Members:

Julian Zerafa (to submit)

Ryan Mizzi

Christian Mercieca

Documentation for Part 2

This is the official documentation for our final solution of Part 2. The document is split into 3 parts, each part was written by a different member.

** Hierarchical Structure will be available on GitHub as well as on the submission on VLE

Part 2:

Documentation Julian Zerafa:

Documentation for Favorites Functionality

Overview

This section covers two scripts used for managing and displaying a user's favorite dishes on the website: favorites.php (server-side logic) and favorites.html (front-end structure).

favorites.php

Purpose:

Manages user's favorite dishes by:

- Displaying favorite dishes.
- Removing dishes from favorites.
- Sending the list of favorite dishes via email.

1. **Initial Setup:**

- Includes necessary files: config.php, send_mail.php, header.html.
- Starts the session and initializes the favorites array in the session if it doesn't exist.

```php
<?php
include 'config.php';
include 'send_mail.php'; // Include the send_mail function
include 'templates/header.html';

// Initialize the session if not already started
if (session_status() === PHP_SESSION_NONE) {
    session_start();
}

// Initialize the favorites array if it doesn't exist
if (!isset($_SESSION['favorites'])) {
    $_SESSION['favorites'] = [];
}
```

2. Fetching the favourite dish using the session

```
$favorites = $_SESSION['favorites'];
```

3. Removing favourite dishes using POST requests

```
if ($_SERVER['REQUEST_METHOD'] == 'POST' &&
isset($_POST['remove_from_favorites'])) {
    $remove_id = intval($_POST['remove_id']);
    if (($key = array_search($remove_id, $favorites)) !== false) {
        unset($favorites[$key]);
        $_SESSION['favorites'] = $favorites;
    }
}
```

4. Prepares and executes an SQL query to fetch the favourite dishes from the database `restaurant`

```
if (!empty($favorites)) {
    // Fetch details of each favorite dish from the database
    $placeholders = implode(',', array_fill(0, count($favorites), '?'));
    $types = str_repeat('i', count($favorites));
    $stmt = $mysqli->prepare("SELECT dishes.id, dishes.name,
dishes.description, dishes.price, dishes.image, categories.name as
category_name
                              FROM dishes
                              JOIN categories ON dishes.category_id =
categories.id
                             WHERE dishes.id IN ($placeholders)");
    $stmt->bind_param($types, ...$favorites);
    $stmt->execute();
    $result = $stmt->get_result();
    $favoriteDishes = $result->fetch_all(MYSQLI_ASSOC);
    $stmt->close();
}
```

5. Option to send the list of favourite dishes to an email.

```
if ($_SERVER['REQUEST_METHOD'] == 'POST' && isset($_POST['send_email'])) {
    $to = $_POST['email'];
    $subject = "Your Favorite Dishes";

    // Prepare the email content with HTML formatting without images
    $message = "<h1>Your Favorite Dishes</h1><div style='font-family: Arial,
sans-serif;'>";
    foreach ($favoriteDishes as $dish) {
        $message .= "<div style='margin-bottom: 20px; border-bottom: 1px solid
#ccc; padding-bottom: 10px;'>
                        <h2 style='color: #333;'>" .
htmlspecialchars($dish['name']) . "</h2>
                        <p><strong>ID:</strong> " .
htmlspecialchars($dish['id']) . "</p>
                        <p><strong>Price:</strong> €" .
number_format((float)$dish['price'], 2, '.', '') . "</p>
                        <p><strong>Description:</strong> " .
htmlspecialchars($dish['description']) . "</p>
                        <p><strong>Category:</strong> " .
htmlspecialchars($dish['category_name']) . "</p>
                    </div>";
    }
    $message .= "</div>";

    // Send the email using the send_mail function
    send_mail($to, $subject, $message);
}
```

6. Including the design and footer

```
include 'templates/favorites.html';
include 'footer.php';
```

Conclusion:

`favorites.php` handles the functionality that allows the user to add/remove favourite dishes and even send the list by email. `favorites.html` is the used to add a more appealing UI to the code allowing for a more organized and user-friendly experience.

Documentation for Dish Functionality

Overview

This section covers two scripts used for displaying and managing details of a specific dish on the website: dish.php (server-side logic) and dish.html (front-end structure).

dish.php

Purpose:

Manages the details of a specific dish by:

- Fetching and displaying dish details.
- Adding or removing the dish from the user's favorites.

1. Initial Setup:
   - Includes necessary files: config.php, header.html.
   - Starts the session and initializes the favorites array in the session if it doesn't exist.

```php
include 'config.php';
include 'templates/header.html';

// Initialize the session if not already started
if (session_status() === PHP_SESSION_NONE) {
    session_start();
}

// Initialize the favorites array if it doesn't exist
if (!isset($_SESSION['favorites'])) {
    $_SESSION['favorites'] = [];
}
```

2. Sanitizing and Validating Dish ID using GET request:

```php
$dish_id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT);

if (!$dish_id) {
    echo "<p>Invalid dish ID.</p>";
    exit;
}
```

3. Fetching Dish Details from the Database:
    - Prepares and executes a query to fetch details of the dish.

```
$query = "SELECT dishes.id, dishes.name, dishes.price, dishes.image,
dishes.description, categories.name as category_name
          FROM dishes
          JOIN categories ON dishes.category_id = categories.id
          WHERE dishes.id = ?";
$stmt = $mysqli->prepare($query);
$stmt->bind_param("i", $dish_id);
$stmt->execute();
$result = $stmt->get_result();
$dish = $result->fetch_assoc();

if (!$dish) {
    echo "<p>Dish not found.</p>";
    exit;
}
```

4. Handling Favourites:
    - Checks if the dish is in the favorites list.
    - Handles adding or removing the dish from the favorites list based on the POST request.

```
$isFavorite = in_array($dish_id, $_SESSION['favorites']);

// Handle adding/removing from favorites
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    if (isset($_POST['add_to_favorites'])) {
        if (!in_array($dish_id, $_SESSION['favorites'])) {
            $_SESSION['favorites'][] = $dish_id;
        }
    } elseif (isset($_POST['remove_from_favorites'])) {
        if (($key = array_search($dish_id, $_SESSION['favorites'])) !== false)
{
            unset($_SESSION['favorites'][$key]);
        }
    }
    header("Location: dish.php?id=$dish_id");
    exit;
}
```

Conclusion

dish.php and dish.html work together to provide functioning logic to any specific dish logic. It handles fetching the dish details, managing the favourites list and dish.html provides a user-friendly UI throughout.

Documentation for Login and Logout Functionality

Overview

This section covers three scripts used for managing user authentication on a website: login.php (login logic), logout.php (logout logic), and login.html (login form layout). It allows the admin to access the necessary resources to update the website from the UI directly, making it a very dynamic solution.

login.php

Purpose:

Handles user login by verifying credentials against the database and starting a session for authenticated users (admin)

1. Initial Setup
    - Starts a session and includes necessary files: config.php and header.html

```
session_start();
include 'config.php';
include 'templates/header.html';

$error = '';
```

2. Handling Login Form Submission
    - Checks if the request method is POST and retrieves the username and password from the form submission.

```
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $username = $_POST['username'];
    $password = $_POST['password'];

    // Query the database to retrieve user credentials
    $stmt = $mysqli->prepare("SELECT username, password FROM users WHERE
username = ?");
    $stmt->bind_param("s", $username);
    $stmt->execute();
    $result = $stmt->get_result();
    $user = $result->fetch_assoc();

    if ($user && password_verify($password, $user['password'])) {
        $_SESSION['username'] = $user['username'];
```

```
        $_SESSION['role'] = 'admin'; // Assuming all users are admins for
simplicity
        header('Location: admin.php');
        exit();
    } else {
        $error = 'Incorrect username or password';
    }
}
```

- The username and password have not been hardcoded in the actual script for security reasons.
- They are stored in a table called `user` and the password has also been hashed for further security.

logout.php

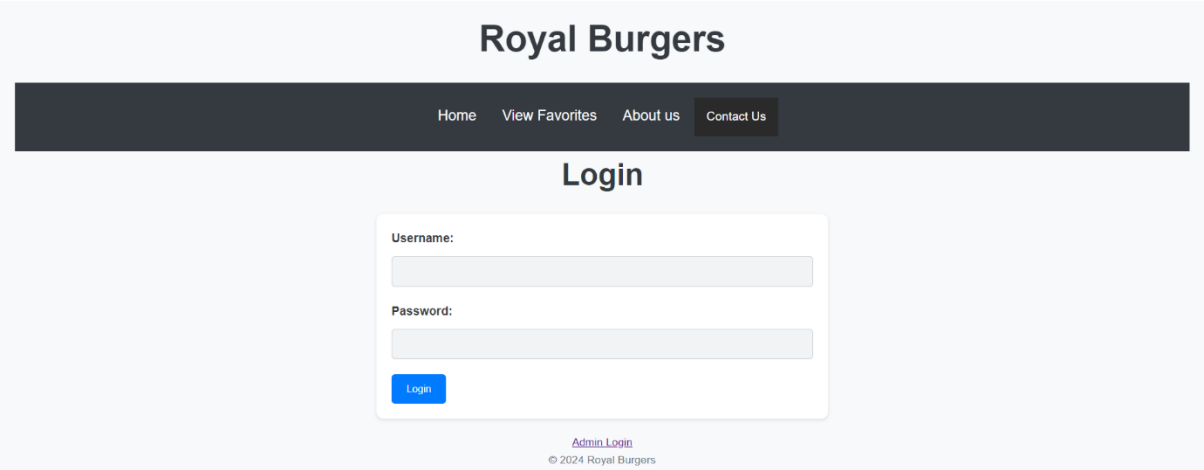
Purpose:

Destroys the session and redirects the user to the login page.

```
<?php
session_start();
session_destroy();
header('Location: login.php');
exit();
?>
```

Conclusion

The `login.php` and `login.html` serve as a way for the admin to gain control to features to edit in the website. `login.html` contains validation such as warning of an incorrect login attempt. This is the `login.php` script when served:



Documentation for Homepage and Menu Functionality

Overview

This section covers the scripts used for displaying the homepage of a restaurant's website: index.php (server-side logic) and index.html (front-end structure).

Purpose:

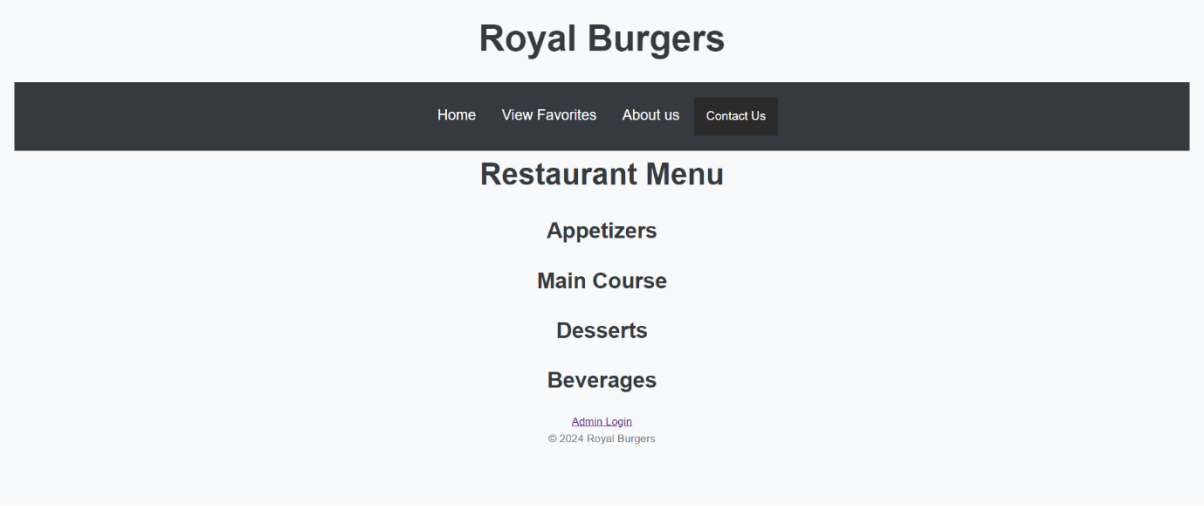
Manages the homepage by fetching the 4 Main Categories (Appetizers, Main Course, Dessert, and Beverages).

Loading the menu:

```
$menu = loadMenu($mysqli);
```

Conclusion:

The `index.php` and `index.html` scripts provide a dynamic and user-friendly UI for the restaurant homepage. The `index.php` script should be the runnable class. This is how `index.php` looks when served.



Documentation Ryan Mizzi:

add_category.php

This PHP script handles adding a new category to the database.

Includes:

- config.php: Database connection.
- header.html: HTML header template.

Session Management:

- Starts a session.
- Checks if the user is logged in. If not, redirects to login.php.

```php
// Check if the user is logged in
if (!isset($_SESSION['username'])) {
    header('Location: login.php');
    exit();
}
```

Error Handling:

- Initializes an $error variable to store any error messages.

Form Submission Handling:

- Checks if the request method is POST.
- Retrieves the category name from the form submission.
- Inserts the new category into the database using a prepared statement.
- On success, redirects to admin.php. On failure, sets an error message.

```php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $category_name = $_POST['category_name'];

    // Insert new category into the database
    $stmt = $mysqli->prepare("INSERT INTO categories (name) VALUES (?)");
    $stmt->bind_param("s", $category_name);
    if ($stmt->execute()) {
        // Category added successfully, redirect to admin panel
        header('Location: admin.php');
        exit();
    } else {
        // Error occurred while adding category
        $error = 'Error: Unable to add category.';
    }
}
```

Includes:

- add_category.html: Form template for adding a new category.
- footer.php: HTML footer template.

add_category.html

This HTML template provides a form to add a new category.

Form:

- A form with method POST that submits to add_category.php.
- An input field for the category name.
- A submit button to add the category.

```
<form method="post" action="add_category.php">
    <label for="category_name">Category Name:</label>
    <input type="text" id="category_name" name="category_name" required>
    <br>
    <input type="submit" value="Add Category">
</form>
```

Error Display:

- Displays any error messages if they exist.

category.php

This PHP script fetches and displays dishes for a specific category.

Includes:

- config.php: Establishes the database connection.
- header.html: Header template.

Input Sanitization:

- Sanitizes the category ID retrieved from the URL using filter_input.
- If the category ID is invalid, it displays an error message and exits.

```php
$category_id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT);

if (!$category_id) {
    echo "<p>Invalid category ID.</p>";
    exit;
}
```

Fetching Category Details:

- Prepares and executes a SQL query to retrieve the name of the category using a prepared statement.
- If the category is not found, it displays an error message and exits.

```php
$query = "SELECT name FROM categories WHERE id = ?";
$stmt = $mysqli->prepare($query);
$stmt->bind_param("i", $category_id);
$stmt->execute();
$result = $stmt->get_result();
$category = $result->fetch_assoc();

if (!$category) {
    echo "<p>Category not found.</p>";
    exit;
}
```

Fetching Dishes:

- Prepares and executes a SQL query to retrieve dishes belonging to the specified category.
- Fetches all dishes into an associative array.

```php
$query = "SELECT id, name, price, image, description FROM dishes WHERE category_id = ?";
$stmt = $mysqli->prepare($query);
$stmt->bind_param("i", $category_id);
$stmt->execute();
$result = $stmt->get_result();
$dishes = $result->fetch_all(MYSQLI_ASSOC);
```

Includes:

- category.html: HTML template to display the category name and dishes.
- footer.php: Footer template.

category.html

This HTML template displays dishes for a specific category.

Displaying Category Name:

- Sets the title of the page to the name of the category.

CSS Styling:

- Defines styles for the menu layout and individual dishes, including responsiveness for different screen sizes.

Displaying Dishes:

- Iterates through the list of dishes.
- Displays each dish's name as a link to view its details.
- If an image is available, displays it along with the dish name.
- Displays the price of each dish.

```
<?php foreach ($dishes as $dish): ?>
    <div class="dish">
        <h3><a href="dish.php?id=<?php echo htmlspecialchars($dish['id']); ?>"><?php echo htmlspecialchars($dish['name']); ?></a></h3>
        <?php if (!empty($dish['image'])): ?>
            <img src="<?php echo htmlspecialchars($dish['image']); ?>" alt="<?php echo htmlspecialchars($dish['name']); ?>">
        <?php else: ?>
            <p>No image available</p>
        <?php endif; ?>
        <p>Price: €<?php echo number_format((float)$dish['price'], 2, '.', ''); ?></p>
    </div>
<?php endforeach; ?>
```

delete_category.php

This PHP script handles the deletion of a category from the database.

Includes:

- config.php: Establishes the database connection.
- header.html: Header template.

Session Check:

- Starts a session to check if the user is logged in.
- If not logged in, redirects to the login page.

```
if (!isset($_SESSION['username'])) {
    header('Location: login.php');
    exit();
}
```

Fetching Categories:

- Prepares and executes a SQL query to fetch all categories from the database.
- Retrieves category IDs and names into an associative array.

```php
$stmt = $mysqli->prepare("SELECT id, name FROM categories");
$stmt->execute();
$result = $stmt->get_result();
$categories = $result->fetch_all(MYSQLI_ASSOC);
```

Category Deletion:

- Checks if the form is submitted and if the category ID is set.
- If so, retrieves the category ID from the POST data.
- Prepares and executes a SQL query to delete the category from the database.
- If deletion is successful, redirects to the admin panel.
- If an error occurs during deletion, it displays an error message.

```php
$stmt = $mysqli->prepare("DELETE FROM categories WHERE id = ?");
$stmt->bind_param("i", $category_id);
```

Includes:

- delete_category.html: HTML template for displaying the delete category form.
- footer.php: Footer template.

update_order.php

This PHP script handles the updating of display orders for categories in the database.

Includes:

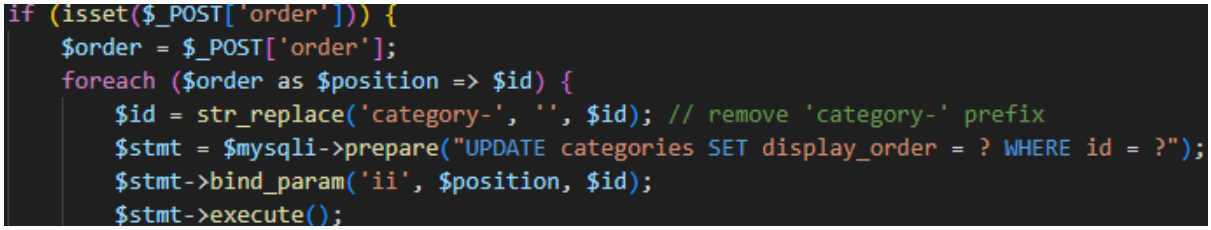
- config.php: Establishes the database connection.

Session Check:

- Starts a session to check if the user is logged in with admin privileges.
- If not logged in as an admin, redirects to the login page.

Update Order:

- Checks if the POST data named 'order' is set.
- Iterates through the received order data, which contains category IDs prefixed with 'category-' and their new positions.
- Removes the 'category-' prefix from each category ID.
- Prepares and executes a SQL query to update the display order of each category in the database based on the received position.
- Binds the position and category ID parameters to the prepared statement.



```php
if (isset($_POST['order'])) {
    $order = $_POST['order'];
    foreach ($order as $position => $id) {
        $id = str_replace('category-', '', $id); // remove 'category-' prefix
        $stmt = $mysqli->prepare("UPDATE categories SET display_order = ? WHERE id = ?");
        $stmt->bind_param('ii', $position, $id);
        $stmt->execute();
```

This documentation outlines the functionality of update_order.php, explaining how it updates the display order of categories in the database based on the received order data.

add_dish.php

This PHP script handles adding a new dish to the database.

Includes:

- config.php: Database connection.
- header.html: HTML header template.

Form Submission Handling:

- Checks if the request method is POST.
- Retrieves the dish details (name, description, price, category ID) and handles image upload.
- Moves the uploaded image to the uploads directory.
- Inserts the new dish into the database using a prepared statement.
- On success, redirects to admin.php.

```php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $name = $_POST['name'];
    $description = $_POST['description'];
    $price = $_POST['price'];
    $categoryId = $_POST['category_id'];

    // Handle the image upload
    $image = $_FILES['image'];
    $imagePath = 'uploads/' . basename($image['name']);
    move_uploaded_file($image['tmp_name'], $imagePath);

    $stmt = $mysqli->prepare("INSERT INTO dishes (name, description, price, category_id, image) VALUES (?, ?, ?, ?, ?)");
    $stmt->bind_param("ssdss", $name, $description, $price, $categoryId, $imagePath);
    $stmt->execute();
    $stmt->close();
```

Fetch Categories:

- Queries the database for all categories to populate the dropdown in the form.

```php
$categories = $mysqli->query("SELECT * FROM categories")->fetch_all(MYSQLI_ASSOC);
```

Includes:

- add_dish.html: Form template for adding a new dish.
- footer.php: HTML footer template.

add_dish.html

This HTML template provides a form to add a new dish.

Form:

- A form with method POST and enctype multipart/form-data that submits to add_dish.php.
- Input fields for dish name, description, price, and image.
- A dropdown to select the category, populated with data from the database.
- A submit button to add the dish.

edit_dish.php

This PHP script handles the editing of an existing dish in the database.

Includes:

- config.php: Database connection.
- header.html: HTML header template.

Form Submission Handling:

- Checks if the request method is POST.
- Retrieves the dish details (id, name, description, price, category ID) from the form submission.
- Fetches the existing image path from the database.
- Checks if a new image is uploaded and handles the image upload process.
    - Validates file type and size.
    - Moves the uploaded file to the images directory.
- Updates the dish details in the database using a prepared statement.
- On success, redirects to admin.php.

```php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $id = $_POST['id'];
    $name = $_POST['name'];
    $description = $_POST['description'];
    $price = $_POST['price'];
    $categoryId = $_POST['category_id'];

    // Fetch existing image path
    $stmt = $mysqli->prepare("SELECT image FROM dishes WHERE id = ?");
    $stmt->bind_param("i", $id);
    $stmt->execute();
    $result = $stmt->get_result();
    $dish = $result->fetch_assoc();
    $stmt->close();
```

Fetch Dish and Categories:

- Fetches the dish details and categories from the database to populate the form fields.

Includes:

- edit_dish.html: Form template for editing a dish.
- footer.php: HTML footer template.

edit_dish.html

This HTML template provides a form to edit an existing dish.

Form:

- A form with method POST and enctype multipart/form-data that submits to edit_dish.php.
- Hidden input field for the dish ID.
- Input fields for dish name, description, price, and image.
- Displays the current image if it exists.
- A dropdown to select the category, populated with data from the database and pre-selects the current category.
- A submit button to save the changes.

delete_dish.php

This PHP script handles the deletion of a dish from the database.

Includes:

- config.php: Contains the database connection details.

Session Handling:

- Checks if a session is already started and starts it if not.
- Verifies if the user is logged in by checking the session variable username.
    - If not logged in, redirects to login.php and exits.

Dish Deletion:

- Checks if the id parameter is set in the GET request.
    - If set, prepares a SQL statement to delete the dish with the specified ID.
    - Executes the statement to delete the dish from the database.
    - Closes the prepared statement.

```php
if (isset($_GET['id'])) {
    $stmt = $mysqli->prepare("DELETE FROM dishes WHERE id = ?");
    $stmt->bind_param("i", $_GET['id']);
    $stmt->execute();
    $stmt->close();
}
```

Redirection:

- Redirects the user to admin.php after the deletion process and exits the script.

This script ensures that only logged-in users can delete dishes and handles the deletion process securely using prepared statements.

admin.php

This PHP script represents the admin dashboard, where authorized users with admin privileges can manage the menu items.

Includes:

- config.php: Establishes the database connection.
- includes/functions.php: Includes necessary functions for menu management.

Session Check:

- Starts a session to check if the user is logged in and has admin privileges.
- If not logged in as an admin, redirects to the login page.

Load Menu:

- Utilizes the loadMenu function from includes/functions.php to retrieve the menu items from the database.

```
// Load the menu
$menu = loadMenu($mysqli);
```

HTML Template:

- Includes the HTML template for displaying the admin dashboard.
- Utilizes jQuery UI sortable feature to allow drag-and-drop reordering of menu categories.
- Sends the updated order to update_order.php via AJAX post request.

- Iterates through each category and its associated dishes to display them.
- Provides edit and delete links for each dish, visible only to admins.

admin.html

This HTML template is responsible for displaying the admin dashboard.

Title:

- Sets the title of the admin dashboard as "Admin Dashboard".

Stylesheet and Scripts:

- Links to the external stylesheet styles.css.
- Includes jQuery and jQuery UI libraries for drag-and-drop functionality.
- Implements JavaScript code to make the menu items sortable and update their order dynamically.

Admin Dashboard:

- Displays the heading "Admin Dashboard".
- Renders the menu categories and their associated dishes in a sortable list.
- Provides edit and delete links for each dish, visible only to admins.

This documentation outlines the functionality of the admin.php script and its associated admin.html template, explaining how they enable admin users to manage menu items via a user-friendly dashboard.

header.html

This HTML file represents the header section of the Royal Burgers website.

Document Type Declaration:

- Specifies the document type as HTML5.

Stylesheet:

- Links to an external CSS file named styles.css to apply styles to the website.

HTML Structure:

- Defines the document's language as English (<html lang="en">).
- Sets the charset to UTF-8 for proper character encoding.
- Sets the viewport for responsive design.

Title and Logo:

- Sets the title of the webpage as "Royal Burgers".
- Displays the logo or brand name as an <h1> heading.

Navigation Menu:

- Provides navigation links to different pages of the website, including Home, Favorites, and About Us.
- Includes a dropdown menu for contacting the restaurant, with options to book a table, send a query, or file a complaint.
- If the current page is the admin panel (admin.php), it displays an additional dropdown menu with admin-specific options, such as adding a new dish, managing categories, handling queries, complaints, bookings, and logging out.

footer.php

This PHP file represents the footer section of the Royal Burgers website.

Session Check:

- Checks if a session is active, and if the user is logged in as an admin.

Footer Content:

- If the user is not logged in, it provides a link to the admin login page.
- If the user is logged in, it offers a link to the admin panel.
- Displays the copyright information, indicating "© 2024 Royal Burgers".

```php
<?php if (!isset($_SESSION['username'])): ?>
    <a href="login.php">Admin Login</a>
<?php else: ?>
    <a href="admin.php">Admin Panel</a>
<?php endif; ?>
```

functions.php

This PHP file contains a function called loadMenu($mysqli) that retrieves menu data from the database.

Function:

- **Name:** loadMenu
- **Parameters:** $mysqli (MySQLi object) - The MySQLi database connection object.
- **Return Type:** Array - An array containing menu categories and their respective dishes.

Function Description:

1. Initializes an empty array called $categories to store menu categories.
2. Executes a SQL query to select category IDs and names from the database, ordered by display order and ID.
3. Iterates over the query results and appends each category to the $categories array.
4. For each category, executes another SQL query to select dish IDs, names, and prices associated with that category.
5. Appends each dish to the 'dishes' sub-array of its respective category in the $categories array.
6. Returns the populated $categories array containing menu data.

```php
while ($row = $result->fetch_assoc()) {
    $categories[] = [
        'id' => $row['id'],
        'name' => $row['name'],
        'dishes' => []
    ];
}

foreach ($categories as &$category) {
    $query = "SELECT id, name, price FROM dishes WHERE category_id = " . $category['id'];
    $result = $mysqli->query($query);
    while ($row = $result->fetch_assoc()) {
        $category['dishes'][] = [
            'id' => $row['id'],
            'name' => $row['name'],
            'price' => $row['price']
        ];
    }
}

return $categories;
```

The `functions.php` file contains a function to load menu data.

config.php

This PHP file defines configuration constants and establishes a connection to the MySQL database.

Configuration Constants:

- **DB_HOST:** The hostname of the MySQL server.
- **DB_NAME:** The name of the MySQL database.
- **DB_USER:** The MySQL username.
- **DB_PASS:** The MySQL password.

Database Connection:

- Establishes a connection to the MySQL database using the mysqli class.
- Checks the connection status and outputs an error message if connection fails.

```php
// Create a connection
$mysqli = new mysqli(DB_HOST, DB_USER, DB_PASS, DB_NAME);
```

Include Functions File:

- Includes the functions.php file using the ROOT constant, which contains the loadMenu function.

The `config.php` file establishes the database connection and includes the necessary functions.

Documentation Christian Mercieca:

User submissions of bookings, queries and complaints:

Submitting bookings: from the home page, the user can select the option from the dropdown menu to submit a booking. This redirects the user to the booking.php, and the user is prompted to enter values for name, telephone number, email address, number of seats and the date and time of his reservation. To make sure that the date of reservation entered is not in the past, some Java Script was used in the booking.php file for client-side validation:

```
if (selectedDateTime < currentDateTime) {
        alert("Please select a future date and time.");
        return false;
    }
```

And similarly, some Java Script was used for server-side validation in submit_booking.php:

```
if ($selectedDateTime < $currentDateTime) {
  echo "Error: The selected date and time is in the past. Please choose a future date and time.";
```

To make the booking more realistic, the maximum number of seats that can be booked is 10, and the user will be given a warning if the submit button is clicked while the number of seats value is greater than 10.

When clicking the submit button, the user will be redirected to the menu, and the database will be updated with a new entry containing the submitted information.

A similar approach was given to the queries and complaints feature, where the user can pick the option to submit a query or a complaint, and is redirected to the respective page (query.php or complaint.php) to input information and submit them to the database.

Admin actions on bookings, queries and complaints:

When the admin is logged in, he/she will have the option to manage bookings, queries and complaints. The admin can confirm bookings, respond to queries and complaints, aswell as have the ability to delete them if desired. When the option to manage bookings is selected from the dropdown menu on the admin interface, the admin is redirected to the admin_confirm_booking.php page to view and manage the bookings, which are retrieved directly from the database.

```
 $sql = "SELECT id, booking_name, telephone_number, number_of_seats, date_time,
email_address, booking_confirmation FROM bookings";
```

The admin can use the fields on the right hand side of the page to input a confirm message (or a response in the case if handling queries or complaints) and after the submit button is pressed, the database is updated with this response while an email is sent to the email address that the user had inputted when submitting his/her booking/query/complaint (both done by confirm_booking.php), with the response provided by the admin:

```
$subject = "Your Booking Confirmation";

$message = "Booking Confirmation: " . $booking_confirmation;
```

The admin is also redirected to the main admin interface. If the admin decides to delete a booking, this can be done using the delete button which runs the delete_booking.php files which deletes the entry;

```
$sql = "DELETE FROM bookings WHERE id = ?";

$stmt = $mysqli->prepare($sql);

$stmt->bind_param('i', $id);
```

and sends the admin to the main admin page admin_index.php:

```
if ($stmt->execute()) {
   header("Location: admin_index.php");
   exit;
```

Similar options are available for queries and complaints, having admin_resolve_queries to display queries, and admin_resolve_complaints to handle complaints. Each of them have their respective php files to handle deleting entries (delete_query and delete_complaint). To pass details and send the email to the user, the files resolve_query.php and resolve_complaint.php along with the base setup done by send_mail.php are used.

Other files:

config.php: Used to start a session and setup a connection with the database.

send_mail.php: Uses PHPMailer in order to the send emails to the clients from the restaurant's email address. The account and SMTP settings are set up, and sets the email's subject and content. The email address is passed from the confirm_booking.php, resolve_query.php and resolve_complaint.php files.

info_page.html: This file is HTML since it is static, and only contains the information about the restaurant such as the opening hours and address, and also the employees and some details about them such as their names and positions in the restaurant.

Database structure:

The database is split up into 6 tables, each handling a specific data type example dishes. The tables that I used are the Bookings, Queries and Complaints tables. Although they contain some similar fields such as email_address and telephone_number tables, the tables contain

certain columns that allow for better keeping of data such as username in the Queries and Complaints tables, and number_of_seats in the Bookings table. Regarding the actual data within, each column has been given custom datatypes, maximum lengths and other special settings in order to better facilitate the data within it. For example, the id field which is hidden to the user, is of type integer, and is set as primary key and automatically increments itself for each row. Another example is the booking_name field being of datatype VARCHAR, having a maximum length of 255 and is required before the booking is submitted by the user:

booking_name VARCHAR(255) NOT NULL

Validations implemented:

Client-side:

- The 'required' attribute was set to the database fields such that when the user tries to submit a booking/query/complaint when a field is empty, it is not allowed.
- The database type for the email_address field was set to email to make sure a valid email format is entered.
- A limit to the maximum number of seats to book was set at a limit of 10.

Server-side:

- Input data is sanitized and validated.
- Certain criteria that are required have been set example the booking date can only be in the future.
- Parameterized queries, prepare statements and placeholders have been used to prevent SQL injections.
- The database fields 'id' for have been set to auto-increment to prevent issues with duplicate values that could happen if the id is set manually.
- PHPMailer error detection and displaying.