누구나 쉽게 이용하는
이커머스 사이트 '띵켓' 제작

팀　　명 :　　띵킹 (Thinking)
지도 교수 :　　이병천 교수님
팀　　장 :　　이다원
팀　　원 :　　이나원

2021. 10. 22
중부대학교 정보보호학과

목 차

프로젝트 요약

프로젝트 주제를 처음 선정할 때, 우리 주변에서 흔히 볼 수 있는 서비스를 만들어보면 어떨까 생각했다. 사용할 때는 편리하고 간단하게 사용할지 몰라도 내가 사용하던 서비스를 내가 개발자가 되어서 만들어내고 기능들을 구현하는 것은 굉장히 어려운 문제일 것이고 도전이 될 것이라고 생각해서 졸업 작품 프로젝트의 주제로 e-commerce 웹 사이트 제작하기를 선정하게 되었다.

프로젝트의 이름은 "Thinket"으로, think에 market을 합친 Thinket 이라는 이름으로 e-commerce 웹 사이트를 제작했다.
사이트를 제작하기 위해 프론트엔드 부분에는 React.js, Redux를 사용했고, 백엔드 부분에는 Node.js, Express, Mongo DB, Mongoose를 사용했다.

기본적으로 웹 사이트에 사용자가 로그인/로그아웃 할 수 있도록 제작을 했고, 웹 사이트의 상품들을 전체적으로 보여주는 메인 페이지를 시작으로, 각각의 상품 상세 페이지, 상품을 장바구니에 담을 수 있도록 장바구니 페이지를 제작하고,
본격적으로 주문을 위한 사용자의 배송 지 정보 등을 입력받고 최종 주문을 하면 주문정보를 확인 할 수 있도록 상세 주문 확인 페이지도 제작했다.
그리고 사용자가 그동안의 주문 내역 리스트를 확인 할 수 있도록 주문 내역 페이지를 제작하였다.

판매자로 등록되어 있는 계정은 자신이 현재 등록한 상품들을 확인 할 수 있도록 상품 목록 페이지와 추가로 판매할 상품을 등록할 수 있도록 기능을 제공하고, 판매자에게 들어온 주문을 확인 할 수 있는 판매자용 주문 내역 페이지도 제작을 하였다.
상품마다 판매자가 정해져 있기 때문에, 해당 판매자가 판매하는 상품들만 따로 모아서 볼 수 있는 페이지도 제작을 진행했다.

마지막으로 관리자로 등록되어있는 계정은 웹 사이트의 관리자이기 때문에 사용자들을 관리할 수 있어야 하므로, 전체 사용자 목록 페이지를 만들어 사용자를 관리할 수 있게 하고, 판매자 권한 또한 사용자 목록 페이지에서 관리자가 설정할 수 있도록 만들었다.
그리고 관리자는 판매자를 불문하고 전체적인 상품 목록과 주문 내역을 확인 할 수 있고, 전체적인 주문을 차트로 볼 수 있는 대시보드 페이지도 같이 만들어서
월별 주문 현황을 차트로 확인할 수 있고, 카테고리 별 판매 현황 또한 확인 할 수 있도록 제작했다.

1. 서론

1.1 연구 배경 및 필요성

팀원이 모두 프론트엔드에 관심을 가지고 있었고, 프론트엔드를 진로의 방향으로 잡고 공부하고 있는 와중에, 우리가 살면서 자주 접하게 되는 e-commerce 웹사이트가 눈에 띄었다. 현존하는 여러 가지 특징들을 가지고 있는 웹 사이트들이 있지만, 그중에 특히나 수많은 기능들이 포함되어있는 e-commerce 웹사이트들이 어떻게 작동을 하고, 어떻게 구성이 되어서 운영이 되고 있는지를 탐구하고, 이를 직접 개발해보며 공부하는 시간을 갖기 위해 목표를 설정하고 프로젝트의 주제로 선정하였다.

1.2 연구 목표

프로젝트의 연구 목표는 웹 사이트의 기본적인 UI 디자인 구현과 e-commerce 웹사이트라면 필수로 가져야하는 기능들을 구현하는 것을 큰 목표로 잡았다.
장바구니, 상품 주문 기능과 상품 검색 및 필터링 기능, 별점 및 리뷰 기능 등을 제공할 수 있도록 하고, 또한 상품을 구매하는 소비자만 웹 사이트를 이용하는 것이 아니라 판매자로서도 사이트를 이용하고 상품을 등록하고 판매할 수 있도록 하는 것을 목표로 하고 프로젝트를 진행했다.

1.3 성과 요약

프로젝트 시작 시 계획하고 제일 주된 목표로 생각했던 실생활에서 많이 이용하는 e-commerce 웹 사이트에서 꼭 필요한 메인 페이지, 상품 상세 페이지, 상품 장바구니 및 주문 기능, 검색, 필터링, 별점 및 리뷰 등 주요 기능들을 개발하였고, 판매자가 상품을 등록해 판매할 수 있도록 기능을 제작하였다. 또, 사이트 관리자는 상품들의 판매 현황과 상품 목록 관리, 사용자 관리를 할 수 있도록 기능을 구현하였다.

2. 관련 연구

2.1 사용 기술

2.1.1 React.js

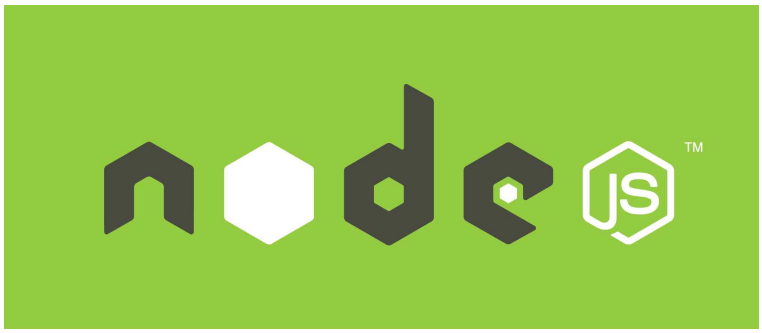
React는 페이스북이 만든 사용자 인터페이스(UI) 구축을 위한 선언적이고 효율적이며 유연한 Javascript 라이브러리이다. 컴포넌트라고 불리는 작고 고립된 코드의 파편을 이용하여 복잡한 UI를 구성하도록 도와주어 상호작용이 많은 UI를 만들 때 생기는 어려움을 줄여준다. 애플리케이션의 각 상태에 대한 간단한 설계만 하면 React는 데이터가 변경됨에 따라 적절한 컴포넌트만 효율적으로 갱신하고 렌더링해준다. 다양한 Front-end 프레임워크들 중에서 React를 사용한 이유는 많이 사용되기도 하고 간단하기 때문이다. 우리 팀은 HTML, CSS, JS만을 이용한 웹사이트 제작 경험이 많았기 때문에 위 세 가지 정도만 알고 있어도 간단히 배울 수 있다는 장점이 있었다. 또한 이용하면서 코드의 가독성과 재사용성이 좋고 유지보수가 편하다는 장점을 느꼈다.

2.1.2 Redux

React 사용 시에 상태관리를 도와주는 툴로는 React Context, Redux, MobX 등이 있는데 우리는 그 중에서 가장 많이 사용되는 Redux를 이용하기로 하였다.

Redux를 사용하면 한번 호출된 데이터는 store에 저장되고 컴포넌트들은 이 store에서 필요한 데이터만 가지고 가면 되기 때문에 효율적이고 다른 컴포넌트들에게 끼치는 영향 없이 모든 컴포넌트가 동일한 데이터를 가져다 쓰기 때문에 모든 UI가 일관된 내용을 유지할 수 있다는 장점이 있었다. 이러한 장점들을 위해 React 만으로 구현하기보다 Redux를 이용하여 크기가 점점 커지고 복잡해지는 데이터, 앱의 상태 관리를 해주었다.

2.1.3 Node.js



Node.js는 확장성 있는 네트워크 애플리케이션 개발에 사용되는 소프트웨어 플랫폼이다. 구글에서 개발한 자바스크립트 엔진을 사용하기 때문에 속도도 빠르고 단일 스레드 이벤트 루프를 통한 높은 처리 성능을 가지고 있다. Node.js는 자바스크립트 언어를 사용하기 때문에 JS를 조금 다룰 줄 알던 우리에게는 새로운 언어를 배우지 않아도 자바스크립트로 backend 로직을 개발할 수 있다는 장점이 컸다. 또한 npm을 통해 다양한 모듈(패키지)을 제공하기 때문에 우리가 필요한 라이브러리와 패키지를 검색, 설치하고 사용 할 수 있어서 좋았다.

2.1.4 Express

Express는 웹 및 모바일 애플리케이션을 위한 일련의 강력한 기능을 제공하는 간결하고 유연한 Node.js 웹 애플리케이션 프레임워크이다.

다양한 웹 프레임워크들이 있지만 현재까지 가장 많이 사용되는 Express를 사용하기로 하였다. 우리가 느낀 Express의 최대 장점은 단순함이었다. 또한 자유롭게 활용할 수 있는 많은 HTTP 유틸리티 메서드 및 미들웨어를 통해 쉽고 빠르게 API를 작성할 수 있었다.

2.1.5 Mongo DB

Mongo DB란 문서 지향 데이터베이스이며 NoSQL 데이터베이스 종류 중에 하나이다. NoSQL DB중 인지도가 1위인 MongoDB를 사용했는데 사용해보니 스키마 관리가 필요 없다는 점이 가장 좋았다. schema-less 이기 때문에 어떤 형태의 데이터이던지 저장 가능하고 JSON 형태로 저장하기 때문에 직관적이어서 데이터를 읽고 쓰기가 빨랐다.

2.1.6 Mongoose

Mongoose는 Node.js와 Mongo DB를 연결해주는 가장 유명한 ODM이다. 위에서 설명했듯이 Mongo DB는 schema-less이기 때문에 어떤 형태의 데이터를 넣어도 저장 가능하다는 장점이 있다.

하지만 이는 반대로 불필요한 데이터가 들어간다거나, 원하는 자료 형과 맞지 않는 데이터가 들어갈 수도 있는 문제가 존재한다. mongoose는 schema가 도입되어 있기 때문에 schema에서 선언한 틀에 맞게 데이터를 넣어 위와 같은 문제점을 보완할 수 있다는 장점이 있다. Mongo DB의 NoSQL의 장점을 살리며 DB를 다룰 수 있다.

2.2 기존 이커머스 사이트 조사

'띵켓'이라는 사이트의 본격적인 구축에 앞서 '띵켓'에서 제공해야할 서비스를 체계적으로 정리할 필요성을 느꼈다.
그러기 위해서 기존 쇼핑몰들이 제공하고 있는 서비스가 구체적으로 무엇인지, 어떤 기능이 있는지, 어떤 구조로 보여 지는지에 대해 조사해보았다.
우리 띵킹조는 수많은 이커머스 사이트들 중 2020년 기준 국내 이커머스 시장 점유율 현황 2위를 차지한 '쿠팡'과 국내 온라인 패션 쇼핑몰 1위를 차지한 '무신사'를 조사해보았다.

2.2.1 쿠팡

첫 번째로 조사해볼 사이트는 쿠팡이다.

전자상거래 사이트 쿠팡은 국내 이커머스 시장 점유율 2위답게 굉장히 로켓배송, 로켓회원, 쿠팡 페이 등등 쿠팡만의 특별한 기능을 많이 지니고 있었지만 그중에서 크게 회원가입 및 로그인 페이지, 메인페이지, 장바구니 페이지, 주문/결제 페이지, My쿠팡(마이페이지), 상품 목록 페이지, 상품 상세페이지로 나누어 구체적으로 조사하였다.
조사하는 과정에서 눈여겨본 필수 기능, 우리가 구현할 기능들은 '노란색 테두리'로 표시하였다.

▶회원가입 및 로그인페이지

[회원가입 페이지]

이메일(아이디), 비밀번호, 비밀번호 재입력, 이름, 휴대폰 번호, 이용약관 동의의 단계를 거쳐 회원가입을 진행한다.
각 정보 입력 시 조건 충족과 미 충족 시의 상황별로 다른 문구들이 출력되며 올바른 정보 입력을 요구하게 된다.

상황1) 이메일/이름/휴대폰 번호 각 양식에 맞지 않게 입력 시
:'올바르게 입력해주세요' 문구를 출력하며 재입력을 요구한다.

상황2) 이미 가입된 계정의 이메일(아이디) 입력시

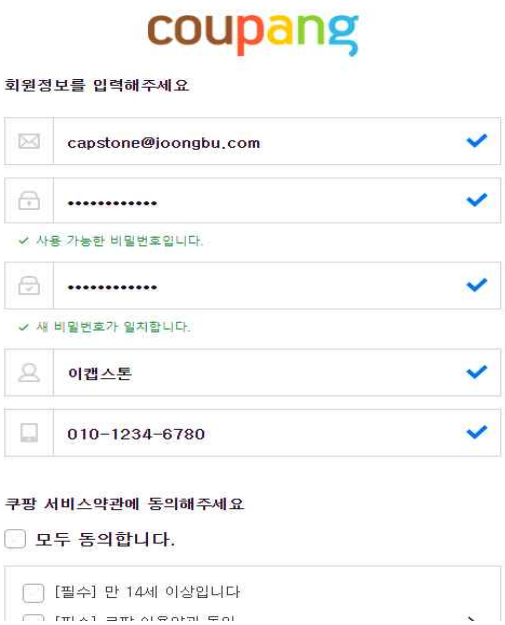
:'이미 가입된 이메일 주소입니다. 다른 이메일을 입력하여 주세요.' 문구를 출력한다.

상황3) 비밀번호 조건 미준수시

:비밀번호 조건별 미준수시 경고문 (영문/숫자/특수문자 2가지 이상 조합 -8~20자, 3개 이상 연속되거나 동일한 문자/숫자 제외, 아이디(이메일)제외)를 출력한다.

상황4) 비밀번호 재입력 시 입력한 비밀번호와 다를 경우
:'새 비밀번호가 일치하지 않습니다.' 문구를 출력한다.

상황5) 모든 조건 충족 시
체크표시 뜨며 회원가입이 가능해진다.



[모든 조건 충족 후 회원가입 가능]

[로그인 페이지]

아이디(이메일)입력, 비밀번호 입력을 거쳐 로그인이 가능하고 자동로그인 기능, 아이디/비밀번호 찾기, 로그인/회원가입 버튼이 존재한다.
회원가입 페이지와 마찬가지로 각 정보 입력 시 조건 충족과 미 충족 시의 상황별로 다른 문구들이
출력되며 올바른 정보 입력을 요구하게 된다.

상황1) 아이디(이메일)를 이메일 형식으로 미 입력할 경우
:'아이디(이메일)는 이메일 형식으로 입력해주세요.' 라는 문구가 출력된다.

상황2) 이메일 또는 비밀번호 잘못 입력 시
:'이메일 또는 비밀번호를 다시 확인하세요. 쿠팡에 등록되지 않은 이메일이거나, 이메일 또는 비밀번호를 잘못 입력하셨습니다.'라는 문구가 출력된다.

상황3) 로그인 성공 시
쿠팡 메인페이지로 전환

▶ 메인페이지
[헤더]
- 즐겨찾기/ 입점신청/ 로그인(로그아웃)/ 회원가입/ 고객센터
- 카테고리/ 상품 검색/ 마이쿠팡(마이페이지)/ 장바구니

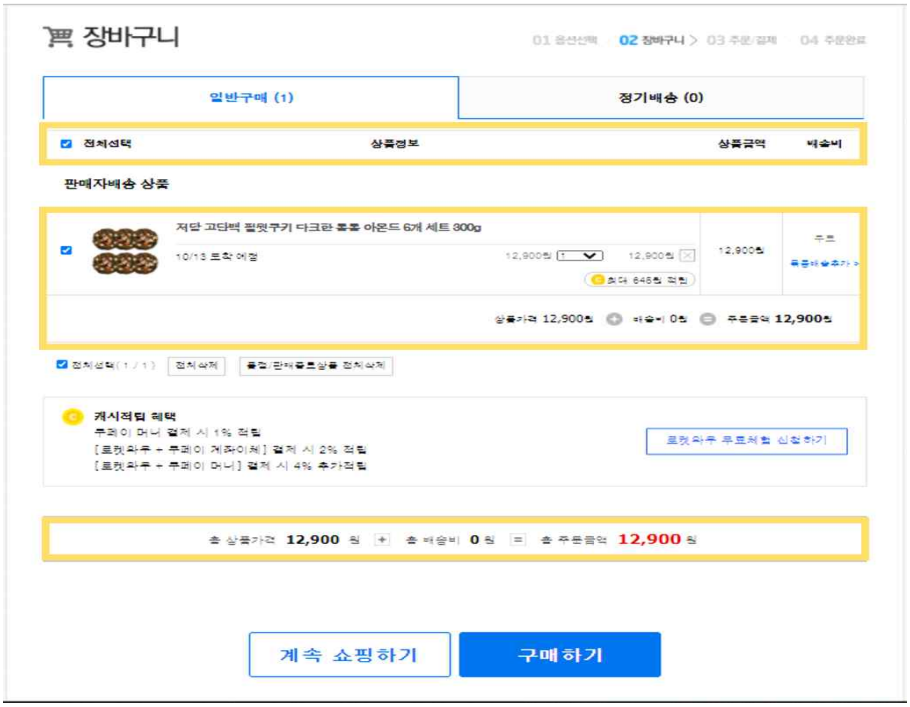
[메인]
- 이미지 슬라이더 -> 이벤트/ 쿠폰을 특정한 알고리즘을 통해 추천해준다.
- 오늘의 발견 (쿠팡이 엄선한 가장 HOT한 상품) -> 상품추천(카테고리를 추천)
- 오늘의 쇼핑 제안 (추천제품) -> 최근 본 상품 및 관련 제품을 추천 받을 수 있다.
-카테고리 별 상품 추천 -> 각 카테고리에 대해 상품 추천을 해준다.

[푸터]
-회사소개/ 인재채용/ 입점, 제휴문의/ 공지사항/ 고객의 소리/ 이용약관/ 개인정보처리방침/ 쿠팡 페이 이용약관/ 쿠팡 페이 개인정보처리방침/ 신뢰관리센터/ 제휴마케팅/ 광고안내

▶ 장바구니 페이지

상황1) 장바구니에 담은 상품이 없을 때
: '장바구니에 담은 상품이 없습니다.' 라고 표시되며 오늘의 추천 상품보기 버튼으로 메인 페이지 이동이 가능할 수 있게끔 한다.

상황2) 장바구니에 담은 상품이 있을 때
: 장바구니에 담은 상품 옵션 확인, 옵션(수량) 변경 가능하고 각 상품별 선택 후 장바구니에서 삭제, 품절/판매종료 상품 삭제가 가능하다.
'상품가격+배송비 = 총 주문금액' 으로 상품가격, 배송비, 총 주문금액이 확인 가능하다.



하단의 계속 쇼핑하기 버튼을 통해 메인 페이지로 이동 할 수 있고, 구매하기 버튼을 통해 주문/결제 페이지로 이동 할 수 있다.

▶ 주문/ 결제 페이지

장바구니에서 구매하기 버튼을 누를 시에 넘어오는 페이지이다.

구매자 정보와 주문내역을 확인하고 결제 정보를 선택할 수 있다.

1) 구매자 정보

: 이름 / 이메일/ 휴대폰 번호

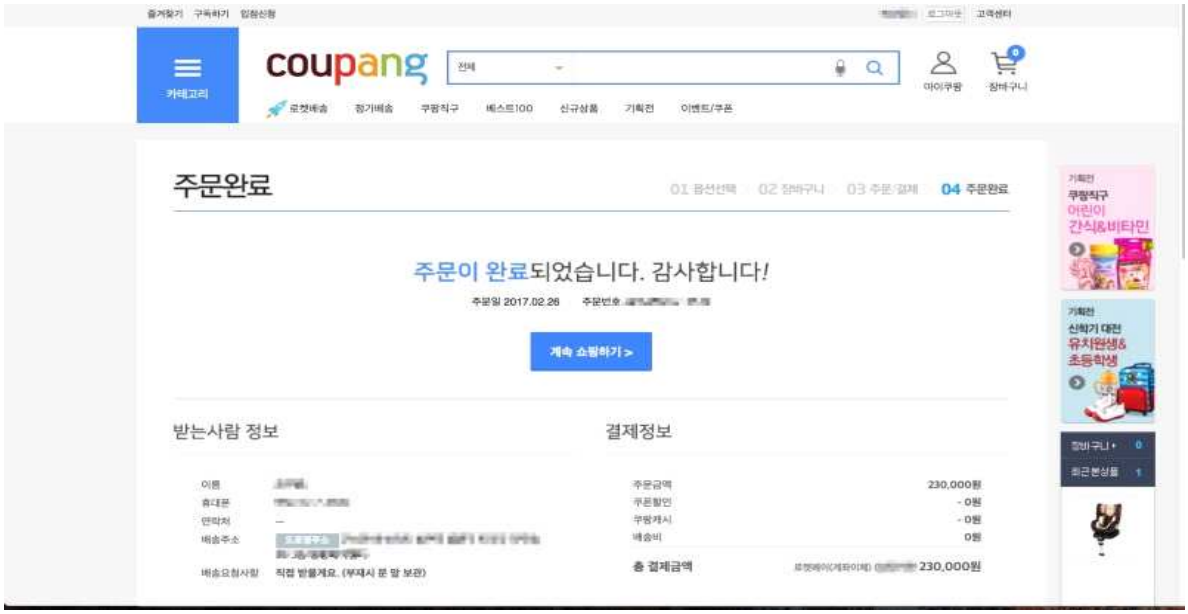
2) 받는 사람 정보

: 배송지 변경 가능 (배송지 추가/ 배송지 선택/ 배송지 수정)
이름 / 배송주소/ 연락처/ 배송 요청사항

3) 배송 n건 중 n
: 도착 예정일, 배송 상품 내역 표시

4) 결제정보
: 총 상품가격, 할인쿠폰 적용 금액 표시, 배송비, 쿠팡캐시, 쿠팡캐시 보유내역, 총 결제금액, 결제방법
-결제방법: 계좌이체, 쿠페이 머니, 신용/체크카드, 법인카드, 휴대폰, 무통장입금
-선택한 결제수단으로 향후 결제 이용에 동의합니다. (선택)

▶ 주문 완료 페이지



: "주문이 완료되었습니다. 감사합니다!" 라는 문구가 표시된다.
주문일, 주문번호가 확인 가능하고 계속쇼핑하기 버튼을 통해 메인 페이지로 돌아갈 수 있고 하단에 이름, 휴대폰, 연락처, 배송주소, 배송 요청사항 등의 받는 사람 정보와 주문금액, 쿠폰할인, 쿠팡캐시, 배송 비, 총결제금액등의 결제정보가 표시된다.

▶ My쿠팡 – 마이페이지

: 왼쪽 사이드 바에서 확인할 수 있듯이 크게 My쇼핑, My혜택, My활동,

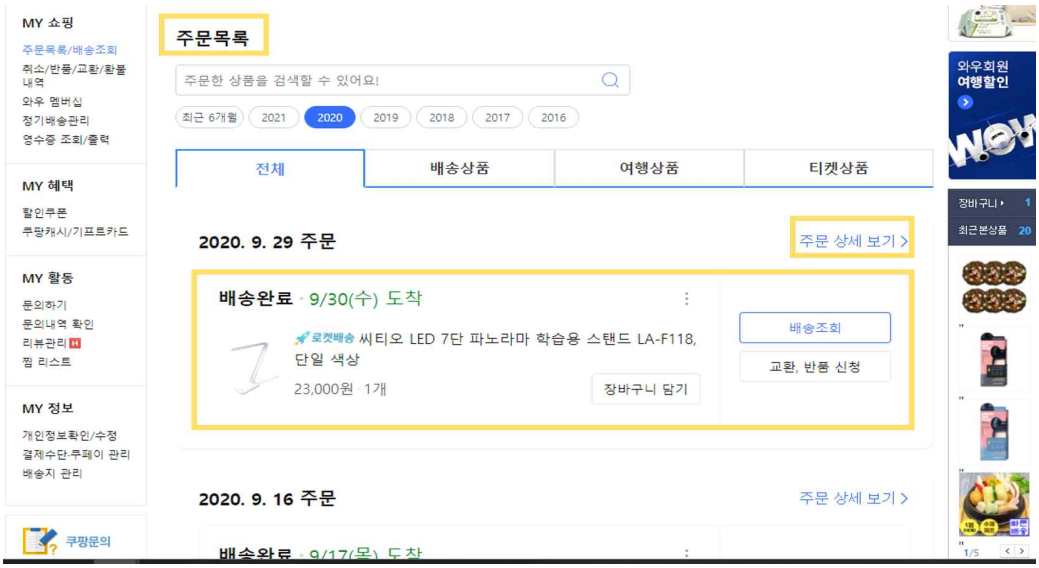
My정보로 나뉘어 있고 주문목록 및 배송조회, 취소/반품/교환/환불내역 확인, 문의하기, 개인정보 확인 및 수정 등의 업무를 수행할 수 있다.

1) My 쿠팡

: 상단의 My쿠팡 바에서 미사용 티켓, 배송 중, 할인쿠폰, 쿠페이 머니, 쿠팡캐시를 확인 가능하다.

2) My 쇼핑

: 내 주문목록, 배송조회 등을 확인할 수 있다.



My쇼핑 내의 카테고리 별 주요 기능을 살펴보도록 하겠다.

첫 번째로, 주문목록/배송조회 부분이다.

여기에서는 주문한 상품 검색가능, 최근6개월~5년 전까지 기간별 조회가 가능하다. 또한, 전체, 배송상품/여행상품/티켓 상품별로 확인가능하고, 주문 상세보기 및 상품별 배송조회, 교환/반품신청, 리뷰작성, 판매자 문의 등의 기능이 제공된다.

특히 주문 상세보기에서는 주문 날짜, 주문번호, 배송여부, 받는 사람(이름), 연락처, 받는 주소, 배송요청사항 등의 받는 사람 정보, 결제정보가 확인 가능하다.

두 번째로, 취소/반품/교환/환불내역에서는 취소접수일, 주문일, 주문번호, 총 환불 금액, 취소 상세 등이 조회 가능하다.

그밖에도 My쇼핑 카테고리 에서는 와우멤버십, 정기배송관리 및 현금영수증, 신용카드 매출전표 등이 확인 가능한 영수증 조회/출력 기능이 제공된다.

3) My혜택

:보유한 할인쿠폰 및 사용 완료한 쿠폰, 쿠팡캐시 등을 확인할 수 있다.

할인쿠폰 페이지에서는 사용가능한 쿠폰 명, 할인 액, 사용조건, 유효기간, 쿠폰종

류가 조회 가능하고 마감임박 할인쿠폰(24시간 기준)조회, 할인쿠폰 교환권 등록, 사용완료/ 기간 만료된 할인쿠폰 조회가 가능하다.

쿠팡캐시/기프트카드 페이지에서는 쿠팡 내에서만 사용가능한 쿠팡캐시의 사용가능캐시, 한 달 이내 소멸캐시, 예치금, 사용내역 등을 조회 가능하고 그밖에 기프트카드를 등록하거나 선물 하는 등의 기능이 제공된다.

4) My활동

:문의하기, 고객센터, 문의내역 조회 등이 가능하다.

첫 번째로, 문의하기 클릭 시 고객센터에 문의내역 페이지로 넘어가게 된다.

고객센터에서는 자주 묻는 질문 검색이 가능하고 최근 문의내역 조회 및 배송문의/ 반품, 교환, 환불/ 정기배송/ 주문, 결제/ 로켓모바일/ 쿠팡캐시/ 회원서비스/ 여행, 티켓 등의 분야별 자주 묻는 질문을 확인 할 수 있다.

그밖에도 고객의 소리, 공지나 이벤트를 확인 할 수 있는 쿠팡 소식 기능 등이 제공된다.

두 번째로는 리뷰 작성, 작성한 리뷰 조회, 쿠팡 체험단 신청이 가능한 리뷰관리 카테고리가 존재한다.

마지막으로 찜한 상품 및 찜한 브랜드 조회가 가능한 찜 리스트 카테고리 에서는 찜한 상품을 바로 장바구니에 담거나 삭제 가능한 기능이 제공 된다

5) My 정보

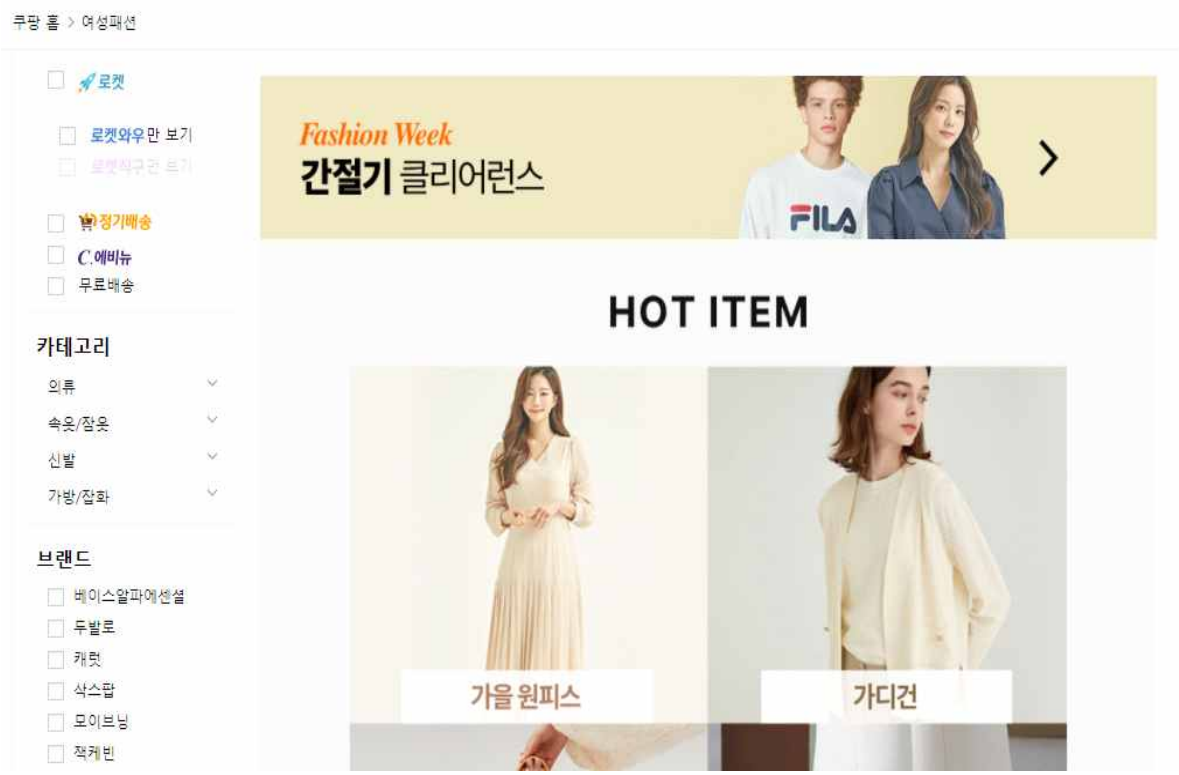
: My정보에서는 개인정보 확인 및 수정 등이 가능하다.

My정보 클릭 시 개인 정보 확인/ 수정을 위해서는 회원 정보 확인 과정을 거쳐야 한다. 내 계정에 해당하는 비밀번호를 확인한 후에 개인정보를 확인하거나 수정할 수 있다.

비밀번호 확인 과정을 거치면 회원정보 수정 페이지로 넘어올 수 있다.
회원정보 수정 페이지에서는 아이디(이메일), 이름, 휴대폰 번호, 비밀번호, 배송지, 수신 설정 등을 수정할 수 있다.
또한 회원정보 뿐만 아닌 결제수단, 쿠페이, 배송지등을 수정 할 수있다.

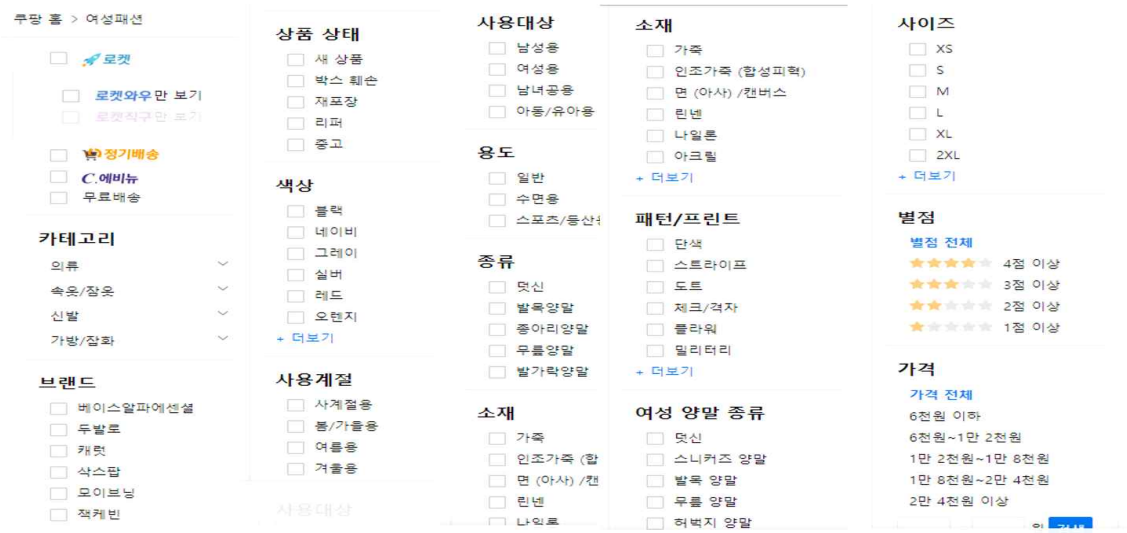
▶ 상품 목록 페이지
: 메인페이지에서 카테고리 클릭 시 보여 지는 페이지이다.
쿠팡엔 정말 다양한 상품군이 존재하기 때문에 여성의류 카테고리 클릭 시의 화면을 대표로 분석해보았다.

[여성의류 카테고리 클릭시]



메인페이지에서 여성의류 카테고리 클릭시에 보여지는 페이지이다.

좌측 카테고리 부분에서 의류(티셔츠, 맨투맨/후드티, 블라우스/셔츠 등), 속옷/잠옷, 신발, 가방/잡화 등의 세부 카테고리를 선택 할 수 있고 우측에서는 인기 아이템 등을 소개 하고 있다.

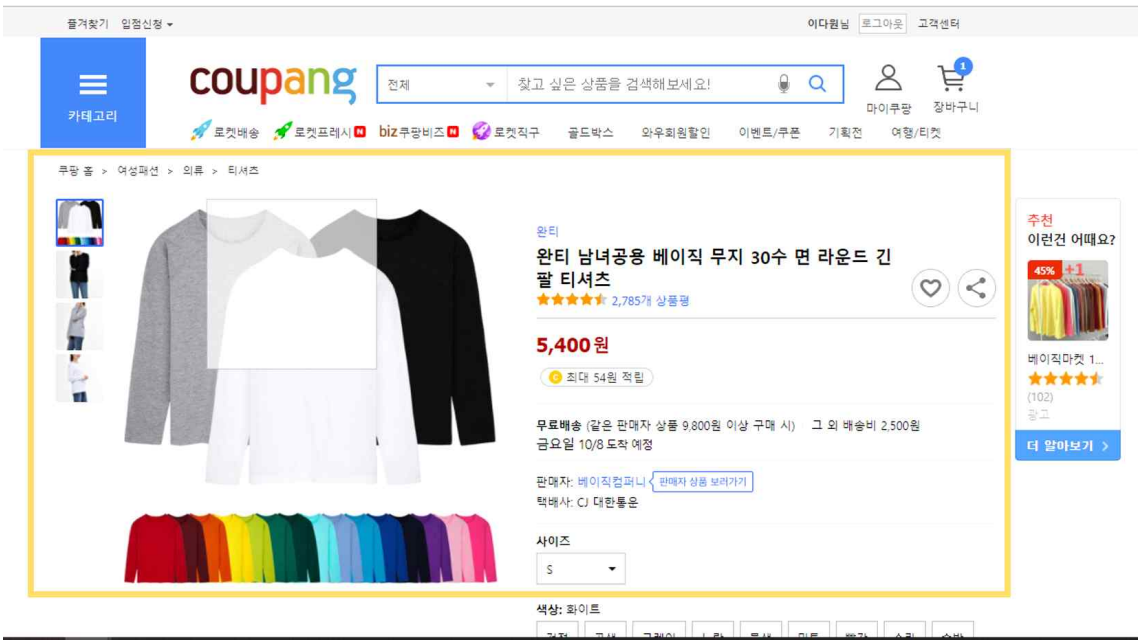


위에 보여 지는 사진은 여성의류 카테고리 클릭 시 좌측에 나타나는 사이드 바를 모두 캡쳐 하여 합쳐놓은 것이다.

브랜드, 상품상태, 색상, 사용계절, 사용대상, 용도, 종류, 소재, 패턴/프린트, 여성양

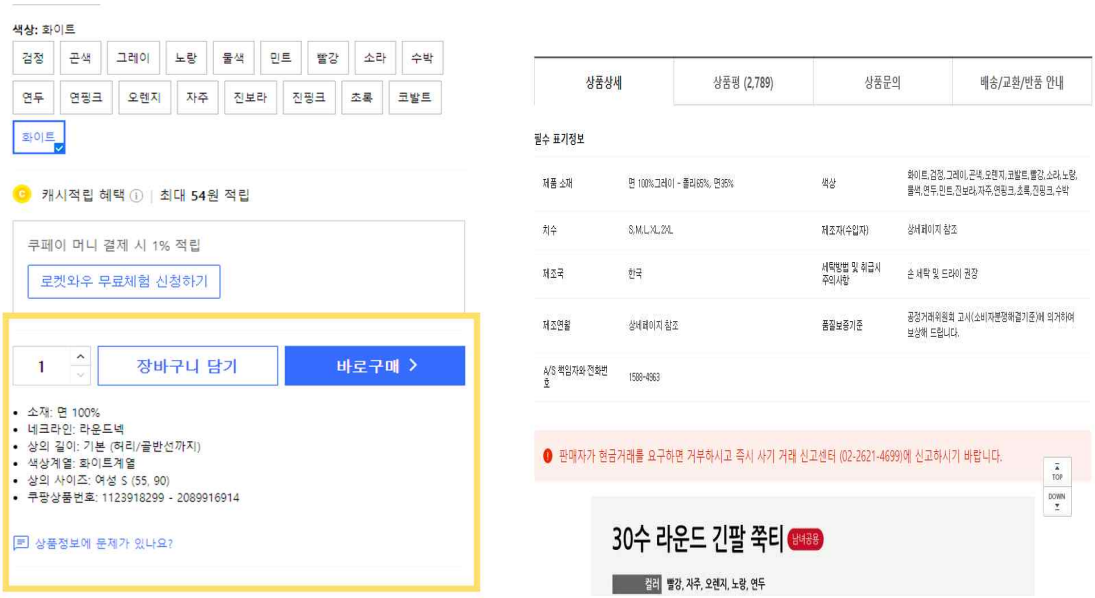
말종류, 사이즈, 별점, 가격 별로 세분화하여 필터링이 가능하다.
또한 상품 목록을 낮은 가격순/ 높은 가격순/ 판매량순/ 최신순 으로 정렬 가능하다.

▶ 상품 상세 페이지
: 상품 목록 페이지에서 하나의 특정 상품 클릭시 확인 가능한 페이지이다.



상품 상세페이지에서는 상품 사진과 함께 옵션(사이즈, 수량)선택이 가능하고 장바구니 담기버튼/ 바로구매 버튼을 이용할 수 있다.

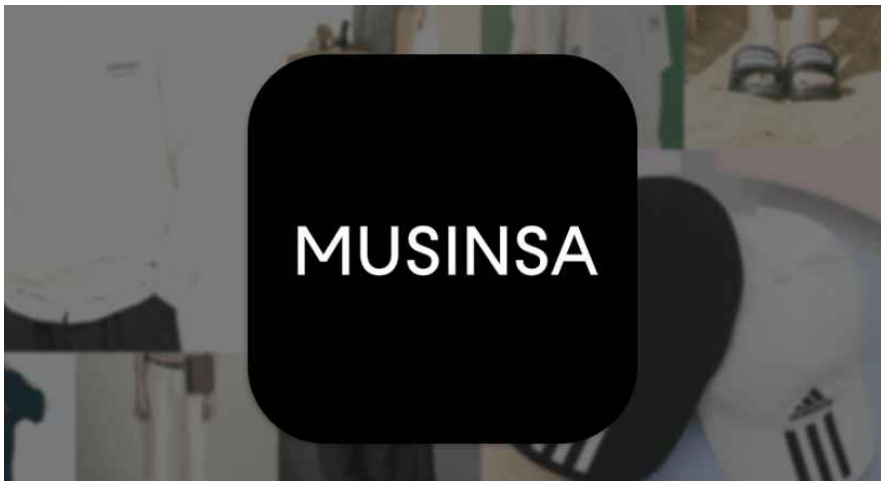
▶ 사이드 바
: 오른쪽 사이드 바에서는 최근 본 상품을 보여주며 내가 본 상품에 대한 관련 추천 상품, 장바구니에 담은 상품 등을 확인 할 수 있다.



또한 제품소재, 사이즈, 제조국 등의 상품 필수 표기 정보(상품 상세 정보), 상품평(리뷰), 상품 문의내역, 배송/교환/반품 안내 사항을 확인할 수 있으며 상품별 쿠팡 상품번호가 존재한다는 것이 확인 가능하다.

2.2.2 무신사

두 번째로 조사해본 사이트는 무신사이다.



'띵켓'은 여러 '의류'브랜드가 입점한 이커머스 사이트라는 컨셉으로 제작하기 때문에 좋은 참고서가 될 것이라 생각했다.

무신사 역시 쿠팡과 마찬가지로 크게 회원가입 및 로그인페이지, 메인페이지, 상품페이지, 상품 상세 페이지, 마이페이지로 나누어 조사해보았다.

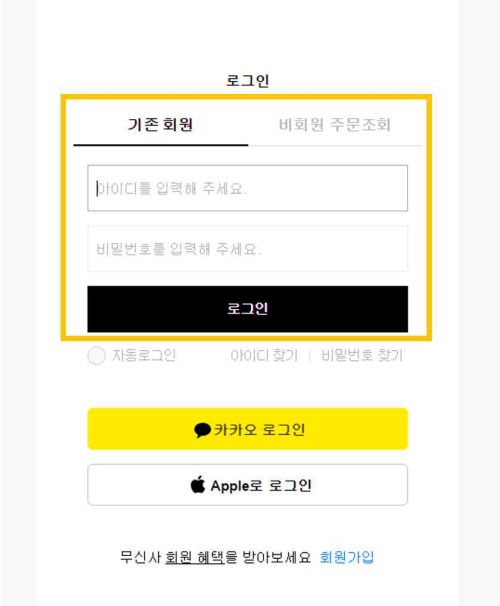
▶ 회원가입 및 로그인 페이지

[회원가입 페이지]

: 아이디, 비밀번호, 비밀번호 재확인, 이메일을 차례로 입력하게 되어있고 아이디 / 비밀번호/ 이메일 조건 미 충족 시 경고문(조건 안내문)이 출력된다.

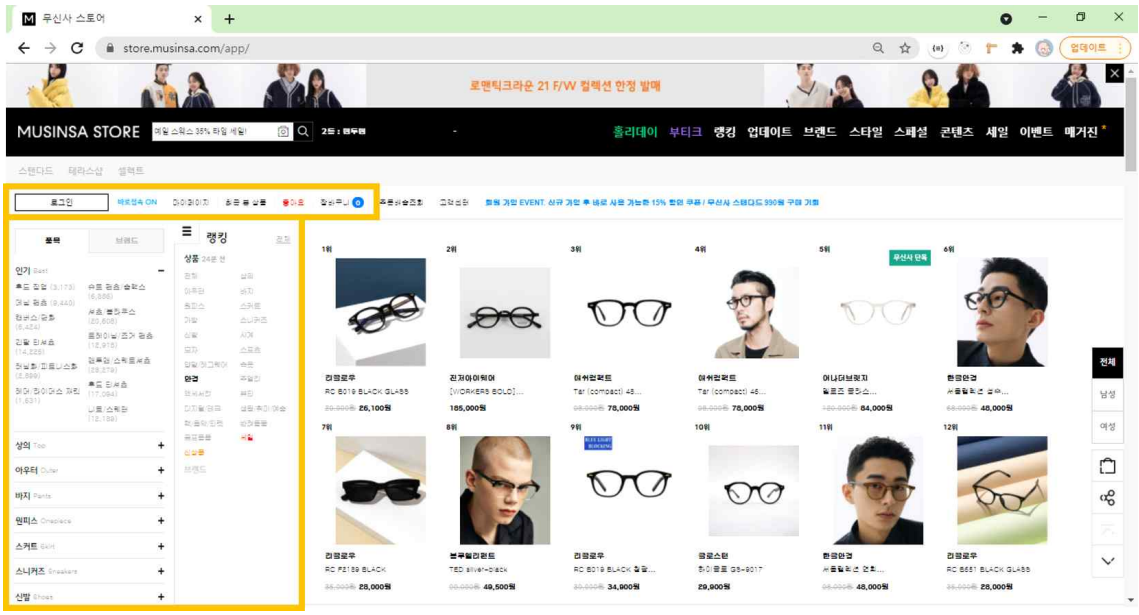
위에 보여 지는 화면은 회원가입 조건 충족시 보여 지는 화면이다.

[로그인 페이지]



: 기존회원과 비회원의 경우로 나누어 로그인이 가능하고 기존회원의 경우 아이디, 비밀번호를 입력하거나 카카오톡 등의 sns계정을 통해 로그인이 가능하게끔 되어있다.

▶ 메인페이지



: 광고 및 프로모션을 보여주는 상단 배너, 로고/ 검색/ 검색순위/ 메뉴바를 확인할 수 있고 로그인, 마이페이지, 최근 본 상품, 좋아요, 장바구니, 주문배송조회, 고객센터 등의 기능을 이용할 수 있게 되어있다.

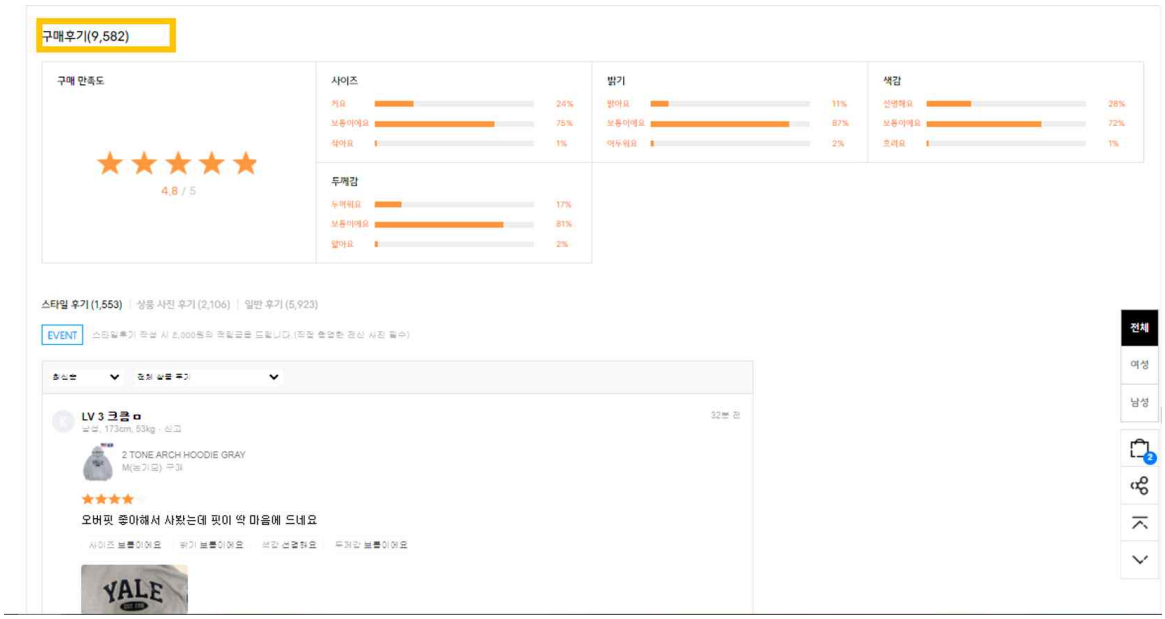
또한 상품별 카테고리를 확인 할 수 있는 품목버튼, 브랜드별 카테고리를 확인하고 브랜드명을 검색할 수 있는 사이드 바가 존재하며 우측으로는 카테고리 별 상품들의 랭킹이 사진과 함께 제공된다.

▶ 상품페이지

: 무신사 역시 너무나 다양한 상품 군과 카테고리가 존재하기 때문에 후드티 카테고리를 기준으로 조사해보았다.

[후드티 카테고리 클릭 시]

브랜드 별로 조회가 가능하고 색상, 가격, 실측(사이즈)별로 필터링 기능이 제공된다. 또한 상품명 검색, 추천순/ 신상품순/ 낮은 가격순/ 높은 가격순/ 할인율순/ 후기순/ 판매순으로 정렬하여 조회가 가능하다. 우측 사이드 바 에서는 추천 상품 목록이 제공된다.



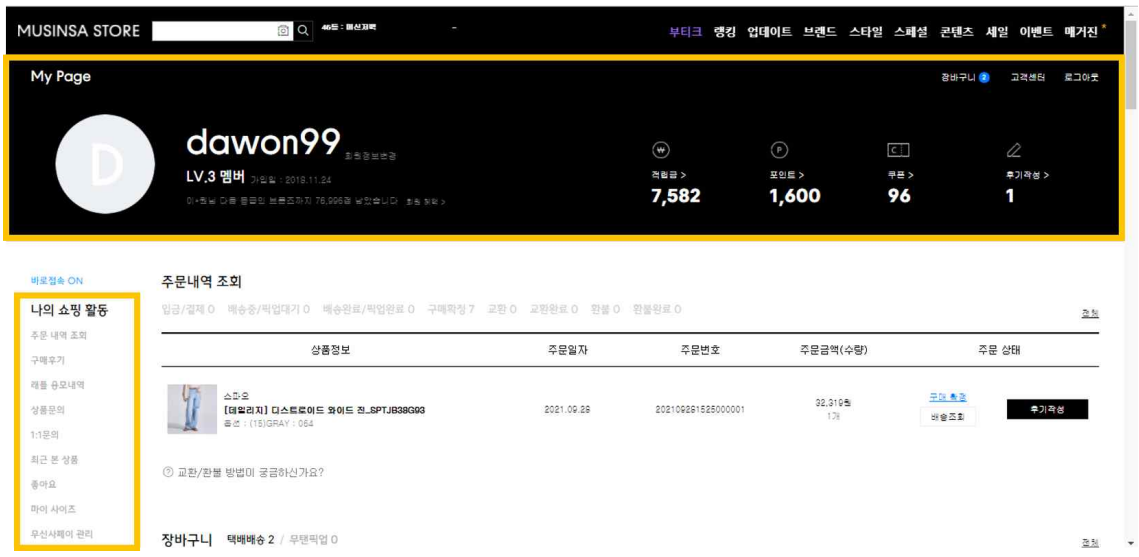
▶ 상품 상세 페이지

: 브랜드, 품번, 시즌, 조회 수, 누적판매, 좋아요, 구매후기, 배송정보, 가격, 사이즈 정보, 바로구매, 찜, 장바구니 버튼, 코디 컷, 상세 사진 등을 확인 할 수 있다.

하단의 구매현황에서는 구매자 연령/성별, 사이즈 기준 표를 한눈에 볼 수 있는 표와 차트가 제공되며 상품 추천, 구매후기-사이즈/ 밝기/ 색감/ 두께감별 별점을 확인 할 수 있다.

▶ 마이페이지

: 마이페이지에서는 아이디, 회원등급, 적립금, 포인트, 쿠폰, 후기작성내역이 조회 가능하다.



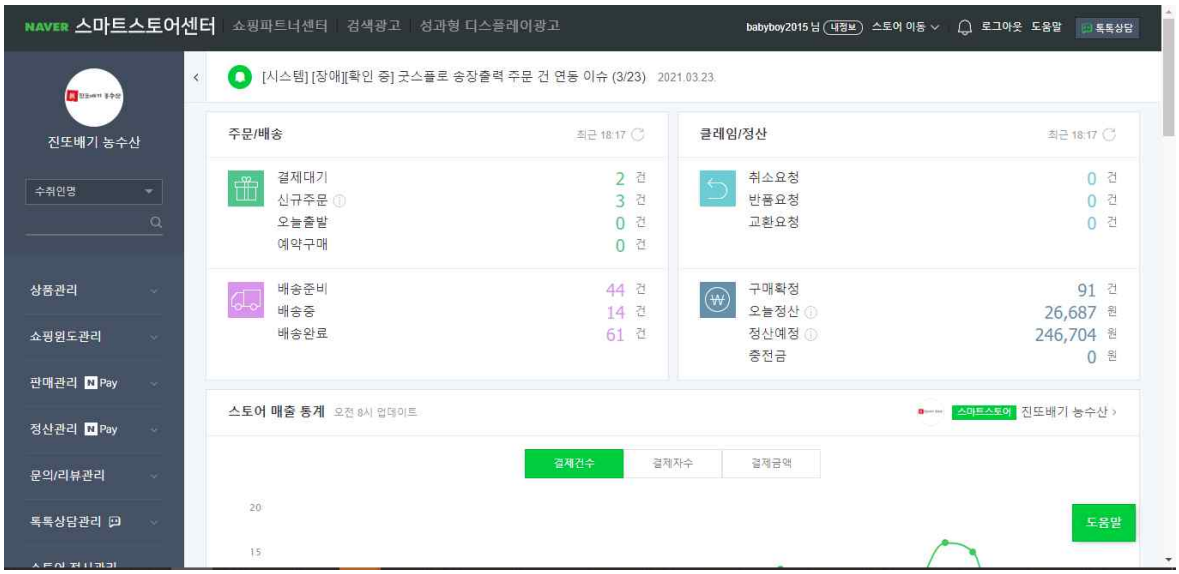
위와 같이 쿠팡, 무신사 사이트를 통해 '구매자' 입장에서의 이커머스 사이트 분석을 할 수 있었다.

하지만 띵켓은 '구매자'가 이용하는 기능뿐만 아니라 '판매자'와 '관리자'가 상품 판매 등을 할 수 있는 기능도 필요한 사이트이기에 '판매자'로서의 사이트 분석도 필

요했다.

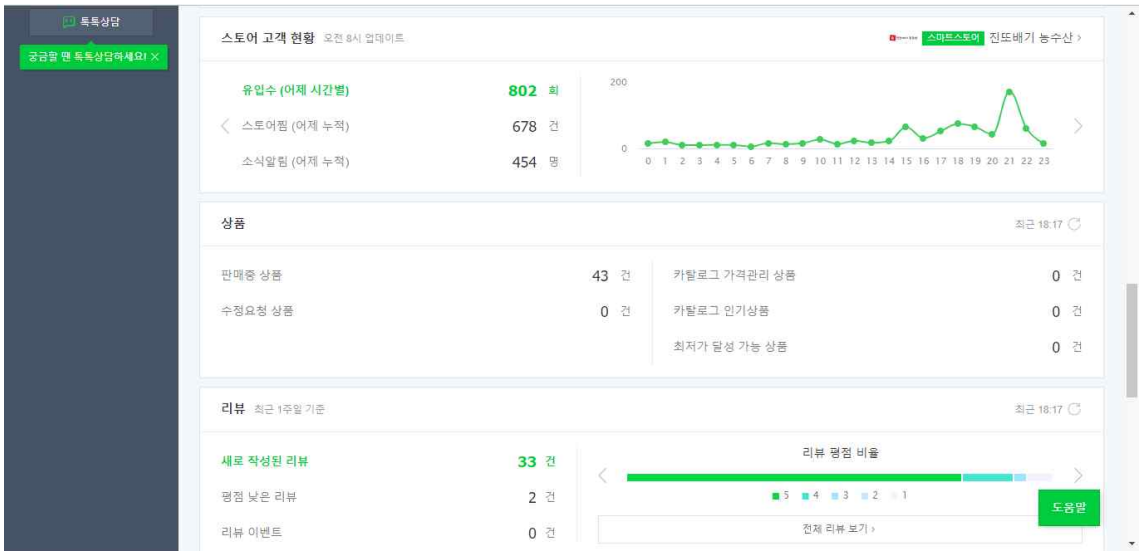
이를 위해 실제 '네이버 스마트스토어'를 이용하여 상품을 판매중인 판매자 페이지 조사를 진행하였다.

2.2.3 네이버 스마트스토어



[스마트스토어센터 메인페이지]
판매자 계정으로 스마트 스토어센터 페이지 진입 시 보여 지는 페이지이다.
크게 주문과 배송, 클레임과 정산의 내역들을 보여준다.

결제건수, 결제자수, 결제금액에 따른 스토어 매출 통계(그래프)를 보여줌으로써 한눈으로 판매자의 실적을 확인할 수 있다.



고객현황과 판매중인 상품의 개수, 리뷰 평점 등을 확인할 수 있다.

좌측의 판매자 카테고리를 분석하여 판매자 페이지에서 필요한 기능을 살펴보았다. 문의와 리뷰 관리, 판매자 정보 관리 및 수정, 상품 조회 및 수정, 상품 등록, 배송 관리, 주문조회, 구매/반품/교환 관리, 판매 분석, 고객 현황조회 등의 기능이 있다는 것을 확인할 수 있었다.

지금까지 세 가지 대표적인 이커머스 사이트 쿠팡, 무신사, 네이버 스마트스토어 페이지 분석을 진행하였다. 이를 통해 '띵켓'의 회원가입 및 로그인 페이지, 메인페이지, 상품 상세 페이지, 마이페이지 제작 및 상품등록, 주문내역관리 등의 이커머스 사이트라면 반드시 갖추어야 할 기능들을 추려냈다.

3. 본론

3.1 서비스 설계

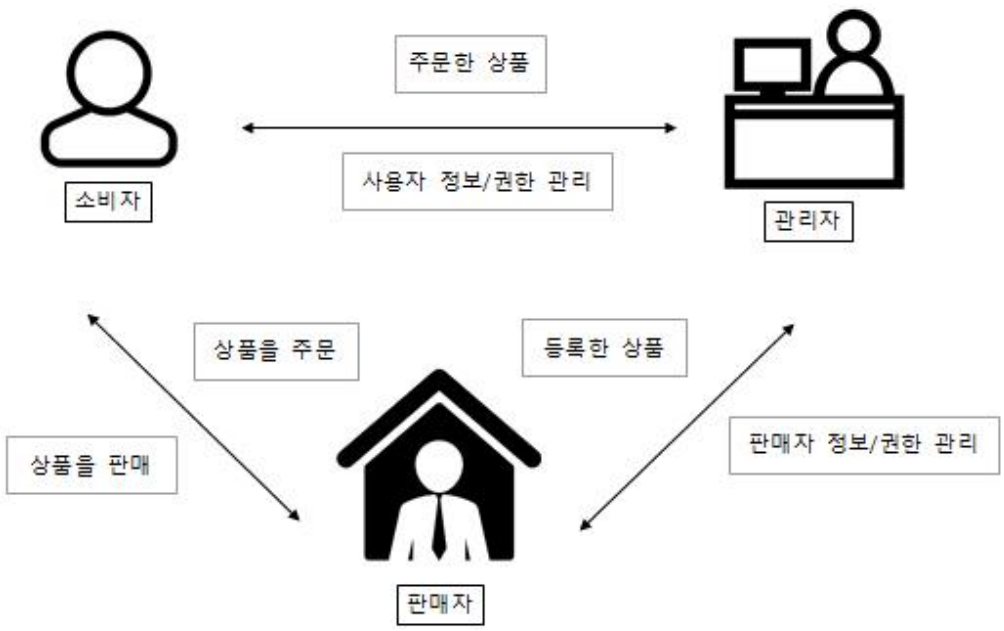
목차 2.2에서 연구하고 조사했던 기존 e-commerce 사이트들의 기능들을 바탕으로 실제 제작할 사이트에 적용할 수 있고 들어갈 만한 기능들을 추린 뒤, 그것을 기준으로 제작을 진행할 수 있도록 서비스 설계를 하였다.

[소비자 기능]

회원가입	로그인	장바구니	상품 검색	카테고리
고객센터	주문(주문내역 확인)	결제	마이페이지	쿠폰 및 혜택
개인정보수정	상품상세페이지	주문 배송 조회	좋아요	최근본 상품
필터링	상품추천	후기(리뷰)	회원등급	적립금

[판매자 기능]

주문 내역 조회	클레임 관리	배송관리
정산	매출통계	상품관리(등록,수정)



위의 표와 그림을 바탕으로 소비자는 장바구니, 상품 주문 기능과 상품 검색 및 필터링 기능, 별점 및 리뷰 기능 등을 이용하여 편리한 상품구매가 이뤄지도록 하고, 판매자는 상품 등록 및 판매, 주문내역 관리 등이 편리하도록 기능을 제공할 수 있도록 하며, 사이트의 관리자는 전체 상품 및 주문내역 관리와 더불어 사용자와 관리자의 정보 및 권한 관리 기능을 제공하는 것을 목표로 하고 프로젝트를 진행했다.

3.2 구현 환경

3.2.1 운영체제

우리는 Windows10 운영체제에서 프로젝트를 시작하였다.

3.2.2 React 앱 생성과정

VSCode 터미널 혹은 cmd창에서 모두 가능하지만 VSCode내 터미널에서 아래와 같은 명령어를 입력하여 React 앱을 생성하였다.

```
npm install create-react-app
```

1) 리액트 프로젝트를 생성해주는 create-react-app 명령어를 이용하기 위해 create-react-app 명령을 설치한다.

```
create-react-app capstone
```

2) create-react-app 명령을 실행하여 capstone이라는 이름의 프로젝트를 생성하였

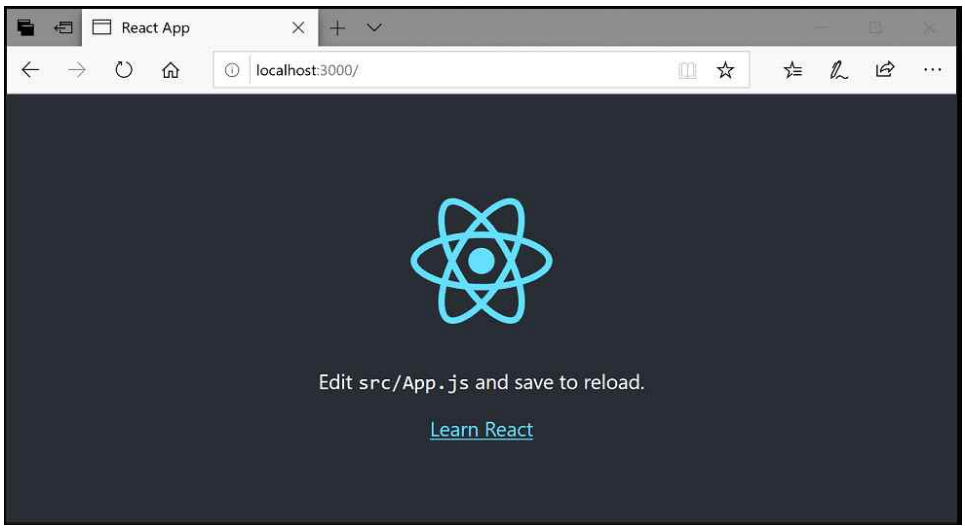
다.

```
cd capstone
```

3) 위에서 생성한 capstone 프로젝트 폴더에 들어간다.

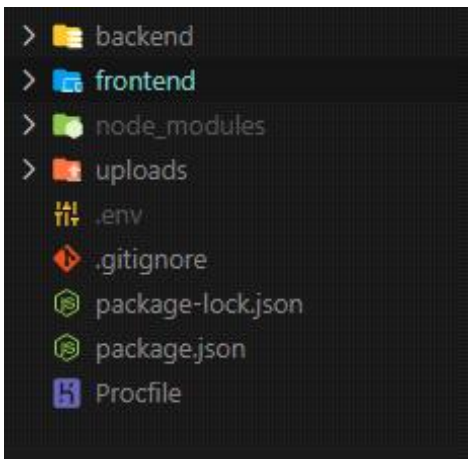
```
npm start
```

4) capstone 프로젝트 폴더에서 npm start 명령을 실행해서 제대로 설치되었는지 확인해보았다.



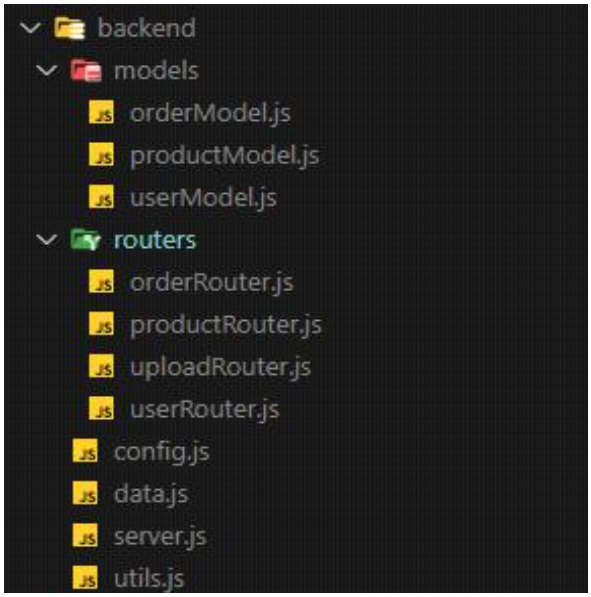
5) npm start 명령을 입력한 후에 뜬 웹브라우저 창이다. 이로서 react-app이 잘 생성되었음을 확인 할 수있다.

3.2.3 폴더구조



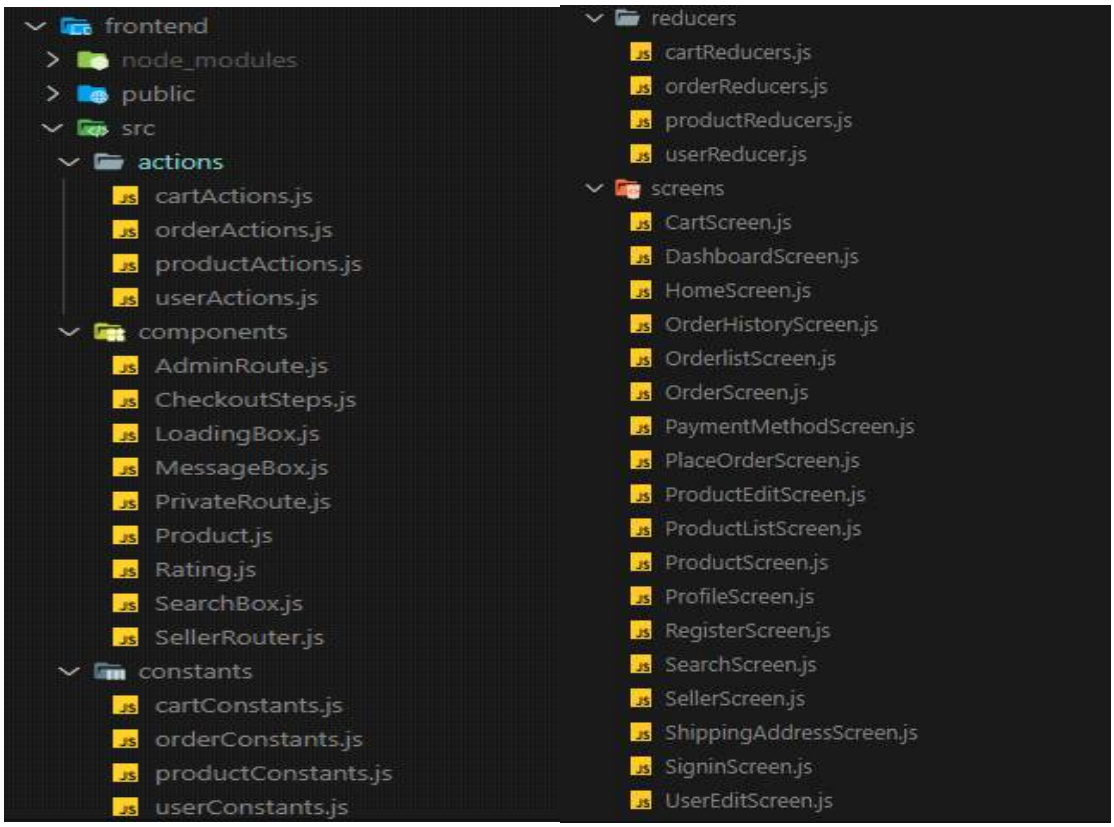
위에서 생성한 React App capstone 폴더 내의 구조이다.
크게 backend, frontend 폴더로 나누어 제작하였다.
'npm install'로 react 앱을 설치하였기 때문에 현재 디렉터리에 node_modules라는 디렉터리가 만들어지고, 그 안에 모듈이 다운로드 된 것이다.

[backend 파일 구조]



backend 파일은 크게 models와 routers 파일로 나누었다.
각각 db스키마 내용이 담긴 model파일들이 담긴 models파일과 서버와 클라이언트의 통신을 위한 인터페이스의 역할을 하는 router파일들이 담긴 routers 폴더이다.

[frontend 파일 구조]

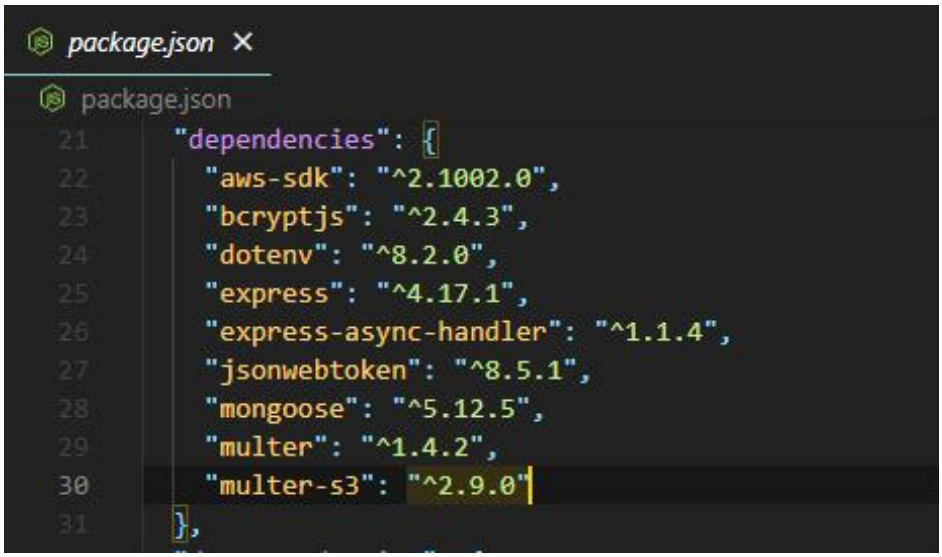


frontend폴더는 크게 actions, components, constants, reducers, screens 폴더로 나

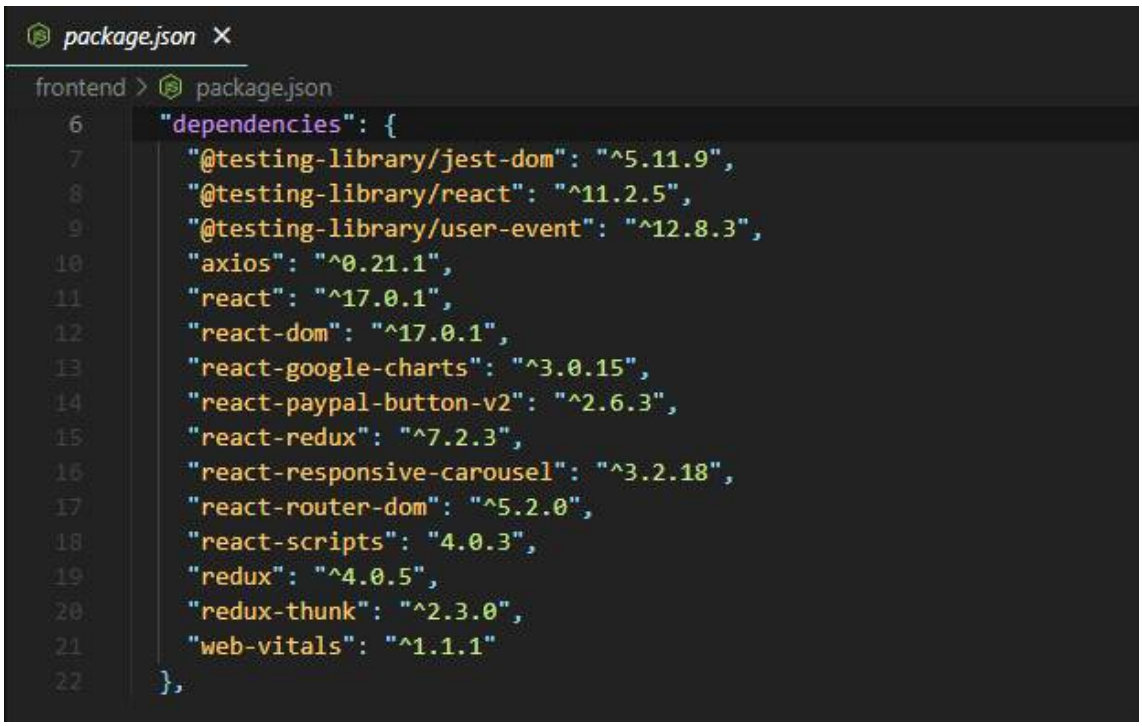
누었다.

3.2.4 설치한 package

터미널에 npm install [패키지명]을 입력하여 설치한 package들을 확인할 수 있다.

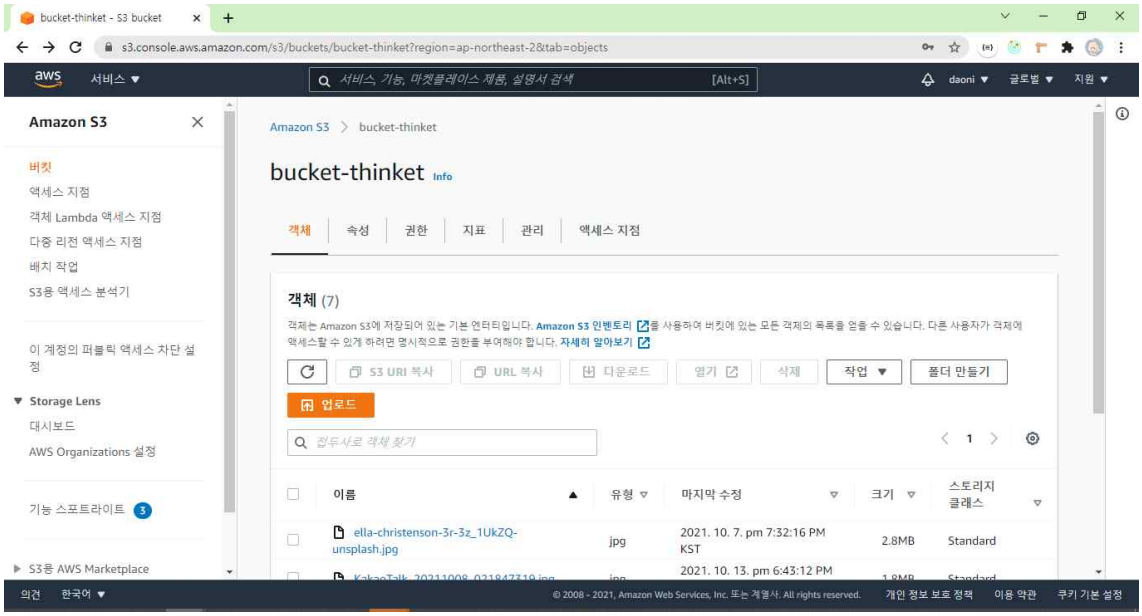


[capstone > package.json]



[capstone > frontend > package.json]

3.2.5 AWS s3



aws s3 서비스를 이용하여 상품 등록 시에 상품 이미지가 업로드 되게 하였다.

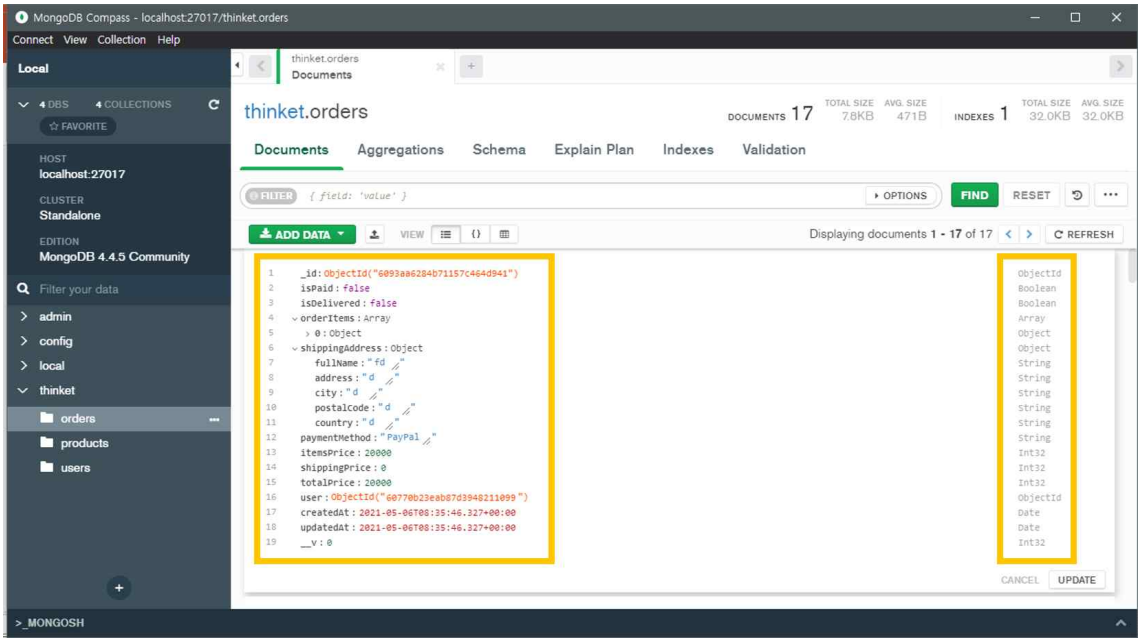
3.3 구현 과정 설명

3.3.1 backend

3.3.1.1 DB 설계

'띵켓'이라는 쇼핑 사이트에서는 구매자와 판매자 계정, 상품(사진, 가격, 설명, 별점 등), 주문내역 등의 다양한 데이터를 다루기 때문에 DB를 각각 주문(orders)/ 상품(products)/ 사용자(users) DB로 세분화하여 구상해보았다.

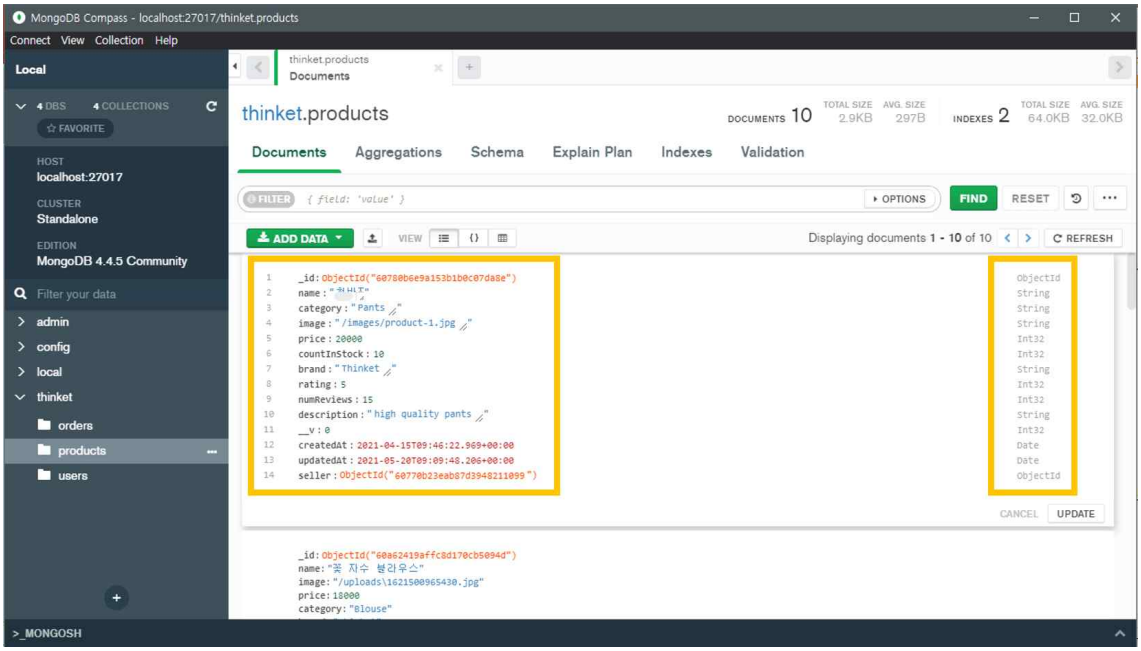
1) orders DB



주문 내역이 저장되는 DB이다. 결제여부, 배송여부, 주문한 상품(상품ID, 상품명, 상품 사진, 가격, 수량), 배송 정보(이름, 주소, 우편번호), 지불방법, 상품가격, 배송비, 총 금액, 주문 시간을 저장한다.

1 각 주문내역에 대한 id를 ObjectId를 이용하여 MongoDB에 데이터를 저장할 때에 자동으로 값이 부여되게끔 하였다. 또한 중복되지 않는 값을 key로 저장해야하기에 ObjectId로 지정하였다.
2,3 결제여부와 배송여부는 참 혹은 거짓의 값만 가져야하니 Boolean 타입을 사용했다.
4,5 주문한 상품은 배열로 저장된다. 주문한 상품이 하나일 경우 array[0]에 데이터가 저장되고 두 개일 경우 array[1]이 추가되는 방식이다.
배열에 저장된 각 객체에는 고유 값을 지니는 id와 상품명, 상품 사진, 상품 가격, 상품id, 수량이 objectid, string, string, int32, objectid, int32 타입 순으로 저장된다.
6, 7-11 배달 주소는 객체로 저장되는데 배송지 정보 입력 시에 기재하는 이름, 도/시, 군/구, 우편번호, 상세주소 모두 문자열인 string값으로 저장된다.
12 결제수단은 string 값으로 저장된다.
13,14,15 상품가격, 배송비, 총 금액의 경우에는 정수 값을 가져야해서 int32 타입으로 지정하였다. int32와 int64는 나타낼 수 있는 값의 범위의 크기의 차이만 지닌 정수형 타입 이기에 int32로 지정하였다.
16 1과 같은 이유로 사용자의 id도 ObjectId로 지정하였다.
17, 18 생성된 날짜, 수정된 날짜는 모두 date로 지정하였다.

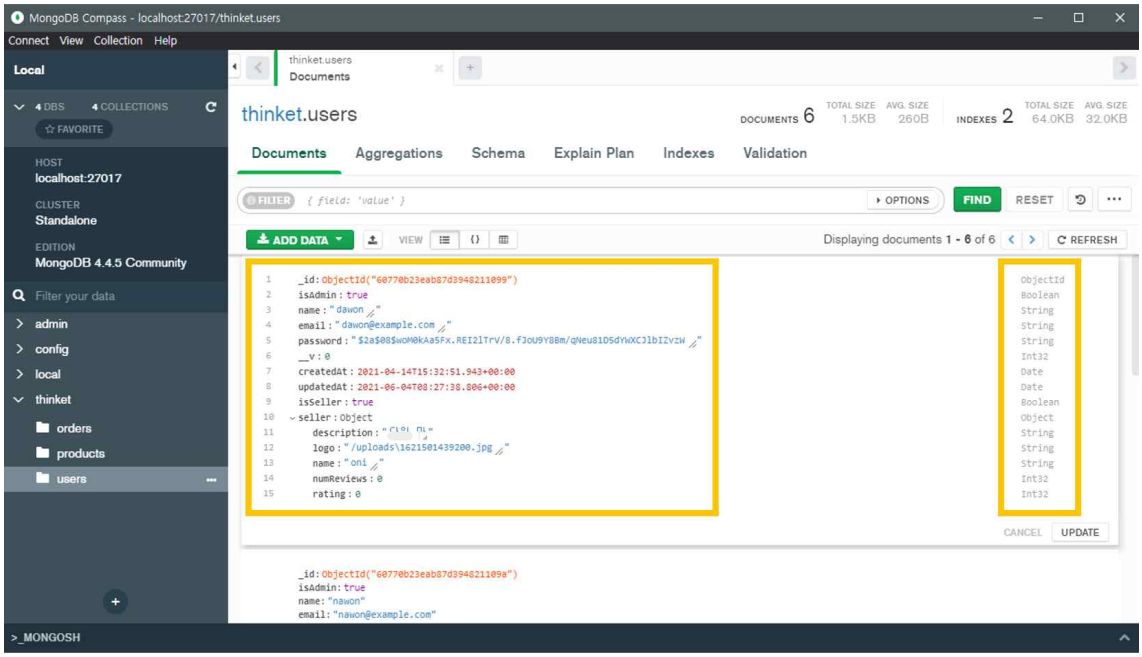
2) products DB



상품명, 카테고리, 상품 사진, 가격, 보유 수량, 브랜드명, 별점, 리뷰개수, 상품설명, 셀러ID를 저장한다.

1 각 상품에 대한 id를 ObjectId를 이용하여 Mongo DB에 데이터를 저장할 때에 자동으로 값이 부여되게끔 하였다. 또한 중복되지 않는 값을 key로 저장해야하기에 ObjectId로 지정하였다.
2,3,4,7,10 상품명, 카테고리, 상품 이미지, 상품 브랜드, 상품의 상세 설명은 문자열인 string 값으로 지정하였다.
5,6,8,9 상품 가격, 수량(재고), 별점, 리뷰개수는 정수 값을 지녀야 하기에 int32로 지정하였다.
12,13 생성 날짜, 수정 날짜는 date로 지정하여 자동으로 생성, 수정 날짜가 저장된다.
14 판매자 역시 중복되지 않는 유일한 값으로 존재하기에 ObjectId로 지정하였다.

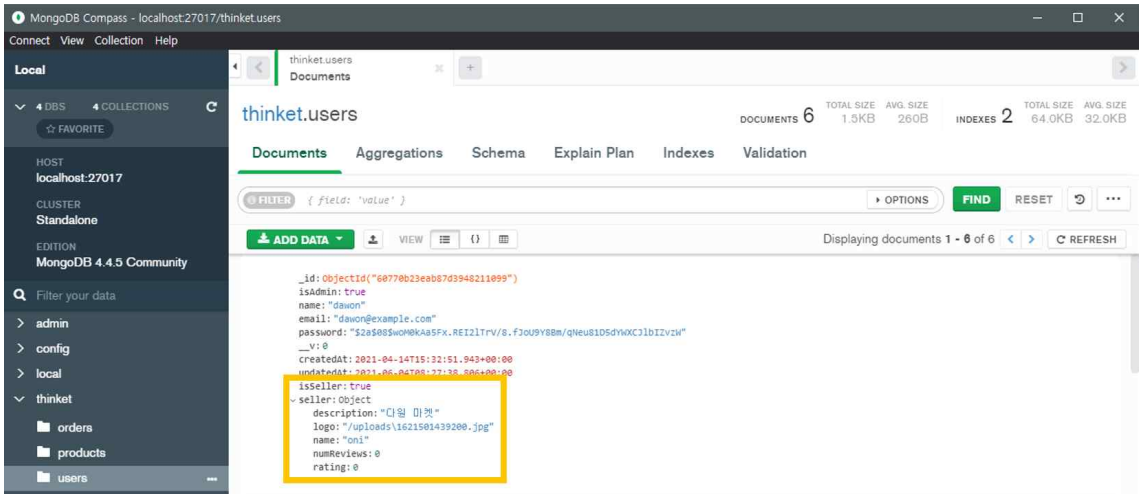
3) users DB



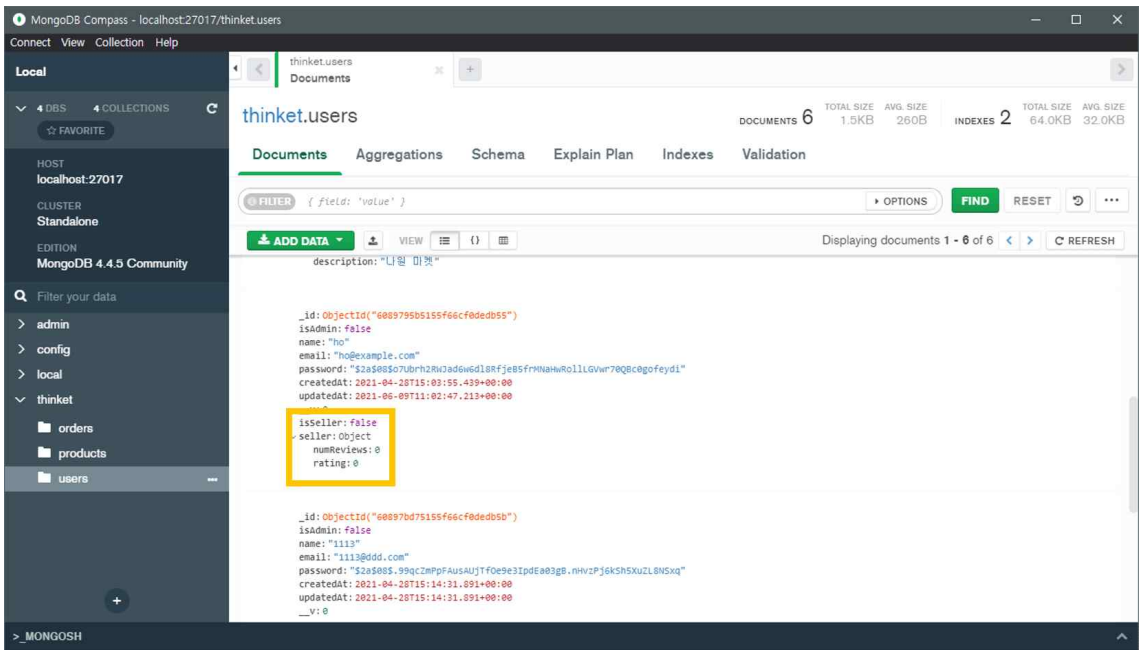
관리자 권한 여부, 이름, 메일 주소, 비밀번호(암호화되어 저장), 판매자 여부가 저장된다.

1 각 사용자에 대한 id를 ObjectId를 이용하여 Mongo DB에 데이터를 저장할 때에 자동으로 값이 부여되게끔 하였다. 또한 중복되지 않는 값을 key로 저장해야하기에 ObjectId로 지정하였다.
2 띵켓에서 'dawon'이라는 회원은 관리자로 지정되어있어서 true로 표시되어 있다. 관리자인지 아닌지에 대한 값은 참 혹은 거짓만 가질 수 있음으로 Boolean 타입을 지정하였다.
3,4,5 이름, 이메일, 비밀번호 는 모두 문자열로 저장되게끔 string으로 지정하였다.
7,8 생성 날짜, 수정 날짜는 date로 지정하여 자동으로 생성, 수정 날짜가 저장된다.
9 이 계정이 셀러인지에 대한 값은 참 혹은 거짓으로만 가져야 하기 때문에 Boolean 타입을 지정하였다.
10 셀러일 경우에만 저장되는 값이다. 판매자에 대한 설명, 로고 이미지, 판매자 혹은 마켓이름은 문자열인 string으로 지정되고 리뷰 개수와 별점은 정수형인 int로 저장된다.

사용자 계정은 판매자가 아닌 경우(isSeller=false)와 판매자가 맞는 경우(isSeller=true)로 나뉘는데 판매자인 경우에는 판매자 설명, 판매자 로고, 판매자 이름, 리뷰개수, 별점이 저장된다.



[판매자 계정일 경우]



[판매자 계정이 아닐 경우]

3.3.1.2 Model

mongoose에서의 model은 데이터베이스(mongo DB)와 연결하는 부분을 담당하고, 데이터베이스에서 데이터를 저장하는 기본 단위인 도큐먼트의 형태를 의미한다.

```
const productSchema = new mongoose.Schema(
    {
        name: {type: String, required: true, unique: true},
        seller: {type:mongoose.Schema.Types.ObjectID, ref: 'User'},
        category: {type: String, required: true},
        image: {type: String, required: true},
        price: {type: Number, required: true},
        countInStock: {type: Number, required: true},
        brand: {type: String, required: true},
        rating: {type: Number, required: true},
        numReviews: {type: Number, required: true},
        description: {type: String, required: true},
        reviews : [reviewSchema]
    },
    {
        timestamps: true,
    }
);

const Product = mongoose.model('Product', productSchema);

export default Product;
```

[Product Model 코드 일부]

프로젝트의 상품에 대한 스키마를 작성한 코드이다. mongoose.Schema로 스키마를 작성해주고, mongoose.model()을 호출해주면 스키마가 등록된다. 이를 이용해서 데이터를 데이터베이스에 넣기 전에 작성한 스키마를 기준으로 먼저 검사한 뒤, 데이터베이스에 저장하게 된다.

3.3.1.3 Router

express에서 router는 서버와 클라이언트의 통신을 위한 인터페이스를 제공해주는 역할을 한다. 따라서 router 파일들은 특정 요청을 받아서 특정 URL이 그 요청에 대해 어떤 기능을 하게 만들 것인지 지정해주는 역할을 해준다고 볼 수 있다.

```
productRouter.get('/:id', expressAsyncHandler(async(req, res) => {
    const product = await Product.findById(req.params.id).populate('seller','seller.name seller.logo seller.rating
    seller.numReviews');
    if(product){
        res.send(product);
    }
    else{
        res.status(404).send({message:'Product Not Found'});
    }
}));

productRouter.post('/', isAuth, isSellerOrAdmin, expressAsyncHandler(async(req, res) => {
    const product = new Product({
        name: 'sample name' + Date.now(),
        seller: req.user._id,
        image: '/images/product-1.jpg',
        price: 0,
        category: 'sample category',
        brand: 'sample brand',
        countInStock: 0,
        rating : 0,
        numReviews: 0,
        description: 'sample description'
    });
    const createdProduct = await product.save();
    res.send({message: 'Product Created', product: createdProduct});
}));
```

[product Router 코드 일부 (get, post)]

```
productRouter.put('/:id', isAuth, isSellerOrAdmin, expressAsyncHandler(async(req,res) => {
    const productId = req.params.id;
    const product = await Product.findById(productId);
    if(product) {
        product.name=req.body.name;
        product.price=req.body.price;
        product.image=req.body.image;
        product.category=req.body.category;
        product.brand=req.body.brand;
        product.countInStock=req.body.countInStock;
        product.description=req.body.description;
        const updatedProduct = await product.save();
        res.send({message:'상품이 업데이트 되었습니다.', product: updatedProduct })
    } else {
        res.status(404).send({message: 'Product Not Found'});
    }
})
);

productRouter.delete('/:id', isAuth, isAdmin, expressAsyncHandler(async (req, res) => {
    const product = await Product.findById(req.params.id);
    if(product) {
        const deleteProduct = await product.remove();
        res.send({message: '상품이 삭제되었습니다', product: deleteProduct});
    } else {
        res.status(404).send({message: 'Product Not Found'});
    }
}));
```

[product Router 코드 일부 (put, delete)]

express.Router()로 생성한 productRouter를 이용해서 각각 해당하는 요청에 대해 어떤 역할들이 수행되어야 하는지를 작성한 코드이다. 모든 부분에 등장하는

expressAsyncHandler는 express router 내부의 예외를 처리하고 express 오류 핸들러로 전달하기 위한 미들웨어 역할을 해준다.

본격적으로 코드를 보면 get / post / put / delete로 다른 의미를 가진다는 것을 알 수 있다. 각각 read / create / update / delete의 의미를 갖는다.

따라서 그 의미에 맞추어서, get 부분에는 id 값에 맞는 내용들을 populate를 이용해서 판매자의 이름, 로고, 별점, 리뷰 수만 가져와 객체로 치환해주고 res.send()로 보내주고, post 부분에는 product model에서 적어주었던 스키마에 맞게 이름, 판매자, 가격, 카테고리 등 상품에 필요한 주요 정보들을 담아서 보내주는 역할을 해준다. 그리고 put 부분에서는 해당 id에 맞는 상품의 내용들을 새롭게 업데이트해서 보내주는 역할을 해주고, 마지막으로 delete 부분에는 원하는 id에 해당하는 상품을 제거하는 역할을 담당하도록 작성하였다.

3.3.1.4 Server

```
app.use('/api/uploads', uploadRouter);

app.use('/api/users', userRouter);

app.use('/api/products', productRouter);

app.use('/api/orders', orderRouter);

app.get('/api/config/paypal', (req, res) =>{
  res.send(process.env.PAYPAL_CLIENT_ID || 'sb');
});

const __dirname = path.resolve();
app.use('/uploads', express.static(path.join(__dirname, '/uploads')));

app.use(express.static(path.join(__dirname, '/frontend/build')));

app.get('*', (req, res) => res.sendFile(path.join(__dirname, '/frontend/build/index.html')));

app.get('/', (req, res) => {
  res.send('Server is ready');
});

app.use((err, req, res, next) => {
  res.status(500).send({message: err.message});
});

const port = process.env.PORT || 5000;

app.listen(port, () => {
  console.log(`Server at http://localhost:${port}`);
});
```

[server.js 코드 일부]

프로젝트의 server.js 파일 안에 들어있는 코드이다. 이 파일 안에는 미들웨어가 모여 있는데, 미들웨어란 요청에 대한 응답 과정 중간에 껴서 어떠한 동작을 해주는 것을 말한다. 즉, 요청과 응답 중간에서 응답을 보내기 전, 미들웨어가 지정된 동작을 수행하는 것이다. express에서는 app.use(미들웨어)로 사용할 미들웨어를 응답을 보내기 전에 넣어주면 된다. 코드에 보이는 express.static()도 미들웨어의 한 종류로, 정적 파일의 기본 경로를 정해주는 역할을 한다. 부연 설명으로 path.join()은 인자로 받은 경로들을 하나로 합쳐서 문자열 형태로 경로를 리턴해주는 역할을 해준다.

3.3.2 frontend

3.3.2.1 Action

프로젝트에서 state에 어떤 변화가 필요할 때, action이라는 것을 발생시키고, 이것은 하나의 객체라고 볼 수 있다. 따라서 action들은 어떤 동작에 대해 선언되어진 객체이다. 아래의 코드를 예로 들어서,

```
export const register = (name, email, password) => async(dispatch) => {
    dispatch({type: USER_REGISTER_REQUEST, payload: { email, password }});
    try {
        const {data} = await Axios.post('/api/users/register', {name, email, password});
        dispatch({type: USER_REGISTER_SUCCESS, payload: data});
        dispatch({type: USER_SIGNIN_SUCCESS, payload: data});
        localStorage.setItem("userInfo", JSON.stringify(data));
    }catch(error) {
        dispatch({
            type: USER_REGISTER_FAIL,
            payload:
                error.response && error.response.data.message
                ? error.response.data.message
                : error.message,
        });
    }
};
```

[user Action 코드 일부]

user Action 파일의 회원가입 부분이다. 백엔드랑 프론트엔드랑 통신을 쉽게 할 수 있도록 axios를 사용했고, dispatch를 사용해서 action을 store에 전달한다. dispatch는 store의 내장 함수 중 하나로 reducer에게 action을 발생하도록 시키는 역할을 한다. 때문에 dispatch의 인수에 reducer로 넘길 객체(type)를 정의 해주어야 한다. 그렇게 호출을 하면 store가 reducer 함수를 실행해, 넘긴 action을 함수가 처리해 새로운 상태를 만들어 준다.

이런 구조를 이용해서 다른 action 파일들도 비슷한 구조로 작성을 해주었다.

```
export const removeFromCart = (productId) => (dispatch, getState) => {
  dispatch ({
    type: CART_REMOVE_ITEM,
    payload: productId});
  localStorage.setItem('cartItems', JSON.stringify(getState().cart.cartItems));
};
```
[cart Action 코드 일부]

```
export const deleteOrder = (orderId) => async(dispatch, getState) => {
  dispatch({type: ORDER_DELETE_REQUEST, payload:orderId});
  const {userSignin:{userInfo}} = getState();
  try {
    const {data} = Axios.delete(`/api/orders/${orderId}`, {
      headers: {Authorization: `Bearer ${userInfo.token}`},
    });
    dispatch({type: ORDER_DELETE_SUCCESS, payload: data});
  } catch(error) {
    const message = error.response && error.response.data.message ? error.response.data.message : error.message;
    dispatch({type: ORDER_DELETE_FAIL, payload: message})
  }
};
```
[order Action 코드 일부]

```
export const createProduct = () => async(dispatch, getState) => {
  dispatch({type: PRODUCT_CREATE_REQUEST});
  const {userSignin: {userInfo}} = getState();
  try {
    const {data} = await Axios.post('/api/products', {}, {
      headers: {Authorization: `Bearer ${userInfo.token}`}
    });
    dispatch({type: PRODUCT_CREATE_SUCCESS, payload: data.product});
  } catch(error) {
    const message =
      error.response && error.response.data.message ? error.response.data.message : error.message;
    dispatch({type: PRODUCT_CREATE_FAIL, payload: message});
  }
};
```
[product Action 코드 일부]

주문내역을 삭제하는 부분과 상품을 생성하는 부분인데, 두 코드가 동일한 구조로 작성되어있는 것을 볼 수 있다. 여기서 getState는 현재 스토어에 있는 내용을 반환해주는 역할을 해준다. 또, headers 부분에 Bearer의 의미는 JWT 혹은 OAuth에 대한 토큰을 사용한다는 의미이다.

3.3.2.2 Constant

constant는 상수라는 뜻으로, 절대 변하지 않는 일정한 값을 말한다.
따라서 바뀌지 않는 값들을 모두 모아놓고 사용 할 수 있도록 분야별로(user, product, order, cart) 파일들을 만들어서 관리해주었다. 위에 action에서 dispatch에

type 부분에 들어갔던 값들이 전부 constant 파일 안에 들어가 있다.

```
export const USER_REGISTER_REQUEST = 'USER_REGISTER_REQUEST';
export const USER_REGISTER_SUCCESS = 'USER_REGISTER_SUCCESS';
export const USER_REGISTER_FAIL = 'USER_REGISTER_FAIL';
```

[user Constant 코드 일부]

```
export const PRODUCT_CREATE_REQUEST = 'PRODUCT_CREATE_REQUEST';
export const PRODUCT_CREATE_SUCCESS = 'PRODUCT_CREATE_SUCCESS';
export const PRODUCT_CREATE_FAIL = 'PRODUCT_CREATE_FAIL';
export const PRODUCT_CREATE_RESET = 'PRODUCT_CREATE_RESET';
```

[product Constant 코드 일부]

위와 같이 constant 값들을 정의 해주는 코드들로 구성되어있는 파일들이다.

3.3.2.3 Reducer

reducer는 현재 상태와 action 객체를 파라미터로 받아와서 새로운 상태를 반환해주는 함수이다. reducer에서 반환하는 상태는 곧 component가 지닐 새로운 상태가 된다.

```
export const userRegisterReducer = (state = {}, action) => {
    switch(action.type) {
        case USER_REGISTER_REQUEST:
            return {loading: true};
        case USER_REGISTER_SUCCESS:
            return {loading: false, userInfo: action.payload};
        case USER_REGISTER_FAIL:
            return {loading: false, error: action.payload};
        default :
            return state;
    }
};
```

[user Reducer 코드 일부]

user의 register 부분 reducer 함수이다. state와 action을 받아서 type 별로 상태를 반환해주고 있는 코드이다. 사용자가 등록을 요청하면 loading 상태를 true로 반환하고, 사용자 등록에 성공 시 loading 상태를 false로 반환하면서 userInfo를 저장한다. 마지막으로 사용자 등록에 실패하면 loading을 false로 반환하고 error를 반환하도록 작성했다. 다른 파일들도 동일하게 작성해주었다.

```
export const orderListReducer = (state = {orders:[]}, action) => {
  switch(action.type) {
    case ORDER_LIST_REQUEST:
      return {loading: true};
    case ORDER_LIST_SUCCESS:
      return {loading: false, orders: action.payload};
    case ORDER_LIST_FAIL:
      return {loading: false, error: action.payload};
    default:
      return state;
  }
};
```

[order Reducer 코드 일부]

```
export const productCreateReducer = (state = {}, action) => {
  switch (action.type) {
    case PRODUCT_CREATE_REQUEST:
      return {loading: true};
    case PRODUCT_CREATE_SUCCESS:
      return {loading: false, success: true, product: action.payload};
    case PRODUCT_CREATE_FAIL:
      return {loading: false, error: action.payload};
    case PRODUCT_CREATE_RESET:
      return {};
    default:
      return state;
  }
};
```

[product Reducer 코드 일부]

3.3.2.4 Component

React에서 component라고 하는 것은 javascript에서의 함수와 유사하다. "props"라고 하는 임의의 입력을 받은 후, 화면에 어떻게 표시되는지를 기술하는 React element를 반환한다. 프로젝트에서 반복되어져 사용되는 함수들을 component로 만들어서 사용해주었다.

```
export default function MessageBox(props) {
  return (
    <div className={`alert alert-${props.variant || 'info'}`}>
      {props.children}
    </div>
  );
}
```
[Message Box component 코드]

MessageBox 컴포넌트의 코드이다. props를 받아서 error 상황 시에 사용자에게 메시지를 띄워주기 위한 컴포넌트로, 여러 가지 발생할 수 있는 error 상황에 쓰이기 때문에 컴포넌트로 만들었다.

```
export default function AdminRoute({component: Component, ...rest}) {
    const userSignin = useSelector(state => state.userSignin);
    const {userInfo} = userSignin;
    return (
        <Route {...rest} render={(props) => userInfo && userInfo.isAdmin ? ( <Component {...props}></Component>):
    (
        <Redirect to="/signin"/>
    )}
    ></Route>
    );
}
```
[Admin Route component 코드]

```
export default function PrivateRoute({component: Component, ...rest}) {
    const userSignin = useSelector(state => state.userSignin);
    const {userInfo} = userSignin;
    return (
        <Route {...rest} render={(props) => userInfo? ( <Component {...props}></Component>):
    (
        <Redirect to="/signin"/>
    )}
    ></Route>
    );
}
```
[Private Route component 코드]

```
export default function SellerRoute({component: Component, ...rest}) {
    const userSignin = useSelector((state) => state.userSignin);
    const {userInfo} = userSignin;
    return (
        <Route {...rest} render={(props) => userInfo && userInfo.isSeller ? ( <Component {...props}></Component>):
    (
        <Redirect to="/signin"/>
    )}
    ></Route>
    );
}
```
[Seller Route component 코드]

그리고 일반 사용자, 판매자, 관리자에 따라서 조건에 맞는 사람만 접근할 수 있도

록 해야 하는 컴포넌트도 있기 때문에, Admin, Seller, Private로 나누어서 조건에 맞는지 확인한 뒤 컴포넌트가 보일 수 있도록 해주었다.

```
export default function Product(props) {
  const {product} = props;
  return (
    <div key={product._id} className="product-card">
      <Link to={`/product/${product._id}`}>
        {/* Product Image */}
        <img className="product-image" src={product.image} alt={product.name}/>
      </Link>
      {/* Card Content */}
      <div className="card-content">
        <Link to={`/product/${product._id}`}>
          <h2>{product.name}</h2>
        </Link>
        <Rating rating={product.rating} numReviews={product.numReviews} />
        {/* Product Price */}
        <div className="row">
          <div className="product-price">
            {product.price}원
          </div>
          <div>
            <Link to={`/seller/${product.seller._id}`}>
              {product.seller.name}
            </Link>
          </div>
```

[Product component 코드]

또, 사이트의 메인페이지에서 상품들이 나열될 때 상품 사진 및 이름, 별점, 가격과 판매자 이름이 공통적으로 들어가기 때문에 컴포넌트로 만들어서 반복적으로 사용하기 편하게 관리해주었다.

3.3.2.5 Screen

Screen은 말 그대로 웹 사이트에서의 하나하나의 화면 들을 의미한다.
우리 웹 사이트에 필요한 화면들은 전부 Screen으로 만들어 주었다.

```
return  (
  <div>
    <h2>베스트 셀러</h2>
    {loadingSellers ? <LoadingBox></LoadingBox>
    :
    errorSellers ? <MessageBox variant="danger">{errorSellers}</MessageBox>
    :
    <>
      {sellers.length === 0 && <MessageBox>No Seller Found</MessageBox>}
      <Carousel showArrows autoPlay showThumbs={false}>
        {sellers.map((seller) => (
          <div key={seller._id}>
            <Link to={`/seller/${seller._id}`}>
              <img src={seller.seller.logo} alt={seller.seller.name}></img>
              <p className="legend">{seller.seller.name}</p>
            </Link>
          </div>
        ))}
      </Carousel>
    </>
    }
    <h2>전체 상품</h2>
    {loading? <LoadingBox></LoadingBox>
    :
    error?<MessageBox variant="danger">{error}</MessageBox>
    :
    <>
      {products.length === 0 && <MessageBox>No Product Found</MessageBox>}
      <div className="row-sorting center">
      {
        products.map((product) => (
        <Product key={product._id} product={product} />
```

[Home Screen 코드 일부]

가장 먼저 사이트에 접속하면 보이는 Home Screen 코드의 일부 이다.
처음 보이는 페이지에는 베스트셀러를 보여주는 carousel 부분이 있는데, LodaingBox 컴포넌트와 MessageBox 컴포넌트를 이용해서 로딩, 에러 발생 시에 해당 컴포넌트로 적절한 대응을 해줄 수 있도록 작성하였다.
그 다음 판매자별로 사진과 이름이 carousel에 나오도록 map을 사용해서 작성해주었다. map은 배열을 받아서 하나하나의 또 다른 배열로 만들어 반환해준다.
그리고 이제 사이트의 상품들이 보이게 해주는 부분이 나오는데, 여기서 이제 만들어 두었던 Product 컴포넌트를 이용해서 상품들을 하나하나 화면에 표현해준다.

```
<div className="col-1">
    <div className="product-card card-content">
        <ul>
            <li>
                <h2>
                    셀러 : <Link to={`/seller/${product.seller._id}`}>
                        {product.seller.seller.name}
                    </Link>
                </h2>
                <Rating rating={product.seller.seller.rating} numReviews={product.seller.seller.
                numReviews}></Rating>
            </li>
            <li>
                <div className="row">
                    <div>가격</div>
                    <div className="price">{product.price}원</div>
                </div>
            </li>
            <li>
                <div className="row">
                    <div>구매가능여부</div>
                    <div>{product.countInStock > 0 ? ( <span className="success">구매가능</
                    span> ) : (
                    <span className="danger">품절</span>)}
                    </div>
                </div>
            </li>
```

[Product Screen 코드 일부]

상품 상세 페이지를 담당하는 코드의 한 부분이다. 상품마다 판매자가 있기 때문에 판매자와 그의 별점을 화면에 표시해주고, 링크를 걸어두어서 특정 판매자가 판매하는 상품만 따로 볼 수 있는 페이지로 이동 할 수 있다.

그리고 상품의 가격과 함께 구매가 가능한지 여부를 보여주는데, 여기서 상품의 수량을 검사해서 수량이 있을 때만 구매가능이라고 표시하고, 없으면 품절로 표시해주도록 했다.

```
<div className="row top">
    <div className="col-2">
        <h1>장바구니</h1>
        {error && <MessageBox variant="danger">{error}</MessageBox>}
        {cartItems.length === 0 ? <MessageBox>장바구니가 비었습니다 <Link to="/">쇼핑 하러가기</Link></
        MessageBox>
        :
        (
            <ul>
                {
                    cartItems.map((item) => (
                        <li key={item.product}>
                            <div className="row">
                                <div>
                                    <img src={item.image} alt={item.name} className="small" />
                                </div>
                                <div className="min-30">
                                    <Link to={`/product/${item.product}`}>{item.name}</Link>
                                </div>
                                <div>
                                    <select value={item.qty} onChange={(e) => dispatch(addToCart(item.
                                    product, Number(e.target.value)))}>
                                        {
                                            [...Array(item.countInStock).keys()].map( (x) => (
                                                <option key={x+1} value={x+1}>{x+1}</option>
                                            ))
                                        }
                                    </select>
                                </div>
                                <div>
                                    {item.price}원
                                </div>
```

[Cart Screen 코드 일부]

장바구니 페이지를 담당하는 코드 일부이다. 장바구니에 들어갈 cartItem이 존재하지 않으면 "장바구니가 비었습니다." 라고 표시해주고, cartItem이 존재하면 상품의 이미지와 상품명, 수량과 가격을 화면에 나타내준다. 이때, 장바구니에서 상품의 수량은 변경이 가능하도록 작업 해주었다.

```
{
  cart.cartItems.map((item) => (
    <li key={item.product}>
      <div className="row">
          <div>
              <img src={item.image} alt={item.name} className="small" />
          </div>
          <div className="min-30">
              <Link to={`/product/${item.product}`}>{item.name}</Link>
          </div>
          <div>
              {item.qty} x {item.price}원 = {item.qty * item.price}원
          </div>
      </div>
    </li>
  ))
}
```

[Place Order Screen 코드 일부]

장바구니에 주문할 상품을 담고 주문하기를 클릭해서 넘어가면 로그인을 한 사용자에 한해서, 사용자 자신의 배송 정보를 입력하고, 원하는 결제 수단을 선택하게 된다. 그 다음 최종 주문을 진행하기 전, 자신이 입력한 배송정보와 결제수단, 장바구니에 담았던 상품들을 확인 할 수 있도록 보여주고(그림 5.4), 그 다음 다시 주문하기 버튼을 눌러 결제를 진행할 수 있는 주문 페이지로 넘어간다.

```
{
  !order.isPaid && (
    <li>
      {!sdkReady? (<LoadingBox></LoadingBox>) :
      (
        <>
        {errorPay && (<MessageBox variant="danger">{errorPay}</MessageBox>)}
        {loadingPay && <LoadingBox></LoadingBox>}
        <PayPalButton
        amount={order.totalPrice} onSuccess={successPaymentHandler}>
        </PayPalButton>
        </>
      )}
    </li>
  )
}
```

[Order Screen 코드 일부]

위에서 봤던 결제 전 주문 확인 페이지와 유사하게 작성한 결제 진행 페이지이다. 이 페이지 역시 사용자가 입력한 배송 정보와 결제 수단, 그리고 최종적으로 들어간 상품 주문 내역을 확인하고 결제할 수 있도록 페이지 내용이 구성되어있다. 지불할 가격을 확인하고 Paypal 버튼(그림 5.5)이 추가되어 결제 진행이 이루어지는 페이지이다.

```
<h1>주문 내역</h1>
{loading ? <LoadingBox></LoadingBox>:
error ? <MessageBox variant="danger">{error}</MessageBox> : (
  <table className="table">
    <thead>
      <tr>
        <th>주문번호</th>
        <th>주문날짜</th>
        <th>총 금액</th>
        <th>결제여부</th>
        <th>배송여부</th>
        <th>상세보기</th>
      </tr>
    </thead>
    <tbody>
      {orders.map((order) => (
        <tr key={order._id}>
          <td>{order._id}</td>
          <td>{order.createdAt.substring(0, 10)}</td>
          <td>{order.totalPrice}</td>
          <td>{order.isPaid ? order.paidAt.substring(0, 10): '결제 전'}</td>
          <td>{order.isDelivered ? order.deliveredAt.subString(0, 10): '배송 전'}</td>
          <td>
            <button type="button" className="small" onClick={() => {props.history.push(`/order/${order._id}`)}
            }>
              상세정보
            </button>
          </td>
        </tr>
      ))}
    </tbody>
```

[사용자 용 Order History Screen 코드 일부]

사용자가 주문을 진행 했으니까 그 내역을 확인 할 수 있도록 주문내역 페이지를 제작했다. 위의 코드는 사용자가 자신이 주문한 내역을 확인 할 수 있는 주문 내역 페이지의 코드이다. 주문번호와 날짜, 금액 및 결제여부와 배송여부 까지 주문한 정보를 확인할 수 있도록 제작했다.

```
<h1 className="orderlist">주문내역</h1>
    {loadingDelete && <LoadingBox></LoadingBox>}
    {errorDelete && <MessageBox variant="danger">{errorDelete}</MessageBox>}
    {loading ? <LoadingBox></LoadingBox>:
    error ? <MessageBox variant="danger">{error}</MessageBox> : (
<table className="table">
<thead>
    <tr>
    <th>주문번호</th>
    <th>고객</th>
    <th>주문날짜</th>
    <th>총 금액</th>
    <th>결제여부</th>
    <th>배송여부</th>
    <th>상세보기</th>
    </tr>
</thead>
<tbody>
    {orders.map((order) => (
    <tr key={order._id}>
        <td>{order._id}</td>
        <td>{order.user.name}</td>
        <td>{order.createdAt.substring(0, 10)}</td>
        <td>{order.totalPrice}</td>
        <td>{order.isPaid ? order.paidAt.substring(0, 10): '결제 전'}</td>
        <td>{order.isDelivered ? order.deliveredAt.subString(0, 10): '배송 전'}</td>
        <td>
        <button type="button" className="small" onClick={() => {props.history.push(`/order/${order.
        _id}`)}}>
            상세정보
```

[판매자 및 관리자용 Order List Screen 코드 일부]

사용자가 주문한 내역을 판매자와 관리자 또한 꼭 확인 할 수 있어야 하므로, 판매자와 관리자용으로 페이지를 따로 제작해주었다.
사용자용 주문 내역 페이지와 굉장히 유사하지만, 판매자와 관리자용에는 상품을 주문한 고객이 누구인지 추가적으로 보여줄 수 있도록 해주었다.

```
<tbody>
  {products.map((product) => (
    <tr key={product._id}>
      <td>{product._id}</td>
      <td>{product.name}</td>
      <td>{product.price}</td>
      <td>{product.category}</td>
      <td>{product.brand}</td>
      <td>
        <button
          type="button"
          className="small"
          onClick={() =>
            props.history.push(`/product/${product._id}/edit`)
          }
        >
          수정
        </button>
        <button
          type="button"
          className="small"
          onClick={() => deleteHandler(product)}
        >
          삭제
        </button>
```

[판매자와 관리자용 Product List Screen 코드 일부]

판매자와 관리자용 상품 목록 페이지 코드 일부이다. 등록한 상품들을 한눈에 보고, 관리 할 수 있도록 상품의 이름, 가격, 카테고리, 브랜드 정보를 확인할 수 있고, 수정 버튼을 누르면 수정 페이지로 넘어가도록 페이지를 구성했다. 또 등록했던 상품을 삭제하고 싶을 수도 있기 때문에 삭제 기능까지 포함해서 제작했다

```
<div>
  <label htmlFor="price">상품 가격</label>
  <input id="price" type="text" placeholder="가격을 입력하세요" value={price} onChange={(e) => setPrice
  (e.target.value)}></input>
</div>
<div>
  <label htmlFor="image">상품 이미지</label>
  <input id="image" type="text" placeholder="상품 이미지를 입력하세요" value={image} onChange={(e) =>
  setImage(e.target.value)}></input>
</div>
<div>
  <label htmlFor="imageFile">상품 이미지 파일 </label>
  <input id="imageFile" type="file" label="이미지를 골라주세요" onChange={uploadFileHandler}></input>
  {loadingUpload && <LoadingBox></LoadingBox>}
  {errorUpload && <MessageBox variant="danger">{errorUpload}</MessageBox>}
</div>
<div>
  <label htmlFor="category">상품 카테고리</label>
  <input id="category" type="text" placeholder="상품 카테고리를 입력하세요" value={category} onChange={
  (e) => setCategory(e.target.value)}></input>
</div>
<div>
  <label htmlFor="brand">상품 브랜드</label>
  <input id="brand" type="text" placeholder="상품 브랜드 입력하세요" value={brand} onChange={(e) =>
  setBrand(e.target.value)}></input>
</div>
<div>
  <label htmlFor="countInStock">상품 수량</label>
  <input id="countInStock" type="text" placeholder="상품 수량을 입력하세요" value={countInStock}
  onChange={(e) => setCountInStock(e.target.value)}></input>
</div>
<div>
```

[판매자용 Product Edit Screen 코드 일부]

판매자는 등록한 상품을 관리하는 것과 더불어서 상품의 정보를 수정 가능하도록 상품 수정 페이지를 만들어 주었다. 상품명과 가격, 이미지, 카테고리와 수량까지 변경 가능하도록 기능을 추가했다. 이 Product Edit 페이지는 상품 등록할 때와 똑같은 폼을 가지고 있기 때문에 그대로 상품 등록 페이지로도 활용을 해주었다.

[Seller Screen 코드 일부]

상품의 상세 페이지에서 해당 상품을 판매하는 판매자를 확인할 수 있는데, 위의 그림 5.10은 그 판매자를 클릭하면 해당 판매자가 판매하는 상품만 볼 수 있는 판매자 상품 페이지를 담당하는 코드이다. 한쪽에 판매자의 로고와 이름, 판매자에 대한 별점과 설명을 보여주고, 그 옆에 이제 판매자가 등록한 상품들을 나열해서 보여주도록 페이지를 구성하였다. 여기서 상품을 보여줄 때, 그 전에 만들어 두었던 Product 컴포넌트를 이용해서 화면에 상품들을 나열해주었다.

```
  <div>
    Sort by{' '}
    <select value={order} onChange={(e) => {
      props.history.push(getFilterUrl({ order: e.target.value }));
    }}>
      <option value="newest">신상품순</option>
      <option value="lowest">저가순</option>
      <option value="highest">고가순</option>
      <option value="toprated">인기순</option>
    </select>
  </div>
</div>
<div className="row top">
  <div className="col-1">
    <h3>카테고리</h3>
    <div>
      {loadingCategories ? (
        <LoadingBox></LoadingBox>
      ) : errorCategories ? (
          <MessageBox variant="danger">{errorCategories}</MessageBox>
        ) : (
            <ul>
              <li>
                <Link className={'all' === category ? 'active' : ''} to={getFilterUrl({category: 'all'})}
                >모든상품</Link>
              </li>
              {categories.map((c) => (
                <li key={c}>
                  <Link className={c === category ? 'active' : ''} to={getFilterUrl({ category: c })}>
                    {c}
                  </Link>
                </li>
```

[Search Screen (검색 결과) 코드 일부]

메인페이지에 있는 검색 창에 검색할 단어를 입력하고 검색을 진행하면 볼 수 있는 검색 결과 페이지의 코드 일부이다. 검색한 단어에 해당되는 상품을 보여주는데, 사용자가 원하는 방식으로 상품이 정렬될 수 있도록 신상품순, 저가순, 고가순, 인기순으로 나누어서 상품들을 정렬할 수 있도록 해주었다. 그리고 검색 결과 중에서 원하는 카테고리, 가격, 별점으로도 또 나누어서 볼 수 있도록 필터링 기능을 제작했다.

```
<h1>사용자 목록</h1>
{loadingDelete && <LoadingBox></LoadingBox>}
{errorDelete && <MessageBox variant="danger">{errorDelete}</MessageBox>}
{successDelete && <MessageBox variant="success">사용자가 성공적으로 삭제되었습니다</MessageBox>}
{
  loading ? (<LoadingBox></LoadingBox>)
  : error ? (<MessageBox variant="danger">{error}</MessageBox>)
  :
  (
    <table className="table">
      <thead>
        <tr>
          <th>ID</th>
          <th>이름</th>
          <th>이메일</th>
          <th>셀러 여부</th>
          <th>관리자 여부</th>
          <th>상태</th>
        </tr>
      </thead>
      <tbody>
        {users.map((user) => (
          <tr key={user._id}>
            <td>{user._id}</td>
            <td>{user.name}</td>
            <td>{user.email}</td>
            <td>{user.isSeller ? 'O' : 'X'}</td>
            <td>{user.isAdmin ? 'O' : 'X'}</td>
            <td>
              <button type="button" className="small" onClick={() => props.history.push(`/user/${user._id}/
              edit`)}>수정</button>
              <button type="button" className="small" onClick={() => deleteHandler(user)}>삭제</button>
```

[관리자용 User List Screen 코드 일부]

```
<form className="form" onSubmit={submitHandler}>
    <div>
        <h1>사용자 수정 {name}</h1>
        {loadingUpdate && <LoadingBox></LoadingBox>}
        {errorUpdate && (<MessageBox variant="danger">{errorUpdate}</MessageBox>)}
    </div>
    {loading? (<LoadingBox/> ):
    error? (<MessageBox variant="danger"> {error} </MessageBox> ): (
    <>
    <div>
        <label htmlFor="name">이름</label>
        <input id="name" type="text" placeholder="이름을 입력해주세요" value={name} onChange={(e) =>
        setName(e.target.value)}></input>
    </div>
    <div>
        <label htmlFor="email">이메일</label>
        <input id="email" type="email" placeholder="이메일을 입력해주세요" value={email} onChange={(e)
        => setEmail(e.target.value)}></input>
    </div>
    <div>
        <label htmlFor="isSeller">셀러</label>
        <input id="isSeller" type="checkbox" checked={isSeller}  onChange={(e) => setIsSeller(e.target.
        checked)}></input>
    </div>
    <div>
        <label htmlFor="isAdmin">관리자</label>
        <input id="isAdmin" type="checkbox" checked={isAdmin}  onChange={(e) => setIsAdmin(e.target.
        checked)}></input>
    </div>
```

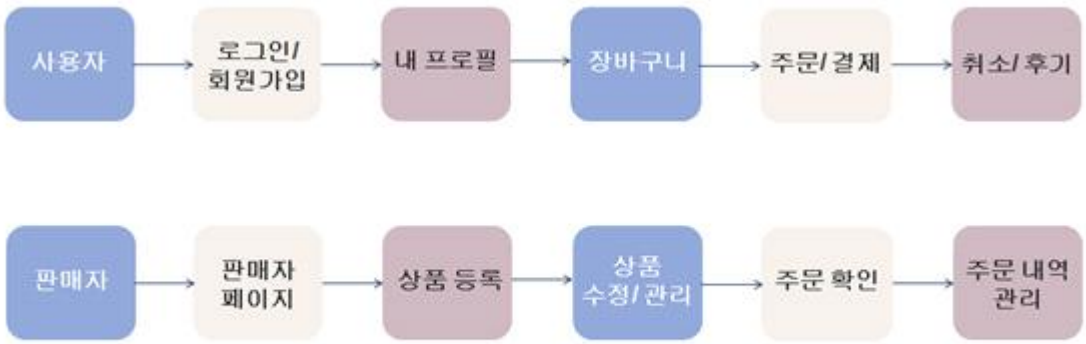
[관리자용 User Edit Screen 코드 일부]

웹 사이트의 관리자는 모든 사용자를 관리할 수 있어야 하므로, 사용자 목록 페이지(그림 5.12)를 만들어서 사용자들의 정보와 권한을 관리할 수 있도록 하였다. 사용자의 ID와 이름, 이메일 주소 같은 기본적인 정보들과 판매자인지 여부, 관리자인지 여부를 포함한 정보까지 확인 할 수 있도록 표로 정리해서 보여주는 페이지이다. 여기서 관리자는 사용자 정보 수정(그림 5.13)을 통해서, 한 사용자에게 판매자의 권한을 줄 수도 있고, 관리자 권한을 줄 수도 있다. 이런 권한 문제는 관리자권한을 가진 사람이 설정할 수 있어야 한다고 생각해서 이 부분에 제작 해주었다.

3.4 주요 기능 데모

3.4.1 데모 시나리오

띵켓 사이트의 데모 시나리오이다.



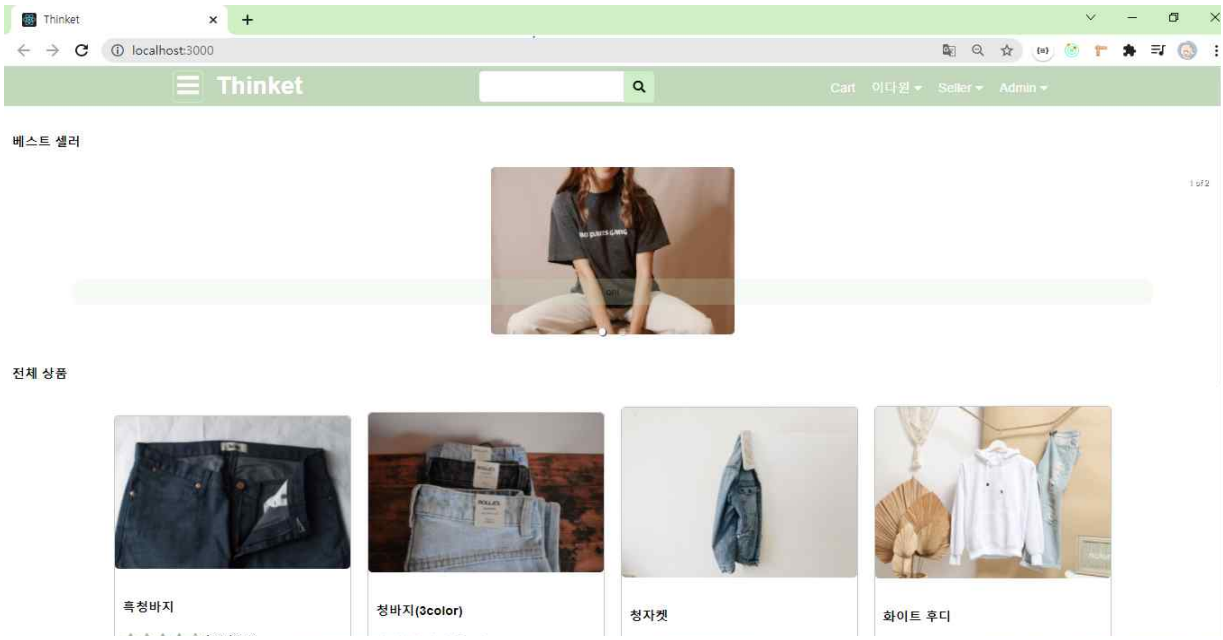
먼저 소비자 계정으로 회원가입을 하고 로그인 한 뒤, 내 프로필 페이지로 들어가서 프로필을 수정할 수 있음을 보여주고, 메인 페이지의 상품들의 상세 페이지를 둘러보고, 상품 검색 및 필터링 기능을 활용한 뒤, 주문할 상품을 장바구니에 담고 주문 단계로 넘어간다. 그 후 배송지를 입력하고, 결제 수단을 선택하는 과정을 거친 뒤 주문에 성공한다. 주문이 완료되면, 자신의 주문내역 페이지로 가서 주문한 내역이 잘 들어가 있는지 확인하고, 배송이 완료되었다는 가정 하에 리뷰를 남기고 마무리 짓는다.

다음으로 판매자로 등록 되어있는 계정으로 로그인하고, 상품 내역 페이지로 가서 자신이 등록해놓은 상품 내역을 확인 판매하고자하는 상품을 등록해본다. 등록한 상품에 대한 수정도 진행해본 뒤, 등록한 상품이 메인 페이지에 잘 올라가 있는지 확인하고, 등록한 상품의 상세 페이지에서 판매자 이름을 클릭해 판매자 페이지도 확인해본다. 그리고 판매하는 상품을 주문한 소비자가 있는지 주문내역 페이지를 통해 확인 할 수 있음을 보여주고, 상세 정보도 확인하여 마무리한다.

마지막으로 관리자 계정으로 로그인 한 뒤, 관리자의 대시보드 페이지에서 볼 수

있는 총 사용자 수와 주문 수, 판매 금액을 확인한다. 그리고 상품 내역과 주문 내역 페이지로 가서 관리자는 판매자 구분 없이 모든 내역을 확인 할 수 있음을 보여주고, 마지막으로 사용자 관리에서 사용자들의 정보 확인과, 판매자 권한을 직접 부여해 해당 계정에 권한이 잘 적용되는지 확인해보고 마무리 짓는 것으로 시나리오를 짜봤다.

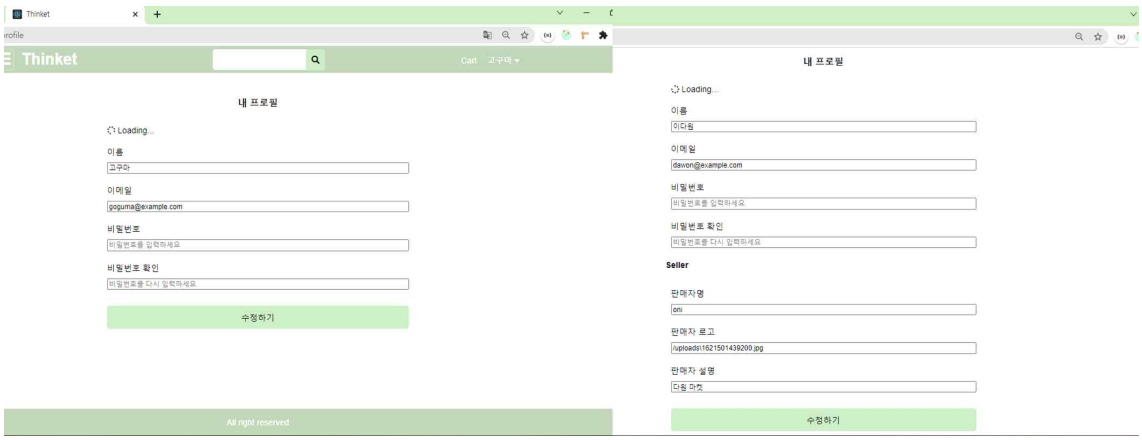
3.4.2 기능 데모



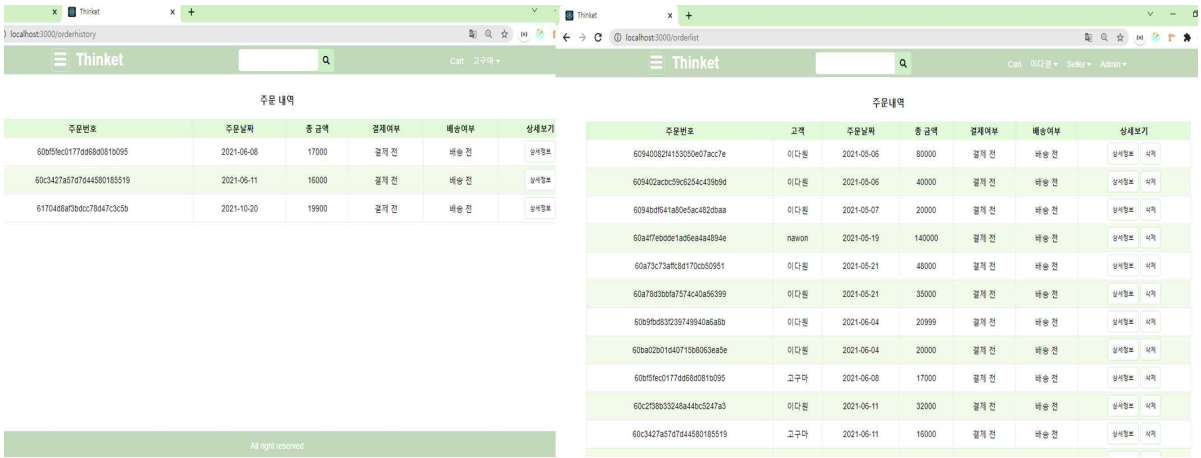
[메인 페이지]
띵켓의 메인 페이지이다. 상단에는 차례로 카테고리 바, 로고, 검색창, 장바구니, 상단바로 구성되어 있고, 중앙에는 각 셀러별 대표 이미지를 슬라이드로 확인 할 수있고, 밑으로는 전체 상품을 확인할 수 있다.

[상단바]
좌측부터 각각 사용자, 셀러, 관리자의 상단 토글바이다. '이다원' 계정은 셀러이자 관리자 계정으로 등록되어 있기 때문에 모두 확인 할 수 있다.
사용자 상단 바에서는 내 프로필, 주문내역, 로그아웃 기능을 이용할 수 있다.
셀러의 상단 바에서는 상품목록, 주문내역을 확인할 수 있다.

관리자 상단 바에서는 대시보드, 상품목록, 주문내역, 사용자 목록을 확인할 수 있다.



[프로필]
좌측부터 각각 일반 사용자, 셀러의 내 프로필 화면이다.
일반 사용자는 이름, 이메일, 비밀번호 변경이 가능하다.
셀러는 사용자와 동일하게 이름, 이메일, 비밀번호 변경이 가능하며 셀러의 판매자명, 로고, 설명을 추가로 변경할 수 있다.

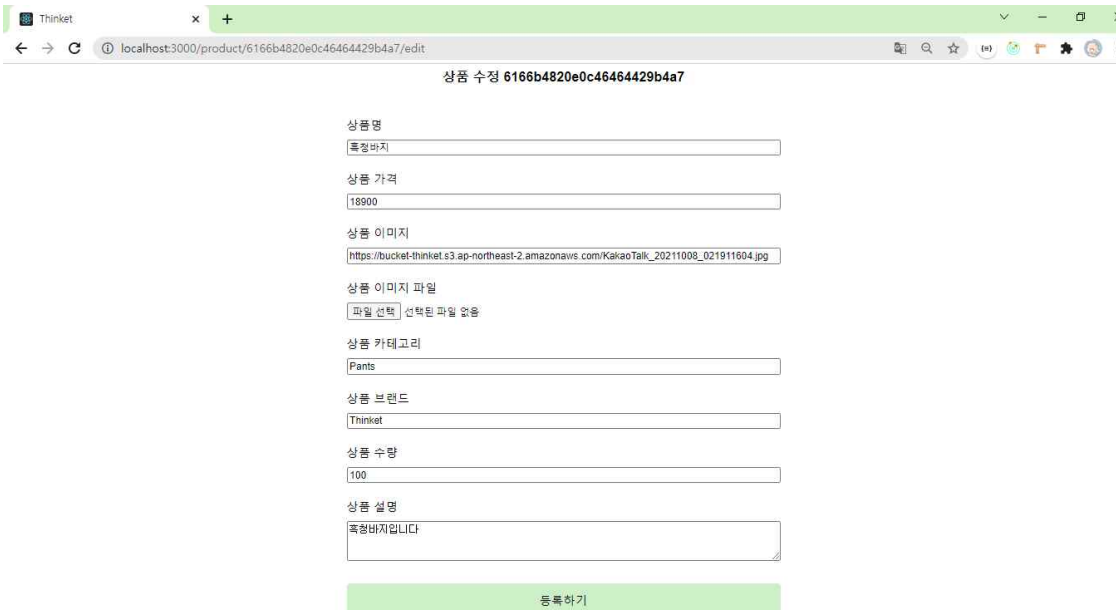


[주문내역]
좌측부터 각각 일반 사용자, 셀러의 주문내역 화면이다.
일반 사용자는 자신이 주문한 내역에 관해서만 볼 수 있고, 셀러는 자신이 등록한 상품에 들어온 주문 내역을 확인 할 수 있다. 사진에는 없지만 관리자 또한 주문내역 확인이 가능한데 관리자는 띵켓 내에 생긴 모든 주문 내역을 확인 할 수 있다.

[상품 목록 페이지]

셀러의 상품 목록 페이지이다.

상품 id, 상품명, 가격, 카테고리, 브랜드, 상품 수정 및 삭제, 추가가 가능하다.



[상품 추가 및 수정]

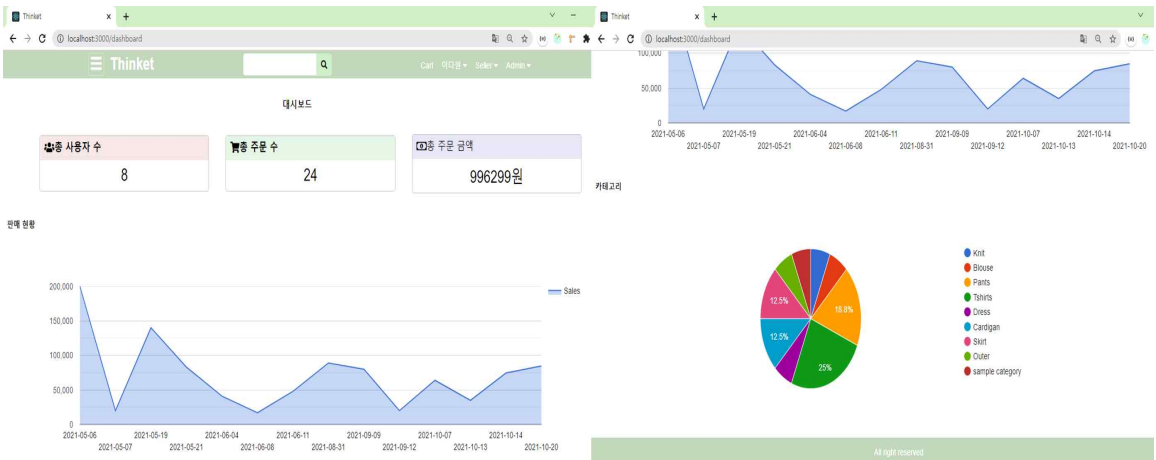
상품 추가 및 수정을 눌렀을 때 보여 지는 페이지이다.

상품명, 가격, 상품이미지, 상품 카테고리, 브랜드, 수량, 설명을 추가 및 수정할 수 있다.

[사용자 목록]

관리자의 사용자 목록 페이지이다.

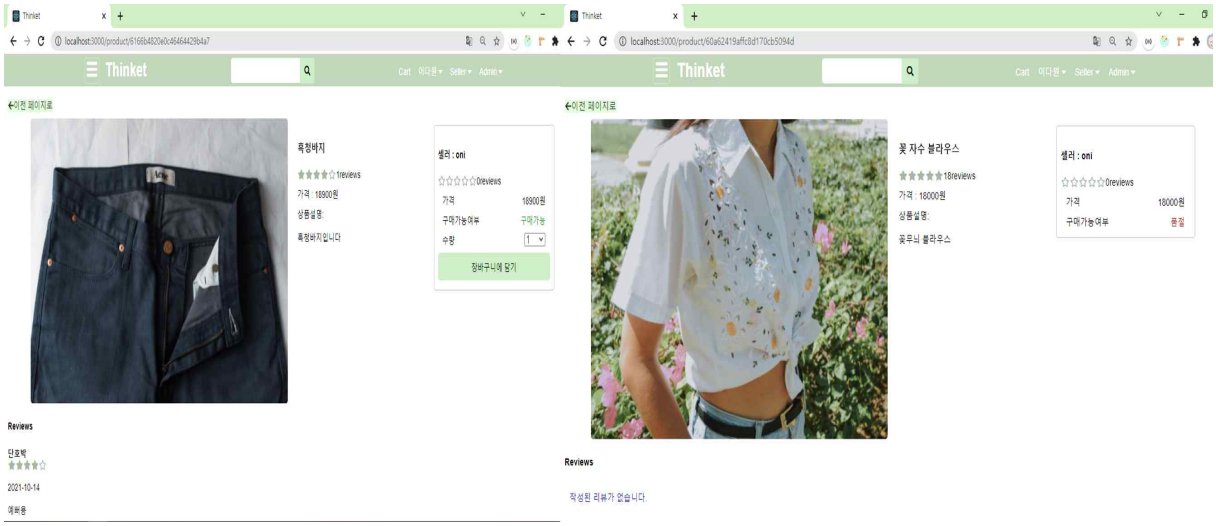
띵켓에 회원가입 된 모든 사용자의 이름, 이메일을 확인 할 수 있으며 수정을 통해 셀러 혹은 관리자 권한을 부여할 수 있고 계정 삭제 또한 가능하다.



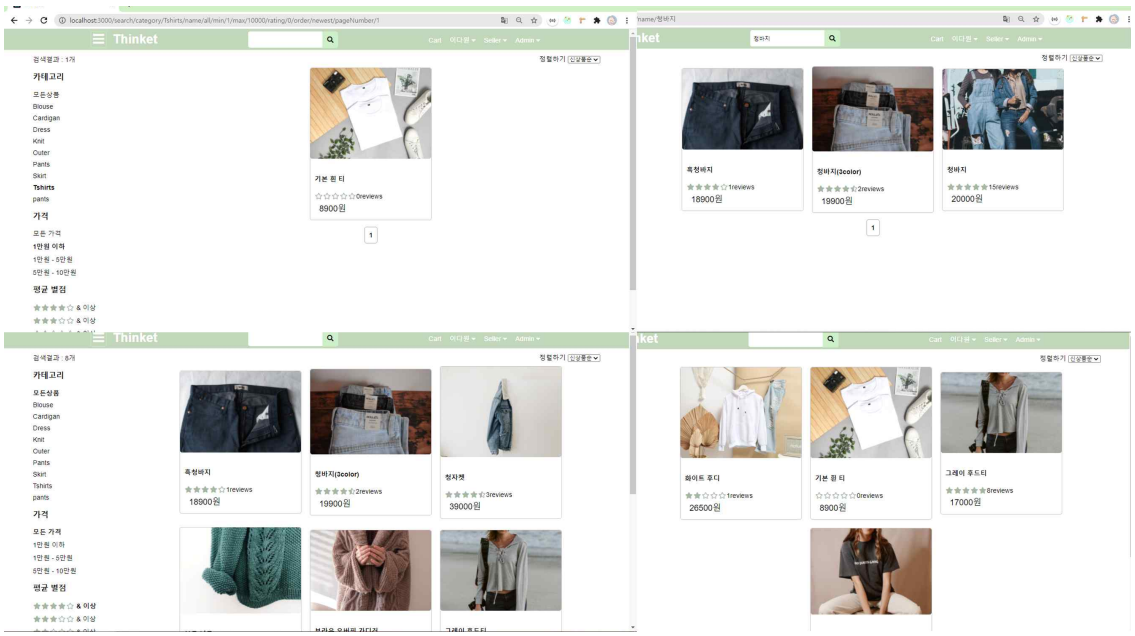
[관리자의 대시보드]

관리자의 대시보드 페이지이다.

총 사용자수와 주문 수, 총 주문 금액을 한눈에 파악할 수 있으며 일자별 판매현황을 확인하고 카테고리 별 상품 현황을 확인할 수 있다.

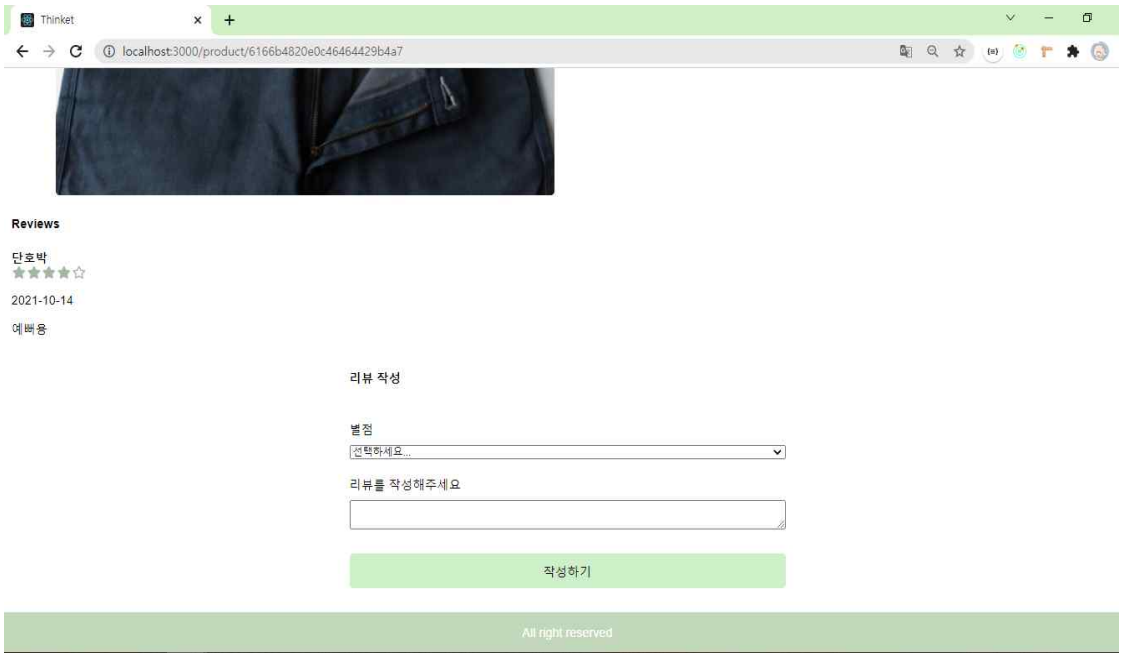


[상품 상세페이지]
좌측은 상품 수량이 있을 때, 우측은 판매 가능한 상품 수량이 없을 때의 상품 상세 페이지이다.
이전 페이지로 돌아갈 수 있는 버튼과 상품명, 평점 및 리뷰 개수, 상품 설명, 가격, 구매 가능 여부, 수량을 확인할 수 있다.
상품 수량이 있을 경우에는 수량을 선택하고 장바구니 담기 기능을 이용할 수 있으며 상품 수량이 없을 경우에는 구매 가능여부에 품절이라고 뜨며 구매할 수 없게 된다.



[정렬, 검색, 카테고리]
위 4개의 화면은 위에서부터 차례로 티셔츠 카테고리의 1만원 이하 상품 필터링, 청바지 검색 시, 별점 4점 이상의 상품, 티셔츠 카테고리 클릭시의 화면이다.

상품을 보기 편하게 정리하기 위해 카테고리 기능을 추가하였고 검색기능을 통해 원하는 상품을 빠르게 찾을 수 있게 하였다. 또한 평균 별점과 가격을 기준으로 필터링하여 원하는 상품을 찾을 수 있다. 위 화면에는 없지만 우측상단의 정렬기준을 통하여 상품을 신상품순, 고가순, 저가순 등으로 정렬하여 볼 수 있다.

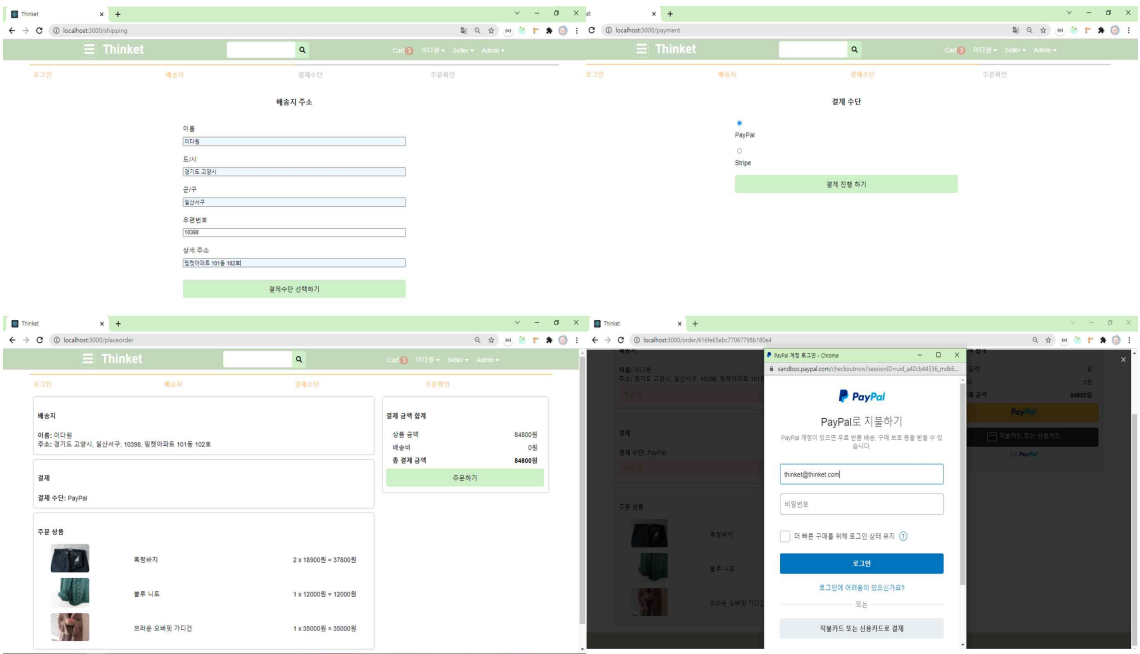


[리뷰 작성]
상품 상세페이지 하단에서 별점 및 리뷰를 남길 수 있다.
리뷰 작성시 이름과 별점, 작성 날짜, 리뷰 코멘트가 기록된다.

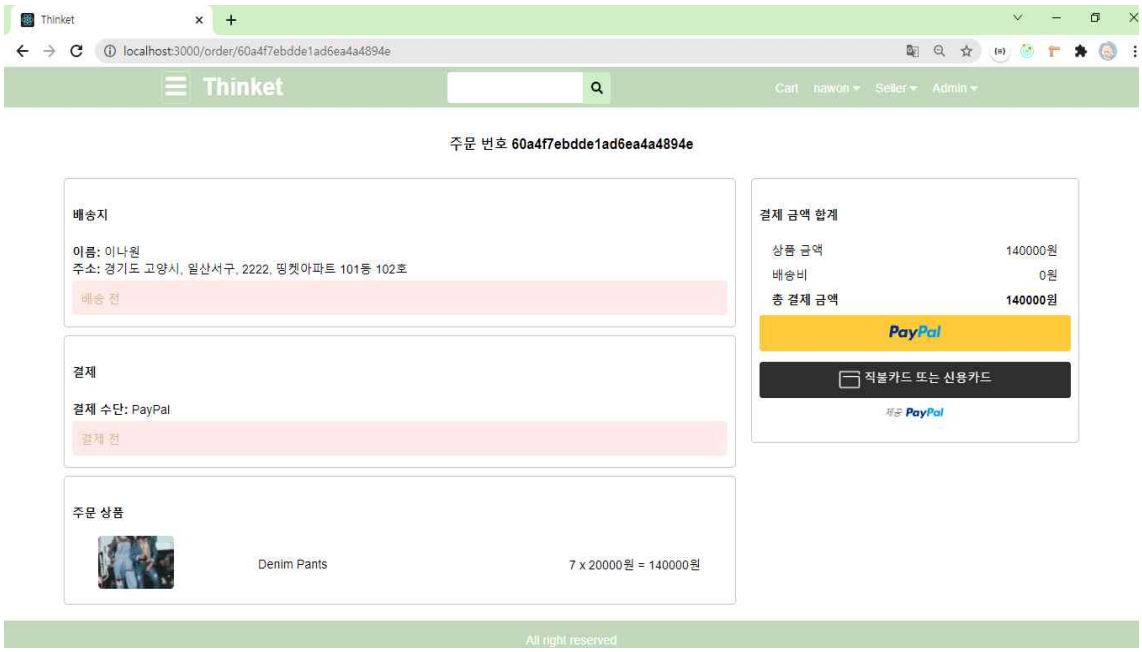
[장바구니 – 상품 없을 때]

장바구니에 담긴 상품이 없을 때 보이는 화면이다.
'장바구니가 비었습니다.' 라는 문구와 함께 쇼핑하러 가기 버튼 클릭시 메인페이지로 넘어가서 상품을 볼 수 있게끔 하였다.

[장바구니 – 상품 있을 때]
장바구니에 상품이 담겨있을 때 보이는 화면이다.
상품들의 이미지, 상품명, 수량, 가격 및 결제 예상금액(총 수량)이 보여지고 주문하기 버튼을 통해 주문과정으로 넘어갈 수 있다.



[주문과정]
위 화면은 주문하기 과정을 차례로 보여준다. 사진의 왼쪽부터 차례로 배송지 주소 입력, 결제수단 선택, 주문확인, 결제 과정이다. 주문확인에서는 입력한 배송 주소와 결제수단, 주문한 상품, 총 지불해야하는 가격을 확인할 수 있다.



[주문완료 후 페이지]
배송지와 결제수단, 주문 상품과 배송여부, 결제여부를 확인할 수 있다.

3.5 결과 분석

3.5.1 프로젝트 기능 분석

3.5.1.1 사용자 관점 분석

띵켓에서 회원가입 및 로그인을 통해 사용자로서 가입, 로그인 되어있을 경우 상품 상세페이지 조회, 상품 구매 및 주문 내역 조회, 장바구니 등의 이용이 가능하다. 또한 상품을 더 편리하게 보게 해주는 카테고리, 필터링 기능을 이용할 수 있고 상품 검색을 통해 원하는 상품의 이름을 알 경우 검색을 통해 조회가 가능하게끔 기능이 제공된다. 또한 별점과 리뷰를 남길 수 있다. 이미 리뷰를 남겼을 경우에는 리뷰 작성 기능이 제한된다.

3.5.1.2 판매자 관점 분석

판매자 계정으로 등록되어 있을 경우 일반 사용자 계정과 같이 장바구니, 상품을 구매하는 등의 기능을 이용할 수 있을 뿐만 아니라 판매자가 자신의 상점을 운영할 수 있게 된다. 내 프로필에 들어가서 회원정보 수정 시에 '셀러' 부분이 추가되어 판매자명, 판매자 로고 이미지, 판매자 설명을 따로 수정할 수 있다. 또한 상점 운영에 필요한 상품의 이미지, 가격, 설명 등을 등록하거나 수정할 수 있다. 자신의 상점에 들어온 주문을 따로 주문내역에서 조회 할 수 있다.

3.5.1.3 관리자 관점 분석

관리자 계정에서는 회원가입 되어 등록된 사용자 계정들을 조회할 수 있으며 각 계정별로 판매자 혹은 관리자의 권한을 부여할 수 있다. 권한을 부여 받은 계정은 판매자 혹은 관리자만이 이용할 수 있는 기능을 사용할 수 있게 된다.
또한 띵켓 내에서 일어난 모든 주문과 상품에 관한 내역을 조회할 수 있다. 셀러 같은 경우에는 각 셀러가 관리하는 상품의 주문 내역 건에 관해서만 조회 가능하지만 관리자는 모든 주문내역이 조회 가능하다. 상품도 마찬가지이다. 상품 목록은 셀러와 관리자만이 조회, 등록, 수정, 삭제가 가능하고 셀러는 셀러 본인이 등록한 상품만 조회 혹은 수정, 삭제 가능하다. 하지만 관리자는 띵켓 내에 등록된 모든 상품에 대한 조회, 등록, 수정, 삭제가 가능하다.
마지막으로 관리자 계정에서는 총 사용자수, 주문 수, 주문 금액 및 날짜별 판매현황을 보여주는 그래프, 카테고리 별 상품 개수 현황을 확인할 수 있는 대시보드라는 기능이 추가적으로 있다.

3.5.2 기존 사례와 비교

기존에 존재하는 쿠팡, 무신사등이 구매자(사용자)에게 제공하는 장바구니, 상품 주문 기능, 상품 검색 및 필터링 기능, 별점 및 리뷰 기능을 구현하였고 판매자로 등록하여 상품 등록, 판매 등이 가능하게 하였을 뿐만 아니라 관리자는 네이버 스마트스토어에서 판매자에게 제공되는 상품 등록 및 관리, 매출 통계(그래프)를 조회할 수 있게끔 구현함으로서 기본적인 이커머스 사이트, 쇼핑몰의 기능을 갖추었다.
그러나 회원 등급, 포인트 지급, 찜(좋아요) 등의 부가적인 기능은 제작하지 못하여서 조금 아쉬움이 남는다.

4. 결론 및 향후 과제

4.1 프로젝트의 결론 및 기대효과

우리가 실생활에서 너무나도 쉽게 접할 수 있고, 많이 이용하고 있는 e-commerce 웹사이트를 직접 구상하고 주요 기능들을 구현함으로써 기본적인 프론트엔드 개발 능력 향상 및 웹 사이트의 UI에 대한 이해를 할 수 있었고, 웹사이트의 구성과 필요한 기능을 구현하며 간접적인 실무 경험을 할 수 있었던 계기가 되었을 것으로 기대가 된다.

4.2 프로젝트의 향후 과제

웹 사이트를 사용자들이 이용함에 있어서 좀 더 편의성을 높여줄 수 있는 기능들은 생각하면 굉장히 많기 때문에 사용자의 시각에서 디자인이나 기능들을 발전 시켜 나아가야 한다고 생각한다.

추가적으로 웹사이트에서 나타날 수 있는 보안 상 취약점들이 많은데, 이를 방지할 수 있는 사이트로 개선하여야 된다고 생각해서, 현재 웹 사이트가 보안적인 문제에 대응할 수 있도록 기능들을 추가해야 할 필요가 있을 것으로 보인다.

보안 문제 해결을 위해 회원가입 시에 본인인증 기능을 추가하거나 개인정보 수정 시에 비밀번호 재 인증을 통하여 비밀번호 일치 시에만 수정할 수 있게 하는 기능, 결제과정에서의 보안기능, 관리자 계정 로그인시 보안코드 추가 등의 기능을 추가하여 외부 공격자로부터 받을 수 있는 보안 위협을 방지하는 것을 목표로 하여 더 더욱 안전한 쇼핑몰 이용이 가능하게끔 할 것이다.

그밖에도 사이트를 개선시킬 수 있는 것이 무엇일까 끊임없이 고민하는 것이 과제로 남을 것이라고 생각한다.

5. 참고자료

React https://ko.reactjs.org/

Redux https://ko.redux.js.org/

Node.js https://nodejs.org/ko/

Express https://expressjs.com/ko/

Mongo DB https://www.mongodb.com/

Mongoose https://mongoosejs.com/

6. 별첨

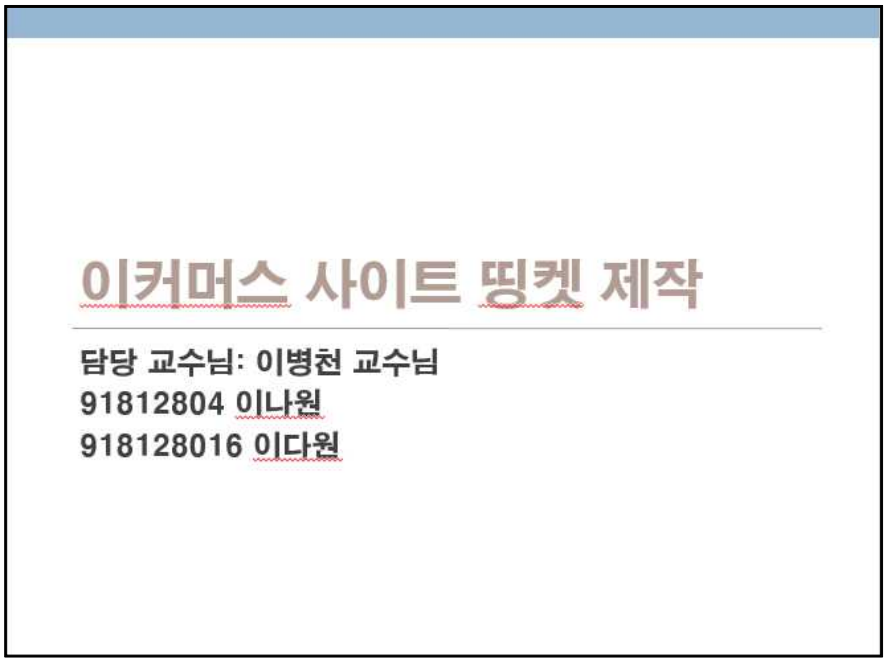
6.1 소스코드 (github 주소)

https://github.com/leenawon/e-commerce-capstone-project

6.2 서비스 주소

https://thinket-site.herokuapp.com/

6.3 발표 ppt



목차

- 주제 선정
- 구상도
- 추진 경과
- 개발 환경 및 개발 내용
- 개발 시스템 운영
- 결론 및 기대효과

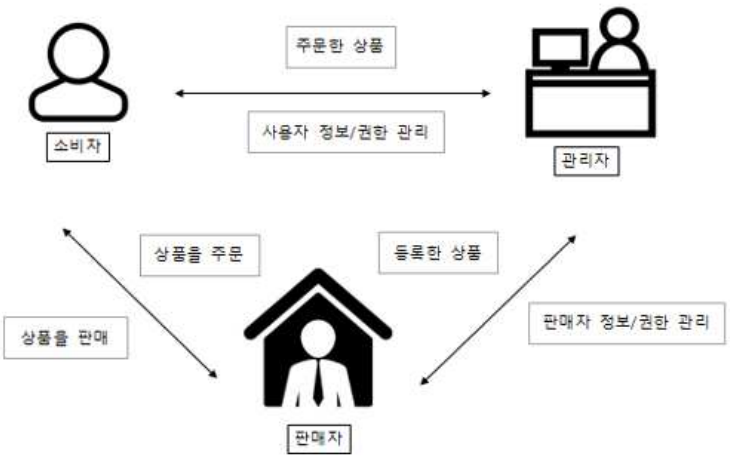
주제 선정

메인 페이지, 상품 페이지, 장바구니, 필터링, 로그인 등의 사이트 기능 구현

제작한 사이트를 실제 이용 가능하게끔 배포 기능을 구현

자주 접하게 되는 e-commerce 웹사이트들이 어떻게 작동을 하고, 어떻게 구성이 되어서 운영이 되는지를 탐구하고 직접 개발해보며 더 나은 웹사이트 제작을 해본다.

구상도



추진 경과

추진 경과 / 수행	3월	4월	5월	6월	7월	8월	9월	10월
자료 조사 및 연구								
프로트에드 개발								
DB 구축 및 연동								
테스트 및 보완								

개발 환경 및 개발 내용

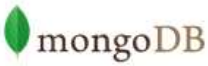
개발 환경

Frontend

Backend

OS

DB

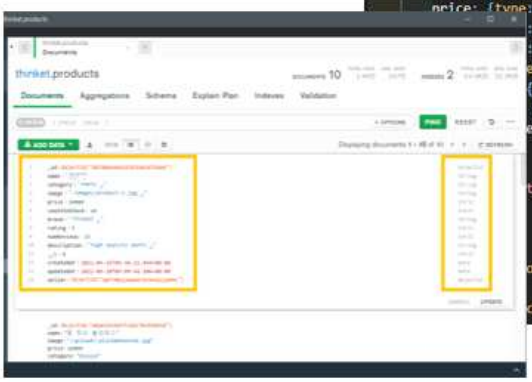


Deploy

개발 환경 및 개발 내용

개발 내용(1/6)

상품 DB

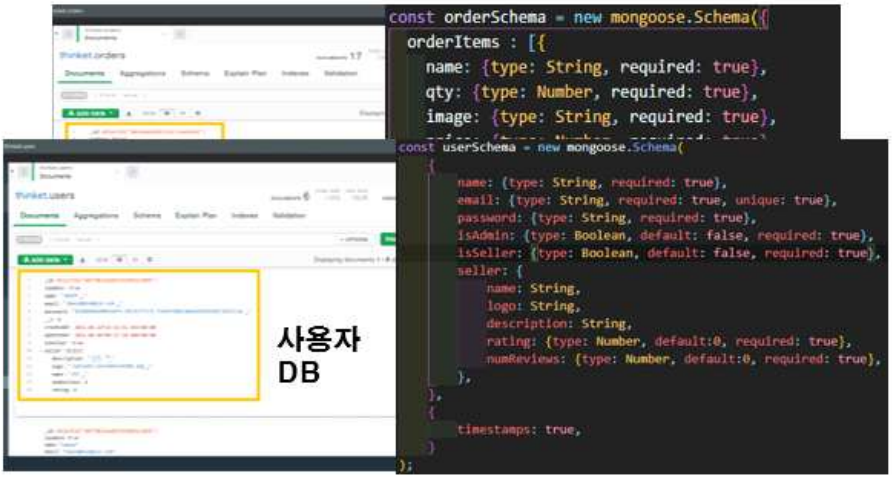



```
const productSchema = new mongoose.Schema(
    {
        name: {type: String, required: true, unique: true},
        seller: {type:mongoose.Schema.Types.ObjectID, ref: 'User'},
        category: {type: String, required: true},
        image: {type: String, required: true},
        price: {type: Number, required: true},
        : {type: Number, required: true},
        : String, required: true},
        : Number, required: true},
        {type: Number, required: true},
        {type: String, required: true},
    eviewSchema]

    true,

oose.model('Product', productSchema);

ct;
```

상품 DB 스키마 작성

개발 환경 및 개발 내용

개발 내용(2/6)




```
const orderSchema = new mongoose.Schema({
  orderItems : [{
    name: {type: String, required: true},
    qty: {type: Number, required: true},
    image: {type: String, required: true},
```

```
const userSchema = new mongoose.Schema(
    {
        name: {type: String, required: true},
        email: {type: String, required: true, unique: true},
        password: {type: String, required: true},
        isAdmin: {type: Boolean, default: false, required: true},
        isSeller: {type: Boolean, default: false, required: true},
        seller: {
            name: String,
            logo: String,
            description: String,
            rating: {type: Number, default:0, required: true},
            numReviews: {type: Number, default:0, required: true},
        },
    },
    {
        timestamps: true,
    }
);
```

개발 환경 및 개발 내용

개발 내용(3/6)

Router (product)

```
productRouter.get('/:id', expressAsyncHandler(async(req, res) => {
    const product = await Product.findById(req.params.id).populate('seller','seller.name seller.logo seller.rating
    seller.numReviews');
    if(product){
        res.send(product);
    }
    else{
        res.status(404).send({message:'Product Not Found'});
    }
}));

productRouter.post('/', isAuth, isSellerOrAdmin, expressAsyncHandler(async(req, res) => {
    const product = new Product({
        name: 'sample name' + Date.now(),
        seller: req.user._id,
        image: '/images/product-1.jpg',
        price: 0,
        category: 'sample category',
        brand: 'sample brand',
        countInStock: 0,
        rating : 0,
        numReviews: 0,
        description: 'sample description'
    });
    const createdProduct = await product.save();
    res.send({message: 'Product Created', product: createdProduct});
}));
```

특정 요청에 따른 기능을 지정

개발 환경 및 개발 내용

개발 내용(4/6)

메인 페이지 작성

```
return (
  <div>
    <h2>베스트 셀러</h2>
    {loadingSellers ? <LoadingBox></LoadingBox>
    :
    errorSellers ? <MessageBox variant="danger">{errorSellers}</MessageBox>
    :
    <>
      {sellers.length === 0 && <MessageBox>No Seller Found</MessageBox>}
      <Carousel showArrows autoPlay showThumbs={false}>
        {sellers.map((seller) => (
          <div key={seller._id}>
            <Link to={`/seller/${seller._id}`}>
              <img src={seller.seller.logo} alt={seller.seller.name}></img>
              <p className="legend">{seller.seller.name}</p>
            </Link>
          </div>
        ))}
      </Carousel>
    </>
    }
    <h2>전체 상품</h2>
    {loading? <LoadingBox></LoadingBox>
    :
    error?<MessageBox variant="danger">{error}</MessageBox>
    :
    <>
      {products.length === 0 && <MessageBox>No Product Found</MessageBox>}
      <div className="row-sorting center">
      {
        products.map((product) => (
        <Product key={product._id} product={product} />
```

베스트 셀러와 전체 상품
보여주는 메인 페이지 개발

개발 환경 및 개발 내용

개발 내용(5/6)

```
export const createProduct = () => async(dispatch, getState) => {
  dispatch({type: PRODUCT_CREATE_REQUEST});
  const {userSignin: {userInfo}} = getState();
  try {
    const {data} = await Axios.post('/api/products', {}, {
                                                          fo.token}`}
                                                          load: data.product});
                                                          sage ? error.response.data.message : error.message;
                                                          d: message});
```

```
export const productCreateReducer = (state = {}, action) => {
  switch (action.type) {
    case PRODUCT_CREATE_REQUEST:
      return {loading: true};
    case PRODUCT_CREATE_SUCCESS:
      return {loading: false, success: true, product: action.payload};
    case PRODUCT_CREATE_FAIL:
      return {loading: false, error: action.payload};
    case PRODUCT_CREATE_RESET:
      return {};
    default:
      return state;
  }
};
```

Action, Reducer

개발 환경 및 개발 내용

개발 내용(6/6)

Component 생성

```
export default function Product(props) {
  const {product} = props;
  return (
    <div key={product._id} className="product-card">
      <Link to={`/product/${product._id}`}>
        {/* Product Image */}
        <img className="product-image" src={product.image} alt={product.name}/>
      </Link>
      {/* Card Content */}
```

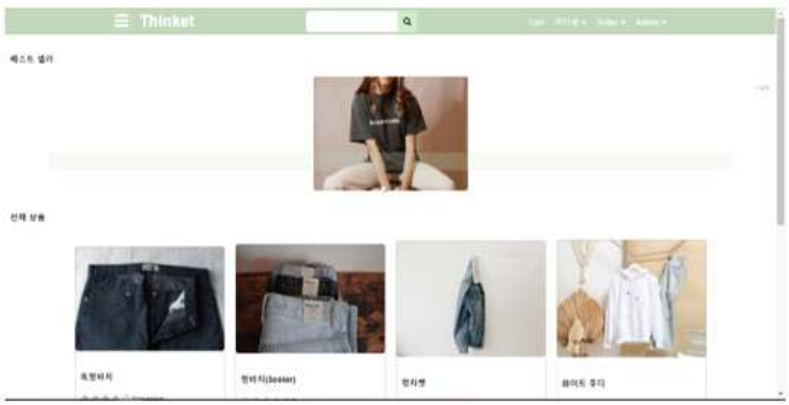
Product component

```
export default function SearchBox(props) {
  const [name, setName] = useState('');
  const submitHandler = (e) => {
    e.preventDefault();
    props.history.push(`/search/name/${name}`);
  }

  return (
    <div>
      {/* Search Form */}
      <form className="search" onSubmit={submitHandler}>
        <div className="row-sorting">
          <input type="text" onChange={(e) => setName(e.target.value)}></input>
          <button type="submit" className="primary">
            <i className="fa fa-search"></i>
          </button>
        </div>
      </form>
    </div>
  )
```

SearchBox component

개발 시스템 운영(1/8)

메인 페이지



개발 시스템 운영(2/8)

상품 검색 / 정렬 / 필터링

개발 시스템 운영(3/8)

상품 상세 페이지 / 장바구니 페이지

개발 시스템 운영(4/8)

주문 과정(1/2)

개발 시스템 운영(5/8)

주문 과정(2/2)

개발 시스템 운영(6/8)

주문 내역 확인

개발 시스템 운영(7/8)

판매자 - 상품 목록

개발 시스템 운영(8/8)

관리자 – 사용자 관리 / 대시보드

결론 및 기대효과

결론

- 이커머스 웹사이트가 가져야하는 기본적인 기능들을 바탕으로 팀만의 웹 사이트를 개발
- 차별화된 기능들을 개발하지는 못했지만 사이트 이용에 필수적인 기능들은 개발 성공

기대효과

- 팀 프로젝트의 목표를 달성하기 위해 함께 노력하며 협동심과 책임감의 필요성과 중요성을 느끼게 된 프로젝트
- 웹사이트를 직접 구상하고 주요 기능들을 구현함으로써 개발 능력 향상의 계기로 마련

Q&A

감사합니다.