

# 게시판 CRUD 만들기

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

# 3장 게시판 만들고, 새 글 작성하기

## create

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

# 게시판 CRUD

- Create : 게시물 생성
- Read: 조회
- Update: 수정
- Delete: 삭제

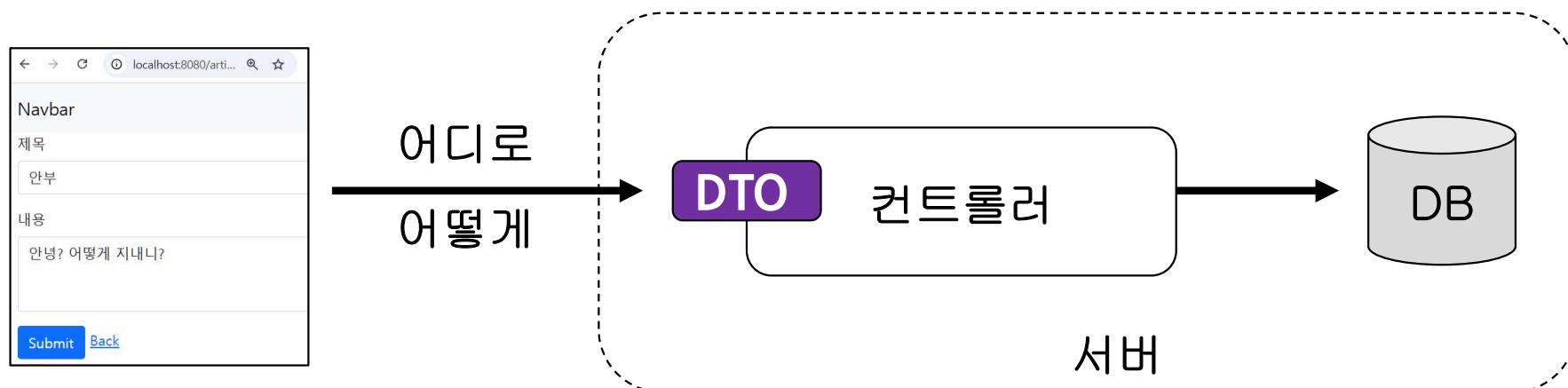
# 폼(form) 데이터와 DTO

- HTML 요소인 <form> 태그에 실려 전송되는 데이터

```
<form action="어디로" method="어떻게">
  <input type="text">
  <textarea></textarea>
  <button type="submit">Submit</button>
</form>
```

- DTO(Data Transfer Object)

- <form> 태그에 실어 보낸 데이터는 서버의 컨트롤러가 DTO라는 객체에 담아 보관
- DTO로 받은 데이터는 최종적으로 DB에 저장됨



# 입력 폼 만들기

- [firstproject - src - main - resources - templates]를 오른쪽 마우스로 클릭, 컨텍스트 메뉴에서 [new - Directory] 선택
  - 새 디렉토리명: articles
- [articles] 디렉토리에서 마우스 오른쪽 버튼, [New - File] 선택
  - 새 파일명: new.mustache
- new.mustache 파일 편집

```
{{>layouts/header}}
```

```
<form action="">
    <input type="text">
    <textarea></textarea>
    <button type="submit">Submit</button>
</form>
```

```
{{>layouts/footer}}
```

# 컨트롤러 만들기

- 컨트롤러 만들기

- [src - main - java - com.example.firstproject - controller] 디렉토리에서 마우스 오른쪽 버튼 클릭
- [New - Java Class] 선택
  - ✓ 새 파일명 ‘ArticleController’ 입력
- ArticleController.java 편집

```
package com.example.firstproject.controller;

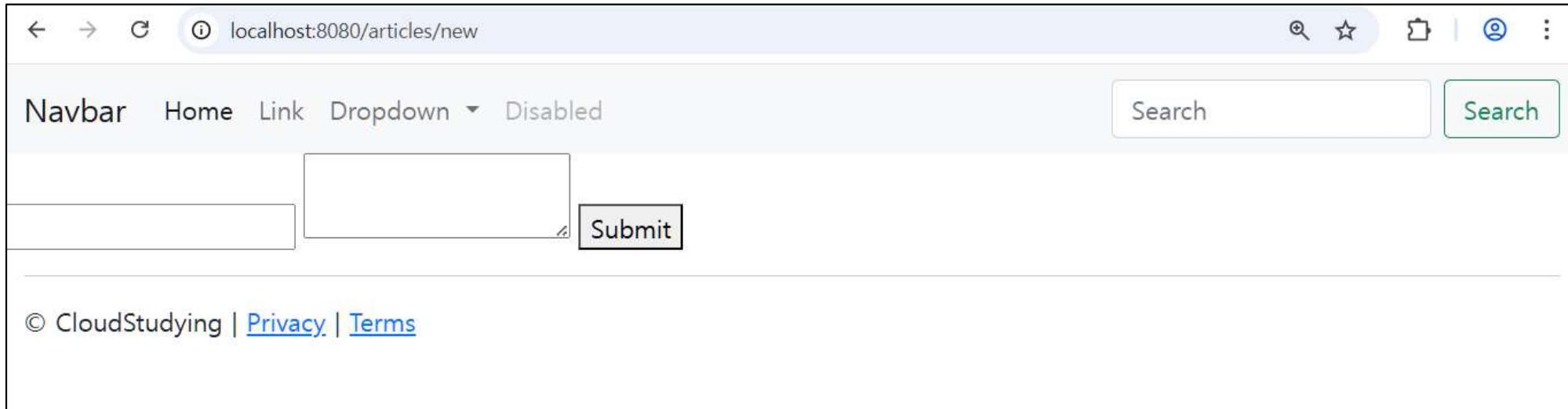
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller //컨트롤러 선언
public class ArticleController {

    @GetMapping("/articles/new") //URL(localhost:8080/articles/new) 요청 접수
    public String newArticleForm(){
        return "articles/new"; //반환값으로 뷰 페이지(articles/new.mustache)의 이름
    }
}
```

# localhost:8080/articles/new 실행결과

- 서버 실행 하기 >
- localhost:8080/articles/new 접속하여 실행결과 확인



# 입력 폼 꾸미기

- new.mustache에 부트스트랩 CSS 코드로 스타일 적용

new.mustache

```
 {{>layouts/header}}
```

```
<form class="container" action="">
```

```
    <div class="mb-3">
```

```
        <label class="form-label">제목</label>
```

```
        <input type="text" class="form-control">
```

```
    </div>
```

```
    <div class="mb-3">
```

```
        <label class="form-label">내용</label>
```

```
        <textarea class="form-control" rows="3"></textarea>
```

```
    </div>
```

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

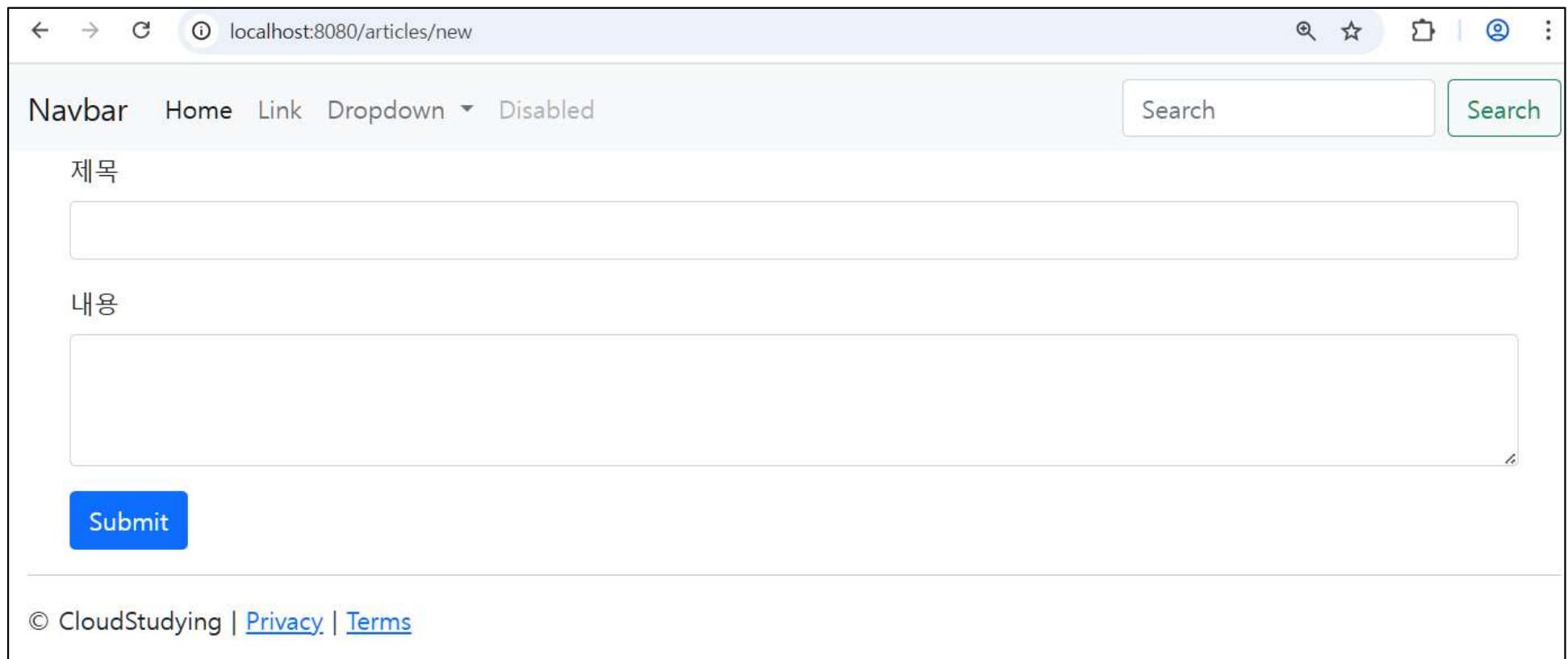
```
</form>
```

```
 {{>layouts/footer}}
```

# localhost:8080/articles/new 실행결과

- [Build - Build project] 클릭, 또는  , Ctrl + F9
  - 자바(\*.java) 코드가 바뀔 때는 서버를 재시작해야 함
  - 머스테치(\*.mustache) 코드가 바뀔 때는 [Build Project]만 해도 됨
- localhost:8080/articles/new 접속하여 실행결과 확인



# 폼 데이터 전송하기

- 폼 데이터를 전송하기 위한 **<form>** 속성
  - action : 폼 데이터를 보내는 곳의 URL을 설정
  - method: 폼 데이터 전송방식 설정
    - ✓ 속성값: post, get
- new.mustache 편집

```
{{>}layouts/header}

<form class="container" action="/articles/create" method="post">
  (중략)
</form>

{{>}layouts/footer}
```

# 폼 데이터 받기

- ArticleController.java 편집: createArticle() 추가

- @PostMapping(url)

- ✓ Post 방식으로 폼데이터가 전송되었을 때 사용

- ✓ Url은 new.mustache 파일에 <form>태그 action="/articles/create"의 속성값으로 설정

(중략)

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

```
@Controller
```

```
public class ArticleController {
```

```
    @GetMapping("/articles/new")
```

```
    public String newArticleForm(){
```

```
        return "articles/new";
```

```
}
```

```
    @PostMapping( "/articles/create" ) //URL 요청 접수
```

```
    public String createArticle(){
```

```
        return "";
```

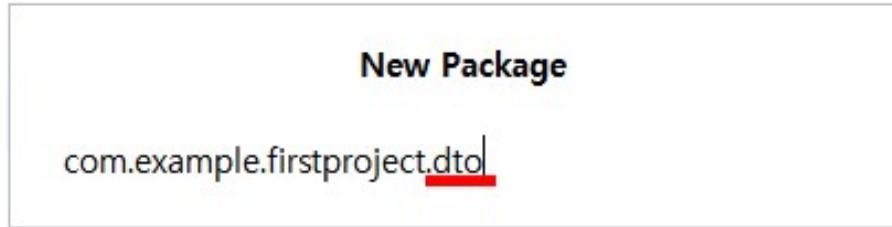
```
        //반환할 뷰 페이지를 작성하지 않음
```

```
}
```

```
}
```

# DTO 만들기

- [src - main - java - com.example.firstproject]에서 마우스 오른쪽 버튼 클릭
- 컨텍스트 메뉴에서 [New - Package] 선택
  - 새 패키지 이름, 'dto' 입력



- [com.example.firstproject]의 dto 패키지에서 마우스 오른쪽 버튼 클릭
- 컨텍스트 메뉴에서 [New - Class] 선택
  - 새 클래스 이름: ArticleForm

# DTO 만들기

- ArticleForm.java 편집

```
package com.example.firstproject.dto;

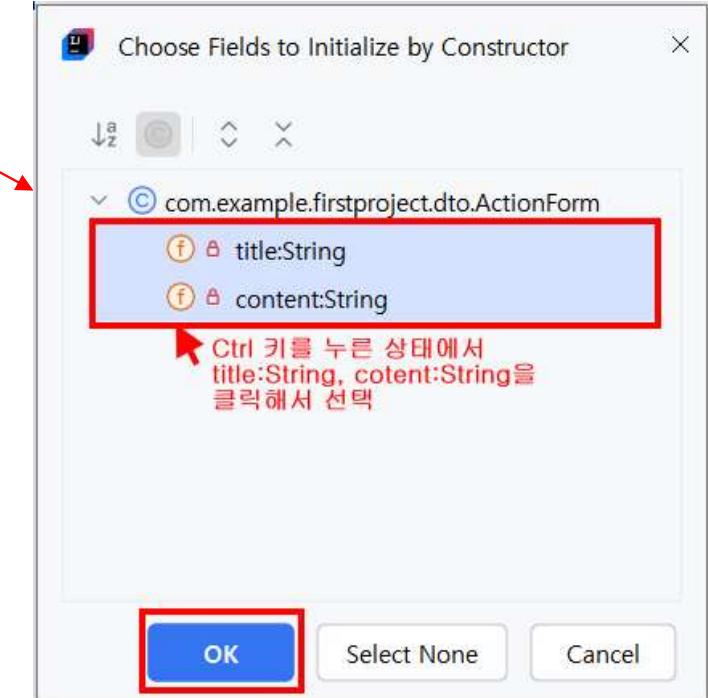
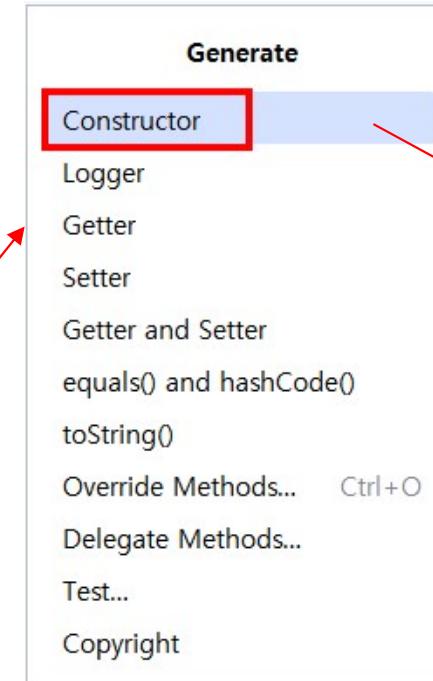
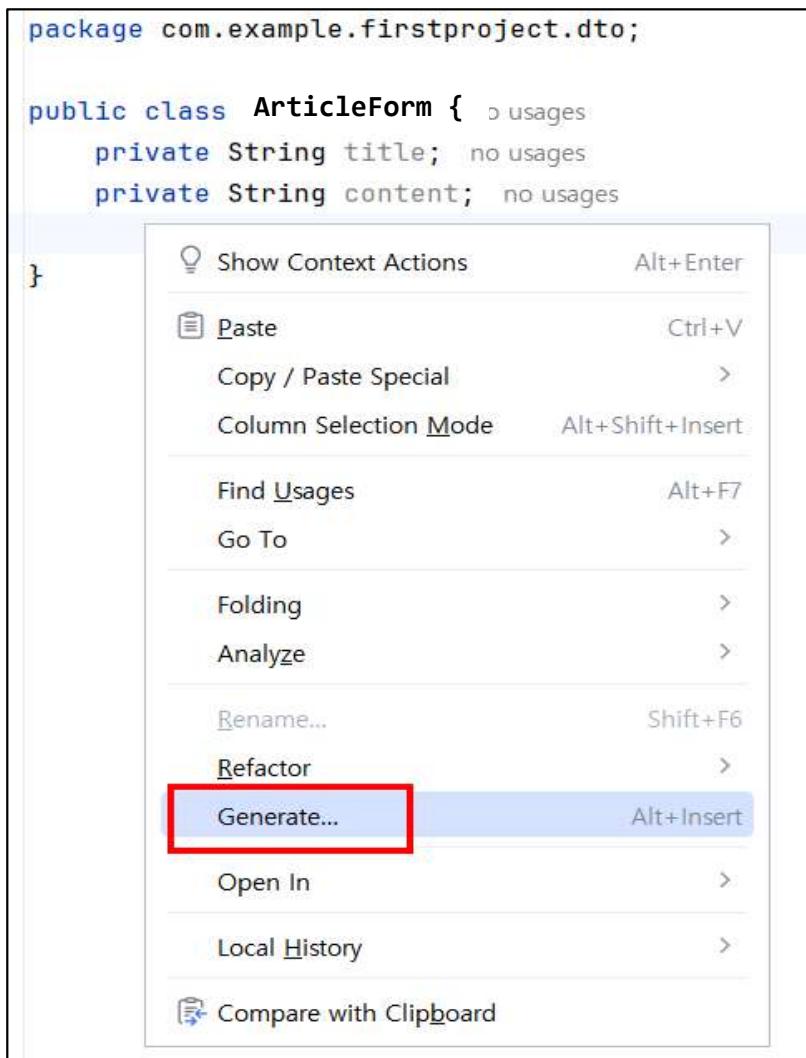
public class ArticleForm {
    private String title;
    private String content;
}
```

# DTO 만들기

- ArticleForm.java 편집

- 생성자 추가

- ✓ Content 필드 다음 줄에서 마우스 오른쪽 버튼 클릭, [Generate - Constructor]



# DTO 만들기

- 생성자 추가 확인

ArticleForm.java

```
package com.example.firstproject.dto;

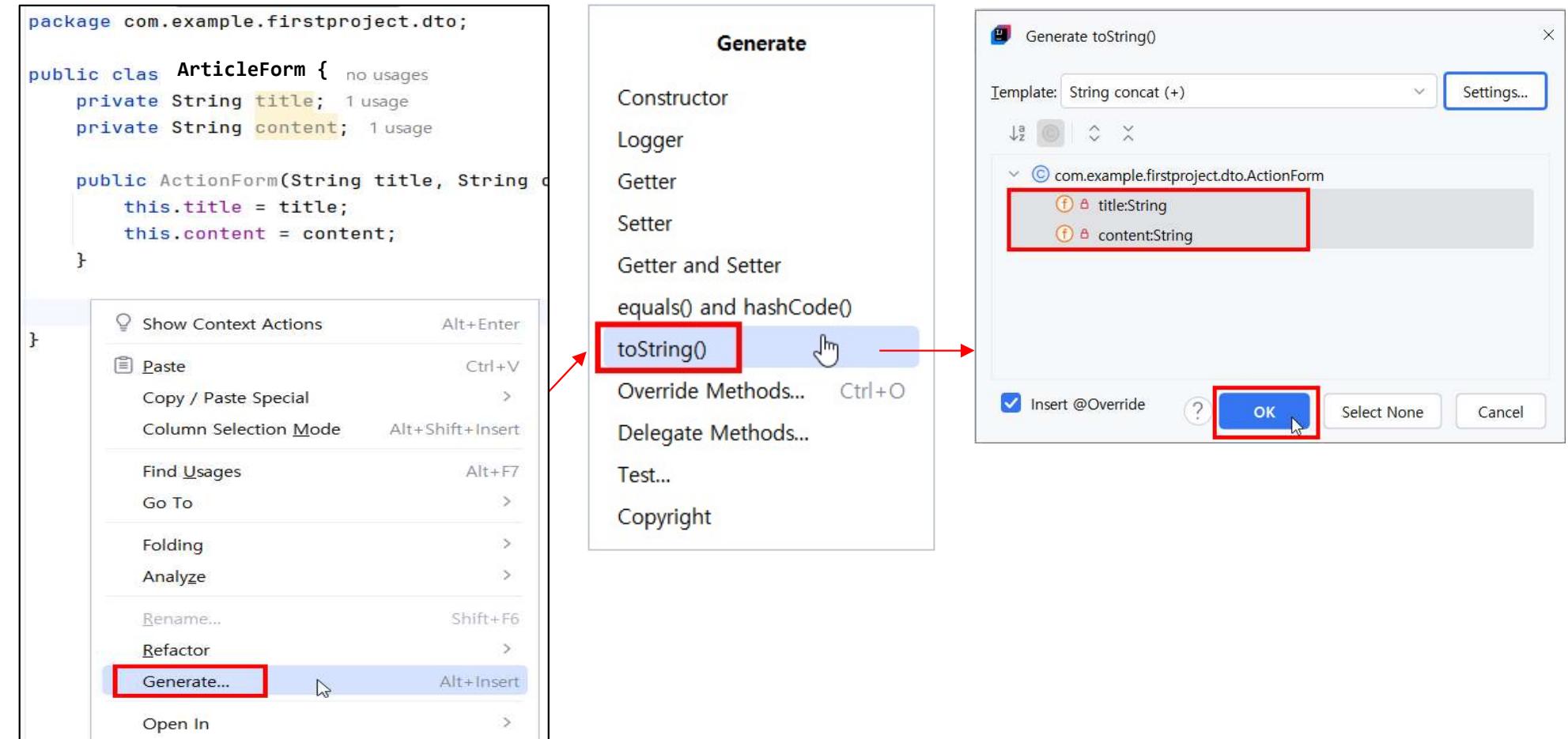
public class ArticleForm {
    private String title;
    private String content;

    public ArticleForm(String title, String content) {
        this.title = title;
        this.content = content;
    }
}
```

# DTO 만들기

- ArticleForm.java에 `toString()` 추가

- ArticleForm.java의 생성자 아래 줄에서 마우스 오른쪽 버튼 클릭
- 컨텍스트 메뉴에서 `[toString()]` 클릭
- Generate `toString()` 창에서 `title:String, content:String`이 선택된 상태에서 `[OK]` 버튼 클릭



# DTO 만들기

- ArticleForm.java에 toString() 생성 확인

```
package com.example.firstproject.dto;

public class ArticleForm {
    private String title;
    private String content;

    public ActionForm(String title, String content) {
        this.title = title;
        this.content = content;
    }

    @Override
    public String toString() {
        return "ActionForm{" +
            "title='" + title + '\'' +
            ", content='" + content + '\'' +
            '}';
    }
}
```

# 전송받은 폼 데이터를 DTO에 담기

- ArticleController.java의 createArticle() 수정
  - 폼 데이터를 createArticle의 매개변수로 받음
    - ✓ createArticle(ArticleForm form)
  - 전송받은 폼데이터가 잘 담겼는지 확인
    - ✓ form.toString() 호출

(중략)

```
import com.example.firstproject.dto.ArticleForm;
```

(중략)

```
@Controller
```

```
public class ArticleController {
```

```
    @PostMapping("/articles/create")
```

```
    public String createArticle(ArticleForm form){
```

```
        System.out.println(form.toString());
```

```
        return "";
```

```
}
```

```
}
```

# 입력 폼과 DTO 필드 연결하기

- new.mustache 파일 수정
  - <input> 태그에 name=“title” 추가
  - <textarea> 태그에 name=“content” 추가

```
{{>layouts/header}}
```

```
<form class="container" action="/articles/create" method="post">
  <div class="mb-3">
    <label class="form-label">제목</label>
    <input type="text" class="form-control" name="title">
  </div>                                         DTO의 title 필드와 연결
  <div class="mb-3">
    <label class="form-label">내용</label>
    <textarea class="form-control" rows="3" name="content"></textarea>
  </div>                                         DTO의 content 필드와 연결
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

```
{{>layouts/footer}}
```

# 전송된 폼 데이터가 DTO에 담겼는지 확인

- 서버 재시작
- localhost:8080/articles/new 접속

localhost:8080/articles/new

Navbar

제목

Hello

내용

How are you?

Submit

© CloudStudying | [Privacy](#) | [Terms](#)

localhost:8080/articles/create

## Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Sep 09 15:15:03 KST 2025  
There was an unexpected error (type=Not Found, status=404).

- IntelliJ 하단 run 탭  
form에서 입력한  
'Hello', 'How are you?' 출력 확인

```
2025-09-09T15:07:20.967+09:00  WARN 29132 --- [nio-8080-exec-1] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/error] onto handler 'org.springframework.boot.autoconfigure.web.error.BasicErrorController#errorPage'
2025-09-09T15:07:21.654+09:00  INFO 29132 --- [nio-8080-exec-1] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/error] onto handler 'org.springframework.boot.autoconfigure.web.error.BasicErrorController#errorPage'
2025-09-09T15:07:21.670+09:00  INFO 29132 --- [nio-8080-exec-1] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/error] onto handler 'org.springframework.boot.autoconfigure.web.error.BasicErrorController#errorPage'
2025-09-09T15:14:19.139+09:00  INFO 29132 --- [nio-8080-exec-1] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/error] onto handler 'org.springframework.boot.autoconfigure.web.error.BasicErrorController#errorPage'
2025-09-09T15:14:19.139+09:00  INFO 29132 --- [nio-8080-exec-1] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/error] onto handler 'org.springframework.boot.autoconfigure.web.error.BasicErrorController#errorPage'
2025-09-09T15:14:19.143+09:00  INFO 29132 --- [nio-8080-exec-1] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/error] onto handler 'org.springframework.boot.autoconfigure.web.error.BasicErrorController#errorPage'

ActionForm{title='Hello', content='How are you?'}
```

example > firstproject > controller > ArticleController

# DTO를 데이터베이스에 저장하기

- H2 데이터베이스
  - 자바로 작성된 관계형 데이터베이스 관리 시스템
  - 스프링 부트가 지원하는 인메모리 DB
    - ✓ 애플리케이션 재구동 때마다 초기화
  - 장점
    - ✓ 용량이 매우 가벼움
    - ✓ 설치가 필요 없음
    - ✓ 웹기반 콘솔을 제공하여 개발용 로컬 DB, 테스트 환경에 용이
- 스프링 부트에서 H2 DB 사용
  - 스프링 부트 프로젝트 생성시 Dependencies에서 추가

# JPA(Java Persistence API)

- 스프링 부트는 자바 언어를 사용하지만 DB는 SQL 언어를 사용
- ORM(Object Relational Mapping)은 SQL를 사용하지 않고 DB를 관리할 수 있는 도구
  - DB의 테이블을 자바 클래스로 만들어 관리
- JPA는 자바 언어로 DB에 명령을 내리는 API로서 ORM 기술의 표준을 사용함
- 스프링 부트는 JPA를 사용하여 데이터베이스를 관리함



- 하이버네티트(Hibernate)
  - JPA의 인터페이스를 구현한 실제 클래스로서 자바의 ORM 프레임워크
  - JPA는 인터페이스 모음이므로, 이를 구현한 실제 클래스가 필요함

# JPA(Java Persistence API)

- JPA 특징
  - 객체 지향적 접근: DB 테이블을 객체로 매팅하여 객체지향적으로 접근
  - 쿼리 생성의 자동화: JPA가 자동으로 SQL 쿼리를 생성하여 DB로 전송함
  - 캐시 및 지연 로딩으로 성능 향상
- JPA의 핵심
  - 엔티티(entity)
    - ✓ 자바 객체를 DB가 이해할 수 있도록 만든 것
    - ✓ 엔티티를 기반으로 테이블이 만들어짐
  - 리파지토리(repository)
    - ✓ 엔티티가 DB 속 테이블에 저장 및 관리될 수 있도록 하는 인터페이스

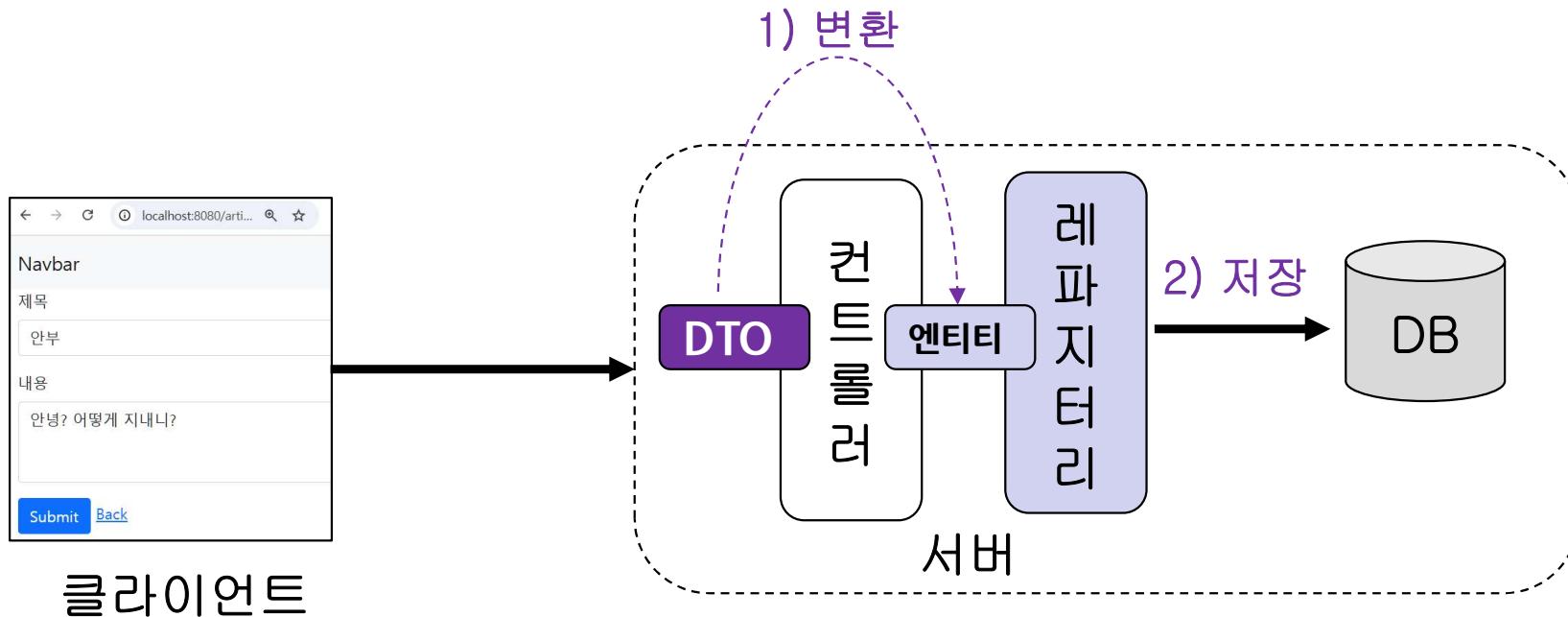
# JPA(Java Persistence API)

- JPA에서 사용하는 주요 어노테이션
  - `@Entity`
    - ✓ `@Entity`가 선언된 클래스를 기반으로 DB에 테이블 생성
  - `@Id`
    - ✓ `@Id`가 선언된 필드를 테이블의 기본 키(primary key)로 매핑
  - `@GeneratedValue`
    - ✓ `@GeneratedValue`가 선언된 필드가 auto increment 컬럼임을 알려줌
  - `@column`
    - ✓ `@column`이 선언된 필드는 테이블의 컬럼에 매핑

# DTO를 DB에 저장

- DTO를 DB에 저장하는 과정

- 1) DTO를 엔티티로 변환하기
- 2) 리파지토리를 이용해 엔티티를 DB에 저장하기



# DTO를 DB에 저장

- controller/ArticleController.java에 DTO를 DB에 저장하는 코드 삽입

```
@Controller  
public class ArticleController {  
    (중략)  
  
    @PostMapping("/articles/create")  
    public String createArticle(ArticleForm form){  
        System.out.println(form.toString());  
        // 1. DTO를 엔티티로 변환  
        // 2. 리파지터리로 엔티티를 DB에 저장  
        return "";  
    }  
}
```

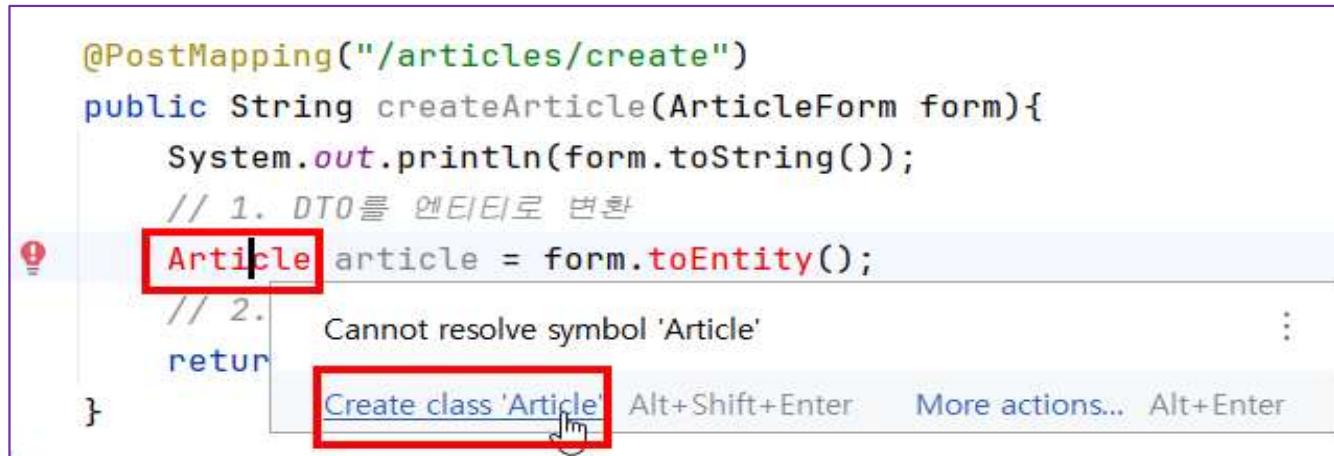
# DTO를 엔티티로 변환하기

- 엔티티 클래스 만들기: Article
  - controller/ArticleController.java 의 createArticle() 수정

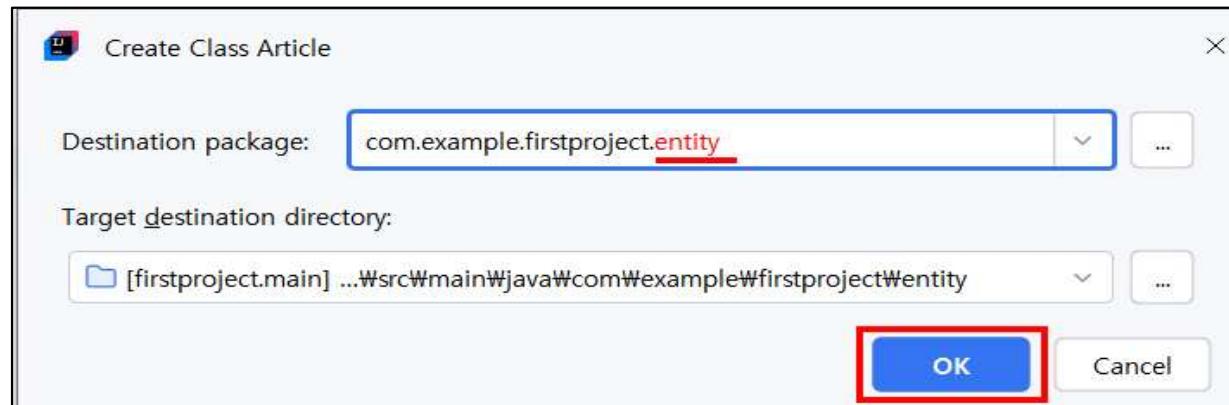
```
@PostMapping("/articles/create")
public String createArticle(ArticleForm form){
    System.out.println(form.toString());
    // 1. DTO를 엔티티로 변환
    Article article = form.toEntity();
    // 2. 리파지터리로 엔티티를 DB에 저장
    return "";
}
```

# DTO를 엔티티로 변환하기

- 빨간색으로 표시된 Article 위에 마우스를 클릭한 후, 잠시 기다리면 인텔리제이가 Article 클래스를 만들 수 있는 링크를 보여줌
- [Create class 'Article']를 클릭



- Destination package 설정 : com.example.firstproject.entity



# 엔티티 만들기

- com.example.firstproject.entity 패키지와 Article 클래스가 만들어졌는지 확인

The screenshot shows the IntelliJ IDEA interface. On the left, the Project tool window displays the project structure:

- build
- gradle
- src
  - main
    - java
      - com.example.firstproject
        - controller
          - ArticleController
          - FirstController
        - dto
          - ArticleForm
        - entity
          - Article

A red box highlights the "entity" folder under "main/java/com.example.firstproject".

The right pane shows the content of the Article.java file:

```
package com.example.firstproject.entity;  
public class Article {  
}
```

# 엔티티 만들기

- Article 클래스 코드 작성

entity/Article.java

```
package com.example.firstproject.entity;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;

@Entity
public class Article {
    @Id
    @GeneratedValue
    private Long id;
    @Column
    private String title;
    @Column
    private String content;
}
```

# 엔티티 만들기

- Article 클래스에 생성자 추가
  - Content 필드 아래 줄에서 [마우스 오른쪽 버튼] 클릭
  - 컨텍스트 메뉴에서 [Generate - Constructor] 선택
  - id:Long, title:String, content:String을 모두 선택 후 [OK] 버튼 클릭
- Article 클래스에 `toString()` 추가
  - 생성자 아래 줄에서 [마우스 오른쪽 버튼] 클릭
  - 컨텍스트 메뉴에서 [Generate - `toString()`] 선택
  - id:Long, title:String, content:String을 모두 선택 후 [OK] 버튼 클릭

# 엔티티 만들기

- 생성자와 `toString()`이 추가된 Article 클래스

```
(중략)
public class Article {
    @Id
    @GeneratedValue
    private Long id;
    @Column
    private String title;
    @Column
    private String content;

    public Article(Long id, String title, String content) {
        this.id = id;
        this.title = title;
        this.content = content;
    }

    @Override
    public String toString() {
        return "Article{" +
            "id=" + id +
            ", title='" + title + '\'' +
            ", content='" + content + '\'' +
            '}';
    }
}
```

# 엔티티 만들기

- ArticleController.java에 빨간색으로 오류가 표시되었던 Article은 오류가 사라진 것 확인

Controller/ArticleController.java

```
import com.example.firstproject.entity.Article;  
  
(중략)  
public String createArticle(ArticleForm form){  
    System.out.println(form.toString());  
    // 1. DTO를 엔티티로 변환  
    Article article = form.toEntity();  
    // 2. 리파지터리로 엔티티를 DB에 저장  
    return "";  
}
```

# DTO를 엔티티로 변환하기

- **toEntity() 메소드 만들기**

- DTO인 ArticleForm 객체를 엔티티 객체로 변환

- toEntity()에 마우스를 올리고 잠시 기다리면

Create method 'toEntity' in 'ArticleForm' 링크가 포함된 창이 뜸

The screenshot shows two panels of an IDE. On the left, the code for `ArticleController.java` is displayed:

```
@PostMapping("/articles/create")
public String createArticle(ArticleForm form){
    System.out.println(form.toString());
    // 1. DTO를 엔티티로 변환
    Article article = form.toEntity();
    // 2. 리파지토리로 엔티티를 DB에
    return "";
}
```

A tooltip window is open over the `toEntity()` call in the controller code, containing the message: "Cannot resolve method 'toEntity' in 'ArticleForm'" and a link "Create method 'toEntity' in 'ArticleForm'". A red arrow points from this tooltip to the corresponding code in `ArticleForm.java` on the right.

The right panel shows the `ArticleForm.java` file:

```
package com.example.firstproject.dto;

import com.example.firstproject.entity.Article;

public class ArticleForm {
    private String title;
    private String content;

    public ArticleForm(String title, String content) {
        this.title = title;
        this.content = content;
    }

    @Override
    public String toString() {
        return "ArticleForm{" +
            "title='" + title + '\'' +
            ", content='" + content + '\'' +
            '}';
    }
}

public Article toEntity() {
```

# DTO를 엔티티로 변환하기

- **toEntity() 코딩**

```
public Article toEntity() {  
    return new Article( id: null, title, content);  
}
```

- ArticleController.java에 가 보면 toEntity()의 빨간색으로 표시된 오류가 사라진 것 확인

# 리파지토리로 엔티티를 DB에 저장하기

- ArticleController.java 편집

- ArticleRepository 타입의 articleRepository 객체 선언
- article 엔티티를 저장하고, 저장된 엔티티를 반환(saved)하는 articleRepository.saved() 메소드 호출

```
@Controller
public class ArticleController {

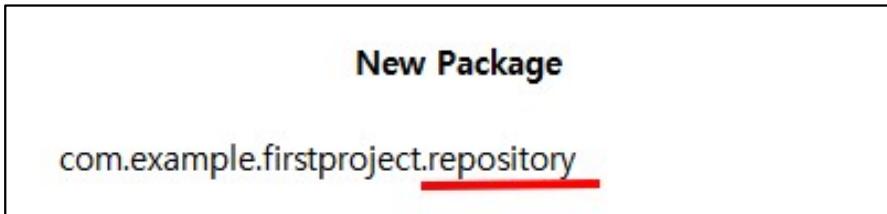
    private ArticleRepository articleRepository; //articleRepository 객체 선언
    // (중략)

    @PostMapping("/articles/create")
    public String createArticle(ArticleForm form){
        System.out.println(form.toString());
        // 1. DTO를 엔티티로 변환
        Article article = form.toEntity();
        // 2. 리파지토리로 엔티티를 DB에 저장
        Article saved = articleRepository.save(article); // article 엔티티를 저장하고
        // saved 객체를 반환
        return "";
    }
}
```

# 리파지토리 만들기

- 리파지토리 만들기

- 프로젝트 탐색기 com.example.firstproject 에서 [마우스 오른쪽 버튼] 클릭
- 컨텍스트 메뉴에서 [New - Package] 선택
  - ✓ 새 패키지 이름: repository



- repository 패키지에서 [New - Java Class] 선택
  - ✓ Interface 선택 후, 새 인터페이스 이름: ArticleRepository

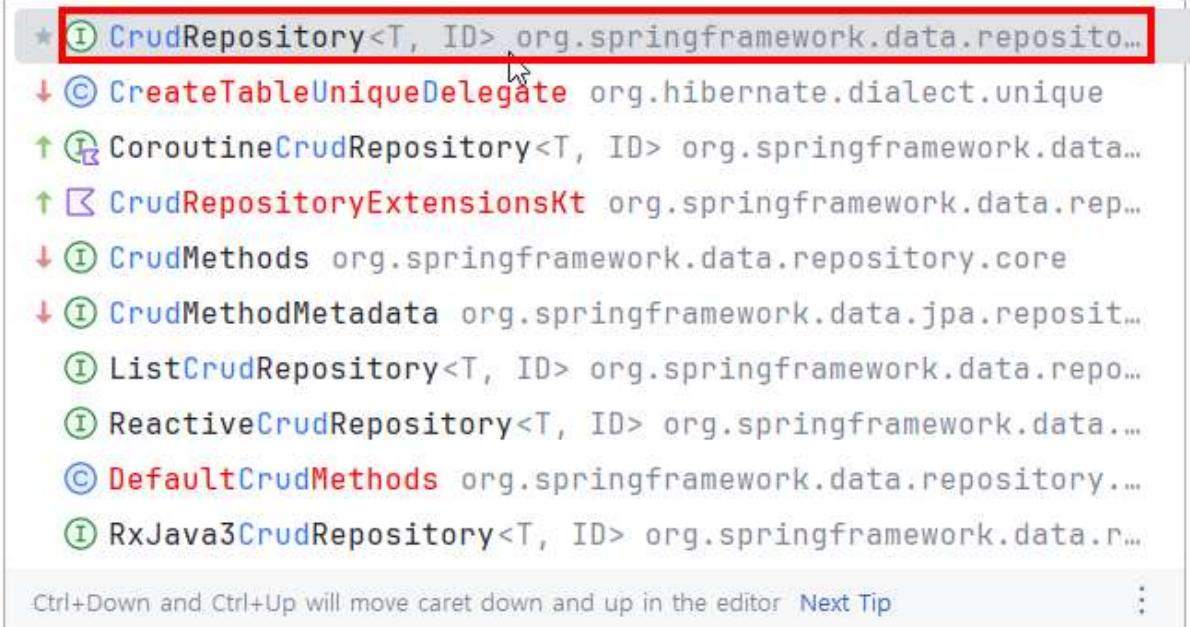


# 리파지토리 만들기

- ArticleRepository.java 편집
  - CrudRepository 인터페이스를 상속

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

public interface ArticleRepository extends CRUD<...> {
}
```



The screenshot shows the IntelliJ IDEA code editor with a code completion dropdown open. The code being typed is 'extends CRUD<...>'. A tooltip at the bottom says 'Ctrl+Down and Ctrl+Up will move caret down and up in the editor' and 'Next Tip'. The dropdown list includes:

- ① CrudRepository<T, ID> org.springframework.data.repository.CrudRepository
- ② CreateTableUniqueDelegate org.hibernate.dialect.unique.CreateTableUniqueDelegate
- ③ CoroutineCrudRepository<T, ID> org.springframework.data.repository.support.CoroutineCrudRepository
- ④ CrudRepositoryExtensionsKt org.springframework.data.repository.kotlin.extensions.CrudRepositoryExtensionsKt
- ⑤ CrudMethods org.springframework.data.repository.core.CrudMethods
- ⑥ CrudMethodMetadata org.springframework.data.jpa.repository.CrudMethodMetadata
- ⑦ ListCrudRepository<T, ID> org.springframework.data.repository.ListCrudRepository
- ⑧ ReactiveCrudRepository<T, ID> org.springframework.data.repository.reactive.ReactiveCrudRepository
- ⑨ DefaultCrudMethods org.springframework.data.repository.support.DefaultCrudMethods
- ⑩ RxJava3CrudRepository<T, ID> org.springframework.data.repository.support.RxJava3CrudRepository

# 리파지토리 만들기

- **CrudRepository<Article, Long>**

- Article: 관리 대상 엔티티의 타입(클래스 명)
- Long: 관리 대상 엔티티의 대푯값(primary key)

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

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

public interface ArticleRepository extends CrudRepository<Article, Long> {
}
```

# CrudRepository<T, ID> 인터페이스

- JPA에서 제공하는 인터페이스
- DB 엔티티에 대해 기본적인 CRUD(Create, Read, Update, Delete) 작업을 쉽게 처리할 수 있게 함
  - CrudRepository를 상속하면, 별도의 SQL 없이 데이터 접근이 가능
- 타입 매개변수 <T, ID>
  - T: 엔티티 클래스
  - ID: 엔티티의 Primary Key의 타입
- 주요 메소드
  - <S extends T> S save(S entity) : 엔티티 저장 또는 업데이트, 저장된 엔티티 객체 반환
  - Optional<T> findById(ID id) : ID로 엔티티 조회
  - boolean existsById(ID id) : 해당 ID의 존재 여부
  - long count() : 전체 엔티티 수 반환
  - void deleteById(ID id) : ID로 엔티티 삭제
  - void delete(T entity) : 특정 엔티티 삭제

# 리파지토리로 엔티티를 DB에 저장하기

- ArticleController.java

- import com.example.firstproject.repository.ArticleRepository; 삽입 확인
- save() 메소드 빨간색 오류 사라짐 확인

```
import com.example.firstproject.dto.ArticleForm;
import com.example.firstproject.entity.Article;
import com.example.firstproject.repository.ArticleRepository;
    (중략)

@Controller //컨트롤러 선언
public class ArticleController {
    (중략)

    @PostMapping("/articles/create")
    public String createArticle(ArticleForm form){
        System.out.println(form.toString());
        // 1. DTO를 엔티티로 변환
        Article article = form.toEntity();
        // 2. 리파지토리로 엔티티를 DB에 저장
        Article saved = articleRepository.save(article); // article 엔티티를 저장하고
                                                        // saved 객체를 반환
        return "";
    }
}
```

# 리파지토리로 엔티티를 DB에 저장하기

- 객체 주입하기:

- `@Autowired` : 스프링 부트가 이미 생성해 놓은 지정된 객체를 지정된 변수에 자동으로 주입해 주는 어노테이션
- 스프링 부트가 미리 생성해 놓은 `ArticleRepository` 객체를 `@Autowired`이 붙어 있는 `articReposiroty` 변수에 연결

```
    @Autowired  
    private ArticleRepository articleRepository; //articleRepository 객체 선언  
  
    @GetMapping("/articles/new") //URL(localhost:8080/articles/new) 요청 접수  
    public String newArticleForm(){  
        return "articles/new"; //반환값으로 뷰 페이지(articles/new.mustache)의 이름  
    }  
  
    @PostMapping("/articles/create")  
    public String createArticle(ArticleForm form){  
        System.out.println(form.toString());  
        // 1. DTO를 엔티티로 변환  
        Article article = form.toEntity();  
        // 2. 리파지토리로 엔티티를 DB에 저장  
        Article saved = articleRepository.save(article); // article 엔티티를 저장하고  
                                                        // saved 객체를 반환
```

# 의존성 주입(DI: Dependency Injection)

- 스프링 프레임워크의 핵심 개념
- 의존성(Dependency)
  - A 클래스가 다른 B 클래스의 기능을 사용할 때, A는 B에 의존한다고 함
- 주입(Injection)
  - 의존하는 객체를 직접 생성하지 않고, 외부에서 전달 받는 방식
  - A 클래스에서 직접 new B()로 객체를 생성하지 않고, 스프링 프레임워크가 생성해 놓은 B 객체를 A에 넣어주는 것
- 특징
  - 클래스 결합도가 감소
  - 테스트 용이
  - IoC(Inversion of Control)
    - ✓ 객체의 생성 및 생명주기 관리 등의 제어권을 개발자가 아닌 스프링이 담당

# 리파지토리로 엔티티를 DB에 저장하기

- 데이터 저장 확인

```
@PostMapping("/articles/create")
public String createArticle(ArticleForm form){
    System.out.println(form.toString());
    // 1. DTO를 엔티티로 변환
    Article article = form.toEntity();
    System.out.println(article.toString()); //DTO가 엔티티로 잘 변환되는지 확인 출력
    // 2. 리파지토리로 엔티티를 DB에 저장
    Article saved = articleRepository.save(article);
    System.out.println(saved.toString()); //article이 DB에 잘 저장되는지 확인 출력
    return "";
}
```

# 리파지토리로 엔티티를 DB에 저장하기

- 서버 재시작
- localhost:8080/articles/new 접속

The screenshot illustrates the process of saving an entity to the database using a repository. It shows a browser with two tabs and a terminal window.

- Left Tab:** Shows a form for creating a new article. The title field contains "aaaa" and the content field contains "1111". A red box highlights the "Submit" button.
- Right Tab:** Shows a Whitelabel Error Page with the URL `localhost:8080/articles/create`. The page displays the following message:

```
This application has no explicit mapping for /error, so you are seeing this as a fallback.  
Thu Sep 18 13:16:15 KST 2025  
There was an unexpected error (type=Not Found, status=404).
```

A red arrow points from the "Submit" button in the left tab to this error page.
- Terminal Window:** Labeled "콘솔 확인" (Console Confirmation). It shows the following log entries:

```
2025-09-18T13:16:14.700+09:00  INFO 15604 --- [nio-8080-e]  
2025-09-18T13:16:14.700+09:00  INFO 15604 --- [nio-8080-e]  
2025-09-18T13:16:14.702+09:00  INFO 15604 --- [nio-8080-e]  
ActionForm{title='aaaa', content='1111'}  
Article{id=null, title='aaaa', content='1111'}  
Article{id=1, title='aaaa', content='1111'}
```

The last three lines are highlighted with a red box.

# H2 데이터베이스에 접속하기

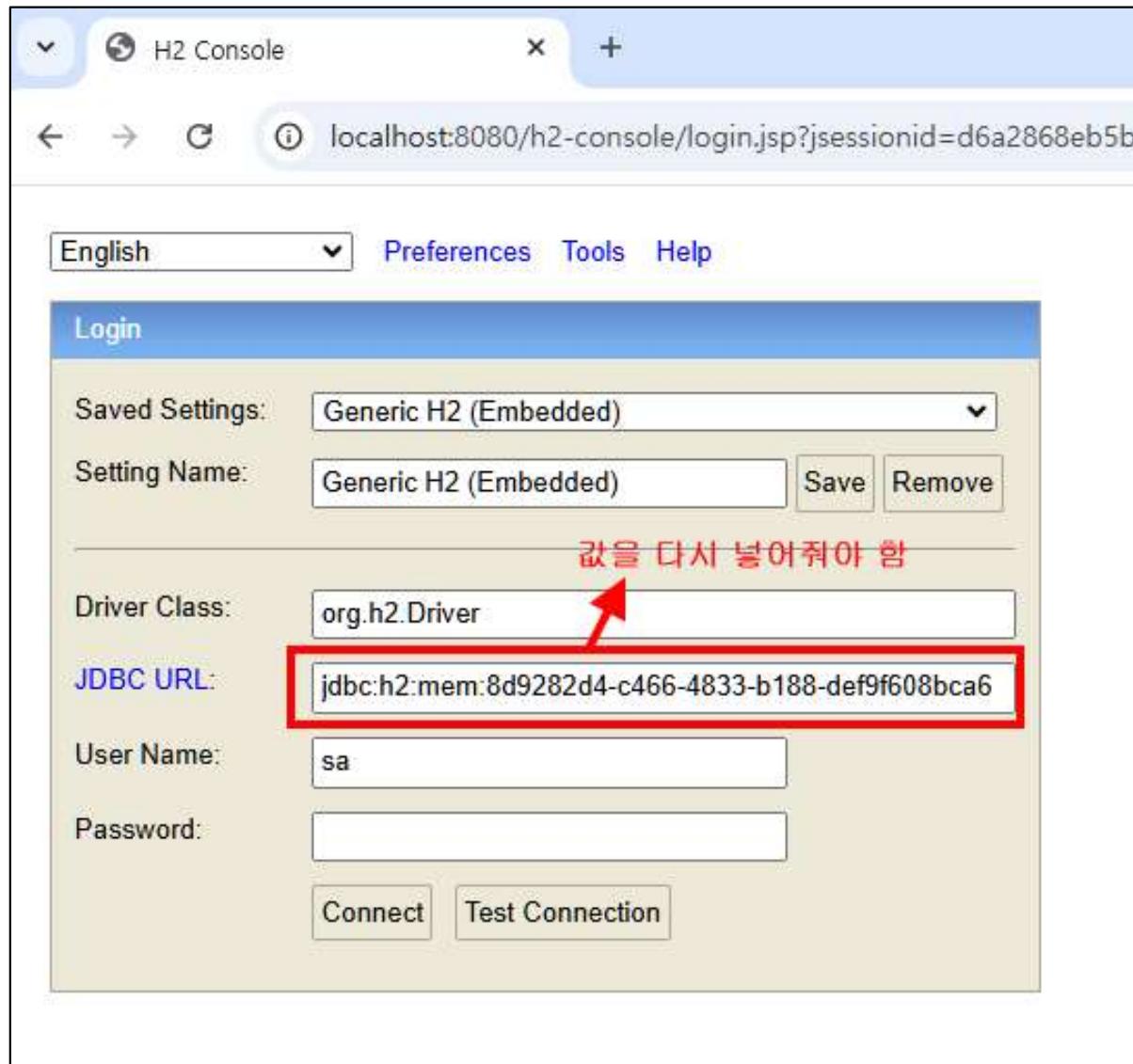
- H2 DB에 웹 콘솔로 접속하기

- 프로젝트 탐색기, [src - main - resources]에서 application.properties 파일 열기
- application.properties에 *spring.h2.console.enabled=true* 추가

```
spring.application.name=firstproject
server.servlet.encoding.force=true
spring.h2.console.enabled=true
```

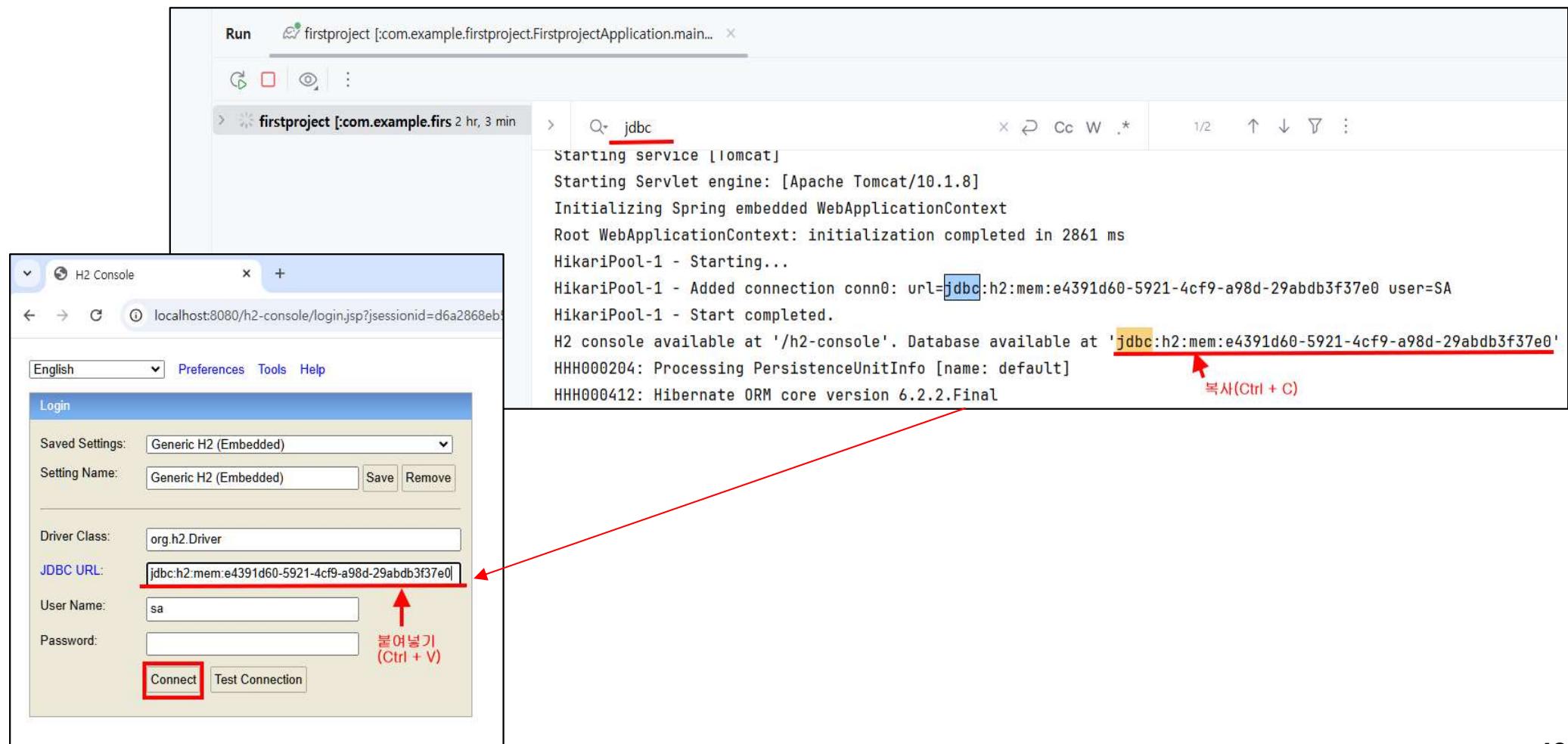
# H2 데이터베이스에 접속하기

- 서버 재시작
- 웹 브라우저에서 *localhost:8080/h2-console* 접속



# H2 데이터베이스에 접속하기

- [Run] 탭에서 Ctrl+F를 클릭후 나타난 검색창에 *jdbc* 입력
- H2 DB url(*jdbc:h2:mem:e4…*) 복사, h2 웹 콘솔의 JDBC URL에 붙여넣기 후, [connect] 버튼 클릭
  - H2는 인메모리 DB이므로, 서버를 재시작 할 때마다 접속 url이 달라짐



# H2 웹 콘솔에서 데이터 조회하기

- H2 DB 웹 콘솔

The screenshot shows the H2 Console web interface running at `localhost:8080/h2-console/login.do?jsessionid=b4a2f49904d532d6472e2cc2e4cfa1c3`. The left sidebar displays the database schema:

- ARTICLE** table with columns: ID, CONTENT, TITLE, Indexes
- INFORMATION\_SCHEMA**
- Sequences
- Users
- H2 2.1.214 (2022-06-13)

The main area contains the following sections:

- Important Commands** table:

	Displays this Help Page
	Shows the Command History
	Executes the current SQL statement
	Executes the SQL statement defined by the text selection
	Auto complete
	Disconnects from the database
- Sample SQL Script** table:

<a href="#">Delete the table if it exists</a>	DROP TABLE IF EXISTS TEST;
<a href="#">Create a new table with ID and NAME columns</a>	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
<a href="#">Add a new row</a>	INSERT INTO TEST VALUES(1, 'Hello');
<a href="#">Add another row</a>	INSERT INTO TEST VALUES(2, 'World');
<a href="#">Query the table</a>	SELECT * FROM TEST ORDER BY ID;
<a href="#">Change data in a row</a>	UPDATE TEST SET NAME='Hi' WHERE ID=1;
<a href="#">Remove a row</a>	DELETE FROM TEST WHERE ID=2;
<a href="#">Help</a>	HELP ...
- Adding Database Drivers** section: Additional database drivers can be registered by adding the Jar file location of the driver to the environment variable (Windows): to add the database driver library C:/Programs/hsqldb/lib/hsqldb.jar, set the environment variable

# H2 웹 콘솔에서 데이터 조회하기

- Article 테이블 조회

The screenshot shows the H2 Web Console interface. On the left, the database tree view shows a connection named 'jdbc:h2:mem:e4391d60-5921-4c' with a selected 'ARTICLE' table. A red box highlights the 'ARTICLE' table, and a red arrow points from it to the 'Run' button in the top toolbar. Another red box highlights the 'Run' button, and a red arrow points from it to the SQL statement input field. The SQL statement 'SELECT \* FROM ARTICLE' is entered in the input field. Below the input field, the results are displayed in a table:

ID	CONTENT	TITLE
(no rows, 7 ms)		

At the bottom left of the results area is an 'Edit' button.

# H2 웹 콘솔에서 데이터 조회하기

- Article 테이블에 레코드 추가
  - localhost:8080/articles/new 접속, 데이터 입력 후, [Submit]

Navbar

제목  
aaaa

내용  
1111

Submit

© CloudStudying | [Privacy](#) | [Terms](#)

Navbar

제목  
bbbb

내용  
2222

Submit

© CloudStudying | [Privacy](#) | [Terms](#)

# H2 웹 콘솔에서 데이터 조회하기

- Article 테이블 조회

The screenshot shows the H2 Web Console interface. On the left, there's a sidebar with database and schema navigation. The main area has a toolbar at the top with various icons and dropdowns for settings like 'Auto commit' (checked), 'Max rows' (set to 1000), and 'Auto complete' (set to Off). Below the toolbar, a connection URL is displayed: 'jdbc:h2:mem:e4391d60-5921-4c11'. A red box highlights the 'Run' button, which is also connected by a red arrow to the executed SQL statement in the central panel.

SQL statement:

```
SELECT * FROM ARTICLE;
```

The results of the query are shown in a table:

ID	CONTENT	TITLE
1	1111	aaaa
2	2222	bbbb

(2 rows, 1 ms)

At the bottom of the results panel is an 'Edit' button.