# SMART RESTURANT ORDERING SYSTEM

**Higher National Diploma in Software Engineering Final Project Report**

**23.2P**

data:image/jpeg;base64,

**Declaration**

“We certify that this project does not incorporate without acknowledgement, any material previously submitted for a Diploma in any institution and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. We also hereby give consent for our project report, if accepted, to be made available for photocopying and for interlibrary loans, and for the title and summary to be made available to outside organizations.”

**Project title:** a Smart Restaurant Ordering System

**Group Members:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Index Number** | **Signature** |
| Udantha Vishwajith | KAHDSE232P-009 |  |
| Anjaliaka de Silva | KAHDSE232P-010 |  |
| Shakya Rathnayaka | KAHDSE232P-011 |  |
| Sanjana Rathnayake | KAHDSE232P-016 |  |

**Course Directors** : Mrs. Anjula Weerasinghe/

: Mr. Sandaruwan Herath **Signature of the Course Directors : …………………………. Date of submission : ………………………….**

**Supervisors** : Mrs. Anjula Weerasinghe/

: Mr. Sandaruwan Herath

**Signature** : ………………………….

**“The project is submitted in partial fulfillment of the requirement of the Higher National Diploma of Software Engineering of National Institute of Business Management”**

**Abstract**

The Smart Restaurant Ordering System is a comprehensive digital solution designed to revolutionize traditional restaurant operations by automating and optimizing the order processing workflow. This project addresses critical operational challenges faced by restaurants including delays in order communication, inefficiencies in payment processing, and difficulties in generating real-time analytics. The system introduces a centralized platform that connects customers, front desk staff, kitchen personnel, and management through integrated digital interfaces.

Utilizing the TALL stack (Tailwind CSS, Alpine.js, Laravel, and Livewire) along with Vue.js components, the system provides a responsive and intuitive user experience across multiple interfaces. The solution features automated order placement, integrated payment processing with multiple payment options, and robust reporting capabilities that provide valuable insights into sales patterns and customer preferences.

Through the implementation of this system, restaurants can significantly reduce order processing times, minimize errors in communication, streamline payment handling, and leverage data-driven decision-making to optimize operations. The project follows an Agile development methodology, ensuring continuous refinement based on stakeholder feedback throughout the development lifecycle.

# Table of Contents

[Declaration 2](#_Toc198669720)

[Table of Contents 4](#_Toc198669722)

[List of Keywords 6](#_Toc198669723)

[Table of Figure 7](#_Toc198669724)

[Acknowledgement 8](#_Toc198669725)

[Chapter 1: Introduction 9](#_Toc198669726)

[1.1 Introduction 9](#_Toc198669727)

[1.2 Organization Structure 9](#_Toc198669728)

[1.3 Current Operations in Organization 10](#_Toc198669729)

[1.4 Users and Responsibilities in the Organization 10](#_Toc198669730)

[1.5 Problem Definition 11](#_Toc198669731)

[1.6 Proposed Solution 11](#_Toc198669732)

[1.7 Project Schedule 13](#_Toc198669733)

[1.8 Chapter Summary 13](#_Toc198669734)

[Chapter 2: Methodology 14](#_Toc198669735)

[2.1 Introduction 14](#_Toc198669736)

[2.2 Data Collection Method(s) 14](#_Toc198669737)

[2.3 Software Process Model 15](#_Toc198669738)

[2.4 Software Development Tools 15](#_Toc198669739)

[2.5 Testing Strategies 16](#_Toc198669740)

[2.6 Implementation Plan 17](#_Toc198669741)

[2.7 Chapter Summary 17](#_Toc198669742)

[Chapter 3: Analysis 18](#_Toc198669743)

[3.1 Introduction 18](#_Toc198669744)

[3.2 UML Diagrams 18](#_Toc198669745)

[3.2.1 Class Diagram 18](#_Toc198669746)

[3.2.2 Sequence Diagrams 20](#_Toc198669747)

[3.2.3 Use Case Diagram 23](#_Toc198669748)

[3.3 Data Flow Diagrams 24](#_Toc198669749)

[3.4 ER Diagram 28](#_Toc198669750)

[3.5 Chapter Summary 29](#_Toc198669751)

[Chapter 4: Solution design 30](#_Toc198669752)

[4.1 Introduction 30](#_Toc198669753)

[4.2 System Design 30](#_Toc198669754)

[4.2.1 Process Flow 32](#_Toc198669755)

[4.3 Interface Design 34](#_Toc198669756)

[4.5 Database Design 61](#_Toc198669757)

[Chapter 5: Conclusion 62](#_Toc198669758)

[References 63](#_Toc198669759)

[Appendix 63](#_Toc198669760)

[User Manual 67](#_Toc198669761)

# List of Keywords

1. ER diagrams - It is a data modeling technique that graphically depicts an information system's components and their interactions.
2. DFD diagrams - A data diagram is a graphical representation of the "flow" of data through an information system that depicts the process characteristics.
3. Use case diagram - It is a graphical representation of the interactions between system components.
4. Database - It is a structured collection of data stored in a computer, particularly one that is accessible in multiple ways.
5. Primary key - It is a key in a relational database that is unique to each record.
6. Foreign key - It is a field in one table that uniquely identifies a row in another table or within the same table.

# Table of Figure

[Figure 1.Class Diagram 18](#_Toc197810465)

[Figure 2.Use Case Diagram 22](#_Toc197810466)

[Figure 3.1.Data Flow Diagrams - Level 0 23](#_Toc197810467)

[Figure 3.1.Data Flow Diagrams - Level 0 23](#_Toc197810467)

[Figure 3.1.Data Flow Diagrams - Level 0 23](#_Toc197810467)

[Figure 3.1.Data Flow Diagrams - Level 0 23](#_Toc197810467)

[Figure 3.1.Data Flow Diagrams - Level 0 23](#_Toc197810467)

[Figure 3.1.Data Flow Diagrams - Level 0 23](#_Toc197810467)

# Acknowledgement

This project could not have been completed without the help of so many people, whose names cannot all be listed. However, our group wishes to convey our heartfelt gratitude and thankfulness to our course directors and supervising lecturers, Mr. Sandaruwan Herath and Mrs. Anjula Weerasinghe, for their invaluable assistance, advice, attention, and time.

We would like to thank all of the lecturers who taught us during this course, as well as the NIBM personnel, for their help and understanding.

Last but not least, we want to express our heartfelt gratitude and appreciation to our parents.

Finally, thank you to everyone who helped make this a success

# Chapter 1: Introduction

## 1.1 Introduction

This chapter introduces the project focused on the development of a **Smart Restaurant Ordering System**. It provides an overview of the organization, highlights the current operations, identifies the existing problems, and presents the objectives and proposed solutions of the project.

## 1.2 Organization Structure

**The Restaurant Environment:**

The restaurant is structured around various departments that ensure seamless operations, including:

* **Front Desk**: Manages customer interaction and order processing.
* **Kitchen Staff**: Prepares food and provides updates on order statuses.
* **Wait Staff**: Assists customers with placing orders, collecting payments, and providing help when requested.
* **Management/Admin**: Oversees the overall functioning of the restaurant, analyzes sales data, generates reports, and makes informed decisions.

**Organizational Flow:**

1. **Customers** interact with the front desk or waiters to place orders.
2. **Front Desk** communicates orders to the system, and payments are processed through either digital channels or cash.
3. **Kitchen Staff** prepares and delivers orders based on the system’s input.
4. **Management** focuses on report generation, sales analysis, and system maintenance.

## 1.3 Current Operations in Organization

**Traditional Ordering Process:**

* **Manual Order Placement**: Customers typically place orders verbally through wait staff or at the front desk.
* **Payment Processing**: Payments are manually processed, with either cash handling or basic card payment processing.
* **Order Status Updates**: Customers often ask for updates on their orders directly from waiters.
* **Report Generation**: Sales data and other performance reports are manually compiled by the admin team at the end of the day, usually requiring significant manual effort.

The current system is prone to inefficiencies, such as delays in communication between wait staff and the kitchen, errors in order details, and challenges in tracking sales and analytics.

## Users and Responsibilities in the Organization

* **Customers -** Place orders, request updates on their orders, make payments, and interact with the staff for assistance.
* **Front Desk -** Manage order placements and cancellations, oversee payment processing, and coordinate communication between customers and the kitchen.
* **Kitchen Staff -** Prepare orders and provide updates on order statuses to the system.
* **Waiters -** Assist customers with placing additional orders, making payments (in case of cash transactions), and responding to customer queries.
* **Admin/Manager -** Generate sales reports, analyze order trends, and make critical decisions to optimize the restaurant’s performance.

## 1.5 Problem Definition

The restaurant currently faces several operational challenges that hinder its efficiency and ability to deliver optimal customer experience. The existing system relies heavily on manual processes, which lead to delays, errors, and miscommunication across different departments. When customers place orders, waiters or the front desk staff must manually communicate these orders to the kitchen, often leading to delays in order preparation, especially during peak hours. This manual system increases the likelihood of errors in order details, such as incorrect items or quantities, which in turn leads to customer dissatisfaction and wasted resources.

Payment processing is another area where the restaurant encounters inefficiencies. The current system does not integrate various payment methods, leading to a fragmented experience. While cash payments are manually handled, card payments or digital wallets are processed separately, often causing delays and confusion during peak business hours. This lack of integration makes it difficult to track payments and leads to inconsistencies in financial records. Furthermore, managing cancellations and refunds is a time-consuming process, as requests must be reviewed manually by the front desk, leading to delays in resolving customer concerns.

In addition to these operational challenges, the management faces significant difficulties in tracking sales data and generating reports. With no real-time analytics in place, the sales team must manually compile end-of-day sales reports, which are both time-consuming and prone to errors. Analyzing order trends or time-based analytics requires significant manual effort, which delays decision-making and prevents the restaurant from making quick adjustments to improve performance. In summary, the restaurant's existing manual processes are inefficient, error-prone, and inadequate for handling the growing demands of modern customers and competitive business environments.

## 1.6 Proposed Solution

To address the problems outlined above, the proposed Smart Restaurant Ordering System offers a comprehensive digital solution aimed at automating and optimizing the restaurant’s operations. This system is designed to streamline every stage of the customer’s experience, from order placement to payment processing, while also providing management with the tools needed for real-time reporting and data-driven decision-making.

The first aspect of the solution involves automating the order placement process. Instead of relying on manual communication between customers, waiters, and the kitchen, the system will allow customers to place orders directly through a centralized platform. The system will instantly relay the order details to both the kitchen and front desk, minimizing delays and reducing the chances of errors. The kitchen staff will receive real-time updates on the status of orders, such as new requests, changes, or cancellations, ensuring that the preparation process runs smoothly and efficiently. This automation will significantly enhance the speed and accuracy of order handling, particularly during peak times.

The proposed system will also streamline payment processing by integrating various payment methods into a single platform. Customers will have the option to pay via credit card, digital wallets, or cash. Digital payments, such as card transactions or wallet payments, will be handled directly through the system, providing a seamless experience for both customers and the front desk staff. In cases where cash payment is preferred, the system will notify the front desk, prompting them to send a waiter to collect the payment. This integration of payment methods will reduce the time spent on processing transactions and eliminate errors associated with manual cash handling. Furthermore, the system will simplify the management of cancellations and refunds, as requests will be processed digitally, significantly reducing response times.

One of the key features of the Smart Restaurant Ordering System is its ability to generate real-time reports. The system will pull data directly from the sales database to create daily sales reports, order analytics, and time-based performance analytics. These reports will provide management with valuable insights into customer trends, peak business hours, and overall restaurant performance. By automating the report generation process, the system will save significant time for the management team, allowing them to focus on making strategic decisions to improve operational efficiency and customer satisfaction. The availability of real-time data will enable the restaurant to make timely adjustments to its operations, such as adjusting staffing levels during peak hours or modifying menu items based on customer preferences.

The Smart Restaurant Ordering System proposes:

* A **centralized digital platform** where customers can place orders, make payments, and receive status updates.
* **Real-time data synchronization** between the front desk, kitchen, and management to ensure a seamless flow of information.
* Integration of **multiple payment gateways**, including credit cards, digital wallets, and cash-handling mechanisms.
* A robust **reporting feature** that provides daily sales reports, time-based analytics, and order trends to help the management team optimize the restaurant's performance.
* A **user-friendly interface** that allows both the front desk and waiters to efficiently interact with customers and the system.

## 1.7 Project Schedule

|  |  |  |  |
| --- | --- | --- | --- |
| Phase | Tasks | Timeframe | |
| 1: Requirements | Requirement gathering, problem definition, system specs | Week 1 - Week 2 | |
| 2: Design | System architecture design, database design | Week 3 - Week 4 | |
| 3: Development | Develop order placement, payment, and reporting modules | Week 5 - Week 8 | |
| 4: Testing | User acceptance testing, integration testing | Week 9 - Week 10 |
| 5: Deployment | System deployment and user training | Week 11 - Week 12 |
| 6: Maintenance | Ongoing support and improvements | Ongoing |

## 1.8 Chapter Summary

In summary, the **Smart Restaurant Ordering System** will streamline the restaurant’s operations by automating order placements, payments, and reporting processes. The current manual methods, which are prone to inefficiencies, will be replaced by a more effective and accurate system, ensuring improved customer satisfaction and better management insights.

# Chapter 2: Methodology

## 2.1 Introduction

The methodology chapter outlines the approaches and processes used to develop the Smart Restaurant Ordering System. This chapter focuses on the methods adopted for gathering relevant data, the software process model chosen for development, the tools used for building the system, the testing strategies employed to ensure the software's reliability, and the implementation plan for the project. Each of these components is essential in creating a robust, user-friendly system that meets the project’s objectives.

## 2.2 Data Collection Method(s)

To ensure that the Smart Restaurant Ordering System effectively meets the restaurant's needs, a variety of data collection methods were employed. The primary data collection method involved **interviews** with key restaurant staff, including waiters, front desk managers, kitchen staff, and administrative personnel. These interviews provided insights into the specific challenges faced by different departments in the current operations, such as order delays, miscommunication, and payment processing issues.

Additionally, **observations** were conducted during peak and off-peak hours to assess the flow of operations and identify bottlenecks in real-time. Customer feedback surveys were also reviewed to understand the pain points from the customer's perspective, particularly regarding order accuracy and payment convenience. These qualitative data collection methods ensured that the system design addressed real-world problems and aligned user expectations.

## 2.3 Software Process Model

The **Agile Software Development Model** was chosen for this project due to its iterative and flexible nature. Agile is ideal for projects where requirements evolve over time, and it allows for regular feedback and continuous improvement. The Agile model promotes close collaboration between developers and stakeholders, enabling regular review of progress and swift adjustments based on feedback.

In each Agile sprint, functional features such as order placement, payment processing, and report generation were developed incrementally. Frequent demonstrations of working modules were given to the restaurant’s management and staff, allowing for adjustments and refinements throughout the development process. This approach ensured that the system was tailored to the restaurant's specific needs while maintaining flexibility to accommodate changes in scope.

## 2.4 Software Development Tools

Several modern software development tools were used throughout the project to ensure the efficiency and quality of the Smart Restaurant Ordering System:

* **Programming Language**: The back end of the system was developed using **Laravel**, a PHP framework known for its robustness and scalability in web application development.
* **Front-End Development**: For the front-end development, the **TALL stack** (Tailwind CSS, Alpine.js, Laravel, and Livewire) was used:
  + **Tailwind CSS** was utilized to create a responsive, modern, and clean user interface with minimal code. Tailwind’s utility-first approach allowed for rapid UI development with custom styles that ensured responsiveness across devices.
  + **Livewire** facilitated the development of dynamic user interfaces by allowing real-time interaction between the user and the server without the need for JavaScript. It simplifies the process of building reactive components like the order list and payment interfaces.
  + **Alpine.js** was used for adding minimal JavaScript functionality to the system’s front end. It allowed for the creation of dynamic, interactive components such as modals and accordions, enhancing the user experience without the complexity of larger JavaScript frameworks.
* **Additionally,** Vue.js is used for its ability to make lightweight individual components.
* **Database**: A **MySQL** database was used for storing order details, payment records, sales data, and other relevant information. Its compatibility with Laravel and ability to handle large amounts of data made it an ideal choice for this project.
* **Version Control**: **Git** and **GitHub** were used for version control and collaborative development, ensuring that all changes were tracked, and the project team could easily revert or merge code when necessary.
* **Testing Tools**: Automated testing was carried out using **PHPUnit** for unit testing, ensuring that the system components were functioning correctly during each development phase.

## 2.5 Testing Strategies

The testing phase was crucial in ensuring the system’s reliability and functionality. The following testing strategies were applied:

* **Unit Testing**: Each module of the system (such as order placement, payment, and report generation) was tested individually to ensure that the functionality worked as expected. Unit tests were run after each sprint using PHPUnit to catch bugs early in the development process.
* **Integration Testing**: Once individual modules were verified, integration tests were carried out to ensure that they worked well together. This was particularly important for features like payment processing, which interacts with multiple parts of the system (e.g., session data, front desk notifications).
* **User Acceptance Testing (UAT)**: After the system reached a stable state, it was deployed in a simulated environment where restaurant staff could interact with the system. Feedback from the UAT was used to fine-tune the system and fix any remaining issues before the final deployment.
* **Performance Testing**: Load tests were conducted to simulate peak-hour conditions and evaluate the system's performance under heavy usage, ensuring that it could handle high volumes of concurrent orders and payment requests.

## 2.6 Implementation Plan

The implementation plan for the Smart Restaurant Ordering System was broken down into several key phases to ensure smooth deployment:

* **Phase 1: Requirement Gathering and Planning** – Initial meetings with restaurant stakeholders to gather requirements and define the project scope.
* **Phase 2: System Design** – Designing the architecture, database schema, and user interface layouts based on gathered requirements.
* **Phase 3: Development** – Incremental development of system features using the Agile model. Each sprint focused on a particular module (e.g., order placement, payment processing, report generation).
* **Phase 4: Testing** – Comprehensive testing to identify and resolve bugs or performance bottlenecks.
* **Phase 5: Deployment and Training** – Deployment of the system to the restaurant’s environment, followed by staff training to ensure proper usage.

## 2.7 Chapter Summary

In this chapter, the methodological framework for developing the Smart Restaurant Ordering System was outlined. Through a combination of interviews, observations, and feedback, the project identified key operational inefficiencies that needed to be addressed. The Agile software development model was chosen to enable flexibility and continuous improvement throughout the development process. Key tools like Laravel, MySQL, and PHPUnit facilitated efficient development and testing. A multi-phase implementation plan was designed to ensure a successful rollout of the system, followed by thorough testing strategies to guarantee its reliability.

# Chapter 3: Analysis

## 3.1 Introduction

The analysis phase of the project involves understanding the existing operational procedures in the restaurant and identifying areas that require digital transformation to streamline and improve efficiency. This chapter outlines the key structural elements of the proposed system, including various diagrams to visualize system functionality and relationships between data entities. Through detailed analysis, the goal is to ensure that the Smart Restaurant Ordering System meets the needs of the restaurant while addressing existing challenges.

## 3.2 UML Diagrams

The **Unified Modeling Language (UML)** diagram provides a visual representation of the system's interaction between actors (such as customers, front desk staff, and kitchen staff) and the system components. It illustrates the various use cases within the system, such as placing an order, processing payments, canceling orders, and generating reports.

The UML diagrams—particularly the use case, class, and sequence diagrams—help provide a clear understanding of system behavior, making it easier to visualize the functional requirements.

### 3.2.1 Class Diagram

The most widely used UML diagram is the class diagram. It is the building block of all object-oriented software systems. We use class diagrams to depict the static structure of a system by showing the system’s classes, their methods and attributes. Class diagrams also help us identify relationships between different classes or objects.

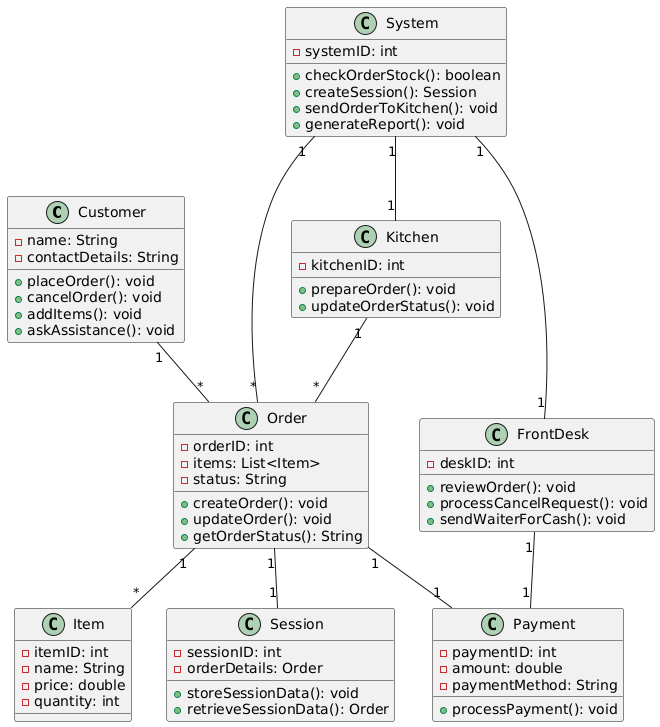


Figure 1.Class Diagram

### 3.2.2 Sequence Diagrams

Depicting the step-by-step process flow, such as when a customer places an order or when a report is generated by the system. These diagrams highlight the flow of control and messages between system actors and components.

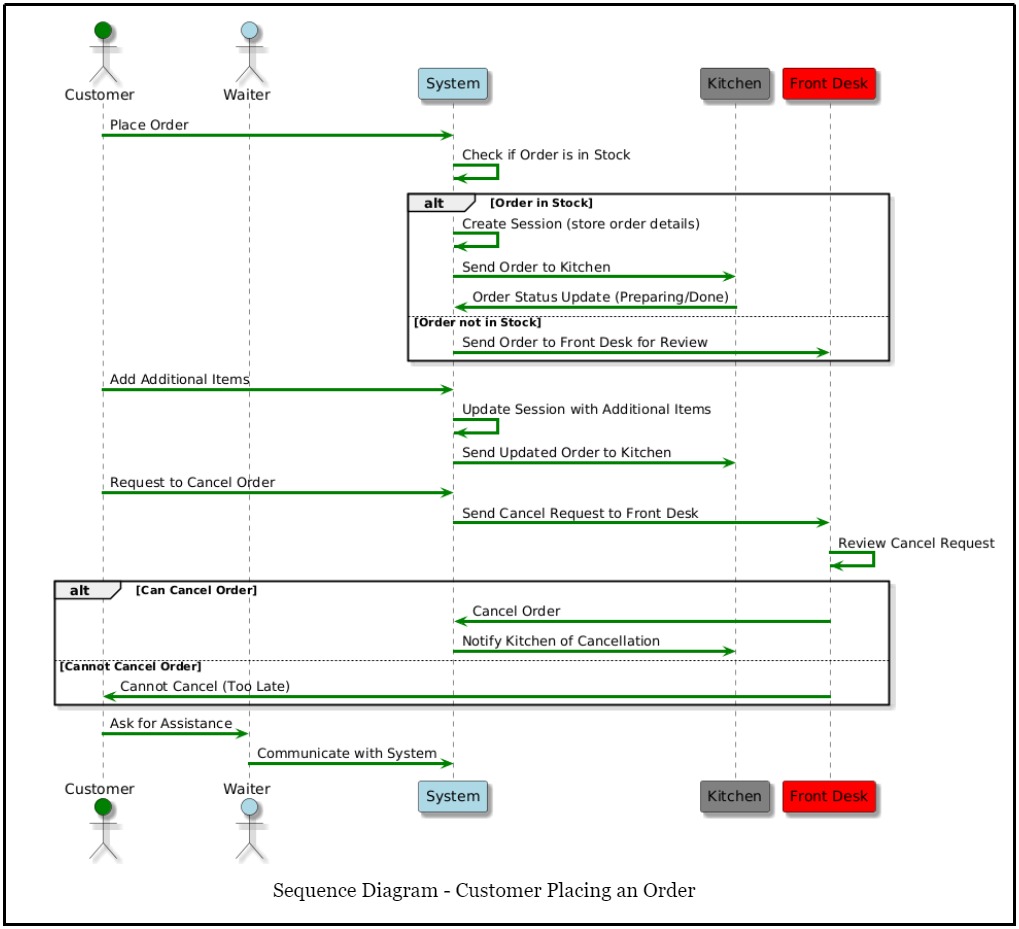
****

Figure 2. 1.Sequence Diagram - Customer Placing an Order

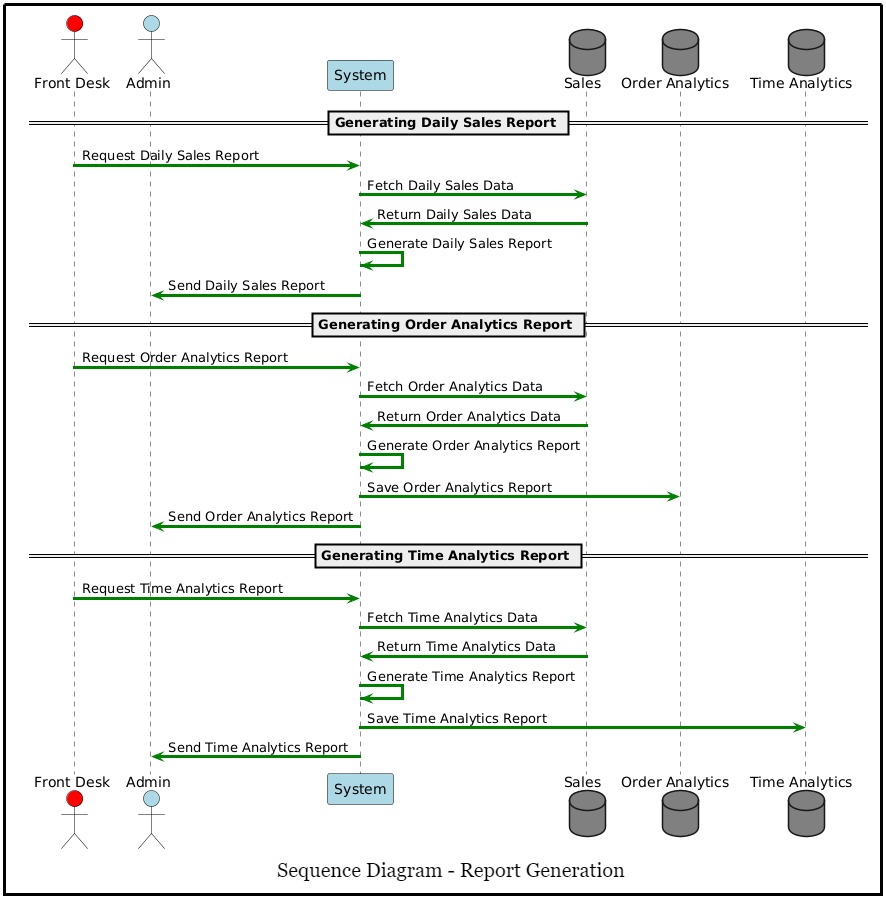
****

Figure 2. 2..Sequence Diagram - Report Genration

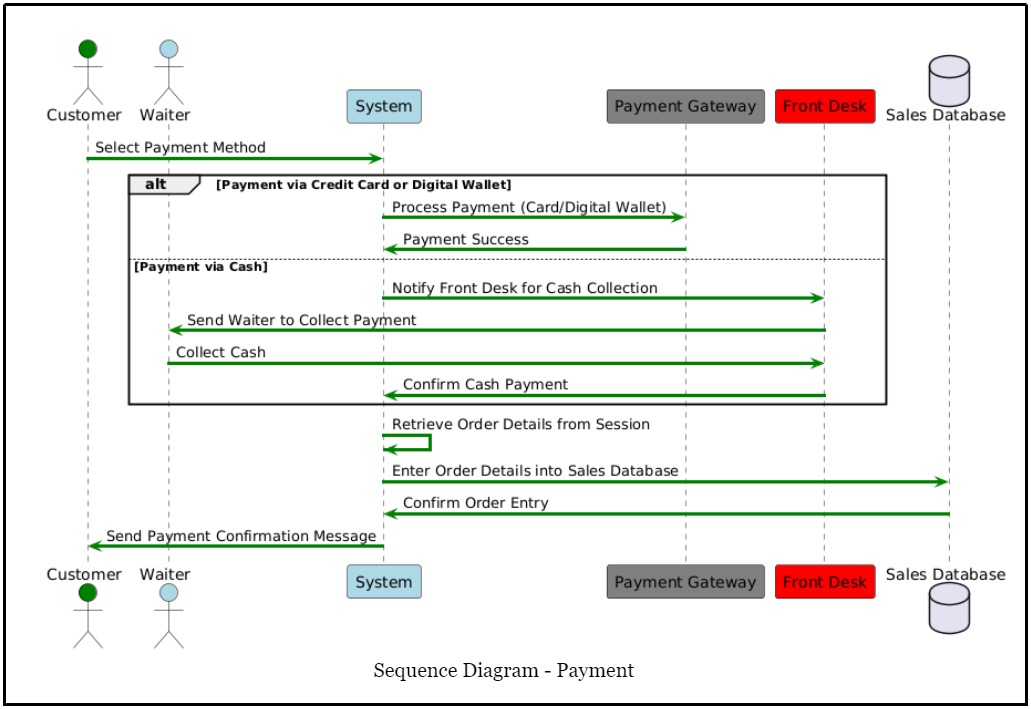
****

Figure 2. 3..Sequence Diagram - Payment

### 3.2.3 Use Case Diagram

Representing the interaction between system users and major functionalities such as placing orders, processing payments, and generating reports.

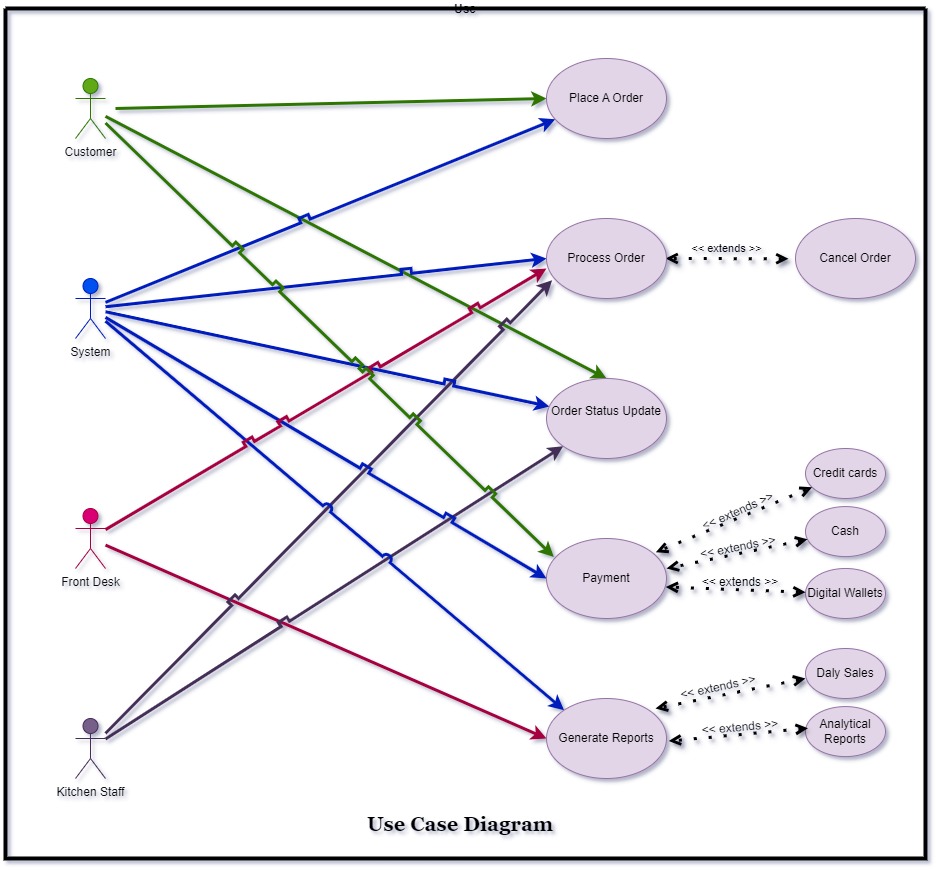
****

Figure 2.Use Case Diagram

## 3.3 Data Flow Diagrams

Data Flow Diagrams (DFD) provide a graphical representation of the data flow of a system that can be understood by both technical and non-technical users. The models enable software engineers, customers, and users to work together effectively during the analysis and specification of requirements.

A diagram of a restaurant system

Description automatically generated

Figure 4 1eee

Figure 3.0.Data Flow Diagrams - Level 0 1e

Figure 3.0.Data Flow Diagrams - Level 0 2

Figure 3.1.Data Flow Diagrams - Level 0

Figure 3. 0.Data Flow Diagrams - Level 0

A diagram of a process flow

Description automatically generated

Figure 3. 1.Data Flow Diagrams - Level 1

A diagram of a flowchart

Description automatically generated

Figure 3. 2.Data Flow Diagrams- Level 2 - Generate Reports

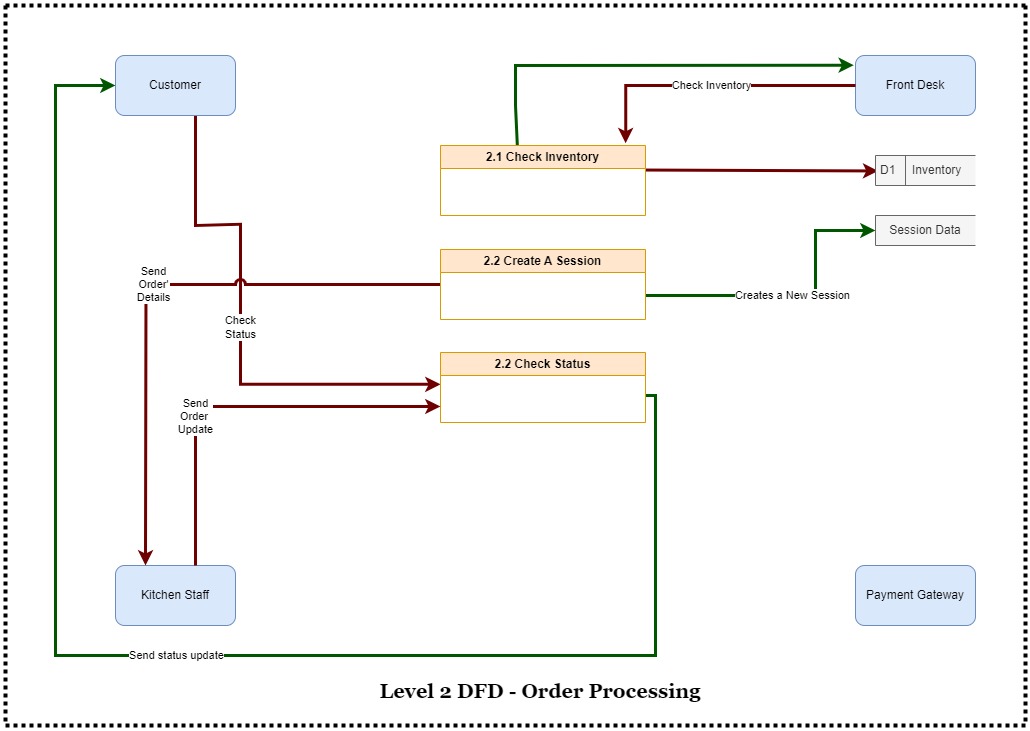


Figure 3. 3.Data Flow Diagrams- Level 2 - Order Processing

## 3.4 ER Diagram

The Entity-Relationship (ER) Diagram focuses on the data structure of the system, identifying the relationships between different entities such as customers, orders, items, payments, and reports. The primary entities and their relationships include:

* **Customer**: Contains customer information like name, contact details, and order history.
* **Order**: Links the customer to the items they order and stores details such as order status, date, and payment information.
* **Item**: Represents the menu items available for ordering, including stock status and pricing.
* **Payment**: Handles various payment methods (credit card, cash, digital wallet) and stores transaction details.
* **Report**: Summarizes the sales, order analytics, and time analytics for management purposes

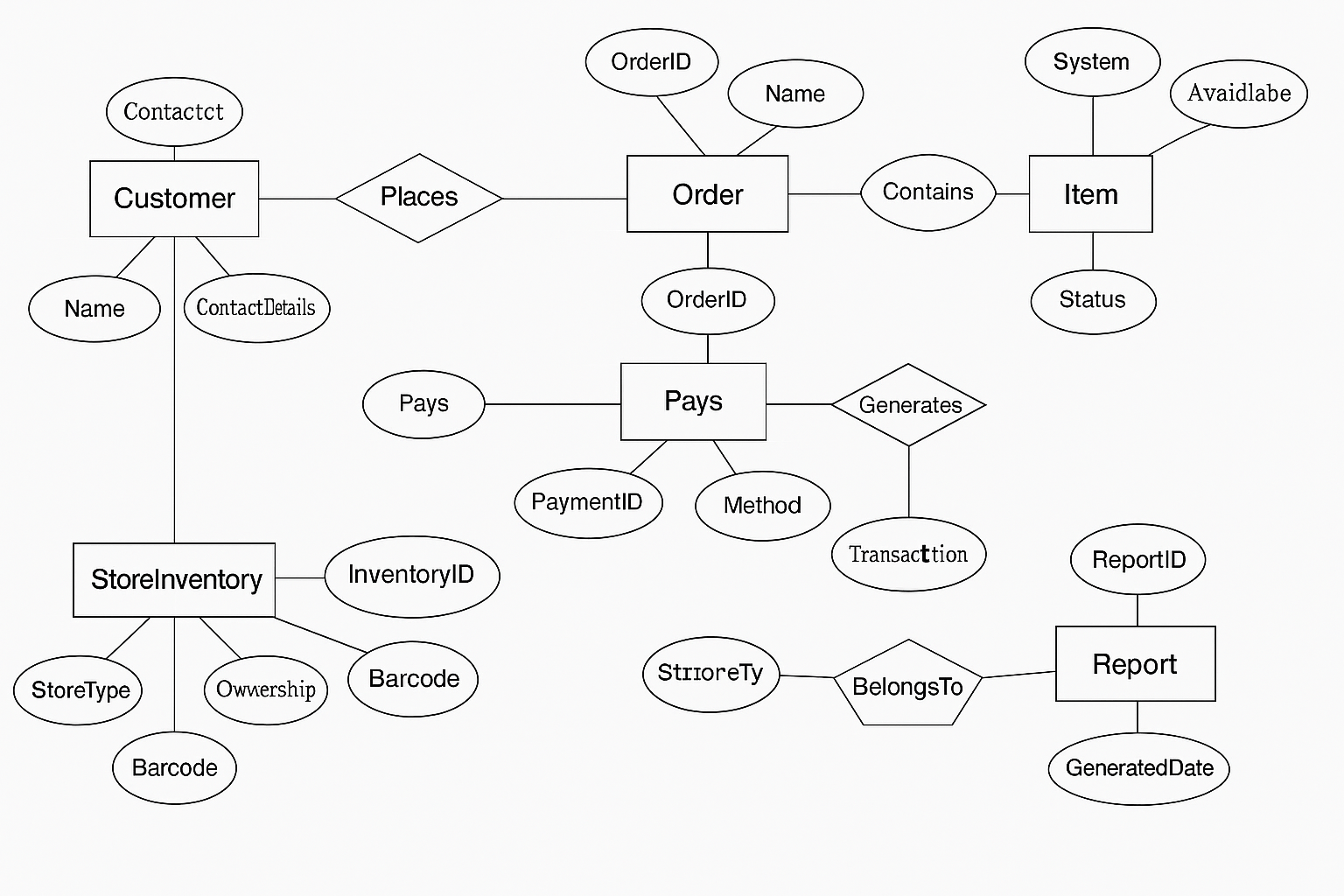


Figure 4. 1.ER Diagram.

.

## 3.5 Chapter Summary

This chapter presented the system's analysis using both UML and ER diagrams to capture the overall structure, processes, and relationships in the Smart Restaurant Ordering System. The diagrams serve as a communication tool between developers, stakeholders, and end-users, ensuring clarity in design and functionality. The insights gained from these models form the foundation for the system's implementation and help ensure that all key requirements are met.

# Chapter 4: Solution design

## 4.1 Introduction

The Solution Design chapter outlines the technical architecture and design choices that will be implemented to build the system. This chapter provides a comprehensive description of how the solution will be structured, including system architecture, interface designs, and database structures. The solution will follow best practices for modularity, scalability, and maintainability while ensuring user-friendly interaction and efficient data management.

In this chapter, we explore System Design, which includes the overall structure of the system, including its core components, interactions, and data flow. Additionally, the Interface Design section focuses on the user experience (UX) and user interface (UI) aspects of the system. The Database Design section covers the storage and retrieval of data necessary for the system's operation.

## 4.2 System Design

The system utilizes multiple physical devices for different stages of order processing and management. These devices serve specific functions for customers, kitchen staff, and restaurant administration:

1. **Display Console** (Customer Interface)



* **Device**: PC or tablet with KIOSK mode activated.
* **Purpose**: Used by customers to browse the menu, customize orders, and submit orders.
* **Location**: Their table.
* **Functionality**: Displays an interactive menu, allows order customization (e.g., choosing extras), and confirms total price.

1. **Front Desk Display** (Desk/Admin Interface)

A person at a desk using a tablet

Description automatically generated

* **Device**: PC or tablet with internet access.
* **Purpose**: Used by staff at the front desk to review and process incoming customer orders.
* **Location**: Front desk.
* **Functionality**: Displays customer orders, allows staff to review and adjust (e.g., unavailable items), and sends finalized orders to the kitchen.

1. **Kitchen Display** (Kitchen Staff Interface)

A computer screen in a kitchen

Description automatically generated

* **Device**: PC or tablet with internet access.
* **Purpose**: Used by kitchen staff to receive and manage order details.
* **Location**: Kitchen.
* **Functionality**: Displays incoming orders, preparation instructions, and updates the order status (e.g., "In Progress", "Ready").

### 4.2.1 Process Flow

The order processing workflow is structured in five key stages to ensure smooth communication between customers, front desk staff, kitchen staff, and the Laravel-based admin system:

1. **Customer Places an Order Using the Display Console**

* **Step 1: Menu Interaction**
* The customer interacts with the **Display Console**, which shows the restaurant's menu.
* The interface allows the customer to:
  1. Browse available items.
  2. Select quantities.
  3. Customize orders (e.g., adding extra toppings or specific cooking instructions).
  4. View the total price.
* **Step 2: Order Submission**
* Once the customer finalizes their selection, they confirm the order with the **Display Console**.
* The order details are wirelessly transmitted to the **central server** (Laravel-based backend) for processing.

1. **Order Processing at the Desk (On Errors)**

* **Step 3: Order Review**
* The **Front Desk Display** receives the order for review. The front desk staff can make necessary adjustments, such as removing unavailable items.
* Once adjustments are made (if any), the order is confirmed, and it proceeds to the kitchen.

1. **Sending Orders to the Kitchen**

* **Step 4: Kitchen Notification**
* Once the desk staff confirms the order, the **Kitchen Display** receives the order.
* Kitchen staff view the item names, quantities, and any specific preparation instructions.
* **Step 5: Order Preparation**
* The kitchen staff prepares the food according to the order specifications.
* As each dish is prepared, kitchen staff update the system with the order status (e.g., "In Progress" or "Ready").

1. **Managing Orders via the Laravel Admin System**

* **Step 6: Real-Time Monitoring**
* The **Laravel-based admin system** continuously monitors the status of all orders.
* It tracks every step of the process, ensuring that restaurant staff are aware of the progress of each order.
* **Step 7: Order Modifications or Cancellations**
  + Customers can modify or cancel their order after placement using the **Display Console**.
  + The Laravel system processes these requests, and notifications are sent to the relevant staff to update the order.
* **Step 8: Completion and Delivery**
* Once the order is marked as "Ready" by the kitchen staff, the front desk or delivery personnel are notified to serve or deliver the order.
* **Step 9: Payment Process**
* After order confirmation, the customer selects a payment method through the **Display Console**:

1. **Credit Card**: The customer can swipe or tap their card on a terminal.
2. **Digital Wallets**: Popular digital wallets such as Apple Pay or Google Pay can be used for contactless payment.
3. **Cash**: If paying by cash, the desk staff handles the payment when the order is delivered.
4. **Post-Order Management and Analytics**

* **Step 10: Data Collection**
* The Laravel admin system collects data for each order, including item popularity, time of order, and customer preferences.
* **Step 11: Reporting and Analytics**
* Reports can be generated to analyze trends, such as peak order times, popular items, and average customer spending.
* The data helps improve restaurant operations, staffing, and marketing strategies.

## 4.3 Interface Design

The user interfaces (UI) and overall user experience (UX) of the Smart Restaurant Ordering System have been crafted using TALL stack (Tailwind CSS, Alpine.js, Laravel, Livewire) and Vue.js. These technologies enable the development of responsive, dynamic, and efficient interfaces that provide a seamless experience for both customers and staff.

1. Interface 1 - Landing page

****

The welcome page of Café Modern Bites' website, built using HTML, Bootstrap, and Tailwind CSS, serves as an inviting entry point for visitors. It features a beautifully presented sushi roll to entice potential customers.   The navigation panel at the top provides easy access to various sections of the site. Users can explore the menu items by clicking on the "Menu" link, add their selections to the cart via the "Cart" icon, and proceed to finalize their orders through the "Checkout" section. The seamless integration of these features ensures a smooth user experience, allowing customers to effortlessly browse, select, and review their orders before completing the checkout process.

**function setupNavigation():**

**if "Home" is clicked:**

**display WelcomePage**

**if "Menu" is clicked:**

**display MenuPage**

**if "Cart" is clicked:**

**display CartPage**

**if "Checkout" is clicked:**

**display CheckoutPage**

**if ShoppingCartIcon is clicked:**

**display CartPage**

**// Call to action button functionality**

**function setupCallToAction():**

**if "Order Now" is clicked:**

**redirect to MenuPage**

**// Chat support**

**function initializeChatSupport():**

**display ChatIcon at BottomLeft**

**if ChatIcon is clicked:**

**open ChatSupportDialog**

**// Main execution flow**

**function main():**

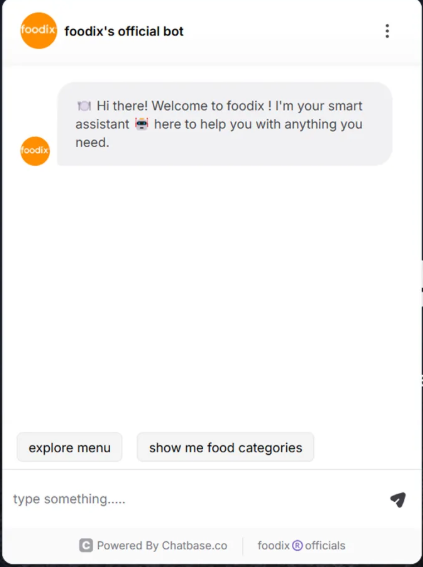
**initializeWebsite()**

**setupNavigation()**

**setupCallToAction()**

**initializeChatSupport()**

1. Interface 2 -Chatbot Popup Menu



The chatbot popup on Café Modern Bites' website is a user-friendly feature designed to enhance the customer experience. When users click the bot icon on the welcome page, the chatbot opens and greets them with a welcoming message. It offers quick access buttons like "explore menu" and "show me food categories," allowing customers to easily navigate through the available options.

This intelligent assistant can provide detailed information about specific items, their quantities, and even check the restaurant's opening and closing hours. By interacting with the chatbot, users can efficiently find what they're looking for without navigating multiple pages, making the ordering process faster and more convenient. The chatbot, powered by Chatbase.co, ensures that customers receive prompt and accurate assistance, contributing to a seamless and enjoyable online dining experience.

**function foodixChatbot():**

**displayMessage("👋 Hi there! Welcome to foodix ! I'm your smart assistant 🤖 here to help you with anything you need.")**

**showButton("explore menu")**

**showButton("show me food categories")**

**// Listen for user input**

**waitForUserInput()**

**// Handle user button clicks**

**function handleButtonClick(buttonText):**

**if buttonText == "explore menu":**

**displayMenuOptions()**

**if buttonText == "show me food categories":**

**displayFoodCategories()**

**// Handle text input from user**

**function handleUserTextInput(userText):**

**if containsKeywords(userText, ["menu", "food", "eat"]):**

**displayMenuOptions()**

**else if containsKeywords(userText, ["hours", "open", "close", "time"]):**

**displayBusinessHours()**

**else if containsKeywords(userText, ["reservation", "book", "table"]):**

**handleReservationRequest()**

**else if containsKeywords(userText, ["order", "delivery", "pickup"]):**

**handleOrderOptions()**

**else if containsKeywords(userText, ["ingredient", "allergy", "dietary"]):**

**handleDietaryQuestions()**

**else:**

**offerGeneralHelp()**

**// Display menu options**

**function displayMenuOptions():**

**displayMessage("Here's our menu. Would you like to see specific categories?")**

**showFoodCategories(["Appetizers", "Main Courses", "Sushi", "Desserts", "Beverages"])**

**// Display food categories**

**function displayFoodCategories():**

**displayMessage("We offer the following categories:")**

**showFoodCategories(["Appetizers", "Main Courses", "Sushi", "Desserts", "Beverages"])**

**displayMessage("Which category would you like to explore?")**

**// Handle business hours inquiry**

**function displayBusinessHours():**

**displayMessage("We are open from 11:00 AM to 10:00 PM Monday through Thursday, and 11:00 AM to 11:00 PM Friday through Sunday.")**

**// Handle reservation request**

**function handleReservationRequest():**

**displayMessage("I'd be happy to help with a reservation. How many people will be dining and what date and time would you prefer?")**

**// Wait for response and process reservation**

**// Handle order options**

**function handleOrderOptions():**

**displayMessage("Would you like to place an order for delivery or pickup?")**

**showButton("Delivery")**

**showButton("Pickup")**

**// Handle dietary questions**

**function handleDietaryQuestions():**

**displayMessage("Our staff can accommodate various dietary restrictions. Please let me know your specific concerns, and I'll provide information about our menu items.")**

**// General help function**

**function offerGeneralHelp():**

**displayMessage("I can help you with our menu, business hours, reservations, ordering, or answer questions about our food. What would you like to know?")**

**// Main chatbot event loop**

**function chatbotEventLoop():**

**while chatbotIsActive:**

**if userClickedButton:**

**buttonText = getButtonText()**

**handleButtonClick(buttonText)**

**if userTypedMessage:**

**userText = getUserText()**

**handleUserTextInput(userText)**

1. Interface 3- Menu Page

A screenshot of a menu

Description automatically generated

The "Choose Menu" page of Café Modern Bites offers a clean, user-friendly layout where customers can browse various food categories. Each category displays high-quality images of the dishes, along with short descriptions highlighting key ingredients and preparation methods. Prices are clearly listed, allowing users to make informed decisions. Customers can easily add items to their cart directly from the menu, ensuring a seamless and efficient ordering experience.

**function handleCategoryClick(categoryName):**

**setActiveCategory(categoryName)**

**displayMenuItems(getItemsForCategory(categoryName))**

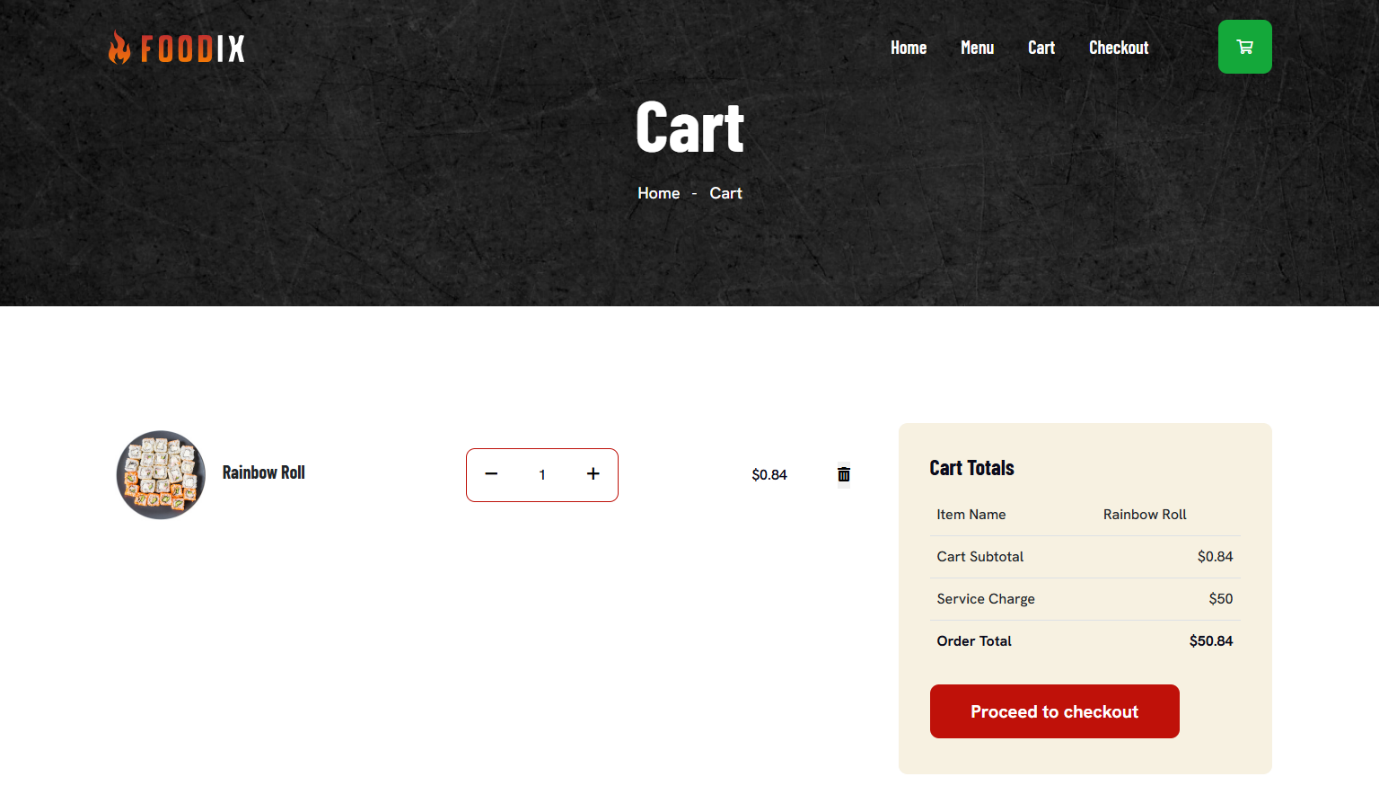
**function addItemToCart(item):**

**addToUserCart(item)**

**updateCartIcon()**

**showConfirmationMessage(item.name + " added to cart!")**

4.Interface 4-Cart Page



The "Cart" page of Café Modern Bites provides customers with a clear and organized view of their selected items. Users can easily adjust the quantity of each item according to their preferences, ensuring they have exactly what they want. The page displays the price of each item, allowing customers to keep track of their spending.

On the right side of the page, the "Cart Totals" card summarizes the order details. It shows the subtotal for all items in the cart, any applicable service charges, and the total order amount. This comprehensive overview helps customers review their selections and understand the final cost before proceeding to checkout. The "Proceed to Checkout" button at the bottom of the card makes it simple for users to finalize their orders, ensuring a smooth and efficient shopping experience.

**function initializeCartPage():**

**// Cart contents section**

**displayCartItems(getUserCart())**

**// Cart summary section**

**displayCartSummary(calculateCartTotals())**

**function displayCartItems(cartItems):**

**clearCartItemsDisplay()**

**if cartItems.length == 0:**

**displayEmptyCartMessage()**

**return**

**for each item in cartItems:**

**createCartItemRow(item)**

**function createCartItemRow(item):**

**row = createNewElement("div", "cart-item-row")**

**// Remove button**

**addRemoveButtonToRow(row, function() { removeItemFromCart(item.id) })**

**appendToCartItemsDisplay(row)**

**function displayCartSummary(totals):**

**summary = createNewElement("div", "cart-totals")**

**for each item in getUserCart():**

**addSummaryItem(summary, item.name, item.price)**

**addSummaryLine(summary, "Cart Subtotal", totals.subtotal)**

**addSummaryLine(summary, "Service Charge", totals.serviceCharge)**

**addSummaryLine(summary, "Order Total", totals.total)**

**appendToCartSummaryDisplay(summary)**

**function updateItemQuantity(itemId, newQuantity):**

**if newQuantity <= 0:**

**removeItemFromCart(itemId)**

**else:**

**setCartItemQuantity(itemId, newQuantity)**

**refreshCartDisplay()**

**updateCartTotals()**

**function removeItemFromCart(itemId):**

**deleteCartItem(itemId)**

**refreshCartDisplay()**

**updateCartTotals()**

**function calculateCartTotals():**

**subtotal = 0**

**for each item in getUserCart():**

**subtotal += item.price \* item.quantity**

**serviceCharge = calculateServiceCharge(subtotal)**

**total = subtotal + serviceCharge**

**return {**

**subtotal: formatCurrency(subtotal),**

**serviceCharge: formatCurrency(serviceCharge),**

**total: formatCurrency(total)**

**}**

**function proceedToCheckout():**

**navigateToPage("checkout")**

5.Interface 5 – Checkout Page

A screenshot of a computer

Description automatically generated

The "Checkout" section of Café Modern Bites' website serves multiple purposes, ensuring a comprehensive and user-friendly experience for customers. On this page, users can review the items they have added to their cart, along with the associated prices, providing a final check before completing their order.

Customers are given the flexibility to choose their preferred payment method from options such as Cash on Delivery, Direct Bank Transfer, and PayPal. This versatility caters to various customer preferences, making the payment process convenient and accessible.

Additionally, the checkout page includes a billing details section where customers can fill in their personal information. By entering their email address, customers have the option to subscribe to the restaurant's email service. This subscription allows them to receive updates and promotions directly, enhancing their engagement with the café.

Overall, the checkout page is designed to streamline the ordering process, offering a clear summary of the cart contents, flexible payment options, and an opportunity for customers to stay connected with Café Modern Bites through email updates.

**function initializeOrderSummary()**

**summary = createNewElement("div", "order-summary")**

**addSummaryTitle(summary, "Your order")**

**cartItems = getUserCart()**

**for each item in cartItems:**

**addSummaryItem(summary, item.name, formatCurrency(item.price \* item.quantity)**

**cartTotals = calculateCartTotals()**

**addSummaryLine(summary, "Cart Subtotal", cartTotals.subtotal)**

**addSummaryLine(summary, "Service Charge", cartTotals.serviceCharge)**

**addSummaryLine(summary, "Order Total", cartTotals.total)**

**appendToRightColumn(summary)**

**function initializePaymentMethods()**

**paymentSection = createNewElement("div", "payment-methods")**

**addSectionTitle(paymentSection, "Payment Method")**

**addPaymentOption(paymentSection, "Cash On Delivery", "cashOnDelivery", true,**

**"Pay with cash remains a simple and reliable choice, transcending the complexities of modern finance.")**

**addPaymentOption(paymentSection, "Direct Bank Transfer", "bankTransfer", false)**

**addPaymentOption(paymentSection, "Paypal", "paypal", false)**

**appendToRightColumn(paymentSection)**

**function addPaymentOption(container, label, value, isDefault, description):**

**option = createNewElement("div", "payment-option")**

**addRadioButton(option, "paymentMethod", value, label, isDefault**

**if description:**

**addDescription(option, description)**

**appendToContainer(container, option)**

**function handleCheckoutSubmission()**

**formData = {**

**firstName: getInputValue("firstName"),**

**lastName: getInputValue("lastName"),**

**city: getInputValue("city"),**

**email: getInputValue("email"),**

**phone: getInputValue("phone"),**

**paymentMethod: getSelectedPaymentMethod()**

**}**

**if not validateCheckoutForm(formData):**

**showValidationErrors()**

**return**

**orderData = processOrder(formData)**

**generateReceipt(orderData)**

**showOrderConfirmation(orderData.orderId)**

**6.Interface - Bill Page**

A screenshot of a menu

Description automatically generated

The receipt from Café Modern Bites provides a clear and concise summary of the customer's order. It lists the quantity and price of each item, along with the subtotal, service charge, and final order total. The payment method is also specified, ensuring transparency in the transaction.

At the bottom, there is a section for the customer's signature, confirming their agreement to pay the total amount as per the card issuer's agreement. This structured layout ensures that all necessary details are easily accessible and verifiable, enhancing the overall customer experience.

**function addPaymentOption(container, label, value, isDefault, description):**

**option = createNewElement("div", "payment-option")**

**addRadioButton(option, "paymentMethod", value, label, isDefault)**

**if description:**

**addDescription(option, description)**

**appendToContainer(container, option)**

**function handleCheckoutSubmission():**

**formData = {**

**firstName: getInputValue("firstName"),**

**lastName: getInputValue("lastName"),**

**city: getInputValue("city"),**

**email: getInputValue("email"),**

**function generateReceipt(orderData):**

**receipt = createNewElement("div", "receipt")**

**addRestaurantHeader(receipt, {**

**name: "Café Modern Bites",**

**address: "1500 Main Ave",**

**cityState: "Long Beach, CA 90712",**

**phone: "505-303-2993"**

**})**

**addOrderDetails(receipt, {**

**date: getCurrentDateTime(),**

**terminal: "TERMINAL 1"**

**})**

**addItemsTable(receipt, orderData.items)**

**addReceiptTotals(receipt, {**

**subtotal: orderData.subtotal,**

**serviceCharge: orderData.serviceCharge,**

**total: orderData.total**

**})**

**addPaymentMethod(receipt, orderData.paymentMethod)**

**addSignatureSection(receipt)**

**if orderData.printReceipt:**

**printReceipt(receipt)**

**else:**

**displayReceipt(receipt)**

**saveReceiptToOrderHistory(orderData.orderId, receipt)**

**function addRestaurantHeader(receipt, restaurantInfo):**

**header = createNewElement("div", "restaurant-header")**

**addTextElement(header, "restaurant-name", restaurantInfo.name)**

**addTextElement(header, "restaurant-address", restaurantInfo.address)**

**addTextElement(header, "restaurant-city-state", restaurantInfo.cityState)**

**addTextElement(header, "restaurant-phone", restaurantInfo.phone)**

**appendToReceipt(receipt, header)**

**function addOrderDetails(receipt, orderInfo):**

**details = createNewElement("div", "order-details")**

**addTextElement(details, "order-datetime", orderInfo.date)**

**addTextElement(details, "order-terminal", orderInfo.terminal)**

**appendToReceipt(receipt, details)**

**function addItemsTable(receipt, items):**

**table = createTable(["Qty", "Item", "Price"])**

**for each item in items:**

**addTableRow(table, [**

**item.quantity,**

**item.name,**

**formatCurrency(item.price \* item.quantity)**

**])**

**appendToReceipt(receipt, table)**

**function addReceiptTotals(receipt, totals):**

**totalsSection = createNewElement("div", "receipt-totals")**

**addTextRow(totalsSection, "CART SUBTOTAL", totals.subtotal)**

**addTextRow(totalsSection, "SERVICE CHARGE", totals.serviceCharge)**

**addTextRow(totalsSection, "ORDER TOTAL", totals.total)**

**appendToReceipt(receipt, totalsSection)**

**function addPaymentMethod(receipt, paymentMethod):**

**paymentSection = createNewElement("div", "payment-method")**

**addTextElement(paymentSection, "payment-type-label", "PAYMENT TYPE: ")**

**addTextElement(paymentSection, "payment-type-value", paymentMethod)**

**appendToReceipt(receipt, paymentSection)**

**function addSignatureSection(receipt):**

**signatureSection = createNewElement("div", "signature-section")**

**addTextElement(signatureSection, "signature-label", "CUSTOMER SIGNATURE")**

**addSignatureLine(signatureSection)**

**addTextElement(signatureSection, "signature-agreement",**

**"I agree to pay the above total amount according to the card issuer agreement.")**

**appendToReceipt(receipt, signatureSection)**

7.Interface 7-Dashboard Page

A screenshot of a computer

Description automatically generated

The dashboard of the restaurant management system provides a comprehensive overview and control over various functionalities. The "Menu Items Management" section allows administrators to manage the entire menu efficiently. Key features include:

Total Menu Items: Displays the total number of items currently listed on the menu.

Active Items: Shows the count of items that are active and available for customers to order.

Categories: Indicates the number of different categories into which menu items are organized.

Administrators can add new items to the menu using the "Add New Item" button, ensuring the menu stays up to date with fresh offerings. Each item in the list has options for editing or deleting, providing flexibility in managing the menu content. This streamlined interface ensures that the restaurant's menu is always accurate and well-maintained, enhancing the overall operational efficiency.

**function initializeMenuItemsManagement():**

**setPageTitle("Menu Items Management")**

**addActionButton("Add New Item", function() { openAddItemModal() })**

**displaySummaryCards()**

**initializeMenuItemsTable()**

**loadMenuItems()**

**function displaySummaryCards():**

**container = createNewElement("div", "summary-cards-container")**

**totalItems = getTotalMenuItemsCount()**

**addSummaryCard(container, "Total Menu Items", totalItems, "blue")**

**activeItems = getActiveItemsCount()**

**addSummaryCard(container, "Active Items", activeItems, "green")**

**categories = getCategoriesCount()**

**addSummaryCard(container, "Categories", categories, "yellow")**

**appendToMainContent(container)**

**function loadMenuItems():**

**menuItems = fetchMenuItems()**

**clearTableContent("menu-items-table")**

**for each item in menuItems:**

**addItemToTable(item)**

**function addItemToTable(item):**

**row = createTableRow()**

**addTableCell(row, item.id)**

**addImageCell(row, item.imageUrl, item.name)**

**addTableCell(row, item.name)**

**addTableCell(row, item.category)**

**addTableCell(row, formatCurrency(item.price))**

**actionsCell = createTableCell()**

**addEditButton(actionsCell, function() { openEditItemModal(item.id) })**

**addDeleteButton(actionsCell, function() { confirmDeleteItem(item.id) })**

**appendToRow(row, actionsCell)**

**appendToTable("menu-items-table", row)**

**function saveNewMenuItem():**

**formData = getFormData("add-menu-item-form")**

**if not validateMenuItemForm(formData):**

**showValidationErrors()**

**return**

**result = saveMenuItem(formData)**

**if result.success:**

**closeModal()**

**showSuccessMessage("Menu item added successfully")**

**loadMenuItems()**

**else:**

**showErrorMessage("Failed to add menu item: " + result.error)**

**function openEditItemModal(itemId):**

**item = getMenuItemById(itemId)**

**modal = createModal("Edit Menu Item")**

**form = createForm("edit-menu-item-form")**

**addFormField(form, "name", "Name", "text", { value: item.name })**

**addFormField(form, "category", "Category", "select", getCategoryOptions(), { value: item.category })**

**addFormField(form, "price", "Price", "number", { min: 0, step: 0.01, value: item.price })**

**addFormField(form, "description", "Description", "textarea", { value: item.description })**

**addFormField(form, "image", "Image", "file", { accept: "image/\*" })**

**addHiddenField(form, "id", item.id)**

**addCurrentImage(form, item.imageUrl)**

**addFormButton(form, "Cancel", function() { closeModal() }, "secondary")**

**addFormButton(form, "Save", function() { updateMenuItem() }, "primary")**

**appendToModal(modal, form)**

**showModal(modal)**

8. Interface 8-Orders Section

A screenshot of a computer

Description automatically generated

The "Orders" section of the restaurant dashboard provides a detailed overview of all customer orders. Each order is listed with its unique ID, quantity, price, subtotal, service charge, total amount, payment method, and timestamp. This comprehensive layout allows administrators to track and manage orders efficiently.

Administrators can view the number of items ordered, their respective prices, and the total cost for each order, including any additional service charges. The payment methods used by customers are also clearly indicated, ensuring transparency in financial transactions.

The timestamp for each order helps in monitoring the order flow and managing delivery or preparation times effectively. This section is crucial for maintaining an organized workflow, enabling quick access to order details, and facilitating smooth operations within the restaurant.

The "Order Payment and Sales Insights" section of the restaurant dashboard offers valuable analytics to help administrators make informed decisions. The "Payment Methods Distribution" pie chart illustrates the proportion of orders paid through different methods, such as Cash on Delivery, Direct Bank Transfer, and PayPal. This visualization aids in understanding customer payment preferences and optimizing financial processes.

The "Sold Items" bar graph provides a clear view of the quantities sold for various items. It helps identify popular menu options and trends in customer choices, enabling better inventory management and strategic planning.

Additionally, the "Download Report" button allows administrators to export this data for further analysis or record-keeping. These insights are crucial for enhancing operational efficiency and tailoring services to meet customer needs effectively.

**function loadOrders():**

**orders = fetchOrders()**

**clearTableContent("orders-table")**

**for each order in orders:**

**addOrderToTable(order)**

**function addOrderToTable(order):**

**row = createTableRow()**

**addTableCell(row, order.id)**

**addTableCell(row, order.itemName)**

**addTableCell(row, order.quantity)**

**addTableCell(row, formatCurrency(order.price))**

**addTableCell(row, formatCurrency(order.subtotal))**

**addTableCell(row, formatCurrency(order.serviceCharge))**

**addTableCell(row, formatCurrency(order.total))**

**addTableCell(row, order.paymentMethod)**

**addTableCell(row, formatDateTime(order.timestamp))**

**appendToTable("orders-table", row)**

**function initializeOrdersAnalytics():**

**container = createNewElement("div", "orders-analytics-container")**

**addSectionTitle(container, "Order Payment and Sales Insights")**

**columnsContainer = createTwoColumnLayout()**

**paymentMethodsData = getPaymentMethodsDistribution()**

**addPieChart(columnsContainer, "Payment Methods Distribution", paymentMethodsData)**

**soldItemsData = getSoldItemsData()**

**addBarGraph(columnsContainer, "Sold Items", soldItemsData)**

**appendToContainer(container, columnsContainer)**

**addButton(container, "Download Report", function() { downloadOrdersReport() })**

**appendToMainContent(container)**

**function getPaymentMethodsDistribution():**

**return [**

**{ label: "Cash On Delivery", value: 35 },**

**{ label: "Direct Bank Transfer", value: 25 },**

**{ label: "Paypal", value: 40 }**

**]**

**function getSoldItemsData():**

**return [**

**{ label: "Rainbow Roll", value: 45 },**

**{ label: "Tuna Nigiri", value: 30 },**

**{ label: "BBQ Pizza", value: 25 },**

**{ label: "Mushroom Pizza", value: 15 },**

**{ label: "Chicken", value: 20 }**

**]**

**function downloadOrdersReport():**

**reportData = generateOrdersReport()**

**downloadFile("orders\_report.csv", reportData, "text/csv")**

**showSuccessMessage("Report downloaded successfully")**

**9.Interface -Item Added Section**

A screenshot of a computer

Description automatically generated

The "Add New Menu Item" form in the restaurant system allows administrators to easily add new items and categories to the database. This feature ensures that the menu is always up-to-date with the latest offerings.

Administrators can input details such as the item name, category, price, and description. They can also upload an image of the item to enhance its visual appeal on the menu. Once all the necessary information is filled out, clicking the "Save Item" button will add the new item to the database.

This functionality streamlines the process of updating the menu, making it simple for administrators to manage and expand the restaurant's offerings efficiently.

**function displayAddNewMenuItemForm()**

**display "Add New Menu Item" with close button "X"**

**display label "Item Name"**

**create input field for itemName**

**display label "Category"**

**create input field for category**

**display label "Price"**

**create input field for price**

**display label "Description"**

**create text area for description**

**display label "Image"**

**create file input with button "Choose File" and text "No file chosen"**

**create button "Close"**

**create button "Save Item"**

**if "Close" button clicked**

**closeForm()**

**if "Save Item" button clicked**

**if validateForm()**

**saveMenuItem(itemName, category, price, description, image)**

**showSuccessMessage("Menu item added successfully")**

**closeForm()**

**else**

**showErrorMessage("Please fill in all required fields")**

**endif**

**endif**

**endfunction**

**function validateForm()**

**if itemName is empty or price is empty**

**return false**

**endif**

**return true**

**endfunction**

**function saveMenuItem(itemName, category, price, description, image)**

**create new menuItem object**

**menuItem.name = itemName**

**menuItem.category = category**

**menuItem.price = price**

**menuItem.description = description**

**menuItem.image = image**

**addToDatabase(menuItem)**

**endfunction**

**4.4 AL and ML Implementation**

The AI/ML components are primarily focused on enhancing the user experience through prediction models for operational insights, such as peak order times and popular menu items.

**Key Features of AI/ML Implementation**

1. **Prediction Model**:
   * A machine learning model is integrated to provide predictive insights into restaurant operations.
   * The model is implemented in Python and executed using the app.py script.
2. **Purpose of the ML Model**:
   * **Peak Time Prediction**: Determines busy hours based on historical data.
   * **Insights into Menu Items**: Identifies most popular orders based on sales data.
3. **Usage Instructions**:
   * **Branch Setup**: Users must switch to the ML branch to access the prediction-related functionality.
   * **Local Execution**:
     1. Clone the repository.
     2. Navigate to the ML branch.
     3. Install necessary Python dependencies from requirements.txt.
     4. Run the Python script using python app.py.
     5. The application runs on http://localhost:5000.
4. **Integration with Flask API**:
   * The Laravel backend communicates with a Flask API running locally to send data and retrieve predictions.
   * Example Routes:
     1. POST /peak-times: Sends order data to the Flask API for processing.
     2. GET /predict-peak-times: Fetches peak time predictions.
5. **Visualization**:
   * Prediction results are displayed in the frontend using charts and dropdowns.
   * **Visual Components**:
     1. A chart for peak hours.
     2. Dropdown menus for visualizing predictions in the UI.
6. **Frontend Integration**:
   * The predict.blade.php file dynamically shows peak order times fetched from the API.
   * Chart libraries like Chart.js are used for displaying insights.
7. **Dependencies**:
   * **Backend**: Laravel (PHP framework).
   * **Machine Learning**: Python with Flask API.
   * **Frontend**: Blade templates and JavaScript for rendering predictions.

**Strengths**

* **Integration of AI for Practical Use**:
  + Predictions, such as peak times, add value to restaurant management.
  + Seamless integration of Laravel and Flask ensures a smooth user experience.
* **Well-Documented Setup**:
  + Instructions in the README.md provide clear steps for running the ML model locally.
* **Visualization**:
  + Use of charts and dropdowns enhances user interaction and data interpretation.

## 4.5 Database Design

A screenshot of a computer

AI-generated content may be incorrect.

# Chapter 5: Conclusion

The Smart Restaurant Ordering System project has successfully developed a comprehensive digital solution that addresses the key operational challenges faced by modern restaurants. By transitioning from manual processes to an automated, integrated system, the project has delivered a platform that enhances efficiency, accuracy, and customer satisfaction across all aspects of restaurant operations.

The system effectively connects all stakeholders - customers, front desk staff, kitchen personnel, and management - through specialized interfaces designed for their specific needs. Customers can now place orders directly through an intuitive interface, while front desk staff and kitchen personnel receive real-time updates that streamline communication and order preparation. The management team benefits from comprehensive analytics and reporting features that support data-driven decision-making.

**5.1 Future Enhancements**

The Smart Restaurant Ordering System has established a solid foundation that can be further enhanced with additional features:

1. **Mobile Application Development**: Creating dedicated mobile apps for customers would enable remote ordering and reservation capabilities, expanding the restaurant's digital footprint.
2. **Inventory Management Integration**: Connecting the ordering system with inventory tracking would enable automatic stock updates and alert staff when items are running low.
3. **Customer Loyalty Program**: Implementing a points-based reward system would encourage repeat business and provide additional customer insights.
4. **Advanced AI-Driven Analytics**: Incorporating more sophisticated data analysis tools could offer predictive analytics for demand forecasting and personalized customer recommendations.
5. **Multi-location Support**: Expanding the system to support chain restaurants with multiple locations would provide centralized management capabilities while maintaining location-specific customizations.

# References

1. Herath, S. (n.d.). *Theoretical basis of ML feature selection – Summary*. Medium. <https://medium.com/image-processing-with-python/theoretical-basis-of-ml-feature-selection-summary-7670e190c8a3>
2. Herath, S. (n.d.). *Theoretical basis of ML model evaluation metrics – Summary*. Medium. <https://medium.com/image-processing-with-python/theoretical-basis-of-ml-model-evaluation-metrics-summary-3cae19129679>
3. Herath, S. (n.d.). *Data normalization & standardization*. Medium. <https://medium.com/image-processing-with-python/data-normalization-standardization-c06547531f05>
4. Herath, S. (n.d.). *Implementing linear regression with Python – A simple example*. Medium. <https://medium.com/image-processing-with-python/implementing-linear-regression-with-python-a-simple-example-8cba6d70c3fb>

# Appendix

**1.Python Model For Predictions :**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

# Load the dataset

file\_path = "sample\_data.csv" # Change to your dataset path

df = pd.read\_csv(file\_path)

# Convert 'order\_date' to datetime format

df["order\_date"] = pd.to\_datetime(df["order\_date"], format="%d/%m/%Y", errors="coerce")

# Extract day of the week from 'order\_date'

df["day\_of\_week"] = df["order\_date"].dt.day\_name()

# Convert 'order\_time' to datetime format and extract the hour of the day

df["order\_time"] = pd.to\_datetime(df["order\_time"], format="%H:%M:%S", errors="coerce")

df["hour\_of\_day"] = df["order\_time"].dt.hour # Extracts the hour

# --- Exploratory Data Analysis (EDA) ---

# 1. Item Popularity

item\_popularity = df.groupby("pizza\_name")["quantity"].sum().reset\_index()

item\_popularity = item\_popularity.sort\_values(by="quantity", ascending=False)

# 2. Peak Hours

peak\_hours = df.groupby("hour\_of\_day")["order\_id"].count().reset\_index()

peak\_hours = peak\_hours.rename(columns={"order\_id": "total\_orders"})

# 3. Peak Days

peak\_days = df.groupby("day\_of\_week")["order\_id"].count().reset\_index()

peak\_days = peak\_days.rename(columns={"order\_id": "total\_orders"})

# # Plot Peak Hours

# plt.figure(figsize=(10,5))

# sns.barplot(x=peak\_hours["hour\_of\_day"], y=peak\_hours["total\_orders"], palette="coolwarm")

# plt.xlabel("Hour of Day")

# plt.ylabel("Total Orders")

# plt.title("Peak Ordering Hours")

# plt.show()

# # Plot Peak Days

# plt.figure(figsize=(10,5))

# sns.barplot(x=peak\_days["day\_of\_week"], y=peak\_days["total\_orders"], order=[

# "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"

# ], palette="coolwarm")

# plt.xlabel("Day of the Week")

# plt.ylabel("Total Orders")

# plt.title("Peak Ordering Days")

# plt.show()

# --- 🟢 Prepare Data for ML Model ---

df\_ml = df.groupby(["hour\_of\_day", "day\_of\_week"])["order\_id"].count().reset\_index()

df\_ml = df\_ml.rename(columns={"order\_id": "total\_orders"})

# Convert categorical 'day\_of\_week' to numerical

df\_ml = pd.get\_dummies(df\_ml, columns=["day\_of\_week"], drop\_first=True)

# Train-Test Split

X = df\_ml.drop(columns=["total\_orders"])

y = df\_ml["total\_orders"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# --- 🟢 Train Initial Model ---

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

# Evaluate Initial Model

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

print(f"Initial Model - MAE: {mae:.2f}, RMSE: {rmse:.2f}")

# --- 🟢 Hyperparameter Tuning ---

param\_grid = {

"n\_estimators": [50, 100, 200],

"max\_depth": [None, 10, 20],

"min\_samples\_split": [2, 5, 10],

"min\_samples\_leaf": [1, 2, 4]

}

grid\_search = GridSearchCV(

RandomForestRegressor(random\_state=42),

param\_grid,

cv=3,

scoring="neg\_mean\_absolute\_error",

n\_jobs=-1

)

grid\_search.fit(X\_train, y\_train)

# Best Model & Parameters

best\_params = grid\_search.best\_params\_

best\_model = grid\_search.best\_estimator\_

y\_pred\_best = best\_model.predict(X\_test)

# Evaluate Tuned Model

mae\_best = mean\_absolute\_error(y\_test, y\_pred\_best)

rmse\_best = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_best))

print(f"Best Parameters: {best\_params}")

print(f"Tuned Model - MAE: {mae\_best:.2f}, RMSE: {rmse\_best:.2f}")

import joblib

# Save the best model

joblib.dump(best\_model, 'restaurant\_peak\_time\_model.pkl')

**User Manual**

**Introduction**

This system is a smart, IoT-enabled restaurant website aimed at streamlining online food ordering, menu browsing, and restaurant management. It leverages modern web technologies for an interactive experience.

**Getting Started**

1. **Access the Website:**  
   Open your web browser and navigate to the restaurant’s website URL.
2. **Homepage Overview:**  
   The homepage typically displays the restaurant’s highlights, featured dishes, and navigation links.

**User Registration and Login**

* **Registering an Account:**
  1. Click on the “Sign Up” or “Register” button.
  2. Fill in your details: name, email, password, and any other required information.
  3. Submit the form to create your account.
* **Logging In:**
  1. Click on the “Login” button.
  2. Enter your registered email and password.
  3. Click to sign in.

**Browsing the Menu**

* Navigate to the **Menu** or **Order** page from the main navigation bar.
* Browse dishes by category (e.g., Starters, Main Course, Desserts, Drinks).
* Each dish may include a description, image, price, and “Add to Cart” button.

**Placing an Order**

1. **Select Dishes:**  
   Click “Add to Cart” for the items you wish to order.
2. **Review Cart:**  
   Navigate to your cart, usually via a cart icon or “View Cart” button.
3. **Checkout:**
   * Review your selected items, adjust quantities as needed.
   * Enter delivery information if required.
   * Choose payment method (Online/On Delivery).
   * Confirm and submit your order.

**Order Tracking**

* After placing your order, you may be redirected to an **Order Status** or **My Orders** page.
* View real-time updates on your order’s preparation and delivery status (if supported).

**Special Features**

Depending on the implementation, the system may offer:

* **IoT Integration:**  
  Automated notifications, smart table service, or kitchen queue displays.
* **Table Reservation:**  
  Book a table in advance through the reservation section.
* **Promotions:**  
  Check for ongoing offers or discounts.

**Profile Management**

* Access your profile via the user icon or menu.
* Update your personal information, change your password, or view order history.

**Contact & Support**

* Use the **Contact Us** page or live chat (if available) for assistance.
* Find restaurant location, phone number, and email address for support.