

Software Project

Project #2

담당교수 : 신영주

제출일 : 2018. 11. 16

학과 : 컴퓨터정보공학부

학번 : 2015722025

이름 : 정용훈

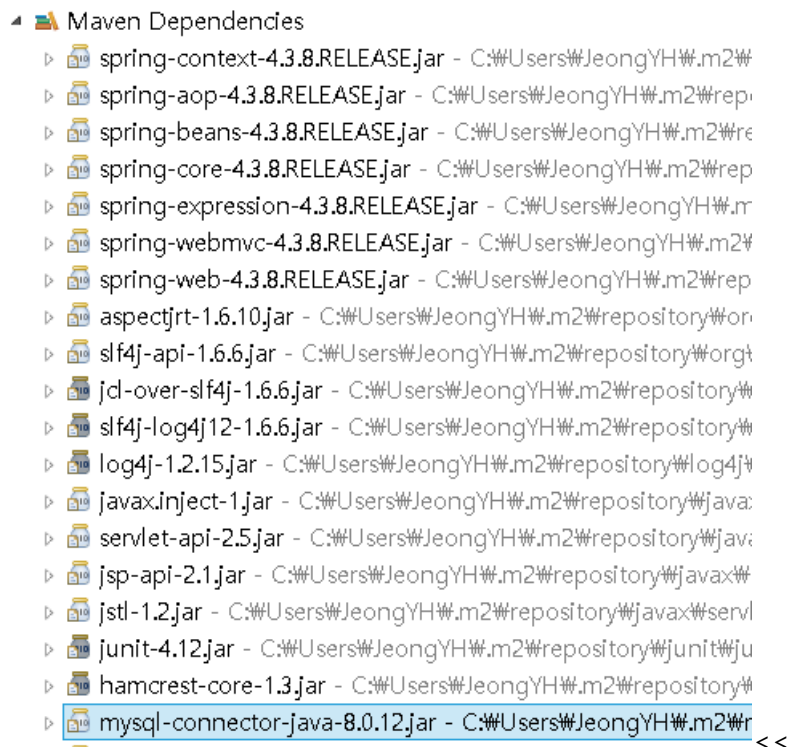
1. 튜토리얼 과정에 대한 설명 및 과정 캡처

지금부터 설명하는 과정은 DB의 table이 작성되고 기본적인 project의 설정이 맞춰진 상태에서 여러가지 test를 진행하는 것부터 시작한다.

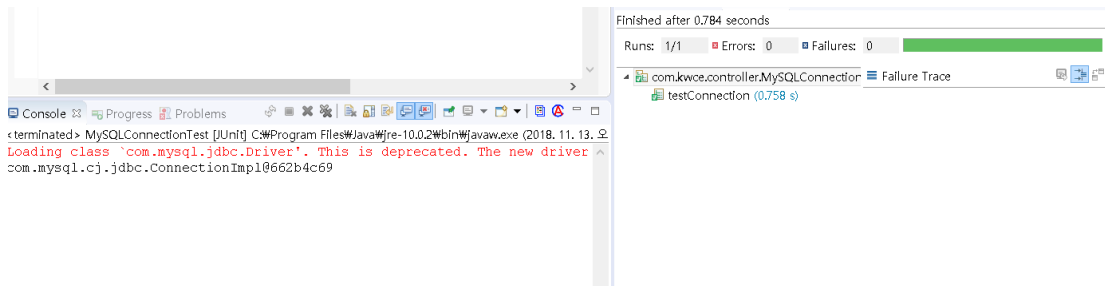
우선 여러가지 테스트와 최종적인 게시판을 구현하기 위해서는 DB와 Project가 잘 연결이 되었는지 확인해야한다.

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.12</version>
</dependency>
```

Test를 위해서는 JDBC라는 라이브러리가 필요하므로 해당 코드를 pom.xml에 넣어준다.

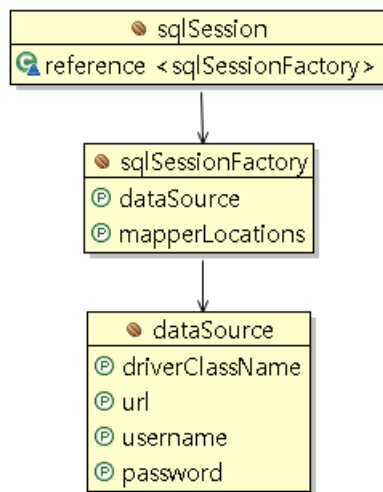


이후 connector-java-8.0.12.jar가 설치된 것을 확인해준다. 그 후 Junit을 통하여 test할 java파일에서 Run 시키면 아래와 같은 그림으로 결과가 나온다.

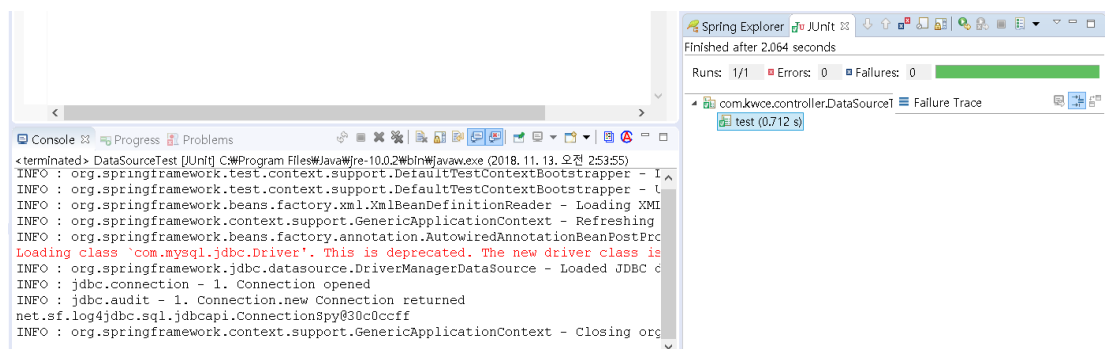


DB가 PDF의 과정에 따라 알맞게 연결된 것으로 확인된다. PDF의 과정에 따라 pom.xml에 모듈들을 추가적으로 생성하며 root-context.xml 파일을 수정한다. 수정을 완료하면 아래와 같이 Beans graph의 연관 관계를 확인할 수 있다.

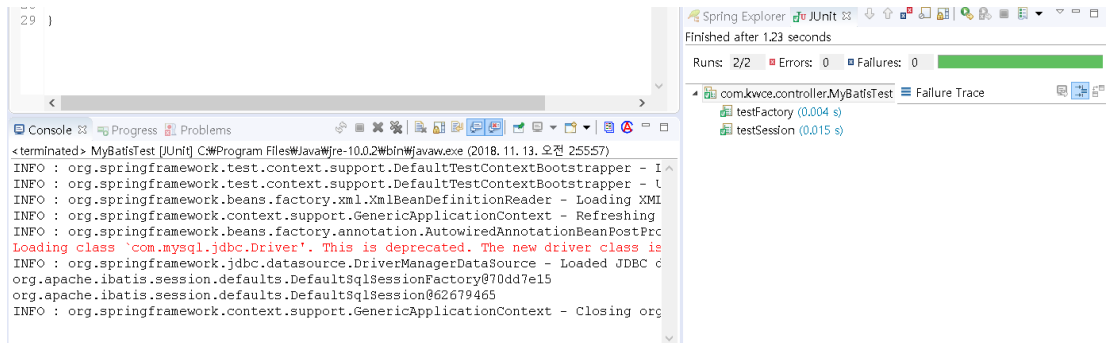
Spring Beans



다음으로는 로그 + MySQL와의 연결 테스트를 진행한다.

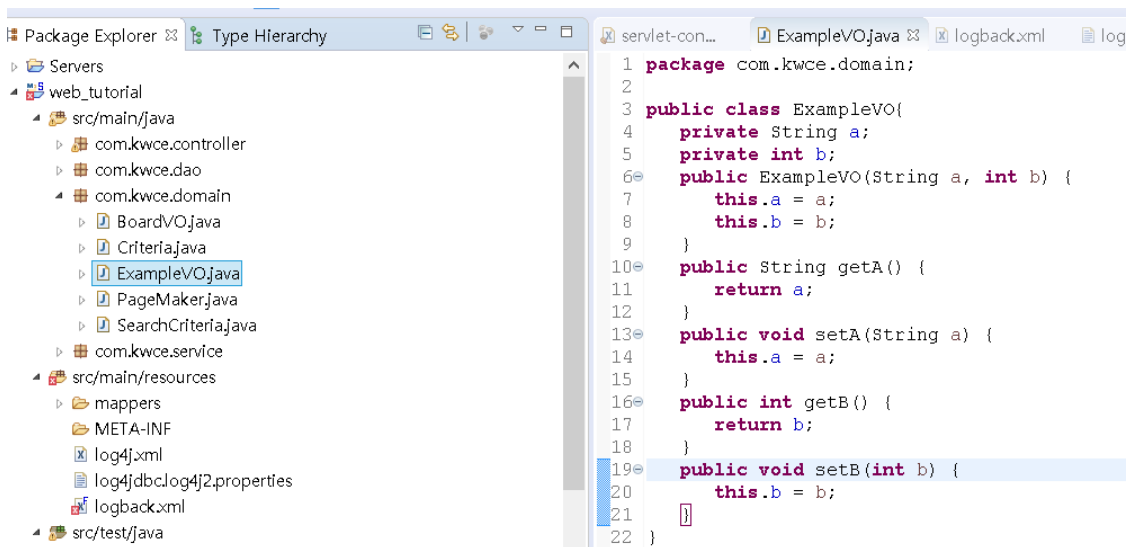


마찬가지로 PDF와 비교했을 때 정상적인 연결이 된 것을 확인할 수 있다. 마지막으로 하게 될 연결테스트는 MyBatis와의 연결 테스트이다.

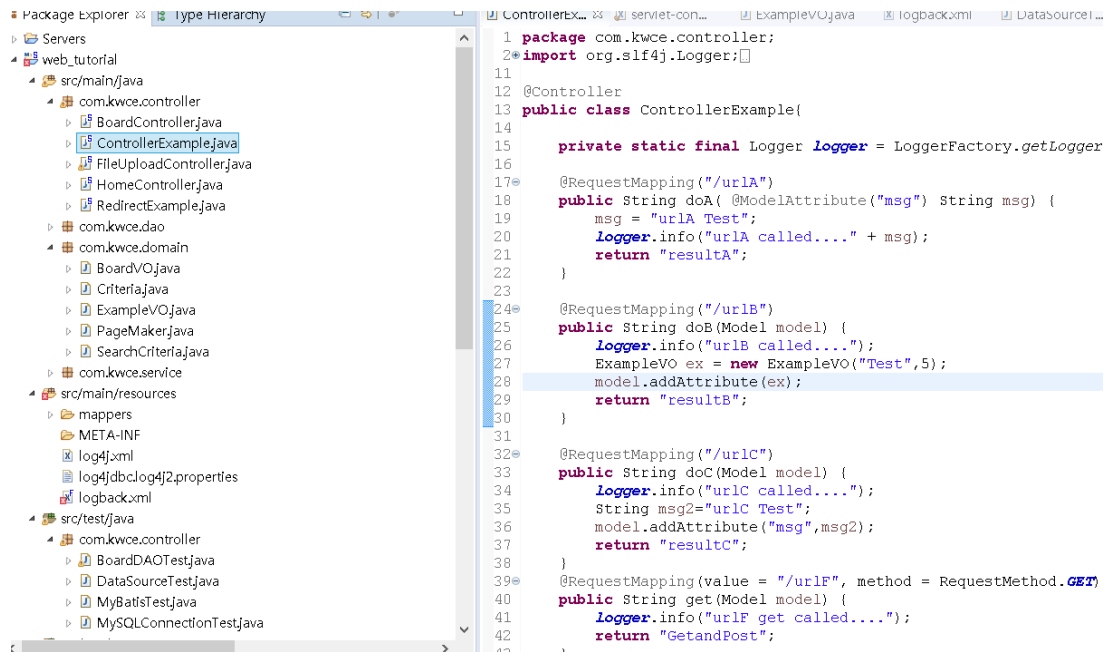


마찬가지로 연결이 된 것을 확인할 수 있다. 해당 과정을 test하기 위해서는 로그 파일을 나누는 작업과 추가사항이 있지만 PDF에 기본적으로 제공되는 것이기 때문에 자세한 설명은 생략했다.

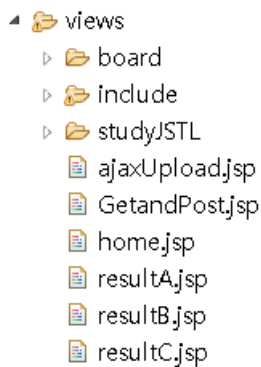
다음으로 설명하게 될 사항은 스프링 MVC에 대한 TEST 사항이다. Test전 PDF를 따라 servlet-context.xml파일을 수정하여 설정을한다.



다음으로 Domain Package를 생성해주며 test하게될 예제를 위해 ExampleVO를 정의한다.



위 사진처럼 test를 위한 Controller를 작성하고 각각의 함수를 정의하여 개발자가 원하는 URL과 내용을 직접 입력할 수 있다. @RequestMapping은 사용자가 인터넷 상에서 실질적으로 해당페이지를 찾을 수 있게 해당하는 URL에서 메소드를 처리할 수 있도록 하는 것이며 @ModelAttribute는 파라미터에 이름을 붙여 처리해서 view로 전달해주는 역할을 한다. 다음으로는 사용자가 실질적으로 볼 수 있는 view 즉, JSP파일에 관련된 설명이다.



다음 그림과 같이 resultA, resultB, resultC와 같은 JSP파일을 생성해 주며 해당 파일은 다음과 같은 내용으로 채워준다.

resultA.JSP

```
ControllerEx... resultA.jsp servlet-con... ExampleVO.java DataSourceT... MyBatisTest... 43
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose
4 <html>
5 <head>
6 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7 <title>Insert title here</title>
8 </head>
9 <body>
10   <span>ResultA ${msg}</span>
11
12 </body>
13 </html>
```

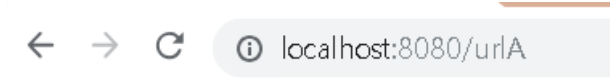
resultB.JSP

```
ControllerEx... resultA.jsp servlet-con... ExampleVO.java MyBatisTest... resultB.jsp 44
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose
4 <html>
5 <head>
6 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7 <title>Insert title here</title>
8 </head>
9 <body>
10   <span>${exampleVO.a}</span>
11   <span>${exampleVO.b}</span>
12
13 </body>
14 </html>
```

resultC.JSP

```
ControllerEx... resultA.jsp servlet-con... ExampleVO.java MyBatisTest... resultB.jsp resultC.jsp 44
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose
4 <html>
5 <head>
6 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7 <title>Insert title here</title>
8 </head>
9 <body>
10   <span>${msg}</span>
11
12 </body>
13 </html>
```

다음은 해당 MVC에 대한 결과를 볼 수 있다.



ResultA

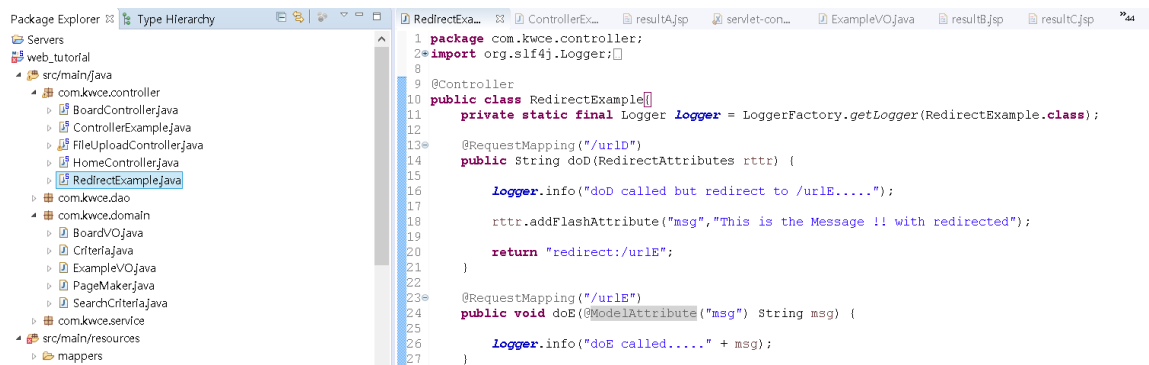


Test 5



urlC Test

해당 결과 값 모두 Controller에서 입력한 상태로 나오는 것을 확인할 수 있으며 URL주소 또한 Controller에서 설정한데로 들어가 페이지를 찾을 수 있는 것을 확인할 수 있다. JSP 파일은 실습을 통해 확인한 것과 같이 View에 대한 설정이며 동작을 하는 것은 Controller를 거치는 것을 확인할 수 있다. 다음은 특정 URL을 호출하지만 return을 통해 다른 URL을 호출하는 방법을 알아보겠다.



해당 코드를 살펴보면 urlD를 호출하지만 return으로 urlE를 호출하는 것을 확인할 수 있다. 해당 코드를 실행해보면 다음과 같다.



호출 결과 URL의 주소가 바로 바뀌는 것을 확인할 수 있다. 이를 이용하여 예제처럼 바로 바뀌는 것이 아니라 특정 행동(버튼 클릭)을 통하여 일정한 이벤트가 수행 되었을 경

우 다른 페이지로 넘어가거나 일정한 이벤트를 다시 수행하게 할 수 있도록 응용이 가능하다.

다음 튜토리얼의 내용은 GET과 POST의 방식차이를 설명한 것이다.

```
@RequestMapping(value = "/urlF", method = RequestMethod.GET)
public String get(Model model) {
    logger.info("urlF get called....");
    return "GetandPost";
}

@RequestMapping(value = "/urlF", method = RequestMethod.POST)
public String post(String StudentID, String name) {
    logger.info("urlF post called....");
    System.out.println(StudentID);
    System.out.println(name);
    return "redirect:/";
}
```

다음과 같이 Controller에 내용을 추가해주며, 아래와 같이 JSP파일을 생성해준다.



```
GetandPost.jsp  RedirectExa...  ControllerEx...  resultA.jsp  resultB.jsp  resultC.jsp  »45
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3  <!DOCTYPE html>
4  <html>
5  <head>
6  <meta charset="UTF-8">
7  <title>GET AND POST</title>
8  </head>
9  <body>
10 <h1>GET and POST TEST</h1>
11
12 <form method="POST" action=../urlF>
13 <table>
14 <tr>
15 <td><label>학번</label></td>
16 <td><input type="text" name="StudentID"></td>
17 </tr>
18
19 <tr>
20 <td><label>이름</label></td>
21 <td><input type="text" name="name"></td>
22 </tr>
23 </table>
24 <input type="submit" value="전송">
25 </form>
26 </body>
27 </html>
```


해당 URL을 찾아 들어가면 다음과 같은 페이지를 확인할 수 있다.

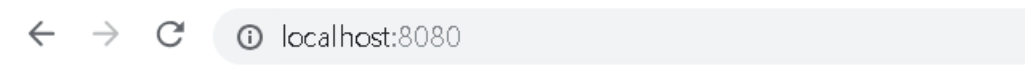


GET and POST TEST

학번

이름

전송 버튼을 클릭하면 다음과 같은 페이지로 넘어간다.



Hello world!

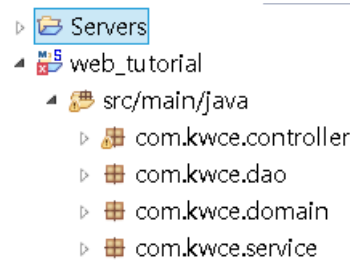
The time on the server is 2018년 11월 13일 오전 3시 33분 59초 KST.

전송을 눌러 다음과 같은 페이지로 넘어가는 이유는 JSP에 있는 form때문에 그러한데 action과 방식인 POST가 일치하며 Controller의 Post함수가 반응하여 return값으로 home.JSP를 설정했기 때문에 해당 페이지로 넘어오게 된 것이다. 좀더 자세한 GET과 POST방식의 전송방법은 4번 문항에서 설명하겠다.

2. 게시판 기능 동작에 대한 과정 설명 및 캡처

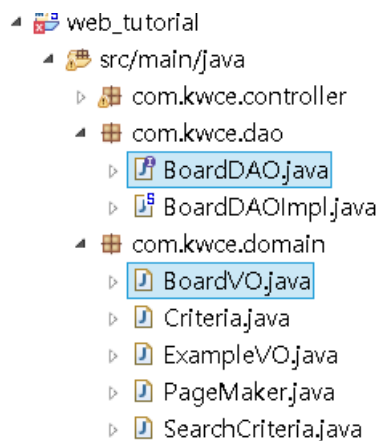
우선 게시판의 동작을 설정하기전 게시판의 틀 구현부터 실행해야 한다. 이제부터 다루게 되는 것은 Data Base와 MyBatis를 같이 다루게 된다. 튜토리얼 1번째 문항에서는 각각의 설정과 DB Table을 설정하였으며 2번째 문항에서는 MVC에 대한 여러가지 예제를 통해 알아보았다. 우리가 구현하게 될 게시판은 MVC도 이용하며 글을 올리고 수정하고 삭제 하기 위한 DB가 존재해야 하기 때문에 앞선 예제를 다룬 것이다.

첫번째로 게시판을 만들기전에 개발 패키지를 구성해야 한다.



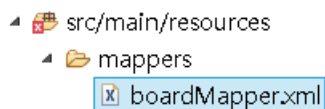
The screenshot shows the package structure of the 'web_tutorial' project. Under 'src/main/java', there are four packages: 'com.kwce.controller', 'com.kwce.dao', 'com.kwce.domain', and 'com.kwce.service'. The text indicates that these packages are to be created and populated with classes.

다음과 같이 패키지를 구성하며 Domain 패키지 내에는 BoardVO를 작성하여 추가하고 DAO패키지에는 Board DAO를 작성하여 추가한다.



This screenshot provides a more detailed view of the package structure. Under 'com.kwce.dao', there are 'BoardDAO.java' and 'BoardDAOImpl.java'. Under 'com.kwce.domain', there are 'BoardVO.java', 'Criteria.java', 'ExampleVO.java', 'PageMaker.java', and 'SearchCriteria.java'. The 'BoardVO.java' and 'BoardDAO.java' files are highlighted with blue boxes.

그 후 Mapper를 작성하기 위해 src/main/resources의 mappers안에 boardMapper.xml 파일을 생성하여 작성한다.



The screenshot shows the 'src/main/resources' directory with a subdirectory 'mappers'. Inside 'mappers', the file 'boardMapper.xml' is highlighted with a blue box.

그 후 Mapper파일을 참조할 수 있도록 root-context.xml파일을 수정하면 된다.

이렇게 설정을 마치고 게시판 작성 전 글생성Test를 통해 확인한다.

생성 TEST

```

        ?)) returned net.sf.log4jdbc.sql.jdbcapi.PreparedStatementSpy@20765ed5
INFO : jdbc.audit - 33. PreparedStatement.setString(1, "33번째 글 제목입니다.") returned
INFO : jdbc.audit - 33. PreparedStatement.setString(2, "33번째 글 내용입니다.") returned
INFO : jdbc.audit - 33. PreparedStatement.setString(3, "2015722033") returned
INFO : jdbc.sqlonly - INSERT INTO tbl_board (title, content, writer) VALUES ('33번째 글 제목입니다.', '33번째 글 내용입니다.', '2015722033')
INFO : jdbc.sqltiming - INSERT INTO tbl_board (title, content, writer) VALUES ('33번째 글 제목입니다.', '33번째 글 내용입니다.', '2015722033')
(executed in 9 msec)
INFO : jdbc.audit - 33. PreparedStatement.execute() returned false
INFO : jdbc.audit - 33. PreparedStatement.getUpdateCount() returned 1
INFO : jdbc.audit - 33. PreparedStatement.isClosed() returned false
INFO : jdbc.audit - 33. PreparedStatement.close() returned
INFO : jdbc.connection - 33. Connection closed
INFO : jdbc.audit - 33. Connection.close() returned
INFO : jdbc.connection - 34. Connection opened
INFO : jdbc.audit - 34. Connection.new Connection returned
INFO : jdbc.audit - 34. Connection.getAutoCommit() returned true
INFO : jdbc.audit - 34. PreparedStatement.new PreparedStatement returned
INFO : jdbc.audit - 34. Connection.prepareStatement(INSERT
        INTO tbl_board

```

Spring Explorer JUnit

Finished after 3.955 seconds

Runs: 4/4 (3 skipped) Errors: 0 Failures: 0

com.kwce.controller.BoardDAOTest [Runner: Failure Trace]

bno	title	content	writer	regdate	viewcnt
25	19번째 글 제목입니다.	19번째 글 내용입니다.	2015722019	2018-11-13 04:12:19	0
26	20번째 글 제목입니다.	20번째 글 내용입니다.	2015722020	2018-11-13 04:12:19	0
27	21번째 글 제목입니다.	21번째 글 내용입니다.	2015722021	2018-11-13 04:12:19	0
28	22번째 글 제목입니다.	22번째 글 내용입니다.	2015722022	2018-11-13 04:12:19	0
29	23번째 글 제목입니다.	23번째 글 내용입니다.	2015722023	2018-11-13 04:12:19	0
30	24번째 글 제목입니다.	24번째 글 내용입니다.	2015722024	2018-11-13 04:12:19	0
31	25번째 글 제목입니다.	25번째 글 내용입니다.	2015722025	2018-11-13 04:12:19	0
32	26번째 글 제목입니다.	26번째 글 내용입니다.	2015722026	2018-11-13 04:12:19	0
33	27번째 글 제목입니다.	27번째 글 내용입니다.	2015722027	2018-11-13 04:12:19	0
34	28번째 글 제목입니다.	28번째 글 내용입니다.	2015722028	2018-11-13 04:12:19	0
35	29번째 글 제목입니다.	29번째 글 내용입니다.	2015722029	2018-11-13 04:12:19	0
36	30번째 글 제목입니다.	30번째 글 내용입니다.	2015722030	2018-11-13 04:12:19	0
37	31번째 글 제목입니다.	31번째 글 내용입니다.	2015722031	2018-11-13 04:12:19	0

SQL에서 표를 통해 확인한 결과 잘 생성된 것을 볼 수 있다.

글 삭제 TEST

```

@Test
public void testDelete() throws Exception {
    dao.delete(1);
}
@Test @Ignore
public void testListPaging() throws Exception {
    int page = 1;
    List<BoardVO> boards = dao.listPaging(page);
    for (BoardVO board : boards) {
        logger.info(board.getBno() + " : " + board.getTitle());
    }
}
@Test @Ignore
public void testListCriteria() throws Exception {

```

Spring Explorer JUnit

Finished after 1.974 seconds

Runs: 4/4 (3 skipped) Errors: 0 Failures: 0

com.kwce.controller.BoardDAOTest [Runner: Failure Trace]

- testListPaging (0.000 s)
- testListCriteria (0.000 s)
- testCreate (1.473 s)
- testDelete (0.466 s)

위 기능을 구현하며 기본적인 게시판을 생성하기 위한 틀이 잡혔다. 페이지 디자인을 위한 파일을 제공받아 다운로드 받은 후 게시판을 위한 Service패키지의 BoardService를 구현하고 Controller을 구현한다.

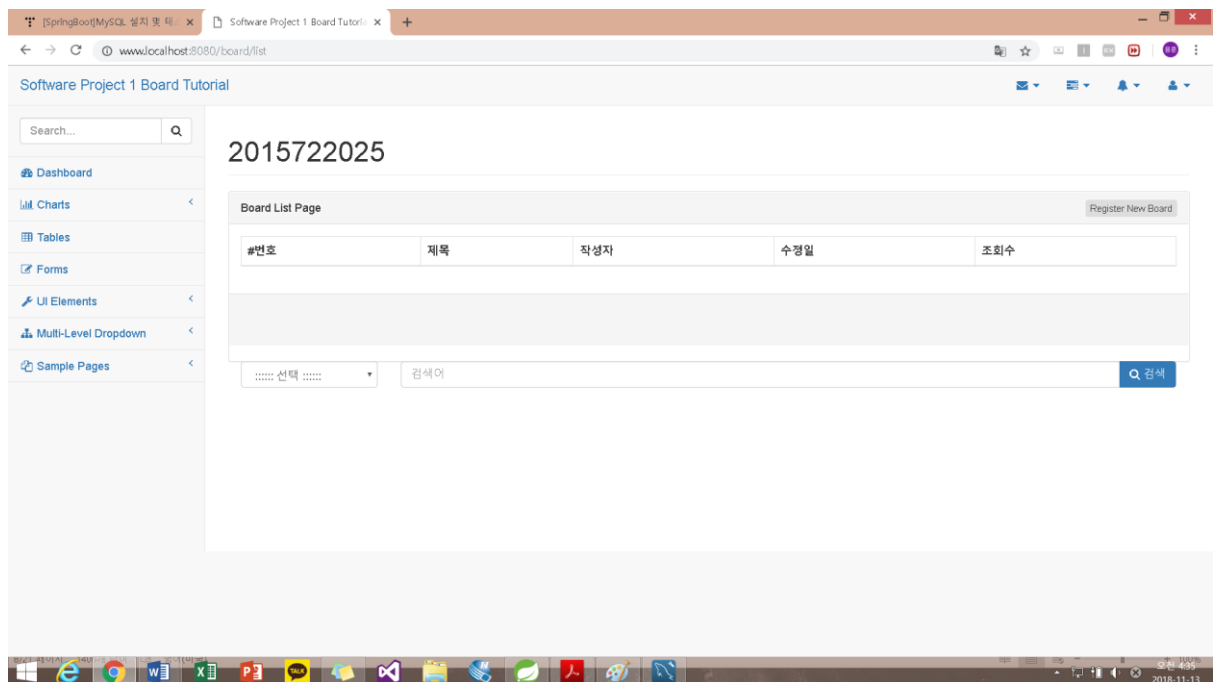
```
@RequestMapping("/board/*")
public class BoardController{

    private static final Logger logger = LoggerFactory.getLogger(BoardCon

    @Autowired
    private BoardService service;

    @RequestMapping(value = "/list")
    public String list(@ModelAttribute("criteria") SearchCriteria criteri
        logger.info("list() : called...");
        logger.info(criteria.toString());
        List<BoardVO> list = service.list(criteria);
        PageMaker pageMaker = new PageMaker();
        pageMaker.setCriteria(criteria);
        pageMaker.setTotalCount(service.listCount(criteria));
        model.addAttribute("list", list);
        model.addAttribute("pageMaker", pageMaker);
        model.addAttribute("totalCount", service.listCount(criteria));
        return "board/list";
    }
}
```

그 후 www.localhost:8080/board/list 해당 URL을 통해 들어가면 다음과 같은 화면을 볼 수 있다.



추가적으로 부트스트랩이 적용된 것도 확인할 수 있다.

글 조회를 눌렀을 때 해당 글을 구분하기 위하여 아래 코드처럼? bon=xxx 를 붙여 페이지를 구분할 수 있도록 설정한다.

```
<c:forEach var="boardVO" items="${list}">
<tr>
<td>${boardVO.bno}</td>
<td><a href="${path}/board/read${pageMaker.makeSearch(pageMaker.criteria.page)}&bno=${boardVO.bno}">${boardVO.title}</a>
<td>${boardVO.writer}</td>
<td><fmt:formatDate pattern="yyyy-MM-dd HH:mm"
value="${boardVO.regdate}" /></td>
<td><span class="badge bg-red">${boardVO.viewcnt}</span></td>
</tr>
</c:forEach>
```

그리고 다시 board/list 링크를 타서 확인하면 다음과 같이 TEST를 통해 구현한 list들을 확인할 수 있다.

2015722025

Board List Page					Register New Board
#번호	제목	작성자	수정일	조회수	
59	59번째 글 제목 입니다.	2015722059	2018-11-14 00:32	0	
58	58번째 글 제목 입니다.	2015722058	2018-11-14 00:32	0	
57	57번째 글 제목 입니다.	2015722057	2018-11-14 00:32	0	
56	56번째 글 제목 입니다.	2015722056	2018-11-14 00:32	0	
55	55번째 글 제목 입니다.	2015722055	2018-11-14 00:32	0	

다음으로 구현할 기능은 게시판에서 글을 등록하는 기능을 구현하는 것이다. 글 등록 페이지는 GET방식과 POST 방식으로 구현하며, 글을 등록하면 해당 내용을 DB로 이동시켜 주면서 저장한다. 우선 글 등록 페이지를 위한 View를 작성한다. 해당 JSP는 아래와 같이 작성한다.

```

1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4 <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
5 <%@include file="../include/header.jsp"%>
6 <div class="row">
7 <div class="col-lg-12">
8 <h1 class="page-header">2015722025</h1>
9 </div><!-- /.col-lg-12-->
10 </div><!-- /.row -->
11 <div class="row">
12 <div class="col-lg-12">
13 <div class="panel panel-default">
14 <div class="panel-heading">Board Register</div><!-- /.panel-heading -->
15
16 <div class="panel-body">
17 <form role="form" action="/board/register" method="POST">
18 <div class="form-group">
19 <label>Title</label><input class="form-control" name='title'>
20 </div>
21 <div class="form-group">
22 <label>Text area</label>
23 <textarea class="form-control" rows= "3" name='content'></textarea>
24 </div>
25 <div class="form-group">
26 <label>Writer</label><input class="form-control" name='writer'>
27 </div>
28 <button id="SubmitBtn" type="submit" class="btn btn-default">Submit Button</button>
29 <button type="reset" class="btn btn-default">Reset Button</button>
30 <button id="ListBtn" class="btn btn-default">List</button>
31 </form>
32 </div><!-- end panel-body -->
33 </div><!-- end panel-body -->
34 </div><!-- end panel -->
35 </div><!-- /.row -->

<script>
$(document).ready(function(){
    var Formobj= $("form[role='form']");

    $("#ListBtn").on("click", function(){
        Formobj.attr("action", "/board/list");
        Formobj.attr("method", "POST");
        Formobj.submit();
    });

    $("#SubmitBtn").on("click", function(){
        if($("#input[name=title]").val()==""){
            alert("제목을 입력하세요!");
            $("#input[name='title']").focus();
            return false;
        }
        if($("#input[name=content]").val()==""){
            alert("내용을 입력하세요!");
            $("#input[name='content']").focus();
            return false;
        }
        if($("#input[name='writer']").val()==""){
            alert("이름을 입력하세요!");
            $("#input[name='writer']").focus();
            return false;
        }
    });
});
</script>
<%@include file="../include/footer.jsp"%>

```

코드 작성이 완료되었으면 다시 게시판으로 이동하여 해당 동작이 정확히 잘되는지 확인 해준다.

2015722025

Board Register

Title

글 등록 TEST

Text area

글 등록 TEST

Writer

글 등록 TEST

Submit Button

Reset Button

List

trial

www.localhost:8080 내용:
저리가 완료되었습니다.

확인

2015722025

Board List Page

Register New Board

#번호	제목	작성자	수정일	조회수
100	글 등록 TEST	글 등록 TEST	2018-11-14 00:41	0
99	99번째 글 제목 입니다.	2015722099	2018-11-14 00:32	0
98	98번째 글 제목 입니다.	2015722098	2018-11-14 00:32	0
97	97번째 글 제목 입니다.	2015722097	2018-11-14 00:32	0
96	96번째 글 제목 입니다.	2015722096	2018-11-14 00:32	0

새롭게 글이 올라간 것을 확인 할 수 있다. 다음 구현하게 될 게시판의 기능은 list에 등록된 글들을 조회하는 기능을 구현하는 것이다. 먼저 조회에 대한 Controller를 추가해야 한다.

```
@RequestMapping(value = "/read", method = RequestMethod.GET)
public String read(@RequestParam("bno") int bno, Model model) throws Exception {
    logger.info("read page get...");
    model.addAttribute(service.read(bno));
    return "board/read";
}
```

위 코드를 BoardController.java에 추가한 후 View를 위한 JSP파일을 작성하도록 한다. JSP의 전체적인 작성코드는 PDF를 따랐으며 아래 코드는 Bno값을 저장했다가 수정과 삭제를 구현할 때 같이 전달해주기 위한 코드이다.

```
<form id="Form" method="post">
<input type="hidden" name="bno" value="${ boardVO.bno } ">
</form>
```

```
$(document).ready(function() {
    var Formobj=$("#Form");
    $("#ModiBtn").on("click",function() {
        Formobj.attr("action","/board/modify");
        Formobj.attr("method","get");
        Formobj.submit();
    });
    $("#ListBtn").on("click",function() {
        self.location="/board/list";
    });
    $("#DelBtn").on("click",function() {
        Formobj.attr("action","/board/remove");
        Formobj.submit();
    });
});
```

해당 코드는 read.jsp에서 modify, list, delete버튼에 대한 이벤트를 구현한 것이다. 여기까지 코드 작성을 마치고 게시판을 확인하면 올라온 글들을 조회할 수 있다.



위 결과화면은 글을 조회한 결과로 Register를 통해 넣은 글을 DB를 기반으로 해서 다시 조회할 수 있는 것을 확인할 수 있다. 다음으로 구현하는 기능은 글을 수정하는 기능이다. 글 수정을 구현하기 전 먼저 Controller를 수정해야한다. Controller에 다음과 같은 코드를 추가해서 넣어준다.

```
@RequestMapping(value = "/modify", method = RequestMethod.GET)
public String modifyGET(@RequestParam("bno")int bno, Model model) throws Exception{
    logger.info("modify page get...");
    model.addAttribute(service.read(bno));
    return "board/modify";
}

@RequestMapping(value="/modify", method = RequestMethod.POST)
public String modifyPOST(BoardVO board, RedirectAttributes rttr) throws Exception{
    logger.info("modify page post.....");
    logger.info(board.toString());
    service.modify(board);
    rttr.addFlashAttribute("msg", "SUCCESS");
    return "redirect:/board/list";
}
```


Controller를 수정했다면 수정페이지를 구현한다. 전체적인 코드는 PDF를 따라가며 다음 코드는 수정 페이지에서의 이벤트를 구현한 것이다.

```
<script>
$(document).ready(function() {
    $("#ListBtn").on("click", function() {
        var formobj=$("#form[role='form']");
        formobj.attr("action", "/board/list");
        formobj.attr("method", "get");
        formobj.submit();
    });
    $("#ModiBtn").on("click", function() {
        if($("#input[name=title]").val()=="") {
            alert("제목을 입력하세요!");
            $("#input[name='title']").focus();
            return false;
        }
        if($("#input[name=content]").val()=="") {
            alert("내용을 입력하세요!");
            $("#input[name='content']").focus();
            return false;
        }
        return true
    });
});
</script>
```

해당 View까지 작성을 완료했다면 게시판에서 수정 기능을 수행할 수 있다.

2015722025

Board Modify Page

Bno

101

Title

글 조회 TEST

Text area

글 조회 TEST

Writer

글 조회 TEST

Modify List

rial

www.localhost:8080 내용:
처리가 완료되었습니다.

확인

2015722025

Board List Page					Register New Board
#번호	제목	작성자	수정일	조회수	
101	글 수정TEST	글 조회 TEST	2018-11-14 01:04	3	

정상적으로 글이 수정된 것을 확인할 수 있습니다. 다음으로는 글을 삭제하는 기능입니다.

글을 삭제하는 기능은 따로 View page를 만들 필요없이 read page에서 delete버튼을 눌러 해당 이벤트가 발생할 수 있도록 기능을 추가해주면 됩니다. 해당 기능을 추가해주는 과정은 아래와 같습니다.

```
@RequestMapping(value = "/remove", method = RequestMethod.POST)
public String remove(@RequestParam("bno") int bno, RedirectAttributes rttr) throws Exception{
    service.remove(bno);
    logger.info("remove page post.....");
    rttr.addFlashAttribute("msg", "SUCCESS");
    return "redirect:/board/list";
}
```

해당 코드를 Controller에 추가했다면 조회 페이지에서 Delete버튼이 동작하는 것을 확인할 수 있다.

2015722025

Board Read Page

Bno

100

Title

글 등록 TEST

Text area

글 등록 TEST

Writer

글 등록 TEST

Modify List Delete

처리가 완료되었습니다.

확인

2015722025

Board List Page Register New Board

#번호	제목	작성자	수정일	조회수
99	99번째 글 제목 입니다.	2015722099	2018-11-14 00:32	0
98	98번째 글 제목 입니다.	2015722098	2018-11-14 00:32	0
97	97번째 글 제목 입니다.	2015722097	2018-11-14 00:32	0
96	96번째 글 제목 입니다.	2015722096	2018-11-14 00:32	0
95	95번째 글 제목 입니다.	2015722095	2018-11-14 00:32	0

해당 게시글이 list에서 삭제된 것을 확인할 수 있다. 기본적인 게시글의 동작 구현은 삭제 구현까지이며 앞으로는 추가구현을 하게 된다. 지금까지 구현한 기능을 살펴보면 DB에 대한 설정을 앞에서 맞춰주고 MVC에 관련한 기능을 계속해서 구현하였는데 View에서는 사용자가 실질적으로 보는 화면을 다루었고 Controller는 View페이지에서 특정 이벤트가 일어나면 함수를 실행시켜 알맞은 기능을 할 수 있게끔 하는 것을 확인할 수 있었다. 이를 이용하여 추가 구현을 할 수 있는데 이는 다음 항목에서 설명할 예정이다.

3. 추가 기능 구현

추가 기능을 구현하는 것 자체가 처음부터 현재 구현을 완료한 상태까지 쉽지 않은 과정이었다. 실질적인 수업진도와 프로젝트의 과정이 맞지 않아 개인적으로는 많이 힘들었던 부분이다. 우선 추가기능을 구현한 현황으로는 조회수, paging, 검색, 파일 & 이미지 업로드이다. 기본적으로 PDF에서 예시한 추가기능을 구현했으며 많은 블로그와 카페를 참고했다.

(1) 조회수

가장 먼저 소개할 기능은 조회수 기능이다. 튜토리얼을 따라 게시판을 구현했다면 글 리스트에 조회수가 보이지만 따로 설정을 해주지 않았기 때문에 해당 글의 조회수가 올라가지 않는 것을 확인할 수 있다. 조회수와 관련된 ViewCnt변수와 함수는 기본적인 게시물 구현에 있어 PDF에서 이미 정의된 내용이다. 실질적으로 +1이 되는 기능이 없기 때문에 이와 같은 문제가 발생한 것이며 이에 대한 기능만 추가해주면 정상적으로 동작할 것으로 예상되었다.

```
<update id="updateViewCnt">
    UPDATE tbl_board
    SET viewcnt = viewcnt + 1
    WHERE bno = #{bno}
</update>
```

해당하는 코드를 boardMapper.xml 에 추가해준다. 다음으로 확인할 것은 ViewCnt를 update해주는 관련 소스 코드이다.

```
public void updateViewCnt(Integer bno) throws Exception;
```

해당 코드는 BoardDAO에서 선언해주는 소스 코드이다.

```
public void updateViewCnt(Integer bno) throws Exception{
    session.update(namespace + ".updateViewCnt", bno);
}
```

다음 코드는 BoardDAOImpl에 선언된 코드로 BoardDAO에서 선언한 함수에 대한 정의를 해준 것이다. Bno의 정보를 바탕으로 작동하는 것을 확인할 수 있다. 이제 실질적으로 read페이지를 읽을 때 마다 Count가 증가되게 해야 하므로 함수를 call을 할 수 있도록 코드를 바꾸어 줘야한다.

```

public BoardVO read(Integer bno) throws Exception {
    // TODO Auto-generated method stub
    dao.updateViewCnt(bno);
    return dao.read(bno);
}

```

ServiceImpl의 소스 코드이며 updateViewCnt의 함수가 들어가 있는 것을 확인할 수 있다.

```

@RequestMapping(value = "/read", method = RequestMethod.GET)
public String read(@RequestParam("bno") int bno, Model model) throws Exception {
    logger.info("read page get...");
    model.addAttribute(service.read(bno));
    return "board/read";
}

```

이는 다음과 같이 read page가 읽힐 때 마다 read함수가 실행되면서 ViewCnt가 1씩 증가하는 효과를 기대할 수 있다. 결과를 확인하면 다음과 같이 볼 수 있다.

2015722025

Board List Page					Register New Board
#번호	제목	작성자	수정일	조회수	
99	99번째 글 제목 입니다.	2015722099	2018-11-14 00:32	4	
98	98번째 글 제목 입니다.	2015722098	2018-11-14 00:32	0	
97	97번째 글 제목 입니다.	2015722097	2018-11-14 00:32	0	
96	96번째 글 제목 입니다.	2015722096	2018-11-14 00:32	0	
95	95번째 글 제목 입니다.	2015722095	2018-11-14 00:32	0	

99번째 글은 4개의 Count가 올라가 있다. 아래 글을 순차적으로 조회하면 다음과 같은 결과가 나온다.

2015722025

Board List Page					Register New Board
#번호	제목	작성자	수정일	조회수	
99	99번째 글 제목 입니다.	2015722099	2018-11-14 00:32	4	
98	98번째 글 제목 입니다.	2015722098	2018-11-14 00:32	3	
97	97번째 글 제목 입니다.	2015722097	2018-11-14 00:32	2	
96	96번째 글 제목 입니다.	2015722096	2018-11-14 00:32	1	
95	95번째 글 제목 입니다.	2015722095	2018-11-14 00:32	0	

정상적으로 조회수가 올라간 것을 확인할 수 있으며 이 데이터는 DB에 저장되며 페이지를 다시 커도 정보가 유지되는 것을 확인할 수 있다.

(2) Paging

다음으로 설명할 추가기능은 paging이다. paging이란 일정 개발자가 정해 놓은 수의 게시글이 넘어가면 페이지를 넘길 수 있도록 설정해주어 깔끔하게 정보를 한눈에 볼 수 있도록 하는 기능을 말한다. 본인이 구현한 Paging의 구현단계로는 2단계로 나뉘는데 단계는 다음과 같다.

첫번째. URI 문자열을 조절하여 원하는 페이지의 데이터가 출력되도록 하기

두번째. List 하단에 페이지 번호를 출력하며, 번호를 클릭하여 해당 페이지로 이동

Paging처리에 대한 원칙과 추가해야 할 구현이 더 있지만 기본적인 Paging을 위한 단계만 거치게 되었다.

먼저 작성하게 될 코드는 Criteria클래스를 작성한다. 이유는 Paging처리를 하는데 조금 더 효율적으로 구현하기 위한 클래스 생성이다. Criteria클래스는 Paging처리르 하는데 기준이 되는 변수들을 하나의 객체로 처리할 수 있도록 도와주는 클래스이다. Criteria클래스는 다음과 같이 작성하도록 한다.

```

1 package com.kwce.domain;
2
3 public class Criteria {
4
5     private int page;
6     private int perPageNum;
7
8     public Criteria() {
9
10        this.page = 1;
11        this.perPageNum = 5;
12    }
13
14    public void setPage(int page) {
15
16
17
18        if (page <= 0) {
19            this.page = 1;
20            return;
21        }
22
23        this.page = page;
24    }
25
26    public void setPerPageNum(int perPageNum) {
27
28
29        if (perPageNum <=0 || perPageNum > 100) {
30            this.perPageNum = 5;
31            return;
32        }
33
34        this.perPageNum = perPageNum;
35    }
36
37    public int getPage() {
38
39        return page;
40    }
41
42    public int getPageStart() {
43
44        return (this.page - 1) * perPageNum;
45    }
46
47    public int getPerPageNum() {
48
49        return this.perPageNum;
50    }
51
52    @Override
53    public String toString() {
54        return "Criteria{" +
55            "page=" + page +
56            ", perPageNum=" + perPageNum +
57            '}';
58    }
59 }

```

변수로 선언된 page는 현재 페이지를 나타내 주며, perpage는 나타내고 싶은 게시글의 개수를 의미한다. 이 함수에서 perpage의 정의를 바꿔주면 한 페이지에서 나타나는 게시글의 개수가 결정된다.

다음은 Criteria타입의 변수를 가지는 추상 메소드를 추가시켜준다.

```
public List<BoardVO>listCriteria(Criteria criteria)throws Exception;
```

정의된 함수를 Impl에서 다음과 같이 재정의 시켜준다.

```
@Override
public List<BoardVO> listCriteria(Criteria criteria)throws Exception{
    return session.selectList(namespace + ".listCriteria",criteria);
}
```

그 후 BoardMapper.xml에 select쿼리를 작성한다.

```
<select id="listCriteria" resultType="com.kwce.domain.BoardVO">
<![CDATA[
SELECT
    bno
    , title
    , content
    , writer
    , regdate
    , viewcnt
FROM tbl_board
WHERE bno > 0
ORDER BY bno DESC, regdate DESC
LIMIT #{pageStart},#{perPageNum}
]]>
</select>
```

BoardService에 추상 메소드를 선언하고 Impl에 재정의를 해준다.

```
public List<BoardVO>listCriteria(Criteria criteria)throws Exception;
```

```
@Override
public List<BoardVO> listCriteria(Criteria criteria)throws Exception{
    return dao.listCriteria(criteria);
}
```

이렇게 paging구현에 대한 준비가 완료되면 Controller와 View를 구현하기전에 Paging처리를 위한 계산 클래스를 작성해주어야 한다.

다음은 하단의 페이지 번호 출력을 도와줄 PageMaker를 아래와 같이 작성한다.

```
1 package com.kwce.domain;
2
3 import org.springframework.web.util.UriComponents;
4
5 public class PageMaker {
6
7     private int totalCount;
8     private int startPage;
9     private int endPage;
10    private boolean prev;
11    private boolean next;
12    private int displayNum = 10;
13    private Criteria criteria;
14
15    public void setTotalCount(int totalCount) {
16        this.totalCount = totalCount;
17        calcData();
18    }
19
20    private void calcData() {
21
22        endPage = (int) (Math.ceil(criteria.getPage() / (double) displayNum) * displayNum);
23
24        startPage = (endPage - displayNum) + 1;
25
26        int tempEndPage = (int) (Math.ceil(totalCount / (double) criteria.getPerPageNum()));
27
28        if (endPage > tempEndPage) {
29            endPage = tempEndPage;
30        }
31
32        prev = startPage == 1 ? false : true;
33
34        next = endPage * criteria.getPerPageNum() >= totalCount ? false : true;
35    }
36
37    public String makeQuery(int page) {
38
39        UriComponents uriComponents = UriComponentsBuilder.newInstance()
40            .queryParams("page", page)
41            .queryParams("perPageNum", criteria.getPerPageNum())
42            .build();
43
44        return uriComponents.toUriString();
45    }
46
47    public String makeSearch(int page) {
48
49        UriComponents uriComponents = UriComponentsBuilder.newInstance()
50            .queryParams("page", page)
51            .queryParams("perPageNum", criteria.getPerPageNum())
52            .queryParams("searchType", ((SearchCriteria) criteria).getSearchType())
53            .queryParams("keyword", encoding(((SearchCriteria) criteria).getKeyword()))
54            .build();
55
56        return uriComponents.toUriString();
57    }
58
59    private String encoding(String keyword) {
60
61        if (keyword == null || keyword.trim().length() == 0) {
62            return "";
63        }
64
65        try {
66            return URLEncoder.encode(keyword, "UTF-8");
67        } catch (UnsupportedEncodingException e) {
68            return "";
69        }
70    }
71}
```



```

80
81 public int getTotalCount() {
82     return totalCount;
83 }
84
85 public int getStartPage() {
86     return startPage;
87 }
88
89 public void setStartPage(int startPage) {
90     this.startPage = startPage;
91 }
92
93 public int getEndPage() {
94     return endPage;
95 }
96
97 public void setEndPage(int endPage) {
98     this.endPage = endPage;
99 }
100
101 public boolean isPrev() {
102     return prev;
103 }
104
105 public void setPrev(boolean prev) {
106     this.prev = prev;
107 }
108
109 public boolean isNext() {
110     return next;
111 }
112
113 public void setNext(boolean next) {
114     this.next = next;
115 }
116
117
118 public int getDisplayNum() {
119     return displayNum;
120 }
121
122 public void setDisplayNum(int displayNum) {
123     this.displayNum = displayNum;
124 }
125
126 public Criteria getCriteria() {
127     return criteria;
128 }
129
130 public void setCriteria(Criteria criteria) {
131     this.criteria = criteria;
132 }
133
134 @Override
135 public String toString() {
136     return "PageMaker{" +
137         "totalCount=" + totalCount +
138         ", startPage=" + startPage +
139         ", endPage=" + endPage +
140         ", prev=" + prev +
141         ", next=" + next +
142         ", displayNum=" + displayNum +
143         ", criteria=" + criteria +
144         '}';
145 }

```

위 PageMaker는 Paging처리를 위한 계산관련 클래스이다. Page는 현재 페이지의 번호이며 perPageNum은 게시글의 개수, totalCount는 전체 게시글의 개수, startPage는 시작페이지 번호, endpage는 끝 페이지 번호, prev는 이전 링크 next는 다음 링크를 뜻한다. 다음 추가하게 될 함수는 전체 게시글의 개수를 구할 수 있는 함수이다. BoardDAO에 다음 함수를 추가한다.

```
public int countBoard(Criteria criteria) throws Exception;
```

마찬가지로 Impl에서 재정의의를 해준다.

```
@Override
public int countBoard(Criteria criteria) throws Exception{
    return session.selectOne(namespace + ".countBoard",criteria);
}
```

다음은 Mapper에 대한 수정이다. xml파일에 아래 코드를 추가적으로 넣어준다.

```
<select id="countBoard" resultType="int">
    <![CDATA[
        SELECT COUNT(bno)
        FROM tbl_board
        WHERE bno > 0
    ]]>
</select>
```

그 후 Service에도 마찬가지로 추상 메소드 추가와 재정의의를 함께 해준다.

```
public int countBoard(Criteria criteria) throws Exception;
```

```
@Override
public int countBoard(Criteria criteria) throws Exception{
    return dao.countBoard(criteria);
}
```

이렇게 PageMaker클래스와 몇 가지 함수까지 작성했다면 Controller와 View와 관련된 JSP를 작성해주면 된다.

```

PageMaker pageMaker = new PageMaker();
pageMaker.setCriteria(criteria);
pageMaker.setTotalCount(service.listCount(criteria));
model.addAttribute("list", list);
model.addAttribute("pageMaker", pageMaker);
model.addAttribute("board", service.listCriteria(criteria));

```

우선 Controller로 /list 부분 함수에 해당 함수를 추가적으로 넣어주면 된다. 다음은 JSP부분으로 list.JSP에 아래와 같이 추가해주면 된다.

```

<div class= "panel-footer">
    <div class="text-center">
        <ul class="pagination">
            <c:if test="${pageMaker.prev}">
                <li>
                    <a href="${path}/board/list${pageMaker.makeSearch(pageMaker.startPage - 1)}">&laquo;</a>
                </li>
            </c:if>
            <c:forEach begin="${pageMaker.startPage}" end="${pageMaker.endPage}" var="idx">
                <li <c:out value="${pageMaker.criteria.page == idx? 'class=active:'}" />>
                    <a href="${path}/board/list${pageMaker.makeSearch(idx)}">${idx}</a>
                </li>
            </c:forEach>
            <c:if test="${pageMaker.next && pageMaker.endPage > 0}">
                <li>
                    <a href="${path}/board/list${pageMaker.makeSearch(pageMaker.endPage + 1)}">&raquo;</a>
                </li>
            </c:if>
        </ul>
    </div>
</div>

```

이렇게 Controller에 대한 수정과 JSP에 대한 list수정이 끝났다면 URL을 통해 결과를 확인할 수 있다.

2015722025

Board List Page					Register New Board
#번호	제목	작성자	수정일	조회수	
99	99번째 글 제목 입니다.	2015722099	2018-11-14 00:32	7	
98	98번째 글 제목 입니다.	2015722098	2018-11-14 00:32	5	
97	97번째 글 제목 입니다.	2015722097	2018-11-14 00:32	2	
96	96번째 글 제목 입니다.	2015722096	2018-11-14 00:32	1	
95	95번째 글 제목 입니다.	2015722095	2018-11-14 00:32	1	

1 2 3 4 5 6 7 8 9 10 »

(3) 검색

다음은 검색에 대한 추가구현 설명입니다. 검색 추가구현은 제목, 내용, 작성자를 통한 검색을 구현하게 되었습니다.

가장 먼저 검색에 대한 SearchCriteria 클래스를 작성합니다.

```
package com.kwce.domain;

public class SearchCriteria extends Criteria {

    private String searchType;
    private String keyword;
```

다음과 같은 형태로 Criteria를 상속받고 있는 것을 확인할 수 있습니다.

다음으로는 list.JSP에 검색창을 먼저 만든다. List.JSP에 아래 코드를 추가적으로 작성해준다.

```
<div class="box-footer">
    <br/>
    <div class="form-group col-sm-2">
        <select class="form-control" name="searchType" id="searchType">
            <option value="n" <:out value="${criteria.searchType == null ? 'selected' : ''}"/>선택 :</option>
            <option value="t" <:out value="${criteria.searchType eq 't' ? 'selected' : ''}"/>제목</option>
            <option value="c" <:out value="${criteria.searchType eq 'c' ? 'selected' : ''}"/>내용</option>
            <option value="w" <:out value="${criteria.searchType eq 'w' ? 'selected' : ''}"/>작성자</option>
            <option value="tc" <:out value="${criteria.searchType eq 'tc' ? 'selected' : ''}"/>제목+내용</option>
            <option value="cw" <:out value="${criteria.searchType eq 'cw' ? 'selected' : ''}"/>내용+작성자</option>
            <option value="tcw" <:out value="${criteria.searchType eq 'tcw' ? 'selected' : ''}"/>제목+내용+작성자</option>
        </select>
    </div>
    <div class="form-group col-sm-10">
        <div class="input-group">
            <input type="text" class="form-control" name="keyword" id="keywordInput" value="${criteria.keyword}" placeholder="검색어">
            <span class="input-group-btn">
                <button type="button" class="btn btn-primary btn-flat" id="searchBtn">
                    <i class="fa fa-search"></i> 검색
                </button>
            </span>
        </div>
    </div>
</div>
```

t, c, w, n이 의미하는 것은 제목, 내용, 작성자, 검색조건 없음을 뜻하는 keyword이다. 그리고 버튼에 대한 이벤트 설정을 아래와 같이 해준다.

```
$(document).ready(function () {

    $("#searchBtn").on("click", function (event) {
        self.location = "list${pageMaker.makeQuery(1)}"
        + "&searchType=" + $("#select option:selected").val()
        + "&keyword=" + encodeURIComponent($("#keywordInput").val());
    });
});
```

검색 버튼 클릭 시 발생하게 되는 이벤트를 설정한 것이다.

해당 작업이 끝났다면 미리 만든 PageMaker에 makeSearch함수와 encoding함수를 추가해준다. makeSearch함수는 검색조건과 키워드를 처리해주며, encoding함수는 인코딩 처리를 위한 함수이다.

```
public String makeSearch(int page) {
    UriComponents uriComponents = UriComponentsBuilder.newInstance()
        .queryParams("page", page)
        .queryParams("perPageNum", criteria.getPerPageNum())
        .queryParams("searchType", ((SearchCriteria) criteria).getSearchType())
        .queryParams("keyword", encoding(((SearchCriteria) criteria).getKeyword()))
        .build();

    return uriComponents.toUriString();
}

private String encoding(String keyword) {
    if (keyword == null || keyword.trim().length() == 0) {
        return "";
    }

    try {
        return URLEncoder.encode(keyword, "UTF-8");
    } catch (UnsupportedEncodingException e) {
        return "";
    }
}
```

해당 작업까지 완료되었다면 검색된 목록과 개수를 return 해주는 함수를 선언해준다.

```
public List<BoardVO> list(SearchCriteria criteria) throws Exception;

public int listCount(SearchCriteria criteria) throws Exception;
```

그 후 재정의를 해준다.

```
@Override
public int listCount(SearchCriteria criteria) throws Exception {
    return session.selectOne(namespace + ".listSearchCount", criteria);
}

@Override
public List<BoardVO> list(SearchCriteria criteria) throws Exception {
    return session.selectList(namespace + ".listPagingSearch", criteria);
}
```

다음으로 Mapper.xml에 대한 수정을 해주어야 한다. 이유는 상황에 맞게 검색을 처리하도록 동작해야 하기 때문에 코드를 추가적으로 작성하는 것이다.

```

<select id="countBoard" resultType="int">
    <![CDATA[
        SELECT COUNT(bno)
        FROM tbl_board
        WHERE bno > 0
    ]]>
</select>
<select id="listSearchCount" resultType="int">
    <![CDATA[
        SELECT COUNT(bno)
        FROM tbl_board
        WHERE bno > 0
    ]]>
    <include refid="search"/>
</select>

<sql id="search">
    <if test="searchType != null">
        <if test="searchType == 't'.toString()">
            AND title LIKE CONCAT('%', #{keyword}, '%')
        </if>
        <if test="searchType == 'c'.toString()">
            AND content LIKE CONCAT('%', #{keyword}, '%')
        </if>
        <if test="searchType == 'w'.toString()">
            AND writer LIKE CONCAT('%', #{keyword}, '%')
        </if>
        <if test="searchType == 'tc'.toString()">
            AND (
                title LIKE CONCAT('%', #{keyword}, '%')
                OR content LIKE CONCAT('%', #{keyword}, '%')
            )
        </if>
        <if test="searchType == 'cw'.toString()">
            AND (
                content LIKE CONCAT('%', #{keyword}, '%')
                OR writer LIKE CONCAT('%', #{keyword}, '%')
            )
        </if>
        <if test="searchType == 'tcw'.toString()">
            AND (
                title LIKE CONCAT('%', #{keyword}, '%')
                OR content LIKE CONCAT('%', #{keyword}, '%')
                OR writer LIKE CONCAT('%', #{keyword}, '%')
            )
        </if>
    </if>
</sql>

```

다음으로는 Service에서도 추상 메소드를 정의하고 재정의가 필요하다.

```
public List<BoardVO> list(SearchCriteria criteria) throws Exception;

public int listCount(SearchCriteria criteria) throws Exception;

@Override
public int listCount(SearchCriteria criteria) throws Exception {
    return dao.listCount(criteria);
}
@Override
public List<BoardVO> list(SearchCriteria criteria) throws Exception {
    return dao.list(criteria);
}
```

마지막으로 Controller를 아래와 같이 정의해준다.

```
@RequestMapping(value = "/list")
public String list(@ModelAttribute("criteria") SearchCriteria criteria, Model model) throws Exception {
    logger.info("list() : called...");
    logger.info(criteria.toString());
    List<BoardVO> list = service.list(criteria);

    PageMaker pageMaker = new PageMaker();
    pageMaker.setCriteria(criteria);
    pageMaker.setTotalCount(service.listCount(criteria));

    model.addAttribute("list", list);
    model.addAttribute("pageMaker", pageMaker);

    model.addAttribute("totalCount", service.listCount(criteria));
    return "board/list";
}
```

구현된 게시판에서 결과를 확인한다.

2015722025

Board List Page					Register New Board
#번호	제목	작성자	수정일	조회수	
94	94번째 글 제목 입니다.	2015722094	2018-11-14 00:32	1	
93	93번째 글 제목 입니다.	2015722093	2018-11-14 00:32	1	
92	92번째 글 제목 입니다.	2015722092	2018-11-14 00:32	0	
91	91번째 글 제목 입니다.	2015722091	2018-11-14 00:32	0	
90	90번째 글 제목 입니다.	2015722090	2018-11-14 00:32	0	
1 2 3					
작성자 ▼ 2209					Q 검색

일정 키워드를 바탕으로 해당 게시글을 검색완료한 모습입니다.

(4) 파일 & 이미지 upload

추가 기능으로 파일과 이미지를 업로드할 수 있도록 하였습니다. 한계점으로는 데이터베이스 이용에 익숙하지 않아 게시판 자체에 이미지와 파일을 업로드하지 못하고 다른 폴더로 보낼 수 있는 업로드를 만들게 되었습니다. 이번 프로젝트에서는 과제 제출을 컨셉으로 버튼을 설정하였으며 구현 과정은 다음과 같습니다.

가장 먼저 File upload관련 라이브러리를 xml에 아래와 같이 추가해주어야 합니다.

```
<!-- 이미지 업로드 관련-->
<!-- https://mvnrepository.com/artifact/org.imgscalr/imgscalr-lib--> <dependency>
  <groupId>org.imgscalr</groupId>
  <artifactId>imgscalr-lib</artifactId>
  <version>4.2</version>
</dependency>

<!-- 파일 업로드 관련 -->
<!-- https://mvnrepository.com/artifact/commons-fileupload/commons-fileupload -->
<dependency>
  <groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>1.3.2</version>
</dependency>
```

그 후 사용자가 보게 될 View인 JSP파일에 다음 코드를 추가합니다.

```
<form id="fileUploadForm" action="board/fileUpload" method="post" enctype="multipart/form-data">
  <input type="file" id="fileUpload" class="btn btn-default" name="fileUpload"/>
  <input type="file" id="fileUpload2" class="btn btn-default" name="fileUpload2"/>
  <input type="button" value="과제 제출" onClick="fileSubmit();" />
</form>
```

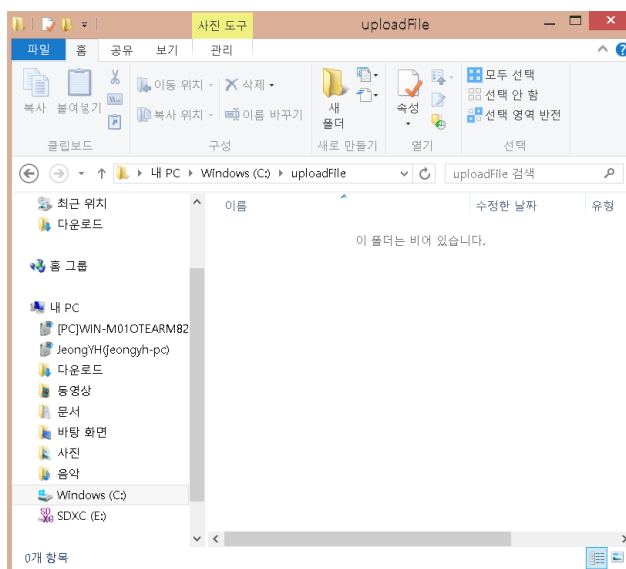

마지막으로 실질적인 버튼 이벤트에 대한 처리를 위하여 BoardController에 업로드와 관련된 함수를 아래와 같이 정의합니다.

```
@RequestMapping(value = "/read")
public void ajaxUpload(MultipartHttpServletRequest multi) {
    //
    String path="C:/uploadFile/";
    File dir = new File(path);
    if(!dir.isDirectory()){
        dir.mkdir();
    }

    Iterator<String> files = multi.getFileNames();
    while(files.hasNext()){
        String uploadFile = files.next();
        MultipartFile mFile = multi.getFile(uploadFile);
        String fileName = mFile.getOriginalFilename();
        try {
            mFile.transferTo(new File(path+fileName));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return;
}
```

해당 함수는 조회 페이지에서 활성화되며, 원래는 글을 올리 때 구현하는 게 맞지만 이유모를 컴파일 오류가 뜨기에 조회 페이지에 넣게 되었습니다. String path는 실제 파일을 업로드할 경로를 뜻하며 페이지에서 이미지나, 파일을 업로드하면 해당 경로로 파일이 올라가게 됩니다.

다음은 파일 업로드에 관련한 결과 화면입니다.



다음 파일은 경로로 설정한 upload파일입니다.

```
String path="C:/uploadFile/";
```

2015722025

Board Read Page

Bno

97

Title

97번째 글 제목 입니다.

Text area

97번째 글 내용입니다.

Writer

2015722097

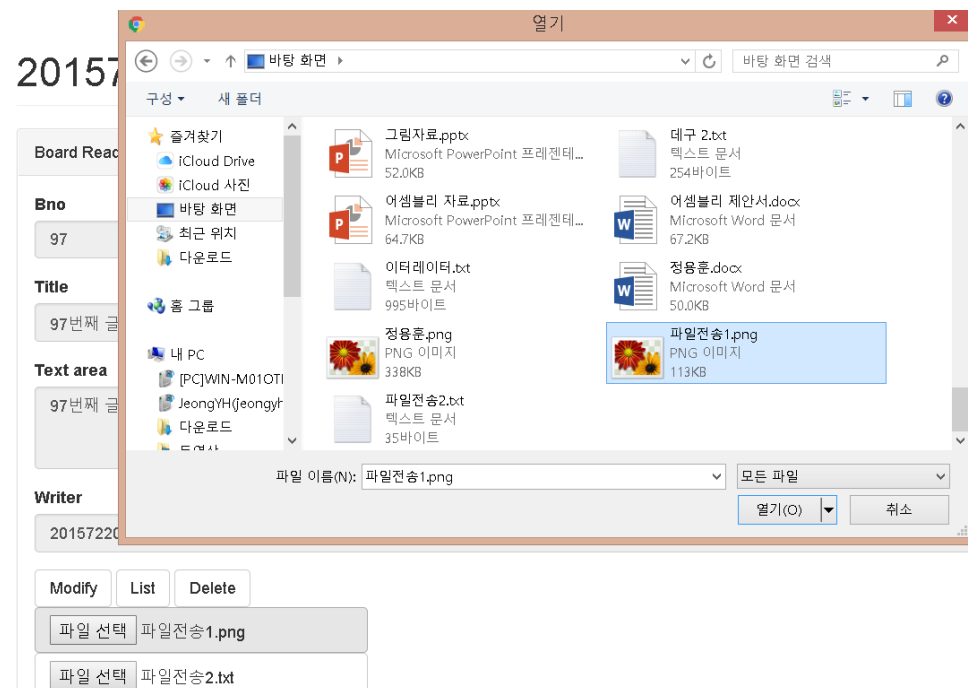
Modify List Delete

파일 선택 선택된 파일 없음

파일 선택 선택된 파일 없음

과제 제출

파일 선택을 통해 upload할 파일을 or 이미지를 선택 후 과제 제출 버튼을 누르게 되면 경로로 설정한 파일에 해당 파일과 이미지가 올라가게 되어있습니다



우선 위 그림과 같이 전송할 파일을 선택합니다.

2015722025

Board Read Page

Bno
97

Title
97번째 글 제목입니다.

Text area
97번째 글 내용입니다.

Writer
2015722097

Modify List Delete

파일 선택 파일전송1.png

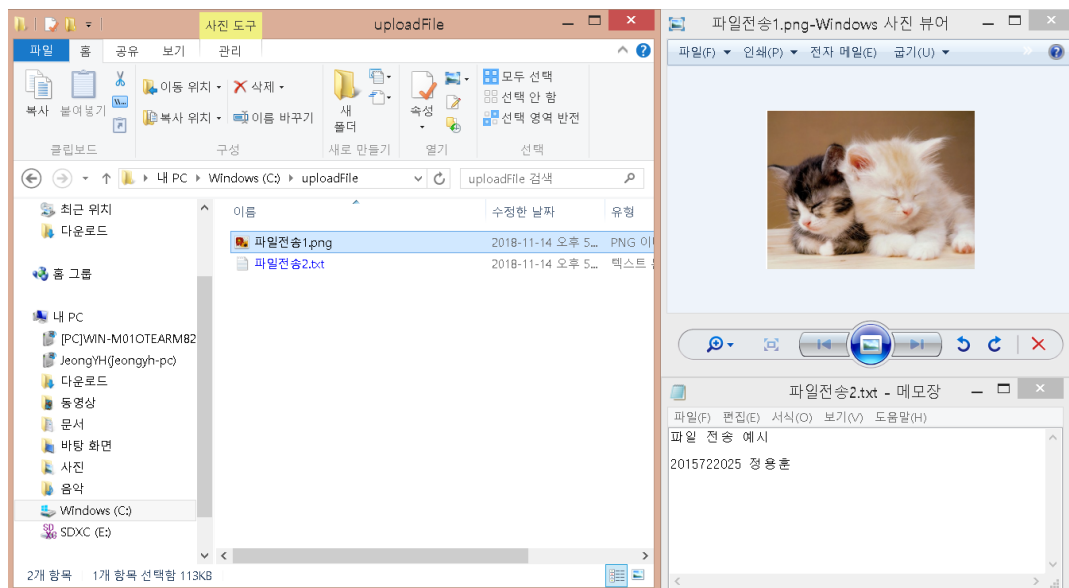
파일 선택 파일전송2.txt

과제 제출

www.localhost:8080 내용:
파일 업로드하였습니다.

확인

파일을 모두 선택하고 과제 제출 버튼을 누르면 다음과 같은 이벤트가 발생하게 됩니다. 그 후 upload 파일을 확인하면 정상적으로 파일이 올라간 것을 확인할 수 있습니다.

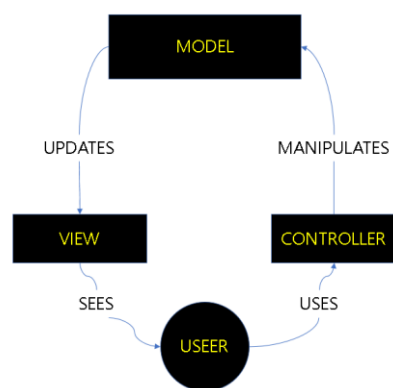


다음과 같이 upload한 파일이 성공적으로 이동한 것을 확인할 수 있습니다. 데이터 베이스를 사용하면 게시판 자체에 올릴 수 있겠지만 해당 부분에 대한 개념이 부족하여 과제 제출 형식으로 기능을 구현하게 되었습니다. 이를 응용하면 실질적으로 우리가 과제 제출할 때 사용하는 페이지도 구현할 수 있다고 생각합니다.

4. 튜토리얼 관련 내용 추가 조사

● MVC 패턴

MVC패턴은 디자인 패턴 중 하나이다. 여기서 디자인 패턴이란 프로그램이나 어떤 특정한 것을 개발하는 중에 발생한 문제점들을 정리해서 상황에 따라 간편하게 적용해서 쓸 수 있는 것을 정리하여 특정한 "규약"을 통해 쉽게 쓸 수 있는 형태로 만든 것을 말한다.

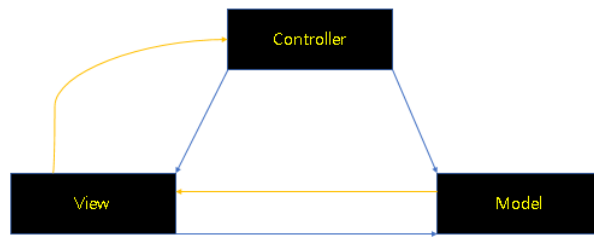


MVC란 약자로 Model-View-Controller를 뜻하며 각각의 명칭과 user의 관계는 옆에 그림과 같이 사용자가 Controller를 조작하면 Controller가 Model을 통하여 데이터를 가져와 데이터를 바탕으로 View를 제어해서 최종적으로 user가 결과를 확인 할 수 있게 되는 것이다. 하지만 실질적인 MVC패턴이 아니라 이해를 돕기 위한 그림이며 실질적인 각 패턴의 관계는 동작의 설명이 끝나고 자세하게 설명하겠다. 아래 설명은 각각 약자의 기본적인 동작을 설명한 것이다.

첫번째로 컨트롤러는 모델에 명령을 보내며 모델의 상태를 변경할 수 있는 것을 뜻한다. (ex 워드 프로세서에서 문서 편집) 또한 컨트롤러가 관련된 뷰에 명령을 보내면 모델의 표시 방법을 바꿀 수 있다. (ex 문서 스크롤)

두번째로는 모델이다. 모델은 모델의 상태가 변화가 생기면 컨트롤러와 뷰에 신호를 보내며, 이와 같은 신호를 통해 뷰에서는 최신의 결과를 계속 보여줄 수 있으며, 컨트롤러는 모델의 변화에 따라 적용가능한 명령을 추가, 제거, 수정할 수 있다. MVC를 다르게 구현하면 뷰나 컨트롤러에서 직접 모델의 상태 정보를 읽어오는 경우도 있다.

세번째로 뷰는 단순히 사용자가 보는 결과물을 생성하기 위하여 모델로부터 정보를 읽어온다.



해당 그림이 실질적인 MVC패턴들의 관계를 나타낸 것이다. 이유는 Controller가 View에도 영향을 미치는 부분이 있어야 하기 때문이다. 이러한 MVC패턴을 사용하는 이유는 일의 효율을 높이기 위해서이다. 패턴을 통하여 역할을 3가지로 나누어 개발을 한다면 유지보수성, 개발프로그램의 확장성, 유연성, 중복코딩이라는 문제점을 막을 수 있기 때문에 MVC패턴을 사용하게 된다.

● Maven

Maven의 사전적 의미로는 이디시어로 “지식의 축적”을 의미한다. 대다수의 maven 사용자들은 maven을 소스 코드로부터 배포 가능한 산출물을 빌드하는 ‘빌드 툴’이라고도 한다. 아래 설명은 Maven에 대한 간략한 설명이다.

- pom.xml 파일을 이용해서 관련된 jar 파일(라이브러리 파일, 모듈)을 다운하고 관리
- 프로젝트를 관리하기 위한 툴(라이브러리만이 아니라, 컴파일, 테스트, 패키지 등 다양한 기능을 제공)
- 기본적으로 사용하는 이클립스에 Maven project가 포함되어 있다.

쉽게 말해서 개발자가 개발을 진행하는데 있어 단순히 자신의 코드만 사용하여 프로그램 개발하는 것이 아니라 많은 종류의 라이브러리들을 활용하여 개발을 하게 된다. 이때 사용되는 라이브러리들의 개수는 프로그램의 크기에 따라 수십개가 훌쩍 넘는 경우가 생기기도 하는데 이렇게 개수가 많아진 라이브러리는 개발자가 관리하는데 어려움이 있을 수 있다. Maven은 이런 문제를 해결하기 위한 도구이며 Maven은 내가 사용할 라이브러리만이 아니라 해당 라이브러리가 작동하는데 필요한 다른 라이브러리들까지 관리해주며 네트워크를 통하여 자동으로 다운받아 주는 편리한 도구이다. 또한 프로젝트 전체적인 라이프 사이클을 관리하는 도구이며, 많은 편리함과 장점이 있어 많이 사용되고 있는 도구이다.

● GET방식과 POST방식

GET과 POST의 차이점을 설명하기전 공통점으로는 두 방법 모두 client에서 server로 데이터를 전송하는 방식을 뜻한다. 두 방식의 차이점은 다음과 같다.

1) *GET

-URL 뒤에 파라미터와 파라미터 값이 쓰여 전송되는 방식

-URL에 이어 붙이기 때문에 길이에 제한이 있어 많은 데이터를 보내기 어렵다.

2) *POST

-form 태그를 이용해 submit하는 방식

-GET 방식에 비해 URL이 짧기 때문에 상대적으로 많은 데이터를 보낼 수 있다.

다음 이미지는 튜토리얼을 통해 실습했던 코드를 조금 변형하여 나타낸 POST와 GET의 전송방법 차이를 나타낸 것이다. (ControllerExample.java)

```
@RequestMapping(value = "/urlF", method = RequestMethod.GET)
public String get(Model model) {
    logger.info("urlF get called....");
    return "GetandPost";
}

@RequestMapping(value = "/urlF", method = RequestMethod.POST)
public String post(String StudentID, String name) {
    logger.info("urlF post called....");
    System.out.println(StudentID);
    System.out.println(name);
    return "redirect:/";
}
```

<<원래 코드

```
@RequestMapping(value = "/urlF", method = RequestMethod.GET)
public String get(Model model) {
    logger.info("urlF get called....");
    return "GetandPost";
}

@RequestMapping(value = "/urlF", method = RequestMethod.POST)
public String post(String StudentID, String name) {
    logger.info("urlF post called....");
    System.out.println(StudentID);
    System.out.println(name);
    return "GetandPost";
}
```

<<변형 코드

전송 방식의 대해서만 알아보기 위하여 GET과 POST방식의 함수를 다른 View로 보내지 않고 전송하는 화면에서 머물게 해주기 위한 return값 변경

(GetandPost.jsp)

```
<form method="POST" action=../urlF>
<table>
<tr>
<td><label>학번</label></td>
<td><input type="text" name="StudentID"></td>
</tr>
```

<<원래 코드

```
<form method="GET" action=../urlF>
<table>
<tr>
<td><label>학번</label></td>
<td><input type="text" name="StudentID"></td>
</tr>
```

<<비교를 위한 변형

전송 방식의 대한 정의로 POST와 GET의 차이를 알기 위하여 POST로 전송 후 결과를 확인하고 GET으로 변경 후 데이터를 전송하여 차이를 확인한다.

POST

← → ↻ localhost:8080/urlF

GET and POST TEST

학번
이름

전송 전

← → ↻ localhost:8080/urlF

GET and POST TEST

학번
이름

전송 후

```
INFO : com.kwce.controller.ControllerExample - urlF post called....  
2015722025  
정용훈
```

데이터를 확인할 수 없기 때문에 Post방식에서는 콘솔에 이동한 데이터가 Print된 것을 확인할 수 있다. 앞서 설명한 것과 마찬가지로 주소에 변화가 없고 데이터가 숨겨진 채 전송된다.

GET

← → ↻ localhost:8080/urlF

GET and POST TEST

학번
이름

전송 전

← → ↻ localhost:8080/urlF?StudentID=2015722025&name=정용훈

GET and POST TEST

학번
이름

전송 후

GET방식은 데이터가 url뒤에 붙어서 함께 전송되는 것을 확인할 수 있다.

- Junit 조사

Junit이란 spring상에서 개발자가 원하는 부분을 독립적인 단위로 테스트할 수 있도록 지원해주는 프레임 워크이다. 다른 요소에 영향을 주지 않으면서 소스코드의 특정 모듈들이 각각 제대로 구동하는지 확인할 수 있으며, main메소드를 가진 클래스들을 따로 여러 개 생성하지 않아도 되게 해준다. 프로젝트에서는 BoardDAOtest 파일을 만들어 사용해보았습니다.

- Front-End

1) JSTL

JSTL은 'Java Server Pages Standard Tag Library'의 약자 이다. Library는 여러 프로그램이 공통으로 사용하는 코드를 모아 놓은 코드의 집합을 말하며, JSTL 또한 공통으로 사용되는 코드의 집합이다. 하지만 우리가 보통 프로그래밍을 할 때 사용하는 Library와는 달리 JSP페이지 안에서 사용할 수 있는 커스텀과 함수를 제공한다. 해당 라이브러리와 기능, 접두어는 아래 표를 확인하면 된다.

라이브러리	기능	접두어
core	언어의 반복문, 제어, 변수 등의 기능 제공	C
fmt	숫자, 날짜, 시간의 포매팅 기능	fmt
sql	데이터베이스의 데이터를 입력/수정/삭제조회 기능 제공	sql
xml	Xml 문서를 처리할 때 필요한 기능 제공	x
Functions	문자열을 처리하는 함수 제공	fn

해당 라이브러리를 쓰기위해선 taglib prefix와 uri를 설정해야 합니다. 다음은 각 라이브러리에 대한 설정 내용입니다.

- (1) Core (prefix: c), URI=<http://java.sun.com/jsp/jstl/core>
- (2) Formatting (prefix: fmt), URI= <http://java.sun.com/jsp/jstl/fmt>
- (3) Database (prefix: sql), URI=<http://java.sun.com/jsp/jstl/sql>
- (4) XML (prefix: x), URI=<http://java.sun.com/jsp/jstl/xml>
- (5) Function (prefix: fn), URI= <http://java.sun.com/jsp/jstl/functions>

JSTL의 **Core** 예시는 아래와 같다.

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4  <%@page import="java.util.ArrayList" %>
5  <!DOCTYPE html">
6  <html>
7  <head>
8  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
9  <title>Insert title here</title>
10 </head>
11 <body>
12 <%
13 ArrayList<Integer> array=new ArrayList<Integer>();
14 array.add(1); array.add(2); array.add(3); array.add(4); array.add(5);
15 array.add(6); array.add(7); array.add(8); array.add(9); array.add(10);
16 request.setAttribute("array", array);
17 %>
18 <h3>JSTL Core TEST</h3>
19 <table border="1">
20 <tr>
21 <td>201572202</td>
22 <td>정용훈</td>
23 </tr>
24 </table>
25
26 <c:out value="Core Test 입니다!"></c:out>
27 <br>
28
29 <c:forEach var="i" items="{array}">
30 <c:if test="{i%2==0}">
31 <c:out value="{i}=짝수"/>
32 </c:if>
33 <c:if test="{i%2==1}">
34 <c:out value="{i}=홀수"/>
35 </c:if>
36
37 <br>
38 </c:forEach>
39
40
41 </body>
42 </html>
```

해당 코드는 JSTL의 코어 라이브러리이며, 홀수 짝수를 forEach를 통해 반복적으로 찾을 수 있도록 작성한 것이다. 쓰인 JSTL로는 c: out, c: if, c: forEach등이 있다. 결과 화면은 다음과 같다.

JSTL Core TEST

201572202	정용훈
-----------	-----

Core Test 입니다!

1=홀수
2=짝수
3=홀수
4=짝수
5=홀수
6=짝수
7=홀수
8=짝수
9=홀수
10=짝수

Core는 언어의 반복문과 제어, 변수 등의 기능을 제공한다. 우리가 평소 배웠던 언어와 유사하기 때문에 이해하기 수월하다.

다음은 JSTL의 **fmt** 라이브러리의 예시이다.

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4 <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
5 <%@page import="java.util.ArrayList" %>
6 <!DOCTYPE html">
7 <html>
8 <head>
9 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
10 <title>Insert title here</title>
11 </head>
12 <body>
13 <h3>JSTL Core TEST</h3>
14 <table border="1">
15 <tr>
16 <td>201572202</td>
17 <td>정용훈</td>
18 </tr>
19 </table>
20
21 <c:out value="Core Test 입니다!"></c:out>
22 <br><br>
23
24 <c:set var="now" value="<%=new java.util.Date() %>"/>
25 <c:out value="{now }"/><br>
26 data : <fmt:formatDate value="{now }" type="date"/><br>
27 time : <fmt:formatDate value="{now }" type="time"/><br>
28 both : <fmt:formatDate value="{now }" type="both"/><br>
29 <br>
30
31
32 </body>
33 </html>
```

fmt라이브러리를 사용하기 위해 추가한 코드이다. 다음은 현재 날짜와 시간을 나타내는 코드로써 해당 결과는 아래 그림과 같이 나온다.

← → ↻ ⓘ localhost:8080/JSTL

JSTL Core TEST

201572202	정용훈
-----------	-----

Core Test 입니다!

Wed Nov 14 20:26:42 KST 2018

data : 2018. 11. 14.

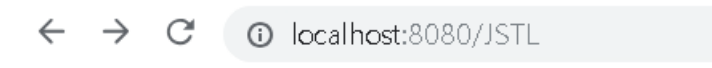
time : 오후 8:26:42

both : 2018. 11. 14. 오후 8:26:42

다음은 JSTL의 **xml** 라이브러리의 예시이다.

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4  <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
5  <%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
6  <%@ taglib prefix = "x" uri = "http://java.sun.com/jsp/jstl/xml" %>
7
8  <!DOCTYPE html">
9  <html>
10 <head>
11 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
12 <title>Insert title here</title>
13 </head>
14 <body>
15 <h3>JSTL Core TEST</h3>
16 <table border="1">
17 <tr>
18 <td>201572202</td>
19 <td>정용훈</td>
20 </tr>
21 </table>
22
23 <c:out value="Core Test 입니다!"></c:out>
24 <br><br>
25
26 <h3>Books Info:</h3>
27
28 <c:set var = "xmltext">
29 <books>
30 <book>
31 <name>Padam History</name>
32 <author>ZARA</author>
33 <price>100</price>
34 </book>
35
36 <book>
37 <name>Great Mistry</name>
38 <author>NUHA</author>
39 <price>2000</price>
40 </book>
41 </books>
42 </c:set>
43
44 <x:parse xml = "${xmltext}" var = "output"/>
45 <b>The title of the first book is</b>:
46 <x:out select = "$output/books/book[1]/name" />
47 <br>
48
49 <b>The price of the second book</b>:
50 <x:out select = "$output/books/book[2]/price" />
51
52 </body>
53 </html>
```

Xml 라이브러리는 문서를 처리할 때 필요한 기능을 제공해주는 라이브러리로서 사용된 x: out은 Xpath의 내용을 출력하는 함수이다. 코드를 위 이미지와 같이 작성 후 결과를 출력하면 아래 이미지와 같이 결과가 나온다.



JSTL Core TEST

201572202	정용훈
-----------	-----

Core Test 입니다!

Books Info:

The title of the first book is: Padam History

The price of the second book: 2000

마지막으로 JSTL의 **Functions** 라이브러리의 예시이다.

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4 <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
5 <%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
6 <%@ taglib prefix = "x" uri = "http://java.sun.com/jsp/jstl/xml" %>
7 <%@ taglib prefix = "fn" uri = "http://java.sun.com/jsp/jstl/functions"%>
8
9 <!DOCTYPE html">
10 <html>
11 <head>
12 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
13 <title>Insert title here</title>
14 </head>
15 <body>
16 <h3>JSTL Core TEST</h3>
17 <table border="1">
18 <tr>
19 <td>201572202</td>
20 <td>정용훈</td>
21 </tr>
22 </table>
23
24 <c:out value="Core Test 입니다!"></c:out>
25 <br><br>
26
27 <c:set var = "string1" value = "This is first String."/>
28 <c:set var = "string2" value = "This is second String." />
29 <p>Length of String (1) : ${fn:length(string1)}</p>
30 <p>Length of String (2) : ${fn:length(string2)}</p>
31
32 </body>
33 </html>
```

Functions라이브러리는 문자열을 처리하는 함수를 제공한다. 이번 Test에서는 문자열의 길이를 반환하는 함수를 호출함으로써 라이브러리에 대한 예시를 알아보았다. 위 코드처럼 수정 후 결과를 확인하면 아래와 같은 결과가 나온다.

← → ↻ ⓘ localhost:8080/JSTL

JSTL Core TEST

201572202	정용훈
-----------	-----

Core Test 입니다!

Length of String (1) : 21

Length of String (2) : 22

2) HTML, JQuery, JavaScript, CSS

HTML

HTML이란 프로그래밍 언어의 한 종류로써 웹을 통해 볼 수 있는 문서를 다루는데 사용하는 언어이다. 특히 인터넷에서 웹을 통해 접근되는 대부분의 web 페이지들은 HTML로 작성된다. HTML은 문서의 글자크기, 글자 색, 글자 모양 등...을 정의하는 명령어이며 홈페이지를 작성하는데 주로 쓰이게 된다. 다음 작성하게 된 예시는 웹에서 코드를 결과 값으로 변환시켜주는 사이트에서 작성한 코드이며 결과 화면이다.

```
<!DOCTYPE html>
<html>
<body>
2015722025 정용훈

<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
<h4>This is heading 4</h4>
<h5>This is heading 5</h5>
<h6>This is heading 6</h6>
<h1>This is heading 7</h1>
<h5>This is heading 8</h5>

</body>
</html>
```

해당 예제 코드는 글자의 크기를 변환시켜주는 코드로써 크기는 h1~h6까지의 크기를 가진다. 해당 코드를 웹에서 변환시켜주면 아래와 같은 결과 화면을 볼 수 있다.

This is heading 1

This is heading 2

This is heading 3

This is heading 4

This is heading 5

This is heading 6

This is heading 7

This is heading 8

결과 화면과 같이 hx의 값에 따라 글자의 크기가 비교되는 것을 확인할 수 있다. html에서는 글자의 크기 만이 아니라 글자 색, 모양 등 개발자가 원하는 대로 디자인을 할 수 있다.

```
1 <!DOCTYPE html>
2 <html lang="ko">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>HTML Paragraph</title>
7 </head>
8
9 <body>
10
11   <p>안녕하세요!</p>
12   <hr>
13   <p>2015722025</p>
14   <hr>
15   <p>정용훈 입니다~</p>
16
17 </body>
18
19 </html>
```

다음은 내용과 수평구분선 출력을 위한 코드입니다. 해당 결과는 다음과 같이 나옵니다!

안녕하세요!

2015722025

정용훈 입니다~

```
1 <!DOCTYPE html>
2 <html lang="ko">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>HTML Paragraph</title>
7 </head>
8
9 <body>
10
11   <pre>
12 2015722025
13 정용훈 입니다~
14 pre 는 현재 제가 쓴 text데로 글이 출력
15 됩
16 니
17 다!
18   </pre>
19
20 </body>
21
22 </html>
```

다음 예시는 pre로 code text에 타이핑한 모습데로 결과화면에 출력되는 것을 확인할 수 있다.

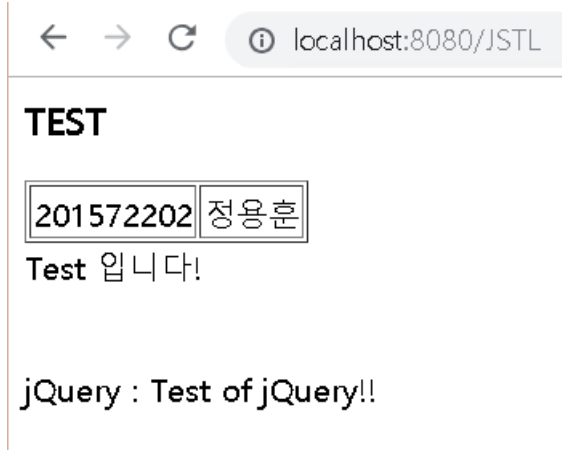
```
2015722025
정용훈 입니다~
pre 는 현재 제가 쓴 text데로 글이 출력
됩
니
다!
```

JQuery

JQuery란 오픈소스 기반의 자바스크립트 라이브러리입니다. JQuery는 웹 사이트에 자바스크립트를 더욱 쉽게 활용할 수 있도록 도와주는 것입니다. 또한 JQuery를 사용하게 되면 짧고 단순한 코드로 웹페이지에 다양한 효과와 이벤트를 적용할 수 있습니다. 이런 기능의 JQuery는 현재 가장 인기 있는 자바스크립트 라이브러리 중 하나입니다. 더 나아가 구버전을 포함한 대부분의 브라우저에서 지원이 가능하며, HTML DOM을 쉽게 조작가능하고, CSS스타일도 간단하게 적용할 수 있는 장점이 있습니다. 많은 기능을 하는 JQuery가 있지만 간단하게 appendTo를 사용하여 내용을 출력하는 예제를 보겠습니다. 코드는 아래 이미지와 같습니다.

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4  <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
5  <%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
6  <%@ taglib prefix = "x" uri = "http://java.sun.com/jsp/jstl/xml" %>
7  <%@ taglib prefix = "fn" uri = "http://java.sun.com/jsp/jstl/functions"%>
8
9
10 <!DOCTYPE html">
11<html>
12<head>
13 <script src="http://code.jquery.com/jquery-latest.min.js"></script>
14 <script src="http://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
15</script>
16 $(document).ready(function() {
17     $("<div><p>jQuery : Test of jQuery!!</p></div>").appendTo("body");
18 });
19 </script>
20 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
21 <title>Insert title here</title>
22 </head>
23 <body>
24 <h3> TEST</h3>
25 <table border="1">
26 <tr>
27     <td>201572202</td>
28     <td>정용훈</td>
29 </tr>
30 </table>
31
32 <c:out value="Test 입니다!"></c:out>
33 <br><br>
34
35 </html>
```

해당 코드를 보면 "Test of jQuery!!"라는 문구가 body부분에는 없지만 appendTo를 통하여 body에서 출력되는 결과를 확인할 수 있습니다. 결과 화면은 아래와 같습니다.



원래 있던 문구들의 밑으로 문구가 들어간 것을 확인할 수 있습니다. 이번 실습은 간단한 예제였지만 이렇게 JQuery를 응용한다면 짧은 코드를 사용하여 다양한 효과를 나타낼 수 있습니다.

```

1 <!DOCTYPE html>
2 <html lang="ko">
3
4 <head>
5     <meta charset="UTF-8">
6     <title>jQuery Event</title>
7     <script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
8     <script>
9         $(function() {
10             $("button").one("click", function() {
11
12                 $("#text").append("클릭 완료!<br>");
13
14                 $(this).click(function() {
15                     $("#text").append("이미 완료 되었습니다!<br>");
16                 });
17             });
18         });
19     </script>
20 </head>
21
22 <body>
23
24     <h1>2015722025 정용훈 JQuery</h1>
25     <button>클릭!</button>
26     <p id="text"></p>
27
28 </body>
29
30 </html>

```

다음은 클릭을 인식하여 첫번째 클릭이면 클릭 완료를 알리며 한번 더 클릭하게 되면 두 이미 완료되었다는 문구를 띄우는 함수입니다. 이렇게 JQuery는 이벤트의 대한 여러가지 제어가 가능합니다.

2015722025 정용훈 JQuery

클릭!



클릭 완료!

처음 눌렀을 경우

2015722025 정용훈 JQuery

클릭!



클릭 완료!

이미 완료 되었습니다!

두번째부터 눌렀을 경우 결과화면

2015722025 정용훈 JQuery

클릭!

클릭 완료!

이미 완료 되었습니다!

이미 완료 되었습니다!

이미 완료 되었습니다!

이미 완료 되었습니다!

이미 완료 되었습니다!

이미 완료 되었습니다!

이미 완료 되었습니다!

이미 완료 되었습니다!

이미 완료 되었습니다!

JavaScript

웹 사이트의 구성은 크게 HTML, CSS, 자바스크립트로 이루어진다. HTML은 웹의 큰 뼈대 역할을 하고 있으며, CSS는 색깔이나 글씨체와 같은 디자인을 관리하게 된다. 자바스크립트는 객체지향 스크립트 언어로 웹 페이지의 이벤트 동작을 담당하게 된다. 예를 들면 어떤 버튼의 동작을 결정하고 관리하는 것이 자바스크립트의 역할이란 것이다.

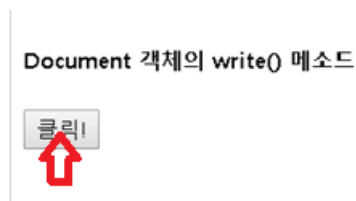
다음은 자바와 자바스크립트의 차이점을 나타낸 표이다.

JavaScript	Java
객체 지향, 객체의 형 간에 차이 없음	클래스 기반
변수 자료형이 선언되지 않음	변수 자료형은 반드시 선언
하드 디스크에 자동으로 작성 불가	하드 디스크에 자동으로 작성 가능

이런 자바스크립트의 기능을 간단한 코드를 통해 알아보도록 하겠다.

```
1 <!DOCTYPE html>
2 <html lang="ko">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>JavaScript Output</title>
7 </head>
8
9 <body>
10
11   <h5>Document 객체의 write() 메소드</h5>
12
13   <button onclick="document.write(2015722025)">클릭!</button>
14
15 </body>
16
17 </html>
```

코드의 결과는 다음과 같이 나온다.



다음과 같이 버튼이 생겼으며 소스코드에 버튼에 대한 이벤트 동작이 설정되어 있기 때문에 버튼을 누르면 다음과 같은 결과가 나오게 된다.

```
<button onclick="document.write(2015722025)">클릭!</button>
```

2015722025 <<결과 화면

```

1 <!DOCTYPE html>
2 <html lang="ko">
3
4 <head>
5     <meta charset="UTF-8">
6     <title>JavaScript Apply</title>
7 </head>
8
9 <body>
10
11     <h1>2015722025 정용훈</h1>
12     <p>자바스크립트를 이용하여 현재 날짜 시간 표시하기!</p>
13     <button onclick="printDate()">클릭!</button>
14     <p id="date"></p>
15     <script>
16         function printDate() {
17             document.getElementById("date").innerHTML = Date();
18         }
19     </script>
20
21 </body>
22
23 </html>

```

스크립트를 이용한 현재 날짜와 시간을 표시하는 코드입니다!

2015722025 정용훈

자바스크립트를 이용하여 현재 날짜 시간 표시하기!



Fri Nov 16 2018 19:39:43 GMT+0900 (한국 표준시)

CSS

CSS란 위에서 언급한 것 과 마찬가지로 웹 문서의 전반적인 디자인을 미리 저장해 둔 스타일 시트를 뜻한다. 기존에는 HTML이 스타일까지 모두 다루었으나 설계하고 수시로 변경하는데 많은 제약이 따르는데, 이를 보완하기 위해 만들어진 것이 스타일 시트이며, 이에 대한 표준안이 바로 CSS이다. CSS통하여 웹 디자인의 글자의 크기, 글자체, 줄 간격, 배경 색상, 배열위치 등을 자유롭게 선택하거나 변경할 수 있으며 관리가 간편하게 할 수 있다. 이 CSS에 대한 간단한 예시 코드는 아래와 같다.

```
1 <!DOCTYPE html>
2 <html lang="ko">
3   <head>
4     <meta charset="utf-8">
5     <title>Test of CSS!</title>
6     <style>
7       #학번 {
8         color:red;
9         font-weight: bold;
10      }
11      #정용훈 {
12        color:blue;
13        font-weight: normal;
14      }
15    </style>
16  </head>
17  <body>
18    <div id="학번">
19      2015722025
20    <div id="정용훈">
21      정용훈
22    </div>
23  </div>
24 </body>
25 </html>
```

해당 코드이며 color를 통하여 색을 변화시켜주었고 글의 굵기 또한 변경해주었다. 해당 코드의 결과화면은 아래와 같다.

2015722025

정용훈


```

1 <!DOCTYPE html>
2 <html lang="ko">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>Test of CSS!</title>
7   <style>
8     #학번 {
9       color:red;
10      font-weight:bold;
11      background-color: yellow;
12    }
13    #정용훈 {
14      color:blue;
15      font-weight:normal;
16      text-decoration: underline;
17    }
18  </style>
19 </head>
20 <body>
21   <div id="학번">
22     2015722025
23     <div id="정용훈">
24       정용훈
25     </div>
26   </div>
27 </body>
28 </html>

```

다음 예제코드는 위에서 실습한 코드에 밑줄과 배경색을 지정해준 것이다. 결과는 다음과 같다.

2015722025
정용훈

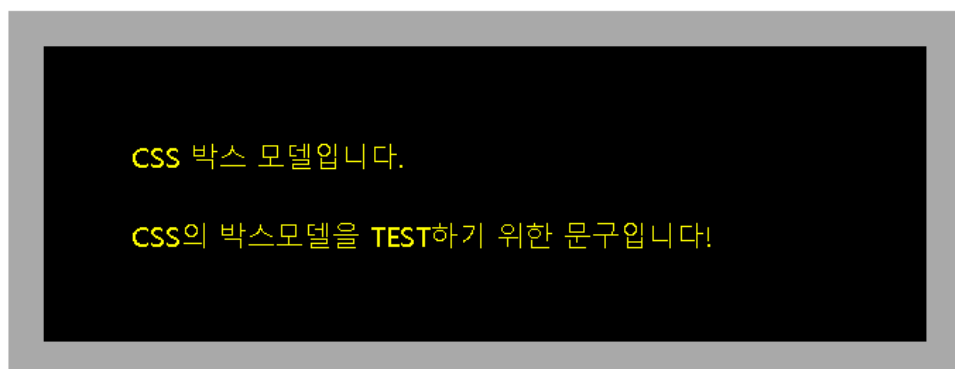
```

1 <!DOCTYPE html>
2 <html lang="ko">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>CSS Box Model</title>
7   <style>
8     div {
9       background-color: black;
10      color: yellow;
11      width: 400px;
12      padding: 50px;
13      border: 20px solid darkgray;
14      margin: 50px;
15    }
16  </style>
17 </head>
18
19 <body>
20
21   <h2>2015722025 정용훈</h2>
22   <div>CSS 박스 모델입니다.<br><br>
23     CSS의 박스모델을 TEST하기 위한 문구입니다!
24   </div>
25
26 </body>
27
28 </html>

```

마지막 CSS의 예시는 박스를 만드는 것입니다. 해당 코드의 결과는 다음과 같습니다.

2015722025 정용훈



CSS사용하면 훨씬 많은 기능의 style을 사용할 수 있습니다.

5. Consideration

이번 소프트웨어 프로젝트의 두번째 프로젝트는 자바 스프링을 사용한 튜토리얼과 게시판에 대한 기본적인 구현, 추가 구현이었다. 고찰에 앞서 솔직한 심정으로 많은 어려움이 있었던 project이다 물론 여러가지 블로그와 정보를 찾아보며 Project를 끝내기는 했지만 수업진도와는 아예 맞지 않았기에 개인적으로는 힘든 프로젝트였다. 우선 튜토리얼을 진행하는데 있어서도 많은 실수와 오타, 버전을 바꿔줘야 하는 문제 등으로 진행이 많이 더디게 되었다. 오류에 대한 경고없이 컴파일이 바로 되기 때문에 오타를 찾는데 시간이 굉장히 많아 걸렸던 것 같다. 튜토리얼의 이해와 진행은 수월했던 것 같다. MVC에 대한 개념은 저번학기 객체지향프로그래밍을 배우면서 배운 MFC와 비슷하다고 생각되었으며 거기에 DB에 대한 개념이 들어간 것뿐이었다. DB와 MVC에 대한 설정은 PDF에 자세하게 나와있기 때문에 어려움없이 설정하였으며 MVC에 대한 튜토리얼도 차근차근 보면 이해가 되기 때문에 큰 어려움이 없었다. 문제는 추가적인 기능에 대한 것이다. 우선 블로그나 여러 정보를 참고하기에는 Project의 틀이 달라서 소스코드를 우리 프로젝트에 맞추는 것이 관건이었다. 조회수기능은 이미 ViewCnt에 대한 정의가 되어있었고 ViewCnt를 업데이트를 하는 방법은 많은 정보가 있기 때문에 수월하게 진행하였다. 하지만 페이징과 검색기능, 업로드 관련기능은 우리가 구현한 게시판 틀에 맞춰야 하기 때문에 시간이 굉장히 많이 걸렸다. 특히 DB에 대한 이해가 많이 부족해서 DB를 따로 설정해야 하는 추가구현은 이번프로젝트에서 못하게 되었다. Ex)댓글 구현 기존의 설정된 DB를 바탕으로 paging과 검색기능을 완성시켰으며 업로드 관련 구현은 DB기반이 아닌 선택된 자료를 원하는 폴더로 업로드 시킬 수 있는 구현을 하게 되었다. 이는 오픈소스를 이용하여 과제 제출 관련 업무를 볼 때 사용할 수 있을 것이라고 생각되었다. 또한 보고서를 쓰면서 JSTL, 자바스크립트, html, SCC에 대하여 배우게 되었는데 현재 우리가 사용하고 있는 웹 페이지들이 역할이 구분되어 작성되는 것을 알 수 있었고 웹페이지를 한번 구현해보고 싶은 마음도 생기게 되었다. 평소 배웠던 언어들과는 달리 가시적으로 웹 사이트의 변경이 보이며 동작을 구현하는 실습을 많이 하면서 MVC이나 JavaScript관련 언어에 관심이 가기 시작한 것 같다. 마지막으로 이번 프로젝트를 통해 전반적인 DB와 MVC에 대한 개념을 익힐 수 있었지만 완벽하게 이해하지 못한 부분이 많기 때문에 조금 아쉬운 Project가 되었다. Project를 진행하기전에 수업 진도에 대한 조정이 많이 필요해 보인다.