

# FINAL PROJECT REPORT

SEMESTER 1, ACADEMIC YEAR: 2024-2025

CT313H: WEB TECHNOLOGIES AND SERVICES

- **Project/Application name:** Restaurant Ordering & Reservation Website
- **GitHub links (for both frontend and backend):** <https://github.com/24-25Sem1-Courses/ct313h03-project-ngothuythanhtram.git>
- **Video Demo link:** <https://youtu.be/qkObzXKYE5U>
- **Student ID 1:** B2111949
- **Student Name 1:** Ngô Thụy Thanh Tâm
- **Student ID 2:** B2111959
- **Student Name 2:** Nguyễn Thị Hoài Thương
- **Class/Group Number:** CT313HM03

## CONTENTS

I. Introduction .....	2
1. Project/application description: .....	2
2. A list of tables and their structures in the database .....	2
3. A task assignment sheet for each member. ....	3
II. Details of implemented features.....	4
1. Feature / Application page 1: Menu Mangement.....	4
2. Feature / Application page 2: Table .....	6
3. Feature / Application page 3: Reservation .....	8
4. Feature / Application page 4: Receipt .....	12
5. Feature / Application page 5: User .....	19

## I. Introduction

### 1. Project/application description:

The Restaurant Ordering & Reservation Website is an user – friendly platform desinged to streamline both table reservations and food orders for customers. It enables customers to place orders online, browse the menu, and either enjoy their meal at the restaurant or arrange for self-pickup.

### 2. A list of tables and their structures in the database

Our project has 6 tables, which are:

1. users
2. menu\_items
3. restaurant\_table
4. reservation
5. receipt
6. order\_item

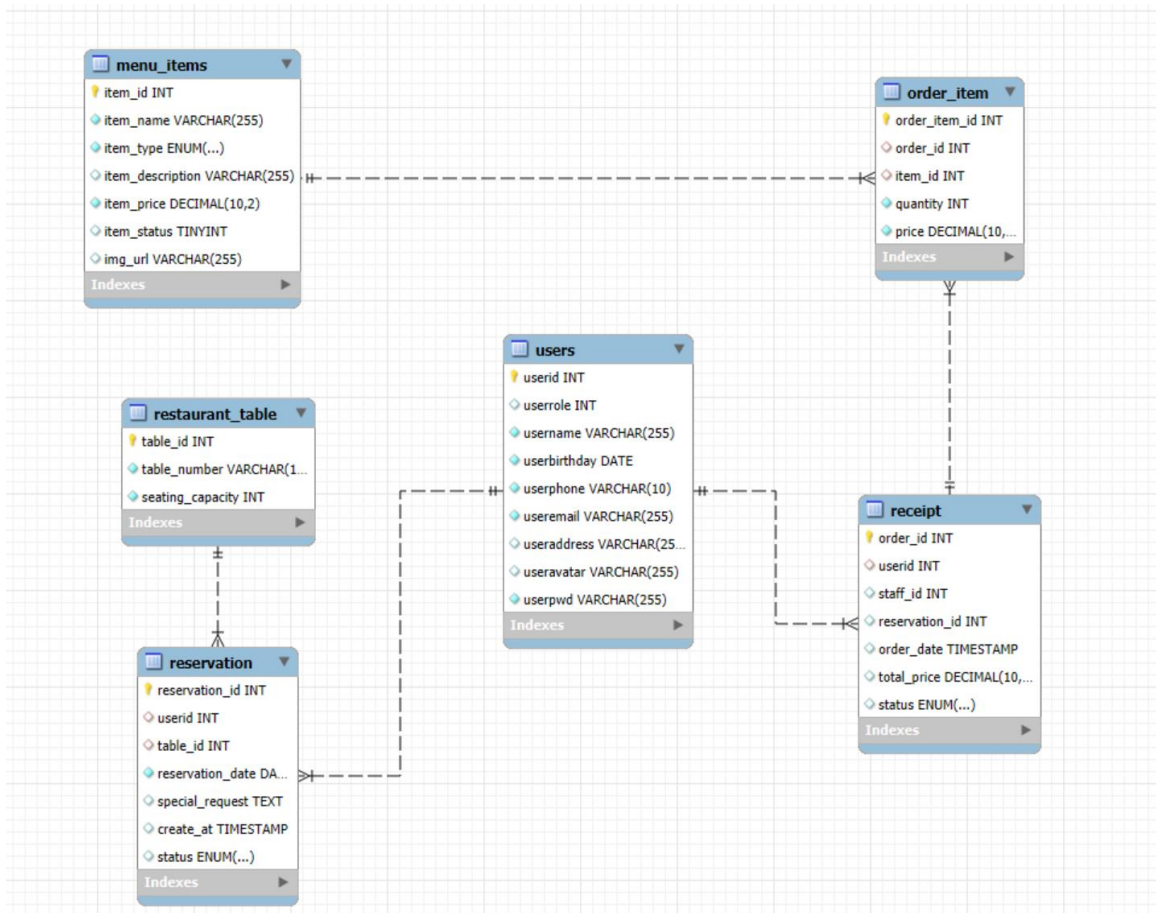


Image 1. Entity-Relationship Diagram

### 3. A task assignment sheet for each member.

Name	Assigned task
<b>TamB2111949</b>	<ul style="list-style-type: none"><li>- Set up project repository.</li><li>- Set up database and design table structures.</li><li>- Write database queries for fetching data.</li><li>- Ensure data consistency and optimize queries for performance.</li><li>- Develop service and controller functions for menu items (CRUD), users (GET), reservations (CREATE, UPDATE, GET) and tables (CREATE, GET, DELETE), receipts (GET, CREATE, UPDATE).</li><li>- Define routes and document API endpoints with Swagger, including parameters, responses, and error handling.</li><li>- Test CRUD operations for menu items, reservations, tables, receipts and GET users.</li><li>- Write detailed project documentation.</li><li>- In charge of designing the UI and implementing the logic service for staff interface, including features for managing the menu, tables, reservations, and receipts.</li></ul>
<b>ThuongB2111959</b>	<ul style="list-style-type: none"><li>- Set up project repository</li><li>- Set up database and design table structures</li><li>- Write database queries for fetching data</li><li>- Ensure data consistency and optimize queries for performance</li><li>- Develop service and controller functions for menu items (GET), users (CRUD), reservations (POST, UPDATE, GET), receipts (CRUD) and tables (GET).</li><li>- Define routes and write document API endpoints, including parameters and responses (Swagger), APIs error.</li><li>- Test CRUD operations for users, receipts.</li></ul>

- Test POST, GET, UPDATE for reservations.
- Test GET for menu items and tables.
- In charge of designing the UI and implementing the logic service for customer interface.
- Write detailed project documentation.

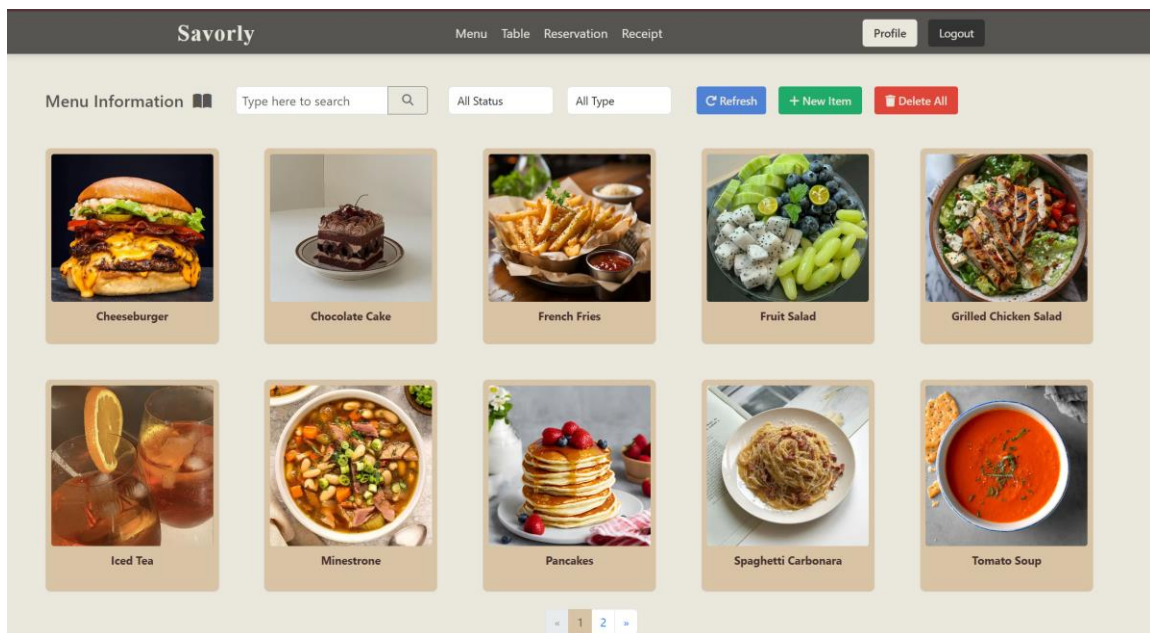
## II. Details of implemented features

### 1. Feature / Application page 1: Menu Management

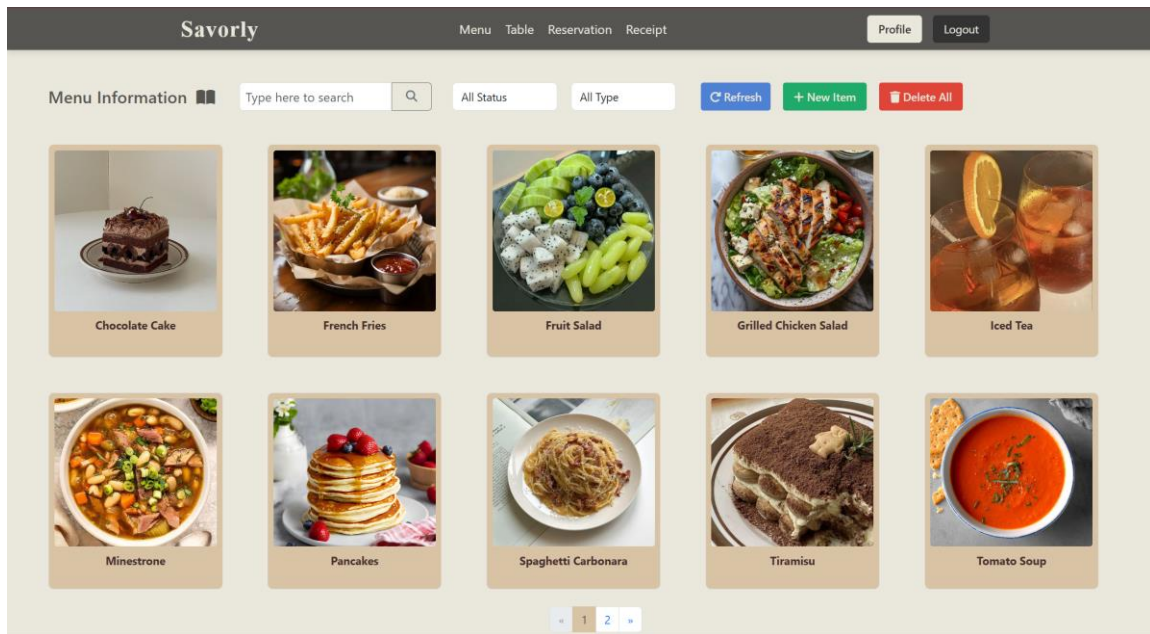
#### Description:

- In staff role: The menu management feature provides staff with tools to view and maintain the restaurant's menu. This includes viewing, adding new items, editing existing ones, and removing items. Validation checks minimize errors, ensuring data consistency.
- In customer role: The menu management feature provides customer with tools to view, search the restaurant's menu and add item from menu to Cart.

#### Screenshots:



*Image 2. Menu before deleting Cheeseburger*



*Image 3. Menu after deleting Cheeseburger*

## Implementation details:

### Libraries Used:

- **SweetAlert2:** Displays confirmation dialogs when deleting menu items, as well as success or error alerts for CRUD operations.

### **Server-Side APIs:**

**DELETE** /api/menu\_items/ByStaff/{item\_id}

**Description:** Staff delete item by id.

**Endpoint:** /api/menu\_items/ByStaff/{item\_id}

**Parameters (required):** item\_id (integer)

**Format Received:**

```
{
  "status": "success",
  "data": {
    "table": {
      "table_id": 0,
      "table_number": "string",
      "seating_capacity": 0
    }
  }
}
```

```
}  
}
```

### **Data Storage:**

This feature reads and stores data in the menu\_items table. The table fields include item\_id, item\_name, item\_type, item\_description, item\_price, item\_status, and img\_url. All database operations are managed using Knex. Image files are uploaded via Multer and stored on the server. When a menu item is deleted, the associated image is removed from the filesystem using unlink from Node.js's module.

### **Client-Side States:**

The **Menu Items State** not only stores fetched menu items, including all items or filtered results based on search criteria,... but it also ensures data consistency by verifying that the menu item name does not already exist in the database before new items are added or existing items are updated. This check helps avoid duplicate entries, maintaining the integrity of the menu data.

Filter/Search State: Holds search keywords or filter conditions (like name, type, price, or status) used to retrieve specific menu items.

Item Edit/Create State: Manages form data for adding or editing a menu item, including name, type, description, price, and the image.

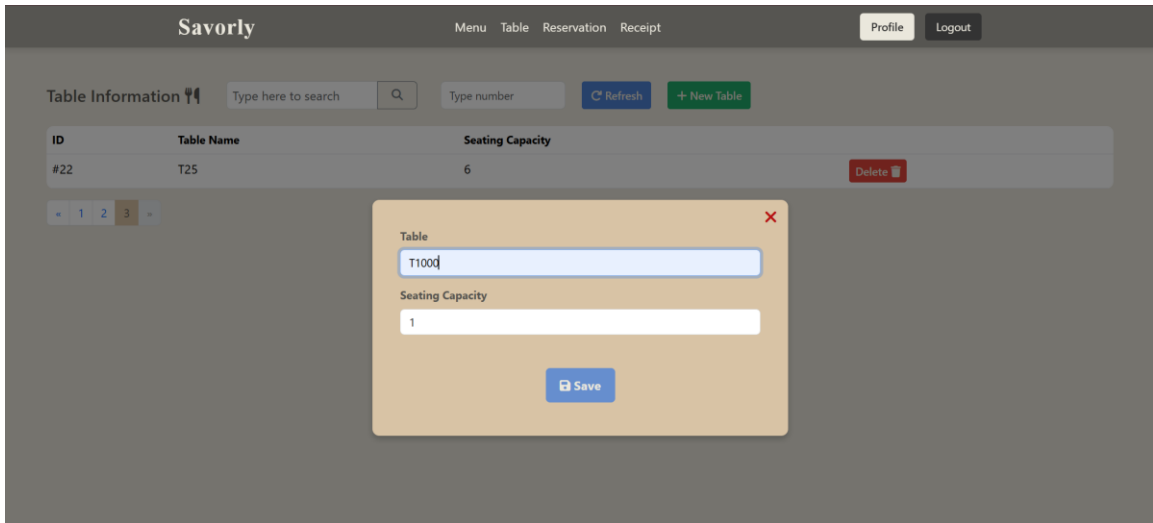
API Status State: Tracks the status of API calls (loading, success, or error) to provide feedback to the user.

## **2. Feature / Application page 2: Table**

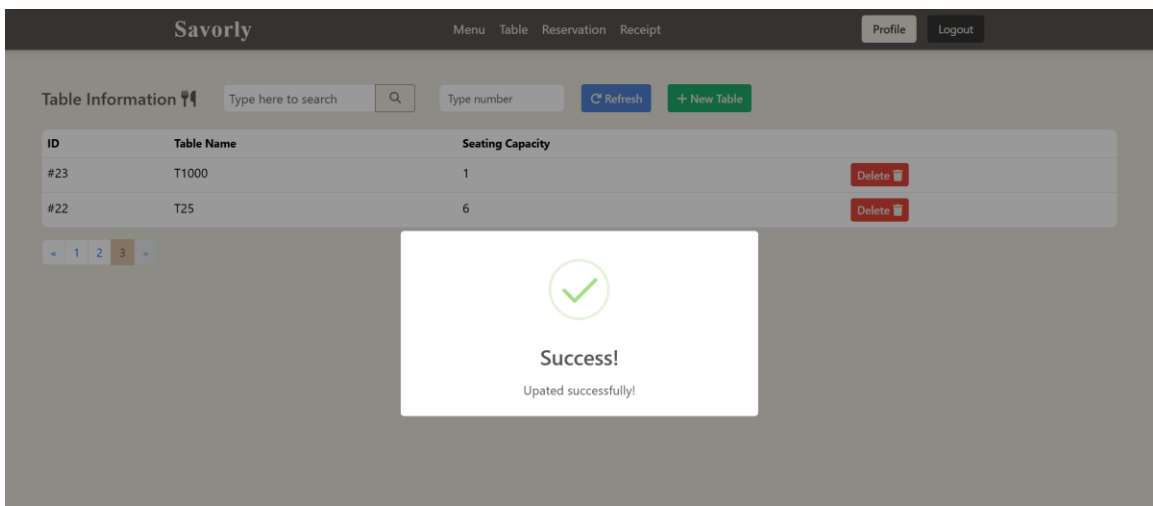
### **Description:**

- In staff role, the table management help staff organize and manage table. It provides table number and its seating capacity. Staff can quickly add new tables or remove existing ones.
- In customer role: The table management feature provides customer with tools to view the restaurant's table. This includes viewing tables.

### **ScreenShots:**



*Image 4. Table Form*



*Image 5. Table added successfully*

### Implementation details:

#### Libraries Used:

- **SweetAlert2:** Displays user-friendly alerts and confirmation dialogs for actions like deleting a table or notifying staff when a table is successfully added or updated.

#### Server-Side APIs:

POST /api/table/ByStaff

**Description:** Staff create a new table.

**Endpoint:** /api/table/ByStaff

**Form format:** multipart/form-data

**Request Body (required) :** table\_number (string), seating\_capacity (integer).

**Format Received:**

```
{
  "status": "success",
  "data": {
    "table": {
      "table_id": 0,
      "table_number": "string",
      "seating_capacity": 0
    }
  }
}
```

### **Data storage**

Table information is stored in a tables table with fields like table\_id, table\_number, seating\_capacity.

### **Client-side states**

Allows staff to add table.

## **3. Feature / Application page 3: Reservation**

### **Description:**

- In staff role, reservation management simplifies booking and updating reservations. Staff can create new reservations by inputting customer details, selecting available tables, and scheduling the desired time. Status updates, such as confirmed, completed, or canceled, can be made seamlessly. Transactions ensure that all related records are accurately updated, maintaining data integrity.
- In customer role, reservation management simplifies booking and updating reservations. Customer can create new reservations by inputting customer details, selecting available tables, and scheduling the desired time. The system will automatically check and notify customers if the table has been booked, ensuring data consistency. Status updates, such as confirmed, completed, or canceled, can be made seamlessly. Transactions ensure that all related records are accurately updated, maintaining data integrity.



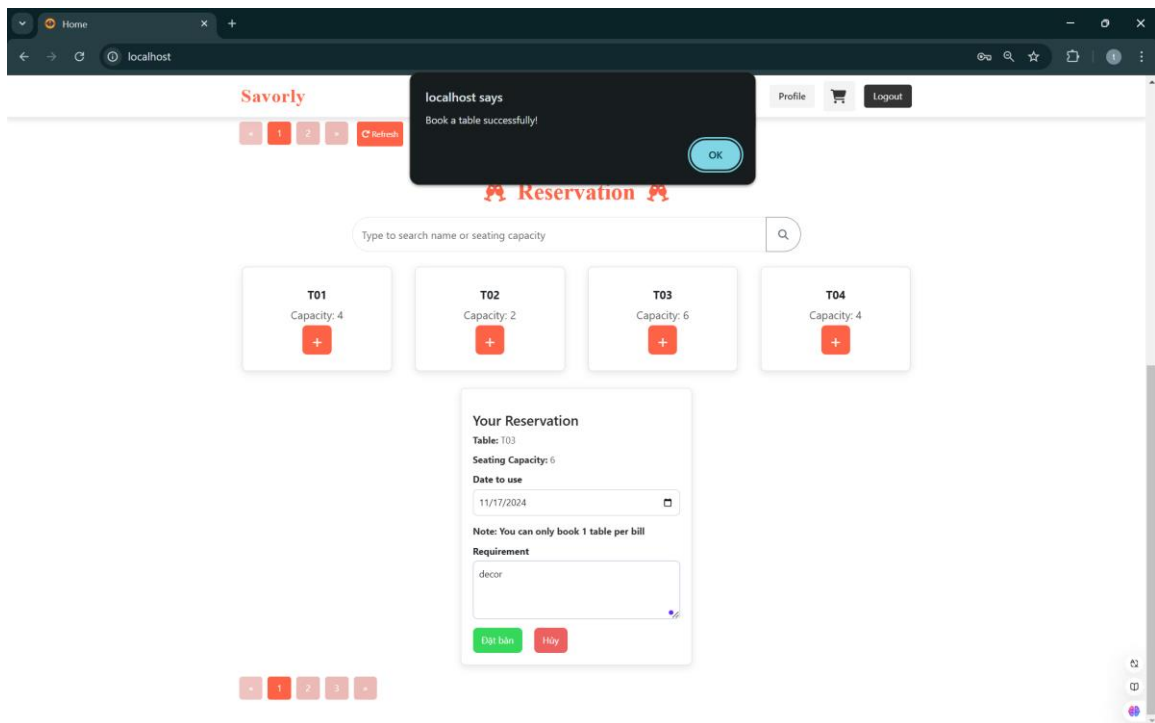
## ScreenShots:

The screenshot shows the Savorly web application interface. At the top, there is a navigation bar with the Savorly logo and links for Menu, Table, Reservation, and Receipt. On the right of the navigation bar are buttons for Profile and Logout. Below the navigation bar, there is a section for Reservation Information with a search bar, a Customer ID input field, and a dropdown menu set to All. There are also Refresh and New Reservation buttons. The main content area displays a Reservation List with IDs #1 through #8. A modal form is open in the center, titled 'Customer Email'. It contains input fields for Customer Email (filled with 'staff@gmail.com'), Table (filled with 'T10'), Date to use (filled with '11/17/2024'), and Requirement (filled with 'Requirement'). A Save button is at the bottom of the modal.

*Image 6. Staff create new reservation*

The screenshot shows the Savorly web application interface after a new reservation has been created. The Reservation List on the left now includes ID #9, which is highlighted. The Reservation Details panel on the right shows the following information: ID #9, Customer ID: 1, Customer Name: staff, Customer Email: staff@gmail.com, Customer Phone: 0987654321, Table: T10, Date to use: 17/11/2024, Requirement: (empty), Reservation Date: 16/11/2024 23:44:36, and Status: confirmed. At the bottom of the page, there is a pagination control showing '1'.

*Image 7. Result after staff creating new reservation*



*Image 8. Customer creating new reservation successfully*

### Implementation details:

#### Libraries Used:

- **SweetAlert2:** Displays user-friendly alerts for actions such as confirming reservation creation or handling errors when required fields are missing.

#### Server-Side APIs:

##### POST /api/reservation/ByStaff

**Description:** Staff create a new reservation for customer.

**Endpoint:** /api/reservation/ByStaff

**Form format:** multipart/form-data

**Request Body (required):** useremail (string), table\_number (string), reservation\_date (string (\$date)), special\_request (string).

#### Format Received:

```
{
  "status": "success",
  "data": {
    "reservation_id": 0,
```

```

    "reservation_date": "2024-11-17",
    "party_size": 0,
    "special_request": "string",
    "status": "booked",
    "user": {
      "username": "string",
      "useremail": "string"
    },
    "table": {
      "table_number": "string",
      "seating_capacity": 0,
      "table_status": "string"
    }
  }
}

```

## POST /api/receipts/addTable/

**Description:** Customer create a new reservtaion.

**Endpoint:** /api/receipts/addTable/

**Form format:** multipart/form-data

**Request Body (required):** table\_number (string), reservation\_date (string (\$date)), speacial\_request (string).

**Format Received:**

```

{
  "status": "success",
  "data": {
    "contact": {
      "order_id": 0,
      "userid": 0,
      "staff_id": 1,
      "reservation_id": 0,
      "order_date": "2024-11-17",
      "total_price": 0,

```

```
        "status": "Pending"
    }
}
}
```

### **Data storage**

Reservation details is stored in a reservations table, possibly containing fields like reservation\_id, user\_email, reservation\_date, and status.

### **Client-side states**

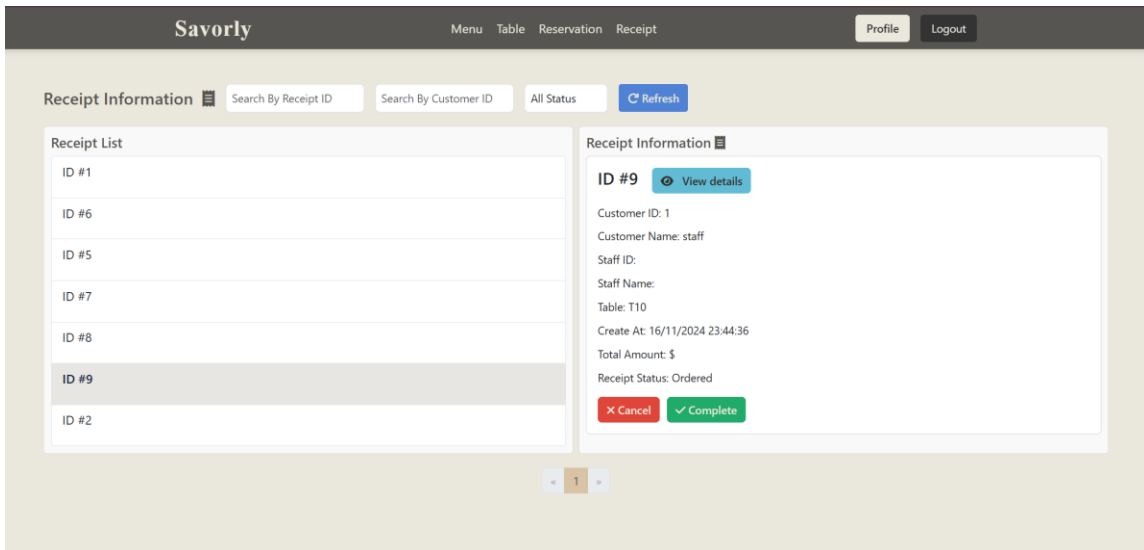
The current reservation state for the user (active reservations, canceled reservations) and user authentication state to retrieve their reservations. For staff, they can also update status of the reservation automatically from the receipt status. The reservation state also handles reservation details such as customer names, times, and assigned tables, ensuring smooth synchronization with the server when changes occur.

## **4. Feature / Application page 4: Receipt**

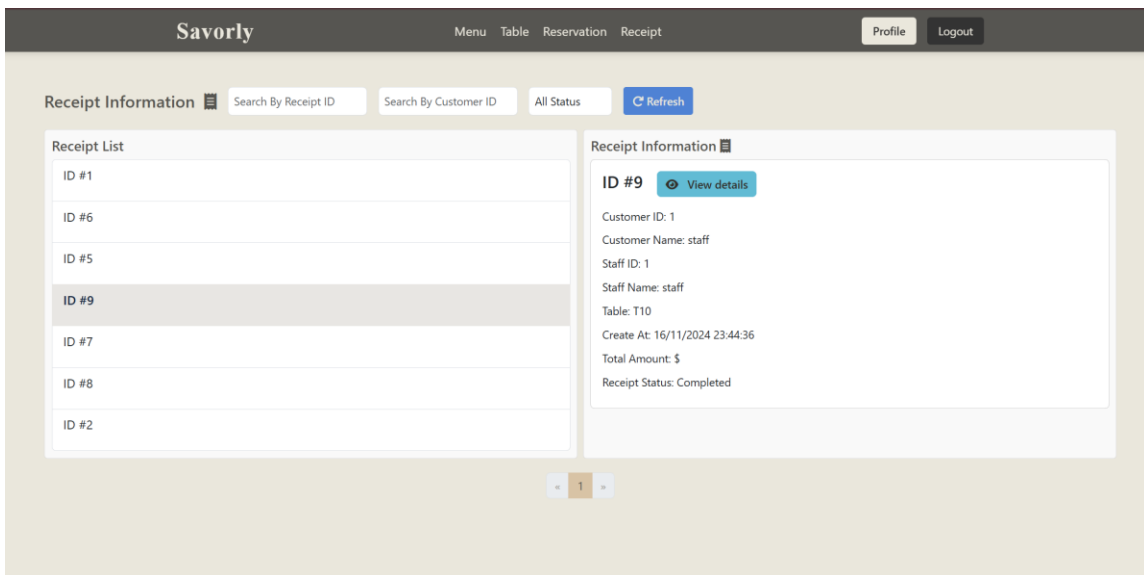
### **Description:**

- In staff role, receipt management allows staff to track the receipt efficiently. Receipts can be created directly from reservations. Staff can update receipt statuses to reflect payment completion or cancelation. Transactions ensure consistency across linked data, such as reservations and tables.
- In customer role, receipt management allows customers can track the receipt efficiently. Customers can create new reservation by providing order dates and optional reservation, and view their invoices with filters. For cart management, they can add items to their cart after verifying item availability or remove items if the items are present in the cart. Regarding reservations, customers can book tables by providing table numbers and reservation dates, with the system ensuring table availability and preventing duplicate bookings. They can confirm orders and reservations to finalize pending orders or cancel existing bookings or orders. Additionally, customers can view the details of their pending orders, including the receipt, reservation, items, and table details. Overall, these features enable customers to seamlessly manage their purchasing experience, from creating and managing orders to handling reservations, with all actions validated for accuracy and reliability.

### **ScreenShots:**



*Image 9. Receipt before staff updating status*



*Image 10. Receipt after staff updating to complete*

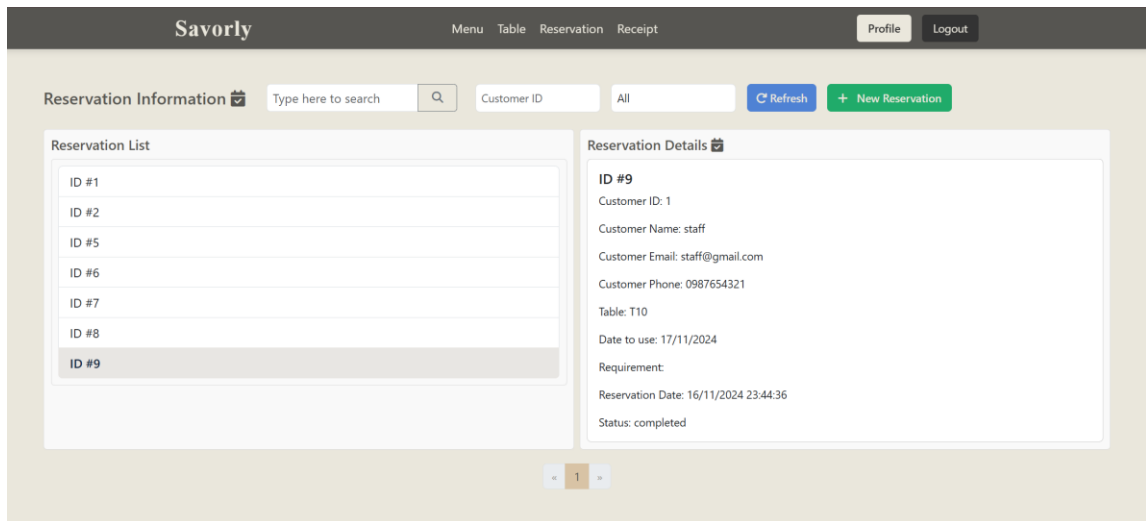


Image 11. Reservation after staff updating receipt status

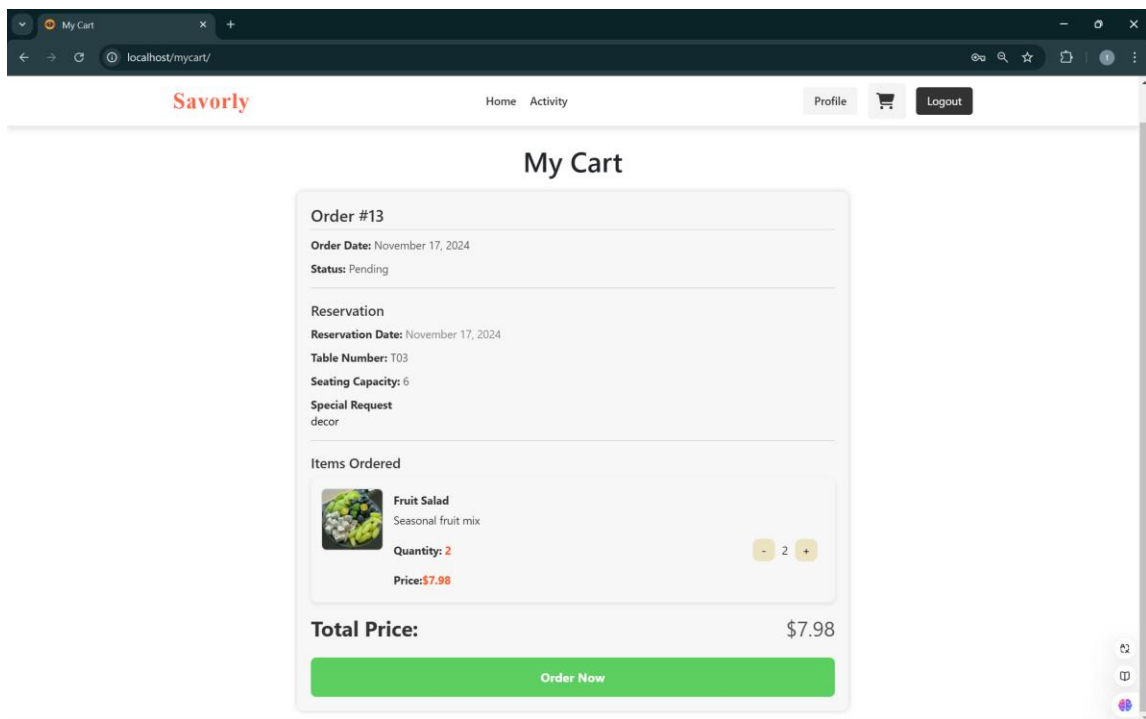
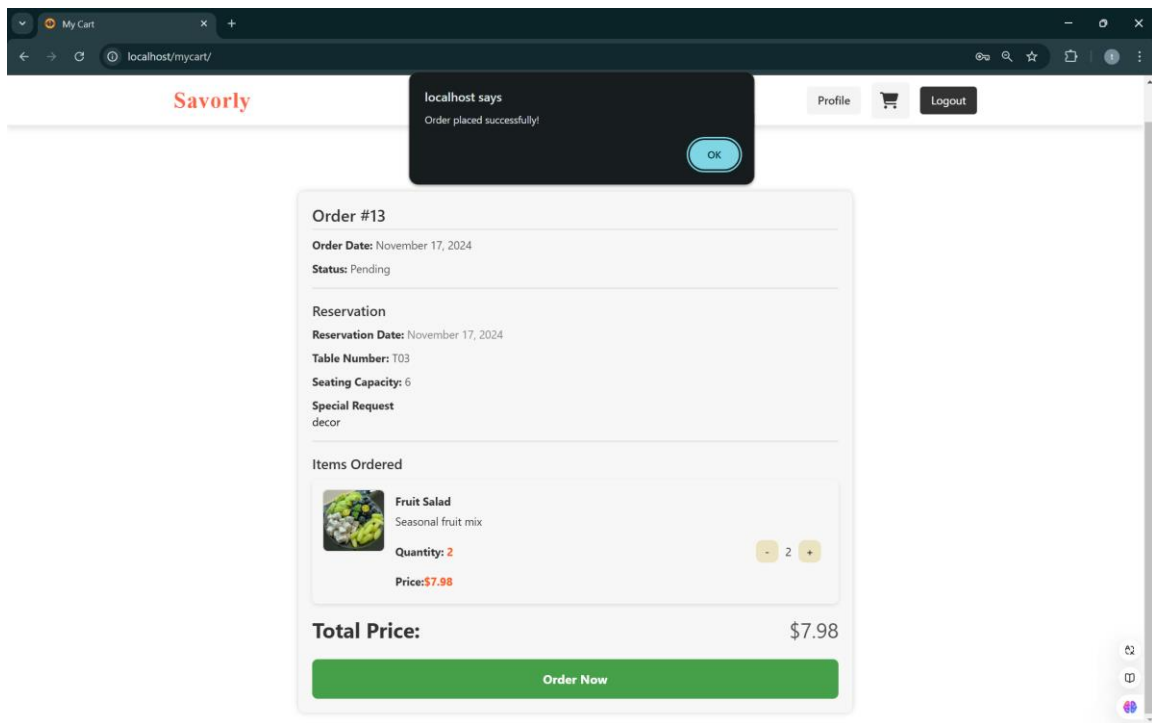
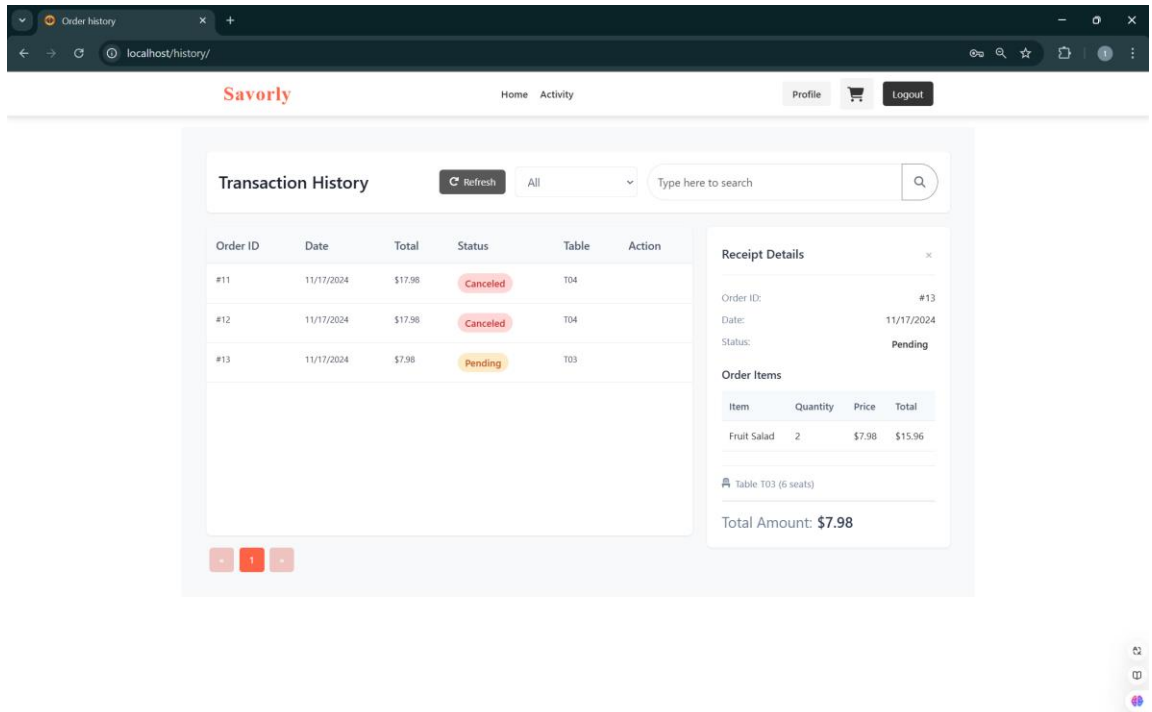


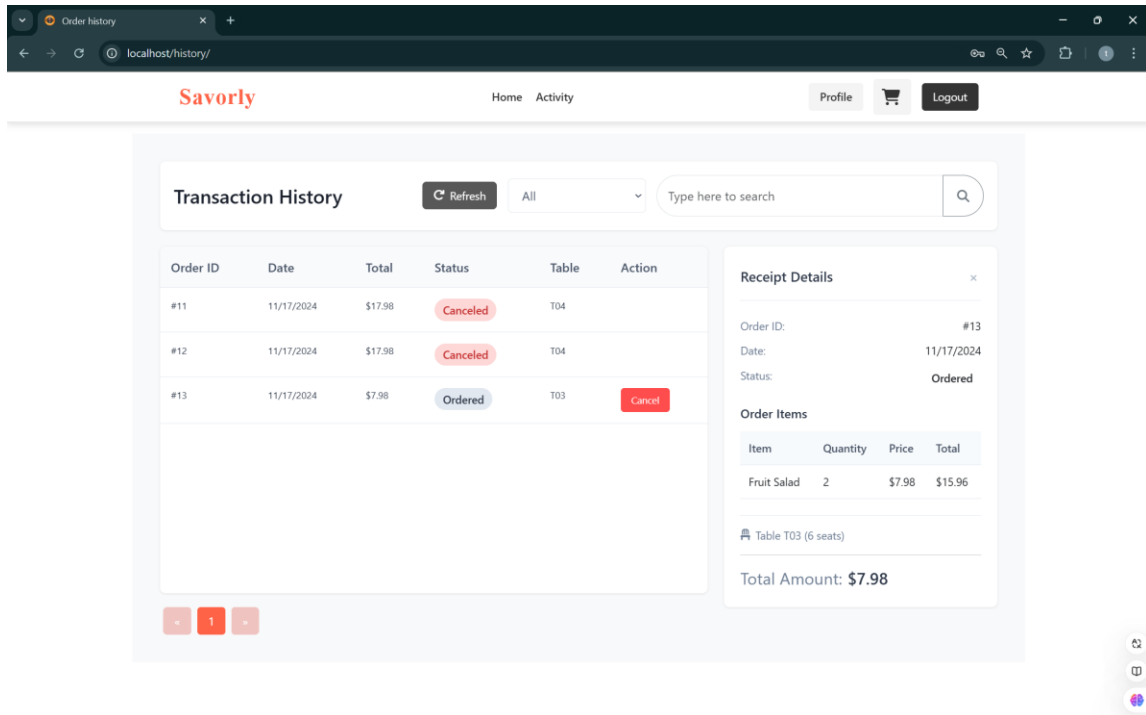
Image 12. Reservation and Ordering Food before customer updating receipt



*Image 13. Reservation and Ordering Food after customer updating receipt status*



*Image 14. Reservation and Ordering Food before customer updating receipt status in History*



*Image 15. Reservation and Ordering Food after customer updating receipt status in History*

### Implementation details:

#### Libraries Used:

- **SweetAlert2:** Displays confirmation dialogs and alerts for actions like confirming, editing receipt status, or handling errors when required fields are missing.

#### Server-Side APIs:

**PUT / api/receipts/verify/ByStaff/{order\_id}**

**Description:** Change status of a specific receipt.

**Endpoint:** /api/receipts/verify/ByStaff/{order\_id}

**Form format:** multipart/form-data

**Parameters (required):** order\_id (integer)

**Request Body (required):** status (string)

**Format Received:**

```
{
  "status": "success",
```



```
"data": {
  "receipt_info": {
    "order_id": 0,
    "userid": 0,
    "user_name": "string",
    "staff_id": 0,
    "staff_name": "string",
    "reservation_id": 0,
    "table_id": "string",
    "order_date": "2024-11-17T05:16:18.451Z",
    "total_price": 0.1,
    "status": "Pending",
    "order_items": [
      {
        "order_item_id": 0,
        "item_id": 0,
        "item_name": "string",
        "quantity": 0,
        "item_price": 0,
        "price": 0.1
      }
    ]
  }
}
```

## POST /api/receipts/addItem

**Description:** Adding item to Cart

**Endpoint:** /api/receipts/addItem

**Form format:** multipart/form-data

**Request Body (required):** item\_id (integer), quantity (number)

**Format Received:**

```
{
```

```
"status": "success",
"data": {
  "customer": {
    "order_item_id": 0,
    "order_id": 0,
    "item_id": 0,
    "quantity": 0,
    "price": 0.1
  }
}
```

### PUT /api/receipts/customer/verify/

**Description:** Customer verify ordering

**Endpoint:** /api/receipts/customer/verify/

**Form format:** multipart/form-data

**Request Body (required):** status (string)

**Format Received:**

```
{
  "status": "success",
  "data": null
}
```

### Data storage

Receipt details are stored in a receipt table, possibly containing fields like order\_id, userid, staff\_id, reservation\_id, order\_date, total\_price, and status.

### Client-side states

The system manages receipts and items for a restaurant-style order process, where client-side state includes details such as user ID, staff ID, reservation ID, order date, total price, and status. When a user adds an item to their receipt, the system first checks if there is an existing "Pending" receipt for the user. If none exists, a new one is created. Items can then be added to the receipt, with their quantities and total prices updated accordingly. If an item is removed or its quantity is reduced to zero,

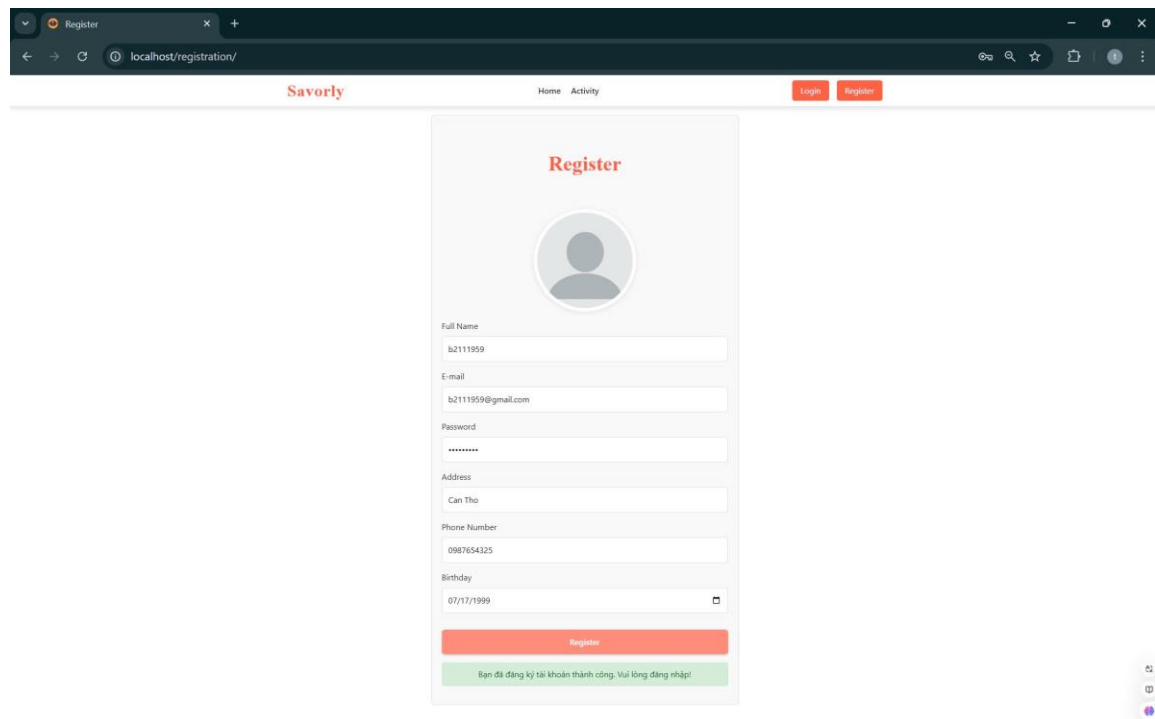
the system adjusts the total price, and if no items remain, the receipt is deleted. Additionally, the system allows the status of an order to be updated (e.g., from "Pending" to "Ordered" or "Canceled") depending on the user's actions. All operations are handled within database transactions to ensure consistency, and users are provided with clear feedback messages after each action.

## 5. Feature / Application page 5: User

### Description:

In customer role, user management allows manage user accounts securely. Customer can create new user by validating user information, including email format, phone number, and password strength. Customers, after authenticating via the login feature can manage their profiles using several functionalities. They can view their profile information, update personal details, and securely logout to end their session. The system ensures only authenticated users access sensitive operations, protecting user data. Moreover, customers have the option to delete their accounts permanently if they wish to stop using the system. Comprehensive input validation and session management enhance security and user experience.

### Screenshots:



The screenshot displays a web browser window with the URL `localhost/registration/`. The page features a navigation bar with the "Savorly" logo, "Home" and "Activity" links, and "Login" and "Register" buttons. The main content area is titled "Register" and includes a user profile icon. Below the icon, there are input fields for "Full Name" (containing "b2111959"), "E-mail" (containing "b2111959@gmail.com"), "Password" (masked with "\*\*\*\*\*"), "Address" (containing "Can Tho"), "Phone Number" (containing "0987654325"), and "Birthday" (containing "07/17/1999"). A red "Register" button is positioned below these fields. At the bottom of the form, a green message box states: "Bạn đã đăng ký tài khoản thành công. Vui lòng đăng nhập!" (Your account registration was successful. Please log in!).

*Image 16. Register successfully*

## Server-Side APIs:

### POST /api/users/registration/

**Description:** Customer create accounts

**Endpoint:** /api/users/registration/

**Form format:** multipart/form-data

**Request Body (required):** userrole (integer) – default =1 and does not allow user input, username (string), userbirthday (string \$date), userphone (string), useremail (string \$email), userpwd (string \$password), useraddress (string) and useravatarFile (string \$binary)

**Format Received:**

```
{
  "status": "success",
  "data": {
    "contact": {
      "userid": 0,
      "userrole": 1,
      "username": "string",
      "userbirthday": "2024-11-17",
      "userphone": "string",
      "useremail": "user@example.com",
      "userpwd": "*****",
      "useraddress": "string",
      "useravatar": "string"
    }
  }
}
```

## Data storage

User details is stored in a users table, including fields like email, and other personal information.

## Client-side states

This feature reads and stores data in the users table. The users fields include userid, userrole, username, userbirthday, userphone, useremail, useraddress,... All database

operations are managed using Knex. Image files are uploaded via Multer and stored on the server. When a user is deleted, the associated image is removed from the filesystem using unlink from Node.js's module.