

E조

MIDTERM PRESENTAION

Index

1 개요

2 역할 분담

3 UX/UI

4 Implementaion

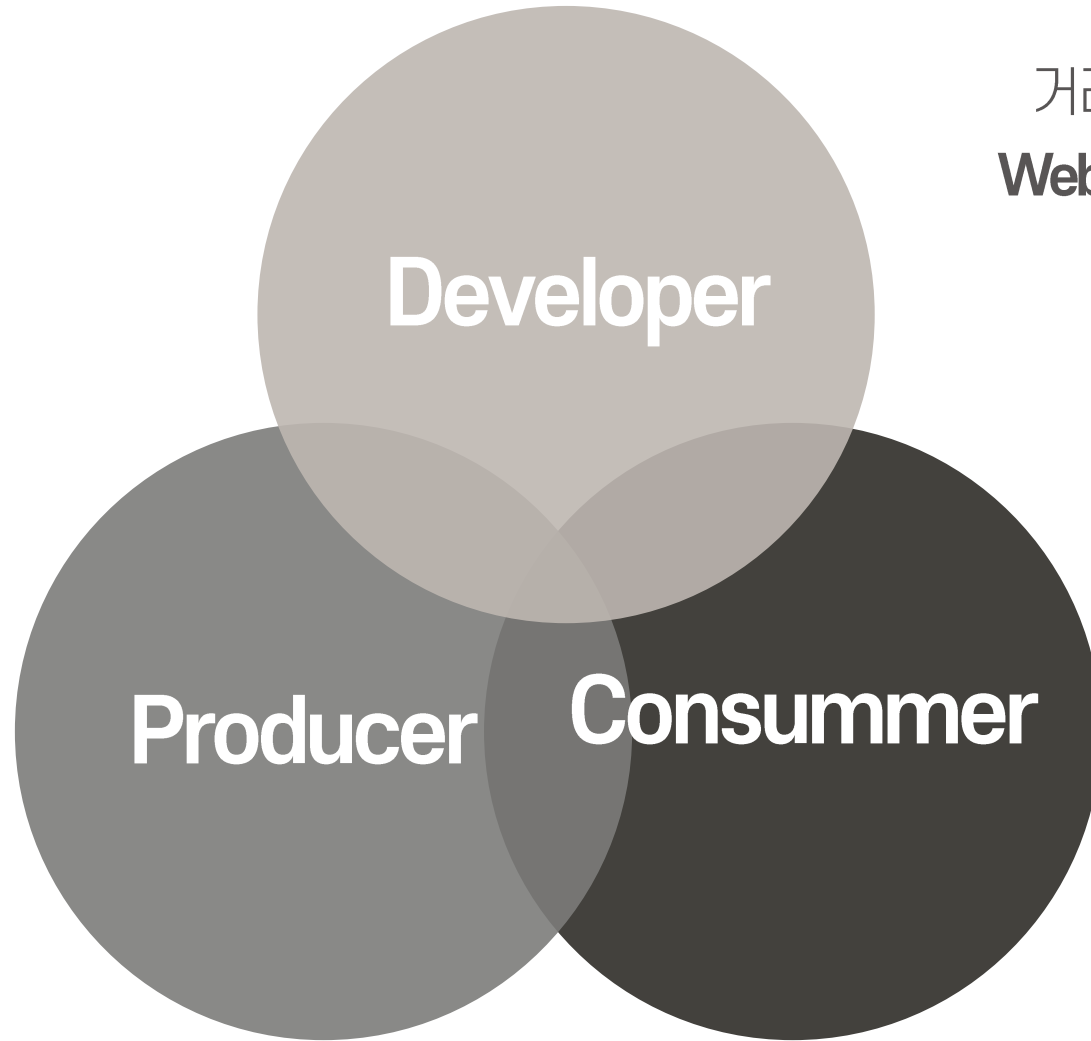
5 Challenges

6 Limitaion

7 Demo

직거래농산물 WebApp

거래를 위한
Web App 제공



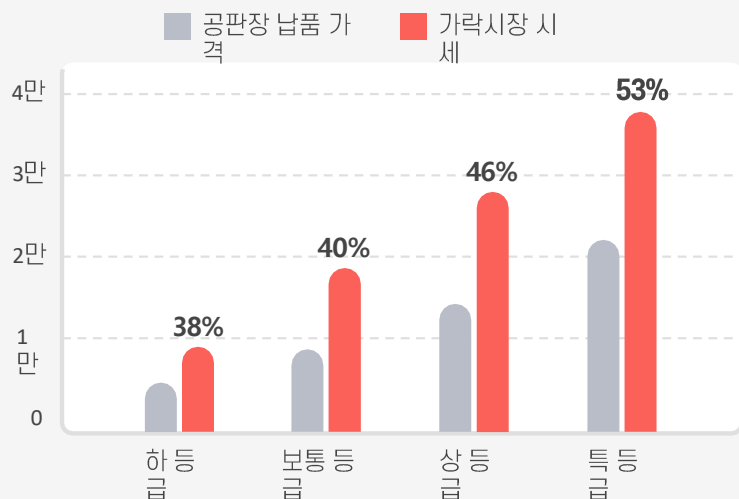
제 값 받고 과일을 판매
하고 싶은

농부를 위한
과일 직거래 플랫폼

신선한 과일을
소량이더라도
합리적인 가격으로 구매할

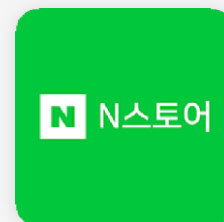
스마트 컨슈머를 위한
과일 직거래 플랫폼

직거래를 하고 싶은데
어디에 어떻게 올려야 할지 모르겠어요.



판매경로1) 공판장

비싸 봤자 소비자 시세의 50%이하에 납품
-> 비효율적인 판매 구조

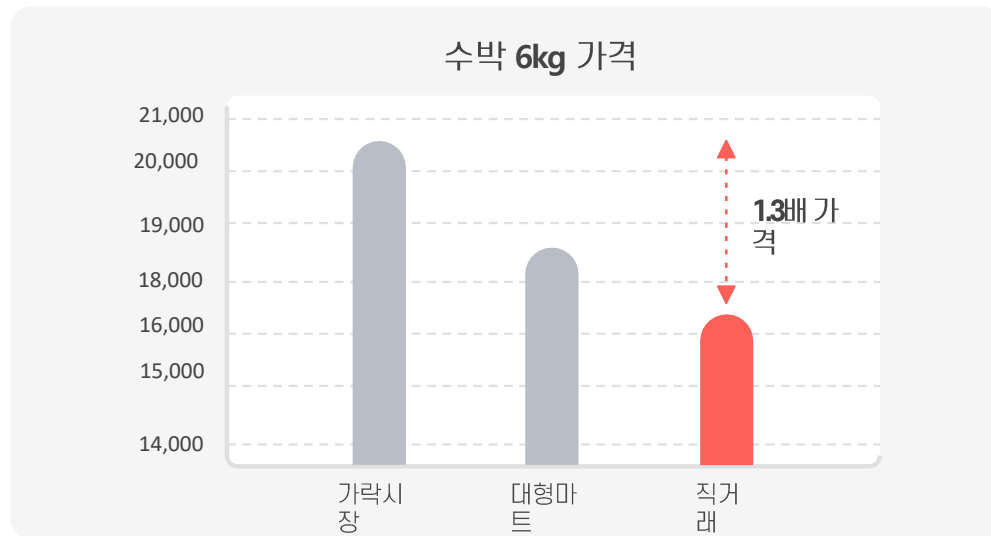


판매경로2) 오픈마켓 직거래

소비자 노출을 위해선 별도의 브랜딩 작업 필요 품
목 별 1,000개 이상의 제품 존재 -> 노출되기 까다로움

“마트/시장 과일 너무 비싸고 신선하지도 않아요.”

“한번 먹을만큼만 구매하고 싶어요.”



저렴함



선주문 후수확 - 신선함

MileStone



Part 1

Development Detail

1103 - 검색, 메인, 상품, 찜하기, 장바구니, 로그인, 회원가입

태그	기능	페이지	상세 기능	기능 코드	설명	착수일	완료일	수정일
1103	검색	검색 페이지	상품 검색	SE-01	주어진 조건에 따라 상품을 검색할 수 있음	2022. 10. 27	2022. 10. 31	
1103	검색	검색 페이지	검색 결과 정렬	SE-02	주어진 조건에 따라 검색된 상품을 정렬할 수 있음	2022. 10. 31	-	
1103	검색	검색 페이지	상점 검색	SE-03	상점 이름으로 상점을 검색할 수 있음	2022. 11. 1	2022. 11. 2	
1103	검색	검색 페이지	필터링 하여 검색	SE-04	검색을 할 때 추가적인 필터링 기능을 제공할 수 있음	2022. 10. 31	2022. 11. 2	
1103	검색	검색 페이지	퍼블리싱	SE-05	반응형으로 퍼블리싱 됨	2022. 10. 17	2022. 10. 27	
1103	검색	검색 상세 페이지 - 상품	퍼블리싱	SE-06	반응형으로 퍼블리싱 됨	2022. 10. 17	2022. 10. 27	
1103	검색	검색 페이지	필터링 아이콘 클릭 시 필터링	SE-07	상단 BEST, 할인 중과 같은 아이콘 클릭 시, 해당 필터로 검색된 결과를 받을 수 있음	2022. 11. 2	2022. 11. 3	
1103	유지보수	검색 페이지	필터를 전역으로 관리	SE-08	필터 조건을 전역으로 관리하여, 새로고침 해도 필터링 된 결과를 받을 수 있음	2022. 11. 2	2022. 11. 3	
1103	검색	검색 상세 페이지 - 상점	퍼블리싱	SE-07	반응형으로 퍼블리싱 됨	2022. 10. 23	2022. 10. 27	
1103	메인	메인 페이지	퍼블리싱	MN-01	반응형으로 퍼블리싱 됨	2022. 10. 17	2022. 10. 27	
1103	메인	오늘의 추천 상품 페이지	퍼블리싱	MN-02	반응형으로 퍼블리싱 됨	2022. 10. 23	2022. 10. 27	
1103	메인	라이브 페이지	퍼블리싱	MN-03	반응형으로 퍼블리싱 됨	2022. 10. 23	2022. 10. 27	
1103	상품 받아오기	상품상세 페이지	상품 정보 받아오기	PO-01	상품을 클릭했을 때, 해당 상품의 정보 (id, 가격, 할인을 등)을 받아서 렌더링 할 수 있음	2022. 10. 23	2022. 10. 23	
1103	상품 받아오기	상품상세 페이지	퍼블리싱	PO-02	반응형으로 퍼블리싱 됨	2022. 10. 17	2022. 10. 27	
1103	상품 받아오기	상품상세 페이지 - 구매 옵션	퍼블리싱	PO-03	반응형으로 퍼블리싱 됨	2022. 10. 17	2022. 10. 27	
1103	상품 받아오기	상품상세 페이지	리뷰 받아오기	PO-04	상품에 작성된 리뷰를 받아와 렌더링 할 수 있음	2022. 10. 23	2022. 10. 24	
1103	상품 받아오기	상품상세 페이지	상품문의 받아오기	PO-05	상품에 작성된 문의사항을 받아와 렌더링 할 수 있음	2022. 10. 23	2022. 10. 24	
1103	상품 받아오기	상품상세 페이지	무게, 개수, 포장 선택 후 구매	PO-06	상품 구매 시, 무게 개수 포장 선택 할 수 있음	2022. 11. 2	2022. 11. 3	
1103	장바구니 담기	장바구니 페이지	장바구니 받아오기	SC-01	장바구니에 담긴 상품을 받아오고 렌더링 할 수 있음	2022. 11. 2	2022. 11. 2	
1103	장바구니 담기	장바구니 페이지	장바구니 담기/삭제	SC-02	장바구니에 상품을 담고 삭제할 수 있음	2022. 11. 2	2022. 11. 2	
1103	장바구니 담기	장바구니 페이지	퍼블리싱	SC-03	반응형으로 퍼블리싱 됨	2022. 11. 1	2022. 11. 2	
1103	찜하기	공통	상품 찜 하기	LK-01	관심 있는 상품에 찜하기 및 취소 할 수 있음	2022. 10. 25	2022. 10. 29	
1103	찜하기	찜하기 페이지	찜한 정보 받아오기	LK-02	유저가 찜한 상품을 받아와 렌더링 할 수 있음	2022. 10. 25	2022. 10. 29	
1103	찜하기	찜하기 페이지	퍼블리싱	LK-03	반응형으로 퍼블리싱 됨	2022. 10. 23	2022. 10. 27	
1103	로그인	로그인 페이지	Oauth 2.0 카카오 로그인	LG-01	SNS 로그인을 통해 유저의 정보를 받아올 수 있음	2022. 10. 21	2022. 10. 25	
1103	로그인	로그인 페이지	익명 로그인	LG-02	회원 가입을 원치 않는 유저는 익명 로그인을 통해 서비스를 이용할 수 있음	2022. 10. 21	2022. 10. 25	
1103	회원가입	랜딩 페이지	퍼블리싱	SU-01	반응형으로 퍼블리싱 됨	2022. 10. 23	2022. 10. 27	
1103	회원가입	약관 동의 페이지	퍼블리싱	SU-02	반응형으로 퍼블리싱 됨	2022. 10. 23	2022. 10. 27	
1103	회원가입	약관 동의 페이지	약관 동의	SU-03	유저는 약관에 동의한 후 페이지 넘길 수 있음	2022. 10. 25	2022. 10. 30	
1103	회원가입	회원가입 페이지	퍼블리싱	SU-04	반응형으로 퍼블리싱 됨	2022. 10. 23	2022. 10. 27	
1103	회원가입	회원가입 페이지	닉네임, 이메일, 번호 설정	SU-05	회원가입 시 닉네임, 이메일, 번호를 입력받아 서버에 저장되어야함	2022. 10. 30	2022. 10. 30	
1103	회원가입	회원가입 페이지	휴대폰 번호 인증	SU-06	인증 코드 전송을 통해 해당 번호가 본인 번호가 맞는지 확인하는 절차가 있어야함	-	-	

Part 1

Development Detail

1117 - 상품 주문, 스토어 받아오기, 리뷰 작성/읽기, 내 정보 수정

태그 ▼	기능 ≡	페이지 ≡	상세 기능 ≡	기능 코드 ≡	설명 ≡	작수일 ≡	완료일 ≡	수정일 ≡
1117	유지보수	검색 페이지	필터 쿼리로 전송	SE-05	필터 검색 시, 필터 내용을 쿼리로 만들어 GET 요청할 수 있음 (유지보수)			
1117	내 정보 수정	마이페이지	정보 수정	MS-01	프로필, 닉네임, 이메일을 수정할 수 있음	2022. 10. 25		
1117	내 정보 수정	마이페이지	알림 설정	MS-02	알림을 켜고 끌 수 있음			
1117	내 정보 수정	마이페이지	퍼블리싱	MS-03	반응형으로 퍼블리싱 됨			
1117	내 정보 수정	마이페이지 - 환경설정	퍼블리싱	MS-04	반응형으로 퍼블리싱 됨	2022. 10. 17	2022. 10. 27	
1117	리뷰 작성/읽기	마이페이지	리뷰 작성	RV-01	구매한 상품에 관하여 리뷰를 작성할 수 있음			
1117	리뷰 작성/읽기	마이페이지	작성한 리뷰 받아오기	RV-02	유저가 작성한 리뷰 목록을 불러와 렌더링 할 수 있음			
1117	리뷰 작성/읽기	마이페이지	리뷰 작성 가능한 상품 받아오기	RV-03	구매는 했지만 아직 리뷰 작성을 안한 상품을 받아와 렌더링 할 수 있음			
1117	리뷰 작성/읽기	마이페이지 - 리뷰작성목록	퍼블리싱	RV-04	반응형으로 퍼블리싱 됨			
1117	리뷰 작성/읽기	마이페이지 - 리뷰작성	퍼블리싱	RV-05	반응형으로 퍼블리싱 됨			
1117	상품 주문	주문/결제 페이지	주문 전) 주문 정보 받아오기	OD-01	어떤 주문을 했는지 받아오고 렌더링 할 수 있음	2022. 11. 1	2022. 11. 2	
1117	상품 주문	주문/결제 페이지	배송 정보 입력	OD-02	배송 받을 주소를 입력할 수 있음			
1117	상품 주문	주문/결제 페이지	주문 정보 전송하기	OD-03	주문한 정보를 서버에 전송함			
1117	상품 주문	주문/결제 페이지	퍼블리싱	OD-05	반응형으로 퍼블리싱 됨	2022. 10. 17	2022. 11. 1	
1117	상품 주문	주문/결제 페이지 - 완료	퍼블리싱	OD-06	반응형으로 퍼블리싱 됨	2022. 10. 17	2022. 11. 1	
1117	상품 주문	마이페이지	주문 후) 주문 내역 받아오기	OD-07	주문을 했던 상품들을 불러오고 렌더링 할 수 있음			
1117	상품 주문	마이페이지	주문하고 상품 불러오기	OD-08	주문을 했던 상품들을 불러오고 렌더링 할 수 있음			
1117	상품 주문	주문/결제 페이지	옵션 정보 불러오기	OD-09	주문할 상품에 대하여 옵션 정보를 받아올 수 있음	2022-11-02	2022-11-03	
1117	스토어 받아오기	스토어페이지	스토어 정보 받아오기	ST-01	스토어 정보를 받아와 렌더링 할 수 있음			
1117	스토어 받아오기	스토어페이지	스토어 팔로우	ST-02	관심 있는 스토어를 팔로우 혹은 취소할 수 있음			
1117	스토어 받아오기	스토어페이지	퍼블리싱	ST-03	반응형으로 퍼블리싱 됨	2022. 10. 17	2022. 10. 27	

Development Detail

1201 - 실시간 채팅, 결제 모듈 적용

태그 ▼	기능 ≡	페이지 ≡	상세 기능 ≡	기능 코드 ≡	설명 ≡	착수일 ≡	완료일 ≡	수정일 ≡
1201	결제 모듈 적용	주문/결제 페이지	결제	OD-04	주문 정보 검증 및 결제 구현			
1201	실시간 채팅	채팅 상세페이지	채팅 내용 받아오기	CH-01	채팅함의 채팅 내역을 시간 순서대로 불러와 렌더링 할 수 있음			
1201	실시간 채팅	채팅 상세페이지	실시간 채팅 구현	CH-02	Realtime database를 통한 실시간 채팅을 할 수 있음			
1201	실시간 채팅	채팅 상세페이지	채팅 상대 차단 및 나가기	CH-03	채팅 상대를 선택하여 차단 및 나가기를 행할 수 있으며, 차단의 경우 사후 1:1 채팅이 불가함			
1201	실시간 채팅	채팅 상세페이지	실시간 채팅 기능 구현	CH-04	realtime database를 활용하여 실시간 채팅이 가능함			
1201	실시간 채팅	채팅 상세페이지	사진, 파일 전송 기능	CH-05	사진, 동영상, 파일, 음성메시지 전송 기능이 가능함			
1201	실시간 채팅	채팅 상세페이지	채팅 설정	CH-06	채팅 알림 끄기, 상점 차단, 신고, 나가기 기능이 가능함			
1201	실시간 채팅	채팅페이지	채팅 함 불러오기	CH-07	채팅 상대별 최신 채팅 내역 하나만 불러와서 렌더링 할 수 있음			
1201	실시간 채팅	채팅페이지	퍼블리싱	CH-08	반응형으로 퍼블리싱 됨	2022. 10. 24	2022. 10. 24	
1201	실시간 채팅	채팅 상세페이지	퍼블리싱	CH-09	반응형으로 퍼블리싱 됨			

39 / 63

구현 완료 기능 수 / 전체 기능 수

Development Detail

Front End

React

TypeScript

Redux/Redux-
Saga

Back End

Firebase Hosting

Firebase Storage

Firestore

Firebase Cloud
Function

Design

Adobe XD

- Design
- Firebase

장병우

- React State Management

김민은

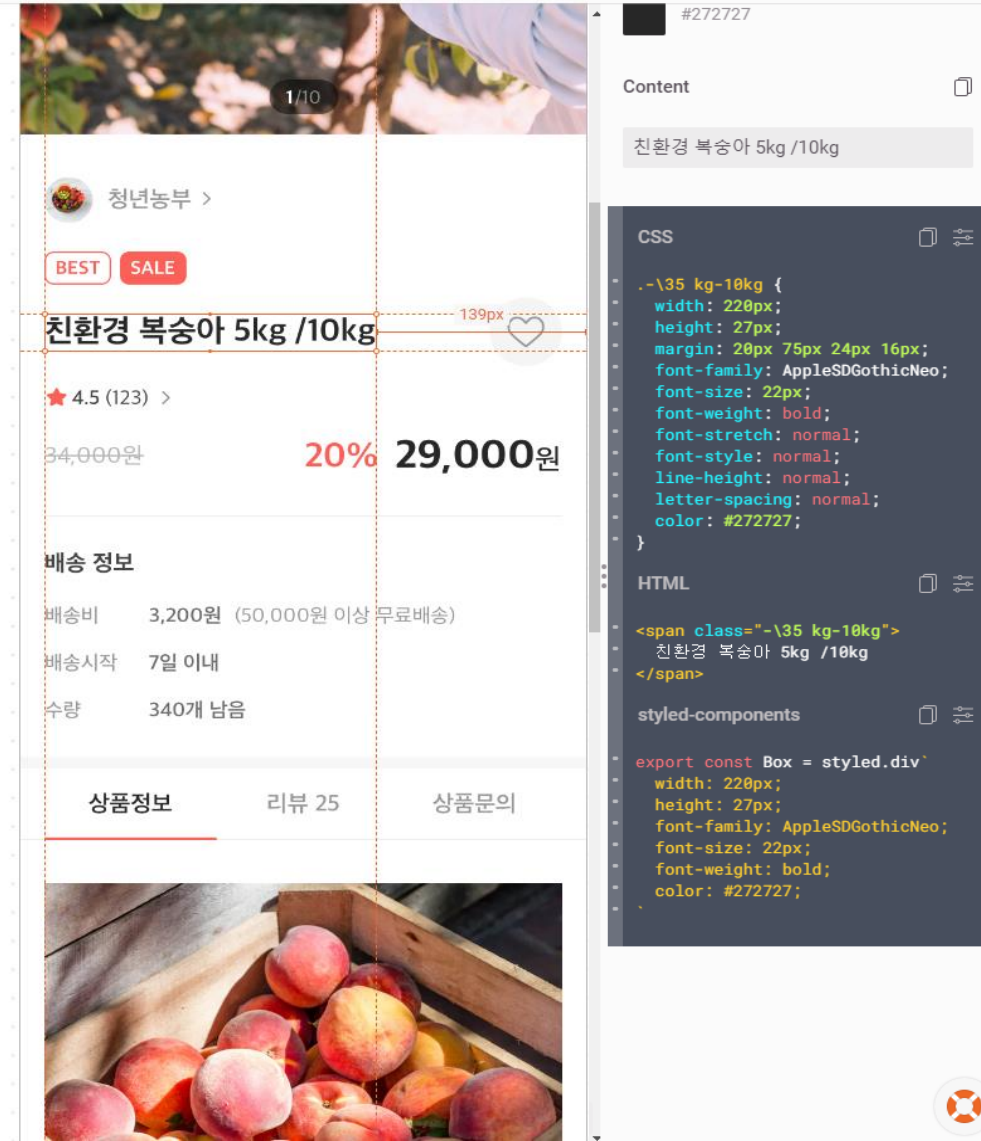
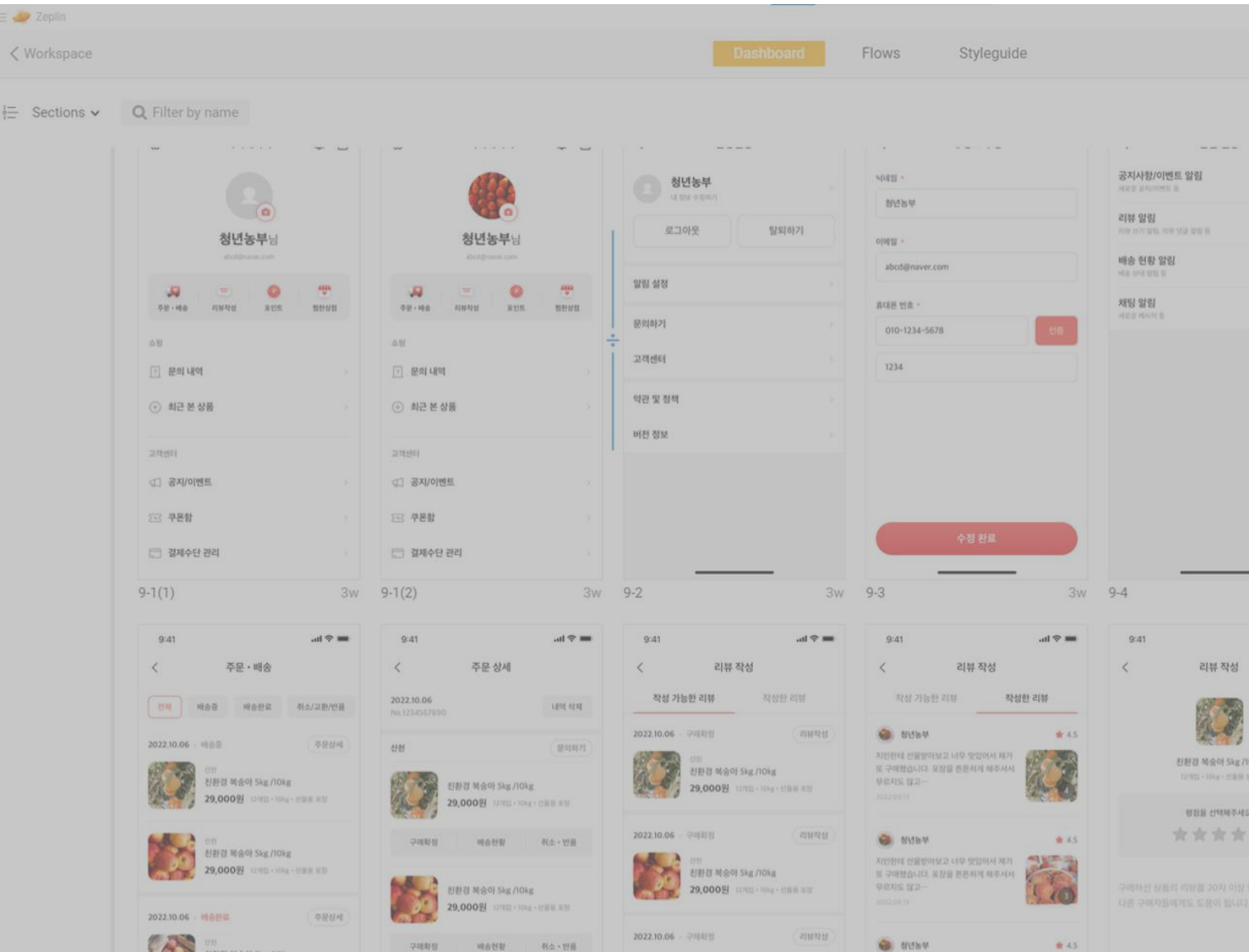
- React Component

심상원

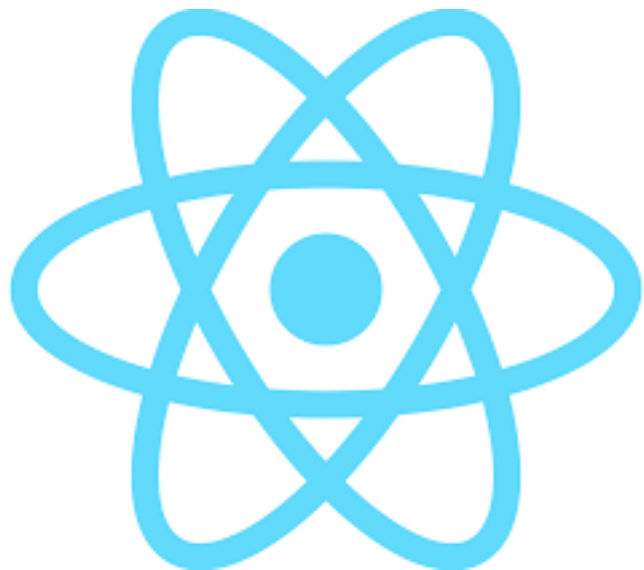
엄태경

- Design with Adobe XD

Part 3 Zeplin



Implementation



복잡하고 관리하기 힘든 양방향 데이터흐름을
단방향흐름으로 바꾸기 위해 Redux를 사용하고,

비동기적인 dispatch(reducer함수 동작) 구현을 위해 Redux-Saga를 사용하였습니다.

Part 3

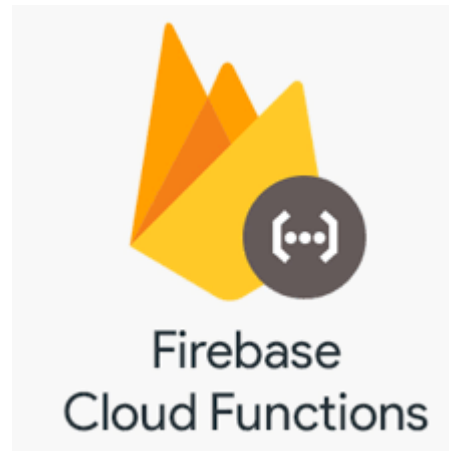
Implementation

```
function ProductPage(props: any) {  
  const params = useParams();  
  const location = useLocation();  
  const navigate = useNavigate();  
  const dispatch = useDispatch();  
  
  const selector: ProductDataType = useSelector((state: RootState) =>  
    state.ProductInfoReducer!.productInfo  
  );  
  
  const modalselector: any = useSelector((state: RootState) =>  
    state.PurchaseReducer.purchaseInfo  
  );  
  
  const [isOpen, setOpen] = useState(false);  
  
  useEffect(() => {  
    dispatch(closeModalAction(modalselector));  
    dispatch(GetProductInfo(params.productId));  
  }, [])
```

```
export const GET_PRODUCT = "GET_PRODUCT";  
export const GET_PRODUCT_SUCCESS = "GET_PRODUCT_SUCCESS";  
export const GET_PRODUCT_FAIL = "GET_PRODUCT_FAIL";  
export const GET_PRODUCT_LOADING = "GET_PRODUCT_LOADING";  
  
export const GetProductInfo = (data: any) => {  
  console.log("call action" + data);  
  
  return {  
    type: GET_PRODUCT,  
    payload: data  
  }  
}
```

```
export function ProductInfoReducer(state = productInfoInitState, action: any) {  
  switch (action.type) {  
    case GET_PRODUCT_SUCCESS:  
      console.log("reducer");  
      console.log(action.payload.photoDataList);  
  
      return {  
        ...state,  
        productInfo: action.payload,  
      };  
  
    case GET_PRODUCT_FAIL:  
      return {  
        ...state,  
        productInfo: action.payload,  
      };  
  
    case GET_PRODUCT_LOADING:  
      return {  
        ...state,  
        productInfo: action.payload,  
      };  
  
    default:  
      return state;  
  }  
}
```

Firebase

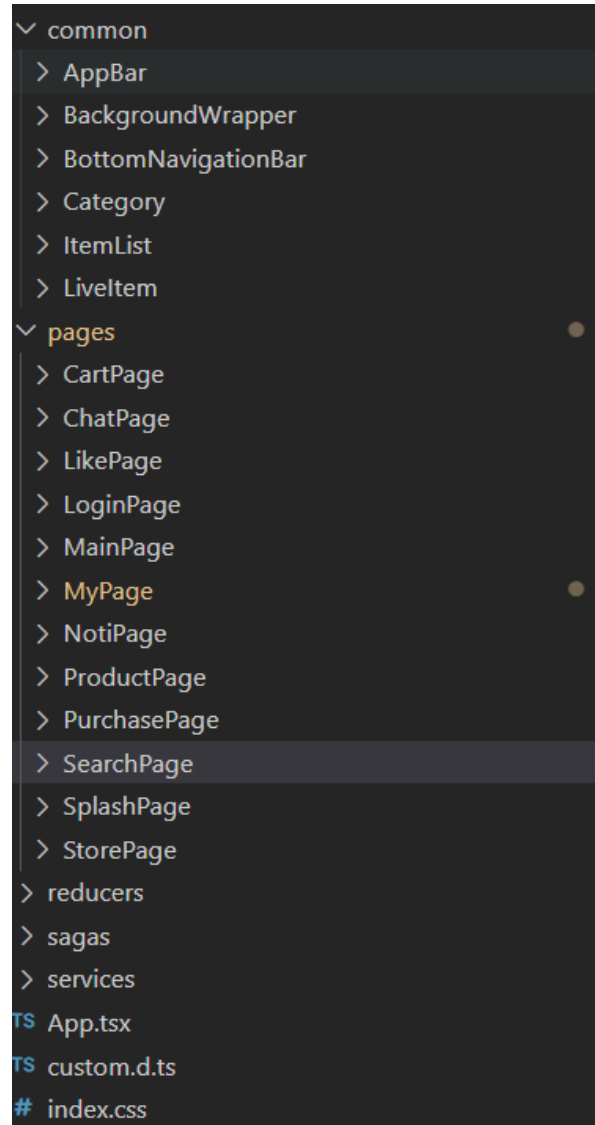


Firestore – Storage – Cloud Function – Authentication - Hosting

BackEnd 대신에 Firebase를 서버로 사용중이며 위의 기능들을 이용중입니다.

Part 4

Frontend Progress



Pages		Components
Login Page	Store Page	AppBar Component
Agree Page	Search Page	CheckBox Component
Like Page	Chat Page	Background Wrapper
Cart Page	Notice Page	Toggle Slider Component
Purchase Page	Review Page	
Splash Page	Event Page	
Product Page		

Frontend Progress



카카오로 계속하기

네이버로 계속하기

Apple로 계속하기

로그인 전 둘러보기

kakao

카카오메일 아이디, 이메일, 전화번호

비밀번호

☒ 로그인 상태 유지 ⓘ

로그인

또는

QR코드 로그인

회원가입

계정 찾기 | 비밀번호 찾기

[이용약관](#) [개인정보 처리방침](#) [운영정책](#) [고객센터](#) [공지사항](#) [한국어](#) ^

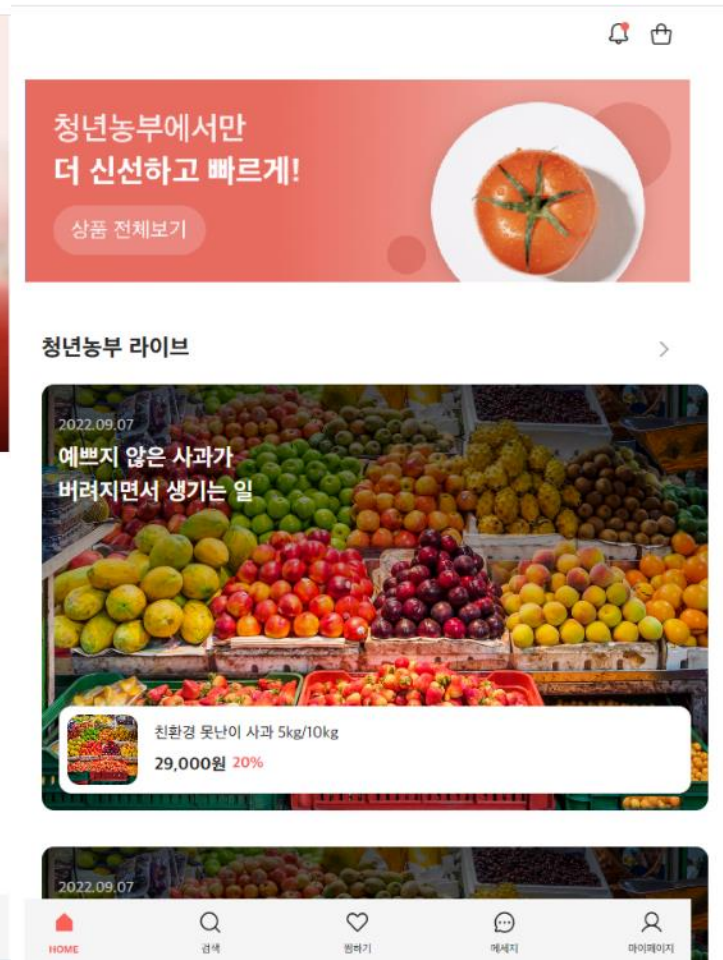
Copyright © Kakao Corp. All rights reserved.

Login Page

- 로그인 페이지 구현
- 카카오 로그인 API 연결
- 로딩 페이지 구현

Part 4

Frontend Progress

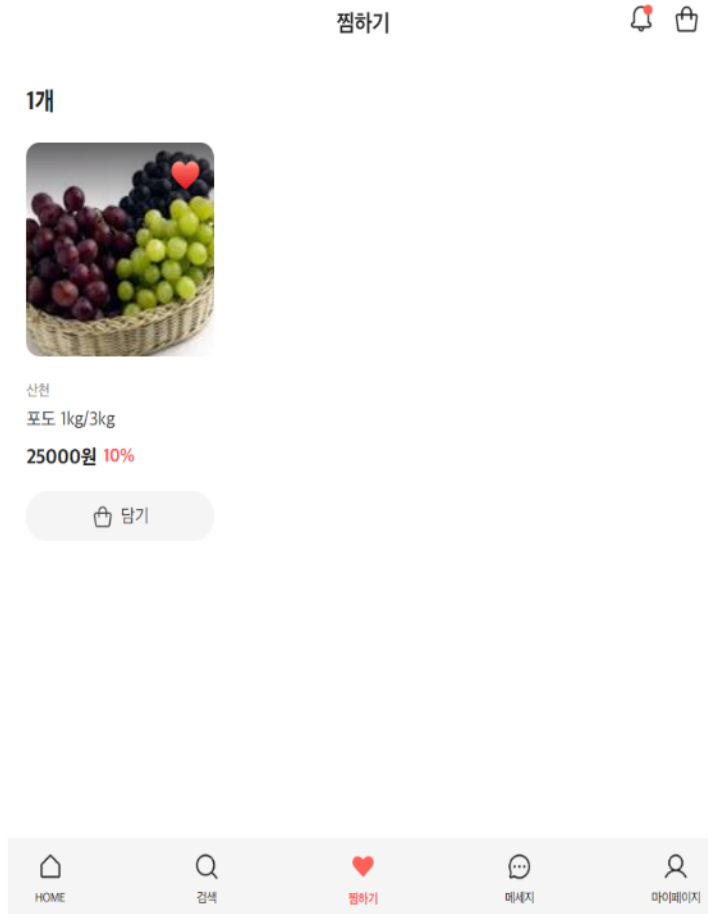
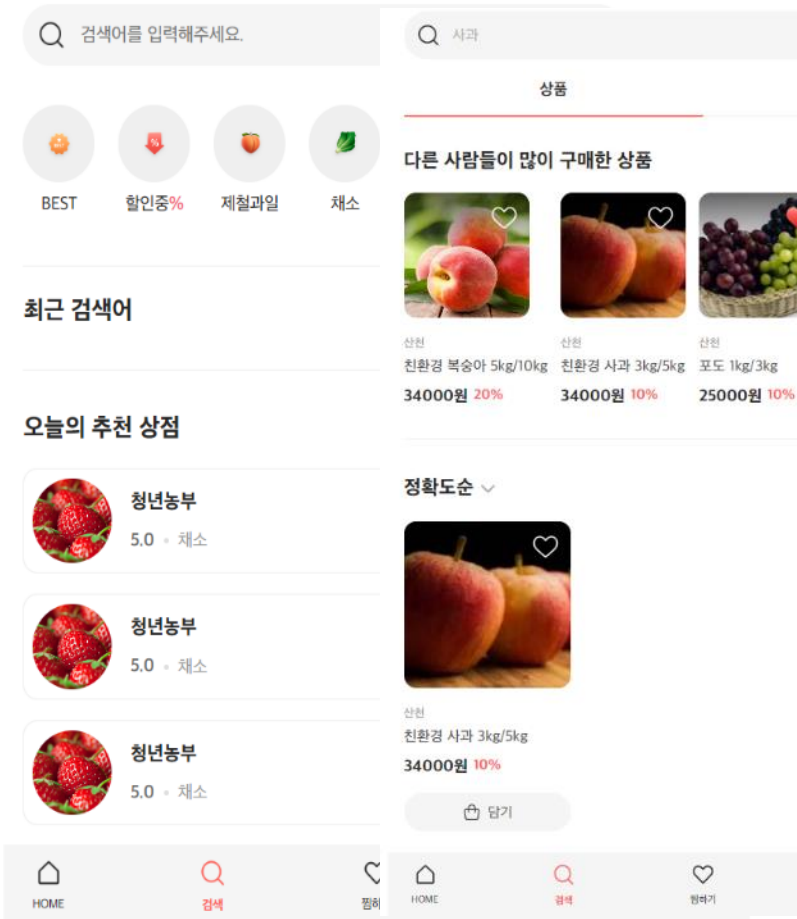


Main Page

- 추천상품 Component 구현 및 적용
- Navigator button 구현 및 적용
- discount Component구현 및 적용

Part 4

Frontend Progress

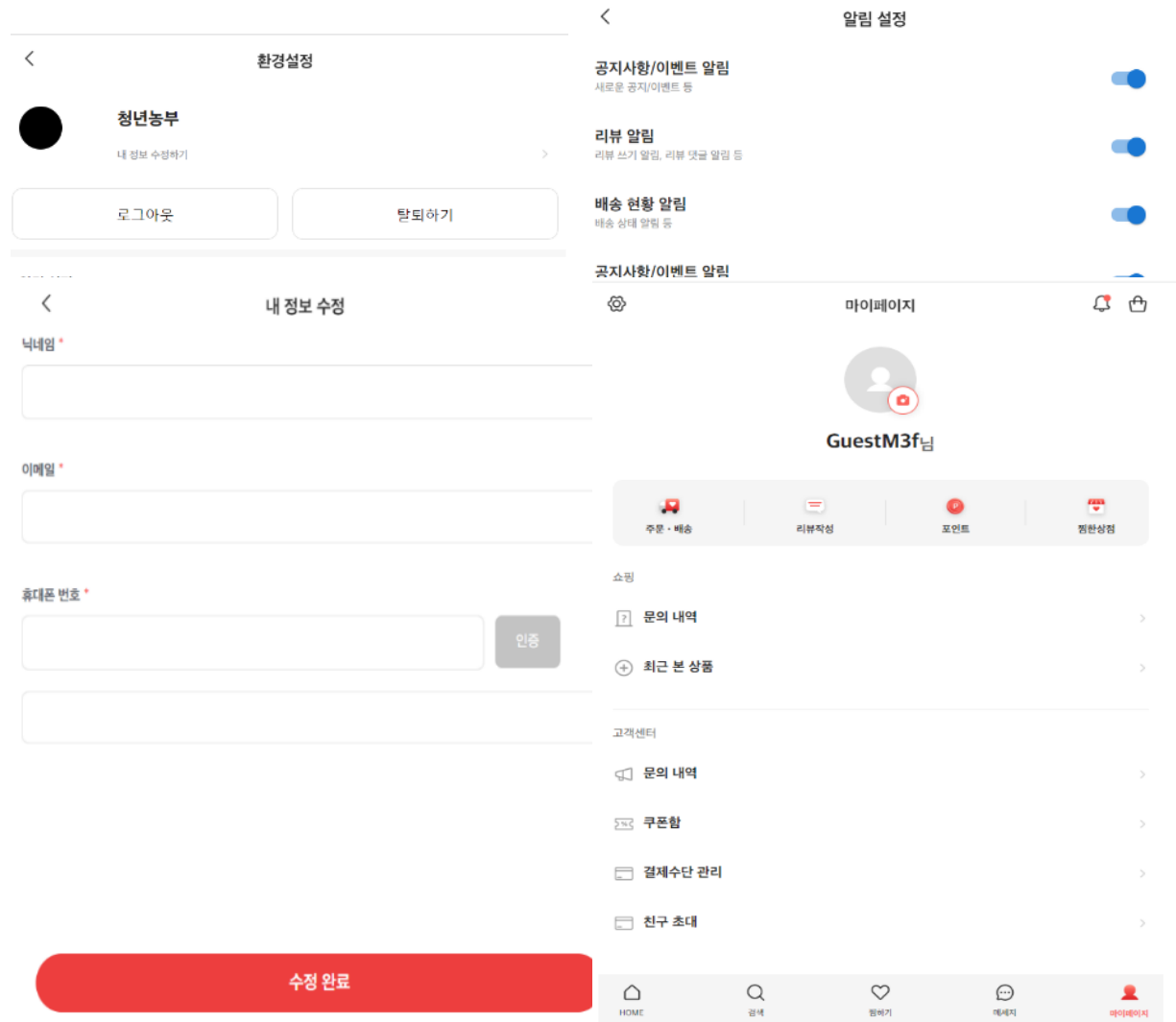


Search Page

- 검색 페이지 구현
- 검색 기능 구현
- 최근 검색어 기능 구현
- 찜하기 기능 구현

Part 4

Frontend Progress

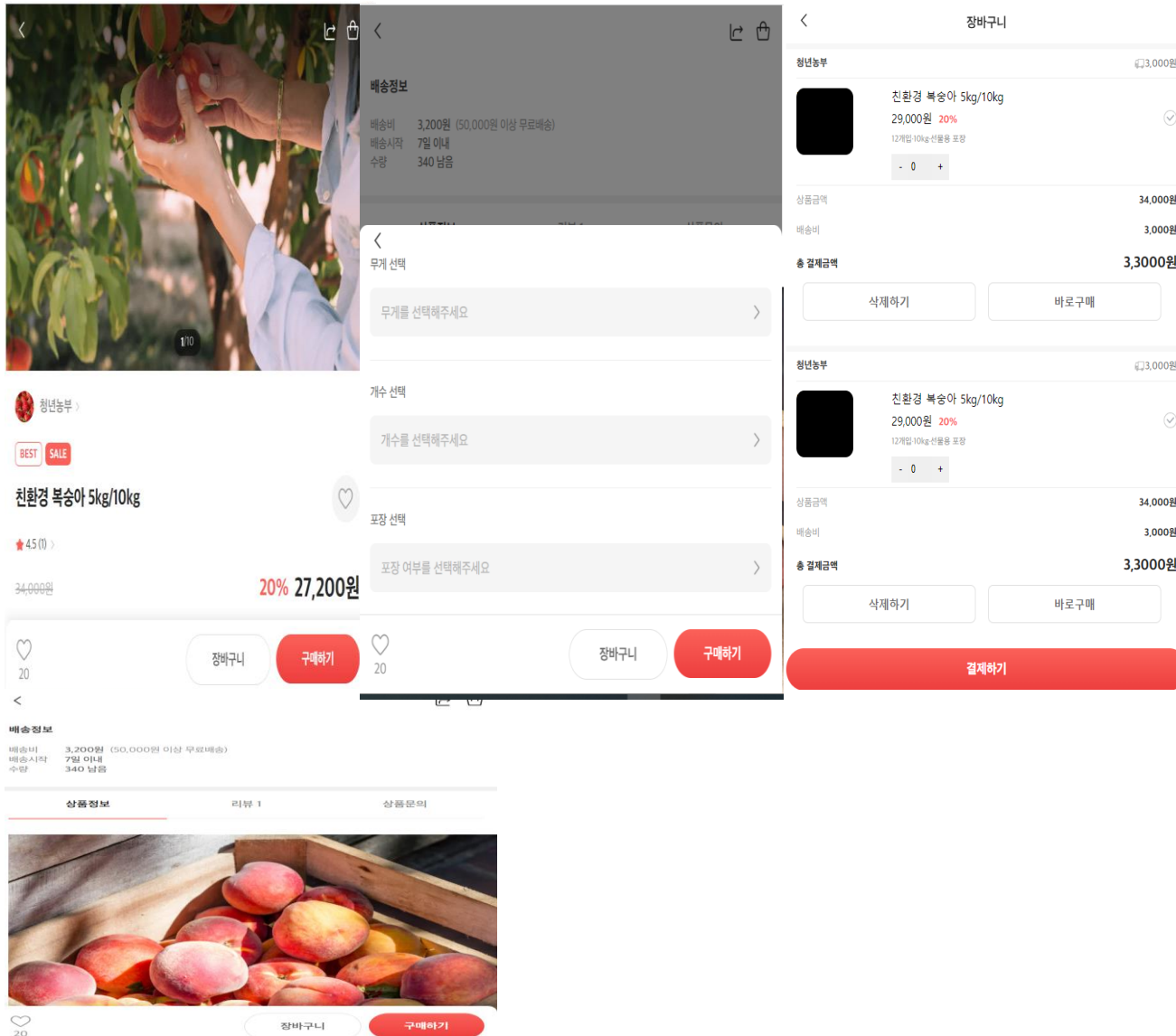


My Page

- 마이페이지 구현 및 navigator 설정
- 알림설정 페이지구현
- 정보수정 페이지구현
- 환경설정 페이지구현

Part 4

Frontend Progress



Store Page

-상품정보 페이지구현

-구매옵션 페이지구현

-장바구니 페이지구현

Firestore : firestore

Firestore는 확장 가능한 완전 관리형 **서버리스** 문서 데이터베이스를 사용해 애플리케이션 개발을 쉽게 할 수 있게 해준다.

서버리스(serverless)란 개발자가 서버를 관리할 필요 없이 애플리케이션을 빌드하고 실행할 수 있도록 하는 클라우드 네이티브 개발 모델입니다.

서버리스 모델에도 서버가 존재하긴 하지만, 애플리케이션 개발에서와 달리 추상화되어 있습니다. 클라우드 제공업체가 서버 인프라에 대한 프로비저닝, 유지 관리, 스케일링 등의 일상적인 작업을 처리하며, 개발자는 배포를 위해 코드를 컨테이너에 패키징하기만 하면 됩니다.

Firestore : firestore

User 구조

The screenshot displays the Firebase Firestore console interface. The left sidebar shows the 'user' collection selected. The main area shows a list of documents under the 'user' collection. One document is selected, showing its fields: 'profile_email', 'profile_img', and 'profile_nickname'. The 'profile' field is expanded, showing its sub-fields: 'is_guest' (true) and 'uid' ('YAoxGuaLSleth0pijmbWSh7GreN2').

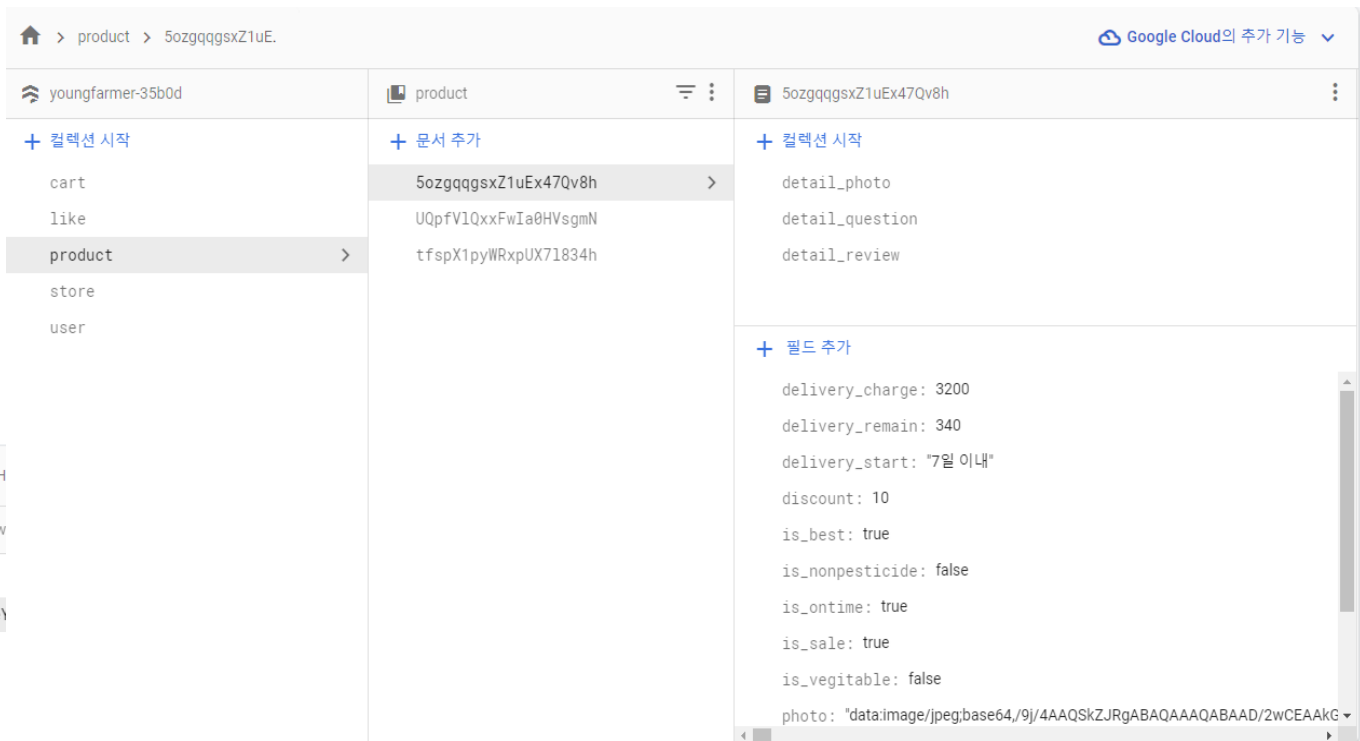
Guest login을 지원하기 때문에
유저의 속성에는 is_guest, uid가 있고

안에 profile이라는 하위 collection을
따로 만들어 email, nickname, img를
관리하고 있다.

Part 4

Firestore : firestore

Product 구조

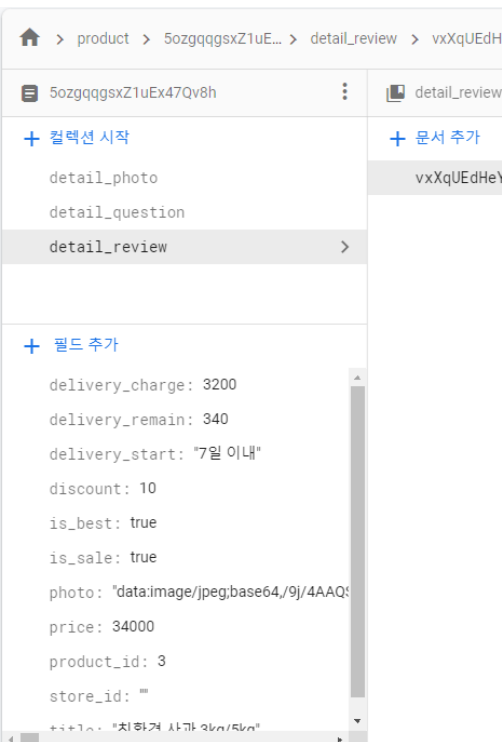


product에는 가격, 상품명 등 판매 과일에 대한 정보가 담겨 있다.

detail_photo, detail_question, detail_review 컬렉션을 따로 만들어 관리하는





하위 컬렉션 구조를 선택하였는데, 전역으로 관리해야 되는 quesiton(문의사항)과 review 특성 상

하위 컬렉션 -> 루트 컬렉션 구조로 변경을 계획 중이다.



Firestore : firestore

Like 구조

 > like > 3bijfy8IdOV5xB... Google Cloud의 추가 기능		
 youngfarmer-35b0d	 like	 3bijfy8IdOV5xBT6iSh7
+ 컬렉션 시작	+ 문서 추가	+ 컬렉션 시작
cart	3bijfy8IdOV5xBT6iSh7 >	+ 필드 추가
like >	Bm3US7nniS9osyAz5QiA CA1YGxWitZuTfB1k59QT YE59GKXYgCCaJ48IkhKa gA1dJ1q5Xd89Jo0E7Nm9 hX1bUc2Rgt3wCJjHCbCp k6q1RCNf0grJg2PHjhdi s1HP235PmddPQVR0Xc7e z6Y2mki9aiAD0KFMYbvE	product_id: 1 uid: "mFiy1YSwJUA7jPmSZ0PvO5aAn12"
product		
store		
user		

like의 경우 전역 관리를 위해

product와 함께 **루트 컬렉션** 구조를 사용하고 있고,

product를 지칭하기 위해 **product_id**와 **uid**를 포함하고 있다.

Firestore : firestore

Cart 구조

🏠 > cart > 2fCtbQvyKL2Rn.. Google Cloud의 추가 기능		
🏠 youngfarmer-35b0d	📁 cart	📄 2fCtbQvyKL2Rnd75YA92
+ 컬렉션 시작	+ 문서 추가	+ 컬렉션 시작
cart >	09b0FYgatq6nWGTxI4wt	+ 필드 추가
like	2fCtbQvyKL2Rnd75YA92 >	option: {"item_weight":"3kg","number_of_item":"12개입","wanna_pave":"선물용 포장"}
product	9Az6Mm9Xxj8Bbz1PyPA7	product_id: 1
store	Lz4PBnPk0I7zjysDHx27	uid: "EKnzCgul49NaVZid0kMZGhaDJGG3"
user	QxN4wAtw3e5Qxu03dAnS	
	bJFvmmInsEp47mSuB8xI	
	cXt15fte0ZdGPe6cdwRd	
	jVvd4MA0re1kG6zP0bo3	

cart의 경우에도 like와 마찬가지로 **루트 컬렉션** 구조를 사용하고 있다.

Firebase : cloud function

Firestore용 Cloud Functions는 Firestore 기능 및 HTTPS 요청에 의해 트리거된 이벤트에 대한 응답으로 백엔드 코드를 자동으로 실행할 수 있는 서버리스 프레임워크입니다

즉 firebase에서 제공하는 serverless 서버이다. 서버리스 서버란 http 요청이 들어오지 않을 때에는 지니고 있는 대부분의 자원을 환원하였다가 요청이 들어올 때 자원을 차지하여 동작하는 서버를 의미한다.

Cloud function 사용중인 기능

로그인 제공업체 선택(1/2단계)

X

기본 제공업체

추가 제공업체

커스텀 제공업체

이메일/비밀번호

Google

Facebook

Play 게임즈

OpenID Connect

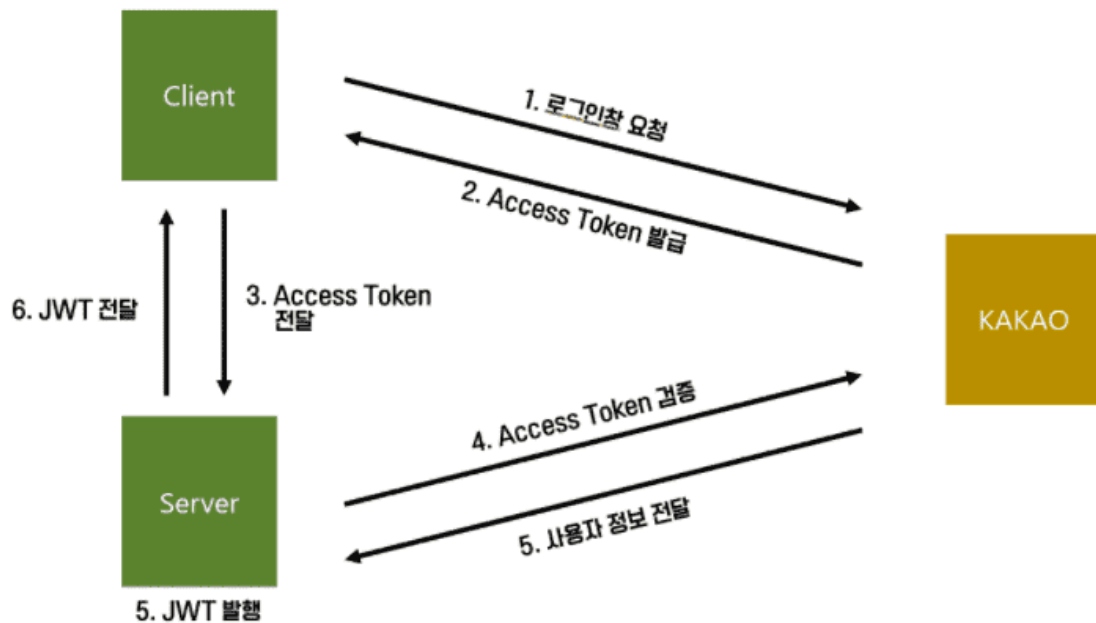
전화

게임 센터

Apple

GitHub

SAML

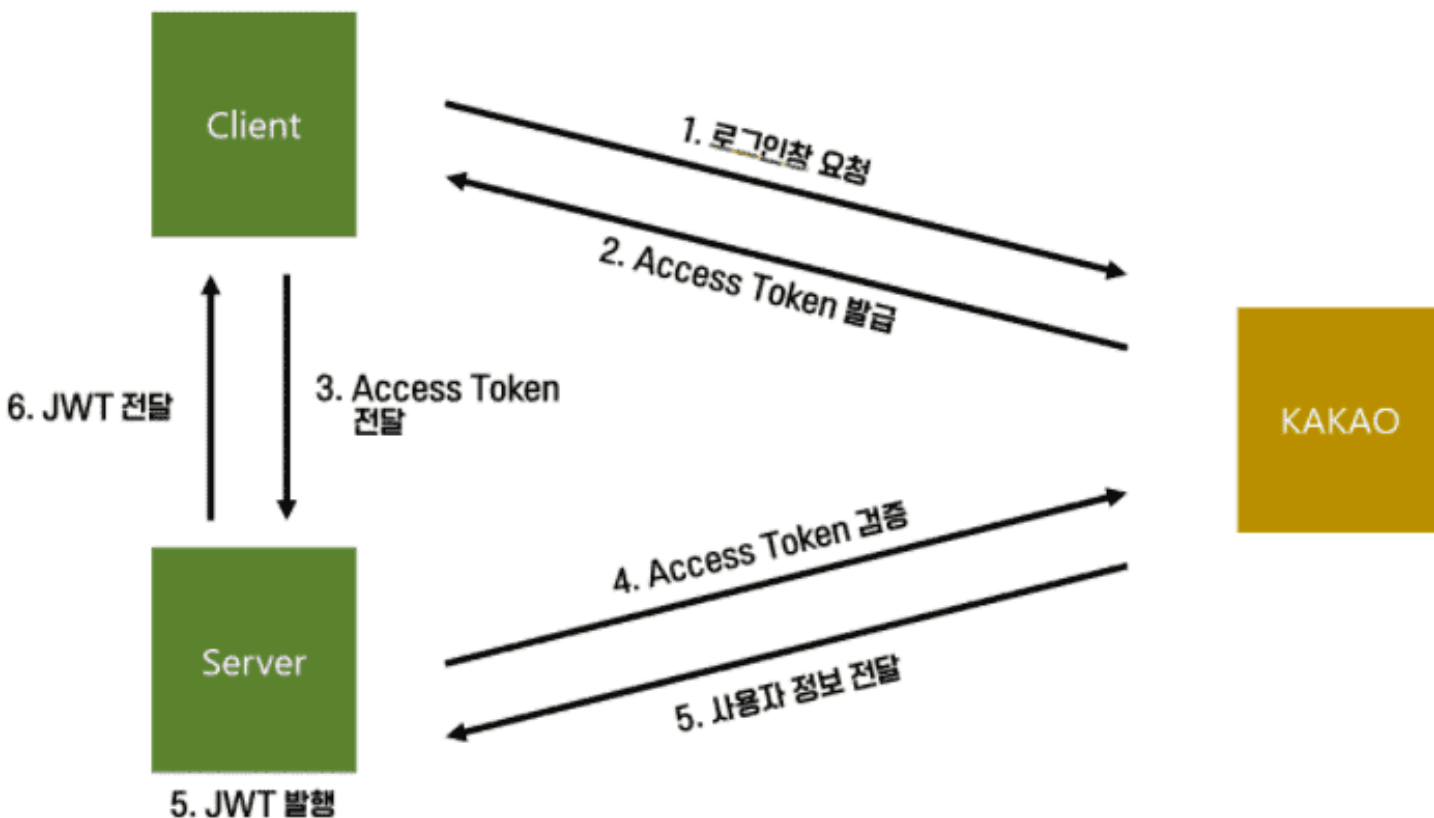


카카오 로그인 api와 firebase auth 기능을 연동 시키기 위해 cloud function을 사용 중이다.

카카오의 경우 **firebase auth**에 정식으로 지원 되지 않기 때문에, JWT 형식의 **Firestore custom token**을 발행하여 auth에 등록해야한다.

Firebase : cloud function

Cloud function 사용중인 기능



카카오로부터 auth token을 받으면,
그 토큰을 cloud function으로 보내어 custom
token을 받는다.

이때 cloud function은 받은 auth token을 카
카오 서버에 보내 실제 유저가 맞는지 확인 후,
맞다면 토큰을 생성하여 반환하는 형식이다.

Firebase : cloud function

Cloud Function 사용중인 기능

```
// actual endpoint that creates a firebase token with Kakao access token
app.post('/verifyToken', (req, res) => {
  const token = req.body.token;
  if (!token) return res.status(400).send({error: 'There is no token.'})
  .send({message: 'Access token is a required parameter.'});

  console.log(`Verifying Kakao token: ${token}`);

  createFirebaseToken(token).then((firebaseToken) => {
    console.log(`Returning firebase token to user: ${firebaseToken}`);
    res.send({firebase_token: firebaseToken});
  });
});

exports.kakaoAPI = functions.region("asia-northeast3").https.onRequest(app);
```

```
/**
 * createFirebaseToken - returns Firebase token using Firebase Admin SDK
 *
 * @param {String} kakaoAccessToken access token from Kakao Login API
 * @return {Promise<String>}         Firebase token in a promise
 */
function createFirebaseToken(kakaoAccessToken) {
  return requestMe(kakaoAccessToken).then((response) => {
    console.log(response);
    const body = JSON.parse(response);
    console.log(body);
    const userId = `kakao:${body.id}`;
    if (!userId) {
      return res.status(404)
        .send({message: 'There was no user with the given access token.'});
    }
    let nickname = null;
    let profileImage = null;
    if (body.properties) {
      nickname = body.properties.nickname;
      profileImage = body.properties.profile_image;
    }
    return updateOrCreateUser(userId, body.kaccount_email, nickname,
      profileImage);
  }).then((userRecord) => {
    const userId = userRecord.uid;
    console.log(`creating a custom firebase token based on uid ${userId}`);
    return firebaseAdmin.auth().createCustomToken(userId, {provider: 'KAKAO'});
  });
};
```

Part 4 **Firebase : cloud function**

Cloud Function 추가할 기능

실 결제는 보안과 밀접 관련이 있으므로 보다 해킹 위험이 적은 서버 측 구현이 필요하다 판단.
⇒ Firebase Cloud Function을 통해 구현

- 1) 거래 검증 및 데이터 동기화 = 결제 금액이 위조되지 않고 올바른지 확인
- 2) 검증 결과 클라이언트에 전송

kakao : 카카오페이 전용 API

[펼치기/감추기](#) | [목록보기](#) | [모두 펼치기](#)

GET /kakao/payment/orders

(카카오페이) 주문내역조회 API

Implementation Notes

(카카오페이) [카카오페이 주문내역조회 API](#)를 래핑한 API 입니다. 지원되는 request가 유사하며 응답되는 response는 카카오페이 API 와 동일합니다.

Response Class (Status 200)

Model | Model Schema

```
{
  "code": 0,
  "message": "string",
  "response": {}
}
```

파라미터 목록

파라미터	Value	Description	파라미터 유형	데이터 유형
payment_request_date	<input type="text" value="(required)"/>	YYYYMMDD 형식의 조회 날짜를 지정하시면 됩니다.	query	string
cid	<input type="text"/>	아임포트 내 설정된 카카오페이 CID를 기본값으로 합니다. 아임포트 계정 내 복수의 카카오페이 CID가 설정된 경우 지정하시면 됩니다.	query	string
page	<input type="text"/>	페이지 숫자(1부터 시작, 1이 기본값)	query	integer

요청하기

Response Content Type

Firestore Authentication

- 유저 로그인 관리를 firebase auth를 사용 중이다.

현재 사용하는 가장 큰 목적은 유저가 로그인 페이지로 접근하지 못하게,
로그인 안된 유저가 메인 페이지로 접근하지 못하도록 막기 위해서이다.

- 구조를 보면 auth를 관리하는 AuthProvider가 routes를 감싸고 있다.

Part 4 Firebase Authentication

Authentication 사용중인 기능

```
/**
 * * 변경 1 -> 2
 * 1. user가 reload될 때마다 null로 설정됨 -> auth 받아오는 중 -> login 페이지로 이동 -> auth 받아옴 -> main 페이지로 이동
 * 2. user가 reload될 때마다 null -> 렌더링 직전 localStorage 값 비교 -> true -> 바로 main -> 이후 auth 변경 때마다 localStorage 변경
 */
export const AuthProvider:FC<Props> = ({children}) :React.ReactElement|null => {
  const [user, setUser] = useState<null | boolean>(getItemWithExpireTime("user")? true : false);
  const dispatch = useDispatch();

  useEffect(()=>{
    getItemWithExpireTime("user")? setUser(true) : setUser(false);
    FirebaseAuth.onAuthStateChanged((data)=> {
      if(data){
        setItemWithExpireTime("user", true, 1000*60*60);
        dispatch(getLikeAction(data.uid));
        setUser(true);
      }
      else{
        removeItem("user");
        setUser(false);
      }
    })
  }, []);
  return (
    <AuthContext.Provider
      value={user!}
    >
      {children}
    </AuthContext.Provider>
  )
}
```

AuthProvider는

FirebaseAuth.onAuthStateChanged
메서드를 통해

유저의 auth 상태가 변경 될 때마다 user
의 로그인 여부를 변경한다.

Part 4

Firebase Authentication

Authentication 사용중인 기능

```
export const AuthContext = createContext(false);
```

```
function App() {
  const params = useParams();
  const location = useLocation();
  const navigate = useNavigate();
  const dispatch = useDispatch();

  return (
    <div style={{display: "flex", alignItems: "center", justifyContent: "center"}}>
      <BottomNavBarChanger>
        <ScrollToTop>
          <AuthProvider>
            <Routes>
              <Route path="/" element = {<PrivateRoute />} >...
            </Route>
              <Route path="/login" element = {<LoginRoute />} >...
            </Route>
              <Route path="*" element = {<h1>Page Not Found</h1>} />
            </Routes>
          </AuthProvider>
        </ScrollToTop>
      </BottomNavBarChanger>
    </div>
  );
}
```

```
export const PrivateRoute:FC<NavigateProps> = ({children, ...props}):any => {
  const currentUser = useContext(AuthContext);

  // console.log("++=====++" + currentUser + "++=====++");

  return (
    currentUser? <Outlet/>
    // : <LoginPage />
    : <Navigate to={"/login"}/>
  );
};

export const LoginRoute:FC<NavigateProps> = ({children, ...props}):any => {
  let currentUser = useContext(AuthContext);
  console.log("currentUser = " + currentUser);

  return (
    currentUser?
    <Navigate to={"/main"} /> : <Outlet/>
  );
};
```

변경된 유저의 로그인 정보는

`useContext(AuthContext)`를 통해 확인할 수 있는데,

이를 이용해 `PrivateRoute`과 `LoginRoute Wrapper`를 제작하였다.

이를 통해 로그인/비로그인 유저의 비정상적인 행동을 막을 수 있다.

Firebase Hosting

<https://youngfarmer-beta.web.app>

Challenges

1

Firebase 검색기능

2

회원가입 - 인증코드 전송

3

Redux 최적화

4

프론트엔드 성능

Challenges(Firebase 검색기능)

Firestore > Build

Was this helpful?  

Full-text search






[Send feedback](#)

Most apps allow users to search app content. For example, you may want to search for posts containing a certain word or notes you've written about a specific topic.

Cloud Firestore doesn't support native indexing or search for text fields in documents. Additionally, downloading an entire collection to search for fields client-side isn't practical.

To enable full text search of your Cloud Firestore data, use a dedicated third-party search service. These services provide advanced indexing and search capabilities far beyond what any simple database query can offer.

Before continuing, research then choose one of the search providers below:

- [Elastic](#) 
- [Algolia](#) 
- [Typesense](#) 

Solution: External search service

Elastic



Algolia

Typesense

Challenges(Firebase 검색기능)

Solution 1

content: "i like bookstore"

▼ keyword

0 "i like"

1 "b"

2 "bo"

3 "book"

4 "books"

5 "store"

6 "ooks"

title: "bookstore"

Solution 2

```
const q = query(
  collection(db, "post"), // 포스트 컬렉션
  where("title", ">=", keyword),
  where("title", "<=", keyword + "₩uf8ff"),
);
```

```
const resSnap = await getDocs(q);
```

Solution 3

- [Elastic](#)
- [Algolia](#)
- [Typesense](#)

Solution 4

모든 Product를 받아온 후 자체 검색

```
async function getSearchAPI(payload:any) {
  let search = payload.search;
  console.log(search);

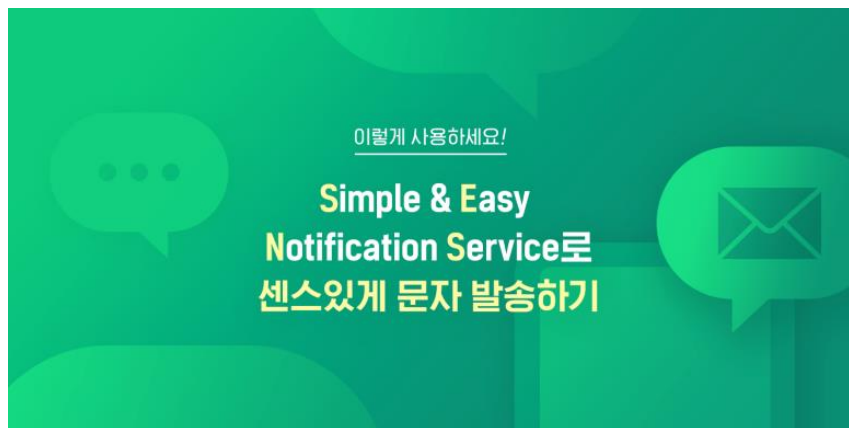
  const productRef = collection(db, "product");
  // ! 검색 기능 도입 필요 (firebase 자체 지원X, algolia 같은 3rd party 필요)
  const q = query(productRef,
  );
  const fbdata = await getDocs(q);

  let searchDataList:any = [];
  fbdata.docs.forEach((doc) => {
    const data = doc.data();
    if(`${data.title}`.includes(search)){
      console.log("include");
      searchDataList.push(data);
    }
  })
  console.log(searchDataList);

  return searchDataList;
}
```

Challenges(회원가입 인증코드 전송)

1. 메시지 형태 (비용 0)



2. Email 형태
(일정수준까지 비용 X)



Nodemailer

1. 유저 클릭

```
<div style={{display: "flex"}}>
  { /* // ! 무게, 개수, 포장 선택 유무에 따라 팝업 창 띄울지 결제/장바구니로 갈지 결정 */ }
  <BottomBoxShoppingCart onClick={e} => {
    if(modalselector.open_modal) {
      dispatch(cartAddAction(product_id, modalselector.select_item_info));
      dispatch(closeModalAction(modalselector));
    }
    else{
      dispatch(openModalAction(modalselector));
    }
  } } style={{marginRight: "9px"}}>
장바구니 </BottomBoxShoppingCart>
```

2. Cart 담기 액션 발생

```
export const cartAddAction = (product_id:number, sub_info: any) => {
  return {
    type: CART_TRY,
    payload: {
      type: CART_ADD_TRY,
      product_id: product_id,
      sub_info: sub_info
    }
  };
}
```

```
async function addCartAPI(payload:any) {
  console.log("addCartAPI");

  let uid = FirebaseAuth.currentUser!.uid;
  const product_id = Number(payload.product_id);

  const cartRef = collection(db, "cart");
  const q = query(cartRef, where("product_id", "==", product_id), where("uid", "==", uid));
  const fbdata = await getDocs(q);

  // 장바구니 담기 가능
  if(fbdata.empty){
    const result = await addDoc(cartRef, {
      uid: uid,
      product_id:product_id,
      option: JSON.stringify(payload.sub_info)
    });
  }
  // 장바구니 수정
  else {
    const result = await updateDoc(fbdata.docs[0].ref, {
      uid: uid,
      product_id:product_id,
      option: JSON.stringify(payload.sub_info)
    });
  }

  const cartfbDataList = await getDocs(
    query(cartRef, where("uid", "==", uid))
  );

  const cartDataList = cartfbDataList.docs.map((doc) => {
    return doc.data();
  });

  return cartDataList;
}
```


Challenges(Redux 최적화)

4. Reducer에서 상태 변경

```
export function CartReducer(state = cartInitState,
  switch (action.type) {
    case GET_CART_SUCCESS:
      return {
        ...state,
        carts: action.payload,
      };
    case GET_CART_FAIL:
      return {
        ...state,
        carts: action.payload,
      };
    case CART_ADD_SUCCESS:
      return {
        ...state,
        carts: action.payload,
      };
    case CART_ADD_FAIL:
      return {
        ...state,
      };
    case CART_CANCEL_SUCCESS:
      return {
        ...state,
        carts: action.payload,
      };
    case CART_CANCEL_FAIL:
      return {
        ...state,
        // likes: action.payload,
      };
  }
}
```

문제 발생!!

이전 상태 -> 현재 상태로 변경

=> 과거 장바구니 담았던 내역이
한번 렌더링 된 이후 변경

=> 깔끔하지 않은 UX

5. Store로부터 UI 수정

```
function CartPage(props: any) {
  const params = useParams();
  const location = useLocation();
  const navigate = useNavigate();
  const dispatch = useDispatch();

  const cartSelector: CartProductDataType[] = useSelector((state) =>
    state.SearchDetailReducer.cartProducts
  );

  const [allCheck, setAllCheck] = useState(false);
  const [order, setOrder] = useState(false);

  const orderSelector: OrderDataType[] = useSelector((state: RootState) =>
    state.OrderReducer.orders
  );

  if(cartSelector) {
    return (
      <AppFrame>
        <AppBarComponentOnlyBack title={"장바구니"} />
        <div style={{margin: "80px 16px 20px 16px"}}>
          <CartTopComp allCheck={allCheck} setAllCheck={setAllCheck} />
        </div>
        <div style={{paddingBottom: "88px",}}>
          {
            cartSelector.map((cartProduct) => {
              return(
                <CartProductComponent allCheck={allCheck} />
              )
            })
          }
        </div>
      </AppFrame>
    )
  }
}
```

Challenges(프론트엔드 성능측면)

사용 폰트 10 종류(40MB),
Asset 폴더 (30MB)

62

성능

76

접근성

100

권장사항

92

검색엔진 최적화

62

성능

같은 추정치이며 달라질 수 있습니다. 이러한 측정항목에서 성능 점수가 직접 계산됩니다. 계산기 보기

▲ 0-49 ■ 50-89 ● 90-100

측정항목

▲ First Contentful Paint

5.0 초

■ Speed Index

5.0 초

▲ Largest Contentful Paint

5.6 초

■ Time to Interactive

7.0 초

● Total Blocking Time

30 밀리초

● Cumulative Layout Shift

0.041

추천

예상 절감치

▲ 차세대 형식을 사용해 이미지 제공하기

3.6 s ▼

▲ 사용하지 않는 자바스크립트 줄이기

2.7 s ▼

▲ 텍스트 압축 사용

1.95 s ▼

■ 레거시 JavaScript를 최신 브라우저에 제공하지 않기

0.3 s ▼

이러한 권장사항은 페이지를 더 빠르게 로드하는 데 도움이 될 수 있습니다. 성능 점수에는 직접적인 영향을 미치지 않습니다.

진단

▲ 효율적인 캐시 정책을 사용하여 정적인 애셋 제공하기 - 리소스 10개 발견됨 ▼

▲ 웹폰트가 로드되는 동안 텍스트가 계속 표시되는지 확인하기 ▼

▲ First Contentful Paint (3G) - 10830 ms ▼

▲ 네트워크 페이로드가 커지지 않도록 관리하기 - 총 크기: 6,395KiB ▼

■ 기본 스레드 작업 최소화하기 - 2.3 초 ▼

Limitaion

Tool이 아닌 Platfrom

정상적으로 WebApp을 구동하기
위해서 실 판매를 할 농가를
직접 찾아야함.

현재는 가상데이터를 이용하여 구현
중.

Firestore의 한계

실 서비스 구현 속도가 올라가는 반면,
여러 제약 사항과 성능 저하를 일으킴.

감사합니다!