Department of Mathematics and Computer Science
Faculty of Data Analysis
UNIME

Web Programming Report
Food Recipes website

Student: Alikhan Alashybay (536353)
Student: Alina Atayeva (533021)
Professor: Armando Ruggeri

Academic Year 2023/2024

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 our project, we 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, we chose to implement a Food Recipe website. Our goal was to craft a platform where people can easily discover and share recipes. The platform has two UIs (User Interfaces), one for guests and one for administrators of the website. The admins are the moderators of the website. Admin's UI also allows them to manage the users themselves. Guests have a basic interface where they can change their basic information, browse recipes, and also share their own.

2. Technology Stack

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

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

In our project, we aimed for simplicity and clarity by breaking down our work into smaller parts. Each part, or component, serves a specific purpose, making the overall code more organized and easier to understand. This approach helps us manage different aspects of the website without getting overwhelmed by a massive amount of code. Similarly, in our CSS (styling) work, we adopted a straightforward strategy. Instead of having specific styles for every little element, we used general styles that apply broadly. This not only reduces the complexity of our code but also maintains a clean and uncluttered workspace for a smoother development process.

We leveraged the power of app.js to streamline functionality. This central file served as the go-to hub for calling functions and implementing various actions like creating, updating, and deleting. By consolidating these operations in one place, we kept our codebase neat and well-organized. This approach not only simplified the coding process but also made it easier to troubleshoot and enhance functionalities. In essence, app.js became the backbone of our project, orchestrating different actions seamlessly for a smoother user experience.

3. Database Structure

The database consists of three tables: food_recipes, recipe_categories, and users.

Table	Action	Rows	Type	Collation	Size	Overhead
food_recipes	Browse Structure Search Insert Empty Drop	6	InnoDB	utf8mb4_general_ci	16.0 KiB	-
recipe_categories	Browse Structure Search Insert Empty Drop	12	InnoDB	utf8mb4_general_ci	16.0 KiB	-
users	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_general_ci	16.0 KiB	-
3 tables	Sum	22	InnoDB	utf8mb4_general_ci	48.0 KiB	0 B

Food_recipes tables consist of the unique identification numbers for each recipe, a title, a description, additional information about the steps of the recipe, or anything besides that, the author, and a category name. Where the author is bound by the session id and user id consequently, and by matching the user id to the name, it writes the name of the user as the author of the recipe created. Category name is chosen on the web page of several choices available, when one is chosen, the recipe is then bound to the category title, from the next table called “category_name”. The user id is set to autoincrement so that each new user will get the next id from the previously created user, this ensures that no users will end up having the same id number.

	id	title	description	additional_info	author	category_name
Edit Copy Delete	19	New food	some new things	1, do 2, bla bla		
Edit Copy Delete	21	ALikhan's	fooooood!	hello world		
Edit Copy Delete	22	new	some	saf	Alikhan admin	Breakfast
Edit Copy Delete	23	cool	is it worth it	i like to train	Alikhan admin	Dinner
Edit Copy Delete	24	asf	as	asdf	Alikhan admin	Salad
Edit Copy Delete	25	New Show from kila	the best ever food you will ever try in your live	its me)	Alikhan admin	Appetizer

Recipe_categories table is made of two attributes (columns): id and title. The id of the category is the primary key, and the title is basically the name of the category.

	id	title
Edit Copy Delete	1	Vegetarian
Edit Copy Delete	10	Breakfast
Edit Copy Delete	11	Lunch
Edit Copy Delete	12	Dinner
Edit Copy Delete	13	Appetizer
Edit Copy Delete	14	Salad
Edit Copy Delete	15	Main-course
Edit Copy Delete	16	Side-dish
Edit Copy Delete	17	Dessert
Edit Copy Delete	18	Snack
Edit Copy Delete	19	Soup
Edit Copy Delete	20	Holiday

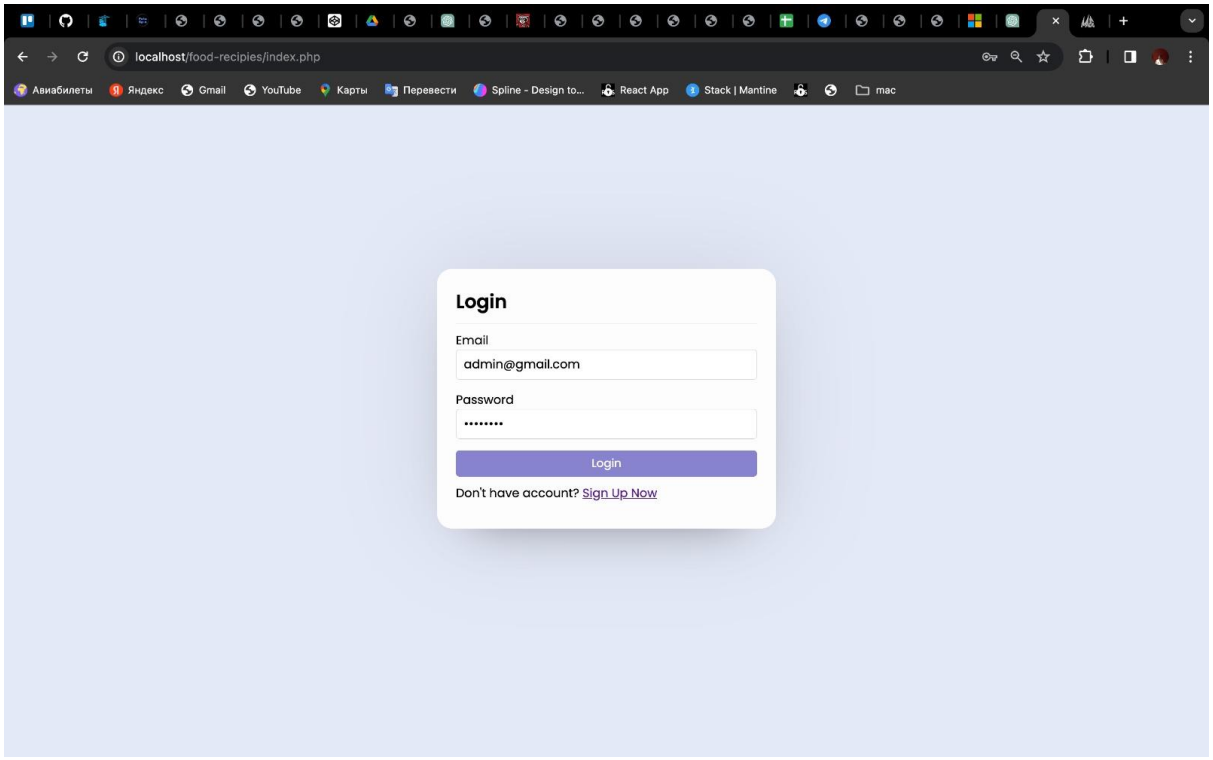
The users table includes the following attributes: id of the user, name, and surname of the user, email address, password, and the is_admin. The password is hashed once a new user is registered for security measures, and as we know the hashed passwords cannot be "unhashed", so this sensitive information 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.

	id	name	surname	email	password	is_admin
Edit Copy Delete	19	Alikhan	admin	admin@gmail.com	$2y$10$W3nJ3JUOhIPOL17Ktlj2buNZZXpw/cP//ySkTtF2U6n...	1
Edit Copy Delete	29	change	asdf	asd@gmail.com	$2y$10$275oMfPFqj1gO3Xix2QXWeRLqY7tzOsx3hyVjUdOLEd...	0
Edit Copy Delete	31	Alikhan	Guest	guest@gmail.com	$2y$10$UCQfwtP/CDHUeQDlZbDEkOYU1nyI8kp8aVUrSUX0h9P...	0
Edit Copy Delete	32	Alina	Atayeva	alina@gmail.com	$2y$10$vrGX7V6ipnlTe7rATZ2ry.RpjG3SWsQL7b3X7oIwLTW...	0

4. Web Pages

- Authorization page

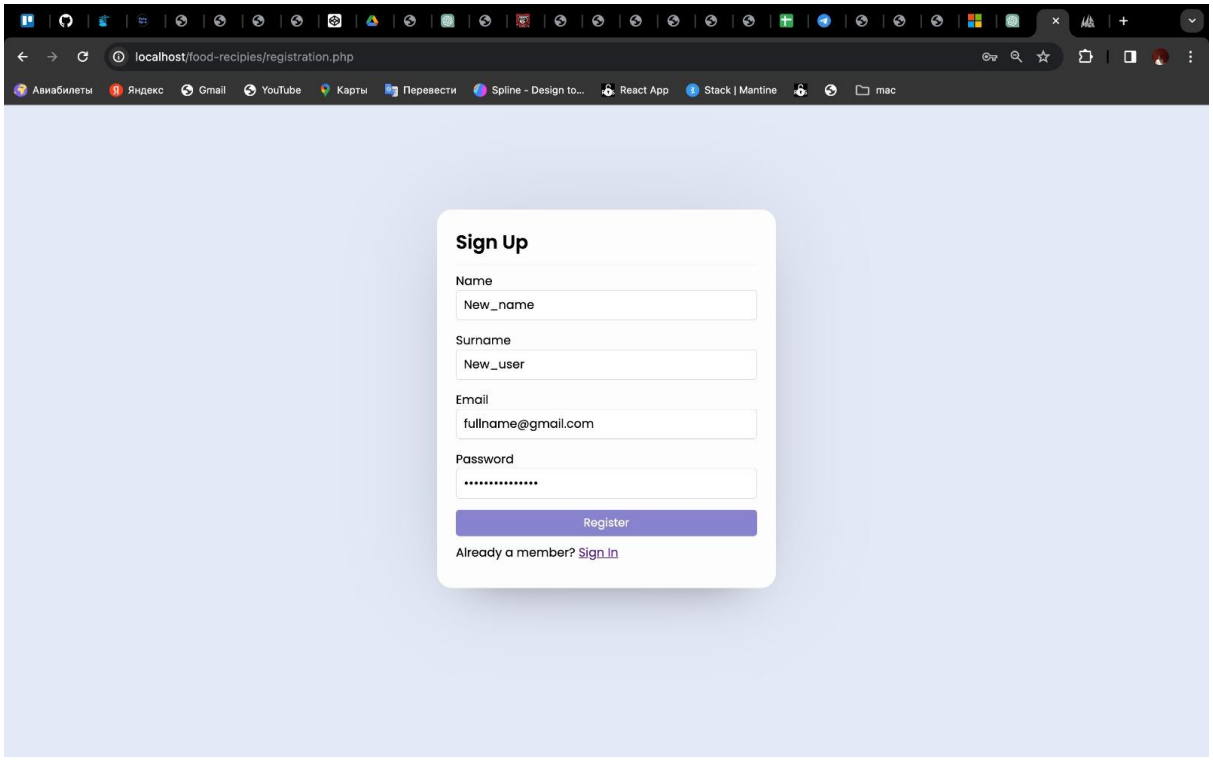
It is the index page, where the user has a choice to either authorize/login or register as a new user. The login page asks for the email of the user, which is the username in our case, and the password. Below the login page, there is a "Sign Up Now" link, by pressing which a user will be transferred 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.



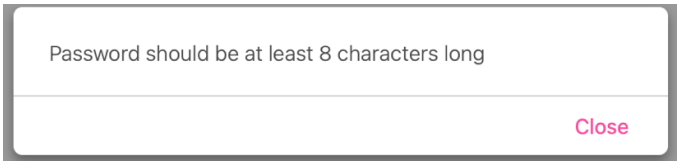
When login is successful the following message pops up, by pressing on “close” button the user will be redirected to the home page of the website.

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.

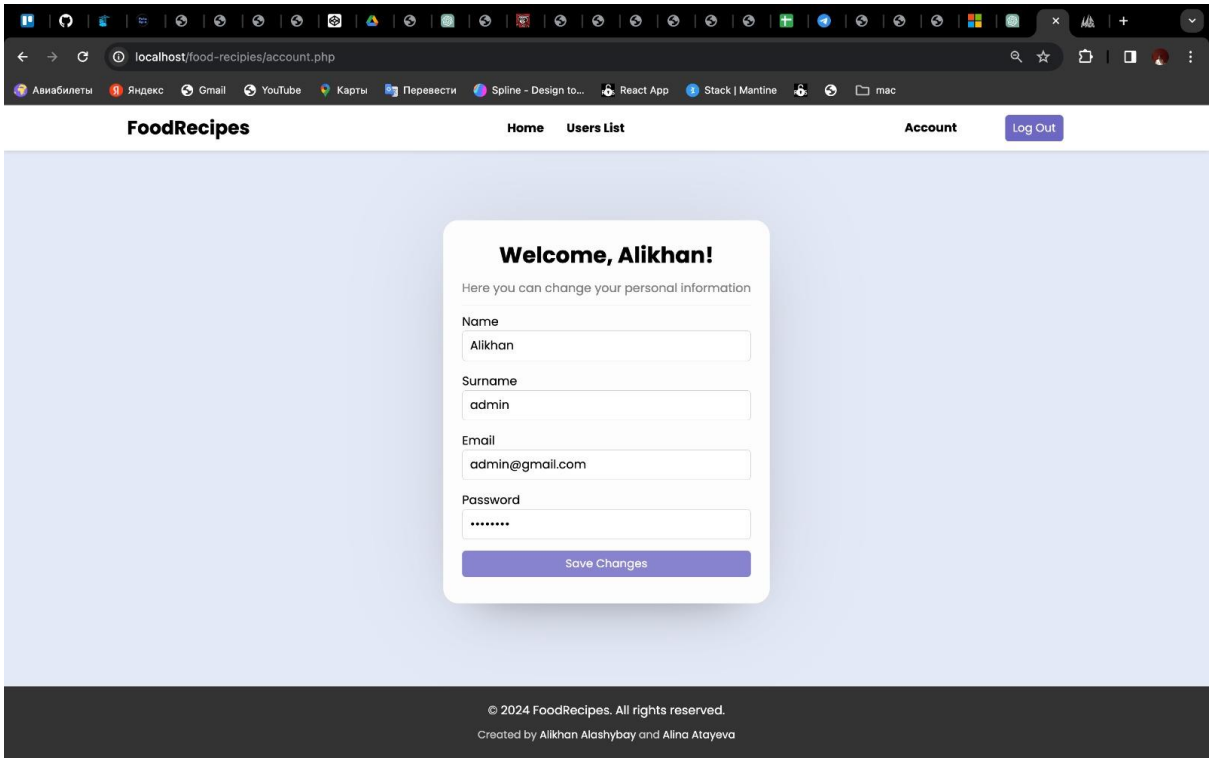


When the user has created a strong password that complies with the rules, the “registration successful” message appears on the page. By clicking “close” the user is redirected to the login page, where he/she can enter the login information to access the system.

- 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 user-

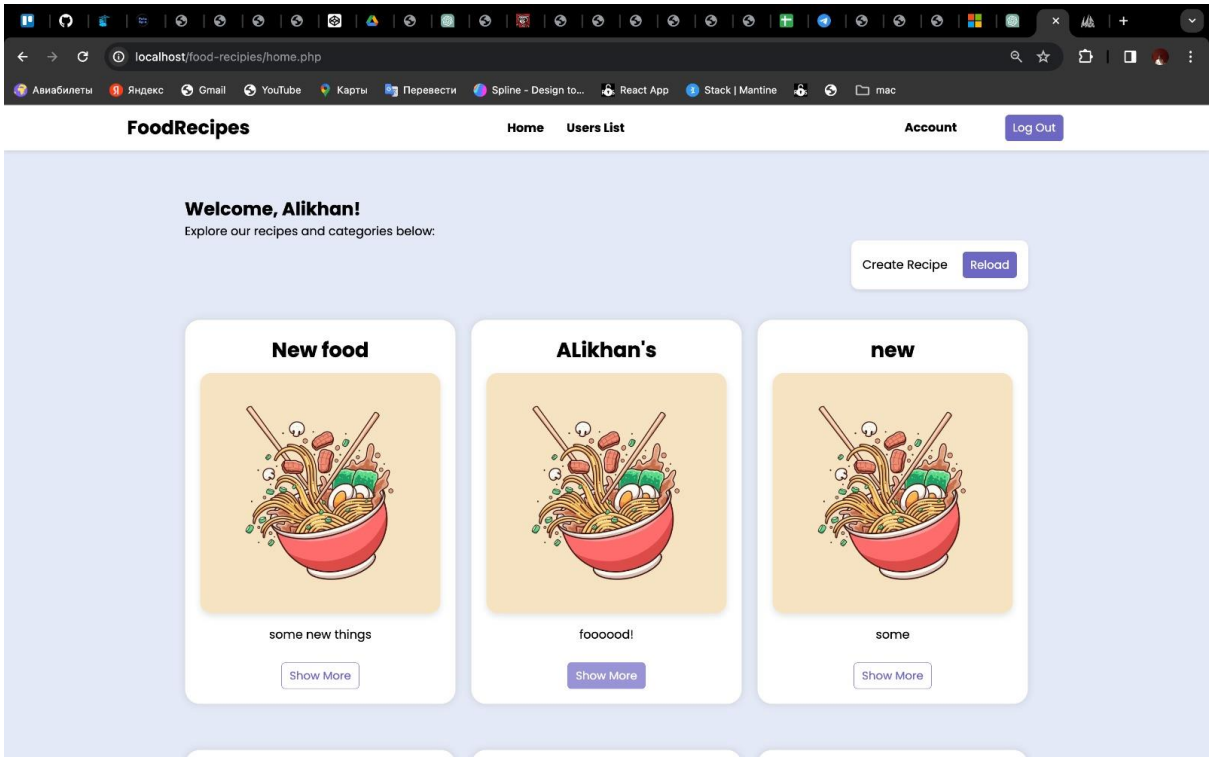
centric design, individuals can seamlessly interact with their account information, facilitating a straightforward process for updating and saving changes. The Account component embodies our commitment to providing users with control over their personal data in a secure and accessible manner.



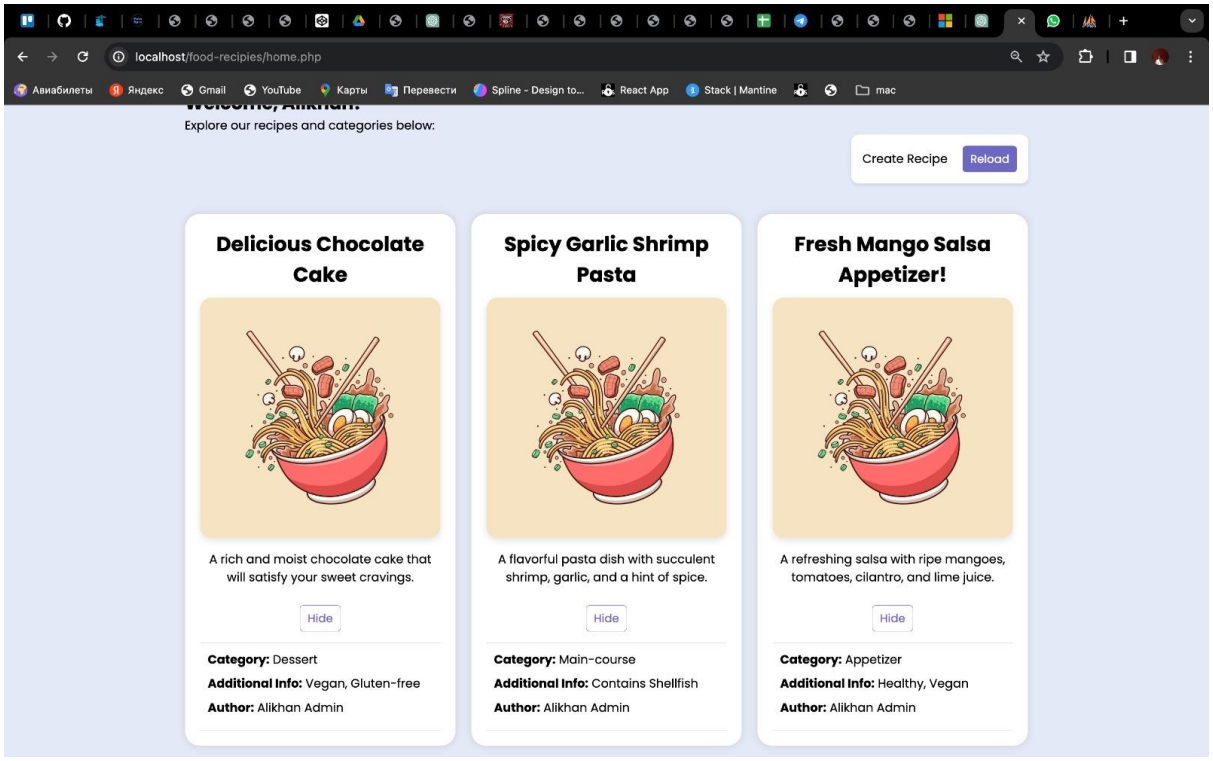
The account page provides an option for the user to change their personal information. All fields should be filled so that the SQL query UPDATES these fields in the table. After done, the user must press the “Save Changes” button, prompting the code to call for the SQL query and modify the changes.

- Home page

The Home page serves as a dynamically personalized welcome interface. Leveraging user authentication data, it greets users by name, fostering a tailored and engaging experience. Beyond this personalized touch, the Home page is structured with recipe cards. These cards act as visual cues, presenting curated recipes to captivate user interest. The design prioritizes simplicity and intuitiveness, aiming to enhance user engagement and facilitate seamless navigation. This academic approach ensures a user-centric and intellectually sound foundation for our web application.



The home page includes the cards with the recipes and each card can be expanded using the “show more” button.

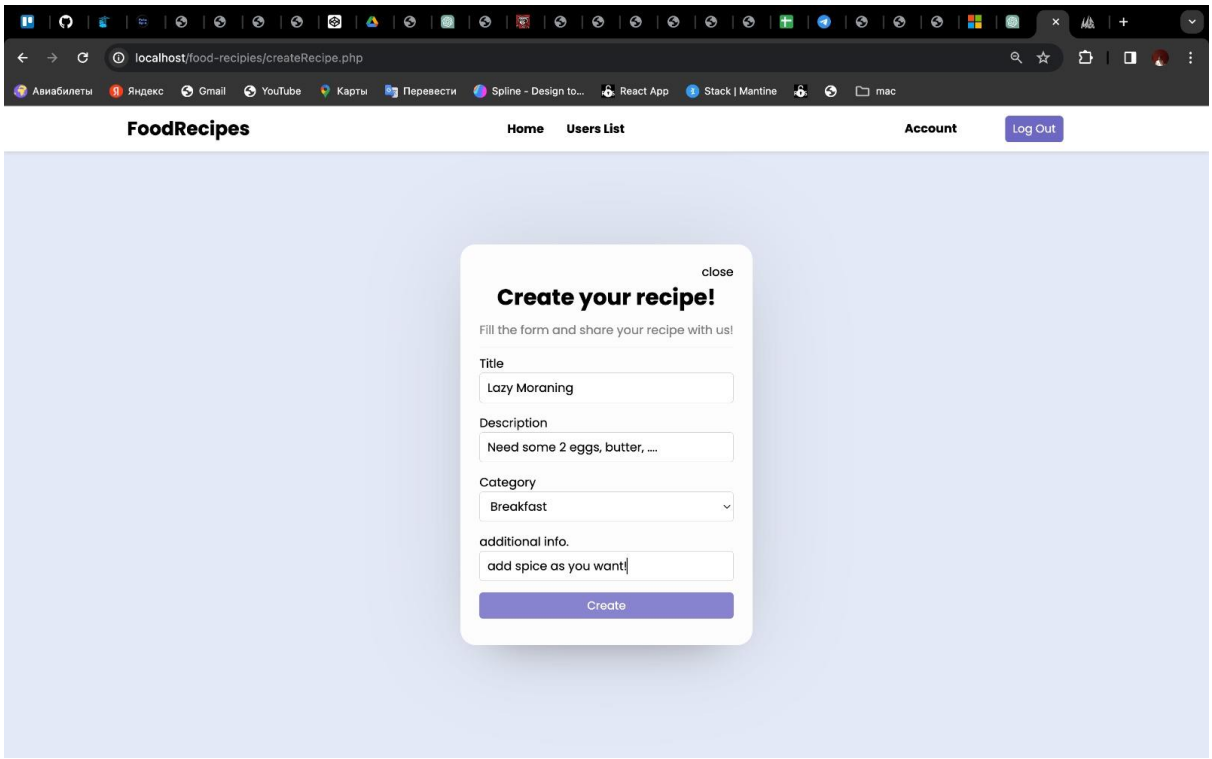


Each card shows information about the category name, information on how the recipe steps, and the author’s name.

There is also a reload button on the right top of the web page for all pages. Its function is quite simple and necessary. When making a request, there might be a scenario when the request is not sent, it reloads the page and sends the request.

- Create recipe page

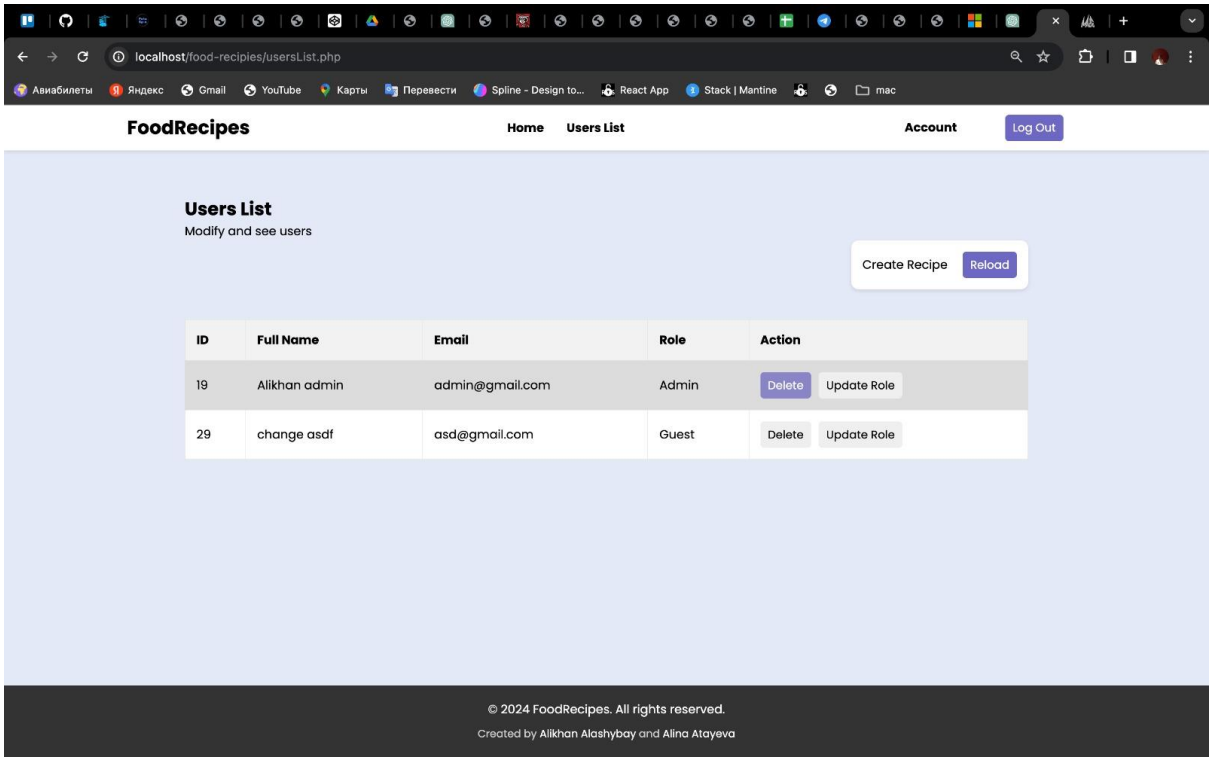
Both guests and admins can create recipes. Hence, the link to the Create recipe page is available for both guest users and admins. The create recipe page includes a form, where users can fill in the fields with the relative information on the recipe creation. The title, description, category, and additional information are inserted into a database via SQL query and later pulled to the recipe card from the database.



- Users' list page

In the Users List functionality designed for administrators, a comprehensive view of all users and their current roles is provided. This feature allows administrators to oversee the user base, facilitating efficient management. Admins have the authority to perform actions such as deleting users or changing their roles directly from this interface. The process is seamless and doesn't necessitate manual page updates; the changes are reflected instantly within the function. This real-time responsiveness enhances the user experience for administrators, ensuring a dynamic and efficient approach to user management. The Users List feature

encapsulates an effective and user-friendly administrative tool, offering a centralized hub for user oversight and streamlined role management.



After the update, there is no need for a manual page update to see the result as it's inside of the function.

5. Key Features

5.1. Database connection

```php
<?php
    $conn = mysqli_connect('localhost', 'root', '', 'food-recipies');

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

This .php file is served to connect to the database which is called "food-recipies". This is a standard code used for connecting to a specific database on localhost. Later, all of 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 below is the index page of the website, and it is the login page. The structure of the HTML document is conventional, starting with a doctype declaration and including metadata in the head section. The `meta` tags specify character set, viewport settings for responsiveness, and compatibility with Internet Explorer. The page is linked to an external stylesheet for styling.

index.php

```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/style.css">
    <title>Login</title>
</head>
```

Within the body, PHP code is embedded to manage user sessions and handle form submissions. The `include_once` statement is used to incorporate session management logic from an external PHP file. Checks are performed to redirect a user already logged in to the home page. The actual login process involves validating the email format, checking for empty fields, querying the database to verify the email's existence, and using `password_verify` to compare the entered password with the hashed password stored in the database. Appropriate alerts are displayed based on different scenarios, such as an invalid email, incorrect password, or a successful login.

```php
<?php
    include_once "utils/session.php";
    if (isset($_SESSION['id'])) {
        header("Location: home.php");
    }
    include_once "php/conn.php";
    if(isset($_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 = 'home.php';</script>";
                } else {
                    echo "<script>alert('Incorrect password');</script>";
                }
            } else {
                echo "<script>alert('Email does not exist');</script>";
            }
```

The HTML form is structured to collect user credentials with fields for email and password. The form uses the POST method, and upon submission, the user input is processed by the embedded PHP code. Additionally, there is a link provided for users who do not have an account, directing them to the registration page.

```
            <header> Login </header>
            <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>
```

The appearance and layout of the login page are managed by an external stylesheet referenced in the head section called “style.css”. This separation of concerns enhances the maintainability and organization of the code. Overall, the login page combines HTML and PHP to create a functional and secure authentication mechanism for users.

5.2. User Registration

In the realm of authentication, we crafted a user-friendly experience. The authorization encompasses two essential parts: login and registration. These components enable users to create accounts and securely log in to access personalized features. To ensure security and prevent unauthorized access, we implemented a session mechanism. This session feature acts like a digital key, ensuring users can navigate through the website seamlessly while keeping unwanted visitors at bay. In essence, authorization serves as the digital bouncer, allowing only authenticated users to explore the different corners of our website.

The logic implemented in the registration process of the web platform follows a clear and secure path. When a user registers, their information is added to the database, and they are seamlessly redirected to the login page. Subsequently, during the login process, the system checks if the entered credentials exist in the database. If a match is found, the user gains access; otherwise, entry is denied. To safeguard user privacy, we've incorporated a default PHP hashing function, ensuring that sensitive information is protected.

The PHP block at the beginning of the file checks whether the user is already logged in using the '$_SESSION' variable. If so, it redirects the user to the home page. This ensures that a logged-in user is not allowed to access the registration page.

```php
<?php include_once "php/conn.php";
    include_once "utils/session.php";
    if (isset($_SESSION['id'])) {
        header("Location: home.php");
    }
```

A session is started using the following function, which is located in the utils folder, in the session.php file.

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

?>
```

The PHP block within the form processing handles user registration. It validates the input fields, checks for existing email addresses in the database, hashes the password and inserts the user's information into the database.

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

Below is the code for hashing the passwords.

```php
$password = password_hash($password, PASSWORD_DEFAULT);
```

Other parts are the html codes of the elements we see on the page. They are hence styled in CSS. The CSS codes are stored in one CSS file called "style.css" in the "Styles" folder.

```html
<header>Sign Up</header>
<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="index.php">Sign In</a>
    </div>
</form>
```

The PHP code below is a basic authentication check to ensure that a user is authenticated before accessing certain content. It includes a file named "session.php," which is expected to contain session-related functionality. The script checks whether the 'id' key is set in the session (`$_SESSION`). If the 'id' is not set, indicating that the user is not authenticated, the script redirects them to the "index.php" page. This authentication mechanism is a common practice to restrict access to certain parts of a website to authenticated users. It emphasizes the importance of session management for user authentication and redirects unauthorized users to the login page for security purposes.

utils > isAuth.php

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

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

5.3. Admin Privileges

There is an authorization check to ensure that only users with admin privileges can access this endpoint. If the user is not an admin, a 403 Forbidden response is sent, and the script terminates.

```php
//auth logic
if(!isAdmin()) {
    http_response_code(403);
    die();
}
```

A SQL query is executed to retrieve all users from the database.

```php
// get users
$sql = "SELECT * FROM users";
$result = mysqli_query($conn, $sql);
```

The script checks whether the query was successful (**isset($result)**). If not, it returns a JSON response with an error message.

```php
if(!isset($result)) {
    return json_encode([
        "message" => "Users not found"
    ]);
}
```

The script fetches the first row of the result set using **mysqli_fetch_assoc($result)** and echoes the user data in JSON format.

```php
// get users
$users = mysqli_fetch_assoc($result);
echo json_encode($users);
```

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
utils > isAdmin.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
controllers > deleteUser.php
<?php
$userId = isset($_REQUEST['id']) ? $_REQUEST['id'] : null;

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

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

    $sql = "DELETE FROM users WHERE id = '$userId'";
    $result = mysqli_query($conn, $sql);

?>
```

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.

```php
// Assuming the current user is an admin
$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();
}
```

- 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.

controllers > updateRole.php

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

    if (!$userId) {
        header("Location: ../home.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.

utils > updateUserRole.php

```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 = 'home.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.

```html
<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>
```

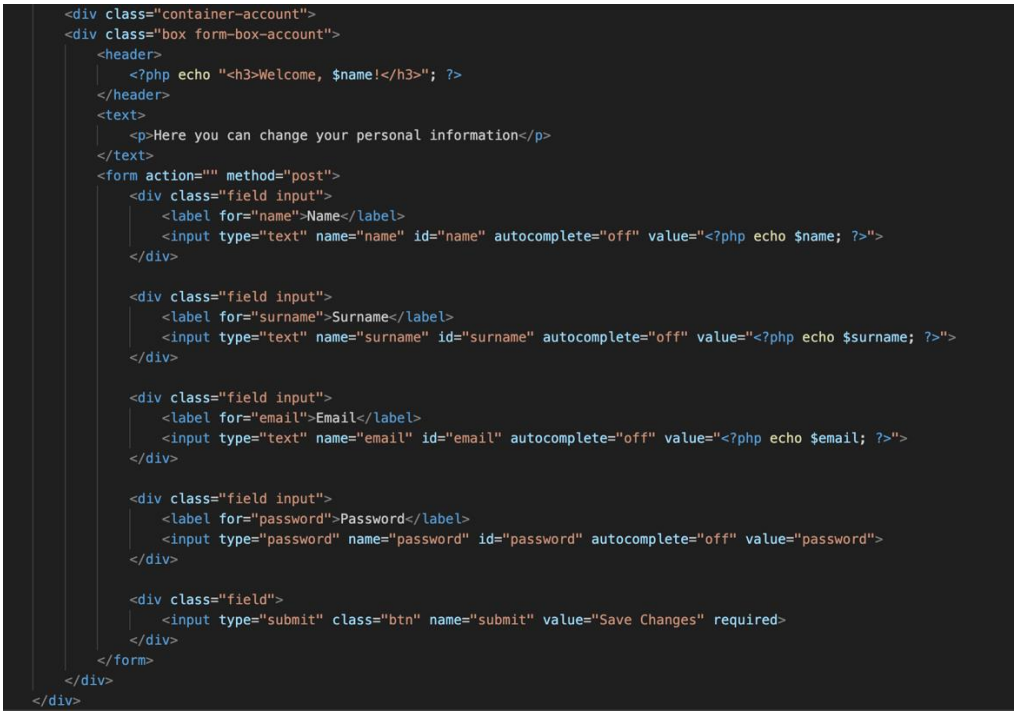
The dynamic rendering of user data is achieved through PHP, fetching and displaying user information from the database. The page provides a table layout with columns for user attributes and action buttons allowing administrators to delete users or update their roles. The user roles are presented as "Admin" 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
<?php
    include_once "utils/session.php";
    if (!isset($_SESSION['id']) || $_SESSION['is_admin'] != 1) {
        header("Location: home.php");
    }

    include_once "php/conn.php";

    $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'];

    include('partials/header.php');

    // Get all users
    $sql = "SELECT * FROM users";
    $result = mysqli_query($conn, $sql);
    $users = mysqli_fetch_all($result, MYSQLI_ASSOC);
    $full_name = $row['name'] . " " . $row['surname'];
?>
```

```php
<?php
    foreach ($users as $user) {
        echo "<tr>";
        echo "<td>{$user['id']}</td>";
        $full_name = $user['name'] . " " . $user['surname'];
        echo "<td>{$full_name}</td>";
        echo "<td>{$user['email']}</td>";
        if ($user['is_admin'] == 1) {
            echo "<td>Admin</td>";
        } else {
            echo "<td>Guest</td>";
        }
        echo "<td>";
        echo "<button class='table-btn' onclick='deleteUser({$user['id']})'>Delete</button>";
        echo "<button class='table-btn' onclick='updateUserRole({$user['id']})'>Update Role</button>";
        echo "</td>";
        echo "</tr>";
    }
?>
```

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.



```
<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>
    </form>
</div>
</div>
```

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.

```php
// 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' WHE
                $result = mysqli_query($conn, $sql);
                if ($result) {
                    echo "<script>alert('Changes saved'); window.location.href = 'home.php';</script>";
                } else {
                    echo "<script>alert('Changes failed');</script>";
                }
            }
```

```php
} 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 = '
    $result = mysqli_query($conn, $sql);
    if ($result) {
        echo "<script>alert('Changes saved'); window.location.href = 'home.php';</script>";
    } else {
        echo "<script>alert('Changes failed');</script>";
    }
}
```

5.5. Recipe Creation and Browsing

- Create recipe

This HTML file represents the "Create Recipe" page of a web application. It includes 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.

The form allows users to submit a new recipe, including details such as title, description, category, and additional information. Client-side validation is implemented to ensure that the description is at least 20 characters long, and that all fields are filled appropriately. Server-side checks are performed to avoid empty or invalid inputs, and a list of recipe categories is dynamically generated from the database.

```
<header>
    <h3>Create your recipe!</h3>
</header>
<text>
    <p>Fill the form and share your recipe with us!</p>
</text>
<form  method="POST" enctype="multipart/form-data">

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

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

    <div class="field input">
        <label for="category_name">Category</label>
        <select name="category_name" id="category_name" autocomplete="off" required>
            <option value="0">Select category</option>
            <?php
                $sql = "SELECT * FROM recipe_categories";
                $result = mysqli_query($conn, $sql);
                while ($row = mysqli_fetch_assoc($result)) {
                    echo "<option value='".$row['title']."'>".$row['title']."</option>";
                }
            ?>
        </select>
    </div>

    <div class="field input">
        <label for="additional_info">additional info.</label>
        <input type="text" name="additional_info" id="additional_info" autocomplete="off" required>
    </div>
```

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

```
// get current user
$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'];

$full_name = $name . " " . $surname;

if (isset($_POST['submit'])) {
    $title = $_POST['title'];
    $description = $_POST['description'];
    $category_name = $_POST['category_name'];
    $additional_info = $_POST['additional_info'];
    $author = $full_name;

    if (strlen($description) < 20) {
        echo "<script>alert('Description should be at least 20 characters long');</script>";

    } else {
        if (empty($title) || empty($description) || empty($additional_info) || empty($category_name)) {
            echo "<script>alert('Please fill in all fields');</script>";
        } else {
            $sql = "INSERT INTO food_recipes ( title, description, category_name, additional_info, author) VALUES ('$title', '$desc
            $result = mysqli_query($conn, $sql);
            if ($result) {
                echo "<script>alert('Recipe created'); window.location.href = 'home.php';</script>";
            } else {
                echo "<script>alert('Recipe creation failed');</script>";
            }
        }
    }
}
```

- Browse recipe

The provided PHP script functions as an API endpoint for retrieving food recipe information from a database. It begins by setting the response header to indicate JSON content. After connecting to the database, the script executes an SQL query to fetch all records from the "food_recipes" table. Error handling ensures a graceful response in case of issues. The fetched data is organized into a JSON string using `json_encode` and sent as the API response, facilitating seamless integration with client-side applications.

controllers > getRecipes.php

```php
<?php
    header('Content-Type: application/json');
    include("../php/conn.php");

    $sql = "SELECT * FROM food_recipes";
    $result = mysqli_query($conn, $sql);

    if (!$result) {
        die('Error: ' . mysqli_error($conn));
    }

    $food_recipes = mysqli_fetch_all($result, MYSQLI_ASSOC);

    echo json_encode($food_recipes);
?>
```

These recipes are displayed on the home page of the user, using this API. The following HTML file is the code for the home section of a food recipe website. It includes PHP code to check whether a user is logged in and fetches the user's name for a personalized greeting. The main content of the page features a dynamic welcome message, a search bar, and a section for displaying recipes. The layout structure is organized using div elements, and external files, such as "app.js" and partials like the header and footer, are included for better maintainability. These will be discussed in the latter section of the report.

```
<main>
    <div class="main-box top">
        <div class="top">
            <div>
                <?php echo "<h2>Welcome, $name!</h2>"; ?>

                <p>Explore our recipes and categories below:</p>
            </div>
        </div>

        <?php include('partials/searchBar.php') ?>

        <div class="bottom" id="recipes">


        </div>
    </div>
</main>

<?php include('partials/footer.php') ?>
div>
```

5.6. Navigation and Design

- Log out button: logout is a simple function that deletes a session, which means the user can't be in page's home, account, or user list(if admin).
    i. `session_start()`: Initiates a new or resumes the existing session.
    ii. `$_SESSION = array()`: Clears all session variables by assigning an empty array to `$_SESSION`. This step is taken to ensure that all session data is unset.
    iii. Cookie Management:
        a. `session_get_cookie_params()`: Retrieves the current session cookie parameters.
        b. `setcookie(...)`: Sets the session cookie with an expiration time in the past, effectively deleting the cookie.
    iv. `session_destroy()`: Destroys the current session, deleting all session data on the server.
    v. `header("Location: ../index.php")`: Redirects the user to the index.php (or login) page after the logout.
    vi. `die()`: Terminates the script execution.

```php
<?php
    session_start();

    $_SESSION = array();

    if (ini_get("session.use_cookies")) {
        $params = session_get_cookie_params();
        setcookie(
            session_name(),
            '',
            time() - 42000,
            $params["path"],
            $params["domain"],
            $params["secure"],
            $params["httponly"]
        );
    }

    session_destroy();

    header("Location: ../index.php");
    die();
?>
```

This script ensures a clean logout by destroying the session, clearing session variables, and redirecting the user to the login page. Additionally, it takes care of clearing the session cookie on the client side.

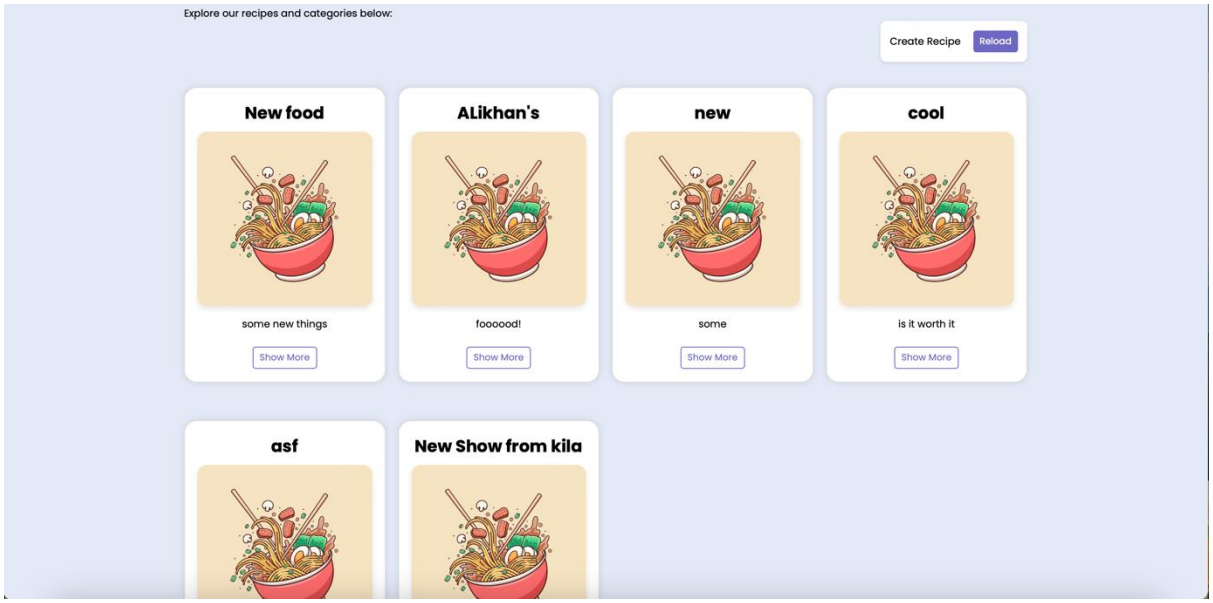
- Auth logic

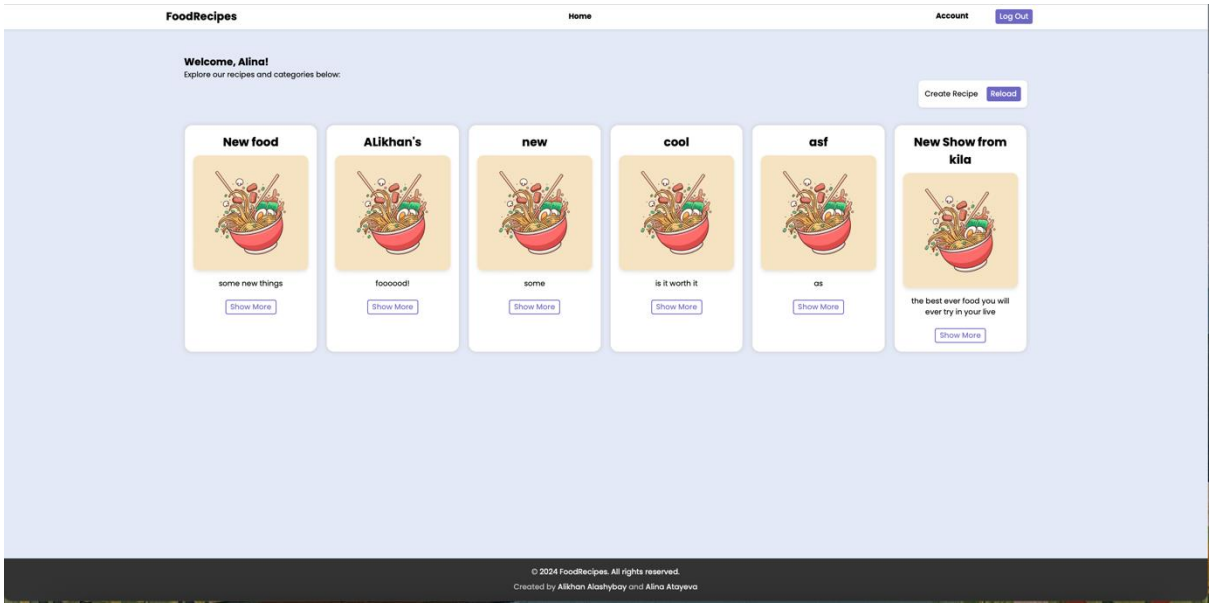
In the website's navigation system, we've implemented a security feature known as "auth logic." This ensures that only authenticated users with an active session are permitted to navigate through various URLs. If a user attempts to access pages other than registration and login without an active session, 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.

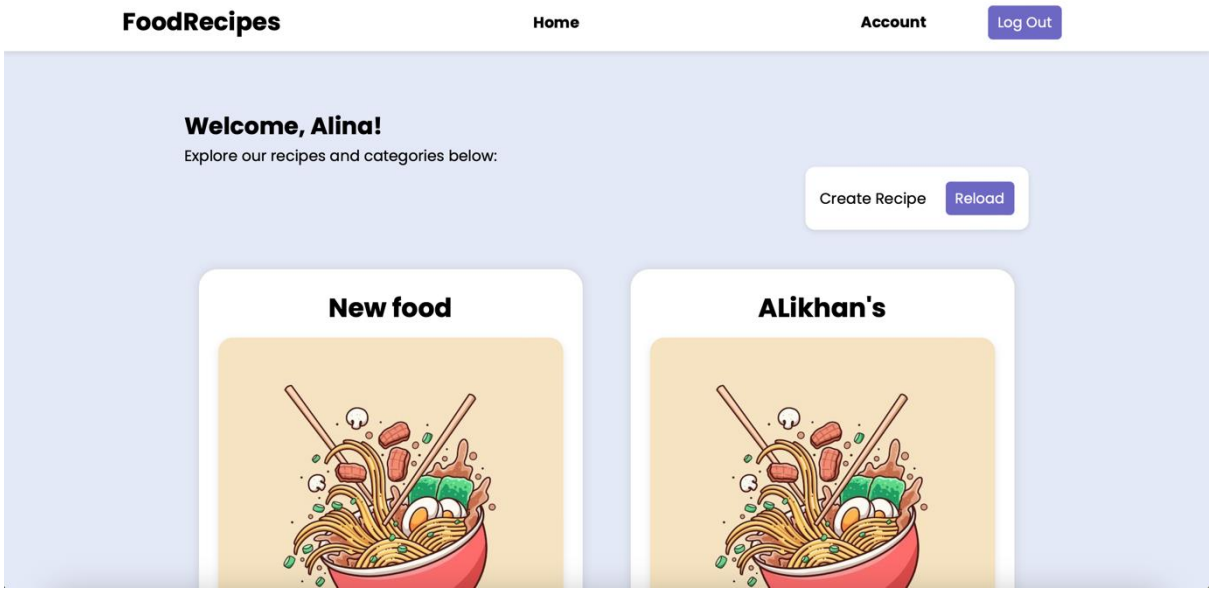
```php
<?php
    require_once("session.php");

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

- Responsiveness

Recipe cards on the home page will be shown in 4-5 columns and 1-2-3 columns depending on screen size.


Explore our recipes and categories below:
Create Recipe
Reload
New food
some new things
Show More
ALikhan's
foooood!
Show More
new
some
Show More
cool
is it worth it
Show More
asf
New Show from kila


FoodRecipes
Home
Account
Log Out
Welcome, Alina!
Explore our recipes and categories below:
Create Recipe
Reload
New food
some new things
Show More
ALikhan's
foooood!
Show More
new
some
Show More
cool
is it worth it
Show More
asf
as
Show More
New Show from kila
the best ever food you will ever try in your live
Show More
© 2024 FoodRecipes. All rights reserved.
Created by Alikhan Alashybay and Alina Atayeva


FoodRecipes
Home
Account
Log Out
Welcome, Alina!
Explore our recipes and categories below:
Create Recipe
Reload
New food
ALikhan's

This is done using CSS. In the “style.css” file:

- ‘display: grid;’: This sets up a grid container.
- ‘grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));’: This line is responsible for the responsive column layout. It uses the ‘auto-fill’ keyword to create as many columns as can fit in the available space, and ‘minmax(300px, 1fr)’ ensures that each column will be at least 300 pixels wide but can grow to occupy equal fractions of the available space ‘(1fr)’.
- ‘gap: 20px;’: This sets the gap between grid items to 20 pixels.
- ‘justify-content: center;’: This centers the columns horizontally.

With this setup, on larger screens, you should see more columns, and on smaller screens, the columns will adjust to maintain a minimum width of 300 pixels. The ‘auto-fill’ and ‘minmax’ combination is a common technique for creating responsive grid layouts.

```css
/* ---------------------------------------- */
/* RECIPE CARD */
#recipes {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
  gap: 20px;
  justify-content: center;

}
.recipe-card {
  border: 1px solid #ddd;
  margin: 20px auto;
  padding: 20px;
  background-color: #fff;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
  text-align: center;
  border-radius: 20px;
  max-width: 400px;
}

.recipe-header {
  font-size: 18px;
  margin-bottom: 10px;
  display: flex;
  justify-content: center;
  align-items: center;
}
```

- Header

The "header.php" file serves as a crucial component in our project, defining the top section of each webpage for a consistent user experience. It opens with our site's distinctive logo, a clickable link leading users back to the main page, labeled "FoodRecipes." The navigation bar is strategically organized, featuring links like "Home" to allow users easy access to essential sections. What makes it dynamic is the conditional inclusion of the "Users List"

link, which only appears if the logged-in user is identified as an administrator, as illustrated in the code snippet:

```php
<?php
    if ($_SESSION['is_admin'] == 1) {
        echo "<a href='usersList.php'>Users List</a>";
    }
?>
```

This PHP logic ensures that only administrators see and can navigate to the "Users List" page. To the right of the navigation, we have quick-access links for managing the user's account and facilitating a smooth logout process. The overall design and functionality of this header component promote a cohesive look and feel across our web pages, contributing to a user-friendly and efficient browsing experience.

```php
<div class="nav">
    <div class="logo">
        <a href="home.php" id="logo">FoodRecipes</a>
    </div>
    <nav class="nav-links">
        <a href="home.php">Home</a>

        <?php
            if ($_SESSION['is_admin'] == 1) {
                echo "<a href='usersList.php'>Users List</a>";
            }
        ?>

    </nav>
    <div class="right-links">
        <a href="account.php">Account</a>
        <a href="utils/logout.php"><button class="btn">Log Out</button></a>
    </div>
</div>
```

- Footer

The "footer.php" file is a key element in maintaining a polished and informative closure to each webpage within our web development project. It encapsulates essential details such as the copyright information dynamically generated to display the current year using PHP's date function:

```php
<footer>
    <div class="footer-content">
        <p>&copy; <?php echo date("Y"); ?> FoodRecipes. All rights reserved.</p>
```

This ensures that the copyright year is always up-to-date. Beneath this, we've incorporated a credits section acknowledging the creators of the website, providing direct links to their GitHub profiles. For instance:

```
        <div class="credits">
          Created by<a href="https://github.com/Alashybay" target="_blank" rel="noreferrer noopener"class="link"> Alikhan Alashybay</a>
          and <a href="https://github.com/alinatayeva" target="_blank" rel="noreferrer noopener"  class="link">Alina Atayeva</a>
        </div>
    </div>
</footer>
```

This establishes transparency and recognition for the individuals behind the project. The footer's overall design aligns with the website's aesthetics and branding, offering users a consistent and visually appealing conclusion to their browsing experience.

- App.js

The "app.js" file serves as a pivotal component within our web application, contributing to the enhancement of the user experience through various client-side functionalities. The script begins by defining functions for user-related actions, such as deletion and role updates. These functions employ the Fetch API to communicate with the server, prompting users for confirmation before executing actions and subsequently refreshing the page to reflect the changes.

Moreover, the script includes a function named `drawCards`, responsible for dynamically fetching recipe data from the server and creating HTML elements to represent each recipe. It ensures a seamless integration of recipe cards into the user interface, featuring titles, images, descriptions, and additional information. The ability to toggle extra content visibility is facilitated through the `toggleMoreContent` function, enhancing user interaction with the recipe cards. Overall, "app.js" is instrumental in managing the client-side dynamics of our application, providing essential features that contribute to a more responsive and engaging user interface.

```
window.addEventListener( type: 'load',  listener: drawCards);

function toggleMoreContent(btn) {
    var additionalContent = btn.parentNode.querySelector('.additional-content');

    if (additionalContent.style.display === 'none' || additionalContent.style.display === '') {
        additionalContent.style.display = 'block';
        btn.innerText = 'Hide';
    } else {
        additionalContent.style.display = 'none';
        btn.innerText = 'Show More';
    }
}
```

```
JS app.js > drawCards > then() callback
50  function reloadPage() {
51      window.location.reload();
52    }
53
54  function drawCards() {
55      const element = document.getElementById( elementId: 'recipes');
56
57
58      fetch( input: 'controllers/getRecipes.php')
59          .then( onfulfilled: response => response.json())
60          .then( onfulfilled: data => {
61              data.forEach(recipe => {
62                  const recipeContainer = document.createElement( tagName: 'div');
63                  recipeContainer.classList.add( tokens[0]: 'recipe-card');
64                  recipeContainer.id = 'card';
65
66                  const template = `
67                      <div class="recipe-header">
68                          <h2>${recipe.title}</h2>
69                      </div>
70                      <div class="recipe-content">
71                          <img class="recipe-image" src="public/images/icon.jpeg" alt="Image was damaged">
72                          <p class="recipe-description">${recipe.description}</p>
73                          <button class="btn show-more-btn" onclick="toggleMoreContent(this)">Show More</button>
74                          <div class="additional-content" style="display: none;">
75                              <p><strong>Category:</strong> ${recipe.category_name}</p>
76                              <p><strong>Additional Info:</strong> ${recipe.additional_info}</p>
77                              <p><strong>Author:</strong> ${recipe.author}</p>
78                          </div>
79                      </div>
80                  `;
81
82                  recipeContainer.innerHTML = template;
83
84                  element.appendChild( node: recipeContainer);
85              });
```

6. Folders Structures

In our Web Programming class project, the project structure is organized into specific folders, each serving a distinct purpose:

- API/controllers:

- This folder houses endpoints or actions responsible for handling various operations such as fetching, updating, and deleting specific columns in the database. These controllers act as the bridge between the front-end and the database, ensuring smooth data transactions.

- App.js:

- This JavaScript file serves as a central hub for calling functions across the application. It orchestrates the execution of various functionalities, ensuring a cohesive and organized codebase.

- Partials:

- The Partials folder contains reusable components or partials. These components are static in nature but are frequently used throughout the application. They serve as building blocks, promoting code reusability and maintaining consistency in the user interface.

- PHP:

- The PHP folder, aptly named for its purpose, handles the connection to the database. It encapsulates functionalities related to database operations, and it can also be thought of as the "db" module within the project structure.

This well-organized folder structure enhances code maintainability, separation of concerns, and overall project scalability. The use of distinct folders for specific functionalities streamlines development and contributes to a modular and efficient web application architecture.