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:				
Result				
Marks by 1st Assessor		Signature of the 1st Assessor		**Agreed Mark**
Marks by2nd Assessor		Signature of the 2nd Assessor		
Comments on the Agreed Mark.				
For Office use only (hard copy assignments)				
Receipt date		Received by		

Table of contents

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

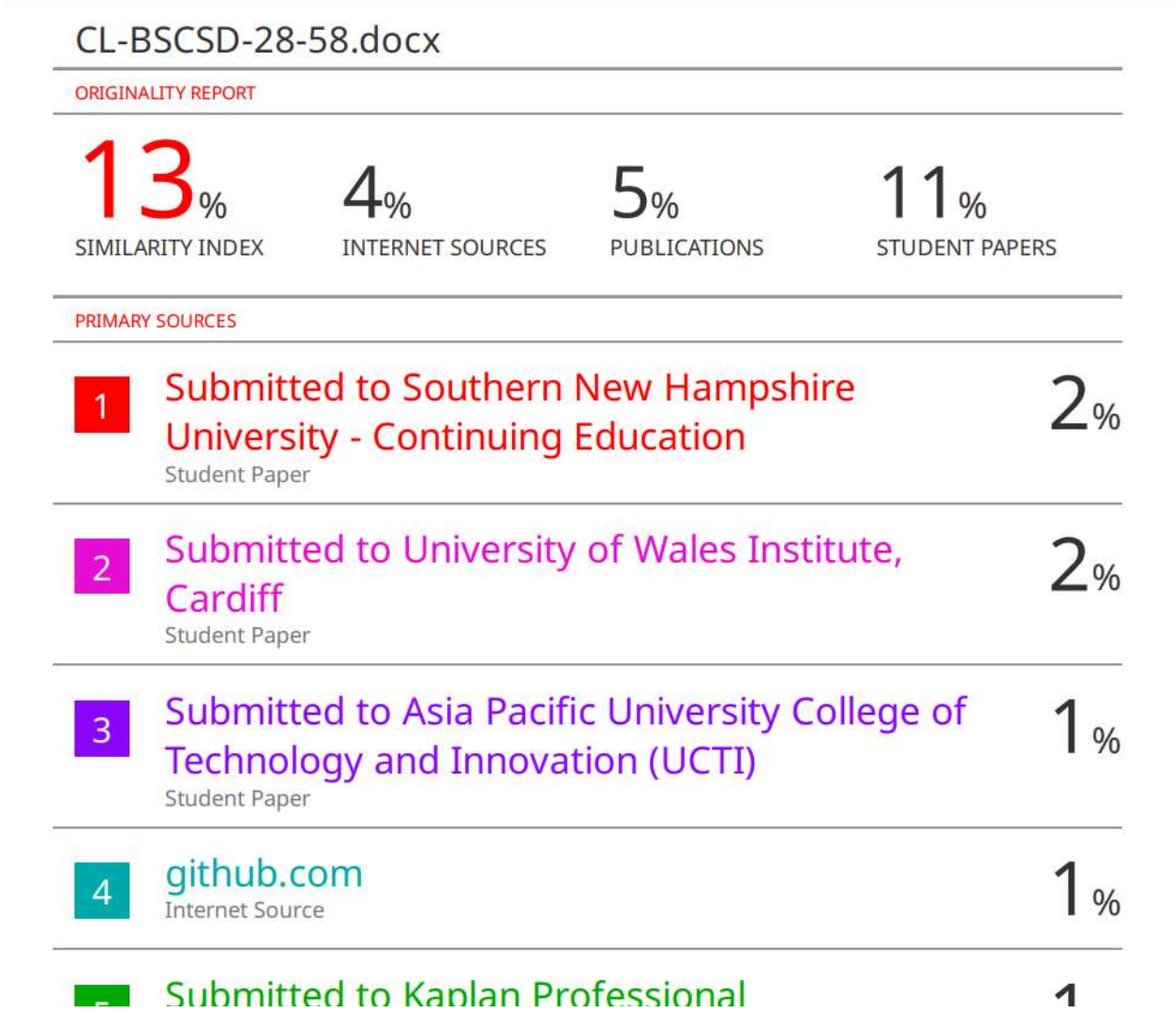
Expected Outcomes

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

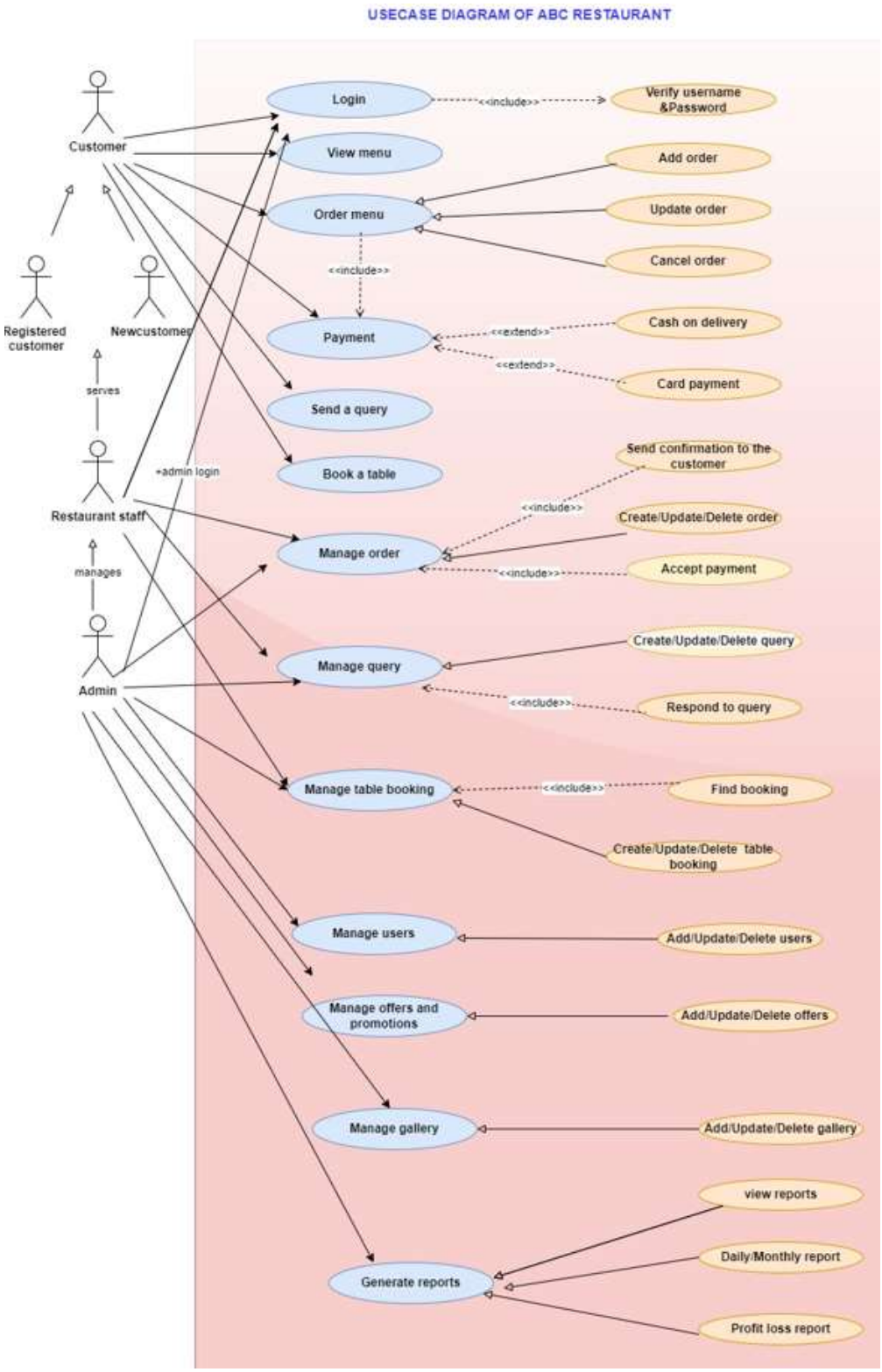
13%	4%	5%	11%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

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

7	Submitted to University of Canterbury Student Paper	1 %
8	Submitted to National College of Ireland Student Paper	1 %
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, Shanping 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

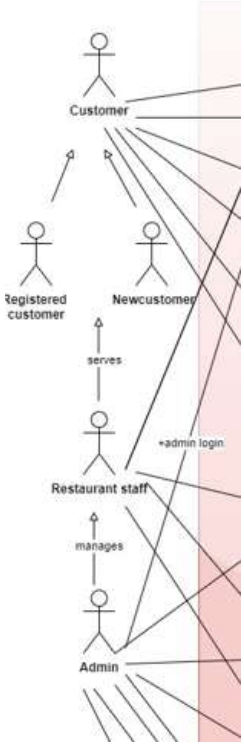
Information Hiding

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

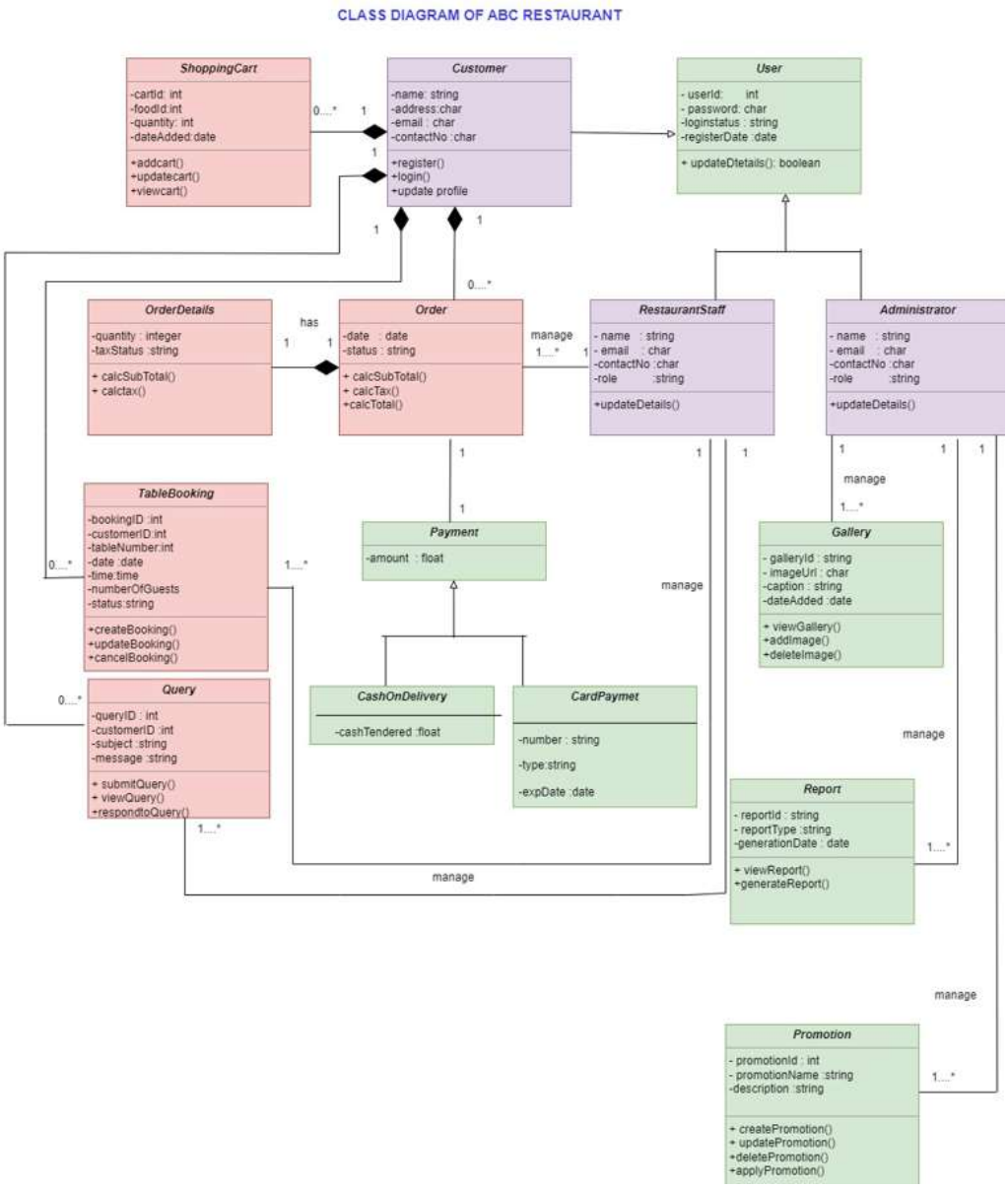
Objects can be treated as instances of their parent class or their own class, allowing for flexible code. For example, a method that takes an Order object as a parameter can work with different types of orders (e.g., dine-in, delivery).

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.

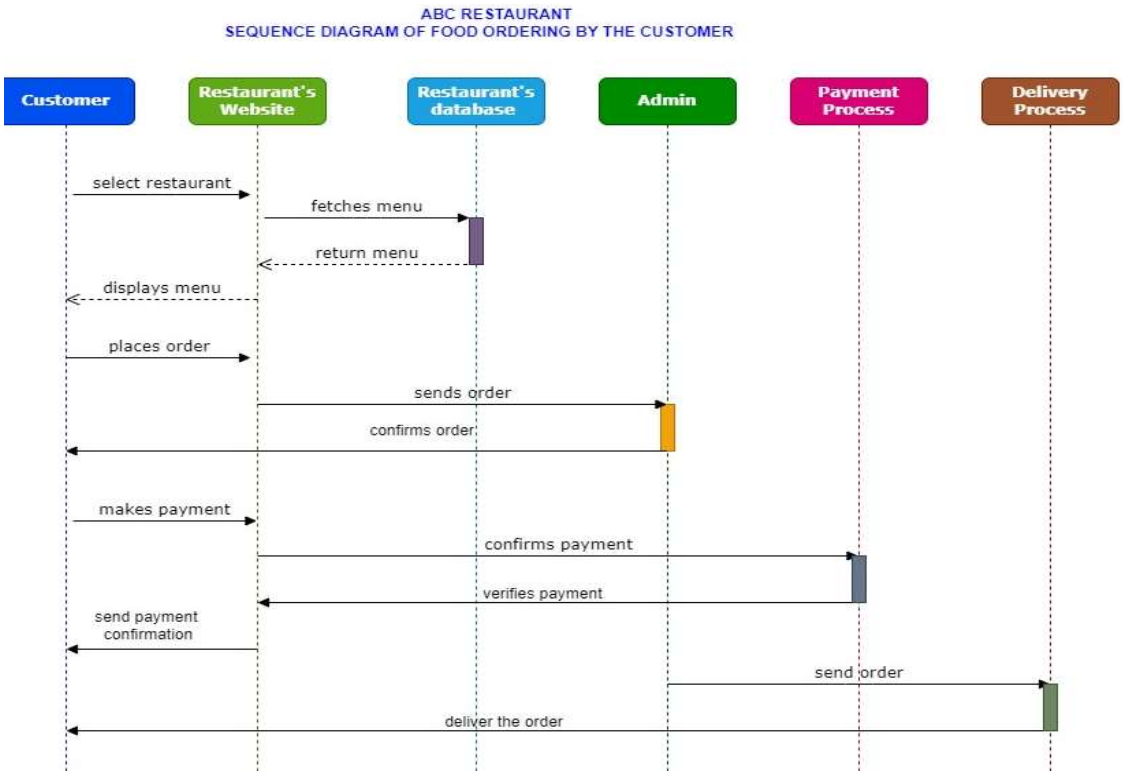
Class-Represents a set of objects with common properties (attributes) and behaviours (operations).

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

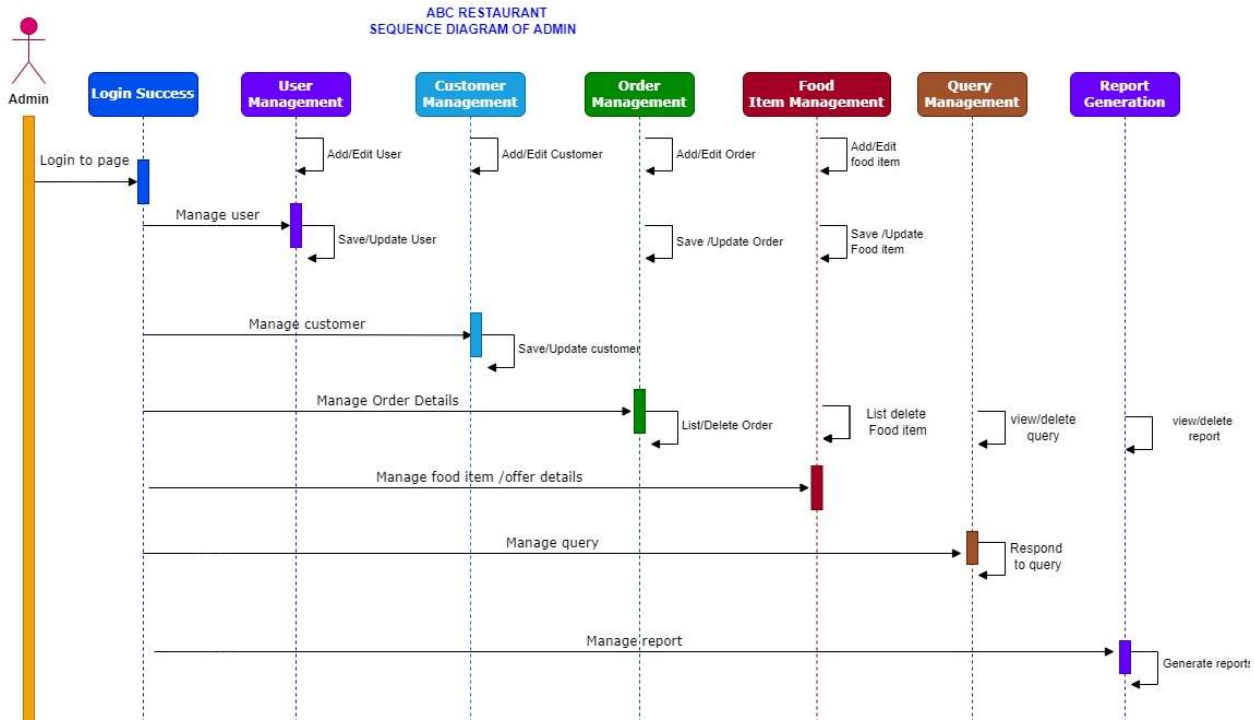
Restaurant prepares the food.

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

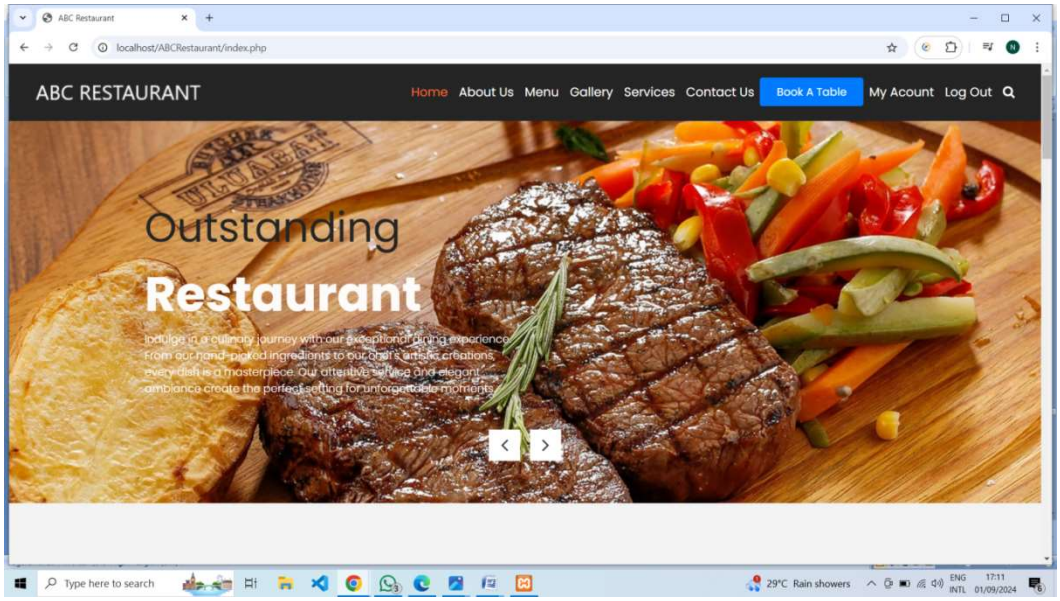
Homepage

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.

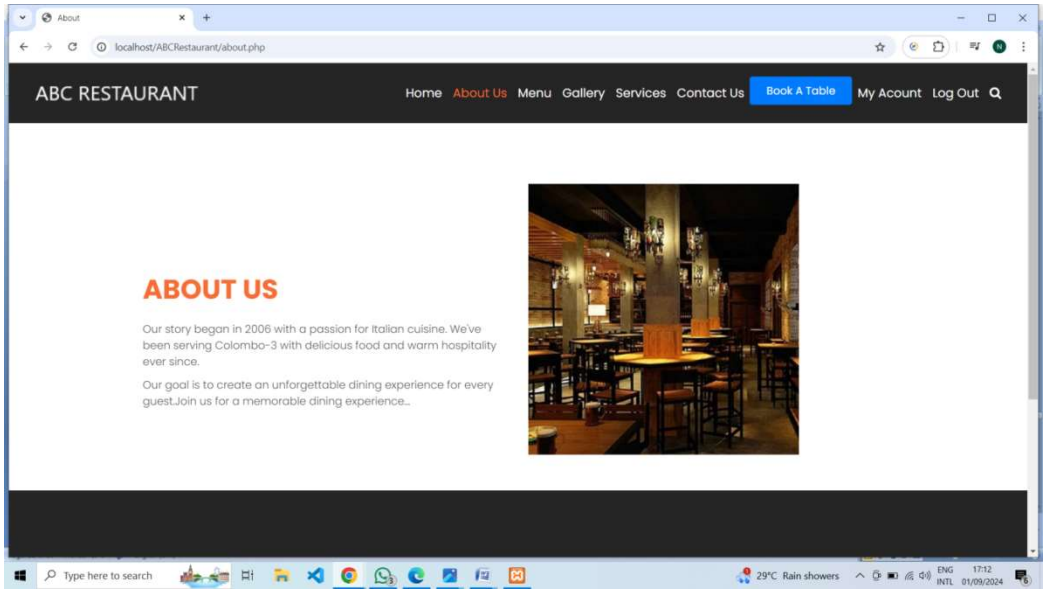
Featured Items -A section highlighting popular or recommended menu items.

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



About us

Small description about ABC Restaurant

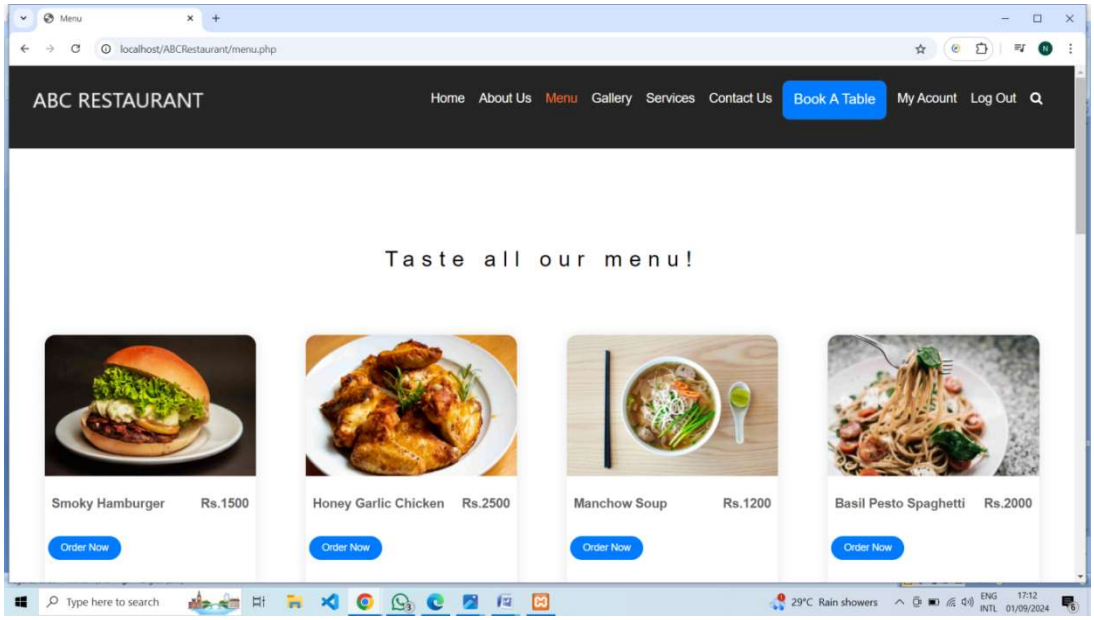


Menu Page

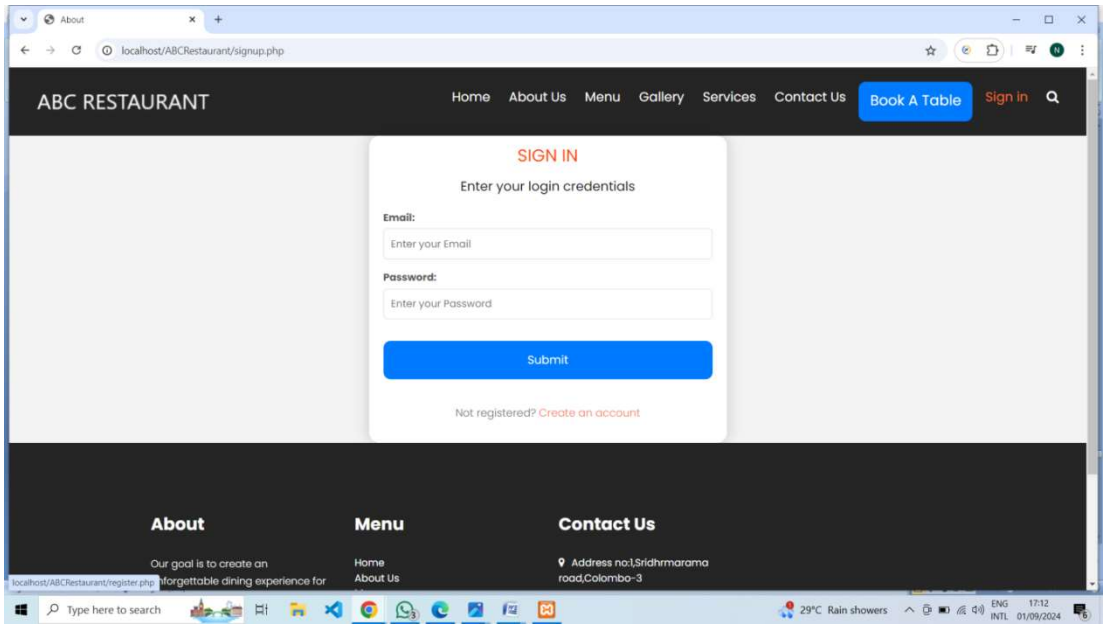
Menu Categories- A list of menu categories (e.g., Appetizers, Main Courses, Desserts).

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

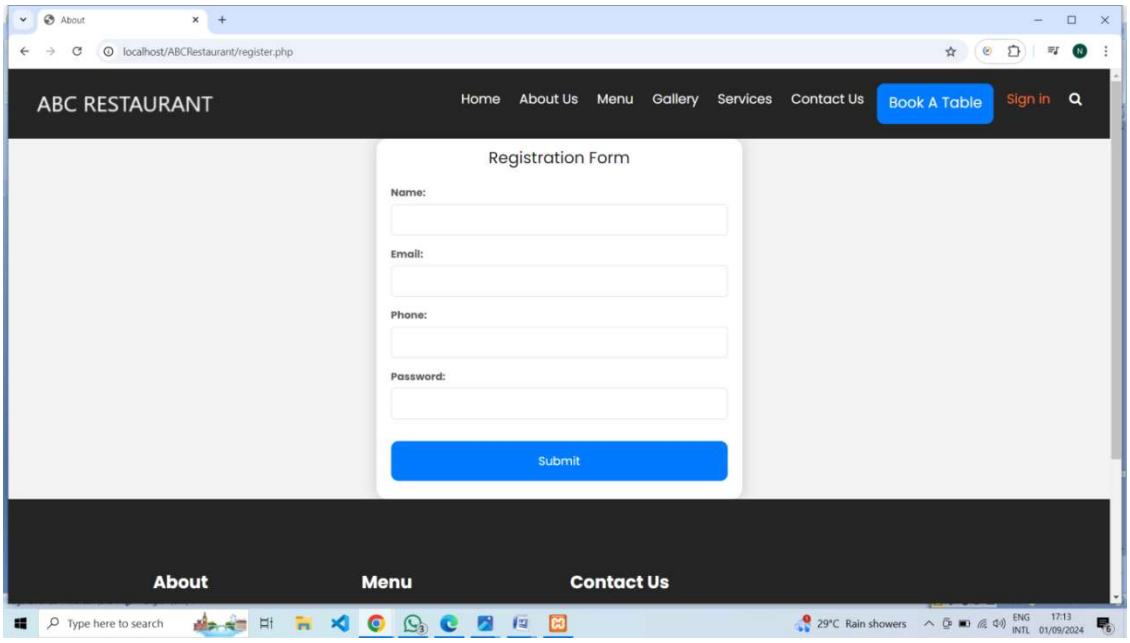
Add to Cart Button- A button for each item to add it to the cart.



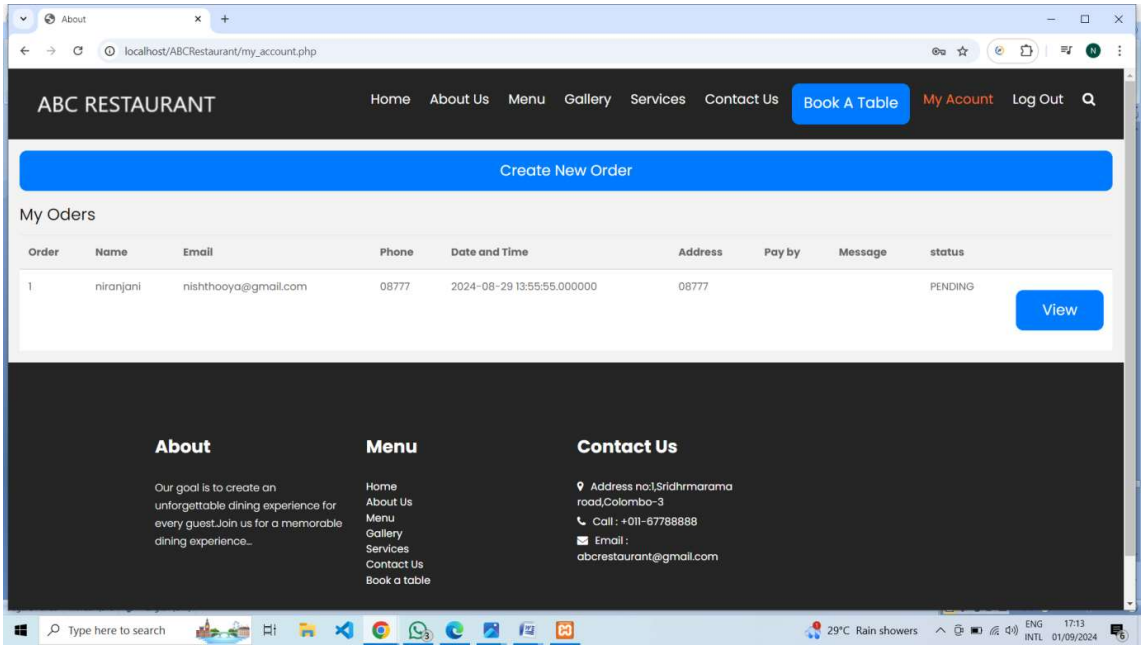
Once you click the order now button it redirects to “sign in” page



If customer doesn’t have an account it advices to create an account redirects to “”registration page”

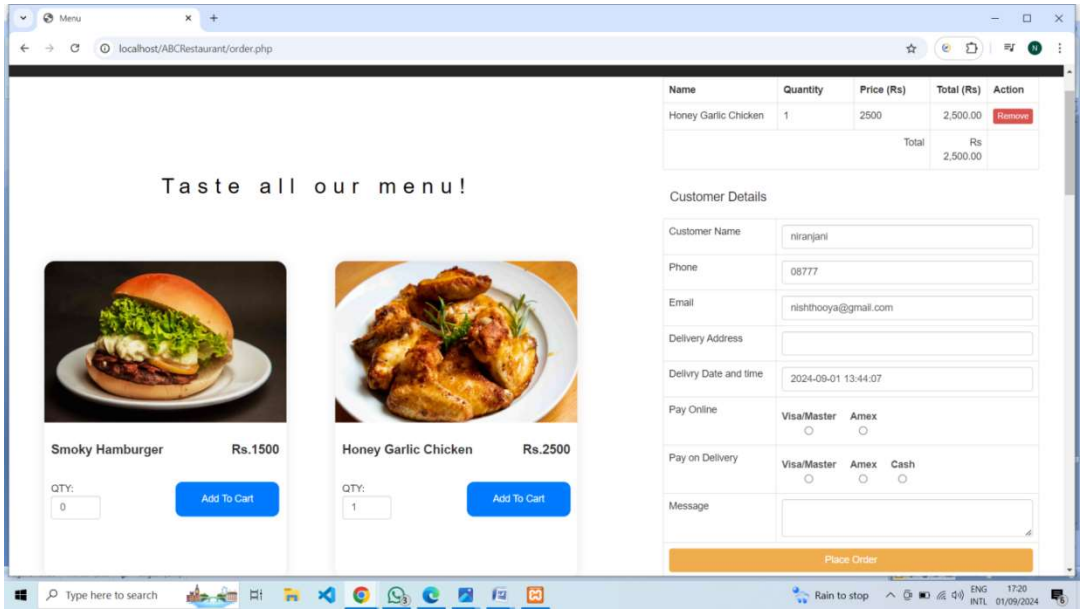


After login success customer can view his previous orders as well as he can able to make new order through this page.

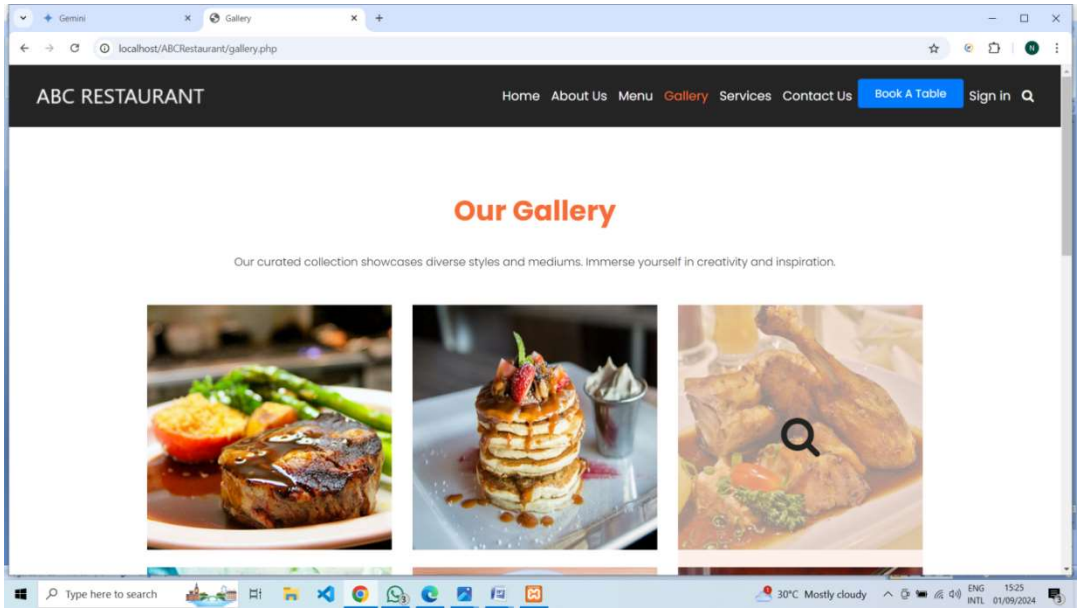


After clicking create an order it redirects to below page from this you can make a new order once you select the desired food item and quantity it will display in a small dialog box from this you

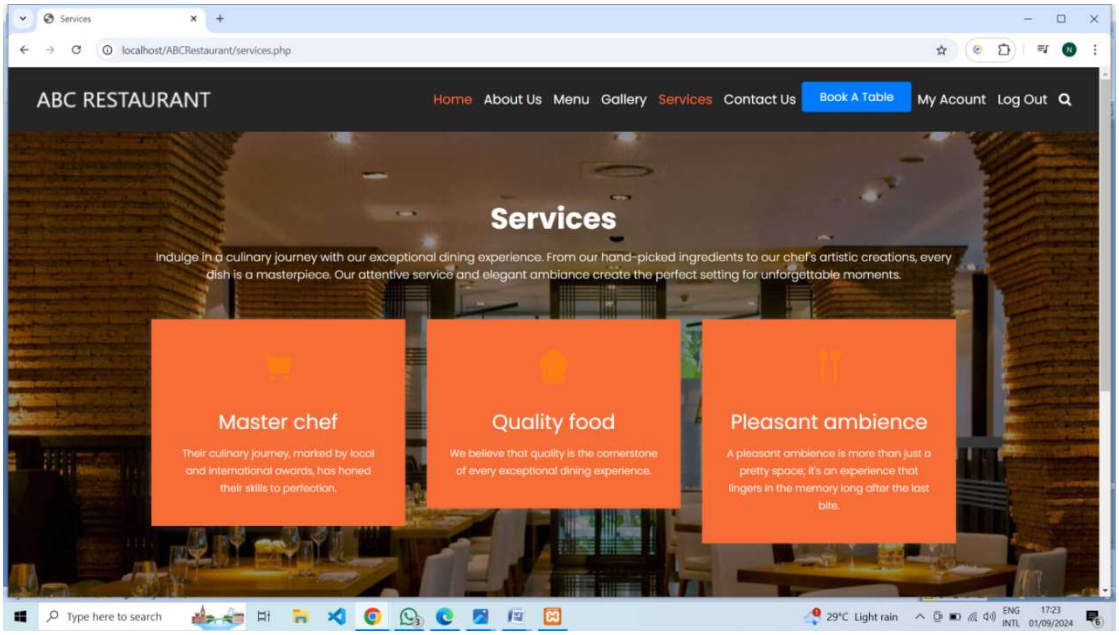
can give the required details and confirm the order. Confirmation message will be sent to the particular customers email. (Since I have run the project via local host I couldn't attached the screen shot of email evidence)



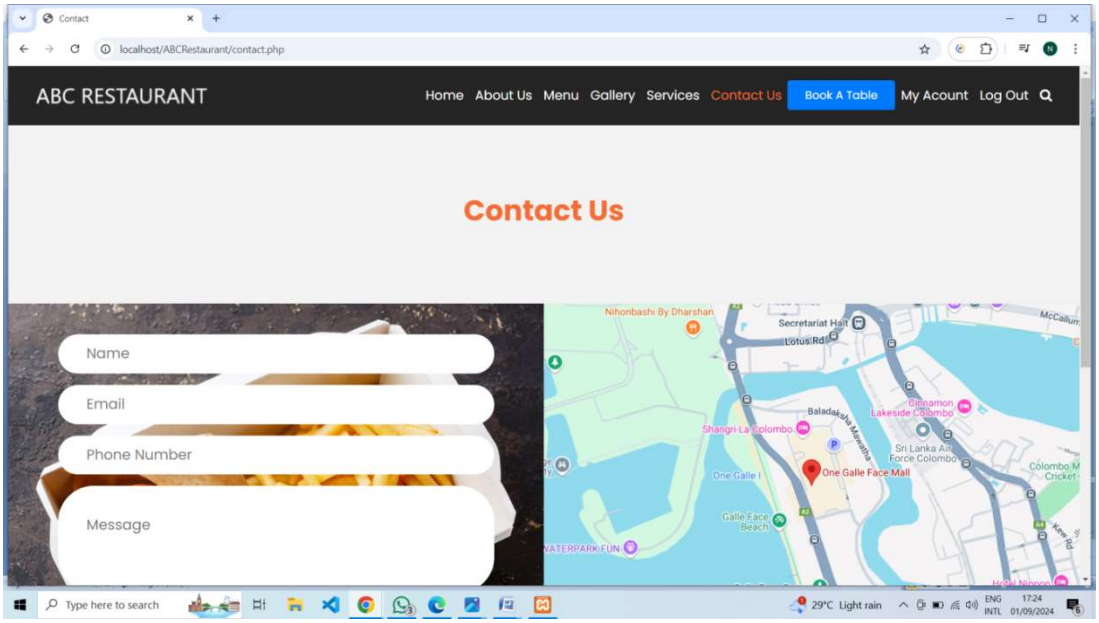
This is the gallery page with some attractive food images.

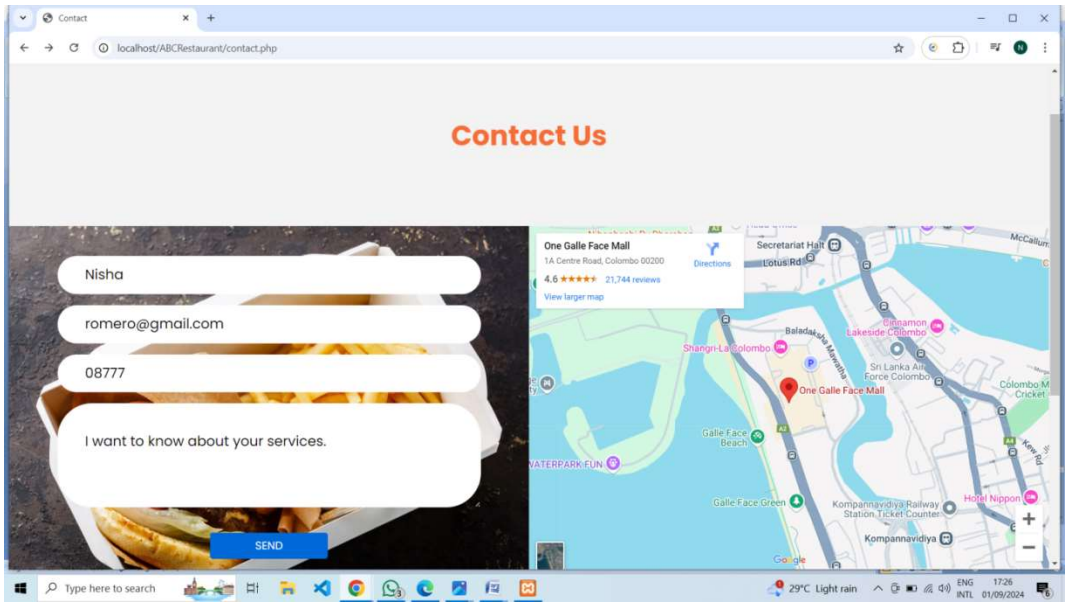


This is the services page with some attractive content to grab the customers easily.

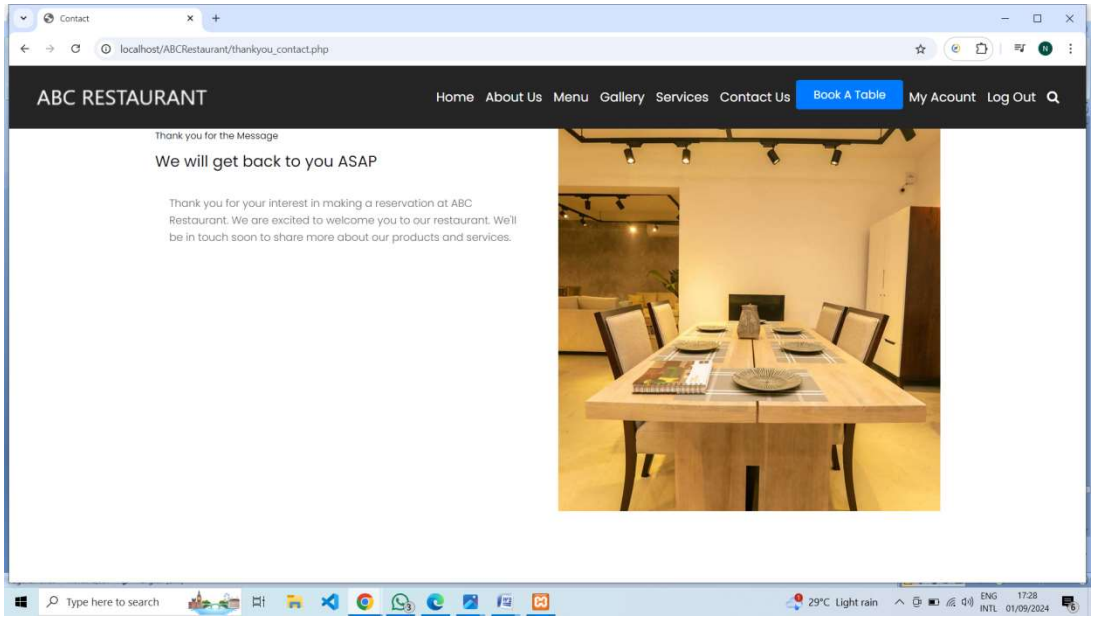


Contact us page with the map of the restaurant

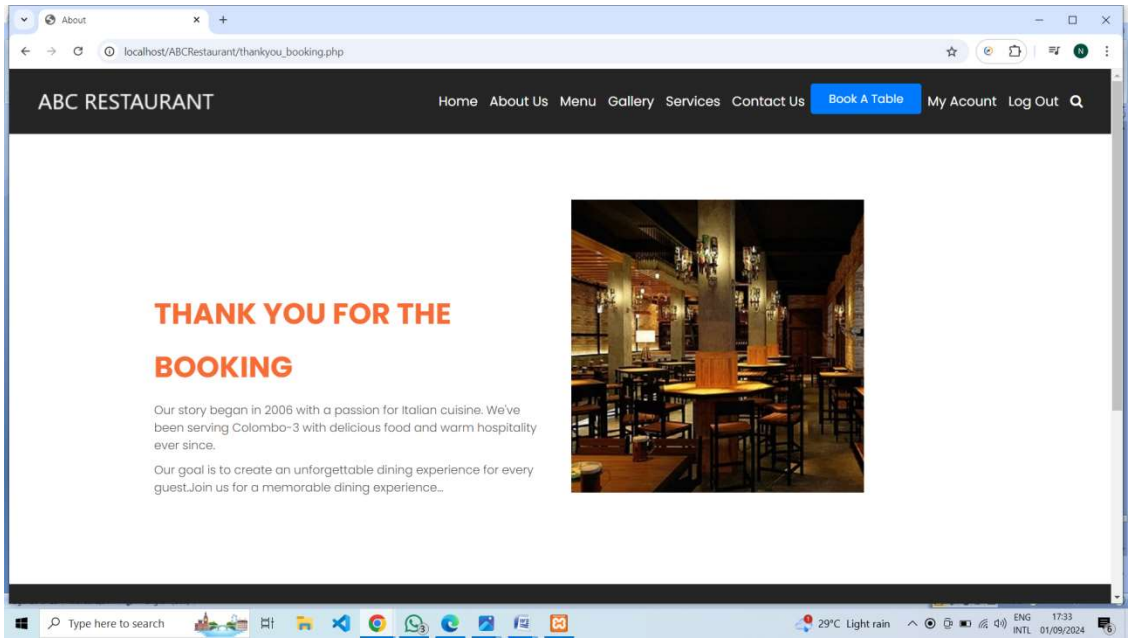




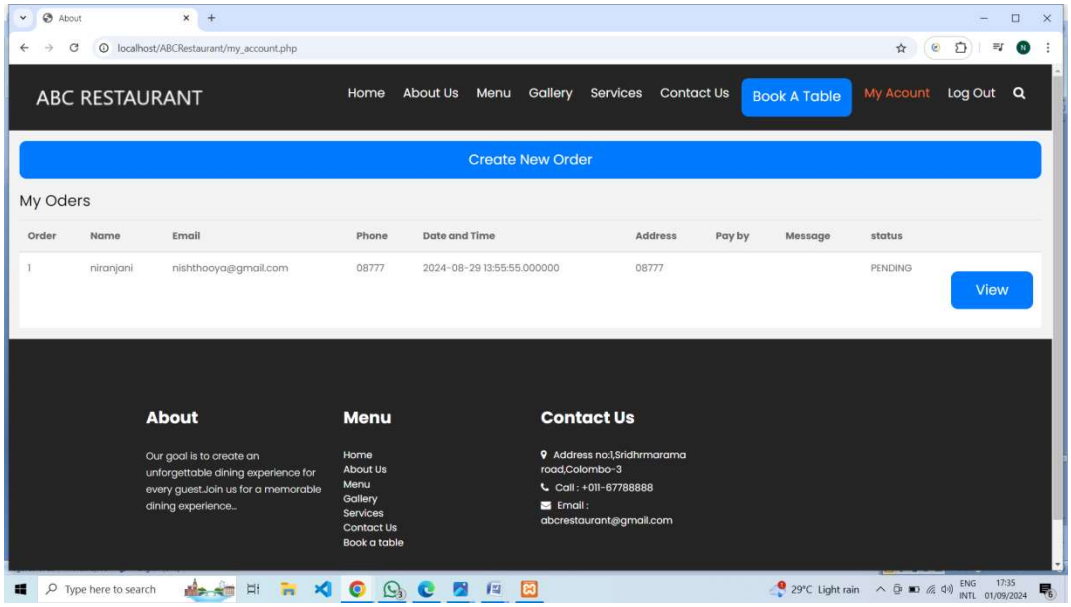
After filled up the contact us form it will be redirected to" thank you message" page



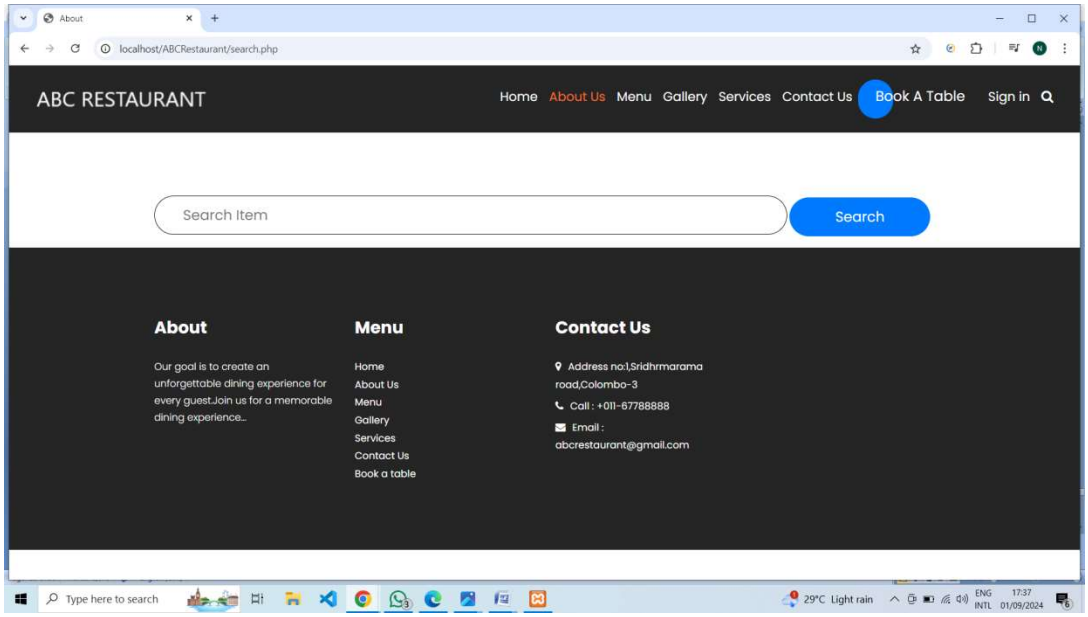
Book a table page customer can make the table reservation.After filled up the booking form customer will get the thankyou message.



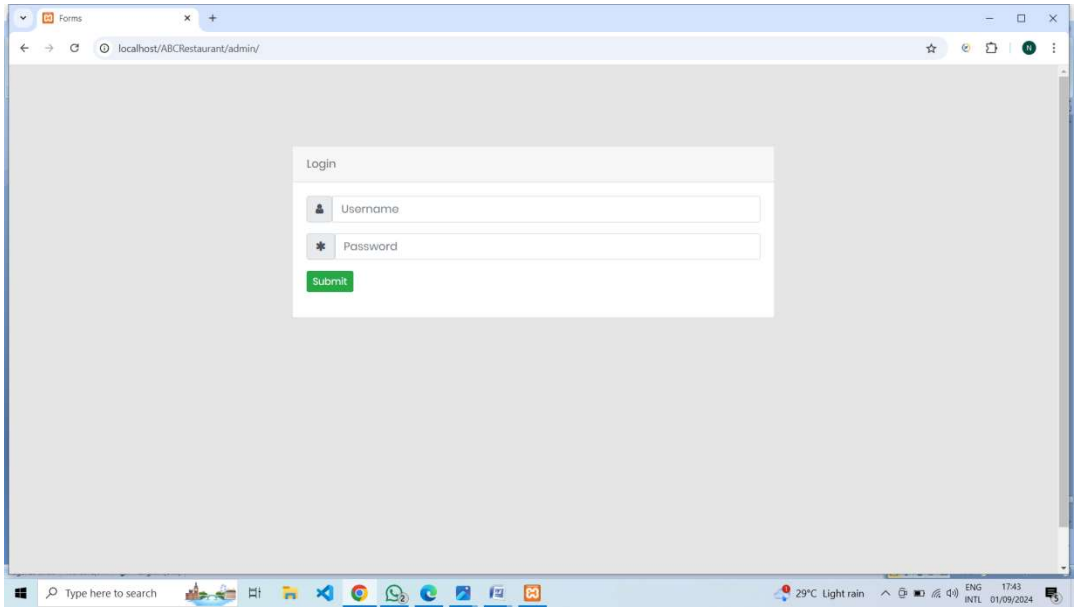
My account-Registered customer can view the order details and history of orders.



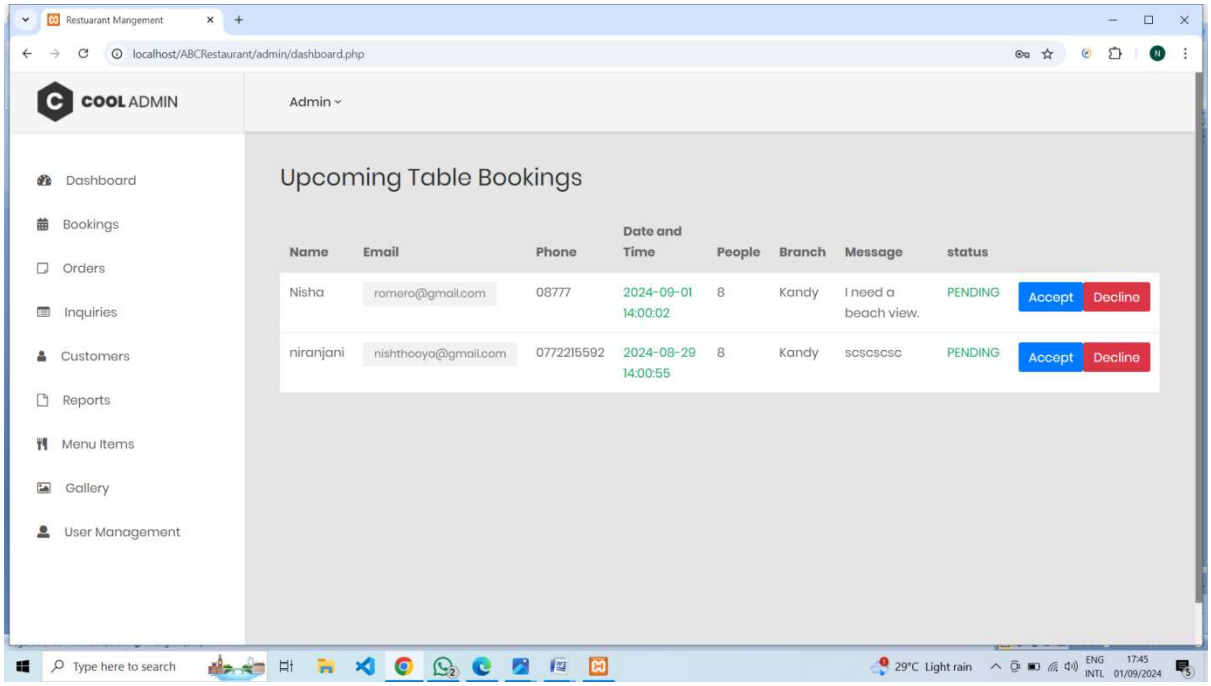
Search page



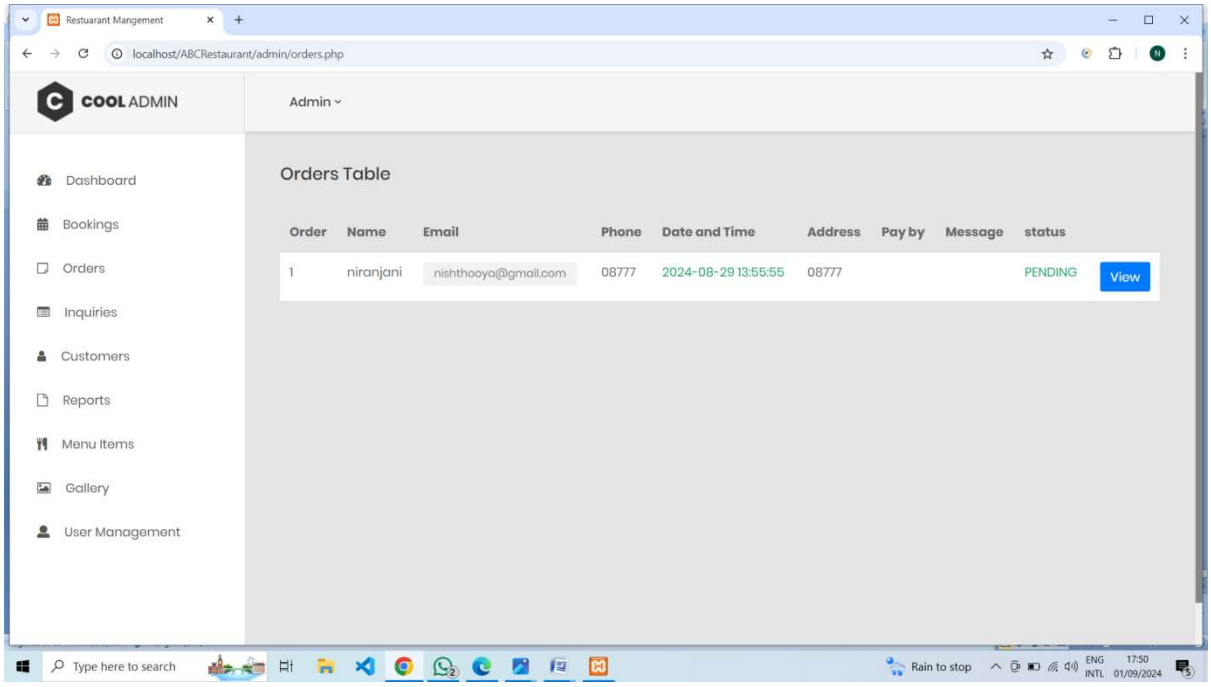
Admin dashboard -Login

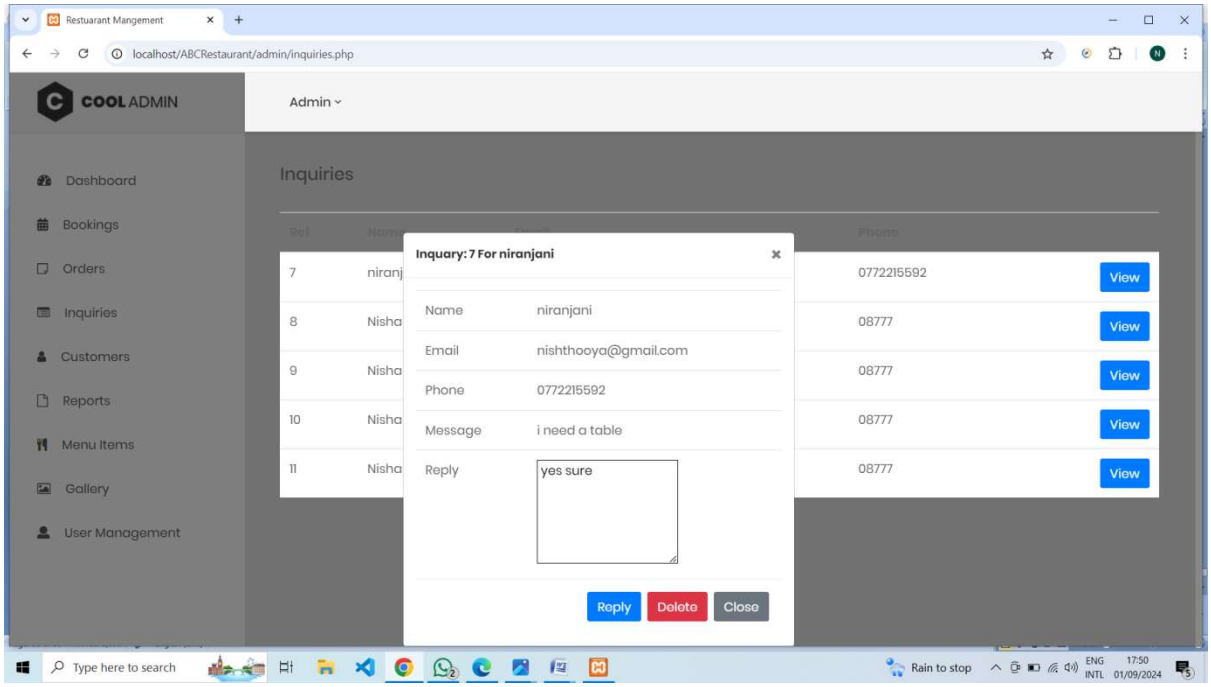


Can be seen the table bookings made by customers.

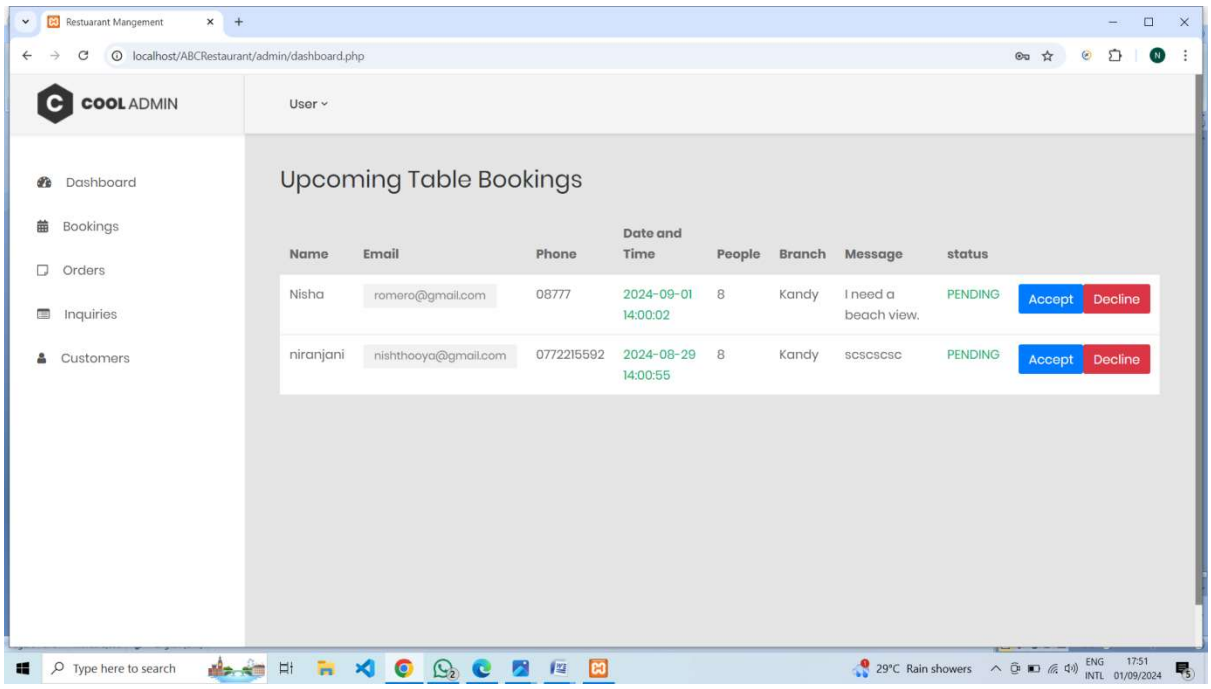


Can be seen the food orders made by customers.





Restaurant staff's dashboard



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.

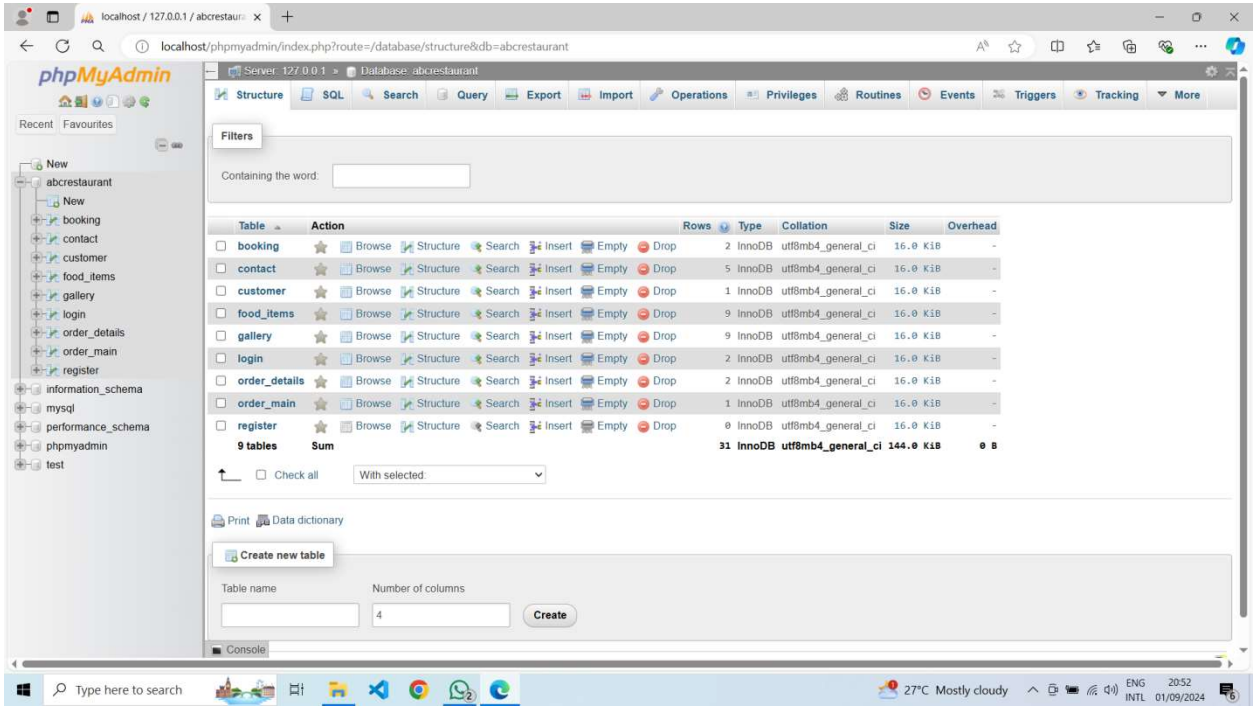
Adaptability

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 .



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 bugshave been reported.
- No functional bugs.
- 80% of the bugs reported have beensolved.

Test Environment

IDE: Eclipse

Programming Language: Java

Testing Framework: Selenium TestNG

Selenium WebDriver: ChromeDriver

Test Cases Executed- 7

Test Results

Passed: 6

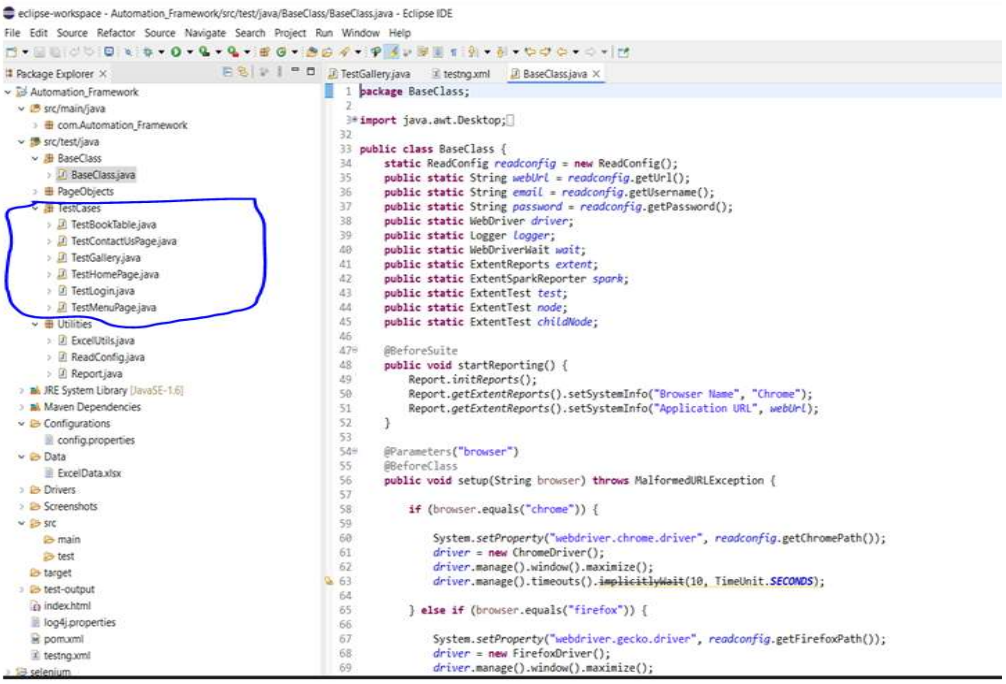
Failed: 1(invalid credentials)

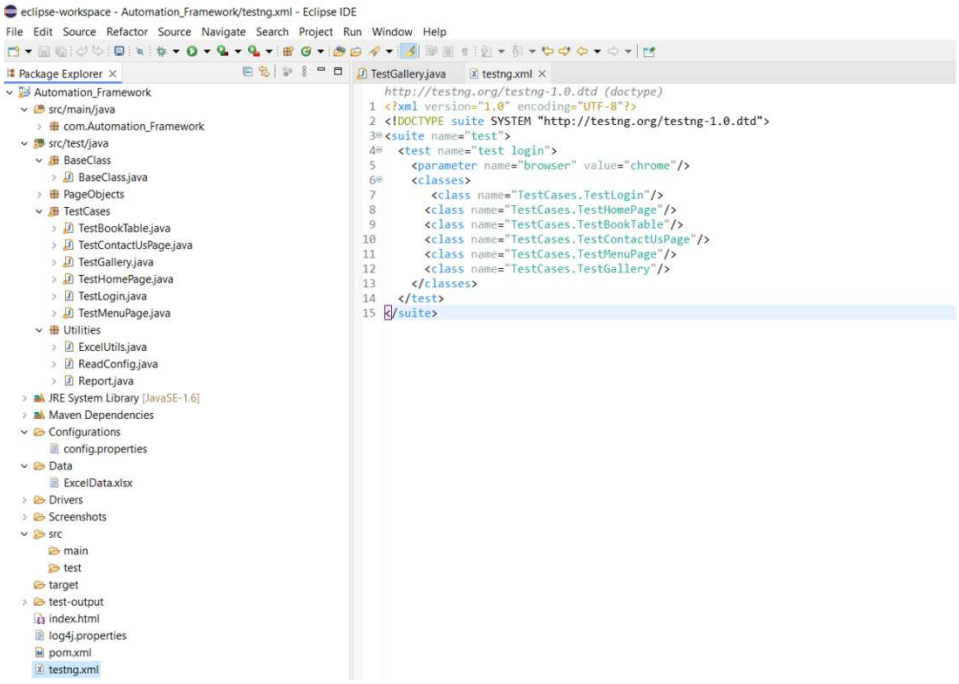
Failed Test Case

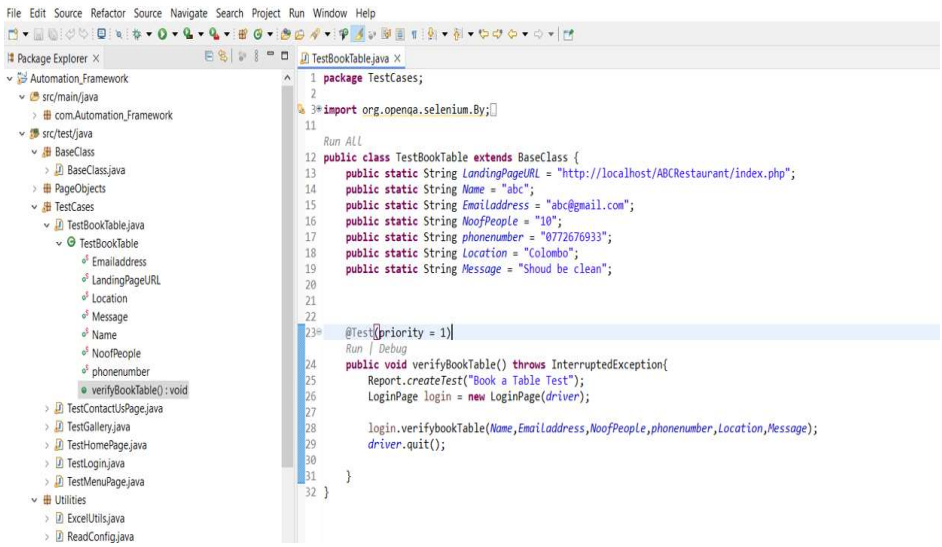
When try to test the Login page with the invalid credentials it doesn't give the access to login. To prevent the unauthorized access it doesn't provide the access. Even though the test case fails, the actual and expected results are indeed the same.

Test cases

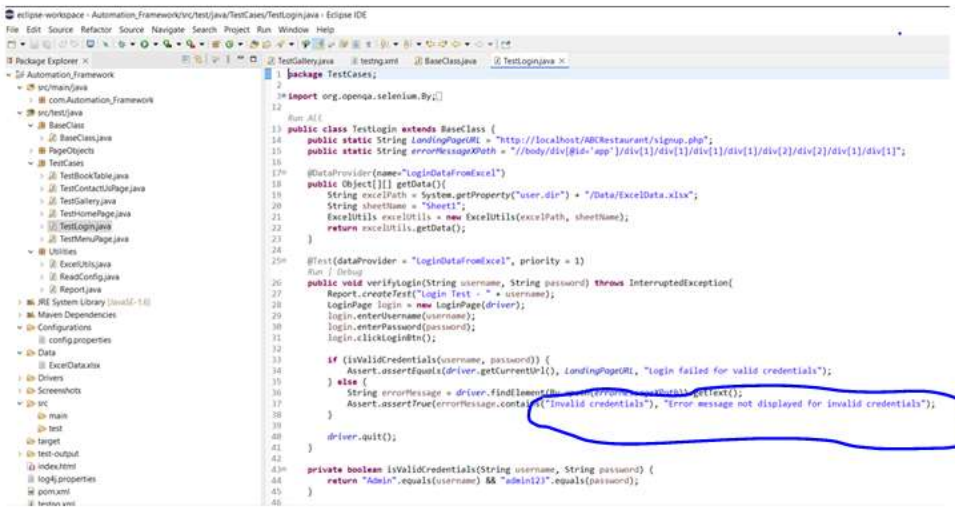
Test (Automation) has been to 6 test cases.


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
Automation_Framework
src/main/java
com.Automation_Framework
src/test/java
BaseClass
BaseClass.java
PageObjects
TestCases
TestBookTable.java
TestContactUsPage.java
TestGallery.java
TestHomePage.java
TestLogin.java
TestMenuPage.java
Utilities
ExcelUtils.java
ReadConfig.java
Report.java
JRE System Library [JavaSE-1.6]
Maven Dependencies
Configurations
config.properties
Data
ExcelData.xlsx
Drivers
Screenshots
src
main
test
target
test-output
index.html
log4j.properties
pom.xml
testng.xml
selenium
TestGallery.java
testng.xml
BaseClass.java
package BaseClass;
import java.awt.Desktop;
public class BaseClass {
static ReadConfig readconfig = new ReadConfig();
public static String webUrl = readconfig.getUrl();
public static String email = readconfig.getUsername();
public static String password = readconfig.getPassword();
public static WebDriver driver;
public static Logger logger;
public static WebDriverWait wait;
public static ExtentReports extent;
public static ExtentSparkReporter spark;
public static ExtentTest test;
public static ExtentTest node;
public static ExtentTest childNode;
@BeforeSuite
public void startReporting() {
Report.initReports();
Report.getExtentReports().setSystemInfo("Browser Name", "Chrome");
Report.getExtentReports().setSystemInfo("Application URL", webUrl);
}
@Parameters("browser")
@BeforeClass
public void setup(String browser) throws MalformedURLException {
if (browser.equals("chrome")) {
System.setProperty("webdriver.chrome.driver", readconfig.getChromePath());
driver = new ChromeDriver();
driver.manage().window().maximize();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
} else if (browser.equals("firefox")) {
System.setProperty("webdriver.gecko.driver", readconfig.getFirefoxPath());
driver = new FirefoxDriver();
driver.manage().window().maximize();

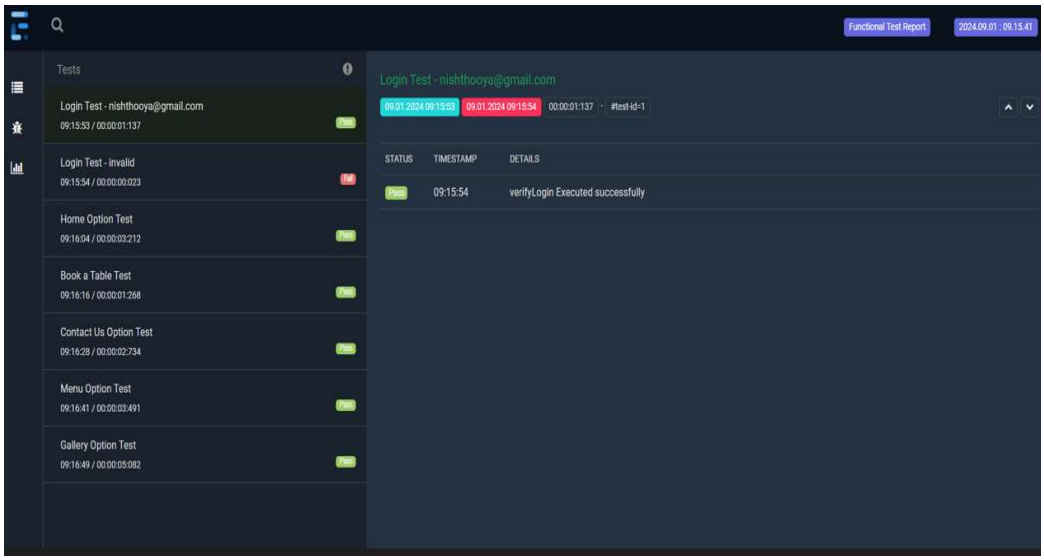

eclipse-workspace - Automation_Framework/testng.xml - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Package Explorer
Automation_Framework
src/main/java
com.Automation_Framework
src/test/java
BaseClass
BaseClass.java
PageObjects
TestCases
TestBookTable.java
TestContactUsPage.java
TestGallery.java
TestHomePage.java
TestLogin.java
TestMenuPage.java
Utilities
ExcelUtils.java
ReadConfig.java
Report.java
JRE System Library [JavaSE-1.6]
Maven Dependencies
Configurations
config.properties
Data
ExcelData.xlsx
Drivers
Screenshots
src
main
test
target
test-output
index.html
log4j.properties
pom.xml
testng.xml
TestGallery.java
testng.xml
http://testng.org/testng-1.0.dtd (doctype)
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="test">
<test name="test login">
<parameter name="browser" value="chrome"/>
<classes>
<class name="TestCases.TestLogin"/>
<class name="TestCases.TestHomePage"/>
<class name="TestCases.TestBookTable"/>
<class name="TestCases.TestContactUsPage"/>
<class name="TestCases.TestMenuPage"/>
<class name="TestCases.TestGallery"/>
</classes>
</test>
</suite>



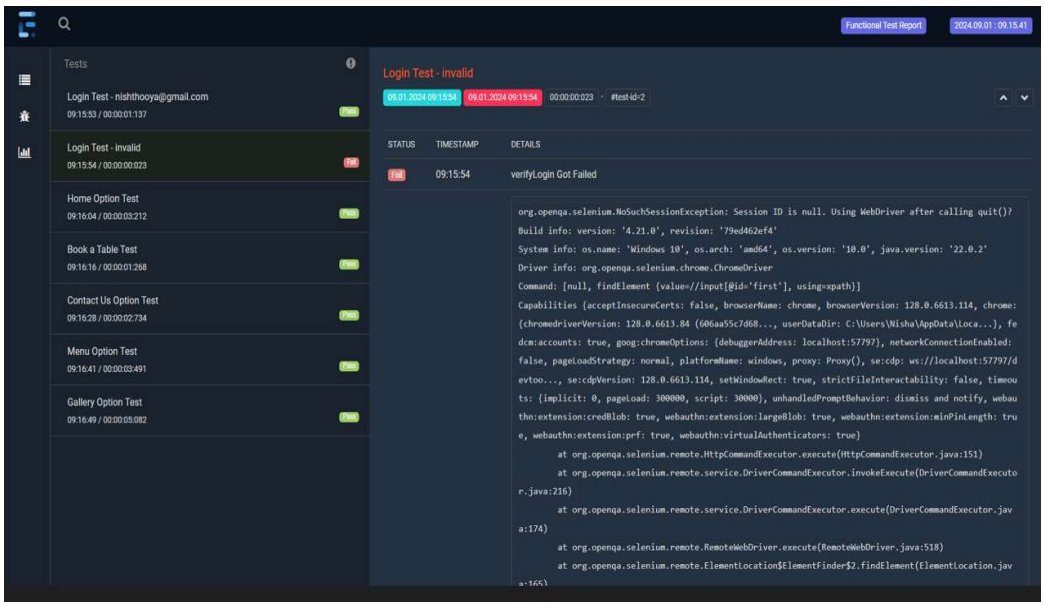
Below screen shot helps to justify the TDD. Verify successful login with valid credentials and unsuccessful login with invalid credentials.



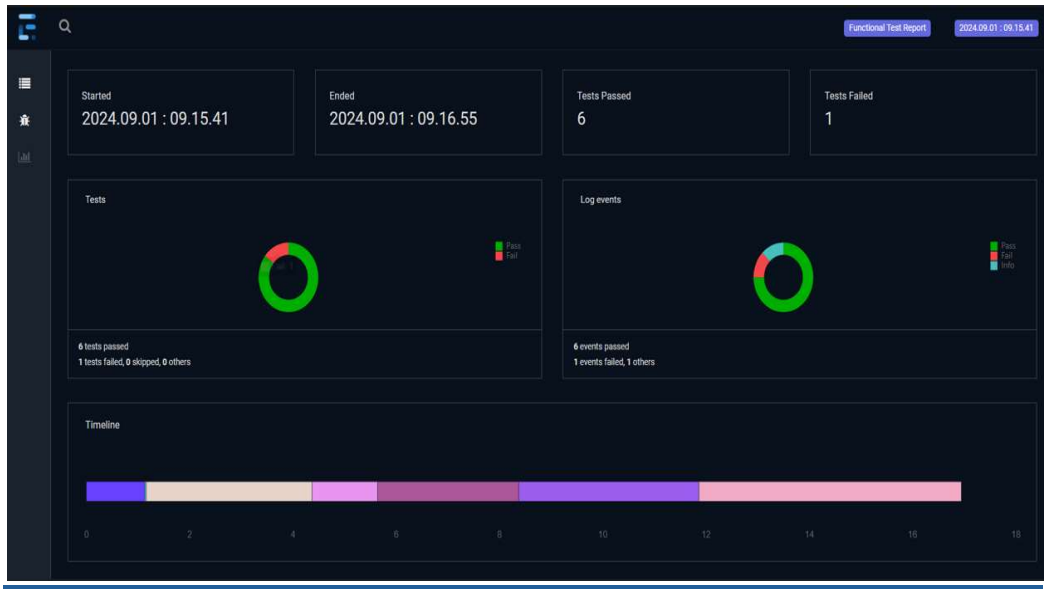
Functional report of test cases .According to that 6 were pass and one 1 was fail.



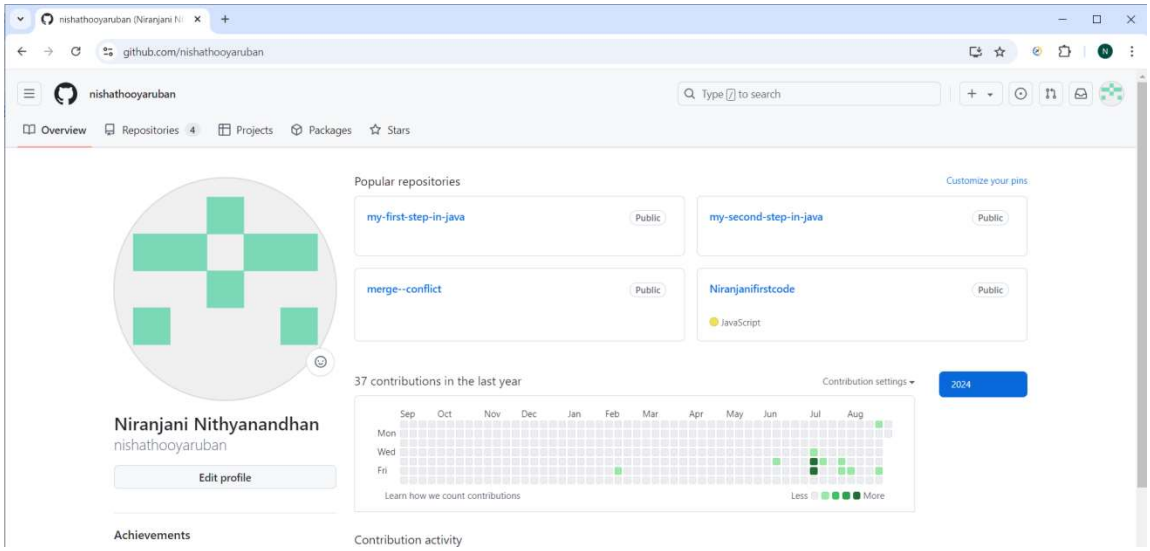
When find out the reason of the failure it shows the reason. (Invalid credentials)



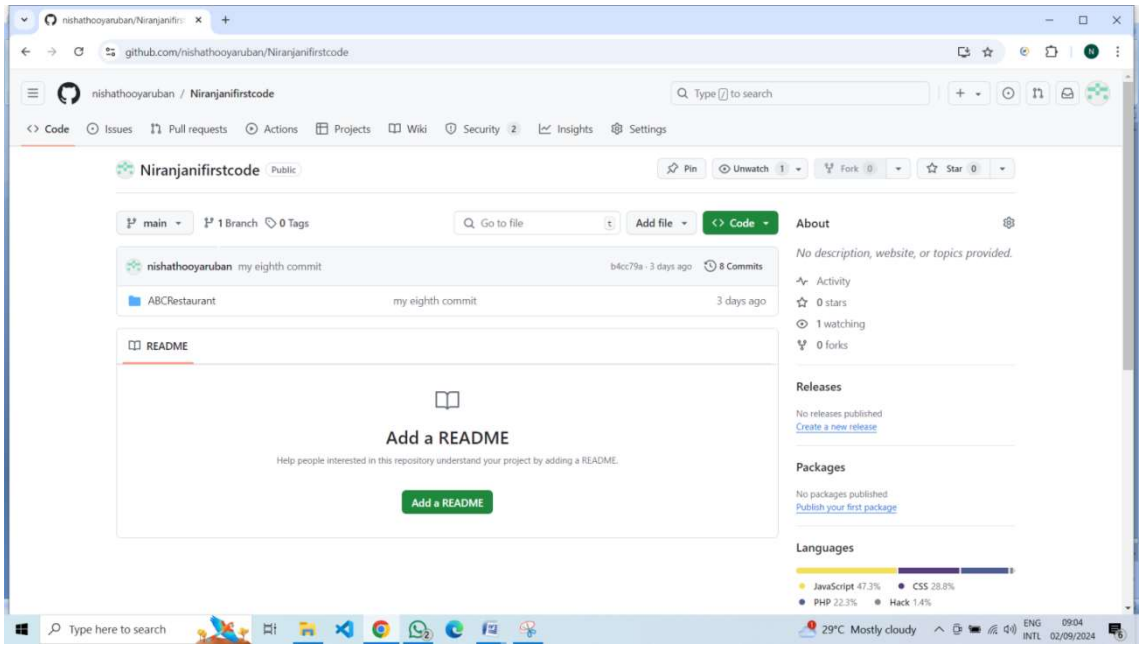
Final overall report

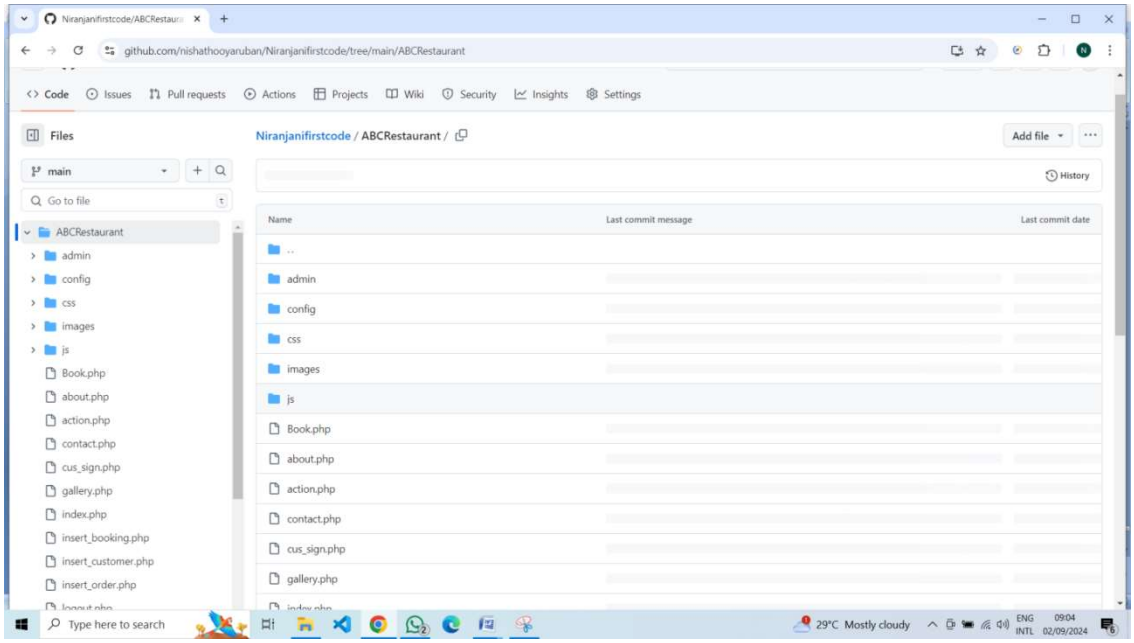


Work with github

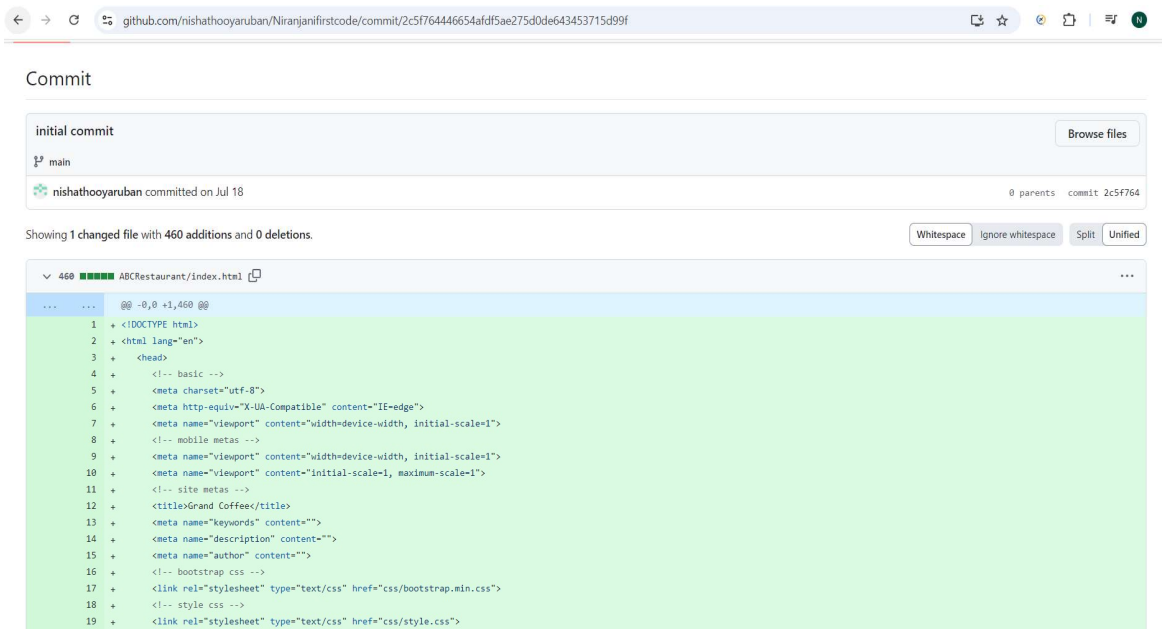


I have created a repository named “Niranjanifirstcode” .

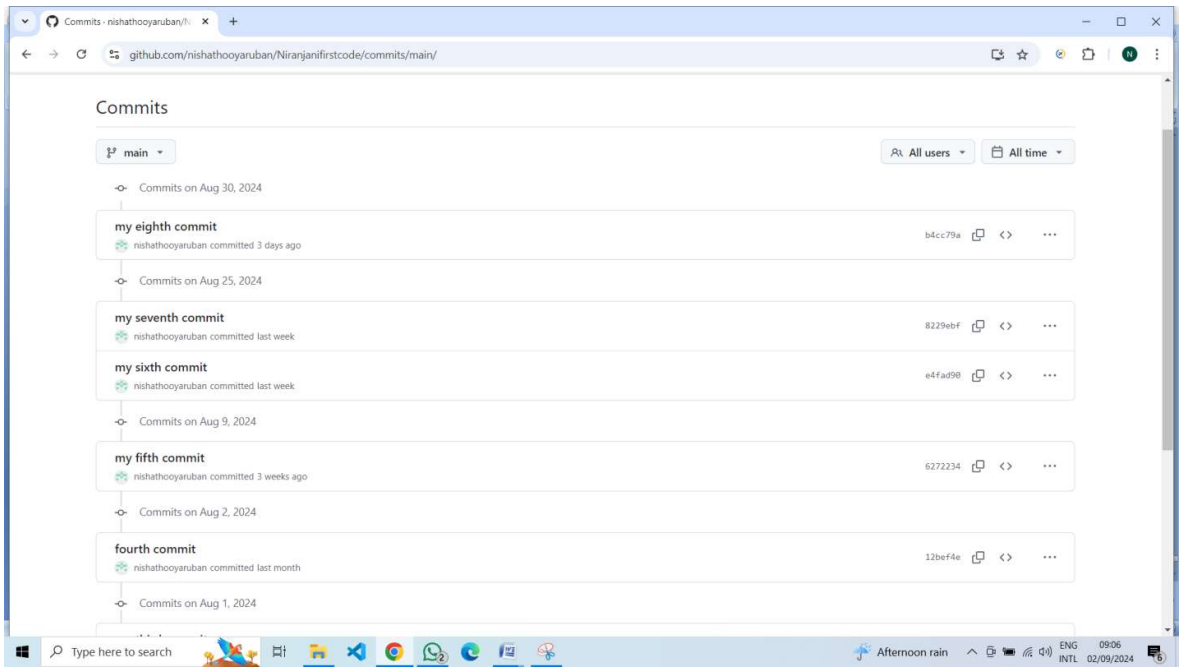




My initial commit has done with some front end codes



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



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.