조회

지난시간에는 insert문을 사용해 글을 작성하여 DB에 저장하고 화면과 연동해 보았다. 이번 시간에는 작성한 글의 목록과 그 상세 내역을 조회하는 페이지를 작성해 보겠다.

1) 목록조회 기능

작성한 글들의 목록을 조회하는 기능이다. 기존 구조에서 findAll()이라는 메서드를 추가하고 매퍼에도 반영해 준다. 그다음 목록을 보여주는 화면을 구성하기 위해 list.html파일을 추가해 주었다.

#BoardController

```
@Controller
@RequiredArgsConstructor
@Slf4j
public class BoardController {
    private final BoardService boardService;

    @GetMapping("/save")
    public String save(){
        return "save";
    }
    @PostMapping("/save")
    public String save(BoardDTO boardDTO){
        boardService.save(boardDTO);
        return "redirect:/list"; //지난시간에는 글을 작성하면 리다이렉션이 되지 않았다.
        //반환타입을 String으로 변경하고 /list url로 리다이렉션 해준다.
```

```
        //글작성이 성공하면 localhost:8080/list로 페이지가 넘어간다.
    }
    @GetMapping("/list")
    public String findAll(Model model){
        List<BoardDTO> boardDTOList = boardService.findAll();
        model.addAttribute("boardList", boardDTOList);
        return "list";
    }

}
```

반환된 boardDTO가 담긴 리스트 형태의 자료구조를 "boardDTOList"라는 이름으로 모델에 담아서 반환한다. Model객체는 파라미터에 선언하면 스프링이 자동으로 생성해 준다.

DB에서 값을 성공적으로 반환받으면 list페이지로 화면을 전환한다.

#BoardService

```
@Service
@RequiredArgsConstructor
public class BoardService {
    private final BoardRepository boardRepository;

    public void save(BoardDTO boardDTO){
        boardRepository.save(boardDTO);
    }
```

```
    public List<BoardDTO> findAll() {
        return boardRepository.findAll();
    }
}
```

사실 현재까지는 Service계층에서 하는 일이 없다. 기능이 복잡해지면 비즈니스 로직이 들어가면서 Service계층에서 할 일이 생긴다. 비록 기능은 없지만 MVC구조를 지키기 위해서 Service를 거쳐서 가도록 한다.

#BoardRepositoty

```
@Repository
@RequiredArgsConstructor
public class BoardRepository {
    private final SqlSessionTemplate sql;

    public void save(BoardDTO boardDTO) {
        sql.insert("Board.save", boardDTO);
    }

    public List<BoardDTO> findAll() {
        System.out.println("findAll");
        return sql.selectList("Board.findAll");
    }
}
```

Board라는 namespace를 갖는 매퍼에서 id가 findAll인 sql을 찾아가도록 한다. "findAll"이라고 쳐야 하는데 "findALl"이라고 대소문자를 잘못 기입하여 한참 동안 오류를 찾았다. 이런 별것 아닌

문제에 시간 쏟는 게 제일 아깝다.

#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>
    <select id="findAll" resultType="board">
        select id, boardTitle, boardWriter, boardHits, date_format(createdAt, "%Y-%m-%d") as createdAt
            from board_table
            order by id desc
    </select>
</mapper>
```

select요청으로 board_table에 있는 데이터 목록을 가져와서 BoardDTO객체에 담아서 반환한다. resultType이 BoardDTO라도 앞서 BoardRepository에서 sql.selectList() 메서드를 호출했다면 마이바티스가 자동으로 리스트 타입으로 담아서 반환해 준다.

목록조회에서는 글 내용이 필요 없기 때문에 그 외의 데이터만 받아오도록 한다. 지금처럼 데이터의 양이 적을 때는 전부 받아와도 크게 문제가 되지 않지만 실제 서비스에서는 글 내용의 용량이 매우 큰 경우도 있기 때문에 실제 보이지 않는 데이터를 주고받느라 서버나 DB에 부하가 걸릴 수 있기 때문이다.

id는 자동으로 시간순에 따라 자동으로 생성되게 DB테이블을 생성할 때 설정해 주었기 때문에 id가 높은 순서대로 내림차순(desc) 정렬하게 되면 최신순으로 데이터를 정렬해서 반환한다.

#list.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        table, tr, td, th {
            border: 1px solid black;
            border-collapse: collapse;
        }
        th, td {
            padding: 10px;
        }
    </style>
<body>
<a href="/save">글쓰기</a>
<table>
    <tr>
        <th>번호</th>
        <th>제목</th>
```

```
        <th>작성자</th>
        <th>작성시간</th>
        <th>조회수</th>
    </tr>
    <tr th:each="board: ${boardList}">
        <td th:text="${board.id}"></td>
        <td>
            <a th:text="${board.boardTitle}" th:href="@{|/{board.id}|}"></a>
        </td>
        <td th:text="${board.boardWriter}"></td>
        <td th:text="${board.createdAt}"></td>
        <td th:text="${board.boardHits}"></td>
    </tr>
</table>
</body>
</html>
```

thymeleaf에서는 모델에서 받아온 값을 태그 내에서 바로 사용이 가능하다. 기존 jsp에서는 태그를 새로 생성해서 매우 귀찮고 지저분한 방식으로 forEach문을 구현해야 했지만, thymeleaf는 tr태그 내부에서 forEach문을 구현할 수 있다. 사용법도 기존의 Java나 javascript의 forEach문의 형태와 비슷하다. boardList는 컨트롤러에서 모델에 추가한 데이터이고 이를 board라는 변수로 각각 받아서 td태그에 사용할 수 있다.

또한 th:href라는 문법을 통해 동적으로 url로 리다이렉션 해줄 수 있다. 위 코드에서는 글의 번호인 id를 localhost:8080/id의 형태로 요청한다.

#thymleaf 공식문서 href사용법

<!-- Will produce 'http://localhost:8080/gtvg/order/details?orderId=3' (plus rewriting) -->

<a href="details.html"

th:href="@{http://localhost:8080/gtvg/order/details(orderId=${o.id})}">view

<!-- Will produce '/gtvg/order/details?orderId=3' (plus rewriting) -->

view

<!-- Will produce '/gtvg/order/3/details' (plus rewriting) -->

view

https://www.thymeleaf.org/doc/tutorials/3.1/usingthymeleaf.html#link-urls

Tutorial: Using Thymeleaf

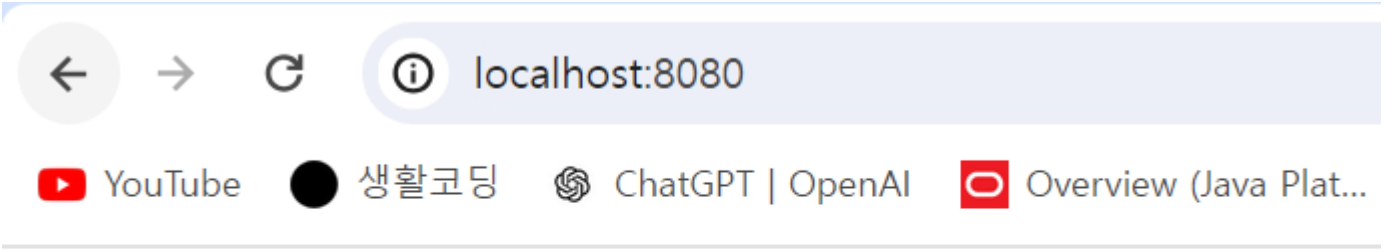
1 Introducing Thymeleaf 1.1 What is Thymeleaf? Thymeleaf is a modern server-side Java template engine for both web and standalone environments, capable of processing HTML, XML, JavaScript, CSS and even plain text. The main goal of Thymeleaf is to provide a

www.thymeleaf.org

변수는 "${}"로 동적으로 삽입해 주지만 url은 "@{}"라는 방식으로 동적으로 삽입할 수 있다. 절대경로와 상대경로 모두 사용 가능하다.

이제 잘 작동하는지 테스트해보자

#글목록 조회 테스트



hello high garden!!!

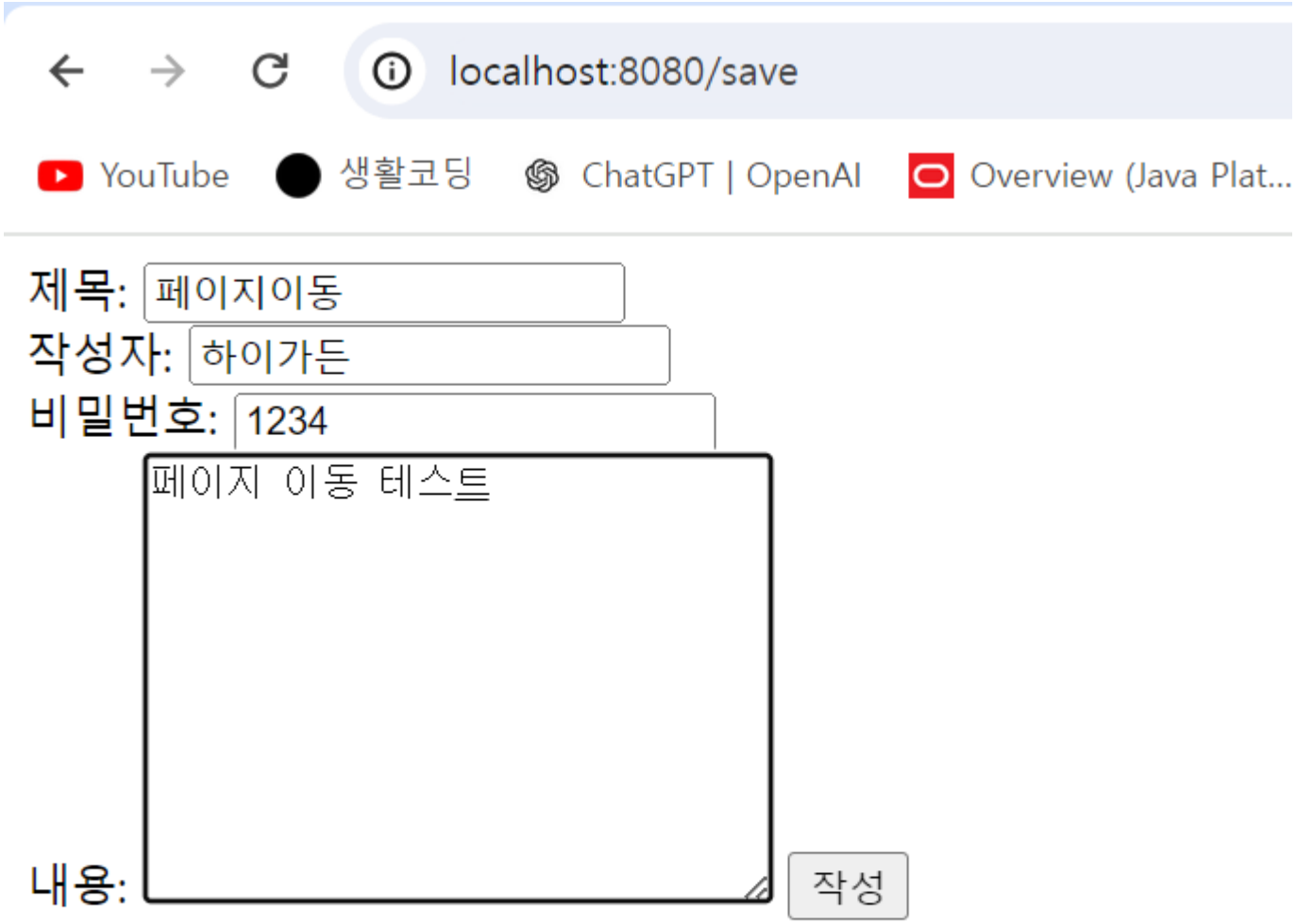
글작성 글목록

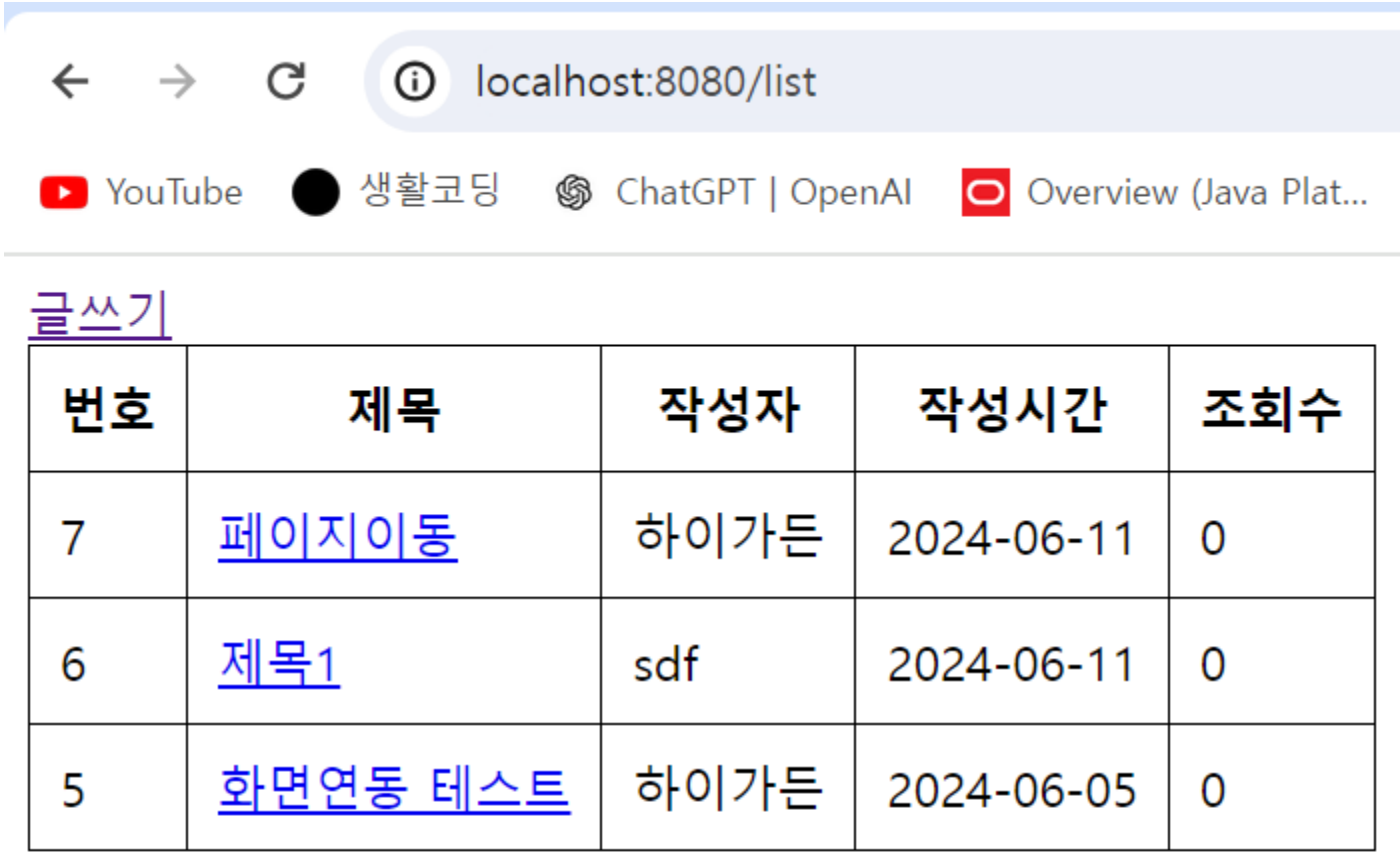
번호	제목	작성자	작성시간	조회수
6	제목1	sdf	2024-06-11	0
5	화면연동 테스트	하이가든	2024-06-05	0

인덱스페이지 ,글목록조회

글목록 버튼을 누르니 목록이 잘 조회된다. 이전에 작성한 글 목록이 조회되는 것을 확인할 수 있다.

#새로운 글 작성 테스트


localhost:8080/save
YouTube
생활코딩
ChatGPT | OpenAI
Overview (Java Plat...
제목: 페이지이동
작성자: 하이가든
비밀번호: 1234
페이지 이동 테스트
내용:
작성



번호	제목	작성자	작성시간	조회수
7	페이지이동	하이가든	2024-06-11	0
6	제목1	sdf	2024-06-11	0
5	화면연동 테스트	하이가든	2024-06-05	0

글작성 ,목록화면이동

BoardController의 save메서드에서 글작성이 성공적으로 수행되면 목록으로 페이지 이동이 일어나도록 수정하였다. 글작성이 성공하자 목록페이지로 이동하는 것을 확인할 수 있다.

2) 상세조회 기능, 조회수

글목록에서 글을 눌러 상세조회를 하게 되면 조회수가 된다. 즉, 상세조회를 하는 select기능과 해

당 글에 대한 조회수를 올려주는 update기능이 함께 고려되어야 한다.

#BoardController

```
    @GetMapping("/{id}")
     public String findById(@PathVariable("id") Long id, Model model) {
        //조회수 처리
        boardService.updateHits(id);
        //상세내용 가져오기
        BoardDTO boardDTO = boardService.findById(id);
        model.addAttribute("board", boardDTO);
        return "detail";
    }
```

@PathVariable이라는 어노테이션을 사용하면 자동으로 url요청을 변수에 담을 수 있다. 클라이언트 측에서 localhost:8080/id 형태로 요청을 했으니 이 패턴을 읽고 id라는 변수에 담아준다.

일단 요청이 들어오면 updateHits()라는 메서드로 조회수를 올린다. 그다음 글 상세내용을 조회하는 메서드가 실행된다.

그 다음 요청이 성공적으로 마무리되면 Model객체에 "boardDTO"를 담아서 뷰로 보내고 /detail로 페이지 이동한다.

#BoardService

```
    public void updateHits(Long id) {
        boardRepository.updateHits(id);
    }
```

```
public BoardDTO findById(Long id) {
    return boardRepository.findById(id);
}
```

마찬가지로 아직은 별다른 기능이 없다. 컨트롤러의 요청을 레포지토리로 연결해 주는 작업만 수행한다.

#BoardRepository

```
public void updateHits(Long id) {
    sql.update("Board.updateHits", id);
}
public BoardDTO findById(Long id) {
    return sql.selectOne("Board.findById", id);
}
```

각각의 요청을 매퍼로 연결해 준다. 조회수기능은 반환값이 따로 필요 없지만 상세조회는 글내용에 대한 데이터를 반환해야 하기 때문에 sql.selectOne으로 데이터를 받아 BoardDTO객체에 담아 반환한다.

#board-mapper.xml

```
<update id="updateHits" parameterType="Long">
    update board_table
    set boardHits = boardHits+1 where id=#{id}
</update>

<select id="findById" parameterType="Long" resultType="board">
    select id, boardTitle, boardWriter, boardPass, boardContents, boardHits,
```

```
        date_format(createdAt, "%Y-%m-%d") as createdAt
    from board_table
    where id=#{id}
</select>
```

기본적인 update와 select sql문이다. update문은 boardHits의 카운트를 현재값에서 1 추가해 주는 sql문이고 select는 id값을 가지고 해당하는 글을 조회한다.

#detail.html

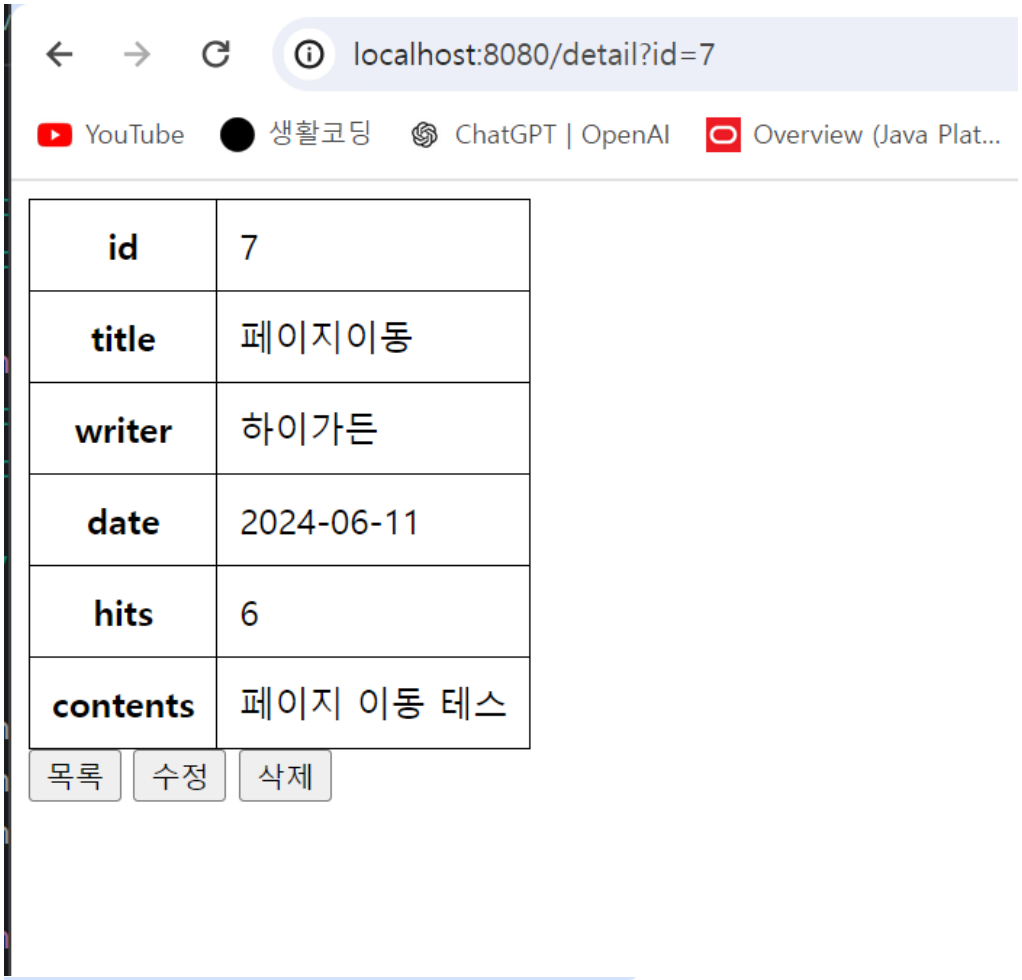
```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>detail</title>
    <style>
        table, tr, td, th {
            border: 1px solid black;
            border-collapse: collapse;
        }
        th, td {
            padding: 10px;
        }
    </style>
</head>
<body>
```

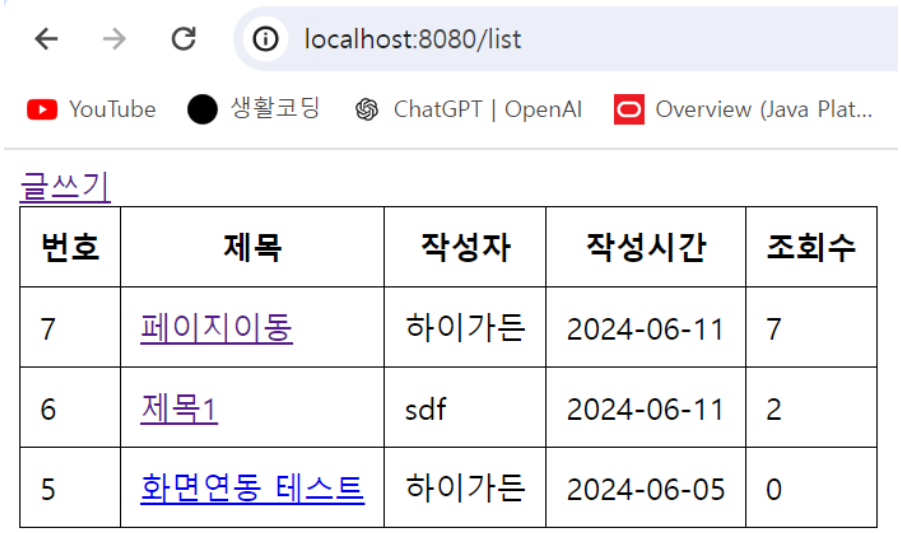
```
<table>
    <tr>
        <th>id</th>
        <td th:text="${board.id}"></td>
    </tr>
    <tr>
        <th>title</th>
        <td th:text="${board.boardTitle}"></td>
    </tr>
    <tr>
        <th>writer</th>
        <td th:text="${board.boardWriter}"></td>
    </tr>
    <tr>
        <th>date</th>
        <td th:text="${board.createdAt}"></td>
    </tr>
    <tr>
        <th>hits</th>
        <td th:text="${board.boardHits}"></td>
    </tr>
    <tr>
        <th>contents</th>
```

```
        <td th:text="${board.boardContents}"></td>
    </tr>
</table>
<button onclick="listReq()">목록</button>
<button onclick="updateReq()">수정</button>
<button onclick="deleteReq()">삭제</button>
</body>
<script th:inline="javascript">
    const listReq = () => {
        location.href = "/list";
    }
    const updateReq = () => {

    }
    const deleteReq = () => {

    }
</script>
</html>
```

상세조회한 데이터를 받아서 화면에 뿌려준다. 버튼이 세 개 있는데 목록화면으로 돌아가는 기능과 수정, 삭제버튼이 있다. 아직 수정기능과 삭제기능은 구현되지 않았기 때문에 함수만 버튼을 눌렀을 때 작동할 함수의 뼈대만 정의해 둔다.

#상세조회 테스트


localhost:8080/detail?id=7
YouTube
생활코딩
ChatGPT | OpenAI
Overview (Java Plat...
id
7
title
페이지이동
writer
하이가든
date
2024-06-11
hits
6
contents
페이지 이동 테스
목록
수정
삭제


localhost:8080/list
YouTube
생활코딩
ChatGPT | OpenAI
Overview (Java Plat...
글쓰기
번호
제목
작성자
작성시간
조회수
7
페이지이동
하이가든
2024-06-11
7
6
제목1
sdf
2024-06-11
2
5
화면연동 테스트
하이가든
2024-06-05
0

상세조회, 조회수 증가

url에 파라미터로 id=7이 잘 요청되었고 해당하는 데이터가 잘 불러와졌다. 글이 상세조회 됨에 따라서 조회수도 잘 증가하는 것을 확인할 수 있다.

이제 글작성과 조회기능은 마무리되었다. 다음 시간에는 수정과 삭제기능을 추가해 보도록 하자.