

**CMU B.Sc. (HONS) SE/B.Sc. (Hons) SE- ASSIGNMENT FEEDBACK SHEET -ICBT CAMPUS**

**Student Details ( Student should fill the content)**

Name	Niranjani Nithyanandhan
Student ID	CL/BSCSD/28/58

**Scheduled unit details**

Unit code	CIS6003	
Unit title	Advanced Programming	
Unit enrolment details	Year	3
	Study period	
Lecturer	Ms. Madhuni Prasadini	
Mode of delivery	Full Time	

**Assignment Details**

Nature of the Assessment	Course work		
Topic of the Case Study	Reservation system		
Learning Outcomes covered			
Word count	4000		
Due date / Time	7th September 2024		
Extension granted?	Yes	No	Extension Date
Is this a resubmission?	Yes	No	Resubmission Date

**Declaration**

I certify that the attached material is my original work. No other person's work or ideas have been used without acknowledgement. Except where I have clearly stated that I have used some of this material elsewhere, I have not presented it for examination / assessment in any other course or unit at this or any other institution

Name/Signature		Date	
----------------	--	------	--

**Submission**

Return to:				
<b>Result</b>				
Marks by 1 <sup>st</sup> Assessor		Signature of the 1 <sup>st</sup> Assessor		<b>Agreed Mark</b>
Marks by 2 <sup>nd</sup> Assessor		Signature of the 2 <sup>nd</sup> Assessor		
<b>Comments on the Agreed Mark.</b>				
<b>For Office use only (hard copy assignments)</b>				
Receipt date		Received by		

## **Table of contents**

<b>Introduction .....</b>	5
<b>Executive Summary .....</b>	6
<b>Turnitin report.....</b>	8
<b>System design with UML diagrams .....</b>	10
Objects .....	13
Classes.....	13
Encapsulation .....	14
Inheritance .....	14
Polymorphism.....	15
Abstraction.....	15
Actors.....	18
Objects .....	18
Sequence.....	18
Actors.....	20
Objects .....	20
Sequence.....	21
<b>Design patterns.....</b>	23
<b>Creational Patterns .....</b>	23
<b>Structural Patterns.....</b>	23
<b>Behavioral Patterns .....</b>	24
<b>Evaluating Design Patterns .....</b>	25
<b>Creational Patterns .....</b>	25
<b>Structural Patterns.....</b>	26
<b>Behavioral Patterns .....</b>	26
<b>Design pattern for ABC Restaurant .....</b>	27
<b>Architecture development .....</b>	27
Homepage .....	27
About us .....	28

Menu Page .....	28
Database Schema for ABC Restaurant .....	38
Tables.....	38
Relationships .....	39
Keys.....	40
<b>The database schema for ABC Restaurant effectively supports the business logic by ....</b>	40
<b>Testing.....</b>	42
<b>TEST PLAN.....</b>	42
Project Information.....	42
<b>TEST LEVELS AND TYPES .....</b>	42
<b>TEST ENVIRONMENT.....</b>	43
<b>ASSUMPTIONS .....</b>	43
<b>EXIT CRITERIA .....</b>	43
<b>Work with github .....</b>	48
<b>References.....</b>	51

## **Introduction**

ABC Restaurant, a popular chain in Sri Lanka, seeks to enhance its operations and customer experience by implementing an online reservation system. This system will provide customers with a convenient way to book tables, view menus, and access other restaurant information.

The primary objectives of this project are,

- To develop a user-friendly online reservation system for ABC Restaurant.
- To enable customers to make reservations, view menus, and access restaurant information.
- To streamline the reservation process and improve customer satisfaction.
- To provide restaurant staff with a tool to manage reservations and track restaurant performance.

This project will focus on developing an online reservation system that includes the following functionalities:

- Customer Registration -Allow customers to create accounts and manage their profile information.
- Reservation Management- Enable customers to make, modify, and cancel reservations.
- Menu Display- Showcase the restaurant's menu items with descriptions and prices.
- Payment Processing- Integrate with a payment gateway to facilitate online payments.
- Staff Management- Provide tools for restaurant staff to manage reservations and view customer information.
- Reporting- Generate reports on restaurant performance, customer trends, and sales data.

## **Executive Summary**

The ABC Restaurant Management System is a comprehensive software application designed to streamline the operations of ABC Restaurant. The system provides functionalities for managing orders, reservations, menus, staff, and customer information. By automating various tasks and providing valuable insights, the system aims to improve efficiency, reduce costs, and enhance customer satisfaction.

## **Key Features**

**Order Management** - Create, modify, and track orders.

**Reservation Management** - Manage table reservations and seating arrangements.

**Menu Management** -Add, edit, and remove menu items.

**Staff Management** - Manage staff schedules, roles, and permissions.

**Customer Management** - Track customer information, preferences, and loyalty program details.

**Reporting** - Generate reports on sales, customer behaviour, and restaurant performance.

## **Benefits**

**Increased Efficiency**- Streamline operations and reduce manual tasks.

**Improved Customer Experience**- Provide a seamless and convenient dining experience.

**Enhanced Decision-Making**-Access real-time data and analytics to make informed decisions.

**Cost Savings**- Optimize resource allocation and reduce operational expenses.

## **Project Timeline**

Timeline for development one month, testing, and implementation took around two weeks

## **Expected Outcomes**

A fully functional and user-friendly restaurant management system.

Improved operational efficiency and customer satisfaction.

Increased profitability and return on investment.

## Turnitin report

CL-BSCSD-28-58.docx

ORIGINALITY REPORT

**13%**  
SIMILARITY INDEX

**4%**  
INTERNET SOURCES

**5%**  
PUBLICATIONS

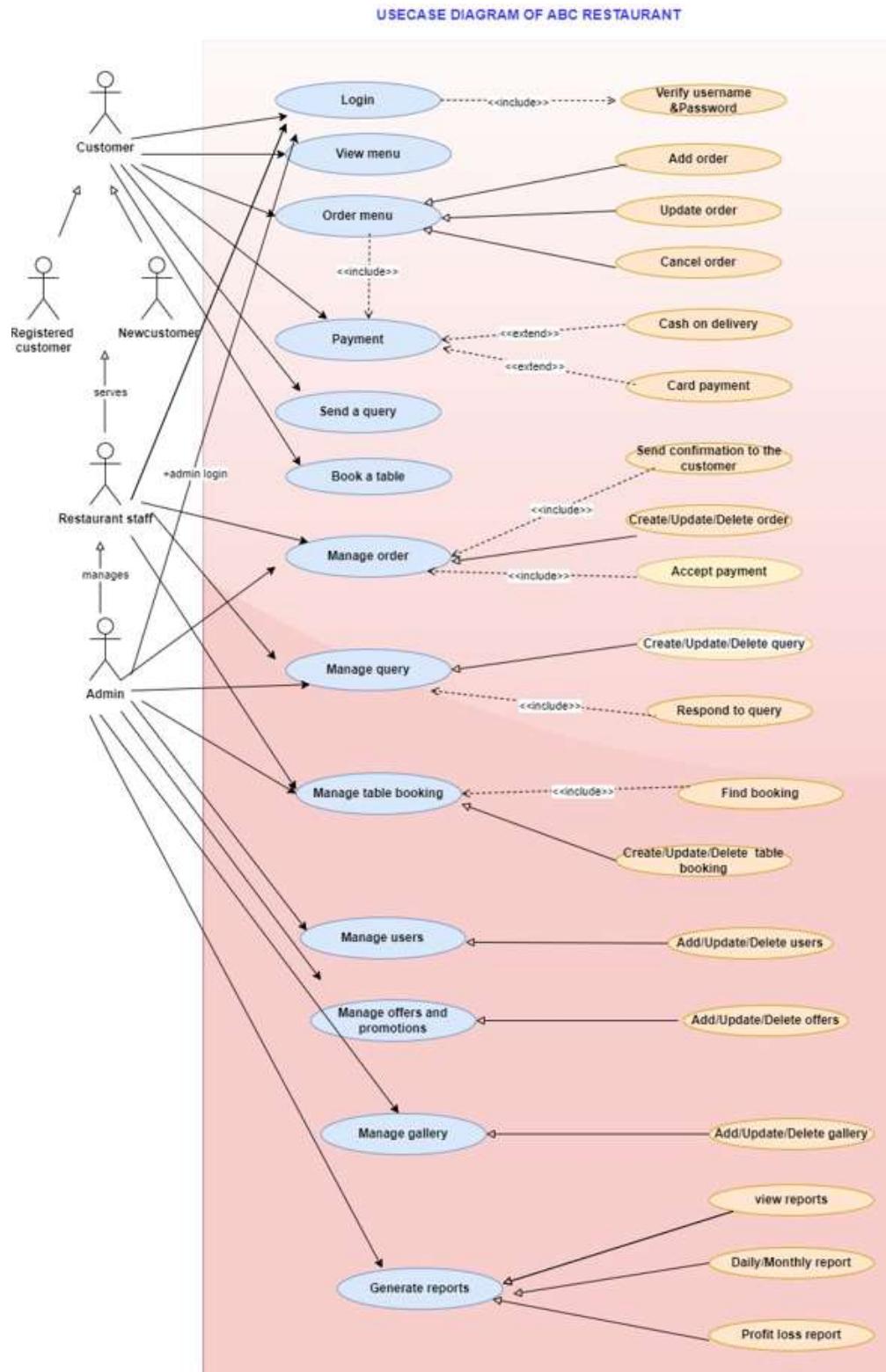
**11%**  
STUDENT PAPERS

PRIMARY SOURCES

- |          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Submitted to Southern New Hampshire University - Continuing Education</b><br>Student Paper            | <b>2%</b> |
| <b>2</b> | <b>Submitted to University of Wales Institute, Cardiff</b><br>Student Paper                              | <b>2%</b> |
| <b>3</b> | <b>Submitted to Asia Pacific University College of Technology and Innovation (UCTI)</b><br>Student Paper | <b>1%</b> |
| <b>4</b> | <b>github.com</b><br>Internet Source   | <b>1%</b> |
| <b>-</b> | <b>Submitted to Kaplan Professional</b>  | <b>1</b>  |

7	Submitted to University of Canterbury Student Paper	1 %
8	Submitted to National College of Ireland Student Paper	1 %
<hr/>		
9	Submitted to Centria UAS Student Paper	1 %
10	1library.co Internet Source	<1 %
11	Ronald J. Leach. "Introduction to Software Engineering", CRC Press, 2018 Publication	<1 %
12	Xin Xia, David Lo, Pavneet Singh Kochhar, Zhenchang Xing, Xinyu Wang, Shaping Li. "Experience report: An industrial experience	<1 %

## System design with UML diagrams



The use case diagram for ABC Restaurant provides a high-level overview of the interactions between actors (users) and the system. It helps visualize the functionalities that the system needs to provide.

### **Actor**

Represents a role that a user plays in the system. In this case, the actors are,

- Customer
- Restaurant Staff
- Admin

### **Use Case**

Represents a functionality that the system provides. In this case, the use cases are,

- Search for restaurant
- View menu
- Make reservations
- View offers
- Submit queries
- Register
- Login
- Manage reservations
- View customer queries
- Respond to queries
- Manage restaurant information
- Manage user accounts
- View reports
- Generate reports
- Manage gallery
- Manage offers
- Manage customers
- Manage payment process

### **Capture System Functionality**

The use case diagram effectively captures the core functionalities that the ABC Restaurant system needs to provide. It helps in understanding the scope of the system and the interactions between different user roles.

### **Identify Actors and Their Goals**

The diagram clearly identifies the actors involved in the system and their goals. This helps in designing the system to meet the needs of different user groups.

### **Define System Boundaries**

The use case diagram helps define the boundaries of the system by outlining what functionalities are within the scope of the system and what are outside.

### **Visualize System Interactions**

The diagram provides a visual representation of how actors interact with the system, making it easier to understand the system's workflow.

### **Serve as a Basis for Requirements Gathering**

The use case diagram can be used as a basis for gathering detailed requirements for each use case, ensuring that the system meets the needs of all stakeholders.

### **Support System Design**

The use case diagram can be used as a starting point for designing the system's architecture, classes, and interactions.

**Object-Oriented Programming (OOP)** is a programming paradigm that models real-world entities as objects. These objects have properties (attributes) and behaviours (methods). In the context of the ABC Restaurant use case diagram, OOP concepts hav applied as follows

## Objects

**Customer** -Represents a customer with attributes like name, address, contact information, and methods like register, login, makeReservation, and viewOrders.

**Restaurant** -Represents a restaurant with attributes like name, address, contact information, and methods like viewMenu, manageReservations, and processOrders.

**Order** -Represents an order with attributes like orderID, customerID, items, and status, and methods like placeOrder, cancelOrder, and viewOrder.

**Menu** -Represents a menu with attributes like menuID, items, and methods like addItem, removeItem, and viewMenu.

**Same as booking (a table), query, gallery, payment, report**

## Classes

**CustomerClass** -Defines the common properties and behaviors for all customers.

**RestaurantClass**-Defines the common properties and behaviors for all restaurants.

**OrderClass** -Defines the common properties and behaviors for all orders.

**MenuClass** -Defines the common properties and behaviors for all menus.

**Same as booking (a table),query,gallery,payment,report**

## Encapsulation

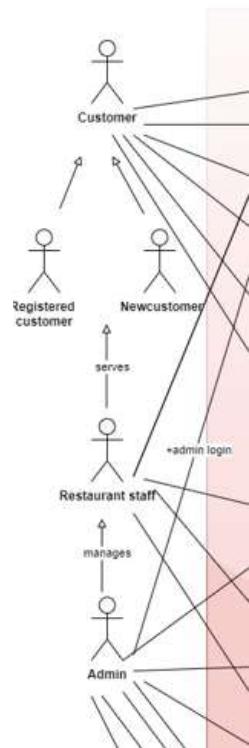
### Information Hiding

The internal implementation of objects is hidden from external users, providing better control and security. For example, the Order class might have methods to calculate the total amount of an order, but the specific calculation logic is hidden from other classes.

## Inheritance

### Hierarchical Relationships

Classes can inherit properties and methods from parent classes. For example, an Admin class might inherit properties and methods from the User class, with additional privileges. Same as registered customer.



## Polymorphism

### **Dynamic Binding**

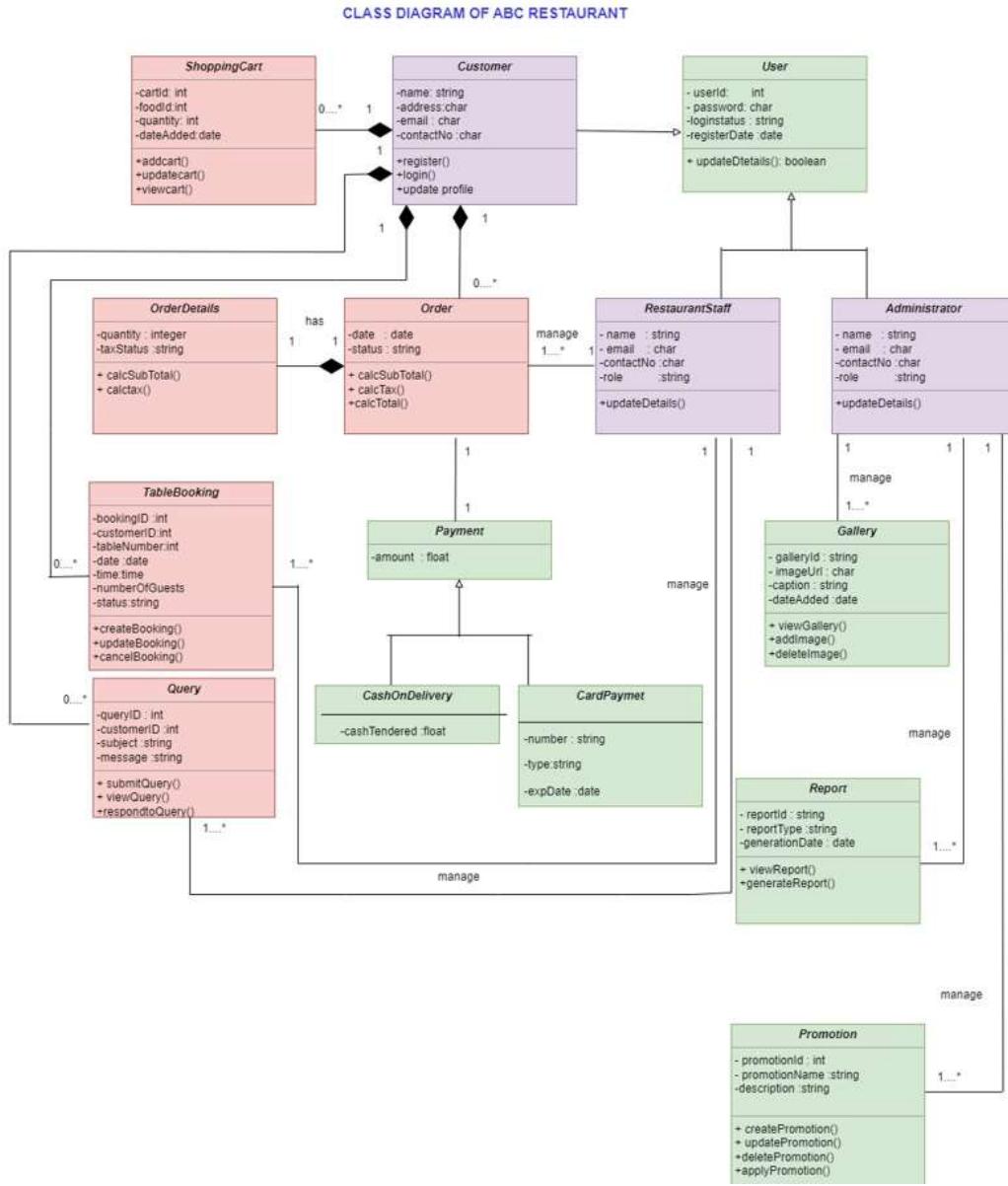
Objects can be treated as instances of their parent class or their own class, allowing for flexible code. For example, a method that takes an Order object as a parameter can work with different types of orders (e.g., dine-in, delivery).

## Abstraction

### **Focus on Essential Features**

Objects are defined by their interfaces (methods) rather than their internal implementation. For example, a Restaurant class might have a method to view the menu, without revealing the specific details of how the menu is stored or retrieved.

By applying these OOP concepts, the use case diagram for ABC Restaurant can be translated into a well-structured and maintainable software design.



The class diagram for ABC Restaurant provides a static view of the system's structure, showing the classes, their attributes, operations, and relationships. It helps in understanding the components of the system and how they interact.

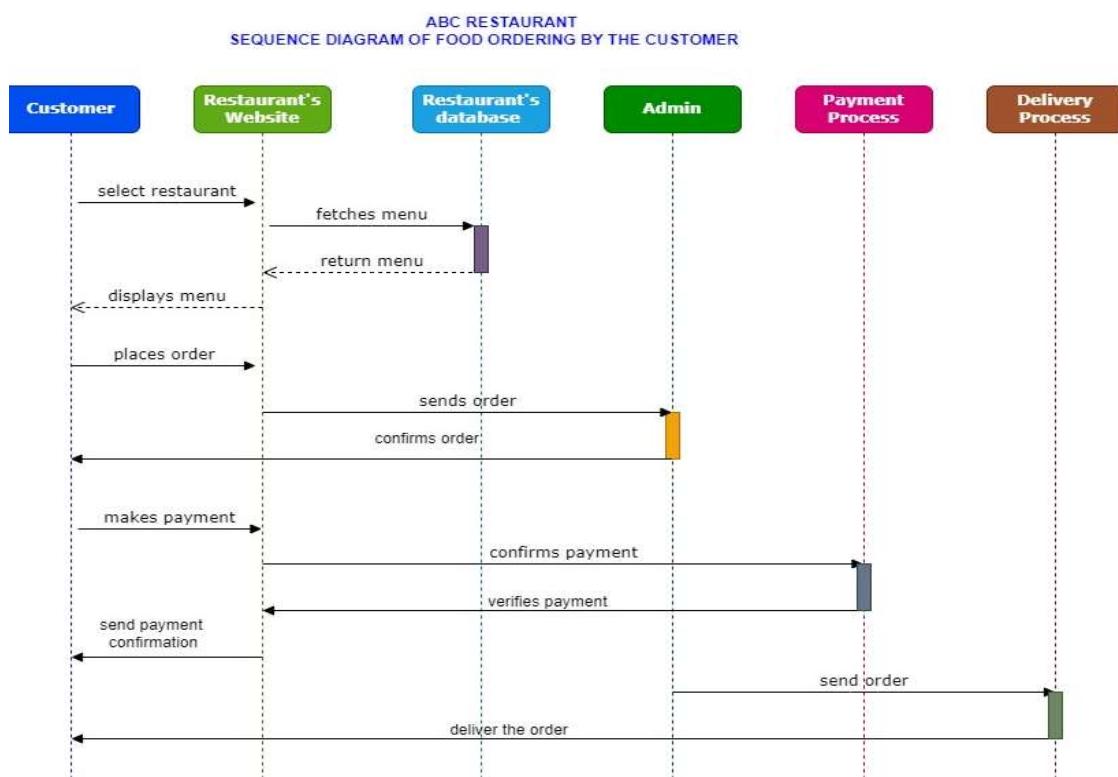
**Class**-Represents a set of objects with common properties (attributes) and behaviours (operations).

**Attributes**- Properties or characteristics of a class.

**Operations** -Actions or functions that a class can perform.

## **Relationships** - Show how classes are connected

The class diagram clearly defines the main components of the ABC Restaurant system, such as Customer, Restaurant, Order, Menu, and Staff ....etc. The diagram specifies the relevant attributes and operations for each class, providing a clear understanding of the data and functionality required. It illustrates the relationships between classes, such as the association between a Customer and multiple Orders, or the aggregation of Menu Items within a Menu. The class diagram can be used as a blueprint for designing the system's architecture, database schema, and object-oriented code. The diagram provides a visual representation of the system's structure, making it easier for stakeholders to understand and communicate about the design.



This sequence diagram provides a simplified overview of a food ordering system. The actual sequence may vary depending on specific requirements and implementation details.

## Actors

- Customer
- System

## Objects

- Customer
- Order
- Menu
- Payment

## Sequence

### **Customer browses the menu**

Customer selects a menu item.

### **System displays item details**

System retrieves item details from the database.

System displays item name, price, description, and image.

### **Customer adds item to cart**

Customer clicks the "Add to cart" button.

System adds the item to the customer's cart.

### **Customer places order**

Customer clicks the "Place order" button.

## **System processes order**

System verifies the customer's information.

System calculates the total amount.

System prompts the customer to choose a payment method.

## **Customer selects payment method**

Customer chooses a payment method (e.g., credit card, cash on delivery).

## **System processes payment**

If credit card payment

- System redirects the customer to a payment gateway.
- Payment gateway processes the payment.
- System receives payment confirmation.

If cash on delivery

- System records the cash payment.

## **System confirms order**

System sends an order confirmation email to the customer.

## **Restaurant receives order**

System sends the order to the restaurant's kitchen.

## **Restaurant prepares order**

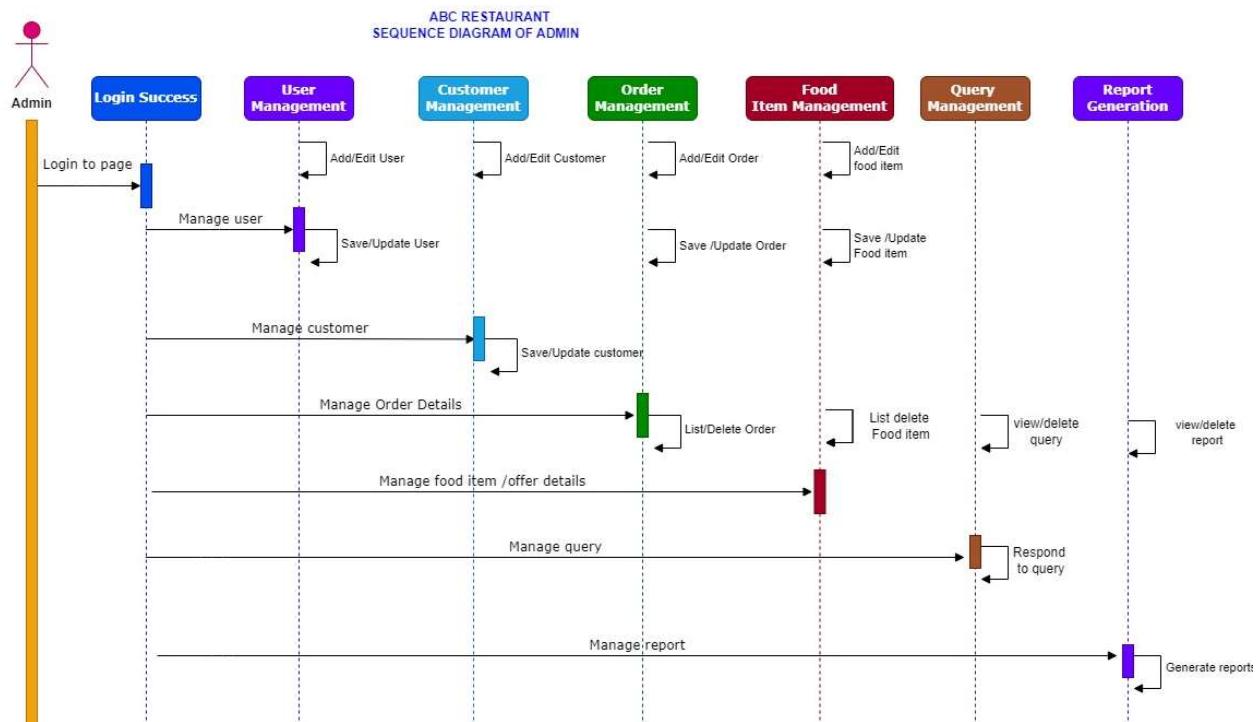
Restaurant prepares the food.

## Restaurant delivers order

Restaurant delivers the order to the customer's address.

## Customer receives order

Customer receives the order and enjoys the food.



## Actors

- Admin
- System

## Objects

- Admin
- User
- Restaurant

- Menu
- Order
- Report
- Query
- Customer

## Sequence

### **Admin logs in**

Admin enters their credentials.

System verifies credentials and grants access.

### **Admin manages users**

Admin selects the "Manage Users" option.

System displays a list of users.

Admin can add, edit, or delete users.

### **Admin manages restaurants**

Admin selects the "Manage Restaurants" option.

System displays a list of restaurants.

Admin can add, edit, or delete restaurants.

### **Admin manages menus**

Admin selects the "Manage Menus" option.

System displays a list of menus.

Admin can add, edit, or delete menu items.

### **Admin views reports**

Admin selects the "View Reports" option.

System generates various reports (e.g., sales, orders, and customer data).

Admin can filter and analyze the reports.

### **Admin generates promotions**

Admin selects the "Generate Promotions" option.

System allows admin to create new promotions with specific details (e.g., discount, validity).

### **Admin manages offers**

Admin selects the "Manage Offers" option.

System allows admin to create, edit, or delete offers.

### **Admin manages gallery**

Admin selects the "Manage Gallery" option.

System allows admin to upload, edit, or delete images

## Design patterns

Design patterns are reusable solutions to common software design problems. They provide proven templates for solving specific design issues, improving code quality, maintainability, and flexibility. Here are some of the most common design patterns categorized by their purpose:

### Creational Patterns

These patterns deal with object creation mechanisms. They help abstract the instantiation process, making the system more flexible and easier to change.

**Singleton** - Ensures a class has only one instance and provides a global point of access.

**Factory Method**- Defines an interface for creating an object, but lets subclasses decide which class to instantiate.

**Abstract Factory**- Provides an interface for creating families of related or dependent objects without specifying their concrete classes.

**Builder**- Separates the construction of a complex object from its representation, allowing the same construction process to create different representations.

**Prototype**-Creates new objects by copying an existing object.

### Structural Patterns

These patterns deal with how classes and objects are composed to form larger structures. They focus on relationships between objects.

**Adapter** - Converts the interface of a class into another interface clients expect.

**Bridge** -Decouples an abstraction from its implementation, so the two can vary independently.

**Composite** - Composes objects into tree structures to represent part-whole hierarchies.

**Decorator** -Attaches additional responsibilities to an object dynamically.

**Façade** -Provides a unified interface to a set of interfaces in a subsystem.

**Flyweight** -Reduces the number of objects by sharing common data.

**Proxy** - Provides a surrogate or placeholder for another object to control access to it.

## Behavioral Patterns

These patterns concern how objects interact and communicate with each other. They address algorithmic and object interactions.

**Template Method** - Defines the skeleton of an algorithm in an operation, deferring some steps to subclasses.

**Strategy** -Defines a family of algorithms, encapsulates each one, and makes them interchangeable.

**Observer** -Defines a one-to-many dependency between objects, so that when one object changes state, all its dependents are notified and updated automatically.

**Iterator**- Provides a way to access the elements of an aggregate object sequentially without exposing its internal representation.

**Visitor**- Represents an operation to be performed on the elements of an object structure.

**State:** Allows an object to alter its behavior when its internal state changes.

**Memento**- Captures and restores the internal state of an object.

**Command**- Encapsulates a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

**Mediator** - Defines an object that encapsulates how a set of objects interact.

**Interpreter**- Defines a grammar for a language and provides an interpreter to evaluate expressions in the language.

## Evaluating Design Patterns

Choosing the right design pattern for a specific problem requires careful consideration of several factors. Here's a comparative analysis of some common design patterns, highlighting their strengths and weaknesses

### Creational Patterns

#### Singleton

**Strengths**- Ensures only one instance of a class, useful for global objects.

**Weaknesses** -Can make the code less testable and harder to maintain.

#### Factory Method

**Strengths** - Promotes flexibility and extensibility, allows for easy addition of new classes.

**Weaknesses**- Can introduce additional complexity.

#### Abstract Factory

**Strengths** -Creates families of related objects, useful for complex systems.

**Weaknesses** -Can be overkill for simpler systems.

## Structural Patterns

### Adapter

**Strengths** - Adapts existing interfaces to match new requirements.

**Weaknesses** - Can introduce additional complexity.

### Decorator

**Strengths** - Adds new responsibilities to objects dynamically.

**Weaknesses** - Can make code less readable if used excessively.

### Composite

**Strengths**- Represents part whole hierarchies, useful for tree-like structures.

**Weaknesses** -Can be complex to implement for large hierarchies.

## Behavioral Patterns

### Strategy

**Strengths**- Encapsulates algorithms and makes them interchangeable.

**Weaknesses**- Can introduce additional complexity.

### Observer

**Strengths**- Defines a one-to-many dependency between objects.

**Weaknesses** - Can lead to performance overhead if not used carefully.

## **Iterator**

**Strengths** - Provides a way to access elements of an aggregate object without exposing its internal representation.

**Weaknesses**- Can be less efficient than direct access.

## **Design pattern for ABC Restaurant**

Based on the requirements of ABC Restaurant I have chosen MVC (Model-View-Controller). It Separates concerns into distinct layers, making the system more modular and easier to maintain. Benefits are improved testability, flexibility, and scalability. The Model layer can represent the restaurant's data (e.g., menu items, orders, customers), the View layer can handle the user interface (e.g., displaying menus, taking orders), and the Controller can manage user interactions and update the Model and View.

## **Architecture development**

### **Homepage**

**Header** - Restaurant logo, search bar, navigation menu (e.g., Home, Menu, Reservations, About Us, Contact Us)

**Hero Image** - A visually appealing image showcasing the restaurant's ambiance or signature dish.

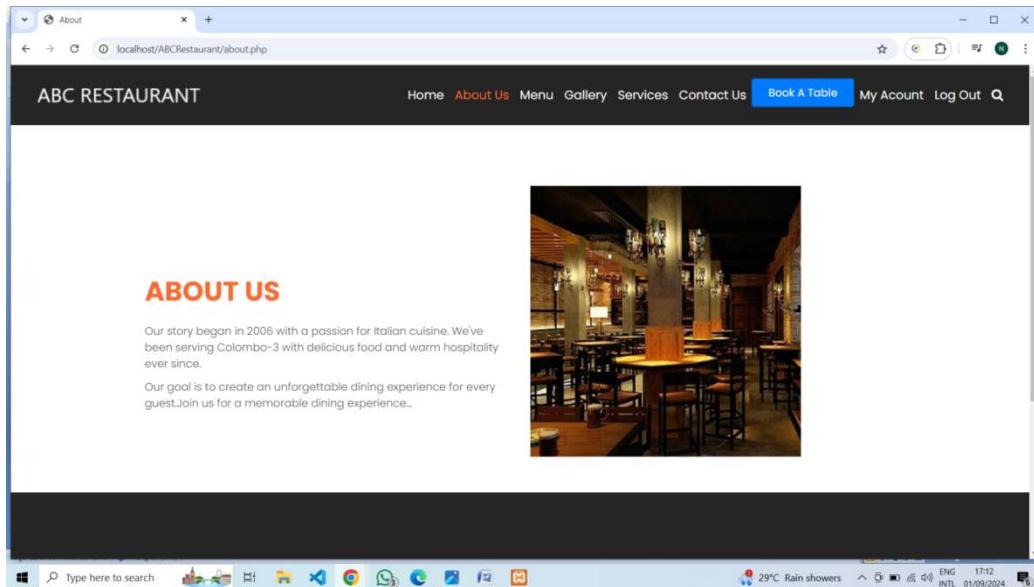
**Featured Items** -A section highlighting popular or recommended menu items.

**Call to Action** - A prominent button encouraging users to make a reservation or view the menu.



## About us

Small description about ABC Restaurant

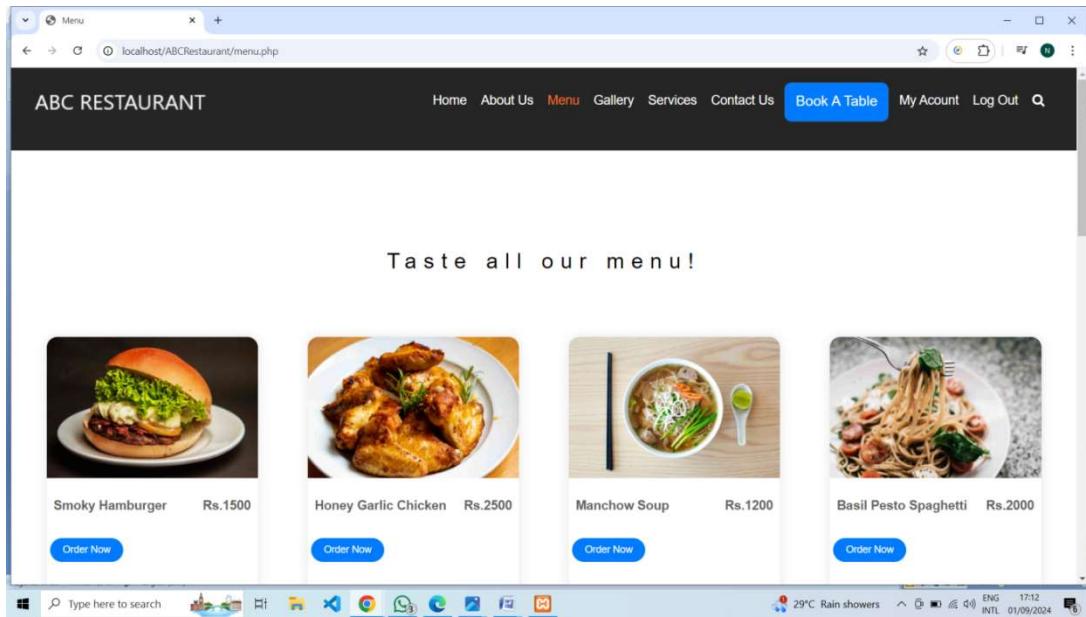


## Menu Page

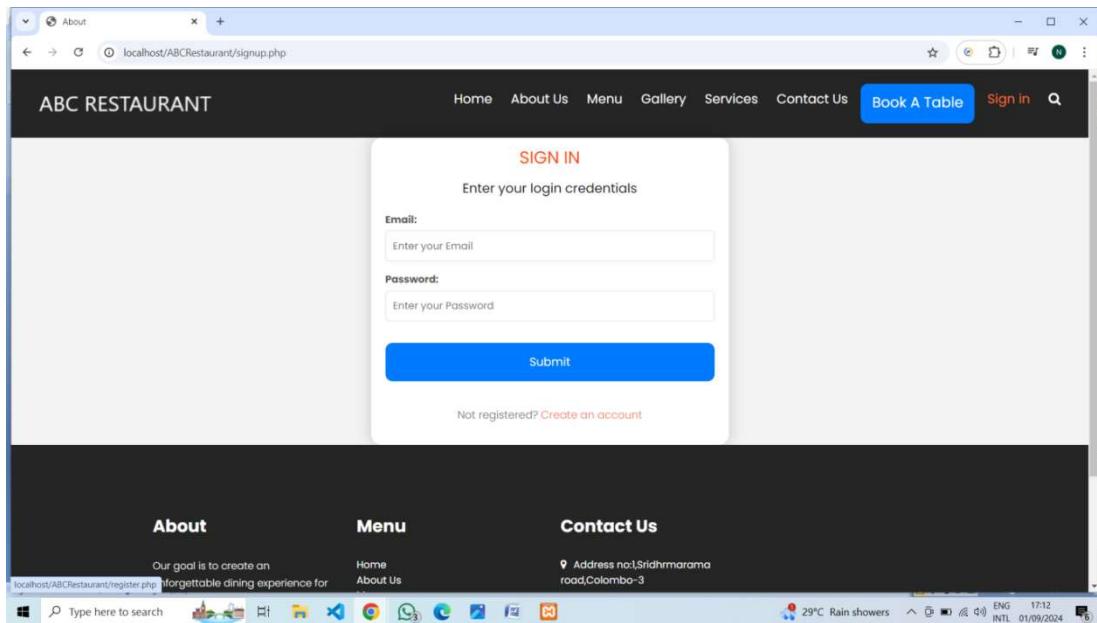
**Menu Categories-** A list of menu categories (e.g., Appetizers, Main Courses, Desserts).

**Item Cards** -Each item card should display the item's name, price, description, and image.

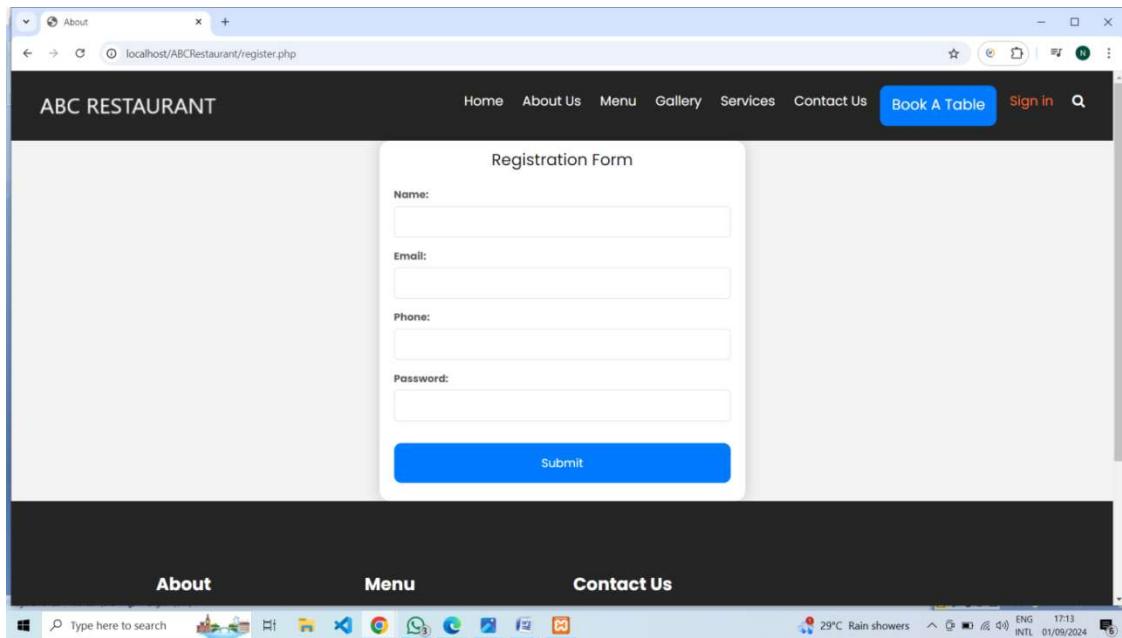
**Add to Cart Button-** A button for each item to add it to the cart.



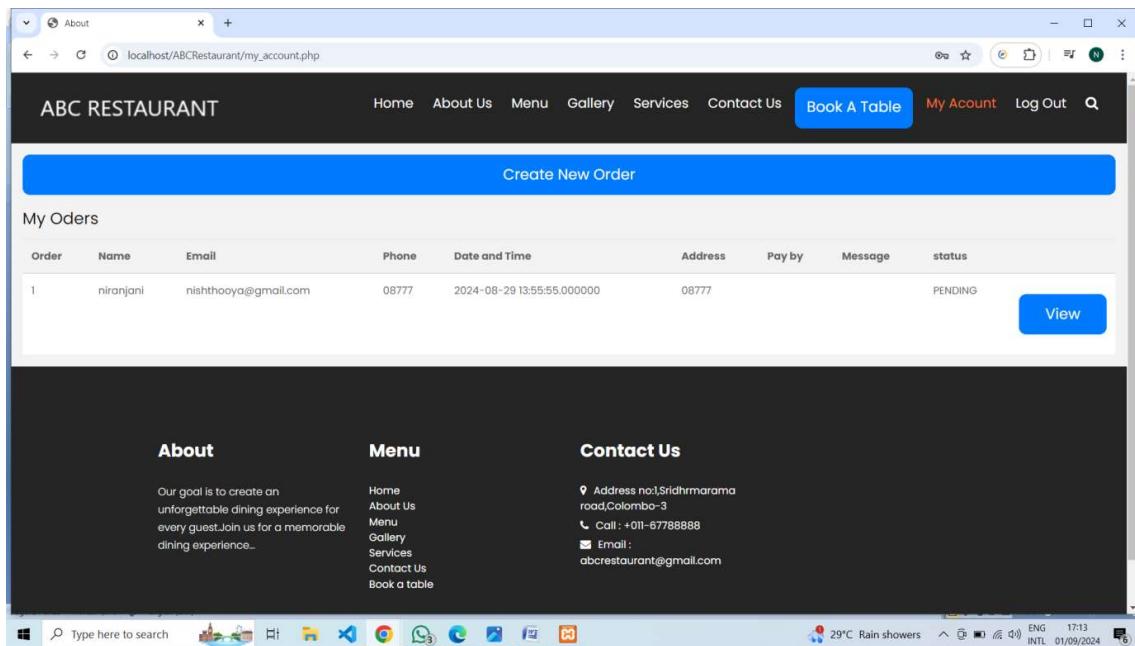
Once you click the order now button it redirects to “sign in” page



If customer doesn't have an account it advises to create an account redirects to “”registration page”



After login success customer can view his previous orders as well as he can able to make new order through this page.



After clicking create an order it redirects to below page from this you can make a new order once you select the desired food item and quantity it will display in a small dialog box from this you

can give the required details and confirm the order. Confirmation message will be sent to the particular customers email. (Since I have run the project via local host I couldn't attached the screen shot of email evidence)

The screenshot shows a web browser window titled "Menu" with the URL "localhost/ABCRestaurant/order.php". The page displays a shopping cart summary and a form for customer details.

Name	Quantity	Price (Rs)	Total (Rs)	Action
Honey Garlic Chicken	1	2500	2,500.00	<a href="#">Remove</a>
			Total	Rs 2,500.00

**Customer Details**

Customer Name: niranjan  
 Phone: 08777  
 Email: nishthooya@gmail.com  
 Delivery Address:   
 Delivry Date and time: 2024-09-01 13:44:07  
 Pay Online: Visa/Master Amex    
 Pay on Delivery: Visa/Master Amex Cash     
 Message:   
 Place Order

This is the gallery page with some attractive food images.

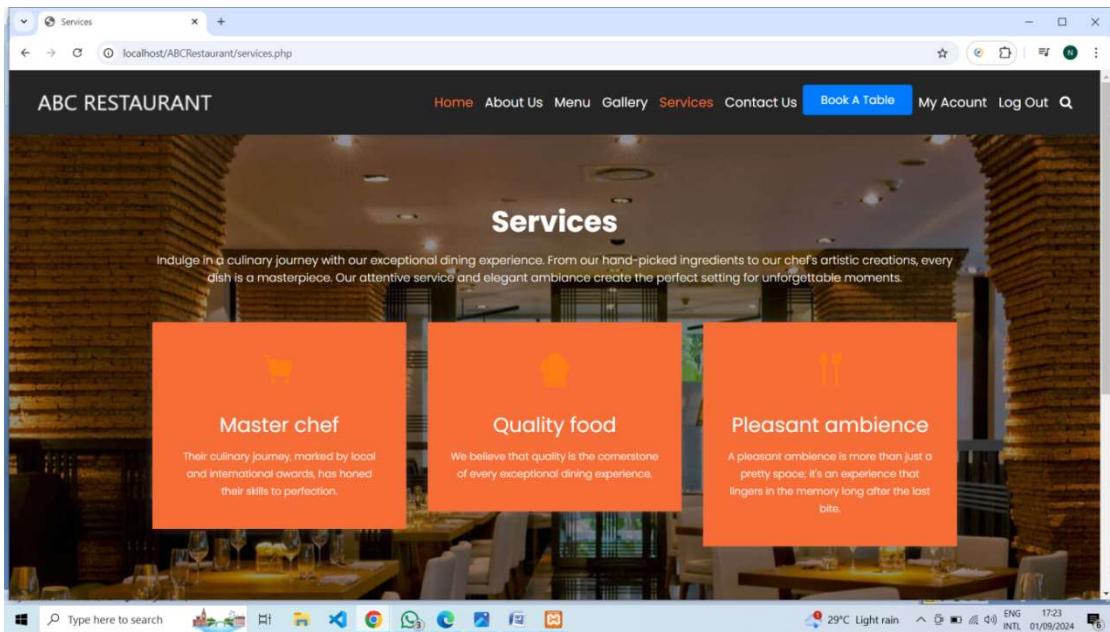
The screenshot shows a web browser window titled "Gemini" with the URL "localhost/ABCRestaurant/gallery.php". The page features a navigation bar with links to Home, About Us, Menu, **Gallery**, Services, Contact Us, Book A Table, Sign in, and a search icon.

## Our Gallery

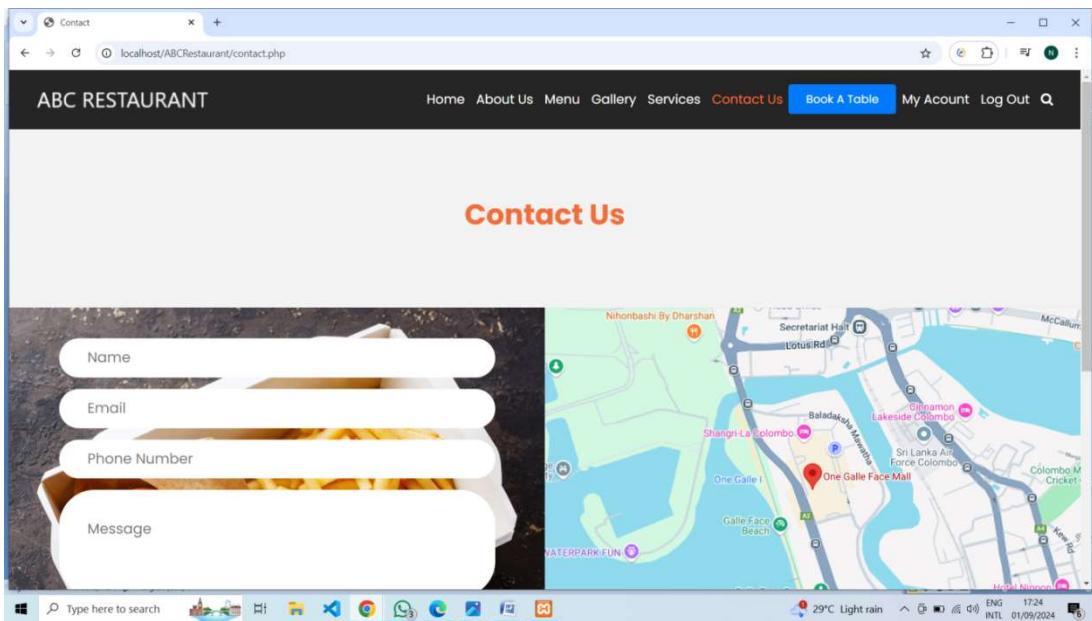
Our curated collection showcases diverse styles and mediums. Immerse yourself in creativity and inspiration.

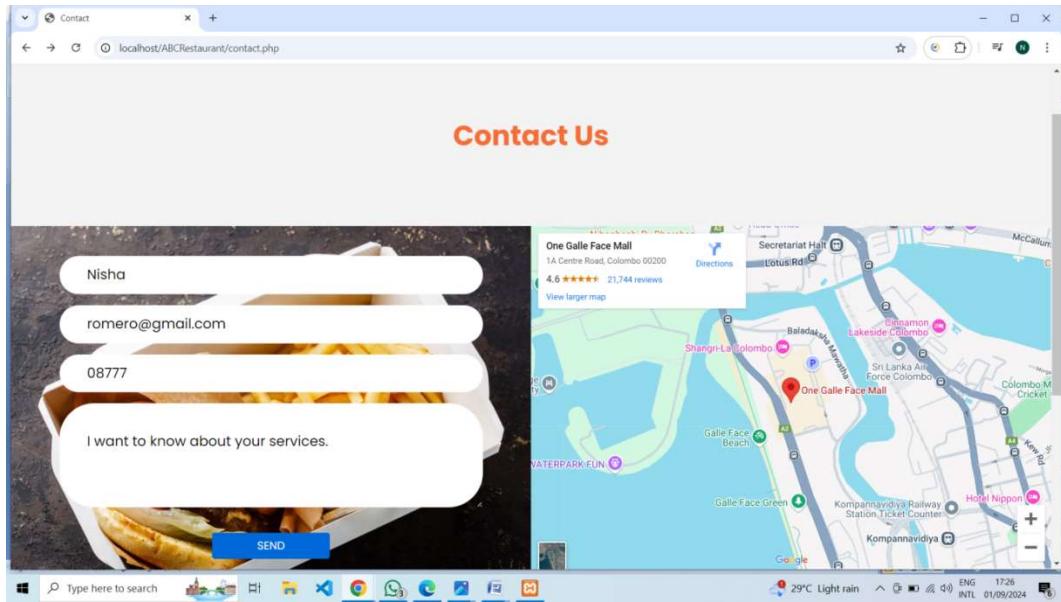
30°C Mostly cloudy 15:25 ENG INTL 01/09/2024

This is the services page with some attractive content to grab the customers easily.

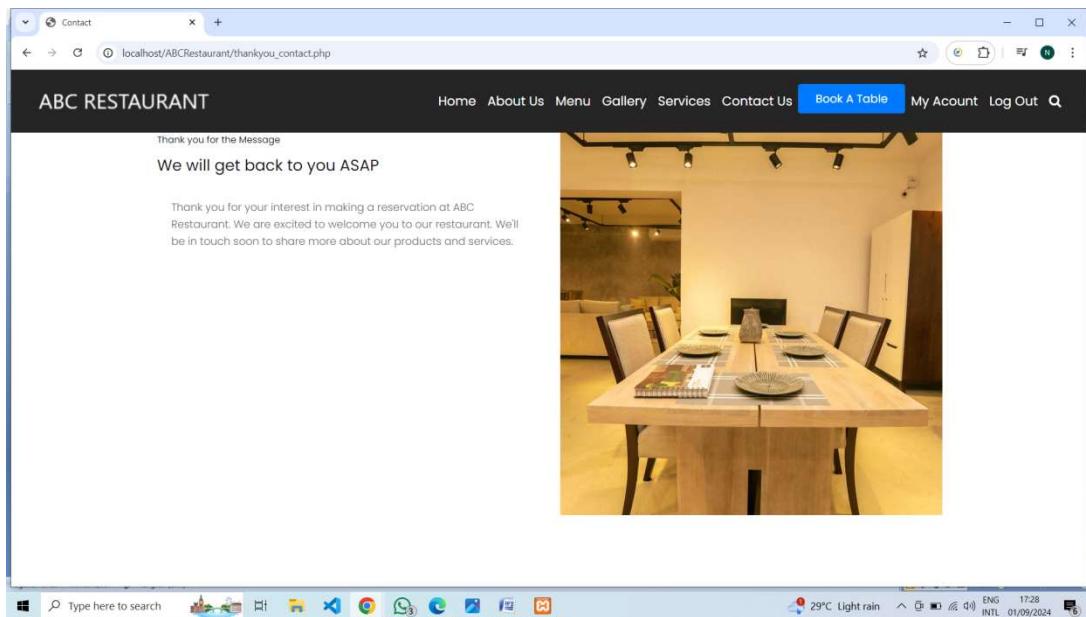


Contact us page with the map of the restaurant

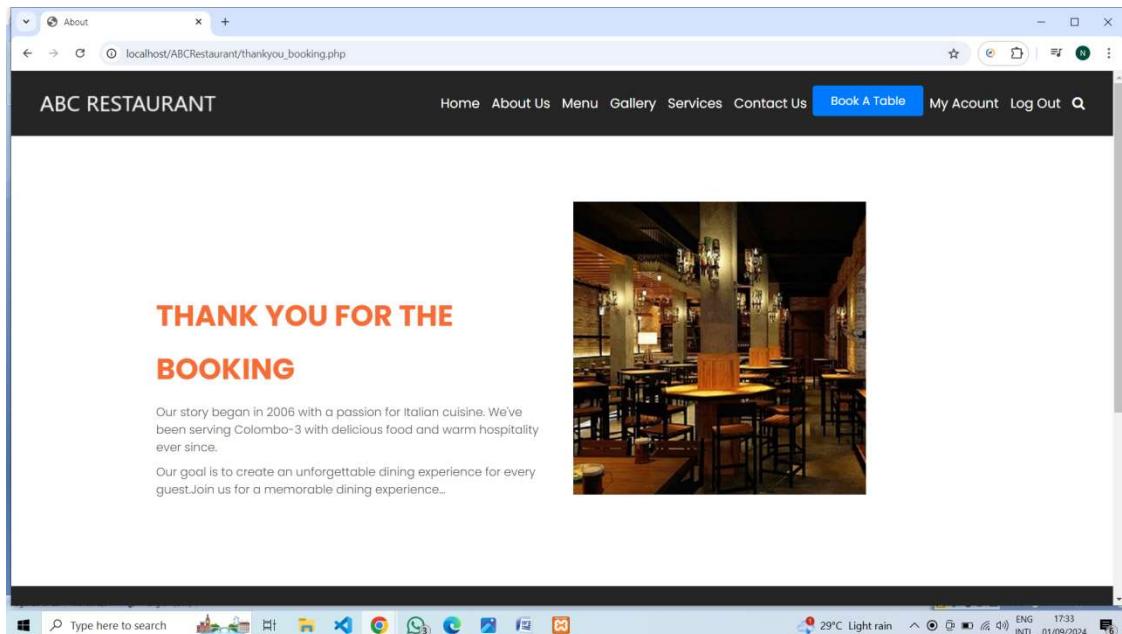




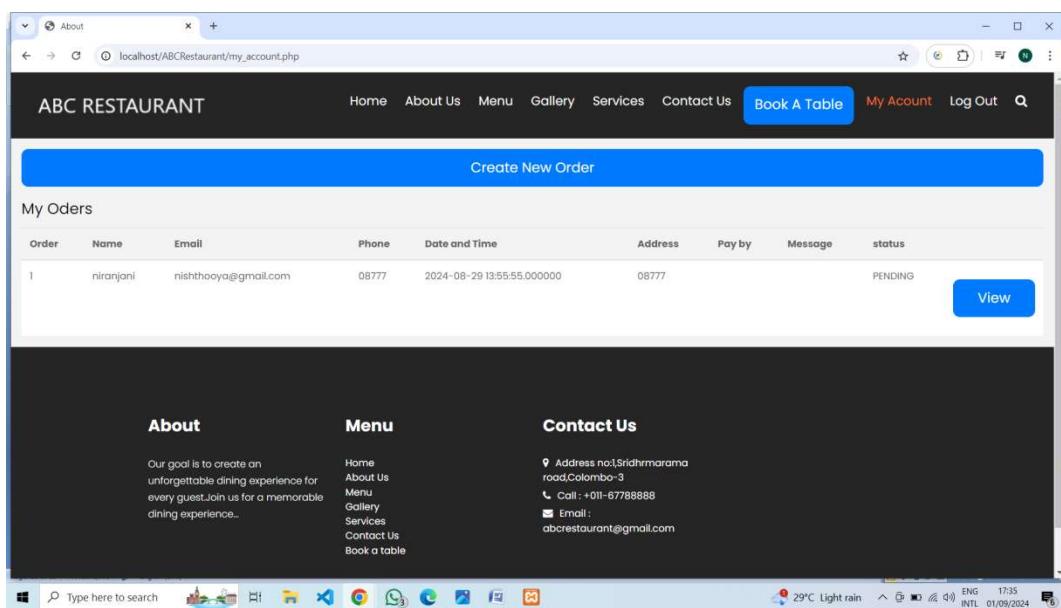
After filled up the contact us form it will be redirected to "thank you message" page



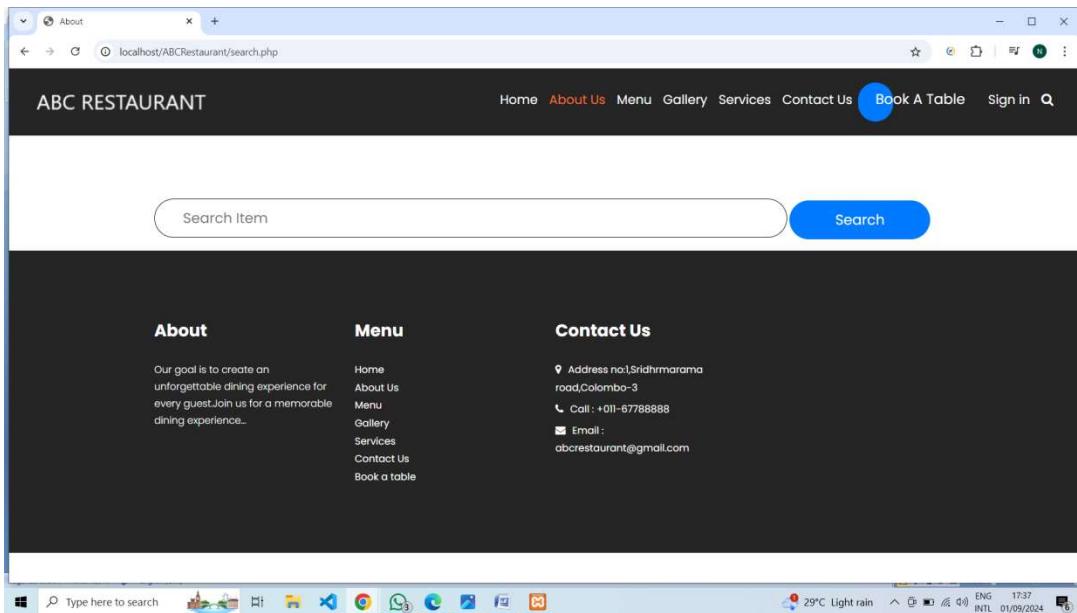
Book a table page customer can make the table reservation. After filled up the booking form customer will get the thankyou message.



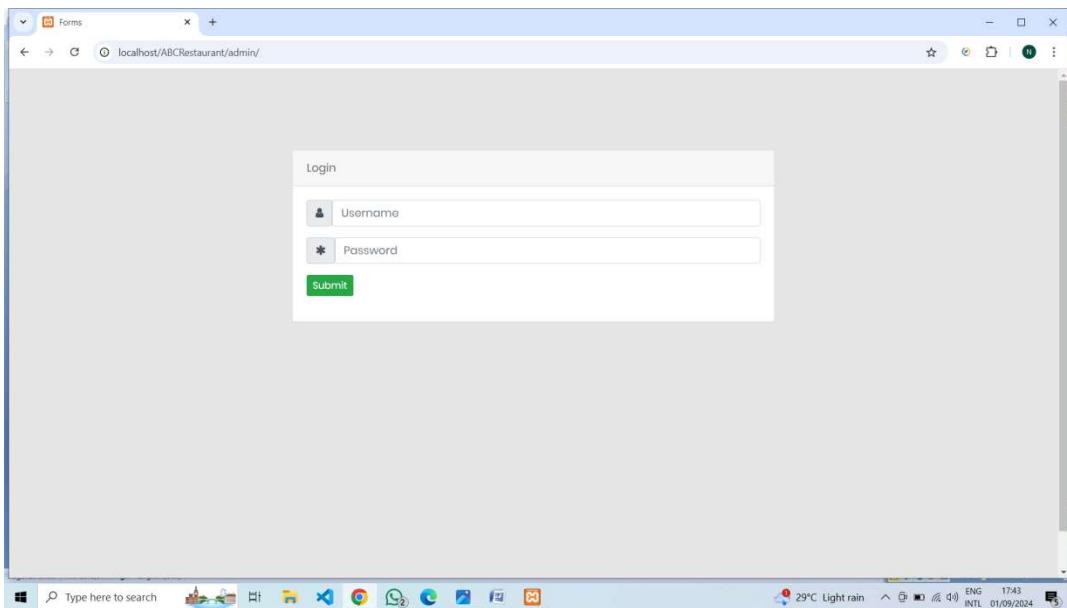
My account-Registered customer can view the order details and history of orders.



## Search page



## Admin dashboard -Login



Can be seen the table bookings made by customers.

The screenshot shows a Windows desktop environment with a browser window open to 'localhost/ABCRestaurant/admin/dashboard.php'. The page title is 'Restaurant Management'. On the left, a sidebar menu under 'COOL ADMIN' includes 'Dashboard', 'Bookings' (selected), 'Orders', 'Inquiries', 'Customers', 'Reports', 'Menu Items', 'Gallery', and 'User Management'. The main content area is titled 'Upcoming Table Bookings' and displays a table with two rows of booking information:

Name	Email	Phone	Date and Time	People	Branch	Message	status	Actions
Nisha	romero@gmail.com	08777	2024-09-01 14:00:02	8	Kandy	I need a beach view.	PENDING	<button>Accept</button> <button>Decline</button>
niranjani	nishthooya@gmail.com	0772215592	2024-08-29 14:00:55	8	Kandy	scscscsc	PENDING	<button>Accept</button> <button>Decline</button>

The status bar at the bottom shows '29°C Light rain' and the date '01/09/2024'.

Can be seen the food orders made by customers.

The screenshot shows a Windows desktop environment with a browser window open to 'localhost/ABCRestaurant/admin/orders.php'. The page title is 'Restaurant Management'. On the left, a sidebar menu under 'COOL ADMIN' includes 'Dashboard', 'Bookings', 'Orders' (selected), 'Inquiries', 'Customers', 'Reports', 'Menu Items', 'Gallery', and 'User Management'. The main content area is titled 'Orders Table' and displays a table with one row of order information:

Order	Name	Email	Phone	Date and Time	Address	Pay by	Message	status	Actions
1	niranjani	nishthooya@gmail.com	08777	2024-08-29 13:55:55	08777			PENDING	<button>View</button>

The status bar at the bottom shows 'Rain to stop' and the date '01/09/2024'.

Inquiries

Reg	Name
7	niranjani
8	Nisha
9	Nisha
10	Nisha
11	Nisha

Inquiry: 7 For niranjani

Name	niranjani
Email	nishthooya@gmail.com
Phone	0772215592
Message	i need a table
Reply	yes sure

Phone

0772215592	<button>View</button>
08777	<button>View</button>

## Restaurant staff's dashboard

User

### Upcoming Table Bookings

Name	Email	Phone	Date and Time	People	Branch	Message	status	Actions
Nisha	romero@gmail.com	08777	2024-09-01 14:00:02	8	Kandy	I need a beach view.	PENDING	<button>Accept</button> <button>Decline</button>
niranjani	nishthooya@gmail.com	0772215592	2024-08-29 14:00:55	8	Kandy	scscscsc	PENDING	<button>Accept</button> <button>Decline</button>

## Database Schema for ABC Restaurant

### Tables

#### 1. Customers

- customerID (primary key, auto-increment)
- firstName
- lastName
- email
- phoneNumber
- address

#### 2. Orders

- orderId (primary key, auto-increment)
- customerId (foreign key referencing Customers.customerId)
- orderDate
- orderStatus (e.g., pending, processing, completed)
- totalPrice

#### 3. OrderItems

- orderItemId (primary key, auto-increment)
- orderId (foreign key referencing Orders.orderId)
- menuItemId (foreign key referencing MenuItems.menuItemId)
- quantity
- price

#### 4. MenuItems

- menuItemId (primary key, auto-increment)
- menuId (foreign key referencing Menus.menuId)
- name

- description
- price
- image

## 5. Menus

- menuId (primary key, auto-increment)
- name
- description

## 6. Staff

- staffId (primary key, auto-increment)
- name
- role (e.g., manager, waiter, chef)
- contactNumber

## 7. Reservations

- reservationId (primary key, auto-increment)
- customerId (foreign key referencing Customers.customerId)
- reservationDate
- reservationTime
- numberOfGuests
- tableNumber

## Relationships

- **One-to-Many**
  - A customer can have many orders.
  - A menu can have many menu items.
  - An order can have many order items.
  - A staff member can handle many orders.

- **Many-to-Many**
  - A customer can make many reservations.
  - A reservation can be for multiple guests.

## Keys

- **Primary keys:** Unique identifiers for each row in a table.
- **Foreign keys:** References to primary keys in other tables, establishing relationships between entities.

**The database schema for ABC Restaurant effectively supports the business logic by**

### **Representing key entities**

The tables in the schema represent the core entities of the restaurant system, such as customers, orders, menu items, staff, and reservations. This provides a structured way to store and manage data.

### **Establishing relationships**

The foreign key relationships between tables define the connections and dependencies between entities. For example, the customerId foreign key in the Orders table establishes a relationship between orders and customers, allowing you to retrieve all orders for a specific customer.

### **Enforcing data integrity**

The use of primary and foreign keys ensures data consistency and prevents invalid data from being entered into the system.

## Supporting queries and reports

The well-designed schema facilitates efficient querying and reporting. For example, you can easily retrieve all orders for a specific date range, calculate total sales, or analyze customer preferences.

## Adaptability

The schema can be easily extended to accommodate new features or changes in business requirements. For example, if the restaurant decides to introduce a loyalty program, you can add a new table to store customer points and rewards.

## Normalization

The schema can be normalized to reduce data redundancy and improve data integrity. Normalization involves dividing large tables into smaller, more focused tables, which can improve performance and prevent data anomalies.

**Database –phpMyAdmin 127.0.0.1 has been included all the necessary tables .**

The screenshot shows the phpMyAdmin interface for a database named 'abcrestaurant'. The left sidebar lists various databases and the current one selected. The main area shows the structure of the 'abcrestaurant' database, specifically the list of tables. Below the table list, there is a summary of the database's size and overhead. At the bottom, a 'Create new table' dialog is open, allowing the user to define a new table's name and number of columns.

Table	Action	Rows	Type	Collation	Size	Overhead
booking	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	16.0 Kib	-
contact	Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_general_ci	16.0 Kib	-
customer	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	16.0 Kib	-
food_items	Browse Structure Search Insert Empty Drop	9	InnoDB	utf8mb4_general_ci	16.0 Kib	-
gallery	Browse Structure Search Insert Empty Drop	9	InnoDB	utf8mb4_general_ci	16.0 Kib	-
login	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	16.0 Kib	-
order_details	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	16.0 Kib	-
order_main	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	16.0 Kib	-
register	Browse Structure Search Insert Empty Drop	8	InnoDB	utf8mb4_general_ci	16.0 Kib	-
<b>9 tables</b>	<b>Sum</b>	<b>31</b>	<b>InnoDB</b>	<b>utf8mb4_general_ci</b>	<b>144.0 Kib</b>	<b>0 B</b>

**Note1:** Considering the word count, I hope to explain the each and every functionality of the admin dashboard on my viva session.

**NOTE2:** I have attached the project in separately in the zip file.

## Testing

### TEST PLAN

#### Project Information

Project Name	ABC Restaurant
Project Description	A restaurant management system that allows staff to manage orders, reservations, menus, and customer information.
Project Manager	N.Niranjani
Release Number/ Version	Version 1
Date of Test Plan Creation	2024-09-01
Date of Last Update	2024-09-01

#### FEATURES TO BE TESTED

A list of features to be tested across user journeys can be seen here –

<http://localhost/ABCRestaurant/index.php>

#### TEST LEVELS AND TYPES

- Test Levels

- Unit Testing
  - Feature Testing
  - System Testing
- Test Types
    - Smoke Testing
    - Functional Testing
    - Order Management (create, update, cancel orders)
    - Reservation Management (create, update, cancel reservations)
    - Menu Management (add, edit, delete menu items)
    - Customer Management (add, edit, view customer information)
    - Reporting (generate reports on sales, orders, reservations)
    - Usability Testing
    - Security Testing
    - Regression Testing

## **TEST ENVIRONMENT**

- Testing Link
- Laptop and Internet connection
- Browser - Chrome
- OS types - Mac, Windows, Android, iOS

## **ASSUMPTIONS**

- Developers have completed coding for the features to be tested.
- Testers have access to necessary test data and documentation (e.g., user stories, API references).

## **EXIT CRITERIA**

- Test execution is complete and bugs have been reported.
- No functional bugs.
- 80% of the bugs reported have been solved.

## **Test Environment**

IDE: Eclipse

Programming Language: Java

Testing Framework: Selenium TestNG

Selenium WebDriver: ChromeDriver

## **Test Cases Executed- 7**

### **Test Results**

Passed: 6

Failed: 1(invalid credentials)

### **Failed Test Case**

When try to test the Login page with the invalid credentials it doesn't give the access to login. To prevent the unauthorized access it doesn't provide the access. Even though the test case fails, the actual and expected results are indeed the same.

### **Test cases**

Test (Automation) has been to 6 test cases.

eclipse-workspace - Automation\_Framework/src/test/java/BaseClass/BaseClass.java - Eclipse IDE

```

File Edit Source Refactor Source Navigate Search Project Run Window Help
Package Explorer TestGalleryjava testng.xml BaseClass.java
1 package BaseClass;
2
3 import java.awt.Desktop;
4
5 public class BaseClass {
6     static ReadConfig readconfig = new ReadConfig();
7     public static String webUrl = readconfig.getUrl();
8     public static String email = readconfig.getUsername();
9     public static String password = readconfig.getPassword();
10    public static WebDriver driver;
11    public static Logger logger;
12    public static WebDriverWait wait;
13    public static ExtentReports extent;
14    public static ExtentSparkReporter spark;
15    public static ExtentTest test;
16    public static ExtentTest node;
17    public static ExtentTest childNode;
18
19    @BeforeSuite
20    public void startReporting() {
21        Report.initReports();
22        Report.getExtentReports().setSystemInfo("Browser Name", "Chrome");
23        Report.getExtentReports().setSystemInfo("Application URL", webUrl);
24    }
25
26    @Parameters("browser")
27    @BeforeClass
28    public void setup(String browser) throws MalformedURLException {
29        if (browser.equals("chrome")) {
30            System.setProperty("webdriver.chrome.driver", readconfig.getChromePath());
31            driver = new ChromeDriver();
32            driver.manage().window().maximize();
33            driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
34        } else if (browser.equals("firefox")) {
35            System.setProperty("webdriver.gecko.driver", readconfig.getFirefoxPath());
36            driver = new FirefoxDriver();
37            driver.manage().window().maximize();
38        }
39    }
40}

```

eclipse-workspace - Automation\_Framework/testng.xml - Eclipse IDE

```

File Edit Source Refactor Source Navigate Search Project Run Window Help
Package Explorer TestGalleryjava testng.xml
1 http://testing.org/testng-1.0.dtd (doctype)
2 <?xml version="1.0" encoding="UTF-8"?>
3 <!DOCTYPE suite SYSTEM "http://testing.org/testng-1.0.dtd">
4 <suite name="test">
5     <test name="test login">
6         <parameter name="browser" value="chrome"/>
7         <classes>
8             <class name="TestCases.Testlogin"/>
9             <class name="TestCases.TestHomePage"/>
10            <class name="TestCases.TestBookTable"/>
11            <class name="TestCases.TestContactUsPage"/>
12            <class name="TestCases.TestLoginPage"/>
13            <class name="TestCases.TestGallery"/>
14        </classes>
15    </test>
16 </suite>

```

File Edit Source Refactor Source Navigate Project Run Window Help

Package Explorer X

```

1 package TestCases;
2
3 *import org.openqa.selenium.By;*
4
5 Run All
6
7 public class TestBookTable extends BaseClass {
8     public static String LandingPageURL = "http://localhost/ABCRestaurant/index.php";
9     public static String Name = "abc";
10    public static String EmailAddress = "abc@gmail.com";
11    public static String NoofPeople = "10";
12    public static String phonenumber = "0772676933";
13    public static String Location = "Colombo";
14    public static String Message = "Should be clean";
15
16
17 @Test(priority = 1)
18 Run | Debug
19 public void verifyBookTable() throws InterruptedException{
20     Report.createTest("Book a Table Test");
21     LoginPage login = new LoginPage(driver);
22
23     login.verifybookTable(Name,EmailAddress,NoofPeople,phonenumber,Location,Message);
24     driver.quit();
25 }
26
27
28
29
30
31
32 }
```

src/main/java  
 com.Automation\_Framework  
 src/test/java  
 BaseClass  
 PageObjects  
 TestCases  
 TestBookTable.java  
 TestBookTableF  
 Emailaddress  
 LandingPageURL  
 Location  
 Message  
 Name  
 NoofPeople  
 phonenumber  
 verifyBookTable():void  
 TestContactUsPage.java  
 TestGallery.java  
 TestHomePage.java  
 TestLoginPage.java  
 TestMenuPage.java  
 Utilities  
 ExcelUtils.java  
 ReadConfig.java

Below screen shot helps to justify the TDD. Verify successful login with valid credentials and unsuccessful login with invalid credentials.

eclipse-workspace - Automation\_Framework\src\test\java\testCases\TestLogin.java - Eclipse IDE

File Edit Source Refactor Source Navigate Project Run Window Help

Package Explorer X

```

1 package TestCases;
2
3 *import org.openqa.selenium.By;*
4
5 Run All
6
7 public class TestLogin extends BaseClass {
8     public static String LandingPageURL = "http://localhost/ABCRestaurant/signup.php";
9     public static String errorMessagePath = "/body/div[@id='app']/div[1]/div[1]/div[2]/div[1]/div[1]";
10
11 @TestdataProvider(name="LoginDataFromExcel")
12 public Object[][] getData(){
13     Properties prop = new Properties();
14     String excelPath = System.getProperty("user.dir") + "/Data/ExcelData.xlsx";
15     String sheetName = "Sheet1";
16     ExcelUtils excelUtils = new ExcelUtils(excelPath, sheetName);
17     return excelUtils.getData();
18 }
19
20 @Test(dataProvider = "LoginDataFromExcel", priority = 1)
21 Run | Debug
22 public void verifyLogin(String username, String password) throws InterruptedException{
23     LoginPage login = new LoginPage(driver);
24     login.enterUsername(username);
25     login.enterPassword(password);
26     login.clickLoginBtn();
27
28     if (!isValidCredentials(username, password)) {
29         Assert.assertEquals(driver.getCurrentUrl(), LandingPageURL, "Login failed for valid credentials");
30     } else {
31         String errorMessage = driver.findElement(By.xpath("//div[@class='errorText']"));
32         Assert.assertTrue(errorMessage.getText().contains("Invalid credentials"), "Error message not displayed for invalid credentials");
33     }
34
35     driver.quit();
36 }
37
38 private boolean isValidCredentials(String username, String password) {
39     return "Admin".equals(username) && "admin123".equals(password);
40 }
```

src/main/java  
 com.Automation\_Framework  
 src/test/java  
 BaseClass  
 TestCases  
 TestBookTable.java  
 TestContactUsPage.java  
 TestGallery.java  
 TestHomePage.java  
 TestLoginPage.java  
 TestLogin.java  
 TestMenuPage.java  
 Utilities  
 ExcelUtils.java  
 ReadConfig.java  
 Report.java  
 JRE System Library [JavaSE-1.8]  
 Maven Dependencies  
 Configurations  
 config properties  
 Data  
 ExcelData.xlsx  
 Screenshots  
 src  
 main  
 test  
 target  
 test-output  
 index.html  
 pom.xml  
 testng.xml

Functional report of test cases .According to that 6 were pass and one 1 was fail.

The screenshot shows a 'Functional Test Report' interface. On the left, a sidebar lists various test cases: 'Login Test - nishthooya@gmail.com', 'Login Test - invalid', 'Home Option Test', 'Book a Table Test', 'Contact Us Option Test', 'Menu Option Test', and 'Gallery Option Test'. Each entry includes a timestamp and a green circular icon. On the right, a detailed view of the 'Login Test - invalid' test is shown. It has a timestamp of '09:15:54' and a status of 'Failed'. The 'DETAILS' section contains the message 'verifyLogin Got Failed'.

When find out the reason of the failure it shows the reason. (Invalid credentials)

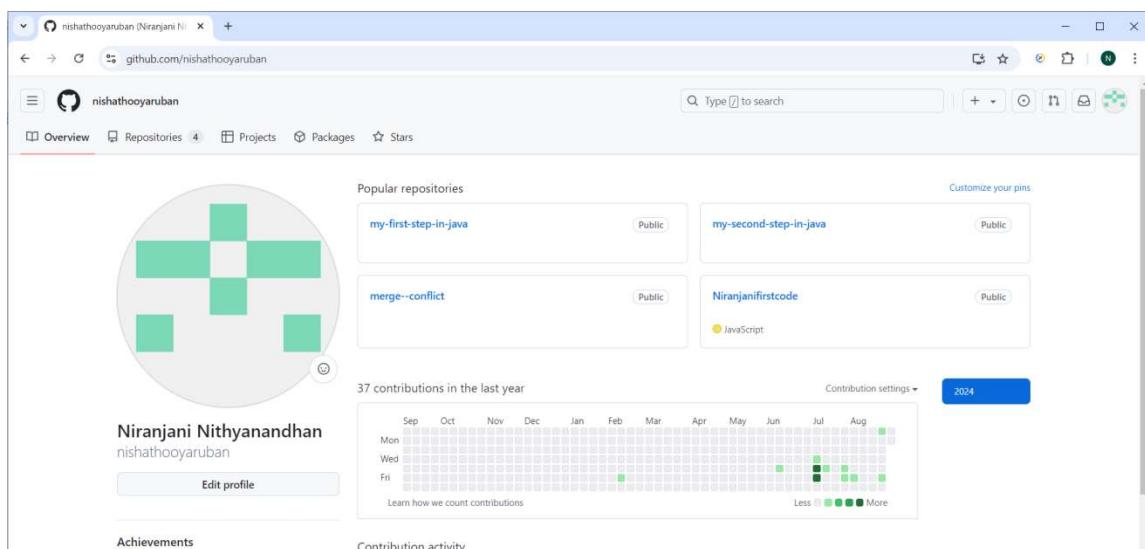
This screenshot is from the same 'Functional Test Report' interface. The 'Login Test - invalid' test is now highlighted with a red background and a red circular icon, indicating it failed. The 'DETAILS' section displays a detailed error message:

```
org.openqa.selenium.NoSuchSessionException: Session ID is null. Using WebDriver after calling quit()
Build info: version: '4.21.0', revision: '79ad462ef4'
System info: os.name: 'Windows 10', os.arch: 'amd64', os.version: '10.0', java.version: '22.0.2'
Driver info: org.openqa.selenium.chrome.ChromeDriver
Command: [null, findElement (value/input[@id='first'], usingxpath)]
Capabilities {acceptInsecureCerts: false, browserName: chrome, browserVersion: 128.0.6613.114, chrome: {chromedriverVersion: 128.0.6613.84 (606aa55c7d68..., userDataDir: C:\Users\Nisha\AppData\Loca...), fe: cdm:accounts: true, goog:chromeOptions: {debuggerAddress: localhost:57797}, networkConnectionEnabled: false, pageLoadStrategy: normal, platformName: windows, proxy: Proxy(), se:cdp: ws://localhost:57797/dvtoo..., se:cdpVersion: 128.0.6613.114, setWindowRect: true, strictFileInteractability: false, timeouts: {implicit: 0, pageLoad: 300000, script: 30000}, unhandledPromptBehavior: dismiss and notify, webauthn:extension:credBlob: true, webauthn:extension:largeBlob: true, webauthn:extension:minPinLength: true, webauthn:extension:prf: true, webauthn:virtualAuthenticators: true}
at org.openqa.selenium.remote.HttpCommandExecutor.execute(HttpCommandExecutor.java:151)
at org.openqa.selenium.remote.service.DriverCommandExecutor.invokeExecute(DriverCommandExecutor.java:216)
at org.openqa.selenium.remote.service.DriverCommandExecutor.execute(DriverCommandExecutor.java:174)
at org.openqa.selenium.remote.RemoteWebDriver.execute(RemoteWebDriver.java:518)
at org.openqa.selenium.remote.ElementLocation$ElementFinder$2.findElement(ElementLocation.java:165)
```

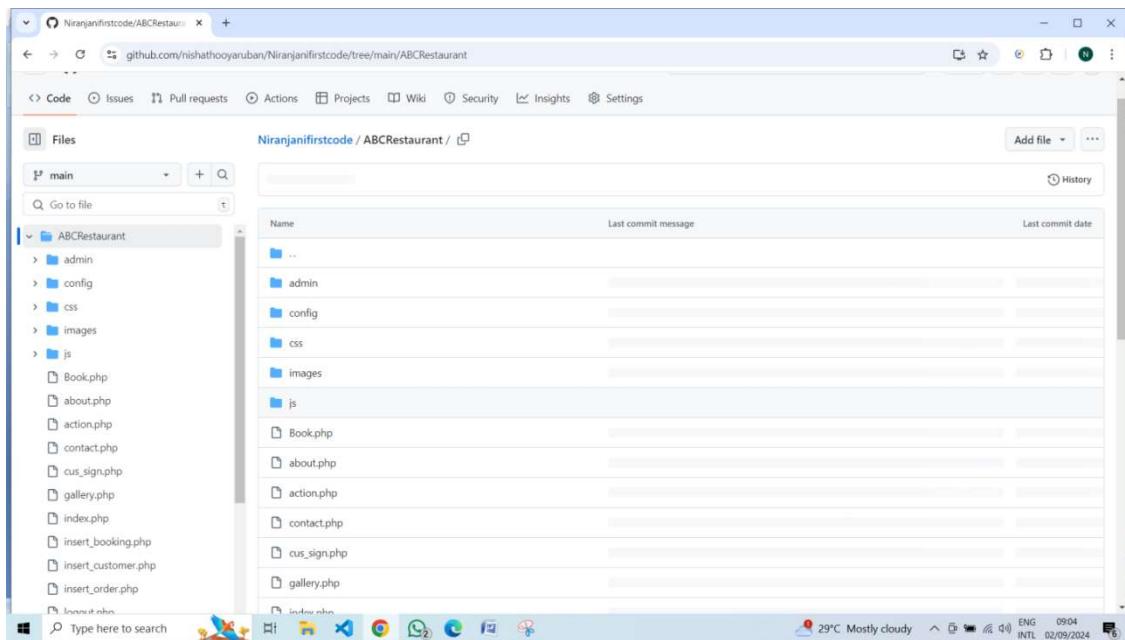
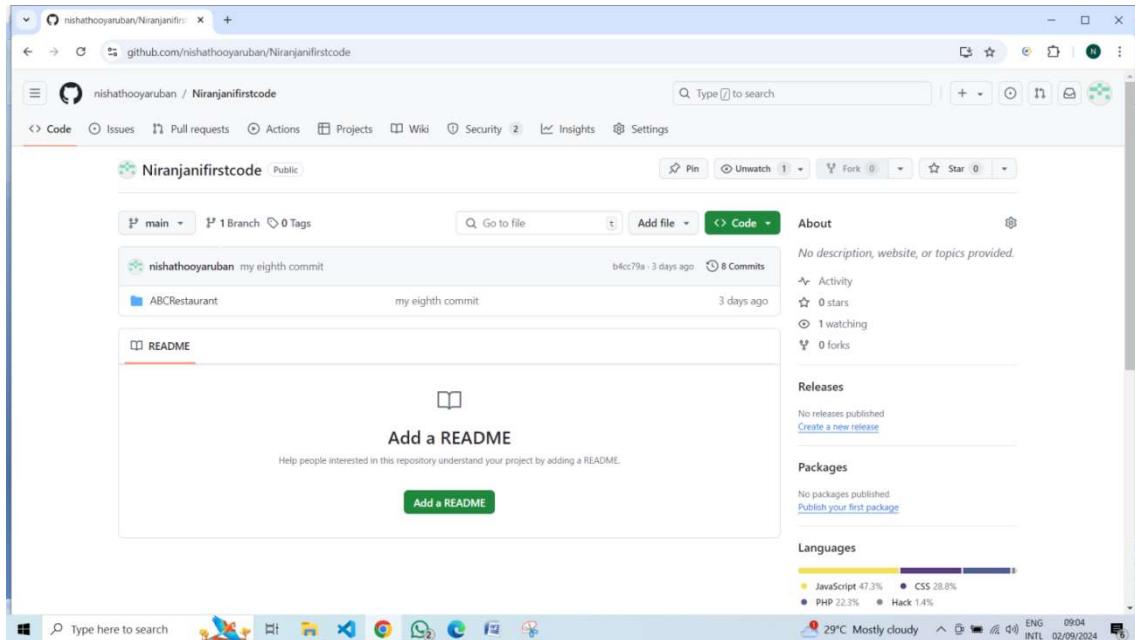
## Final overall report



## Work with github



I have created a repository named “Niranjanifirstcode” .



My initial commit has done with some front end codes

Commit

initial commit

main

nishathooyaruban committed on Jul 18

Showing 1 changed file with 460 additions and 0 deletions.

ABCRestaurant/index.html

```

@@ -0,0 +1,460 @@
1 + <!DOCTYPE html>
2 + <html lang="en">
3 +   <head>
4 +     <!-- basic -->
5 +     <meta charset="utf-8">
6 +     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7 +     <meta name="viewport" content="width=device-width, initial-scale=1">
8 +     <!-- mobile metas -->
9 +     <meta name="viewport" content="width=device-width, initial-scale=1">
10 +    <meta name="viewport" content="initial-scale=1, maximum-scale=1">
11 +    <!-- site metas -->
12 +    <title>Grand Coffee</title>
13 +    <meta name="keywords" content="">
14 +    <meta name="description" content="">
15 +    <meta name="author" content="">
16 +    <!-- bootstrap css -->
17 +    <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
18 +    <!-- style css -->
19 +    <link rel="stylesheet" type="text/css" href="css/style.css">

```

My last commit with some changes...I have engaged with last one month from the date I started the project.

Commits · nishathooyaruban/...

Commits

main

All users · All time

- Commits on Aug 30, 2024
- my eighth commit** · nishathooyaruban committed 3 days ago · b4cc79a
- Commits on Aug 25, 2024
- my seventh commit** · nishathooyaruban committed last week · 8229ebf
- my sixth commit** · nishathooyaruban committed last week · e4fad90
- Commits on Aug 9, 2024
- my fifth commit** · nishathooyaruban committed 3 weeks ago · 6272234
- Commits on Aug 2, 2024
- fourth commit** · nishathooyaruban committed last month · 12bef4e
- Commits on Aug 1, 2024

<https://github.com/nishathooyaruban> -my github link

## References

- Fowler, M. (2003). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.
- Schmidt, D. C., Fayad, M. E., & Johnson, R. E. (2004). *Service-Oriented Architecture: A Conceptual Overview*. Communications of the ACM, 47(6), 29-36. Richardson, M., & Ruby, S. (2007). *RESTful Web Services*. O'Reilly Media.
- Freeman, S., Robson, E., Sierra, K., & Bates, B. (2004). *Head First Design Patterns*. O'Reilly Media.
- Bloch, J. (2008). *Effective Java: Programming Language Guide*. Addison-Wesley Professional.