

E조

엄태경

김민은

장병우

심상원

아이템 변경



아이템 변경 사유

1. 로봇틱스 분야를 자신 있게 이끌 수 있는 인원 X
2. 객체탐지 기술은 오픈소스 돌리기만 하면 완료 -> 실질적인 산출물은 대부분 HW
3. 산출물을 보여줄 방법 (시뮬레이터? 런닝머신? 소형 컨베이어벨트?)

=> App이나 Web

목차

1. 간략한 아이템 소개

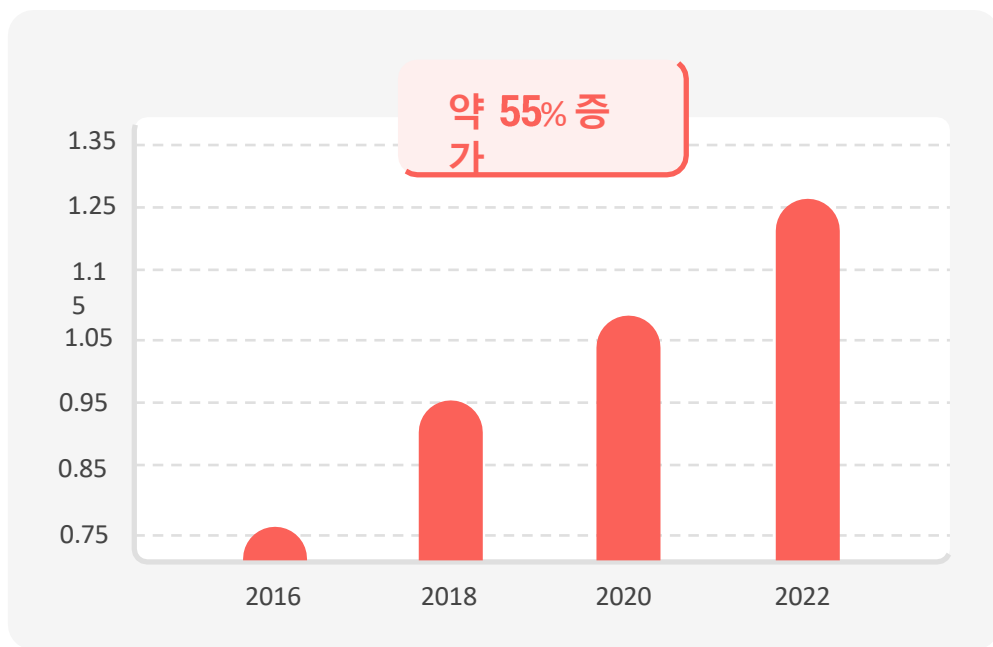
2. 사용 라이브러리/기술 소개

3. 구현 계획

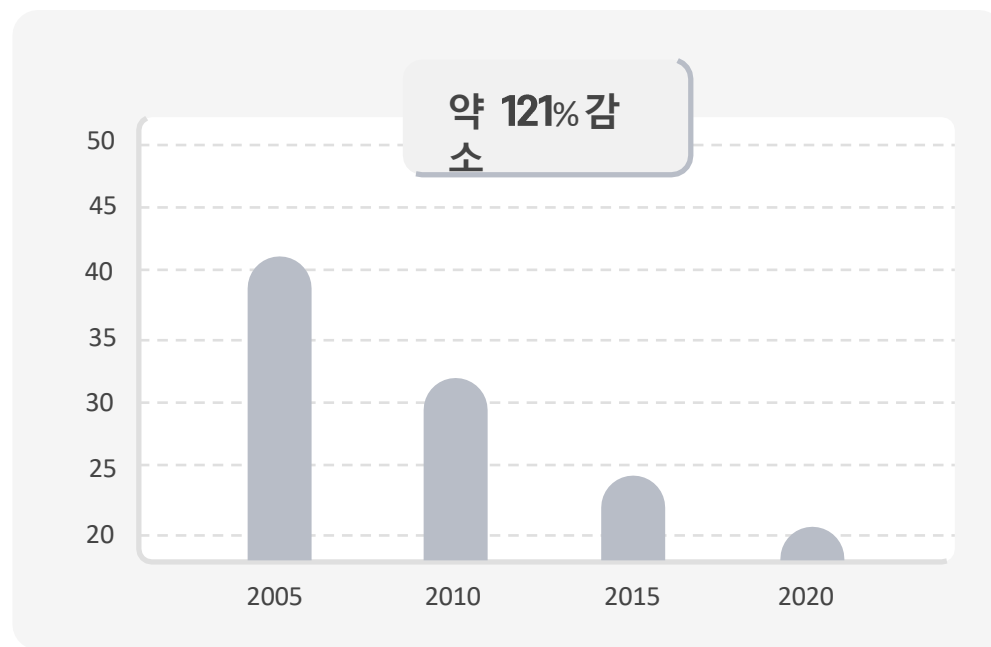
01

Problem

보수적이던 기존 농부는 줄어들고, 새로운 것을 적극 수용하는 청년층 농부가 급격히 증가



빠르게 성장 중인 청년 농부



급격히 감소 중인 기존 농부 (60대 이상)

Problem 기존 과일 유통구조

불필요한 유통단계로 인해 복잡하며 비효율적인 기존 구조
-> 유통단계 간소화를 통해 농업인, 소비자 모두에게 윈윈

기존 구조



기존 유통구조

농산물 유통구조 매우 복잡, 제 값을 못 받고 헐 값에 판매

직거래 구조

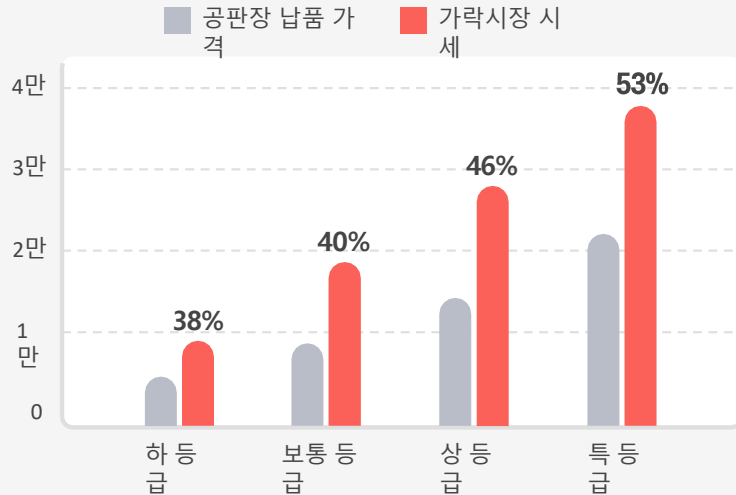


직거래 유통구조

농산물 제 값에 받고, 소비자도 싼 가격에 신선한 과일 구매

Problem 농업인의 고 충

“ 직거래를 하고 싶은데 어디다 어떻게 올려야 할지 모르겠어요. ”



판매경로1) 공판장

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

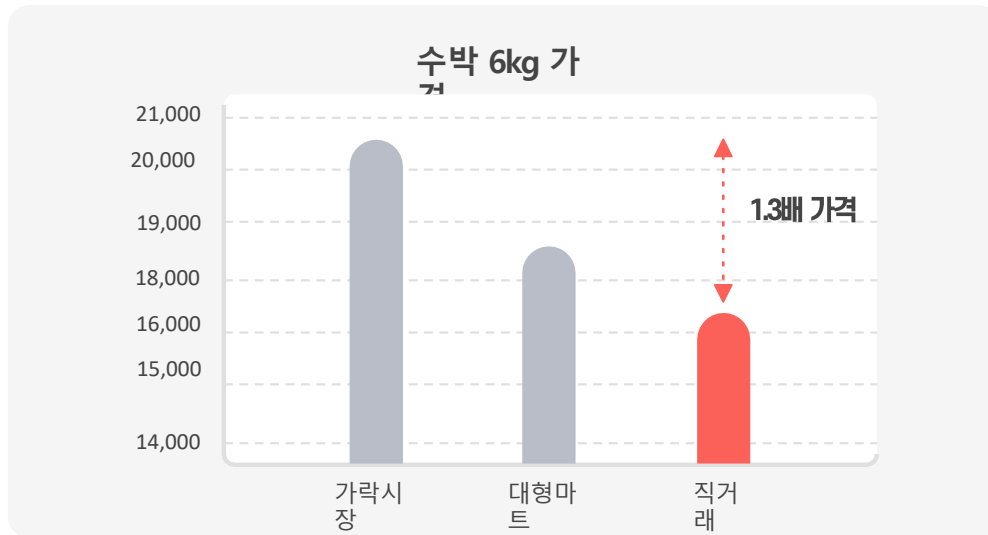


판매경로2) 오픈마켓 직거래

소비자 노출을 위해선 별도의 브랜딩 작업 필
요 품목 별 1,000개 이상의 제품 존재 -> **노출되기 까
다로움**

Problem 소비자의 고
충

“ 마트/시장 과일 너무 비싸고 신선하지도 않아요. ”



저렴
함

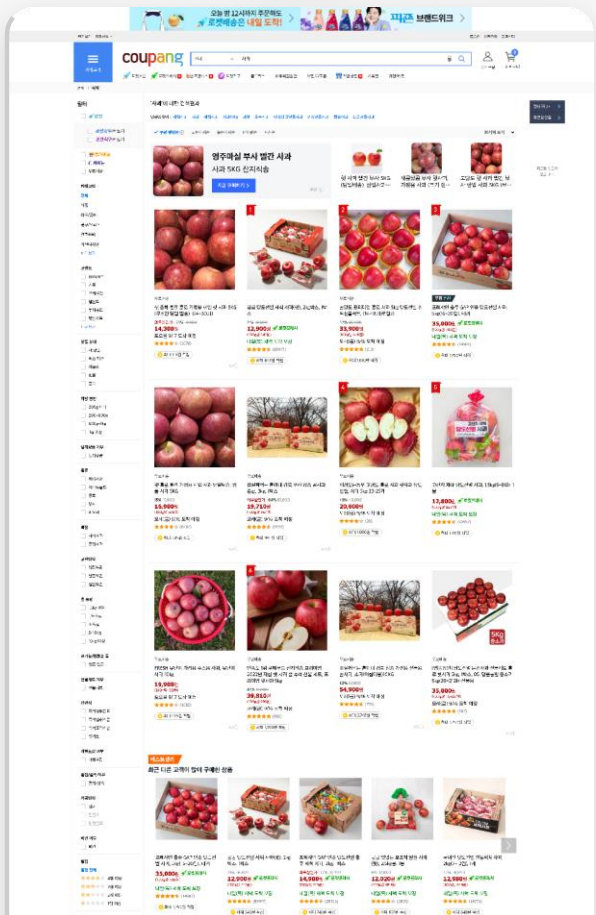
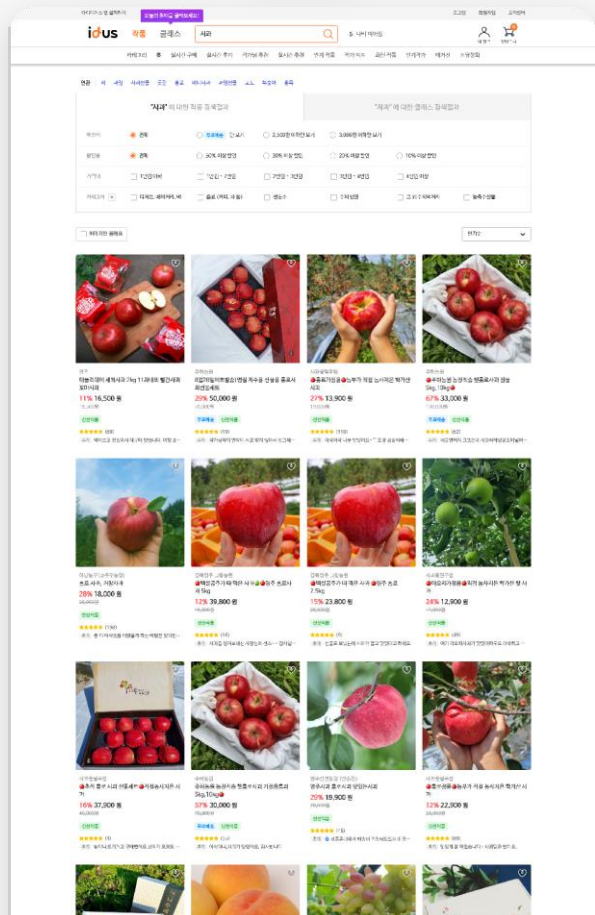


선주문 후수확 - 신
선함

02

Solution

Solution 경쟁사 분석



사과를 검색했을 때 나오는 결과 수 = 1776개

단순히 맛있고 신선한 사과를 싸게 먹고 싶었을 뿐인데 고민이 너무 많아짐

상품의 변별력은 떨어지고
그저 키워드 싸움 뿐인 현재의 오픈마켓

경쟁사

우수한 품질의 농수축산물을 생산하는 생산자와 소비자를 연결하는 오픈마켓



아이디어스

농수산물 매출액 120억



쿠팡

로켓 프레시 연간 매출액 2조 3200억



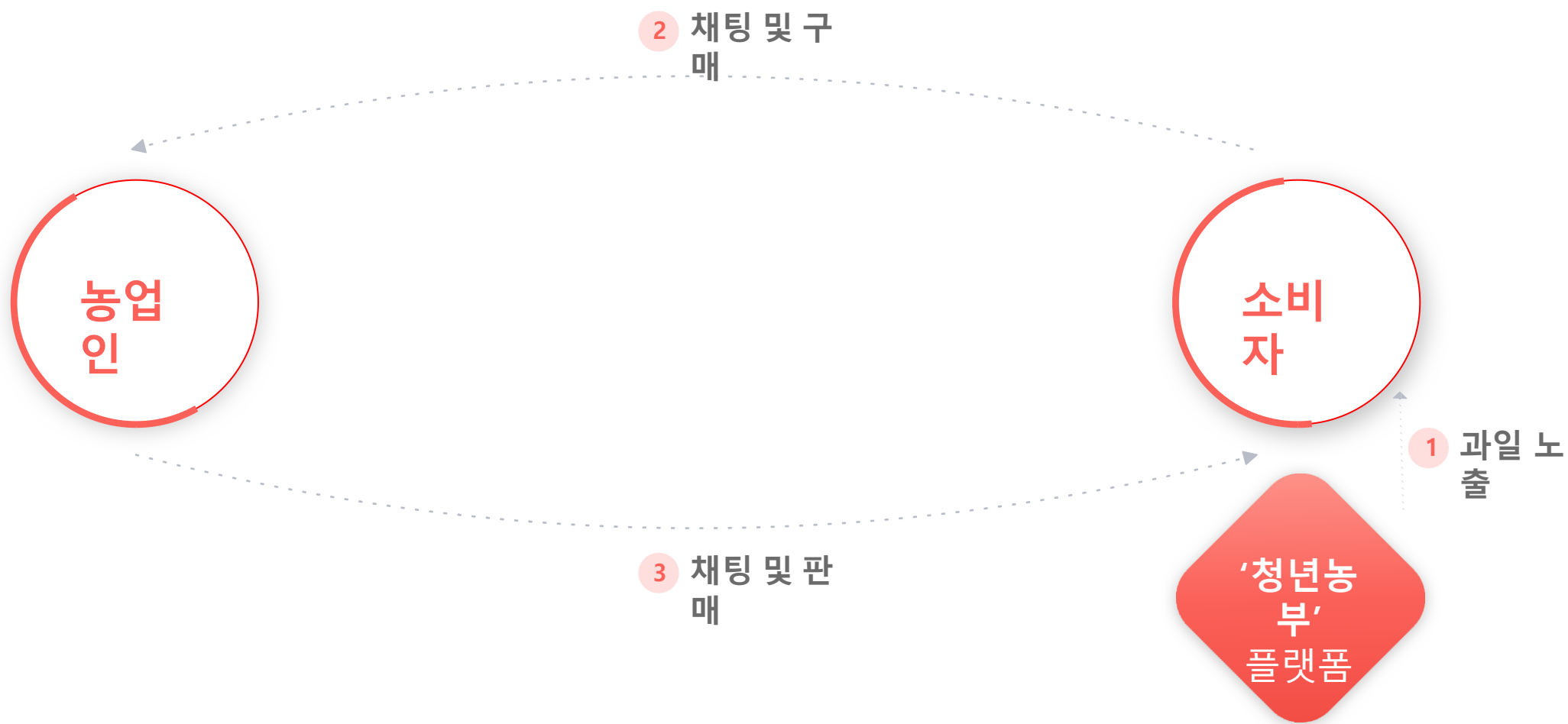
각 농가만의 **스토리/특장점**을 살릴 수 있는 **과일 전**



플랫폼

농가와의 1:1 채팅을 통한
국내 최초 커스터마이징
기능

소비자와 농업인의 진짜 직접 거래

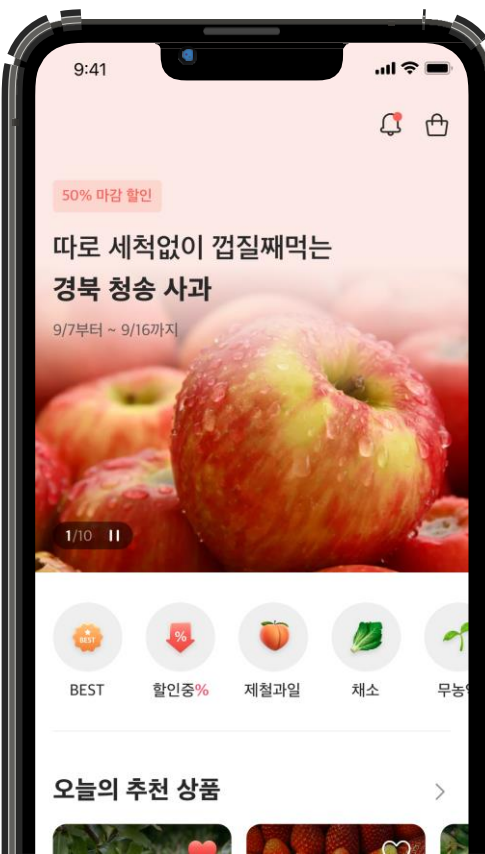


Solution 차별화 포인
트

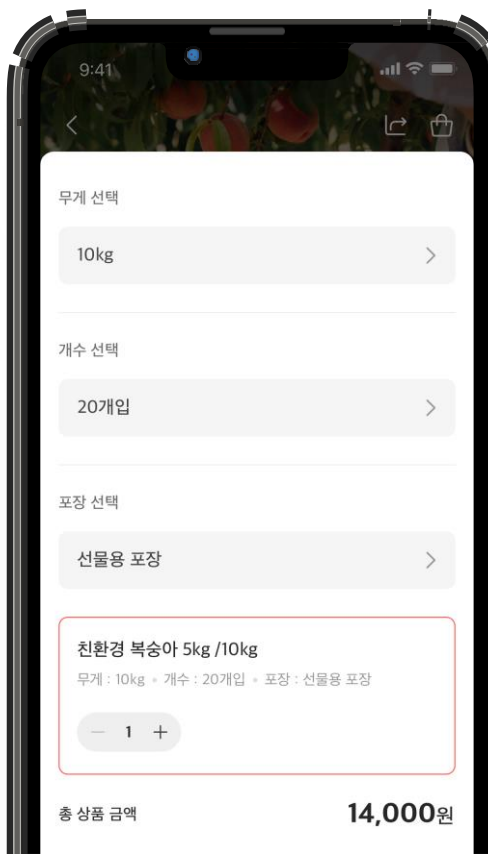
	차별화 포인 트	내 용
소비자	큐레이 션	품종 별 우수한 농가만 입점시켜, 선주문 후수확한 신선한 과일을 고민 없이 바로 받아먹을 수 있음
	커스터마이 징	소수의 농가만 입점 되어 있어, 농부와의 실시간 채팅을 통한, 커스텀 과일 주 문 가능
농가	라이브 커머 스	라이브 커머스를 통한 과일 판매 촉진, 인지도 향상

갓 수확한 신선한 과일을, 고민 없이, 집 안에서 받아볼 수 있는 '청년농부'

큐레이션 기능



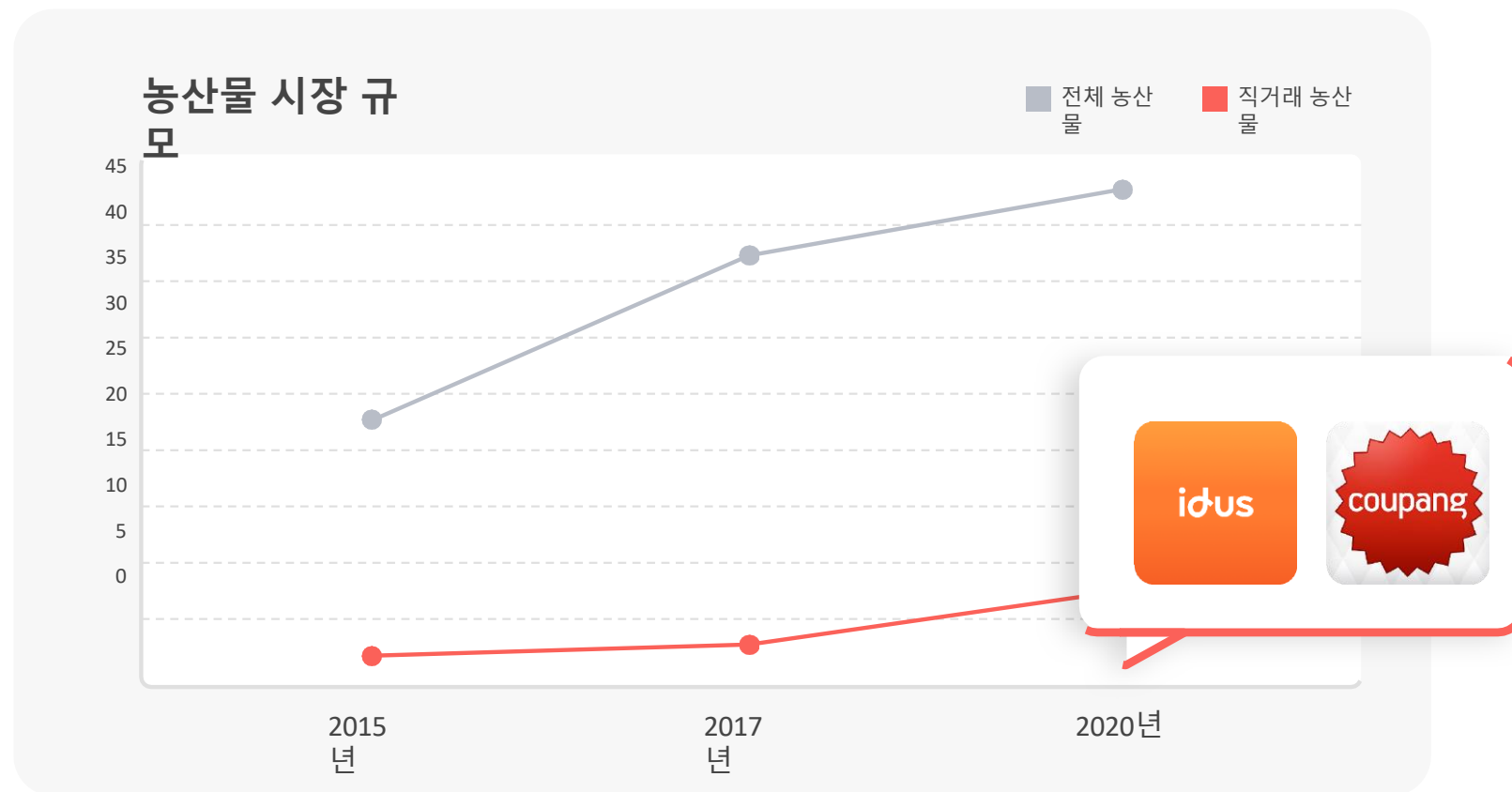
커스터마이징 구매



라이브커머스



전체 농산물 거래 규모 > 직거래 시장 규모 > 경쟁사 시장 규모

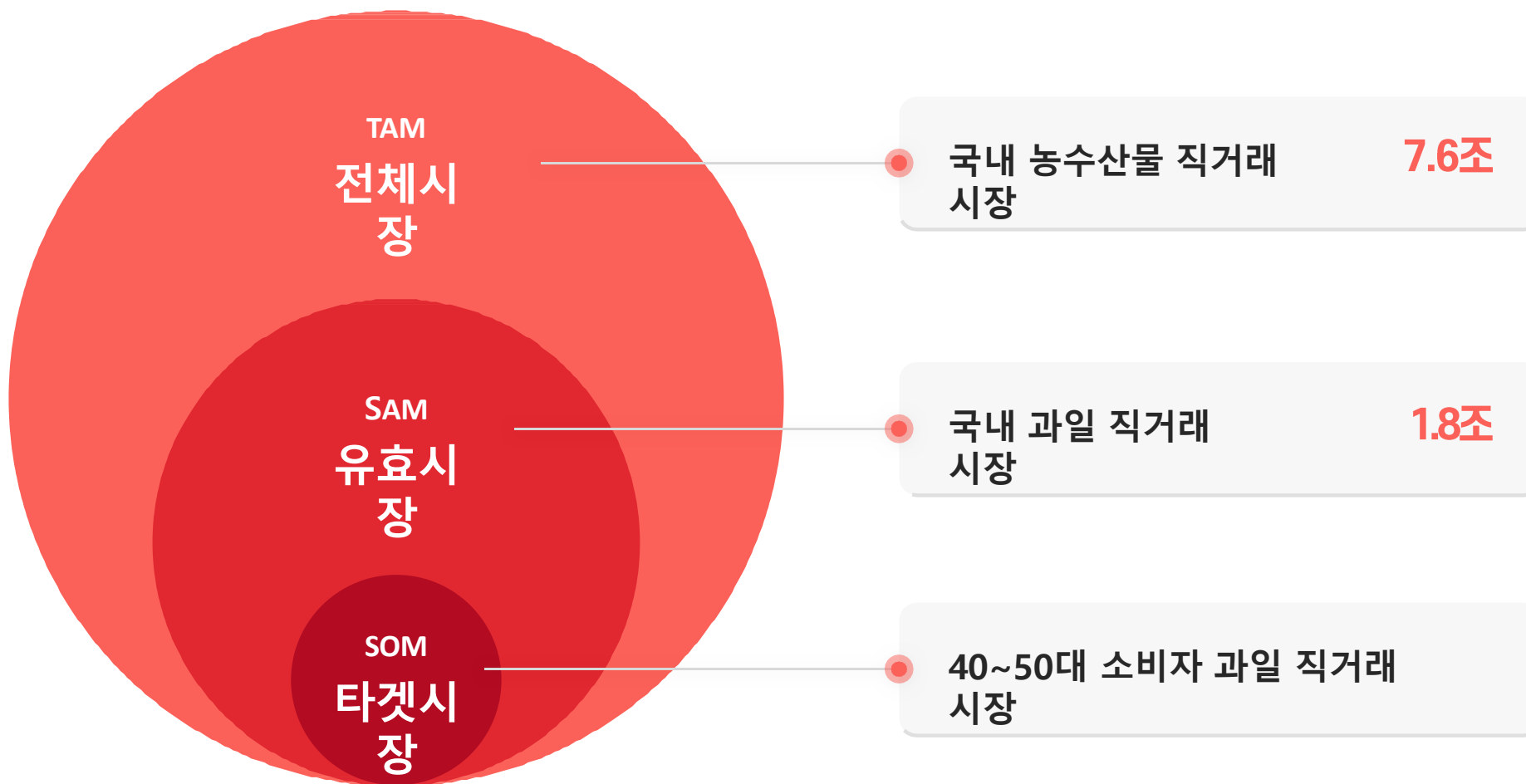


03

Scale-Up

판매처 농가 5개 네트워크 보유, 긍정적 반응





산출물 소개

1. 실제 거래가 가능한 WebApp
2. FrontEnd: React + Typescript
3. BackEnd: Firebase

Front End

1. React + Typescript

2. Redux, Redux-Saga

3. Atomic Design

4. CSS 적용 방법

Front End – React+Typescript

React는 웹 프레임워크로, 자바스크립트 라이브러리의 하나로서 사용자 인터페이스를 만들기 위해 사용된다.

1. Component 기반 구조 => Atomic Design Pattern

UI(View)를 여러 컴포넌트(component)를 쪼개서 만들
기능 단위, UI 단위로 캡슐화 시켜 코드를 재사용할 수 있기에 유지보수 및 관리가 용이해짐

2. Virtual Dom

DOM은 Document Object Model의 약자로 html, xml, CSS 등을 트리 구조로 인식하고, 데이터를 객체로 간주하고 관리
Virtual DOM은 가상의 Document Object Model로서 이벤트가 발생할 때마다 Virtual DOM을 만들고,
다시 그릴 때마다 실제 DOM과 비교하고 전후 상태를 비교해, 변경이 필요한 최소한의 변경사항만 실제 DOM에 반영해,
앱의 효율성과 속도를 개선할 수 있음

3. Props and State => Props Drilling => Redux

Props: 부모 컴포넌트에서 자식 컴포넌트로 전달해 주는 데이터

State: 동적인 데이터를 다룰 때 사용

Front End – React+Typescript

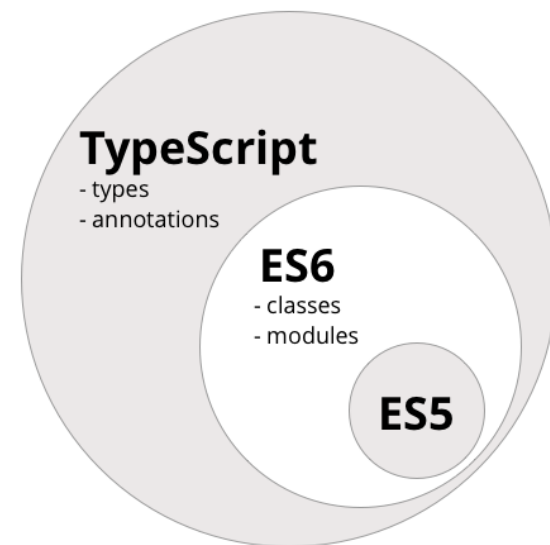
타입스크립트(TypeScript)는 자바스크립트(JavaScript)를 기반으로 정적 타입 문법을 추가한 프로그래밍 언어

1. 컴파일 언어, 정적 타입 언어

장점1) 코드 작성 단계에서 타입을 체크해 오류를 확인할 수 있고 미리 타입을 결정하기 때문에 실행 속도가 매우 빠름

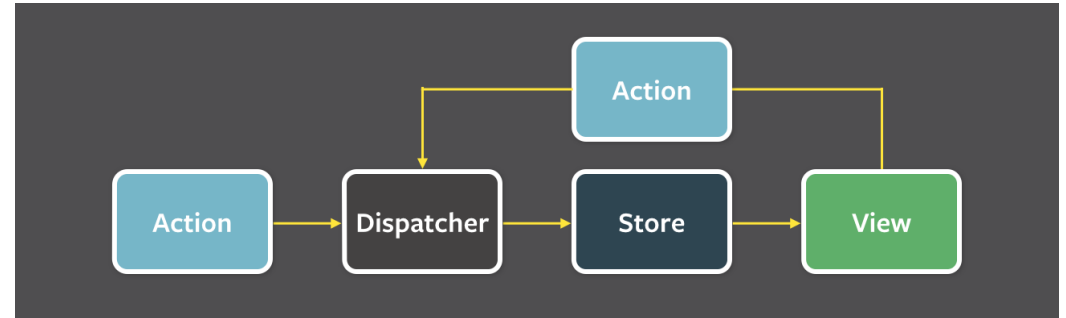
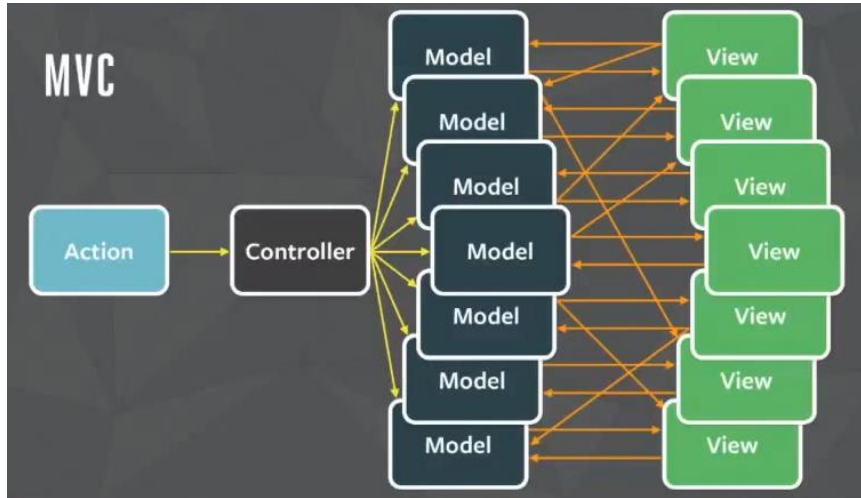
단점1) 코드 작성 시 매번 타입을 결정해야 하기 때문에 번거롭고 코드량이 증가하며 컴파일 시간이 오래 걸림

=> 장점2) 높은 가독성 및 유지보수의 편의성: 장점

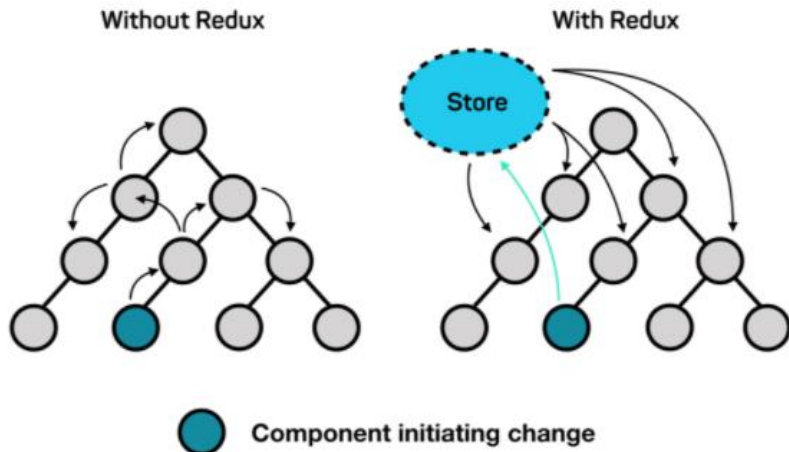


Front End – Redux, Redux-Saga

Javascript 상태관리 라이브러리



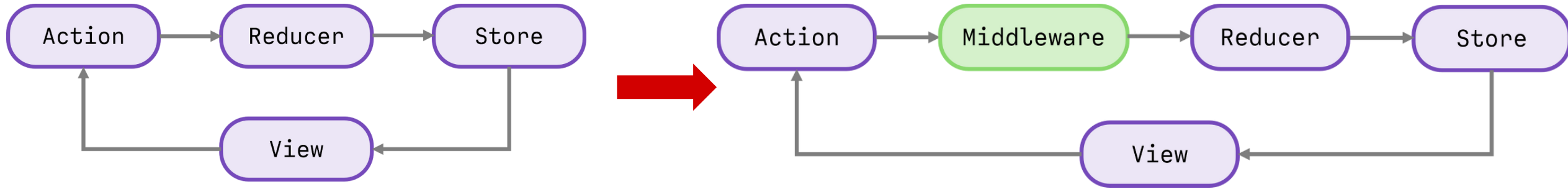
모든 View가 Store를 바라보는 단방향 구조



- Props Drilling 사전 예방 => 성능 향상
- View와 Controller를 격리시킴으로써 보다 쉬운 유지 보수

Front End – Redux, Redux-Saga

Redux-saga는 react/redux 애플리케이션의 데이터 fetching이나 브라우저 캐시에 접근하는 순수하지 않은 비동기 동작들을 더 쉽고 보기 좋게 만드는 것을 목적으로 하는 라이브러리



action을 발생 => reducer를 통해 State를 변화시켜 store를 갱신

Action과 reducer 사이에서 흐름을 제어하는 미들웨어
중간에서 비동기 작업 수행

API 호출 로직의 효율적인 관리

async/await 라는 기능으로 비동기 동작들을 처리할 수 있는데 굳이 코드량을 늘려가며 redux-saga를 도입할 이유가 있을까?

=> redux-saga에서는 더 세부적인 컨트롤 가능

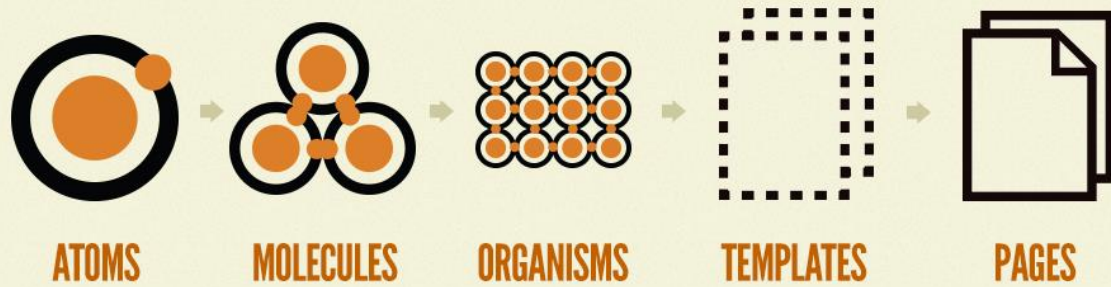
ex) 사용자 부주의로 동일 api 여러 번 호출하는 경우 => 가장 마지막 호출의 response만 받아올 수 있도록 제어 가능

=> Presentational 컴포넌트와 Container 컴포넌트의 명확한 분리

=> 다른 페이지에서 동일 API 필요 시 같은 코드 중복 작성 필요 X

Front End – Atomic Design Pattern

작은 단위의 컴포넌트를 재사용성이 강하고 단단하게 잘 설계함으로써 점진적으로 지속적으로 개발하기 용이하게 하는 방법론



Overall

```
ubuntu@DESKTOP-00KI5C5:/mnt/c/src/youngFarmer$ tree -L 2 ./src
./src
├── App.tsx
├── api
│   └── axios.ts
├── assets
│   ├── fonts
│   └── images
├── common
│   ├── AppBar
│   ├── BottomNavigationBar
│   ├── Category
│   ├── ItemList
│   └── LiveItem
├── custom.d.ts
├── firebaseConfig.json
├── index.css
├── index.tsx
├── pages
│   ├── ChatPage
│   ├── LikePage
│   ├── LiveListPage
│   ├── LoginPage
│   ├── MainPage
│   ├── MyPage
│   ├── ProductPage
│   ├── SearchDetailPage
│   ├── SearchPage
│   ├── SplashPage
│   └── TodayRecommendPage
├── reducers
│   ├── BottomNavigationBarReducer.ts
│   ├── SearchReducer.ts
│   ├── SplashReducer.ts
│   └── index.ts
├── sagas
│   ├── SplashSaga.ts
│   └── index.ts
├── services
│   └── BottomNavBarChanger.tsx
└── setupProxy.js
```

Page

```
ubuntu@DESKTOP-00KI5C5:/mnt/c/src/youngFarmer$ tree -L 2 ./src/pages/SearchPage
./src/pages/SearchPage
├── SearchActions.ts
├── SearchConstants.ts
├── SearchPage.tsx
├── atoms
│   ├── RecentSearch.tsx
│   └── RecommendStore.tsx
├── components
│   ├── RecentSearch.tsx
│   └── RecommendStore.tsx
```

Reducer

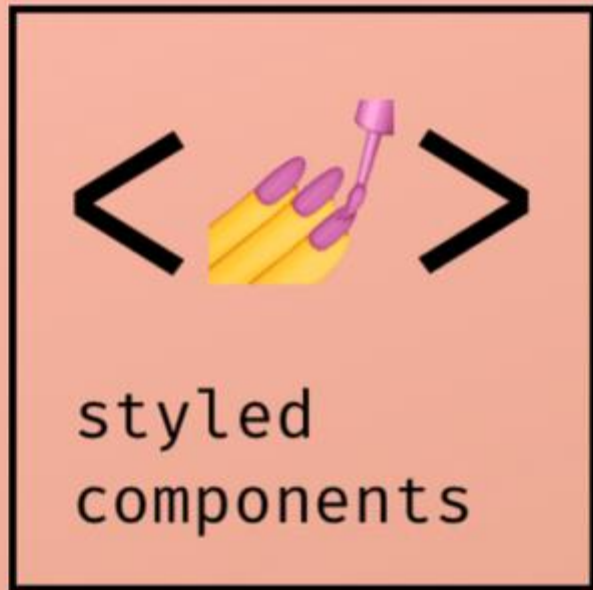
```
ubuntu@DESKTOP-00KI5C5:/mnt/c/src/youngFarmer$ tree -L 2 ./src/reducers
./src/reducers
├── BottomNavigationBarReducer.ts
├── SearchReducer.ts
├── SplashReducer.ts
└── index.ts
```

Saga

```
ubuntu@DESKTOP-00KI5C5:/mnt/c/src/youngFarmer$ tree -L 2 ./src/sagas
./src/sagas
├── SplashSaga.ts
└── index.ts
```

Front End – CSS 적용 방법

기존 돔을 만드는 방식인 css, scss 파일을 밖에 두고, 태그나 id, class이름으로 가져와 쓰지 않고, 동일한 컴포넌트에서 컴포넌트 이름을 쓰듯 스타일을 지정하는 것을 styled-components라고 부릅니다. css 파일을 밖에 두지 않고, 컴포넌트 내부에 넣기 때문에, css가 전역으로 중첩되지 않도록 만들어주는 장점이 있습니다.



Back End

시간이 촉박한 관계로 다른 팀 진도를 따라잡기 위해 Firebase 선택

Firebase

1. Firebase Hosting

2. Firebase Storage

3. Firestore

4. Firebase Cloud Function



AWS

1. EC2

2. S3

3. RDS

4. Lambda

핵심 기능

1. 유저 관리

Firebase Auth(Google, Github, Facebook) + SNS Login

2. 과일 직거래

1. 찜하기, 장바구니
2. 검색
3. 리뷰

[Firestore, Firebase cloud]

4. 실시간 채팅

[Socket I.O. vs Firestore vs Firebase Realtime Database]

5. 결제

[I'mport 테스트 모드]

3. (Future Work..?)라이브 커머스

1. 라이브 스트리밍

[WebRTC]

핵심 기능 – 실시간 채팅

대부분의 기능을 Firebase로 구현하므로, Firebase 선택 => Google이 추천한 실시간 데이터베이스 사용

주요 고려사항

두 데이터베이스의 공통 핵심 기능 외에도 아래에 나열된 고려사항이 앱의 성공에 어떤 영향을 미칠지 생각해 보세요.

데이터베이스의 역할	<div>내 앱에서 데이터베이스를 사용하는 목적은...</div> <div>주로 데이터 동기화(기본 쿼리 사용)입니다.</div> <div>고급 쿼리, 정렬, 트랜잭션입니다.</div>
데이터 작업	<div>내 앱에서 데이터베이스를 사용하면...</div> <div>수 GB 이하의 데이터가 자주 변경됩니다.</div> <div>수백 GB 내지 TB의 데이터가 변경되며, 훨씬 더 높은 빈도로 읽힙니다.</div>
데이터 모델	<div>원하는 데이터 구조는...</div> <div>간단한 JSON 트리입니다.</div> <div>컬렉션으로 정리된 문서입니다.</div>
가용성	<div>가용성 요구사항은...</div> <div>99.999%의 매우 높은 업타임을 보장해야 합니다.</div> <div>99.95% 이상의 업타임을 보장해야 합니다.</div>

주요 고려사항

두 데이터베이스의 공통 핵심 기능 외에도 아래에 나열된 고려사항이 앱의 성공에 어떤 영향을 미칠지 생각해 보세요.

데이터베이스의 역할	<div>내 앱에서 데이터베이스를 사용하는 목적은...</div> <div>고급 쿼리, 정렬, 트랜잭션이 필요하지 않으면 실시간 데이터베이스를 사용하는 것이 좋습니다.</div>
데이터 작업	<div>내 앱에서 데이터베이스를 사용하면...</div> <div>디지털 화이트보드 앱처럼 앱이 소규모 업데이트를 잇달아 보내는 경우 실시간 데이터베이스를 사용하는 것이 좋습니다.</div>
데이터 모델	<div>원하는 데이터 구조는...</div> <div>비정형 JSON 데이터의 경우 실시간 데이터베이스를 사용하는 것이 좋습니다.</div>
가용성	<div>가용성 요구사항은...</div> <div>가용성이 매우 높아야 하지만 결정적이지 않은 경우 Cloud Firestore 또는 실시간 데이터베이스를 사용하는 것이 좋습니다.</div>
로컬 데이터에 대한 오프라인 쿼리	<div>내 앱은 연결이 제한되거나 없는 기기에서 쿼리를 수행해야 하는 빈도가...</div> <div>사용자가 계속 온라인 상태일 것으로 예상되는 경우 Cloud Firestore 또는 실시간 데이터베이스를 사용하는 것이 좋습니다.</div>
데이터베이스 인스턴스 수	<div>내 개별 프로젝트에서는...</div> <div>단일 Firebase 프로젝트에 여러 데이터베이스를 추가할 수 있으므로 실시간 데이터베이스를 사용하는 것이 좋습니다.</div>

핵심 기능 - 결제

실 결제는 보안과 밀접한 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

핵심 기능 - 라이브커머스

Live Stream API > 문서 > 가이드

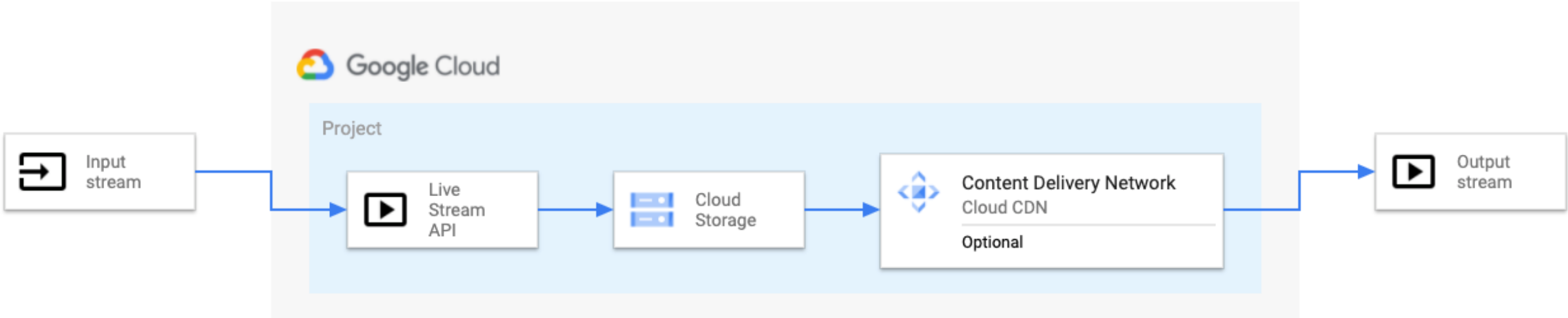
도움이 되었나요?  

Live Stream API 개요

[의견 보내기](#)

이 페이지에서는 Live Stream API를 간략히 설명합니다. Live Stream API는 여러 기기 플랫폼용으로 메자닌 실시간 신호를 HTTP 동적 적응형 스트리밍(DASH/MPEG-DASH), HTTP 실시간 스트리밍(HLS)을 비롯한 소비자 직접 스트리밍 형식으로 트랜스코딩합니다.

먼저 입력 엔드포인트를 만든 다음 실시간 SRT 또는 RTMP 입력 신호를 HLS 또는 DASH 출력 스트림으로 트랜스코딩하는 채널 리소스를 만듭니다. Live Stream API에서 만든 출력 스트림은 Cloud Storage 버킷에 저장됩니다. 그런 다음 Cloud Storage 버킷을 Cloud CDN의 백엔드로 구성할 수 있습니다.





50% 마감 할인

따로 세척없이 껍질째먹는 경북 청송 사과

9/7부터 ~ 9/16까지



1/10 ||



BEST



할인중%



제철과일



채소



무농약

오늘의 추천 상품



감사합니다!

과일 진짜 직거래 플랫폼