



**Project Title: Online Food Ordering Web Application**

**Team Members:**

1. Muhammad Usman
2. Aravind Reddy
3. Ritwik Reddy
4. Damodar Reddy

## The Story:

Motivation behind developing this application is to facilitate the people with this online application. During the Covid-19 time the e-commerce companies like amazon, boosted their business more than we think about can be and not only it facilitates the business owners to boost their business, also it facilitates the customers to get the thing by just clicking a few buttons. Our application is facilitating the customers in the same way, during and after the covid time. Let's understand the customers. We are targeting both restaurant owners and as well as food buyers. For restaurant owners our app will help them to boost their business by maximizing the reach of their hotel miles away to customers. Unlike Social media marketing, where the customers will only get to know about the business. Our application will not only marketize their business, also it will allow their target audience to buy the food from their restaurants. For food buyers, this will help allow them to get their favorite food by sitting in their home or in the school and office. In this way buyers will be saving their time and commute cost. This application is having the best features that will results in both sellers and buyers benefit.

## The Data:

The data we used in our application is mostly string format data that is the name of customers and restaurants and their corresponding menus. All the restaurant data was collected from the google by visiting each restaurant website.

We used NoSQL database, so our data is non-relational data. To give the high-level overview of over data schema, we have attached the schema of one restaurant from our database.

```
4 ▾ {  
5   "id": "66330",  
6   "name": "Jimmy John's Sandwiches",  
7   "address": "9201 E State Rte 350, Raytown, MO 64133, USA",  
8   "cuisines": "Gourmet Sandwiches",  
9   "rating": "4.4",  
10  "reviews": "865",  
11  "feature_image": "https://img.cdn4dd.com/cdn-cgi/image/fit=cover,wi  
12  "thumbnail_image": "https://img.cdn4dd.com/cdn-cgi/image/fit=cover,wi  
13  "menu": [  
14    {  
15      "id": 663307,  
16      "name": "SPICY EAST COAST ITALIAN",  
17      "desc": "A tasty Italian sandwich, made with a double serving of  
18      "price": 12  
19    }, {  
20      "id": "899166",  
21      "name": "BEACH CLUB",  
22      "desc": "Made with tasty hand-sliced turkey breast, provolone ch  
23    }]  
}
```

## Application Screen Shots.

### 1. Login/Registration

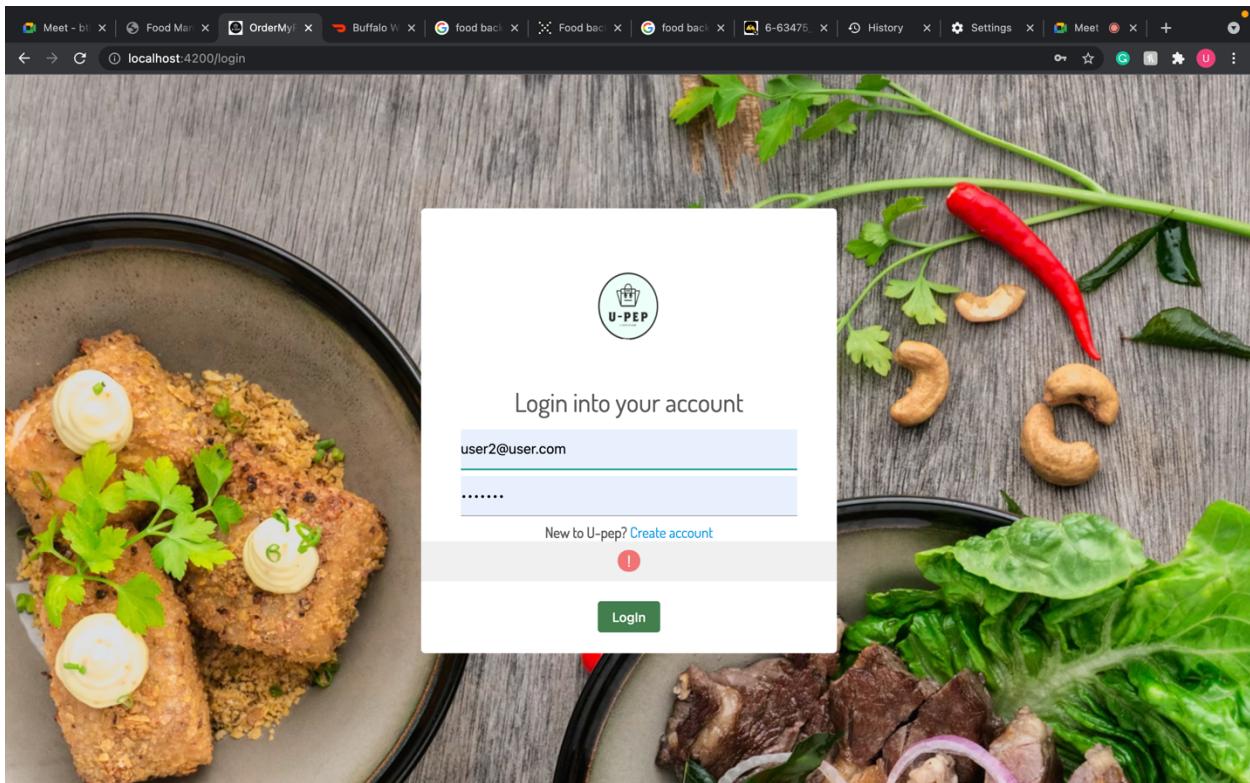


Figure 1: User Login/Registration

## 2. Home Page

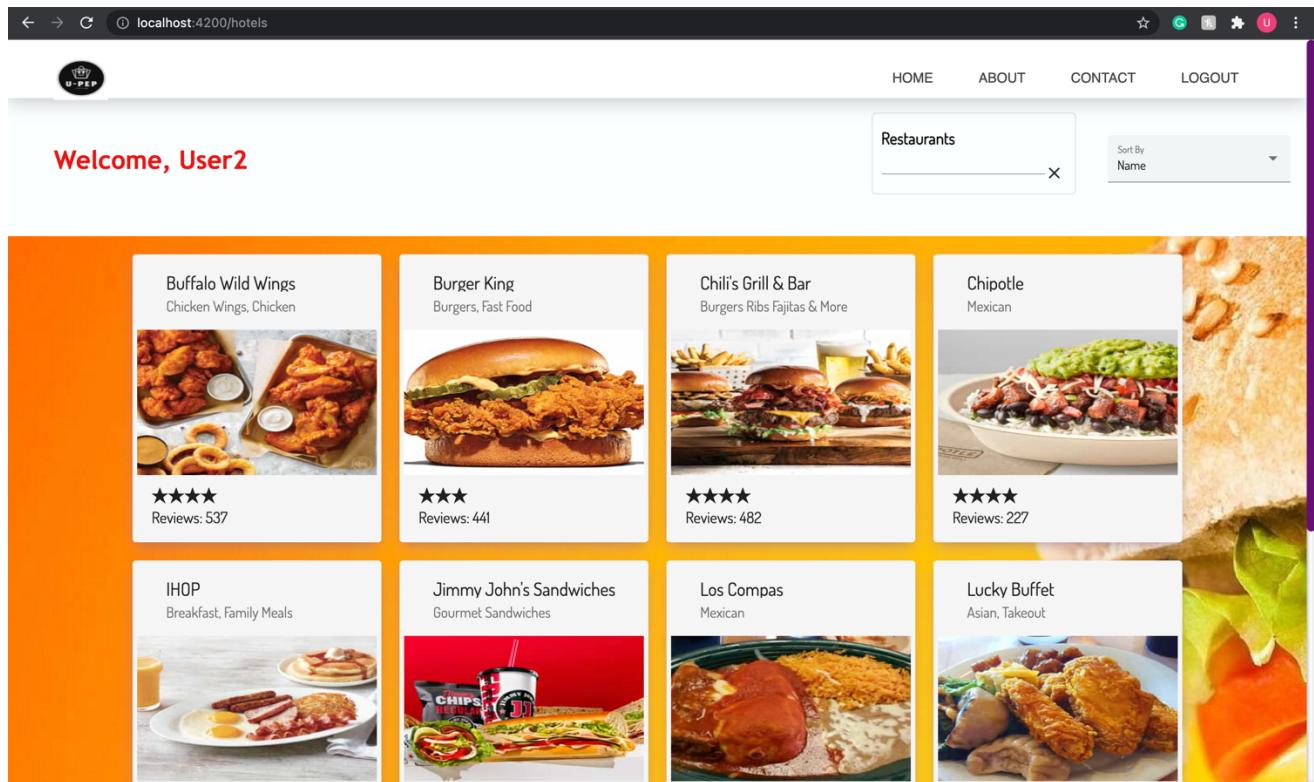


Figure 2: Available Hotels

### 3. Menu Page

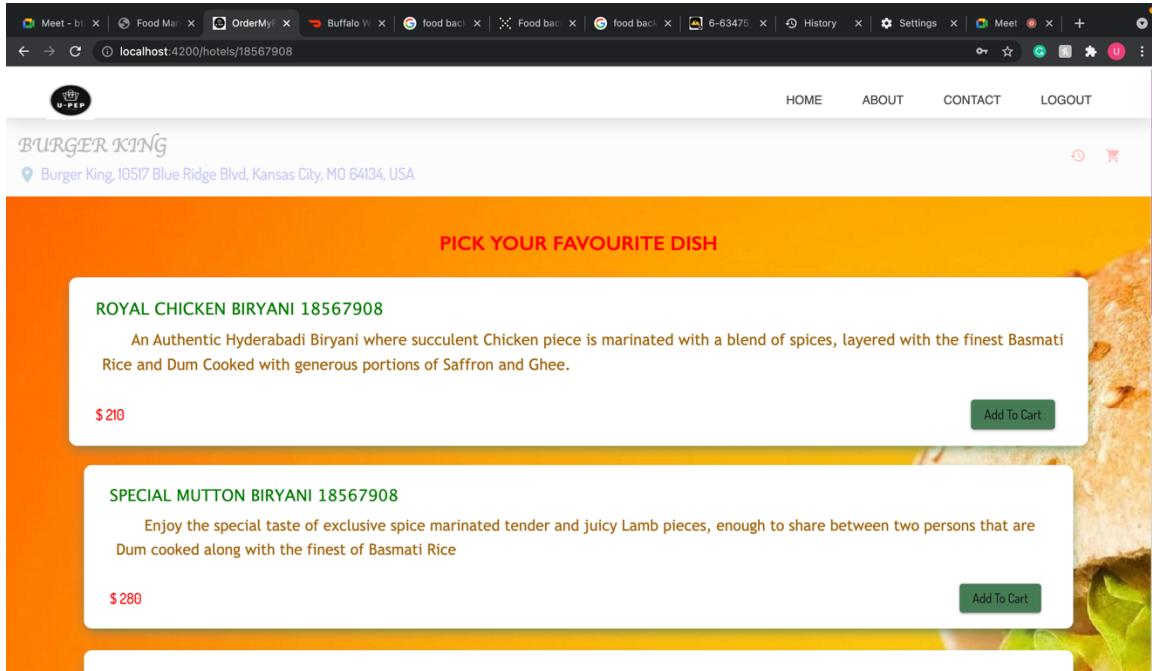


Figure 3: List of menus

#### 4. Cart

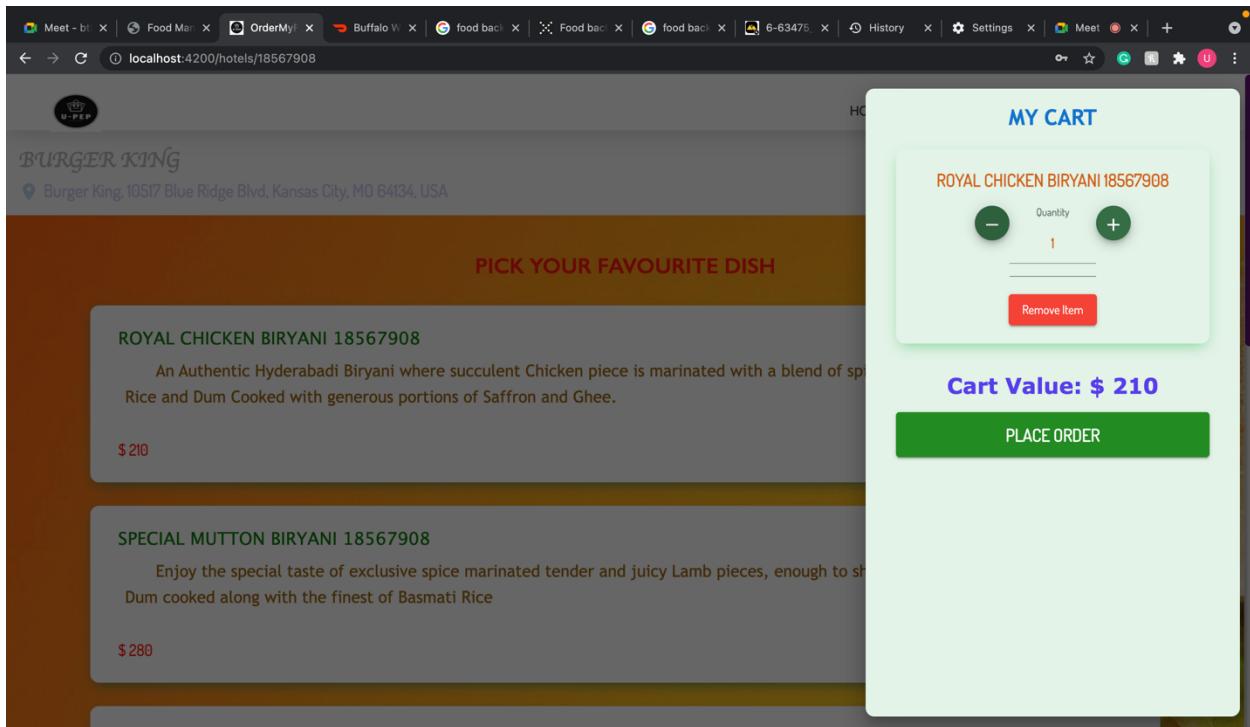


Figure 4: Cart

## 5. Chat Bot

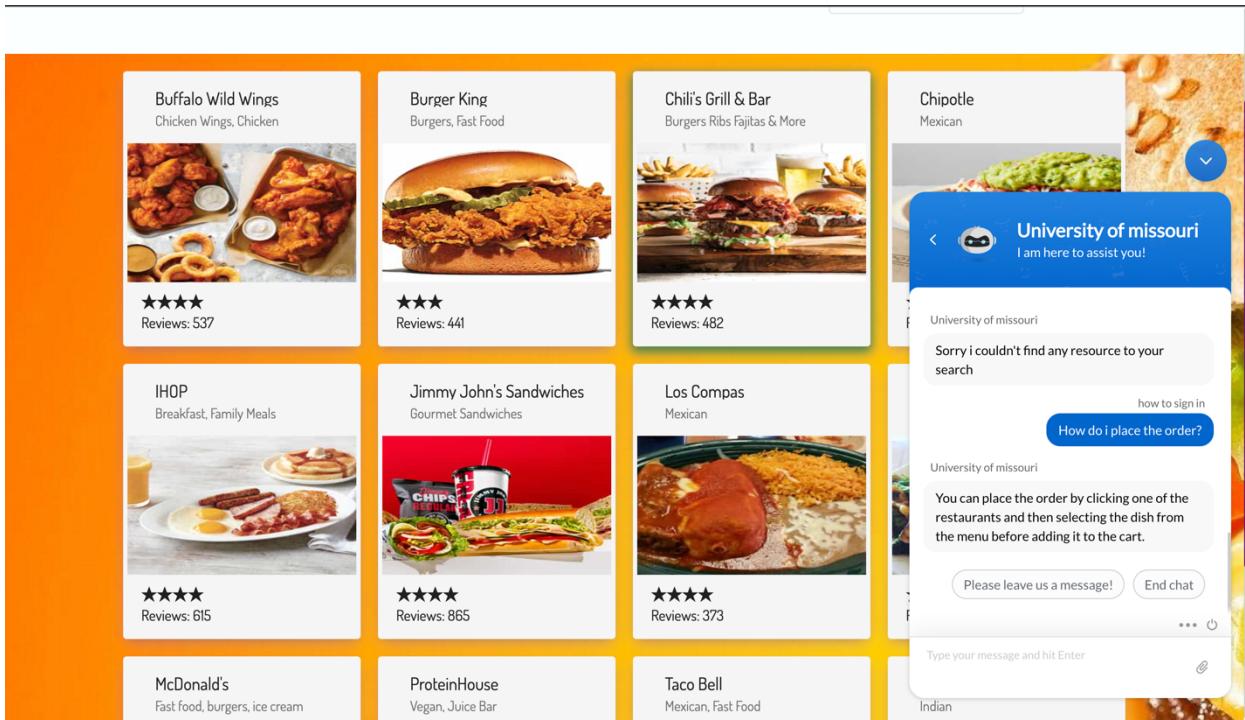


Figure 5: Chatbot

## 6. Order Confirmation

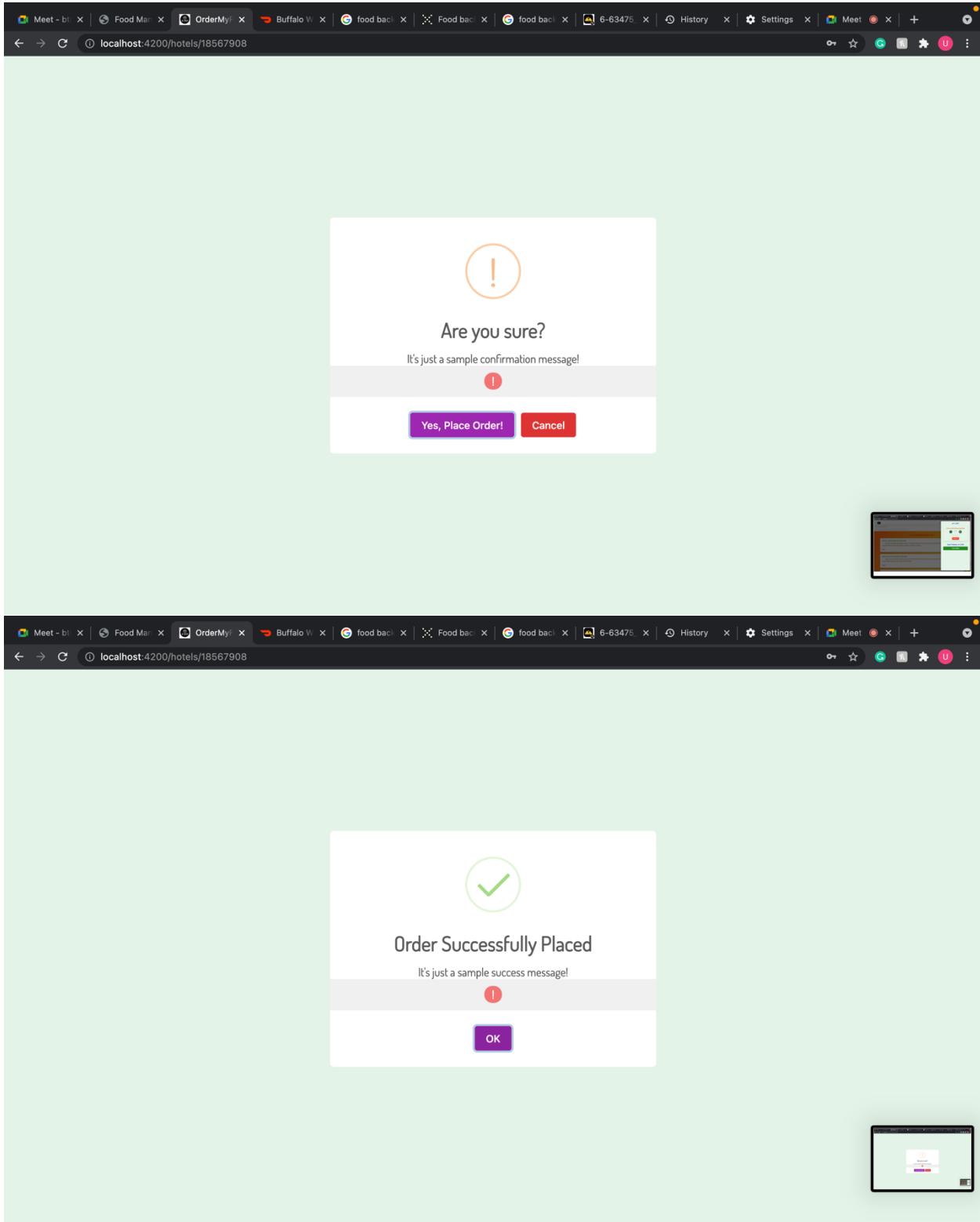


Figure 6: Order Confirmation

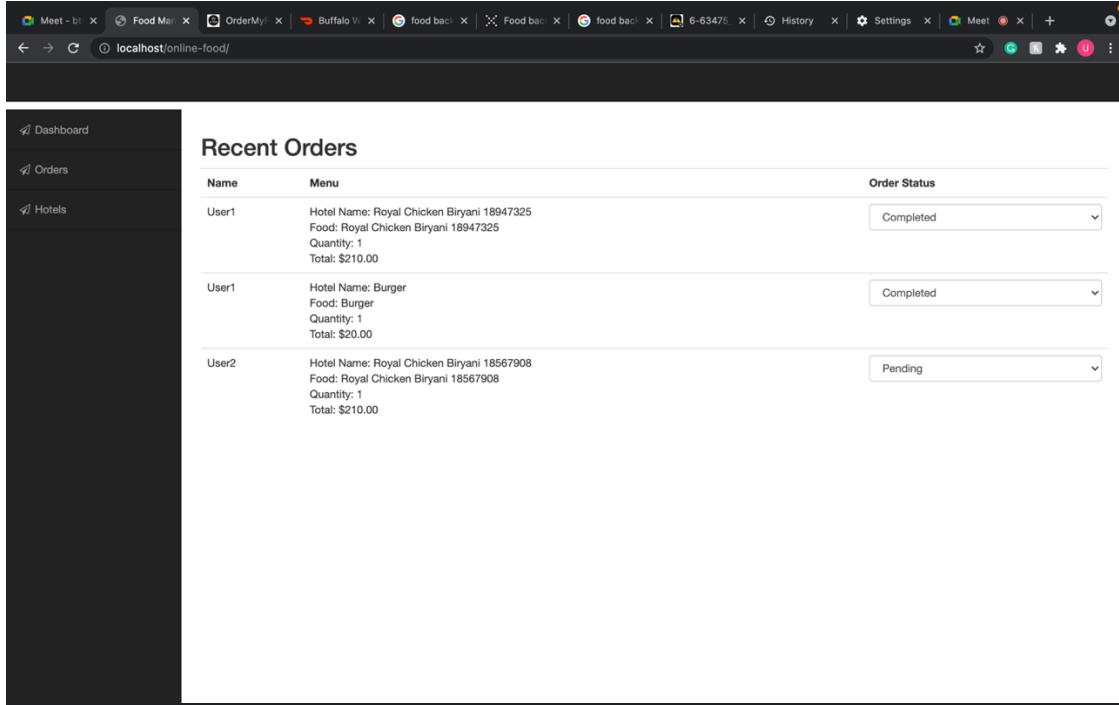
## 7. Admin Panel

The screenshot shows a web browser window with the URL `localhost/online-food/dashboard.php`. The page has a dark sidebar on the left with icons for Dashboard, Orders, and Hotels. The main area features three summary cards: 'Customers' (2), 'Hotels' (12), and 'Orders!' (3). Below these is a section titled 'Hotels' with a table listing five establishments:

Name	Address	View
Jimmy John's Sandwiches	9201 E State Rte 350, Raytown, MO 64133, USA	<a href="#">View Menu</a> <a href="#">Delete Hotel</a>
Buffalo Wild Wings	1806 NW Chipman Rd, Lee's Summit, MO 64081	<a href="#">View Menu</a> <a href="#">Delete Hotel</a>
Taj Mahal	7521 Wornall Rd, Kansas City, MO 64114, USA	<a href="#">View Menu</a> <a href="#">Delete Hotel</a>
Lucky Buffet	2931 S Noland Rd, Independence, MO 64055, USA	<a href="#">View Menu</a> <a href="#">Delete Hotel</a>
ProteinHouse	1345 Main St, Kansas City, MO 64105, USA	<a href="#">View Menu</a> <a href="#">Delete Hotel</a>

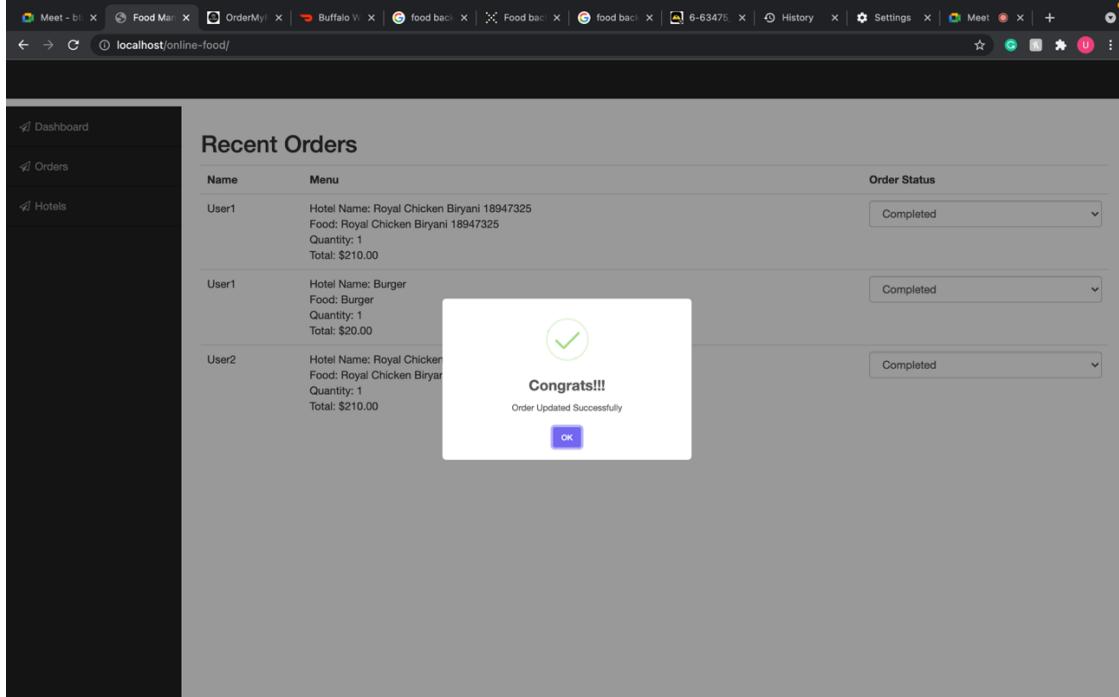
Figure 7: Admin Panel Dashboard

## 8. Order Management Page



**Recent Orders**

Name	Menu	Order Status
User1	Hotel Name: Royal Chicken Biryani 18947325 Food: Royal Chicken Biryani 18947325 Quantity: 1 Total: \$210.00	Completed
User1	Hotel Name: Burger Food: Burger Quantity: 1 Total: \$20.00	Completed
User2	Hotel Name: Royal Chicken Biryani 18567908 Food: Royal Chicken Biryani 18567908 Quantity: 1 Total: \$210.00	Pending

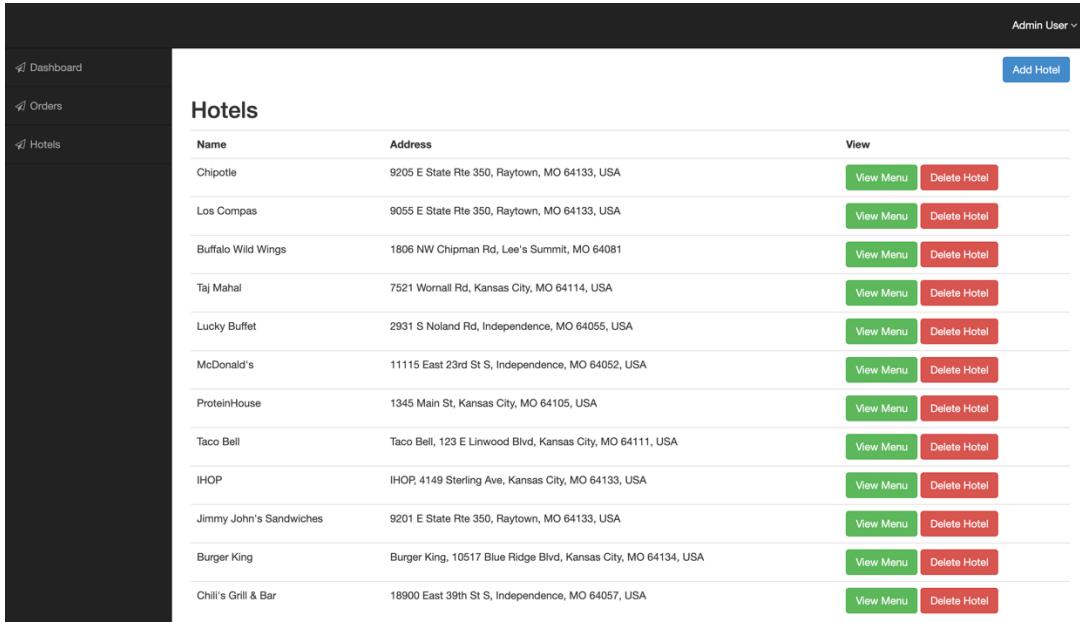
**Recent Orders**

Name	Menu	Order Status
User1	Hotel Name: Royal Chicken Biryani 18947325 Food: Royal Chicken Biryani 18947325 Quantity: 1 Total: \$210.00	Completed
User1	Hotel Name: Burger Food: Burger Quantity: 1 Total: \$20.00	Completed
User2	Hotel Name: Royal Chicken Biryani 18567908 Food: Royal Chicken Biryani 18567908 Quantity: 1 Total: \$210.00	Completed

Figure 8: Order Accept/Reject

**Description:** From this screen Admin can manage the orders from the various users. Managing order includes, accepting and rejecting the incoming orders.

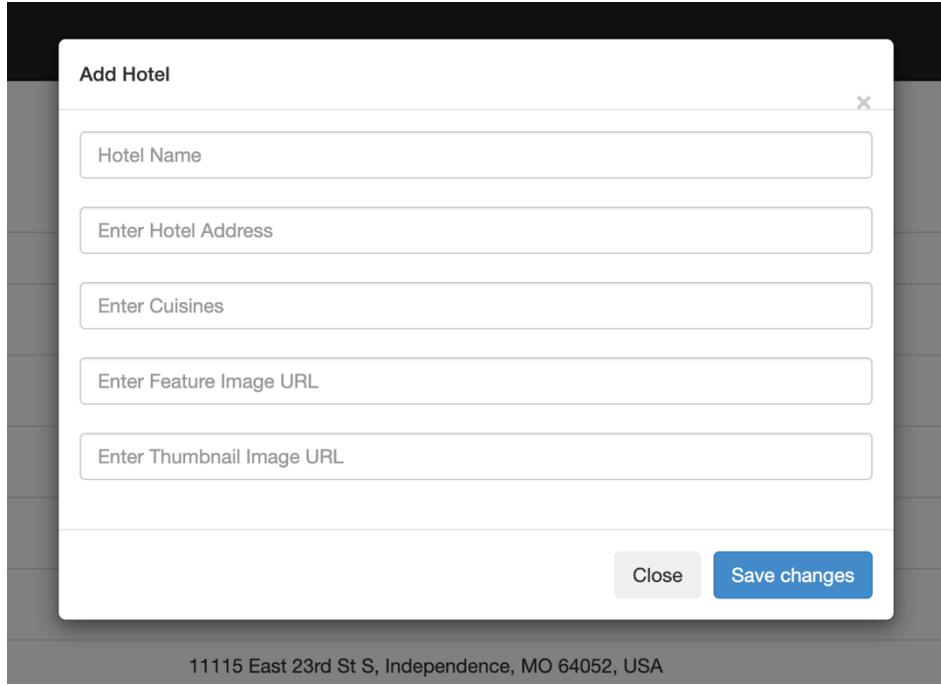
## 9. Hotels Managing Page



The screenshot shows a web application interface for managing hotels. On the left is a sidebar with navigation links: Dashboard, Orders, and Hotels. The main content area has a header "Hotels". Below it is a table listing various restaurants with their names and addresses. Each row includes a "View" button (green) and a "Delete Hotel" button (red). A blue "Add Hotel" button is located in the top right corner of the main content area.

Name	Address	View	Delete Hotel
Chipotle	9205 E State Rte 350, Raytown, MO 64133, USA	<button>View Menu</button>	<button>Delete Hotel</button>
Los Compas	9055 E State Rte 350, Raytown, MO 64133, USA	<button>View Menu</button>	<button>Delete Hotel</button>
Buffalo Wild Wings	1806 NW Chipman Rd, Lee's Summit, MO 64081	<button>View Menu</button>	<button>Delete Hotel</button>
Taj Mahal	7521 Wornall Rd, Kansas City, MO 64114, USA	<button>View Menu</button>	<button>Delete Hotel</button>
Lucky Buffet	2931 S Noland Rd, Independence, MO 64055, USA	<button>View Menu</button>	<button>Delete Hotel</button>
McDonald's	11115 East 23rd St S, Independence, MO 64052, USA	<button>View Menu</button>	<button>Delete Hotel</button>
ProteinHouse	1345 Main St, Kansas City, MO 64105, USA	<button>View Menu</button>	<button>Delete Hotel</button>
Taco Bell	Taco Bell, 123 E Linwood Blvd, Kansas City, MO 64111, USA	<button>View Menu</button>	<button>Delete Hotel</button>
IHOP	IHOP, 4149 Sterling Ave, Kansas City, MO 64133, USA	<button>View Menu</button>	<button>Delete Hotel</button>
Jimmy John's Sandwiches	9201 E State Rte 350, Raytown, MO 64133, USA	<button>View Menu</button>	<button>Delete Hotel</button>
Burger King	Burger King, 10517 Blue Ridge Blvd, Kansas City, MO 64134, USA	<button>View Menu</button>	<button>Delete Hotel</button>
Chili's Grill & Bar	18900 East 39th St S, Independence, MO 64057, USA	<button>View Menu</button>	<button>Delete Hotel</button>

Figure 9: List of Hotels



The screenshot shows a modal dialog titled "Add Hotel". It contains five input fields: "Hotel Name", "Enter Hotel Address", "Enter Cuisines", "Enter Feature Image URL", and "Enter Thumbnail Image URL". At the bottom right of the modal are two buttons: "Close" and "Save changes". The background of the page shows a partial view of a restaurant listing with one item visible: "11115 East 23rd St S, Independence, MO 64052, USA".

Figure 10: Add Hotel

Description: Admin can view and add the hotel by clicking add new button.

## 10. Menu Managing Page

Name: Chipotle  
Address: 9205 E State Rte 350, Raytown, MO 64133, USA  
Specialty: Mexican

Dish	Description	Price	Action
Burrito Bowl	your choice of freshly grilled meat or sofritas served in a delicious bowl with rice beans or fajita veggies and topped with guac salsa queso blanco sour cream or cheese	\$8.00	<a href="#">Edit</a> <a href="#">Delete</a>
Burrito	your choice of freshly grilled meat or sofritas wrapped in a warm flour tortilla with rice beans or fajita veggies and topped with guac salsa queso blanco sour cream or cheese	\$13.00	<a href="#">Edit</a> <a href="#">Delete</a>

Figure 11: Menu List

Add Menu

Name: Jimmy John's Sandwiches

Address: 9201 E State Rte 350, Raytown, MO 64133, USA

Specialty: Gourmet Sandwiches

Dish	Description	Price
SPICY EAST COAST ITALIAN	A sandwich made with a spicy chicken salad, provolone cheese, and topped with easy mayo and onion.	\$12.00
BEACH CLUB	Made with a choice of fresh-baked 8" or Giant 16" French bread, lettuce, tomato, and mayo.	\$20.00

Menu Name:

Enter Description:

Menu Price:

[Close](#) [Save changes](#)

Figure 12: Add menu

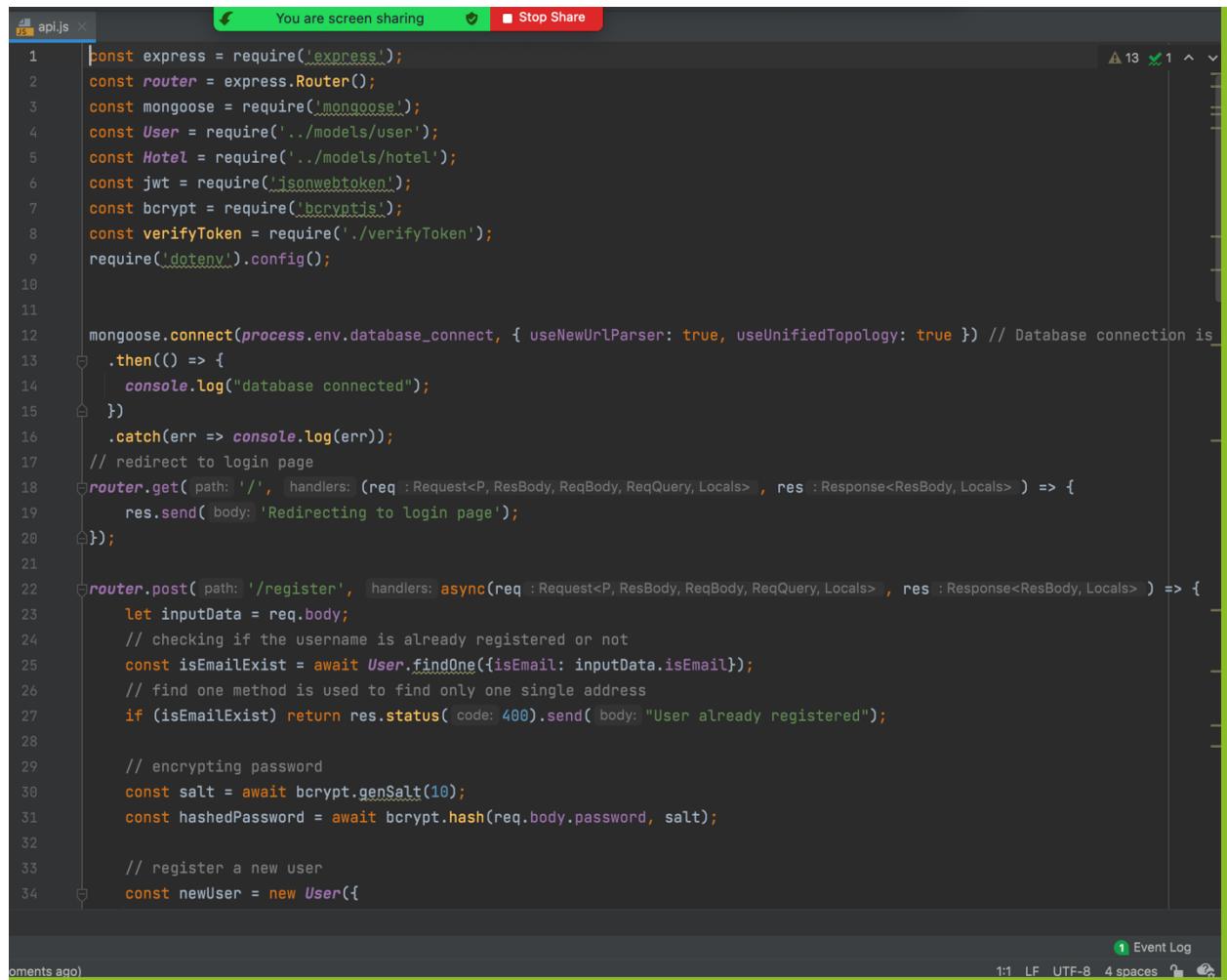
Description: Admin can view and add the menu for the specific hotel.

## Improvement:

In this increment, we mainly fixed some bugs in the admin panel and add new functionality that were not present in the previous sprint. We have discussed the improvement parts below.

1. The total of the order has been fixed in this sprint by using the correct sum formula.
2. In this sprint we have add the functionality, where user can add the restaurants using admin panel. In the previous sprint user must insert the new restaurant detail by visiting the database.
3. User can now delete the menu from the admin panel, whereas in the previous sprint user cannot delete the menu from the admin panel and if he wanted to delete it, he must need to visit the database to do that.
4. We have also added the chat bot into our website.
5. Refactoring of the whole code and we also added appropriate comments.

## Code Snippets:



```

1  const express = require('express');
2  const router = express.Router();
3  const mongoose = require('mongoose');
4  const User = require('../models/user');
5  const Hotel = require('../models/hotel');
6  const jwt = require('jsonwebtoken');
7  const bcrypt = require('bcryptjs');
8  const verifyToken = require('../verifyToken');
9  require('dotenv').config();

10
11
12  mongoose.connect(process.env.database_connect, { useNewUrlParser: true, useUnifiedTopology: true }) // Database connection is
13  .then(() => {
14      console.log("database connected");
15  })
16  .catch(err => console.log(err));
17 // redirect to login page
18 router.get( path: '/', handlers: (req :Request<P, ResBody, ReqBody, ReqQuery, Locals>, res :Response<ResBody, Locals>) => {
19     res.send( body: 'Redirecting to login page');
20 });
21
22 router.post( path: '/register', handlers: async(req :Request<P, ResBody, ReqBody, ReqQuery, Locals>, res :Response<ResBody, Locals>) => {
23     let inputData = req.body;
24     // checking if the username is already registered or not
25     const isEmailExist = await User.findOne({isEmail: inputData.isEmail});
26     // find one method is used to find only one single address
27     if (isEmailExist) return res.status( code: 400).send( body: "User already registered");
28
29     // encrypting password
30     const salt = await bcrypt.genSalt(10);
31     const hashedPassword = await bcrypt.hash(req.body.password, salt);
32
33     // register a new user
34     const newUser = new User({

```

Figure 13: Backend API Code

The screenshot shows a code editor window with two tabs: 'api.js' and 'hotel.js'. The 'hotel.js' tab is active, displaying the following code:

```

1  const mongoose = require('mongoose');
2
3  const Schema = mongoose.Schema;
4
5  const hotelSchema = new Schema({
6      id: String,
7      name: String,
8      address: String,
9      cuisines: String,
10     rating: String,
11     reviews: String,
12     feature_image: String,
13     thumbnail_image: String,
14     menu: [{{
15         id: Number,
16         name: String,
17         desc: String,
18         price: Number
19     }}]
20 });
21
22 module.exports = mongoose.model('hotel', hotelSchema, 'hotels');
23

```

The code defines a Mongoose schema for a 'hotel' document. It includes fields for id, name, address, cuisines, rating, reviews, feature\_image, thumbnail\_image, and menu. The menu is represented as an array of objects, each with id, name, desc, and price.

Figure 14: Hotel Schema

The screenshot shows a code editor window with a single file containing router logic. The relevant part is:

```

// getting a list of hotels from the database
router.get( path: '/hotels', verifyToken, async (req : Request<P, ResBody, ReqBody, ReqQuery, Locals>, res : Response<any, Record<string, any>>)
  try{
    const hotels = await Hotel.find();
    res.json(hotels);
  } catch(err) {
    res.status( code: 404).send( body: `request is not processed - ${err}`);
  }
};

// finding selected hotel using hotel ID
router.get( path: '/hotels/:hotelId', verifyToken, async (req : Request<P, ResBody, ReqBody, ReqQuery, Locals>, res : Response<any, Record<string, any>>
  try{
    const hotel = await Hotel.findOne({id: req.params.hotelId});
    res.json(hotel);
  } catch(err) {
    res.status( code: 404).send( body: `request is not processed - ${err}`);
  }
};

```

This code defines two routes: one for '/hotels' which returns a list of all hotels, and another for '/hotels/:hotelId' which returns a specific hotel by its ID. Both routes use the 'verifyToken' middleware and handle errors by sending a 404 status with an error message.

Figure 15: Getting Hotels List

```

.userLoginMethod = async() => {
    // async method is being called for the login
    await Swal.fire({
        // alert pop-up is used as a login html file using sweet alerts
        // defining and declaring all the parameters required to create a user understandable html
        imageUrl: 'assets/logo1.png',
        imageWidth: 250,
        imageHeight: 150,
        imageAlt: 'LOGO',
        title: '<strong>Login into your account</strong>',
        html:
            '<input class="swal2-input" placeholder="Enter your Email" required> id="username" ' +
            '<input class="swal2-input" placeholder="Enter your password" required> id="securityKey" type="password" ' +
            '<b>New to U-pep?</b>&nbsp' +
            '<a href="/register">Create account</a> ',
        allowEscapeKey: false,
        focusConfirm: false,
        confirmButtonText: 'LogIn',
        confirmButtonColor: '#437e4d',
        allowOutsideClick: false,
        preConfirm: () => { // preconfirm method is used in this method we are going to extracting the user given inputs
            this.userLogin = {
                gmail: (document.getElementById('username') as HTMLInputElement).value, // value of the user given email
                securityKey: (document.getElementById('securityKey') as HTMLInputElement).value // value of the user given password
            }
        }
    }).then((final) => { // after extracting the date checking is done basically validation is going on
        if (final.isConfirmed) { // if statement is used and pre defined isConfirmed method
            if (!this.userLogin.gmail || !this.userLogin.securityKey) { // checking whether the input fields are empty or not
                const userFault = this.errorCustomization( statusText: "Information Missing", statusMessage: "no fields should be empty" );
                this.errorDisplay(userFault); // setting the customized error message to the errorDisplay method
            }
        } else { // if the details are present the login method in the service layer is called
            this.serviceAuth.userLogin(this.userLogin).subscribe( // connecting to the backend in the service layer is done and
        }
    })
}

```

Figure 16: Login Page

## Work Distribution:

- Muhammad Usman & Ritwik Reddy taking care of the Backend framework and managing overall project.
- Aravind Reddy & Damodar Reddy is working on the front-end design of the website.
- All team are working on collecting the data and completing the documentation.

## **Issue & Blockages with the Project:**

1. Due to lack of experience of our team members and with short time period, it was difficult to include all the features in our applications. However, we included all the major features that is required for online food ordering app.
2. Second challenge was to develop the admin panel. We first tried the auto generated Adminbro admin panel that is design for node applications, and we successfully deployed that admin panel. However, there was a problem with iterating the nested array from the database and after discussing with professor Vijaya, we created the admin panel in PHP which was easy, and it was easily connected with the mongodb database.
3. In the beginning we are planning to add online grocery stores in our application, but the business logic of online grocery store was not matching with the online food ordering application. If we add grocery stores in our application, then it would become difficult for grocery store owners to maintain their stocks in the database system. Due to this issue, we excluded this feature from our application.

## **GitHub Link:**

[https://github.com/MuhammadUsman94/Web-Mobile Project](https://github.com/MuhammadUsman94/Web-Mobile_Project)

## **Video Link:**

<https://youtu.be/RzBavUCuIIA>

## **Presentation Link:**

[https://github.com/MuhammadUsman94/Web-Mobile\\_Project/blob/main/Power%20Point/Web%20and%20Mobile%20PT.pptx](https://github.com/MuhammadUsman94/Web-Mobile_Project/blob/main/Power%20Point/Web%20and%20Mobile%20PT.pptx)

## **References:**

- <https://www.spaceotechnologies.com/create-online-food-ordering-system-guide/>
- <https://www.youtube.com/watch?v=WEqOr-clmMM>

- <https://www.youtube.com/watch?v=dMQFT270axQ>
- <https://www.youtube.com/watch?v=UrfhqE7I-3o>