개발자 천하늘 포트폴리오

2024

NIMONEMO

직장인 N잡 사이트

* 본 프로젝트는 RESTful API 형식으로 제작되었습니다.

https://github.com/soundspeedskye

https://github.com/myj1217/nimonemo

CHEONHANEUL

TEAM GITHUB

ROLE

*“상상,
그 이상의
개발자”*

Front

- 커뮤니티
  - ► CRUD, 리스트 페이지 구현
- 결제
  - ► 결제 모달, 주문확인 모달 페이지 구현

Back

- 커뮤니티
  - ► CRUD 기능 구현
  - ► 리스트, 페이징 구현
- 파일
  - ► 파일처리, 이미지 미리보기, 썸네일 구현
- 결제
  - ► IMPORT API - 카카오페이 결제 기능 구현

NIMONEMO

프로젝트 기획

01

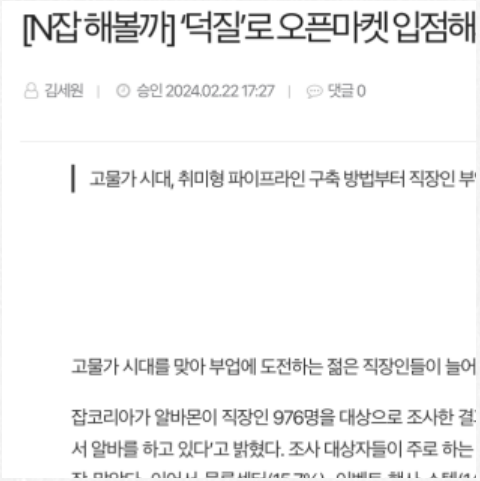
직장인들은
이미
N잡ing

02

가장 높은
선호도는
핸드메이드

03

접근성이
쉬워지는
플랫폼들



04

부업을
넘어선
꿈의 실현

NIMONEMO

idus

01
아이디어스

대한민국 최대 온라인
핸드메이드 쇼핑몰

02
엣시

전세계 최대 온라인
핸드메이드 쇼핑몰
아이디어스의
해외 버전

Etsy

inafactory

03
이너팩토리

크리에이터,
신인 작가들의
굿즈 제작 쇼핑몰

참고
레퍼런스

개발환경

서버

Apache Tomcat

MySQL

개발 툴

IJ

Visual Studio

Redux

spring boot

React

tailwindcss

개발 언어

Java

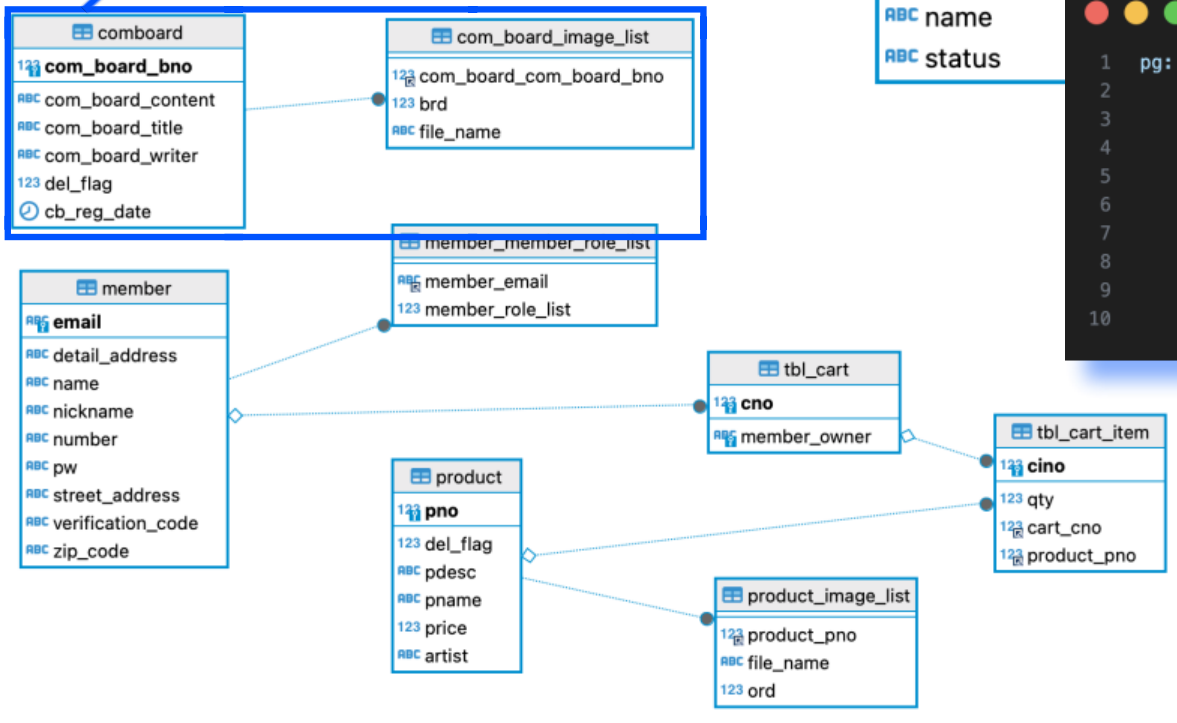
Spring Data JPA

JavaScript

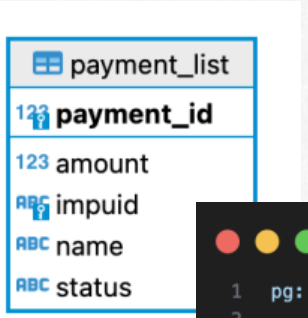
파일 구조

Project Structure
ERD Diagram

community



payment




```
pg: "kakaopay",
      pay_method: "card",
      merchant_uid: `mid_${new Date().getTime()}`,
      name: "테스트 상품",
      amount: 100,
      buyer_email: buyerInfo.email,
      buyer_name: buyerInfo.name,
      buyer_tel: buyerInfo.tel,
      buyer_postcode: buyerInfo.postcode,
      buyer_addr: buyerInfo.address,
```

구현 화면

1 · 2 · 3 · 4

게시글 리스트업 페이징 처리

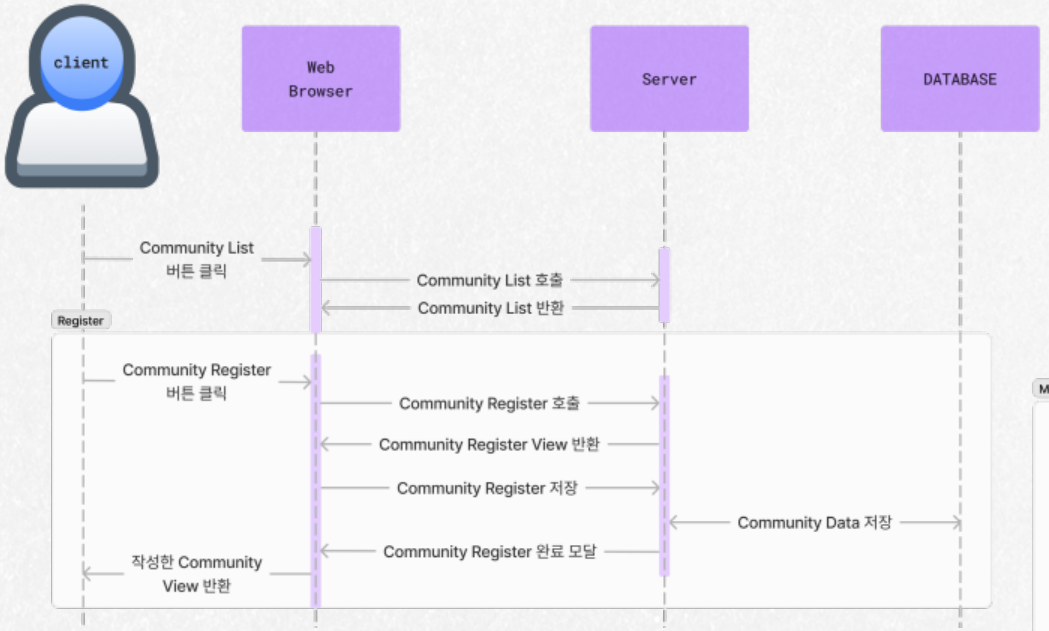
게시글 CRUD

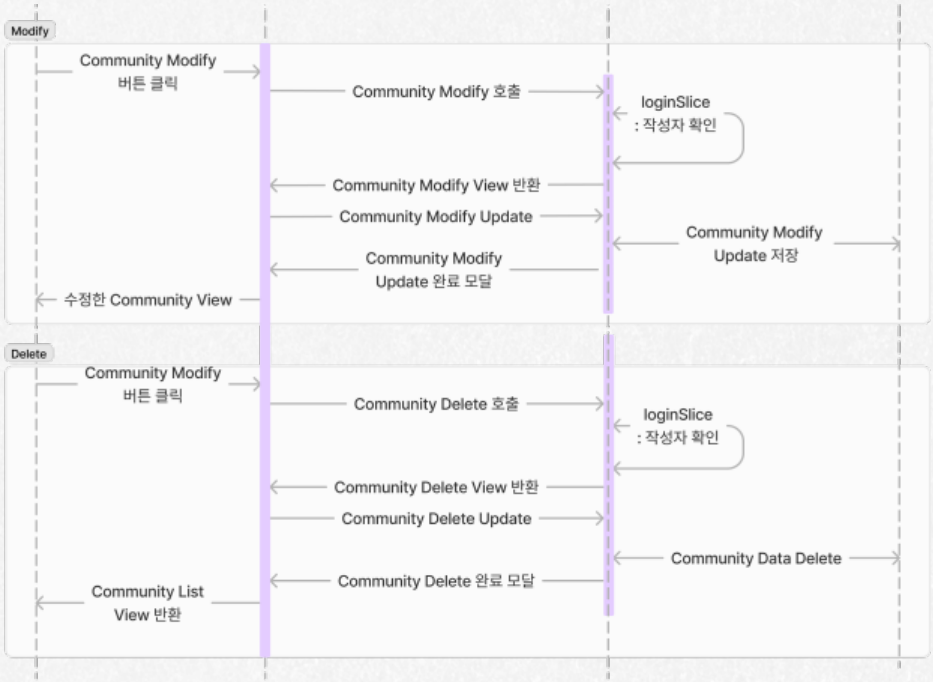
이미지 CRUD

IMPORT API 카카오페이 결제

구현 화면

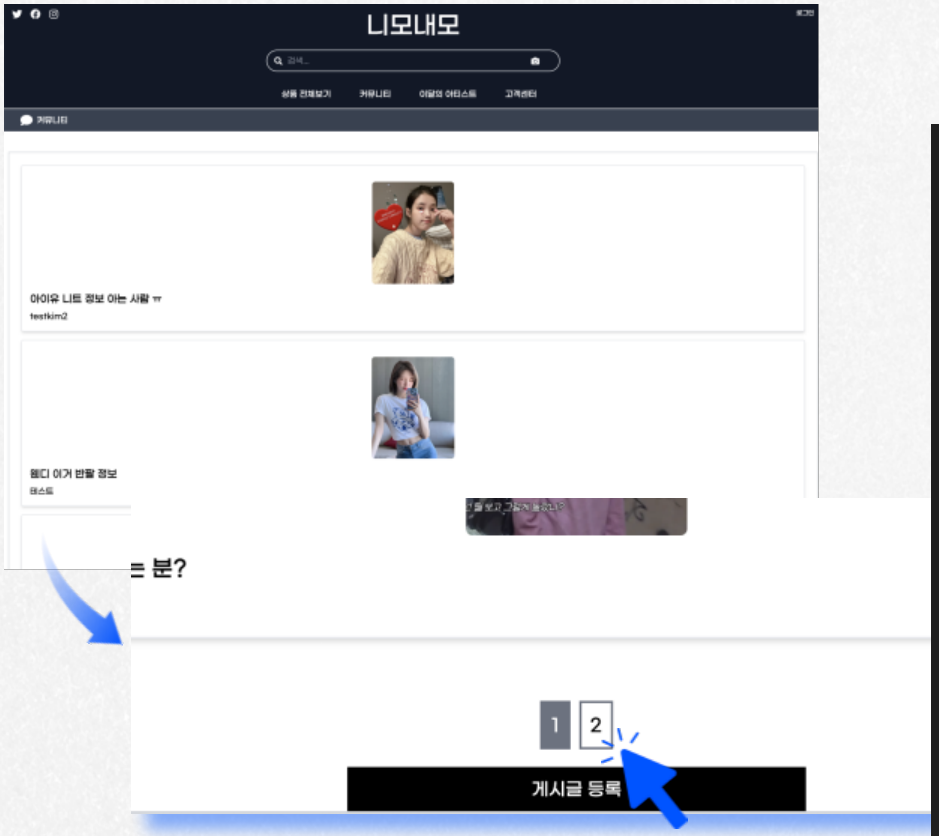
커뮤니티 UML Diagram





구현 화면

1. 게시글 리스트업
페이징처리



1. 최신 등록된 순서대로 리스트에 업데이트

2. 게시글 10개마다 리스트 생성

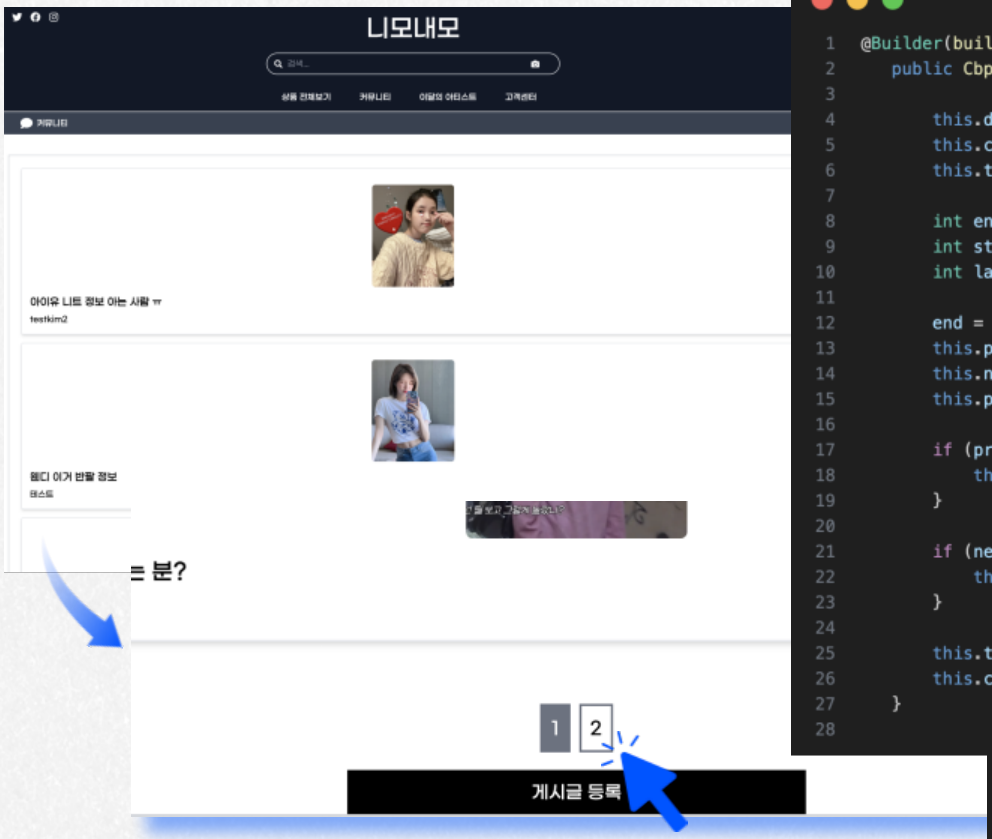
FRONT

```
const initState = {
  dtoList: [],
  pageNumList: [],
  pageRequestDTO: null,
  next: false,
  totalCount: 0,
  prevPage: 0,
  nextPage: 0,
  totalPage: 0,
  current: 0,
};

const ComListComponent = () => {
  const { exceptionHandle } = useCustomLogin();
  const { page, size, refresh, moveToList, moveToRead } = useCustomMove();

  //serverData는 나중에 사용
  const [serverData, setServerData] = useState(initState);

  useEffect(() => {
    comList({ page, size })
      .then((data) => {
        console.log(data);
        setServerData(data);
      })
      .catch((err) => exceptionHandle(err));
  }, [page, size, refresh]);
```

NIMONEMO

구현 화면

1. 게시글 리스트업
페이징처리




```java
@Builder(builderMethodName = "withAll")
    public CbpResponseDTO(List<E> dtoList, CbpRequestDTO cbpRequestDTO, long totalCount) {

        this.dtoList = dtoList;
        this.cbpRequestDTO = cbpRequestDTO;
        this.totalCount = (int) totalCount;

        int end = (int) (Math.ceil(cbpRequestDTO.getPage() / 10.0)) * 10;
        int start = end - 9;
        int last = (int) (Math.ceil((totalCount / (double) cbpRequestDTO.getSize())));

        end = end > last ? last : end;
        this.prev = start > 1;
        this.next = totalCount > end * cbpRequestDTO.getSize();
        this.pageNumList = IntStream.rangeClosed(start, end).boxed().collect(Collectors.toList());

        if (prev) {
            this.prevPage = start - 1;
        }

        if (next) {
            this.nextPage = end + 1;
        }

        this.totalPage = this.pageNumList.size();
        this.current = cbpRequestDTO.getPage();
    }
```

BACK

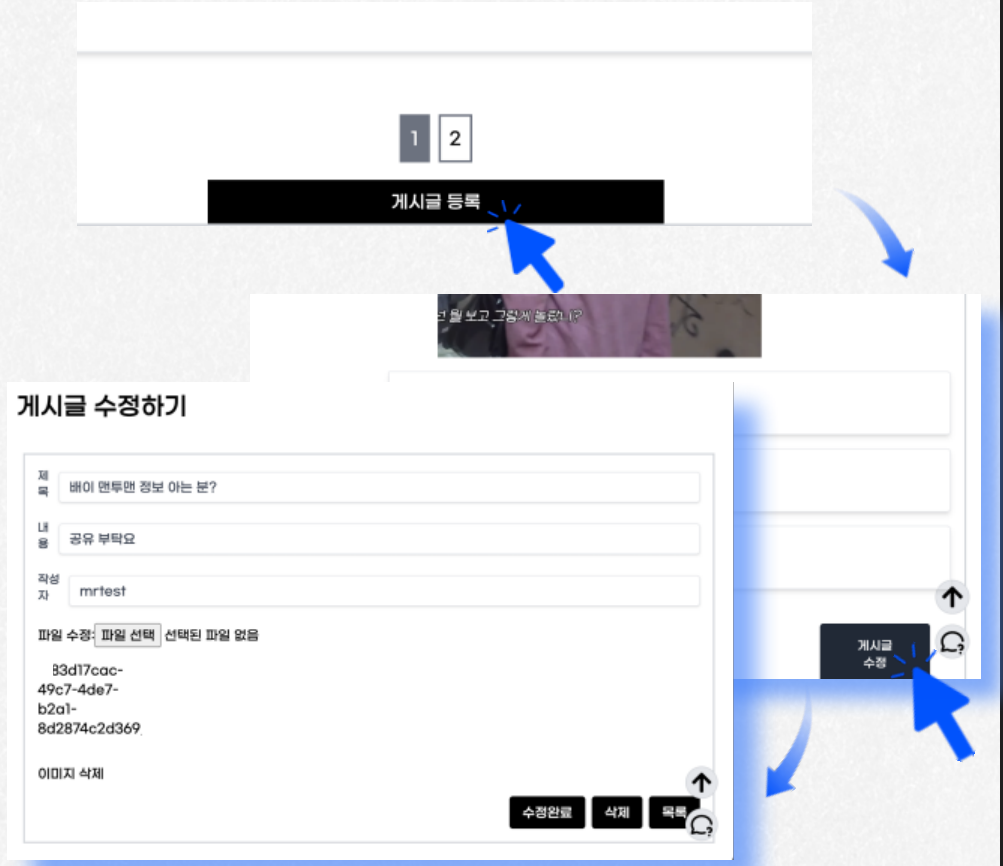
```java
@Override
    public CbpResponseDTO<ComBoardDTO> getCbList(CbpRequestDTO cbpRequestDTO) {
        log.info("getList.............");

        Pageable pageable = PageRequest.of(
                cbpRequestDTO.getPage() - 1,  //페이지 시작 번호가 0부터 시작하므로
                cbpRequestDTO.getSize(),
                Sort.by("comBoardBno").descending());

        Page<Object[]> result = comBoardRepository.selectList(pageable);
```

NIMONEMO

1. 최신 등록된 순서대로 리스트에 업데이트
2. 게시글 10개마다 리스트 생성

구현 화면

2. 게시글 CRUD



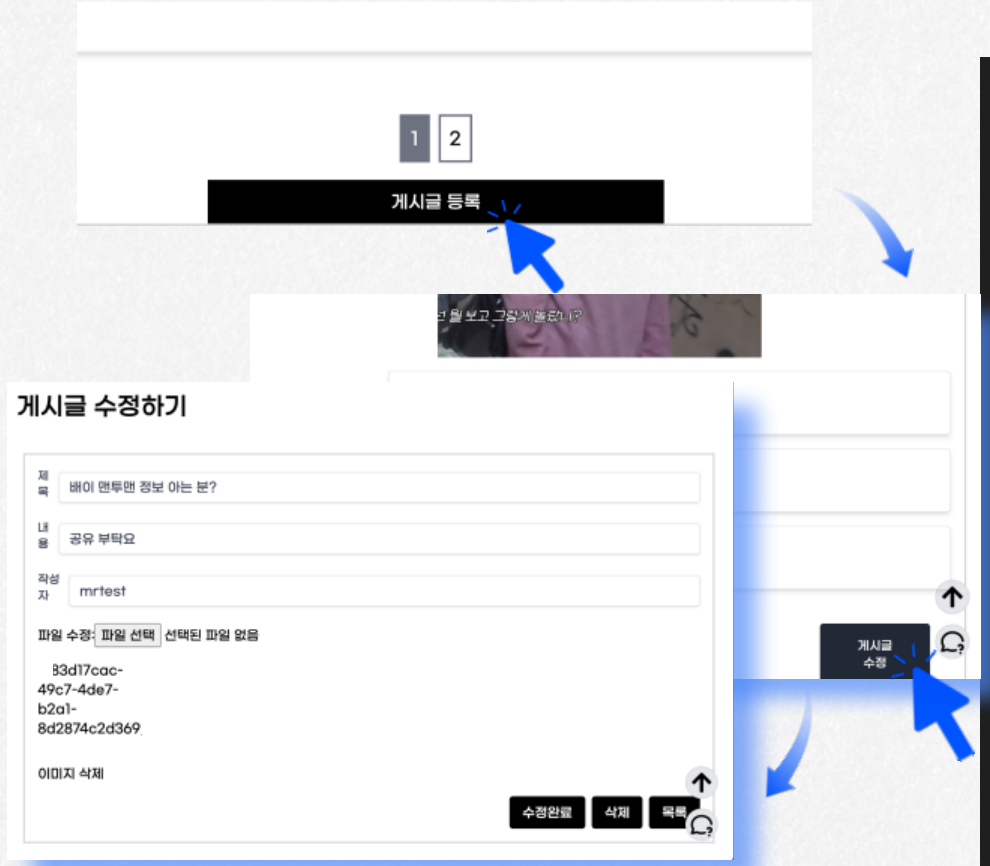
1. 게시글 리스트 하단에 게시글 등록 버튼
2. 게시글 수정 페이지에서 수정 및 삭제 선택

FRONT

```
// 게시글 등록
export const regComboard = async (comboard) => {
  const header = { headers: { "Content-Type": "multipart/form-data" } };
  const res = await jwtAxios.post(`${host}/register`, comboard, header);

  return res.data;
};

// 게시글 상세보기
export const getComBoard = async (cbno, page, size) => {
  const res = await axios.get(`${host}/${cbno}`, {
    params: {
      page: page,
      size: size,
    },
  });

  return res.data;
};

// 게시글 수정
export const modComBoard = async (cbno, data) => {
  const header = { headers: { "Content-Type": "multipart/form-data" } };
  const res = await jwtAxios.put(`${host}/${cbno}`, data, header);

  return res.data;
};

// 게시글 삭제
export const delComBoard = async (cbno) => {
  const res = await jwtAxios.delete(`${host}/${cbno}`);

  return res.data;
};
```

구현 화면

2. 게시글 CRUD



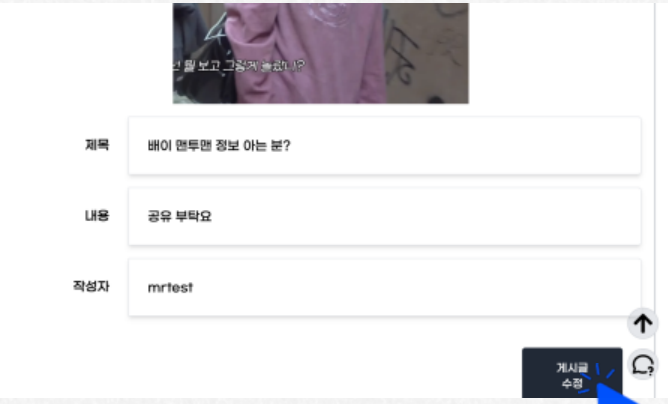
1. 게시글 리스트 하단에 게시글 등록 버튼
2. 게시글 수정 페이지에서 수정 및 삭제 선택

BACK

```java
@Override
    @Transactional
    public void modComBoard(ComBoardDTO comBoardDTO) {
        // 게시글 번호로 게시글 정보를 조회합니다.
        Optional<ComBoard> optionalComBoard =
                comBoardRepository.findById(comBoardDTO.getComBoardBno());

        // Optional에서 게시글 정보를 가져옵니다.
        ComBoard comBoard = optionalComBoard.orElseThrow(() ->
                new IllegalArgumentException("해당 게시글이 존재하지 않습니다."));

        // 게시글 정보를 변경합니다.
        comBoard.changeTitle(comBoardDTO.getComBoardTitle());
        comBoard.changeContent(comBoardDTO.getComBoardContent());

        // 업로드 파일을 초기화합니다.
        comBoard.clearList();

        // 업로드 파일을 추가합니다.
        List<String> uploadFileNames = comBoardDTO.getUploadFileNames();

        if (uploadFileNames != null && uploadFileNames.size() > 0) {
            uploadFileNames.stream().forEach(uploadName -> {
                comBoard.addImageString(uploadName);
            });

            // 변경된 게시글 정보를 저장합니다.
            comBoardRepository.save(comBoard);
        }
    }

    @Override
    public void delComBoard(Long comBoardBno) {
        comBoardRepository.updateToDelete(comBoardBno, true);
    }
```

구현 화면

3. 파일 처리



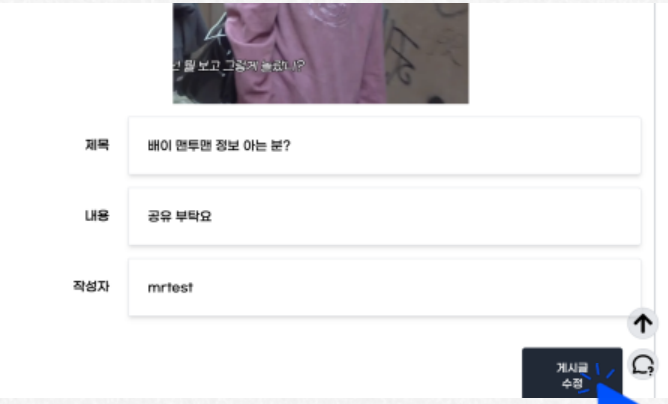
BACK

```java
@PostMapping("/upload")
    public ResponseEntity<String> handleFileUpload(
            @Valid @RequestParam("file")
                MultipartFile file, BindingResult bindingResult) {
        if (bindingResult.hasErrors()) {
            // 유효성 검사에 실패한 경우
            return new ResponseEntity<>("Invalid file", HttpStatus.BAD_REQUEST);
        }

        if (!file.isEmpty()) {
            try {
                // 업로드할 디렉토리 경로 확인 및 생성
                String currentDir = System.getProperty("user.dir");
                File uploadDir = new File(currentDir + UPLOAD_DIR);
                if (!uploadDir.exists()) {
                    uploadDir.mkdirs();
                }

                // 파일 이름 생성 (중복 방지)
                String fileName = StringUtils.cleanPath(file.getOriginalFilename());
                String uniqueFileName = UUID.randomUUID().toString() + "_" + fileName;

                // 업로드된 파일을 서버에 저장
                String filePath = uploadDir.getAbsolutePath() + "/" + uniqueFileName;
                File dest = new File(filePath);
                file.transferTo(dest);

                return new ResponseEntity<>("File '" + fileName + "' uploaded successfully", HttpStatus.OK);
            } catch (IOException e) {
                return new ResponseEntity<>("Failed to upload file", HttpStatus.INTERNAL_SERVER_ERROR);
            }
        } else {
            return new ResponseEntity<>("No file uploaded", HttpStatus.BAD_REQUEST);
        }
    }
```

1. 게시글 등록 시 파일 첨부 가능
2. 게시글 수정 페이지에서 파일 수정 및 삭제 가능

구현 화면

3. 이미지 CRUD



BACK

```java
List<String> uploadNames = new ArrayList<>();

  for (MultipartFile multipartFile : files) {
    String savedName = UUID.randomUUID().toString()
                        + "_" + multipartFile.getOriginalFilename();
    Path savePath = Paths.get(uploadPath, savedName);

    try {
      Files.copy(multipartFile.getInputStream(), savePath);
      String contentType = multipartFile.getContentType();

      if(contentType != null && contentType.startsWith("image")){
        Path thumbnailPath = Paths.get(uploadPath, "s_"+savedName);
        Thumbnails.of(savePath.toFile())
        .size(200, 200)
        .toFile(thumbnailPath.toFile());
      }

      uploadNames.add(savedName);
    } catch (IOException e) {
      throw new RuntimeException(e.getMessage());
    }
  }//end for
  return uploadNames;
}
```

1. 게시글 등록 시 파일 첨부 가능
2. 게시글 수정 페이지에서 파일 수정 및 삭제 가능

NIMONEMO

구현 화면

4. IMPORT API
카카오페이 결제

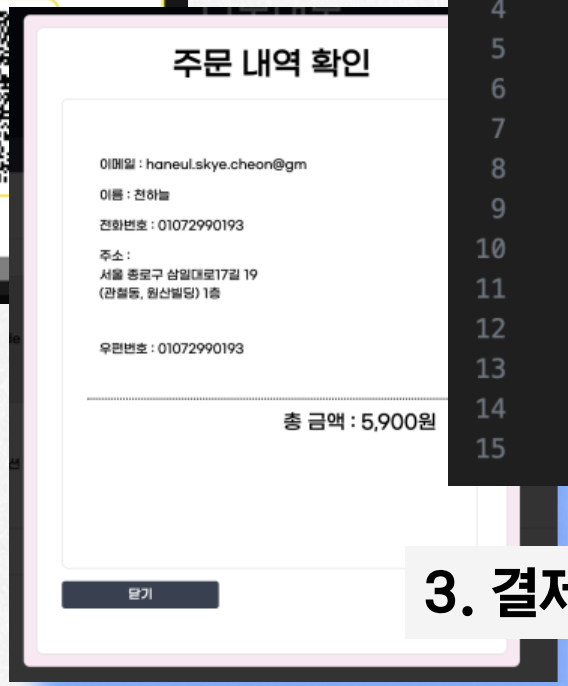
FRONT

```
1  const ClickChargeBtn = (pg_method) => {
2      const { IMP } = window;
3      IMP.init("imp23143410");
4      IMP.request_pay(
5        {
6          pg: "kakaopay",
7          pay_method: "card",
8          merchant_uid: `mid_${new Date().getTime()}`,
9          name: "테스트 상품",
10         amount: 100,
11         buyer_email: buyerInfo.email,
12         buyer_name: buyerInfo.name,
13         buyer_tel: buyerInfo.tel,
14         buyer_postcode: buyerInfo.postcode,
15         buyer_addr: buyerInfo.address,
16         m_redirect_url: "http://localhost:3000/cart",
17       },
18       function (rsp) {
19         // callback
20         if (rsp.success) {
21           alert("결제 성공");
22           closeModal(); // 결제 성공 시 모달을 닫습니다.
23         } else {
24           alert("결제 실패");
25           console.log(rsp);
26         }
```

1. 장바구니에서 선택 상품 주문 시 상품 총 갯수, 합계 금액, 주문자 정보 포함한 모듈 생성
2. 결제하기 클릭 시 카카오페이 API 호출

구현 화면

4. IMPORT API
카카오페이 결제



FRONT

```
function OrderComplete({ totalPrice, closeModal }) {
  const [orderInfo, setOrderInfo] = useState({
    email: "",
    name: "",
    number: "",
    streetAddress: "",
    detailAddress: "",
    zipCode: "",
  });

  const orderState = useSelector((state) => state.loginSlice);

  useEffect(() => {
    setOrderInfo({ ...orderState });
  }, []);
```

3. 결제 완료 후 주문 내역 모듈로 확인 가능

NIMONEMO

마주한 어려움

- **Spring Security**
  "ERROR ACCESS TOKEN"

```
▼ {error: 'ERROR_ACCESS_TOKEN'} i
    error: "ERROR_ACCESS_TOKEN"
```

Database에 테스트 셋업한 더미 데이터들이 보이지 않아 확인한 결과, **Spring Security의 토큰** 기반 인증에 문제가 있음을 확인했다.

SecurityFilterChain 클래스를 수정하고 **JWTCheckFilter에 예외를 추가**하여 문제를 해결했다.

```java
@Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        http.cors(httpSecurityCorsConfigurer -> {
            httpSecurityCorsConfigurer.configurationSource(corsConfigurationSource());
        });
        http.sessionManagement(sessionConfig ->
                        sessionConfig.sessionCreationPolicy(SessionCreationPolicy.STATELESS));
        http.csrf(config -> config.disable());  // csrf 토큰 사용 안함.

        //API 서버로 로그인할수 있게 설정.
        http.formLogin(config ->{
            config.loginPage("/api/member/login");
            config.successHandler(new APILoginSuccessHandler());
            config.failureHandler(new APILoginFailHandler());
        });
```

```java
public class JWTCheckFilter extends OncePerRequestFilter{

    @Override
    //필터링 하지않고 건너뛸 조건을 정의할 수 있는데
    //해당 메서드가 true 를 반환하면 건너뛰고, false 를 반환하면 필터가 작동한다.
    protected boolean shouldNotFilter(HttpServletRequest request) throws ServletException{

        // Preflight요청은 체크하지 않음
        if(request.getMethod().equals("OPTIONS")){
            return true;
        }
        String path = request.getRequestURI();

        //경로의 호출은 체크하지 않음
        if(path.startsWith("/api/member/")) {
            return true;
        }
        if(path.startsWith("/api/products/")) {
            return true;
        }
        if(path.startsWith("/api/comBoard/")) {
            return true;
        }
        return false;
    }
```

마주한 어려움

- **Redux Toolkit**
  Reducer과 Slice의 첫 사용

CartComponent에서 선택한 **상품의 총 개수와 총 금액을 결제까지** 연결하는데 어려움이 있었다.

Redux의 Slice를 이용하여 **useCustomCart라는 hook**을 만들어 cartItem을 효율적으로 관리하고 **map 메서드로 연결하는 과정**을 통해 Redux Toolkit에 대한 이해도를 높일 수 있었다.

```jsx
useEffect(() => {
  if (!Array.isArray(cartItems)) {
    return; // cartItems가 배열이 아니면 처리 중단
  }

  // 선택된 장바구니 항목의 총 개수
  const selectedQty = cartItems.reduce((acc, curr) => {
```

```jsx
export const getCartItemsAsync = createAsyncThunk("getCartItemsAsync", () => {
  return getCartItems();
});

export const postChangeCartAsync = createAsyncThunk(
  "postCartItemsAsync",
  (param) => {
    return postChangeCart(param);
  }
);

const initState = [];
const cartSlice = createSlice({
  name: "cartSlice",
  initialState: initState,

  extraReducers: (builder) => {
    builder
      .addCase(getCartItemsAsync.fulfilled, (state, action) => {
        console.log("getCartItemsAsync fulfilled");
        return action.payload;
      })
      .addCase(postChangeCartAsync.fulfilled, (state, action) => {
        console.log("postCartItemsAsync fulfilled");
        return action.payload;
      });
  },
});

export default cartSlice.reducer;
```

느낀 점

- **React-Router-Dom v6 활용**
- **Axios 이용한 Data API 호출과 연결**

React-Router-Dom v6의 RouterProvider를 사용하고 routes의 path, element를 활용하여 components를 더 용이하게 배열할 수 있었다.

Backend의 Database를 Axios를 이용하여 API 호출하고 연결하는 과정을 경험하게 되었다.

아쉬운점

- **IMPORT API의 더 많은 활용**

특정 간편 결제 서비스가 아닌 여러 카드 결제 서비스까지 제공하기 위해 IMPORT API를 선택했으나 DB와의 연결에 어려움을 겪어 단순한 간편 결제서비스만 해결한 점이 다소 아쉽다.

Corrections

Thank you.

haneul.skye.cheon@gmail.com