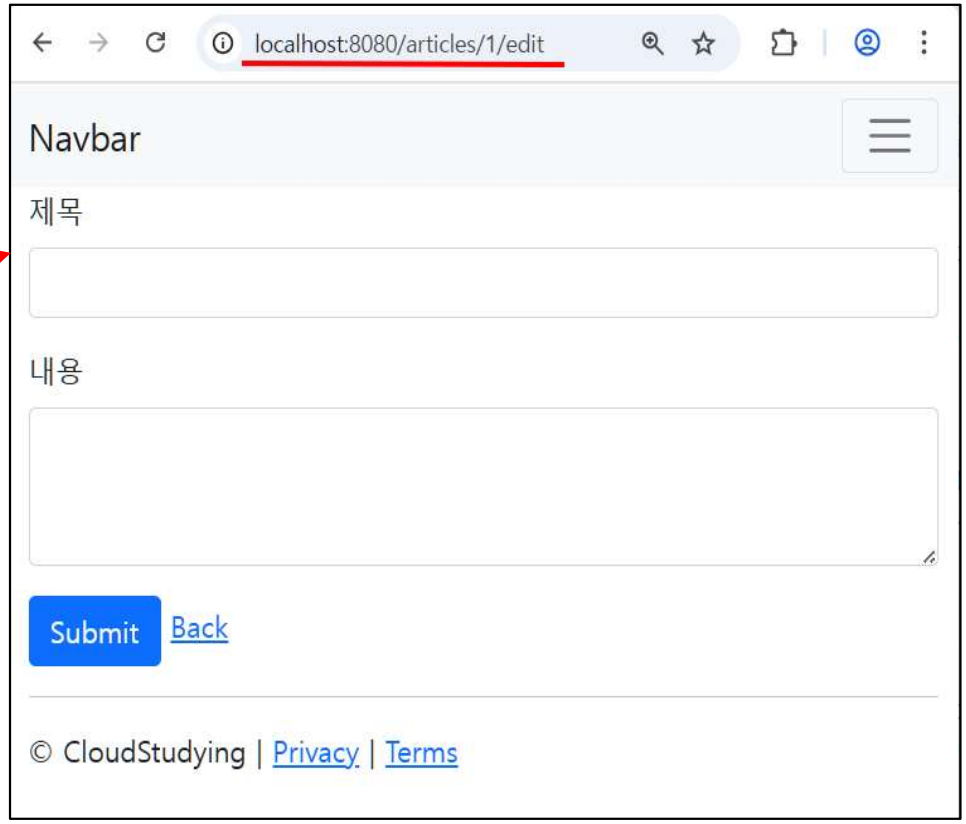
action 속성 값 삭제

링크 수정

수정 폼 만들기

- **서버 재시작**
  - localhost:8080/articles/new 접속해 새 게시물 작성, [Submit]
  - 〈상세 페이지〉에서 [Edit] 클릭





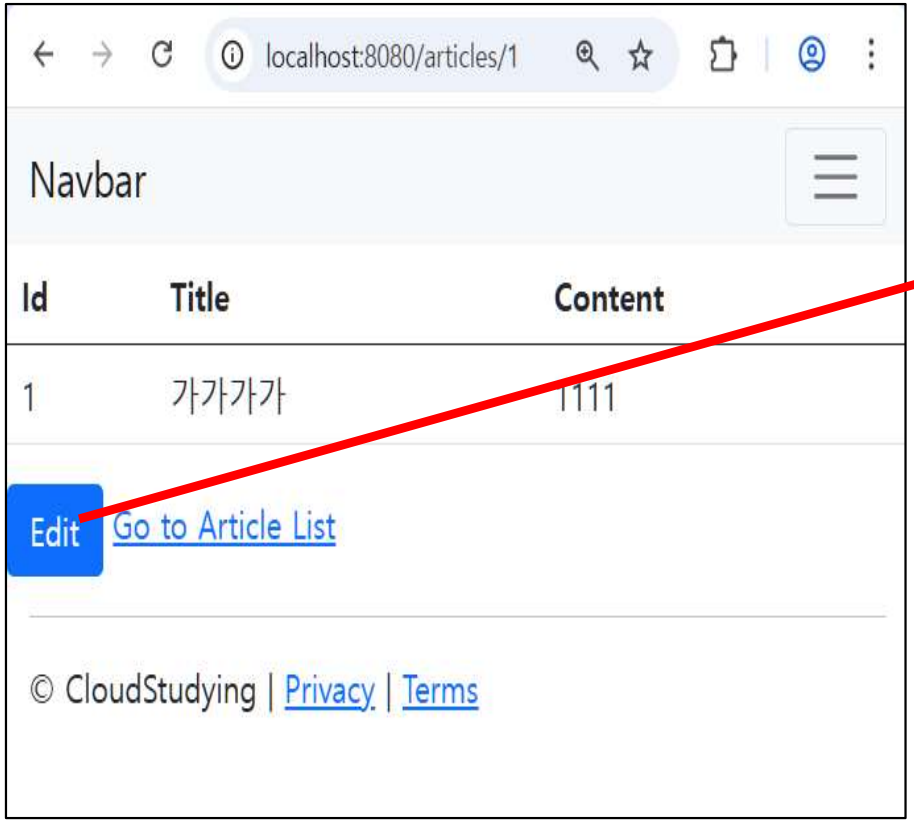
수정 폼 만들기

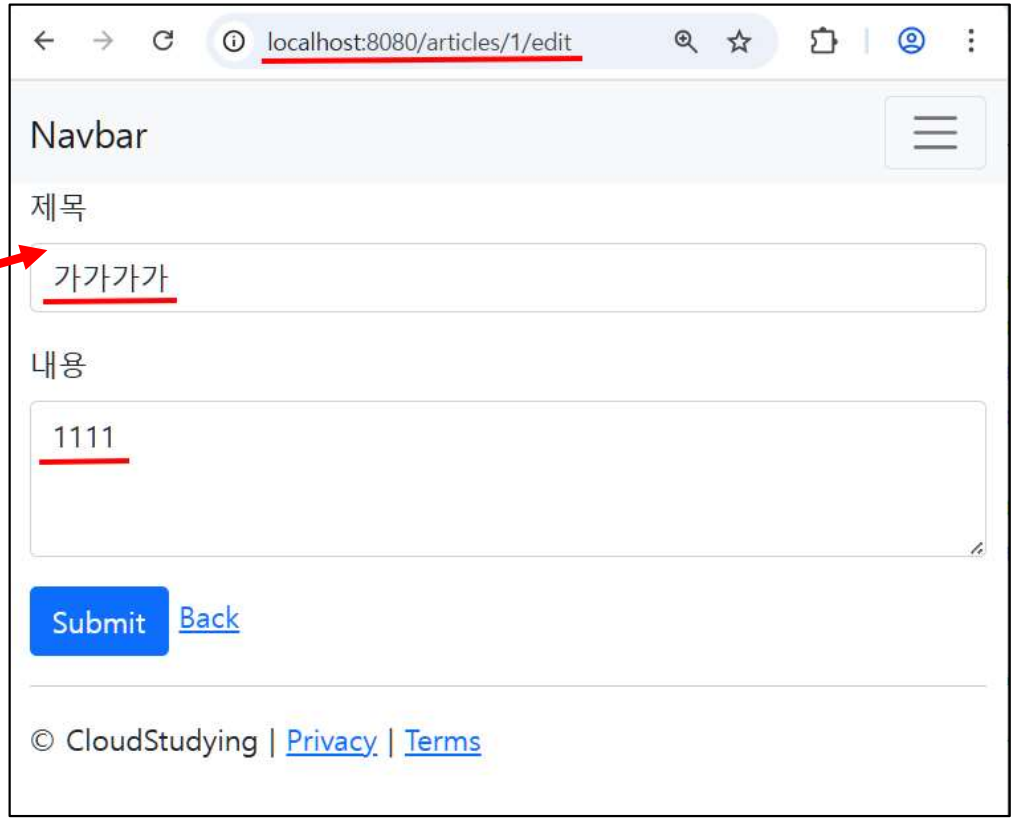
- **edit.mustache 파일 편집**
  - Title, content에 게시물의 데이터를 출력

```
{{>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}}
```

수정 폼 만들기

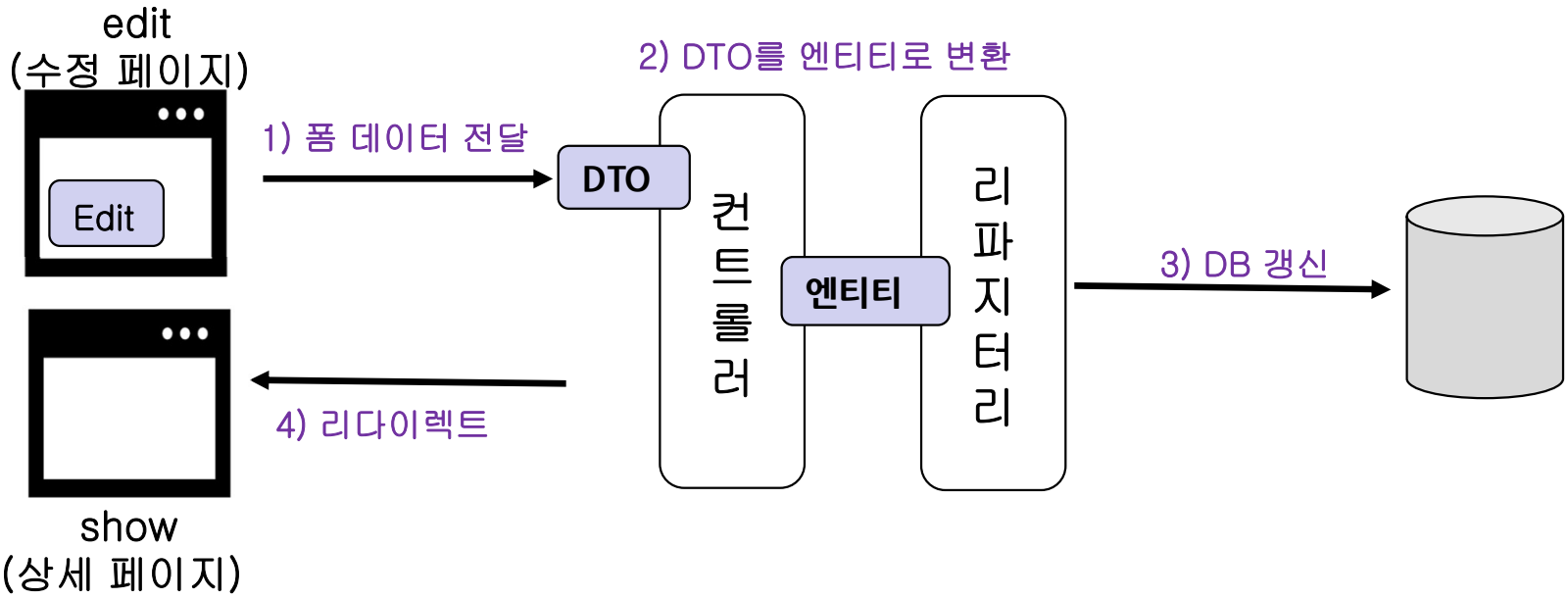
- 프로젝트 빌드
- localhost:8080/articles/1 접속, [Edit] 클릭





수정 폼의 데이터로 DB 갱신하기

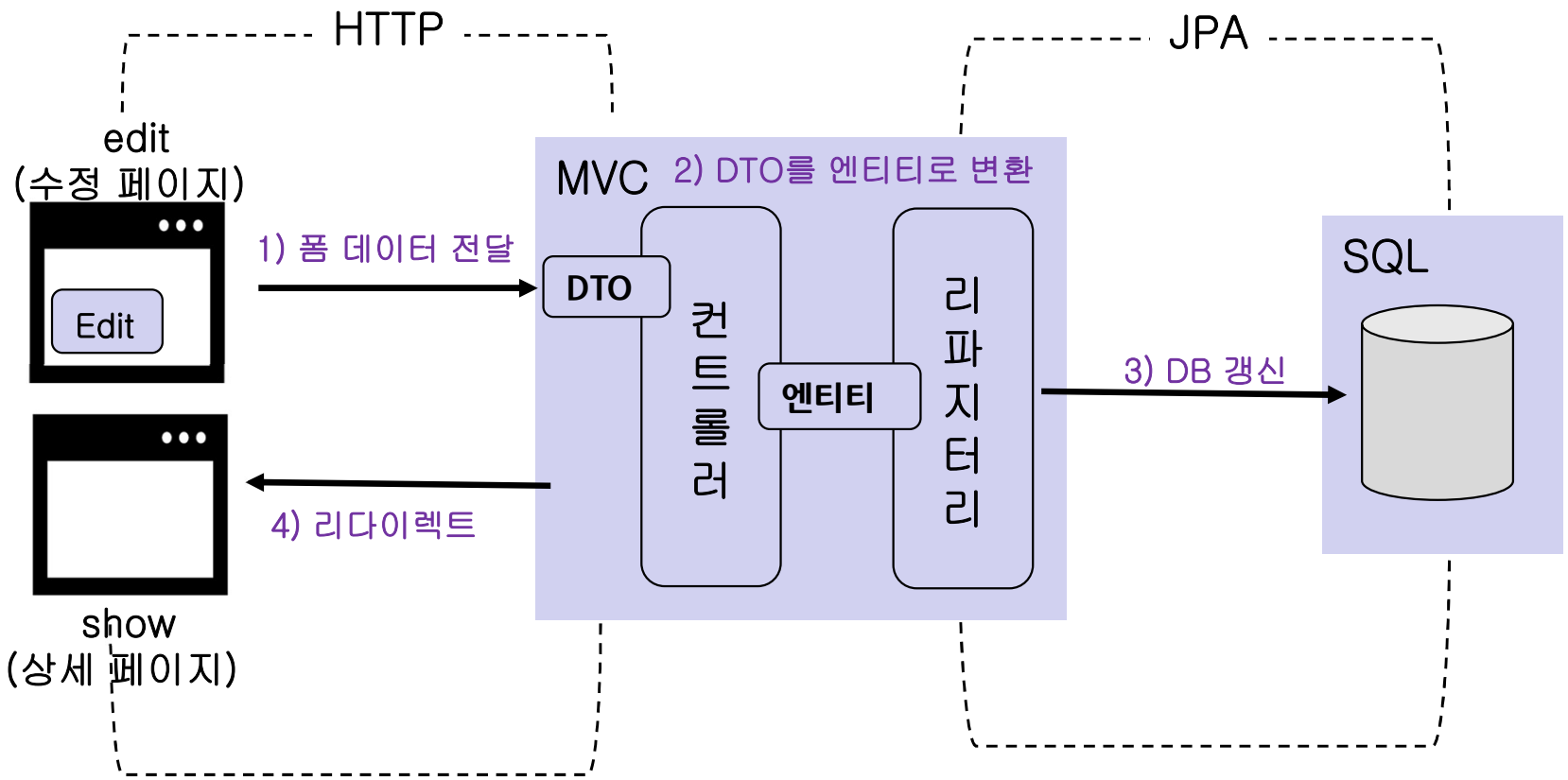
- **데이터를 수정해 DB에 반영한 후, 결과를 볼 수 있게 〈상세 페이지〉로 리다이렉트하기**
  1) 폼 데이터(수정 요청 데이터)를 DTO에 담아 컨트롤러에서 받음
  2) DTO를 엔티티로 변환
  3) DB에서 기존 데이터를 수정 데이터로 갱신
  4) 수정 데이터를 〈상세 페이지〉로 리다이렉트



클라이언트와 서버 간 데이터를 처리를 위한 4가지 기술

- MVC(Model - View - Controller)
  - 서버 역할을 분담해 처리하는 기법
- JPA(Java Persistence API)
  - 서버와 DB간 소통에 관여하는 기술
- SQL(Structured Query Language)
  - DB 데이터를 관리하는 언어
- HTTP(HyperText Transfer Protocol)
  - 웹서버와 클라이언트간 데이터를 주고 받기 위한 통신 규약

클라이언트와 서버 간 데이터를 처리를 위한 4가지 기술



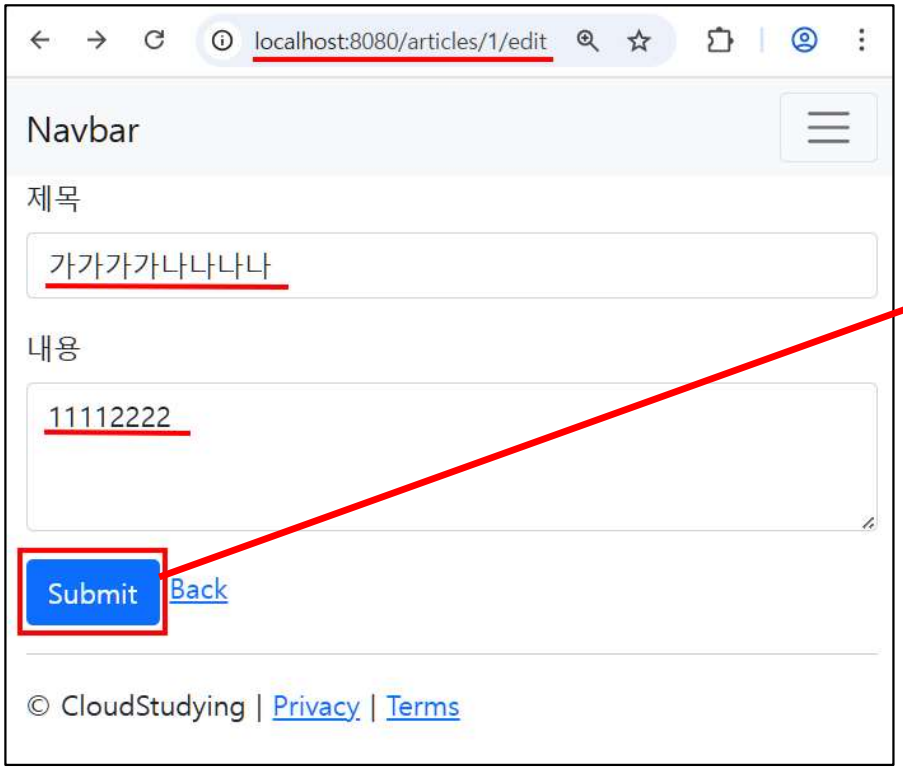
HTTP 메서드

- **HTTP는 클라이언트의 다양한 요청을 메서드를 통해 서버로 보냄**
- **HTTP의 대표적인 4개의 메서드**
  - POST: 데이터 생성 요청
  - GET: 데이터 조회 요청
  - PATCH(PUT): 데이터 수정 요청
  - DELETE: 데이터 삭제 요청
- **CRUD를 위한 SQL문과 HTTP 메서드**

데이터 관리	**SQL**	**HTTP**
데이터 생성(Create)	INSERT	POST
데이터 조회(Read)	SELECT	GET
데이터 수정(Update)	UPDATE	PATCH(PUT)
데이터 삭제(Delete)	DELETE	DELETE

HTTP 메서드

- 데이터 수정한 후 에러 페이지 확인



더미 데이터 설정하기

- **H2가 인메모리 DB이기 때문에 서버를 껐다 켤 때마다 매번 데이터가 사라지므로 이런 버거로움을 개선 하기 위해 더미(dummy) 데이터를 자동 생성해서 사용**
- **[src - main - resources]에서 마우스 오른쪽 버튼 클릭**
- **컨텍스트 메뉴에서 [New - File]**
  - 새 파일명: data.sql
  - 참조: 기본 내용만 실습하므로, Plugins supporting *.sql files found 메시지가 뜨면, Ignore extensions을 클릭
- **data.sql 파일에 sql 쿼리 추가**

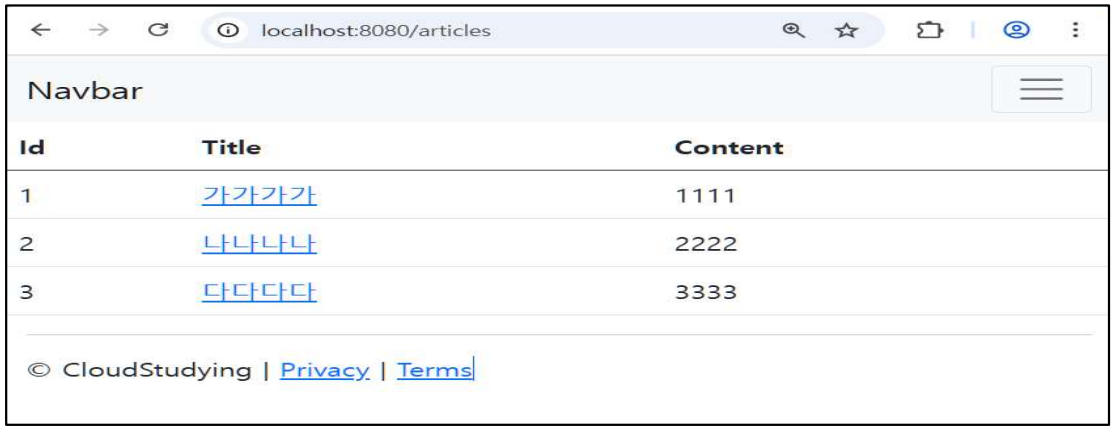
```
INSERT INTO article(id, title, content) VALUES (1, '가가가가', '1111');
INSERT INTO article(id, title, content) VALUES (2, '나나나나', '2222');
INSERT INTO article(id, title, content) VALUES (3, '다다다다', '3333');
```

더미 데이터 설정하기

- [src - main - resources] 디렉토리에 있는 application.properties 열기
- application.properties에 옵션 추가

```
spring.application.name=firstproject
server.servlet.encoding.force=true
spring.h2.console.enabled=true
spring.jpa.defer-datasource-initialization=true
```

- 서버 재시작
- localhost:8080/articles 접속



〈수정 페이지〉 변경하기

- **edit.mustache 파일 수정**
  - 폼 데이터를 보낼 곳(URL)
    - ✓ 〈form〉 태그의 action 속성 값: "/articles/udpate"
  - 폼 데이터를 보내는 방법
    - ✓ 〈form〉 태그의 method 속성 값: "post"
    - ✓ PATCH를 사용하지 않고, post를 사용한 이유
      - 〈form〉 태그가 PATCH 메서드를 지원하지 않기 때문
      - 〈form〉 태그는 전송방법으로 get과 post만을 지원

edit.mustache

```
{{>layouts/header}}

{{#article}}
<form class="container" action="/articles/update" method="post">
    <input name="id" type="hidden" value="{{id}}">
    <div class="mb-3">
        <label class="form-label">제목</label>
        <input type="text" class="form-control" name="title" value="{{title}}">
    </div>
```

서버로 id를 전송해야 하지만,
화면에 표시할 필요는 없음

수정 데이터 받아 오기

- **update() 메서드 기본 틀 만들기**
  - 수정 데이터를 받아 처리하는 컨트롤러의 메서드
  - ArticleController.java에 작성

```
public class ArticleController {

    @Autowired
    private ArticleRepository articleRepository;
            (중략)

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

    @PostMapping("/articles/update")
    public String update(){
        return "";
    }

}
```

수정 데이터 받아 오기

- **수정 데이터를 DTO에 담기**
  - update() 매개변수로 DTO 받기
  - 수정 데이터를 잘 받았는지 로그 찍어보기

ArticleController.java

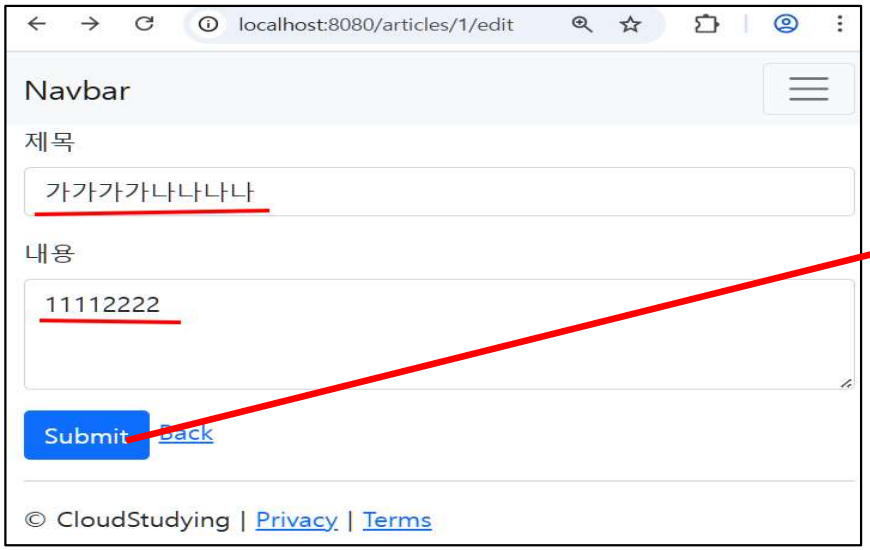
```
@PostMapping("/articles/update")
public String update(ArticleForm form){
    log.info(form.toString());
    return "";
}
```

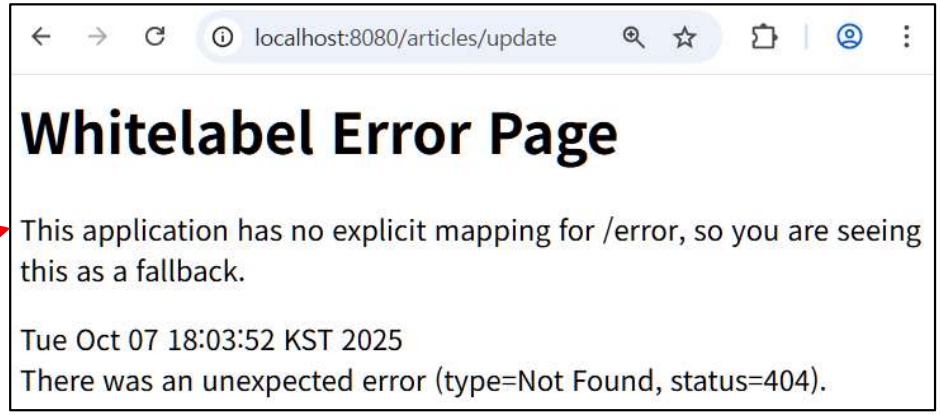
  - ArticleForm에 id 추가하기
    - ✓ 수정 폼에 〈input〉 태그로 id를 추가했으므로 DTO인 ArticleForm에도 추가

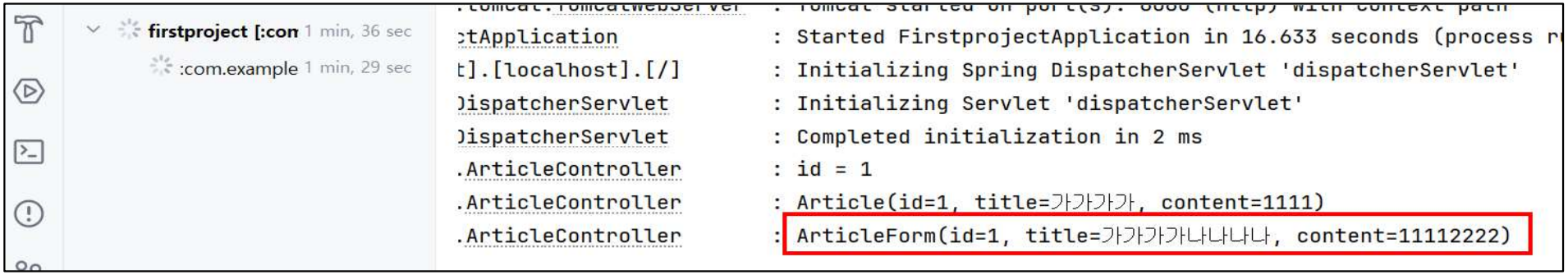
```
public class ArticleForm {
    private Long id;          <-- 1) id 필드 추가
    private String title;
    private String content;

    public Article toEntity() {
        return new Article(id, title, content);
    }                          null --> id로 수정
}
```

수정 데이터 받아 오기

- 서버 재시작
- localhost:8080/articles/1 접속 후, [Edit] 클릭
- 글을 수정(title, content)을 한 후, [Submit]
  - 로그 확인: 수정된 데이터를 잘 받았는지 확인







DB에 저장하고 결과 페이지로 리다이렉트하기

- DTO를 엔티티로 변환하기
- 엔티티를 DB에 저장하기
- 수정 결과 페이지로 리다이렉트하기

ArticleController.java

```
@PostMapping("/articles/update")
public String update(ArticleForm form){
    log.info(form.toString());
    // 1. DTO를 엔티티로 변환하기
    // 2. 엔티티를 DB에 저장하기
    // 3. 수정 결과 페이지로 리다이렉트하기
    return "";
}
```

DB에 저장하고 결과 페이지로 리다이렉트하기

- **DTO를 엔티티로 변환하기**

ArticleController.java

```
@PostMapping("/articles/update")
public String update(ArticleForm form){
    log.info(form.toString());
    // 1. DTO를 엔티티로 변환하기
    Article articleEntity = form.toEntity();
    log.info(articleEntity.toString());
    // 2. 엔티티를 DB에 저장하기
    // 3. 수정 결과 페이지로 리다이렉트하기
    return "";
}
```

DB에 저장하고 결과 페이지로 리다이렉트하기

- **엔티티를 DB에 저장하기**
  - DB에 기존 데이터 가져오기
  - 기존 데이터 값을 갱신하기 : 수정 대상의 존재 여부 확인후 갱신

ArticleController.java

```
@PostMapping("/articles/update")
public String update(ArticleForm form){
    log.info(form.toString());
    // 1. DTO를 엔티티로 변환하기
    Article articleEntity = form.toEntity();
    log.info(articleEntity.toString());
    // 2. 엔티티를 DB에 저장하기
    // 2.1 DB에서 기존 데이터 가져오기
    Article target = articleRepository.findById(articleEntity.getId()).orElse( other: null);
    // 2.2 기존 데이터 값을 갱신하기
    if(target != null){
        articleRepository.save(articleEntity);
    }
    // 3. 수정 결과 페이지로 리다이렉트하기
    return "";
}
```

2) 데이터 저장

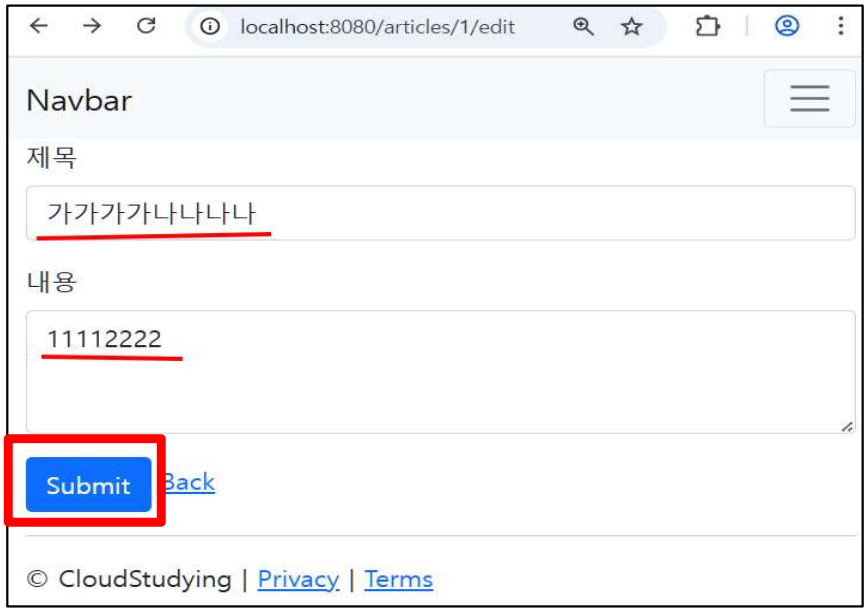
1) DB에서 데이터 찾기

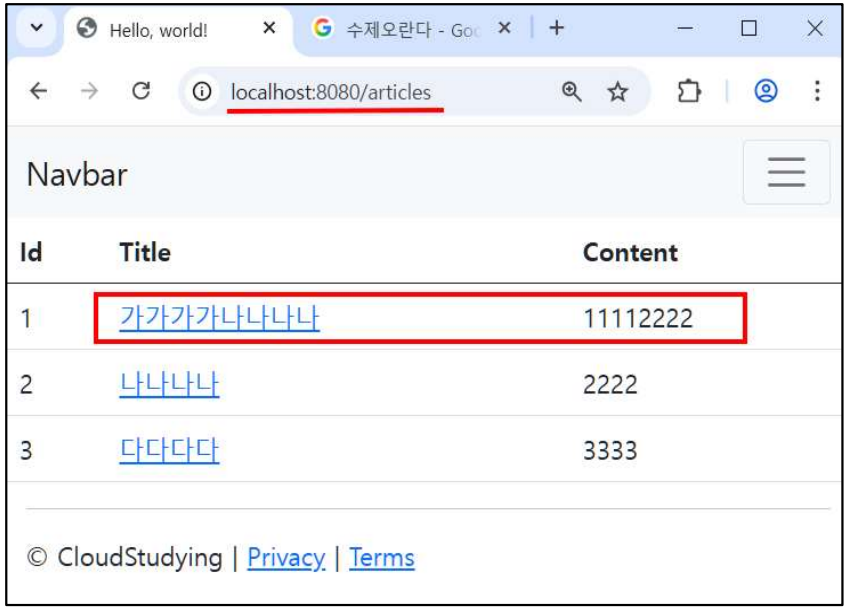
3) 데이터가 없으면 null 반환

If문으로 수정 대상의 존재 여부 확인 후 갱신

DB에 저장하고 결과 페이지로 리다이렉트하기

- 서버 재시작
- localhost:8080/articles 접속
- 게시글 하나를 선택해 [Edit], 수정 후 [Submit]
- localhost:8080/articles 접속: 수정 글 확인





DB에 저장하고 결과 페이지로 리다이렉트하기

- **결과 페이지로 리다이렉트 하기**
  - 게시글 하나를 수정해서 [Submit] 하면, 해당 게시글의 상세 페이지가 열리도록 함
    - ✓ Id 1번 글을 수정하면, /articles/1 페이지를 리다이렉트
    - ✓ Id 2번 ″ /articles/2
  - ArticleController.java의 update() 메서드의 리턴 값 수정 ″

ArticleController.java

```
@PostMapping("/articles/update")
public String update(ArticleForm form){
    log.info(form.toString());
    // 1. DTO를 엔티티로 변환하기
    Article articleEntity = form.toEntity();
                (중략)

    // 3. 수정 결과 페이지로 리다이렉트하기
    return "redirect:/articles/" + articleEntity.getId();
}
```

DB에 저장하고 결과 페이지로 리다이렉트하기

- 서버 재시작
- localhost:8080/articles 접속
- 게시글 하나를 선택해 [Edit], 수정 후 [Submit]
  - 수정한 게시글의 상세 페이지 확인