중고거래 플랫폼 개발

팀　　명 : 안전거래해조
지 도 교 수 : 양환석 교수님
팀　　장 : 이종민
팀　　원 : 이재혁
박민서
정범규
박정수

2024. 11.
중부대학교 정보보호학과

목 차

6. 별 첨

1. 서론

1.1 연구배경

최근 중고 거래 시장은 경제적 부담을 완화하고 자원의 재활용을 촉진하는 중요한 소비 형태로 자리 잡고 있다. 2023년 기준으로 약 1억 7,300만 건의 중고 거래가 이루어졌으며, 이는 지속 가능한 소비와 환경 보호에 대한 인식이 확산되고 있음을 보여준다. 경제적 효율성과 자원의 순환을 동시에 달성할 수 있는 중고 거래는 소비자들 사이에서 점차 중요한 선택지로 떠오르고 있으며, 특히 플랫폼을 통해 더욱 편리하게 거래할 수 있는 시스템이 필요하다. 이러한 시장 변화는 중고 거래의 중요성을 더욱 부각시키고 있으며, 이를 통한 지속 가능한 소비의 가능성을 확대하고 있다.

1.2 연구 목적 및 주제선정

중고 거래 시장의 급격한 확장은 지속 가능한 소비를 촉진하는 중요한 역할을 하고 있다. 중고 물품의 거래는 자원 재활용을 극대화하고, 폐기물을 줄여 환경 보호에 기여하는 동시에, 소비자들의 경제적 부담을 경감시키는 효과가 있다. 이러한 중고 거래 시장의 급성장 배경을 심도 있게 분석하고, 이를 통해 지속 가능한 소비로의 전환을 도모하는 사회적·경제적 의미를 도출하는 것을 목적으로 한다. 나아가 중고 거래가 소비자들에게 어떤 긍정적 영향을 미치고, 환경에 미치는 장기적 효과를 탐구한다.

중고 거래 시장의 성장과 더불어 소비자들이 신뢰하고 안심할 수 있는 거래 환경을 조성하는 것이 필수적이다. 현존하는 중고 거래 플랫폼들은 다양한 거래 경험을 제공하지만, 여전히 신뢰성과 거래의 편의성 측면에서 많은 개선이 필요하다. 이러한 문제점들을 해결하기 위해 사용자들이 더욱 편리하게 이용할 수 있는 중고 거래 플랫폼의 개발 방향을 제시하고, 플랫폼의 신뢰성을 강화하기 위한 방법을 모색할 것이다. 이를 통해 향후 중고 거래 시장의 발전에 기여할 수 있는 실질적 방안을 도출하고자 하며, 소비자와 환경 모두에게 긍정적 영향을 미치는 지속 가능한 거래 문화를 확립하는 데 이바지하고자 한다.

2. 관련 연구

2.1 JavaScript

JavaScript는 웹 브라우저에서 동작하는 프로그래밍 언어로, 주로 웹 페이지의 동적 동작을 구현하는데 사용된다. HTML이 웹페이지의 구조를, CSS가 스타일을 정의한다면, JavaScript는 그 위에 사용자와의 상호작용, 애니메이션, 데이터 처리 등의 기능을 추가하는 역할을 한다. JavaScript는 웹 브라우저 내에서 실행되며, 사용자가 웹페이지를 조작할 때 실시간으로 반응하는 인터페이스를 구현할 수 있게 한다. 또한, 비동기 처리와 같은 기능을 통해 서버와 데이터를 주고받으며 웹 애플리케이션을 보다 복잡하고 기능적으로 구현할 수 있다. 최근에는 Node.js와 같은 환경을 통해 서버 사이드에서도 JavaScript를 사용할 수 있게 되면서, 프론트엔드와 백엔드 모두에서 활용 가능한 범용적인 언어이다.

2.2 React

React는 Facebook에서 개발한 JavaScript 기반의 프론트엔드 라이브러리로, UI를 컴포넌트 단위로 모듈화하여 재사용성과 유지보수성을 높이는 데 중점을 둔다. 컴포넌트는 특정 기능을 담당하는 독립적인 UI 요소로, 여러 페이지에서 반복적으로 사용될 수 있으며, 이를 통해 개발자는 코드 중복을 줄이고 유지보수를 간편하게 할 수 있다. 또한 React는 가상 DOM(Virtual DOM) 기술을 통해 브라우저의 실제 DOM과 가상 DOM 간의 차이만을 계산하여 효율적으로 UI를 업데이트함으로써 성능을 최적화한다. 이러한 특징 덕분에 React는 빠른 UI 변경이 필요한 애플리케이션에서 뛰어난 성능을 발휘하며, 광범위한 커뮤니티와 생태계를 통해 풍부한 자료와 오픈소스 라이브러리를 제공받을 수 있다. 이러한 장점들로 인해 React는 웹 애플리케이션 개발에서 필수적인 도구이다.

2.3 MySQL

MySQL은 오픈 소스 관계형 데이터베이스 관리 시스템으로, SQL을 통해 데이터를 관리하며 다양한 플랫폼에서 빠르고 안정적으로 동작한다. 데이터를 테이블 형식으로 저장하고, 트랜잭션 관리와 데이터 복제를 통해 안정성과 성능을 높인다. 주로 웹 애플리케이션에서 많이 사용되며, 확장성이 뛰어나 소규모부터 대규모 트래픽까지 처리할 수 있다. MySQL은 사용자 권한 관리와 데이터 암호화를 통해 강력한 보안을 제공한다. 클러스터링 및 고가용성 기능을 통해 대규모 트래픽에도 대응 가능하다. 데이터 분석 도구와 통합되어 분석 작업에 활용되며, 오픈 소스 커뮤니티의 활발한 지원과 지속적인 업데이트가 이루어진다.

2.4 Spring Boot

Spring Boot는 스프링 프레임워크를 기반으로 한 경량 애플리케이션 프레임워크로, 개발자가 더 빠르고 효율적으로 애플리케이션을 개발할 수 있도록 돕는다. 내장된 톰캣, 제티 같은 서버를 제공해 별도의 서버 설정 없이 실행이 가능하다. 의존성 관리와 자동 구성 기능을 통해 복잡한 설정을 줄이고, @SpringBootApplication 같은 어노테이션을 통해 주요 설정을 간단하게 처리한다. 마이크로서비스 아키텍처에서 많이 사용되며, 독립적이고 확장 가능한 애플리케이션을 쉽게 개발할 수 있다. 스프링 부트 스타터 패키지를 통해 필요한 의존성을 쉽게 추가할 수 있고, H2 같은 임베디드 데이터베이스와 통합해 테스트 환경 구성이 간편하다. DevTools를 통한 핫 리로딩 기능으로 개발 편의성을 높이고, 강력한 보안 설정을 제공하여 안정적인 애플리케이션을 구축할 수 있다.

2.5 JSON Web Token

JWT(제이슨 웹 토큰, JSON Web Token)은 JSON 형식으로 사용자 정보를 안전하게 전달하는 방식이다. 서버는 사용자의 인증이 완료되면 JWT를 생성하여 클라이언트에 전달하고, 클라이언트는 이 토큰을 API 요청 시 함께 전송해 인증을 유지한다. JWT는 3가지 부분(헤더, 페이로드, 서명)으로 구성되며, 서명을 통해 변조 여부를 확인할 수 있다. 토큰 기반 인증을 통해 세션 상태를 서버에 저장하지 않고도 사용자를 인증할 수 있어 확장성이 좋다. JWT는 주로 OAuth 2.0과 같은 인증 프로토콜에서 사용되며, 무겁지 않고 쉽게 분산 시스템에서 사용할 수 있다. 클라이언트 측에서 토큰을 저장(쿠키 또는 로컬 스토리지)하고, API 요청 시 이를 헤더에 포함해 전송한다. 유효기간이 포함된 JWT는 만료 후 재발급이 필요하며, 보안을 위해 HTTPS 통신과 함께 사용하는 것이 권장된다.

2.6 Web Socket

웹소켓은 웹 클라이언트와 서버 간의 양방향 통신을 실시간으로 가능하게 하는 통신 프로토콜이다. 웹소켓을 사용하면 클라이언트와 서버가 하나의 지속적인 연결을 통해 실시간으로 데이터를 주고받을 수 있다. 이는 일반적인 HTTP 요청-응답 방식과는 다르게, 연결이 한 번 성립되면 끊지 않고 데이터를 계속 주고받을 수 있다는 특징을 가진다.

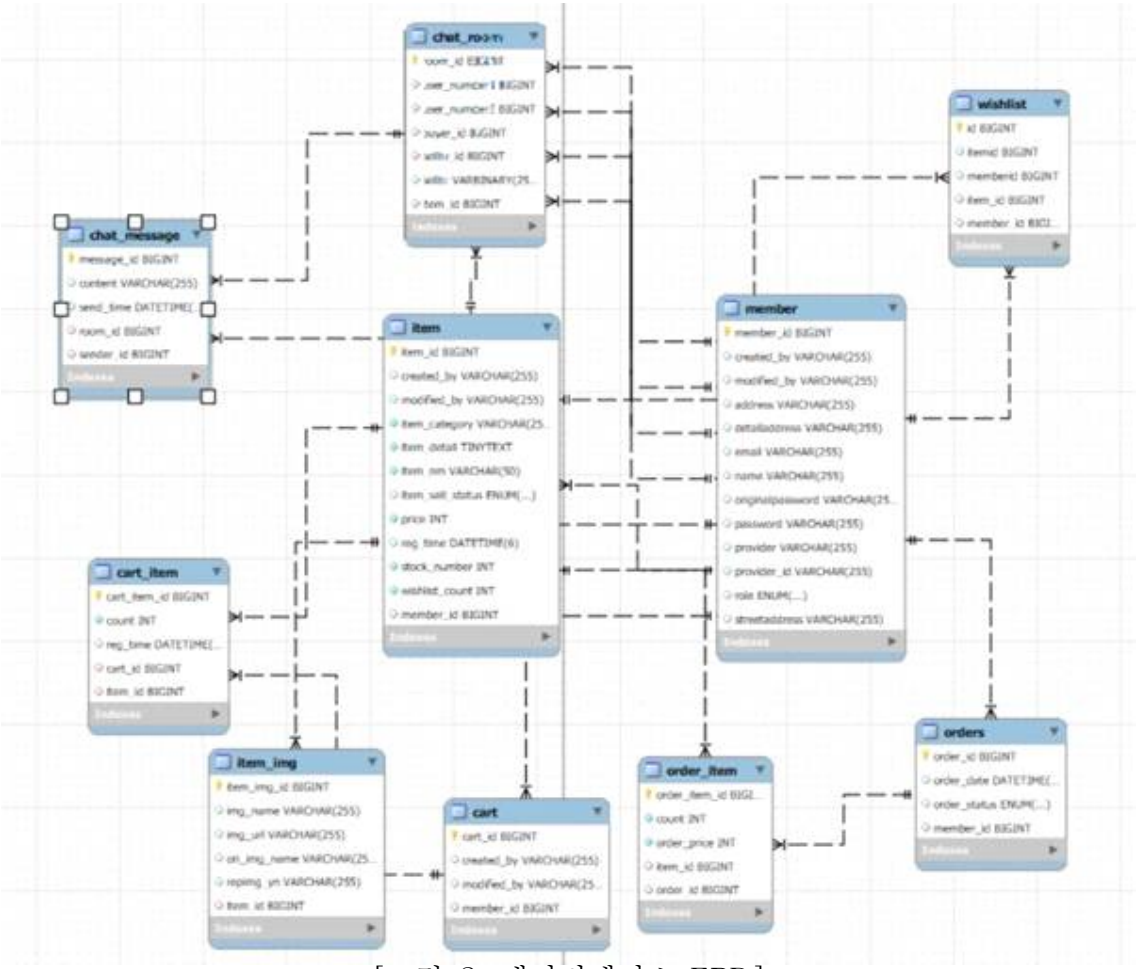
3. 본론

3.1 서비스 구상

그림 1은 웹 애플리케이션의 서비스 구상도이다. 사용자는 React.js로 구성된 인터페이스를 통해 로그인하고, JWT를 이용해 인증된 후 REST API를 통해 서버와 통신한다. Spring Boot는 이 요청을 처리하고, MySQL 데이터베이스와 상호작용하여 필요한 데이터를 관리한다. 판매자는 물건을 등록하고, 판매 등록 후에는 물건의 정보와 이미지 등이 DB에 저장된다. 구매자는 물건을 장바구니에 담거나 구매할 수 있으며 판매자와 웹 소켓을 통해 1대1 채팅을 할 수 있다.

[그림 1. 서비스 구상도]

그림 2는 플랫폼의 가변 데이터를 저장하는 DB에 대한 ERD(Entity Relationship Diagram)이다. member와 user테이블은 사용자 정보를 저장하며, item과 item_img는 상품과 이미지 정보를 관리한다. wishlist는 사용자가 찜한 상품을, cart와 cart_item은 장바구니 내역을 저장한다. orders와 order_item은 주문 정보와 각 주문에 포함된 상품을 관리한다. 마지막으로 chat_room과 chat_message는 사용자 간 채팅 기능을 지원하는 구조로 설계되었다.



[그림 2. 데이터베이스 ERD]

3.2 개발환경

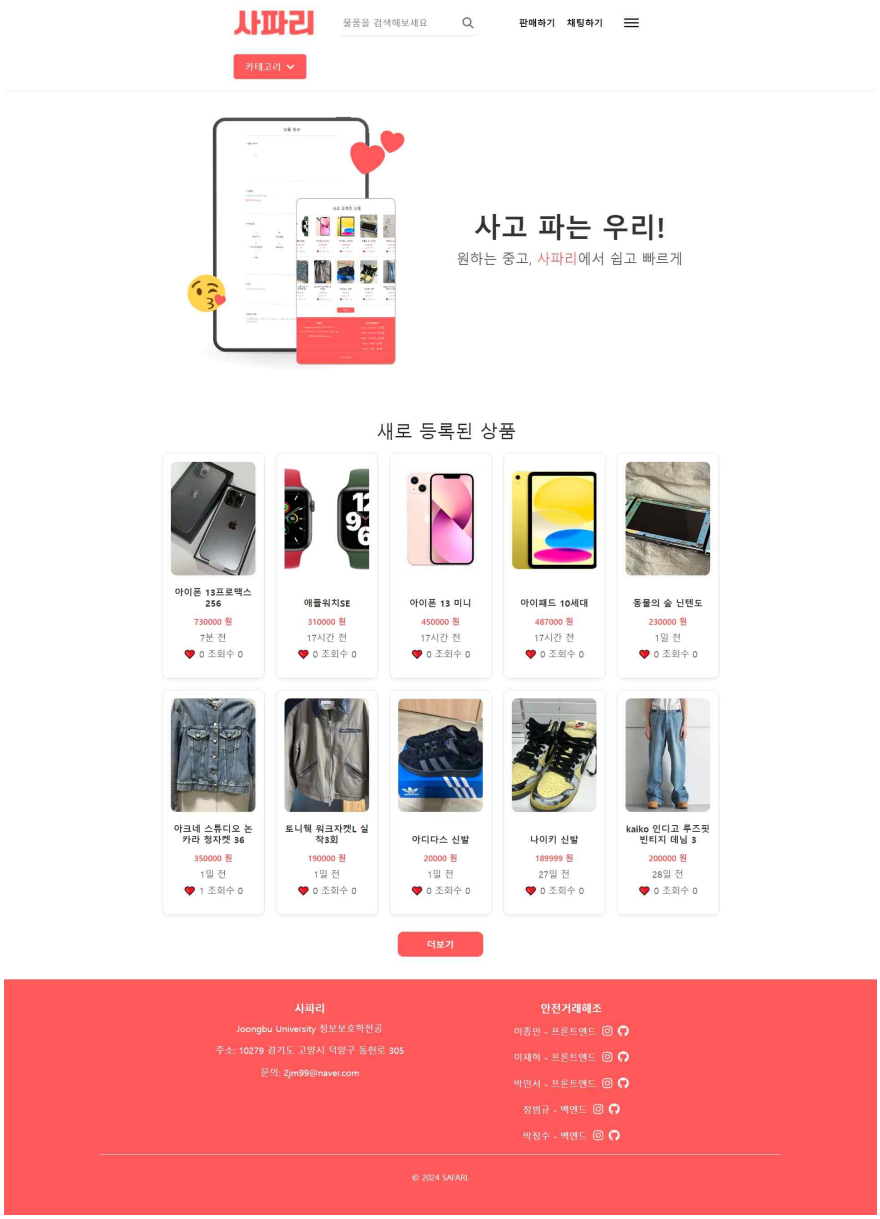
프로그래밍 언어: JavaScript
프레임워크: React, Spring Boot
데이터베이스: MySQL
인프라: AWS, JSON Wen Token
도구: Visual Studio Code, IntelliJ IDEA, GitHub

4. 서비스 구성

4.1 메인페이지

[메인페이지 구성요소]

상단 메뉴에는 상품 검색, 판매하기, 채팅하기, 로그인/회원가입, 마이페이지. 그리고 배너, 새로등록된 상품으로 구성.



[그림 3. 메인페이지]

4.2 로그인/회원가입

회원가입은 사용자가 이름, 이메일, 비밀번호, 주소 등의 정보를 입력하고, 이메일 인증을 통해 계정을 생성하는 과정이다. 사용자는 입력된 이메일로 인증번호를 발송받고, 이를 입력하여 인증을 완료해야 한다. 이메일 인증은 handleSendEmail 함수에서 처리되며, handleConfirmCode 함수에서 인증번호가 확인된다. 모든 정보가 입력되고 인증이 완료되면, handleSubmit 함수를 통해 서버로 회원가입 요청이 전송되고, 서버는 해당 정보를 데이터베이스에 저장한다. 회원가입이 성공적으로 완료되면 사용자는 로그인 페이지로 리다이렉트 된다.

```
const handleSubmit = async (e) => {
  e.preventDefault();

  try {
    console.log("전송할 데이터: ", formData);

    const response = await axios.post(
      'http://localhost:8080/members/new',
      {
        name: formData.name,
        email: formData.email,
        password: formData.password,
        zipcode: formData.zipcode,
        streetadr: formData.streetadr,
        detailadr: formData.detailadr,
        role: formData.role,
      },
      {
        headers: {
          "Content-Type": "application/json",

        },
      }
    );

    console.log('가입 성공', response.data);
    navigate('/auth'); // 가입 성공 후 리다이렉트
  } catch (error) {
    alert(`회원가입에 실패했습니다. 다시 시도해주세요: ${error.response ? error.response.data : error.message}`);
  }
};
```

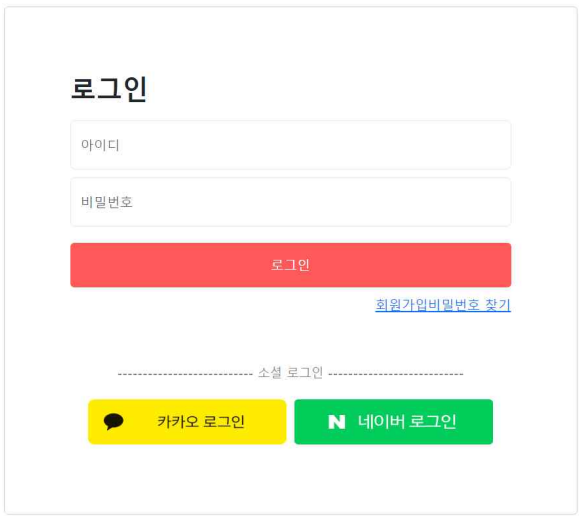
[그림 4. 회원가입 소스코드]

[그림 5. 회원가입 UI]

로그인은 사용자가 이메일과 비밀번호를 입력한 후 서버로 인증 요청을 보내는 과정이다. handleSubmit 함수에서 axios를 사용해 입력된 이메일과 비밀번호를 서버로 전송하며, 서버는 이를 검증한 후 JWT(JSON Web Token)를 발급한다. 발급된 JWT는 클라이언트의 로컬 스토리지(Local Storage)에 저장되며, 이는 이후 클라이언트의 모든 API 요청에 사용된다. 이때 요청 시 JWT는 HTTP 헤더에 포함되어 백엔드와 통신하며, 서버는 토큰을 검증하여 사용자의 인증 상태를 확인하고 요청을 처리한다.

```
const handleSubmit = async (e) => {
  e.preventDefault();
  try {

    const response = await axios.post('http://localhost:8080/members/login', {
      email: formData.id,
      password: formData.password
    });

    //로컬에 토큰 저장
    localStorage.setItem('token', response.data.token);
    console.log(response.data.token);
    localStorage.setItem('isLoggedIn', 'true');


    navigate("/");
  } catch (error) {
    alert("아이디 또는 비밀번호를 확인해주세요");
    setFormData({ id: '', password: '' });
  }
};

const handlePasswordFind = async () => {
  try {
    const response = await axios.post(`http://localhost:8080/members/sendEmail?memberEmail=${email}`);
    alert(response.data); // 성공 메시지
    setShowModal(false); // 모달 닫기
    setEmail(''); // 입력값 초기화
  } catch (error) {
    alert('이메일 전송에 실패하였습니다.'); // 오류 메시지
  }
};
```

[그림6. 로그인 소스코드]



[그림 7. 로그인 UI]

그림 8은 Spring Boot 애플리케이션에서 이메일 인증과 임시 비밀번호 생성 및 전송기능을 제공하는 MailService 클래스이다. 이 서비스는 이메일 인증을 위해 난수를 생성하고, 이를 포함한 인증 메일을 사용자의 이메일 주소로 발송한다. 인증 메일은 JavaMailSender를 사용해 HTML 형식으로 전송되며, 사용자는 해당 인증 번호를 통해 인증절차를 완료 할 수 있다. 또한, 임시 비밀번호를 랜덤으로 생성하여, 사용자의 비밀번호를 임시비밀번호로 변경한 후 이를 이메일로 전송하는 기능도 구현했다. 임시 비밀번호는 BCryptPasswordEncoder를 사용해 암호화하여 데이터베이스에 저장되며, 사용자는 이 임시비밀번호로 고르인한 후 비밀번호를 변경할 수 있다. 이 서비스는 이메일을 통한 인증 및 비밀번호 재설정 절차를 간편하게 처리하여, 사용자 계정의 보안을 강화한다.

```
public static void createNumber() { number = (int)(Math.random() * (90000)) + 100000; }

1개 사용 위치  jeongbeomgyu
public MimeMessage CreateMail(String mail) {  Complexity is 3 Everything is cool!
    createNumber();
    MimeMessage message = javaMailSender.createMimeMessage();

    try {
        message.setFrom(senderEmail);
        message.setRecipients(MimeMessage.RecipientType.TO, mail);
        message.setSubject("[당근마켓 이메일 인증 번호입니다.]");
        String body ="";
        body += "<h1>" + "당근마켓 메일 인증" + "</h1>";
        body += "<h3>" + "당근마켓에 오신 것을 환영합니다!" + "</h3>";
        body += "<h3>" + "요청하신 인증 번호입니다." + "</h3>";
        body += "<h1>" + number + "</h1>";
        body += "<h3>" + "상단의 인증 번호를 입력창에 입력해주세요. 감사합니다." + "</h3>";
        message.setText(body, charset: "UTF-8", subtype: "html");
    } catch (MessagingException e) {
        e.printStackTrace();
    }

    return message;
public static void createNumber() { number = (int)(Math.random() * (90000)) + 100000; }

1개 사용 위치  jeongbeomgyu
public MimeMessage CreateMail(String mail) {  Complexity is 3 Everything is cool!
    createNumber();
    MimeMessage message = javaMailSender.createMimeMessage();

    try {
        message.setFrom(senderEmail);
        message.setRecipients(MimeMessage.RecipientType.TO, mail);
        message.setSubject("[당근마켓 이메일 인증 번호입니다.]");
        String body ="";
        body += "<h1>" + "당근마켓 메일 인증" + "</h1>";
        body += "<h3>" + "당근마켓에 오신 것을 환영합니다!" + "</h3>";
        body += "<h3>" + "요청하신 인증 번호입니다." + "</h3>";
        body += "<h1>" + number + "</h1>";
        body += "<h3>" + "상단의 인증 번호를 입력창에 입력해주세요. 감사합니다." + "</h3>";
        message.setText(body, charset: "UTF-8", subtype: "html");
    } catch (MessagingException e) {
        e.printStackTrace();
    }

    return message;
```

[그림 8. 메일 인증을 통한 인증번호 및 임시 비밀번호 발급]

그림 9는 비밀번호 분실 시 이메일 인증으로 임시 비밀번호를 발급받아 로그인하는 과정.

[그림9. 임시 비밀번호로 로그인]

4.3 상품등록

상품 등록 과정은 사용자가 상품 정보를 입력하고 이미지를 업로드한 후, 서버에 이를 전송하는 방식으로 진행된다. 사용자는 상품명, 카테고리, 가격, 재고 수량 등 필수 정보를 입력하고, 추가적으로 상세 설명과 이미지를 업로드한다. 이 과정에서 Formik을 사용하여 입력된 데이터를 처리하며, Yup을 사용해 입력된 값에 대한 유효성을 검사한다. 이미지는 handleImageChange 함수로 처리되어, 파일 선택 시 미리보기로 표시되며, 사용자는 선택한 이미지를 삭제할 수 있다.

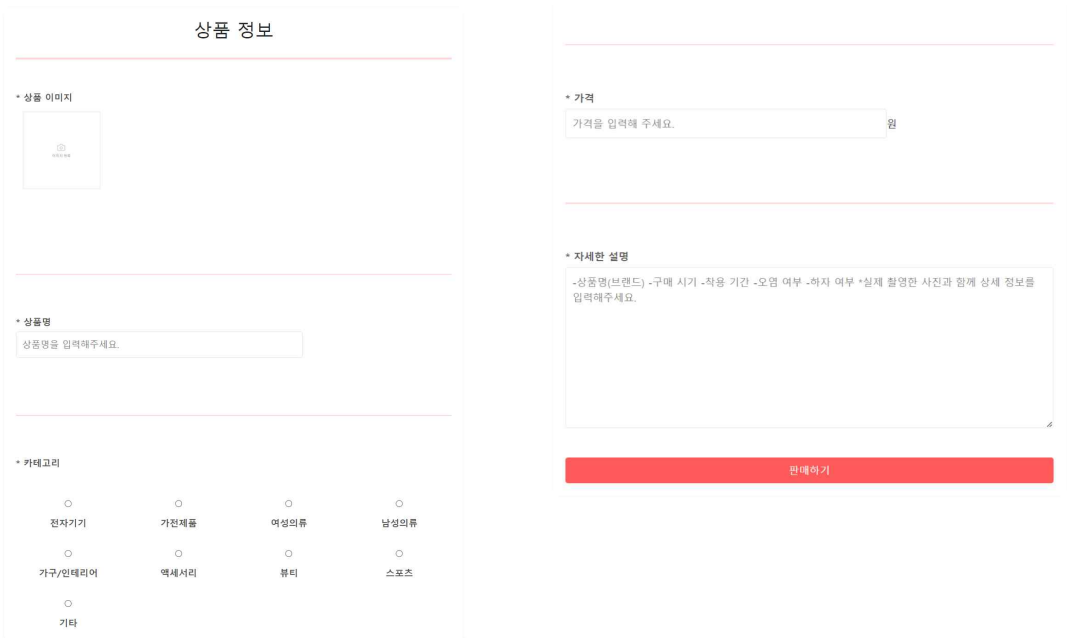
사용자가 입력을 완료하고 제출 버튼을 누르면, onSubmit 함수가 호출되어 서버와 통신한다. 이 함수는 입력된 정보를 FormData 객체에 담아 fetch API를 통해 서버의 /admin/item/new 엔드포인트로 전송한다. 이때, 로컬 스토리지에 저장된 JWT 토큰이 Authorization 헤더에 포함되어 사용자 인증이 이루어진다. 서버는 전송된 데이터를 바탕으로 상품을 등록하고, 성공적인 응답이 오면 사용자에게 등록 성공 메시지를 표시하며, 상품 목록 페이지로 리다이렉트된다. 오류가 발생하면 적절한 오류 메시지를 사용자에게 알리는 구조로 되어 있다.

```
const onSubmit = async (values) => {
  formData.append("itemNm", values.itemNm);
  formData.append("itemCategory", values.itemCategory);
  formData.append("price", values.price);
  formData.append("itemDetail", values.itemDetail);
  formData.append("stockNumber", values.stockNumber);
  formData.append("itemSellStatus", values.itemSellStatus);

  // 이미지 추가
  images.forEach(image => {
    formData.append("itemImgFile", image);
  });

  const token = localStorage.getItem('token'); // 로컬에서 토큰 가져오기
  try {
    const response = await fetch('/admin/item/new', {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${token}`,
      },
      body: formData,
    });

    if (response.ok) {
      const data = await response.json(); // JSON 응답 파싱
      alert(data.message); // 메시지를 알림으로 표시
      navigate('/Product');
    } else {
      const errorResponse = await response.json();
      alert(`상품 등록 중 에러가 발생하였습니다: ${errorResponse.message}`);
    }
  } catch (error) {
    console.error('Error saving product:', error);
    alert('상품 등록 중 서버와의 연결에 문제가 발생하였습니다.');
  }
};
```

[그림 10. onSubmit 함수]

```
const handleImageChange = (event) => {
  const files = Array.from(event.target.files);
  setImages(files);

  const newPreviews = [];
  files.forEach(file => {
    const reader = new FileReader();
    reader.onloadend = () => {
      newPreviews.push(reader.result);
      if (newPreviews.length === files.length) {
        setPreviews(newPreviews);
      }
    };
    reader.readAsDataURL(file);
  });
};
```

[그림 11. handleImageChange 함수]



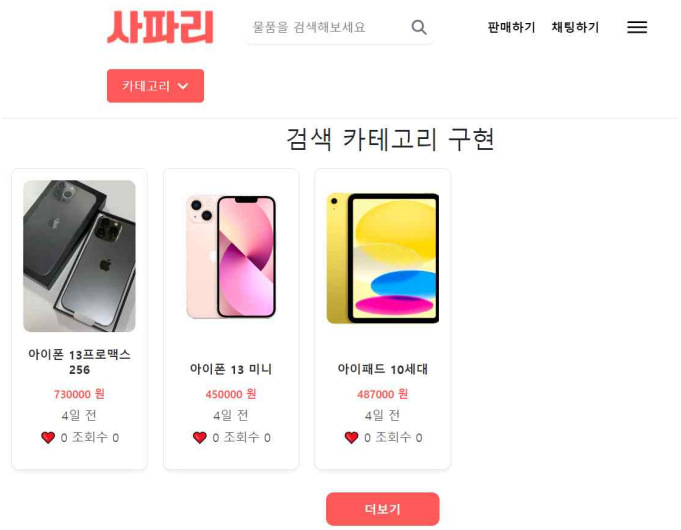
[그림 12. 상품등록 UI]

4.4 상품검색 및 상세정보

등록된 상품을 검색하는 과정은 사용자가 입력한 검색어를 서버에 전송하고, 그에 따른 검색 결과를 받아 화면에 표시하는 방식으로 이루어진다. 먼저 useLocation 훅을 이용해 URL의 쿼리 파라미터에서 검색어를 추출한 후, axios를 사용하여 검색어와 현재 페이지 정보를 서버에 전달한다. 서버는 해당 검색어에 맞는 상품 목록을 응답으로 보내고, 클라이언트는 이 데이터를 setResults로 상태에 저장하여 상품 목록을 렌더링한다. 각 상품은 이미지, 이름, 가격, 판매 상태 등을 포함하며, 사용자가 원하는 경우 '더 보기' 버튼을 통해 추가 결과를 로드할 수 있다.

```
const fetchSearchResults = async () => {
  const params = new URLSearchParams(location.search);
  const query = params.get('q') || '';
  const page = currentPage;

  try {
    const response = await axios.get(`/search`, {
      params: { searchQuery: query, page }
    });
    const data = response.data;

    setResults(data.items || []);
    setCurrentPage(data.currentPage);
    setTotalPages(data.totalPages);
  } catch (error) {
    console.error('검색 결과를 가져오는 중 오류가 발생했습니다:', error);
    setError('검색 결과를 가져오는 중 오류가 발생했습니다.');
  } finally {
    setLoading(false);
  }
};
```

[그림 13. 상품 검색 API 요청 및 결과 처리 코드]

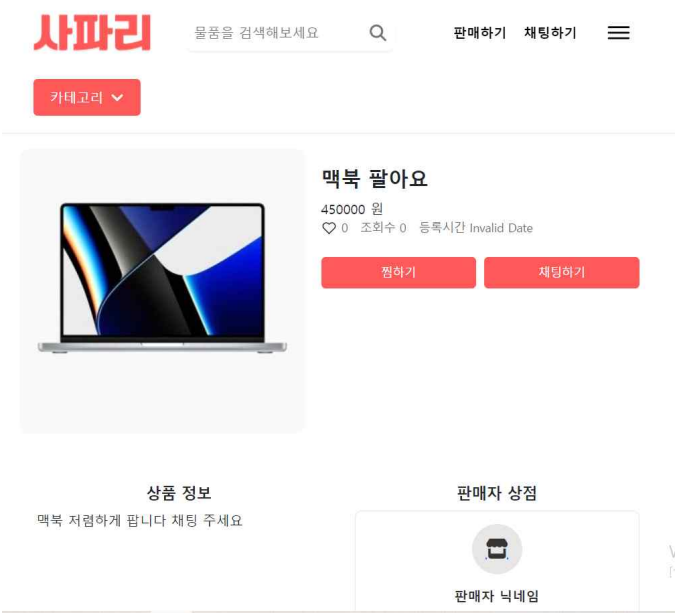


[그림 14. 상품검색페이지 UI]

상품 상세 페이지는 사용자가 선택한 상품의 세부 정보를 제공하며, useParams를 통해 URL에서 상품 ID를 추출하고, axios를 사용해 서버에서 상품 정보를 가져온다. 이 과정에서 JWT 토큰을 이용해 인증이 이루어진다. 상품 정보는 이미지, 이름, 가격, 등록 시간, 조회수 등을 포함하며, 사용자는 '찜하기' 버튼 기능을 사용할 수 있다. '채팅하기' 버튼을 통해 판매자와 실시간 채팅을 할 수 있으며, WebSocket을 통해 채팅방이 연결된다. 채팅창은 Modal 컴포넌트를 통해 모달 형식으로 열리며, 모달을 닫으면 WebSocket 연결이 종료된다. 또한, 판매자의 프로필 정보도 함께 제공된다.

```
useEffect(() => {
  const fetchProduct = async () => {
    const token = localStorage.getItem('token');
    try {
      const response = await axios.get(`http://localhost:8080/item/${id}`, {
        headers: {
          Authorization: `Bearer ${token}`,
        },
      });

      const productData = response.data;
      const mainImage = productData.itemImgDtoList.length > 0 ? productData.itemImgDtoList[0].imgUrl : null;

      setProduct({ ...productData, imgUrl: mainImage });
      setLikes(productData.likes || 0);
      setLiked(productData.liked || false);
    } catch (error) {
      if (error.response && error.response.status === 401) {
        setError('인증에 실패했습니다. 로그인 후 다시 시도해주세요.');
      } else {
        setError('상품을 불러오는 중 오류가 발생했습니다.');
      }
    } finally {
      setLoading(false);
    }
  };
```

[그림 15. 상품 정보 요청 코드]



[그림 16. 상품 상세페이지 UI]

4.5 채팅

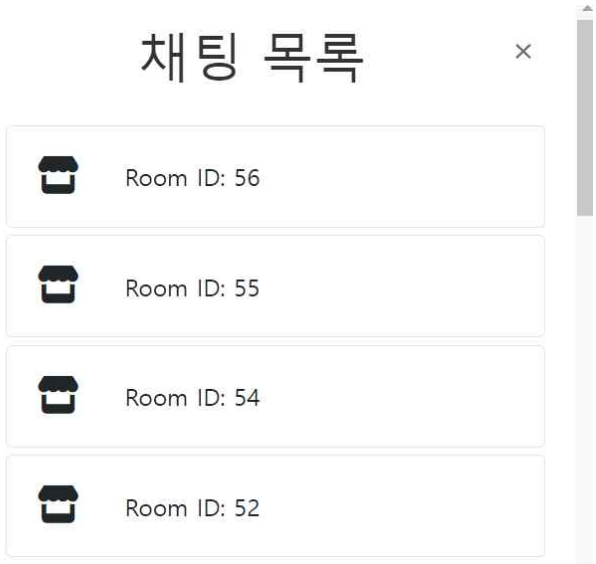
Spring Boot에서 Web Socket을 사용한 1대1 채팅방 기능이며, 다음과 같은 요소들을 포함한다. 사용자가 특정 아이템에 대해 대화하기 위해 채팅방을 생성하며, 이미 존재하는 방이 있다면 해당 방을 불러온다. 사용자는 채팅방에서 메시지를 전송할 수 있으며, 모든 메시지는 데이터베이스에 저장되어 대화 기록을 유지한다. 특정 채팅방의 모든 메시지를 조회하는 기능이 제공되며, 마지막 메시지를 확인할 수 있는 기능도 포함된다. Web Socket을 활용하여 실시간 메시지 전송이 가능하며, 새로운 메시지가 도착했을 때 사용자에게 알림을 제공한다. 사용자는 자신의 채팅방 목록을 조회하여 현재 진행 중인 대화들을 쉽게 확인하고 접근할 수 있다. 이러한 기능들은 사용자 간의 원활한 소통을 지원하며, 실시간 채팅 경험을 제공한다.

```
public List<String> getLastMessageList(Long userNumber, List<ChatRoom> chatRoom){  Complexity is 6 It's time to do something...

    List<String> messageList = new ArrayList<>();
    Pageable pageable = PageRequest.of(0,1, Sort.by("sendTime").descending());
    String lastMessage = "";
    for(int i = 0 ; i < chatRoom.size(); i++) {
        Page<String> pageList = chatMessageRepository.selectLastMessageByChatRoom(chatRoom.get(i).getRoomId(), pageable);
        System.out.println(pageList.getContent());
        if(pageList.getContent().size() != 0){
            lastMessage = pageList.getContent().get(0);
        }else{
            lastMessage = "대화내용이 없습니다.";
        }
        messageList.add(lastMessage);
    }

    System.out.println(messageList);
    return messageList;
}
```

```
// 채팅방 생성 또는 기존 방 가져오기

public ChatRoom createOrGetChatRoom(Long itemId, Long user2Id) {  Complexity is 6 It's time to do something...
    Item item = itemRepository.findById(itemId).orElseThrow(() -> new EntityNotFoundException("Item not found"));
    Member user1 = item.getMember(); // 상품을 등록한 판매자
    Member user2 = userRepository.findById(user2Id).orElseThrow(() -> new EntityNotFoundException("Buyer not found"));

    ChatRoom chatRoom = chatRoomRepository.findByUser1AndUser2AndItem(user1, user2, item);

    // 이미 존재하는 방이 없다면 생성
    if (chatRoom == null) {
        chatRoom = ChatRoom.createChatRoom(user1, user2, item);
        chatRoomRepository.save(chatRoom);
    }
    return chatRoom;
```

[그림 17. 1대1 채팅 소스코드]

[그림 18. 1대1 채팅 UI]



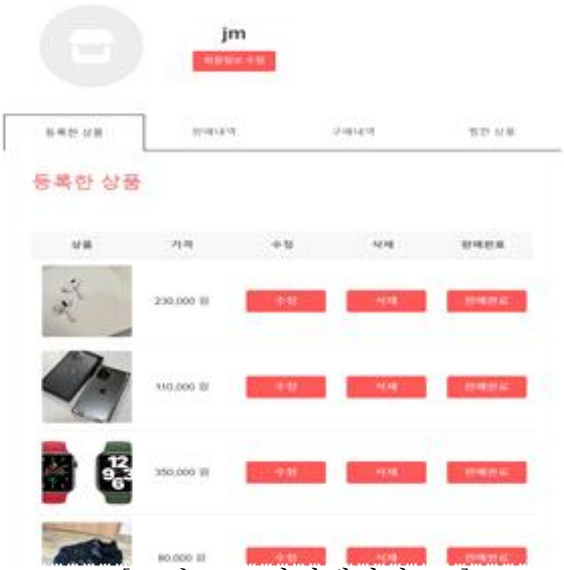
[그림 19. 채팅룸 UI]

4.6 마이페이지

마이페이지는 사용자가 자신의 프로필과 상품 정보를 관리할 수 있는 공간으로, 여러 섹션을 통해 등록한 상품, 판매내역, 구매내역, 찜한 상품을 확인할 수 있도록 구성되어 있다. 페이지가 로드되면, 백엔드 서버에서 JWT 토큰을 사용해 인증된 사용자의 정보를 가져와 화면에 표시한다. 사용자는 화면 상단의 버튼을 통해 각 섹션으로 이동할 수 있으며, 각 버튼은 activeSection 상태에 따라 활성화된다. 기본적으로 '등록한 상품' 섹션이 표시되며, 사용자가 판매를 완료하면 자동으로 '판매내역' 섹션으로 전환되는 기능도 포함되어 있다. 모든 섹션은 조건부 렌더링을 통해 activeSection의 값에 따라 적절한 컴포넌트가 화면에 표시되도록 구현되어 있다.

```
useEffect(() => {
  const fetchUserInfo = async () => {
    try {
      const response = await fetch('http://localhost:8080/members/myInfo', {
        method: 'GET',
        headers: {
          'Authorization': `Bearer ${localStorage.getItem('token')}`, // JWT 토큰 헤더에 추가
          'Content-Type': 'application/json',
        },
      });

      if (response.ok) {
        const data = await response.json();
        setUserId(data.name);
      } else {
        console.error('Error fetching user info:', response.statusText);

      }
    } catch (error) {
      console.error('Error fetching user info:', error);
    }
  };

  fetchUserInfo();
}, []);

const showSection = (sectionName) => {
  setActiveSection(sectionName);
};
```

[그림 20. 사용자 정보를 가져오는 코드]



[그림 21. 마이페이지 UI]

다음은 마이페이지에서 사용자가 등록한 상품 목록을 보여주고, 해당 상품에 대해 수정, 삭제, 판매 완료 등의 작업을 할 수 있는 기능을 제공한다. 페이지가 로드되면 componentDidMount를 통해 서버로부터 사용자가 등록한 상품 목록을 가져와 상태에 저장하고, 이를 화면에 렌더링한다. 상품 목록은 JWT 토큰을 이용해 서버에 인증된 요청을 보내 가져오며, 상품은 판매 완료, 수정, 삭제가 가능하다. '판매 완료' 버튼을 클릭하면 해당 상품은 목록에서 제거되며, 상태가 갱신된다. 수정 버튼을 통해 상품 정보를 수정할 수 있고, 삭제 버튼을 클릭하면 서버로 상품 삭제 요청이 전송되어 상품이 삭제된다.

```
fetchProducts = async () => {
    const token = localStorage.getItem('token'); // 토큰 가져오기
    if (!token) {
        this.setState({ error: '토큰이 없습니다. 로그인 해주세요.', loading: false });
        return;
    }

    try {
        // 사용자 등록 상품을 가져오는 엔드포인트
        const response = await axios.get('http://localhost:8080/user/items', {
            headers: {
                Authorization: `Bearer ${token}`
            }
        });

        console.log("API response for products:", response.data);
        const availableProducts = Array.isArray(response.data)
            ? response.data.filter(product => product.itemSellStatus !== 'SOLD_OUT')
            : (response.data.items || []).filter(product => product.itemSellStatus !== 'SOLD_OUT');

        this.setState({
            products: availableProducts.reverse(),
            loading: false
        });
    } catch (error) {
        console.error("상품을 가져오는 중 오류 발생:", error);
        this.setState({
            error: '제품을 가져오지 못했습니다',
            loading: false
        });
    }
};
```

[그림 22. 등록한 상품 내역 요청 코드]

```
handleDelete = async (productId) => {
    if (window.confirm('정말로 이 상품을 삭제하시겠습니까?')) {
        const token = localStorage.getItem('token');
        try {
            await axios.post(`http://localhost:8080/admin/item/delete/${productId}`, null, {
                headers: {
                    Authorization: `Bearer ${token}`,
                },
            });
            this.setState((prevState) => ({
                products: prevState.products.filter((product) => product.id !== productId),
            }));
            alert('상품이 삭제되었습니다.');
        } catch (error) {
            console.error("상품 삭제 중 오류 발생:", error);
            alert('상품 삭제 중 오류가 발생했습니다.');
        }
    }
};
```

[그림 23. 상품 삭제 핸들러]

```
handleSoldOut = async (productId) => {
    if (window.confirm("이 상품을 판매 완료로 표시하시겠습니까?")) {
        const token = localStorage.getItem('token');

        const orderData = {
            itemId: productId,
            count: 1,
        };

        try {
            // 주문 생성 엔드포인트로 POST 요청
            await axios.post('http://localhost:8080/order', orderData, {
                headers: {
                    Authorization: `Bearer ${token}`,
                    'Content-Type': 'application/json',
                },
            });

            // 상태에서 해당 상품 제거
            this.setState((prevState) => ({
                products: prevState.products.filter((product) => product.id !== productId),
            }));

            alert("상품이 판매 완료로 표시되었습니다.");

            // 판매내역 섹션으로 전환
            this.props.onSoldOut();
        } catch (error) {
            console.error("판매 완료 처리 중 오류 발생:", error);
            if (error.response && error.response.data) {
                alert(`판매 완료 처리 중 오류가 발생했습니다: ${error.response.data}`);
            } else {
                alert("판매 완료 처리 중 오류가 발생했습니다.");
            }
            if (error.response && error.response.status === 401) {
                alert("인증 정보가 없습니다. 다시 로그인해주세요.");
```

[그림 24. 상품 판매완료 핸들러]

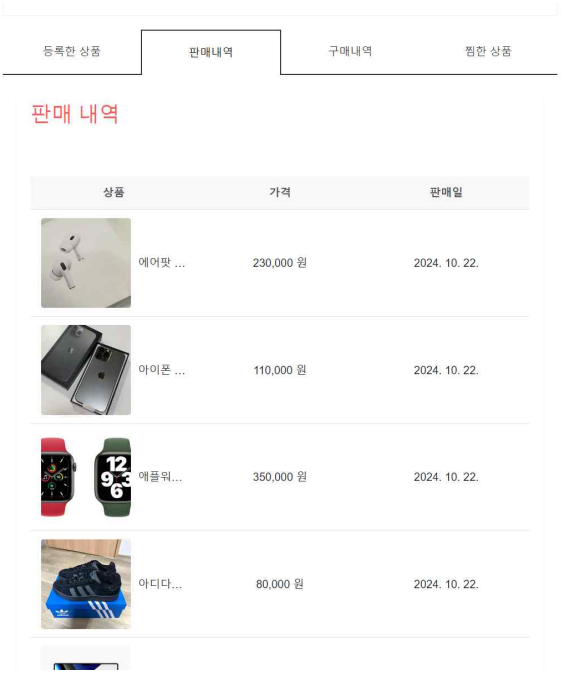
사용자가 판매 완료한 상품들의 내역을 보여주는 페이지로, 서버에서 판매된 상품 데이터를 가져와 화면에 표시하는 역할을 한다. 페이지가 로드되면 componentDidMount를 통해 서버로부터 판매 내역을 요청하고, JWT 토큰을 사용하여 인증된 요청을 보낸다. 서버로부터 응답받은 데이터는 sale 상태에 저장되며, 판매된 상품 목록은 테이블 형식으로 렌더링된다. 각 상품의 이미지, 이름, 가격, 판매일이 표시되며, 만약 판매 내역이 없을 경우 '판매 내역이 없습니다.'라는 메시지가 출력된다.

```
fetchSaleData = async () => {
    try {
        const token = localStorage.getItem('token');
        const response = await axios.get('http://localhost:8080/orders', {
            headers: {
                Authorization: `Bearer ${token}`,
            },
        });

        console.log("API response:", response.data);

        this.setState({
            sale: response.data.orders,
            loading: false,
            error: null,
        });
    } catch (error) {
        console.error("판매 내역을 가져오는 중 오류 발생:", error);
        this.setState({
            error: "판매 내역을 가져오는 데 실패했습니다.",
            loading: false,
        });
    }
}
```

[그림 25. 판매내역 요청 코드]

```
<tbody>
    {sale.map(order => (
        order.orderItemDtoList && order.orderItemDtoList.map(item => (
            <tr key={`${order.orderId}-${item.itemNm}`}>
                <td>
                    <div className={styles.ProductsProductInfo}>
                        <div className={styles.ProductImageWrapper}>
                            <img
                                src={`http://localhost:8080${item.imgUrl}`}
                                alt={item.itemNm}
                                className={styles.ProductsProductImage}
                            />
                        </div>
                        <span className={styles.ProductsProductName}>{item.itemNm}</span>
                    </div>
                </td>
                <td>
                    {item.orderPrice != null ? `${item.orderPrice.toLocaleString()} 원` : ''}
                </td>
                <td>{new Date(order.orderDate).toLocaleDateString()}</td>
            </tr>
        ))
    ))}
</tbody>
```

[그림 26. 판매내역 렌더링 코드]



[그림 27. 판매내역 UI]

사용자가 찜한 상품 목록을 관리하는 기능을 제공한다. 페이지가 로드되면, useEffect 훅을 통해 JWT 토큰을 이용하여 서버로부터 사용자 정보를 가져오고, 해당 사용자의 찜한 상품 목록을 조회하여 화면에 표시한다. 서버에 요청할 때는 Authorization 헤더에 저장된 JWT 토큰을 포함시켜 인증된 사용자만 접근 가능하도록 구현되어 있다. 사용자는 찜한 상품 목록에서 '삭제' 버튼을 클릭하여 상품을 삭제할 수 있으며, 이때 서버로 삭제 요청을 전송한 후, 로컬 상태에서 해당 상품을 제거하여 화면에서 실시간으로 반영된다.

```
useEffect(() => {
    const fetchMemberInfoAndLikedProducts = async () => {
        const token = localStorage.getItem('token');
        if (!token) {
            console.error('Token not found. Please login.');
            return;
        }

        try {
            const memberInfoResponse = await axios.get('http://localhost:8080/members/myInfo', {
                headers: {
                    Authorization: `Bearer ${token}`,
                },
            });
            const memberData = memberInfoResponse.data;
            setMemberId(memberData.id);

            const wishlistResponse = await axios.get(`http://localhost:8080/wishlist/user`, {
                params: { memberId: memberData.id },
                headers: {
                    Authorization: `Bearer ${token}`,
                },
            });

            setLikedProducts(wishlistResponse.data.data || []);
        } catch (error) {
            if (error.response && error.response.status === 401) {
                console.error('Unauthorized access. Please log in.');
            } else {
                console.error('Error fetching data:', error);
            }
        }
    };
```

[그림 28. 찜한 상품목록 요청 코드]

```
try {
    await axios.post('http://localhost:8080/wishlist/add', null, {
        params: { memberId, itemId: productId },
        headers: {
            Authorization: `Bearer ${token}`,
        },
    });
```

[그림 29. 찜한 상품 삭제 요청 코드]

[그림 30. 찜한 상품 목록 UI]

사용자가 등록한 상품 정보를 수정할 수 있는 기능을 제공한다. 페이지가 로드되면, JWT 토큰을 활용하여 서버로부터 해당 상품의 정보를 가져오고, 이를 폼의 초기값으로 설정한다. 사용자는 상품명, 카테고리, 가격, 설명, 상태(판매 중/품절), 재고 수량 등 다양한 정보를 수정할 수 있으며, 상품 이미지를 추가하거나 삭제하는 기능도 포함되어 있다. 사용자가 수정한 정보는 FormData 객체로 서버에 전송되며, 서버로부터의 응답에 따라 성공 또는 실패 알림을 사용자에게 전달하고, 성공 시 마이페이지로 리다이렉트된다.

```
useEffect(() => {
  const fetchProduct = async () => {
    try {
      const token = localStorage.getItem('token');
      const response = await fetch(`http://localhost:8080/item/${id}`, {
        method: 'GET',
        headers: {
          'Authorization': `Bearer ${token}`,
        },
      });

      if (response.ok) {
        const data = await response.json(); // 상품 정보 받아오기
        setProduct(data);
        setPreviews(data.itemImgDtoList.map(img => img.imgUrl));
      } else {
        const errorData = await response.json();
        console.error('Error fetching product:', errorData);
        alert('상품을 불러오는 데 실패했습니다.');
      }
    } catch (error) {
      console.error('Error fetching product:', error);
      alert('상품을 불러오는 중 오류가 발생했습니다.');
    }
  };
```

[그림 31. 상품정보 요청 코드]

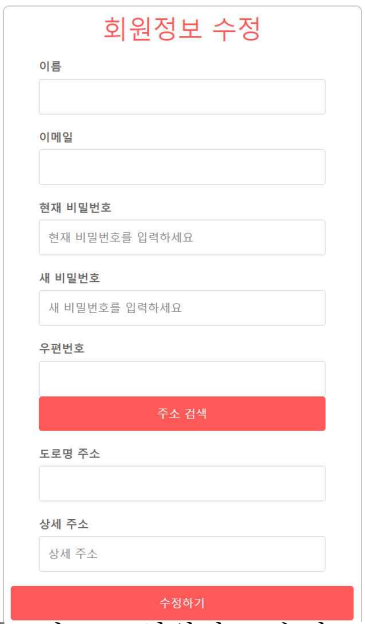
```
const onSubmit = async (values) => {
  const formData = new FormData();
  formData.append("itemNm", values.title);
  formData.append("itemCategory", values.categories);
  formData.append("price", values.price);
  formData.append("itemDetail", values.description);
  formData.append("itemSellStatus", values.status);
  formData.append("stockNumber", values.stockNumber);

  if (images.length > 0) {
    images.forEach((image) => {
      formData.append("itemImgFile", image);
    });
  } else {
    console.log("No images to upload");
  }
  for (const pair of formData.entries()) {
    console.log(`${pair[0]}: ${pair[1]}`);
  }

  try {
    const response = await fetch(`http://localhost:8080/admin/item/${id}`, {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${localStorage.getItem('token')}`,
      },
      body: formData,
    });

    if (response.ok) {
      alert('상품이 성공적으로 업데이트되었습니다.');
      navigate('/MyPage');
    } else {
      const errorResponse = await response.text();
      console.error('Error response:', errorResponse);
      alert(`상품 업데이트 중 오류가 발생했습니다: ${errorResponse}`);
```

[그림 32. 상품 수정 데이터 서버 제출 코드]

사용자가 자신의 회원정보를 수정할 수 있는 페이지를 구현한 것이다. 페이지가 로드되면, JWT 토큰을 이용하여 서버에서 사용자 정보를 가져와 폼에 초기값으로 설정한다. 사용자는 이름, 이메일, 현재 비밀번호, 새 비밀번호, 주소 등을 수정할 수 있으며, DaumPostcode API를 이용해 주소를 검색하고 선택할 수 있다. 비밀번호 변경 과정에서는 먼저 현재 비밀번호의 일치 여부를 확인한 후, 일치하면 새로운 비밀번호와 함께 나머지 정보를 서버에 제출하여 회원정보를 업데이트한다.

```
const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const token = localStorage.getItem('token');


    const checkPasswordResponse = await axios.get('http://localhost:8080/members/checkPwd', {
      params: { checkPassword: userInfo.originalpassword },
      headers: {
        'Authorization': `Bearer ${token}`,
      },
    });

    if (checkPasswordResponse.data) {

      const response = await axios.put('http://localhost:8080/members/updateForm', {
        name: userInfo.name,
        email: userInfo.email,
        password: userInfo.newPassword,
        zipcode: userInfo.address,
        streetadr: userInfo.streetaddress,
        detailadr: userInfo.detailaddress,
      }, {
        headers: {
          'Authorization': `Bearer ${token}`,
        },
      });

      if (response.status === 200) {
        alert('회원정보가 성공적으로 수정되었습니다.');
        navigate('/mypage');
      }
    } else {
      alert('현재 비밀번호가 올바르지 않습니다.');
    }
```

[그림 33. 회원정보 수정 데이터 서버 제출 코드]



[그림 34. 회원정보 수정 UI]

5. 결론

5.1 개선사항

개선 방안은 시리얼 넘버 시스템을 도입하여 중고 거래의 신뢰성을 강화하는 것이다. 동작 과정은 다음과 같다. 판매자는 제품의 고유번호를 플랫폼에 등록하고, 이 번호는 데이터베이스에 저장된다. 구매자는 제품을 수령한 후, 해당 제품의 시리얼 넘버를 입력해 데이터베이스에 저장된 정보와 비교한다. 만약 번호가 일치할 경우, 구매자는 해당 제품의 출고일과 제품 스펙 등의 정보를 확인할 수 있다. 이를 통해 제품의 진위 여부를 검증할 수 있다.

5.2 기대효과

중고 거래 플랫폼을 통해 사용자들이 간편하게 물품을 사고팔 수 있는 환경을 제공하여, 거래가 보다 쉽고 신속하게 이루어진다. 이는 소비자들의 적극적인 참여를 유도하며, 시장의 활성화와 성장을 촉진한다. 또한, 중고 거래는 자원 재활용을 촉진하고 불필요한 폐기물을 줄여 환경 보호에 기여한다. 동시에 소비자들은 저렴한 가격에 물품을 구매하거나 불필요한 물품을 판매해 경제적 부담을 줄일 수 있어, 사용자 만족도와 시장 신뢰성을 높이는 긍정적인 효과를 기대할 수 있다.

6. 별첨

6.1 팀원소개

이종민(팀장)
역할: 프론트엔드 개발
깃허브 주소: https://github.com/jongmini0514

이재혁(팀원)
역할: 프론트엔드 개발
깃허브 주소: https://github.com/ljh5918

박민서(팀원)
역할 프론트엔드 개발
깃허브 주소: https://github.com/minseodao

정범규(팀원)
역할: 백엔드 개발
깃허브 주소: https://github.com/jeongbeomgyu

박정수(팀원)
역할: 백엔드 개발
깃허브 주소: https://github.com/pjs96328

6.2 소스코드

팀 프로젝트 - https://github.com/ljh5918/SAFARI

6.3 시현영상

시현영상 링크 - https://youtu.be/ORAE8ff-Wjk

6.4 발표자료(별첨)