Application for young farmer

Team E

Kim Mineun[1], Sim Sangwon[2], Eaum Taekyung[2], Jang Byungwoo[3]

[1]Department of Math Education, SKKU

[2]Department of Mathematics, SKKU

[3]Department of Electronic and Electrical Engineering, SKKU

October, 2022

Abstract

Compared to the size of the agricultural products market, the size of the online direct trading market is still insignificant. Therefore, judging that it is a blue ocean and developing and servicing an online brokerage platform project was targeted. Naver, Google, Kakao, etc. various platform account ID Convenience is ensured by enabling login. The essential functions of an online brokerage platform were implemented and the basic elements were equipped. There is a shopping cart, review, steaming function, and payment function. With the introduction of a real-time chat function, you can communicate with the farmer in real time. A variety of customizations that are not possible with only selected items are possible. Access token-based payment counterfeit verification. A more secure payment guarantee system was established. Simple and fast development process for demo was pursued. FireAuth and Firestore were used. Performed React state management through Redux and Saga. Developed a react-based webApp composed of atoms, components, and pages based on atomic design.

Keywords : Fruit brokerage platform, React, Firebase

1 Introduction

The online agricultural products market is growing rapidly every year. Nevertheless, compared to the size of the agricultural products market, the agricultural products online market is remarkably small. This means that there are many farmers and consumers who are not using it yet. Also, even for farmers using the existing platform, opinions on the new platform positive.

In the case of large brokerage platforms, it is difficult to be seen at the top due to competitors. Also, as part of a strategy to diversify sales channels, It is

judged that there is a lot of interest in the new platform as well. Because mobile accessibility is much more convenient, the goal was to create a WebApp-based mediation platform.

Basic platform functions, including the chat function, were set as the primary goal. Also, according to the remaining time of the project, we aimed to create a practically usable WebApp mediation platform equipped with detailed functions such as token-based payment forgery verification logic.

In order to take advantage of the advantages of collaborative development, Atomic design was adopted. It was used to implement Page by implementing Atom and Components. Reduce unnecessary code by facilitating the reuse of components that are used redundantly. Sub-components were divided in detail to increase collaboration convenience.

We developed React-based frontend, which is widely used for implementing responsive WebApps. Redux was used to improve React performance by preventing a chain of unnecessary rendering due to status changes in React. Redux-Saga was additionally used for asynchronous management of Status. For the convenience of implementing the demo version, the backend server was replaced with Firebase. To implement the login function, FireAuth and API provided by the platform were used. Firestore was used as a DB for storing various product information and data.

As a result, React and Firebase-based WebApp mediation platform was implemented. Most of the targeted detailed functions (such as adding to the shopping cart, etc.) have been implemented. Access token-based payment forgery verification logic has been implemented to prevent transaction tampering attacks, and we have tried to pay attention to payment-related security in preparation for the launch of the platform. We have completed the implementation of a brokerage platform where real transactions can be made, consisting of a UI/UX that is consistent and comfortable in the red line.

2 Design for the Proposed Service

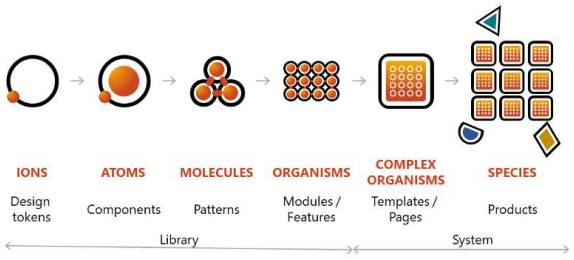

Figure 1: Atomic design

2.1 Atomic Design pattern

We proceeded with development using Atom Design pattern. This design pattern creates a small unit component, a larger component with that small component, and eventually a page. It was possible to increase code reuse because it makes large components from small components.

2.1.1 Atom

As it is the smallest component of the unit, it was made as abstract as possible to increase code reuse. Only CSS of components were defined using the stylized component package.

2.1.2 Molecule and Organism

By combining atoms, we created specific components to be used on the page. In this case, a specific value such as the positions of atoms was set using Flexbox.

2.1.3 page

The pages were constructed by arranging the molecules and organisms made by combining atom in order.

2.2 pages

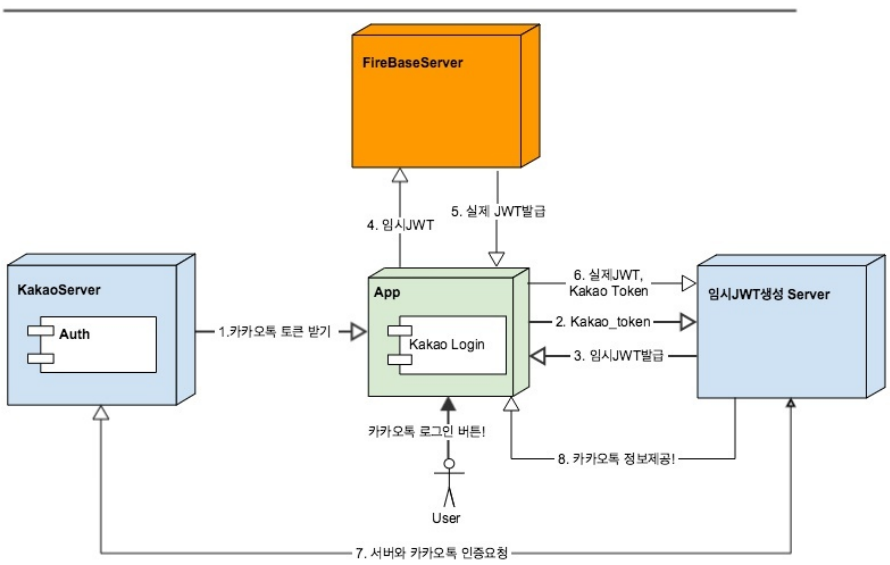

Figure 2: Kakao and Firebase login

2.2.1 Login Page

It has implemented SNS login functions such as Kakao.

Kakao Login page When authentication between the Kakao server and the user is completed, Kakao issues something called Access Token. This Access Token must be handed over to a separate server to receive a JWT (JSON Web Token) type Custom Token. This is because Firebase authentication is possible only with the returned Custom Token.

To explain the overall process, first, when the user logs in with Kakao, the Kakao server delivers the authorization code to the REDIRECT URL. After that, the app requests tokens with an authorization code, and Kakao issues tokens. The issued token is transmitted to the Firebase server, and the server sends a signal to Kakao to check if the normal token is correct. If correct, create a JWT-formatted Firebase custom token on the server and respond to the app. The App logs in to Firebase with the custom token received.

2.2.2 Payment Page

Payment page conducts payment validation. The payment agency we are currently using is I'MPORT. When the user calls the payment module, information required for payment is transmitted to the I'MPORT. After payment at I'MPORT, payment information is transmitted to the youth farmer server, and the young-farmer server checks whether the payment information received from frontend and the payment information provided by I'MPORT are the same. If it is correct, store it in Firestore and refund it if it is different.

2.2.3 Other pages

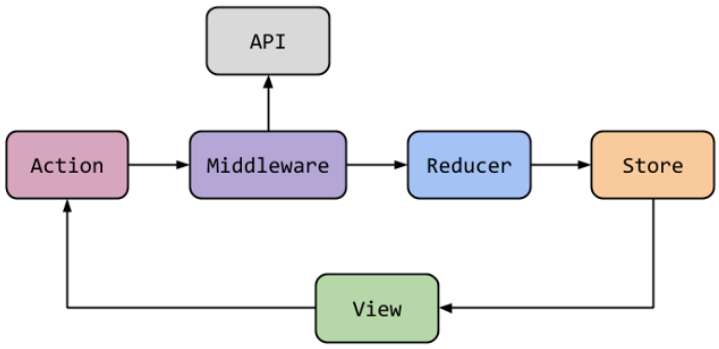

Figure 3: Redux-Saga architecture

Our platform needs a lot of pages, and communication between pages is important. For example, when you press like button for a product, the corresponding like sign should also be displayed on other pages. We used the redux library for this state management.

Meanwhile, Redux had an inconvenience in asynchronous processing. Initially, after communicating with the server using Promise, asynchronous processing was implemented through the Callback function, but this made it difficult to manage the project, such as readability of the code. So, we used a different method. Redux-Saga, a library that helps with asynchronous processing of Redux, was able to increase the readability of code, and as a result, it was able to benefit a lot from project management such as collaboration.

We implemented the core functions of the platform using these technologies. As explained earlier, these libraries have improved a lot in our development, but there are also disadvantages. The library should define an action to implement any one function, define an API for that action, define a Callback function of the API, and define how to convert state in Redux. In other words, a lot of work was required to implement a very simple function in the page.

3 Implementation

3.1 Page UX/UI

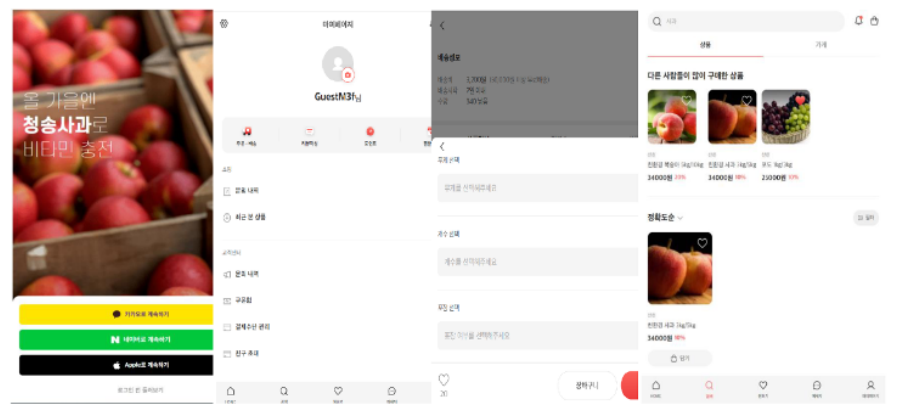

Figure 4: Pages view

This is the actual page of the platform we developed. Overall, it was designed with a neat UX/UI.

As for the design, Figma was used, and collaboration was conducted through Zeplin.

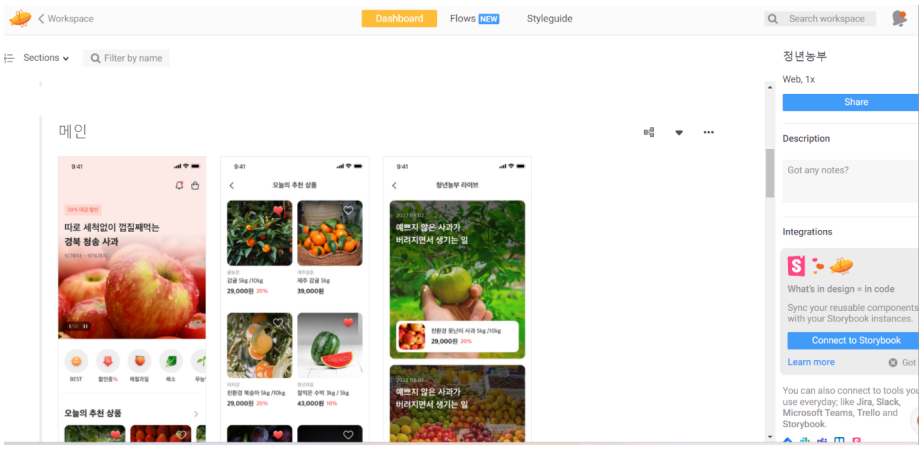

Figure 5: Zeplin

3.2 Frontend

We used React, which is currently the most used library for frontend development. Using Flexbox, the component was designated as a page layout, and the status was managed using Redux. In addition, Redux-Saga was used for asynchronous processing of Redux. It consists of 14 pages, 14 reducer, and 12 Saga scripts.

3.3 Backend

We used Firebase for quick development. The functions provided by Firebase, Cloud Firestore, Firebase Authentication, Cloud storage, Cloud Function, and Hosting were used.

3.3.1 Challenge

There were difficulties in retrieving data within Firebase. In fact, Firebase's official document also introduces the use of additional software for full-text search. However, all such software is charged, so we tried to do something else.

Method 1: Tokenize all the key words. For example, the text called "I like book store"
0: "I like"
1: "b"
2: "bo"
3: "ook"
It is a method of dividing into token and storing it in Firebase as shown in . The difficulty of development is easy, but this has the disadvantage of having to manually tokenize and using a lot of data.

Method 2: Use "+ ₩uf8ff" ₩uf8ff gives the best priority for Firebase searches. Finds all values that begin with query text because of +"₩uf8ff". The problem with this method is that it is found only through the preceding letters. For example, you can only search by setting the search term "I like book" to "I".

The third method is use paid software. We chose Method 1 from the above three methods. Since there are things that need to be developed as a priority, the simplest way to implement them was selected.

3.4 Real Implementation Frontend, Backend

I will explain the process of communication between frontend and backend. An example is a process of receiving product information when a product page is loaded.

```
useEffect(() => {
    dispatch(closeModalAction(modalselector));
    dispatch(GetProductInfo(params.productId));
}, [])
```

Figure 6: Call action

Call GetProductInfoAction through useEffect Hook. Deliver the product Id as a factor.

```
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
    }
}
```

Figure 7: Action detail

The called Action puts the product Id in the payload and sends it to Saga.

It catches the action using the generator function and receives product information from Firebase using ProductID. Depending on the successful loading, the corresponding action is sent to the reducer.

Initialize the Product State by matching the action sent by the API. As such, frontend and backend communicate using node.js library and Firebase.

```
async function getProdcutAPI(payload:any) {
    const product_id = Number(payload);

    const productRef = collection(db, "product");
    const q = query(productRef, where("product_id", "==", product_id), limit(1));
    const fbdata = await getDocs(q);
    const productData = fbdata.docs[0].data();

    const reviewRef = collection(db, "review");
    const q2 = query(reviewRef, where("product_id", "==", product_id));
    const fbdata2 = await getDocs(q2);
    console.log(fbdata2.empty);

    let reviewDataList = [];

    for (const doc of fbdata2.docs) {
```

Figure 8: Get product API

```
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.productInfo,
                storeInfo: action.payload.storeInfo,
            };

        case GET_PRODUCT_FAIL:
            return {
                ...state,
                productInfo: action.payload.productInfo,
                storeInfo: action.payload.storeInfo,
            };

        case GET_PRODUCT_LOADING:
```

Figure 9: Product reducer

4 Evaluation

4.1 Frontend evaluation

What is important in the platform is the loading speed of the page. We made three efforts to improve the loading speed.

1. Change image file extension: png, jpg $\rightarrow$ *webp*
2. Change font: tff $\rightarrow$ $woff$
3. Delete unnecessary typescript

The picture is a comparison of the performance of the Login Page before and after the performance improvement effort through Light House. In general, Light House is divided into scores 0–49 (slow), 50–89 (requires improvement), and 90–100 (good). Our improvement did not increase the score of Light House, but we can see that the performance has improved on detailed indicators such as Fast content paint and Speed index.

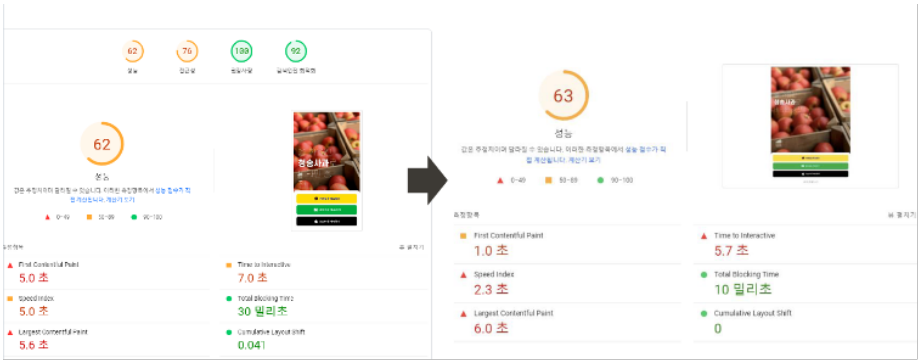

Figure 10: Frontend performance measurement with Light House

4.2 Goal Attainment

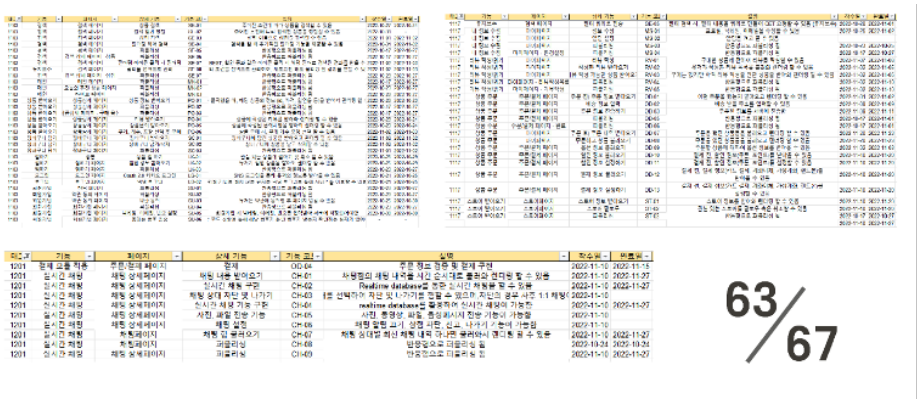

Figure 11: Goal manage

94 percent of the functions intended to be implemented were implemented. In particular, a platform that satisfies the minimum conditions that can be actually traded was developed by implementing a payment function and a chat function.

5 Related work

5.1 React

React is a web framework and is used to create a user interface as one of the JavaScript libraries. One of the advantages of React is that it has a component-

based structure. This makes the code easier to understand and is highly reusable because it is managed on a UI basis. Therefore, even if the size of the application increases, it is easy to maintain and manage it. Based on these advantages, React will be used for front development.

Atomic design pattern Atomic design pattern refers to a design pattern that is easy to develop continuously and gradually based on reusable and rigid design by constructing small components. As we can see in Figure 1, makes it possible to build step by step as like Just as molecules make atoms, which make up something bigger. As frameworks used in recent web front development, such as React, proceed with development on a component-by-component basis, these component-centered design patterns have attracted more attention.

The design design phase forms the atom - molecular - organism - template - page phase. The atomic phase is the component of the smallest unit. Therefore, it must have various states and be designed to be abstract but as inclusive as possible. The molecular phase develops into a more complex unit by weaving atoms. Molecules can have their own properties, and they can use them to create functions in atoms. The organism stage is created by weaving molecules, and when the organism is completed, it has the final appearance of the component. The template step serves to determine the positions and places of the created organism and components, and the template is used to draw and display components on each grid in the page step.

Redux Redux is a JavaScript health management library. This library makes the state predictable and easy to test by making it available only pure functions with Reducer. That is, it is easy to see what action has been taken and how the data has been changed. In addition, an abstract concept called Store is placed so that all states are managed only in the Store. This makes state management easier.

5.2 Figma

It is a web-based UI/UX design and prototyping collaboration tool. Since it is a web-based tool, it does not require installation separately, so it is not limited to the operating system. Taking advantage of the characteristics of the web-based program, it is optimized for collaboration as several people can check the art board at the same time and work in real-time online. UI/UX design is performed using Figma.

5.3 Firebase

Firebase is a mobile application development platform owned by Google. It can also be said to be a collection of tools that can develop, improve, and develop apps. In the original development process, developers must develop and set up various UIs such as analysis, authentication, database, configuration settings, file storage, and push messages. Firebase helps developers reduce the

burden on these UIs and focus on the app's user experience (UX). Developers do not need to build middleware between apps and services, and the client SDK provided by Firebase directly interacts with the backend components. Query was also originally required to be written on both frontend and backend, but with Firebase, the client app only needs to write code to send queries to the database. Originally, the frontend code only calls the API of the back-end, and it is the code of the back-end that actually performs the operation. However, if you use Firebase, you can cross these back-end tasks and immediately turn them over to the client (terminal). This administrator function can be performed through the console window of Firebase.

We will use Firebase with these features for backend side development. With the help of FirebaseUI SDK, we will build an overall backend and implement a payment module using Firebase cloud function.

5.4 Access Token

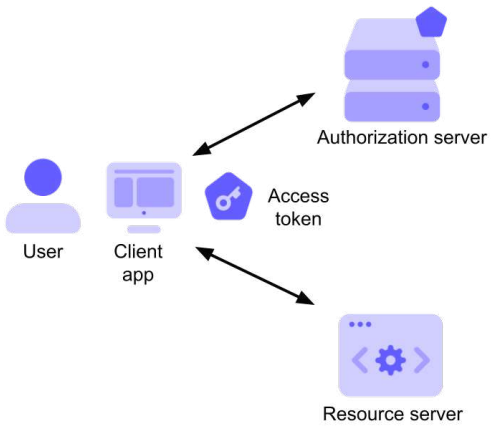

Figure 12: Access token scenario

The above figure makes it possible to know these Access tokens at a glance. If the user is authorized to log in through the Authorization server, Firebase in our App, an access token is issued from the server side. An Access Token is an object that describes the security context of a process. In our WebApp project, this token contains the user's ID and permissions. Our WebApp is currently playing the role of identifying the user when making a payment using the issued access token. In addition, this access token will be used as a certificate when additionally implementing areas requiring user identification and security.

6 Limitations and Discussions

6.1 Synchronization

Let's say you have a product list at the top and a wish list at the bottom. Suppose there is a product A at both the top and bottom. Click the heart-shaped button of a product in the wish list at the bottom to change the status. What will happen. A normal website handles this state change asynchronously, so the heart shape of the product at the top will also change while we can't perceive it with our eyes.

However, for convenience of implementation, we processed some state changes synchronously. The task of notifying the server of the change in Like state was also processed synchronously. Because of this, it takes about 1 to 2 seconds for the state change to be processed and reflected in the previously assumed state of rendering in different components. This is an undesirable phenomenon, and we selected it as a limiting item because we thought it was a matter that should be modified when creating an actual distribution version by building our own back-end server that is not based on Firebase in the future.

6.2 Access Token

Currently we are authenticating users using only Access tokens. The validity period is set at 3 hours, and if 3 hours pass, the user will be logged out. This experience will not be pleasant. However, the more fundamental problem is that tokens are stateless. The story is that the server does not keep state. The server does not have control over the token once issued. Because of this, if the issued token is stolen, the user has no choice but to hand over control of the account to the hacker. The server also has no choice but to wait for the token to expire. In the future, we plan to improve these problems by replacing the vulnerabilities of these access tokens with refresh tokens.

7 Conclusion

We took the theme of the project to create a WebApp-based fruit brokerage platform. A step-by-step design was carried out based on Atomic Design. For UX/UI, the frontend was designed based on Figam, React, Redux, and Redux-Saga, and the backend was designed based on Friebase. The shopping cart, product selection, payment system, etc., which are necessary functions to play a role as an intermediary platform, were set as implementation goals and successfully performed. We have completed the creation of a fruit brokerage platform that can actually be used, which was the original goal, and we have a plan to connect it to actual business through brokerage commissions through demand surveys on the producer side. Therefore, we will do our best to deploy real WebApp beyond the demo version by implementing additional functions such as live commerce through sustainable development and replacing Firebase on the server side with AWS, etc.