# High-Level Design for the Micro-Frontend Insurance App

**1. Overall Architecture:**

The system follows a **Micro-Frontend Architecture** where three separate apps work together as individual units but share data and user flows through a container application. The structure allows each part of the system (container, user details, and insurance details) to operate independently and asynchronously.

The three apps are:

1. **Container App (Shell)**: Manages authentication, routing, and hosting micro-frontends (User and Insurance).
2. **Users App (Micro-Frontend)**: Displays user profile details and eligibility for insurance offers.
3. **Insurance App (Micro-Frontend)**: Manages and displays insurance details and payment functionality.

The **Container App** is the entry point and controls which micro-frontend (User or Insurance) gets loaded based on user interaction. Data is passed between these apps using **local storage** and **event-driven communication**.

[GitHub - pankajexpressian/nagp\_client\_side\_architecture](https://github.com/pankajexpressian/nagp_client_side_architecture/tree/main)

A diagram of a container

Description automatically generated

**2. Key Components and Flow:**

**2.1 Container App:**

* **Responsibilities:**
  + **Authentication**: Handles user login by validating the email format and checking if the user exists.
  + **Event Dispatching**: Sends events to micro-frontend apps (User and Insurance) to display messages.
  + **Data Initialization**: Seeds user and insurance data into local storage on startup.
  + **Routing**: Dynamically loads the respective micro-frontend apps (Users App, Insurance App) based on user actions.
  + **UI Layout**: Includes a sidebar navigation with links/buttons for Profile, Insurance, and Logout.
  + **State Management**: Passes user and insurance data to the micro-frontends while loading them from the container app.
* **Key Functionalities:**
  + **Login**: Validates the email and signs in the user.
  + **Sidebar**: Provides options to load the User and Insurance micro-frontends.
  + **User Interaction**: Redirects based on user’s choice and triggers events like "Send Message".
  + **Logout**: Clears data and redirects back to login screen.

**2.2 Users App (Micro-Frontend):**

* **Responsibilities:**
  + **Profile Display**: Displays user profile information fetched from local storage.
  + **Eligibility Check**: Uses a background worker to process user data (age, income, gender) and determine insurance eligibility.
  + **Event Listener**: Listens to messages/events from the Container App and displays relevant notifications or alerts to the user.
* **Key Functionalities:**
  + **Data Fetching**: Retrieves user details from local storage and displays it.
  + **Offer Calculation**: Background worker runs eligibility checks and calculates insurance offers.
  + **Message Handling**: Listens for any events (such as "Send Message") from the Container App and shows relevant messages.

**2.3 Insurance App (Micro-Frontend):**

* **Responsibilities:**
  + **Insurance Details**: Fetches and displays user’s insurance details from local storage.
  + **Payment Gateway**: Displays payment options and allows the user to pay the premium.
  + **Fun Facts API**: Simulates API calls to show random fun facts using a background worker.
  + **Event Listener**: Listens to events from the Container App for message display.
* **Key Functionalities:**
  + **Display Insurance Data**: Shows existing insurance details pulled from local storage.
  + **Premium Payment**: Allows the user to view and interact with payment options.
  + **API Simulation**: Background worker fetches fun facts periodically to simulate API interaction.
  + **Message Handling**: Listens for events from the Container App to show notifications.

**3. Data Flow & Communication:**

* **Data Storage**: Local storage is used for storing user and insurance data. The Container App seeds the data initially and passes it to the micro-frontends.
* **Event Handling**: The Container App can trigger events (e.g., "Send Message") that the User and Insurance micro-frontends listen for. When the event is received, the micro-frontends update their UI accordingly.
* **Local Storage Usage**:
  + **Container App** seeds user/insurance data.
  + **Users App** and **Insurance App** pull relevant data from local storage.
* **Background Workers**:
  + The **Users App** runs a worker to calculate the best insurance offers based on user data (age, income, etc.).
  + The **Insurance App** runs a worker to simulate periodic API calls that display fun facts.

**4. Technical Stack:**

* **Frontend Framework**: React
* **CSS Preprocessor**: SCSS
* **Bundling**: Webpack for packaging and bundling all micro-frontends.
* **Communication**: Local storage for sharing data and custom event listeners for inter-frontend communication.
* **Background Workers**: For asynchronous tasks like calculating insurance offers and fetching fun facts.

**5. Deployment Flow:**

1. **Local Development**: Each micro-frontend (Container, Users, and Insurance) runs on separate ports (3000, 3001, 3002).
   * Container app runs on **localhost:3000**.
   * Users’ app runs on **localhost:3001**.
   * Insurance app runs on **localhost:3002**.
2. **Launch Flow**:
   * First, the **Container App** is loaded (at port 3000).
   * On successful login, it loads the **Users App** (at port 3001) or the **Insurance App** (at port 3002), depending on user actions.
3. **Data Flow**:
   * The Container App seeds and shares data with the micro-frontends via local storage.
   * Micro-frontends (Users and Insurance) retrieve this data and display the necessary information to the user.
   * Background workers handle tasks like eligibility checks and fun fact fetching.

**6. Security Considerations:**

* **Cross-Site Scripting (XSS)**: The Container App ensures that email input is sanitized to prevent XSS attacks. Input validation is done on the login page to prevent malicious scripts.
* **Data Integrity**: Data stored in local storage is isolated per app and only accessible through the container or micro-frontends. In real app No sensitive data will be stored without proper protection.

**7. User Interaction Flow:**

1. **Login**:
   * User enters email on the login screen of the Container App.
   * Container App validates the email format, checks if the user exists, and redirects to the landing page (either User or Insurance view).
2. **User Profile**:
   * On clicking "View Profile", the **Users App** is loaded, displaying user details and insurance eligibility.
3. **Insurance Details**:
   * On clicking "View Insurance Details", the **Insurance App** is loaded, displaying insurance details and premium payment options.
4. **Message Handling**:
   * The **Container App** dispatches events (e.g., "Send Message") that are listened to by the **Users App** and **Insurance App** for dynamic updates.

## Requirement Checklist:

|  |  |
| --- | --- |
| Requirement | Status |
| Build container app | Done |
| Build One MFE for insurance details and premium payment | Done |
| Build Second MFE for user details | Done |
| Load both MFEs on container app | Done |
| Share data between containers and MFEs | * Data being shared using local storage. * Container app is passing data to MFEs dynamically. * Container app sending events to MFEs. |
| Store and Retrieve data from local storage | Done |
| Add worker to one of the MEFs to do anything | Done |
| Using webpack for bundling | Done |
| Use any CSS pre-processor | Used SCSS loader |
| Implement any one OWASP top 10 | Implemented XSS (prevented cross site scripting) |