

DB연동

1) 의존성 추가

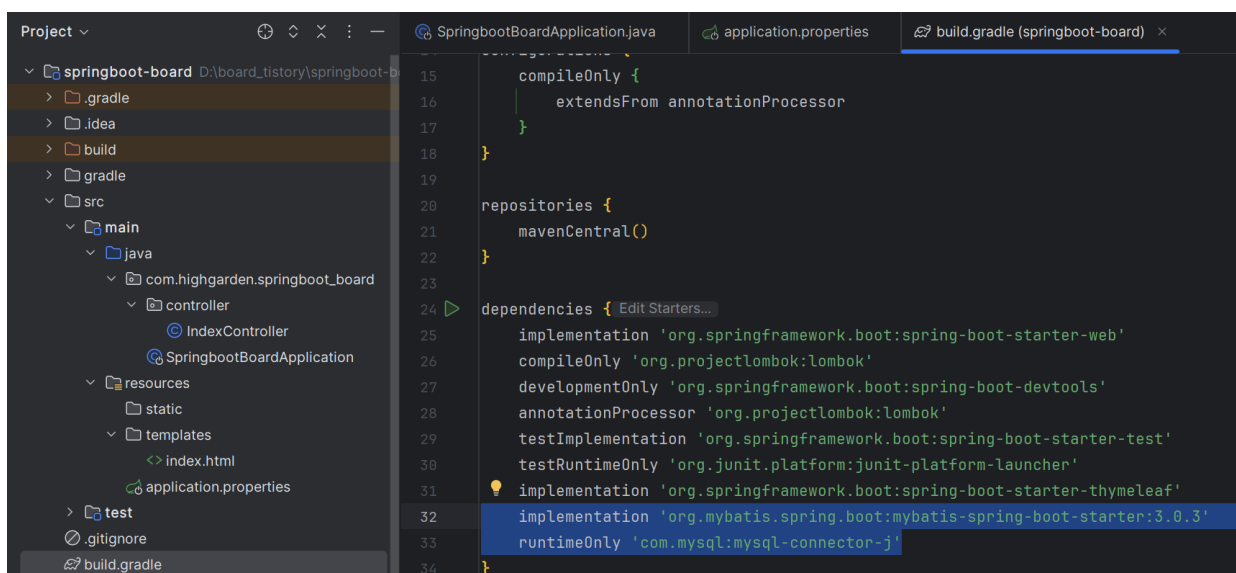
스프링 부트 서버와 mysql을 각각 세팅하였지만 이 둘을 연결해주기 위해서는 database의 드라이버가 필요하다. 기존 java프로젝트에서는 버전호환을 직접 확인하고 다운로드 받아서 적용시켜 주어야 했지만 스프링 부트는 build.gradle파일에 추가만 해주면 드라이버를 자동으로 세팅해준다.

데이터 액세스는 mybatis를 통해 하기로 했기 때문에 관련 의존성도 마찬가지로 추가해준다.

```
implementation 'org.mybatis.spring.boot:mybatis-spring-boot-starter:3.0.3'//mybatis
```

```
runtimeOnly 'com.mysql:mysql-connector-j'//mysql 드라이버
```

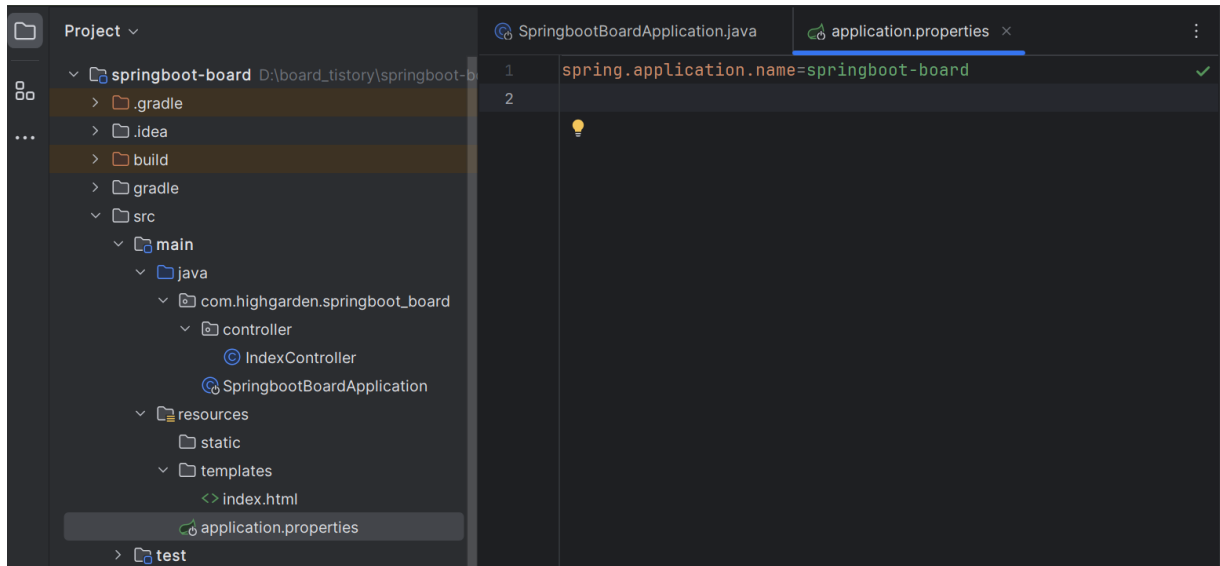
build.gradle의 dependencies 부분에 위 코드를 붙여넣어준다.



mysql, mybatis의존성추가

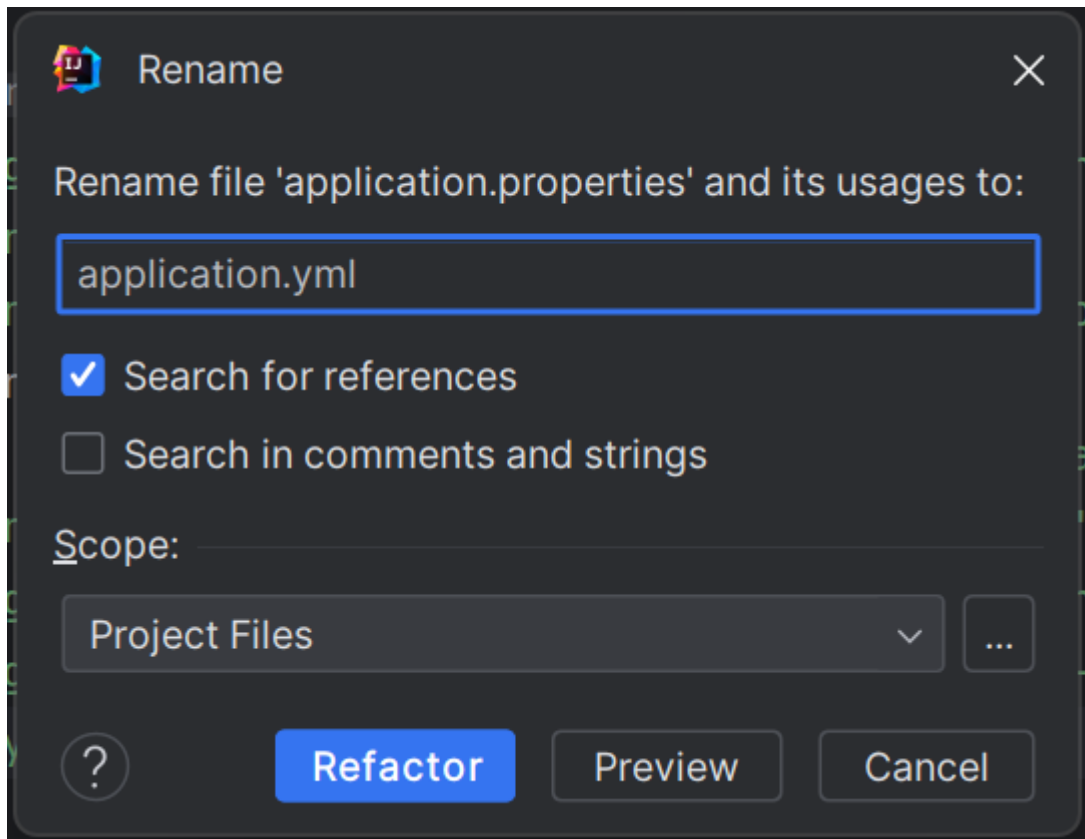
2) application.properties 수정

spring boot 프로젝트를 spring initializr를 통해 생성했을 때 application.properties라는 파일이 있는것을 확인할 수 있다. 어플리케이션에서 사용할 각종 설정정보들을 관리할 수 있도록 해주는 파일이다. 여기에 db연동관련 설정정보들을 작성할 수도 있지만 조금 더 직관적이고 보기 편한 yml형식으로 파일을 변경하겠다.



application.properties

application.properties의 이름을 application.yml로 변경하여 확장자를 변경해준다.



yml로 확

장자 변경

그다음 application.yml 파일에 다음과 같이 작성해준다.

#어플리케이션 포트 -> 디폴트가 8080

server:

port: 8080

database 연동 설정

spring:

datasource:

driver-class-name: com.mysql.cj.jdbc.Driver

url: jdbc:mysql://localhost:{port번호}/{db이름}?serverTimezone=Asia/Seoul&characterEncoding=UTF-8

username: {유저이름}

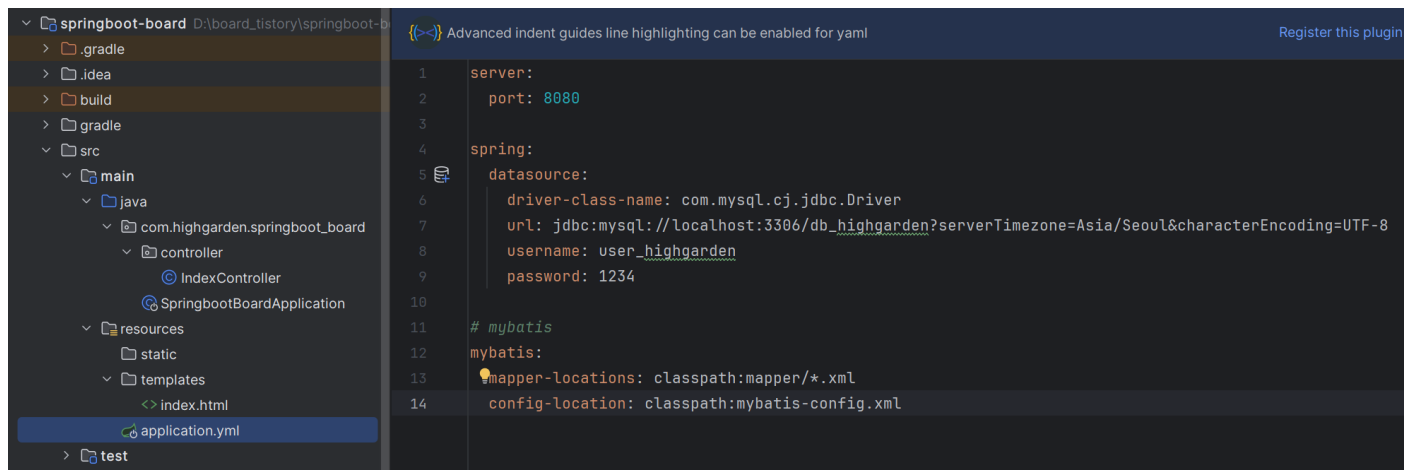
password: {비밀번호}

mybatis

mybatis:

mapper-locations: classpath:mapper/*.xml

config-location: classpath:mybatis-config.xml



application.yml작성완료

yml은 들여쓰기를 통해 설정정보를 읽어내기 때문에 들여쓰기가 매우 중요하다. 들여쓰기를 꼭 지켜서 작성하도록 하자.

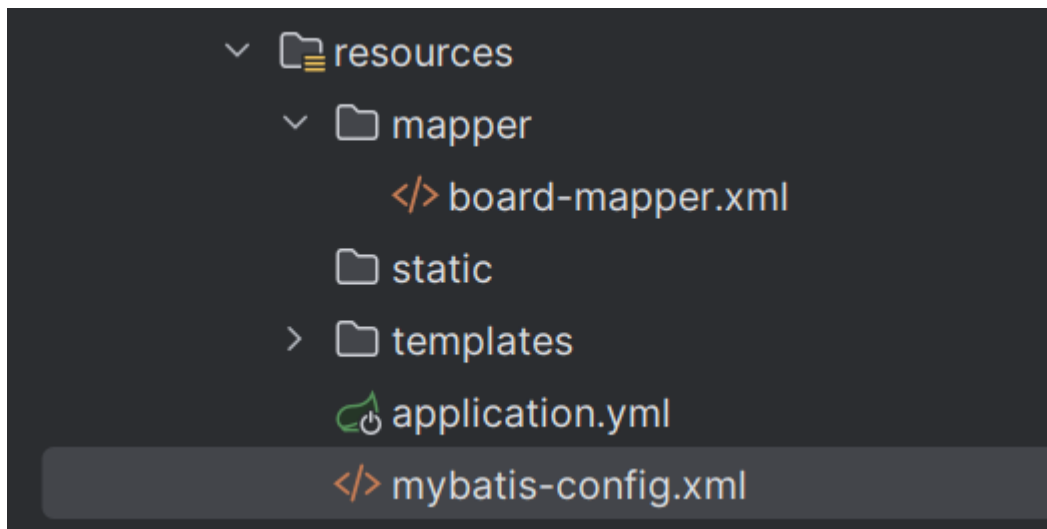
각종 설정정보를 기입하면 spring에서 이 파일을 읽고 사용할 수 있다. 기본적으로 스프링부트에 제공하는 아파치톰캣의 포트번호는 8080을 사용하지만 포트번호를 변경할 수도 있기 때문에 명시적으로 8080이라고 적어준다.

그 다음은 database 연동관련된 설정정보를 적어준다. 어떤 드라이버를 사용할 것인지와 그 database에 접속할 때 필요한 url, database접근 정보(db명, 유저명, 비밀번호)를 기입한다. mysql은 기본적으로 3306포트를 사용하도록 되어있다. 하지만 mysql을 버전에따라 여러 개 설치되어있다면 기존 3306포트와는 다르게 설정되어있을 수 있다. 현재 연동하고자 하는 포트번호가 3306이 맞는지 잘 확인하도록 하자.

마지막으로는 mybatis를 사용하기 위한 설정정보를 기입한다. mybatis를 사용하기 위해서는 mybatis-config와 mapper가 필요하다. 스프링에서 그 파일들을 식별하기 위한 경로정보를 기입해 준다.

3) mapper.xml, mybatis-config.xml 생성

xml파일을 생성해보도록 하자. sql문을 관리하게 될 board-mapper.xml을 resources/mapper 경로 아래에 생성해준다. mybatis-config.xml은 resources아래에 생성한다.



application.yml에 해당 xml파일들을 찾을 수 있도록 경로를 지정해주었기 때문에 경로를 틀리게 작성하면 스프링이 이 파일들을 읽을 수 없다. 정확하게 파일을 생성하도록 하자.

그 다음 각 xml 파일을 작성해본다.

mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
```

```
"http://mybatis.org/dtd/mybatis-3-config.dtd">
```

```
<configuration>
```

```
<typeAliases>

    <!--클래스 alias가 들어갈 공간-->

    <typeAlias type="com.highgarden.springboot_board.controller.IndexController"
alias="IndexController"> </typeAlias>

    <!--indexController예시-->

</typeAliases>

</configuration>
```

mybatis-config.xml에는 mybatis와 관련된 환경설정 정보들을 기입할 수 있는데 이 프로젝트에서는 typealiases만 사용하도록 한다. mybatis 에서 클래스정보를 가져오려면 전체경로를 사용해야 하는데 전체경로를 매번 적어주는 것은 귀찮기 때문에 축약어로 클래스를 읽어올 수 있도록 해주는 것이 typealiases이다.

예를 들어 **com.highgarden.springboot_board.controller.indexController** 라는 클래스에 접근하려면 기존에는 경로를 전부 써주어야하지만 alias설정을 해놓으면 **IndexController**라는 축약어로 접근이 가능해진다.

board-mapper.xml

```
<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

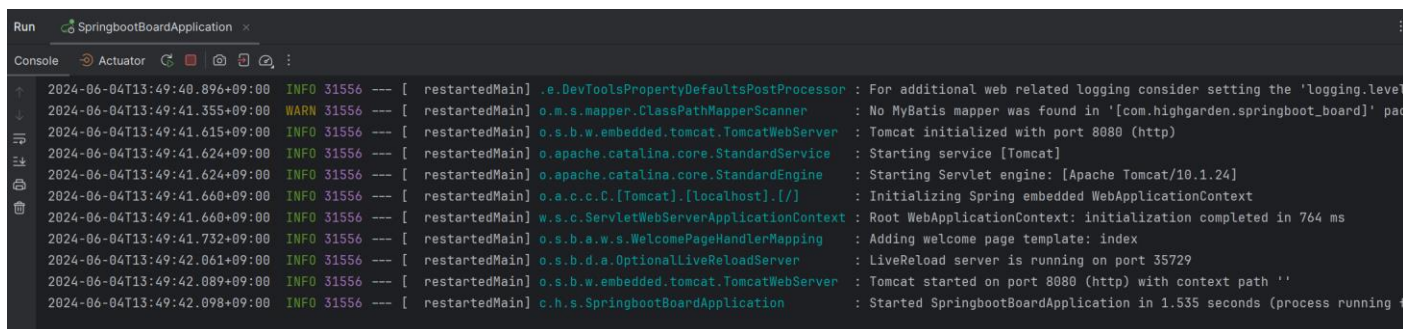
<mapper namespace="Board">

    <!-- sql문이 작성되는 공간 -->

</mapper>
```

실제로 db에서 사용할 sql들을 관리하는 파일이 mapper.xml 이다. 본 프로젝트에서는 게시판 만들 것이기 때문에 **board-mapper.xml**이라는 이름으로 파일을 만들었다. CRUD를 구현할 때 작성되는 sql을 여기에 작성한다.

여기까지 작성했으면 스프링 프로젝트를 실행해보자. 설정이 잘 되지 않았으면 에러가 발생할 것이다.

A screenshot of a Java IDE's console window showing the startup logs of a Spring Boot application named 'SpringbootBoardApplication'. The logs are timestamped and include log levels (INFO, WARN) and log messages. The application starts successfully, initializing Tomcat on port 8080 and the Spring application context. A warning message indicates that no MyBatis mapper was found, which is expected for this stage of development.

```
Run SpringbootBoardApplication
2024-06-04T13:49:40.896+09:00 INFO 31556 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.org.springframework.web' property to 'DEBUG'
2024-06-04T13:49:41.355+09:00 WARN 31556 --- [ restartedMain] o.m.s.mapper.ClassPathMapperScanner : No MyBatis mapper was found in '[com.highgarden.springboot_board]' package
2024-06-04T13:49:41.615+09:00 INFO 31556 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2024-06-04T13:49:41.624+09:00 INFO 31556 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-06-04T13:49:41.624+09:00 INFO 31556 --- [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.24]
2024-06-04T13:49:41.660+09:00 INFO 31556 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-06-04T13:49:41.660+09:00 INFO 31556 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 764 ms
2024-06-04T13:49:41.732+09:00 INFO 31556 --- [ restartedMain] o.s.b.a.w.s.WelcomePageHandlerMapping : Adding welcome page template: index
2024-06-04T13:49:42.061+09:00 INFO 31556 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2024-06-04T13:49:42.089+09:00 INFO 31556 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-06-04T13:49:42.098+09:00 INFO 31556 --- [ restartedMain] c.h.s.SpringbootBoardApplication : Started SpringbootBoardApplication in 1.535 seconds (process running for 1.535s)
```

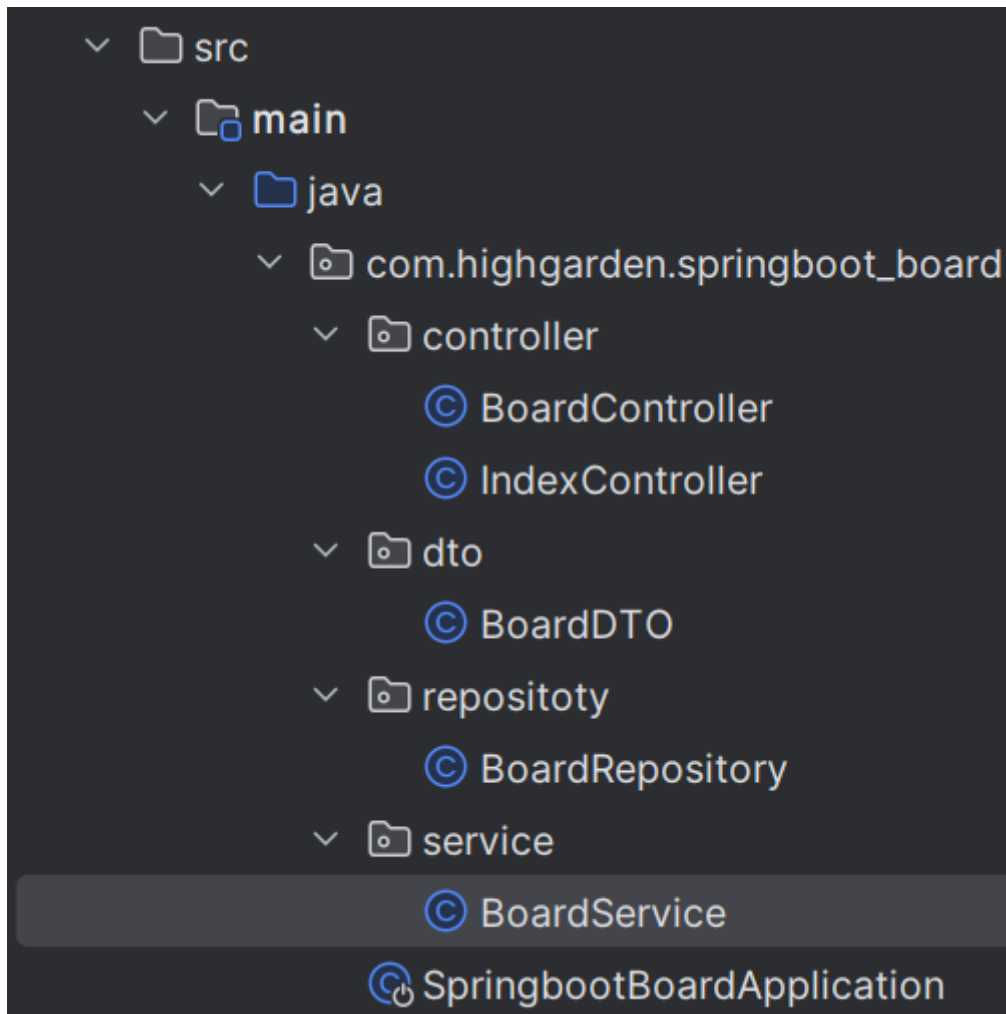
실행성공

어플리케이션이 성공적으로 실행되었다.

4) DB에 insert 해보기

이제 실제로 어플리케이션을 통해 DB작업을 할 수 있도록 코드를 작성해야 한다. 화면구성에 앞서 백엔드 코드를 작성해보자.

백엔드 코드는 mvc패턴에 맞게 프로젝트 경로를 세팅한다.



controller, service, repository, dto 경로 아래에 각각의 클래스들을 생성해준다.

#BoardDTO.class

@Getter

@Setter

@ToString

```
public class BoardDTO {
```

```
    private long id;
```

```
    private String boardWriter;
```



```
private String boardPass;

private String boardTitle;

private String boardContents;

private int boardHits;

private String createdAt;

}
```

@Getter: getter 메서드를 자동생성

@Setter: setter 메서드를 자동생성

@ToString: ToString 메서드를 통해 DTO 내부정보를 printout할 수 있게 해줌

가장 먼저 데이터를 담게 될 DTO 코드를 작성한다. DB를 세팅할 때 생성한 테이블 컬럼명과 동일하게 작성한다.

BoardController.class

@Controller

@RequiredArgsConstructor

```
public class BoardController {

    private final BoardService boardService;

    @PostMapping("/save")

    public void save(BoardDTO boardDTO){

        boardService.save(boardDTO);

    }

}
```

```
    }  
}
```

@Controller: 컨트롤러 클래스라는 것을 스프링이 알수 있게함.

@RequiredArgsConstructor: 초기화되지 않은 final 필드를 자동으로 생성자 주입해줌

@PostMapping: post요청을 받아서 해당 메서드로 매핑해줌

#BoardService.class

@Service

@RequiredArgsConstructor

```
public class BoardService {  
  
    private final BoardRepository boardRepository;  
  
    public void save(BoardDTO boardDTO){  
  
        boardRepository.save(boardDTO);  
  
    }  
}
```

@Service: 서비스 클래스 빈 등록

#BoardRepository.class

@Repository

@RequiredArgsConstructor

```
public class BoardRepository {
```

```

private final SqlSessionTemplate sql;

public void save(BoardDTO boardDTO) {

    sql.insert("Board.save", boardDTO);

}

}

```

@Repository: 레포지토리 클래스 빈 등록

mybatis를 사용하기 위해서 SqlSessionTemplate객체를 사용한다. insert메서드의 파라미터로 mapper의 namespace인 Board로 접근해서 sql문을 수행한다. 위 board-mapper.xml에서 해당 매퍼의 namespace를 Board로 지정해놓았다.

board-mapper.xml

```

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="Board">

    <insert id="save" parameterType="board">

        insert into board_table(boardTitle, boardWriter, boardPass, boardContents)

        values("#{boardTitle}, #{boardWriter}, #{boardPass}, #{boardContents})

    </insert>

</mapper>

```

위에서 작성했던 board-mapper.xml에 insert sql을 추가해준다. 제목, 작성자, 게시글비밀번호, 내

용 외에는 자동생성되도록 구현할 예정이기 때문에 위와같이 작성한다. parameterType에 "board"라고 되어있는 부분은 sql의 매개변수의 타입이 board라는 뜻이다. 제공되는 기본형 변수에 board라는 것은 없기 때문에 이것이 사용자 정의 매개변수라는 것을 알 수 있다.

mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

    "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

    <typeAliases>

        <typeAlias type="com.highgarden.springboot_board.dto.BoardDTO" alias="board"></typeAlias>

    </typeAliases>

</configuration>
```

board-mapper.xml에서 작성한 insert sql문의 매개변수 타입이 board였는데 이것은 BoardDTO클래스의 alias(축약어)이다. alias를 설정하지 않으면 parameterType을 적을 때 마다 전체경로를 적어 주어야 하기 때문에 매우 번거롭다.

5) Postman API 테스트

DB연동을 위한 최소한의 코드작성은 끝났다. 화면을 아직 만들지 않았기 때문에 백엔드 서버로 바로 요청해서 테스트 해야 하는데 이를 위해 postman이라는 서비스를 사용했다.

<https://www.postman.com/>

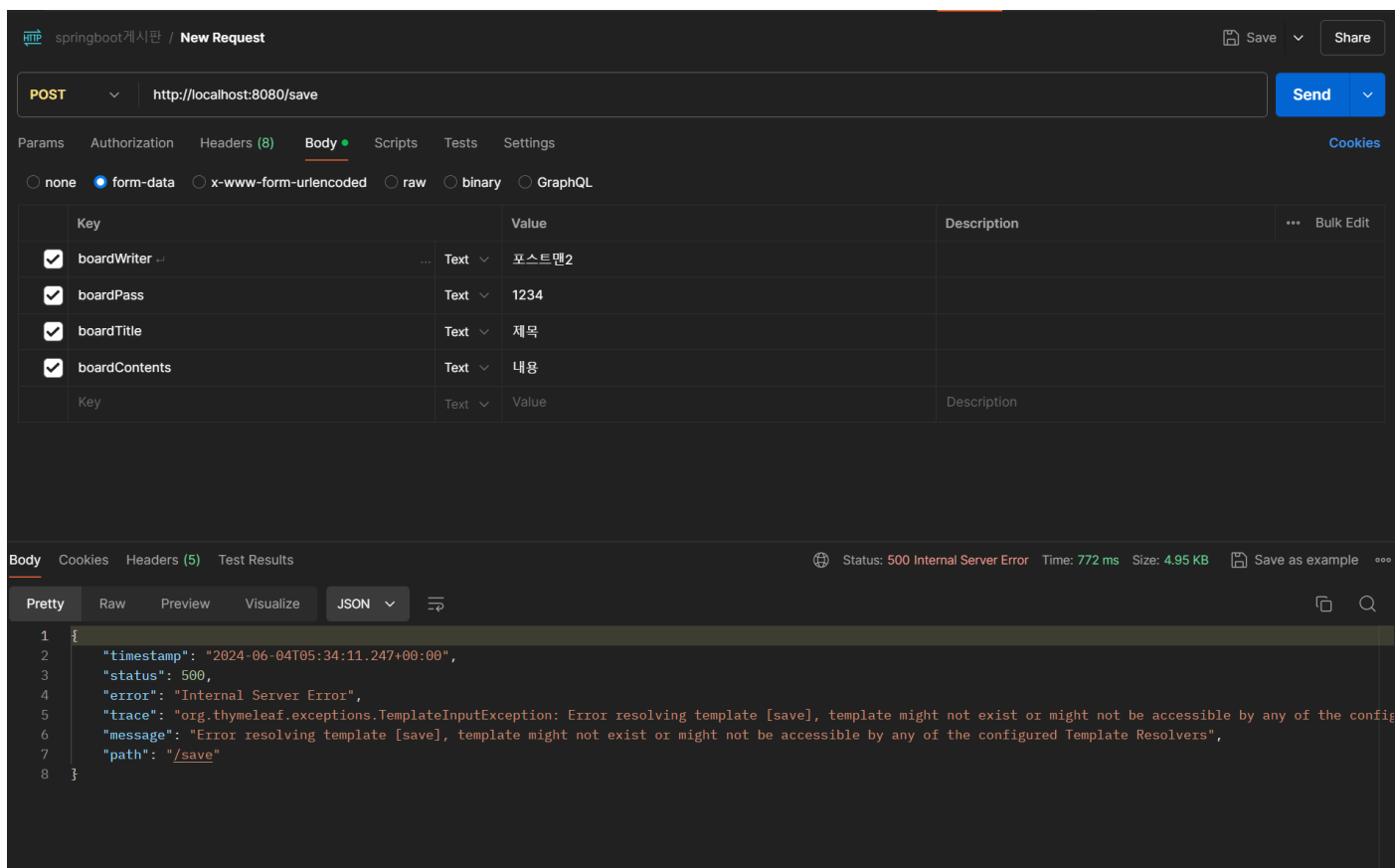
-

[Postman API Platform | Sign Up for Free](#)

Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster.

www.postman.com

post맨은 api를 테스트하기 위해 사용하는 툴로 화면 없이 테스트를 해볼 수 있다.



포스트맨 api테스트

insert문은 post요청으로 이루어지기 때문에 http method를 post로 놓고 서버를 구동시킨 후 localhost:8080/save로 body에 값을 담아 요청한다.

BoardDTO의 필드명과 body에 담긴 변수명이 일치하지 않으면 제대로 값이 담기지 않기 때문에 잘 확인하도록 한다.

요청이 수행되면 결과값이 아래에 반환되게 되는데 **500번 에러**가 발생했다. 서버에서 무언가 처리가 잘 안되었다는 뜻인데 읽어보니 thymereaf템플릿을 찾을 수 없다는 메시지이다. 스프링은 컨트롤러가 뷰를 반환하지 않으면 요청 method를 기반으로 뷰를 찾는데, BoardController의 save 메서드 반환타입을 void로 했기 때문에 save.html을 스프링이 자동으로 탐색한 것이다. 하지만 화면을 아직 만들지 않았기 때문에 해당하는 뷰를 반환할 수 없었고 500에러가 발생한 것이다. 이 문제는 나중에 뷰를 생성하고 연결해주면 해결된다.

500번 에러가 발생했지만 요청은 정상적으로 수행되었기 때문에 DB에 데이터가 잘 들어갔는지 확인해보도록 하자.

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane displays the 'highgarden_db' database with its tables: 'board_file_table' and 'board_table'. The 'board_table' is selected. The main editor shows the SQL query: `use highgarden_db;` followed by `select * from board_table;`. The 'Result Grid' at the bottom displays the query results in a table format.

| id | boardTitle | boardWriter | boardPass | boardContents | boardHits | createdAt | fileAttached |
|----|------------|-------------|-----------|---------------|-----------|---------------------|--------------|
| 1 | 제목 | 포스트맨1 | 1234 | 내용 | 0 | 2024-06-04 14:31:09 | 0 |
| 2 | 제목 | 포스트맨2 | 1234 | 내용 | 0 | 2024-06-04 14:34:11 | 0 |
| * | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

mysql워크벤치에서 select문을 사용해 데이터를 검색해보았다. 테스트를 위해 2건을 insert해보았는데 입력한 내용이 전부 잘 들어가 있는것을 확인 가능하다.

이것으로 DB와 스프링 어플리케이션의 연동을 마쳤다. 다음시간부터는 본격적으로 기능들을 구현해보자.