

# 목차

1. 프로젝트 설명
  - a. 프로젝트 폴더 계층 구조
  - b. 사용 기술 및 언어
2. **ERD**
3. 게시판 및 댓글 관련 기능
  - a. 게시판 **CRUD**
  - b. 답변형 게시판 **CRUD**
  - c. 게시글 검색
  - d. 대댓글(중첩 페이징)
4. 유저 관련 기능
  - a. 내가 쓴 글 확인
5. 관리자 관련 기능
  - a. 글 및 회원관리

## 1-a. 프로젝트 폴더 계층구조(MVC 모델)

**\*HealthChain->src->main**에 가보면 **java**폴더와 **webapp**폴더가 있는데 이 두 폴더 기준으로 설명드립니다. 아래 구조에서 설명안된 것들은 설정관련 파일들 이라 생각하시면 됩니다.

- **java**
  - **api** : api 관련 기능.
  - **controller** : MVC 중 Controller.
  - **model** : MVC 중 Model(DTO).
    - **dao** : DAO
  - **service** : MVC 중 Model의 service 레이어.
  - **xml files** : Mybatis SQL 코드들.
- **webapp**
  - **index.jsp** : 프로그램 진입점.
  - **view** : MVC 중 View.

## 1-b. 사용기술 및 언어

- **기술** : JSP, Servlet, ORACLE, Mybatis, AJAX
- **언어** : JS, JAVA, SQL

## 2. ERD

**ERD 주소** : <https://www.erdcloud.com/d/r6jwhxYsMndAXySjf>


**설명** : 의사면허 테이블 같은 경우에는 단순히 면허코드를 체크하기 위해 만든 테이블입니다. **ERD**에서 이와 같은 테이블을 만들면 안되지만 실제 면허코드를 다룰 수는 없으니 상황을 가정해서 만들었습니다.

## 3-a. 게시판 CRUD

776	frbNotice1	관리자	2025-01-23	12
807	게시글작성	nickName1	2025-02-13	97
806	게게시	nickName1	2025-02-13	1
805	tjhahsgshg	nickName1	2025-02-12	4
804	sdfhdsfhs	nickName1	2025-02-12	0
802	fhadfhadf	nickName1	2025-02-12	1

- **create** : 게시글 생성 기능입니다. 관리자는 글쓰면 공지사항이 되게했습니다(관리자글을 **SQL**에서 **UNION ALL**을 통해 위쪽으로 배치 나머지는 아래로 배치해 구현했습니다. 기존 게시글에서 관리자 글만 뺀것이니 **PK**는 중복되지 않습니다).
- **read** : 게시글 조회 기능입니다. 게시글 조회시 조회수가 1씩 증가하게 됩니다. 조회에 필요한 게시글 **PK**를 서버로 보내야 합니다. 추천기능이 있어 게시글을 추천할 수 있습니다.
- **update** : 게시글 수정 기능입니다. **read**와 동일하게 게시글 **PK**가 필요하며 추가적으로 유저아이디가 필요합니다. 세션에서 해당 작성자에 대한 검증이 필요하기 때문입니다.
- **delete** : 게시글 삭제 기능입니다. **update**와 기능은 다르나 프로세스가 같아 **update**에 보내는 것과 같은 유형의 데이터가 필요합니다. 실제로는 삭제가 되지 않고 삭제했다는 표시만 됩니다.

### 3-b. 답변형 게시판 **CRUD**

3497	df	nickName1	2025-02-12	1
3495	게시글	nickName1	2025-02-12	0
3493	ssssssssssssssssssssss	nickName1	2025-02-12	16
 3494	dddddddddddddddddd	관리자	2025-02-12	2
3471	csTitle234	nickName1	2025-02-04	7
 3490	csTitle234-2	관리자	2025-02-06	2
 3472	csTitle234-1	관리자	2025-02-04	3
3470	csTitle233	nickName1	2025-02-04	0

- **create** : 답변형 게시판같은 경우에는 계층형 쿼리를 사용하기 위해 답글을 달 대상인 부모 게시글 PK가 필요합니다.
- **read, update** : 일반적인 게시글 **read, update**와 동일합니다. 답변형 게시글을 포함한 게시판 이지만, 읽거나 수정하는데는 일반적인 게시판 글과 동일하기 때문입니다.
- **delete** : 답변형 게시판 구조를 참고하시면 이해하시기 편하실 겁니다.

**delete 프로세스 설명** : 답변형 게시판의 **delete** 프로세스는 일반형 게시판의 **delete** 프로세스와 다릅니다. 아래의 표를 한번 생각해 봅시다.

게시글 1
게시글 1-1 (게시글 1에 대한 답글)
게시글 1-1-1 (게시글 1-1에 대한 답글)

게시글 1을 삭제했다고 생각해봅시다. 그럼 다음과 같이 됩니다.

삭제된 게시글 입니다.
게시글 <b>1-1</b> (게시글 <b>1</b> 에 대한 답글)
게시글 <b>1-1-1</b> (게시글 <b>1-1</b> 에 대한 답글)

여기서 삭제된 게시글 이라는 게시글은 표시해도 되고 표시하지 않아도 됩니다.

그리고 다음과 같은 상황도 한번 생각해 봅시다.

어떤 폴더가 있는데, 이 폴더안에 여러파일들을 넣었습니다. 그리고 이 파일이 담겨진 폴더를 삭제를 하면 안에있는 파일들도 같이 삭제가 됩니다. 두 가지 상황을 예로 한번 들어봤는데 먼저 첫번째 상황에 대해 생각을 해봅시다.

- 삭제된 게시물
  - 표시한다 (가정 **A**)
  - 표시안한다 (가정 **B**)

가정 **A**일때, 사용자 입장에서 보면 표시하는게 맞을까요? 삭제된 게시물은 사용자 입장에서보면 확인 할수 없는 게시글일 뿐더러 불필요한 데이터 연동비용만 증가하게 되는 데이터라 생각이 됩니다. 실제로 이렇게 하는 웹페이지도 있지만 저는 장기적으로 볼때는 그다지 좋다고 생각하지는 않습니다.

가정 **B**일때, 그럼 게시글 **1-1**, 게시글 **1-1-1**은 어떻게 처리를 해야 할까요? 그리고 두 번째 상황에 대하여 사용자의 행위는 폴더를 삭제했는데 결과는 폴더와 파일들이 같이 삭제된 상황입니다. 그럼 이와 같은 현상들이 발생하는 근본적인 원인이 뭘까요? 바로 답변형 게시판 구조의 종속성 문제 즉, 트리 구조의 문제점에서 기인을 한다 생각합니다.

트리구조는 **CRUD**에 대하여 **R**의 경우에는 빠른 조회가 가능할 뿐만 아니라 탐색과정에서 별다른 프로세스를 요구하지 않습니다. 하지만, **C**, **D**, **U**같은 경우에는 별도의 프로세스를 요구를 합니다. 바로 노드 재배치 프로세스를 요구합니다.

이러한 재배포치 프로세스에 대해 설명을 하자면, 트리는 모든 노드가 연결되어 있습니다. **D**를 예시로 설명하자면, **DELETE** 연산은 노드의 삭제를 의미하고 노드간의 체이닝이 끊어짐을 의미합니다. 그리고 이 체이닝이 끊어짐에 따라 트리가 여러개로 나뉘게 됩니다. 그리고, 트리를 다시 원래의 **1**개로 만들어야하기 때문에 서로 나뉘어진 트리를 병합하는데 이 프로세스가 노드 재배포치 프로세스 입니다.

그럼 가정 **B**와 같은 상황일 때 어떻게 해야 할까요? 분리된 게시판 트리를 어디에 배치해 두어야 할까요? 배치할 마땅한 장소가 있을까요? 저는 없다 생각했습니다. 그래서 자식게시글 또한 삭제되게 했습니다(실제로는 삭제되지 않고 삭제되었다 표시가 됩니다). 그리고 우리는 모두 이러한 시스템에 대해 이미 알고 있고 암묵적으로 동의한 상태라 생각했습니다.

## 3-c. 게시글 검색

"1" 으로 검색하신 결과

게시글아이디	제목	작성자	작성일자	조회수
805	tjhahsgshg	nickName1	2025-02-12	4
785	forTestTitle1	nickName2	2025-01-29	442
781	gk1	nickName1	2025-01-26	13
778	1	nickName3	2025-01-26	1
767	frbTitle231	nickName1	2025-01-18	0
757	frbTitle221	nickName1	2025-01-18	0
755	frbTitle219	nickName1	2025-01-18	0
754	frbTitle218	nickName1	2025-01-18	0

**설명** : **ERD**를 보시면 아시겠지만, 테이블이 서로 나뉘어져 있습니다. 페이지를 여태 한 테이블을 기준으로 해왔는데 전체 테이블을 대상으로 한다니 상황이 막막하기만 하지만 여러 테이블을 합쳐 한 개의 테이블로 다루면 페이지가 되지 않을까 생각을 했습니다.

하나의 테이블로 다루는 **SQL**문을 저는 **UNION ALL** 연산을 이용했습니다. **UNION ALL**은 중복허용 및 공통컬럼필요 라는 특징을 가지고 있는데, 각 게시판 테이블은 구조가 비슷하니 공통컬럼만 가져왔습니다.

하지만 **UNION ALL**을 쓰면 **PK**가 중복되는 경우가 생기니 **PK**중복에 대한 설명은 다음과 같습니다.

각 게시판 **PK**가 중복이 되는 경우에는, 각 테이블에 카테고리라는 컬럼을 추가해 해결했습니다. 서버는 클라이언트로부터 **PK**와 카테고리를 받고 카테고리를 통해 어느 테이블인지, **PK**를 통해 테이블의 어느 데이터인지를 식별할 수 있습니다.

### 3-d. 대댓글(중첩페이징)

The screenshot displays a web application interface for managing replies. It consists of two main sections, each with a yellow header bar. The top section has a header bar with the text '답변2' and '2025-03-29'. Below the header bar is a search bar with the text '이름' and '2025-03-29'. The search bar has a green button labeled '검색' and a green button labeled '목록'. Below the search bar is a list of replies, each with a green button labeled '댓글'. The bottom section has a header bar with the text '답변1' and '2025-03-29'. Below the header bar is a search bar with the text '이름' and '2025-03-29'. The search bar has a green button labeled '검색' and a green button labeled '목록'. Below the search bar is a list of replies, each with a green button labeled '댓글'. The interface also includes a pagination control at the bottom with a green button labeled '1' and a green button labeled '이동'.

**설명** : 페이징안에 페이징이 존재하는 구조입니다. 일반적인 페이징은 부모**PK**가 필요가 없으나 중첩 페이징은 부모**PK**등이 필수로 필요합니다.

왜냐하면 자식 데이터가 부모 데이터 안에 포함되기 때문입니다.

**구현** : 서버랑 **DB**쪽은 패러미터 받은걸로 기존의 페이징 처리를 하면 됩니다. 클라이언트에서 데이터를 받아서 동적으로 생성하고 보여줘야 하기 때문에 메인구현은 클라이언트가 담당합니다. 또한, 클라이언트는 서버로 데이터 보낼 때, 부모의 **PK**등을 보내야합니다. 데이터 동기화는 필수입니다. 왜냐하면 서버로부터 부모 데이터가 도착하지 않았는데 해당 부모 데이터 안에 자식 데이터를 넣을 수는 없기 때문입니다.

## 4-a. 내가 쓴 글 확인

마이페이지

회원정보 수정	내가 쓴 게시물 전체 ▾
내가 쓴 게시물	
내가 쓴 댓글	
탈퇴	
	자유게시판   게시물작성   2025-02-13
	자유게시판   게시판   2025-02-13
	자유게시판   tjhahgshgl   2025-02-12
	자유게시판   sdfhdsfhs   2025-02-12
	자유게시판   fhadfhdff   2025-02-12
	자유게시판   agasfhfasdf   2025-02-12
	자유게시판   adgad   2025-02-12
	자유게시판   asdasdddddddddddd   2025-02-12
	자유게시판   gk1   2025-01-26
	자유게시판   허   2025-01-26
	자유게시판   안녕하세요   2025-01-21
	자유게시판   frbTitle234   2025-01-18
	자유게시판   frbTitle233   2025-01-18
	자유게시판   frbTitle232   2025-01-18
	자유게시판   frbTitle231   2025-01-18
	페이지 이동 : <input type="text"/> 이동 1 2 3 4 5 6 7 8 9 10 다음

**설명** :

- 회원정보 수정 : 회원정보를 수정할 수 있습니다. 유저아이디는 **PK**이므로 수정이 안되게 하고 비밀번호나 전화번호, 이메일 등을 수정할 수 있습니다.



- 내가 쓴 게시글 및 댓글 : 내가 쓴 게시글이나 댓글을 클릭시 해당 게시판으로 이동합니다.

**회원정보 수정 구현** : 이전과 같은 비밀번호, 전화번호, 이메일등을 사용하는지를 체크합니다. 전화번호나 이메일 등은 해당 데이터를 사용하는 다른 사람이 없는지를 체크합니다.

**게시글 및 댓글이동 구현** : 요청이 마이페이지에서 왔는지 아니면 일반게시판이나 검색에서 왔는지에 대한 판별 코드가 필요합니다. 이에 대해 자세한 설명을 하자면 다음과 같습니다.

우선 일반게시판을 통해 오든 검색에서오든 마이페이지에서 오든 보는 화면은 같습니다. 다만 마이페이지일 경우에는 다른 기능을 하게 만들어야 합니다.

마이페이지에서 해당 게시글이나 댓글을 누르면 스위치가 동작한다는 표시(**ON**)를 브라우저 세션에 해놓고, 브라우저(**JSP**)에서 데이터를 로딩할 때, 해당 **ON**표시를 체크를 합니다. **ON**표시가 없으면 일반적인 루트(게시글 및 검색)를 통해 온것이고 **ON**표시가 있으면 마이페이지로부터 온 것이라 할 수 있습니다. **ON**표시를 확인했으면 해당 서버에서 받은 페이지를 가지고 데이터 로딩이 끝날 때, 해당 페이지를 클릭하는 기능을 넣었습니다.

이렇게 추가적인 클릭기능을 넣은이유는 기존의 기능은 그대로 유지하되, 해당 기능만 추가하여 사용하기 위해 위와같이 클릭하는 기능을 넣었습니다.

그다음, 한가지 중요한 처리를 해야하는데, 바로 **ON**표시를 **OFF**표시로 해놓아야 합니다. **OFF**표시로 하지 않으면 브라우저는 모든 경로가 전부 마이페이지로 온 거로 간주하기 때문입니다.

조금 더 이해를 돕기 위해 비슷한 예시를 들어 설명을 하자면, 마치 화장실을 이용하는 과정과 비슷합니다. 화장실을 이용하기 위해서 해당 화장실칸이 잠금이 되지 않았는지(**ON**표시확인) 잠금이 되지 않았으면 잠금장치를 잠그고 볼 일을 보고 난뒤 다시 잠금을 풀고(**OFF**표시) 다시 와도 방금 예시의 프로세스들이 반복되는 행위와 비슷하다고 생각하시면 됩니다.

**PK로 페이지를 구하는 연산 방법** : 위의 과정에서 서버는 해당 게시글이나 댓글의 **PK**를 통해 페이지를 구해야 합니다. **ORACLE SQL**에서는 **ROWNUM**을 이용하면 해당 **PK**가 몇 번째에 해당되는지 알 수 있기에 쉽게 알 수 있습니다. 즉, **PK의 순서번호**를 통해 페이지를 구할 수 있습니다.

**1**페이지당 **15**개 표시 기준 과 **0~14**의 표시 기준으로 설명을 하면 다음과 같습니다.

순서번호	페이지
<b>0 ~ 14</b>	<b>1</b>
<b>15 ~ 29</b>	<b>2</b>
<b>30 ~ 44</b>	<b>3</b>

**15**개 표시기준을 변수화(**limCol**로 변수화)를 한뒤에 표시를 하면 다음과 같습니다.

순서번호	페이지
<b>limCol * 0 ~ limCol * 1 - 1</b>	<b>1</b>
<b>limCol * 1 ~ limCol * 2 - 1</b>	<b>2</b>
<b>limCol * 2 ~ limCol * 3 - 1</b>	<b>3</b>

각각의 섹션을 **n( >= 0)**이라 하고 이를 좀더 수식화를 하면 다음과 같습니다.

순서번호 = **limCol \* n + x, (0 <= x < limCol).**

순서번호 / **limCol** = **n** 이므로 **n + 1**한게 바로 페이지 번호가 됩니다.

## 5-a. 글 및 회원관리

### 관리자 페이지

회원목록	회원목록					
공지사항 관리	전체 ▼					
고객센터 답변관리	userid9	nickName9	일반회원	세부정보	정지상태아님	탈퇴
	userid8	nickName8	의사회원	세부정보	정지상태아님	탈퇴
	userid7	nickName7	일반회원	세부정보	정지상태아님	탈퇴
	userid6	nickName6	의사회원	세부정보	정지상태아님	탈퇴
	userid5	nickName5	일반회원	세부정보	정지상태아님	탈퇴
	userid4	nickName4	의사회원	세부정보	정지상태아님	탈퇴
	userid3	nickName3	일반회원	세부정보	정지상태아님	탈퇴
	userid2	nickName2	의사회원	세부정보	정지상태아님	탈퇴
	userid1	nickName1	일반회원	세부정보	정지상태아님	탈퇴
	페이지 이동 : <input type="text"/> 이동 1					

### 설명 :

- 회원 목록
  - 드롭다운식으로 전체, 질문, 자유게시판등으로 보기가 가능합니다.
  - 세부정보 : 해당 유저 정보가 나타납니다.
  - 정지상태관리 : 해당 유저의 정지상태를 결정 -> 보는건 가능하지만 글쓰기등이 제한됩니다.
  - 탈퇴 : 해당 유저를 탈퇴시키는 기능입니다.
- 공지사항 및 고객센터 답변 관리 : 각 게시판의 공지사항 및 고객센터 답변리스트가 표시되며 클릭하면 해당 공지사항으로 이동합니다.

