소프트웨어프로젝트1

2차 프로젝트

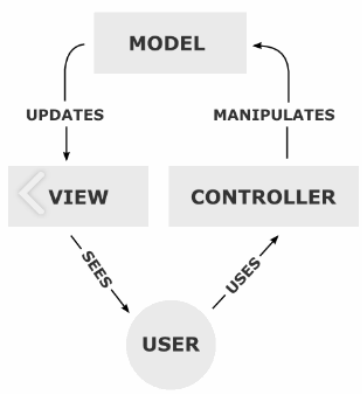
학과:컴퓨터정보공학부

학번: 2018202074

이름: 김상우

Introduction

위 프로젝트는 MVC pattern을 기반으로 작성되어야 한다. MVC패턴은 Model, View, Controller로 구성되며 이들이 역할을 분할하고 이를 바탕으로 작동을 처리되는 패턴이다. 통상적으로 User가 Controller에 Request를 주면 Controller가 해당 값을 Model로 보내고 이 Model을 베이스로 View를 User에게 보여주게 된다. 해당 View를 이용해 다시 Controller로 값을 보내면서 동작되게 된다.



이를 그림으로 표현하면 다음과 같아질 것이다. 각 부분에 대해 Detail하게 설명하면 다음과 같이 될 것이다.

Controller:사용자의 접근(URL)을 바탕으로 요청을 파악하는 역할이다. URL에 따라 Method를 호출하며 Repository를 바탕으로 하는 Service를 통해 Logic을 처리하여 Model에 저장하도록 하는 역할이다 .결과적으로 Model로부터 값을 받아 View를 call해주는 역할을 한다고도 볼 수 있다.

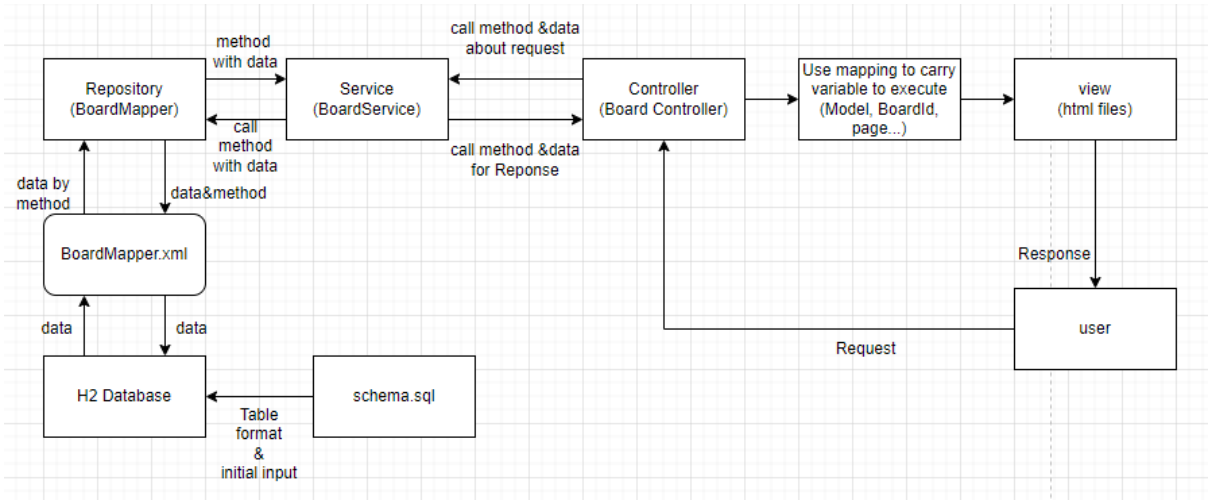
Model:Controller에서의 Method를 바탕으로 저장 및 변경되는 데이터들이 저장되는 공간, 즉 DataBase이다. Controller에 의해서만 변경이 가능하다.

View:Controller가 User로부터 받은 Request를 바탕으로 Model을 호출해 받아온 값들을 바탕으로 User에게 보여주기 위한 부분이다. 일반적으로 HTML, JSP등을 사용하며 Browser를 바탕으로 출력되는 것이 통상적이다.

위의 3구조를 바탕으로 MVC pattern은 진행된다. 이러한 상태를 유지한 상태로 내부 처리에 대해 다양한 방식이 존재한다.

우선 간단하게 Java, HTML을 바탕으로 소통하는 방식이 있다. HTML이 View를, Java 파일이 Controller의 역할을 하며 즉, JSP가 모든 요청을 처리하게 되는 것이다. 단, 구조화가 안되어 있기 때문에 유지 보수가 힘들다는 단점이 있다.

다른 방식으로는 내부처리를 더 세분화하는 방식이다. User는 오직 Controller와 View만 소통을 하게 되며 Controller를 통해 내부에 존재하는 Service를 통해 DAO 등을 이용해 Database와 소통하게 되는 방식이다. 이전과 다르게 Front와 Back이 파일을 기준으로 나뉘어 유지 보수 및 체계화에 유리하다는 장점이 있다. 해당 프로젝트를 진행함에 있어 이 방식을 사용할 것이다.



위를 기반으로 해당 프로젝트에서 사용할 큰 틀은 위의 그림과 같다. 추후 코드와 함께 구체적으로 서술된다. 위의 큰 구조를 보면 확인할 수 있듯이 User는 View로부터 밖에 해당 프로젝트에 대해 확인할 방법이 없으며 이를 기반으로 Controller에만 URL 및 Browser를 통해 접근할 수 밖에 없다. 이렇게 접근된 Controller는 Service, Repository를 통해 H2 Database, 즉 값을 저장하는 Model에 접근하게 될 것이다. 이때 명령어들과 Data를 Model에 보냄에 있어 Service, Repository를 사용하게 되는 것이다. 마찬가지로 Model 또한 들어온 값에 맞게 수정된 후 필요한 값을 Repository와 Service를 기반으로 Controller에 보내준다. 이에 해당된 결과는 Controller에서 View를 부를 때 함께 변수로서 전송되어 이를 기반으로 View가 작동하여 다시 User에게 보이게 되는 것이다.

각 과정에 대해 방식은 다음과 같이 이뤄지게 될 것이다.

User로부터의 Request: Html의 input 및 textarea를 이용, 이를 이용해 문자열 형태의 값들을 받게 될 것이다. URL 및 link의 경우 첫 URL의 경우 Web Brower 입력을 통해, link의 경우 HTML의 a나 button을 통해 구현이 가능할 것이다. 해당 결과로 입력되는 URL의 경우 Controller의 @GetMapping @PostMapping 등의 URL를 받을 수 있는 Annotation을 바탕으로 진행될 것이다.

User로의 Response:HTML을 Web browser를 통해 출력하며 이는 Controller의 Return에 의해 경로 탐색을 통한 html확인 후 출력으로 제공된다. 해당 html의 내용은 Controller가 받아온 변수들을 바탕으로 동작하게 된다.

Controller->H2 Database:user로부터 받은 값들과 기존에 지니던 값들을 HTML내 Javascript등을 통해 변수화한 후 Service, Repository를 통해 Model인 H2database로 전송하게 된다. 들어온 값들을 바탕으로 H2 Database값은 변경된다.

H2 Database->Controller:H2 Database가 가지고 있던 데이터들 중 입력되었던 XML기반의 명령어들을 바탕으로 Repository, Service의 명령어들을 바탕으로 Controller로 전송하게 된다. 이는 User에게 보여지는 View에 영향을 미치게 된다.

Board(사용할 class)

```java
package com.example.kw2018202074.domain;
import lombok.*;
import javax.persistence.*;
//import to use java persistence and lombok
17개 사용 위치
@Getter     //annotation to get function auto
@Entity     //mapping to table
@Setter     //annotation to set function auto
@NoArgsConstructor //make Constructor
@AllArgsConstructor//make Constructor that take every field value
@Table(name="BOARD")   //to use table
public class Board {
    @Id                                                 //boardId is main ket
    @GeneratedValue(strategy = GenerationType.IDENTITY) //we use DataBase's identity column
    private Long boardId;                               //main key variable
    1개 사용 위치
    private String title;                               //title of board
    1개 사용 위치
    private String content;                             //contents of board
    1개 사용 위치
    private String name;                                //upload or update time of board
    1개 사용 위치
    private int read;                                   //views of board

    @Builder                                            //make builder auto
    public void BoardEntity(String title, String content, String name, int read){
        this.title = title;
        this.content=content;
        this.name=name;
        this.read=read;
    }
}
```

해당 class의 경우 위와 같이 생성자 및 Getter, Setter등을 통해 기본적인 class에 대한 annotation들을 받고 있으며(@Getter @Setter @NoargsConstructor @AllArgsConstructor) 객체 매핑을 위해 @Entity또한 사용하였다.) Table Mapping을 위해 @Table을 주었다.

우리는 해당 프로젝트에서 글번호, 제목, 내용, 작성일, 조회수를 담는다는 것을 알고 있다. 그 때문에 다음과 같이 Board 내 다음과 같은 변수들을 만들어 주었다.

Key Value로 사용될 boardId(글번호로서 database에서 또한 main key값으로 사용하기위해 @Id, @GenerateValue(strategy=GenerationType.IDENTITY로 Annotation해주었다.))와 각각 제목, 내용, 작성 및 수정일, 조회수를 저장할 title, content, name, read를 선언해주었다.

추가적으로 @Builder를 통해 해당 값들을 받아올 수 있게 해주었다.

BoardMapper(Repository)

```java
package com.example.kw2018202074.mapper;
import com.example.kw2018202074.domain.Board;
import org.springframework.stereotype.Repository;
import java.util.List;

4개 사용 위치
@Repository                     //repository annotation
public interface BoardMapper {  //interface to define functions
    1개 사용 위치
    List<Board> getList();      //function to change boards to list
    1개 사용 위치
    Board getBoard(Long boardId);//search and take board that have this boardId
    1개 사용 위치
    void uploadBoard(Board board);//upload board to database
    1개 사용 위치
    void updateBoard(Board board);//update board in database
    1개 사용 위치
    void deleteBoard(Long boardId);//delete board in database
    1개 사용 위치
    int updateView(Long boardId);//plus 1 about view

}
```

위는 Repository로 사용할 BoardMapper의 모습이다.

BoardController(Controller)

아래는 해당 프로젝트에서 Controller로서 작동하는 BoardController에 대한 설명이다.
@Repository를 선언하여 해당 interface를 Repository로 사용할 수 있도록 하였다.

getList의 경우 database로부터 데이터들을 받아 list형태로 받을 수 있도록 하였다.

getBoard의 경우 boardId를 받아 이를 기반으로 Board값을 받을 수 있도록 하였다.

uploadBoard의 경우 board를 받아 이를 database에 올릴 수 있도록 하였다.

updateBoard의 경우 board를 받아 이를 database에 올릴 수 있도록 하였다.

deleteBoard의 경우 boardId를 받아 database에서 boardId를 기반으로 board를 찾고 해당 board를 제거하도록 한다.

updateView의 경우 조회수를 올리는 데 사용한다.

BoardMapper.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--xmlversion and encoding type-->
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!--mapper link-->
<mapper namespace="com.example.kw2018202074.mapper.BoardMapper">
<!--    mapper place-->
    <select id="getList" resultType="com.example.kw2018202074.domain.Board">
        SELECT
            *
        FROM tbl_board;
-- select all elements in database
    </select>
<!--get all board list in database-->
    <select id="getBoard" parameterType="Long" resultType="com.example.kw2018202074.domain.Board">
        SELECT
        *
        FROM tbl_board where boardId=#{boardId};
-- select board by boardId in database
    </select>
<!--search and get board from database-->
    <insert id="uploadBoard" parameterType="com.example.kw2018202074.domain.Board">
        <if test = "!(content.isEmpty()) ">
        <if test = "!(title.isEmpty()) ">
-- if no input about content or title
        INSERT INTO tbl_board (title,content,name,read)
        VALUES
        (#{title}, #{content}, #{name}, 0);
-- make and put board that made by input values in database
        </if>
        </if>
    </insert>
<!--insert board in database-->
```

위는 BoardMapper, 즉 Repository에서 사용될 명령어들에 대한 동작을 xml파일로 작성해둔 것이다. Xml의 버전은 1.0으로 encoding의 경우 UTF-8을 기반으로 할 것이다. 또한 스펙문서(DTD)를 받아오기 위해 !DOCTYPE을 이용, 해당 링크를 통해 값을 받아오게 되었다.

Mapper를 통해 해당 파일을 참고할 Repository를 언급해둔 모습이다.

Select를 명령어로 getList를 id값으로 받고 내부에 SELECT * FROM tbl_board를 선언하여 tbl_board, 즉 database에 있는 모든 board값을 받아온다. 이때 ResultType은 새로 선언된 Board임을 Result Type에서 서술하고 있다.

Select를 명령어로 getBoard를 id값으로 받고 내부에 SELECT * FROM tbl_board where boardId=#{boardId}를 선언하여 tbl_board, 즉 database에 있는 모든 board 값 중 parameter로 선언되는 Long 타입의 boardId를 가지는 board를 받아온다. 이때 ResultType은 새로 선언된 Board임을 Result Type에서 서술하고 있다.

Insert를 명령어로 upload를 ide값으로 받고 내부에 있는 if문을 통해 content와 title이 비지 않을 경우 내부 명령을 실행하도록 하였다. INSERT INTO tbl_board(title, content, name, read)를 통해

data base에 해당 변수들에 대하여 VALUE 아래 있는 인자로 받아진 값들을 넣어준다.

```
    <update id="updateBoard" parameterType="com.example.kw2018202074.domain.Board">
        <if test = "!(content.isEmpty()) ">
        <if test = "!(title.isEmpty()) ">
-- if no input about content or title
        UPDATE tbl_board
        SET title =#{title} , content=#{content}, name=#{name}
-- change and put board that made by input values in database
        WHERE boardId=#{boardId};
        </if>
        </if>
    </update>
<!--update board that have boardId in database-->
    <update id="updateView" parameterType="Long">
        UPDATE tbl_board
        SET read = read + 1
        WHERE boardId=#{boardId};
    </update>
<!--plus 1 to board's view in database-->
    <delete id="deleteBoard" parameterType="Long">
        DELETE
        FROM tbl_board
        WHERE boardId=#{boardId};
    </delete>
<!--delete board that have boardId in database-->
</mapper>
```

Update를 명령어로 id를 updateBoard를 받고 내부에 있는 if문을 통해 content와 title이 비지 않을 경우 내부 명령을 실행하도록 하였다. UPDATE tbl_board SET title=#{title}, content=#{content}, name=#{name}를 통해 data base에 해당 변수들에 대하여 인자로 받은 값들을 넣어주게 되는 것을 알 수 있으며 WHERE boardId=#{boardId}를 통해 이는 boardId가 같은 board에만 해당되는 것을 알 수 있다.

Update를 명령어로 id를 updateView를 받고 내부에 있는 UPDATE tbl_board SET read= read+1를 통해 data base에 해당 read변수에 대하여 1을 더해주게 되는 것을 알 수 있으며 WHERE boardId=#{boardId}를 통해 이는 boardId가 같은 board에만 해당되는 것을 알 수 있다.

delete를 명령어로 id를 deleteBoard를 받고 DELETE FROM tbl_board를 통해 data base에 있는 board에 대해 삭제를 진행함을 알 수 있으며 WHERE boardId=#{boardId}를 통해 이는 boardId가 같은 board에만 해당되는 것을 알 수 있다.

BoardService(Service)

```java
//import to use some instructions to make BoardService
package com.example.kw2018202074.service;
import com.example.kw2018202074.domain.Board;
import com.example.kw2018202074.mapper.BoardMapper;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.util.List;

3개 사용 위치
@Service //chang to Bean and put them in root container(calculate java logic)
public class BoardService {
    7개 사용 위치
    private final BoardMapper boardMapper;
    //to use repository
    public BoardService(BoardMapper boardMapper){
        this.boardMapper = boardMapper;
    }
    //setter
    3개 사용 위치
    public List<Board> boardList(){return boardMapper.getList();}
    //variable to take all data
    1개 사용 위치
    public Board getBoard(Long boardId){return boardMapper.getBoard(boardId);}
    //find board(use boardId)
    1개 사용 위치
```

Service로 사용하기 위해 @service를 annotation으로 받는 class BoardService를 제작하였다. 이는 BoardMapper Class 변수 boardMapper를 받아 작동한다. 생성자로 BoardMapper를 인자로 받는 BoardService를 사용하며 boardMapper값을 넣어준다.

boardList함수: boardMapper의 getList를 이용해 database에 저장된 값들을 받아 List의 형태로 변경해준다.

getBoard함수: boardMapper의 getBoard를 이용, database에 저장된 값들 중 boardId를 이용해 board값을 받아온다.

```java
    @Transactional
    public void uploadBoard(Board board) { boardMapper.uploadBoard(board); }
    //make new board and upload that in data base
    1개 사용 위치
    @Transactional
    public void updateBoard(Board board){boardMapper.updateBoard(board);}
    //find board and change data
    1개 사용 위치
    @Transactional
    public void deleteBoard(Long boardId) { boardMapper.deleteBoard(boardId); }
    //find board and delete that
    1개 사용 위치
    @Transactional
    public int updateView(Long boardId) { return boardMapper.updateView(boardId); }
    //update view by click
}
```

@Transactional을 통해 작업들에 오류 발생시 복구처리를 해주었다.

uploadBoard함수는 인자로 받는 board를 이용, BoardMapper의 uploadBoard를 이용해 board를 DataBase에 upload할 수 있도록 하였다.

updateBoard함수는 인자로 받는 board를 이용, BoardMapper의 updateBoard를 이용해 board를 DataBase에 update할 수 있도록 하였다.

deleteBoard함수는 인자로 받는 boardId를 이용, BoardMapper의 deleteBoard를 이용해 boardID를 갖는 DataBase의 board를 찾아 delete할 수 있도록 하였다.

UpdateView함수는 인자로 받는 boardId를 이용, BoardMapper의 updateView를 이용해 boardID를 갖는 DataBase의 board를 찾아 read값을 1만큼 올릴 수 있도록 하였다.

```java
package com.example.kw2018202074.controller;
import com.example.kw2018202074.domain.Board;
import com.example.kw2018202074.service.BoardService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
//import functions that used in BoardController
@Slf4j          //annotation about logging framework
@Controller     //controller annotation
public class BoardController {  //controller class
    9개 사용 위치
    private BoardService service;   //service variable to take data to make controller
    public BoardController(BoardService boardService) { this.service = boardService; }
    //setter
    @GetMapping("/")                //about http://localhost:8080/
    public String init(Model model){
        model.addAttribute( attributeName: "list", service.boardList());    //to take all datas about database
        model.addAttribute( attributeName: "page",  attributeValue: 0);          //current page = 0(1)
        return "/boards/start";                                                 //go to start
    }
```

구현을 위해 Import를 해준 값들이 보인다. 대표적으로 Service와 domain으로부터 Repository와 class로 사용할 부분을 받고 있는 것을 확인할 수 있다.

Logging framwork와 Controller로서의 사용을 위해 annotation을 위와 같이 달아주고 URL을 통한 이동을 감지 및 이벤트 적용을 해주기 위한 함수들을 아래 선언해준다.

우선 Repository를 사용하기 위해 BoardService를 불러오고 내부에서 쓰기 위해 service를 선언해주고, Setter를 선언해주었다.

GetMapping을 통해 http://localhost:8080으로 들어갔을 경우를 확인해준다. 해당 URL이 입력되었을 경우 boards내 start.html을 불러오고 해당 파일에서 Database의 Board들에 대해 받아올 수 있는 boardList를 list로, page값을 0으로 쓸 수 있도록 선언한다.

BoardController(Controller)

```
@GetMapping("/board/start")     //about http://localhost:start
public String start(Model model, int page){                     //when start html
    model.addAttribute( attributeName: "list", service.boardList());   //to take all data about database
    model.addAttribute( attributeName: "page", page);                    //take page from current variable
    return "/boards/main";                                               //go to main.html
}

@GetMapping("/board/main")                                      //when take /board/main
public String main(Model model){                                 //when go to main html
    model.addAttribute( attributeName: "list", service.boardList());    //to take all datas about database
    model.addAttribute( attributeName: "page",  attributeValue: 0);      //current page is zero(when end of upload)
    return "/boards/main";                                               //go to main.html
}
@GetMapping("/board/view")                                      //when take /board/main
public String viewBoard(Model model, Long boardId){              //when go to view html
    service.updateView(boardId);                                  //update view about board that have spectiboardId
    model.addAttribute( attributeName: "view",service.getBoard(boardId));//find and put board into view.html
    return "/boards/view";                                           //go to view.html
}
```

GetMapping을 통해 http://localhost:8080/board/start로 들어갔을 경우를 확인해준다. 해당 URL이 입력되었을 경우 boards내 start.html을 불러오고 해당 파일에서 Database의 Board들에 대해 받아올 수 있는 boardList를 list로, page값은 이전의 page로 쓸 수 있도록 선언한다.

GetMapping을 통해 http://localhost:8080/board/main로 들어갔을 경우를 확인해준다. 해당 URL이 입력되었을 경우 boards내 start.html을 불러오고 해당 파일에서 Database의 Board들에 대해 받아올 수 있는 boardList를 list로, page값은 이전의 page로 쓸 수 있도록 선언한다.

GetMapping을 통해 http://localhost:8080/board/view로 들어갔을 경우를 확인해준다. 해당 URL이 입력되었을 경우 boards내 view.html을 불러오고 해당 파일에서 Database에 저장된 board에 대해 검색 후 받아올 수 있는 boardId 이전으로부터 받고 이 값을 바탕으로 찾은 Board를 view로 쓸 수 있도록 선언한다.

```
    @GetMapping("/board/upload")                                 //when take /board/upload
    public String uploadBoardForm() { return "/boards/upload"; }  //go to upload.html

    @PostMapping("/board/upload")                                //end of /board/upload
    public String uploadBoard(Board board){                      //when we need to upload board
        service.uploadBoard(board);                               //do upload board
        return "redirect:/board/main";                            //go back to main
    }
    @PutMapping("/board/update")                                 //when go to update
    public String updateBoard(Board board){                       //when we need to update board
        service.updateBoard(board);                               //update current board
        return "redirect:/board/main";                            //go back to main
    }

    @DeleteMapping("/board/delete")                              //when go to delete to delete board
    public String deleteBoard(Long boardId){                      //when we need to delete board
        service.deleteBoard(boardId);                             //delete current board
        return "redirect:/board/main";                            //go back to main
    }
}
```

GetMapping을 통해 http://localhost:8080/board/upload로 들어갔을 경우를 확인해준다. 해당 URL이 입력되었을 경우 boards내 upload.html을 불러온다.

PostMapping을 통해 http://localhost:8080/board/upload에서 나올 경우를 확인해준다. 해당 URL이 입력되었을 경우 boards내 main.html을 불러온다. 이때 submit된 Board를 upload해준다.

PutMapping을 통해 http://localhost:8080/board/update로 들어갔을 경우를 확인해준다. 이때 submit된 board를 update하고 boards내 main.html을 불러온다.

DeleteMapping을 통해 http://localhost:8080/board/update로 들어갔을 경우를 확인해준다. 이때 boardId를 통해 Database에서 board를 찾아 delete하고 boards내 main.html을 불러온다.

Application

```java
package com.example.kw2018202074;
import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
//import to execute
1개 사용 위치
@SpringBootApplication  //application about spring
@MapperScan(basePackages = "com.example.kw2018202074") //scan package in this project
public class Application {
    public static void main(String[] args) { SpringApplication.run(Application.class, args); }
    //program start
}
```

위는 실행을 위한 Application 코드이다. Packages로서 해당 Package를 받는 모습이 보이며 Spring Application.run을 통해 이를 실행하는 모습이 보인다.

Main, start.html(Title을 제외하고 동일한 코드)

둘은 동일한 코드임에도 2개의 html을 사용하였다. 이는 해당 html들이 page라는 인자를 받을 필요가 있는데, 이는 처음 시작을 이 html로 하여 page에 대해 input을 줄 수 없으므로 이를 위한 여분 html을 제작한 것이다. 이들은 게시판이 출력되는 부분에 대한 html이다.

```
<!DOCTYPE html>
<html lang="en">
<html xmlns xmlns:th="http://www.w3.org/1999/xhtml" : th="http://www.thymeleaf.org">
<!--to use thymeleaf and xml namebase to xhtml document-->
<head>
  <meta charset="UTF-8">
  <title>Main</title>
<!--  we use UTF-8 and set title-->
  <style type="text/css">
    table {
      width: 100%;
      border: 1px solid #444444;
      border-collapse: collapse;
    }
    /*table design*/
    th, td {
      border: 1px solid #444444;
    }
    /*make border about th td */
    div{
      text-align: center;
    }

  </style>
</head>
```

Html을 사용하기 위해 DOCTYPE으로 선언하고 이후 xhtml형식과 thymeleaf를 쓰기위해 해당 link를 연결하는 모습이다. 이후 head부분에서 UTF-8을 사용함을 명시하고 Title을 임의로 Main으로 정한다.(start.html의 경우 이부분이 Start로 되어있다.) 이후 좀더 깔끔하게 보이게 하기 위해 <style>을 이용, table에 대해 너비 100%, 경계표시, collapse를 통한 비중 부여를 진행하였다. Th,td에 대해서도 경계를 주고 div를 사용하는 경우 위치가 중앙에 올 수 있도록 하였다.

```
<body>
    <div>게시판 리스트</div>
  <table class="table table-hover" style="margin-left: auto;margin-right: auto">
<!--    table style and class-->
    <thead>
    <th>글번호</th>
    <th>제목</th>
    <th>작성일</th>
    <th>조회수</th>
<!--    format keys about table-->
    </thead>
```

테이블을 선언하고 margin(여백)을 왼쪽과 오른쪽에 넣어 중앙을 맞춘 모습이다. 또한 위 프로젝트에서 table은 table-hover에 기반해 제작될 것이다. 테이블의 값의 종류들인 글번호, 제목, 작성일, 조회수를 가장 위에 출력해주었다.

```
    <tr th:each="i :${#numbers.sequence(list.size()-1,0,-1)}" class="align-text-top">
      <td th:if="${i<(list.size()-page*8) && i>=(list.size()-page*8-8)}">[[${list[i].boardId}]]</td>
      <td th:if="${i<(list.size()-page*8) && i>=(list.size()-page*8-8)}">
        <a th:href="@{/board/view(boardId=${list[i].boardId})}">
          [[${list[i].title}]]</a></td>
      <td th:if="${i<(list.size()-page*8) && i>=(list.size()-page*8-8)}">[[${list[i].name}]]</td>
      <td th:if="${i<(list.size()-page*8) && i>=(list.size()-page*8-8)}">[[${list[i].read}]]</td>
<!--      they print elements only range about page-->
    </tr>
<!--    loop and print elements in database-->
  </table>
  <div style='position:static;left:50%;'>
  <nav>
<!--    for paging-->
    <div class="pagination">
      <tr th:each="i :${#numbers.sequence(0,list.size(),1)}" >
<!--        loop for list.size(based database)-->
      <td  th:if="${i==page && i<((list.size()-1)/8)+1}" class="active"><a style=
      "color: red;display: inline" href="#">[[${i}+1]]</a></td>
<!--        about current page-->
      <td ><a style="display: inline;color: blue" th:if="${i!=page && i<((list.size()-1)/8)+1}"
              th:href="@{/board/start(page=${i})}">[[${i}+1]]</a></td>
<!--        for other pages case(restart after change page)-->
      </tr>
    </div>
  </nav>
  </div>
```

이후 table내의 elements들을 출력할 수 있게 해주었다. List의 크기-1~0번의 값만큼 변수 i를 반복시키도록 하였으며 조건문 내에서 볼 수 있듯 i<(list.size()-page*8)이며 i>=list.size()-page*8-8인 경우에만 해당 i를 boardId로 가지는 board의 elements(boardId, title, name, read)를 출력할 수 있게 해주었다. 특히 title의 경우 <a>를 이용, title자체를 링크로 걸어주었다. 이때, link로 /board/view로 이동하게 하였으며 이때 boardId라는 변수로서 현재 boardId값을 보내주게 하였다.

이후 Paging을 위해 nav를 선언, 내부에서 pagination을 class로 선언해주고 i를 0~list의 크기만큼 반복해주었다.(paging에 나오는 값은 실 크기보다 작게 될 것이므로) 이때 page와 i값이 같고 해당 값이 list의 크기-1을 8로 나눈 값+1보다 작을 경우 붉게, 이때 page와 i값이 다르고 해당 값이 list의 크기-1을 8로 나눈 값+1보다 작을 경우 파랗게 출력하며 후자의 경우는 /board/start로 page값이 i값인 상태로 이동하게 해두었다.

```
  <div style="...">
  <button type="button" class="navyBtn" onClick="location.href='board/upload'">게시글 작성</button>
<!--  move to upload html to upload board-->
    </div>
</div>

</body>
</html>
```

끝으로 클릭시 board/upload로 이동하는 button를 배치해주었다.

Upload.html

이는 Upload, 즉 board에 제목과 내용을 입력하고 해당 값을 Database에 추가하기 위한 페이지에 대한 html이다.

```
<!DOCTYPE html>
<html xmlns xmlns:th="http://www.w3.org/1999/xhtml" : th="http://www.thymeleaf.org">
<!--to use thymeleaf and xml namebase to xhtml document-->
<head>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">
  <meta charset="UTF-8">
  <title>Upload</title>
<!--  set title and use UTF-8, can link style sheet-->
</head>
```

Html을 사용하기 위해 DOCTYPE으로 선언하고 이후 xhtml형식과 thymeleaf를 쓰기위해 해당 link를 연결하는 모습이다. 이후 head부분에서 UTF-8을 사용함을 명시하고 Title을 임의로 Upload으로 정한다. 이후 깔끔한 디자인을 위해 link를 통해 stylesheet로서 다음 link를 연결하였다.

```
<body>
<div style="text-align: center;">게시판 리스트</div>
<div class="container">
  <form th:action="@{upload}" method="post">
<!--    upload this data, method is post-->
      <div style="text-align: center;">
          <label class="col-sm-2 control-label">제목</label>
      </div>
<!--      to print data in 1-line-->
      <input type="text" class="form-control" width ='250' name="title">
<!--        take title from user-->
    <input type="hidden" id='currentDate' name="name">
<!--    take current year-month-day(connect with javascript)-->
      <!--      to print data in 1-line-->
      <div style="text-align: center;">
      <label class="col-sm-2 control-label">내용</label>
      </div>
    <textarea class="form-control" name="content"></textarea>
<!--        take contents from user-->
    <div style=" text-align: center;">
    <button type="submit">등록</button>
    </div>
  </form>
</div>
```

가장 상단 중앙에 문자열, 게시판 리스트를 띄워준다. 이후 class를 container로 받는 div를 선언 후 내부에 Upload를 위해 반환하게 될 값을 위해 form으로 이를 묶는다. 이때 action과 method를 통해 form이 submit될 경우 upload를 빠져나감을 알 수 있다. 이후 div를 이용해 중앙에 고정된 제목과 input을 통한 제목 입력 칸 출력, div style을 이용해 중앙에 출력한 내용과 내용 입력

칸을 출력한다. 문자열 제목, 내용은 label을 통해 입력칸은 각각 input, textarea를 통해 출력된다.

또한 충간에 hidden으로서 id를 currentDate로 가지며 name으로서 이를 반환하게 될 부분을 input으로 표현하였다. 이후 하단 중앙에 submit를 위한 버튼을 생성하였다.

```
<script>
    document.getElementById('currentDate').value = new Date().toISOString().substring(0, 10);
    // take today's Year, Month, Day and give to form
</script>
</body>
</html>
```

아래 부분은 자바스크립트를 이용한 부분이다. getElementById를 통해 Id를 이용, hidden input이었던 부분에 해당 값을 줄 수 있도록 한다. 이때, 해당값은 Date().toISOString().substring(0,10)의 결과로 이는 현재 날의 년도-월-일 순으로 문자열을 반환해주는 명령어이다. 해당 값은 위에서 사용되어 반환값에 있어 name이라는 이름으로 submit될 것이다.

View.html

View.html은 게시판에서 제목을 누를 시 내용확인을 위해 화면에 보이게 되는 부분에 대한 html로, 삭제와 수정, 게시판으로 돌아가기를 위한 버튼들을 가지고 있다.

해당 html에서는 수정에 대한 처리도 진행하고 있다.

```
<!DOCTYPE html>
<html xmlns xmlns:th="http://www.w3.org/1999/xhtml" : th="http://www.thymeleaf.org">
<head>
  <script src="https://code.jquery.com/jquery-1.12.0.min.js"></script>
  <script src="js/vendor/modernizr-3.8.0.min.js"></script>
  <script>window.jQuery || document.write('<script src="@{/js/vendor/jquery-3.4.1.min.js}"><\/script>')
  </script>
  <!--  script about j query-->
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">
<!--  take stylesheet-->
  <meta charset="UTF-8">
  <title>View</title>
<!--  use UTF-8 and set Title-->
</head>
```

Html을 사용하기 위해 DOCTYPE으로 선언하고 이후 xhtml형식과 thymeleaf를 쓰기위해 해당 link를 연결하는 모습이다. 이후 head부분에서 UTF-8을 사용함을 명시하고 Title을 임의로 View로 정한다. 이후 깔끔한 디자인을 위해 link를 통해 stylesheet로서 다음 link를 연결하였다.

이때, script를 이용, jquery를 활용하기 위해 source로서 해당 link들로부터 참조를 받아오는 모습이다. 또한 Modernizr(javascript library)를 사용하기 위해서도 참조를 받아오는 모습이다.

```
<body>
<div class="container">
  <div style="text-align: center;">
    <p id="listtitle">[[${view.title}]]</p>
    <p id="change">조회수 : [[${view.read}]]</p>
    <p id="title">생성일자 : [[${view.name}]]</p>
    <p id="title2"></p>
  </div>
<!--  print table main keys-->
  <div id="content">
    <textarea readonly="readonly"
              class="form-control"
              th:text="${view.content}">
    </textarea>
  </div>
<!--  format to take content input from user-->
```

Class로 containter를 갖는 div를 선언, html이 불릴 때 받아온 view(board)를 바탕으로 제목, 조회수, 생성일자를 출력한다.(추후 변경을 위해 p마다 id를 주었다.) 또한 내용을 출력해주는 데, 변경이 불가능하도록 readonly를 선언해주며 text를 view의 content로 해주었다.

```
<!--  format to take content input from user-->
  <div style=" text-align: center;">
    <button id="back" class="navyBtn display: inline" onClick="location.href='main'">이전으로</button>
    <button id="updateBtn" class="btn btn-light btn-sm display: inline" >수정</button>
    <button id="deleteBtn" class="btn btn-light btn-sm display: inline" >삭제</button>
  </div>
<!--  button to check return or delete or update-->
  <form id="form" th:action="@{/}" method="post">
  </form>
<!--  to post using input datas-->
</div>
```

이후 중앙정렬을 위해 text-align:center가 style인 div를 선언 후 내부에 button들을 생성한다. 해당 버튼들을 display:inline을 통해 한 줄에 출력된다.

이전으로 가기 위한 button의 경우 main으로 이동을 위해 onClick값으로 location.href='main'을 받는다. 수정과 삭제를 위한 button의 경우 색 조정을 위해 btn-light를 class에 주었다.

이 버튼들 또한 id를 넣어 추후 변경이 가능하도록 하였다.

이후 form을 설정, 해당 view를 떠날 시 http://localhost:8080/로 이동할 수 있도록 하였다.

```
<script th:inline="javascript">
// script base javascript
  $(document).on('ready',function (e){
// ready to click button
    var form= $("#form");
    var boardId= [[${view.boardId}]];
    $(document).on('click', '#deleteBtn', function (e){
      // when click this button
      $('#form').attr("action","delete");
      form.append("<input type='hidden' name='boardId' value='"+boardId+"'>");
      form.append("<input type='hidden' name='_method' value='delete'>");
      form.submit();
      // take boardId and delete board using this Id
    })
```

그 아래 java script를 통해 위 html을 control하게 만들어주었다. 모든 button이 ready상태일 때 form으로 받는 값과 현재 view의 boardId에 대해 변수를 제작한다. #deleteBtn을 Id로 갖던 삭제 버튼이 click, 즉 눌렸을 경우 form에 delete가 들어가도록 하고 이를 위한 인자로서 boardId가 제공된다. 또한 delete를 할 수 있도록 _method로서 delete를 주고 있는 모습이다. 이후, 이를 submit하여 반영할 수 있도록 한다.

```
$(document).on('click','#updateBtn',function (e){
  //when click this button
  var str="<input class='form-control' width='250'" +
          "placeholder='"+[[${view.title}]]+"' id='updateTitle'>";
  // to take Title from user
  var str2="<textarea class='form-control' placeholder='"+[[${view.content}]]+"'" +
          "id='updateContent'></textarea>";
  // to take content from user
  var str3=new Date().toISOString().substring(0, 10);
  //make update time
  var str4 ="<label class=\"col-sm-2 control-label\">제목</label>"
  var str5 ="<div>게시판 리스트</div>"
  var str6= "<label class=\"col-sm-2 control-label\">내용</label>"
  //print format
  if(str && str2 && str3 && str4 && str5 &&str6){
    $("#change").html(str4);
    $("#listtitle").html(str5);
    $("#title").html(str);
    $("#title2").html(str6);
    $("#content").html(str2);
    $("#name").html(str3);
    $("#back").hide();
    $("#deleteBtn").hide();
  }
  //change them to string and hide back, delete button
  $("#updateBtn").attr("id","updateConfirmBtn");
  //change update button state
})
```

위는 #updateBtn을 Id로 갖는 수정 버튼이 눌렸을 때 발동한다. 기본적으로 view.title, 즉 현재 view가 나타내는 제목을 출력해주며 user가 값을 넣을 수 있는 input과 기본적으로 view.content, 즉 현재 view가 나타내는 내용을 출력해주며 user가 값을 넣을 수 있는 textarea를 각각 변수 str, str2로서 저장하고 이후 현재의 년,월,일 또한 str3로서 받아 board의 name으로 넣을 수 있게 해준다. 이후 format을 맞추기 위해 문자열 제목, 내용을 출력하는 label과 게시판 리스트라는 문자열을 띄워주기 위한 변수(str4, str6, str5)도 만들어낸다. 이들이 참인 경우 해당 id에 해당하는 부분을 .html을 이용, 변경해준다. 이후 수정버튼이 한번 이상 눌림을 나타내기 위해 updateBtn을 id로 갖던 수정 button의 id를 updateConfirmBtn으로 변경해준다.

또한 .hide()함수를 이용, 이전으로가는 버튼과 삭제 버튼을 hide, 즉 화면에서 없앤다.

```
    $(document).on('click','#updateConfirmBtn',function (e){
      //when click this button(after click)
      $('#form').attr("action","update");
      //we will update this form
      var str="<input class='form-control' width='250'" +
              "placeholder='"+[[${view.title}]]+"' id='updateTitle'>";
      // to take Title from user
      var str2="<textarea class='form-control' placeholder='"+[[${view.content}]]+"'" +
              "id='updateContent'></textarea>";
      var updateTitle= $('#updateTitle').val();
      var updateContent=$('#updateContent').val();
      // variable about user input(title, content)
      var curd = new Date().toISOString().substring(0, 10);
      // take year, month, day and input title, contents from user
      if(updateTitle && updateContent){
        //if we took title and content from user
        form.append("<input type='hidden' name='_method' value='put'>");
        form.append("<input type='hidden' name='boardId' value='"+boardId+"'>");
        form.append("<input type='hidden' name='title' value='"+updateTitle+"'>");
        form.append("<input type='hidden' name='content' value='"+updateContent+"'>");
        form.append("<input type='hidden' name='name' value='"+curd+"'>");
        form.submit();
        //save them and update time to board
      }
      else{//input title or content is empty
        $("#content").html(str2);
        $("#title").html(str);
        $("#updateBtn").attr("id","updateConfirmBtn");
        //reset input box and initailize button
      }
      //submit them
    })
  })
</script>

</body>
</html>
```

수정 button의 id값이 수정 버튼이 한번 이상 눌릴 경우인 updateConfirmBtn인 경우에 click이 될 경우 다음 script가 진행된다. Form에 대해 action으로 update를 부여한다. 기본적으로 view.title, 즉 현재 view가 나타내는 제목을 출력해주며 user가 값을 넣을 수 있는 input과 기본적으로 view.content, 즉 현재 view가 나타내는 내용을 출력해주며 user가 값을 넣을 수 있는 textarea를 각각 변수 str, str2로서 저장하고 이들에 입력될 값, 즉 유저로부터 입력받는 제목과 내용을 저장하는 변수 updateTitle, updateContent를 만든다. 이후 현재 년-월-일을 저장하는 curd변수를 제작한다. updateTitle, updateContent가 모두 존재할 경우 _method를 put, boardId를 현재 view의 boardId, title을 입력된 제목, content를 입력된 내용, name(작성 및 수정일을 담는 변수)을 현재 년-월-일 로하는 값들을 append를 통해 받는 form을 submit하게 된다.

만약 제목과 내용 중 하나 이상 입력되지 않을 경우 str1과 str2를 해당 위치에 재차 선언하며 수정 버튼의 id또한 updateConfirmBtn으로 설정한다.

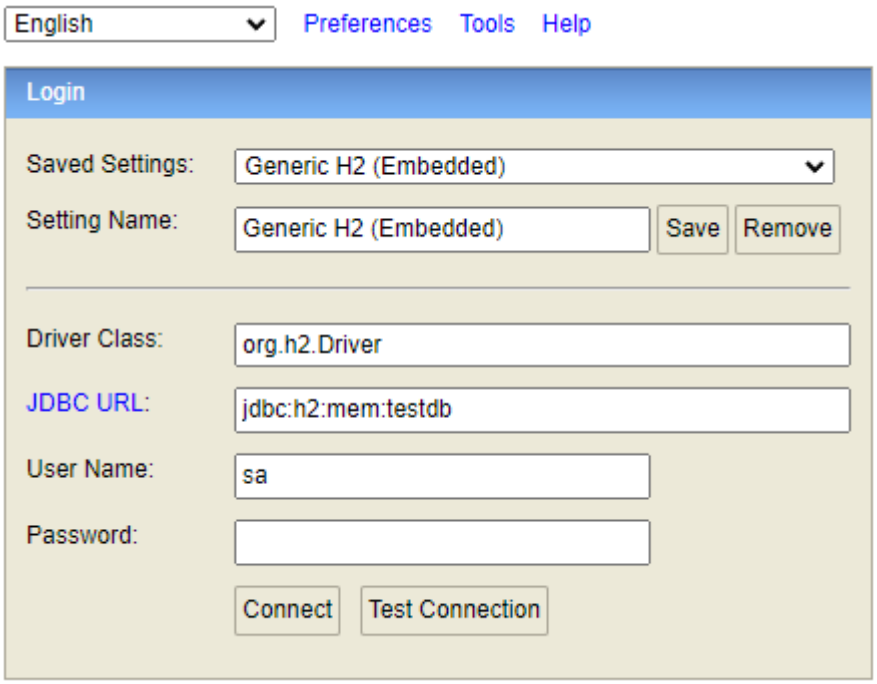
Schema.sql

```
CREATE TABLE tbl_board
(
    boardId Long auto_increment,
    title   varchar(30) not null,
    content varchar(30) not null,
    name varchar (60) not null,
    read varchar (30) not null,
    primary key (boardId)
);
-- base of table(variables)
```

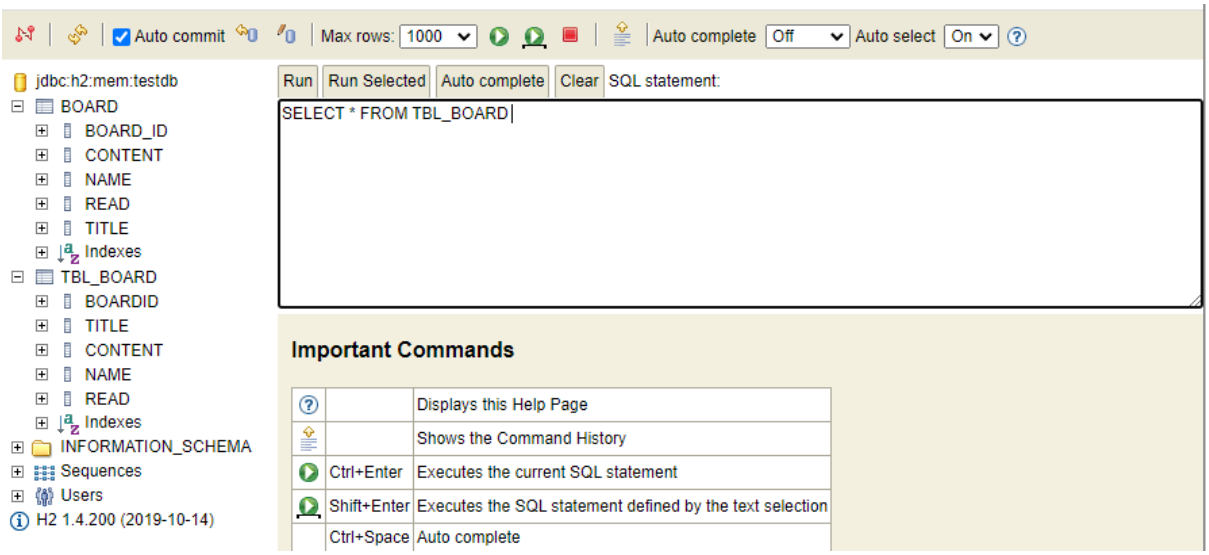
해당 파일에는 h2 Database에 저장될 데이터의 모습을 입력해주었다. Class board와 마찬가지로 title, content, name, read 값을 가지며 boardId를 주요 key로 갖는 모습이다. 이때, boardId의 자동 상승을 위해 auto_increment를 해준 것이 확인된다.

```
INSERT INTO tbl_board(title, content, name, read) VALUES ('title1', 'content1', 'name1', '0');
INSERT INTO tbl_board(title, content, name, read) VALUES ('title2', 'content2', 'name2', '0');
INSERT INTO tbl_board(title, content, name, read) VALUES ('title3', 'content3', 'name3', '0');
INSERT INTO tbl_board(title, content, name, read) VALUES ('title4', 'content4', 'name4', '0');
INSERT INTO tbl_board(title, content, name, read) VALUES ('title5', 'content5', 'name5', '0');
INSERT INTO tbl_board(title, content, name, read) VALUES ('title6', 'content6', 'name6', '0');
INSERT INTO tbl_board(title, content, name, read) VALUES ('title7', 'content7', 'name7', '0');
INSERT INTO tbl_board(title, content, name, read) VALUES ('title8', 'content8', 'name8', '0');
INSERT INTO tbl_board(title, content, name, read) VALUES ('title9', 'content9', 'name9', '0');
INSERT INTO tbl_board(title, content, name, read) VALUES ('title10', 'content10', 'name10', '0');
INSERT INTO tbl_board(title, content, name, read) VALUES ('title11', 'content11', 'name11', '0');
INSERT INTO tbl_board(title, content, name, read) VALUES ('title12', 'content12', 'name12', '0');
INSERT INTO tbl_board(title, content, name, read) VALUES ('title13', 'content13', 'name13', '0');
INSERT INTO tbl_board(title, content, name, read) VALUES ('title14', 'content14', 'name14', '0');
INSERT INTO tbl_board(title, content, name, read) VALUES ('title15', 'content15', 'name15', '0');
```

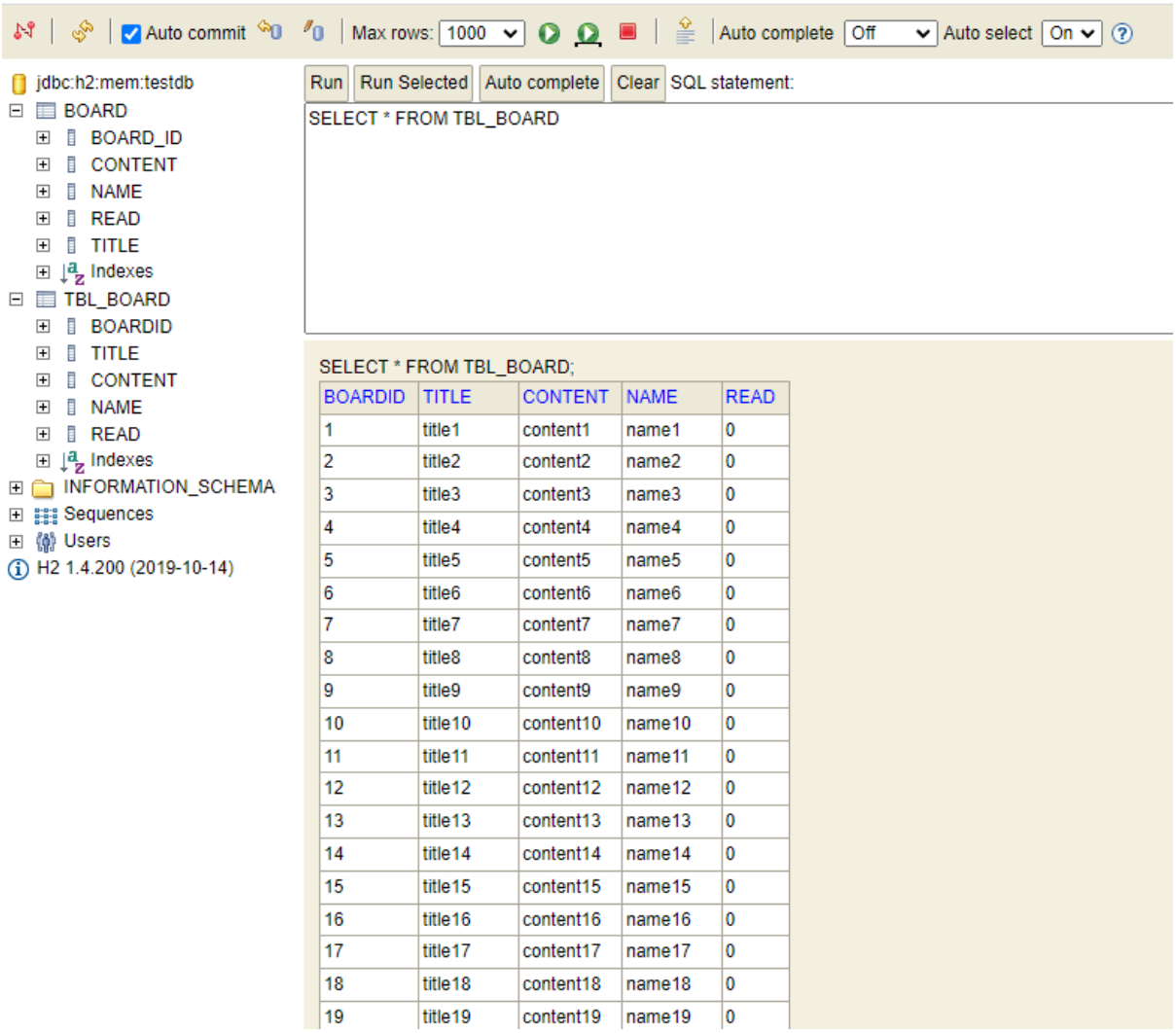
테스트를 위해 해당 database에 값들을 넣은 모습이다.(총 120개) 우리는 이 값을 바탕으로 원하던 동작들이 잘 작동하는 지 확인해볼 생각이다.(Condition3)



H2 Database Console을 통해 위 값들이 옳은 주소에 맞는 값으로 들어갔는지 확인해보기로 했다.



오른쪽에서 BOARD class와 TBL_BOARD에 각 알맞은 변수들이 배치됨을 알 수 있다. TBL_BOARD에 해당 값들이 잘 들어갔는지 확인하기 위해 SELECT * FROM TBL_BOARD를 통해 확인해보기로 했다.



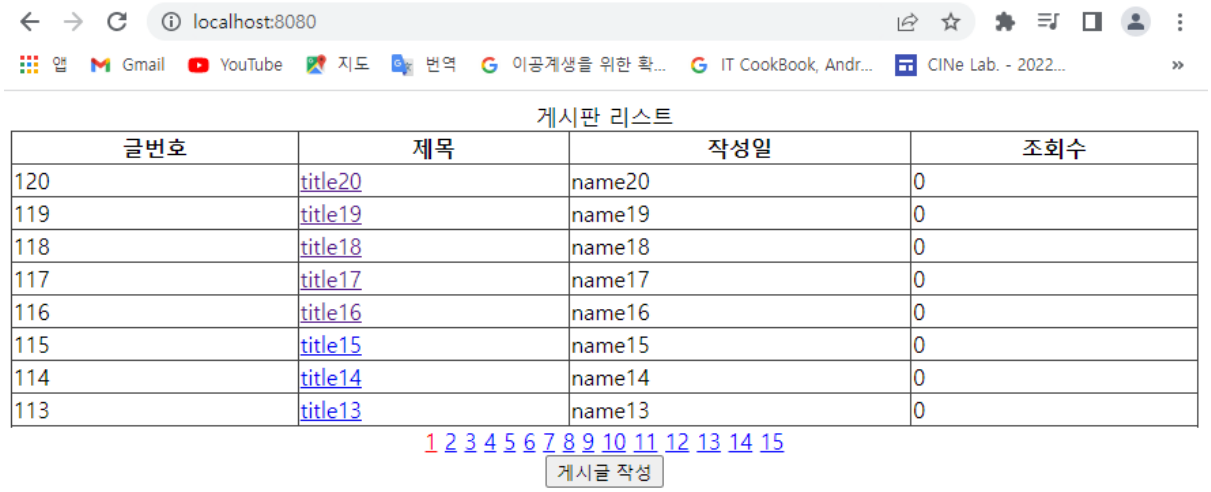
BOARDID	TITLE	CONTENT	NAME	READ
1	title1	content1	name1	0
2	title2	content2	name2	0
3	title3	content3	name3	0
4	title4	content4	name4	0
5	title5	content5	name5	0
6	title6	content6	name6	0
7	title7	content7	name7	0
8	title8	content8	name8	0
9	title9	content9	name9	0
10	title10	content10	name10	0
11	title11	content11	name11	0
12	title12	content12	name12	0
13	title13	content13	name13	0
14	title14	content14	name14	0
15	title15	content15	name15	0
16	title16	content16	name16	0
17	title17	content17	name17	0
18	title18	content18	name18	0
19	title19	content19	name19	0

114	title14	content14	name14	0
115	title15	content15	name15	0
117	title17	content17	name17	0
118	test	for test	2022-05-19	1
119	title19	content19	name19	0
120	title20	content20	name20	0
121	Spring boot	hello Java	2022-05-19	2

(120 rows, 2 ms)

Edit

위의 표에서 알 수 있듯, 120개의 board들이 schema.sql에서 선언된 형태의 table로 sql파일에서 실행한 만큼 즉, 의도한 대로 잘 입력된 모습이다.(Condition 2,3)



우선, **localhost:8080**로 접속시 바로 게시판이 뜨는 것을 잘 확인할 수 있었다.(Condition 1,4)

Page는 8개의 row로 이루어지며 글번호, 제목, 작성일, 조회수로 이루어져 있다. 게시글 작성을 위한 버튼과 제목에 대해 link가 걸려있는 것 또한 확인이 가능했고, Page number가 게시판 하단에 나열된 것(입력은 120개로 120/8=15, 적절한 수의 page number이다), 그리고 현재 Page가 빨갛게 출력되는 것도 확인할 수 있다.(Condition 5) 과정은 아래와 같다.

```
@GetMapping("/")                    //about http://localhost:8080/
public String init(Model model){
    model.addAttribute( attributeName: "list", service.boardList());    //to take all datas about database
    model.addAttribute( attributeName: "page",  attributeValue: 0);          //current page = 0(1)
    return "/boards/start";                                                  //go to start
}
```

(Controller를 통해 URL감지 후 service로 boardList를 통해 h2 Database의 board들을 list형태로 받아 온다. Page를 0으로 하고 start.html를 view로서 호출)

```
public List<Board> boardList(){return boardMapper.getList();}
//variable to take all data
```

```
List<Board> getList();      //function to change boards to list
```

각각 Service와 Repository의 명령어이다. Controller로부터 Service의 boardList가 불리고 이를 바탕으로 Repository의 getList가 불린 것을 확인할 수 있다. 이는 아래 xml파일로 연결되어 값을 받는다..

```
<select id="getList" resultType="com.example.kw2018202074.domain.Board">
    SELECT
        *
    FROM tbl_board;
select all elements in database
</select>
-get all board list in database-->
```

다음은 버튼 '게시글 작성'을 통해 upload를 위한 page에 들어간 모습이다. View를 통해 upload.html을바탕으로 출력된 Title, content, button이 확인된다.(위에서 서술) @GetMapping("/board/upload")를 바탕으로 Controller에서 진행되었다.(과정은 위에서 서술)

위는 제목만 입력시, 내용만 입력시, 아무것도 입력하지 않을 시의 모습이다. Controller의 @PostMapping("/board/upload")를 바탕으로 main.html로 가게 되었지만, XML내 uploadBoard에 관한 명령에서 if를 통해 값이 없는 경우 upload가 진행되지 않게 하여 게시판에 upload가 적용되지 않아 게시판이 유지되는 모습이다.

이후 다시 들어가 값들을 다음과 같이 넣어주고 등록을 눌렀다.

Controller의 @PostMapping("/board/upload")를 바탕으로 main.html을 View로 호출하며 uploadBoard에 관해 xml의 if를 통과하여 진행된다.

게시판 리스트

글번호	제목	작성일	조회수
121	hello	2022-05-19	0
120	title20	name20	0
119	title19	name19	0
118	title18	name18	0
117	title17	name17	0
116	title16	name16	0
115	title15	name15	0
114	title14	name14	0

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

게시글 작성

Xml의 insert가 진행되고 그 결과를 받아, 게시판 가장 상단에 제목을 입력된 hello로 하며 작성일을 띄워주는 모습이 확인된다. 글번호 또한 가장 큰 글번호에서 1이 추가된 모습이다.(Condition 6) page의 경우도 121/8=15.125로 1개를 더 담기 위해 16이 추가되었다.(html base)

hello

조회수 : 1

생성일자 : 2022-05-19

world!

이전으로 수정 삭제

다음은 hello 링크를 타고 들어간 모습이다.@GetMapping("/board/view")를 기반으로 값이 page값과 함께 넘어가게 되었다. 제목과 함께 조회수, 생성일자가 출력되고 있으며 내용 또한 출력되고 있다.(html를 기반으로 Controller에 의해 호출된 Database의 값이 출력됨.) 이전으로 돌아갈 수 있는 버튼, 수정을 위한 버튼, 삭제를 위한 버튼 또한 확인된다.(Condition 7)

게시판 리스트

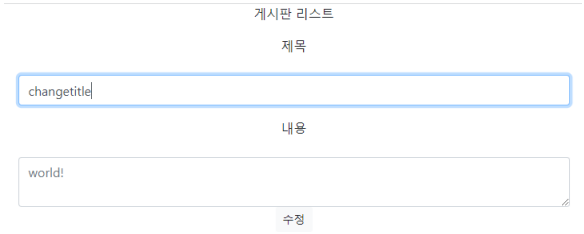
글번호	제목	작성일	조회수
121	hello	2022-05-19	1
120	title20	name20	0
119	title19	name19	0
118	title18	name18	0
117	title17	name17	0
116	title16	name16	0
115	title15	name15	0
114	title14	name14	0

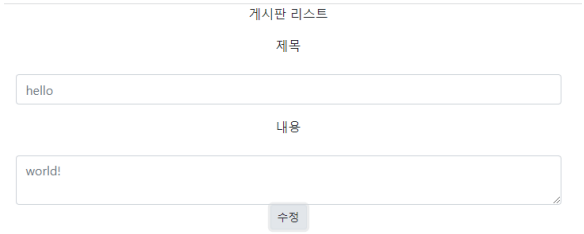
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

게시글 작성

위는 이전으로 버튼을 누른 결과이다. Page 1의 게시판으로 무사히 다시 돌아왔음을 확인했다. 이는 다른 page에 있는 게시글에 대해서도 마찬가지였다. @GetMapping("/board/view")내의 updateView에 의해 조회수가 상승한 것 또한 확인된다.

위는 수정 버튼을 누른 경우이다. 기존에 지닌 title과 content값이 예시로 올라와 있음을 확인할 수 있다. Html을 base로 Controller를 통해 H2 Database에서 받아온 view의 title, content내용이다.





위 수정 버튼에 대해 제목만 입력, 내용만 입력, 아무것도 입력하지 않은경우, 모두 입력을 테스트해보았다.

앞의 3가지 조건에 대해서는 다음과 같이 이전 값을 띄워주는 수정을 위한 화면을 띄워주는 것을 확인할 수 있었다. 상단에서 서술한 대로 Java Script로 인해 이미 받아왔던 값들을 다시 띄워주는 것이다.

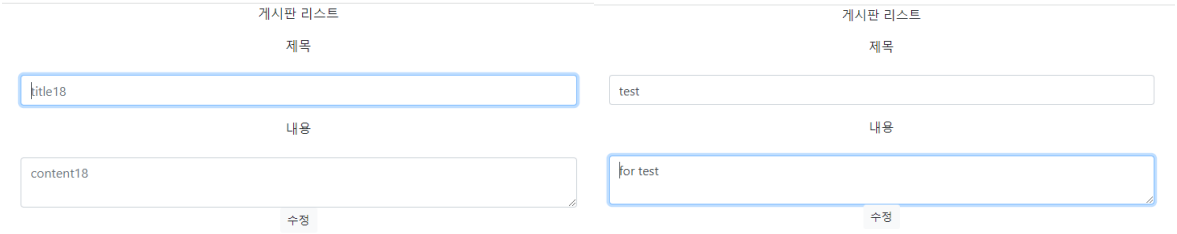
게시판 리스트

글번호	제목	작성일	조회수
121	Spring boot	2022-05-19	2
120	title20	name20	0
119	title19	name19	0
118	title18	name18	0
117	title17	name17	0
116	title16	name16	0
115	title15	name15	0
114	title14	name14	0

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

게시글 작성

반면 제목과 내용을 모두 입력한 경우, Javascript와 xml의 updateBoarad에서의 조건문을 통과할 수 있으므로 update가 성공적으로 이뤄지게 된다. Controller의 @PutMapping("/board/update")에 의해 main.html이 불려 게시판이 나오고 변경된 제목과 변경된 작성일이 잘 출력되고 있는 것을 확인할 수 있었다. 거듭된 수정에도 조회수는 수정을 제외한 값 만을 출력하는 것을 확인할 수 있었다.(view를 call할때만 updateView가 진행되므로 문제 없다.)



작성일의 재확인을 위해 다음에 대해서도 이를 진행하였다.

게시판 리스트

글번호	제목	작성일	조회수
121	Spring boot	2022-05-19	2
120	title20	name20	0
119	title19	name19	0
118	test	2022-05-19	1
117	title17	name17	0
116	title16	name16	0
115	title15	name15	0
114	title14	name14	0

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

게시글 작성

작성일이 잘 바뀌며 조회수도 1만이 상승함을 확인할 수 있었다.(Javascript를 바탕으로 얻은 현재 년, 월, 일을 넘겨주고 있다.)

이번엔 글번호 116의 title16에 대해 삭제버튼을 누름으로서 Delete를 진행하였다. @DeleteMapping("/board/delete")가 호출되고, service의 deleteBoard가 BoardId를 인자로 실행된다. 이후 H2Database가 이를 바탕으로 제거된다. 또한 return에 의해 main.html이 view로 불린다.

Main.html이 불리고 그 과정에서 delete가 정상적으로 진행된 H2DataBase의 값을 받아오게 되었다. 이를 통해 H2 DataBase내의 board가 정상적으로 지워짐을 확인했다.(Condition 8)

=>모든 Condition을 만족하며 이로서 프로젝트가 성공적으로 진행되었음을 알 수 있었다.