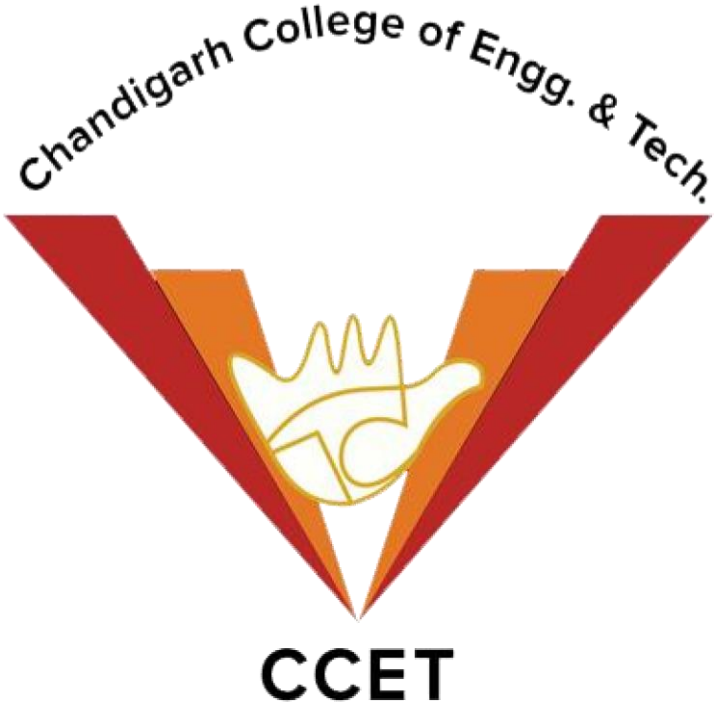
Software requirement Specification (SRS)



Dining Restaurant website

Submitted To:
Dr. Santosh Kumar Yadav Lecturer

CSE DEPARTMENT

Submitted By:
Vishwash (9535)
Eklavya (9508)
Aryan Rathi (9505)
Peyush Sharma (9524)
6TH SEMESTER

Table of Contents

Software requirement specification (Dining Restaurant website)

1. Introduction

1.1 Purpose:

The purpose of this project is to develop a web-based Restaurant Point of Sale (POS) and customer website system. This system allows customers to make reservations, manage their profiles, and earn loyalty points. For the restaurant staff, the system offers functionalities to process orders, manage tables, handle payments, and generate reports, among other administrative tasks.

1.2 Scope:

The scope of the system includes both customer-facing and staff-facing functionalities:

- Customer Side: Allows users to make reservations, register accounts, and view their profile points.
- Staff Side: Enables staff to take orders, send them to the kitchen, process payments, print receipts, and manage user preferences. Additionally, it provides the ability to view charts, generate reports, and manage CRUD operations (Create, Read, Update, Delete).

1.3 Definitions, Acronyms, and Abbreviations:

- POS: Point of Sale, the system used to process restaurant transactions.
- CRUD: Create, Read, Update, Delete – basic database operations.
- XAMPP: A free and open-source cross-platform web server solution stack used to run PHP and MySQL.
- MySQL: A relational database management system used to store restaurant data, such as customer accounts and orders.

1.4 References:

This section would typically list external documents or resources referenced during the development of the system, such as:

- **PHP Documentation**: For understanding specific PHP functionality used in the project.
- **MySQL Documentation**: For setting up and managing the database.
- **XAMPP Documentation**: For configuring the local environment where the system runs.

1.5 Overview:

This section summarizes the overall structure of the document and outlines what will be covered. It would introduce the different modules (Customer Side and Staff Side) and provide a brief overview of how the system supports both user groups (customers and restaurant staff). Additionally, it would mention the setup process for running the system locally using NetBeans and XAMPP.

---

2. Overall Description

2.1 Product Perspective:

The Restaurant POS and Website project is a web-based solution that provides a dual interface: one for **customers** and another for **staff**. The customer interface allows users to make reservations, manage their profiles, and view loyalty points, while the staff interface supports restaurant operations such as order processing, payments, table management, and report generation. The system uses PHP and MySQL for backend processing and data storage, and HTML, CSS, and JavaScript for the user interface. It is built using PHP 7.4 and runs locally through XAMPP, making it a self-hosted solution for restaurants.

2.2 Product Functions:

The product provides several essential functions:

- **Customer Side**:
  - Make reservations online.
  - Register for an account to track reservations and view profile points.
- **Staff Side**:
  - Take orders and send them to the kitchen.
  - Process customer payments (both cash and card).
  - Print receipts.
  - Manage CRUD operations for the restaurant's data.
  - View customer preferences.
  - Generate reports and visualize data through charts and graphs.

2.3 User Characteristics:

- **Customers**: Users visiting the restaurant who will primarily interact with the system to make reservations and manage their profiles.
- **Restaurant Staff**: Waitstaff, kitchen staff, and cashiers who will use the POS system to take orders, process payments, and manage day-to-day restaurant operations.
- **Administrators**: Restaurant managers or owners who will use the system to oversee staff operations, review reports, and maintain data in the system.

2.4 Constraints:

- The system is designed to run on a local server using XAMPP. This requires proper configuration, including starting Apache and MySQL services.
- It may not be scalable for large-scale restaurant chains unless modifications are made to host it on cloud servers.
- Any downtime in the local server environment may interrupt the availability of the POS system.
- The system may need further development to support advanced security features for handling sensitive customer data like payment details.

2.5 Assumptions and Dependencies:

- It is assumed that users (staff and customers) have basic computer literacy.
- The system depends on the correct installation and configuration of **XAMPP** (or a similar local server stack), PHP 7.4, and MySQL for proper functionality.
- The system relies on uninterrupted local network connectivity to ensure communication between the customer and staff interfaces.
- It is assumed that the restaurant has the necessary hardware for receipt printing and order processing.

---

3. Specific Requirements

This section outlines the detailed functional, non-functional, external interface, and data requirements for the system.

3.1 Functional Requirements:

The functional requirements describe the core capabilities of the Restaurant POS and Website system. Based on the README, the following functionalities are essential:

- **Customer Side Functionalities**:
  - **Make Reservations**: Customers must be able to make reservations through the website.
  - **Account Registration**: Customers should have the ability to register and create an account to manage their profiles and view loyalty points.
  - **View Profile Points**: Registered users should be able to view their earned points within their profile section.
- **Staff Side Functionalities**:
  - **Order Processing**: Staff must be able to take customer orders and send them to the kitchen in real-time.
  - **Payment Processing**: The system must support both cash and card payments. Staff should be able to process payments through the POS interface.
  - **Receipt Printing**: The system should allow staff to print receipts after completing a payment.
  - **CRUD Operations**: Staff must be able to perform Create, Read, Update, and Delete operations on menu items, customer information, and orders.
  - **View Customer Preferences**: Staff must be able to view and manage individual customer preferences (such as dietary restrictions or favorite menu items).
  - **Generate Reports**: The system must be able to generate reports (e.g., daily sales, customer statistics) for administrative use.

3.2 Non-Functional Requirements:

The non-functional requirements define the quality attributes of the system, focusing on how the system operates rather than what it does.

- **Performance**:
  - The system should respond promptly to user actions, especially for real-time order processing and payment operations.
  - It must handle multiple simultaneous users without significant delay, ensuring smooth operation for both the customer-facing and staff-facing components.

- **Security**:
    - Customer and staff data must be securely stored in the database. Sensitive information (like login credentials) should be encrypted.
    - The system should provide different access levels for customers, staff, and administrators to ensure role-based security.
- **Usability**:
    - The user interface must be intuitive for both customers and staff, with clear navigation for placing orders, processing payments, and managing reservations.
- **Reliability**:
    - The system should be stable and handle any unexpected errors gracefully to minimize downtime during service hours.
- **Scalability**:
    - Though currently designed for local deployment, the system should be adaptable to accommodate future scaling to cloud-based environments or multiple restaurant locations.
    -

3.3 External Interface Requirements:

The system interacts with external components and systems. These requirements define those interfaces.

- **Database Interface**:
    - The system relies on a **MySQL database** to store all relevant data (customer accounts, orders, payments, reports, etc.).
- **Receipt Printer Interface**:
    - The system must be able to interface with standard receipt printers to print order receipts after payment processing.
- **Payment Gateway**:
    - For card payments, the system must integrate with a payment gateway that can process credit or debit card transactions.
- **Local Server Interface**:
    - The system is designed to run on a **local server** (XAMPP), which hosts the website and POS functionalities. Proper communication between the local Apache server and the MySQL database is crucial for the system to function.

3.4 Data Requirements:

The data requirements describe the type and structure of the data managed by the system.

- **Customer Data**:
  The system must store customer information such as names, contact details, reservation history, profile points, and login credentials. All customer data should be securely stored in the database.
- **Order Data**:
  The system must store order information, including the items ordered, the total price, payment method, and the customer who placed the order. This data is essential for tracking sales and generating reports.
- **Payment Data**:
  The system must store payment data, including transaction details, payment methods (cash, card), and timestamps. For card transactions, sensitive data

should not be stored, and compliance with payment gateway standards is required.

- **Staff Data**:
  The system must store staff login credentials, their role (e.g., waiter, cashier, Kitchen, staff), and any associated data needed for operational purposes.
- **Report Data**:
  The system must generate and store reports on restaurant operations, including daily sales, customer trends, and inventory. These reports must be stored for future reference and accessible to authorized users.

---

4. System Architecture

This section describes the structural organization of the system, how components interact, the database design, and API specifications.

4.1 Architectural Overview:

The system follows a **client-server architecture** where the customer-facing and staff-facing components (front end) interact with the backend services via HTTP requests. The **Apache server** (via XAMPP) serves the PHP scripts, which handle the business logic and database interactions. The system is divided into the following key components:

- **Customer Interface**: The front end of the website for customers is built using **HTML, CSS, and JavaScript**. This interface allows customers to make reservations, view their profile, and manage account information.
- **Staff POS Interface**: The staff-facing POS system is also built using HTML, CSS, and JavaScript but is optimized for restaurant staff to take orders, send them to the kitchen, and process payments.
- **Backend (PHP 7.4)**: The backend is developed in **PHP**. It handles all the business logic, database queries, and interactions between the customer and staff sides. It processes customer requests (e.g., making reservations) and staff operations (e.g., processing payments).
- **Database (MySQL)**: The system uses a **MySQL database** to store customer information, orders, payments, and staff data. The database also stores reports and other important data for generating analytics.

4.2 Data Flow and Component :

In this system, the data flows between several key components. Here's an outline of the data flow:

- **Customer Side**:
  1. A customer makes a request (e.g., a reservation) via the web interface.
  2. The front end sends this request to the server-side PHP scripts.
  3. The PHP backend interacts with the **MySQL database**, processes the request, and stores the necessary data.
  4. The server responds to the customer interface, confirming that the reservation is made.
- **Staff Side (POS)**:
  1. Staff takes an order and processes a payment.

2. The system sends the order to the kitchen and logs the transaction in the **MySQL database**.
3. Once the payment is processed, the system updates the database with the payment details and prints a receipt.

4.3 Database Schema:

The **MySQL database** schema consists of multiple tables to store all relevant data for the restaurant's operations. A possible schema might include the following tables:

- **Customers**: Stores customer details (name, contact info, account data, profile points, and reservation history).
- **Staff**: Stores information about staff members (name, login credentials, roles such as waiter, cashier, or kitchen staff).
- **Orders**: Stores order details such as menu items ordered, quantities, total amount, customer ID, and timestamps.
- **Payments**: Stores payment information, including the payment method (cash, card), transaction ID, and order ID.
- **Reservations**: Stores customer reservation data, including date, time, number of guests, and reservation status.
- **Reports**: Stores sales reports, customer activity reports, and staff performance metrics, which can be generated and viewed by administrators.

4.4 API Specifications:

Since the system is primarily designed to run on a local server using XAMPP, the APIs used are internal to the system. However, the system could have the following API endpoints to handle data exchanges between the front end and back end:

- **Reservation API**:
  - **POST /api/reservations**: Allows customers to create a new reservation.
  - **GET /api/reservations/{customerId}**: Retrieves reservation details for a specific customer.
- **Order API**:
  - **POST /api/orders**: Allows staff to create a new order and send it to the kitchen.
  - **GET /api/orders/{orderId}**: Retrieves order details by ID.
- **Payment API**:
  - **POST /api/payments**: Processes a customer's payment for an order (both cash and card).
- **Report API**:
  - **GET /api/reports/sales**: Retrieves daily, weekly, or monthly sales reports.
  - **GET /api/reports/customers**: Retrieves customer statistics and activity reports.

The APIs facilitate communication between the customer-side and staff-side components while maintaining data integrity in the **MySQL database**.

---

5. Visualized Interaction

5.1 User Registration Flow:

This flow details how customers create accounts and log in to the system to access features like reservations and profile management.

1. **Customer Accesses Website**:
    - The customer navigates to the restaurant's website.
2. **Registration Page**:
    - The customer clicks the "Register" button and is redirected to a registration form.
3. **Input User Details**:
    - The customer provides the required details such as name, email, and password.
4. **Submit Registration**:
    - The system sends the form data to the PHP backend, which interacts with the **MySQL database** to create a new customer record.
5. **Account Confirmation**:
    - Upon successful registration, the system confirms the account creation. The customer can now log in with their credentials.
6. **Login**:
    - After registration, the customer can log in using their email and password to access their profile, view reservations, and check profile points.

5.2 Order Management Flow (Task Management Equivalent):

This flow describes how restaurant staff manage customer orders, including sending orders to the kitchen.

1. **Staff Logs into POS System**:
    - Staff logs into the system using their credentials (waiter, cashier, etc.).
2. **Select Table/Customer**:
    - The staff selects the customer's table and inputs the order into the system.
3. **Place Order**:
    - The order is confirmed, and the system sends the order details to the kitchen via the backend PHP script.
4. **Kitchen Receives Order**:
    - The kitchen staff can view the incoming order through their dashboard and begin preparing the requested items.
5. **Order Status Updated**:
    - Once the order is prepared, the kitchen updates the order status to "Ready," notifying the waiting staff.

5.3 Payment Processing Flow (Messaging Equivalent):

This flow explains how payments are processed in the system.

1. **Customer Requests Bill**:
    - The customer asks for the bill, and the staff brings up the order details in the POS system.
2. **Select Payment Method**:
    - The staff selects the payment method (cash or card) in the system.
3. **Process Payment**:
    - If paying by card, the system integrates with a **payment gateway** to process the transaction. For cash, the transaction is simply logged in the database.
4. **Receipt Printed**:

    - After processing the payment, the system prints a receipt for the customer using a connected receipt printer.
5. **Transaction Logged**:
    - The payment is logged in the **MySQL database**, and the daily sales report is updated.

5.4 Reservation Management Flow (Project Posting Equivalent):

Since there's no project bidding in the restaurant system, this flow covers how customers make reservations.

1. **Customer Logs In**:
    - The customer logs into their account using their credentials.
2. **Navigate to Reservation Section**:
    - The customer selects the option to make a reservation from the main menu.
3. **Select Date/Time and Number of Guests**:
    - The customer inputs the preferred reservation date, time, and number of guests.
4. **Submit Reservation**:
    - The reservation request is submitted to the backend, where the **PHP script** interacts with the **MySQL database** to store the reservation details.
5. **Confirmation**:
    - The customer receives a confirmation message, and the reservation is updated in their profile.

---

6. Future Enhancement

The Restaurant POS and Website system can be enhanced to improve its functionality, scalability, and user experience by adding new features and capabilities.

6.1 AI-Powered Task Automation:

- The system could integrate **AI** to automate some of the routine tasks. For example:
    - **Order Suggestion**: Based on customer preferences and order history, the system could suggest menu items to returning customers.
    - **Kitchen Automation**: AI could automatically group similar orders (e.g., all orders requiring frying) to optimize kitchen workflow.
    - **Table Allocation**: AI could help assign tables based on customer preferences and restaurant capacity to optimize space utilization and customer satisfaction.

6.2 Advanced Analytics:

- Introducing **Advanced Analytics** would enable restaurant managers to gain deeper insights into their operations. This feature could include:
    - **Sales Forecasting**: Based on historical sales data, the system could predict future sales trends, helping managers optimize staffing and inventory.
    - **Customer Insights**: Advanced analytics could analyze customer behaviour, such as peak reservation times, favourite menu items, and average spend per visit.
    - **Operational Efficiency**: Analytics could highlight inefficiencies, such as average order preparation time, wait times for tables, and customer feedback analysis.

6.3 Third-Party Integrations:

- To enhance functionality, the system could integrate with popular third-party tools. Examples include:
    - **Payment Gateways**: Integrate with widely used payment platforms such as **PayPal**, **Stripe**, or mobile payment systems like **Apple Pay** or **Google Pay** for smoother payment processing.
    - **Accounting Systems**: Integration with accounting software like **QuickBooks** or **Xero** to streamline financial reporting and taxation.
    - **CRM Systems**: Integrate with Customer Relationship Management tools to manage customer data more effectively and improve customer engagement through targeted promotions.

6.4 Custom Workflow Support:

- The system could allow for customizable workflows based on the restaurant's specific needs. For example:
    - **Custom Order Process**: Restaurants could set up different workflows for special orders, like takeout, catering, or delivery.
    - **Role-Based Access**: Customizable user roles and workflows would allow different staff members to have varying levels of access and responsibilities in the system. Managers could create specific workflows for kitchen staff, waiters, and cashiers.
    - **Menu Customization**: Restaurants could dynamically customize their menus for events, daily specials, or seasonal changes and integrate them into the POS workflow.

6.5 Offline Mode:

- Implementing an **Offline Mode** would ensure that the system continues to function during temporary network outages. Features could include:
    - **Order Taking and Payments**: The POS system could continue to take orders and process payments locally, storing the data until connectivity is restored.
    - **Data Synchronization**: Once the network is restored, the system would automatically sync any offline transactions with the central database, ensuring no data is lost.
    - **Limited Offline Features**: Offline mode could provide limited functionality, such as taking orders and accessing cached customer data, but disable features like real-time reporting until online connectivity is restored.

---

7. Testing and Validation

7.1 Unit Testing:

- **Unit testing** involves testing individual components of the system in isolation to ensure that each piece of code performs its intended function. For the Restaurant POS and Website project, unit tests would focus on:

- **Reservation Module**: Ensure that customers can make reservations, and the data is stored correctly in the database.
- **Payment Processing**: Verify that payment methods (both cash and card) work correctly and that the correct information is logged in the **MySQL database**.
- **Order Handling**: Ensure that staff can take orders and send them to the kitchen, with no errors or inconsistencies in the data flow.
- **User Authentication**: Test login and registration functions to ensure that only authorized users can access their profiles and perform actions.

7.2 Integration Testing:

- **Integration testing** ensures that different modules of the system work together seamlessly. This is particularly important for a system like the Restaurant POS and Website, where different components need to interact.
  - **Customer and Staff Interaction**: Verify that customer actions (such as making a reservation) are reflected on the staff side and vice versa.
  - **Order and Kitchen Synchronization**: Ensure that orders placed by waitstaff are correctly routed to the kitchen and that status updates (e.g., "Order Ready") are reflected on the waitstaff's POS interface.
  - **Payment Gateway**: If third-party payment gateways are integrated, ensure that payments are processed correctly and the relevant transaction data is saved.
  - **Database Interaction**: Confirm that the **PHP backend** communicates effectively with the **MySQL database** to store and retrieve data without any errors.

7.3 System Testing:

- **System testing** involves validating the complete, integrated system to ensure it meets the required specifications. It focuses on the overall functionality of the Restaurant POS and Website project.
  - **End-to-End Functionality**: Test the full flow from customer registration, reservation, and payment to ensure the entire workflow operates as expected.
  - **Data Integrity**: Verify that all data, including reservations, orders, and payments, is correctly stored and can be retrieved without errors. This ensures that no data is lost or corrupted during operations.
  - **User Interface Testing**: Ensure that the customer and staff interfaces are intuitive and work across different devices (e.g., desktop, tablets, POS terminals).

7.4 Performance Testing:

- **Performance testing** ensures that the system performs efficiently under different conditions and workloads. This is particularly important for a POS system that will be used during peak restaurant hours.
  - **Load Testing**: Simulate multiple customers and staff accessing the system at the same time to ensure that it can handle high traffic without slowing down.
  - **Stress Testing**: Push the system beyond its normal operational limits to identify its breaking point (e.g., during busy restaurant hours with numerous simultaneous orders and payments).

- **Response Time**: Measure the time it takes for key operations (such as placing an order, processing a payment, or retrieving a report) to ensure that the system remains responsive.
- **Resource Utilization**: Monitor the system's use of resources (e.g., memory, CPU) during peak usage to ensure that it operates efficiently without consuming excessive server resources.

---

8. Risks and Mitigation

8.1 Data Breach:

A **data breach** occurs when unauthorized individuals gain access to sensitive customer or staff data, such as personal details, login credentials, or payment information.

- **Risk**:
  - The system stores customer and staff information in the **MySQL database**, which may be targeted by attackers seeking to steal or exploit sensitive data.
- **Mitigation**:
  - **Data Encryption**: Ensure that sensitive information (like passwords) is encrypted before storing it in the database.
  - **Role-Based Access Control**: Implement role-based access control (RBAC) to ensure that only authorized users can access certain parts of the system, such as customer payment data.
  - **Regular Security Audits**: Conduct regular security checks and audits to identify potential vulnerabilities in the system.
  - **Two-Factor Authentication (2FA)**: Implement 2FA for staff and administrators to add an extra layer of security during login.

8.2 System Downtime:

System downtime refers to periods when the POS or website is unavailable, preventing staff from processing orders and customers from making reservations.

- **Risk**:
  - If the system crashes or the local server (XAMPP) experiences issues, the restaurant operations may be interrupted, affecting business efficiency.
- **Mitigation**:
  - **Redundancy**: Implement backup servers or failover systems to ensure that operations continue during server outages.
  - **Regular Backups**: Schedule regular backups of the database so that the system can be restored quickly in case of a failure.
  - **Monitoring Tools**: Use monitoring tools to track the system's performance and detect potential issues before they cause downtime.
  - **Scheduled Maintenance**: Plan for scheduled maintenance during non-operational hours to minimize disruption.

8.3 Missed Deadlines:

This risk relates to delays in system updates, new feature development, or delivering system outputs (such as reports or order processing).

- **Risk**:
  - If updates to the system or the implementation of new features (such as third-party integrations or AI-based enhancements) are delayed, it could lead to customer dissatisfaction or operational inefficiency.
- **Mitigation**:
  - **Agile Development Practices**: Implement agile development methodologies, allowing the team to release features incrementally and manage deadlines effectively.
  - **Clear Communication**: Ensure that any delays are communicated transparently to stakeholders, and provide alternatives or temporary solutions when necessary.
  - **Testing and Staging Environment**: Create a separate staging environment for testing updates and new features to avoid delays during production.

8.4 User Error or Misuse:

User error or misuse occurs when staff or customers unintentionally perform incorrect actions, such as inputting incorrect data, misplacing orders, or making wrong payments.

- **Risk**:
  - Restaurant staff or customers may misuse the system, leading to incorrect orders, incorrect payment processing, or data inconsistencies.
- **Mitigation**:
  - **User Training**: Provide comprehensive training to staff to ensure they understand how to use the system effectively.
  - **Intuitive User Interface**: Design the user interface to be intuitive, reducing the likelihood of errors by providing clear instructions and feedback for each action.
  - **Input Validation**: Implement input validation to ensure that all data entered into the system (e.g., customer reservations, order details, payment information) is accurate and follows the correct format.
  - **Error Logging**: Log all user errors and provide immediate feedback to users to prevent mistakes from escalating.

---

9. Use Case Scenarios

9.1 Order Creation and Assignment:

This scenario illustrates how waitstaff takes customer orders and assigns them to the kitchen.

- **Actors**: Waitstaff, Kitchen Staff, Customers
- **Flow**: Waitstaff logs into the POS system, takes the customer's order, and inputs it into the system. The order is sent to the kitchen for preparation. The kitchen staff marks the order as "Ready" when completed.
- **Exception**: If a menu item is unavailable, the system notifies the waitstaff to modify the order.

9.2 Real-Time Messaging Between Team Members:

Staff members use the POS messaging system to communicate in real time.

- **Actors**: Waitstaff, Kitchen Staff, Management
- **Flow**: The kitchen staff can send real-time messages to waitstaff for clarifications on orders or special requests. The waitstaff responds via the system to ensure accuracy.
- **Exception**: If a staff member is offline, the message is delayed until they log in.

9.3 Order Completion and Feedback:

This describes how orders are completed, and feedback is collected from customers.

- **Actors**: Waitstaff, Customers
- **Flow**: After the customer finishes their meal, waitstaff processes the payment through the POS. The system prompts the customer for feedback after marking the order as complete.
- **Exception**: If the payment fails, an alternate method is prompted.

This summary outlines the primary interactions within the system for managing orders, communication, and customer feedback.

---

10. Conclusion

The Restaurant POS and Website system enhances restaurant operations by integrating customer reservations, order management, and payment processing. It improves staff efficiency and customer experience with a user-friendly interface. Future enhancements like AI automation and advanced analytics will further optimize the system. With strong security and testing protocols, the system ensures reliability and scalability, making it a valuable tool for modern restaurants.
