Vietnamese-German University

NGON

Authentic Vietnamese Cuisine Restaurant

Vương Thiệu Luân - 10421035
Phạm Trần Trường An - 10421001
Nguyễn Trực Lĩnh - 10421088
Nguyễn Ngọc Hoàng Nam - 10421042
Nguyễn Nho Minh Trí - 10421061
Cao Thăng Long - 10421089
Từ Trung Tín - 10421106

I.Table of contents

II. Introduction

a. About NGON

Welcome to NGON, a newly established Vietnamese restaurant dedicated to passing on the delicious tastes and aromas of authentic Vietnamese cuisine. Our menu honors traditional recipes passed down through generations, prepared with diligent care, and using the finest ingredients sourced locally and from across Vietnam. Join us and embark on a culinary adventure that celebrates the timeless flavors and traditions of Vietnam.

b. About our team

Meet the passionate and dedicated team behind NGON, our diverse group of chefs, servers, and shippers, who share a common objective: to provide an exceptional dining experience that captures the true spirit of Vietnam. Each member of our team contributes significantly to bringing the flavors of Vietnam to life in each dish we serve, combining expertise, creativity, and a deep respect for tradition. From our skilled chefs who create authentic recipes with precision and care to our friendly shippers who provide efficient and reliable services, every interaction with our team exudes warmth and hospitality. Join us and discover the passion, dedication, and talent that define the heart of our team at NGON.

c. About Vietnamese Cuisine

Vietnamese cuisine is renowned for its balance of fresh ingredients, aromatic herbs, and complex yet harmonious flavors, resulting in a delightful fusion of influences from Southeast Asia and beyond. From fragrant herbs like basil, cilantro, and mint to zesty citrus and pungent fish sauce, each dish is carefully prepared to achieve the perfect balance of flavors and textures. Join us in celebrating the flavors, traditions, and stories of Vietnam through our authentic cuisine. Whether you're a seasoned foodie or a curious newcomer, Vietnamese cuisine promises a sensory experience that will leave you wanting more. Chào mừng bạn.(Welcome!)

d. About our website

Our website offers a convenient way for you to explore, discover, and enjoy our finest selection of Vietnamese cuisine. Simply place your order online, and we'll deliver it right to your door. We provide a variety of payment options, from cash on delivery to Visa/Mastercard, to ensure a smooth transaction process. Our goal is to bring the essence of Vietnamese culinary tradition directly to your home, allowing you to enjoy the spirit of Vietnamese cuisine in the comfort of your very own home.

III. Requirements

a. Use-case diagram

- The Client:
  - The use-case “Login”: Clients need to login with a valid email and password which have been sign up to access our website and order food.
  - The use-case “Sign up”: Clients need to sign up with a valid email and password to be able to login.
  - The use-case “View Restaurant Menu”: Clients can browse through our restaurant's menu directly on the website.
  - The use-case “Forgot Password”: Clients can reset their password if they forget it.
  - The use-case ”View Cart”: Clients can check their current order and its total price in the shopping cart.
  - The use-case ”Edit profile”: Clients have the option to update their personal information, including their password, addresses, payment details, name, and phone number.
  - The use-case “View Order History”: Clients can review both past and current orders, including those currently out for delivery.
  - The use-case “Place Order”: Clients can confirm and place their orders.
  - The use-case “Cancel Order”: Clients can cancel an order, changing its status to "Canceled."

- The use-case “Logout”: Clients can log out of their account when they're finished using the website.

- Salesman
  - The use-case “Add New Item”: Salesmen can include new products into the menu.
  - The use-case “Delete Item” : Salesmen have the capability to remove existing products from the menu.
  - The use-case “Edit Item”: Salesmen can modify product details on the menu, such as price, name, description, and image.
  - The use-case “Place Order”: Salesmen can access order details once a client has confirmed their order.
  - The use-case “View Report”: Salesmen can review all orders placed through the website.

b. Use-case table

1. Login

Name	Description
Actor	Client
Main function	Login
Brief description	The Client logs in to their account to start ordering food
Context information	Client want to order food on the web page
Triggers	Client device must be connecting to a network
Flow of events summary	1. Client go to the login page 2. Client input the registerd email and corresponding password 3. System provides the requested login.

2. Forget password

Name	Description
Actor	Client
Main function	Change Password
Brief description	The client changes their account password in their profile
Context information	Forget password
Triggers	Client forgets their password and must confirm the email that the system has sent to change password

Name	Description
Flow of events summary	1. Client go to the forget password page 2. Client input the email address of the account which the client has forgot password. 3. Client receive an email from the system to verify the account 4. Client input the new password. 5. System save the client's new password.

3. View cart

Name	Description
Actor	Client
Main function	View Cart
Brief description	- The client can view their cart ID and the date that they ordered. - The client views their cart to check the number of items chosen, the price of each item, the image, category and quantities of the dish and notes for the chef. - The Client can add items, remove items or increase the quantity of each item if needed.
Context information	Client has finished choosing their dish from the menu or wants to view
Triggers	The Client has already login to the webpage
Flow of events summary	1. Client login to the webpage. 2. Client click on the Cart Icon 3. System open the Cart Page 4. Client can change the quantity of the item or removes the current exisiting item. 5. If the Client having choose a dish, the cart is empty

4. Edit profile

Name	Description
Actor	Client, Salesman
Main function	Edit Profile
Brief description	- The client manages to add or update their address and add their payment method. - The salesman managed to change the client information in the profile.
Context information	The address information in the profile has not been set or wrong. The payment has not been set
Triggers	Client has already login to the web page and change information

Name	Description
Flow of events summary	1. Client open their profile page. 2. Client manage to add or update their address 3. Client

5. View Order History

Name	Description
Actor	Client
Main function	View Order History
Brief description	The client views their order has been delivered and details such as order number, date of order and total amount of items of those orders.
Context information	Client want to reorder base on their past order, view the past orders or view the ongoing order
Triggers	Client has already login to the page
Flow of events summary	1. Client opens the order history page. 2. Client reorders their past order, views the past orders, or views the ongoing order.

6. View Restaurant Menu

Name	Description
Actor	Client
Main function	View Restaurant Menu
Brief description	The client views our restaurant menu and item details such as dish ID, category, image.
Context information	Client want to order food from the Restaurant menu
Triggers	Client must be logged in
Flow of events summary	1. Client opens the menu. 2. Client chooses the item to add to their cart.

7. Place an Order

Name	Description
Actor	Client, Salesman

Name	Description
Main function	Place an Order
Brief description	The Client accesses the online menu, selects items to order, and submits the order. It also notifies the Salesman about the new order.
Context information	Finish choosing item in the menu
Triggers	Clients must login to our webpage. Client must choose at least one item to add to their cart
Flow of events summary	1. Client go to menu page to choose dishes to order. 2. Clients go to Cart page to see existing dishes and make some changes if needed. 3. Clients can add more item to the cart before going to the cart page for payment 4. Client chooses delivery address. 5. Client click on button place order. 6. Salesman gets notify about Client's order and start to prepare the order. 7. System notify the Client that order has been placed.

8. Cancel Order

Name	Description
Actor	Client
Main function	Cancel Order
Brief description	The Client cancels their order before it is prepared or delivered
Context information	Wrong order
Triggers	Client chooses the wrong item or simply wants to cancel the order. Client must already login to our webpage. The Order status must in Pending
Flow of events summary	1. Client opens the order history page. 2. Client chooses the ongoing order and click on the Trash Icon.

9. Logout

Name	Description
Actor	Client

Name	Description
Main function	Logout
Brief description	The client logs out of their account after finishing ordering
Context information	Order delivered
Triggers	Receive order from the shipper
Flow of events summary	Client logs out of the account

10. Sign up

Name	Description
Actor	Client
Main function	Signup
Brief description	The clien create an account on our website
Context information	Clients wants to get food on our restaurant, therefore they must have an account
Triggers	Clients device must be connected to internet
Flow of events summary	1. Clients first go to our Sign up page 2. Clients than input their email address and password 3. Clients than have to verify their email via OTP send to their mail boxes 4. After inputting the OTP, the account is created

11. Add new item

Name	Description
Actor	Salesman
Main function	Add new item
Brief description	The salesman adds new item to the restaurant menu
Context information	When there are new items on the menu
Triggers	The restaurant has a new dish. Salesman login to the web page using salesman's account.
Flow of events summary	1. Salesman opens the restaurant menu. 2. Salesman adds new item to the menu. 3. Salesman confirm the new item.

12. Delete item

Name	Description
Actor	Salesman
Main function	Delete item
Brief description	The salesman deletes an item from the restaurant menu
Context information	When the item no longer available
Triggers	The restaurant no longer serves that dish. Salesman login to the web page using salesman's account.
Flow of events summary	1. Salesman opens the restaurant menu. 2. Salesman removes that item from the menu.

13. Edit Item

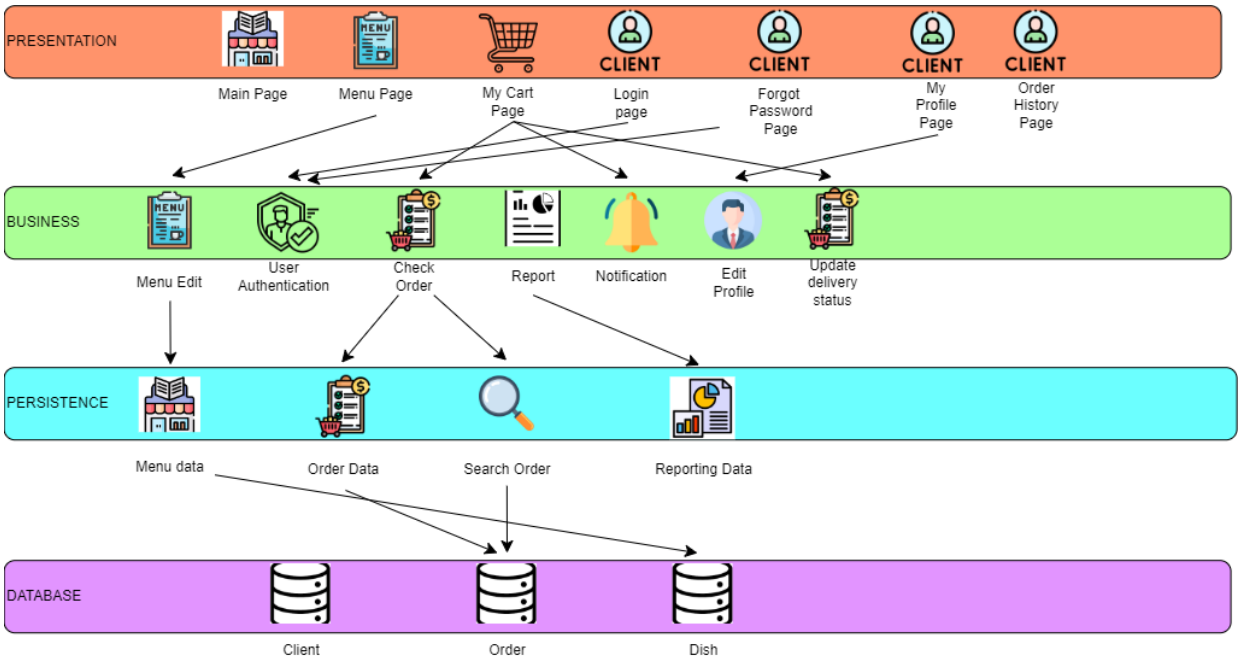
Name	Description
Actor	Salesman
Main function	Edit Item
Brief description	The salesman edits item price or information in the restaurant menu
Context information	
Triggers	The material price of that dish goes up or the restaurant discounts that dish. The information of that dish is wrong or not full. Salesman login to the web page using salesman's account.
Flow of events summary	1. Salesman opens the restaurant menu. 2. Salesman edits item price or information in the menu.

14. View Report

Name	Description
Actor	Salesman
Main function	View Report
Brief description	The salesman check report about profit that restaurant has been made in that day or problem that occured
Context information	At the end of the day

Name	Description
Triggers	After a day of restaurant operations
Flow of events summary	1. Salesman opens system database. 2. Salesman checks orders and transactions of the day using a dashboard

c. Layer Architecture



1. Presentation layer:

- The topmost layer of the program is where users interact with it. It has a number of pages that the users, or clients, can access:
    - Menu Page: Shows the goods that are available.
    - My Cart Page: Displays the goods that have been added.
    - Forgot Password Page: The page in which the user can change their password if the Customer forget the password.
    - Place Order Page: This page allows customers to complete their orders.
    - Login Page: For verifying user identity.
    - My Account Page: Allows users to see and modify account information.
    - Order History Page: Allows users to examine previous orders.

2. Business Layer:

- This layer handles data processing between the presentation and persistence levels and houses the application's business logic.
    - Menu Edit: Controls how menu items are edited.
    - User authentication: Manages user verification and login processes.
    - Check Order: Handles order status and verification.
    - Report: Produces different reports by using data.
    - Notification: Oversees the distribution of user notifications.
    - Edit Profile: Enables users to make changes to their personal data.
    - Update Delivery Status: This modifies the order's delivery status.

3. Persistence Layer:

- Data storage and retrieval are handled by this layer. It communicates with the database directly:
    - Menu Data: Holds details about the items in the menu.
    - Order Data: Oversees order-related data.

- Search Order: Enables you to look for specific order information.
- Reporting Data: Holds information needed to produce reports.

4. Database Layer:

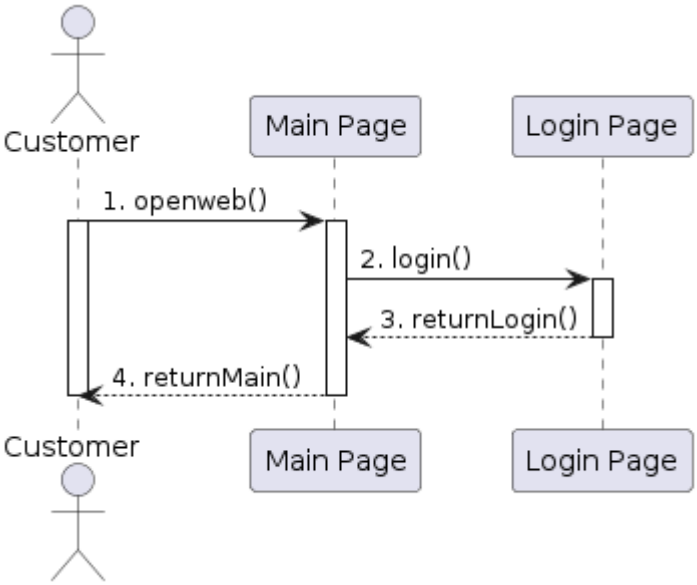
- The real data is kept in this lowest layer. It has multiple databases for diverse kinds of data:
  - Client Database: Holds data about users.
  - Dish Database: Maintains all the dish of the Restaurant.
  - Order Database: Provides details about user-placed orders.

5. Interaction flow:

- Presentation to Business: In order to handle user requests and actions, the presentation layer communicates with the business layer. For instance, the User Authentication component receives a request from the login page when a user logs in.
- Business to Persistence: To save or retrieve data, the business layer communicates with the persistence layer. For example, to preserve the order details, the Place Order Page communicates with the Order Data component when an order is placed.
- Persistence to Database: To carry out CRUD (Create, Read, Update, Delete) actions on the stored data, the persistence layer makes direct database access.

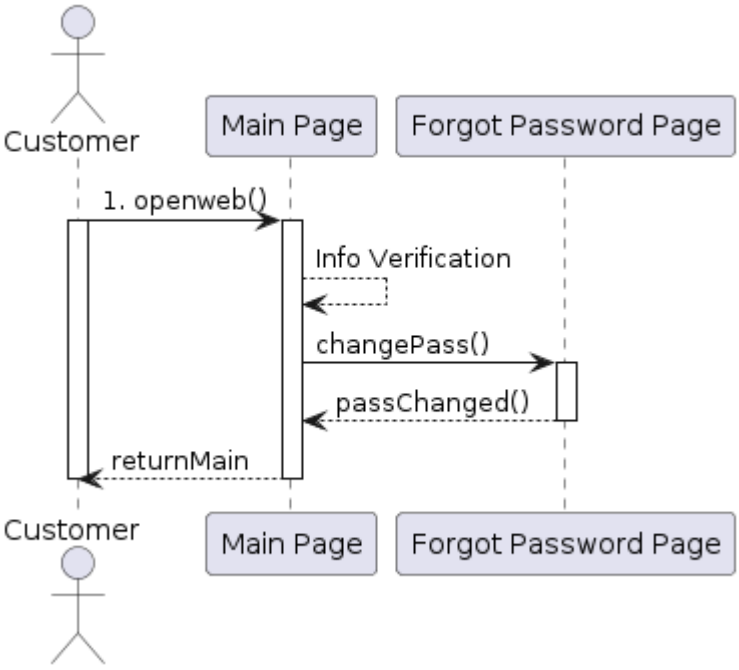
d. Sequence Diagram

1. Login Sequence Diagram



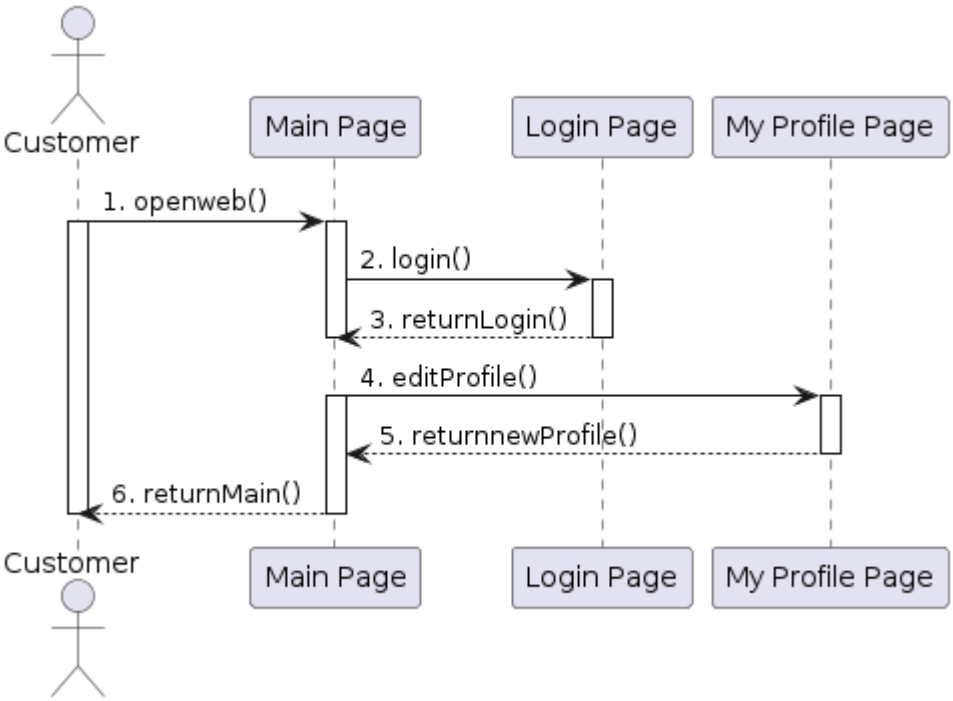
- Before the Customer can use any of our web page services, the customer will have to Login first.
- This requires the Customer to go to our webpage than Login via email address.

2. Change Password Sequence Diagram



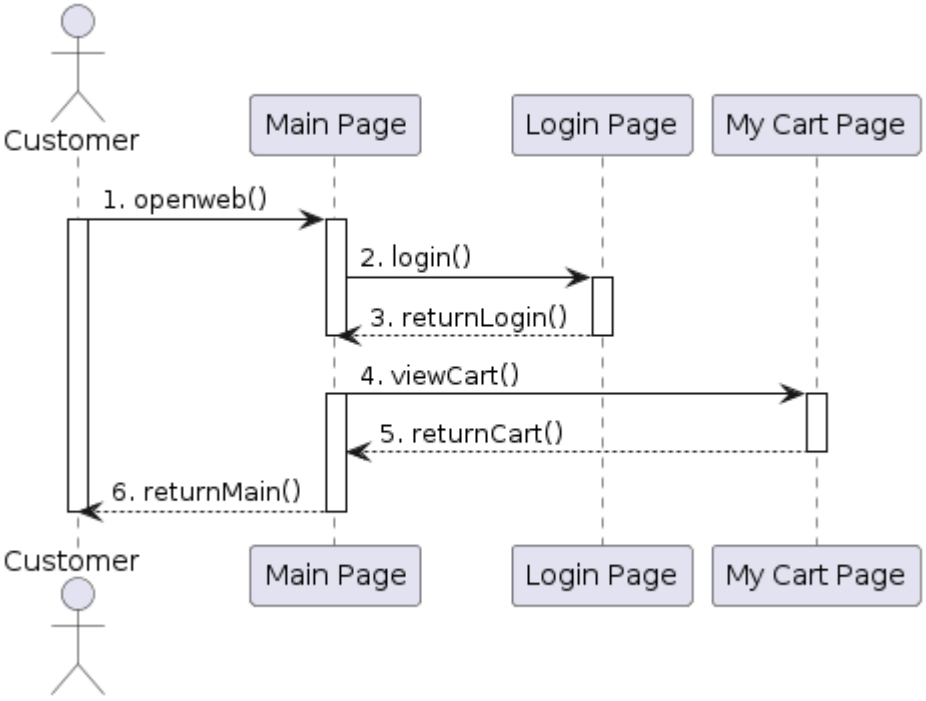
- When the Customer forget their password. The customer can change it.
- The Custumer must first go to our main page and verify the account.
- The step is required to assure that the changing password account belong to the customer.
- After verification, the customer will be forwarded to the Forgot Password Page to enter the new password.
- The process is successfull when the Customer is return back to the main page.

3. Edit Profile Sequence Diagram



- The Customer can edit their personal information.
- There are some information that require the Customer to added before hand for instance Delivering address and payment method before the Customer can Place an Order.
- The Customer must first login to the web page
- Then the Customer can click on my profile to edit their personal information.
- After that, the System will save the Customer information in the database and return the customer back to the main page.

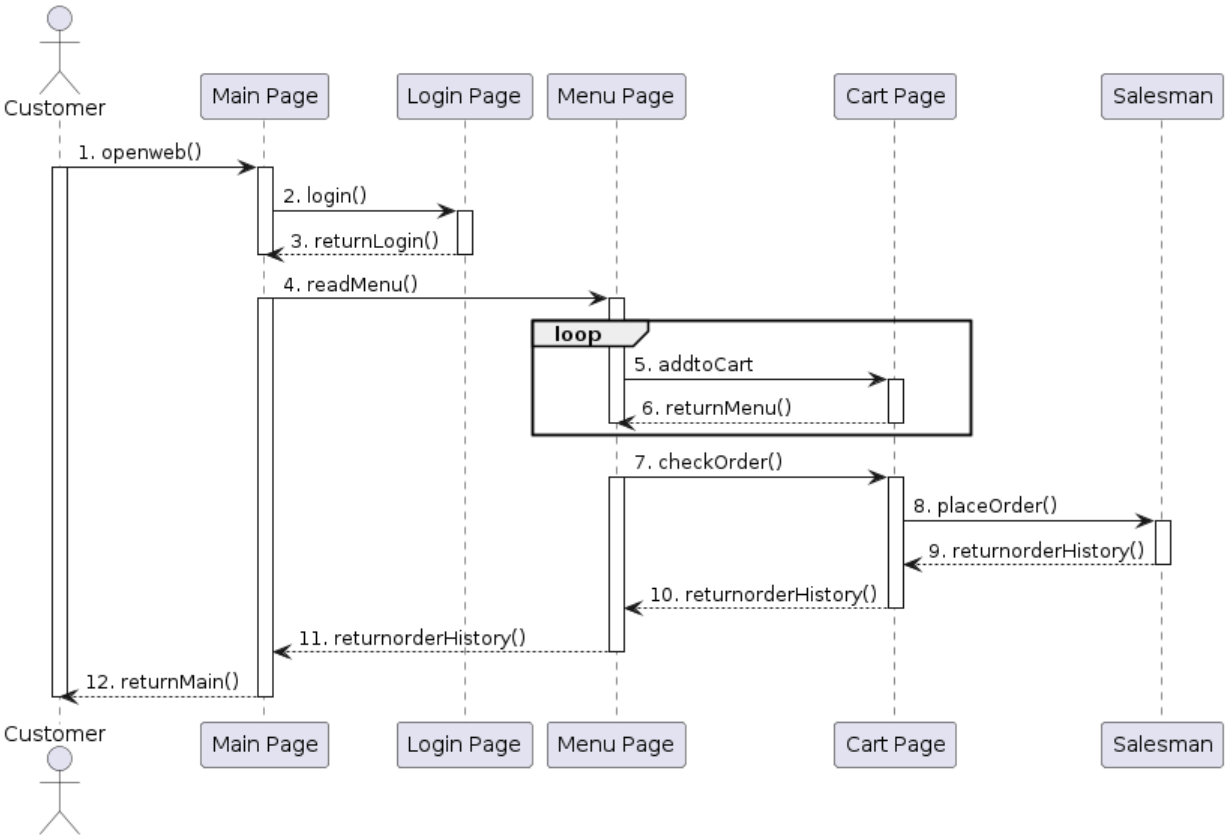
4. View Cart Sequence Diagram



- Before the Customer place an order, they can check what is currently in their cart.
- This process doesn't require the Customer to choose any dish before hand, although it does require the customer to login to the web page with their personal account.
- To view the Cart, the customer can simply click on the cart icon, this will open the My Cart Page.

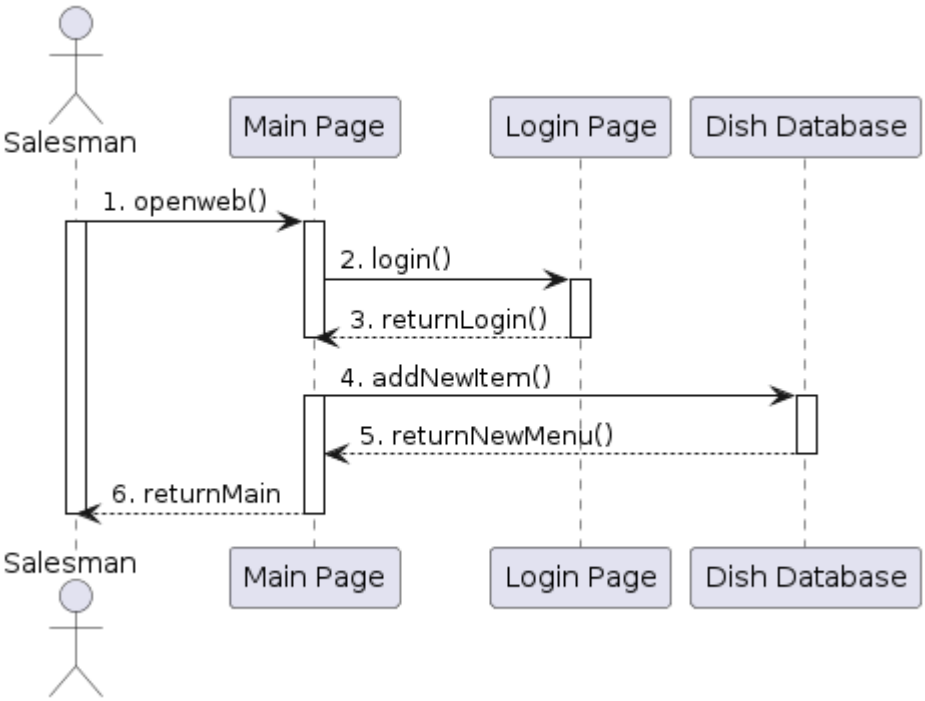
- If the Customer haven't choose any item, the cart will be shown empty.

5. Place an Order Sequence Diagram.



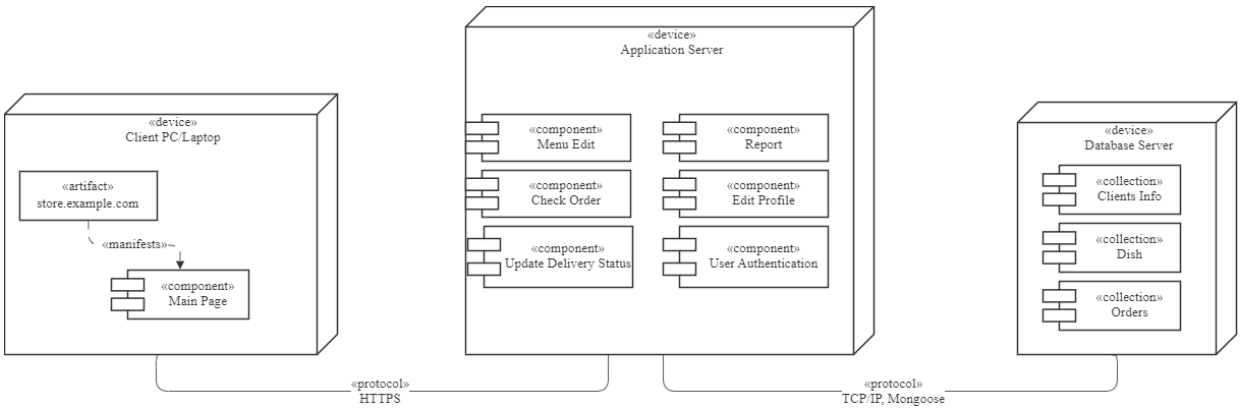
- Before the Customer can Place an Order, the Customer have to login to our webpage using their account.
- Then on the Main Page, the Customer can read our menu by simple click on the Menu Icon on the main page.
- The Customer than can add item to cart, and go to the Cart page to modify the quantity, delete or leave a chef notes for dishes that have been added to the Cart if necessary. This process can be repeat multiple times.
- After choosing the all the desir dishes, the Customer have to go to the Cart page for checking the Order.
- This step included checking the total amount, choosing the desire delivering address and payment method.
- Finally, the Customer click on the place order button, which will notify the Salesman about the new order.
- The Salesman will notify via the webpage to the customer if the order is being prepared.

6. Add New Item to the Menu sequence Diagram.



- Before adding new item to the menu, the Salesman must login to the webpage using the Salesman account.
- After logging in, the salesman can add new item to the menu at the menu page.
- The new item must be provided with all the necessary information before adding to the Dish database.
- After all the confirmation, the new item will be displayed on the menu and can be view by the Customer.

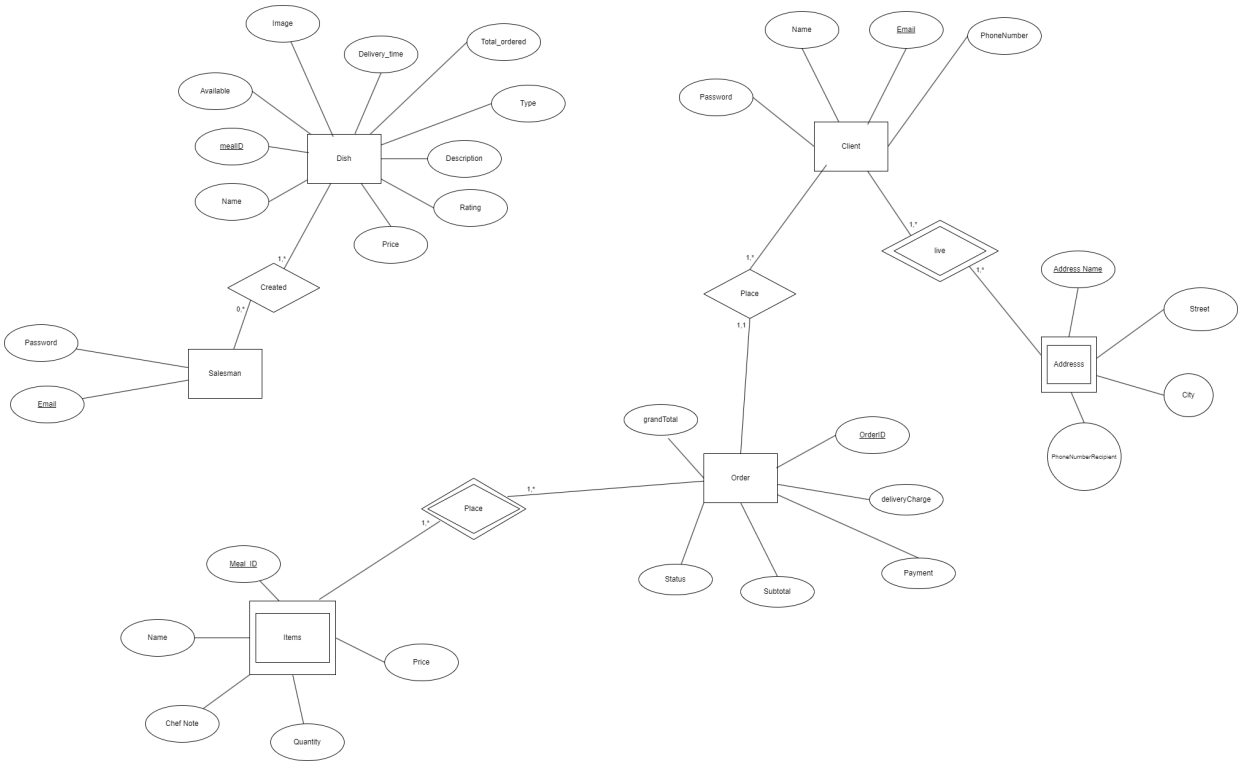
e. Deployment diagram



- Client PC/Laptop: This node represents the user's interface, where the "Main Page" component is accessed via "www.some-example.com". It's connected to a "User Browser Station", indicating that users interact with the system through a web browser.
- Application Server: This central node hosts several components that provide different functionalities:
  - `Check Order`: Allows users to modify information like addresses and payment method before checkout
  - `Report`: Likely generates reports for users or admins.
  - `Edit Profile`: Enables users to update their profile details.
  - `User Authentication`: Manages user login and security.

    - `Menu Edit`: Allows Salesman to add new items to the menu or change information like price and description of the current item.
    - `Notification`: Send a Notification message to the Salesman and the Chef when the Customer place an Order.
    - `Update delivery status`: Allow the Chef and the Salesman to change the Order status. These status included: Preparing, Delivering, Delivered.
- Database Server: Stores and manages data in three collections:
    - `Clients`: Contains client information.
    - `Dish`: Holds all the dish of the Restaurant
    - `Orders`: Stores order details.
- Communication Protocols:
    - `HTTPS`: Secure communication between the Client PC/Laptop and the Application Server.
    - `TCP/IP`, `Message`: Protocols used between the Application Server and the Database Server for data transfer.

f. Database Diagrams



1. The entitiy Client:

- The Entity represent an record Client in the Client database.
- This stores Client personal Information like Name, PhoneNumber, Password,....
- Each Clients can have more than 1 addresses.
- Each Clients can place more than 1 order.

2. The weak entity Address

- Each record in the Address datanse is represented by the entity Address
- This database stores 1 address for each record and will be mapped to the corressponding Clients
- 1 Addresses can belong to many Clients.
- Each Addresses are only unique if they are from the same order.

3. The entity Order:

- Each record in the Order database is represented by the entity Order.
- Each Order will have it subTotal, which is the total price without addictional fees.
- The Delivery fee, which is separated from the subTotal
- And the GrandTotal, which is the actual amount of money the Client has to pay for that order.
- Each Order can have a list of Items
- 1 Orders belongs to only 1 Client, and the Order can cotain that Client information as well.

4. The Weak Entity Items:

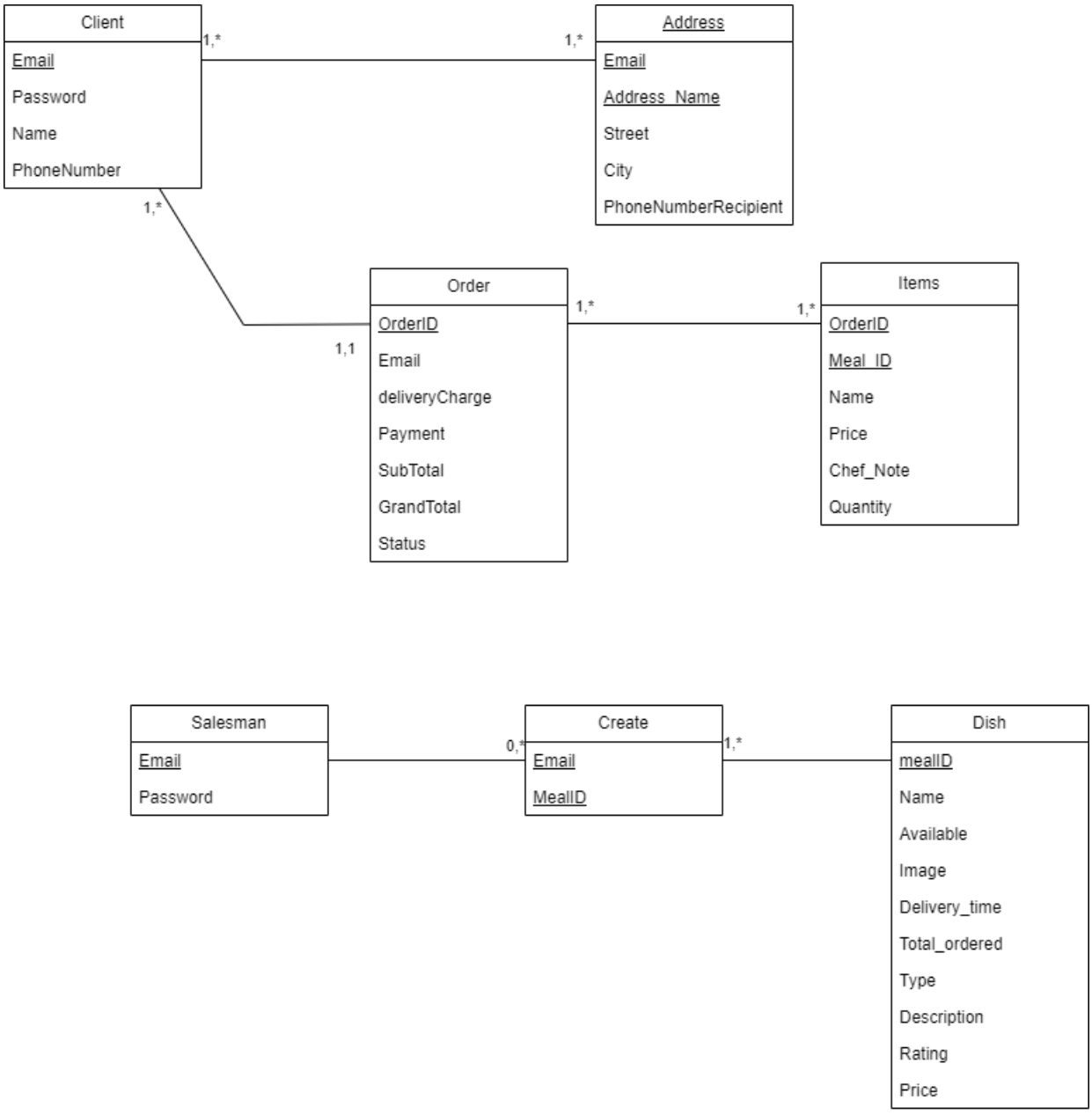
- Each record in the Items database is represented by the entity Items
- This stores the price of an Item multiply by it's quantity.
- 1 Item can be in many Orders.
- Each Item are only unique if they are from the same order.

5. The entity Dish:

- Each record in the Dish database will is represented like the entity Dish
- This database stores the DishID to differentiate different dishes
- The Salesman would have to fillin all of these information in the database when creating a new dish.

6. The entity Salesman:

- Each entity Salesman represent a record in the Salesman databased.
- Only the entities in the Salesman database can access the webpage as the Salesman
- This database also used to establish differences between a Client and a Salesman.
- The Salesman can created a Dish for the Menu.



- The table Client will stores all the Client information
    - Each Record in here represent a Client.
- The table Address will stores all the Client's addresses
    - Each Record in here represent a Address belong to a Client
- The table Order represent an Order that have been placed on our web
    - Each Record represent a Order that have been placed by the Client

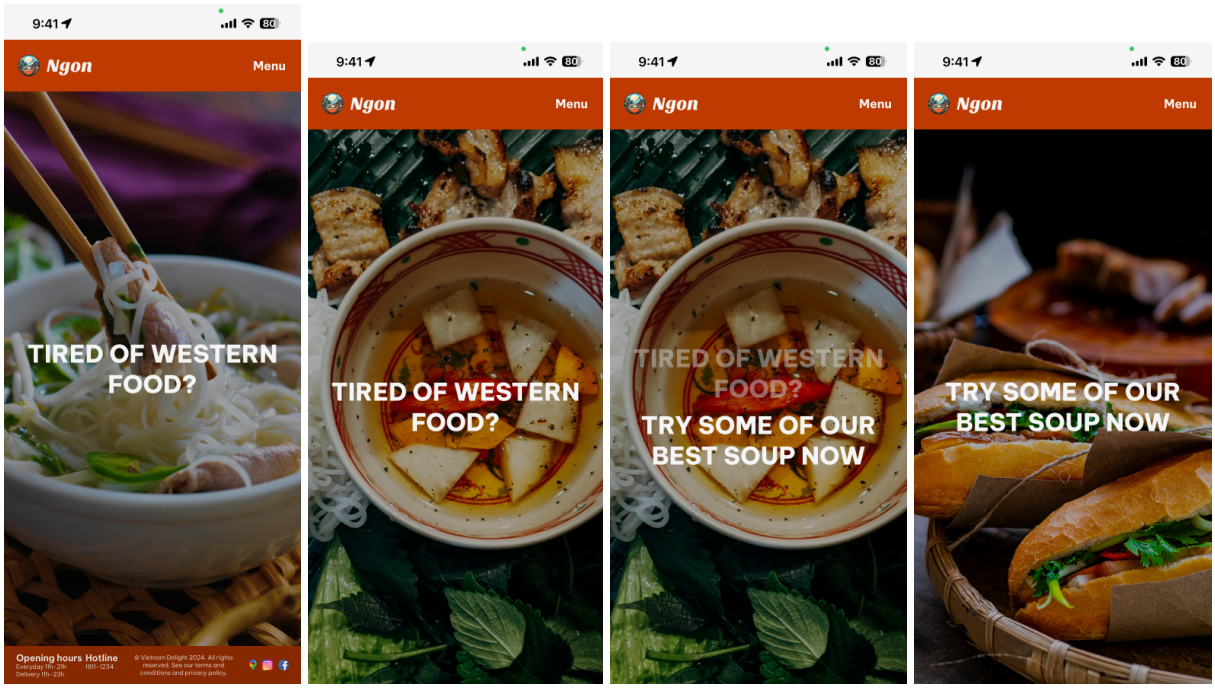
- The table Items will stores all the items of each order.
    - Each Record here shows which Item in which specific Order.
- The table Salesman represent a Salesman.
    - Each Record represent a Salesman, which are authorize to editMenu item and so on.
- The table Dish represent a Dish on the Menu.
    - Each Record represent a Dish.
- The table Create is the relationship table between Dish and Salesman.
    - Each Record represent which Salesman create that Dish.

IV. UI Mockup

- The website is diligently created, with a responsive style. This means that it intelligently adapts its user interface to the device from which it is accessible. Whether the Client is using a PC or a mobile phone, the website provides an excellent viewing experience adapted to the Client's device.

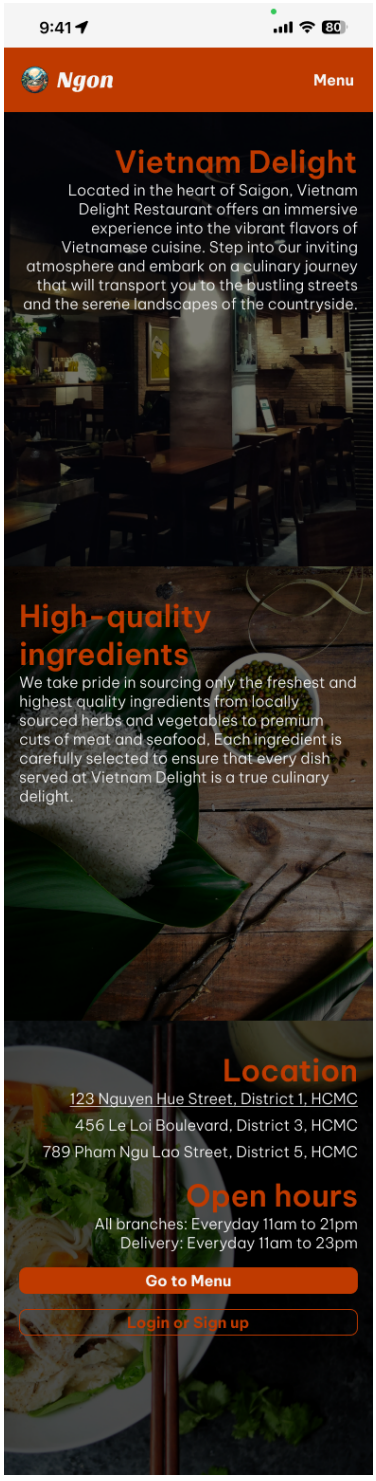
A. Mobile

1. OnBoard



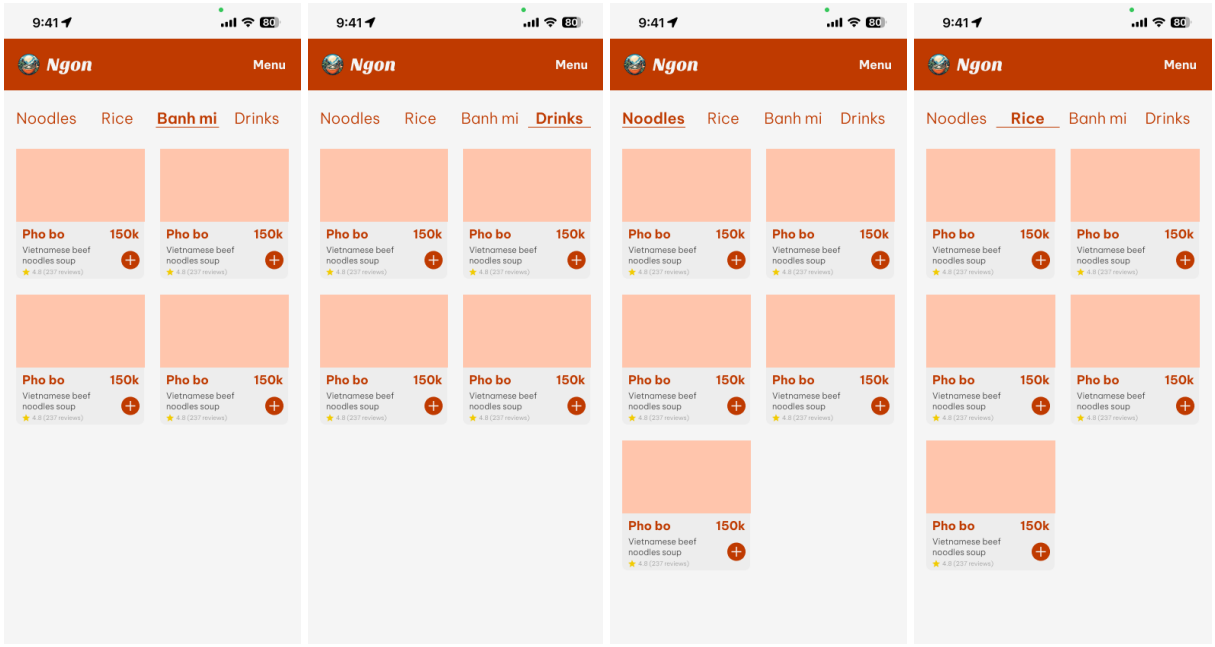
- From left to right, all of these images will be run on the screen.
- The footer will be displayed for all boarding
- The boarding will last 600 ms each
- After the boarding, it will display the Menu or the Client can just tap on the Menu at the top right corner.
- The footers social media and other information are all hyperlinks

2. Our Brand

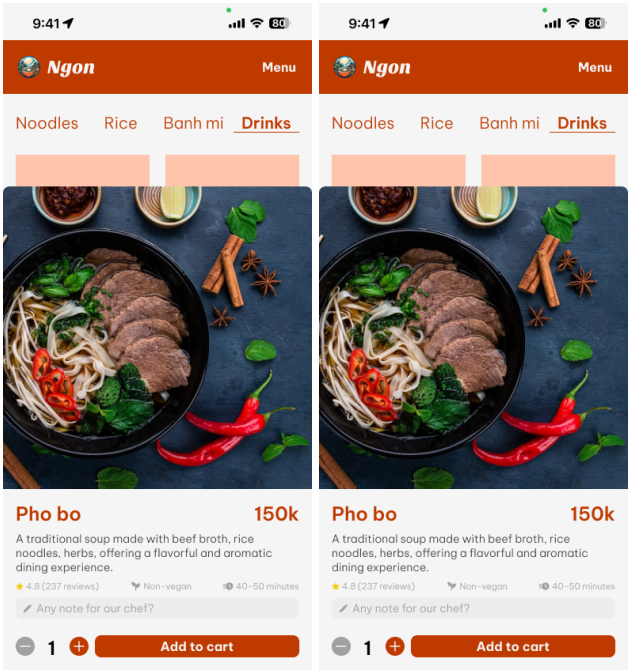


- When the Client click on our Brand, this UI will appear
- The Client can scroll down the moble device to see the rest of the page
- The address links are all hyperlinks
- The Client can click on the "**Go to menu button**" to see the menu
- The Client can click on the "**Login or Sign up button**" to go to the login section

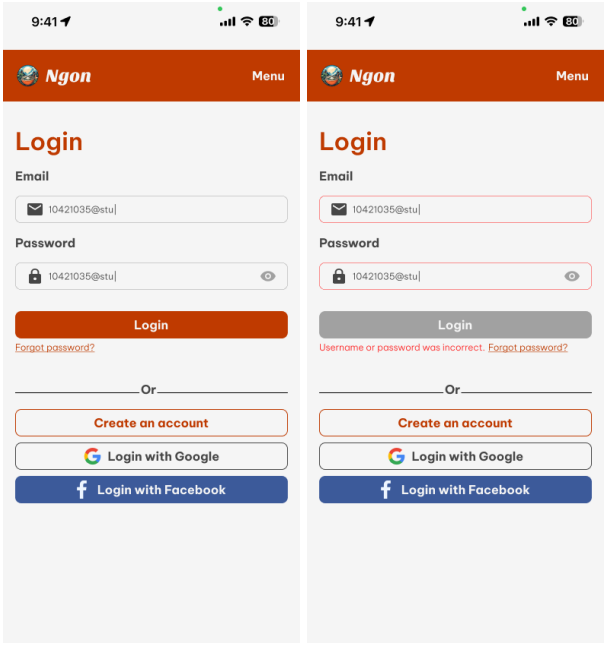
3. The Menu



- This will be the display of the menu for all the types of food that our website has to offered
- Each dish will have it's own price and a English-Vietnamese name in orange and a short english description in gray
- The Client can either click on the plus button to directly add that dish to the cart with quantity of 1.
- To edit the number of quantity, the Client can click on the picture instead, and this will open another UI
- This also allow the Client to add some notes to the chef as well as read a detail description of a dish.
- The bellow UIs are for when the Client click on the dish

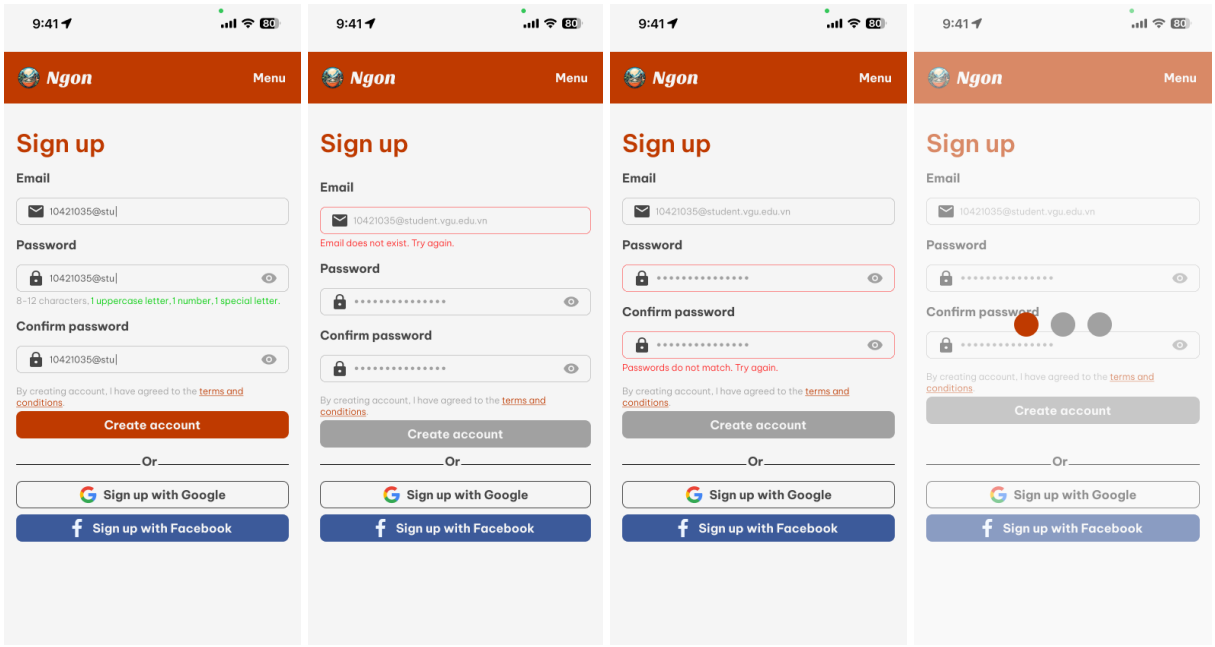


4. Login Section



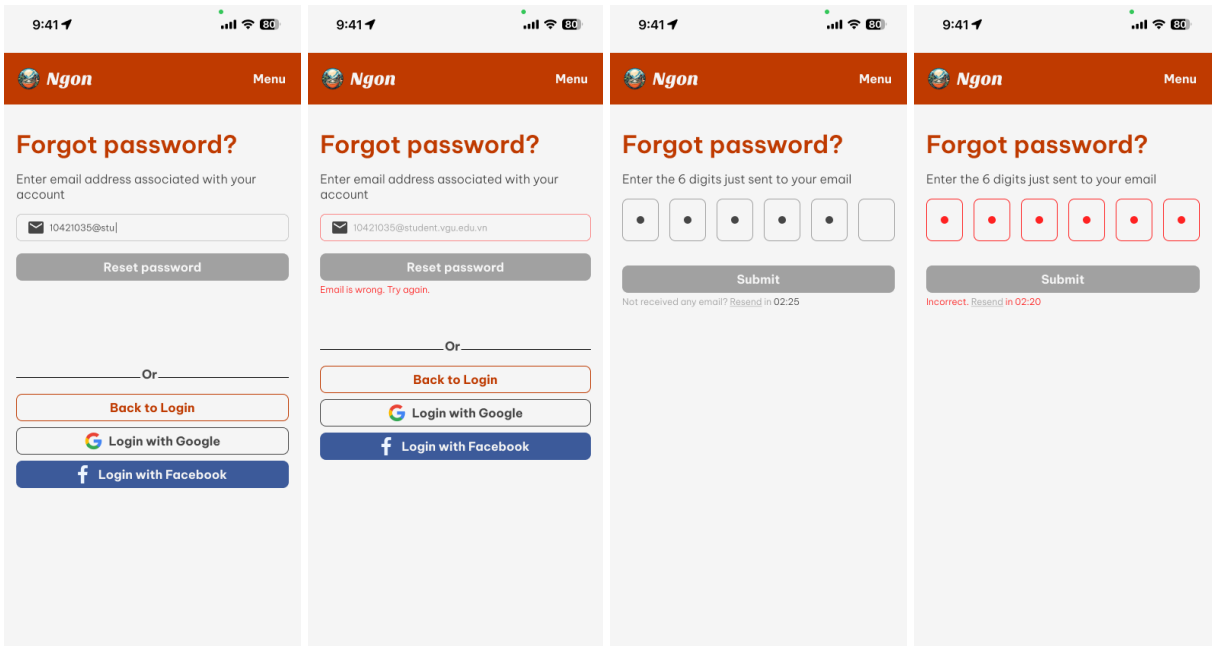
- The Client have to Login to before being able to use our website
- If the Client don't have an account, the Client can either '**Create an account**' which will open another sign up UI
- The Client can also login with their Facebook account or Google account
- The Client can click '**Forgot password**' which will open another UI for the forget password section
- An error notification will appear if the account does not exist in the database.

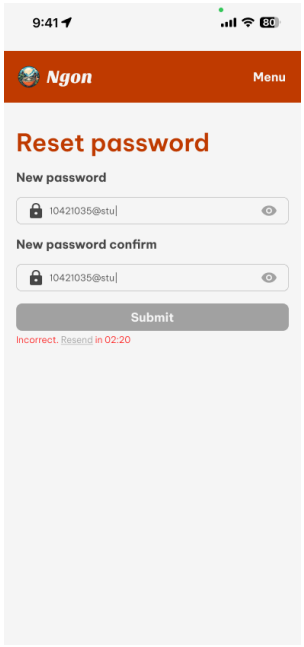
5. Sign up section



- The Client can sign up on our website with just an email address
- The password being created must statisfied some standard for security
- After that, the Client will receive an OTP for authentication.
- An invalid email address or password that don't statisfied our standard will return an error
- The Client can also choose to sign up with an Google account or an Facebook account.
- The website generates an error if the email address or password do not match.

6. Forget password section

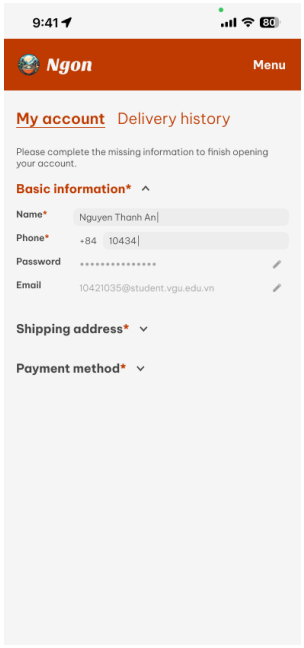




- The Client have to input the email address which the Client want to get back
- Then an OTP authentication will be send to the user email.
- After the Client have input the OTP, this will open another UI for the Client to reset the password.
- The Client can choose to go back to the login page or just login with Google or Facebook account
- The website will display an error if the provided email does not exist in our database, if the OTP sent to the email is not inputted correctly, or if the two new passwords entered do not match.

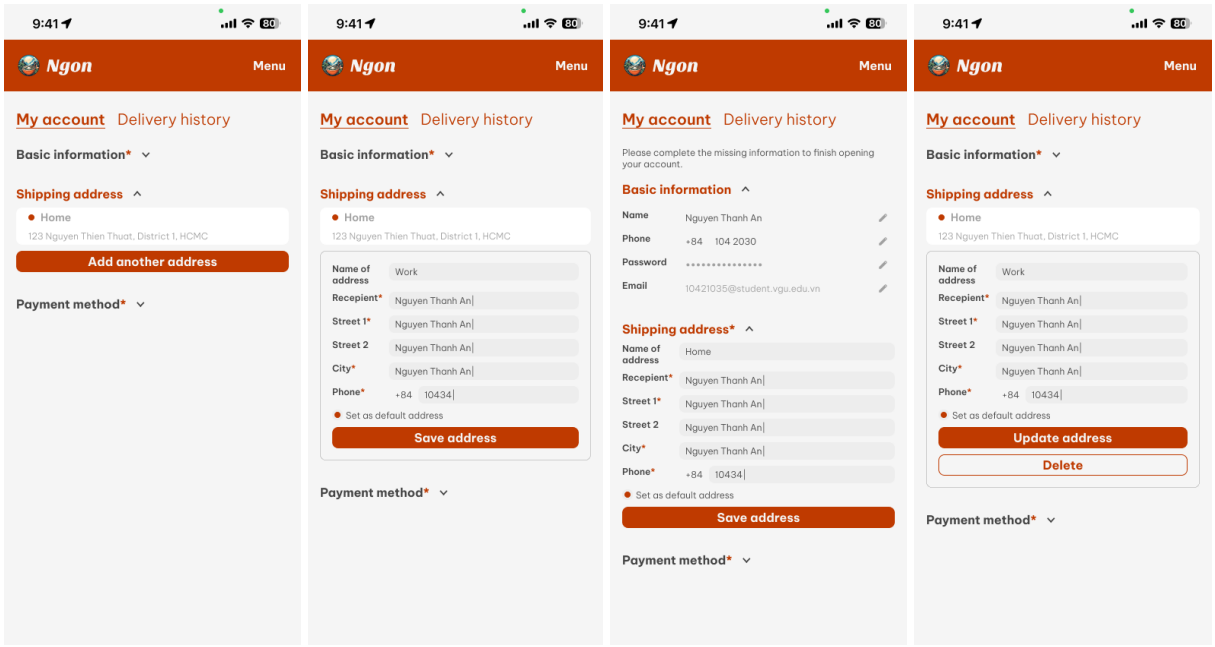
7. My Account section

a. Adding basic information



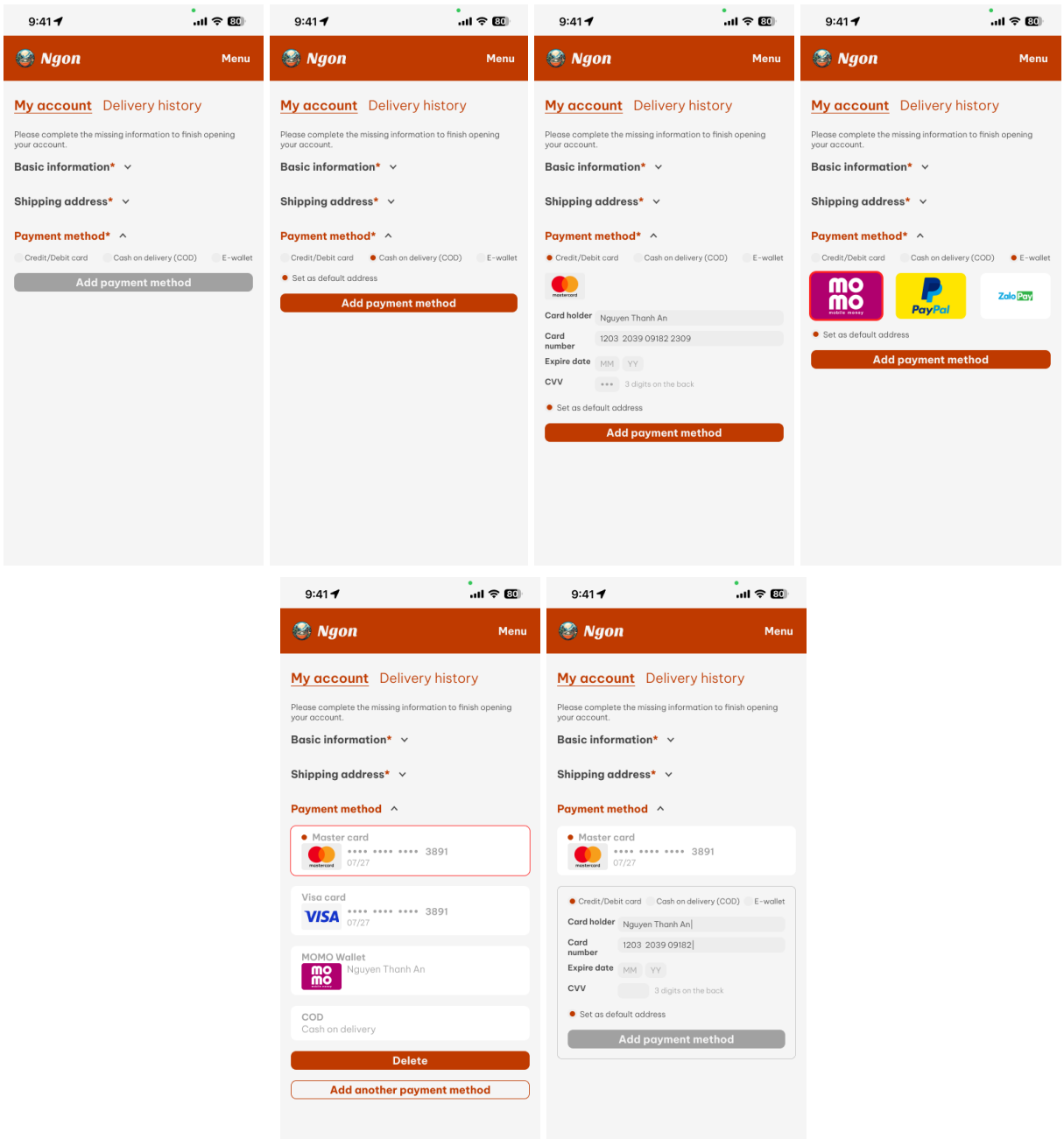
- A drop down panel UI will be used for My Account Section
- For the 'Basic Information', the Client can add in their name and their phonenumber as optional
- The Password and Email are compulsory.
- If the Client want to change password, it will automatically open to the Reset Password page (Forget password section).

b. Adding Shipping Addresses



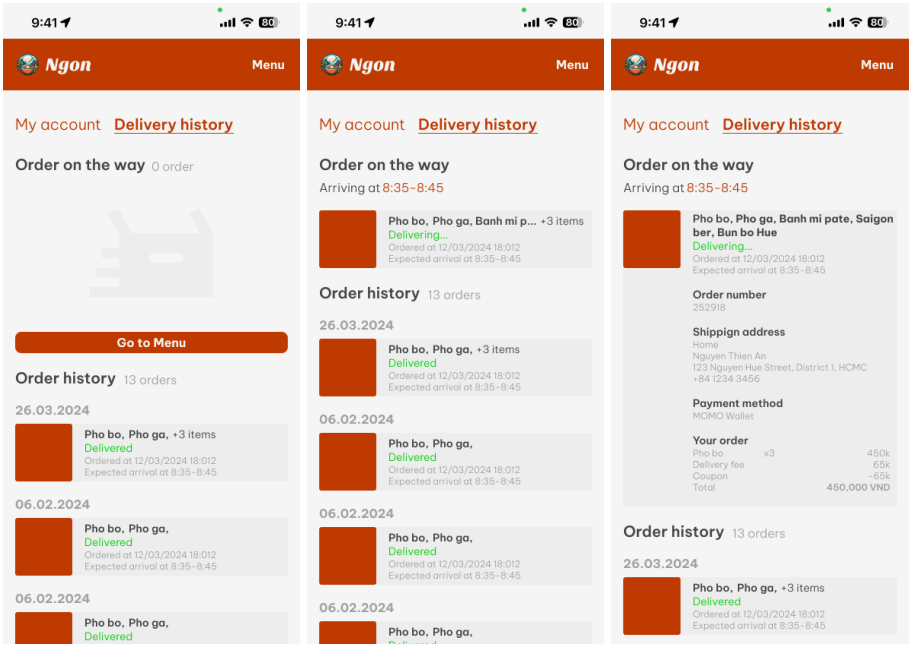
- For the "Shipping Address", the Client have to fillin all the necessary information to be able to add an address.
- The Client can add multiple addresses and can choose which addresses to be shipped to later at the Payment UI.
- The Client can also update or delete an exsisting address.

c. Adding Payment method



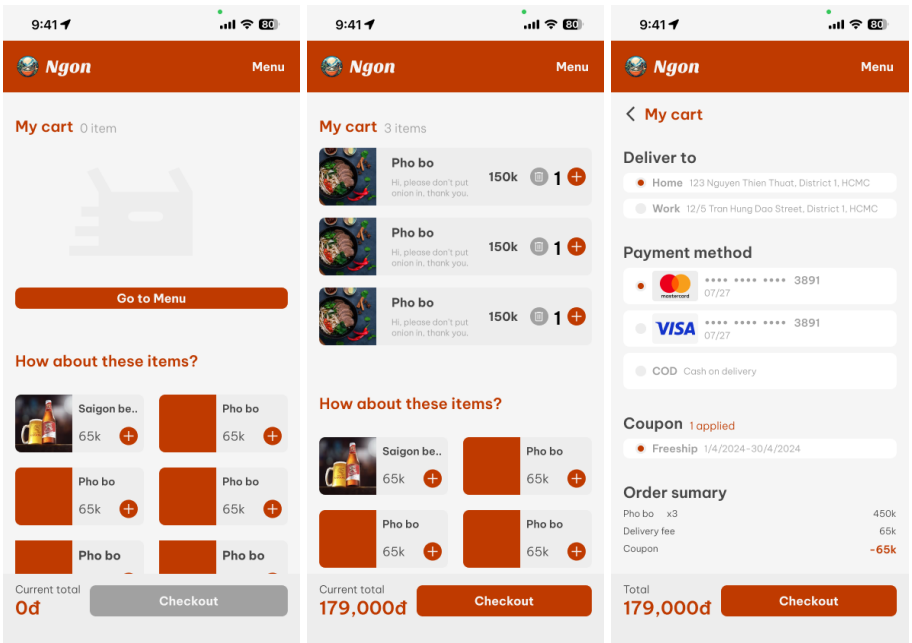
- The Client must also set a payment method as comppulsory
- The Client can set COD as default payment.
- The Client can also add Credit Card or EWallet as payment method as well
- When the Client add a Credit card, by inputing the card number, the website can automatically recognize it as a Visa or Mastercard.
- When the Client add a EWallet, by clicking on the logo, this will open the page to that EWallet for authorization
- The Client can add multiple payment method and can delete an exsisting payment method
- The Client can choose which payment method to use at the end of the Payement UI.

8. Order History section



- With this function, the Client can easily view their own previous orders.
- When click on any previous orders, the Client can see the price, payment method as well as the delivering address.
- This section also displays all the orders that are waiting for delivery.

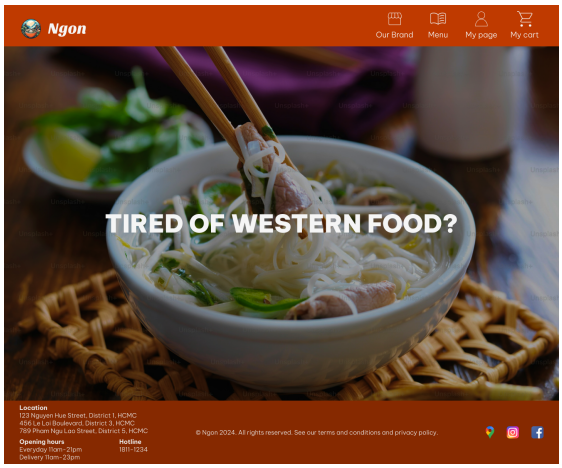
9. My Cart section

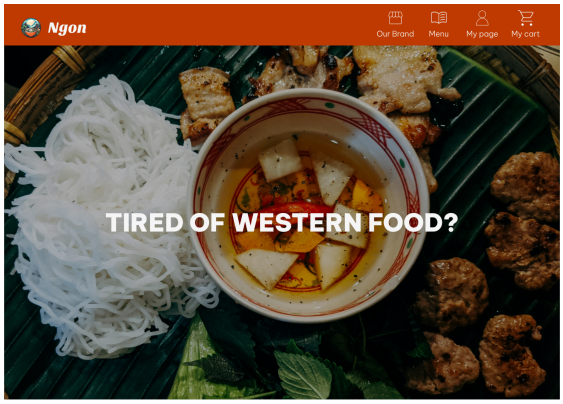


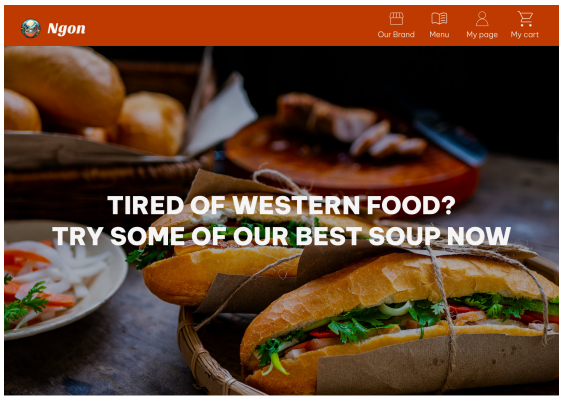
- When the Client add an item to the Cart, it will display in this UI
- This UI also shows recommendations.
- After the Client have choose all the desire dishes as well as setting the quantities for each dish, the Client can click on the "Check out" button
- The final step for an Order to be executed by the chef is to choose a delivering address as well as payment method. The Client can also see the order summary which includes the total prices as well as taxes and shipping fee.
- After Checkout successfully, the Client will automatically lead the the Order History.

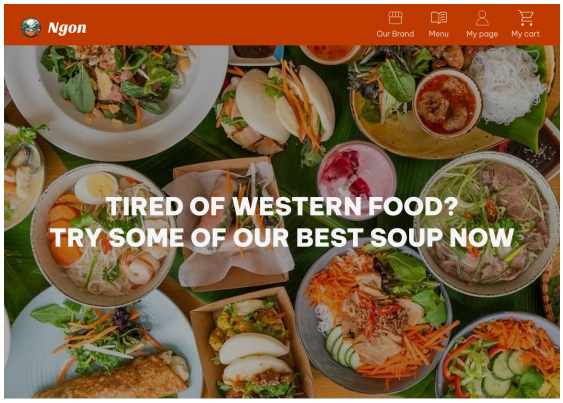
B. Website

1. On Boarding



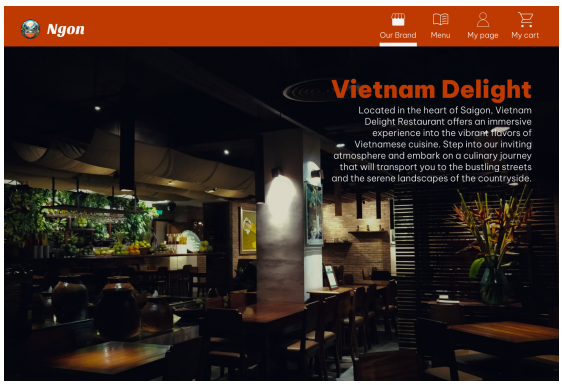


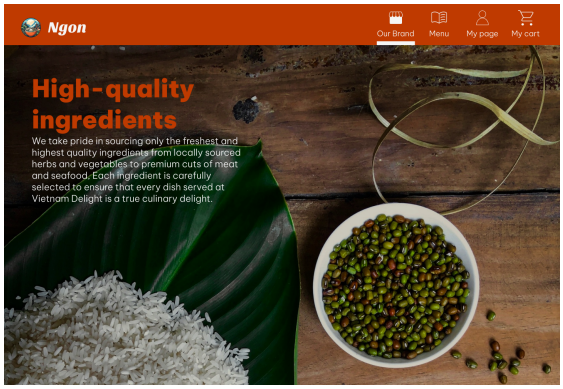


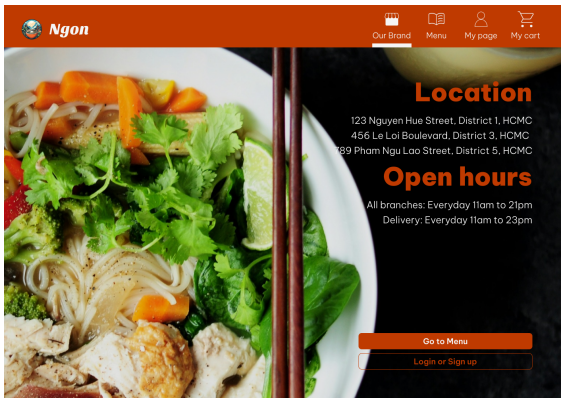


- Each image will be displayed from left to right on the screen, with the footer appearing consistently throughout.
- Each image will be shown for 600 milliseconds before transitioning to the next. Following the image sequence, the menu will be displayed, or the client can simply tap on the menu icon at the top right corner.
- The social media links and additional information in the footer will be clickable hyperlinks.

2. Our Brand

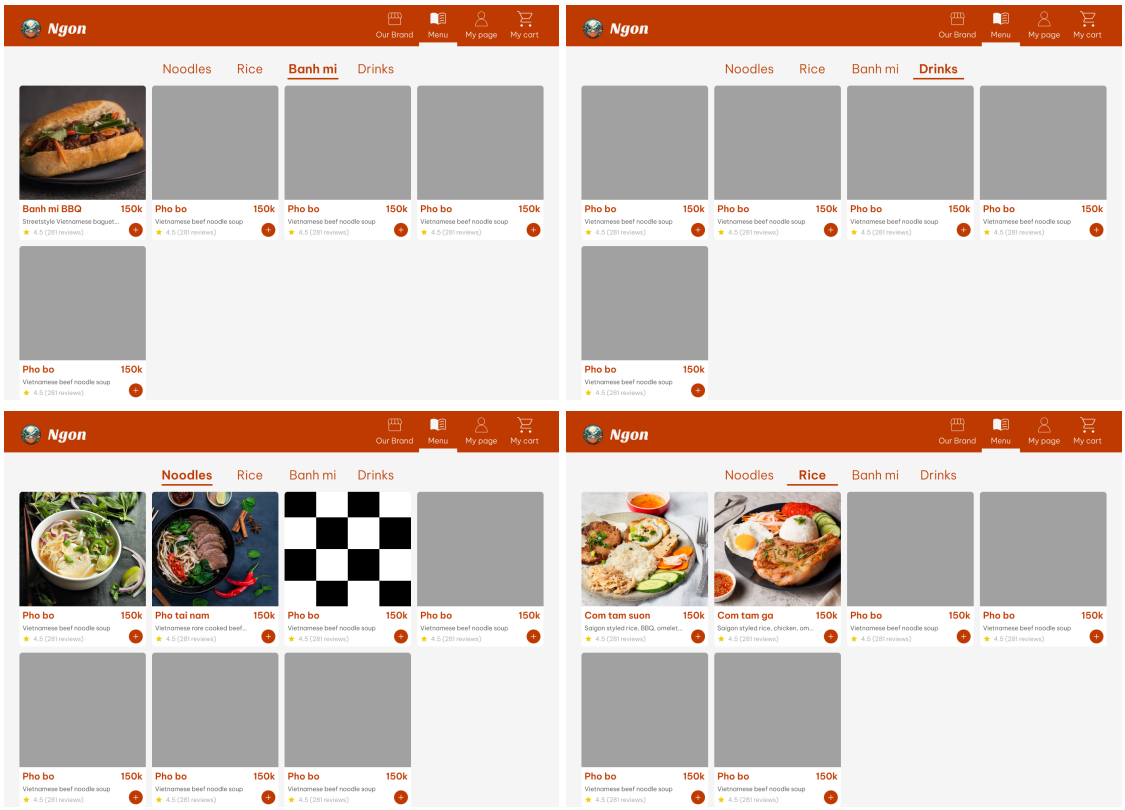




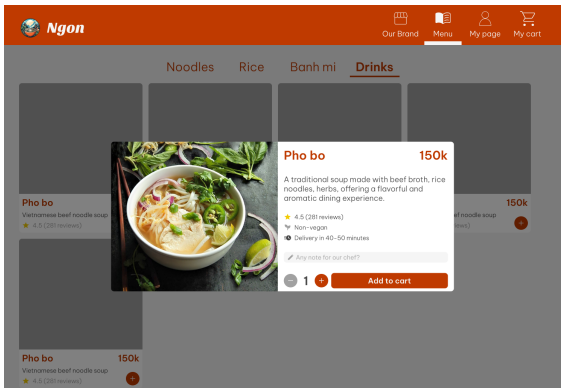


- When the Client scroll down, the page will change from 1 UI to another UI.
- The client has the option to click on the "Go to menu button" to access the menu. Likewise, clicking on the "Login or Sign up button" will direct them to the login section.
- Additionally, all address links are clickable hyperlinks

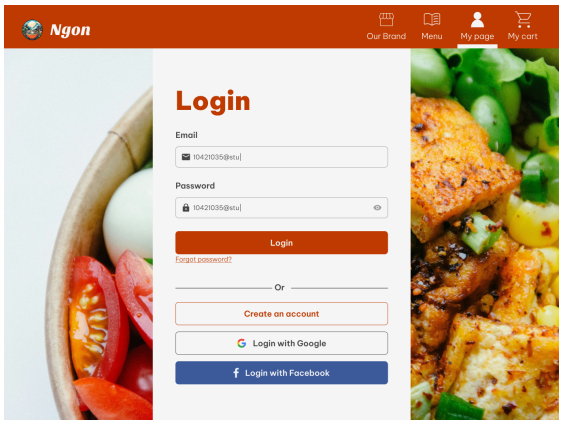
3. The Menu

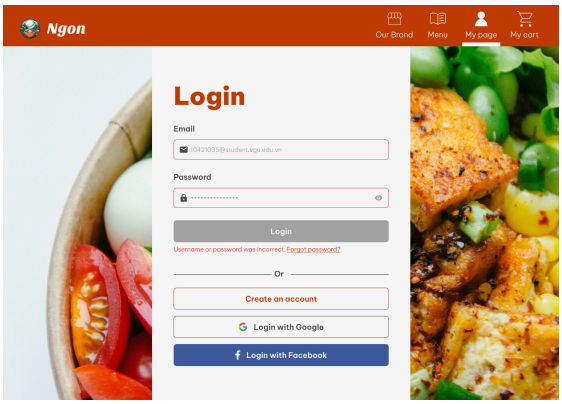


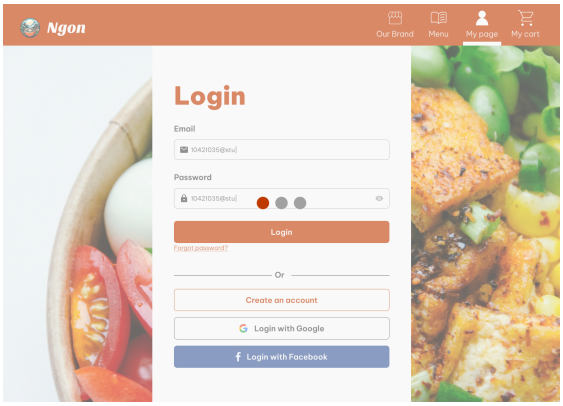
- This is how the menu will be displayed, showcasing all the types of food available on our website.
- Each dish will feature its own price, an English-Vietnamese name highlighted in orange, and a brief English description in gray.
- The client has the option to click on the plus button to instantly add the dish to their cart with a quantity of 1. Alternatively, they can click on the picture to open another UI where they can adjust the quantity and add notes for the chef.
- This UI also provides the option to read a detailed description of the dish.
- The following UIs are for when the client clicks on a specific dish.
- The Client can escape this view by clicking the outside of the dish.



4. Login Section

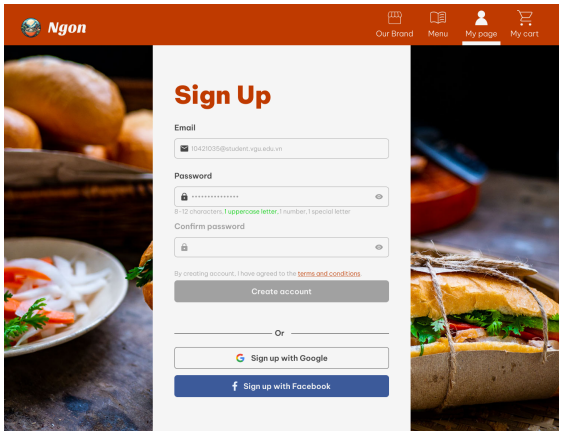


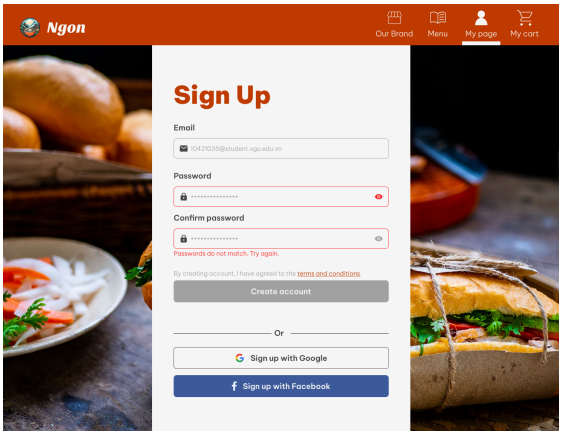


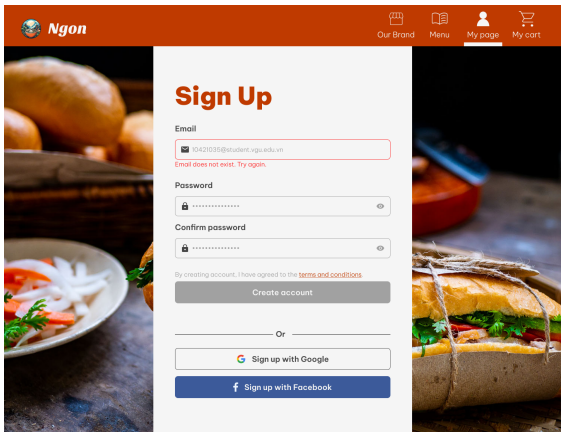


- Before accessing our website, the client must log in.
- If they don't have an account, they can choose to 'Create an account,' which will open a signup UI.
- Alternatively, they can log in using their Facebook or Google account.
- If the client forgets their password, they can click on 'Forgot password,' which will open a UI for the password recovery section.
- An Error notification will pop up when the account don't exsist in the database.

5. Sign up Section

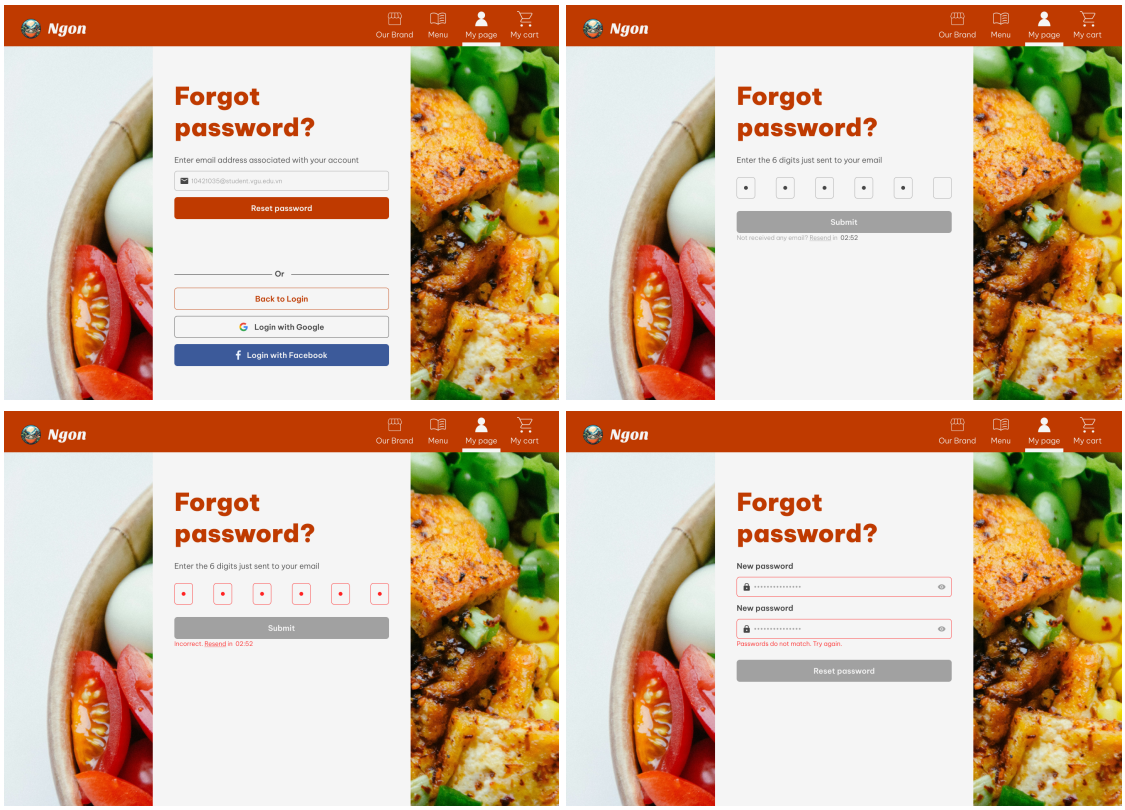


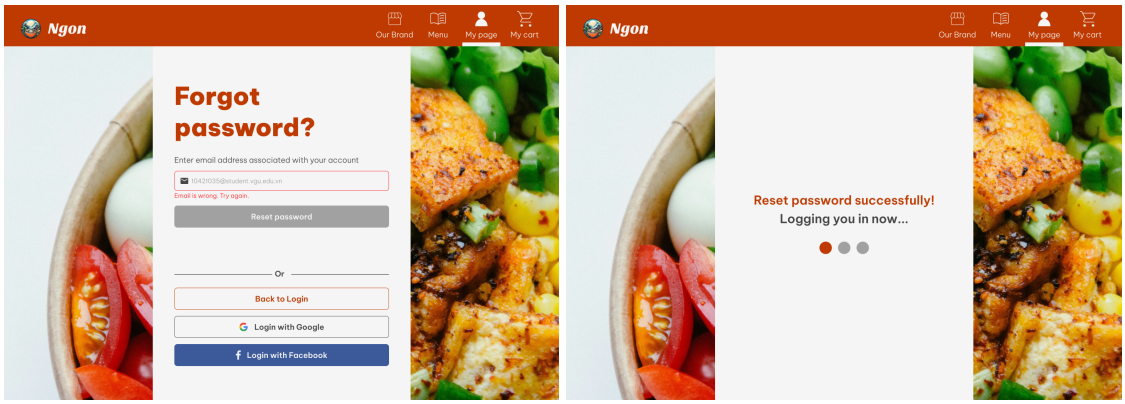




- Clients can easily sign up on our website using just their email address.
- However, the password they create must meet certain security standards.
- The Client will then recevie an OTP for authentication.
- If the email address or password provided does not meet these standards, an error message will be displayed.
- Additionally, clients have the option to sign up using their Google or Facebook accounts.
- The website will return an error when the email doesn't exist or when the confrimation password don't match.

6. Forget Password Section

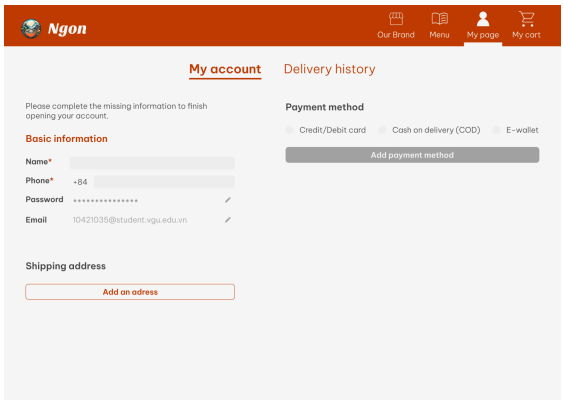




- The client needs to input the email address they wish to receive the reset instructions.
- Subsequently, an OTP authentication will be sent to the user's email.
- Once the client has input the OTP, another UI will open for them to reset the password.
- From there, the client can choose to return to the login page or directly log in using their Google or Facebook account.
- The website will return an error if the Email doesn't exist in our database, when the OTP sent to the email is not inputed correctly or when the 2 new passwords don't match.

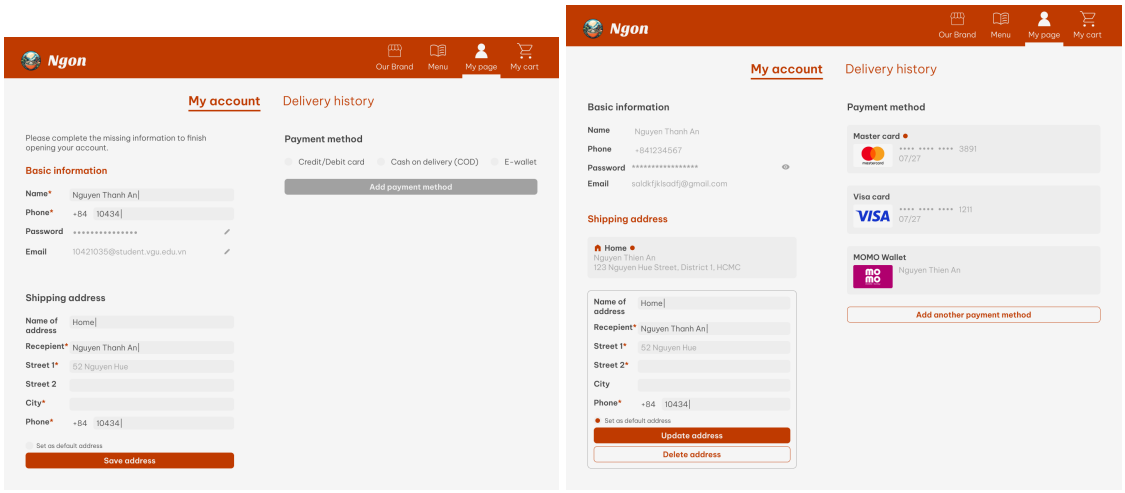
7. My Account Section

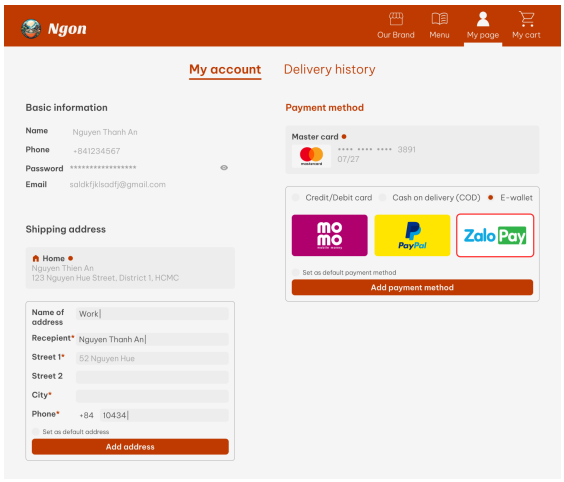
a. Adding basic information



- In the 'Basic Information' section, clients can optionally add their name and phone number.
- However, providing a password and email is mandatory.
- If the client wishes to change their password, the system will automatically redirect them to the Reset Password page. (Forget password section)

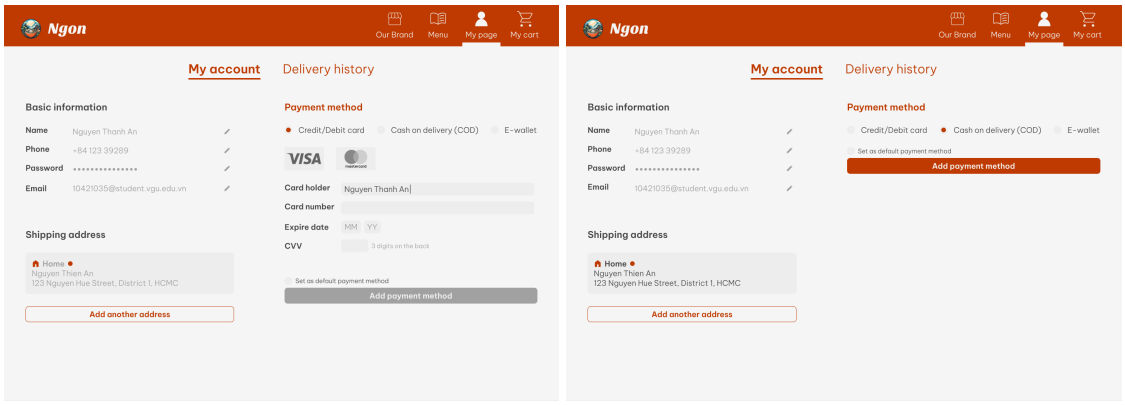
b. Adding Shipping Addresses

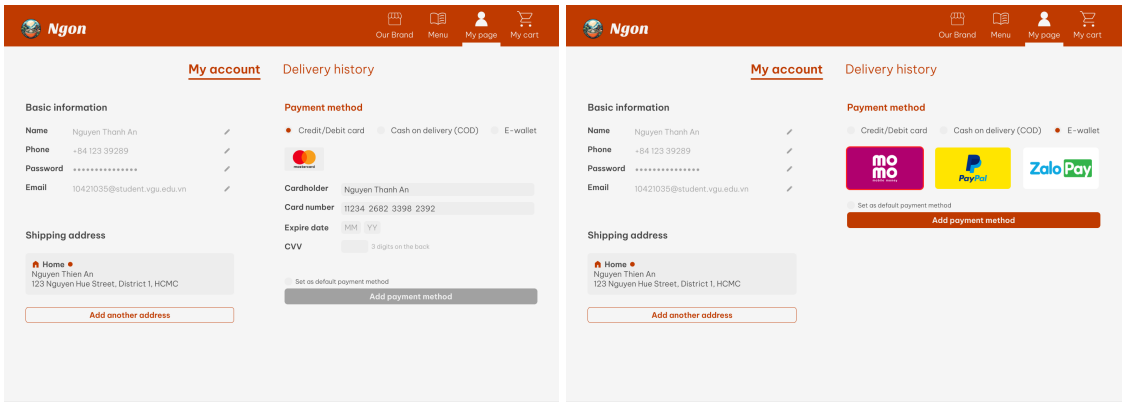


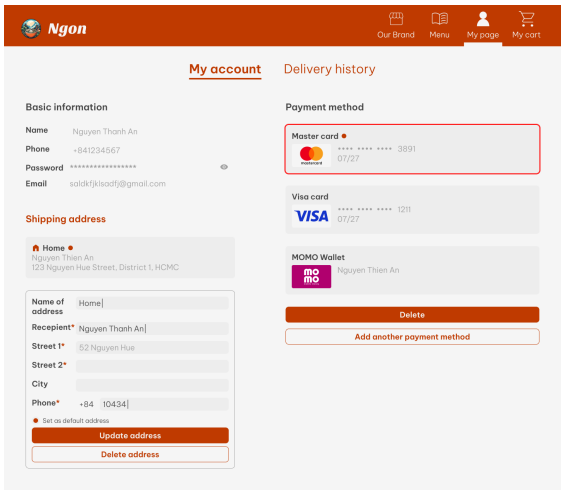


- In the "Shipping Address" section, clients must provide all necessary information to add an address successfully.
- They can add multiple addresses and choose which one to ship to later during the Payment UI stage.
- Additionally, clients have the option to update or delete existing addresses as needed.

c. Adding Payment method

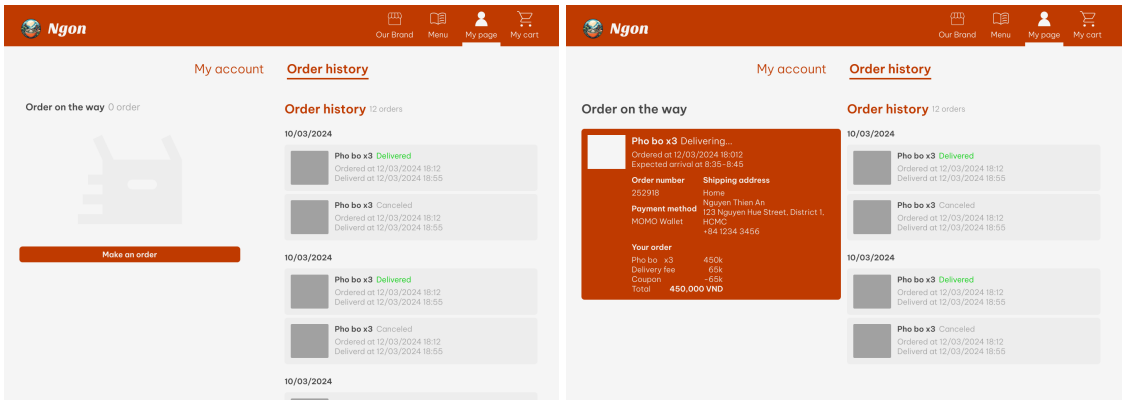


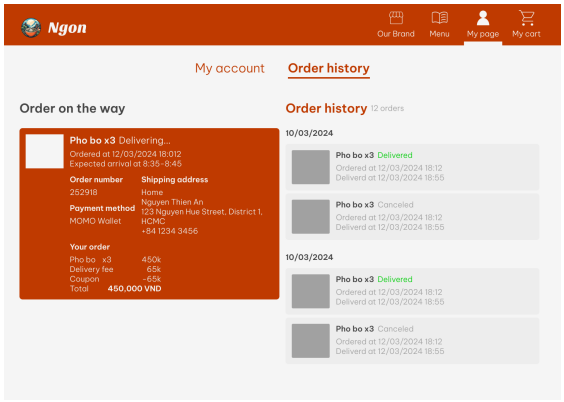




- Setting a payment method is mandatory for clients.
- They can choose Cash on Delivery (COD) as the default payment method.
- Alternatively, they can add Credit Card or EWallet as payment methods.
- When adding a Credit Card, the website can automatically recognize whether it is a Visa or Mastercard based on the inputted card number.
- For EWallets, clicking on the logo will redirect the client to the respective EWallet page for authorization.
- Clients can add multiple payment methods and delete existing ones.
- During the Payment UI stage, clients can select their preferred payment method for the transaction.

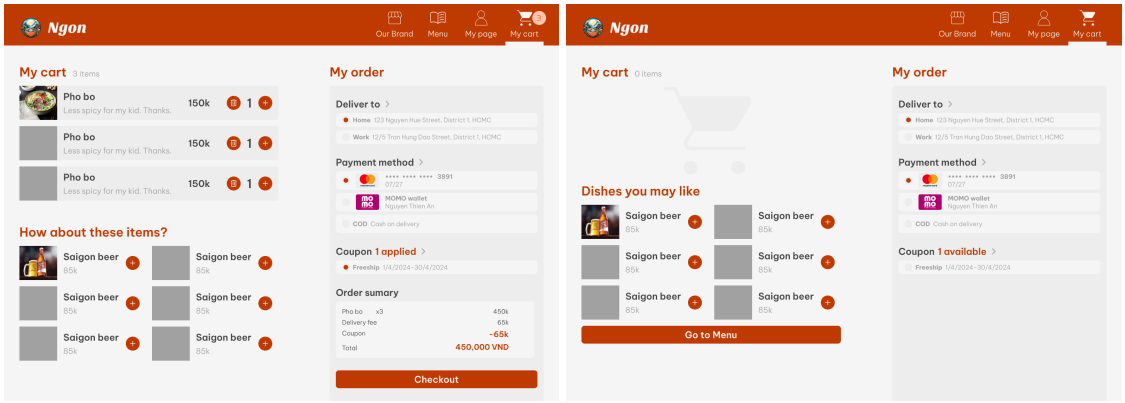
8. Order History Section

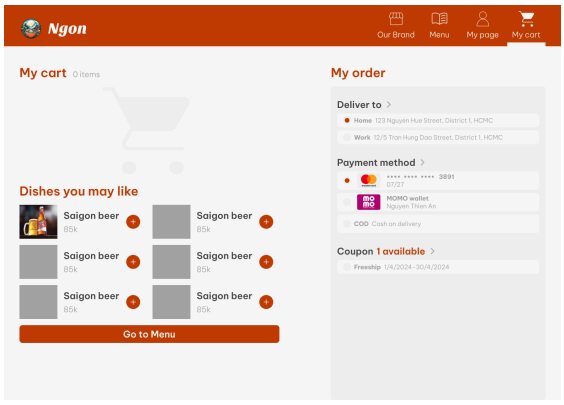




- This feature allows clients to conveniently view their past orders.
- Clicking on any previous order reveals details such as the price, payment method, and delivery address.
- Additionally, this section displays all orders awaiting delivery.

9. My Cart Section





- When the client adds an item to the Cart, it will be displayed in this UI.
- Additionally, the UI shows recommendations.

- Once the client has selected all desired dishes, add some chef notes if wanted and set the quantities for each dish, they can proceed by clicking on the "Check out" button.
- The final step before the order is executed by the chef involves choosing a delivery address and payment method.
- The client can also view the order summary, which includes the total prices, taxes, and shipping fees. Upon successful checkout, the client will be automatically redirected to the Order History page.

V. Back-end coding

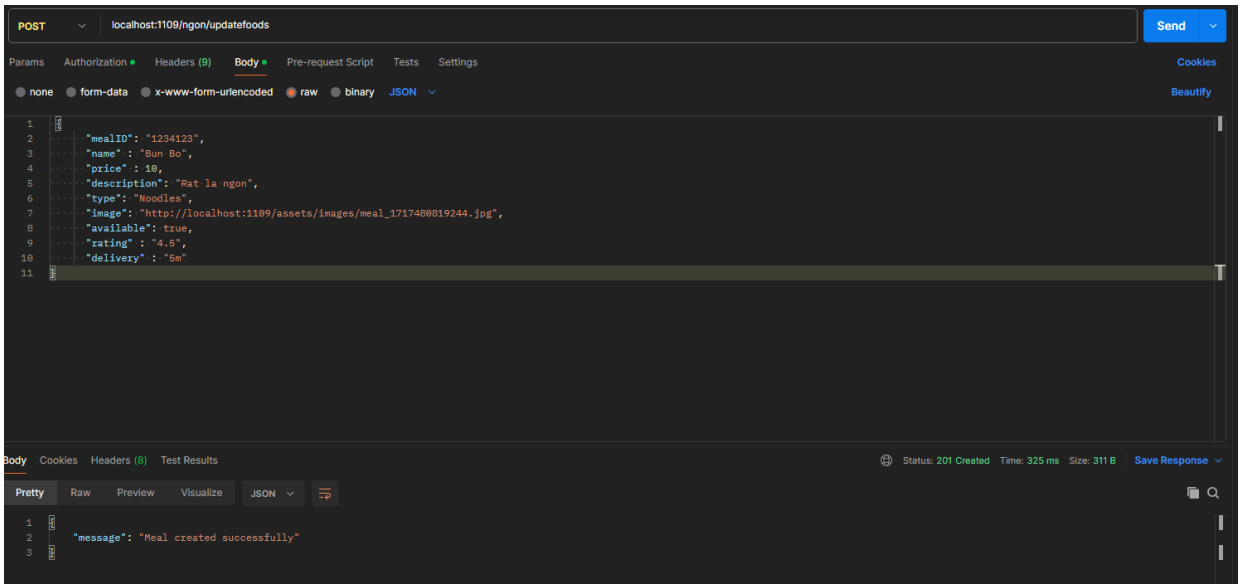
Menu Edit Component

1. Add and Update exsiting Meal

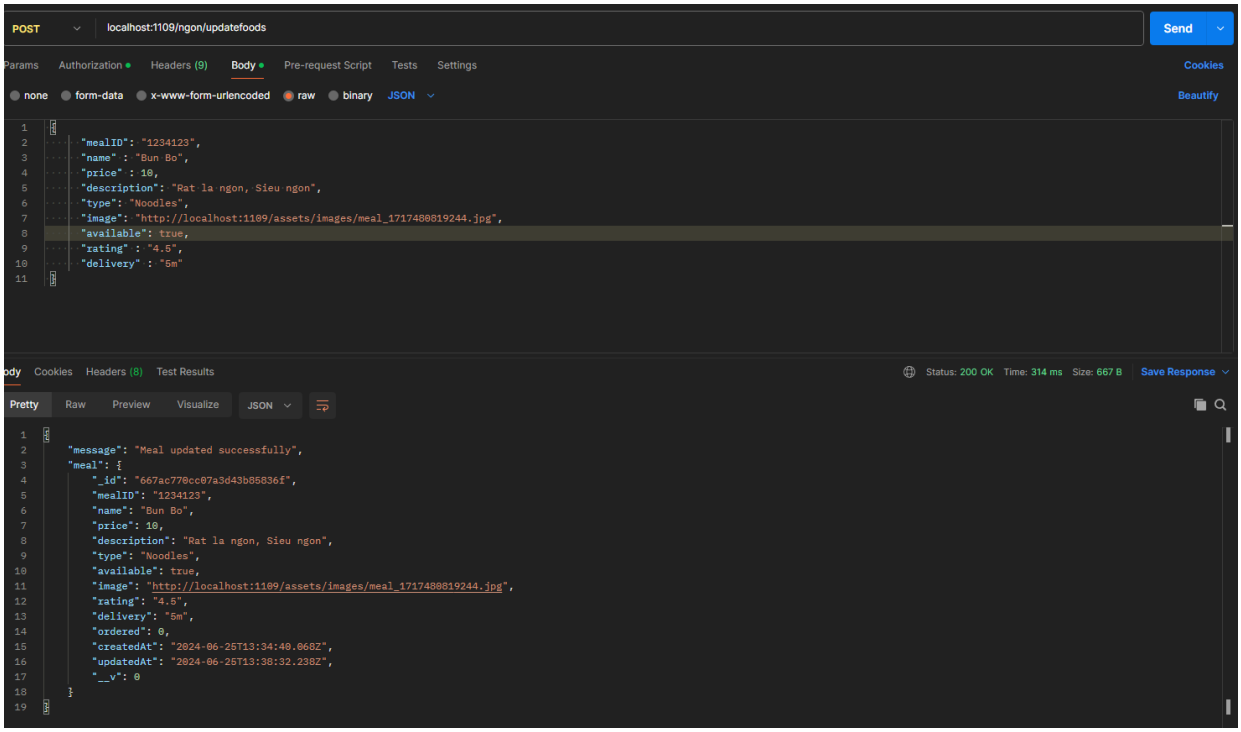
```
module.exports = async (req, res) => {
    try {
        let meal = await Meals.findOne({ mealID: req.body.mealID });

        if (meal) {
            meal = await Meals.findOneAndUpdate({ mealID: req.body.mealID
}, req.body, { new: true });
            return res.status(200).send({ message: 'Meal updated
successfully', meal });
        }

        meal = new Meals(req.body);
        await meal.save();
        res.status(201).send({ message: 'Meal created successfully' });

    } catch (error) {
        console.log(error);
        res.status(400).send(error);
    }
}
```

This function from the application server is used to add meals to the database.

- First, it will find a meal in the ‘Meals’ collection with the ‘mealID’ provided in the request body.
- Next, if a meal exists, it proceeds to update that meal. It updates the existing meal with the data from the request body and returns the updated document to ensure that updated document is returned.
- After that, it will send a response code 200 indicating success, including the message “Meal updated successfully”.
- Otherwise, if no meal is found, it will proceed to create one with data from the request body and save them in the database.
- Finally, it sends a 201 status code, showing that a new resource has been added with “Meal created successfully”.



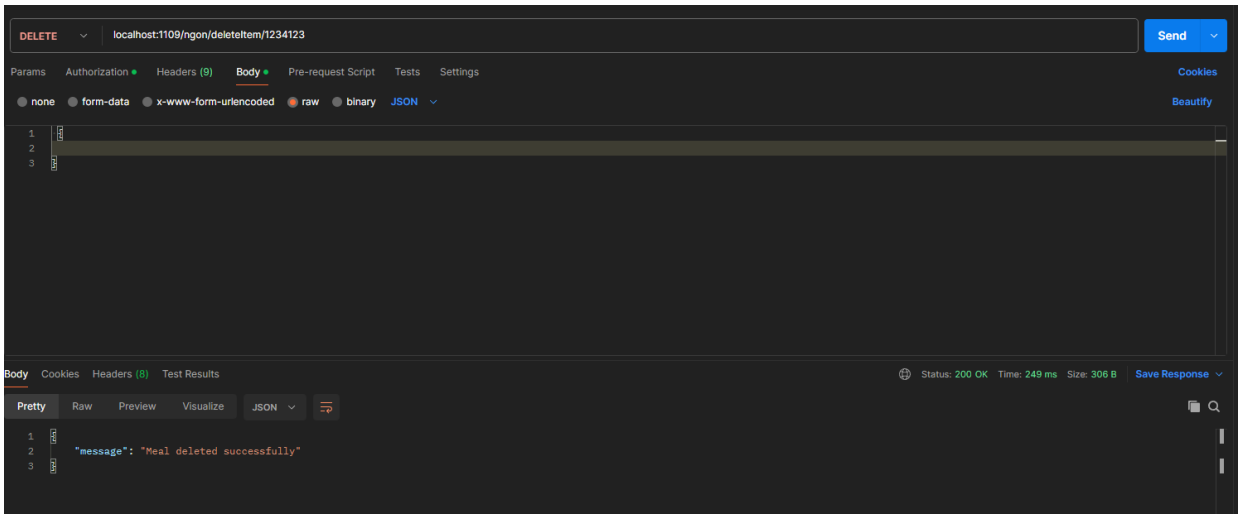
- The below is the same test API for update the existing meal, changing the description from "Rat la ngon" to “Rat la ngon, Sieu ngon”



2. Delete Meal

```
module.exports = async (req, res) => {
    try {
        const meal = await Meals.findOneAndDelete({ mealID:
req.params.mealID });

        if (!meal) {
            return res.status(404).send({ message: "Meal not found" });
        } else {
            res.status(200).send({ message: "Meal deleted successfully" });
        }

    } catch (error) {
        res.status(500).send(error);
    }
}
```

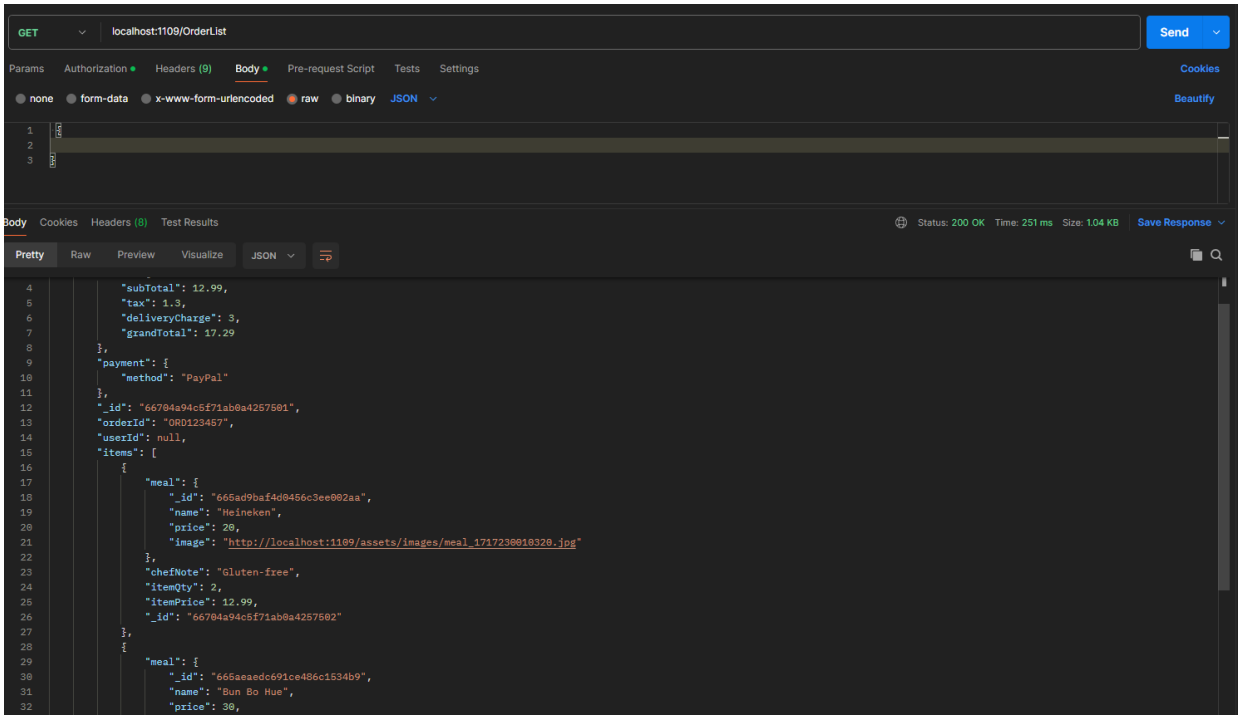
- The function first finds and deletes a specific meal from the 'Meals' collection with the given 'mealID'.
    - If found, it is then deleted and assigned to the 'meal' variable.
    - If no meal is found, a 404 status code will be displayed.
    - Otherwise, the response code 200 will be shown, indicating "Meal deleted successfully".
- Following is the test API for delete a recent added meal with ID 1234123:



Check Order Component

```
module.exports = async (req, res) => {
  try {
    const orders = await Order.find()
      .populate('userId', 'name phone')
      .populate('items.meal', 'name price image') // Populate meal details
      .exec();
    res.json(orders);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};
```

- In Check Order, it will first fetch all orders from the 'Order' collection.
- For each order, it will write name and phone number into
  - 'userId' field,
  - a group of name,
  - price and
  - image from 'Meals' collection will be written into 'meal' field within 'items' array.
- Next, the populated orders will be sent as a JSON response.
- The under API test if it listed order correctly

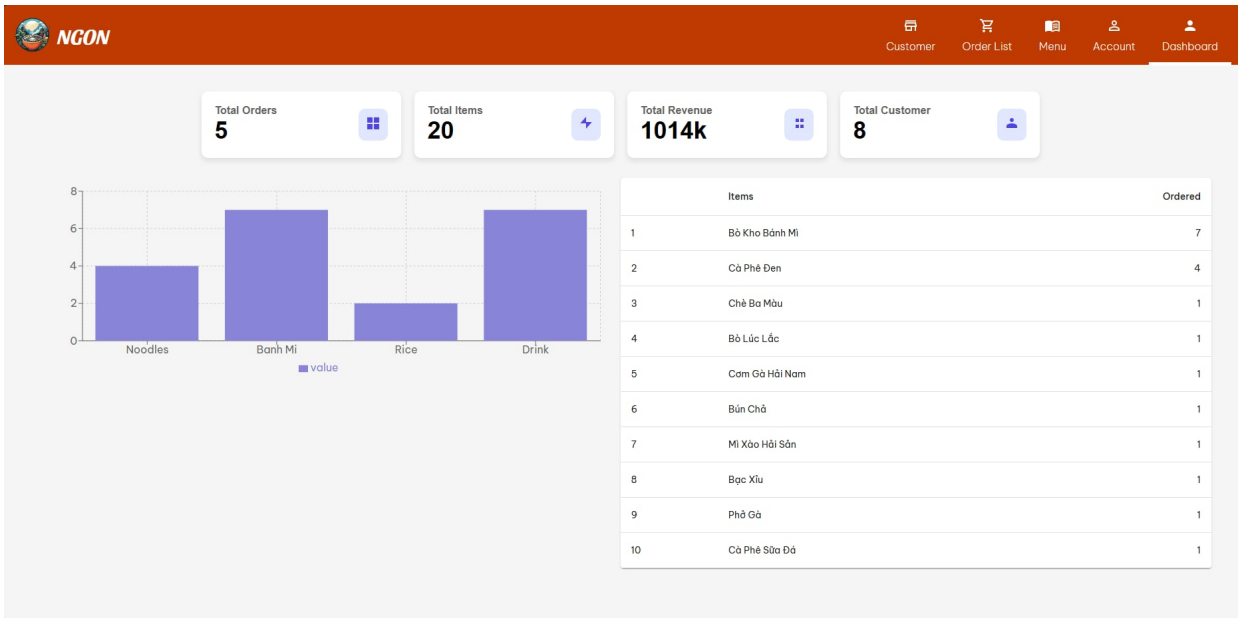


Report Component

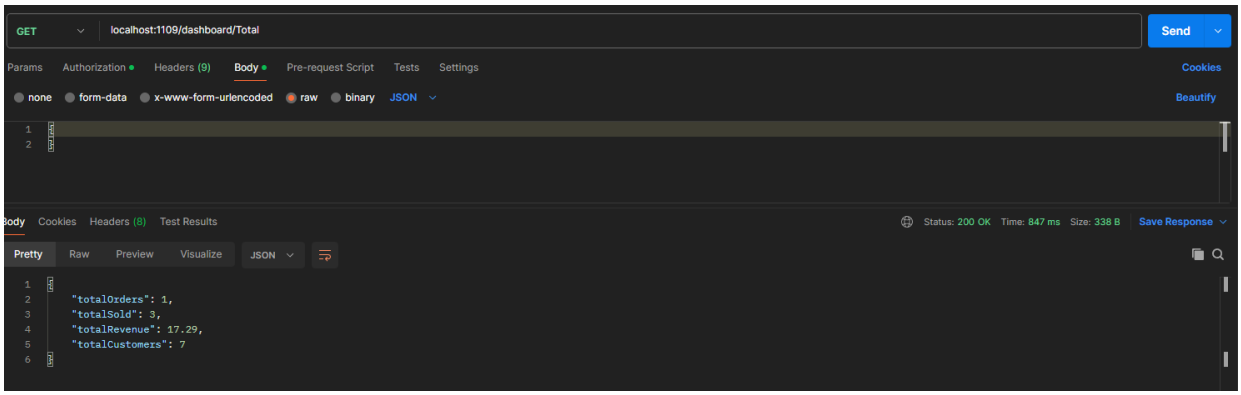
1. General Report Generating

```
router.get('/dashboard/total', authAdmin, async (req, res) => {
  try {
    const totalOrders = await Order.countDocuments();
    const totalSold = await Order.aggregate([
      {
        $unwind: '$items',
      },
      {
        $group: { _id: null, total: { $sum: '$items.itemQty' } },
      },
    ]);

    const totalRevenue = await Order.aggregate([
      {
        $group: { _id: null, total: { $sum: '$bill.grandTotal' } },
      },
    ]);

    const totalCustomers = await User.countDocuments();
    res.json({
      totalOrders,
      totalSold: totalSold[0].total,
      totalRevenue: totalRevenue[0].total,
      totalCustomers,
    });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});
```

- The 'Total' dashboard first counts the total number of orders.
- Secondly, it calculates the total number of items sold by aggregating over the Order collection, unwinding the items array, and summing up the itemQty field.
- Thirdly, it calculates the total revenue by aggregating over the Order collection again, this time summing up the grandTotal field from the bill of each order.
- Fourthly, it counts the total number of customers by counting documents in the User collection.
- The results of these operations are then sent back to the client as a JSON object containing the total number of orders, total items sold, total revenue, and total number of customers.
- If an error occurs during any of these operations, the catch block logs the error and responds with a 500 status code and a JSON object containing an error message, indicating a server error.
- And here is the UI dashboard:



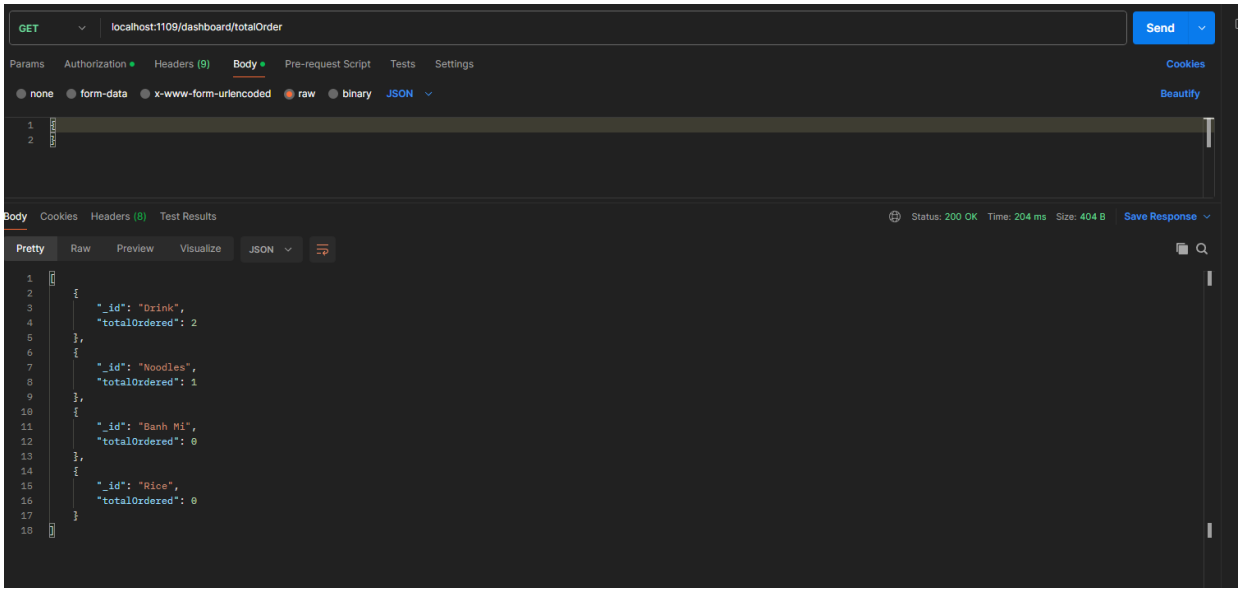
- The following API test if the Total dashboard calculate correctly:



2. Total Order of each Type of food

```
router.get('/dashboard/totalOrder', authAdmin, async (req, res) => {
  try {
    const result = await Meals.aggregate([
      {
        $match: {
          type: { $in: ['Noodles', 'Drink', 'Banh Mi', 'Rice'] },
        },
      },
      {
        $group: {
          _id: '$type',
          totalOrdered: { $sum: '$ordered' },
        },
      },
    ]);

    res.status(200).send(result);
  } catch (err) {
    res.status(500).send({ message: err.message });
  }
});
```

- In this function, **$match** is used to filters the documents in the Meals collection to include only those where the type field matches one of the specified sample values:
    - "Noodles",
    - "Drink",
    - "Banh Mi", or
    - "Rice".
- In the second stage, **$group**, it groups the filtered documents by their type and calculates the total number of orders for each type by summing up the values in the ordered field.
- The result of this aggregation—a list of document objects where each object represents a meal type and its corresponding total orders—is then sent back to the client with a 200 HTTP status code.
- If an error occurs during this process, the catch block captures the error and sends back a 500 HTTP status code along with the error message, indicating an internal server error.

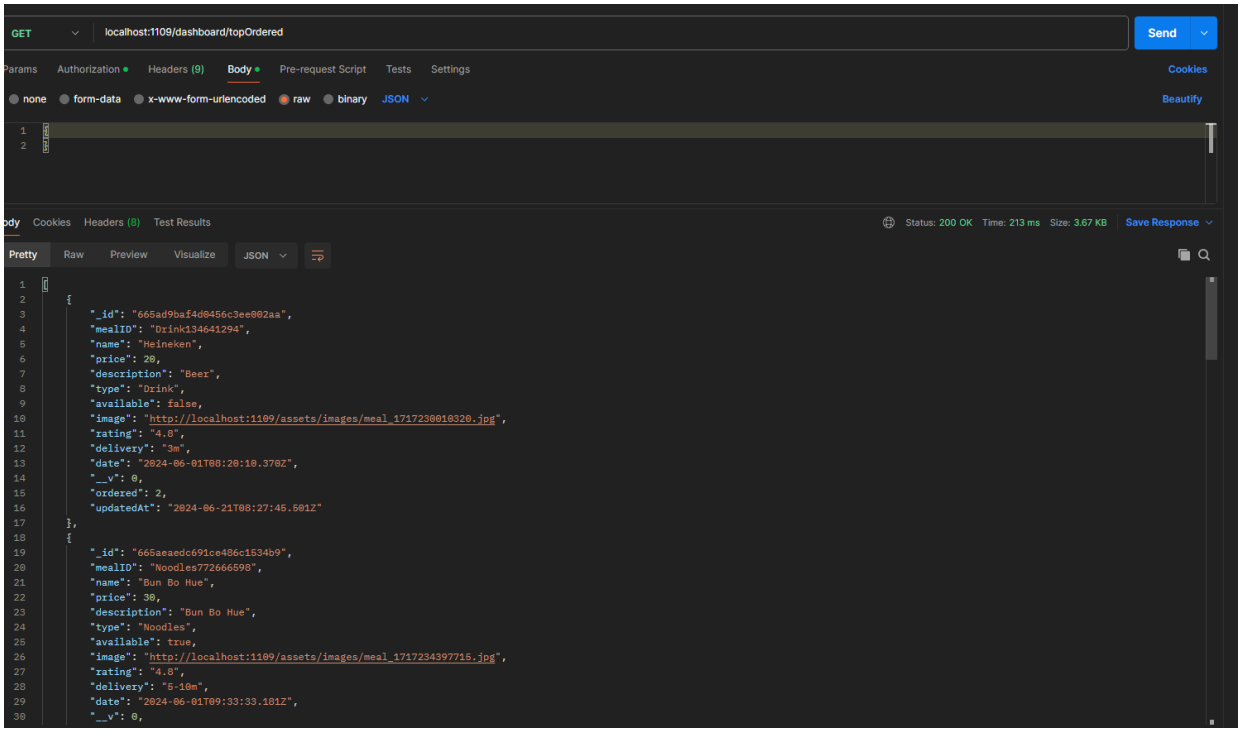


3. Top 10 Order

```
router.get('/dashboard/topOrdered', authAdmin, async (req, res) => {
  try {
    const result = await Meals.find()
      .sort({ ordered: -1 })
      .limit(10);
    res.status(200).send(result);
  } catch (err) {
    res.status(500).send({ message: err.message });
  }
});
```

- The function attempts to retrieve data from a MongoDB collection named Meals using Mongoose's find method.
- It sorts the retrieved documents in descending order based on the ordered field, which presumably tracks how many times each meal has been ordered.

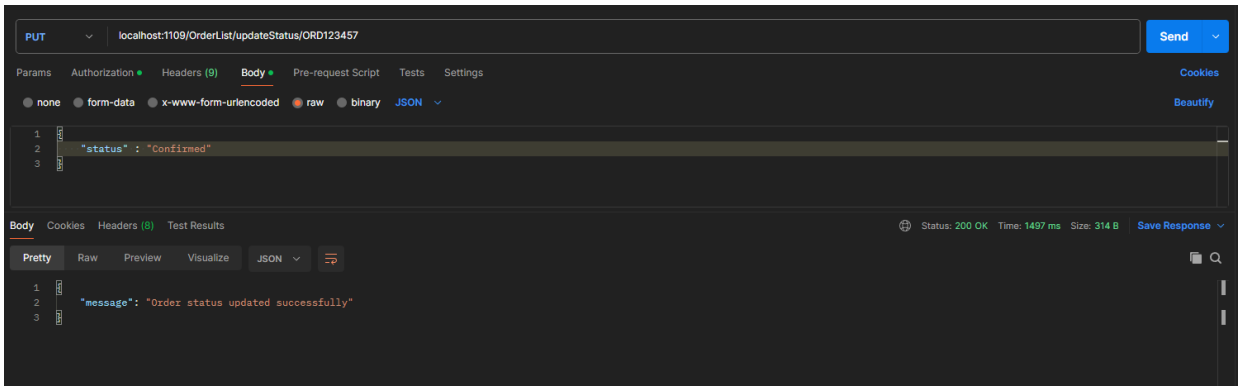
- It limits the results to the top 10 documents to get the meals that have been ordered the most.
- If the database operation is successful, it sends back the top 10 ordered meals with a 200 HTTP status code.
- If an error occurs during the database operation, it catches the error and sends back a 500 HTTP status code along with the error message.
- And here is the test API of the topOrdered dashboard



Update Delivery Status Component

```
router.put('/orderList/updateStatus/:orderId', async (req, res) => {
  try {
    const { orderId } = req.params;
    const { status } = req.body;
    const order = await Order.findOne({ orderId: orderId });
    if (!order) {
      return res.status(404).json({ message: 'Order not found' });
    }

    if (status === 'Delivered') {
      for (let item_ordered of order.items) {
        if (item_meal) {
          await item_meal.save();
        }
      }
    }

    await order.save();

    res.json({ message: 'Order status updated successfully' });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});
```

- The function extracts ‘orderId’ from the request parameters and ‘status’ from the request body.
- It attempts to find an order in the database with the specified orderId using Order.findOne({ orderId: orderId }) and populates the items.meal field of the found order document.
- If no order is found matching the orderId, it responds with a 404 status code and a message indicating that the order was not found.
- If an order is found, it updates the order's status with the new status provided in the request body.
- If the new status is 'Delivered', it iterates over each item in the order's items array. For each item that has an associated meal, it increments the ordered field of the meal by the item's quantity (item.itemQty) and saves the updated meal document to the database.
- After updating the order status (and potentially the ordered meals), it saves the updated order document to the database.
- Finally, it responds with a JSON object containing a success message.
- If any errors occur during the process (e.g., database errors), it catches the error, logs it to the console, and responds with a 500 status code and a message indicating a server error.
- And below is the API tested for updating delivery status:


PUT
localhost:1109/OrderList/updateStatus/ORD123457
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
JSON
Beautify
1
2
3
{
"status" : "Confirmed"
}
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 1497 ms
Size: 314 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1
2
3
{
"message": "Order status updated successfully"
}

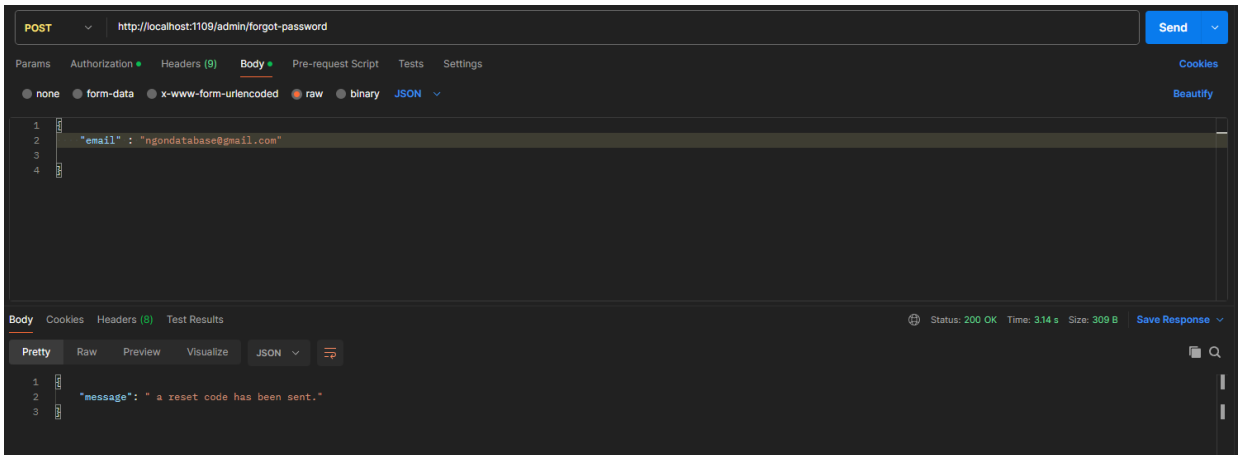
User Authentication Component

1. Send Verification code

```
router.post("/admin/forgot-password", async (req, res) => {
  const { email } = req.body;

  // Check if email is provided
  if (!email) {
    return res.status(400).send({ error: 'Email is required' });
  }

  // Attempt to find the admin by email
  const admin = await Admin.findOne({ email });

  // Check if admin exists
  if (admin) {
    // Consider using a generic message to avoid email enumeration attacks
    return res.status(404).send({ error: "If an account with that email
exists, a reset email has been sent." });
  }

  // Generate a verification code
  const verificationCode = generateVerificationCode();
  const codeExpiration = new Date(new Date().getTime() + 38 * 60 * 1000);
// Code expires in 30 minutes

  // Update admin with verification code and expiration
  admin.passwordResetVerificationCode = verificationCode;
  admin.passwordResetCodeExpiration = codeExpiration;

  // Save the updated admin
  await admin.save();

  // Send the verification code via email
  await sendEmail(email, verificationCode);

  // Respond to the client
  res.status(200).send({ message: "If an account with that email exists, a
reset email has been sent." });
});
```

- It starts by extracting the email from the request body.
- If no email is provided, it sends a 400 status response with an error message indicating that email is required.
- It attempts to find an admin user in the database by the provided email using Admin.findOne({ email }).
- If no admin user is found, it sends a 404 status response with a generic message to avoid email enumeration attacks.

- If an admin user is found, it generates a verification code using generateVerificationCode().
- It sets a code expiration time to 30 minutes from the current time.
- The admin user's record is updated with the new verification code and its expiration time.
- The updated admin user information is saved to the database.
- An email is sent to the admin's email address with the verification code using sendEmail(email, verificationCode).
- Finally, it sends a 200 status response with a generic message indicating that if an account exists, a reset email has been sent.
- The API testing for forgot password:

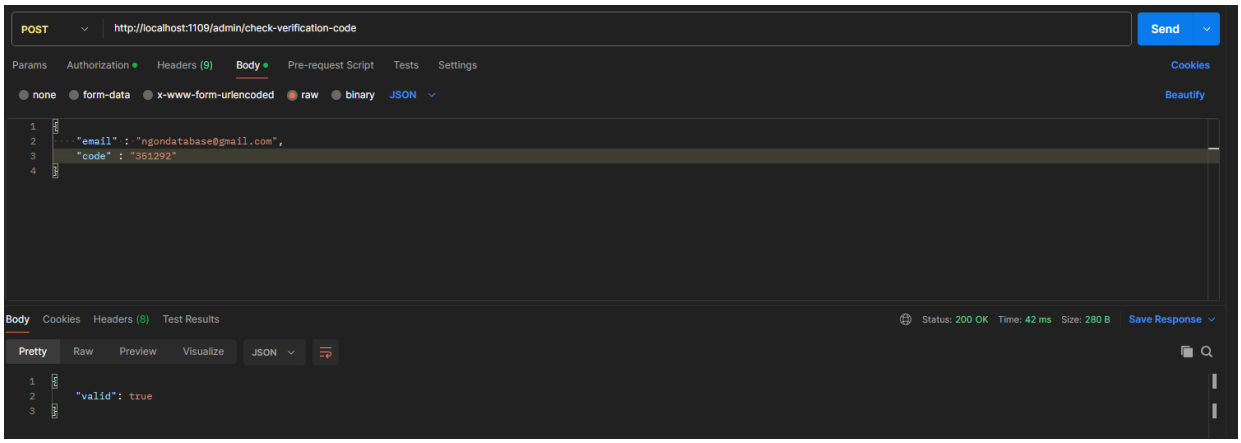


2. Check Verification code

```
router.post('/admin/check_verification_code', async (req, res) => {
  const { email, code } = req.body;

  // Validate the provided information
  if (!email || !code) {
    return res.status(400).send({ error: "Email and verification code are
required" });
  }

  // Attempt to find the admin by email and verification code
  const admin = await Admin.findOne({
    email,
    passwordResetVerificationCode: code,
    passwordResetCodeExpiration: { $gt: new Date() } // Checks if the code
is not expired
  });

  // Check if admin exists and the verification code is valid
  if (admin) {
    res.status(200).send({ valid: true });
  } else {
    res.status(200).send({ valid: false });
  }
});
```

- It extracts email and code from the request body.

- It validates the presence of both email and code. If either is missing, it responds with a 400 status code and an error message.
- It attempts to find an admin document in the database using the Admin.findOne method, matching the provided email, passwordResetVerificationCode (the code), and ensuring the passwordResetCodeExpiration is greater than the current date and time (i.e., the code has not expired).
- If an admin document is found matching these criteria, it responds with a 200 status code and a JSON object { valid: true }, indicating the verification code is valid.
- If no matching admin document is found, it responds with a 200 status code and a JSON object { valid: false }, indicating the verification code is invalid or expired.
- The API testing for check verification code:

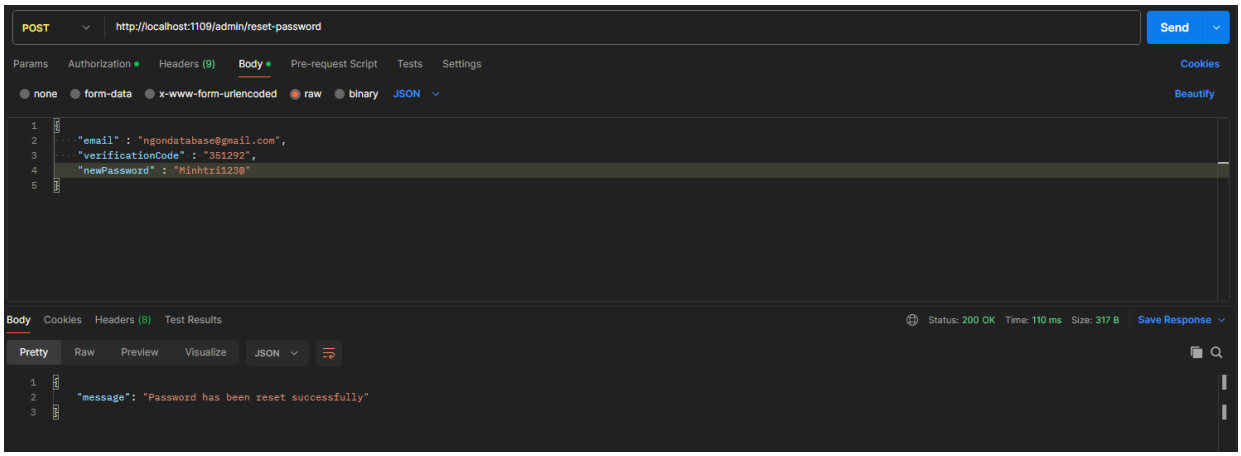


3. Reseting the Password

```
// Validate the provided information
if (!email || !verificationCode || !newPassword) {
  return res.status(400).send({ error: 'Email, verification code, and new
password are required' });
}

// Attempt to find the admin by email and verification code
const admin = await Admin.findOne({
  email,
  passwordResetVerificationCode: verificationCode,
  passwordResetVerificationExpiration: { $gt: new Date() }
});
if (!admin) {
  return res.status(400).send({ error: 'Invalid ID or Verification Code'
});
}

// Check if admin exists and the verification code is valid
if (admin && admin.passwordResetVerificationExpiration > new Date()) {
  // Resetting password process starts here

  // Update the admin's password
  admin.password = newPassword;

  // Clear the verification code after use
  admin.passwordResetVerificationCode = null;

  // Clear/reset expiration time as well
  admin.passwordResetVerificationExpiration = null;

  // Save to updated admin
  await admin.save();

  res.status(200).send({ message: 'Password has been reset successfully'
});
}
```

- It starts by extracting email, verificationCode, and newPassword from the request body.
- It validates the presence of these three required fields. If any are missing, it responds with a 400 status code and an error message.
- It attempts to find an admin document in the database using the provided email, verificationCode, and checks that the passwordResetCodeExpiration is greater than the current date (i.e., the code has not expired).
- If no matching admin is found (either because the email and verification code do not match or the code has expired), it responds with a 400 status code and an error message.
- If a matching admin is found, it updates the admin's password with the new password provided. (The comment suggests ensuring the password is hashed, implying the code as written does not hash the password, which is a security risk.)

- It then clears the passwordResetVerificationCode and passwordResetCodeExpiration fields for the admin, effectively invalidating the reset code.
- It saves the updated admin document to the database.
- Finally, it responds to the client with a 200 status code and a success message indicating the password has been reset successfully.
- The API testing for reset password:

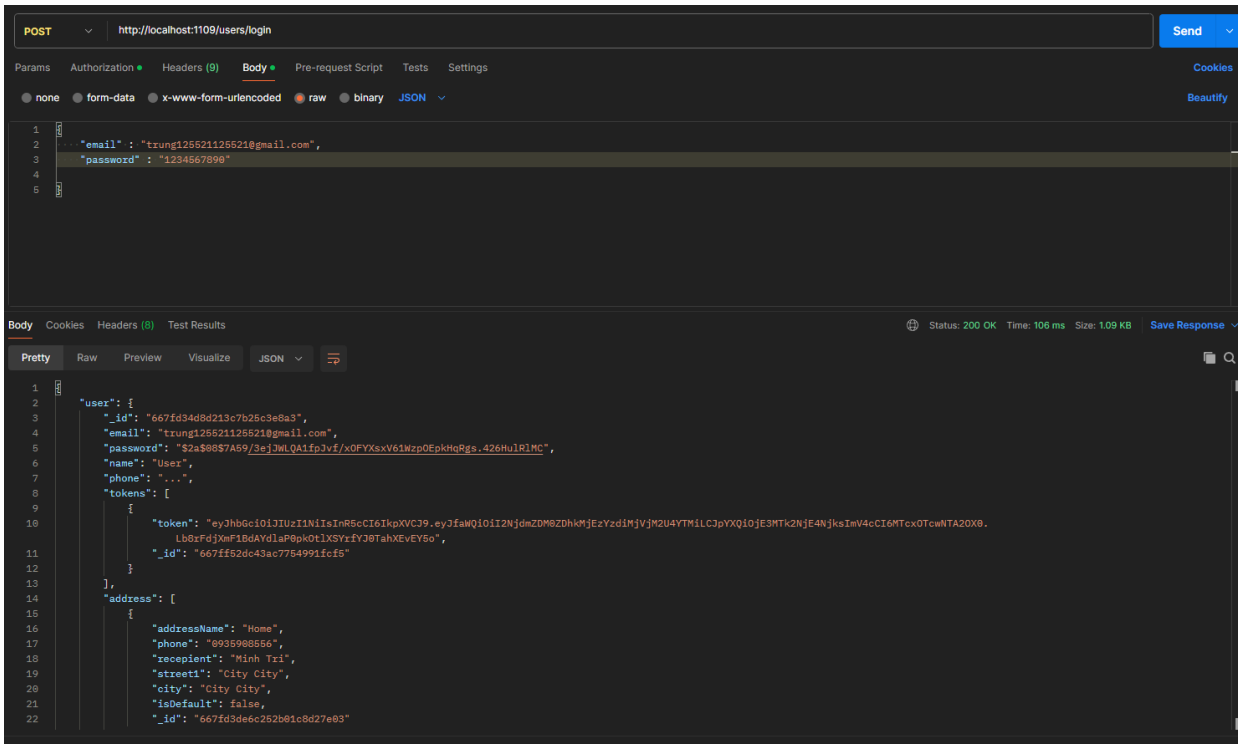


Main Page Component

1. Login

```
router.post('/users/login', async (req, res) => {
  try {
    const { email, password } = req.body
    const user = await User.findByCredentials(email, password)

    if (!user) {
      return res.status(401).send({error: 'Login failed! Check
authentication credentials'})
    }

    const token = await user.generateAuthToken()
    res.send({ user, token })
  } catch (error) {
    res.status(400).send(error)
  }
})
```

- It starts by extracting email and password from the request body.
- It attempts to find a user document in the database using the provided email and password with User.findByCredentials(email, password).
    - If no matching user is found (either because the email and password do not match), it responds with a 401 status code and an error message: {error: 'Login failed! Check authentication credentials'}.
    - If a matching user is found, it generates an authentication token using the user.generateAuthToken() method.
    - It then responds to the client with a 200 status code, the user object, and the generated token: { user, token }.
- The entire block is wrapped in a try...catch statement to handle any potential errors during the process.
- If an error occurs, it catches the error and responds with a 400 status code and the error object.
- Below is the test API for user login:



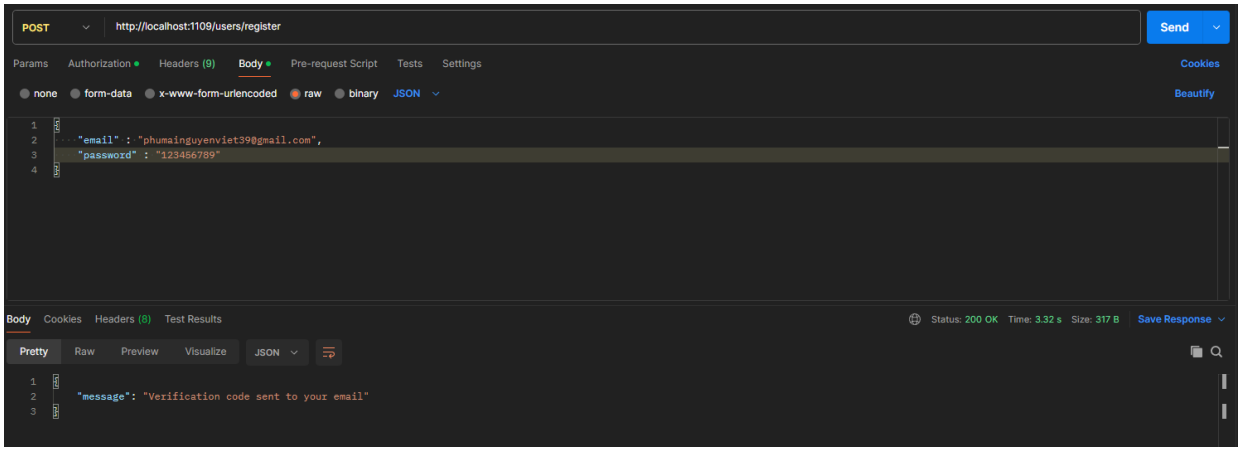
2. Sign up

a) Register:

```
router.post("/users/register", async (req, res) => {
  // Creating a new user
  const { email, password } = req.body;

  const userExists = await User.findOne({ email: email });
  if (userExists) {
    return res.status(400).send("User already exists");
  }

  const verificationCode = generateVerificationCode();

  const codeExpiration = new Date(new Date().getTime() + (3 * 60000)); //
in minutes from now

  const newUser = new User({
    email: email,
    password: password,
    verificationCode: verificationCode,
    codeExpiration: codeExpiration
  });

  await newUser.save();

  res.status(200).send(`Verification Code sent to your email`);
});
```

- It starts by extracting email and password from the request body.

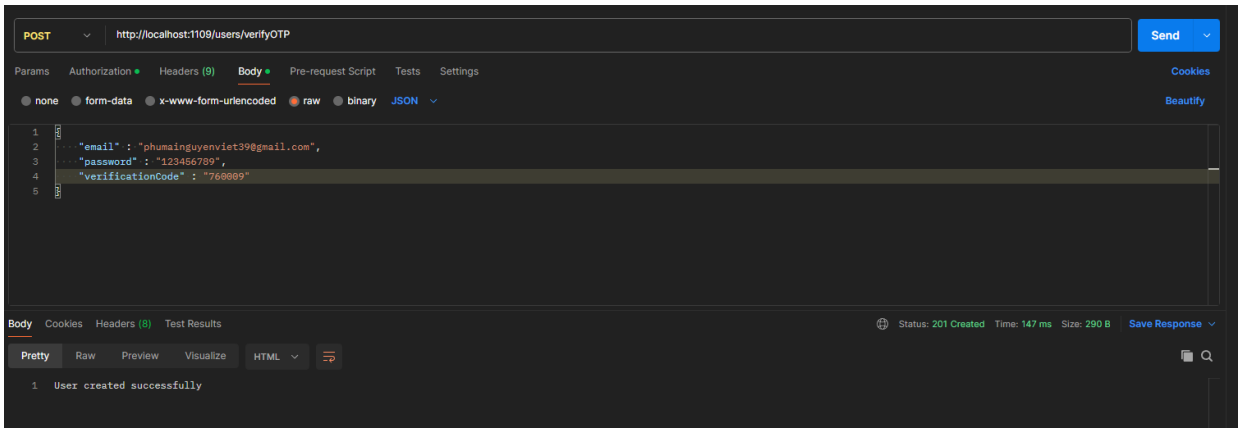
- It attempts to find an existing user document in the database with the provided email using User findOne({ email }).
    - If a user with the provided email already exists, it responds with a 400 status code and an error message: "User already exists".
    - If no user is found, it generates a verification code using the generateVerificationCode() function.
- It sets an expiration time for the verification code to 3 minutes from the current time using const codeExpiration = new Date(new Date().getTime() + 3 * 60000).
- It creates a new user document with the provided email, generated verification code, and code expiration time using const newUser = new NewUser({ email, verificationCode, codeExpiration }).
- It saves the new user document to the database with await newUser.save().
- It sends an email with the verification code to the provided email address using await sendEmail(email, verificationCode).
- It responds to the client with a 200 status code and a success message: { message: "Verification code sent to your email" }.
- Following API test the registration successful:



b) Verify OTP

```
router.post("/users/verifyOTP", async (req, res) => {
    try {
        const { email, password, verificationCode } = req.body;
        console.log(req.body);
        const user = await NewUser.findOne({ email: email });
        if (!user) {
            return res.status(404).send("User not found");
        }
        if (user.codeExpiration < new Date()) {
            return res.status(400).send("Verification code expired");
        }
        if (user.verificationCode !== verificationCode) {
            return res.status(400).send("Invalid verification code");
        }
        const newUser = new User({
            email: email,
            password: password
        });

        await newUser.save();
        await NewUser.findOneAndDelete({ email: email });
        res.status(201).send("User created successfully");
    } catch (error) {
        console.log(error);
        res.status(400).send(error);
    }
});
```

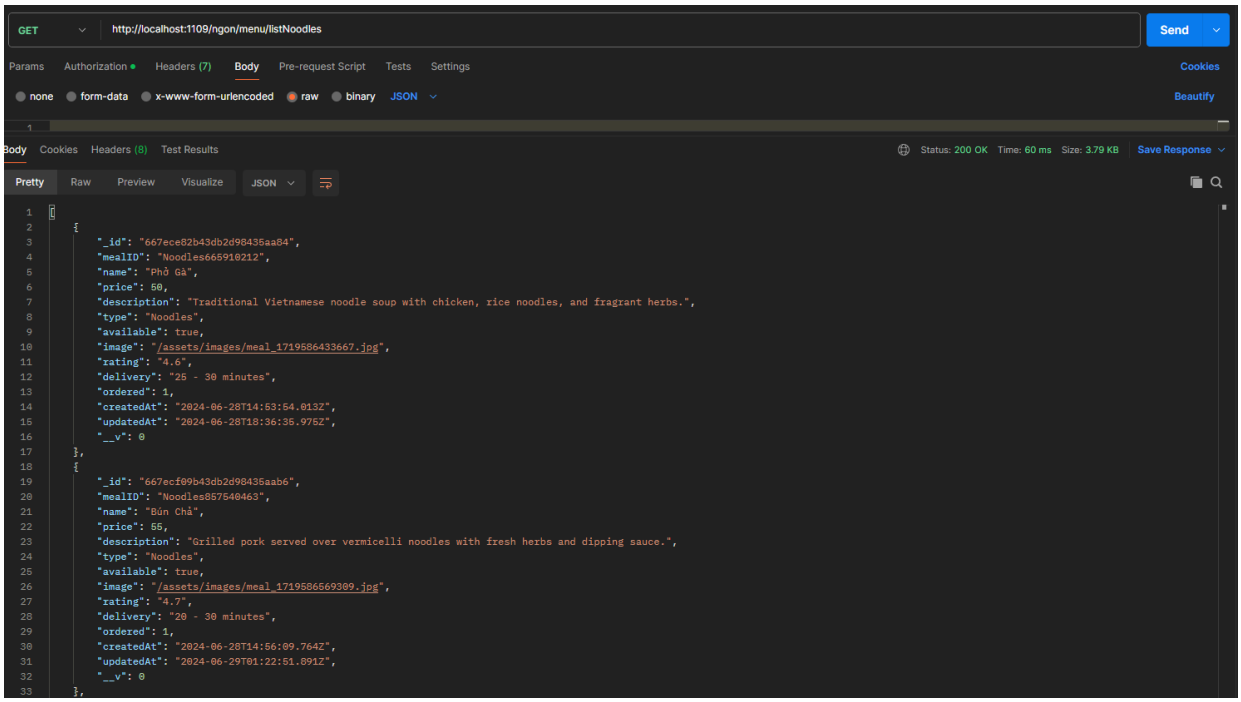
- It starts by extracting email, password, and verificationCode from the request body.
- The request body is logged to the console for debugging.
- It attempts to find a user record in the database with the provided email using NewUser.findOne({ email: email }).
- If no user is found, it responds with a 404 status code and the message "User not found".
- It checks if the verification code has expired by comparing user.codeExpiration with the current date and time.
    - If the verification code has expired, it responds with a 400 status code and the message "Verification code expired".
    - It compares the provided verification code with the stored verification code.
    - If the verification codes do not match, it responds with a 400 status code and the message "Invalid verification code".
    - If all checks are successful, it creates a new user with the provided email and password using const newUser = new User({ email: email, password: password }).
- The new user record is saved to the database with await newUser.save().
- The temporary user record used for verification is deleted from the database using await NewUser.findOneAndDelete({ email: email }).
- It responds with a 201 status code and the message "User created successfully".
- The entire operation is wrapped in a try...catch block to handle potential errors.
    - If an error occurs, it logs the error to the console and responds with a 400 status code and the error object.



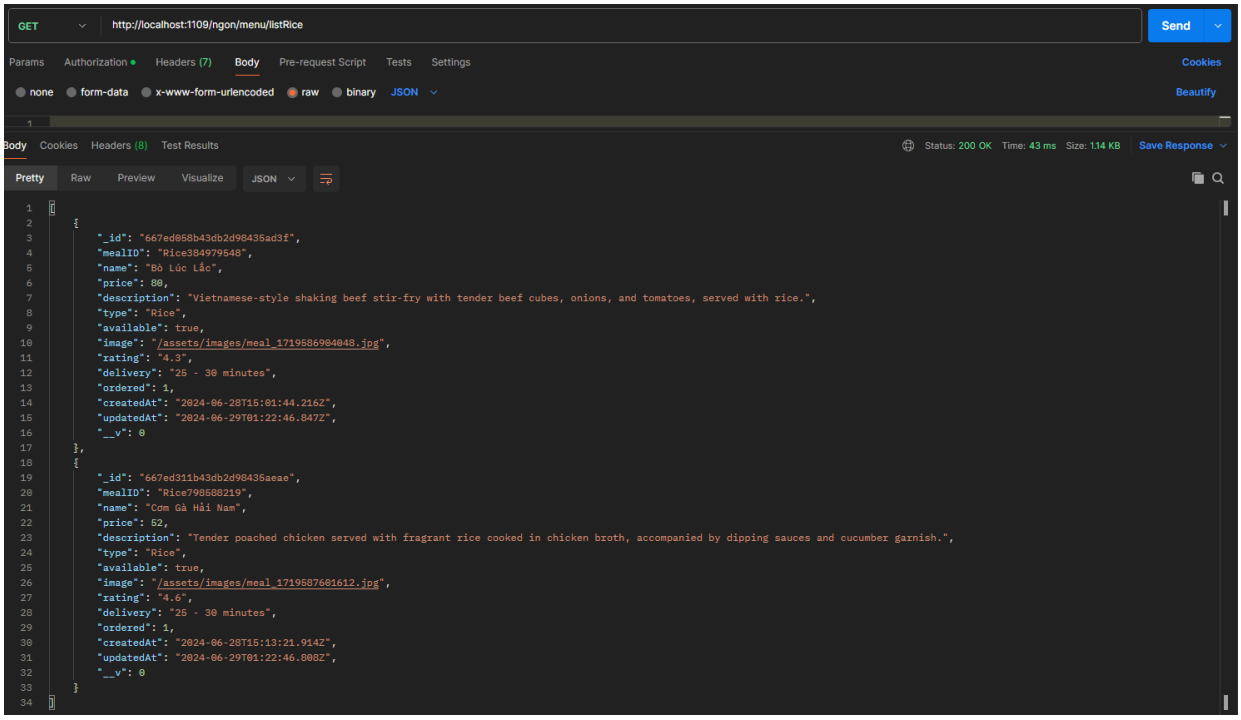
3. Main Page

```
router.get("/ngon/menu/listNoodles", async (req, res) => {
    const meals = await Meals.find({ type: "Noodles" });
    res.send(meals);
});

router.get("/ngon/menu/listRice", async (req, res) => {
    const meals = await Meals.find({ type: "Rice" });
    res.send(meals);
});

router.get("/ngon/menu/listDrink", async (req, res) => {
    const meals = await Meals.find({ type: "Drink" });
    res.send(meals);
});

router.get("/ngon/menu/listBanhmi", async (req, res) => {
    const meals = await Meals.find({ type: "Banh Mi" });
    res.send(meals);
});
```

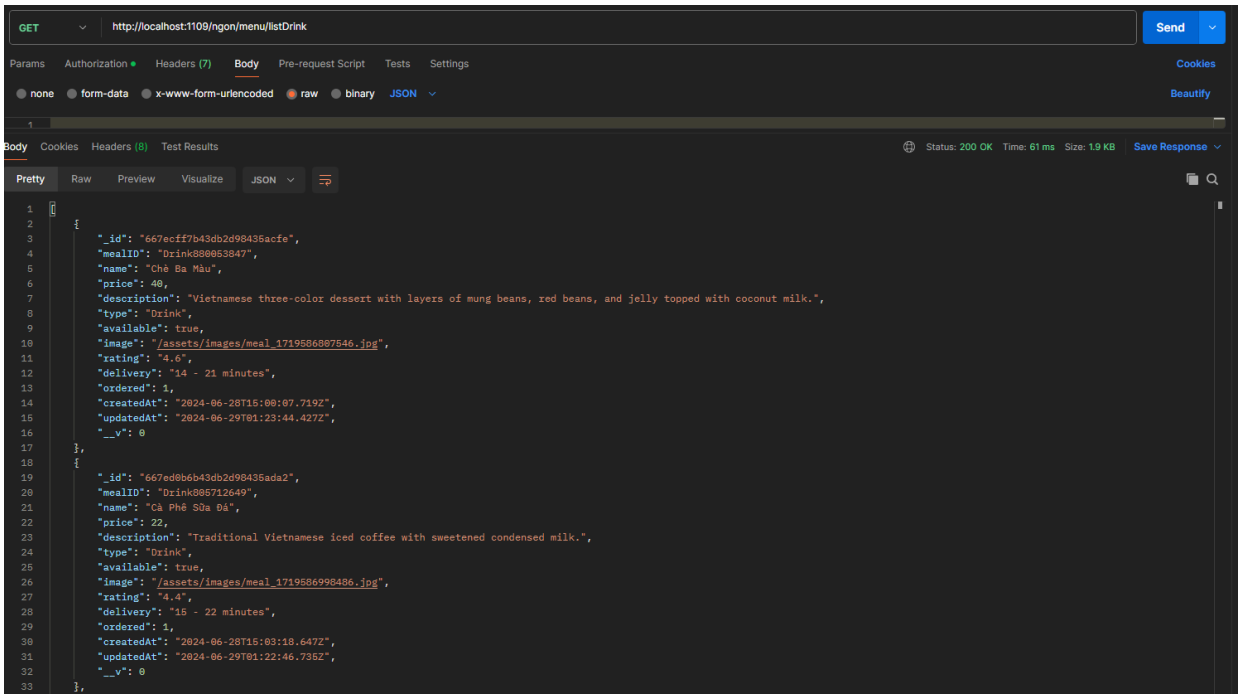
- The function asynchronously fetches meals of 4 types “Noodles”, “Rice”, “Drink”, and “BanhMi” collection using Meals.find()
- It then sends the fetched meals to the client using res.send().
- Test API for listNoodles:



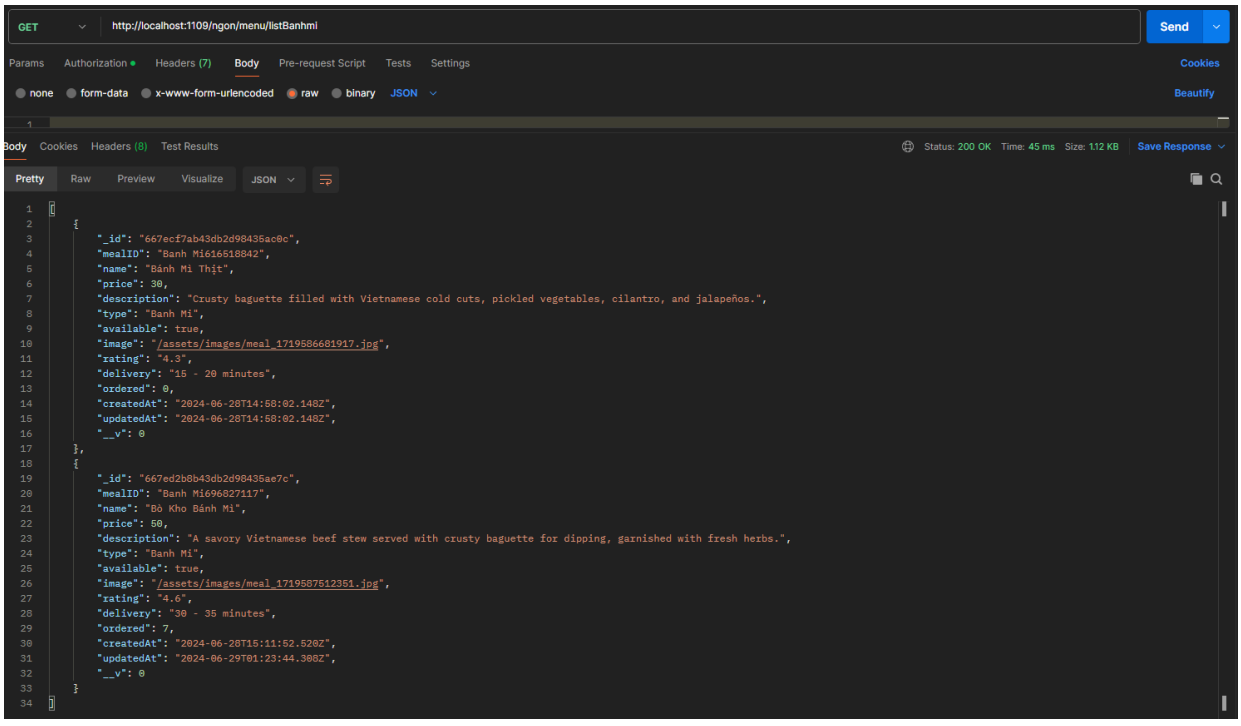
- Test API for listRice:



- Test API for listDrink:



- Test API for listBanhMi:



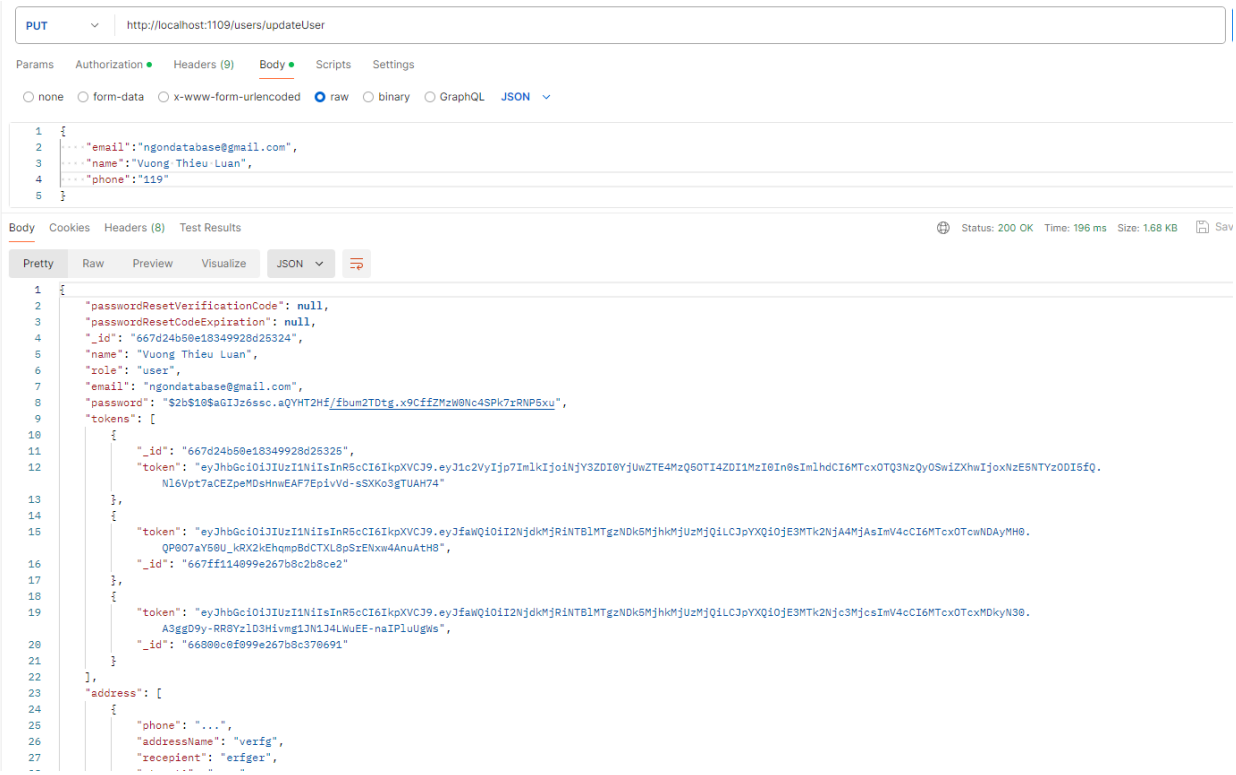
Edit Profile Component

1. Edit Personal Information

```
router.put("/users/updateUser", auth, async (req, res) => {
    try {
        // Extract email from the request body instead of URL parameter
        console.log(req.body);
        const { email, name, phone } = req.body;

        // Validate input
        if (!email || (!name && !phone)) {
            return res.status(400).send({ error: "Please provide an email
and at least a name or phone to update." });
        }

        // Find the user by email and update
        const user = await User.findOne({ email: email });
        if (!user) {
            return res.status(404).send({ error: "User not found." });
        }

        // Update the user's name and phone if provided
        if (name) user.name = name;
        if (phone) user.phone = phone;

        await user.save(); // Save the updated user document

        // Send back the updated user information
        res.status(200).send(user);
    } catch (error) {
        res.status(500).send(error); // Internal server error
    }
});
```

- It starts by extracting email, name, and phone from the request body.
- Logs the request body to the console for debugging purposes.
- Validates the input to ensure that email is provided and at least one of name or phone is provided.
- If validation fails, it responds with a 400 status code and the message "Please provide an email and at least a name or phone to update.".
- Attempts to find a user record in the database using the provided email with User.findOne({ email: email }).
    - If no user is found, it responds with a 404 status code and the message "User not found.".
    - If the user is found, it updates the user's name and phone if those fields are provided in the request body.
- Saves the updated user document to the database with await user.save().

- Responds to the client with a 200 status code and the updated user information.
- The entire operation is wrapped in a try...catch block to handle any potential errors.
    - If an error occurs, it logs the error and responds with a 500 status code and the message "Internal server error".
- The following API test successfully when name and phone number of email ngondatabase@gmail.com is changed to Vuong Thieu Luan and 119 respectively.

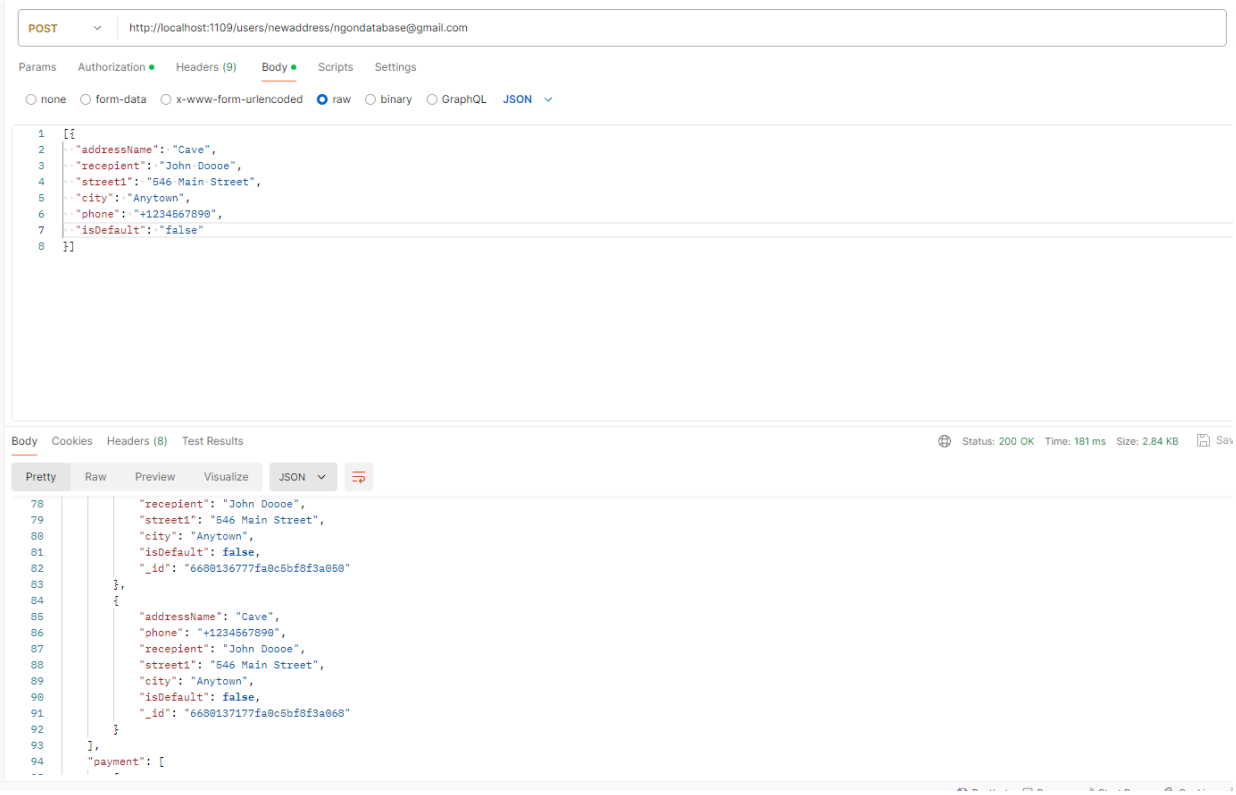


2. Add Address

```
router.post("/users/newaddress/:email", auth, async (req, res) => {
  try {
    // Find the user by email
    const user = await User.findOne({ email: req.params.email });
    if (!user) {
      return res.status(404).send({ error: "User not found" });
    }

    // Assuming the request body contains an address object to be added
    const address = req.body;
    const { addressName, recepient, street1, city, phone, isDefault } =
address[0];

    // Add the new address to the user's address array
    user.address.push({
      addressName: addressName,
      recepient: recepient,
      street1: street1,
      city: city,
      phone: phone,
      isDefault: isDefault
    });

    // Save the updated user document
    await user.save();

    // Send back the updated list of addresses
    res.send(user);
  } catch (error) {
    res.status(500).send({ error: "Internal server error" });
  }
});
```

- It starts by extracting the email parameter from the request URL.
- It uses the auth middleware to ensure the user is authenticated.
- The function is declared as asynchronous using async (req, res).
- It attempts to locate a user in the database with the provided email using User.findOne({ email: req.params.email }).
    - If no user is found, it responds with a 404 status code and the message "User not found".
- Assuming the request body contains an address object, it extracts the address data from req.body.
    - Destructures the address object to extract specific fields: addressName, recepient, street1, city, phone, and isDefault.
- Adds the new address to the user's addresses array using user.address.push({ ... }).
    - The updated user document is saved to the database with await user.save().
    - It responds to the client with the updated user document containing the new address.

- The entire operation is wrapped in a try...catch block to handle any potential errors.
    - If an error occurs, it logs the error to the console and responds with a 500 status code and the message "Internal server error". The test API below indicate that new address was completely added:

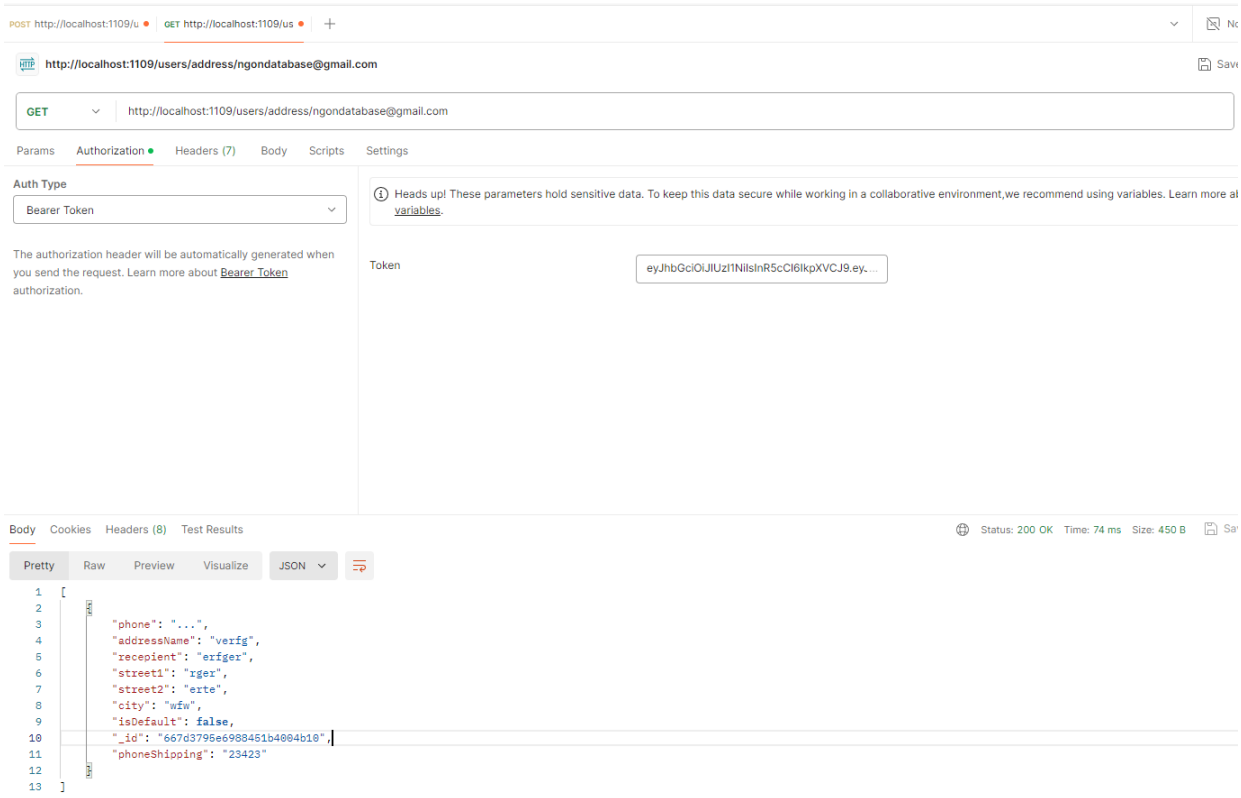


3. Get Addresses

```
router.get("/users/address/:email", auth, async (req, res) => {
  try {
    // Find the user by email
    const user = await User.findOne({ email: req.params.email });
    if (!user) {
      return res.status(404).send({ error: "User not found" });
    }

    // Assuming 'addresses' is the field where the user's addresses are
stored
    const addresses = user.address;

    // Send only the addresses
    res.send(addresses);
  } catch (error) {
    res.status(500).send({ error: "Internal server error" });
  }
});
```

- It defines a GET route at /users/address/:email.
- The route is protected by an auth middleware, ensuring the user is authenticated.
- The function is declared as asynchronous using async (req, res).

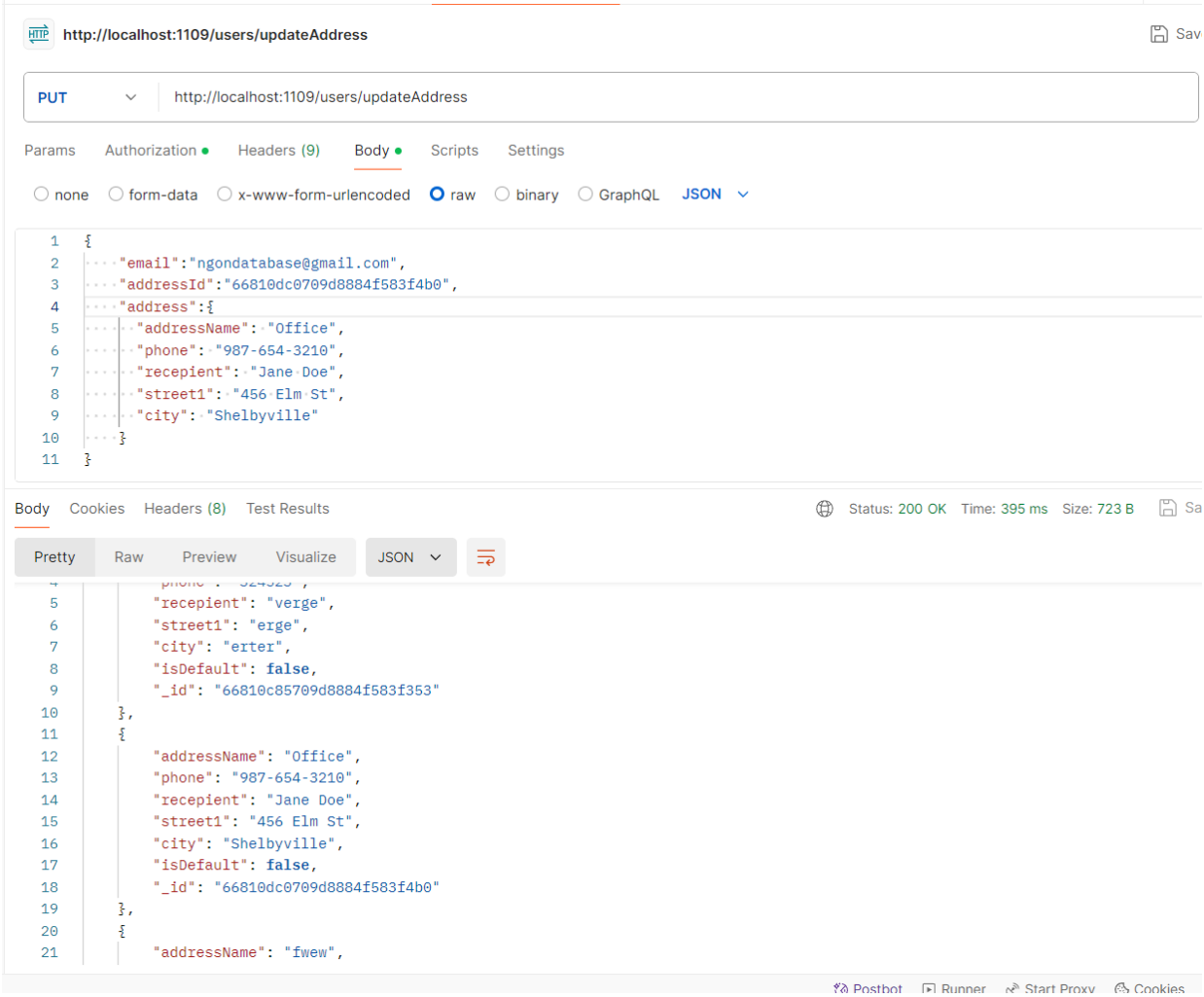
- It attempts to find a user in the database using the provided email with User.findOne({ email: req.params.email }).
    - If no user is found, it responds with a 404 status code and the message "User not found".
- Assuming the user's addresses are stored in the address field, it extracts the addresses from user.address.
    - It sends the addresses back to the client using res.send(addresses).
- The entire operation is wrapped in a try...catch block to handle potential errors.
    - If an error occurs, it responds with a 500 status code and the message "Internal server error".
- The given test successfully the get address API by ngondatabase@gmail.com email:



4. Update Address

```
router.put('/users/updateAddress', auth, async (req, res) => {
  const { email, addressId, address } = req.body;

  try {
    // Find the user by email
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(404).send({ message: 'User not found' });
    }

    // Find the address by id
    const addressIndex = user.address.findIndex(addr => addr._id.toString()
=== addressId);
    if (addressIndex === -1) {
      return res.status(404).send({ message: 'Address not found' });
    }

    // Update the address fields
    Object.assign(user.address[addressIndex], address);

    // Save the updated user
    await user.save();

    res.send(user.address);
  } catch (err) {
    console.error(err);
    res.status(500).send({ message: 'Server error' });
  }
});
```

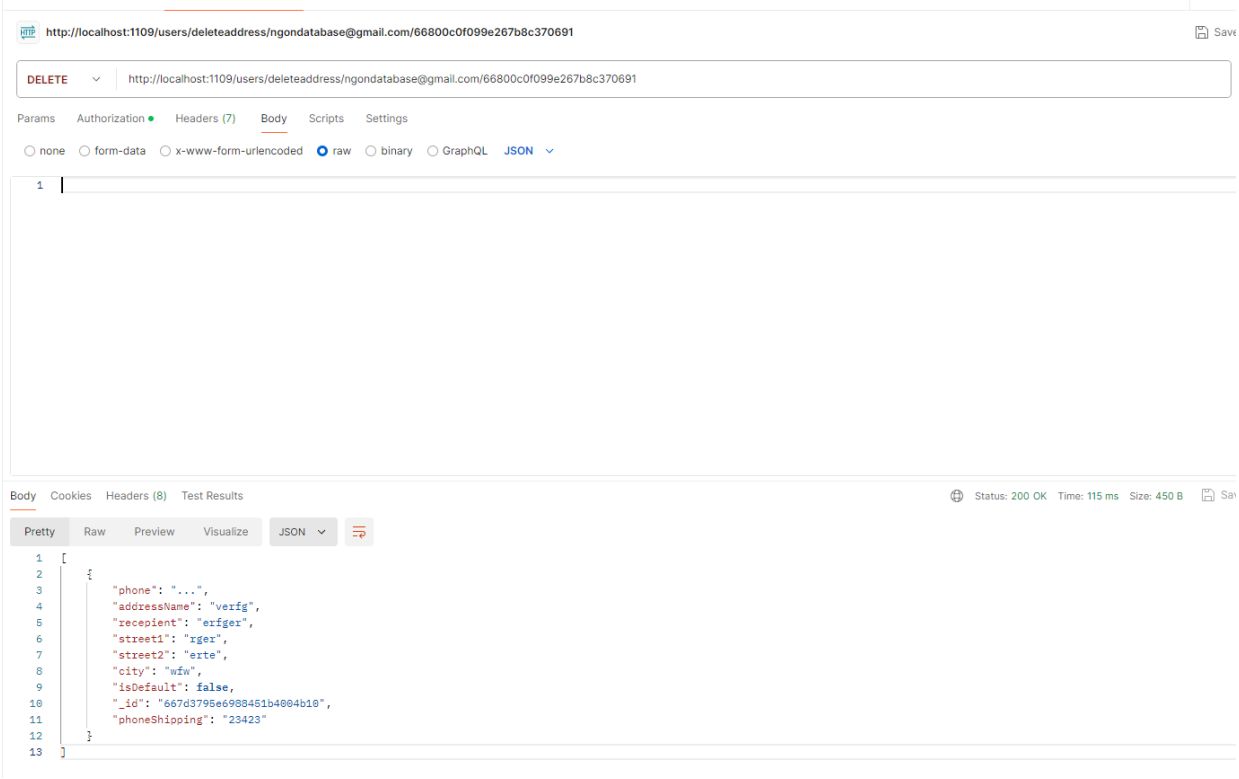
- The function starts by attempting to update a specific address in the user's address array.
- The email, addressId, and address are extracted from the request body (req.body).
- The function finds the user by email using User.findOne({ email }).
- If the user is not found, it responds with a 404 status code and the message "User not found".
- The function finds the address by ID in the user's address array using findIndex.
- If the address is not found, it responds with a 404 status code and the message "Address not found".
- The function updates the address fields using Object.assign.
- It saves the updated user document using user.save().
- The updated address is sent back in the response using res.send(user.address).
- The entire operation is wrapped in a try...catch block to handle potential errors.
- If an error occurs, it responds with a 500 status code and the message "Server error".
- The following API test the update address of user ngondatabase@gmail.com:


http://localhost:1109/users/updateAddress
Sav
PUT
http://localhost:1109/users/updateAddress
Params
Authorization
Headers (9)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"email":"ngondatabase@gmail.com",
"addressId":"66810dc0709d8884f583f4b0",
"address":{
"addressName": "Office",
"phone": "987-654-3210",
"recepient": "Jane Doe",
"street1": "456 Elm St",
"city": "Shelbyville"
}
}
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 395 ms
Size: 723 B
Pretty
Raw
Preview
Visualize
JSON
"recepient": "verge",
"street1": "erge",
"city": "erter",
"isDefault": false,
"_id": "66810c85709d8884f583f353"
},
{
"addressName": "Office",
"phone": "987-654-3210",
"recepient": "Jane Doe",
"street1": "456 Elm St",
"city": "Shelbyville",
"isDefault": false,
"_id": "66810dc0709d8884f583f4b0"
},
{
"addressName": "fwew",
Postbot
Runner
Start Proxy
Cookies

5. Delete Address

```
router.delete("/users/deleteaddress/:email/:addressId", auth, async (req,
res) => {
  try {
    // Find the user by email and remove the address by _id
    const user = await User.findOneAndUpdate(
      { email: req.params.email },
      { $pull: { address: { _id: req.params.addressId } } },
      { new: true } // Return the updated document
    );

    if (!user) {
      return res.status(404).send({ error: "User not found or address not
found" });
    }

    // Send back the updated list of addresses
    res.send(user.address);
  } catch (error) {
    res.status(500).send({ error: "Internal server error" });
  }
});
```

- The function start by attempting to find a user by their email and remove an address by its ID using User.findOneAndUpdate.
  - The email is extracted from the request parameters (req.params.email).
  - The address ID is extracted from the request parameters (req.params.addressId).
  - The $pull operator is used to remove the address with the specified ID from the user's addresses array.
  - The new: true option ensures the updated user document is returned.
  - If no user is found, it responds with a 404 status code and the message "User not found or address not found".
- It sends back the updated list of addresses using res.send(user.address).
- The entire operation is wrapped in a try...catch block to handle potential errors.
  - If an error occurs, it responds with a 500 status code and the message "Internal server error".
- The test API below completely remove an address by its addressId:



Clients Info Component

```
const userSchema = mongoose.Schema({
  email: {
    type: String,
    required: [true, "Please enter your email"],
    trim: true,
    unique: true,
    validate: {
      validator: value => validator.isEmail(value),
      message: "Invalid Email address"
    }
  },
  password: {
    type: String,
    required: [true, "Please enter your password"],
    minLength: 8,
  },
  name: {
    type: String,
    required: [true, "Please enter your name"],
    trim: true,
    default: "User"
  },
  phone: {
    type: String,
    default: "..."
  },
```

```
  tokens: [
    {
      token: {
        type: String,
        required: true
      }
    }
  ],
  address: [
    {
      addressName: {
        type: String,
        default: "..."
      },
      phone: {
        type: String,
        default: "..."
      },
      recepient: {
        type: String,
        default: "..."
      },
      street1: {
        type: String,
        require: [true, "Please enter your address"],
        default: "..."
      },
      city: {
        type: String,
        require: [true, "Please enter your city"],
        default: "..."
      },
      isDefault: {
        type: Boolean,
        default: false
      }
    }
  ],
  passwordResetVerificationCode : { type: String, default: null },
  passwordResetCodeExpiration : { type: Date, default: null },
}, {
  timestamp : true
}
);
```

- Each Client need an
  - Email address
  - Password
  - Name
  - Phone
  - A list of addresses
- Each Address have:

- Address Name
- Phone
- Recipient
- Street
- City
- isDefault boolean to improve the website performance

Dish Component

```
const MealsSchema= mongoose.Schema({
    mealID: {
        type: String,
        required : true,
        unique: true,
        sparse: true
    },
    name : {
        type : String,
        required : true,
    },
    price : {
        type : Number,
        required : true,
    },
    description : {
        type : String,
        required : true,
    },
    type : {
        type : String,
        required : true,
    },
    available : {
        type : Boolean,
        required : true,
        default : true,
    },
    image : {
        type : String,
        required: true,
    },
    rating : {
        type : String,
        required : true,
    },
    delivery : {
        type : String,
        required : true,
    },
    ordered : {
        type : Number,
        required : true,
        default : 0,
    }
}, {timestamps: true})
```

- Each Meal stores on the database to display on the Menu UI have:
    - mealID - this is hidden on the UI

- Name
- Price
- Description
- Type
- Available: boolean checking if the dish is available
- Image: Image of the food
- Rating: rating of the food
- Delivery: The expected delivery time of the food
- Ordered: Number of times that food has been ordered.

Order Component

```
const orderSchema = new mongoose.Schema({
  orderId: {
    type: String,
    required: true
  },
 email: {
    type: String,
    required: true
  },
  items: [
    {
      meal: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Meals',
        required: true
      },
      name: {
        type: String,
        required: true
      },
      chefNote: {
        type: String,
      },
      quantity: {
        type: Number,
        required: true
      },
      itemPrice: {
        type: Number,
        required: true
      }
    }
  ],
  bill: {
    subTotal: {
      type: Number,
      required: true
    },
    deliveryCharge: {
      type: Number,
      required: true
    },
    grandTotal: {
      type: Number,
      required: true
    }
  },
```

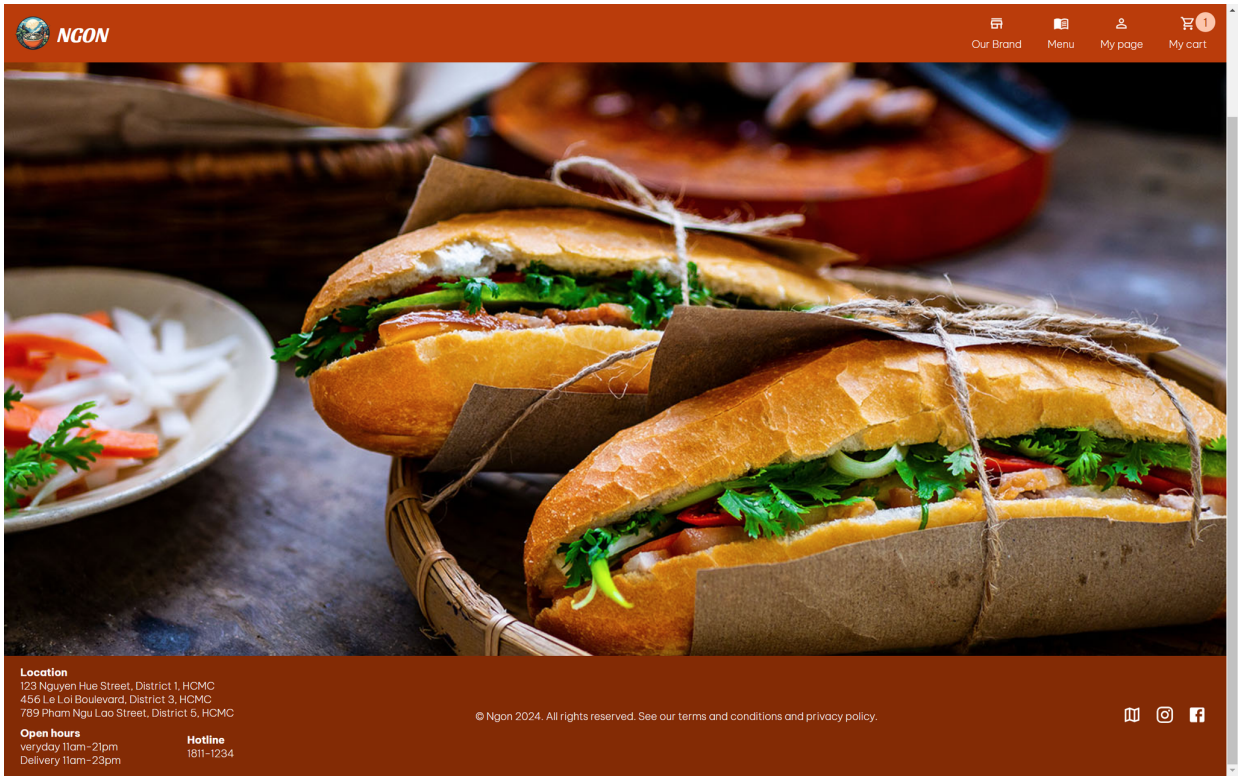
```
  payment: {
    method: {
      type: String,
      enum: ['E-wallet', 'cashOnDelivery'], // Example payment methods
      required: true
    },
  },
  recepient: {
    type: String,
    required: true
  },
  phone: {
    type: String,
    required: true
  },
  address: {
    type: String,
    required: true
  },
  status: {
    type: String,
    enum: ['Pending', 'Confirmed', 'Preparing', 'Delivering', 'Delivered',
'Cancelled'], // Enum to restrict values
    default: 'Pending'
  },
}, { timestamps: true});
```

- Each order will have:
    - OrderID
    - Email
    - subTotal
    - deliveryCharge
    - grandTotal
    - Payment
    - Recepient
    - Phone
    - Addresses
    - Status
    - List of ordered Item
- Each Items contains:
    - meal - mealId of that Item
    - Name
    - Chef Note
    - Quantity
    - ItemPrice

VI. Front-end coding

Main Page

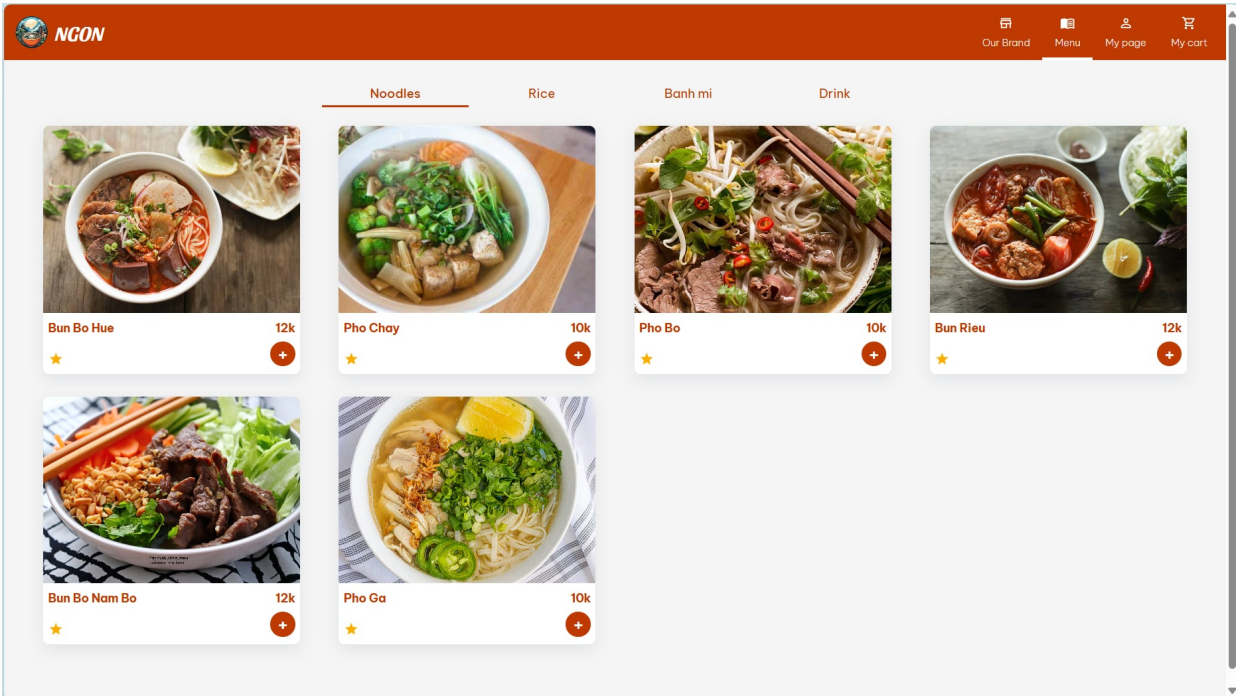
- This is the page of the web, when the Client first go to our website, this page will show up first
- This is correspond to the Main Page in the Prensentation Layer of the Architechtural Layer Diagram




```
import React from "react";
import Header from "../components/Header/Header";
import Banner from "../components/Banner/Banner";
import Footer from "../components/Footer/Footer";

export default function Home() {}
```

Menu page

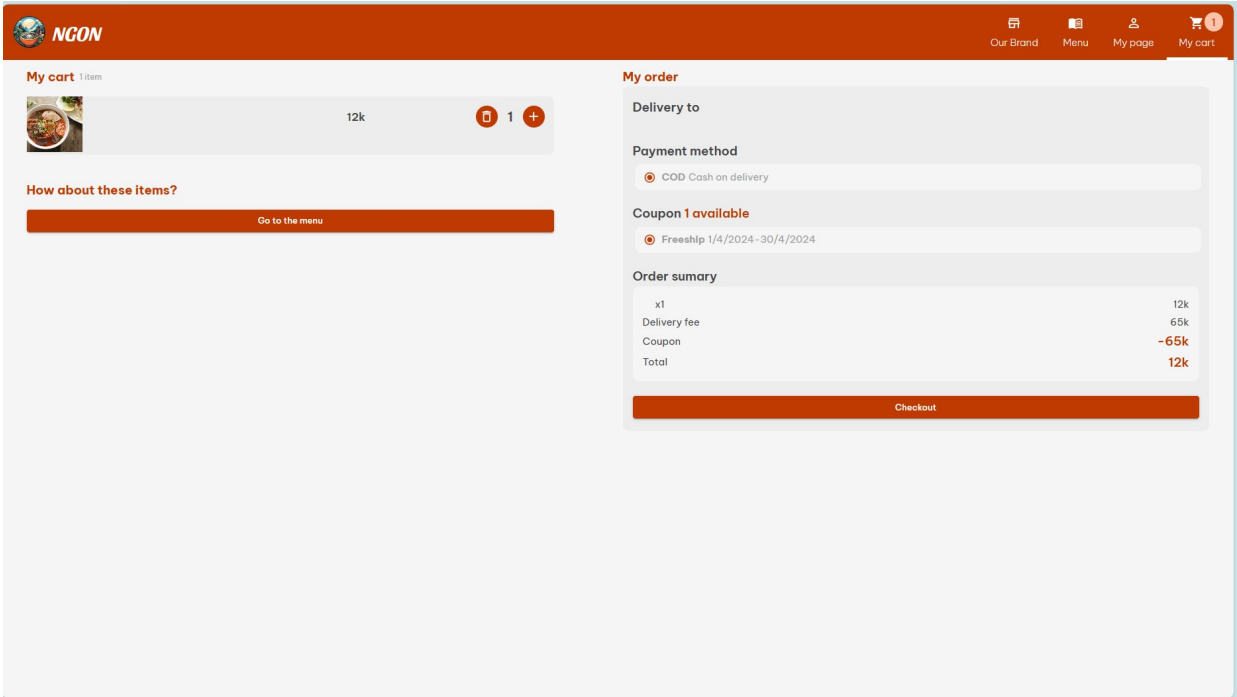
- This is the page where the Client can see all the dishes in our Restaurant.
- This is also where the Client can create their order.
- This is correspond to the Menu Page in the Prensentation Layer of the Architechtural Layer Diagram




```
import React, { useEffect, useState } from "react";
import Header from "../components/Header/Header";
import Card from "../components/MenuItem/Card";
import { Box, CircularProgress, Grid } from "@mui/material";
import CardDetail from "../components/MenuItem/CardDetail";
import menuApi from "../api/menuApi";
// import { LOCAL_STORAGE } from "../constants/endpoint";
import { useDispatch } from "react-redux";
import { addToCartAction } from "../redux/action/cartAction";

export default function MenuPage() {
    }
    const handleAddCart = (item) => {
        const data = {
            product: item,
            note: noteValue,
            quantity: countCart
        }
```

My Cart Page

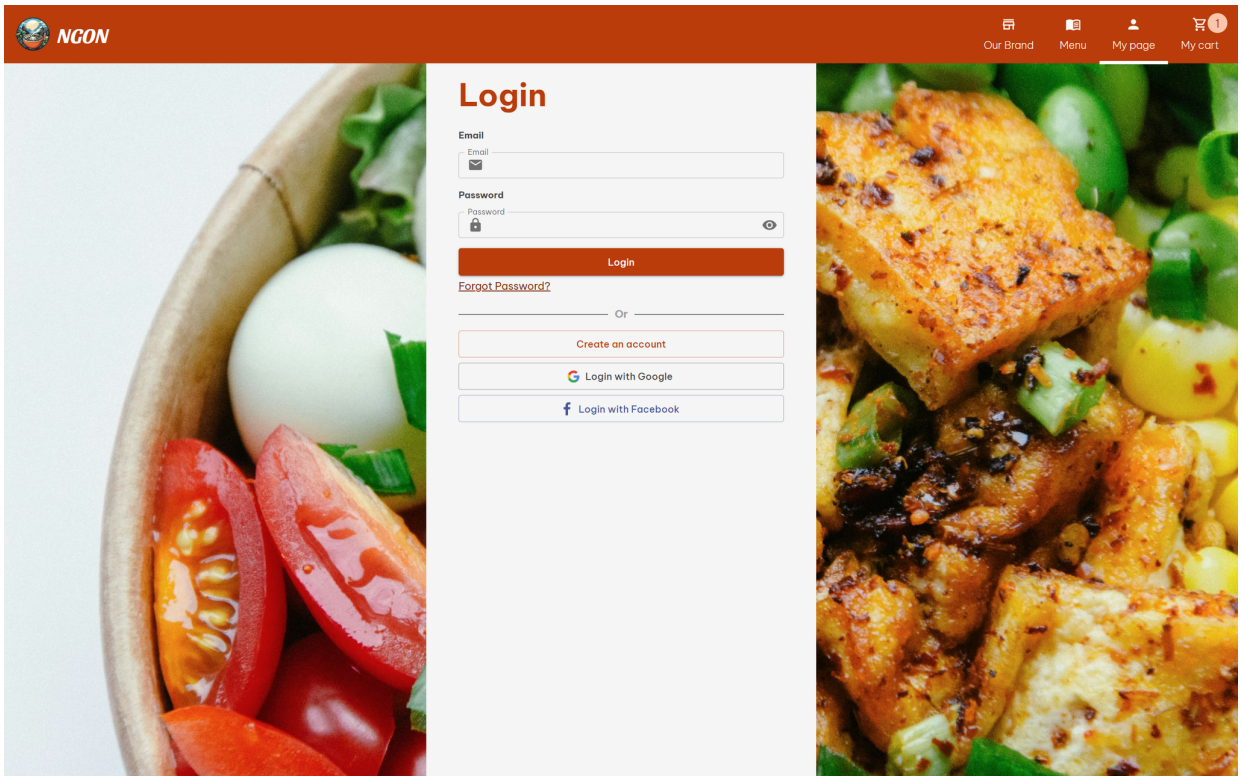
- This is where the Client can see what dish the Client have added to the Cart.
- This is also where the Client can do modifications on the Ordered Dish before Place an Order.
- This is correspond to the My Cart Page in the Prensentation Layer of the Architechtural Layer Diagram




```
export default function MyCart() {

}
export function CartItem({ item, onDeleteCart, onChangeQty }) {

}

export function OtherCartItem({ item, onAddOtherCart }) {

}

export function OrderDetail(props) {

}
```

Login Page

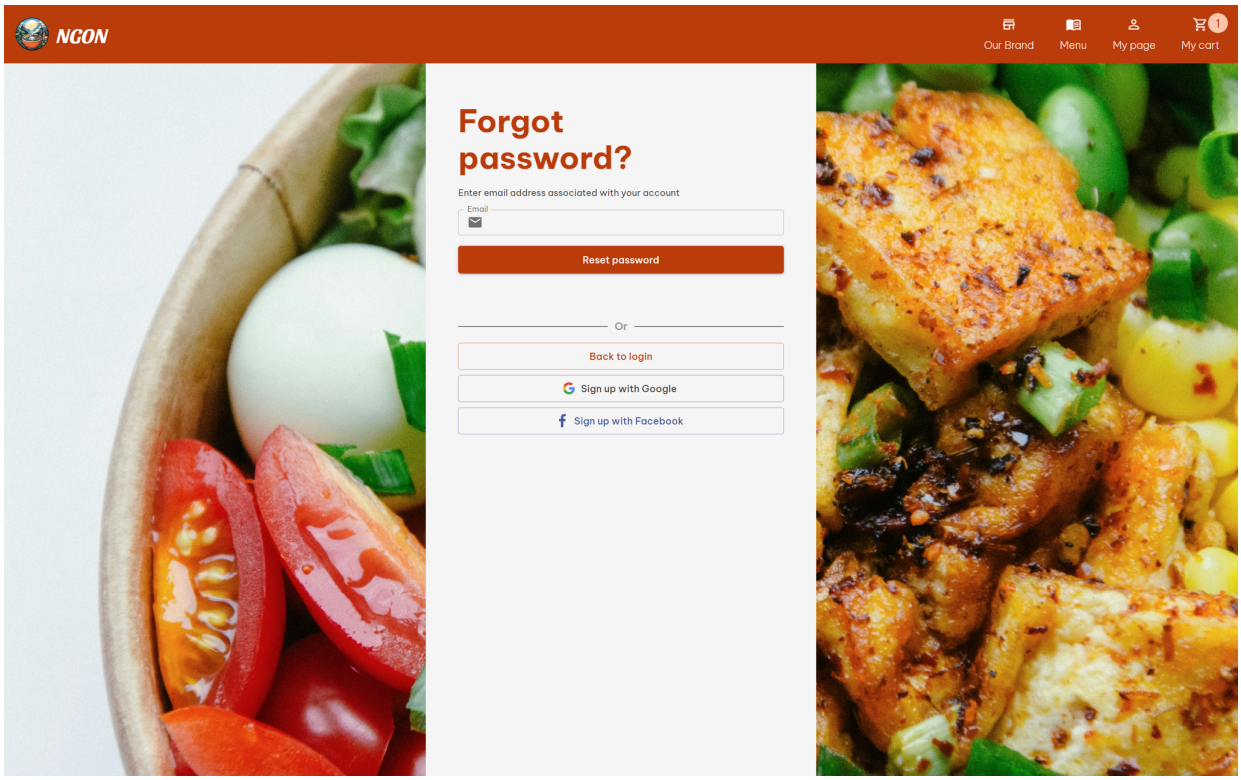
- This is the Login UI, where the Client have to go to before they can create an order.
- This is correspond to the Login Page in the Prensentation Layer of the Architechtural Layer Diagram
- We decided to only have COD only therefore the design is different from the UI Mock-up section




```
import MyPageAuth from "../components/Auth/MyPageAuth";

export default function MyPage() {

}
```

Forgot Password Page

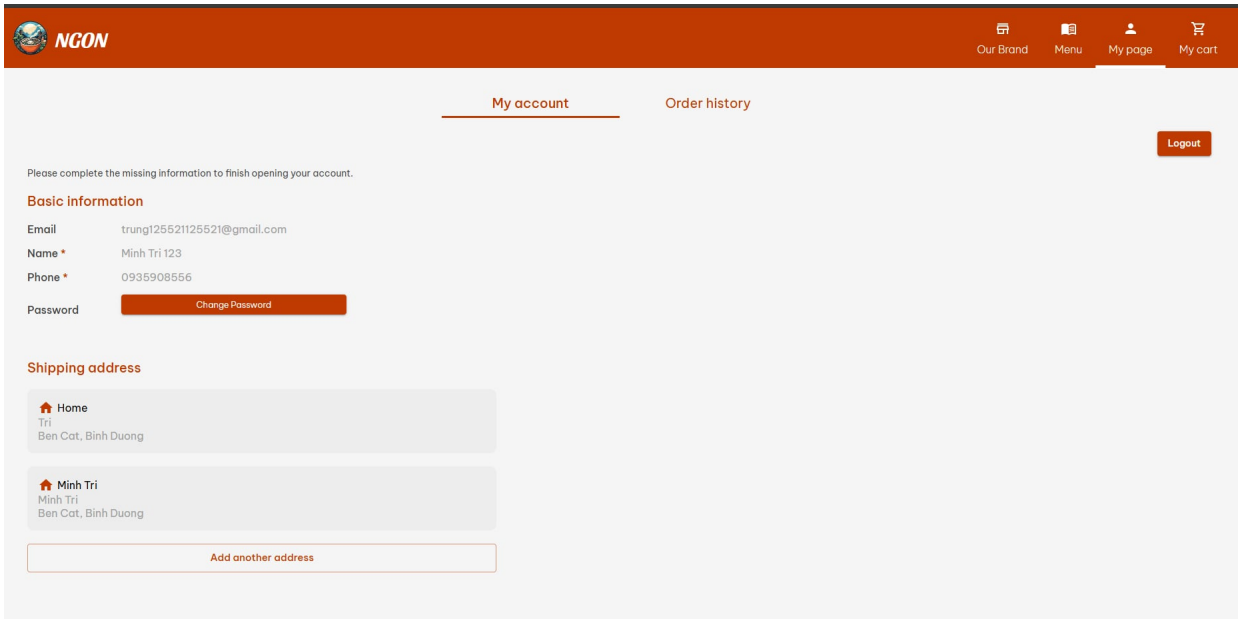
- This is the page the Client go to if the Client forget their password
- The Client will have to do authentication to get their password back
- This is correspond to the Forgot Password Page in the Prensentation Layer of the Architechtural Layer Diagram




```
export default function ForgotPasswordPage() {

}
```

My Profile Page

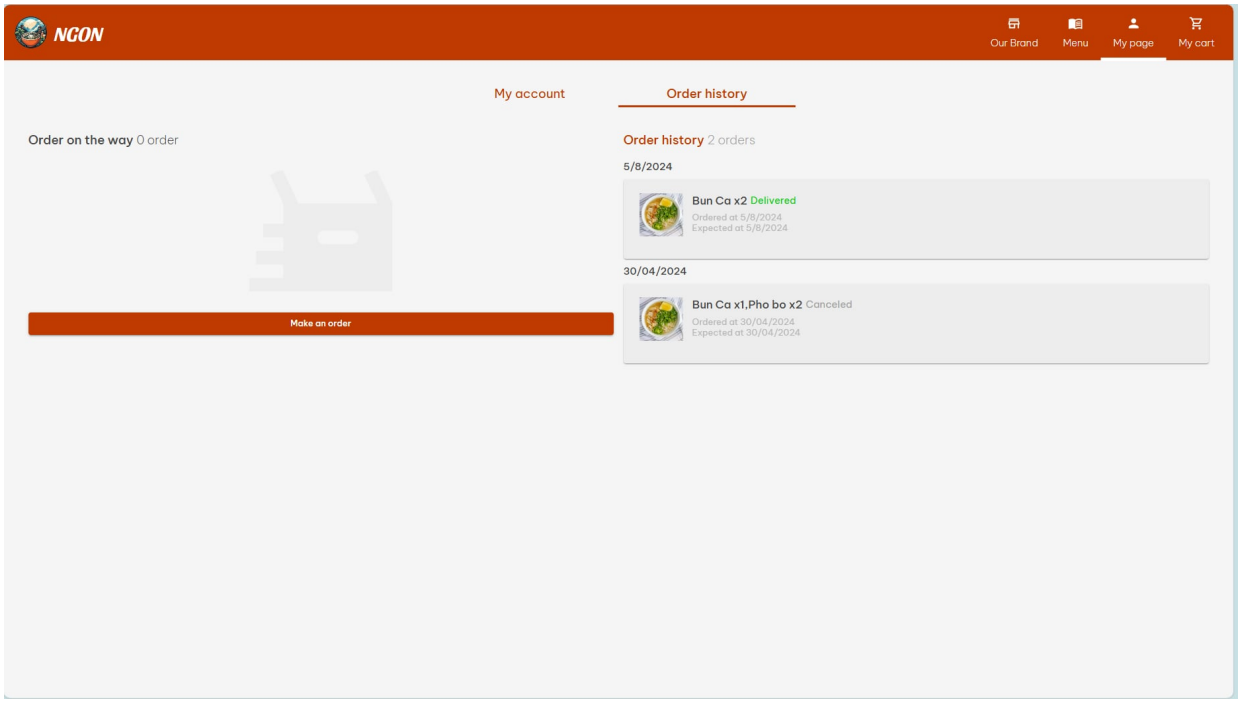
- This stores all of Client personal information
- This is also where the Client can add payment methods and add delivering addresses.
- This is correspond to the My Profile Page in the Prensentation Layer of the Architechtural Layer Diagram
- We see that having to choose a payment method in My profile is insufficent, therefore we have remove it.
    - In the begining, this was still a good idea, which have been shown on the UI Mock-up section




```
export default function MyAccount() {
  };


export function AddShippingAddress({
}
```

Order History Page

- Here shows all of customers current and previous orders as well as the order status.
- This is correspond to the Order History Page in the Prensentation Layer of the Architechtural Layer Diagram




```
export default function OrderHistory() {
}
```

VII. Change Management

Additional Change

- Modify the place order function such that the Client can use Coupon for discount if the Client have received one.
- The Clients will be ranked based on their total spending on the website:
  - Spending lesser than 1 milion VND: Bronze
  - Spending between 1 to 5 million VND: Silver:
    - Discount 3% for foods only
  - Spending between 5 to 10 million VND: Gold
    - Discount 5% for foods only
  - Spending higher than 10 million VND: Diamond
    - Discount 10% for foods only

Changes made

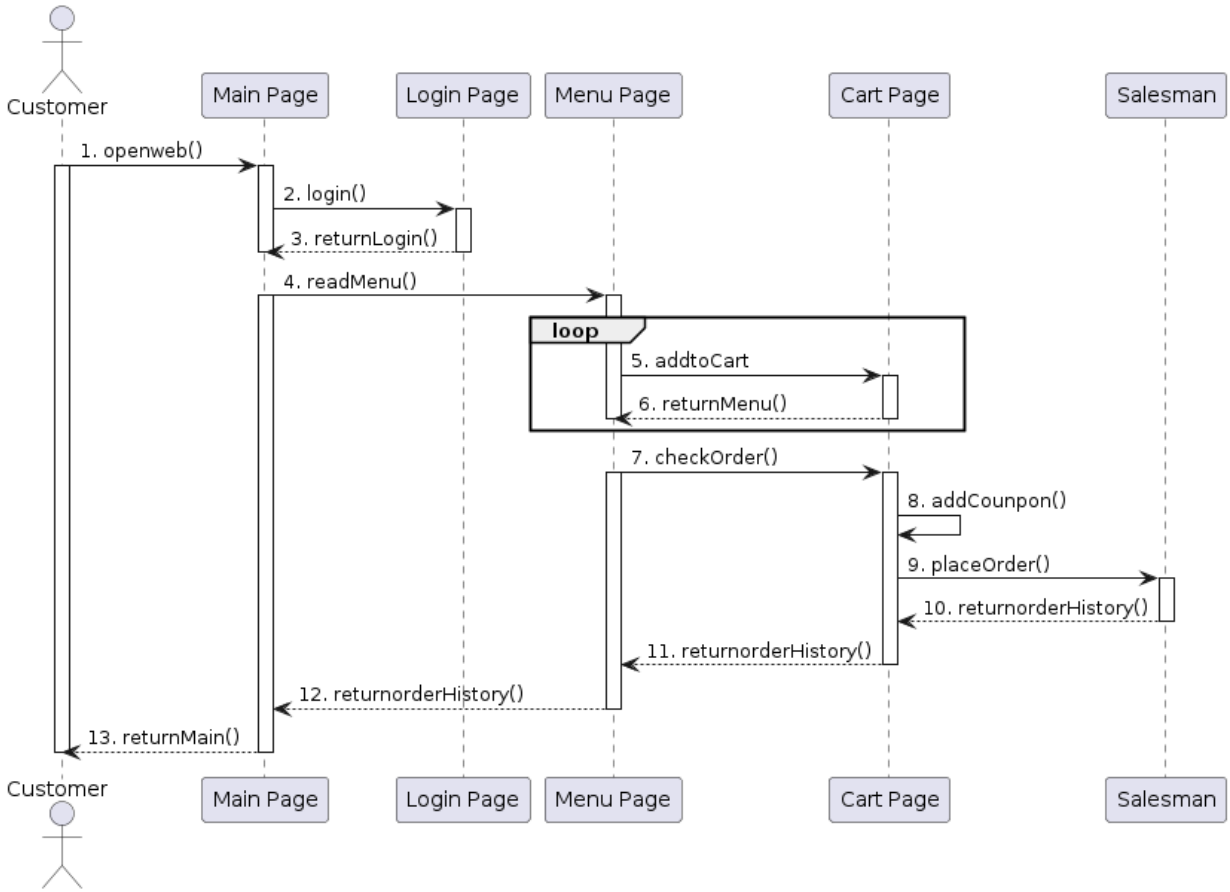
1. Use case diagram

- The Client can apply a coupon if he/she received one.
- This Coupon doesn't necessarily need to be applied, without it, the Client can still place an order.

2. Use case table

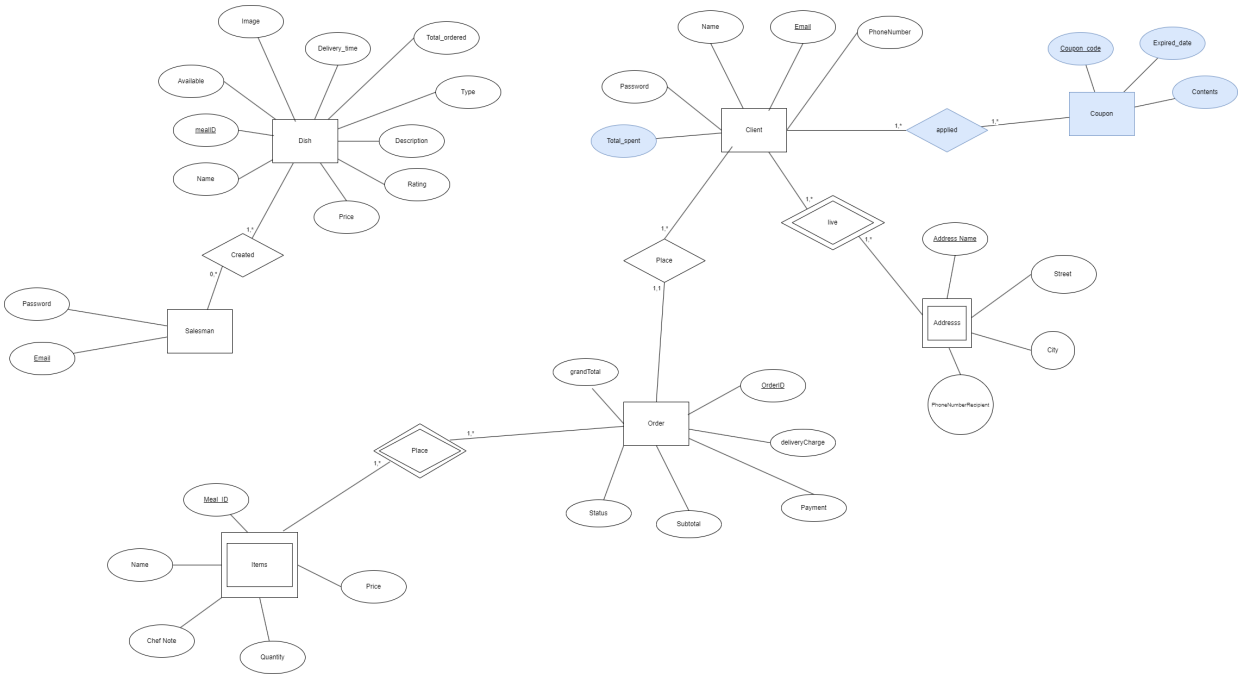
Name	Description
Actor	Client, Salesman
Main function	Place an Order
Brief description	The Client accesses the online menu, selects items to order, and submits the order. **The Client can apply a coupon before checkout, this coupon will give the Client discount on the current order.** It also notifies the Salesman about the new order.
Context information	Finish choosing item in the menu
Triggers	Clients must login to our webpage. Client must choose at least one item to add to their cart. **To use the coupon, Client must receive the Coupon code and the Coupon must be within expiration date.**
Flow of events summary	1. Client go to menu page to choose dishes to order. 2. Clients go to Cart page to see existing dishes and make some changes if needed. 3. Clients can add more item to the cart before going to the cart page for payment 4. Client chooses payment method and delivery address. **5. If Client have a Coupon, Client can input the Coupon code for discount** 6. Client click on button place order. 7. Salesman gets notify about Client's order and start to prepare the order. 8. System notify the Client that order has been placed.

3. Sequence Diagram

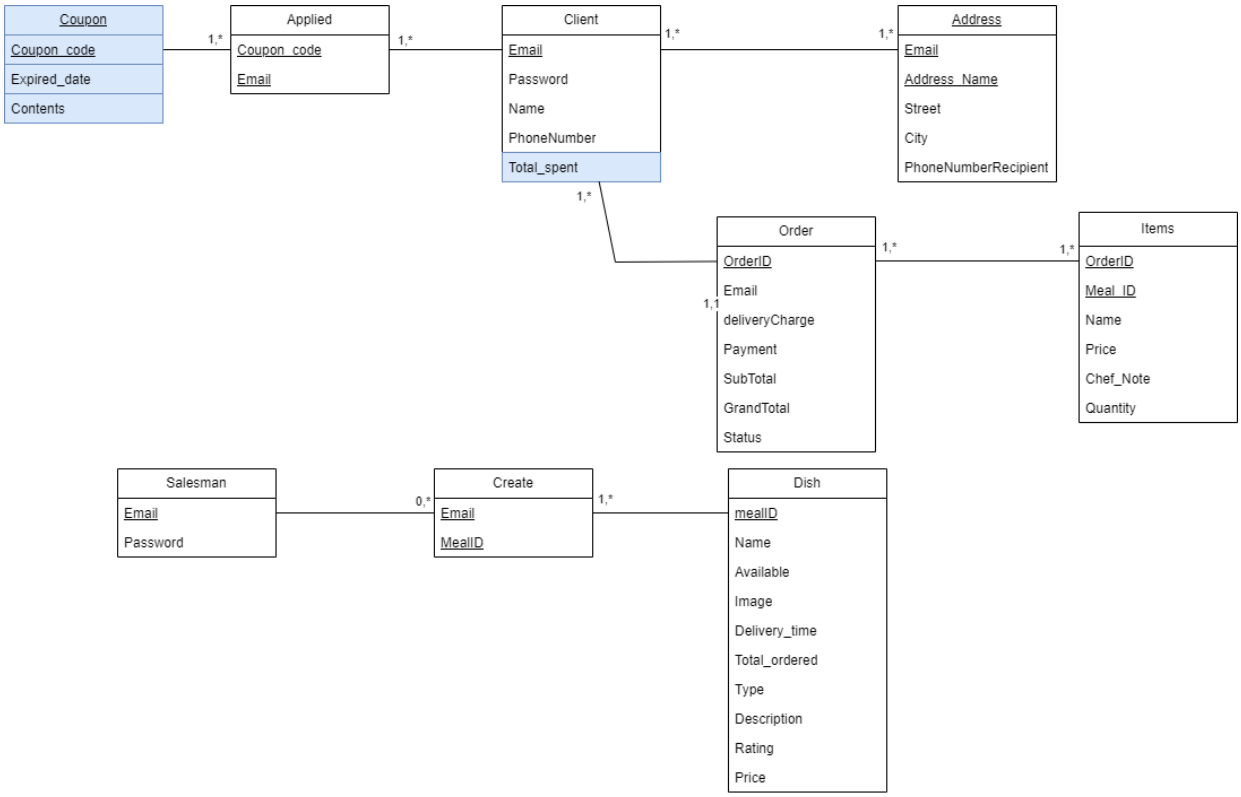


- If the Client have a Coupon, he/she can addCoupon() before placeOrder().
- This will give the Client's order a discount.

4. Database schema



- We added a new entity called Coupon:
  - Each record in the entity represents a record in Coupon database
  - One Coupon can be used by many Clients
  - One Clients can have many Coupons but Clients can only use 1 Coupon at a time.
  - The Coupon each have a coupon_id,which are unique for each coupon,expired_date and the content, for instance: 10% discount.

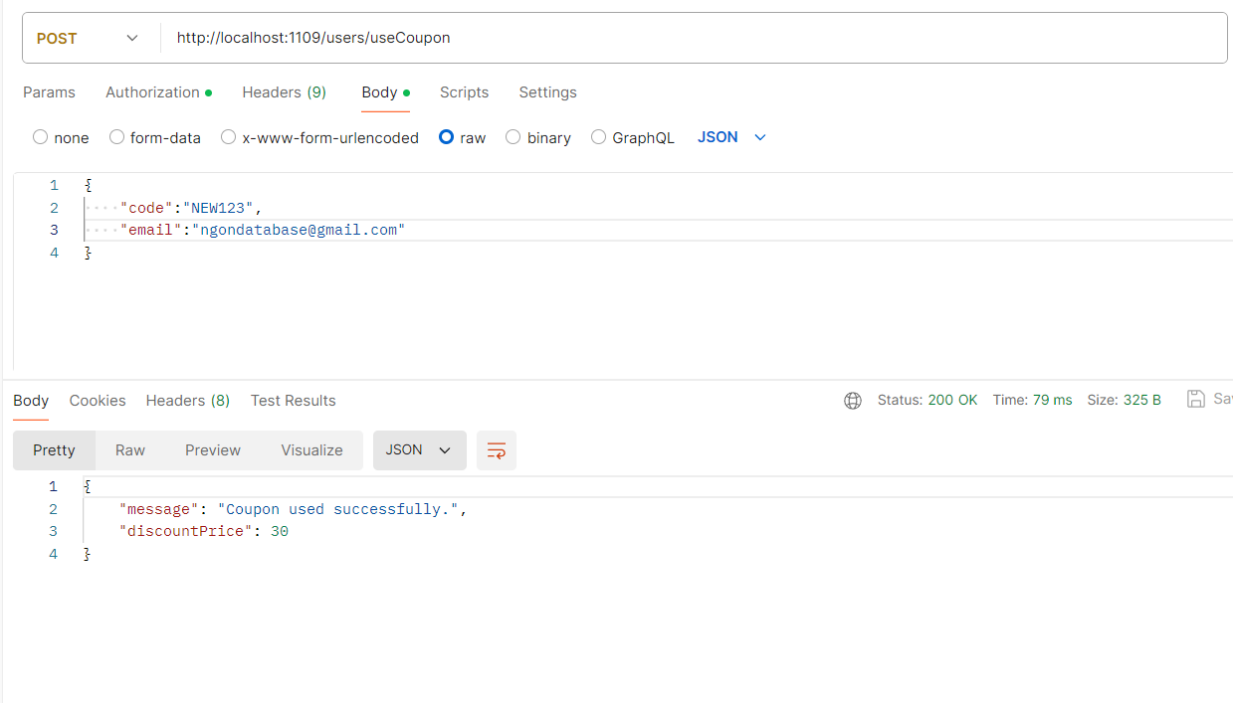


- Due to the new changes, we have modify the database as well.
- The coupon table is added, this table stores the information of each Coupon
    - Each Record represent a coupon
- The table applied is the relationship table between Client and Coupon
    - Each Record represent which Coupon uses by which Client.
- The table Client now have a new attribute called **"Total_spent"**
    - This help the API to ranked the customer depends on their total spent on our web.

5. Back-end

```
router.post('/users/useCoupon', auth, async (req, res) => {
  const { code, email } = req.body;

  // Validate the input
  if (!code || !email) {
    return res.status(400).json({ message: 'Coupon code and email are
required.' });
  }

  try {
    // Find the coupon by code
    const coupon = await Coupon.findOne({ code });

    if (!coupon) {
      return res.status(404).send({ message: 'Coupon not found.' });
    }

    // Use the coupon with the provided email
    // await coupon.useCoupon(email);
    if (!coupon.isValid(email)) {
      res.send({ message: 'Coupon is invalid or has been used.',
discountPrice: 0})    ;
    } else{
      res.send({ message: 'Coupon used successfully.', discountPrice:
coupon.discountPrice});}

  } catch (error) {
    if (error.message === 'Coupon is invalid or has been used the maximum
number of times.') {
      res.status(400).send({ message: error.message });
    } else {
      res.status(500).send({ message: 'Internal server error.' });
    }
  }
});
```

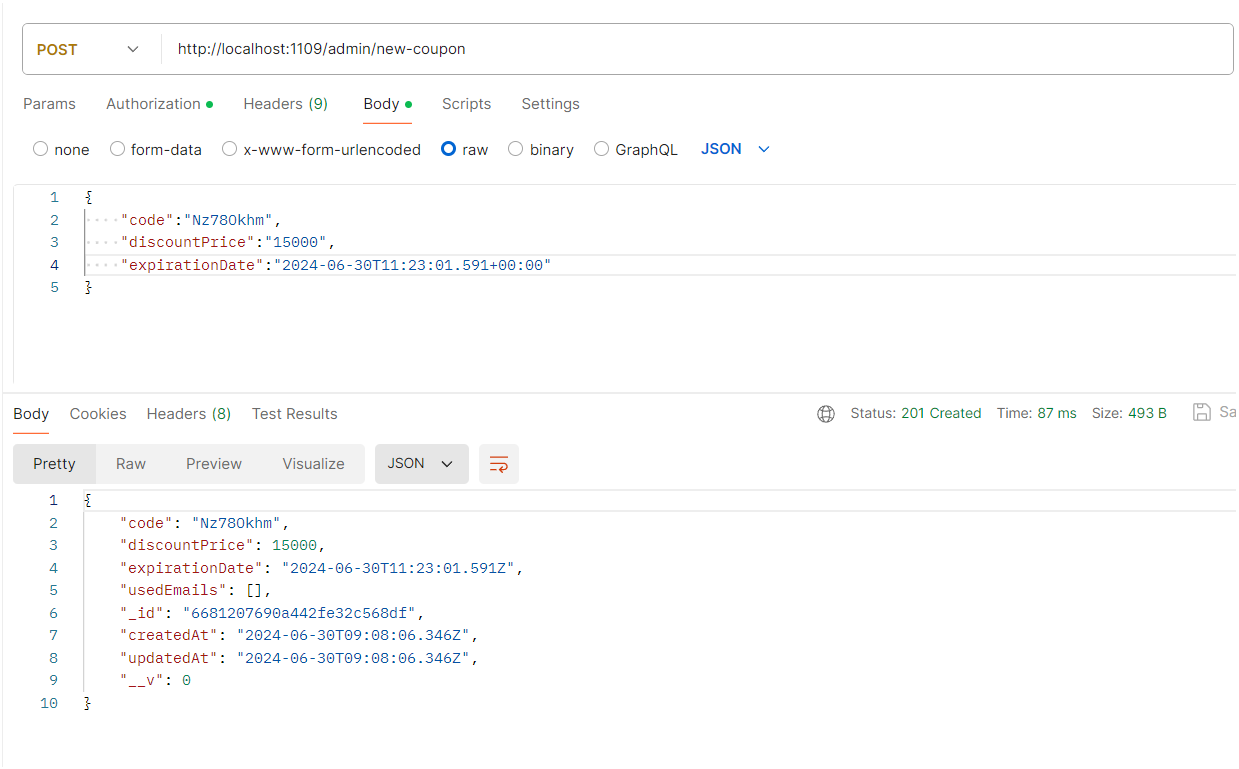
- This is the API useCoupon for the User
- When the User input a Coupon, it will check a few scenarios:
    - If the Coupon doesn't exists, return code 404 with message: "Coupon not found"
    - If the Coupon is valid, it will sent message: "Coupon used successfully." and will applied discount to the food immediately.
    - If the Coupon is invalid, it will return message: "Coupon is invalid or has been used."
- The code is surrounded with a try catch:
    - If an error occurs, it responds with a 500 status code and the message "Internal server error".
- Below is the API testing for useCoupon:




```
router.post('/admin/new-coupon', authAdmin, async (req, res) => {
  const { code, discountPrice, expirationDate } = req.body;
  console.log(req.body);
  // Validate the input
  if (!code || discountPrice == null || !expirationDate) {
    return res.status(400).send({ message: 'All fields are required.' });
  }

  try {
    // Create a new coupon
    const newCoupon = new Coupon({
      code,
      discountPrice,
      expirationDate
    });

    // Save the coupon to the database
    const savedCoupon = await newCoupon.save();
    res.status(201).send(savedCoupon);
  } catch (error) {
    // Handle duplicate code error
    if (error.code === 11000) {
      res.status(409).send({ message: 'Coupon code already exists.' });
    } else {
      res.status(500).send({ message: 'Internal server error.' });
    }
  }
});
```

- This API defines a route that handles POST requests to the URL path.
- authAdmin: This is a middleware function that likely checks if the request is being made by an authenticated admin user. Only if this middleware passes will the request proceed to the next step.

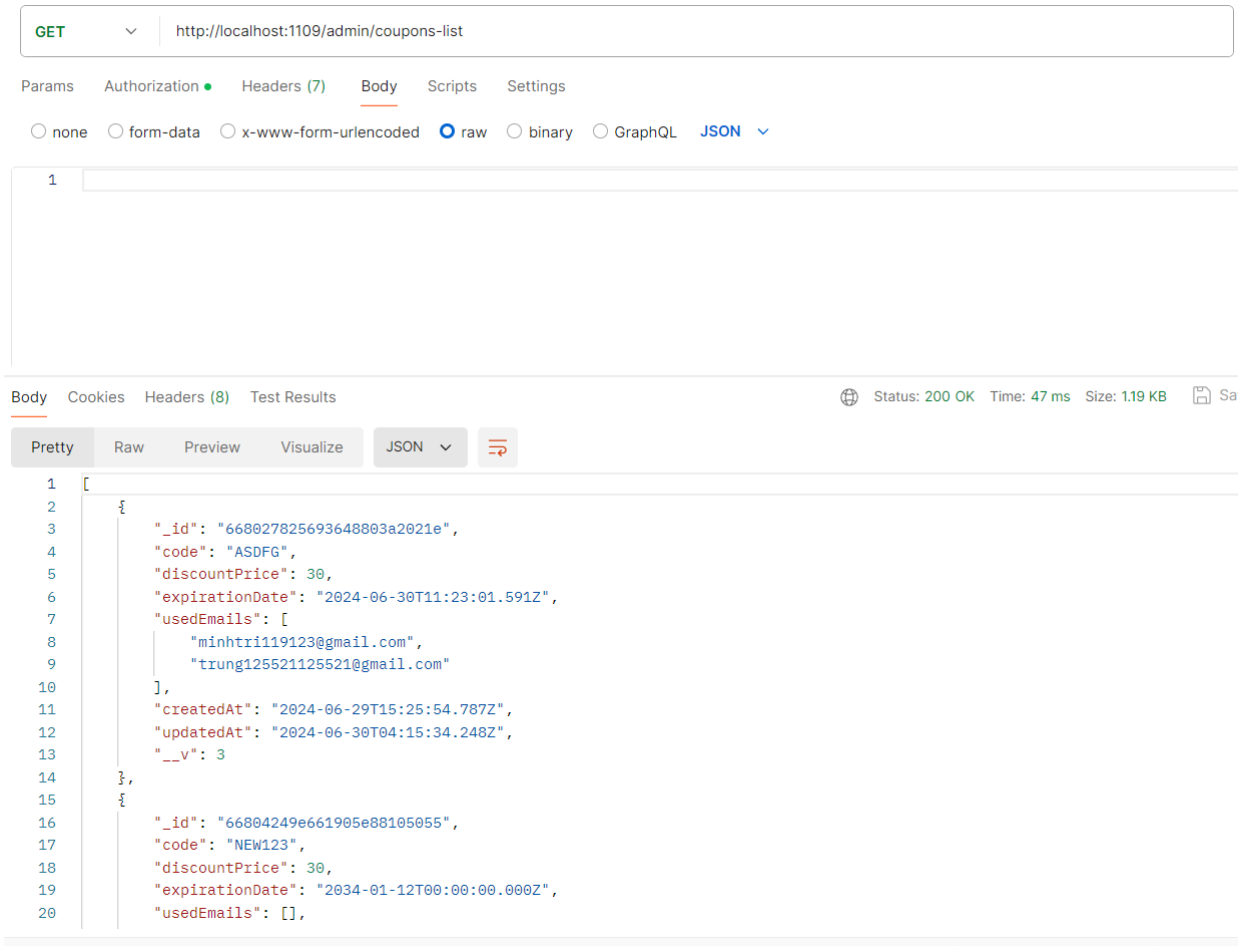
- The code, discountPrice, and expirationDate properties from the request body are being extract.
- If any of the required fields (code, discountPrice, expirationDate) are missing or invalid, the message "All fields are required." and status code 400 is return.
- A new instance of the Coupon model is created with the provided coupon code, discountPrice, and expirationDate.
- The new coupon is saved to the MongoDB database.
    - The await keyword ensures that the function waits for the save operation to complete before proceeding.
- savedCoupon: This variable will hold the saved coupon document.
- If the coupon is saved successfully, a 201 status code (Created) and the saved coupon document are sent to the client.
- If any error occurs during the execution of the try block, the catch block will catch it




```
router.get('/admin/coupons-list', authAdmin, async (req, res) => {
  try {
    const coupons = await Coupon.find({});
    res.send(coupons);
  } catch (error) {
    res.status(500).send({ message: 'Internal server error.' });
  }
});
```

- This defines a route that handles GET requests to the API couponlist
- authAdmin: This is a middleware function that likely checks if the request is being made by an authenticated admin user. Only if this middleware passes will the request proceed to the next step.
- The route handler is defined as an asynchronous function using the async keyword.
    - This allows the use of await within the function to handle asynchronous operations.

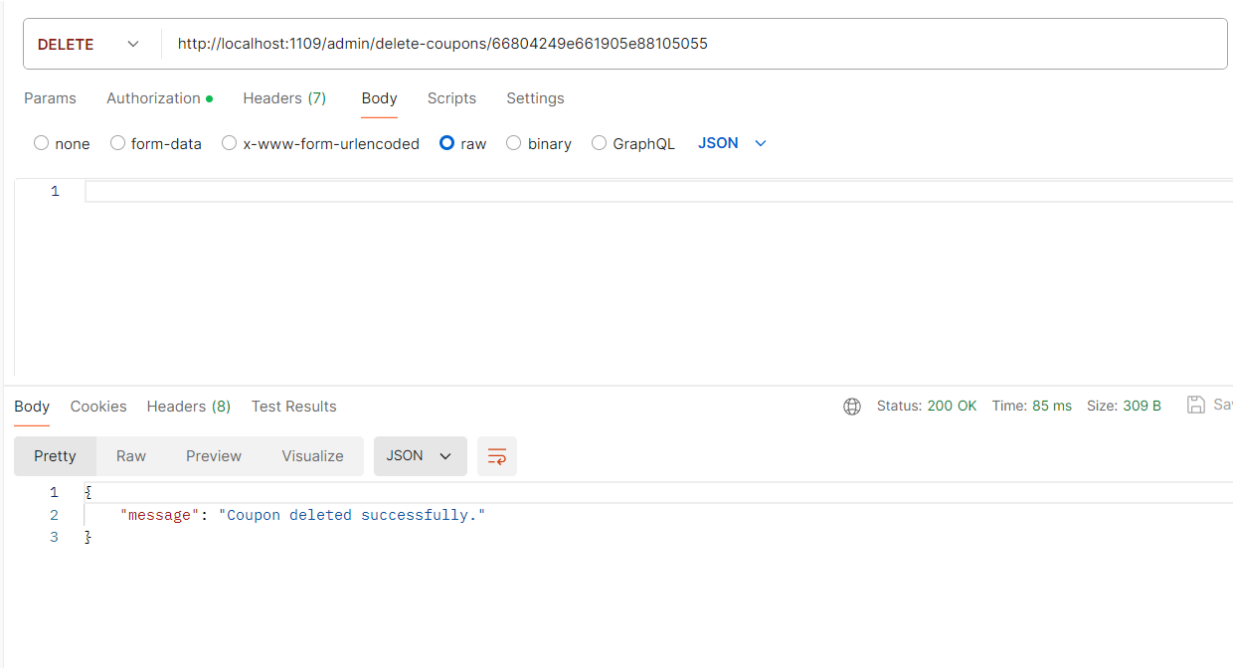
- Coupon.find({}): This line queries the MongoDB database for all documents in the Coupon collection.
    - The await keyword ensures that the function waits for the database operation to complete before proceeding.
- coupons: This variable will hold the array of coupon documents retrieved from the database.
- If the database query is successful, the array of coupons is sent back to the client in the response.
- If any error occurs during the execution of the try block, the catch block will catch it.
    - res.status(500).send({ message: 'Internal server error.' }): This sends a response with a 500 status code (Internal Server Error) and a message indicating an internal server error




```
router.delete('/admin/delete-coupons/:id', authAdmin, async (req, res) => {
  try {
    const { id } = req.params;
    const deletedCoupon = await Coupon.findByIdAndDelete(id);
    if (!deletedCoupon) {
      return res.status(404).send({ message: 'Coupon not found.' });
    }
    res.status(200).send({ message: 'Coupon deleted successfully.' });
  } catch (error) {
    res.status(500).send({ message: 'Internal server error.' });
  }
});
```

- This API defines a route that handles DELETE requests to the URL path.

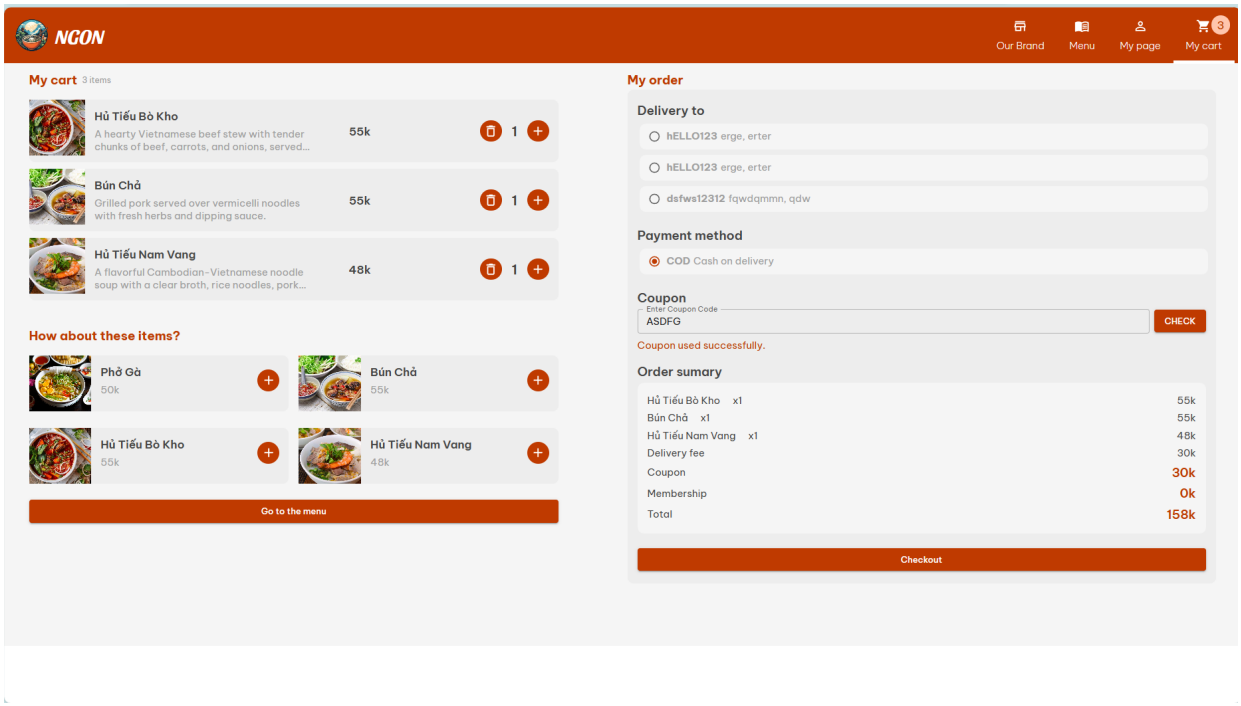
  - `:id`: This is a route parameter that will capture the ID of the coupon to be deleted.
  - authAdmin: This is a middleware function that likely checks if the request is being made by an authenticated admin user. Only if this middleware passes will the request proceed to the next step.
- The route handler is defined as an asynchronous function using the async keyword.
  - This allows the use of await within the function to handle asynchronous operations.
- This line extracts the id parameter from the request URL.
  - The id represents the unique identifier of the coupon to be deleted.
- Coupon.findByIdAndDelete(id): This line queries the MongoDB database to find and delete the document with the specified id from the Coupon collection.
  - The await keyword ensures that the function waits for the database operation to complete before proceeding.
- deletedCoupon: This variable will hold the deleted coupon document, or null if no document with the specified id was found.
  - If deletedCoupon is null, it means no coupon with the specified id was found in the database. In this case, a 404 status code (Not Found) and an appropriate message are sent to the client.
  - If the coupon was found and deleted successfully, a 200 status code (OK) and a success message are sent to the client.
  - If any error occurs during the execution of the try block, the catch block will catch it.
- res.status(500).send({ message: 'Internal server error.' }): This sends a response with a 500 status code (Internal Server Error) and a message indicating an internal server error.



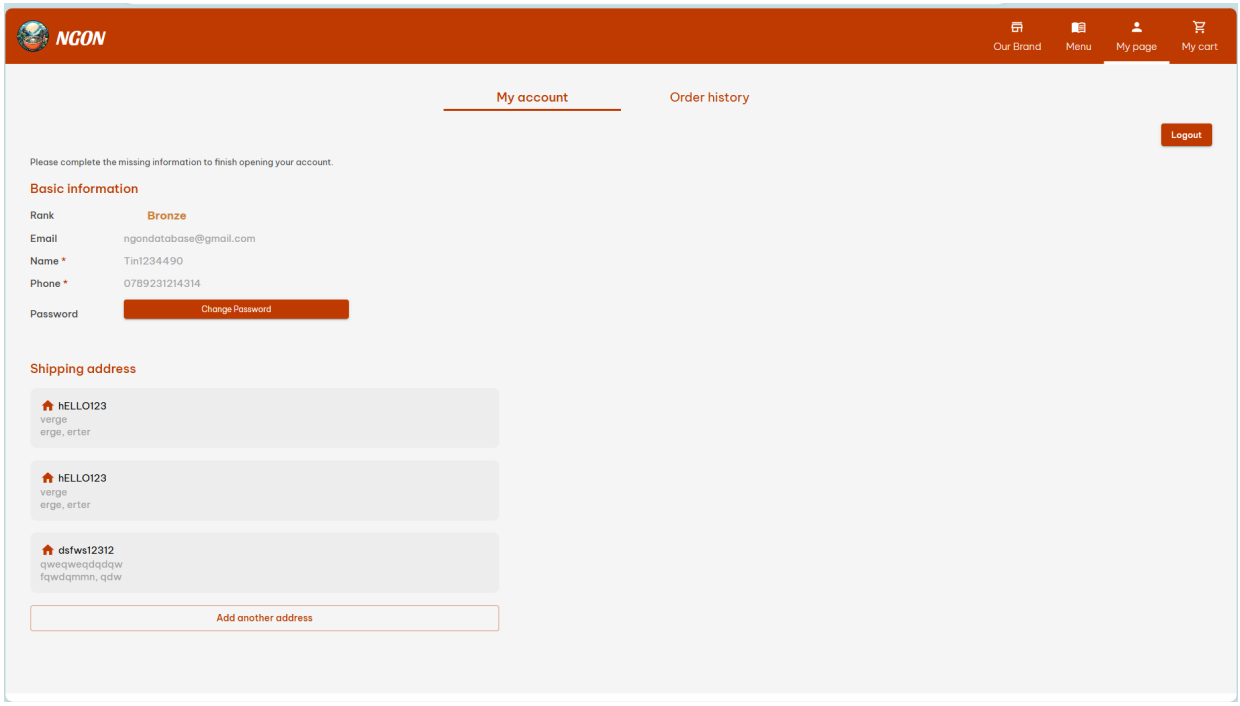
6. Front-end

- For the front-end, we add a coupon slot for the user to put their coupon before place an order.

```
<div className="my-order-coupon title-top">
  <p className="my-order-title">Coupon
    </p>
      <FormControl>
    </FormControl>
```


NGON
Our Brand
Menu
My page
My cart
3
My cart 3 items
Hủ Tiếu Bò Kho
A hearty Vietnamese beef stew with tender chunks of beef, carrots, and onions, served...
55k
1
Bún Chả
Grilled pork served over vermicelli noodles with fresh herbs and dipping sauce.
55k
1
Hủ Tiếu Nam Vang
A flavorful Cambodian-Vietnamese noodle soup with a clear broth, rice noodles, pork...
48k
1
How about these items?
Phở Gà
50k
Bún Chả
55k
Hủ Tiếu Bò Kho
55k
Hủ Tiếu Nam Vang
48k
Go to the menu
My order
Delivery to
hELLO123 erge, erter
hELLO123 erge, erter
dsfws12312 fqwdqmmn, qdw
Payment method
COD Cash on delivery
Coupon
Enter Coupon Code
ASDFG
CHECK
Coupon used successfully.
Order sumary
Hủ Tiếu Bò Kho x1 55k
Bún Chả x1 55k
Hủ Tiếu Nam Vang x1 48k
Delivery fee 30k
Coupon 30k
Membership 0k
Total 158k
Checkout

```
  <p>Rank</p>

const [rank, setRank] = useState("");

useEffect(() => {
  const fetchTotalSpending = async () => {
  try {
    const response = await userApi.getTotalSpending(user.email);

    const totalSpending = response.data.totalSpending;
    if (totalSpending < 1000) {
      setRank("Bronze");
    } else if (totalSpending < 5000) {
      setRank("Silver");
    } else if (totalSpending < 10000) {
      setRank("Gold");
    } else {
      setRank("Diamond");
    }
    } catch (error) {
     console.error("Error fetching total spending:", error);
  }
};

  fetchTotalSpending();
}, []);

const [memberShip, setMemberShip] = useState(0);

  useEffect(() => {
    const fetchTotalSpending = async () => {
      try {
        const response = await userApi.getTotalSpending(user.email);
        console.log(response);
        const totalSpending = response.data.totalSpending;
        if (totalSpending < 1000) {
          setMemberShip(0);
        } else if (totalSpending < 5000) {
          setMemberShip(3);
        } else if (totalSpending < 10000) {
          setMemberShip(5);
        } else {
          setMemberShip(10);
        }
      } catch (error) {
        console.error("Error fetching total spending:", error);
      }
    };

    fetchTotalSpending();
  }, []);
```


NGON
Our Brand
Menu
My page
My cart
My account
Order history
Logout
Please complete the missing information to finish opening your account.
Basic information
Rank
Bronze
Email
ngondatabase@gmail.com
Name *
Tin1234490
Phone *
0789231214314
Password
Change Password
Shipping address
hELLO123
verge
erge, erter
hELLO123
verge
erge, erter
dsfws12312
qweqweqdqdqw
fqwdqmmn, qdw
Add another address

VIII. Security

- Our webpage employs a hashing technique to securely encrypt clients' passwords. This encryption ensures that even in the event of a database breach, the passwords remain protected and inaccessible to attackers. Additionally, we have implemented robust authentication protocols for clients who wish to change their passwords or when they first signup on our website.

Advantage of our security:

1. Password Protection:
    - Hashing ensures that passwords are not stored in plain text.
    - Even if an attacker gains access to the database, they won't immediately obtain the original passwords.
2. Resistance to Rainbow Tables:
    - Hashing makes it difficult for attackers to use precomputed tables (rainbow tables) to reverse-engineer passwords.
    - Each password is hashed uniquely, preventing easy lookups.
3. Data Integrity:
    - Hashing provides data integrity.
    - If a password is modified, the hash changes, alerting the system to potential tampering.
4. Salting:
    - By adding a unique salt to each password before hashing, the system further enhances security.
    - Salting prevents attackers from using precomputed tables even if they know the hashing algorithm.
5. Performance:
    - Hashing is computationally efficient during password verification.
    - It doesn't require expensive decryption steps.

Disadvantage of our security:

1. No Recovery:
    - Hashed passwords cannot be recovered.
    - If a user forgets their password, the system can't retrieve it.
    - Instead, we have password reset mechanisms which require authentication via email.
2. Collision Risk:
    - Although rare, hash collisions (two different inputs producing the same hash) can occur.
    - Properly chosen hash algorithms minimize this risk.
    - To decrease the chance of this happening, every password created must be complex (required at least 8 character, 1 uppercase , 1 special character)
3. Limited Complexity:

- Hashing doesn’t inherently enforce password complexity rules (e.g., minimum length, character types).
- It’s essential to validate passwords before hashing.

4. Single Iteration:
    - Basic hashing uses a single iteration, which can be vulnerable to brute-force attacks.
    - Modern systems use multiple iterations (key stretching) to mitigate this.
5. Algorithm Vulnerabilities:
    - The choice of hashing algorithm matters.
    - Some older algorithms (e.g., MD5, SHA-1) are no longer secure due to vulnerabilities.
    - Use modern, cryptographically strong algorithms (e.g., bcrypt, Argon2)

Access Control:

- The Client:
    - The Client are Users who have created an Account on our Web page.
    - If the User don't have an account, they cannot place an order or view the cart.
    - The Client can only view the Menu, view cart, place an order, view previous orders as well as edit their own personal information.
- The Salesman:
    - The Salesman account are created by the technician, which involves modification on the actual database itself.
    - We don't provide an UI for Salesman registration to further enhanced security.
    - The Salesman can see all Clients who have an account on our webpage, he can also sees the Client personal information like phone number and name.
    - The Salesman cannot see the Client's Password since it has been hash.
    - The Salesman cannot see the Client's card informations as well.
    - The Salesman can see the total amount of money each Client spent on our website, he can also see the website revenue and the total profit for each type of dish.
    - The Salesman can also edit, add and delete a new dish on the menu.
- Other Users or Unkown Users:
    - With out an Acoount, these Users can only view our Menu, our main page as well as our Branding.
    - The Users also knows other social accounts that we have on our website.

Password recovery

- When the Client forgot their password, the Client will uses our Forgot password function
    - This requires the Client to first input the email address that they use to create the Account on our website.
    - Than the server will sent an authentication email to the Client via their email address.
    - The Client than have to input the authentication pin with 6 characters on our website before the Client can input the new password.

IX. Testing

Test Case ID	Test Steps	User Input	Expected Output	Actual Result	Execution Status
Signup Function	1. User open the signup UI 2. User input the email 3. User input the password 4. User input OTP code 5. User press the signup button	User input email and password and also check their mail box for OTP code	Display a popup informing signup successful.	The website will send verification code to your email, and redirect you to the OTP Verification Page. After input the OTP, a pop up will notify verify successfully and redirect to the login page	pass
Login Function	1. User open the Login Menu 2. User input the email and password that has been signup 3. User press the Login button	User registered email and password	User will be redirect to the Main Page	It redirect you directly to your account's profile.	pass
Add to cart	1. User opens the menu 2. User click on the plus icon to add the dish to the cart 3. User click on the cart icon on the top right	- User click on the plus icon to add dish - User click on the Cart Icon	The choosen dish should appear in the Cart UI	The chosen dish will be add to the cart and can be seen with a picture, description and its price	pass

Test Case ID	Test Steps	User Input	Expected Output	Actual Result	Execution Status
Modification dish in Cart	1. User have already choosen a dish in the cart 2. User click on the Cart Icon to go to the Cart UI 3. User click on the plus to change the quantity of the dish order 4. User can remove the dish from the cart.	- User click on the plus of the dish in the cart UI - User click on the trash icon on the dish in the Cart UI	- The dish quantity should increase if the User click on the plus icon - The dish should be remove from the cart when click on the trash icon.	You can increase the quantity or remove the existing meals on the cart. The subtotal of the cart will fluctuate based on the cart modification	pass
Cancel Order Function	1. User go to the menu 2. User click 3. User go to the Cart Page 4. User choose the payment method 5. User choose the delivery address 6. User click the Order button 7. User go to the Order History Tab 8. User click on the trash icon	User create an Order and User go to the Order History Function	The User's Order which just been placed should be deleted	The User will be navigated to the Order History site and the order which has been placed will be labeled as Canceled	pass

Test Case ID	Test Steps	User Input	Expected Output	Actual Result	Execution Status
Place Order function	1. User go to the menu 2. User click 3. User go to the Cart Page 4. User choose the payment method 5. User choose the delivery address 6. User click the Order button	- User provide delivering address - User choose payment method - User click on the order button	- The order detail should be display on the User's Screen. - When go to the order history page, the User should see their order	The popup message appears after placing an order informing that you have successfully place an order. When the User go to the Order History Page, the Order will show up and be in pending state.	pass
Forgot password function	1. User click on the forget password link 2. User input the email address of the lost account 3. User check their email inbox for the authentication key 4. User input the authentication 5. User input the new password.	- User provide the registered emain - User provide the authentication key - User provide the new password	- The User should be able to login using the new password	The user will provide the website the email that used for sign up , then it will send the verification code and redirect to the OTP Verification page. After filling the OTP, the user can change the password.	pass

Test Case ID	Test Steps	User Input	Expected Output	Actual Result	Execution Status
View Restaurant Menu Function	1. User click on the Menu page 2. User click on other tabs of the menu page 3. User click on a dish to see the dish's description	- User click on the menu - User click on the dish	- The dish detail should be displayed when click on the dish - The User should be able to see all the menu and it's content.	The menu contains many dishes along with their description, price, category, rating and delivery time. There also have buttons which can change dish type. Each dish types has various dishes in it.	pass
Add new item to the menu function	1. The Salesman login to the Salesman account 2. The Salesman go to the Menu page 3. The salesman click on the New button 4. The Salesman input the information of the new dish 5. The Salesman click on the Save button.	- Salesman input the new dish description	- The User should see the new dish when the Salesamn add new item to the menu	The new item edit page will appear when you press “+New” in the menu section on the screen which let you add the information of a dish. After adding all the necessary information, the new dish will show up on the menu. We can click on the dish agian and update the exsiting information, this is update dish.	pass

Test Case ID	Test Steps	User Input	Expected Output	Actual Result	Execution Status
The view report function	1. The Salesman login to the Salesman's account 2. The salesman go to the dashboard tab	The Salesman click on the Dashboard tab	All the sales statistics should appear here when there is an order.	This is a dashboard that shows all the information involving the restaurant revenue. This included the number of times a dish is ordered, the total amount of registerd Clients, the total revenue of the restaurant, a bar chart that visualize which type of dish is the best seller, the total items currently on the menu and the total of order in state delivered	pass