Università
degli Studi di
Messina

Web Programming project

Report

Course: Web-programming

Professor: Armando Ruggeri

Student: Tairbek Akhayev, 551094

Table of Contents

1. Project Overview

1.1. Objective

The objective of this project is to create a web platform, which allows users to register and log in using SQL database interaction. The backend development of the platform must be written in any of the programming languages, among them PHP, Node.js, Python, etc., to expose APIs. Frontend development to navigate the platform, written in either HTML, CSS, or JavaScript with jQuery/Angular. For my project, I used various backend-oriented and frontend-oriented programming languages to implement various functions and APIs, which will be discussed later in this report.

1.2. Scope

For the class on Web Programming, I chose to implement a Food Delivery website. My goal was to create a platform where users can easily browse food options and manage their accounts. The platform features two distinct user interfaces: one for customers and another for administrators.

The **customer interface** allows users to browse the menu, customize orders and manage their profile details.

The **administrator interface** is designed for managing the website's operations. Admins can manage user accounts for all users.

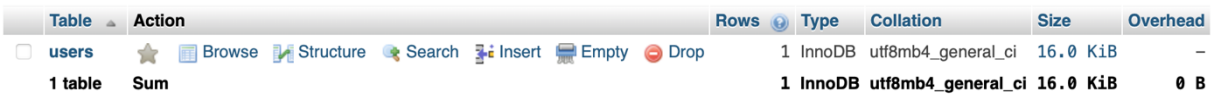
2. Technology Stack

The development of the Food Delivery website involved the use of the following technologies:

- **Frontend:** HTML, CSS, JavaScript
- **Backend:** PHP
- **Database:** MySQL
- **Styling:** CSS

3. Database Structure

The database consists of 1 table: users.



This table includes the following attributes: id of the user, name, and surname of the user, email address, password, and the is_admin. The password is hashed once a new user is registered for security measures, and as we know the hashed passwords cannot be “unhashed”, so this sensitive information is kept private and secure. The is_admin attribute describes the role of the user, precisely whether the user is an admin account or a guest account. Depending on this description, if the is_admin attribute is set to 1, the account is the admin account, and hence different UI will be available for the administrator, with additional features to the user features, which will be discussed later. On the other hand, when the is_admin attribute is set to 0, the user is the guest account.

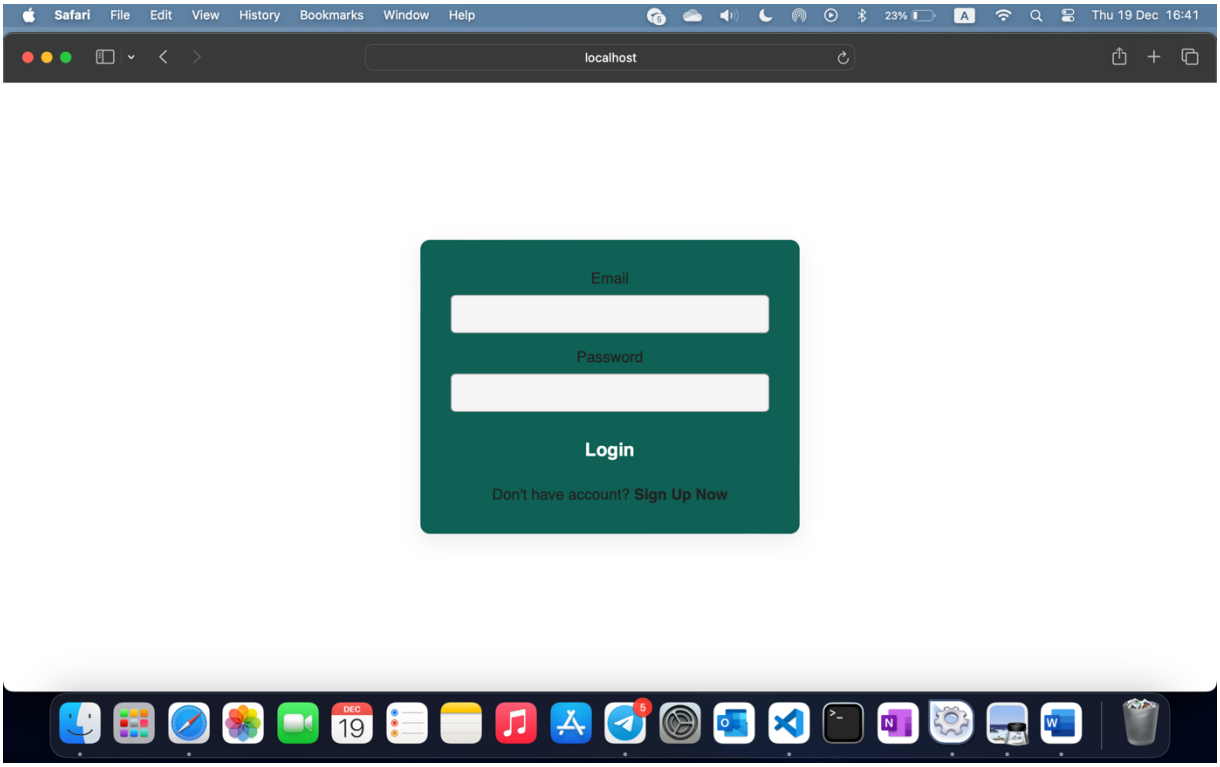
#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**id**	int(11)			No	*None*		AUTO_INCREMENT	Change Drop More
2	**name**	varchar(100)	utf8mb4_general_ci		No	*None*			Change Drop More
3	**surname**	varchar(100)	utf8mb4_general_ci		No	*None*			Change Drop More
4	**email**	varchar(100)	utf8mb4_general_ci		No	*None*			Change Drop More
5	**password**	varchar(255)	utf8mb4_general_ci		No	*None*			Change Drop More
6	**is_admin**	tinyint(1)			No	*None*			Change Drop More

	id	name	surname	email	password	is_admin
Edit Copy Delete	35	Tair	Admin	admin@gmail.com	$2y$10$IJDqkXUMJS.rEdBfwRz1TOWoO9yW1XlnzmTvhvO72tS...	1
Edit Copy Delete	38	Tair	Guest	guest@gmail.com	$2y$10$PLBrF9mvTArVivYvcHjGie2FQweC10dmHjTSuHKMaB7...	0

4. Web Pages

Authorization page

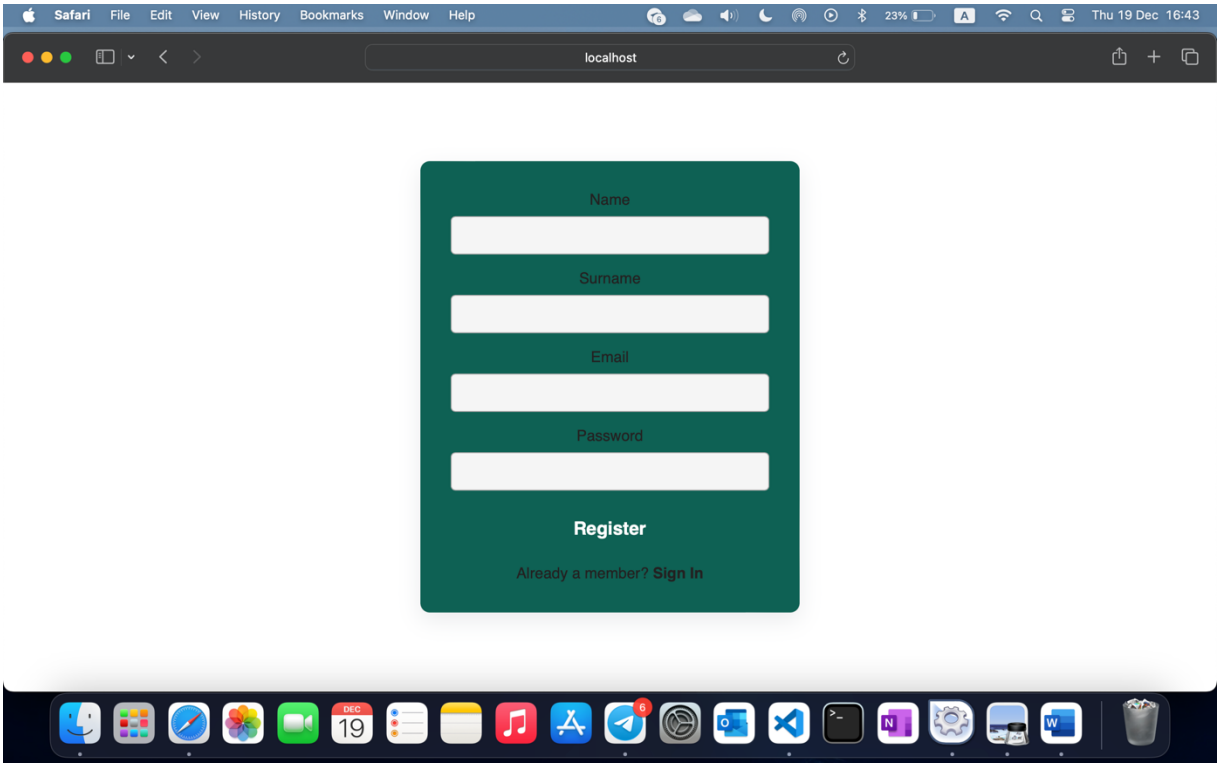
It is the index page, where the user has a choice to either authorize/login or register as a new user. The login page asks for the email of the user, which is the username in our case, and the password. Below the login page, there is a “Sign Up Now” link, by pressing which a user will be transferred to a sign-up page. This dual functionality ensures a seamless onboarding experience, accommodating both new and returning users. This is a clear and simple interface, with no distracting content, to enrich the user experience and guide them through the website.



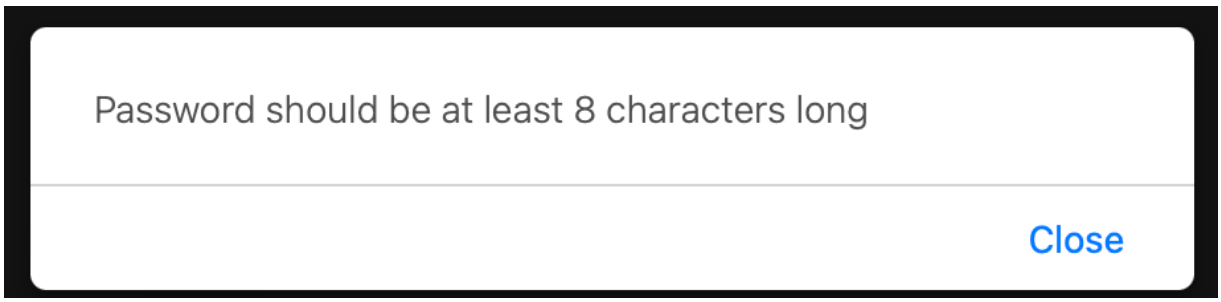
When login is successful the following message pops up, by pressing on “close” button the user will be redirected to the home page of the website.

In the case of the scenario where the user is not yet authorized, the users are directed to a different page, where they are prompted to register. The interface asks for a name, surname, email address, and password. All fields should be filled with information, and if not, the system will highlight the empty fields with red, asking the users to fill the fields. After all the fields are filled, these data are then transferred to the SQL database, where data is inserted into the user table, using the SQL query written in the .php file.

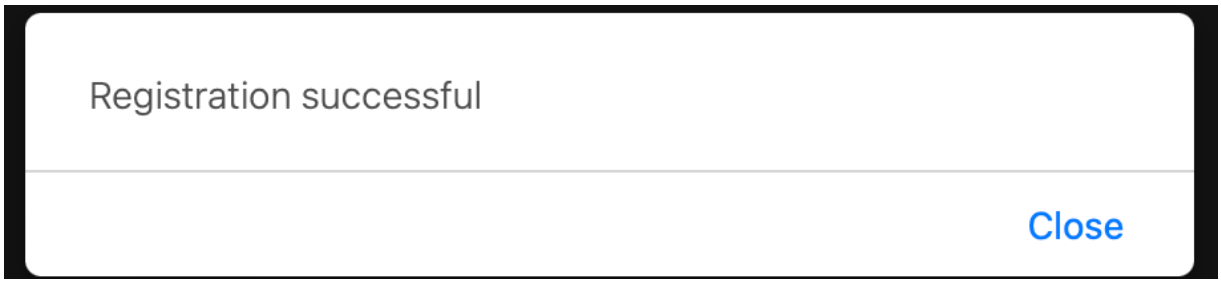
There is also a link to return to the previous page in case the user mistakenly presses the “Sign Up Now” button or remembers that they already have an account. The “Sign In” link redirects the user to the previous page with login.



For the security measures, the password should be at least 8 characters long, and the code checks for that. If the password is less than 8 characters long, the following message will pop out on the page, clearing all fields, and prompting the user to fill in the fields again, but with the proper length of the password.

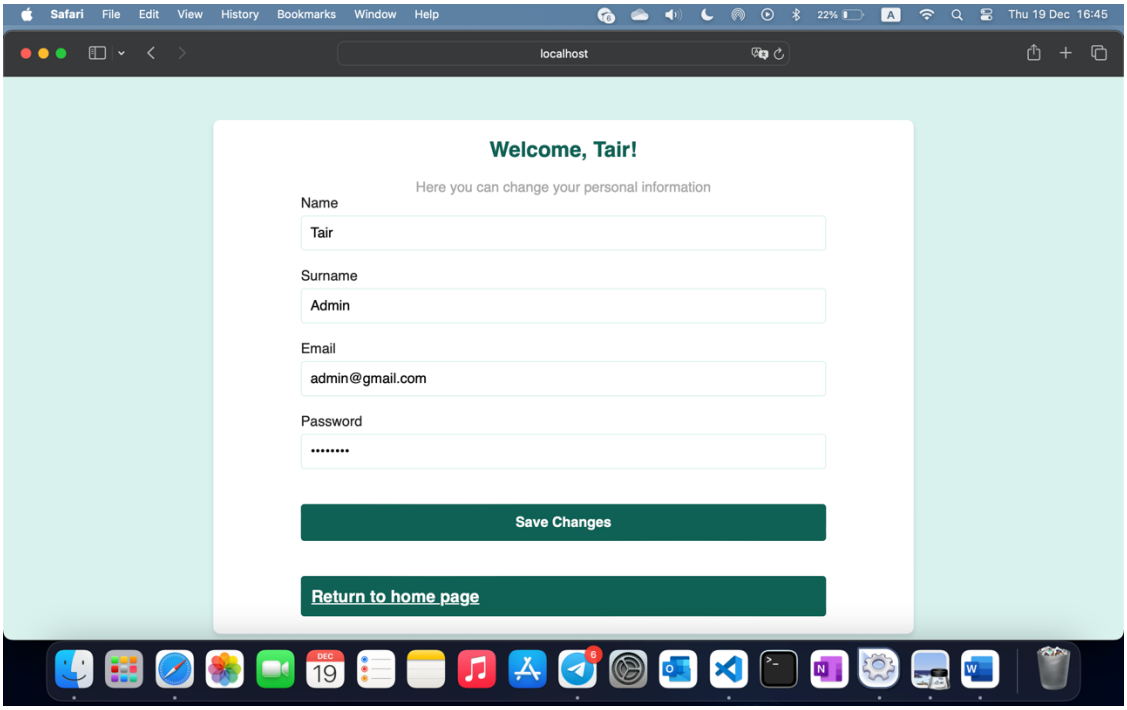


When the user has created a strong password that complies with the rules, the “registration successful” message appears on the page. By clicking “close” the user is redirected to the login page, where he/she can enter the login information to access the system.



- Account page

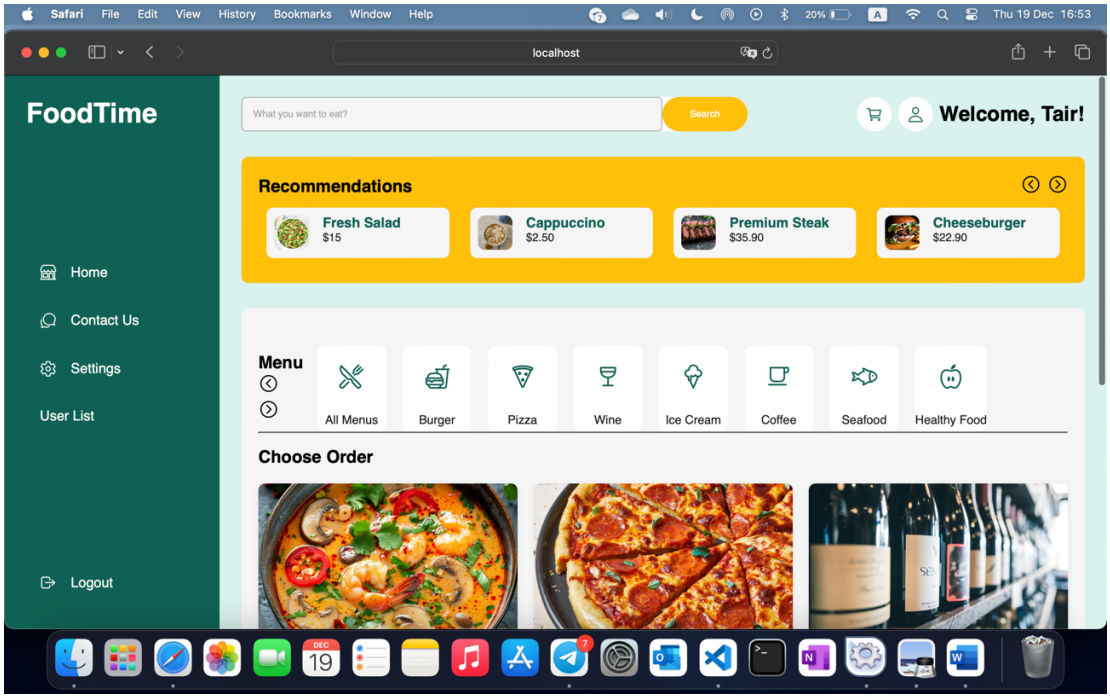
In the Account page, user empowerment is a key focus. Users are granted the autonomy to modify and update their personal data. Through a systematic approach, the current user's information is retrieved from the database, and these details fill corresponding input fields. This ensures that users have a transparent view of the data they are modifying. With this usercentric design, individuals can seamlessly interact with their account information, facilitating a straightforward process for updating and saving changes. The Account component embodies our commitment to providing users with control over their personal data in a secure and accessible manner.



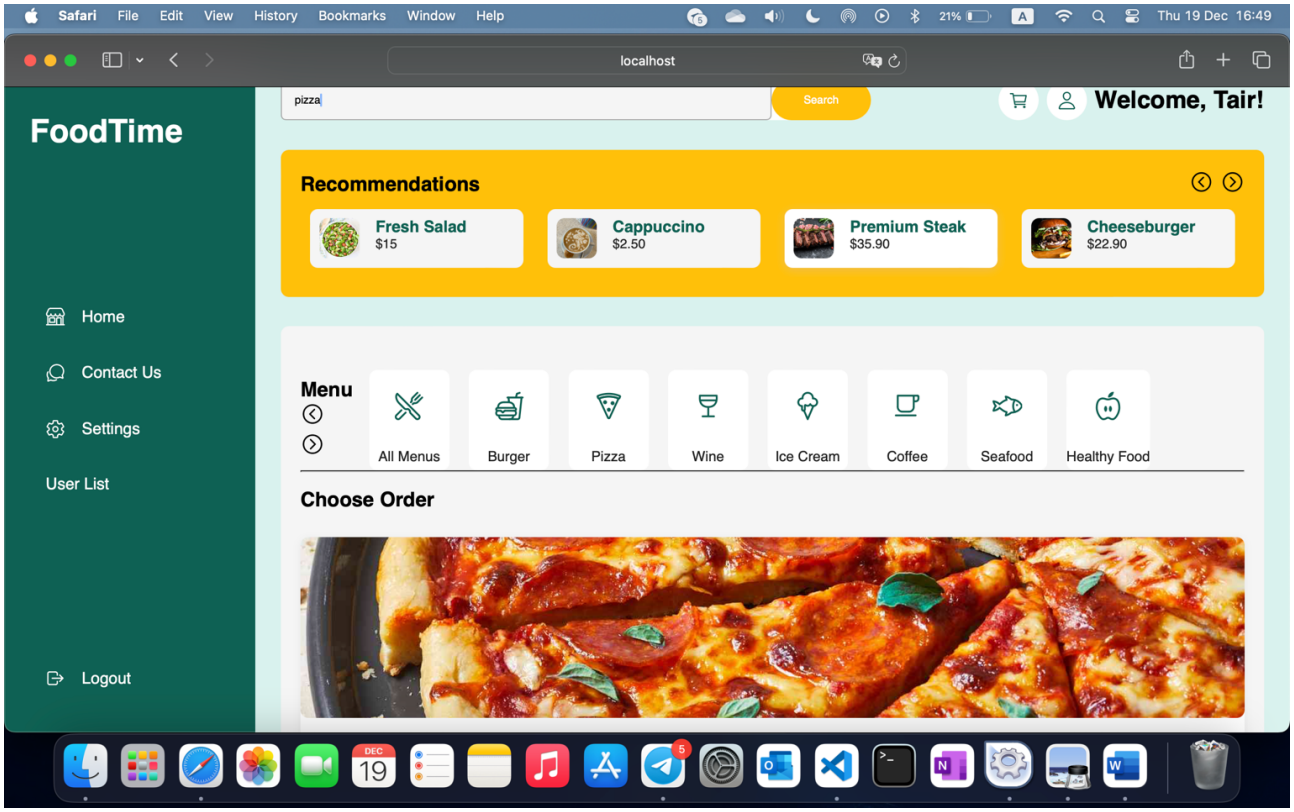
The account page provides an option for the user to change their personal information. All fields should be filled so that the SQL query UPDATES these fields in the table. After done, the user must press the “Save Changes” button, prompting the code to call for the SQL query and modify the changes.

- Home page

The Home page of the Food Delivery website serves as a dynamically personalized interface. By leveraging user authentication, it welcomes users by name, creating a more engaging and user-centric experience. The design focuses on functionality and aesthetics to ensure seamless navigation and user satisfaction.



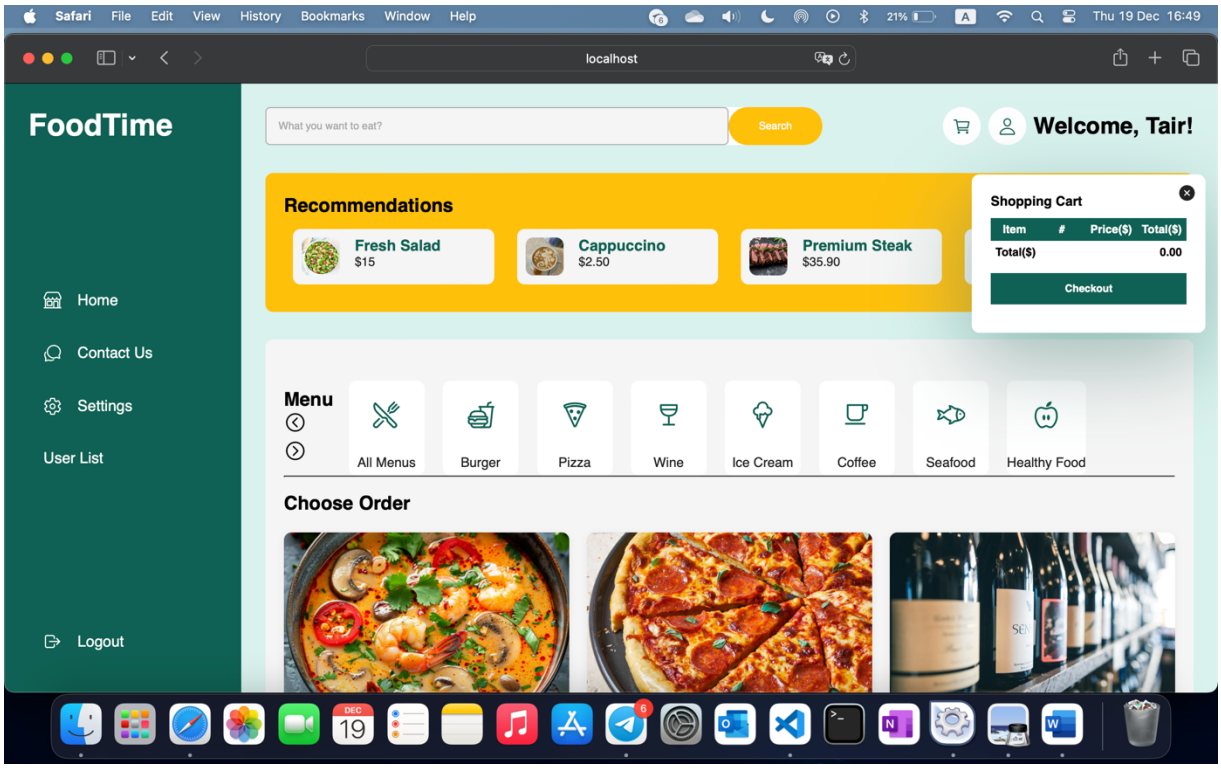
The Home page includes a navigation sidebar with quick links to essential sections like "Contact Us" and "Settings," and, for administrators, an additional "User List" section for managing user accounts if you’re admin. A prominent **search bar** allows customers to explore available food options easily.



The main content area is organized into two sections:

1. **Highlighted Recommendations**: This section displays curated meal options as visually appealing cards with images, descriptions, and prices. Customers can navigate through recommendations using navigation arrows.
2. **Menu and Ordering Section**: Here, customers can browse categorized menu items (e.g., burgers, pizzas, healthy food) or explore all available dishes. Each item card includes a description, price, and an "Add to Cart" button, streamlining the order placement process.

Additionally, a **shopping cart popup** is available for users to review and modify their orders before proceeding to checkout, enhancing usability and user control.

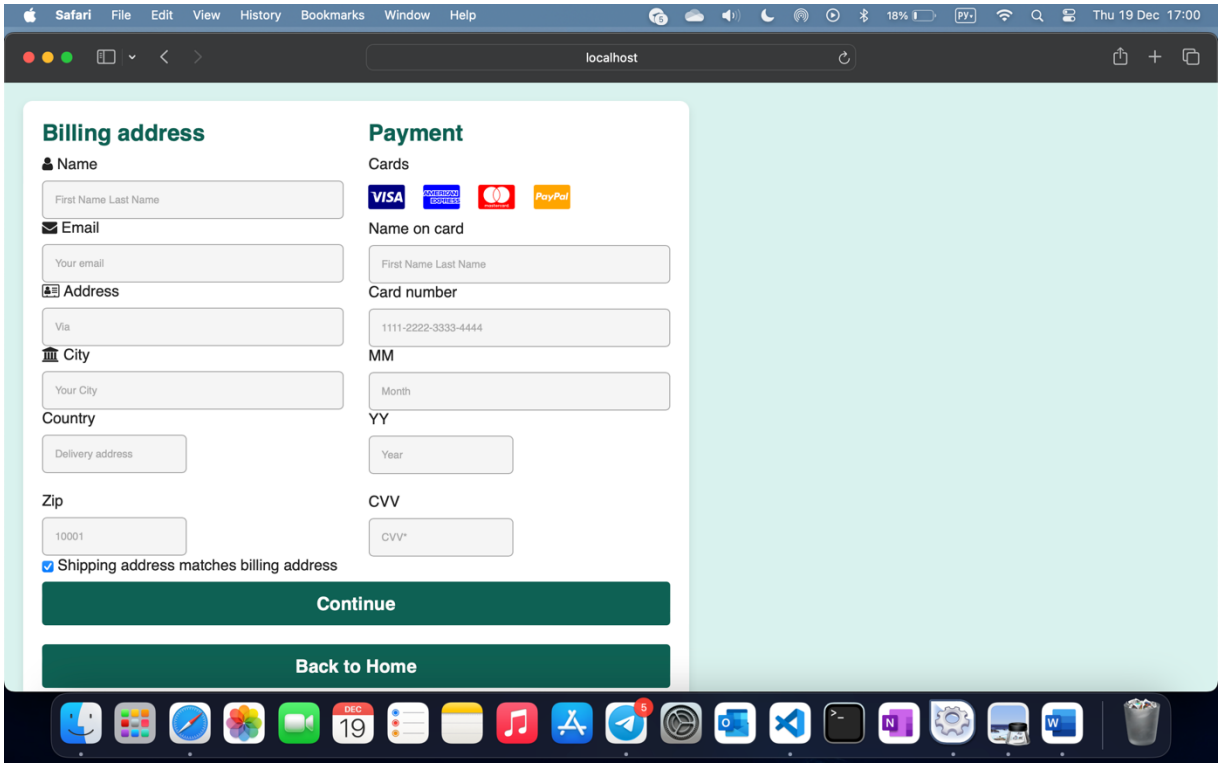


The thoughtful layout and personalized elements make the Home page an intuitive and welcoming entry point for the platform, ensuring an optimized user experience tailored to a modern food delivery service.

- Checkout page

The **Checkout page** is an essential part of the food delivery process, providing a seamless experience for users to review and complete their orders. Items are added to the shopping cart from various sections of the website, such as the **menu pages**, by clicking the "Add to Cart" button. Each time an item is added, it is stored in the cart along with its name, quantity, and price.

On the **Cart page**, users can view all added items. They have the flexibility to adjust the quantity of each item by using plus (+) and minus (-) buttons next to the quantity field. This allows users to modify their order to better suit their needs before proceeding. As the quantity is updated, the total cost of the order is recalculated in real-time, providing immediate feedback.



The **Checkout page** is where users can finalize their food delivery orders. This page is structured to gather the necessary information for completing the purchase, including billing and shipping details, as well as payment information.

Billing Address Section:

The **Billing address** form collects personal information such as the user's name, email, address, city, country, and postal code. This section ensures that the user's billing details are entered correctly before moving forward with the payment.

Payment Section:

In the **Payment** section, users are prompted to enter their payment details, including the cardholder's name, card number, expiration date, and CVV. Icons for different types of credit and debit cards (Visa, MasterCard, Amex, and PayPal) are provided for better user recognition.

Validation:

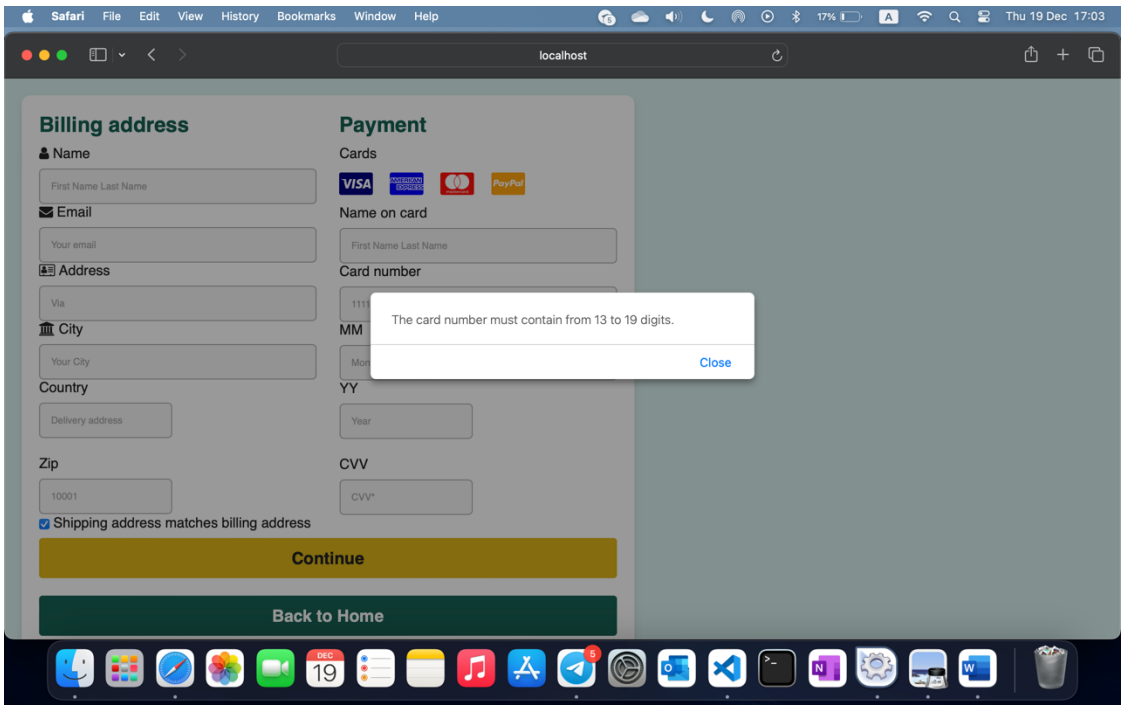
A key feature of the page is the **form validation** script. Before the user can submit the form, the system checks that the card number follows the correct format, the expiration date is valid (i.e., not expired), and the CVV is exactly three digits long. If any of these fields are incorrect, an alert is shown, and the form submission is prevented, ensuring that invalid information is not processed.

```
function validateForm(event) {
    const cardNumber = document.getElementById("ccnum").value;
    const expirationMonth = document.getElementById("expmonth").value;
    const expirationYear = document.getElementById("expyear").value;
    const cvv = document.getElementById("cvv").value;

    // Валидация номера карты
    if (!validateCardNumber(cardNumber)) {
        alert("The card number must contain from 13 to 19 digits.");
        event.preventDefault(); // Останавливает отправку формы
        return;
    }

    // Валидация даты истечения
    if (!validateExpirationDate(expirationMonth, expirationYear)) {
        alert("The card expiration date is incorrect.");
        event.preventDefault();
        return;
    }

    // Валидация CVV
    if (cvv.length !== 3) {
        alert("CVV must be 3 digits long.");
        event.preventDefault();
        return;
    }
}
```

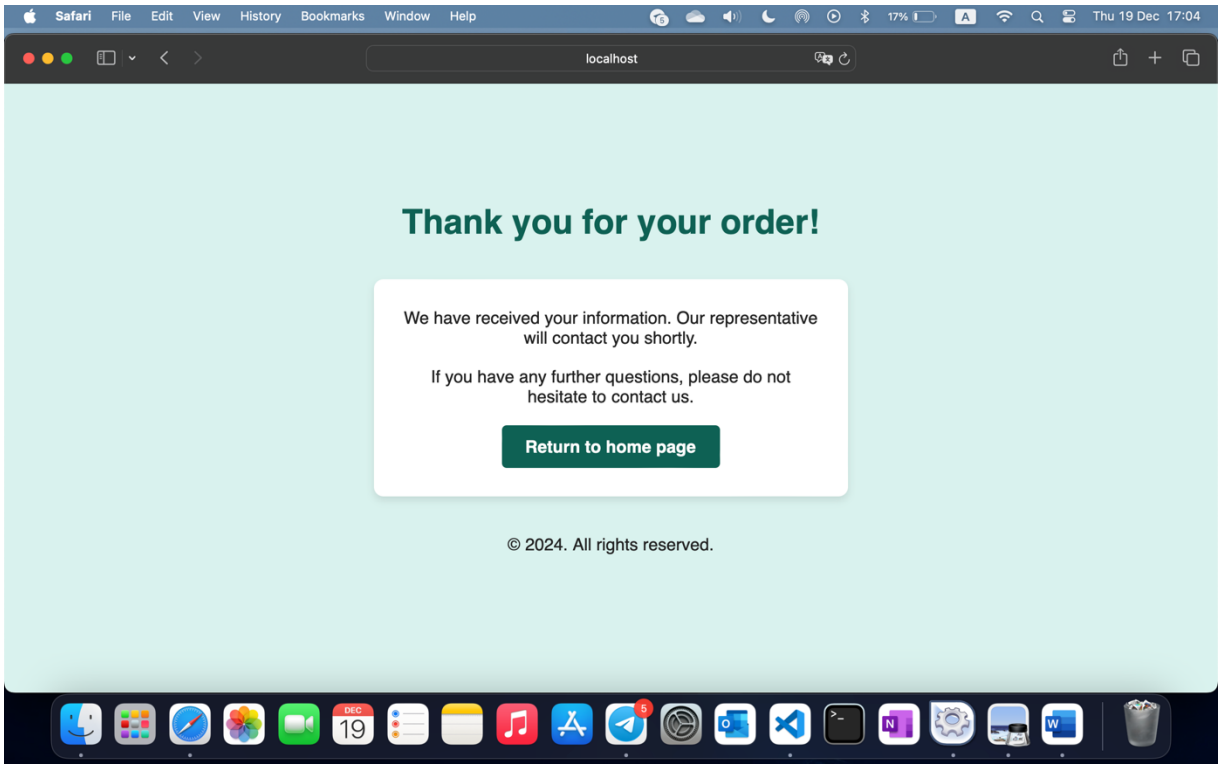


Shipping Address:

Users have the option to select whether the **shipping address** matches the billing address. If checked, the system assumes that both addresses are the same, making the process more streamlined.

Checkout Process:

Once all the information is filled out and validated, users can click the **Continue** button to proceed to a "Thank You" page, confirming their order and providing a final summary. Alternatively, if they wish to go back to the home page, a button is available to navigate them to the main menu.

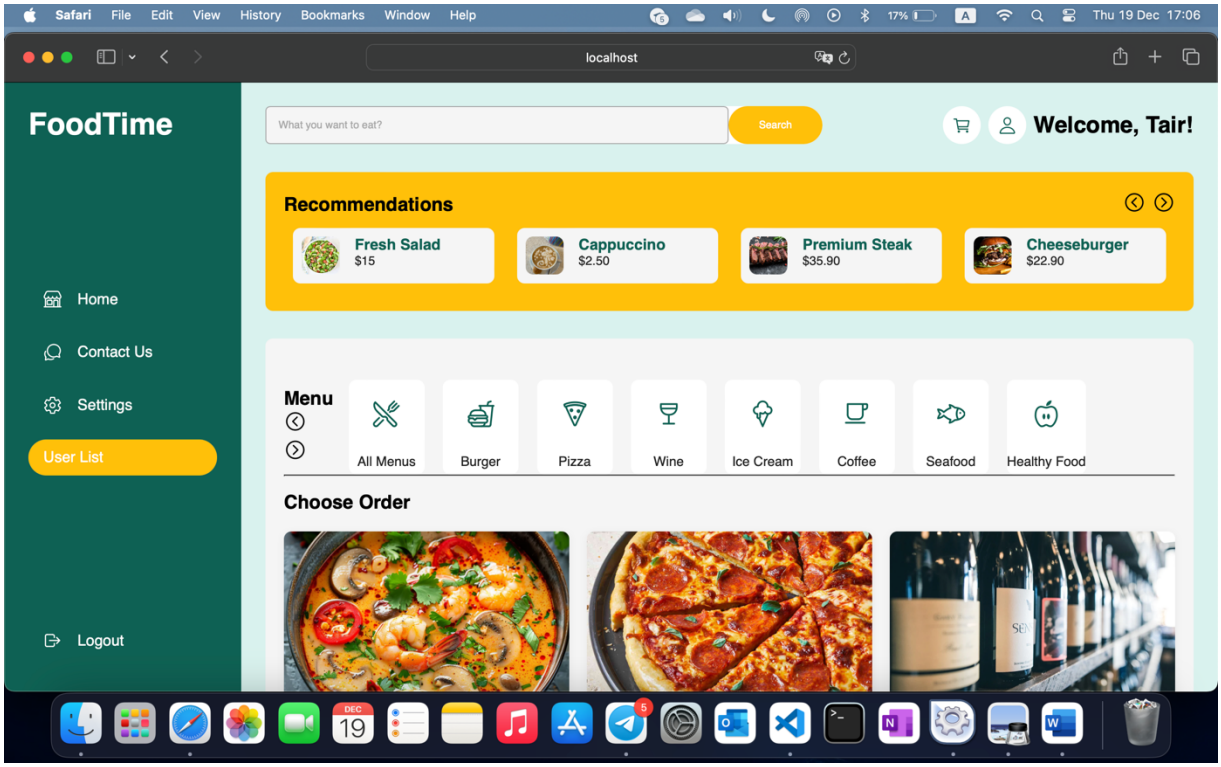


Navigation and User-Friendly Design:

The page is designed to ensure ease of navigation with clear labels and input fields. The responsive layout adapts to various devices, making the checkout process seamless on both desktop and mobile platforms.

- Users' list page

The **userList.php** page is designed for administrators to view and manage the list of users in the system. The page allows the admin to see all registered users, display their roles, and take actions such as deleting users or updating their roles.

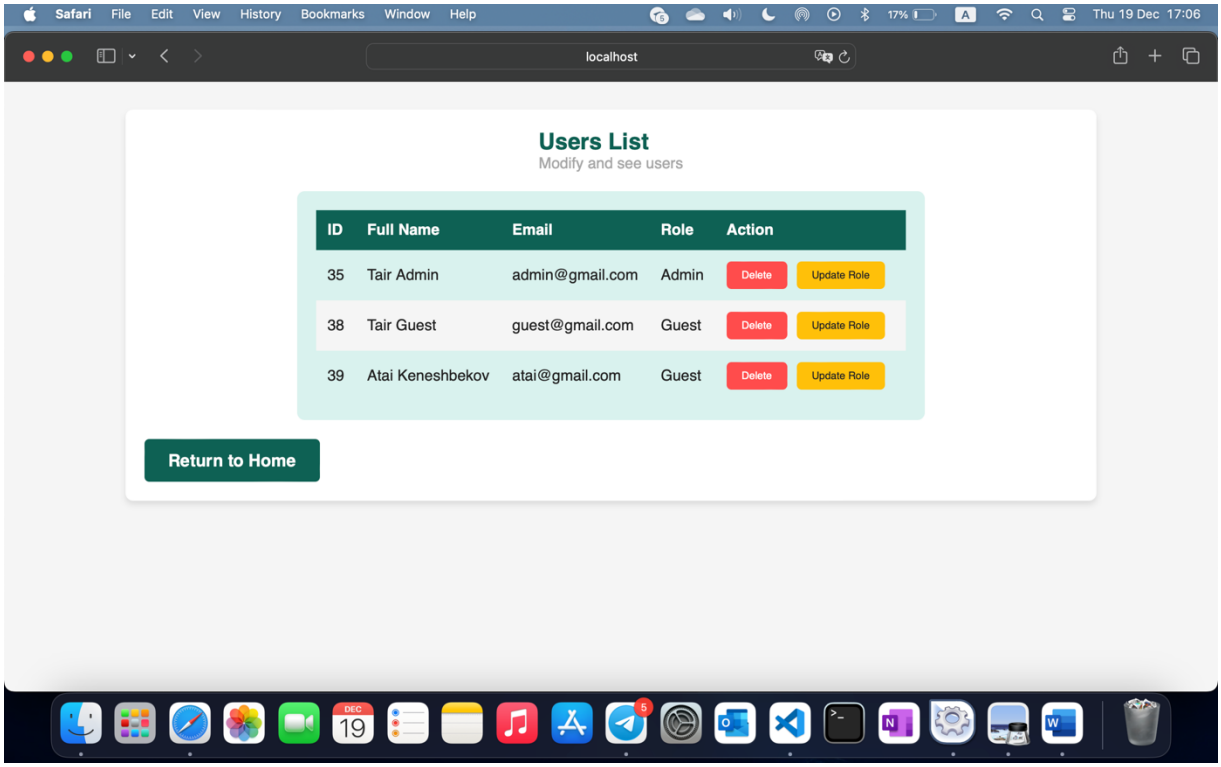


Key Features:

1. **Session Management**: The page starts by verifying that the current user is logged in and has admin privileges. The `session.php` file is included to check the session data, ensuring that only users with admin rights (`is_admin == 1`) can access this page. If the user is not an admin, they are redirected or prevented from viewing the content.
2. **Database Connection**: The page uses a MySQL connection (`conn.php`) to fetch user data from the database. Initially, it retrieves the logged-in user's details (like their name) and then fetches all the users in the system to populate the user list.
3. **Displaying Users**: The page dynamically generates an HTML table displaying the following information for each user:
    - **ID**: The unique identifier of the user.
    - **Full Name**: A combination of the user's first and last name.
    - **Email**: The user's email address.
    - **Role**: Indicates whether the user is an **Admin** or a **Guest**.
    - **Action**: Provides two buttons for each user:
        - **Delete**: Allows the admin to remove a user from the system.
        - **Update Role**: Enables the admin to modify the user's role (e.g., change a Guest to an Admin).
4. **Table Structure**: The user list is displayed in a table with the following columns:
    - `ID`: A unique identifier for each user.
    - `Full Name`: The concatenation of the user's first and last names.
    - `Email`: The user's email address.
    - `Role`: Displays either "Admin" or "Guest" based on the user's `is_admin` field in the database.
    - `Action`: Contains two buttons to perform actions such as deleting the user or updating their role.
5. **Action Buttons**: The action buttons, such as **Delete** and **Update Role**, are linked to JavaScript functions (presumably defined in `logic.js`):

- `deleteUser(id)`: Deletes the user with the specified `id` from the database.
- `updateUserRole(id)`: Opens a dialog or provides an interface for changing the user's role (e.g., toggling between Admin and Guest).

6. **Return to Home Button**: Below the table, there is a **Return to Home** button that redirects the admin back to the home page (`index.php`).

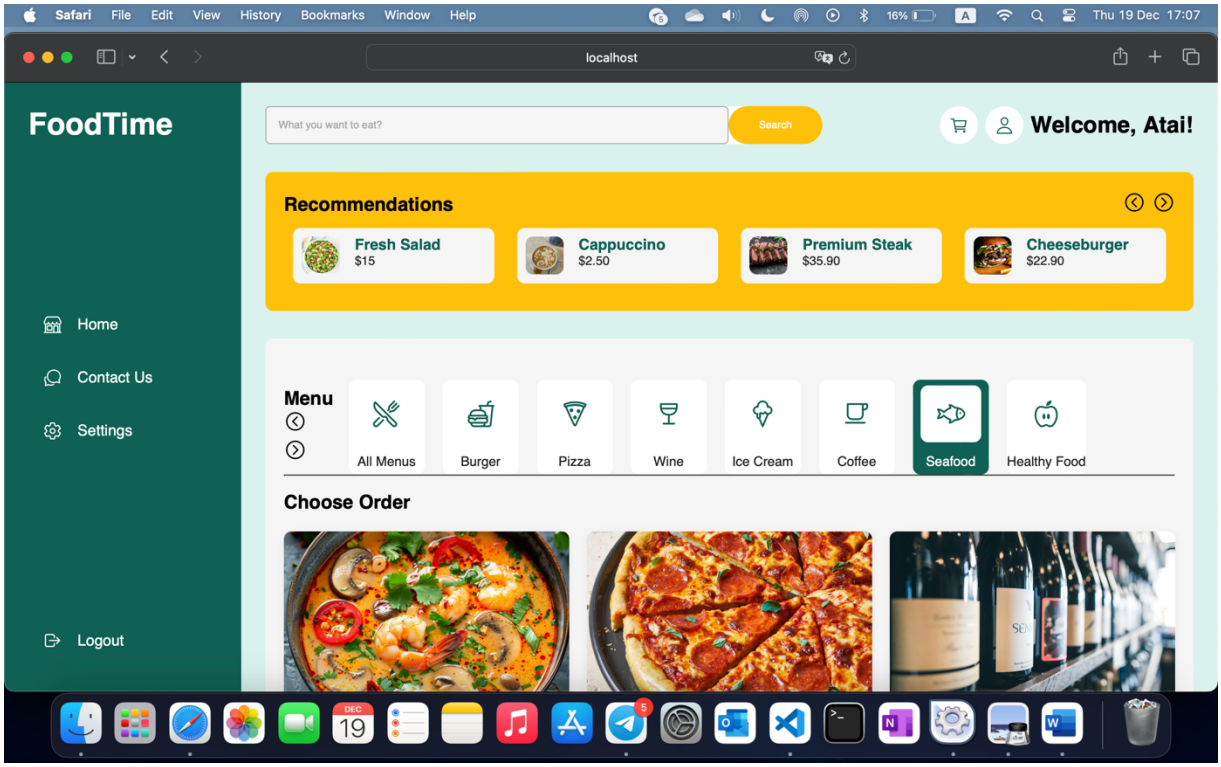


JavaScript Functions (`logic.js`):

The script file `logic.js` is included at the end of the page, which likely contains the implementation for handling the user actions:

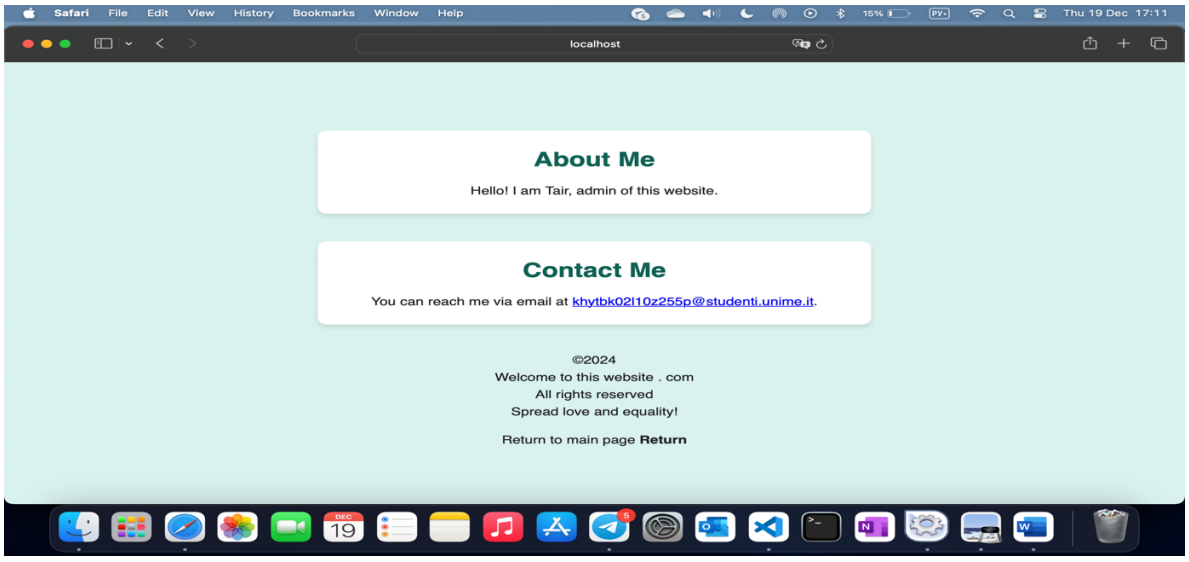
- **deleteUser(id)**: Sends a request to the server to delete the user with the given `id`. It might involve an AJAX request that triggers a server-side script to perform the deletion without reloading the page.
- **updateUserRole(id)**: Initiates an action to update the role of the user. This could open a prompt or a modal for the admin to select a new role for the user.

If you’re not admin:



Contact-us page

The **Contact Us** page offers a straightforward and welcoming way for users to reach out. It begins with a brief introduction in the **About Me** section, where the admin introduces themselves, ensuring a personal touch In the **Contact Me** section, users are provided with a direct email link, making communication easy and efficient. The page is designed to offer clear and open communication channels, reinforcing the admin's availability and commitment to user engagement. Additionally, a footer message promotes positivity and equality, enhancing the overall user experience. Finally, a link back to the main page ensures seamless navigation throughout the site.



5. Key Features

5.1. Database connection

```php
<?php
    $conn = mysqli_connect(hostname: 'localhost', username: 'root', password: '', database: 'food_delivery');

    if (!$conn) {
        die("Connection failed: " . mysqli_connect_error());
    }
?>
```

This .php file is served to connect to the database which is called “food_delivery”. This is a standard code used for connecting to a specific database on localhost. Later, all the other .php files will be using this file to check for the connection to the database, instead of writing this chunk of code in each file over and over again. This not only saves space but also offers an efficient coding and project design.

5.2. Login

The HTML file provided is the **Login** page of the website. It follows a standard structure, starting with the doctype declaration and includes essential metadata in the head section. The `meta` tags define the character set, set the viewport for responsiveness, and ensure compatibility with Internet Explorer. The page is linked to an external stylesheet, **style.css**, to manage the visual presentation.

In the body of the document, PHP code is embedded to handle user login functionality. The `include_once` statements bring in session management and database connection logic from external PHP files. The page checks if a user is already logged in, and if so, redirects them to the homepage to prevent redundant login attempts.

The login process is implemented through a form that collects the user’s email and password. PHP validates the email format and checks that all fields are filled. It queries the database to verify if the entered email exists and uses `password_verify` to check if the provided password matches the hashed password stored in the database. If the credentials are incorrect, the user is alerted with an error message. If the login is successful, the user is redirected to the homepage with a success message.

The form is designed using standard HTML and uses the POST method to submit user input, which is then processed by the embedded PHP code. A link to the **Registration** page is also provided for users who don’t have an account, encouraging new users to sign up.

The visual style of the login page is controlled by an external CSS file referenced in the head section, **style.css**, allowing for easier management and better organization of the code. Overall, the page combines HTML and PHP to provide a secure and functional authentication system for users.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
     <link rel="stylesheet" href="styles/style.css">
    <title>Login</title>
</head>
<body>
      <div class="container">
        <div class="box form-box">
            <?php
                include_once "utils/session.php";
                include_once "php/conn.php";
                if(isset($_POST['submit'])){
                    $email = $_POST['email'];
                    $password = $_POST['password'];
                    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
                        echo "<script>alert('Invalid email format');</script>";
                    } else if (empty($password)) {
                        echo "<script>alert('Please fill in all
fields');</script>";
                    } else {
                        $sql = "SELECT * FROM users WHERE email = '$email'";
                        $result = mysqli_query($conn, $sql);
                        $num_rows = mysqli_num_rows($result);
                        if ($num_rows > 0) {
                            $row = mysqli_fetch_assoc($result);
                            if (password_verify($password, $row['password'])) {
                                session_start();
                                $_SESSION['id'] = $row['id'];
                                $_SESSION['name'] = $row['name'];
                                $_SESSION['surname'] = $row['surname'];
                                $_SESSION['email'] = $row['email'];
                                $_SESSION['is_admin'] = $row['is_admin'];
                                setcookie("id", $row['id'], time() + (86400 * 30),
"/");
                                echo "<script>alert('Login successful');
window.location.href = 'index.php';</script>";
                            } else {
                                echo "<script>alert('Incorrect
password');</script>";
                            }
                        } else {
                            echo "<script>alert('Email does not exist');</script>";
                        }
                    }
                }
            ?>
            <form action="" method="post">
```

```
                <div class="field input">
                    <label for="email">Email</label>
                    <input type="text" name="email" id="email" autocomplete="off"
required>
                </div>

                <div class="field input">
                    <label for="password">Password</label>
                    <input type="password" name="password" id="password"
autocomplete="off" required>
                </div>

                <div class="field">
                    <input type="submit" class="btn" name="submit" value="Login"
required>
                </div>
                <div class="links">
                    Don't have account? <a href="registration.php">Sign Up Now</a>
                </div>
            </form>
        </div>
      </div>
</body>
</html>
```

5.2. User Registration

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="styles/style.css">
    <title>Registration</title>
</head>
<body>
      <div class="container">
        <div class="box form-box">
            <?php include_once "php/conn.php";
                include_once "utils/session.php";

                if(isset($_POST['submit'])){
                    $name = $_POST['name'];
                    $surname = $_POST['surname'];
                    $email = $_POST['email'];
                    $password = $_POST['password'];
                    $is_admin = 0;

                    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
```

```
                        echo "<script>alert('Invalid email format');</script>";
                    } else if (empty($name) || empty($surname) || empty($password))
{
                        echo "<script>alert('Please fill in all
fields');</script>";
                    } else if (strlen($password) < 8) {
                        echo "<script>alert('Password should be at least 8
characters long');</script>";
                    } else {
                        $sql = "SELECT * FROM users WHERE email = '$email'";
                        $result = mysqli_query($conn, $sql);
                        $num_rows = mysqli_num_rows($result);
                        if ($num_rows > 0) {
                            echo "<script>alert('Email already exists');</script>";
                        } else {
                            $password = password_hash($password, PASSWORD_DEFAULT);
                            $sql = "INSERT INTO users (name, surname, email,
password, is_admin) VALUES ('$name', '$surname', '$email', '$password',
'$is_admin')";
                            $result = mysqli_query($conn, $sql);
                            if ($result) {
                                echo "<script>alert('Registration successful');
window.location.href = 'index.php';</script>";
                            } else {
                                echo "<script>alert('Registration
failed');</script>";
                            }
                        }
                    }
                }
                ?>

            <form method="POST">
                <div class="field input">
                    <label for="name">Name</label>
                    <input type="text" name="name" id="name" autocomplete="off"
required>
                </div>

                <div class="field input">
                    <label for="surname">Surname</label>
                    <input type="text" name="surname" id="surname"
autocomplete="off" required>
                </div>

                <div class="field input">
                    <label for="email">Email</label>
                    <input type="text" name="email" id="email" autocomplete="off"
required>
                </div>
```

```
                <div class="field input">
                    <label for="password">Password</label>
                    <input type="password" name="password" id="password"
autocomplete="off" required>
                </div>

                <div class="field">

                    <input type="submit" class="btn" name="submit" value="Register"
required>
                </div>
                <div class="links">
                    Already a member? <a href="login.php">Sign In</a>
                </div>
            </form>
        </div>
      </div>
</body>
</html>
```

The HTML file provided is the **Registration** page for the website. The structure of the document follows standard HTML conventions, starting with the doctype declaration, and includes metadata such as the character set and viewport settings for responsiveness. It is also linked to an external stylesheet, **style.css**, which is responsible for the page's visual layout.

Within the body, embedded PHP code manages the registration logic.
The `include_once` statements integrate database connection and session management functionality from external PHP files.

```
<?php
    // start session
    session_start();

?>
```

```
<?php
    $conn = mysqli_connect('localhost', 'root', '', 'food_delivery');

    if (!$conn) {
        die("Connection failed: " . mysqli_connect_error());
    }
?>
```

Upon form submission, the PHP script checks for valid email format, ensures that all required fields are filled, and enforces a minimum password length of 8 characters. If the email already exists in the database, the user is alerted with an error message.

When the form inputs pass validation, the password is hashed using `password_hash` to enhance security, and a new user record is inserted into the database. If the registration is successful, the user is redirected to the homepage with a success message. In case of a failure, an error message is displayed to the user.

```
$password = password_hash($password, PASSWORD_DEFAULT);
                            $sql = "INSERT INTO users (name, surname, email,
password, is_admin) VALUES ('$name', '$surname', '$email', '$password',
'$is_admin')";
                            $result = mysqli_query($conn, $sql);
```

```
if ($result) {
                                echo "<script>alert('Registration successful');
window.location.href = 'index.php';</script>";
                            } else {
                                echo "<script>alert('Registration
failed');</script>";
                            }
```

The registration form contains fields for the user's name, surname, email, and password, using the POST method to send the data for processing. There is also a link provided for users who already have an account, guiding them to the **Login** page.

The layout of the registration page is controlled by the external **style.css** file, separating the content from the presentation and making the code more maintainable. In summary, this page provides a secure, user-friendly interface for new users to register on the website.

5.3. Admin Privileges

There is an authorization check to ensure that only users with admin privileges can access this endpoint.

The PHP code below checks if the currently authenticated user has the role of an admin. It requires the inclusion of a "session.php" file, which is likely to handle session-related functionalities, and a "conn.php" file, which probably establishes a connection to the database. The code retrieves the current user's ID from the session (`$_SESSION['id']`) and then queries the database to retrieve the corresponding user's role. If the user's role is identified as an admin (where 'is_admin' is equal to 1), the script returns true, indicating that the user has administrative privileges. If the user is not identified as an admin, the script returns false. This code is a part of an authentication mechanism that checks the user's role to

determine their level of access or privileges within the website. It's a common practice to control and restrict access to certain functionalities based on user roles for security reasons.

```php
<?php
    require_once("session.php");
    require_once("conn.php");

    // get current user id
    $userId = $_SESSION['id'];

    // get current user role
    $sql = "SELECT is_admin FROM users WHERE id = '$userId'";
    // if admin return true
    $result = mysqli_query($conn, $sql);
    $row = mysqli_fetch_assoc($result);
    if ($row['is_admin'] == 1) {
        return true;
    }
    // if not admin return false
    return false;


?>
```

Admins can update and delete users.

- Delete users

The "deleteUser.php" file in the controllers folder handles the deletion of a user from the database. It first checks whether a user ID is provided in the request; if not, it redirects the user to the home page. Assuming a user ID is present, it includes the necessary connection to the database and executes a SQL DELETE query to remove the user with the specified ID. The result of the deletion operation is then processed, and an appropriate alert message is generated, informing the user whether the deletion was successful or not. Overall, this file seems to encapsulate the control flow and database interaction logic associated with user deletion.

```php
<?php
include_once "utils/session.php";
include_once "php/conn.php";

if (!isset($_SESSION['id']) || !isset($_GET['id'])) {
    header("Location: index.php");
    exit();
}

// Assuming the current user is an admin
```

```php
$admin_id = $_SESSION['id'];
$user_id_to_delete = $_GET['id'];

// Check if the current user is an admin
$sql = "SELECT is_admin FROM users WHERE id = '$admin_id'";
$result = mysqli_query($conn, $sql);
$row = mysqli_fetch_assoc($result);

if ($row['is_admin'] != 1) {
    header("Location: index.php");
    exit();
}

// Perform the deletion
$sql_delete = "DELETE FROM users WHERE id = '$user_id_to_delete'";
$result_delete = mysqli_query($conn, $sql_delete);

if ($result_delete) {
    echo "<script>alert('User deleted successfully'); window.location.href =
'usersList.php';</script>";
    exit();
} else {
    echo "<script>alert('User deletion failed'); window.location.href =
'usersList.php';</script>";
    exit();
}
?>
```

The "deleteUser.php" file in the utils folder contains utility functions related to user management. It includes the inclusion of session handling and assumes the current user is an admin. The file then performs user deletion based on certain conditions, such as checking the admin status of the current user. If the conditions are met, it proceeds to execute the SQL DELETE query to remove the specified user from the database. The file includes alert messages to notify the user about the outcome of the deletion operation. This file seems to serve as a collection of utility functions for common user-related tasks.

- Update users

The "updateRole.php" file in the controllers folder for updating user roles is responsible for toggling the admin status of a user. It checks if a user ID is provided; if not, it redirects the user to the home page. Assuming a user ID is present, it includes the necessary database connection and executes a SQL UPDATE query to toggle the "is_admin" field in the users table. Additionally, if the user updates their own role, it updates the session to reflect the

change. The file concludes by redirecting the user to the users' list page. This controller file handles the logic associated with updating user roles.

```php
<?php
$userId = isset($_REQUEST['id']) ? $_REQUEST['id'] : null;

    if (!$userId) {
        header("Location: ../index.php");
        die();
    }

    include_once "../php/conn.php";

    // update user role, if 1 make 0, if 0 make 1
    $sql = "UPDATE users SET is_admin = IF(is_admin = 1, 0, 1) WHERE id =
'$userId'";
    $result = mysqli_query($conn, $sql);


    // update session if the user is updating his own role
    include_once "../utils/session.php";
    if ($userId == $_SESSION['id']) {
        $_SESSION['is_admin'] = $_SESSION['is_admin'] == 1 ? 0 : 1;
    }

    header("Location: ../usersList.php");
    die();

?>
```

The "updateUserRole.php" file in the "utils" folder handles the update of user details such as name, surname, email, and password. It expects a form submission, retrieves the input values, and executes a SQL UPDATE query to modify the corresponding user's information in the database. The result of the query is processed, and an alert message is generated to notify the user about the success or failure of the update operation. This utility file focuses on the user's personal information update.

```php
<?php
include_once "php/conn.php";

if(isset($_POST['submit'])) {
    $name = $_POST['name'];
    $surname = $_POST['surname'];
    $email = $_POST['email'];
    $password = $_POST['password'];
```

```
    $sql = "UPDATE users SET name = '$name', surname = '$surname', email =
'$email', password = '$password' WHERE id = '$user_id'";
    $result = mysqli_query($conn, $sql);
    if ($result) {
        echo "<script>alert('Changes saved'); window.location.href =
'index.php';</script>";
    } else {
        echo "<script>alert('Changes failed');</script>";
    }
}
?>
```

- User's list

The following HTML code serves as the users' list page for a web application. It begins by including essential components such as the session handler, database connection, and header. The page is restricted to users with admin privileges, redirecting others to the home page. The main content displays a list of users with their details, including ID, full name, email, role, and action buttons.

```
<div class="bottom">
                <table>
                    <thead>
                        <tr>
                            <th>ID</th>
                            <th>Full Name</th>
                            <th>Email</th>
                            <th>Role</th>
                            <th>Action</th>
                        </tr>
                    </thead>
                    <tbody>
```

The dynamic rendering of user data is achieved through PHP, fetching and displaying user information from the database. The page provides a table layout with columns for user attributes and action buttons allowing administrators to delete users or update their roles. The user roles are presented as "Admin" or "Guest" based on the "is_admin" field in the database. Overall, this page facilitates the administration of users by providing essential functionalities in a clear and organized manner.

```
<?php
    require_once("session.php");
    require_once("conn.php");

    // get current user id
    $userId = $_SESSION['id'];
```

```php
    // get current user role
    $sql = "SELECT is_admin FROM users WHERE id = '$userId'";
    // if admin return true
    $result = mysqli_query($conn, $sql);
    $row = mysqli_fetch_assoc($result);
    if ($row['is_admin'] == 1) {
        return true;
    }
    // if not admin return false
    return false;


?>
```

5.4. Account

This HTML file represents the account page of a web application. It starts by including essential components such as the session handler, database connection, and header. The page is restricted to authenticated users, redirecting others to the login page. The main content dynamically fetches the current user's information from the database and populates the form fields with the retrieved data.

```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel= stylesheet href="styles/account.css">
    <title>Account Page</title>

</head>
<body>
        <?php include_once "utils/session.php";
            if (!isset($_SESSION['id'])) {
                header("Location: index.php");
            }
            include_once "php/conn.php";
            // get current user's info
            $user_id = $_SESSION['id'];
            $sql = "SELECT * FROM users WHERE id = '$user_id'";
            $result = mysqli_query($conn, $sql);
            $row = mysqli_fetch_assoc($result);
            $name = $row['name'];
```

```
            $surname = $row['surname'];
            $email = $row['email'];
            $password = $row['password'];
            $is_admin = $row['is_admin'];

            // update user's info with logic
            if(isset($_POST['submit'])){
                $name = $_POST['name'];
                $surname = $_POST['surname'];
                $email = $_POST['email'];
                $password = $_POST['password'];
                if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
                    echo "<script>alert('Invalid email format');</script>";
                } else if (empty($name) || empty($surname) || empty($password)) {
                    echo "<script>alert('Please fill in all fields');</script>";
                } else if (strlen($password) < 8) {
                    echo "<script>alert('Password should be at least 8 characters
long');</script>";
                } else {
                    $sql = "SELECT * FROM users WHERE email = '$email'";
                    $result = mysqli_query($conn, $sql);
                    $num_rows = mysqli_num_rows($result);
                    if ($num_rows > 0) {
                        $row = mysqli_fetch_assoc($result);
                        if ($row['id'] != $user_id) {
                            echo "<script>alert('Email already exists');</script>";
                        } else {
                            if ($password != $row['password']) {
                                $password = password_hash($password,
PASSWORD_DEFAULT);
                            }
                            $sql = "UPDATE users SET name = '$name', surname =
'$surname', email = '$email', password = '$password' WHERE id = '$user_id'";
                            $result = mysqli_query($conn, $sql);
                            if ($result) {
                                echo "<script>alert('Changes saved');
window.location.href = 'index.php';</script>";
                            } else {
                                echo "<script>alert('Changes failed');</script>";
                            }
                        }
                    } else {
                        if ($password != $row['password']) {
                            $password = password_hash($password, PASSWORD_DEFAULT);
                        }
                        $sql = "UPDATE users SET name = '$name', surname =
'$surname', email = '$email', password = '$password' WHERE id = '$user_id'";
                        $result = mysqli_query($conn, $sql);
                        if ($result) {
                            echo "<script>alert('Changes saved');
window.location.href = 'index.php';</script>";
```

```
                        } else {
                            echo "<script>alert('Changes failed');</script>";
                        }
                    }
                }
            }
        ?>
        <div class="container-account">
        <div class="box form-box-account">
            <header>
                <?php echo "<h3>Welcome, $name!</h3>"; ?>
            </header>
            <text>
                <p>Here you can change your personal information</p>
            </text>
            <form action="" method="post">
                <div class="field input">
                    <label for="name">Name</label>
                    <input type="text" name="name" id="name" autocomplete="off"
value="<?php echo $name; ?>">
                </div>

                <div class="field input">
                    <label for="surname">Surname</label>
                    <input type="text" name="surname" id="surname"
autocomplete="off" value="<?php echo $surname; ?>">
                </div>

                <div class="field input">
                    <label for="email">Email</label>
                    <input type="text" name="email" id="email" autocomplete="off"
value="<?php echo $email; ?>">
                </div>

                <div class="field input">
                    <label for="password">Password</label>
                    <input type="password" name="password" id="password"
autocomplete="off" value="password">
                </div>

                <div class="field">
                    <input type="submit" class="btn" name="submit" value="Save
Changes" required>
                </div>

                <a href="index.php" class="btn">Return to home page</a>
            </form>
        </div>
    </div>
</body>
</html>
```

The form allows users to update their personal information, including name, surname, email, and password. Client-side validation is implemented to ensure that the email format is correct, and all fields are filled appropriately. Additionally, the password must be at least 8 characters long. Server-side checks are performed to avoid duplicate emails, and password hashing is handled to maintain security.

The PHP logic within the file manages the update process, validating and processing user input, and providing feedback messages. The page maintains a clean and user-friendly design, facilitating users in modifying their account details. The footer and header are included to ensure consistency with the overall site structure.

5.6. Navigation and Design

- Auth logic

In the website's navigation system, we've implemented a security feature known as "auth logic." This ensures that only authenticated users with an active session are permitted to navigate through various URLs. If a user attempts to access pages other than registration and login without an active session, the system will prevent them from doing so. This mechanism helps enhance security by restricting unauthorized access to certain parts of the website and ensures that only logged-in users can explore its full functionality.

```php
<?php
    require_once("session.php");

    // check if user authed
    if(!isset($_SESSION['id'])) {
        header("Location: index.php");
        die();
    }
?>
```

- App.js

The provided JavaScript code is designed to manage various interactive elements and functionalities of a website, such as the mobile navigation menu, product carousel, shopping cart, and search functionality. Here's a breakdown of what each section does:

1. Mobile Menu Toggle

- The code listens for a click event on a **menu-toggle** element (typically a hamburger icon).
- When clicked, it toggles the `is-active` class on the **menu-toggle** and the `active` class on the **sidebar**. This is typically used to show or hide the navigation menu on mobile devices.
- Additionally, when a user clicks on the **sidebar**, the menu will close if the screen width is less than or equal to 768px, ensuring the mobile menu can be closed once a link is clicked.

```
const mobile = document.querySelector('.menu-toggle');
const mobileLink=document.querySelector('.sidebar');

mobile.addEventListener("click", function(){
    mobile.classList.toggle("is-active");
    mobileLink.classList.toggle("active");

});

//close menu when click
mobileLink.addEventListener("click",function(){
    const menuBars=document.querySelector(".is-active");
    if(window.innerWidth<=768 && menuBars){
        mobile.classList.toggle("is-active");
        mobileLink.classList.toggle("active");
    }
});
```

2. Horizontal Scrolling for Menus

- The code implements horizontal scrolling for two different sections: **highlight-wrapper** and **filter-wrapper**.
- It listens for click events on **back** and **next** buttons to scroll the content left or right by a specified amount (`step` or `stepFilter`). This functionality is useful for showcasing items in a carousel-like layout.

```
var step=100;
var stepFilter=60;
var scrolling=true;

$(".back").bind("click",function(e){
    e.preventDefault();
    $(".highlight-wrapper").animate({
        scrollLeft:"-="+step+"px"
    });
});
```

```
•
•   $(".next").bind("click",function(e){
•       e.preventDefault();
•       $(".highlight-wrapper").animate({
•           scrollLeft:"+="+step+"px"
•       });
•   });
•
•   $(".back-menus").bind("click",function(e){
•       e.preventDefault();
•       $(".filter-wrapper").animate({
•           scrollLeft:"-="+step+"px"
•       });
•   });
•
•   $(".next-menus").bind("click",function(e){
•       e.preventDefault();
•       $(".filter-wrapper").animate({
•           scrollLeft:"+="+step+"px"
•       });
•   });
•
```

3. Shopping Cart Popup

- Functions `toggleCartPopup` and `closeCart` control the visibility of the shopping cart popup. When the cart icon is clicked, it either shows or hides the cart by toggling the `active` class.
- The popup allows users to view the items in their cart and proceed to checkout.

```
function toggleCartPopup(){
    const cartPopup=document.getElementById('cart-popup');
    cartPopup.classList.toggle('active');
}

//for close cart popup
function closeCart(){
    const cartPopup=document.getElementById('cart-popup');
    cartPopup.classList.remove('active')
}
```

4. Adding Items to Cart

- The function `addToCart` adds a product to the shopping cart by creating a new row in the cart table. It checks if the product already exists in the cart, and if so, it increments the quantity.

- For new items, it adds a row with buttons for increasing or decreasing the item quantity, and updates the total price for each item and the entire cart.

```
function addToCart(itemName, price) {
    const cartItems = document.getElementById('cart-
items').getElementsByTagName('tbody')[0];

    // Проверяем, есть ли товар в корзине
    const existingRow = Array.from(cartItems.getElementsByTagName('tr')).find(row
=> {
        return row.querySelector('.item-name').textContent === itemName;
    });

    if (existingRow) {
        // Если товар уже в корзине, увеличиваем количество
        const countElement = existingRow.querySelector('.item-count');
        countElement.textContent = parseInt(countElement.textContent) + 1;
        updateRowTotal(existingRow, price);
    } else {
        // Добавляем новый товар в корзину
        const newRow = `
            <tr>
                <td class="item-name">${itemName}</td>
                <td>
                    <button class="decrease-qty">-</button>
                    <span class="item-count">1</span>
                    <button class="increase-qty">+</button>
                </td>
                <td class="item-price">${price.toFixed(2)}</td>
                <td class="item-total">${price.toFixed(2)}</td>
            </tr>
        `;
        cartItems.insertAdjacentHTML('beforeend', newRow);
    }

    updateCartTotal();
}
```

5. Updating Cart Totals

- The function `updateRowTotal` recalculates the total price for an item in the cart whenever the quantity changes.
- The function `updateCartTotal` updates the total sum for all items in the cart by iterating through all rows and summing the item totals.

```
function updateRowTotal(row, price) {
    const count = parseInt(row.querySelector('.item-count').textContent);
    const totalElement = row.querySelector('.item-total');
    totalElement.textContent = (count * price).toFixed(2);
    updateCartTotal();
}

function updateCartTotal() {
    let totalSum = 0;
    document.querySelectorAll('.item-total').forEach(el => {
        totalSum += parseFloat(el.textContent);
    });
    document.getElementById('cart-total').textContent = totalSum.toFixed(2);
}
```

6. Handling Quantity Changes

- Event listeners are added to the cart items for the increase and decrease buttons.
- When clicked, the `handleQuantityChange` function adjusts the quantity of the item in the cart, updates the item's total price, and recalculates the cart's total sum. If the quantity reaches zero, the item is removed from the cart.

```
function handleQuantityChange(button, change) {
    const row = button.closest('tr'); // Находим родительскую строку
    const countElement = row.querySelector('.item-count'); // Элемент количества
    const price = parseFloat(row.querySelector('.item-price').textContent); // Цена товара
    let currentCount = parseInt(countElement.textContent) + change;

    if (currentCount > 0) {
        countElement.textContent = currentCount; // Обновляем количество
        row.querySelector('.item-total').textContent = (price * currentCount).toFixed(2); // Обновляем стоимость
    } else {
        row.remove(); // Удаляем строку, если количество 0
    }

    updateCartTotal(); // Обновляем итоговую сумму корзины
}

```

7. Search Functionality

- The `searchMenu` function listens for user input in a search field and filters the displayed menu items based on the product name. It hides items that don't match the search query and shows those that do.
- This allows users to easily search through a list of menu items or products.

```
function searchMenu() {
    var input = document.getElementById("searchInput").value.toLowerCase();
// Получаем текст из поля ввода
    var menuItems = document.querySelectorAll(".detail-card"); // Получаем
все карточки

    menuItems.forEach(function(item) {
        var productName = item.querySelector(".detail-name
h4").innerText.toLowerCase(); // Название блюда

        if (productName.includes(input)) {
            item.style.display = ""; // Показываем элемент
        } else {
            item.style.display = "none"; // Скрываем элемент
        }
    });
}
```

8. Redirect to Checkout

- The function `redirectToCheckout` checks if there are any items in the cart. If the cart is not empty, the user is redirected to the checkout page. If the cart is empty, an alert prompts the user to add items before proceeding to checkout.

```

function redirectToCheckout() {
    const cartItems = document.getElementById('cart-
items').getElementsByTagName('tbody')[0];
    if (cartItems && cartItems.getElementsByTagName('tr').length > 0) {
        window.location.href = 'checkout.php';
    } else {
        alert('Your cart is empty. Please add items to the cart before
proceeding to checkout.');
    }
}

```

Summary

This script provides essential interactivity for an e-commerce or restaurant-style website, including:

- **Responsive mobile menu**: Toggle and close functionality.
- **Horizontal scrolling**: For carousel-like navigation of items.
- **Shopping cart**: Adding items, updating quantities, showing total prices, and managing the cart popup.
- **Search**: Filtering items based on user input.
- **Checkout**: Redirecting users to the checkout page only if the cart contains items.

The code ensures smooth user interactions with dynamic features like navigation, product management, and checkout, enhancing the user experience on both desktop and mobile devices.