# Frameworks and Architectures for the Web

# Final Project Document

Group 4: Chuanyu Wei, Xuhui Chang, Yan Zhang, Yuke Gu

# Web Design of the Client-Side Application

### 1. Problem Domain

Our project involves developing a web shop for the fictional brand MuseMove, which specializes in women's yoga outfits and accessories. The main goal of the web shop is to provide an intuitive and user-friendly online shopping experience, allowing users to browse products, view detailed product information, add items to their cart, and complete purchases.

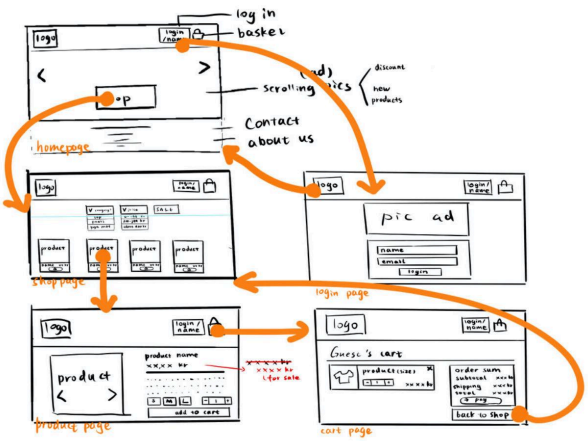
### 2. Information Architecture and Website Design Principles

2.1 Used Taxonomies:

The primary categories used in our taxonomy are:

- Category
  - All Categories
  - Tops
  - Bottoms
  - Yoga Mats
- Price
  - All Prices
  - 0 - 99 kr
  - 100 - 199 kr
  - 200 kr and above

2.2 Hierarchies of Information:




The website follows a hierarchical structure that enables users to easily navigate through the various levels of information. On every page of the website, users can navigate to the home page, the cart page, and log in and out.

- Home Page: The entry point of the website. From this, users can go to the shop page to shop by categories, and go to new and discount product pages.
- Shop Page: Accessible from the home page, where users can shop by categories. And it links to pages of all individual products hierarchically.
- Product Categories: Users can filter products by categories such as Tops, Bottoms, or Yoga Mats, as well as by price range.
- Individual Product Pages: Each product category links to individual product pages, with more detailed product details.

## 2.3 Website Design

### 2.3.1 Layout

The layout of each page is designed to enhance the user experience. Key components include:

- Home Page: Features a large carousel component that displays newly arrived and discount products, along with a prominent "Go Shopping" button.
- Login Page: A simple layout of a banner picture and a login form centered in the middle.
- Shop Page: Displays products in a grid view with filters for categories and price ranges. Each product display includes a picture, name, size options, and price.
- Product Detail Page: Provides detailed information about a specific product, including multiple pictures, product name, price, description, size options, and quantity selection.
- Cart Page: Displays the user's cart contents of each added product with name, image, prize, quantity and quantity controls, and an order summary with responsive total calculation, and a "Continue Shopping" button.

### 2.3.2 Visual Design:

The visual design of the website uses a grayish blue color as the theme color, which matches the style and color palette of our products. Buttons and icons use a darker shade of this color for hover effects. The design follows a minimalist approach, using as few colors as possible and a single font across all pages.

### 2.3.3 Responsiveness

The website is designed to be responsive, ensuring compatibility across various devices and screen sizes. This ensures that users have a consistent and seamless experience whether they access the site from a desktop, tablet, or smartphone.

### 2.3.4 User Interface (UI)

The UI is designed to be intuitive and user-friendly. Key elements include:

- Navigation Bar: Present on every page, providing quick access to the home page, cart page, and login/logout functionality.
- Carousel Component: Used on the homepage to catch users' attention with visual effects.
- Product Grid: On the shop page, displaying products in an organized manner with filtering options.
- Product Details: On the product detail page, providing comprehensive information about each product.
- Cart Summary: On the cart page, summarizing the user's selections and total costs.

# Design of the RESTful API

### 1. Web Service Endpoints

Our RESTful API is designed to handle various operations related to products, users, and shopping carts.

| Endpoint | HTTP Method | Description |
|---|---|---|
| /products | GET | Retrieve all products or filter by category/price |
| /products/categories | GET | Retrieve all categories |
| /products/{productId} | GET | Retrieve a specific product |
| /user | POST | Create a new user |
| /user/{userId}/basket | POST | Create a basket for a user |
| /basket/{basketId} | GET | Retrieve the contents of a basket |
| /basket/{basketId}/items | POST | Add a specific product of a certain quantity to a basket |
| /basket/{basketId}/items/{itemId} | PUT | Update the quantity of a product in the basket |
| /basket/{basketId}/items/{itemId} | DELETE | Remove a product from the basket |

### 2. Explanation of Endpoints

- `/products` (GET): Retrieves information about all products or applies filters by category and/or price range.
- `/products/categories` (GET): Retrieves a list of all product categories.

- `/products/{productId}` (GET): Retrieves detailed information about a specific product.
- `/user` (POST): Creates a new user in the system.
- `/user/{userId}/basket` (POST): Creates a shopping basket for a specific user.
- `/basket/{basketId}` (GET): Retrieves the contents of an existing basket.
- `/basket/{basketId}/items` (POST): Adds a specified product with a certain quantity to the user's shopping basket.
- `/basket/{basketId}/items/{itemId}` (PUT): Updates the quantity of an existing item in the basket.
- `/basket/{basketId}/items/{itemId}` (DELETE): Removes a specific item from the user's basket.

# Software Architecture of the Developed System

## 1. Logical Architecture

The logical architecture of our online shopping system is divided into three main layers: the Presentation Layer, the Application Layer, and the Data Layer.

1.1 Presentation Layer
Components Developed: The client-side web application.
Role: This layer is responsible for the user interface, allowing customers to browse products, view product details, and manage their shopping basket.

1. 2 Application Layer
Components Developed: The server-side RESTful web service.
Role: This layer handles the business logic of the application. It processes requests from the client-side application, manages user sessions, and orchestrates operations on the data.

1.3 Data Layer
Components Developed: Data storage mechanisms.
Role: This layer is responsible for storing and managing product information, categories, and shopping basket contents.
Technology: Data is stored in JSON files or a database. The data format used in HTTP messages is JSON.

## 2. Physical Architecture

2.1 Web Server
Components Deployed: Client-side web application.
Role: Serves the front-end of the application to users. It handles the presentation layer, delivering the React-based single-page application to users' browsers.
Technology: Hosting the React application which interacts with the RESTful API.

2.2 Application Server
Components Deployed: Server-side RESTful web service.
Role: Handles business logic and processes client requests. It interacts with the data layer to retrieve and manipulate data as needed.
Technology: Running Node.js and Express to provide the backend services required

2.3 Cross–Origin Resource Sharing
Express CORS middleware is used to enable HTTP requests from the web server for all origins.

### 3. Client-Side React Application

We use React functional components to build the client-side web application. In index.tsx, the App component is wrapped in AuthProvider, which provides logged-in user's information and basket information. In the App component, we use React Router to manage all navigation and routing within the single-page application, and a Navbar component is rendered on top of all routes to make sure that the Navbar is always displayed on top of the page.

The other main components we have are Home, Login, Cart, Shopping, and ProductDetails.

- Navbar
  The Navbar component handles the user's state and many navigations. It gets the logged-in user's information from the Context API. If the user's information doesn't exist, it can navigate to the Login component; otherwise, it displays the user's first name and a logout button. Utilizing the useHistory and useLocation hook, it makes the user navigate back to the previous page after logging in, and navigate to Home when logging out.

- Home
  The Home component has two sub-components, MyCarousel and AboutUsSection, and a button that navigates to the Shopping component.

- Login
  The Login component has two sub-components, Banner and LoginForm. The LoginForm uses Formik and Yup framework to validate filled-in information per field and on submit. When a user successfully submits the form, RESTful API is called to create a new user and then create a basket for the user. Then, user's information and basket ID are passed to Context API to be used in other components.

- Shopping
  It has a Shopping component which fetches and displays a list of products, with category and price filtering, and two subcomponents the Filters and the ProductCard. It has the useState hook which manages state for products, category filter, price filter, and error messages, and the useEffect hook which fetches products from the server when category or price filters change.
    - The Filters has a FliterBar component with two sub components: CategoryFilter and PriceFilter. They display categories fetched from the server in a dropdown. The CategoryFilter calls 'onCategoryChange' when a category is selected, while the PriceFilter calls 'onPriceChange'. It uses the useState hook to manage state within the

components (categories, prices, and error messages), and the useEffect hook to fetch data from the server when the component mounts and handles side effects.
    - The ProductCard component displays product details such as the product image, name, price, and available sizes. It also handles adding the product to the shopping cart when the "Add to Cart" button is clicked. It includes a useAuth hook which retrieves the basketId for the current user.

- ProductDetails: Detailed product information displayed using React components.
    - It uses the useParams hook from react-router-dom to get the product ID from the URL and fetches product data from a RESTful API. The component manages loading, error handling, quantity selection, and size selection using useState.
    - When the component mounts, it fetches product details based on the product ID and updates the state. If an error occurs, an error message is displayed.Users can select a product size (if available) and quantity before adding the product to their cart. The useAuth hook retrieves the basket ID from the Context API. The handleAddToCart function creates an item object with the product details and adds it to the user's basket via a POST request.

- Cart
  1) Once entering the cart webpage, if a user is logged in, the user's basket data is fetched from the UserContext API. 2) When the user adds an item to the cart, the product information is fetched using GET and the item is displayed by being POSTed in the basket. The user can adjust the quantity of the added item or delete it via a PUT and DELETE request. 3) The calculateSubtotal function computes and updates the total price of the current items in the current basket to be shown in the Subtotal part, and to be used to decide whether there is shipping fee (30 kr if Subtotal is below 350 kr) and computed together with the shipping fee as the Total part the Order Summary. 4) Once the user is finished shopping, the ContinueShopping function navigates the user back to the shopping page.

React Patterns: Use of Context API for state management and functional components with hooks (useState, useEffect, useContext, useParams)..

Third-Party Libraries: React-Bootstrap for UI components, Formik and Yup for form validation, React Router for navigation.

**4. RESTful API Application**

- Express Server: Manages API endpoints for handling product data, user data, and shopping cart operations.
- Express Patterns: Middleware for handling requests, routers for modularizing routes, and controllers for separating business logic.
- Data storage: Product information, user data, and cart details are stored in JSON files.

# Appendix: API Specifitions

### List of resources

| Resource path | POST | GET | PUT | DELETE |
|---|---|---|---|---|
| /products | | Get info about all products or filter by category/price | | |
| /products/categories | | Retrieve all categories | | |
| /products/{productId} | | Retrieve a specific product | | |
| /user | create user | | | |
| /user/{userId}/basket | Create basket for a user | | | |
| /basket/{basketId} | | Get basket content | | |
| /basket/{basketId}/items | Adds a specific product of certain quantity to a basket | | | |
| /basket/{basketId}/items/{itemId} | | | Update amount of a single product | Remove a single product from basket |

# Path: /products

**Method:** GET

**Summary:** Retrieve information about all products or apply filters by category and/or price range.

**URL Params:**

Optional query parameters for filtering:

category (string): Name of the product category to filter by.

minPrice (number): Minimum price to filter by.

minPrice (number): Minimum price to filter by.

**Body:** -

**Success Response:**

Code: 200 OK

Body Content: Array of products with most important details.


```
[
{
    "productId": "top04",
    "productName": "Seamless High-Elasticity Long Sleeve Sports Top",
    "price": 329,
    "currency": "kr",
    "category": "Top",
    "sizes": [
      "S",
```

```
      "M",
      "L"
    ],
    "image": "./image/top/4/638440469900730000 (2).jpeg",
    "description": "Seamless High-Elasticity Long Sleeve Sports Top",
    "url": "product.top04.html"
  },
  // More products...
]
```

**Error Response:**

Code: 404 Not Found

Body Content:

```
{ "error": "No products found" }
```

**Sample Call:**

```
//all products
async function fetchProducts() {
    try {
        let response = await fetch('/products', {
            method: 'GET',
            headers: {
                'Content-Type': 'application/json;charset=utf-8'
            }
        });

        if (!response.ok) {
            // Assuming the server responds with a JSON containing an error field in case
of errors
            let errorData = await response.json();
            throw new Error(errorData.error);
        }

        let data = await response.json();
        console.log(data);
    } catch (error) {
        console.error('Error:', error);
    }
}


//filtered products
async function fetchFilteredProducts() {
    try {
        let queryParameters = new URLSearchParams();

        if (categoryFilter) queryParameters.append('category', categoryFilter);
```

```
        if (minPriceFilter) queryParameters.append('minPrice', minPriceFilter);
        if (maxPriceFilter) queryParameters.append('maxPrice', maxPriceFilter);

        let response = await fetch(`/products?${queryParameters.toString()}`, {
            method: 'GET',
            headers: {
                'Content-Type': 'application/json;charset=utf-8'
            }
        });

        if (!response.ok) {
            // Assuming the server responds with a JSON containing an error field in case
of errors
            let errorData = await response.json();
            throw new Error(errorData.error || 'Network response was not ok');
        }

        let data = await response.json();
        console.log(data);
    } catch (error) {
        console.error('Error:', error);
    }
}
```

# Path: /products/categories

**Method:** GET

**Summary:** Get all categories.

**URL Params:** -

**Body:** -

**Success Response:**

Code: 200 OK

Body Content: Products JSON data

```
{
  "categories": [
    {
      "name": "All Categories",
      "value": "all"
    },
    {
      "name": "Top",
      "value": "Top"
```

```
    },
    {
      "name": "Bottom",
      "value": "Bottom"
    },
    {
      "name": "Yoga Mat",
      "value": "Yoga Mat"
    }
  ],
  "prices": [
    {
      "name": "All Prices",
      "value": "all"
    },
    {
      "name": "0 - 99kr",
      "value": "0-99"
    },
    {
      "name": "100 - 199kr",
      "value": "100-199"
    },
    {
      "name": "> 199kr",
      "value": "199-"
    }
  ]
}
```


**Error Response:**

Code: 500 Internal Server Error

Body Content:


```
 {"error": "An unexpected error occurred"}
```


**Sample Call:**

```
let response = await fetch('/products/categories', {
    method: 'GET'
});
let data = await response.json();
if (!response.ok) {
    throw new Error(data.error);
}
console.log(data);
```

# Path: /products/{productId}

**Method:** GET

**Summary:** Get all details about a specific product.

**URL Params:** productId: string, required in path

**Body:** -

**Success Response:**

Code: 200 OK

Body Content:


```
{
  "productId": "top01",
  "productName": "V-Shaped Droplet Sports Bra Low Intensity CityWalk",
  "price": 120,
  "currency": "kr",
  "category": "Top",
  "sizes": [
    "S",
    "M",
    "L"
  ],
  "image": "./image/top/1/638410919194070000.jpg",
  "description": "V-Shaped Droplet Sports Bra Low Intensity CityWalk",
  "url": "product.top01.html"
}
```


**Error Response:**

Code: 404 NOT FOUND

Body Content:


```
{"error": "Product with id p1 does not exist"}
```


**Sample Call:**

```
fetch(`/products/${productId}`, {
    method: 'GET',
    headers: {
        'Content-Type': 'application/json;charset=utf-8'
    }
})
.then(response => {
    if (!response.ok) {
```

```
            throw new Error('Product not found.');
        }
        return response.json();
    })
    .then(productDetails => {
        console.log(productDetails);
    })
    .catch(error => console.error('Error:', error));
```

# Path: /user

**Method:** POST

**Summary:** Create a new user.

**URL Params:** None

**Body: User JSON data**

{"firstName": "Jane", "lastName:"Smith", "email": "jane@example.com"}

**Success Response:**

Code: 201 CREATED

Body Content: For example:

```
{
 "firstName": "Jane",
 "lastName": "Smith",
 "email": "jane@example.com",
 "userId": "6ed23fe8-9e16-45fc-bf51-c386ce7581fe",
 "message": "User created successfully."
}
```

**Error Response:**

Code: 400 BAD REQUEST

Body Content:

```
{"error": "Email address is already in use."}
```

**Sample Call:**

```
let user = {"userName": "Jane", "email": "jane@example.com"};
let response = await fetch('/user', {
    method: 'POST',
    headers: {'Content-Type': 'application/json;charset=utf-8'},
    body: JSON.stringify(user)
}).then(response => {
    if (!response.ok) {
        throw new Error('An error occurred.');
    }
    return response.json();
}).then(data => {
    console.log(data); // Prints the success response including the userId
}).catch(error => console.error('Error:', error));
```

# Path: /user/{userId}/basket

**Method**: POST

**Summary**: Creates a shopping basket for a specific user

**URL Params**:

userId: the unique identifier of the user

**Body**:

```
{"userId": "uniqueUserId"}
```

**Success response**:

Code: 201 Created

Body content: For example:

```
{“basketId”: "basketID", "userId": "uniqueUserId",  "products":[]}
```

**Error Response**:

Code: 404 Not Found

Body Content:

```
 {error: "User not found."}
```

**Sample Call**:

```
fetch('/user/{userId}/basket, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({"userId": "uniqueUserId",  "products":[]}
),
})
.then(response => response.json())
.then(data => console.log(data))
.catch((error) => console.error('Error:', error));
```

# Path: /basket/{basketId}

**Method**: GET

**Summary**: Retrieves the contents of an existing basket

**URL Params**:

basketId: the unique identifier of a basket

**Body**: -

**Success response**:

Code: 200 OK

Body content:

```
{“basketId”: "basketID", "userId": "uniqueUserID",
"products": [{"productId": "productID", "name": "ProductName", "quantity": 1, "price":
19.99}, ...]}
```

**Error Response**:

Code: 404 Not Found

Body Content**:**

```
 {error: "Basket not found."}
```

**Sample Call**:

```
fetch('/basket/{basketId}', {
  method: 'GET',
})
.then(response => response.json())  \\ parse the JSON response from the server
.then(data => console.log(data))  \\ log this data to the console
.catch((error) => console.error('Error:', error));  \\ catch and log the error
```

# Path: /basket/{basketId}/items

**Method**: POST

**Summary**: Adds a specified product of certain quantity to the user's shopping basket.

**URL Params**:

basketId: the unique identifier of the basket

**Body**: {"productId": "productID", "name": "ProductName", "quantity": 1}

**Success response**:

Code: 201 Created

Body Content:

```
{“itemId”: "itemID", "productId": "productID", "name": "ProductName", "quantity": 1}
```

**Error Response**:

Code: 404 Not Found

Body Content:

```
 {error: "Basket not found."}
```

**Sample Call**:

```
fetch('/basket/{basketId}/items, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({productId: "productID", quantity: 1}),
})
.then(response => response.json())
.then(data => console.log(data))
.catch((error) => console.error('Error:', error));
```

# Path: /basket/{basketId}/items/{itemId}

**Method**: PUT
**Summary**: update the quantity of an existing item in the basket

**URL params**:

basketId: The unique identifier of the basket.

itemId: The unique identifier of the item to update.

**Body**: {quantity: 3}

**Success Response**:

Code: 200 OK

Body Content:  (Returns the updated cart item)

```
{
  "itemId": "item456",
  "productId": "p2",
  "quantity": 3
}
```

**Error Response**:

Code: 404 Not Found

Body Content:

```
{error: "basket not found"}
//or
{error: "item not found"}
```

**Sample Call**:

```
fetch('/basket/basket123/items/item456', {
  method: 'PUT',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({ quantity: 3 })
})
.then(response => response.json())
.then(data => console.log(data));
```

# Path: /basket/{basketId}/items/{itemId}

**Method**: DELETE

**Summary**: Removes a specific item from a user's basket.

**URL Params:**

basketId: The unique identifier of the basket.

itemId: The unique identifier of the item to be removed.

**Body**: -

**Success Response**:

Code: 204 NO CONTENT

**Error Response**:

Code: 404 Not Found

Body Content:

```
{error: "basket not found"}
//or
{error: "item not found"}
```

**Sample Call:**

```
fetch('/basket/basket123/items/item456', {
  method: 'DELETE'
})
.then(response => response.json())
.then(data => console.log(data));
```