

스프링 부트 웹 프로젝트

chapter08

다대일관계 처리하기

제공된 자료는 훈련생의 수업을 돕기 위한 것으로, 타인과 공유하시면 안됩니다.

Contents

part.1

회원 관리 기능

part.2

회원 목록 조회

part.3

회원가입

part.4

회원 상세 조회

part.5

메인화면과 사이드바

해당 프로젝트는 기존 코드를 활용할 예정입니다.

이전에 작성한 "project7"을 복사하여 "project8"를 새로 생성하세요.

 project7 project8

프로젝트에서 에러가 발생했으면, setting.gradle 파일을 열어서 프로젝트의 이름을 변경하세요.

그리고 gradle을 리프레쉬하면 에러가 사라집니다.

```
✓ [IDE] > project8 [spring-study main]
  > [IDE] > src/main/java
  > [IDE] > src/main/generated
  > [IDE] > src/main/resources
  > [IDE] > src/test/java
```

```
settings.gradle ×
1 rootProject.name = 'project07'
2
```

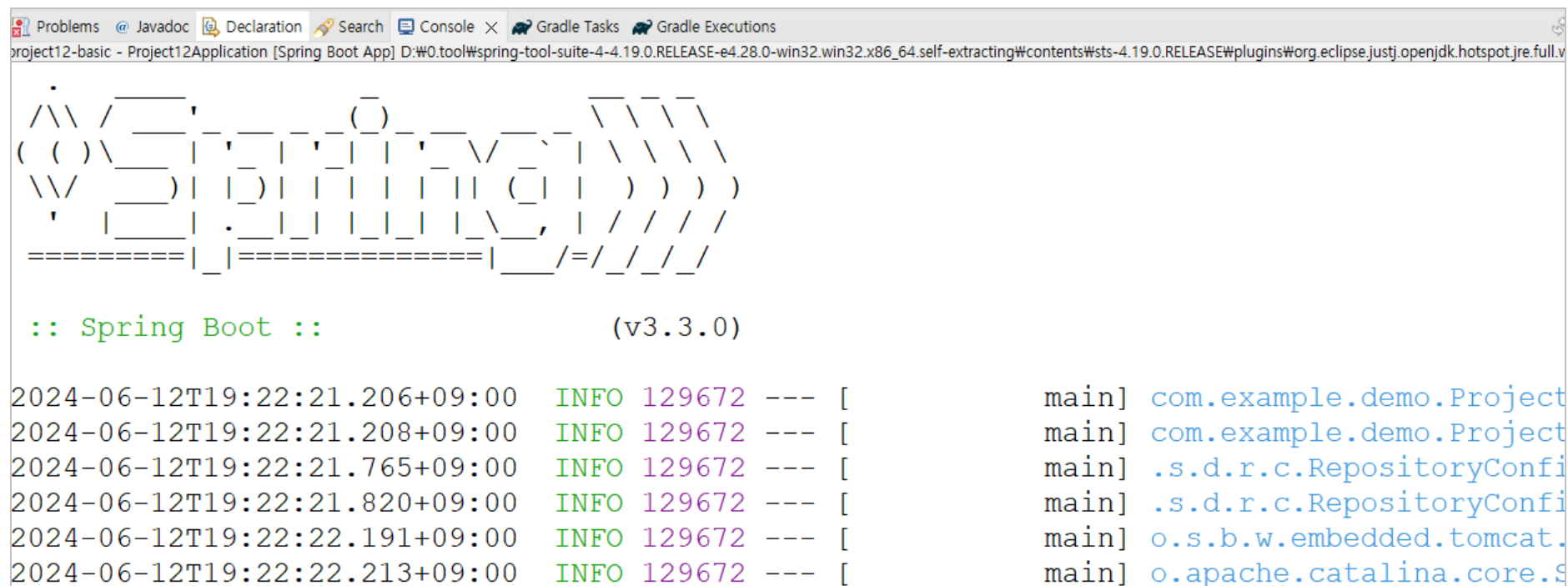


```
settings.gradle ×
1 rootProject.name = 'project8'
2
```

이전에 작성했던 단위테스트는 모두 삭제하세요.

```
▼ > src/test/java  
▼ > com.example.demo
```

프로젝트가 정상적으로 실행되는지 확인하세요.



```
project12-basic - Project12Application [Spring Boot App] D:\000\spring-tool-suite-4-4.19.0.RELEASE-e4.28.0-win32.win32.x86_64.self-extracting\contents\sts-4.19.0.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.v...  
  
:: Spring Boot ::                (v3.3.0)  
  
2024-06-12T19:22:21.206+09:00 INFO 129672 --- [main] com.example.demo.Project12Application: Starting Project12Application (v3.3.0) on ...  
2024-06-12T19:22:21.208+09:00 INFO 129672 --- [main] com.example.demo.Project12Application: The following profiles are active: ...  
2024-06-12T19:22:21.765+09:00 INFO 129672 --- [main] .s.d.r.c.RepositoryConfigurationDelegate: Loading spring data from ...  
2024-06-12T19:22:21.820+09:00 INFO 129672 --- [main] .s.d.r.c.RepositoryConfigurationDelegate: Loading spring data from ...  
2024-06-12T19:22:22.191+09:00 INFO 129672 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer: Tomcat initialized with port(s): ...  
2024-06-12T19:22:22.213+09:00 INFO 129672 --- [main] o.apache.catalina.core.StandardEngine: Starting Servlet engine: ...
```

기능 목록

- 사용자 등록
- 사용자 목록 조회
- 사용자 상세 조회
- 사용자 수정 및 삭제 (생략)

URL 설계

기능	URL	GET/POST	기능	Redirect URL
목록	/member/list	GET	목록화면	
등록	/member/register	GET	입력화면	
	/member/register	POST	등록처리	/member/list
조회	/member/read	GET	조회화면	

게시판은 일반적으로 여러 정보가 함께 처리되어 게시물을 구성한다.
게시물 정보에는 회원 정보도 포함되어 있다.

Start Bootstrap	Toggle Menu																										
회원관리	Board List Page 게시물 등록																										
게시물관리	<table><tr><th>#</th><th>제목</th><th>작성자</th><th>등록일</th></tr><tr><td>5</td><td>5번글</td><td>user2</td><td>2023/08/06</td></tr><tr><td>4</td><td>4번글</td><td>user1</td><td>2023/08/06</td></tr><tr><td>3</td><td>3번글</td><td>user3</td><td>2023/08/06</td></tr><tr><td>2</td><td>2번글</td><td>user2</td><td>2023/08/06</td></tr><tr><td>1</td><td>1번글</td><td>user1</td><td>2023/08/06</td></tr></table>			#	제목	작성자	등록일	5	5번글	user2	2023/08/06	4	4번글	user1	2023/08/06	3	3번글	user3	2023/08/06	2	2번글	user2	2023/08/06	1	1번글	user1	2023/08/06
#	제목	작성자	등록일																								
5	5번글	user2	2023/08/06																								
4	4번글	user1	2023/08/06																								
3	3번글	user3	2023/08/06																								
2	2번글	user2	2023/08/06																								
1	1번글	user1	2023/08/06																								
	Board 엔티티	Member 엔티티																									

회원 관리 기능

회원과 게시물의 관계

회원과 게시물은 1:N 관계이다.

회원 데이터가 '1'이고 게시물 데이터가 'N'이 되어, 한 명의 회원이 여러 게시물을 작성할 수 있다.

게시물의 작성자 컬럼에는 동일한 회원 아이디가 여러 번 입력될 수 있다는 의미이다.

테이블 구조로 보면, 게시물 테이블의 작성자는 사용자 테이블의 아이디를 참조한다.

회원과 게시물 데이터

	id	name	password		no	title	content	writer_id
1	user1	둘리	1234	1	1번글	오늘 날씨가..	☞	user1
2	user2	또치	1234	2	2번글	집에 오늘 길은..	☞	user2
3	user3	도우너	1234	3	3번글	오늘 점심에..	☞	user3
4	user4	마이콜	1234	4	4번글	차가 막혀서..	☞	user1
5	user5	고길동	1234	5	5번글	이번 개봉 영화..	☞	user2

회원과 게시물 테이블

사용자	논리 이름*	데이터 타입	널 허용
아이디	VARCHAR(20)	N·N	
패스워드	VARCHAR(255)	N·N	
이름	VARCHAR(20)	N·N	
등록일	DATE	NULL	
수정일	DATE	NULL	

게시물	논리 이름*	데이터 타입	널 허용
글번호	INT	N·N	
제목	VARCHAR(50)	N·N	
내용	VARCHAR(255)	NULL	
작성자	VARCHAR(20)	N·N	
등록일	DATE	NULL	
수정일	DATE	NULL	

회원 엔티티를 생성한 후, BaseEntity를 상속받아 날짜 필드를 추가한다.

회원 리파지토리를 생성한 후, JpaRepository를 상속받는다.

Member

```
public class Member extends BaseEntity {
```

MemberRepository

```
public interface MemberRepository extends JpaRepository<Member, String> {  
}
```


Q. 단위테스트와 리파지토리를 이용하여 회원 등록, 조회, 수정, 삭제를 처리한다.

MemberRepositoryTest

```
@Test
public void 회원등록() { }

@Test
public void 회원목록조회() { }

@Test
public void 회원단건조회() { }

@Test
public void 회원수정() { }

@Test
public void 회원삭제() { }
```

tbl_member 테이블

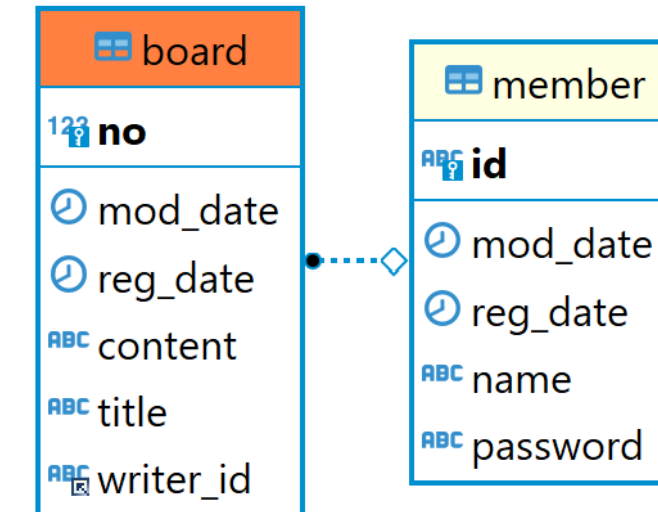
	id ↑	name	password	mod_date	reg_date
1	user1	둘리	1234	2024-09-29 18:31:32.473	2024-09-29 18:31:32.473
2	user2	또치	1234	2024-09-29 18:31:32.502	2024-09-29 18:31:32.502

게시물 엔티티에서 작성자 필드 변경하기

게시물 테이블의 작성자 컬럼은 회원 테이블의 아이디를 참조한다.

따라서 외래키(FK)를 사용하는 엔티티의 구조를 변경해야 한다.

Board 엔티티에서 Writer 필드의 자료형을 Member 타입으로 변경하고, 다대일 관계를 설정하기 위해 @ManyToOne 어노테이션을 적용한다.



Board

```

public class Board extends BaseEntity {
    ...
    @ManyToOne
    Member writer;
}
  
```

SQL

Hibernate:

```

alter table if exists board
add constraint FK_i57kt4qb1qssjljxotb26a4h0
foreign key (writer_id)
references member (id)
  
```

외래키가 추가됨

게시물 서비스에서 작성자 필드 처리

게시물 엔티티의 구조가 변경되어 변환 메소드에서 에러가 발생하므로, 이를 수정해야 한다.

dtoToEntity() 메소드에서 Member 엔티티를 생성한 후, 작성자 필드에 저장한다.

entityToDto() 메소드에서 Member 엔티티에서 id를 꺼내 작성자 필드에 저장한다

BoardService

```
default Board dtoToEntity(BoardDTO dto) {  
    Member member = Member.builder().id(dto.getWriter()).build();  
    Board entity = Board.builder()  
    ...  
    .writer(member)  
}
```

```
default BoardDTO entityToDto(Board entity) {  
    BoardDTO dto = BoardDTO.builder()  
    ...  
    .writer(entity.getWriter().getId())  
}
```

Board와 BoardDTO

Board	BoardDTO
<pre>public class Board extends BaseEntity { int no; String title; String content; Member writer; //작성자</pre>	<pre>public class BoardDTO { int no; String title; String content; String writer;</pre>

게시물 등록 테스트

기존 게시물 테이블을 삭제한 후, 새로 생성한다.

그리고 Board리파지토리와 이용하여 새로운 게시물을 등록한다.

엔티티에 참조관계가 설정되어 있으므로, 게시물의 작성자는 tbl_member에 존재하는 회원의 아이디를 사용해야 한다.

BoardRepositoryTest

```
@Test  
void 게시물등록() { }
```

tbl_member

	id	name	password
1	user1	둘리	1234
2	user2	또치	1234

결과 화면

	no	title	content	writer_id	mod_date	reg_date
1	1	안녕하세요	안녕하세요	user1	2024-09-29 18:17:40.490	2024-09-29 18:17:40.490
2	2	반갑습니다	반갑습니다	user1	2024-09-29 18:20:01.224	2024-09-29 18:20:01.224

게시물 등록 테스트

Board서비스를 이용하여 새로운 게시물을 등록한다.

BoardServiceTest

```
@Test  
void 게시물등록() { }
```

결과 화면

	123no ▼	ABCtitle ▼	ABCcontent ▼	ABCwriter_id ▼	mod_date ▼	reg_d
1	1	안녕하세요	안녕하세요	user1	2024-10-01 20:49:06.716	2024-10
2	2	반갑습니다	반갑습니다	user1	2024-10-01 20:49:06.798	2024-10
3	3	하이	하이	user1	2024-10-01 20:49:13.392	2024-10

게시물 조회 테스트

Board리파지토리를 이용하여 특정 게시물을 조회한다.

SQL 쿼리에서 BOARD 테이블과 MEMBER 테이블이 JOIN 되어 처리된다.

결과를 보면 조회한 게시물에 작성자의 회원정보가 포함되어 있다.

BoardRepositoryTest

```
@Test
public void 게시물조회() {
}
```

결과 화면

```
w1_0.password,
w1_0.reg_date
from
board b1_0
left join
member w1_0
on w1_0.id=b1_0.writer_id
```

```
Board(no=1, title=안녕하세요, content=안녕하세요, writer=Member(id=user1, password=1234, name=둘리))
```

회원 삭제 테스트

Member리파지토리를 이용하여 모든 회원 정보를 삭제한다.

이때 게시물을 작성한 회원은 바로 삭제할 수 없다.

게시물이 회원을 참조하고 있기 때문에, 게시물을 먼저 삭제한 후에 회원을 삭제해야 한다.

MemberRepositoryTest

```
@Test
public void 첫번째회원삭제 () {

}

@Test
public void 두번째회원삭제 () {

}
```

tbl_board 테이블

	123 no ▼	ABC title ▼	ABC content ▼	ABC writer_id ▼
1	1	안녕하세요	안녕하세요	user1
2	2	반갑습니다	반갑습니다	user1
3	3	하이	하이	user1

tbl_member 테이블

	ABC id ▼	ABC name ▼	ABC password ▼
1	user1	둘리	1234
2	user2	또치	1234

게시물 리파지토리에 삭제 메소드 추가

특정 회원이 작성한 게시물을 일일이 삭제하기 번거롭기 때문에, 모든 게시물을 일괄 삭제하는 메소드를 추가한다.
그런데 메소드를 호출하면 트랜잭션 처리가 거부되어 에러가 발생된다.

이때 리파지토리에 @Transactional 어노테이션을 설정하면 정상적으로 Commit 처리된다.

commit: SQL 작업 결과를 데이터베이스에 영구적으로 반영하는 것

BoardRepository

```
// 특정 작성자가 작성한 모든 게시물을 삭제하는 메소드
@Query("delete from Board where writer = :member")
public void deleteWriter(@Param("member") Member member);
```

에러

: No EntityManager with actual transaction available for current thread - cannot reliably process 'remove' call

MemberDTO

컨트롤러와 뷰 사이에서 회원 데이터를 전달하는 객체를 만든다.

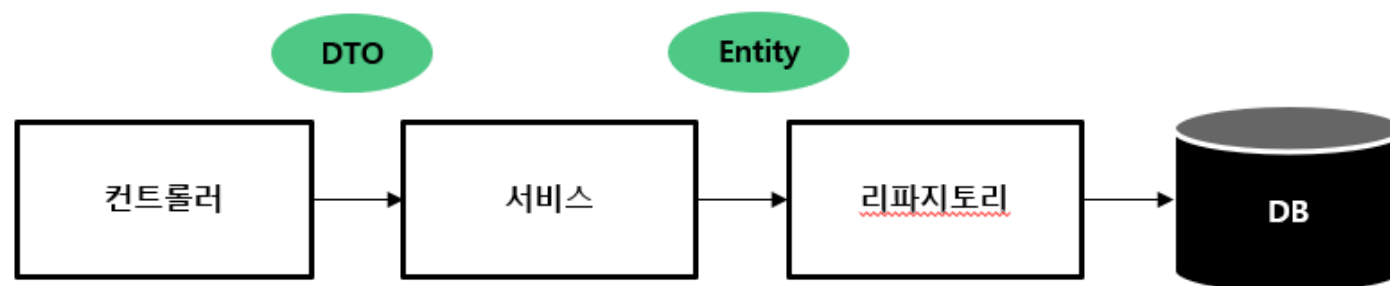
MemberService와 MemberServiceImpl

컨트롤러와 리파지토리 사이에서 사용자 요청을 전달하는 서비스를 만든다.

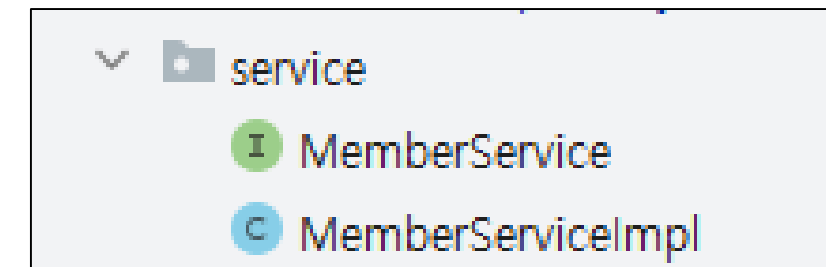
서비스는 인터페이스와 구현클래스로 구성되어 있다.

DTO와 엔티티간의 변환 메소드를 추가한다.

Spring MVC 구조



서비스 구조



목록 조회 기능

회원 목록을 출력하며, 각 회원의 요약된 정보를 표시한다.

결과 화면

Member List Page		
아이디	이름	등록일
user3	고길동	2023/03/01
user2	또치	2023/03/01
user1	또치	2023/03/01
1		

DTO 리스트 출력

서비스에서 목록 처리

- 회원 목록을 조회하는 메소드를 추가한다.
- 매개변수로 페이지번호를 수집하고, 해당 페이지 정보를 반환한다.
- 엔티티를 DTO로 변환하는 메소드를 추가한다.
- 구현 클래스에서는 상속받은 목록 메소드를 구현한다.
- entityToDTO() 메소드를 사용하여 엔티티 리스트를 DTO 리스트로 변환한다

인터페이스

```
Page<MemberDTO> getList(int pageNumber);

default MemberDTO entityToDto(Member entity) {
    ...
}
```

구현 클래스

```
@Override
public Page<MemberDTO> getList(int page) {
    ...
}
```

서비스의 테스트

- 단위테스트를 통해 회원 목록을 조회한다.
- 엔티티 리스트가 DTO 리스트로 변환되었는지 확인한다.

MemberServiceTest

```
@SpringBootTest
public class MemberServiceTest {

    @Autowired
    MemberService service;
```

```
@Test
public void 회원목록조회() {

}
```

엔티티 객체를 DTO 객체로 변환 확인

```
✓ 테스트 통과: 1 / 1개 테스트 - 249ms
MemberDTO(id=user5, password=1234, name=고길동, regDate=2023-08-08)
MemberDTO(id=user4, password=1234, name=마이콜, regDate=2023-08-08)
MemberDTO(id=user3, password=1234, name=도우너, regDate=2023-08-08)
MemberDTO(id=user2, password=1234, name=또치, regDate=2023-08-08)
MemberDTO(id=user1, password=1234, name=둘리, regDate=2023-08-08)
```

컨트롤러에서 목록 처리

- 회원 목록 화면을 반환하는 메소드를 추가한다.
- 페이지번호를 수집하기 위한 파라미터를 추가한다.
- 서비스를 사용하여 게시물 리스트를 조회한다.
- 화면에 게시물 리스트를 전달한다.

BoardController

```
@GetMapping("/list")
public void list(@RequestParam(defaultValue = "0") int page, Model model) {
    Page<MemberDTO> list = service.getList(page);
    model.addAttribute("list", list);
}
```

회원 목록 조회

컨트롤러와 화면에서 목록 처리

정적 페이지 활용하기

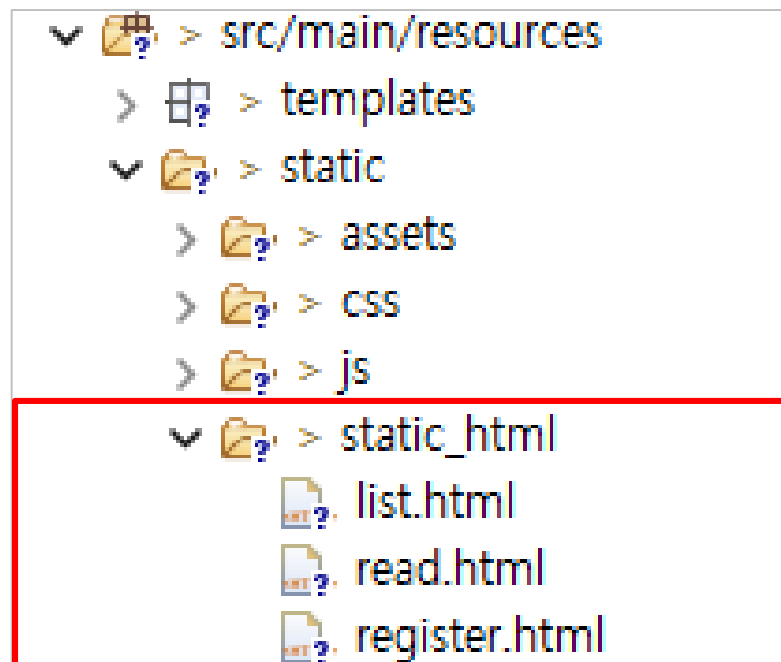
해당 프로젝트는 미리 준비된 정적 페이지를 사용한다.

정적 파일은 이미 기본적인 구조를 가지고 있으며, 동적인 부분만 타임리프로 처리한다.

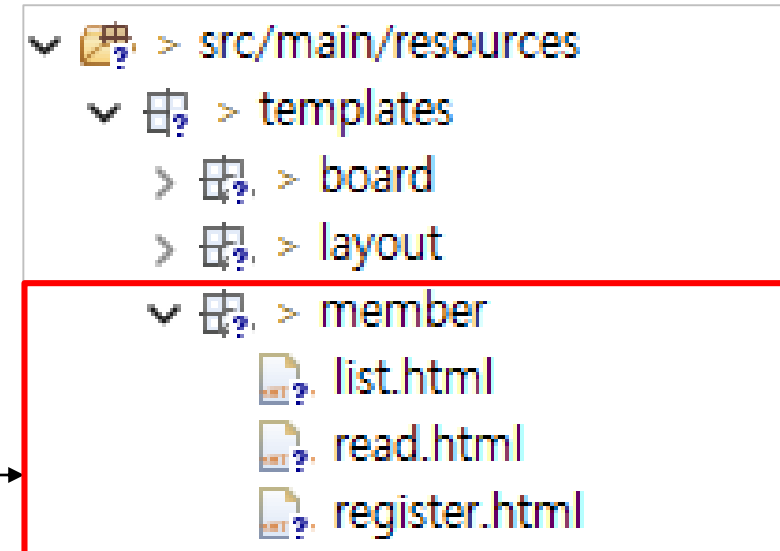
spring study의 Project08에 있는 HTML 파일을 복사하여 본인의 프로젝트에 저장한다.

(/resources/static/static_html/아래파일 → 복사 → resource/templates/member/)

예제 프로젝트



내 프로젝트

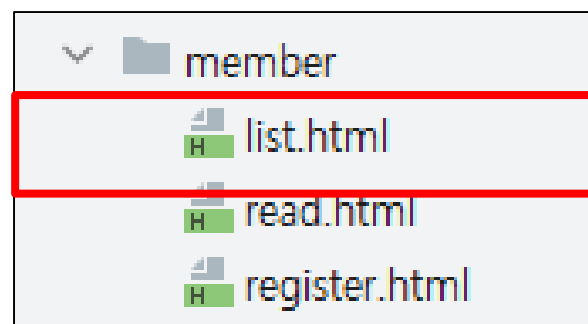


정적 페이지 활용하기

list.html을 브라우저에서 열어서 내용을 확인한다.

파일 선택 > 우클릭 > Open With 메뉴 > Web Brower 메뉴

정적 파일



결과 화면



레이아웃 적용

- 목록 화면을 조각으로 만들고, 레이아웃을 적용한다.
- basic.html 파일에 list.html의 코드 조각을 전달한다.

list.html

```
<th:block th:replace="~/layout/basic :: setContent(~{this::content} )">

  <th:block th:fragment="content">

기존 HTML 내용



  </th:block>

</th:block>
```


리스트 출력

- 전달받은 회원 리스트를 테이블을 이용해서 출력한다.
- 반복문을 사용해서 리스트를 처리한다.
- “회원의 아이디”를 클릭하면 상세화면으로 이동한다.
- 날짜는 “년/월/일” 포맷으로 출력한다.

list.html

```
<tr th:each="dto : ${list}">
  <th>
    <a th:href="@{/member/read(id = ${dto.id}, page= ${list.number + 1})}">
      [[${dto.id}]]
    </a>
  </th>
  <td>[[${dto.name}]]</td>
  <td>[[${#temporals.format(dto.regDate, 'yyyy/MM/dd')}]</td>
</tr>
```

회원 목록 조회

컨트롤러와 화면에서 목록 처리

테스트

1. localhost:8080/member/list 주소로 접속한다.
2. 회원 목록이 정상적으로 출력되는지 확인한다.

결과 화면

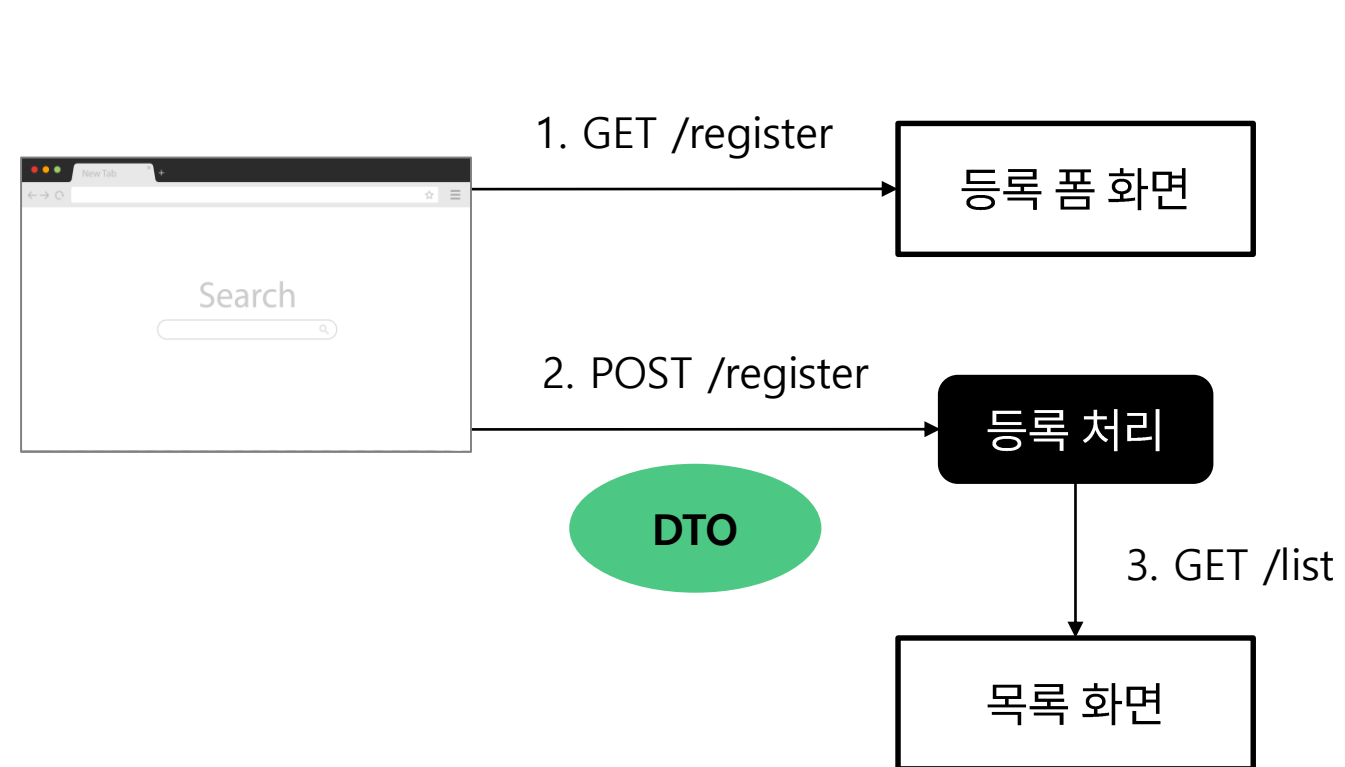
Member List Page		
아이디	이름	등록일
user30	둘리	2024/09/29
user29	둘리	2024/09/29
user28	둘리	2024/09/29
user27	둘리	2024/09/29
user26	둘리	2024/09/29
user24	둘리	2024/09/29
user25	둘리	2024/09/29
user23	둘리	2024/09/29
user22	둘리	2024/09/29
user21	둘리	2024/09/29
<div>1 2 3</div>		

회원 가입

등록 처리

등록 기능

- 사용자에게 회원 등록 폼을 보여준다.
- 사용자로부터 회원 정보를 입력 받는다.
- 사용자로부터 전달받은 회원정보를 등록한 후 목록화면으로 이동한다.



<등록 처리 흐름>

등록 성공

Member List Page		
아이디	이름	등록일
user4	자두	2023/03/01
user3	고길동	2023/03/01
user2	또치	2023/03/01
user1	또치	2023/03/01

등록 실패

localhost:8080 내용:
아이디가 중복되어 등록에 실패하였습니다

확인

<등록 화면>

서비스에서 등록 처리

- 서비스에 회원을 등록하는 메소드를 추가한다.
- DTO타입을 파라미터로 수집하고, 처리결과를 boolean값으로 반환한다.
- DTO를 엔티티로 변환하는 일반 메소드를 추가한다.
- 회원의 아이디가 사용가능 한지 확인하고, 등록을 진행한다.
- dtoToEntity() 메소드를 사용하여 전달받은 DTO를 엔티티로 변환한다.

인터페이스

```
boolean register(MemberDTO dto);

default Member dtoToEntity(MemberDTO dto) {
    ...
}
```

구현 클래스

```
@Override
public boolean register(MemberDTO dto) {
    String id = dto.getId();
    MemberDTO getDto = read(id);
    if (getDto != null) {
        System.out.println("사용중인 아이디입니다.");
        return false;
    } ...
}
```

회원 가입

등록 처리

서비스의 테스트

- 단위테스트를 통해 새로운 회원을 추가한다.
- 엔티티 객체가 DTO 객체로 변환되었는지 확인한다.

MemberServiceTest

```
@Test
public void 회원등록() {

}
```

회원 테이블

	id	mod_date	reg_date	name	password
1	user1	2023-08-07 19:11:05.796	2023-08-07 19:11:05.796	둘리	1234
2	user2	2023-08-07 19:11:05.809	2023-08-07 19:11:05.809	또치	1234
3	user3	2023-08-07 19:11:05.811	2023-08-07 19:11:05.811	도우너	1234
4	user4	2023-08-07 19:11:05.813	2023-08-07 19:11:05.813	마이콜	1234
5	user5	2023-08-07 19:11:05.814	2023-08-07 19:11:05.814	고길동	1234

컨트롤러에서 등록 처리

- GET방식으로 등록 폼 화면을 반환하는 메소드를 추가한다.
- POST방식으로 회원 등록을 처리하는 메소드를 추가한다.
- 등록폼에서 전달한 회원 정보를 파라미터로 받아서 처리한다.
- 등록에 성공하면, 목록화면으로 이동한다.
- 등록에 실패하면, 실패메세지를 전달하면서 다시 회원가입 화면으로 이동한다.

MemberController

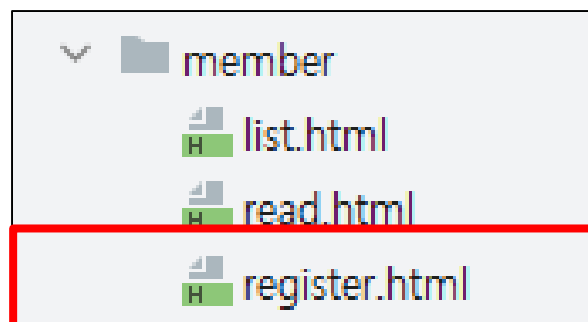
```
@GetMapping("/register")
public void register() { }

@PostMapping("/register")
public String registerPost(MemberDTO dto, RedirectAttributes redirectAttributes) {
    boolean isSuccess = service.register(dto);
    ...
}
```

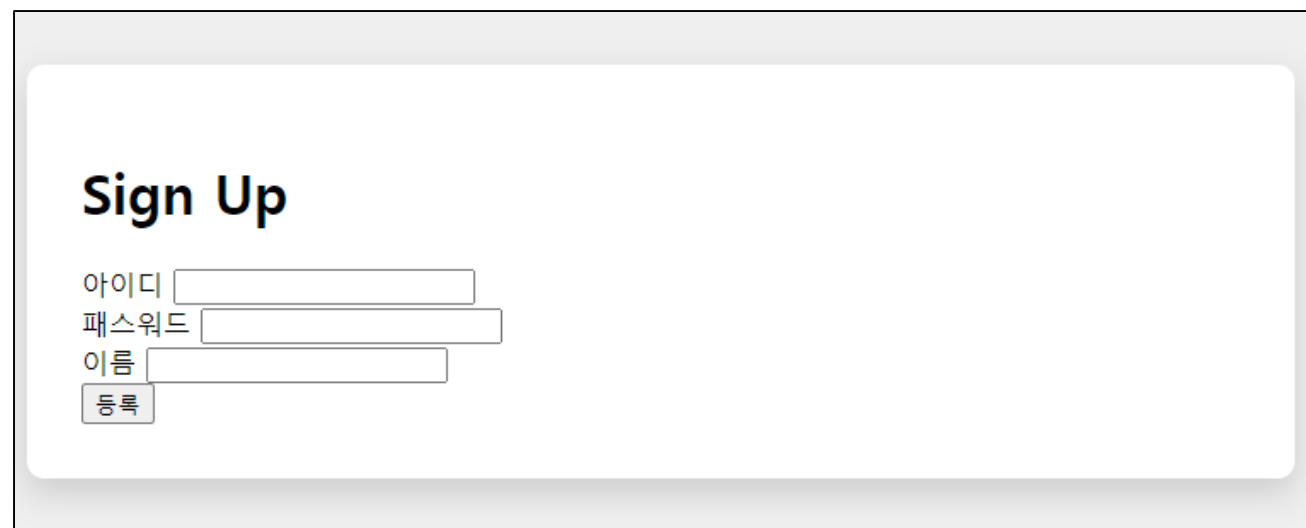
정적 페이지 활용하기

- register.html을 브라우저에서 열어서 내용을 확인한다.
- 파일 선택 > 우클릭 > Open With 메뉴 > Web Brower 메뉴
- 등록 화면은 템플릿을 적용하지 않는다.

정적 파일



결과 화면

A screenshot of a web form titled 'Sign Up'. The form contains three input fields: '아이디' (ID), '패스워드' (Password), and '이름' (Name). Below the input fields is a button labeled '등록' (Register).

입력양식 만들기

- 사용자에게 회원정보를 입력 받는 화면을 만든다.
- form태그와 input태그로 입력 필드를 구성한다.
- action속성에 데이터를 전송할 서버 URL을 지정한다.
- 데이터가 DTO타입으로 수집되므로, 입력필드에 DTO 필드와 매칭되는 name속성을 설정한다.
- 등록 버튼을 클릭하면, 등록처리가 호출되고 사용자가 입력한 데이터가 서버에 전달된다.

register.html

```
<form th:action="@{/member/register}" th:method="post">
...
<div>
    <label >아이디</label>
    <input type="text" name="id">
</div>
```

MemberDTO

```
public class MemberDTO {

    String id;

    String password;

    String name;
```


등록처리와 모달창

- 등록에 실패한 경우에 사용자에게 실패 메시지를 보여준다.
- 자바스크립트를 사용해서 컨트롤러에서 전달받은 메시지를 출력한다.

list.html

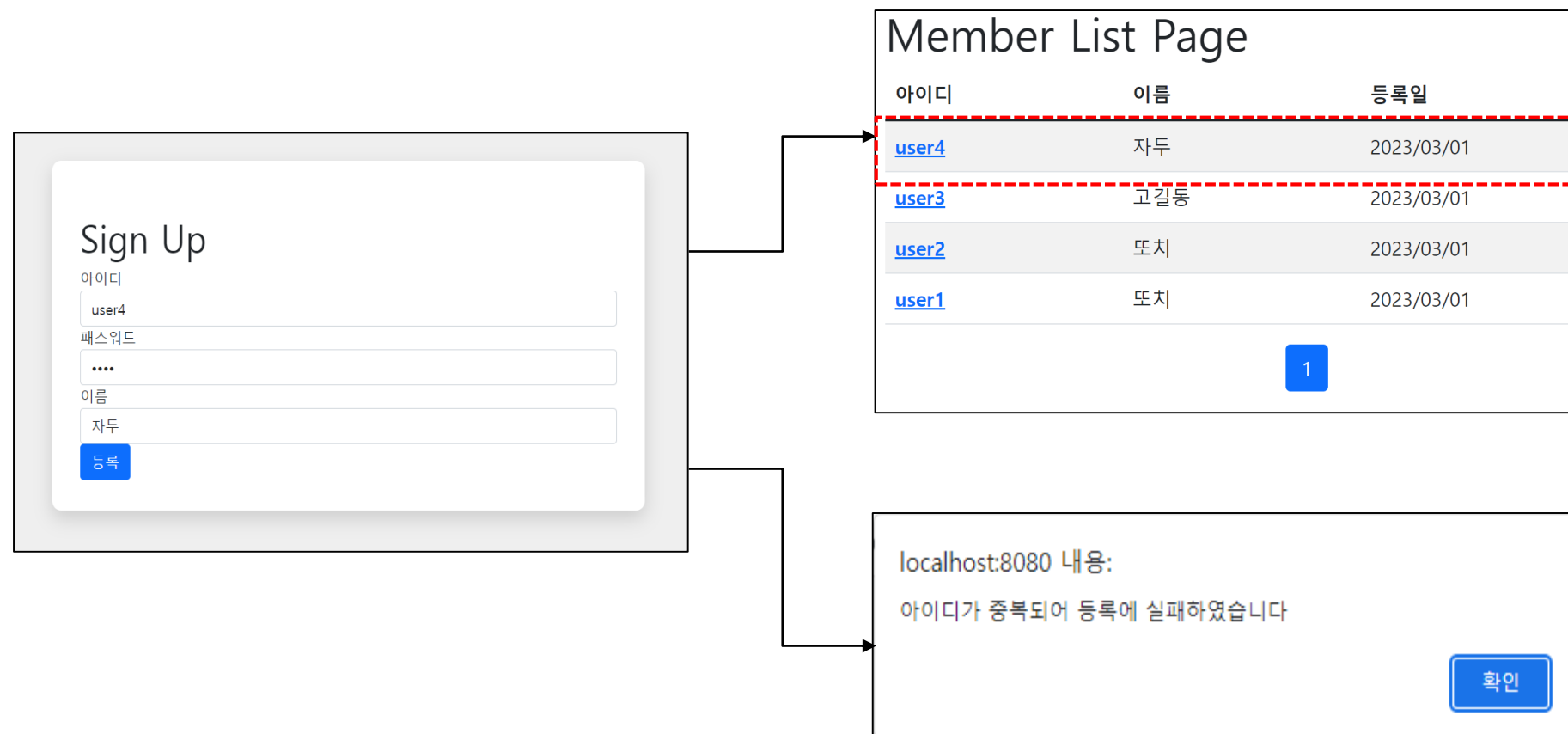
```
<script th:inline="javascript">
    $(document).ready(function(){
        var msg = [{${msg}]}
        if(msg != null){
            alert(msg);
        }
    });
</script>
```

회원 가입

컨트롤러와 화면에서 등록 처리

테스트

1. 회원가입 화면이 정상적으로 나타나는지 확인한다.
2. 회원가입 화면에서 회원정보를 입력하고 [등록] 버튼을 클릭한다.
3. 회원 등록에 성공하면 목록페이지로 이동하는지 확인한다.
4. 회원 등록에 실패하면 에러메시지가 표시되는지 확인한다.



상세 조회 기능

특정 회원 정보를 출력한다.

회원의 모든 정보를 표시하며, 해당 정보는 읽기만 가능하고 수정할 수 없다.

결과 화면

Member Read Page	
아이디	user3
패스워드
이름	고길동
등록일	2023/03/01 23:38:55
수정일	2023/03/01 23:38:55
목록으로	

DTO 출력

서비스에서 조회 처리

- 특정 회원을 조회하는 메소드를 추가한다.
- 매개변수로 회원의 아이디를 수집하고, 회원정보를 반환한다.
- 구현 클래스에서는 상속받은 조회 메소드를 구현한다.
- entityToDTO() 메소드를 사용하여 엔티티 객체를 DTO 객체로 변환한다.

인터페이스

```
MemberDTO read(String id);
```

구현 클래스

```
@Override
public MemberDTO read(String id) {
    Optional<Member> result = repository.findById(id);
    Member member = result.get();
    return entityToDto(member);
    ...
}
```

서비스의 테스트

- 단위테스트를 통해 특정 회원 정보를 조회한다.
- 엔티티 객체가 DTO 객체로 변환되었는지 확인한다.

MemberServiceTest

```
@Test
public void 회원단건조회() {

}
```

결과 화면

✓ 테스트 통과: 1 / 1개 테스트 - 204ms

MemberDTO(id=user1, password=1234, name=둘리, regDate=2023-08-07T19:11:00)

컨트롤러에서 조회 처리

- 게시물 상세 화면을 반환하는 메소드를 추가한다.
- 회원아이디와 페이지번호를 파라미터로 수집한다.
- 서비스를 사용하여 회원 정보를 조회한다.
- 화면에 회원 정보를 전달한다.

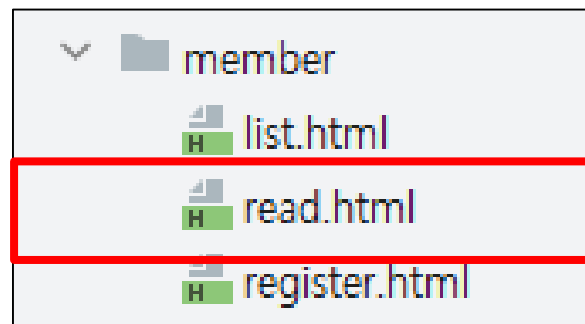
BoardController

```
@GetMapping("/read")
public void read(String id, @RequestParam(defaultValue = "0") int page, Model model) {
    MemberDTO dto = service.read(id);
    model.addAttribute("dto", dto);
    model.addAttribute("page", page);
}
```

정적 페이지 활용하기

- read.html을 브라우저에서 열어서 내용을 확인한다.
- 파일 선택 > 우클릭 > Open With 메뉴 > Web Brower 메뉴

정적 파일



결과 화면

Member Read Page

아이디

user1

패스워드

....

이름

둘리

등록일

2023/03/01 10:00:00

수정일

2023/03/01 10:00:00

목록으로

레이아웃 적용

- 목록 화면을 조각으로 만들고, 레이아웃을 적용한다.
- basic.html 파일에 read.html의 코드 조각을 전달한다.

read.html

```
<th:block th:replace="~/layout/basic :: setContent(~{this::content} )">

  <th:block th:fragment="content">

기존 HTML 내용



  </th:block>

</th:block>
```


조회 화면 만들기

- 전달받은 회원 정보를 입력필드를 이용하여 출력한다.
- 값을 변경할 수 없도록 읽기모드로 설정한다.
- 날짜는 "년/월/일 시:분:초" 패턴으로 출력한다.
- 목록으로 버튼을 클릭하면 목록화면으로 이동한다.
- 링크에 현재 페이지번호를 파라미터로 전달한다.

read.html

```
<label>아이디</label>
```

```
<input type="text" th:value="${dto.id}" readonly>
```

```
<a th:href="@{/member/list(page=${page})}">
```

```
    <button>목록으로</button>
```

```
</a>
```

회원 상세 조회

컨트롤러와 화면에서 조회 처리

테스트

1. 목록화면에서 [아이디]를 클릭한다.
2. 상세화면에서 회원 정보가 정상적으로 출력되는지 확인한다.

목록 화면

Member List Page		
아이디		이름
user3		고길동
user2		또치
user1		또치

상세 화면

Member Read Page	
아이디	user3
패스워드
이름	고길동
등록일	2023/03/01 23:38:55
수정일	2023/03/01 23:38:55
목록으로	

메인화면과 사이드바 메인화면

컨트롤러에서 메인화면 처리

1. 새로운 HomeController를 생성한다.
2. BoardController의 메인 메소드를 HomeController로 옮긴다.
3. main.html 파일을 /template/home 폴더 아래로 옮긴다.

HomeController

```
@Controller
public class HomeController {

    @GetMapping("/")
    public String home() {
        return "/home/main";
    }
}
```

메인화면과 사이드바 메인화면

테스트

localhost:8080/ 주소로 접속한다.

메인화면이 정상적으로 출력되는지 확인한다.

결과 화면



메인화면과 사이드바 사이드바

템플릿에서 메뉴 처리

- 기존에 사용하지 않는 메뉴를 삭제한다.
- 회원관리와 게시물관리 메뉴를 추가한다.
- 홈 이동 링크를 변경한다.

Start Bootstrap	Toggle Menu
Dashboard	Board List Page 게시물 등
Shortcuts	# 제목
Overview	
Events	
Profile	
Status	

가짜 메뉴

basic.html

```
<div class="list-group list-group-flush">
  <a class="list-group-item list-group-item-action list-group-item-light p-3" href="#">Dashboard</a>
  <a class="list-group-item list-group-item-action list-group-item-light p-3" href="#">Shortcuts</a>
  <a class="list-group-item list-group-item-action list-group-item-light p-3" href="#">Overview</a>
  <a class="list-group-item list-group-item-action list-group-item-light p-3" href="#">Events</a>
  <a class="list-group-item list-group-item-action list-group-item-light p-3" href="#">Profile</a>
  <a class="list-group-item list-group-item-action list-group-item-light p-3" href="#">Status</a>
</div>
```

2개만 남기고 삭제하기

```
<a href="/member/list">회원관리</a>
<a href="/board/list">게시물관리</a>
..
```

메뉴 추가

```
<li class="nav-item active"><a class="nav-link" href="/">Home</a></li>
```

홈 링크 수정

메인화면과 사이드바 사이드바

테스트

1. 사이드바에 회원관리와 게시물관리 메뉴가 나타나는지 확인한다.
2. 회원관리 메뉴를 클릭하면 회원 목록으로 이동하는지 확인한다.
3. 게시물관리 메뉴를 클릭하면 게시물 목록으로 이동하는지 확인한다.

결과 화면

