Darliza Sanchez

CIS 344

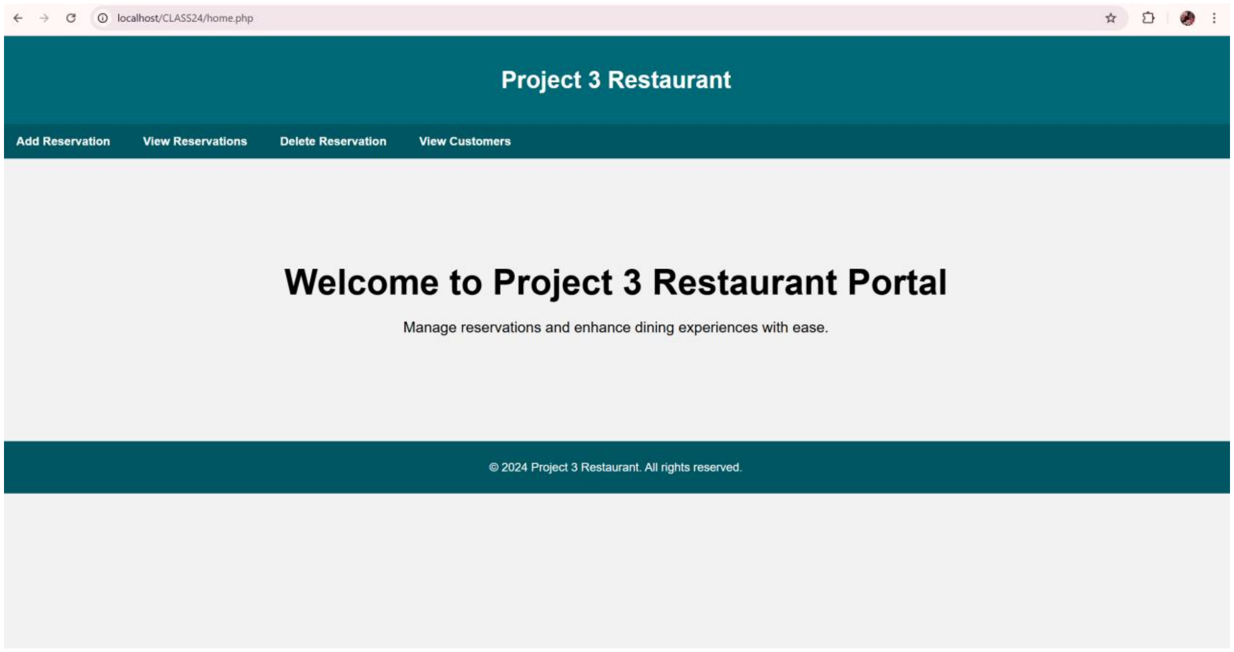
Project 3

12/10/24

Report

Project 3 Restaurant Management System is a dynamic, web-based system designed to handle reservations, dining preferences, and customer interactions in a straightforward manner. The project implements a robust PHP backend with a well-structured MySQL database and a user-friendly frontend interface to offer restaurant managers a seamless experience. In addition, the system adds and views reservations, managing customer preferences in an organized structure. The project handles real-world scenarios through stored procedures and a relational database schema, efficiently, with the integrity of data preserved.

Home.php



Welcome to Project 3 Restaurant Portal

Manage reservations and enhance dining experiences with ease.

© 2024 Project 3 Restaurant. All rights reserved.

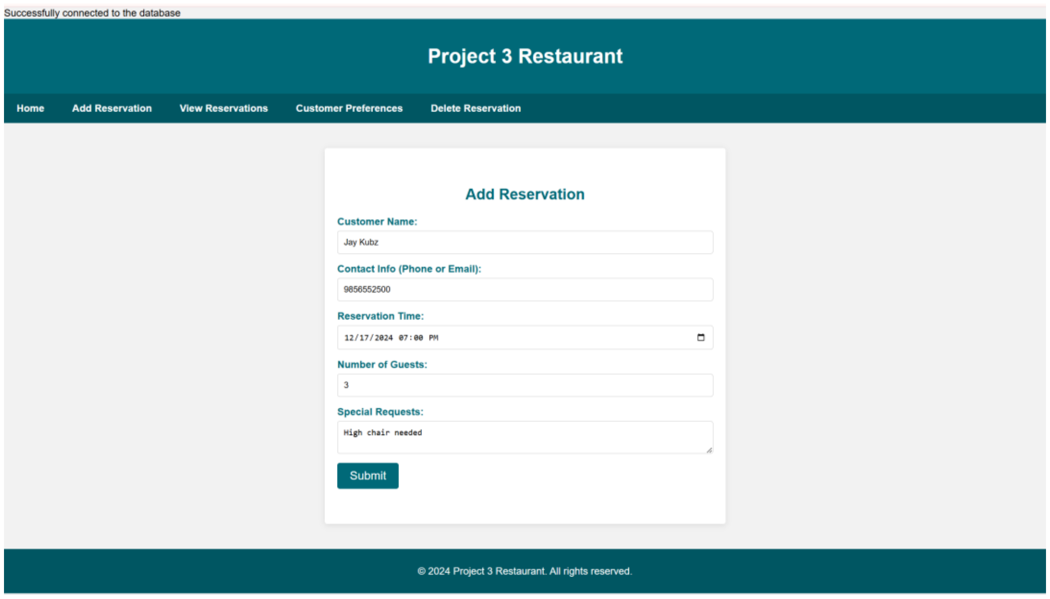
The home.php dynamically create a navigation menu and footer. A visually appealing home page with a hero section. Uses PHP to make the site easily maintainable and secure, escaping user input. Contains clear and simple CSS to create a professional design.

It first starts with some dynamic variables like $restaurantName: This variable holds the name of the restaurant ("Project 3 Restaurant"), making it easier to update the name in one place or $currentYear: Dynamically fetches the current year using the date() function. This ensures that the footer always displays the correct year without manual updates. $navItems is an associative array where the keys like "Add Reservation", "View Reservations") represent the labels displayed on the navigation menu. The values correspond to the action query parameters used in RestaurantServer.php. This array ensures the navigation menu is easily extendable. Adding a new item to the menu only requires appending another key-value pair here.

The <style> tag contains embedded CSS, which defines the visual appearance of the page. It consists of Body Styling that Sets the font and applies a soft gray background color. Header Styling that Creates a centered header with a dark teal background and white text. A Navigation Bar that Creates a horizontal menu with links and Links have a hover effect that changes the background color. Lastly a Footer Styling and header section. I did get some help from google on the designing part as I wanted it to look a bit more pleasing for users.

The navigation bar loops through the $navItems array using foreach: The $label is displayed as the link text (e.g., "Add Reservation"). The $action is encoded using urlencode() to ensure safe inclusion in the URL query string (e.g., RestaurantServer.php?action=addReservation). This dynamic approach ensures the navigation bar reflects all menu items defined in the $navItems array. There is also the hero section displays a large banner section (section.hero) with: A welcoming headline (<h2>) and a subheading (<p>) so that the home page didn't look empty.

AddReservation.php



Successfully connected to the database

Project 3 Restaurant

Add Reservation

Customer Name: Jay Kubz

Contact Info (Phone or Email): 9856552500

Reservation Time: 12/17/2024 07:00 PM

Number of Guests: 3

Special Requests: High chair needed

© 2024 Project 3 Restaurant. All rights reserved.

This page is used for adding reservations to the restaurant system in a user-friendly manner: it contains an input form for reservation details and the messages of success or failure that will appear depending on what happens during the operation.

The Navigation Bar Provides links to navigate between different parts of the website. Dynamic Action Links: Each link uses the action parameter (handled in RestaurantServer.php) to direct users to the respective functionality. This static approach works well but requires adding or removing links manually. There is also a Success or Error Message that Displays feedback to the user after submitting the form. If $message is set (e.g., "Reservation successfully created!"), it will display the message in green. Prevents users from wondering whether their action succeeded or failed.

The Reservation Form Gathers reservation details from the user:

Name (customer_name): Input field for the customer's name.

Contact Info (contact_info): Accepts a phone number or email address for contacting the customer.

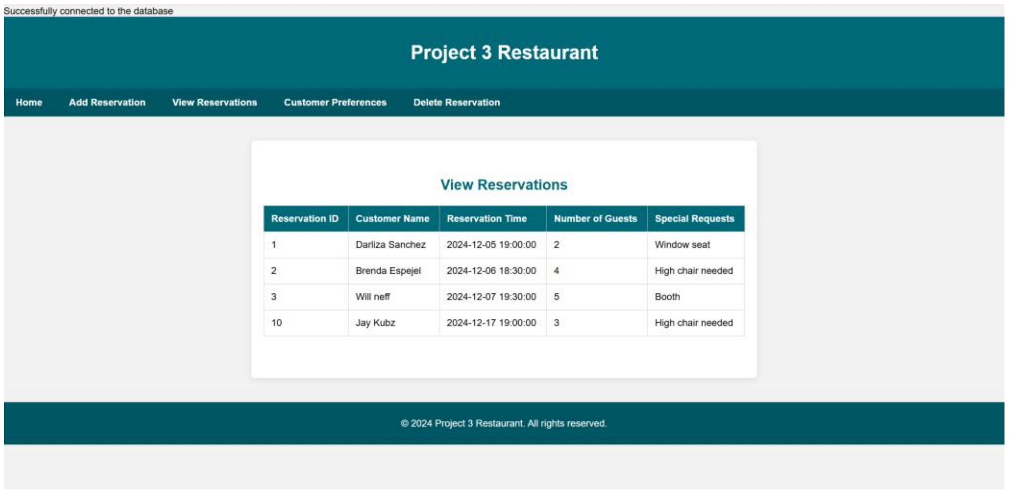
Reservation Time (reservation_time): Uses a datetime-local input type to ensure proper date and time format.

Number of Guests (number_of_guests): Specifies how many people the reservation is for.

Special Requests (special_requests): A text area field for optional notes like dietary restrictions or seating preferences.

Key Details: Validation: required attribute ensures all fields (except "Special Requests") must be filled out before submission. The browser handles basic validation like ensuring number_of_guests is numeric and reservation_time is a valid date-time format. Submission: method="POST" sends the form data securely to RestaurantServer.php for processing. action="RestaurantServer.php?action=addReservation" ensures the correct action is triggered in RestaurantServer.php. A challenge about this page is that the form depends entirely on the addReservation action in RestaurantServer.php. Any issues in the backend (e.g., database connection problems or missing logic) would make the form ineffective.

ViewReservation.php



View Reservations

Reservation ID	Customer Name	Reservation Time	Number of Guests	Special Requests
1	Darliza Sanchez	2024-12-05 19:00:00	2	Window seat
2	Brenda Espejel	2024-12-06 18:30:00	4	High chair needed
3	Will neff	2024-12-07 19:30:00	5	Booth
10	Jay Kubz	2024-12-17 19:00:00	3	High chair needed

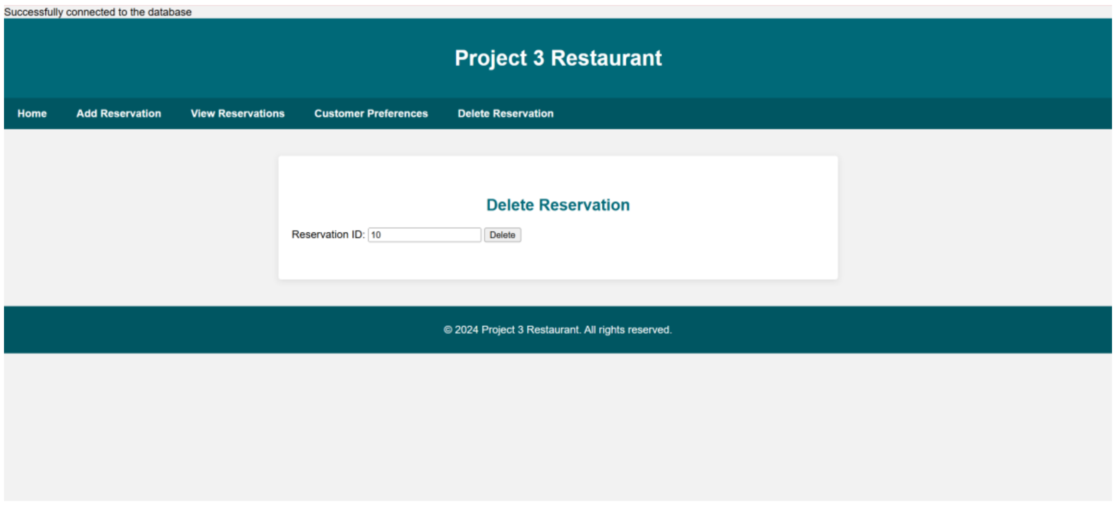
© 2024 Project 3 Restaurant. All rights reserved.

Dynamic Rows (PHP Loop) that Dynamically generates rows in the table for each reservation. How It Works: The if (isset($reservations) && !empty($reservations)) check ensures that there is reservation data to display. If reservations exist, the foreach loop iterates over the $reservations array (provided by RestaurantServer.php) and displays each reservation. Each cell (<td>)

displays a specific property of the reservation, escaped using htmlspecialchars() for security. If no reservations are found, a single row with the message "No reservations found" is displayed.

An interesting feature is that the table rows are generated dynamically based on data fetched from the backend. This approach ensures that the page always reflects up-to-date reservation data without requiring manual updates.

DeleteReservationForm.php


Successfully connected to the database

Project 3 Restaurant

Delete Reservation

Reservation ID: 10 Delete

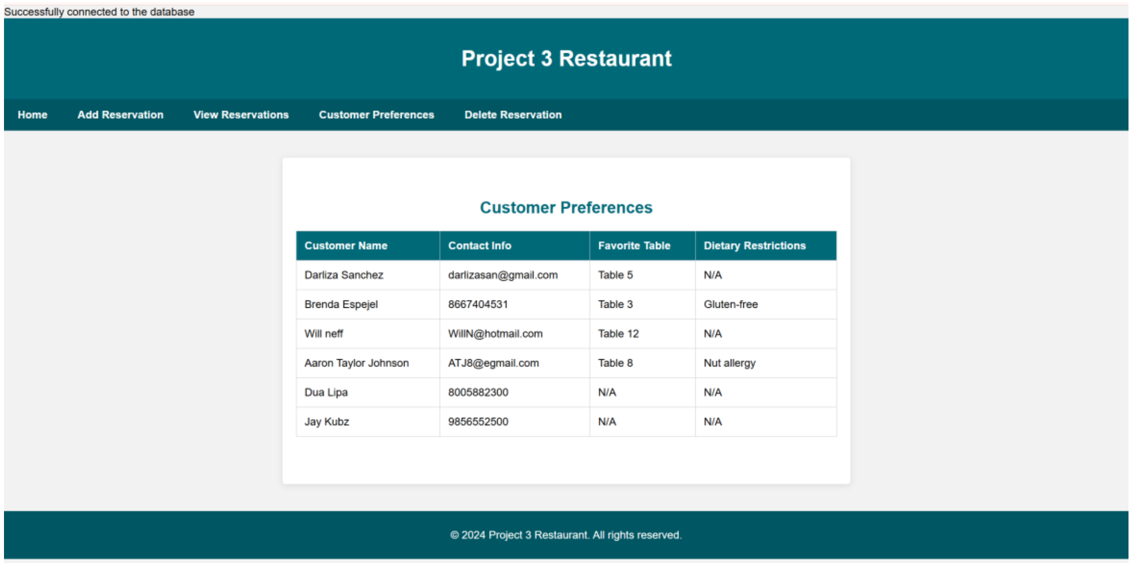
© 2024 Project 3 Restaurant. All rights reserved.

This file provides a simple form interface for deleting a reservation from the system by specifying its Reservation ID. It integrates with the backend (RestaurantServer.php) to process the deletion request.

The content Section Wraps the delete form in a styled container. Form Attributes: method="POST" Specifies that the form will send data to the server via the POST method, which is more secure and appropriate for data modification actions like deletion. action="RestaurantServer.php?action=deleteReservation" Directs the form submission to RestaurantServer.php with the action=deleteReservation parameter, ensuring the backend knows which function to execute.

Reservation ID Field Allows the user to enter the unique reservation_id of the reservation they want to delete. Attributes: type="number": Ensures the input only accepts numeric values, which match the database schema. name="reservation_id": Assigns the data a key for processing in the backend (accessible in PHP as $_POST['reservation_id']). id="reservation_id": Links the field to the <label> for better accessibility. Required: Ensures the field cannot be left empty, reducing the chance of errors.

ViewCustomerPreferences.php


Successfully connected to the database

Customer Preferences

Customer Name	Contact Info	Favorite Table	Dietary Restrictions
Darliza Sanchez	darlizasan@gmail.com	Table 5	N/A
Brenda Espejel	8667404531	Table 3	Gluten-free
Will neff	WillN@hotmail.com	Table 12	N/A
Aaron Taylor Johnson	ATJ8@egmail.com	Table 8	Nut allergy
Dua Lipa	8005882300	N/A	N/A
Jay Kubz	9856552500	N/A	N/A

© 2024 Project 3 Restaurant. All rights reserved.

This page dynamically displays customer details, including their preferences, in a tabular format. The data ($customers) is fetched from the backend.

Table Structure: the structure of the table with clear headers:

Customer Name: Displays the customer's name.

Contact Info: Shows the customer's contact information.

Favorite Table: Indicates the customer's table preference.

Dietary Restrictions: Lists any dietary restrictions.

Dynamic Rows (PHP Loop): Dynamically populates the table rows based on the $customers array fetched from the backend. Row Details include a foreach Loop: Iterates through the $customers array, creating a new row (<tr>) for each customer. Data Cells (<td>): contactInfo: Contact details (e.g., phone number or email). favoriteTable: The customer's preferred table, or "N/A" if no preference is set (?? 'N/A' handles null values). dietaryRestrictions: The customer's dietary restrictions, or "N/A" if none are provided.

The $customers array is passed to this view by RestaurantServer.php, which fetches data from the database using the getCustomerPreferences() method in RestaurantDatabase.php. The process works as follows:

Query Execution: The backend retrieves data from the Customers and DiningPreferences tables.

It uses a LEFT JOIN to ensure customers without preferences are still included.

Data Passing: The $customers array, containing rows of data, is passed to viewCustomerPreferences.php.

Dynamic Rendering: The PHP loop dynamically renders each row based on the array's contents.

One Challenge is that not all customers may have preferences (e.g., favorite table or dietary restrictions) stored in the database. Solution: Using the ?? 'N/A' syntax to display "N/A" for null or missing values.

RestaurantDatabase.php

This class handles all interactions with the database for the restaurant management system.

First the private properties store essential connection details for the database. Constructor and Database Connection: __construct(): Automatically initializes the database connection when a RestaurantDatabase object is created. connect(): Creates a new mysqli connection using the stored credentials.

Method: getAllReservations()

Fetches all reservations, including the associated customer names, using an INNER JOIN. It Joins the Reservations and Customers tables using customerId. Returns the data as an associative array with keys like reservationId, customerName, reservationTime, etc.

Method: addReservation()

Adds a new reservation by calling the stored procedure addReservation. Uses prepare() and bind_param() for secure parameterized queries. Parameters are bound to placeholders (?) using the appropriate data types: s (string) for customerName, contactInfo, reservationTime, and specialRequests. i (integer) for numberOfGuests.

Method: getCustomerPreferences()

Fetches customer preferences by joining the Customers and DiningPreferences tables. LEFT JOIN: Ensures all customers are included, even those without preferences. Returns details such as customer name, contact info, favorite table, and dietary restrictions.

Method: deleteReservation()

Deletes a reservation from the Reservations table based on its ID. Uses a parameterized query to prevent SQL injection. Ensures only the reservation with the specified reservationId is deleted.

The database was a page that was a major challenge and I had to be very careful with because it's easy to make error in. If the credentials or database name are incorrect, the connection will fail. It was major to have all the information correct.

RestaurantServer.php

This file serves as the controller for the restaurant management system. It handles user requests, interacts with the database via RestaurantDatabase.php, and dynamically includes the appropriate views.

First it loads the RestaurantDatabase class, which handles all database interactions.

Handling User Requests (handleRequest())

Purpose: Routes the request based on the action parameter in the query string (e.g., ?action=addReservation). Default Action: If no action is specified, it defaults to home(). It has a Modular design ensures easy scalability, adding new actions is as simple as defining a new case.

Method: viewCustomerPreferences()

Retrieves customer preferences from the database and includes the corresponding view. Fetches data using the getCustomerPreferences() method in RestaurantDatabase.php. Includes viewCustomerPreferences.php to display the data.

Method: home()

Defines the default action, which currently loads the addReservation.php view.

Method: addReservation()

Processes the form submission to add a reservation. How It Works: Checks if the request is POST (i.e., form submission). Retrieves data from $_POST, validates it, and calls addReservation() in RestaurantDatabase.php. On success or failure, sets an appropriate $message and re-includes the form for user feedback.

Method: viewReservations()

Displays reservations, optionally filtered by customer ID. If customerId is provided in the query string, fetches only that customer's reservations using findReservations(). Otherwise, fetches all reservations using getAllReservations().

Method: addSpecialRequest()

Allows users to add or update a special request for a specific reservation.

Method: deleteReservation()

Deletes a reservation by ID. Checks if the request is POST. Retrieves the reservation ID from $_POST and calls deleteReservation() in RestaurantDatabase.php. Redirects back to the viewReservations page after completion.

$portal = new RestaurantPortal(); and $portal->handleRequest();

Creates an instance of RestaurantPortal and calls handleRequest() to process the user's action.

Restaurant_reservations Databse

The SQL script sets up a database schema for a restaurant management system, with tables for customers, dining preferences, and reservations, as well as stored procedures for performing operations like adding reservations, updating special requests, and finding reservations.

Customers Table:

```
CREATE TABLE `customers` (
  `customerId` int(11) NOT NULL,
  `customerName` varchar(50) NOT NULL,
  `contactInfo` varchar(200) DEFAULT NULL
);
```

Stores customer details. customerId is the primary key and auto-incremented. Ensures unique combinations of customerName and contactInfo.

DiningPreferences Table:

```
CREATE TABLE `diningpreferences` (
  `preferenceId` int(11) NOT NULL,
  `customerId` int(11) NOT NULL,
  `favoriteTable` varchar(35) DEFAULT NULL,
  `dietaryRestrictions` varchar(300) DEFAULT NULL
);
```

Stores customer-specific dining preferences. Links to customers via customerId (foreign key)/ Allows null values for favoriteTable and dietaryRestrictions to accommodate customers without preferences.

Reservations Table:

```
CREATE TABLE `reservations` (
  `reservationId` int(11) NOT NULL,
  `customerId` int(11) NOT NULL,
  `reservationTime` datetime NOT NULL,
  `numberOfGuests` int(11) NOT NULL,
  `specialRequests` varchar(150) DEFAULT NULL
);
```

Tracks reservations for customers. Links to customers via customerId. Includes fields for time, guest count, and special requests.

Stored Procedures:

```
CREATE PROCEDURE `addReservation` (
    IN `customerNameInput` VARCHAR(50),
    IN `contactInfoInput` VARCHAR(200),
    IN `reservationTimeInput` DATETIME,
    IN `numberOfGuestsInput` INT,
    IN `specialRequestsInput` VARCHAR(150)
)
```

Adds a reservation by: Checking if the customer already exists, adding a new customer if necessary, Inserting the reservation. Uses DECLARE to store the customerId temporarily. Uses LAST_INSERT_ID() to retrieve the customerId of a newly added customer.

FindReservations

```
CREATE PROCEDURE `findReservations` (IN `customerId` INT)
```

Retrieves all reservations for a given customer ID.

Relationships and Constraints:

Foreign Keys: Enforces referential integrity between tables.

Indexes: Improves performance and ensures unique identification of customers by name and contact info.

Project 3 Restaurant Management System covers core demands for up-to-date restaurants by managing efficiently the reservation flow and customer base. A relational database ensures sound data management, and a backend in PHP performs a fine integration of the Logic-Presentation Layers.