Software Design Document for Food Finder

Prepared by Team 2

May 9, 2023

Contents

1 Introduction

This Software Design Document describes the architecture and system design of Food Finder, a mobile and web-based application designed to help users find and filter restaurants through interactive GUI.

2 System Overview

2.1 User Interface

2.1.1 Mobile Application

The application consists of the following pages:

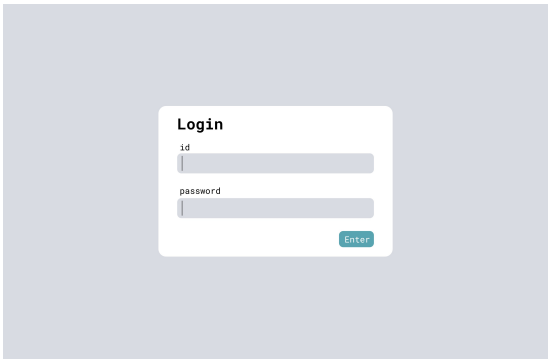
(a) Loading page (b) Search Restaurant (c) Data Map (d) Map View

(e) Map View with Pop-up (f) List View (g) Restaurant Detail

Figure 1: User Interfaces of Mobile Application

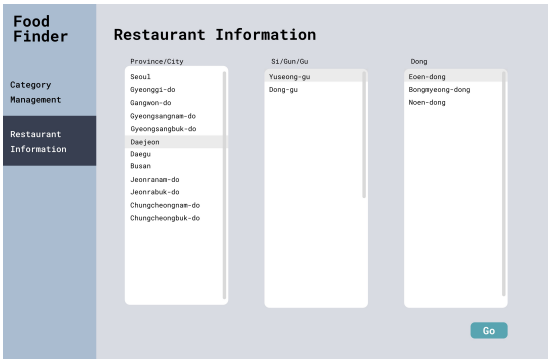
2.1.2 Administrator Portal

The administrator portal consists of the following pages:

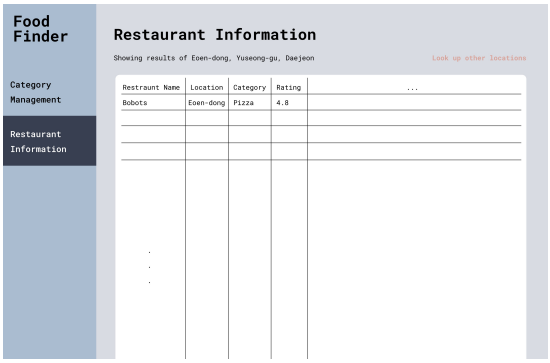


(a) Log-in

(b) Category Management



(c) Select Region



(d) Restaurant Information

Figure 2: User Interfaces of Web Portal

3 Technical Components

3.1 Frameworks and Libraries

3.1.1 Front-end

We currently have two front-ends: a mobile app for app users and a web page for administrator. We use Next.js (version 13) for both front-end. Normally, we cannot use Next.js as mobile app front-end, but we decided to make mobile app by Progressive Web App (PWA). It enables us to use same framework for web and mobile app which can accelerate development speed. We use Kakao Map API to display the restaurants on the map.

For styling, we selected Tailwind CSS (version 3) for CSS framework. It is known to be fast in development.

3.1.2 Back-end

We use Node.js (version 18 LTS) for back-end API server. We will make two servers for mobile app and web page. We use Express.js (version 4) for serving requests and GraphQL for parsing query. We use Prisma (version 4) for javascript ORM and PostgreSQL (version 15) database. Also, we use Meilisearch (version 1) for search engine since it has features such as geosearch, sorting, and filtering. We use NGINX for page routing and static serving front-end mobile app and web pages. For data crawling, we use selectolax.

3.2 Development Tools

We use Github for managing the repository. Since we use github, we use Git as our version control system. We use Github Actions as CI/CD for our repository. We use ESLint for linting, Jest for testing back-end and React Testing Library for front-end. We use Next.js's default bundler which is webpack for easy setup. We will bundle all of the services into Docker and will provide a way to deploy by Docker Compose.

3.3 Platform and Runtime Environment

Since we use docker, we can run on any platform that runs docker. Without docker, we target Linux for one-tier support.

4 Detailed System Design

We organized the functional requirements from software requirement specification in Appendix B.

4.1 Restaurant Information

First we define the data format of the restaurants in JSON, which is compatible with meilisearch. This data format fulfills *REQ* 13.

```
{
  "id": 1,
  "name": "Asobu",
  "introduction": "A best place for KAIST students",
  "address": "14, Eoeun-ro 48beon-gil, Yuseong-gu, Daejeon-si",
  "location": [XX.XX, XX.XX],
  "region": "Eueon-dong",
  "phone": "042-XXX-XXXX",
  "price": 9500,
  "business_hour": {
    "monday": [0, 0, 0, 0, ...],
    ...
  },
  "moods": ["Cozy", "Affordable"],
  "characteristics": ["Sushi", "Donburi"],
  "images": ["path/to/image1", ...],
  "menus": [ {
    "menu A": 9500,
    "menu B": 11000,
    ...
  }, ...],
  "reviews": [ {
    "username": "minji",
    "content": "Good"
  }, ...]
  "rating": 4.6
}
```

Code 1: Example Restaurant Data

`id` is the primary key which uniquely identifies each restaurant. `location` is list of longitude and latitude. `region` is one of the `region` in the dataset of regions. `business_hour` is a dictionary which keys are day of the week. The value is list size of 24, each representing the availability of the restaurant in corresponding hour. `characteristics` is list of category (See 4.1.2). The available list of categories are managed via administrator portal (See Section 4.7).

4.1.1 Region Information

We define the data format of the region in JSON, which is compatible with meilisearch. The *region* is *dong* level, where *displayed* is full name of district starting from city.

```
{
  "id": 1,
  "region": "Gung-dong",
  "location": [XX.XX, XX.XX],
  "displayed": "Daejeon-si, Yuseong-gu, Gung-dong"
}
```

Code 2: Example Region Data

id is the primary key which uniquely identifies each region.

4.1.2 Category Information

We define the data format of the category information. The level of the category is at most 2, *i.e.*, there is no nested *categories* attribute.

```
{
  "name": "Korean",
  "image": "path/to/thumbnail/image",
  "categories": [ {
    "name": "Korean Grill",
    "image": "path/to/thumbnail/image"
  }, ... ]
}
```

Code 3: Example Category Data

4.2 Search Restaurant

The *region search bar* allows users to input a search query and view a list of relevant regions. The *restaurant filter* allow users to apply or delete various filters for their search results. The filters include price, mood, rating, and business hours.

4.2.1 User Interface Components

The main components of the search bar include:

- An input field for users to enter their search keywords.
- A results panel to display the search results. When user input is empty, show the recent *TBD* 1 search history and the button to clear the history.
- When user clicks search result, it generates and shows data map (Section 4.3).

The filter component displays available filters. Users can horizontally swipe to change the filters on the screen. When user clicks on a filter,

- The border of the filter is highlighted.
- The list of options with checkbox is displayed.
- When the user selects one option or multiple options, the selected option or the number of selected options is displayed on the right of the filter, respectively.

The results panel and the filter window are dynamically updated using meilisearch.

4.2.2 Region Search

To fulfill the *REQ* 1, we set the attributes of region demonstrated in 4.1 as below.

By setting *searchableAttributes* as above, the order of results will prioritize matched *dong* over matched higher level of district such as city.

Setting	Attributes
`searchableAttributes`	`region, displayed`

Table 1: Setting of the attributes

4.2.3 Filter Option Generation

In this section, we demonstrate the method for generating filter options. These options depend on the restaurants that match the user's region search. If the user does not input any keywords, display the base options, which are generated beforehand based on all the restaurants in the database. To reduce computational load of querying the entire database, the base options are updated only once a *TBD* 2.

Query the restaurant information dataset `restaurants` matching the user's region search query `region` using faceted search of meilisearch.

```
client.index('restaurants').search(region, {
    facets: ['price', 'rating', 'moods']})
```

Code 4: Faceted Search for Option Generation

The response would return restaurants in queried region with new field: `facteDistribution`.

```
{
  ...
 "facetDistribution":{
    "price":{
      "9500": 6,
      "8000": 4,
      ...
    },
    "rating": {
      "4.7": 1,
      "3.0": 2,
      ...
    },
    "moods": {
      "Cozy": 6,
      "Affordable": 4,
      ...
    }
  }
}
```

Code 5: Example Faceted Search Result

4.2.3.1 Price and Rating

We process the result to generate for options: from zero to first quartile, from first quartile to second quartile, from second to third quartile, and equal or more than third quartile. Options for price are rounded to 1000 KRW, *e.g.*, 8000 KRW if the option is originally 8300 KRW. Options for rating is rounded one decimal place.

4.2.3.2 Mood

We display the top *TBD* 3 moods based on the number of corresponding restaurants.

4.2.3.3 Business Hours

We enable 8 options for the business hours.

- Open now: An option to filter restaurants that are currently open
- 7 options for the day of the week

4.2.4 Test Plan

- Test the filter application and removal process to ensure a smooth user experience.
- Test the filter options to reflect the restaurants in the searched region. It should not display option that is not present in the restaurants of the searched region.

4.3 Data Map

The *data map* visualizes the characteristics of restaurants according to the applied filter. The characteristics are displayed in tree structures, where each node denotes a category of restaurants, per *REQ* 5. The size of the nodes is proportional to the number of restaurants in each category.

4.3.1 User Interface Components

The main components of the data map include:

- A visual representation of the tree structure showing the categories and sub-categories.
  - Each node represent one category.
  - Sub-node representing sub-category is connected to the node representing the category.
  - Node is displayed as the circular image representing the category. The name of the category is displayed below.
- A mini-map to check the selected area.
- A search bar to modify the user's choice.
- A filter function to modify the user's choice.

4.3.2 Data Map Generation

To query the restaurants of each category, we again use faceted search of meilisearch. Given the restaurant information dataset `restaurants` and region search query `region`,

```
client.index('restaurants').search(region, {
    facets: ['characteristics']})
```

Code 6: Faceted Search for Data Map

The response would return restaurants in queried region.

```
{
  "hits": [ {
      "id": 1,
      "name: "Asbou",
      ...
    }, ...],
  "facetDistribution":{
    "characteristics":{
      "Korean Grill":6,
      "Sushi": 4,
      ...
} } }
```

Code 7: Example Faceted Search Result

Next, we process the search result and the category information to construct a list of restaurants by the category. Desired data format is as below.

```
[ {
  "Korean": {
    "count": 3,
    "image": "path/to/thumbnail/image",
    "restaurants": [ ... ],
    "categories": [
        "Korean Grill": {
            "count": 3,
            "image": "path/to/thumbnail/image",
            "restaurants": [ ... ],
        }, ...]
    }
}, ...]
```

Code 8: Example Processed Result

Using the processed result, we construct data map. The *count* becomes the radius of the node, the *image* becomes the image of the node, and the key becomes the text of the node. The *restaurants* attribute is passed as an argument when user clicks the node (See Sections 4.4, 4.5).

4.3.3 Search Bar and Filter Function

To fulfill *REQ* 6 and *REQ* 7, data map includes a search bar and filter function to allow users to modify their choices. When the user inputs a search keyword or applies a filter, the data map will be updated accordingly to display the relevant information using meiliesearch.

4.3.4 Mini-map Function

To fulfill *REQ* 8, data map provides a mini-map that allows users to check the area they choose. When the user hovers over a node, an enlarged map will pop up, showing the selected area.

4.3.5 Test Plan

- Test the data map generation to ensure accurate visualization of the tree structures and categories of restaurants.
- Test the mini-map function to ensure the enlarged map displays the correct area and restaurants.
- Test the search bar and filter functions to ensure accurate and dynamic updates to the data map.
- Test the responsiveness and performance of the data map interface to ensure a smooth user experience.

4.4 Map View of Restaurants

The *map view* displays the location of filtered restaurants on map.

4.4.1 User Interface Components

The map view shows filtered restaurants by pointing pins at their corresponding location. The main components of the map view include:

- A search bar to modify the user's region choice. This fulfills *REQ* 6.
- A button to redirect user back to data map
- A filter component (See Section 4.2) that fulfills *REQ* 7.
- A blue pin to point the location of the filtered restaurants.
- A red pin to point the location of the selected restaurant.
- A button to switch to list view (See Section 4.5).

When user touches blue pin, the pin change to red and the corresponding restaurant information is shown in a pop-up window. Pop-up window include:

- A name of the restaurant
- Whether restaurant is open or closed.
- A rating from 0.0 to 5.0.
- A restaurant's business hour
- The first image of the restaurant

If user clicks the name of the restaurant, user is redirected to restaurant detail (See Section 4.6).

4.4.2 Restaurant Location Map Generation

We use Kakao Map API to display region searched by the user with pins on the map using *location* attribute. We can also set the visual attributes and the click event of the pin using the API.

```
mapOption = {
    center: new kakao.maps.LatLng(XX.XX, XX.XX),
    // location attribute of region
    level: 3 // magnitude of the map
};
var map = new kakao.maps.Map(mapContainer, mapOption);
var markerPosition  = new kakao.maps.LatLng(XX.XX, XX.XX);
// location attribute of restaurant
var marker = new kakao.maps.Marker({ position: markerPosition });
marker.setMap(map);
```

Code 9: Example usage of API

4.4.3 Test Plan

- Integration testing: test the integration of the feature with other components that choosing view type such as list view and map view.
- Test the accuracy of location of pins on the map.

4.5 List View of Restaurants

The *list view* displays the filtered restaurants in a list. This fulfills *REQ* 9, *REQ* 10 and *REQ* 11.

4.5.1 User Interface Components

- A search bar to modify the user's region choice. This fulfills *REQ* 6.
- A vertically scrollable restaurant list. Name, open status, rating, and first image of each restaurant are displayed.
- A button to redirect user back to data map.
- A filter component (See Section 4.2) that fulfills *REQ* 7.
- A button to switch to map view (See Section 4.4)

When the user selects one of the restaurant, it redirects user to the restaurant detail.

4.5.2 Restaurant List Generation

Setting	Attributes
searchableAttributes	*region, price, business_hour, moods, charactersitics, rating*
sortableAttributes	*rating:desc*

Table 2: Setting of the attributes

By setting *sortableAttributes* as above, filtered restaurants can be sorted by rating in descending order. This fulfills *REQ* 10.

4.5.3 Test Plan

- Integration testing: test the integration of the feature with other components that choosing view type such as list view and map view.
- Test if the information is accurate and reflect the selected filters and region.
- Test the accurate and smooth redirection to the restaurant detail.

4.6 Restaurant Detail

The *restaurant detail* shows detailed information about a specific restaurant, such as its name, rating, menu, and reviews. We fulfill *REQ* 11 via search engine.

4.6.1 User Interface Components

The main components of restaurant detail include:

- A button to redirect user back to data map.
- A fixed compartment containing name, rating, open status, introduction and images.
- A menu tab displaying a list of menus in a two-column table.
- A reviews tab displaying a list of reviews which include the name of the person who review and the content.
- A info tab displaying location, business hour, and telephone number. When the user clicks the telephone number, copy the number to the clipboard.

4.7 Web Portal - Administrator

The web portal allows administrators to manage restaurant information and categories. Administrators can log-in to the portal using their credentials, and then add, modify, or delete restaurant information, categories, and region information as needed.

4.7.1 Log-in

The *log-in* page that fulfills *REQ* 12 is consisted of:

- An input for ID
- An input for password
- A log-in button

At successful log-in, a page with a side bar on the left appears. On the bar, the application logo is displayed on the top and 2 tabs appear under it. First tab redirects administrator to *category management* (Section 4.7.3) and the second redirects administrator to *restaurant information* (Section 4.7.4)

4.7.2 Data Crawling

The steps for data crawling is as follows.

1. Confirm the region information and the category information.
2. For each region `r` and category `c`, query the Naver Map with the keyword: `r.region c.name`
3. Crawl the information of the restaurants.
4. If the restaurant of sub-category is not included in the parent category, add it to the parent category.

4.7.3 Category Management

Administrators can manage restaurant categories per *REQ* 14. The page displays multiple lists vertically, each for every first-level category, *e.g.*, Korean and Japanese. Each list is displayed in 2 columns, in alphabetical order. By clicking the *Add Section* button at the top and - button at the right of each list, administrators may add or delete first-level categories, respectively. Additionally, + button at the right of each list allow administrator to add second-level category.

4.7.4 Restaurant Information

4.7.4.1 Select Region

First the administrator can select the region via three levels of scrollable listboxes: province/city level, *si/gun/gu* level, and dong level. Once an administrator clicks on a *dong* level region node, the information of the restaurants in the region is loaded.

4.7.4.2 Restaurant Information

A table of the information of the restaurants in the selected region is loaded from the JSON.

- The table consists of 14 rows, which is the number of different attributes of restaurant information.
- The header row consists of the names of each attribute.
- The × button at the end of each row enables administrator to delete the restaurant information.

Below the table, the button that triggers pop-up on click is displayed. An administrator can add new restaurant information via pop-up per *REQ* 13. An administrator can modify information by clicking the corresponding cell per *REQ* 13. The values in list or dictionary, *e.g.*, `reviews`, are represented in JSON format. Thus, administrator should follow JSON syntax to modify or add the value. Any modification in the table updates the JSON.

4.7.5 Test Plan

Test the mobile application accurately reflects the modifications in web portal after refreshing:

- Adding a new restaurant.
- Removing a restaurant.
- Adding a new category.
- Removing a category.
- Adding a new second level category.
- Removing a second level category.

5 Goals and Milestones

The following milestones are identified to track the progress of the project:

5.1 Milestone 1: Design and Planning

- Date: 4/29 ~ 5/8
- Design the overall software structure for software requirements
- Design user interfaces
- Design system architecture

5.2 Milestone 2: Setup

- Date: 5/9 ~ 5/15
- Setup github repository
- Setup frameworks
- Setup CI/CD

5.3 Milestone 3: Development

- Date: 5/16 ~ 5/22
- Develop web portal for admin
- Develop PWA
- Setup database

5.4 Milestone 4: Testing

- Date: 5/23 ~ 5/29
- Test following the test plans demonstrated in Section 4

5.5 Milestone 5: Deployment

- Date: 5/30 ~ 6/5
- Setup docker
- Deploy to the server

5.6 Milestone 6: Evaluation

- Date: 6/5 ~ 6/12
- Analyze user feedback and system performance

A Glossary

Terms	Definition
Category	The category of food served in the restaurant

B Functional Requirements from SRS

REQ 1: **Delimit an Area**. The system should be able to delimit an area by input of users.
REQ 2: **Database**. DB contains information that can be used for filtering. Depending on filters that user applied, the application queries DB for corresponding filtered information.
REQ 3: **Haptics**. The application takes user actions, such as touch and slide, and reacts accordingly.
REQ 4: **Load data from DB**. The system should be able to get the data associated with the characteristics of restaurants from DB.
REQ 5: **Make Tree Structures**. The system should be able to process the data to make tree structures which each of nodes denotes a category of restaurants. Data maps are created in the following way. The method can be changed later.

1. Divide the zones in proportion to the radius of the node of a category which have a sub-node. Place each node in each zone.
2. Similarly, for sub-nodes, the zones are divided in proportion to the radius. Also, each node is placed in each area. This is repeated until a node with no sub-node appears.
3. Place the main Nodes that do not have sub-nodes in the lower right corner of the screen.
4. Finally, Connect each node appropriately.

The representation of the node can be later changed from a circle to another shape such as a round square.
REQ 6: **Search Bar**. Users should be able to use search bar function to modify their choice.
REQ 7: **Filter**. Users should be able to use filter function to modify their choice.
REQ 8: **Display Location**. The system should display the location of restaurants chosen by data map on the region map.
REQ 9: Operating environment provide **scroll** functions.
REQ 10: **Load restaurant list** from DB.
REQ 11: **Load restaurant information** from DB. If data does not exist, just leave it as blank.
REQ 12: **Administrator Log-In**. In order to administer the system, an administrator should be logged in to the web-portal. The system should make an account for administrator.
REQ 13: **Manage Restaurant Information** In order to have a DB, an administrator should be able to manage the restaurant information. Some of information can be obtained by third-party map service.

- Name: Restaurant name
- Address: Restaurant address
- Phone: Phone number of restaurant
- Price: Average price of dishes
- Rating: A number indicating the rating of a restaurant
- Business Hour: An expression of business hours
- Mood: A word that can describes mood of restaurant
- Characteristics: A list of characteristics of restaurant; Every elements of the list is one of categories.

REQ 14: **Manage Restaurant Category**. In order to have a list of restaurant categories, an administrator should be able to manage the restaurant types.

- Add Category: A new category related to characteristics of a restaurant can be added to the list by an administrator
- Delete Category: A category related to characteristics of a restaurant can be deleted from the list by an administrator.

C To Be Determined List

TBD 1 The number of search history showed will be determined based on the height of the result panel.
TBD 2 The querying frequency will be determined based on the size of the restaurant database and the importance of the base options for better user experience.
TBD 3 The number of options for mood displayed will be determined based on the size of the filter window to maximize the user experience.