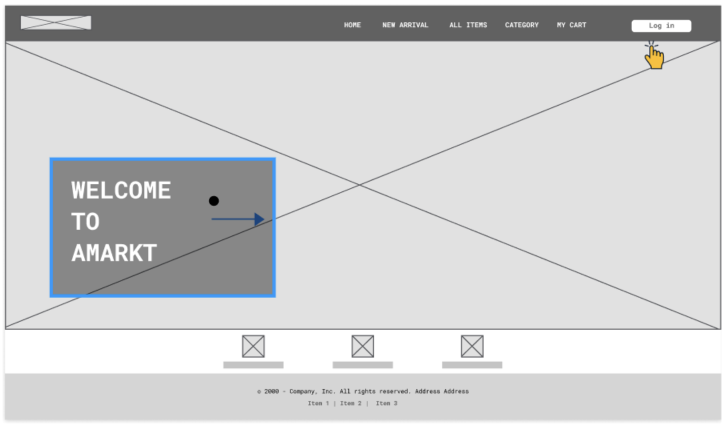
Frameworks and Architecture For the Web
Final Project Report



Group 5

Guanran Tai, Yunjoung Song, Marouan El Haddad, Sandra Baum, Jiashuo Li

27 May 2022

Web design of the client-side application

1. Overall activities on building a dynamic web shop

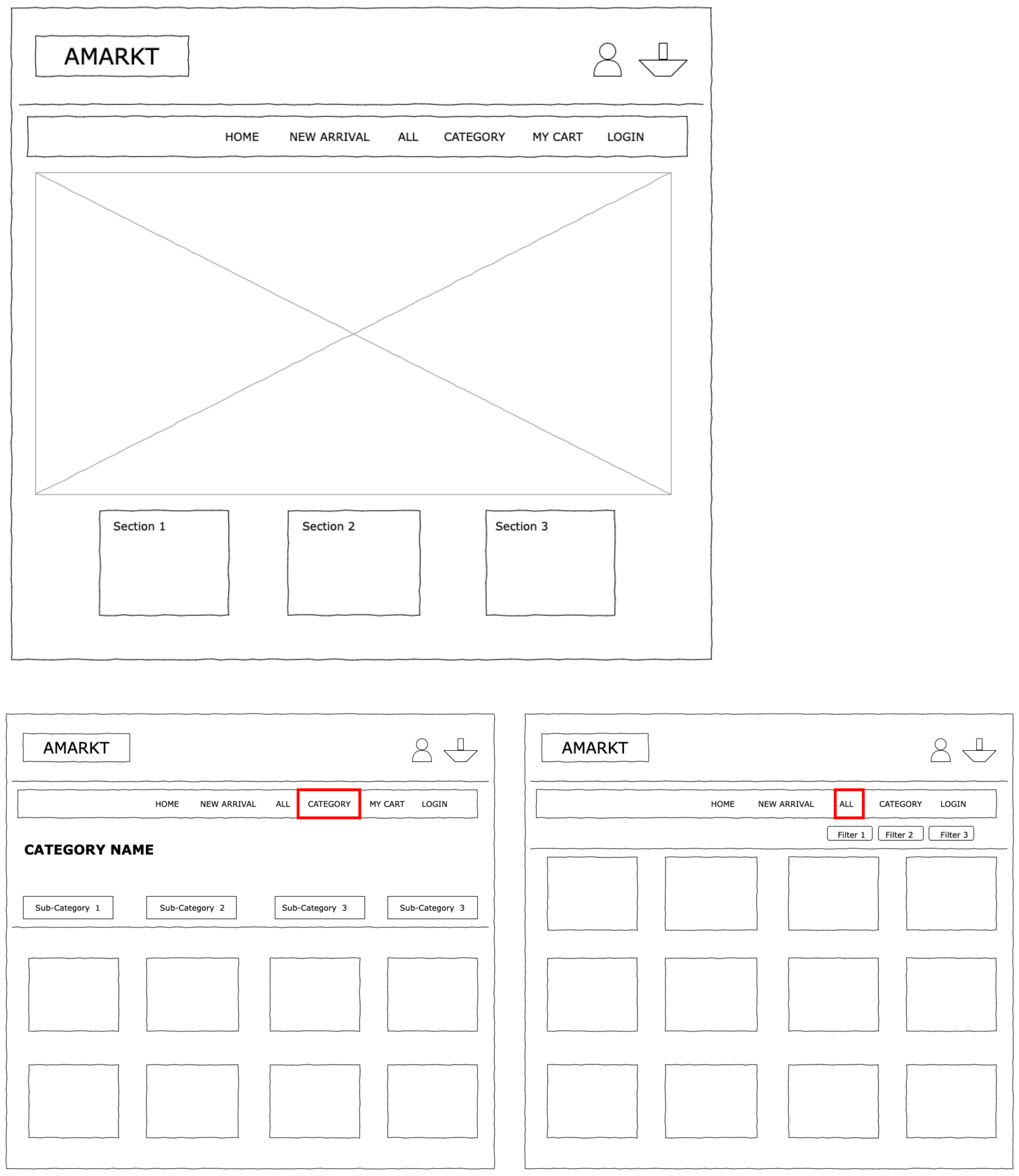
This document shows the development of building an e-commerce website from the Frameworks and Architecture course at IT University of Copenhagen. We will report the initial approach from the beginning stage to the implement stage in order to state the entire developing process of the given tasks.

The first project was intended to train practical hand-on studies in accordance with the theoretical learning from the lecture, Frameworks, and Architecture For the Web. Both practical and theoretical methods were helpful to understand the interactive and dynamic web environment. The following four steps (Site Definition and Planning, Information Architecture, Site Design and Site Structure) provide the process of how the basic website for online shopping is made in practice and represent our logical web architectures for this project activities.

2. Site Definition and Planning

Our group decided to make Asian grocery market website since there is an increase in the awareness of exotic Asian food in the media globally, but the Asian grocery markets are hardly found online in Denmark. After we agreed in consensus about the main concept for the website and shared ideas for the critical features and website design. In addition, we found the fact that the most of Asian online markets need to improve numerous parts regarding usability for customers who are not familiar to that of products. Therefore, we have focused on functional requirements in order to improve user experience such as browsing categories easily, finding products fast using filtering function, and communicating with sellers in case users require some help. After confirming the main concept, we had a brainstorming session to discuss the basic functionality of the website and the flow when users usu our web shop. Thus, the goal of the session was to create initial sketches and descriptions of each page, after that we could create the structure of the website. In addition, we could imagine how users can use and interact while they are shopping. (Figure 1) Eventually, we used the initial wireframe to develop more specific features in order to appeal to potential customers. Based on our desk research, we decided to make our website simpler than other competitors. The structure of screen components were identified to be: navigation bar, main image banner, category division section, and footer. The navigation bar shows the menus as ‘Home’, ‘New’(newly arrived products), ‘All’(all products list), ‘Category’ (product category options), ‘Cart’(user’s shopping cart) and ‘Login’. This is located at the top position on each page, allowing the users to search for desired menu or contents effectively.

(Figure 1. The first wireframe of Group 5)



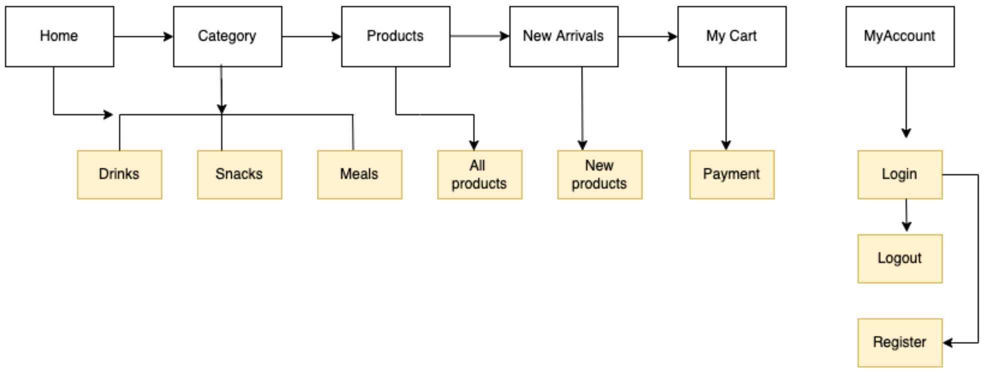
2. Information Architecture

According to the lecture content from Frameworks and architecture for the web course(week4, Website development and design), a website's structure is defined as a structural projection of information architecture. In particular, using a carefully designed category menu helps site visitors to find information easily through clear taxonomy and consistency. From this aspect, we categorized our Asian food items into three categories:

Drink, Snacks, Meals. This is because, these categories would be easier for beginners to choose from. Our website is aimed at retail customers who want to conveniently try Asian food shopping, we wanted to avoid confusing them with too many complex options about Asian food types. Therefore, we planned the three categories as can be seen below (Figure 2). The main categories can be chosen at the navigation bar present at every page and on the homepage under the hero banner, and the sub categories are shown within the category page as filter functions. The sorting buttons were added after the fourth lecture about 'Usability', since we had a valuable lesson to consider for users. The function would be helpful to find subcategory items easily by filtering only what the user wants to see.

Solid and relatable structure is a crucial part of the usability of the website, since user satisfaction is very much linked to how quickly they find information on the site. For these reason, we have chosen 'Hierarchical structure' for our site structural theme rather than 'Sequence structure'. Although our website is currently neither in big scale nor has complex product categories, we decided to implement the 'hierarchical structure' to enable potential future extension of more various categories and new products.

(Figure 2. Sitemap: hierarchical site)



3. Site Design and Structure

We believe that a good website structure is necessary for grouping and categorizing products properly. Since we chose the hierarchical site type, We tried to design carefully each page layout both in terms of User Interface perspective. In addition, the structure is based on user's flow as each Top-down approach – A top-down approach focuses on general categories of the content and then move down to the sub-categories. Therefore, we can logically divide the contents by gradually breaking it up into different categories. This can help inform the users to pleasant shopping activities on the website.

Our website was named 'A-Markt', the name is shortened words to Asian market. We agreed that the name is easy to remember, the site identity. Usually, other Asian supermarket websites have a strange name (They often use the names from their own languages such as Chinese, Thai or Japanese.), it could be difficult to remember for foreign customers. In addition, we considered that the main home page should be used as marketing tools for representing the main concept of our web shop to promote new arrivals or main products. We discussed the entire site design with a consistency, such as the entire layout, color and details, after that we have considered functional non-functional requirements. As a result, we made a mock-up design using Figma based on the initial wireframe.(Figure 3)

Home page – The home page is at the top of the position, It acts as a hub for the visitors of our site. We link to critical information (new arrivals, special offers or popular product images on the home page. From the structure, we will be able to more easily guide users to the most important pages.

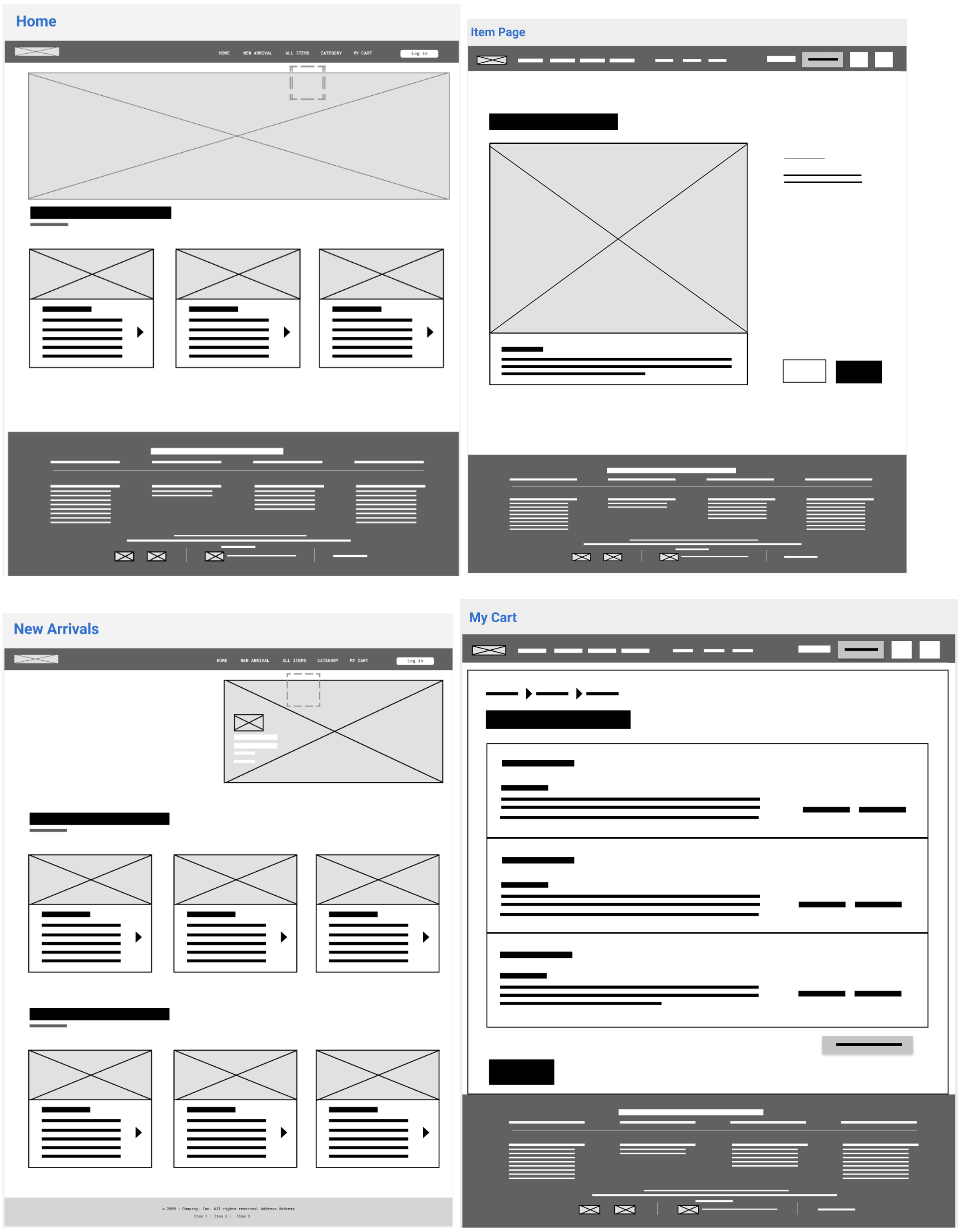
Categories – Categorization is a valuable part of a website's structure. These play a major role in defining a website's structure and flow of users moving. We can help users make decisions faster and easier with good categorization. In addition, users can use categories to reduce the amount of time spent considering a decision. Subcategories provide a structured methodology for browsing and categorizing information in a meaningful manner, especially for websites with complex data.

Item Page – The page includes detailed information of each item. The main function is providing proper information such as images, types, amount, and price for users in order to lead them to put the item into their cart.

New Arrivals/ Products – The individual posts and pages are the basic elements of a website. Depend on user's preference, they can access the pages which are recently updated or all products at one page and sorting orders as they want to see at first with filtering functions on the option bar. This individual pages, we should consider to focus on how to create a meaningful information hierarchy within every page, so the user has less to consider when it comes to consuming content.

My Cart – The page provides detailed information of each item in the user's cart. The function of the cart page is summarizing the items description and show the total price and number of the products which a user put in. It can be a last step to go before the payment.

(Figure 3. The mock-up design of the website)



Design of the RESTful API

The API design table is shown in the appendix.

Software architecture of your developed system

1. overall logical/physical architecture

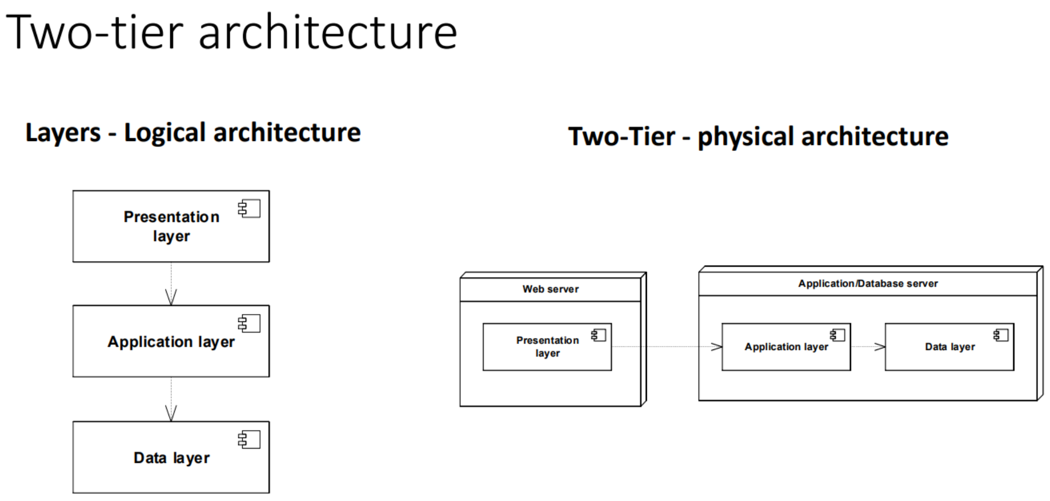


Figure 1: Software architecture: logical/physical

As the picture shown before, we adopted a three-layers-architecture, which includes presentation layer, application layer and data layer. However, in real physical implementation, we use three json files to store the related data, and such json files are together with our service-end-API code, so our online store application uses two-tiers-architecture: web server and Application/Database server.

a. The representation layer

The representation layer is built from frameworks such as react and various components such as React-Bootstrap and Axios. We use React-Bootstrap to build and customize our front-end outlook. React-Bootstrap replaces the Bootstrap JavaScript, each component has been built from scratch as a true React component, without unneeded dependencies like jQuery. Axios is a lightweight HTTP client based on the http service and is similar to the native JavaScript Fetch API. It is promise-based, which gives us the ability to take advantage of JavaScript's async and await for more readable asynchronous code. Based on Axios we could interface with a REST API, making GET, POST and DELETE requests. Besides, we also use the React-Router, a fully-featured client and server-side routing library for building user interfaces. Moreover, we use local storage to store the information like user ID and pass them to the application layer for further operation, for example verification.

React

As is the React developers suggestions, all components were created as functional components and not class components. The data needed for the components was managed by React useState and useEffect hooks.Although having a state increases the complexity of a component, the majority of the components were stateful, as they needed to use data received from the API. Despite that, here were several parent child relations, where data was passed via props from parent to child.These included;

One parent child relationship for managing the cart, where the cart component received the data (ProductNumber, ProductId) about particular users cart content from the API and passed the ProductId over to child element Cartelement, which rendered products with info about price and ProductName. Similar relationship was present in different category components (meals, drinks, snacks), which had two children Addbutton and Removebutton, which recieved ProductId from parent remove or add products via API calls.

A Hierarchy of Components (including React-Router components) can be seen below:

- BrowserRouter
- Routes
- Route
- Menu
  - Link
  - OutLet
- Login
  - Link
- Register
  - Link
- AreLoggedIn
- NewProducts
  - Addbutton
  - Removebutton
  - Link
- ProductList
  - Addbutton
  - Removebutton
- Cart

    - Removebutton

- Conditional
    - Meal
        - Addbutton
        - Removebutton
    - Snack
        - Addbutton
        - Removebutton
    - Drink
        - Addbutton
        - Removebutton
- IndividualProduct
    - Addbutton
    - Removebutton
- Hero
- Categories
    - OutLet
- Footer

React Components

Component	Component Description
Menu	The menu component is the header which contains quick access to different parts of the website. For routing, react router library was used.
Hero	The hero is a non-functional component which only contains images, and is thus only a visual component. The images are not from the API, but are located in the product folder
AreLoggedIn	If the user is not logged in, then the component will show a message when you press the cart button.
Conditional	This is a component which uses conditional rendering, which renders different components depending on which category is pressed on

AddButton	The add button is used in different places on the website, and uses the API endpoint for adding a product to the cart
RemoveButton	The remove button is used in different places on the website, and uses the API endpoint for removing a product from the cart
ProductList	The ProductList is a component used to render all products from the json file in the backend side. It uses the API to get all the products.
IndividualProduct	Whenever a specific product is pressed, this component will show extra information about the product such as the description. This is also a parent component of the addButton and RemoveButton components.
Categories	Categories is a component which uses the API to get all the product categories, and it is featured under the hero banner on the main page. It uses a map function to go through the list of categories, and creates a column for each of the elements in the list.
Meal	The meal component is a stateful component and uses useEffect and useState hooks for state management. It uses map function to go through the list of meal products, and creates a card component for each elements in the list.
Drink	The drink component is a stateful component and uses useEffect and useState hooks for state management. It uses map function to go through the list of drink products, and creates a card component for each elements in the list.
Snack	The snack component is a stateful component and uses useEffect and useState hooks for state management. It uses map function to go through the list of snack products, and creates a card component for each elements in the list.

b. The Application layer

In Application layer, it is built from the RESTful API server. In that server, we use RESTful Model-View-Controller (MVC) pattern. As shown in the picture below, we defined two types of MVC pattern in our implementation: Users' MVC as well as Products' MVC. For each MVC we have designed, the Model component corresponds to all the data-related logic that the APP-user works with. The view component is used for all the http request and http response. And the controller, which contains routers.js to navigate for different requests, and controller.js to implement the interface between Model and View components to process all the incoming requests, manipulate data using the Model component and interact with the Views to return the content of http response.

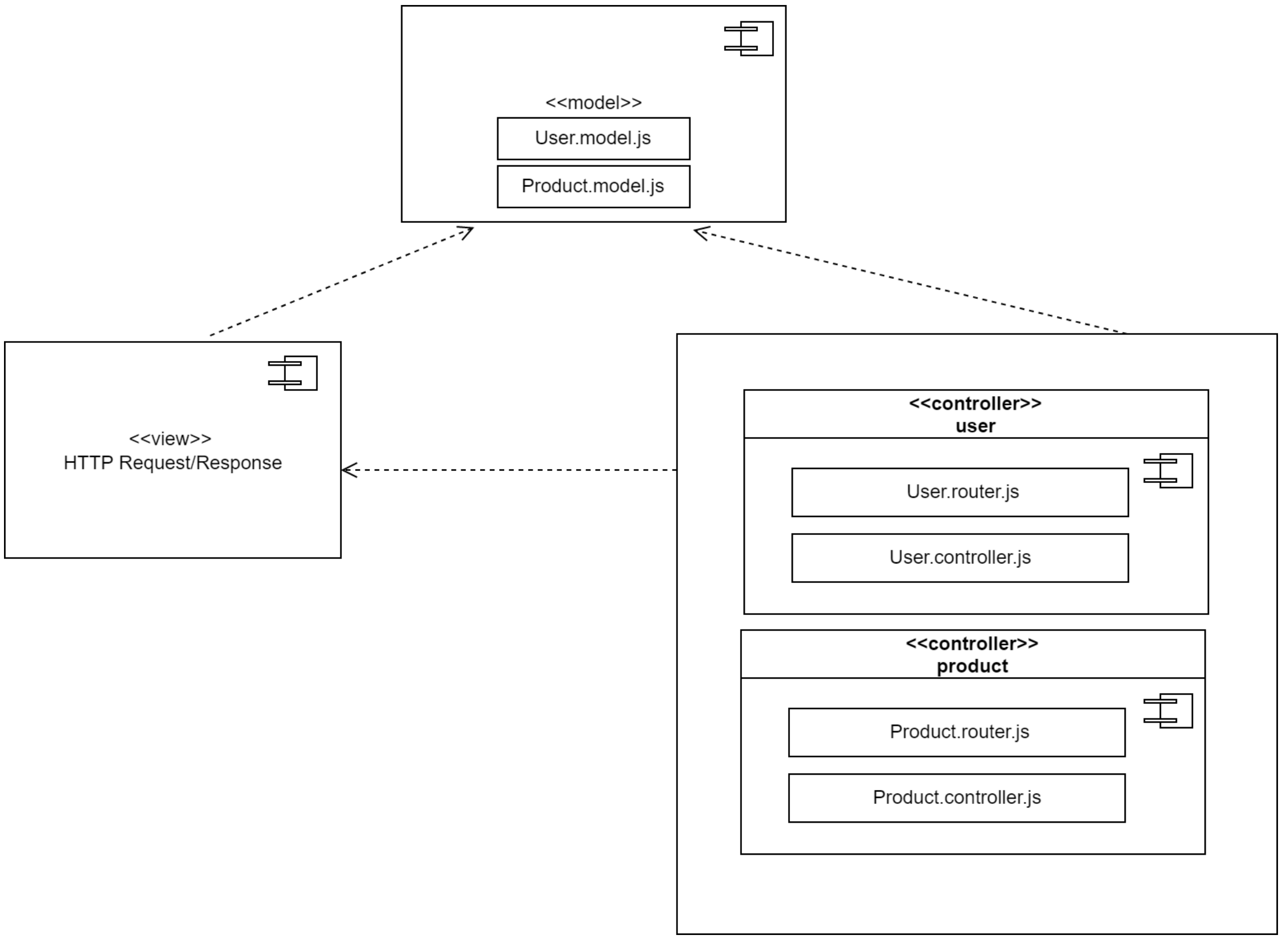


Figure 2: MVC diagram

c. The Data layer

We use Users.json to store the user information, Products.json to store products detail, and Cart.json to store the users' cart information.

d. Advantages and Weakness

We decided to adopt mvc architecture because: 1. It is easy for multiple developers to collaborate and work together. 2. It is easier to debug and update as we have multiple levels properly written in the application. But it does has some disadvantages, like it must have strict rules on methods to avoid problems regarding ambiguous definitions, and to some extent hard to understand the whole architecture.

2. Software styles/patterns

Express standard components

We use express.json. This is a built-in middleware function in Express. It parses incoming requests with JSON payloads.

User/Product router

We define and implement two middlewares named router component(User and Product), to navigate to different controller functions according to different request.

Controller

We define Product and User Controller to manipulate data from product/ User model, and by calling model's functions and getting their results, those controllers will return success or error message.

Model

We use model to implement the data-related logic. In User Model, we design the functions like: registerUser, verify user Information, get/save user information, save cart information, etc. And in product Model, we design functions like: get product By product ID, get product By product category, save product information, etc.

Data store

We use three json file to store different types of data: cart, product and user. In the Cart.json we adopt an array, which stores different customers and their cart information. In the Product.json, we adopt an array of products which include productId, product name, price, description, cactergory and so on. In the user.json we adopt an array of users which include user ID, user password, email, etc.

A Design of the RESTful API

Resource path	POST	GET	PUT	DELETE
/products		Get most important information about all products that are offered.		
/product/:id/		Get all details about a specific product.		
/categories		Get product categories that exist.		
/categories/: categoryId		Get most important information about products for a specific category.		
/user	Post the user information and find if the user registered, if not then register it and return the token.		Put the userId and password, then verify by server and if information is right, then get a token send back from the server side.	
/user/:userId		Get the user information		
/cart/:userId	Add the product to the cart by using productid and productNumber.	Get the information about user's cart	Create the cart if not exist	
/cart/:userId /products/:id				Delete the product from user's cart

B The api description

1. API: /products
   - Path: /products
   - Method: GET
   - Summary: Get most important information about all products that are offered.
   - URL Params: -
   - Body: -
   - Success Response:
     - Code: 200
     - Body Content:

       ```
       [{ "productId": 1, "productName": "Matcha Bubbletea", "price": 34.99,
       "description": "Match Bubbletea is perfect for crunch time.
       It is loaded with antioxidants and sugar. It comes with sweet matcha jelly,
       tea mixture and red beans", "category": "drinks" }, ... ]
       ```
   - Error Response:
     - Code: 404
     - Body Content: 404! This is an invalid URL.
   - Sample Call:

     ```
     let response = await fetch('http://localhost:3000/products',
     {method: 'GET' },
     });
     ```

2. API: /product/:id
   - Path: /product/:id
   - Method: GET
   - Summary: Get all details about a specific product.
   - URL Params: Id: number, required in path
   - Body: -
   - Success Response:
     - Code: 200
     - Body Content:

       ```
       { "productId": 1, "productName": "Matcha Bubbletea", "price": 34.99,
       ↪  "description": "Match Bubbletea is perfect for crunch time. It's loaded
       ↪  with antioxidants and sugar. It comes with sweet matcha jelly, tea
       ↪  mixture and red beans", "category": "drinks" }
       ```
   - Error Response:
     - Code: 404

    - Body Content: product with ID:id doesn't exist

- Sample Call:

```
let id = 4
let response = await fetch(`http://localhost:3000/\${id}/`,
{method: 'GET' },
});
```

3. API: /categories

- Path: /categories
- Method: GET
- Summary: Get product categories that exist.
- URL Params: -
- Body: -
- Success Response:
    - Code: 200
    - Body Content:

```
{
    "drinks": [
        "coffee",
        "tea",
        "juice"
    ],
    "snacks": [
        "chips",
        "candy",
        "cookies"
    ],
    "meals": [
        "noodle",
        "rice",
        "forozen food"
    ]
}
```

- Error Response:
    - Code: 404
    - Body Content: 404! This is an invalid URL.
- Sample Call:

```
let response = await fetch('http://localhost:3000/categories',
{     method: 'GET' },
});
```

4. API: /categories/:categoryId
    - Path: /categories/:categoryId
    - Method: GET
    - Summary: Get most important information about products for a specific category.
    - URL Params: id: number, required in path
    - Body: -
    - Success Response:
        - Code: 200
        - Body Content:

        ```
        [{ "productId": 1, "productName": "Matcha Bubbletea", "price": 34.99,
        ↪  "description": "Match Bubbletea is perfect for crunch time. Its loaded
        ↪  with antioxidants and sugar. It comes with sweet matcha jelly, tea
        ↪  mixture and red beans", "category": "drinks" }...]
        ```
    - Error Response:
        - Code: 404
        - Body Content: product with ID:id doesn't exist
    - Sample Call:

    ```
    let response = await fetch('http://localhost:3000/prodcat/:id',
    {     method: 'GET'},
    });
    ```

5. API: /user
    - Path: /user
    - Method: PUT
    - Summary: Put the userId and password, then verify by server and if information is right, then get a token send back from the server side.
    - URL Params: -
    - Body: Customer JSON data
    - For example:

    ```
    {
        "Id":4,
        "Password": "123"
    }
    ```
    - Success Response:
        - Code: 200
        - Body Content:

        ```
        {
            "message": "Login successful"
        }
        ```

- Error Response:
  - Code: 401
  - Body Content: User not found or password is incorrect
- Sample Call:

```
let response = await fetch("http://localhost:3000/user", {
  method: "PUT",
  headers: {
    "Content-Type": "application/json;charset=utf-8",
  },
  body: JSON.stringify({
    Id: 4,
    Password: "123",
  }),
});
```

6. API: /user/:userId

- Path: /user/:userId
- Method: GET
- Summary: Get the user information.
- URL Params: - userId: number, required in path
- Body: -
- Success Response:
  - Code: 200
  - Body Content:

```
{
    "userInformation": {
        "Id": 4,
        "Email": "Ethan3",
        "FirstName": "cat",
        "LastName": "more",
        "Password": "123"
    }
}
```

- Error Response:
  - Code: 404
  - Body Content: 404! This is an invalid URL.”
- Sample Call:

```
let response = await fetch("http://localhost:3000/user/4", {
  method: "GET"
});
```

7. API: /user

   - Path: /user
   - Method: POST
   - Summary: Post the user information and find if the user registered, if not then register it and return the token.
   - URL Params: -
   - Body: Customer JSON data
   - For example:

     ```
     {
         "Id":78,
         "Password": "123",
         "Email": "taiguanran@gmail.com",
         "FirstName":"Guanran",
         "LastName" :"Tai"
     }
     ```

   - Success Response:
     - Code: 200
     - Body Content:

       ```
       {
           "message": "Registration successful"
       }
       ```

   - Error Response:
     - Code: 401
     - Body Content: User already exists
   - Sample Call:

     ```
     let response = await fetch("http://localhost:3000/user", {
       method: "POST",
       headers: {
         "Content-Type": "application/json;charset=utf-8",
       },
       body: JSON.stringify({ productId: 3, productNumber: 1 }),
     });
     ```

8. API: /cart/:userId(\\d+)

   - Path: /cart/:userId(\\d+)
   - Method: GET
   - Summary: Get the information about user's cart
   - URL Params: userId: number, required in path
   - Body: -

- Success Response:
  - Code: 200
  - Body Content:

```
{
    "cart": [
        {
            "productId": 4,
            "productNumber": 1,
            "price": 15.8
        }
    ]
}
```

- Error Response:
  - Code: 404
  - Body Content: 404! This is an invalid URL.
- Sample Call:

```
let response = await fetch("http://localhost:3000/cart/4", {
  method: "GET",
});
```

9. API: /cart/:userId
   - Path: /cart/:userId
   - Method: PUT
   - Summary: Create the cart if not exist
   - URL Params: userId: number, required in path
   - Body: -
   - Success Response:
     - Code: 200
     - Body Content: Cart created
   - Error Response:
     - Code: 404
     - Body Content: 404! This is an invalid URL.
   - Sample Call:

```
let response = await fetch("http://localhost:3000/cart/4", {
  method: "PUT",
});
```

10. API: /cart/:userId
    - Path: /cart/:userId

- Method: POST
- Summary: Add the product to the cart by using productid and productNumber
- URL Params: userId: number, required in path
- Body: Customer JSON data, for example:

  ```
  {
      "productId":3,
      "productNumber": 1
  }
  ```

- Success Response:
  - Code: 200
  - Body Content: Product added to cart
- Error Response:
  - Code: 404
  - Body Content: 404! This is an invalid URL.
- Sample Call:

  ```
  let response = await fetch("http://localhost:3000/cart/4", {
    method: "POST",
    headers: {
      "Content-Type": "application/json;charset=utf-8"
    },
    body: JSON.stringify({
      productId: 3,
      productNumber: 1,
    }),
  });
  ```

11. API: /cart/:userId/products/:id

- Path: /cart/:userId/products/:id
- Method: DELETE
- Summary: Delete product item from user's cart.
- URL Params: userId: number, Id: number, required in path
- Body: -
- Success Response:
  - Code: 200
  - Body Content: Product deleted from cart
- Error Response:
  - Code: 404
  - Body Content: 404! This is an invalid URL.

- Sample Call:

```
let userId = 4;
let id=3;
let response = await
↪  fetch(`http://localhost:3000/cart/${userId}/produtcs/${id}`, {
  method: "DELETE",
});
```