프로젝트 개발하기

담당자 : 이기정
연락처 : 010-6445-7314

목차

Contents

프로젝트 개요

Project Overview

팀 화해 (파이널 프로젝트)

프로젝트 목표

본 프로젝트를 통해 달성하고자 하는 목표는 대규모적으로 기업들이 모여서 화장품을 파는 최고의 이커머스 기업이 되는것이다

프로젝트 배경

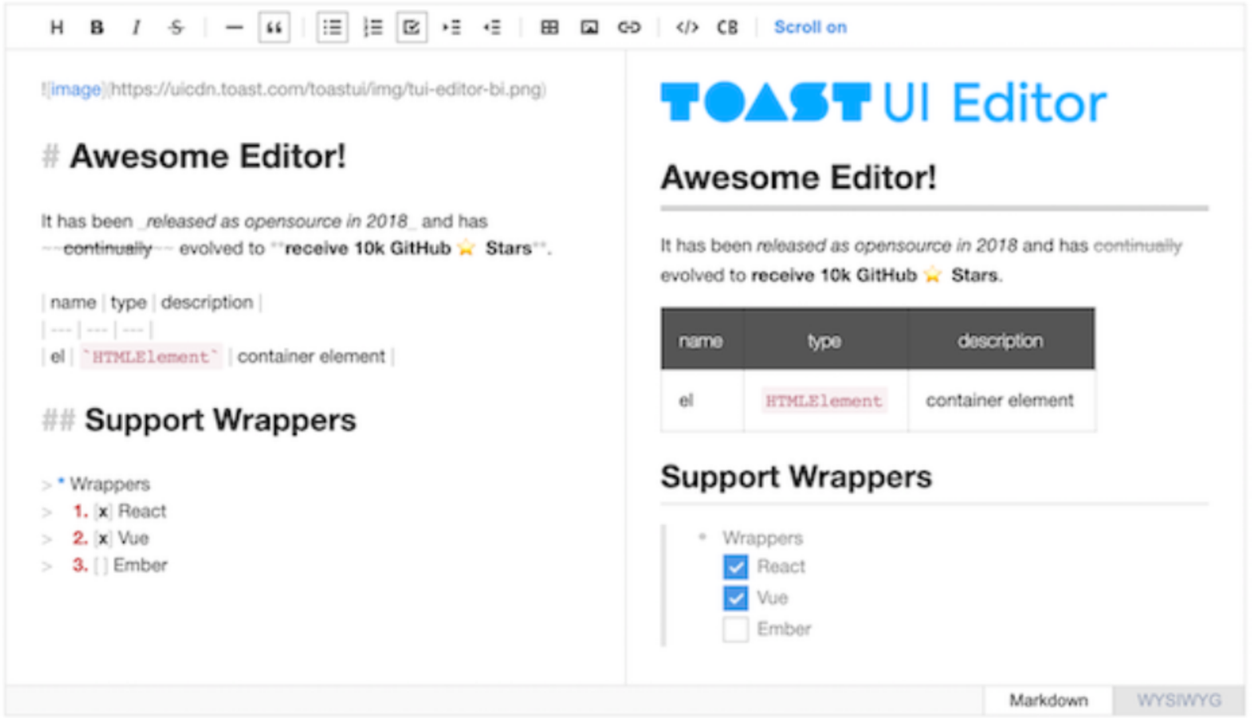
팀화해는 온라인 전자상거래뿐만 아니라 고객들의 취향과 피부타입을 파악해서 알고리즘으로 화장품을 추천해주고 홍보까지 해주는 기능을 만들고싶어서 시작하게 되었다

TOAST UI EDITOR

코드소개에 앞서서 먼저 소개해야될 에디터가 하나있다

1. TOAST UI Editor란?

TOAST UI Editor(이하 'TUI 에디터')는 NHN Cloud에서 개발한 오픈 소스 라이브러리로,
마크다운과 위지윅 방식 모두를 지원하는 무료 에디터이다.



03-1

코드 소개

toast ui editor코드 소개

코드소개(등록 폼 + Toast UI Editor 연동 소개 HTML)

```html
<form th:action="@{/notice/insert}" method="post" id="noticeForm">
    <div>
        <label>제목</label>
        <input type="text" name="boardTitle" required style="width: 80%;">
    </div>

    <br>

    <div>
        <label>게시판 종류</label>
        <select name="boardType" required>
            <option value="1">공지사항</option>
            <option value="2">이벤트</option>
            <option value="4">쿠폰</option>
        </select>
    </div>

    <div>
        <label>내용</label>
        <div id="editor"></div>
        <textarea id="boardContent" name="boardContent" style="display:none;"></textarea>
    </div>

    <!-- 업로드된 이미지 파일명 저장용 -->
    <input type="hidden" name="uploadedFiles" id="uploadedFiles">

    <button type="submit">등록</button>
</form>
```

해당 HTML 코드는 공지사항/이벤트/쿠폰 게시글을 등록하는 입력 폼으로, POST 방식으로 서버에 데이터를 전송합니다.
<input> 태그를 통해 제목을 입력받고, <select> 태그로 게시판 종류를 선택할 수 있으며,
Toast UI Editor로 작성된 본문 내용은 영역에 렌더링됩니다.
에디터 내용을 서버로 전송하기 위해 <textarea id="boardContent">를 숨겨 두었고,
제출 시 자바스크립트를 통해 이 textarea에 HTML 콘텐츠가 복사됩니다.
또한, 이미지 업로드 기능을 지원하기 위해 업로드된 이미지들의 파일명을 저장하는 hidden input(uploadedFiles)을 추가하여, 서버에서 실제 이미지 파일을 처리할 수 있게 구성하였습니다.

코드소개(등록 폼 + Toast UI Editor 연동 소개 JS)

```
const uploadedFilesArr = [];
    // Toast UI Editor 초기화
    const editor = new toastui.Editor({
        el: document.querySelector('#editor'),
        height: '500px',
        width: '90%',
        initialEditType: 'wysiwyg',
        hideModeSwitch: true,
        toolbarItems: [
            ['heading', 'bold', 'italic', 'strike'],
            ['hr', 'quote'],
            ['ul', 'ol'],
            ['table', 'link'],
            ['image']
        ],
        hooks: {
            addImageBlobHook: async (blob, callback) => {
                const formData = new FormData();
                formData.append('image', blob);

                const response = await fetch('/notice/uploadImage', {
                    method: 'POST',
                    body: formData
                });
                const tempUrl = await response.text(); // temp URL 반환
                callback(tempUrl, 'image');

                uploadedFilesArr.push(tempUrl.split('/').pop());
                document.getElementById('uploadedFiles').value = uploadedFilesArr.join(',');
            }
        }
    });

    const form = document.getElementById("noticeForm");
    form.addEventListener("submit", function(e) {
        let html = editor.getHTML();
        document.getElementById("boardContent").value = html;
    })
```

JavaScript에서는 Toast UI Editor를 초기화하여 #editor 영역에 WYSIWYG 방식의 콘텐츠 작성 환경을 제공합니다.
툴바는 글머리, 글자 스타일, 목록, 표, 링크, 이미지 삽입 등으로 구성되어 있으며, addImageBlobHook을 활용해 사용자가 본문에 이미지를 삽입하면 해당 이미지를 FormData로 서버(/notice/uploadImage)에 비동기 전송합니다.
서버로부터 반환된 이미지 URL은 에디터에 삽입되고, 해당 이미지의 파일명은 자바스크립트 배열에 저장됩니다. 이 파일명 목록은 쉼표로 연결되어 hidden input(uploadedFiles)에 설정되어 함께 서버에 전송됩니다.
또한, 폼 제출 시 에디터 내부의 HTML 콘텐츠를 getHTML()로 추출하여 <textarea name="boardContent">에 삽입함으로써 본문 데이터를 정상적으로 전송할 수 있도록 처리하였습니다.

코드소개 (Controller)

```java
// 리스트 페이지
    @GetMapping("/list")
    public String list(
            @RequestParam(value = "page", defaultValue = "1") int currentPage,
            @RequestParam(value = "keyword", required = false) String keyword,
            @RequestParam(value = "type", defaultValue = "all") String type,
            Model model,
            HttpServletRequest request) {

        // 검색 포함 게시글 수 가져오기
        int listCount = noticeService.getListCountWithSearch(1, keyword, type);

        PageInfo pi = Pagination.getPageInfo(currentPage, listCount, 10);

        List<Board> notices = noticeService.selectBoardListWithSearch(pi, keyword, type);

        model.addAttribute("notices", notices);
        model.addAttribute("pi", pi);
        model.addAttribute("loc", request.getRequestURL());
        model.addAttribute("keyword", keyword);
        model.addAttribute("type", type);

       return "notice/list";
    }
```

해당 컨트롤러는 게시글 작성 요청을 처리하며, 클라이언트에서 전달된 이미지 파일명(uploadedFiles)이 존재할 경우,
게시글 본문에 포함된 임시 이미지 경로(/temp/)를 실제 게시용 경로(/notice/)로 변경하여 저장합니다.
또한, 이미지 업로드를 위한 /uploadImage 엔드포인트에서는 사용자가 본문에 삽입한 이미지를 MultipartFile로 수신하여 UUID 기반의 파일명을 생성하고,
서버의 임시 폴더(c:/uploadFilesFinal/temp)에 저장한 뒤, 클라이언트에 접근 가능한 이미지 URL을 반환하여 에디터에 즉시 삽입되도록 처리합니다.

코드소개 (이미지 표시 관련 처리 소개)

```java
// 업로드 (임시 폴더)
    @PostMapping("/uploadImage")
    @ResponseBody
    public String uploadImage(@RequestParam("image") MultipartFile file) throws IOException {
        String tempDir = "c:/uploadFilesFinal/temp";
        String fileName = UUID.randomUUID() + "_" + file.getOriginalFilename();

        File saveFile = new File(tempDir, fileName);
        if (!saveFile.exists()) {
            saveFile.getParentFile().mkdirs();
        }
        file.transferTo(saveFile);

        return "/uploadFilesFinal/temp/" + fileName;
    }
```

Toast UI Editor에서 이미지를 삽입하면 /uploadFilesFinal/temp/파일명 형태의 경로로 본문에 임시 저장됩니다. 게시글 작성 완료 시 Controller에서 이 경로를 /notice/로 치환하고,
이미지 파일도 실제 notice 디렉토리로 이동되기 때문에, 게시글 상세 화면에서 <img> 태그가 올바른 경로(/uploadFilesFinal/notice/파일명)를 참조하게 되어 사용자가 등록한 이미지를 바로 확인할 수 있습니다.

코드소개 (Service 코드 소개 (이미지 파일 이동 및 DB 저장))

```java
public void insertNotice(Board board, String uploadedFiles, HttpSession session) {
        User user = (User)session.getAttribute("loginUser");
        board.setUserNo(user.getUserNo());

        // 게시글 저장 → boardId 생성
        noticeMapper.insertBoard(board);
        int boardId = board.getBoardId();

        if (uploadedFiles != null && !uploadedFiles.isEmpty()) {
            String[] files = uploadedFiles.split(",");
            String tempDir = "c:/uploadFilesFinal/temp";
            String finalDir = "c:/uploadFilesFinal/notice";

            File noticeFolder = new File(finalDir);
            if (!noticeFolder.exists()) {
                noticeFolder.mkdirs(); //  ! 폴더 생성
            }

            for(String fileName : files){
                if(fileName == null || fileName.trim().isEmpty()) continue;

                File tempFile = new File(tempDir, fileName);
                File finalFile = new File(finalDir, fileName);

                // Windows에서 renameTo 실패할 경우 대비
                if (!tempFile.renameTo(finalFile)) {
                    java.nio.file.Path source = tempFile.toPath();
                    java.nio.file.Path target = finalFile.toPath();
                    try {
                        java.nio.file.Files.copy(source, target);
                        tempFile.delete();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }

                // DB 저장
                Attachment attm = new Attachment();
                attm.setAttmName(fileName.substring(fileName.indexOf("_")+1));
                attm.setAttmRename(fileName);
                attm.setAttmPath("/uploadFilesFinal/notice");
                attm.setPositionNo(boardId);
                noticeMapper.insertAttachment(attm);
            }
        }
    }
```

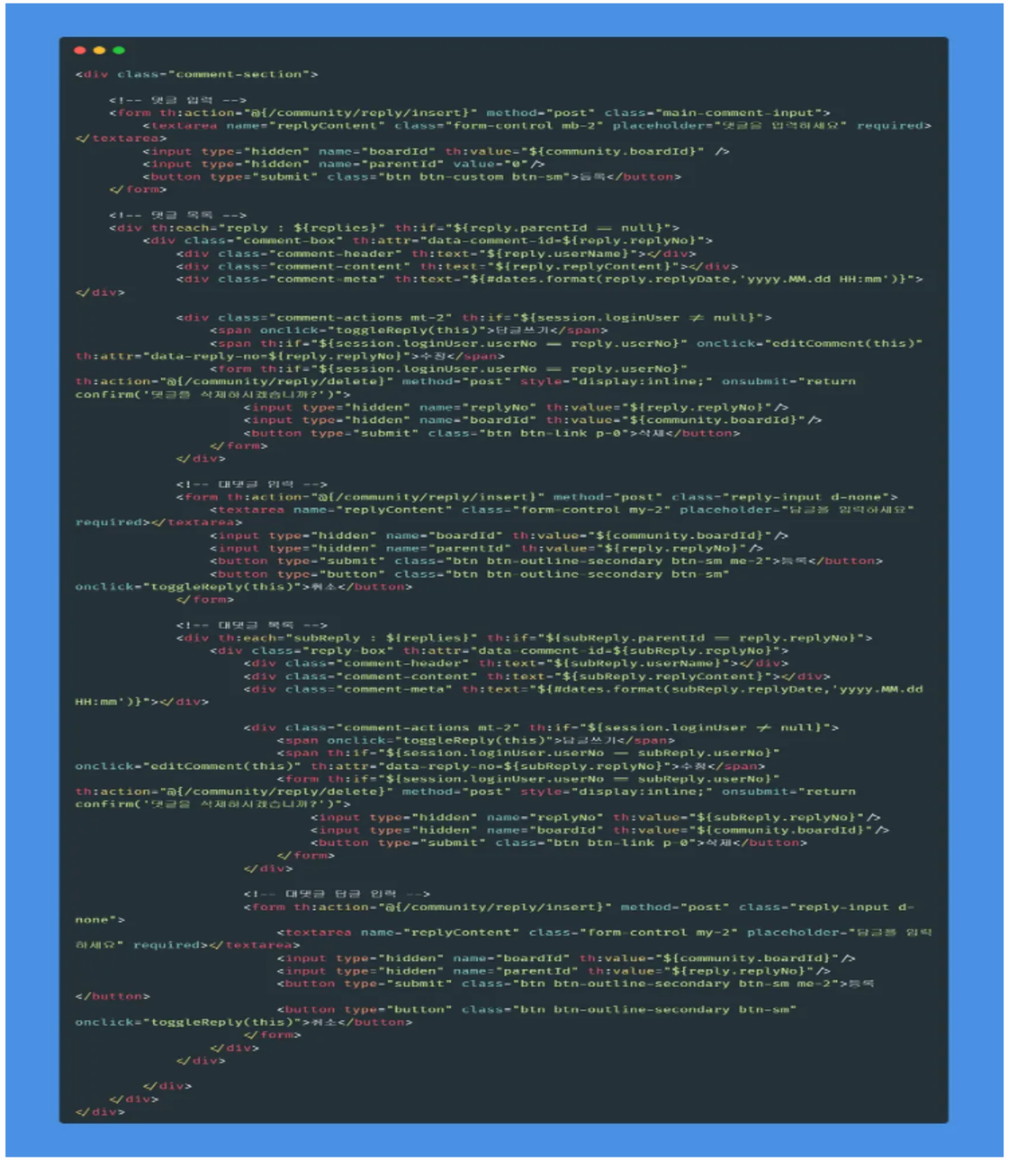
서비스 계층에서는 세션에서 로그인한 사용자의 번호를 Board 객체에 주입하고, 게시글을 먼저 저장하여 boardId를 생성합니다. 이후, 전달받은 이미지 파일명이 존재하면,
임시 디렉토리에 저장된 이미지들을 최종 디렉토리 (c:/uploadFilesFinal/notice)로 이동시키고,
실패 시에는 복사 방식으로 처리하여 안정성을 확보합니다.
각 이미지 파일은 Attachment 객체로 변환되어 원본 이름, 서버 저장 이름, 경로, 게시글 번호 등의 정보를 포함하여 DB에 저장되며, 게시글 본문에서 해당 이미지가 정상적으로 출력될 수 있도록 합니다.

03-2

코드 소개

댓글 코드 소개

코드소개 (댓글/대댓글 작성 및 수정 UI 구성)



```
<div class="comment-section">

    <!-- 댓글 입력 -->
    <form th:action="@{/community/reply/insert}" method="post" class="main-comment-input">
        <textarea name="replyContent" class="form-control mb-2" placeholder="댓글을 입력하세요" required>
</textarea>
        <input type="hidden" name="boardId" th:value="${community.boardId}" />
        <input type="hidden" name="parentId" value="0" />
        <button type="submit" class="btn btn-custom btn-sm">등록</button>
    </form>

    <!-- 댓글 목록 -->
    <div th:each="reply : ${replies}" th:if="${reply.parentId == null}">
        <div class="comment-box" th:attr="data-comment-id=${reply.replyNo}">
            <div class="comment-header" th:text="${reply.userName}"></div>
            <div class="comment-content" th:text="${reply.replyContent}"></div>
            <div class="comment-meta" th:text="${#dates.format(reply.replyDate,'yyyy.MM.dd HH:mm')}">
</div>

            <div class="comment-actions mt-2" th:if="${session.loginUser != null}">
                <span onclick="toggleReply(this)">답글쓰기</span>
                <span th:if="${session.loginUser.userNo == reply.userNo}" onclick="editComment(this)"
th:attr="data-reply-no=${reply.replyNo}">수정</span>
                <form th:if="${session.loginUser.userNo == reply.userNo}"
th:action="@{/community/reply/delete}" method="post" style="display:inline;" onsubmit="return
confirm('댓글을 삭제하시겠습니까?')">
                    <input type="hidden" name="replyNo" th:value="${reply.replyNo}" />
                    <input type="hidden" name="boardId" th:value="${community.boardId}" />
                    <button type="submit" class="btn btn-link p-0">삭제</button>
                </form>
            </div>

            <!-- 대댓글 입력 -->
            <form th:action="@{/community/reply/insert}" method="post" class="reply-input d-none">
                <textarea name="replyContent" class="form-control my-2" placeholder="답글을 입력하세요"
required></textarea>
                <input type="hidden" name="boardId" th:value="${community.boardId}" />
                <input type="hidden" name="parentId" th:value="${reply.replyNo}" />
                <button type="submit" class="btn btn-outline-secondary btn-sm me-2">등록</button>
                <button type="button" class="btn btn-outline-secondary btn-sm"
onclick="toggleReply(this)">취소</button>
            </form>

            <!-- 대댓글 목록 -->
            <div th:each="subReply : ${replies}" th:if="${subReply.parentId == reply.replyNo}">
                <div class="reply-box" th:attr="data-comment-id=${subReply.replyNo}">
                    <div class="comment-header" th:text="${subReply.userName}"></div>
                    <div class="comment-content" th:text="${subReply.replyContent}"></div>
                    <div class="comment-meta" th:text="${#dates.format(subReply.replyDate,'yyyy.MM.dd
HH:mm')}"></div>

                    <div class="comment-actions mt-2" th:if="${session.loginUser != null}">
                        <span onclick="toggleReply(this)">답글쓰기</span>
                        <span th:if="${session.loginUser.userNo == subReply.userNo}"
onclick="editComment(this)" th:attr="data-reply-no=${subReply.replyNo}">수정</span>
                        <form th:if="${session.loginUser.userNo == subReply.userNo}"
th:action="@{/community/reply/delete}" method="post" style="display:inline;" onsubmit="return
confirm('댓글을 삭제하시겠습니까?')">
                            <input type="hidden" name="replyNo" th:value="${subReply.replyNo}" />
                            <input type="hidden" name="boardId" th:value="${community.boardId}" />
                            <button type="submit" class="btn btn-link p-0">삭제</button>
                        </form>
                    </div>

                    <!-- 대댓글 답글 입력 -->
                    <form th:action="@{/community/reply/insert}" method="post" class="reply-input d-
none">
                        <textarea name="replyContent" class="form-control my-2" placeholder="답글을 입력
하세요" required></textarea>
                        <input type="hidden" name="boardId" th:value="${community.boardId}" />
                        <input type="hidden" name="parentId" th:value="${reply.replyNo}" />
                        <button type="submit" class="btn btn-outline-secondary btn-sm me-2">등록
</button>
                        <button type="button" class="btn btn-outline-secondary btn-sm"
onclick="toggleReply(this)">취소</button>
                    </form>
                </div>
            </div>

        </div>
    </div>
</div>
```

이 코드는 커뮤니티 게시판의 댓글 및 대댓글 기능을 위한 HTML 구조로, 사용자는 댓글을 입력하고 등록할 수 있으며, 기존 댓글에 대한 답글(대댓글) 작성 또한 가능합니다.

각 댓글은 parentId 값이 0 또는 null인 경우 최상위 댓글로 간주되며, th:each를 통해 목록을 순회하면서 계층적으로 댓글과 대댓글을 출력합니다.

댓글 작성 시에는 숨겨진 boardId와 parentId를 함께 전송하여 서버에서 어떤 게시글의 어떤 댓글에 대한 응답인지 구분할 수 있도록 했습니다.

댓글 수정은 Bootstrap 모달을 활용하여 별도의 팝업 창에서 처리되며, 수정 버튼을 누르면 모달이 열리고 해당 댓글의 내용이 자동으로 입력되어 편리하게 수정할 수 있습니다.

삭제 버튼은 form 태그로 감싸져 있어 서버에 POST 요청으로 삭제 처리되며, 사용자 확인 알림창(confirm)을 통해 실수 방지를 고려했습니다.

코드소개 (댓글 수정 및 답글 입력 인터랙션 처리)

```
function editComment(el) {
        const commentBox = el.closest('.comment-box, .reply-box');
        const commentContent = commentBox.querySelector('.comment-content');
        const currentText = commentContent.textContent.trim();

        const commentId = el.dataset.replyNo || commentBox.dataset.commentId;

        document.getElementById('editTextarea').value = currentText;
        document.getElementById('editCommentId').value = commentId;

        const editModal = new bootstrap.Modal(document.getElementById('editModal'));
        editModal.show();
    }

    function toggleReply(el) {
        const commentBox = el.closest('.comment-box, .reply-box');
        const replyForm = el.closest('.comment-box, .reply-box').querySelector(':scope > .reply-input');
        if (!replyForm) return;

        if (replyForm.classList.contains('d-none')) {
            replyForm.classList.remove('d-none');
            el.style.display = 'none';
            replyForm.querySelector('textarea').focus();
        } else {
            replyForm.classList.add('d-none');
            el.style.display = 'inline';
        }
    }
```

JavaScript 부분에서는 댓글과 대댓글을 수정하거나 답글 입력 창을 토글하는 동작을 처리합니다.

editComment() 함수는 수정 버튼이 클릭되면 해당 댓글 박스를 찾아 내용을 읽어오고, Bootstrap 모달 내부의 textarea와 hidden input에 데이터를 채워 넣은 뒤 모달을 띄웁니다. 이로써 사용자는 원래 내용을 확인하며 편리하게 댓글을 수정할 수 있습니다.

toggleReply() 함수는 "답글쓰기" 또는 "취소" 버튼 클릭 시, 해당 댓글 박스 내부의 .reply-input 영역을 보여주거나 숨깁니다. 또한, textarea에 자동으로 focus가 가도록 처리하여 사용자 경험을 개선했습니다.

이러한 인터랙션 처리 덕분에 사용자 입장에서 댓글/답글 작성과 수정을 매우 직관적으로 수행할 수 있습니다.

코드소개 (댓글 등록 Controller 코드 소개)

```
// 댓글 등록
    @PostMapping("/reply/insert")
    public String insertReply(Reply reply, HttpSession session) {
        // 로그인 사용자 세팅
        reply.setUserNo(((User) session.getAttribute("loginUser")).getUserNo());

        // 댓글/대댓글 구분
        if (reply.getParentId() ≠ null && reply.getParentId() == 0) {
            // 일반 댓글
            reply.setParentId(null);
        } else if (reply.getParentId() ≠ null) {
            // 대댓글이면 최상위 댓글 ID 가져오기
            Reply parent = communityService.findReplyById(reply.getParentId()); // 부모 댓글 조회
            if (parent.getParentId() == null) {
                // 부모가 최상위 댓글이면 그대로
                reply.setParentId(parent.getReplyNo());
            } else {
                // 부모가 대댓글이면 최상위 댓글 ID로 교체
                reply.setParentId(parent.getParentId());
            }
        }

        communityService.insertReply(reply);
        return "redirect:/community/detail?boardId=" + reply.getBoardId();
    }
```

해당 코드는 커뮤니티 게시글에서 댓글 또는 대댓글을 등록할 때 실행되는 POST 요청 처리 로직입니다.

먼저 HttpSession에서 로그인된 사용자 정보를 조회하여 Reply 객체에 작성자 번호(userNo)를 주입합니다.
그 다음 댓글이 일반 댓글인지 대댓글인지 판단하기 위해 parentId 값을 기준으로 분기 처리를 합니다:

parentId가 0이면 일반 댓글로 간주하고, DB에서는 null로 처리하여 최상위 댓글로 저장되도록 합니다.

반대로 parentId가 null이 아니면 대댓글로 판단되며, 이 경우 부모 댓글 정보를 조회해서 해당 대댓글이 최상위 댓글에 속한 답글인지, 아니면 이미 다른 대댓글에 달린 중첩 답글인지를 판별합니다.

만약 부모 댓글이 최상위 댓글이면 그대로 저장하고,

부모가 다른 대댓글이라면 그 대댓글이 속한 최상위 댓글 ID를 찾아 연결함으로써 대댓글의 계층 구조를 유지할 수 있게 처리합니다.

마지막으로 communityService.insertReply(reply)를 호출하여 댓글 정보를 DB에 저장한 후, 댓글이 작성된 게시글의 상세 페이지로 리다이렉트합니다.

소감

이번 프로젝트에서는 고객 응대에 초점을 맞춘 게시판 기능을 개발하며 실제 서비스에서 자주 접할 수 있는 커뮤니케이션 도구를 구현해보는 경험을 할 수 있었습니다. 특히 대댓글 기능을 통해 사용자 간의 깊이 있는 상호작용을 지원하려고 노력했으며, UI/UX 측면에서도 직관적이고 접근성이 좋은 구조를 지향했습니다.

에디터 부분은 Toast UI Editor를 사용하여 구현하였는데, 이 라이브러리를 선택한 이유는 마크다운과 WYSIWYG(What You See Is What You Get) 방식을 동시에 지원하여 사용자 편의성이 높다고 판단했기 때문입니다. 실제 구현 과정에서도 기본적인 설정 외에 커스터마이징이 잘 되어 있어 사용자의 요구에 맞는 형태로 수정할 수 있었습니다. 이미지 업로드, 코드 블럭, 링크 삽입 등 다양한 기능을 쉽게 적용할 수 있어서 만족스러웠습니다.

기술적인 부분 외에도, 이번 프로젝트를 통해 프론트엔드와 백엔드 간의 데이터 연동을 보다 깊이 이해하게 되었고, 사용자 입장에서 생각하는 훈련도 많이 되었습니다. 단순히 기능을 구현하는 것을 넘어서, 어떤 방식으로 보여주는 것이 효율적인지, 그리고 응대 게시판이 실제 사용자와 기업 간 소통에서 어떤 역할을 하게 될지를 고민하는 과정이 뜻깊었습니다.

아쉬운 점이라면, 시간이 조금 더 있었다면 알림 기능이나 실시간 업데이트 기능(WebSocket 등)을 추가하여 더 완성도 있는 고객 응대 시스템으로 발전시켜 보고 싶었습니다.

전체적으로 이번 프로젝트는 단순한 기능 구현을 넘어, 실제 사용성과 유저 경험(UX)을 고려한 설계와 개발을 고민하게 해준 뜻깊은 기회였으며, 이후 다른 프로젝트에서도 많은 도움이 될 거라 생각합니다.