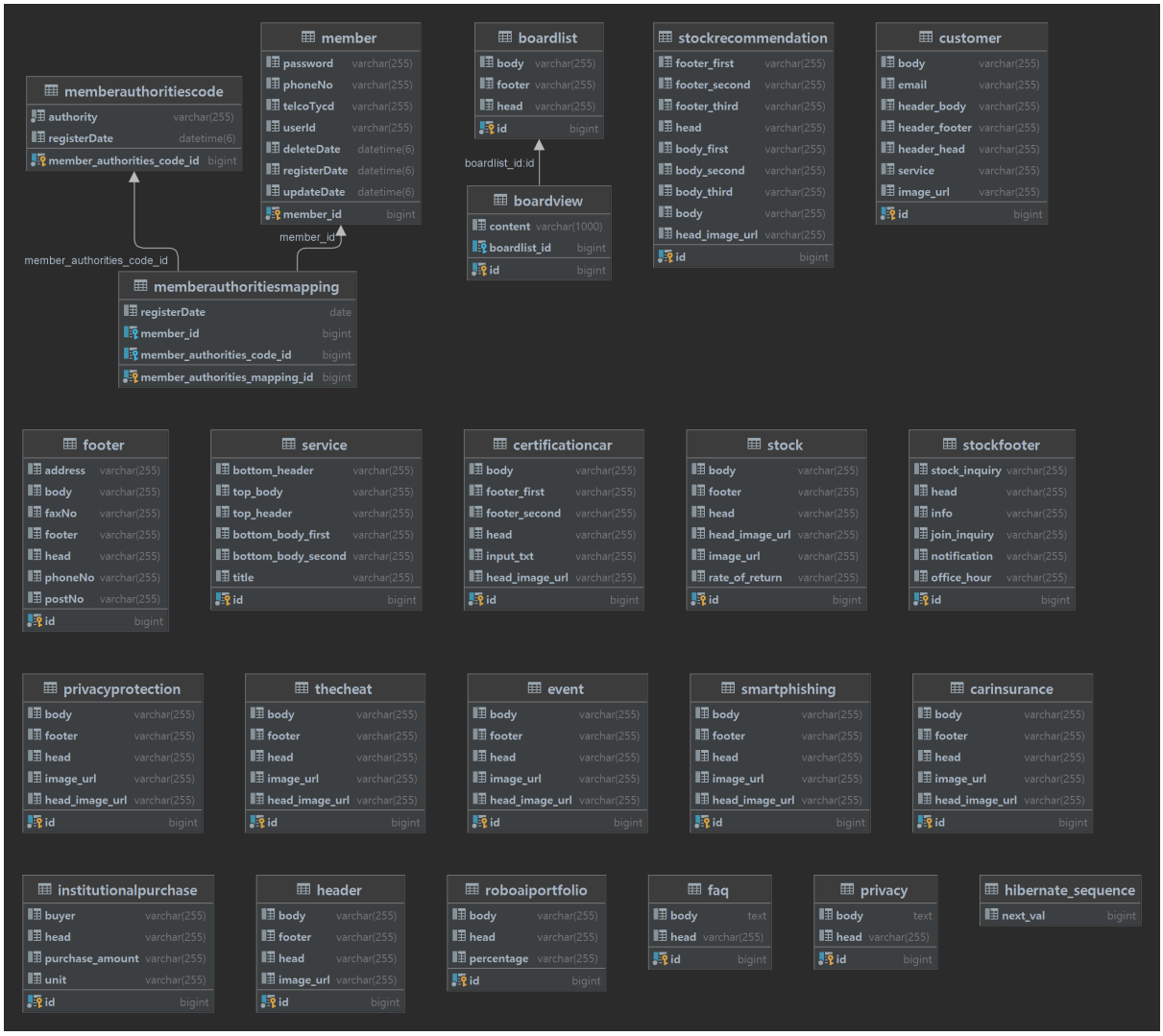
콕뱅크 (인턴_하윤)

Code Review

Database (table 22개)



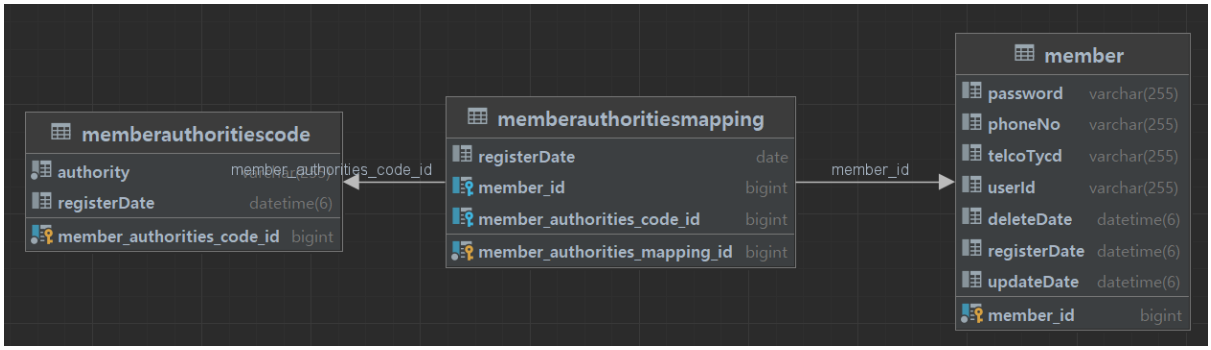
1. `boardlist` 고객센터 공지사항 목록
2. `boardview` 공지사항 세부 내용
3. `stockrecommendation`, `certificationcar`, `carInsurance`, `smartphisihing`, `privacyprotection`, `thecheat` Tables 는 home(index_customer)Page 카테고리 별로 table 분류
4. `customer`은 고객센터 공지사항, FAQ, 이용 약관 및 정책, 서비스 별 문의 카테고리로 분류
5. `event`는 모든 이벤트 목록 저장(전체, 진행, 종료 분류는?)
6. `faq`, `privacy`, `service`, `boarlist`는 고객센터 카테고리별 데이터 분류
7. `header`, `footer` 모든 페이지 공통 영역 처리 (fragment로 공통 영역 지정)

8. `institutionalpurchase`, `roboaiportfolio` 스탁 페이지 메인 카테고리 데이터 분류
9. `stock`, `stockfooter` 스탁 페이지 header, footer로 분류
10. `member`, `memberauthoritiescode`, `memberauthoritiesmapping`은 Spring Security 설정을 위한 Table 생성

Logic

1. MVC패턴 사용. 비즈니스 로직은 모두 Service 계층에서 처리하고 Controller는 클라이언트/서비스를 연결해주는 중계 역할을 하고, Model은 DB와 연동하여 입/출력 데이터를 다룬다. 각 서비스 별 구분을 짓기 위해 MVC 모두 `common`, `main`, `event`, `customer`, `member`로 나누었다.
2. Entity와 DTO는 JPA로 DB설계를 하였다. DAO와 Mapper는 mybatis를 활용하여 스키마 작성을 하였다. 테이블 연관 관계 매핑이나 테이블 설계는 두 개 기술을 활용하여 한다해서 시도하였다.
3. `boardview` - `boardlist` Table이랑 1:1 매핑하여 공지사항 제목과 내용 연결(개행 문자 '\n'는 Thymeleaf에서 적용이 되지 않으므로 Controller 단에서 model.addAttribute를 하여 프론트단에 replace로 적용. `boardview`의 내용은 Text, Img, Text, Img로 구성되어 있어 Img src를 String으로 처리하여 Text '|' Img src '|' Text '|' Img src로 하나의 String으로 Column 지정하고 Service 단에서 DB에서 불러온 Content를 split("|")으로 분류하여 Controller에서 model.addAttribute로 프론트에 넘겨줌
4. `event`는 전체, 진행중, 종료로 세 개로 분류가 되는데, 분류기준은 날짜로 지정하였음. `eventService`에서 `yyyy.MM.dd` ~ `yyyy.MM.dd` 패턴일 때, LocalDateTime.now()기준으로 시작날짜와 종료날짜 사이일 경우, 진행 중 이벤트 List를 만들어 return, 아닐 경우, 종료된 이벤트 List를 만들어 return함. 전체 이벤트 List는 db에서 select * 조회하여 가져오면 됨. 이벤트 목록 출력은 `th:each`로 List의 객체들을 순회하여 출력하였음. `th:onclick`으로 해당 Event의 id값을 parameter로 받아 Controller에서 Param Event id로 Event 객체를 db에서 가져와 event_view에서 출력해주었음. `event` 테이블의 body 컬럼에 진행중 / 종료로 구분 지어 `th:if`를 사용하여 각 이벤트 상태에 맞게 하단 버튼(이벤트 신청/ 이벤트 종료)을 생성하였음.
5. Spring Security를 활용해보았는데, Id, Password, Authority(Admin, Member)로 나누어져 있음. `member`는 회원 가입된 멤버들의 데이터를 저장하고 있음. Security는 BCrpyt 암호화를 사용하므로, password는 암호화 시켜 저장함.(프론트에서 회원 가입을 통해 암호화 로직을 수행해야 하지만, 회원 가입이 없으므로 Test코드에서 PasswordEncoder를 사용하여 직접 암호화 시켜 INSERT 함.) `memberauthoritiesmapping`는 회원 id와 회원 권한을 연결 시켜주는 역할을 수행함.

   `member` : `memberauthoritiesmapping` : `memberauthoritiescode` 의 테이블 관계는 다음과 같다.



예시로, member_id가 1인 '하윤'은 `memberauthoritiesmapping`에 Role_Admin과 Role_Member 두 개의 권한을 가지고 싶다고 가정한다면,

```
INSERT INTO member(member_id, userId, password, registerdate, phoneNo, telcoTycd)
VALUES (1, '하윤', 'adminkr2022!', '2023-04-16', '01090563442', 'S');

INSERT INTO memberauthoritiescode(member_authorities_code_id, authority, registerdate)
VALUES (1, 'ROLE_ADMIN', '2023-04-16');
INSERT INTO memberauthoritiescode(member_authorities_code_id, authority, registerdate)
VALUES (2, 'ROLE_MEMBER', '2023-04-16');

INSERT INTO memberauthoritiesmapping(member_authorities_mapping_id, member_id, member_authorities_code_id)
VALUES (1, 1, 1);
INSERT INTO memberauthoritiesmapping(member_authorities_mapping_id, member_id, member_authorities_code_id)
VALUES (2, 1, 2);
```

위와 같이 INSERT하여 member_id가 1인 회원이 `memberauthoritiescode`의 Admin, Member의 권한 두 개를 가질 수 있게 된다.

Spring Security

Spring Security 설정은 크게 `SecurityConfigure`, `UserDetailsService`, `Handler`로 나누었다. configure에서는 클라이언트에서 서버로 넘어오는 정보들을 어떻게 가공하고 처리할 것 인지에 대한 설정 값을 다룬다. security는 로그인 인증, 인가 뿐만 아니라 회원 권한에 따른 접근 제한을 두어야 하므로 authorizeRequest에서 접근 페이지에 대한 권한 설정을 해주어야 한다. /css, /index, /img등 static 파일들은 Security의 진영 밖으로 두면 관리하지 않는다. formLogin은 <form> 형식의 로그인으로 처리하는 방식이 기본이지만, ajax로 처리하는 방식을 사용. logout()시 로그인 중인 쿠키 스토리지의 `JSESSIONID`를 삭제하도록 해보았지만, 해당 기능이 정상적으로 작동하지 않아 `memberController`에서 session.invalidate()로 설정된 모든 세션을 삭제하였음.

`UserDetailService`는 `CustomUserDetailService`로 커스텀하여 오버라이딩하였다. 로그인할 때 사용한 UserId로 db에서 로그인 멤버를 조회하여, id, pwd, authority를 가져온다. 회원 '하윤'은 Admin, Member 두개의 권한을 가지고 있으므로 member_id와 매핑된 `memberauthoritiesmapping` 의 id를 찾고, mapping_id로 `memberauthoritiescode`의 id에 연결된 authority를 가져온다. (객체 안의 객체를 한꺼번에 select하는 방법은 결국 못 함)

`Handler`는 로그인 성공/실패 후 필요한 비즈니스 로직을 처리하는 영역이다. `Ajax`에서 `Post`로 요청하면 `HttpServletRequest`에서 가로채 인증처리를 한다. 성공 시 반환할 상태 코드와 이동할 페이지는 `MainController`에서 로그인 시 이동할 페이지를 지정하여("/") `Handler`에서 ResponseBody에 이동할 페이지의 attribute를 담아 전달하였다. 로그인 실패도 마찬가지로 상태 코드 및 에러 메세지를 `Ajax`에 `ResponseBody`로 전달한다. `HttpStatusCode = 200`일 경우 팝업 창으로 로그인 완료를 출력하고, 로그인에 성공 시 `memberController`을 통해 `@GetMapping("/")`로 지정된 리턴 값 /member/security.html로 이동하게 된다. 해당 페이지는 로그인을 통해 권한을 가지고 있는 사용자만 접근할 수 있다. (`SecurityConfig - config`에서 /member/**.hasRole("MEMBER")설정을 통해 Member 권한 이상에게 인증/인가 허용)

`xmlns:sec="http://www.w3.org/1999/xhtml` import를 하여 `sec:authorize=`로 권한에 따른 페이지를 보거나 접근할 수 있다. 또한 `name` 이나 `principal=authorities`로 사용자 이름 또는 권한 정보를 확인할 수 있다. 이를 활용하여 메인 페이지에서 로그인 중일 경우에는 권한을 부여 받았기에 로그인 버튼이 로그아웃으로 보인다. `<a class="nav-link" sec:authorize="hasAnyRole('MEMBER', 'ROLE_MEMBER', 'ROLE_ADMIN')">` 반면에, 로그인 버튼은 어느 누구나 볼 수 있기에 `sec:authorize`설정을 하지 않았다. (Spring Security는 기본적으로 id, pwd, 권한 속성 세 개로 인증/인가 절차를 거친다. 사용자 정의에 따라 **`MobileAndUsernamePasswordAuthenticationFilter`**, **`MobileAndUsernamePasswordAuthenticationToken`** 클래스를 활용하여 id, password 뿐만이 아닌 핸드폰 번호 또는 통신사 입력 값까지 인증 절차를 필터링 할 수 있도록 Customize를 할 수 있지만 결국 해내지 못 했다.)

Front(Thymeleaf)

1. 최대한 샘플 화면과 비슷하게 하기 위해 개발자 도구를 활용하여 css를 가져와서 구현하였고, 로그인 접속 및 페이지는 직접 구현해보았다.
2. “화면을 최대한 비슷하게 구현”이라는 요구 사항은 생각보다 어려웠다. 내부 html 코드를 thymeleaf로 리팩토링하고 코드를 간소화하는데 노력했다. 반복된 코드(서비스 배너, 이벤트 목록, 고객센터 배너 별 목록 등)들은 `th:each`로 반복된 코드를 줄였다. 가장 큰 도움을 받은 것은 `th:fragment` 를 활용한 공통 영역 처리였다.
3. 로그인 페이지 구성은 별도로 페이지를 나누지 않고 홈(index_service)페이지에서 한 번에 처리하였다. 퍼블리싱된 html을 조금 수정하여 구현하였다. 위에서 설명하였지만, 처음 메인 페이지 접속 시 인증된 세션 값이 없으므로 로그인 버튼이 나온다. 로그인을 하여 `Spring Security`를 통해 회원 인증이 완료되면 쿠키에 세션 아이디가 저장되고 이 때에는 로그인 버튼이 로그아웃 버튼으로 바뀐다. 다음과 같이 권한에 따른 버튼 기능은 `sec:authorize`로 처리했다. 서비스, 이벤트, 고객센터 메뉴 클릭 시 페이지 이동은 script에서 함수를 구현하여 카테고리 별 `<div th:id>`에 따라 페이지 이동 처리를 하였다. (script 역시 공통 영역으로 처리할 수 있었지만 후순위로 밀려 리팩토링 하지 못했다. Thymeleaf를 사용하는 script의 경우 `<script th:inline="javascript">`를 선언해야 한다.)
4. 이벤트에서 전체, 진행중, 종료 버튼 클릭 시 각각 0, 1, 2로 옵션 값을 두어 Ajax로 `EventController`로 `@RequestParam`에 옵션 값을 같이 서버로 보내 해당 옵션에 따른 서비스 계층에서 처리한 이벤트 목록을 다시 프론트로 보냈다.

아쉬운 점

1. 테이블 설계를 하면서 연관 관계를 활용하여 좀 더 효율적으로 쿼리를 실행할 수 있게 최적화에 대한 고민을 했으면 어땠을까 싶다.
2. 기능 별 카테고리를 나눈다고 테이블을 중구난방으로 생성한 것 같다.
3. DTO는 서비스와 컨트롤러 계층 사이 데이터를 전달할 때, VO는 Mapper(Repository)에서 DB에 데이터를 전달할 때 쓰임을 나누어 설계했어야 했다.
4. 프론트도 Thymeleaf를 사용했지만 좀 더 다양한 기능으로 반복되는 코드를 줄이도록 했어야 했다.
5. 시간 엄수를 지키지 못 했다.