# 1. Introduction

This project report covers the development of a file-based food ordering web service and client interface application, designed to provide a user-friendly experience for customers, restaurant managers, and admins. The project uses a RESTful API to handle login, registration, and menu-viewing functionalities. Future expansions include user role definitions, authentication on the Web Client, and complete WeClient Support for all users roles.

# 2. Business Impact

Using web services for a food ordering platform brings significant advantages:

* **Enhanced User Accessibility**: The APIs provide seamless access to users, allowing customers to search for menus quickly and securely.
* **Increased Efficiency**: Automating login and menu display with token-based authentication reduces manual tasks for restaurant managers and admins.
* **Scalability**: The separation of frontend and backend services allows for independent scaling as needed.
* **Data Security**: Passwords are encrypted before storage, and tokens are used for authenticated sessions, protecting user data.

# 3. Design and Architecture

## 3.1 Web Service Design Architecture

* **Framework**: The backend web service is implemented as a Maven project using RESTful architecture. The endpoints include AuthController for authentication (login and registration) and MenuController for retrieving and filtering menu items.
* **API Endpoints**:
  + **Login (POST)**: Authenticates the user and returns a session token for secure, seamless access.
  + **Register (POST)**: Registers new users, storing their details in a tab-separated file, with encrypted passwords.
  + **View Menus (GET)**: Retrieves a list of menu items, with the ability to filter based on keywords, providing dynamic menu searching.
  + **Place Orders (POST)**: Users can place orders from the Web Client.
  + **View Orders (GET)**: Users can view past orders from the Web Client.

## 3.2 MVC Pattern

* **View**: A separate Maven project serves as the frontend, developed using **Java Swing** for GUI components.
* **View**: A React Application that serves as a complete stand alone for All user roles.
* **Model**: Data models (User, AuthDetails, Menu, Order) encapsulate core application data.
* **Controller**: Manages data flow between the frontend and backend, with AuthController, MenuController, OrderController coordinating the primary application actions.

## 4. Data Structures Used

* **HashMap (Singleton Pattern)**: To ensure efficient data management, a singleton HashMap instance is used to store and retrieve user and menu information. This approach centralizes data handling and allows for easy maintenance of user and menu records.
* **File Storage**: Registered users and menus are saved in a tab-separated file, providing a lightweight, database-free storage solution for rapid access and updates.
* **Token Generation**: Tokens for login are generated using a simple hashing method, which allows secure session handling without the need for a formal database.

## 5. Security Considerations

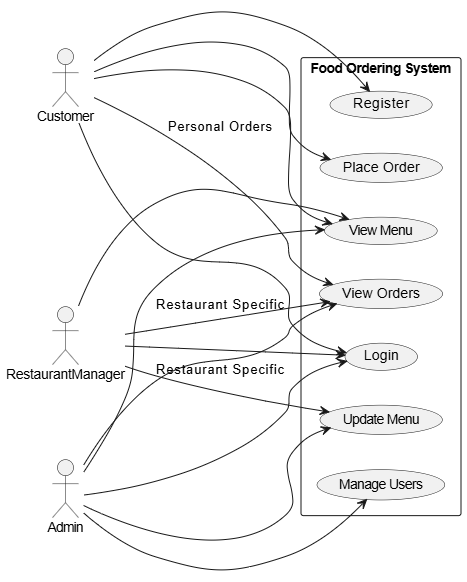
* **Password Encryption**: User passwords are encrypted upon registration and stored securely, reducing the risk of unauthorized access.
* **Token-Based Authentication**: A hashing method is used to generate tokens, allowing secure session management and reducing repeated logins.
* **POST Requests for Sensitive Data**: Login and registration requests use POST to secure user credentials in transit.

## 6. Future Expansion

* **User Types**: Expanding roles for administrators, managers, and regular users, each with varying access and management capabilities.
* **Update Web Client**: Migrate existing functionalities from Java Client to React Application.
* **Add SOAP endpoint**: Implement SOAP end points in the application.

# 7. Use Case Diagram

Not all functionalities are implemented. This is a simple use case Diagram



# 8. Screenshots for use case scenarios

## Java client loginEdited.pngregisterEdited.pngmenuEdited.png

## Web ClientCustomerViewAndCart.PNGCustomerViewPastOrders.PNG

# 9. Additional Functionalities for next deliverables

1. Add more filters in the menu page, example: filter by price, category, or restaurants\
2. Separate view for administrators, and Restaurant managers where they can edit menu, see user details etc…
3. (Optional) Add feature to compare prices with third-party services, different apps