

**ADDIS ABABA UNIVERSITY**

**College of Natural and Computational Science**

**School of Information ScienceDepartment of Information Science**

**Oosad Project on Online Food Ordering System For Gambela Hotel**

**Group Members ……………………………….ID Numbers**

**1. Abeniezer Sileshi…………………..…………….UGR/1474/12**

**2. Adisu Wakeyo………………………………...UGR/0904/13**

**3. Bereket Aynetaw………………………..…..…..UGR/8294/13**

**4. Mikias Abate…………………………..…….UGR/4491/13**

**5. Muluken Nega…………………………..… UGR/8380/13**

**Submitted to: Mr. Aminu M.**

**Submission Date: June 09, 2023**

# 

Contents

[CHAPTER 1 4](#_Toc30708)

[PROPOSAL/INTRODUCTION 4](#_Toc29043)

[1.1 Background of the Organization 4](#_Toc22292)

[1.2 Statement of the Problem 5](#_Toc30919)

[1.3 Objectives of the Project 5](#_Toc281)

[1.3.1 General Objective 5](#_Toc12373)

[1.3.2 Specific Objectives 5](#_Toc22710)

[1.4 Feasibility Study 5](#_Toc20805)

[1.5 Significance of the Project 6](#_Toc5806)

[1.6 Beneficiaries of the Project 6](#_Toc27810)

[1.7 Methodology 6](#_Toc7660)

[1.7.1 Data Collection 6](#_Toc16265)

[1.7.2 System Development Methodology 6](#_Toc3442)

[1.8 Development Tools and Technologies 6](#_Toc20970)

[1.8.1 Frontend Technologies 6](#_Toc26265)

[1.8.2 Backend Technologies 7](#_Toc31803)

[1.8.3. Documentation and Modeling Tools 7](#_Toc4726)

[1.8.4. Deployment Environment 7](#_Toc5066)

[1.9. Scope 7](#_Toc10095)

[1.10. Risks, Assumptions, and Constraints 7](#_Toc14993)

[1.11. Phases and Deliverables of Project 7](#_Toc6191)

[1.12. Work Breakdown Structure 8](#_Toc30390)

[1.13. Project Schedule 9](#_Toc32681)

[CHAPTER 2 11](#_Toc5930)

[BUSINESS AREA ANALYSIS AND REQUIREMENT DEFINITION 11](#_Toc6602)

[2.1 Introduction 11](#_Toc22005)

[2.2 Business Area Analysis 11](#_Toc14077)

[2.2.1 Activities/Functions of the Organization 11](#_Toc28217)

[2.2.2 Problems of the Current system 11](#_Toc10431)

[2.2.3 Forms and Reports of the current system 12](#_Toc19181)

[2.2.4 Players of the Existing System 12](#_Toc22476)

[2.3 Requirements Definitions 12](#_Toc23781)

[2.3.1 Functional Requirement 12](#_Toc30889)

[2.3.2 Non-functional Requirement 22](#_Toc22945)

[2.4 Collaboration Modeling 23](#_Toc14749)

[CHAPTER 3 25](#_Toc29168)

[OBJECT ORIENTED ANALYSIS 25](#_Toc23893)

[3.1 Introduction 25](#_Toc18040)

[3.2 System Use Case 25](#_Toc4041)

[3.2.1 UI Identification 25](#_Toc20887)

[3.2.2 Business Rules Identification 26](#_Toc28441)

[3.2.3 Actor Identification 26](#_Toc31040)

[3.2.4 Designing the Use Case Diagram 27](#_Toc20599)

[3.2.5 Use Case Description 27](#_Toc20666)

[3.3. Conceptual Modeling 36](#_Toc18791)

[3.3.1. Class diagram 37](#_Toc4466)

[3.3.2. Class description 38](#_Toc1995)

[3.4. Sequence diagramming 41](#_Toc19261)

[3.5. User Interface Prototyping 43](#_Toc6049)

[CHAPTER 4 43](#_Toc4024)

[OBJECT ORIENTED DESIGN 43](#_Toc27483)

[4.1. Introduction 43](#_Toc14766)

[4.2. Overview of the Phase I 43](#_Toc25647)

[4.2 Class Type Architecture 44](#_Toc14508)

[(The UI Layer, The Controller Layer, Business/Domain Layer, Persistence Layer, System Classes) 44](#_Toc16867)

[4.3 Design Class Modeling 45](#_Toc18984)

[4.3.1. Class Diagram 46](#_Toc24016)

[4.3.2 Description of classes 47](#_Toc5347)

[4.4 Collaboration Diagram 49](#_Toc29497)

[4.5 Component Diagram / Sub-System Decomposition 50](#_Toc119)

[4.6 Deployment Diagram 51](#_Toc1837)

[4.7 Relational Persistent Model 52](#_Toc12842)

[4.7.1 Mapping the Design Class into Relations 53](#_Toc21635)

[4.7.2 Normalizations 55](#_Toc8572)

[4.7.3. Database Diagram 56](#_Toc1204)

[4.8. User Interface 58](#_Toc2575)

[4.8.1. User Interface Flow Diagram 58](#_Toc1191)

[4.8.2 User Interface Design 59](#_Toc28115)

[CHAPTER 5 62](#_Toc31907)

[OBJECT ORIENTED IMPLEMENTATION 62](#_Toc12708)

[5.1 Introduction 62](#_Toc27907)

[5.2 Implementation Technology 62](#_Toc28596)

[5.3 Testing and testing procedure 63](#_Toc15715)

[5.3.1 Unit Testing 63](#_Toc13951)

[5.3.2 Integration Testing 64](#_Toc14943)

[5.3.3 System Testing 65](#_Toc24599)

[5.3 Deployment/Installation process 67](#_Toc7704)

[CHAPTER 6 69](#_Toc19717)

[CONCLUSION AND RECOMMENDATION 69](#_Toc25119)

[6.1 Conclusion 69](#_Toc26396)

[6.2 Recommendation 70](#_Toc19830)

# 

# CHAPTER 1

# PROPOSAL/INTRODUCTION

## 1.1 Background of the Organization

Gambella Hotel Addis Ababa is a newly established highland hotel located in Addis Ababa the capital and diplomatic seat of Ethiopia. It established in 2021. It offers high-end facilities with premium services to our guests. With their different outlets, their guests can have a unique experience and pleasure throughout their stay with us. It is located in Piazza around Ras Mokenen Bridge near 70 Dereja and is one of the best rated hotels to stay in Addis Ababa. The hotel offers comfortable accommodations, including standard rooms, deluxe rooms, and suites, with amenities such as free Wi-Fi, satellite TV, minibars, and room service.

The hotel also has several on-site dining options, including a restaurant that serves Ethiopian and international cuisine, a cafe that offers coffee and light snacks, and a bar that serves cocktails and other drinks. Other hotel facilities include a fitness center, a sauna, a steam room, and a rooftop terrace with panoramic views of the city.

## 1.2Statement of the Problem

The absence of an online food ordering and delivery system has resulted in a significant loss of potential revenue for Gambella Hotel. Customers are looking for convenience, and an online ordering system would enable them to order food and have it delivered to their location without having to physically visit the hotel. Additionally, the hotel's staff is currently overwhelmed with managing orders, and there is a possibility of errors in taking and delivering orders manually.

## 1.3 Objectives of the Project

### 1.3.1 General Objective

The general objective of the project is to build web based an online food ordering system for Gambella Hotel.

### 1.3.2 Specific Objectives

The specific objectives of the project are as follows:

* To develop a user-friendly interface for customers to place orders online.
* To create a backend system for managing orders and deliveries.
* To provide a system for delivery tracking and order status updates.
* To conduct thorough testing to ensure the system's functionality, performance, and security.

## 1.4 Feasibility Study

A feasibility study is an important step in determining the viability of a food ordering system. Here are some key areas that would be evaluated as part of the feasibility study:

**Technical feasibility:**

* The hotel's existing infrastructure will be assessed to determine if it can support the food ordering system.
* The development team will need to ensure that the system is compatible with the hotel's existing hardware and software.
* The system will need to be able to integrate with the hotel's existing systems, such as the hotel management system, and payment gateway.
* The development team will need to ensure that the system is scalable and can handle a large volume of orders.

**Economic feasibility:**

* The development costs, including hardware and software costs, will be evaluated to determine if they are within the hotel's budget.
* The ongoing maintenance costs will also be evaluated to ensure that the hotel can afford to maintain the system.
* The potential return on investment will be calculated to determine if the system is economically viable.

**Operational feasibility:**

* The hotel's current operational processes will be assessed to determine how the food ordering system will integrate with these processes.
* The development team will need to ensure that the system is user-friendly and easy to use for both guests and staff.
* The hotel staff will need to be trained on how to use the new system.
* The potential impact on hotel operations will be evaluated to determine if the system will cause any disruption.

## 1.5 Significance of the Project

The project's significance is to provide a convenient and efficient way for customers to order food online and have it delivered to their location. This will increase customer satisfaction, loyalty, and revenue for Gambella Hotel. The online ordering system will also improve the efficiency of the ordering process and reduce the likelihood of errors in managing orders manually.

## 1.6 Beneficiaries of the Project

The primary beneficiaries of the project are the hotel itself and its customers. The hotel will benefit from increased revenue and improved operational efficiency, as the online platform will streamline the food ordering process and potentially attract more customers. Customers, on the other hand, will benefit from the convenience of being able to order food online, potentially saving time and effort.

## 1.7 Methodology

### 1.7.1 Data Collection

Data will be collected through interviews with hotel staff and customers to gather information about the ordering process, delivery process, and customer preferences. The information collected will be used to design and develop the system.

### 1.7.2 System Development Methodology

The project will use an agile development methodology, which involves iterative development cycles, continuous feedback, and collaboration between the development team and stakeholders. The agile methodology will allow for flexibility in responding to changes in requirements and addressing issues as they arise.

## 1.8 Development Tools and Technologies

### 1.8.1 Frontend Technologies

The frontend of the system will be developed using HTML, CSS, and JavaScript. Bootstrap will be used as a front-end framework to ensure that the system is responsive and compatible with various devices.

### 1.8.2 Backend Technologies

The backend of the system will be developed using PHP, MySQL, and Laravel. Laravel is a PHP framework that provides a modular and scalable approach to web application development.

### 1.8.3. Documentation and Modeling Tools

For the object-oriented system analysis and design of an online food ordering and delivery project for Gamebella Hotel in PHP, we may consider using various documentations and modeling tools, such as UML diagrams, use case diagrams, sequence diagrams, activity diagrams, and class diagrams. We may also use tools like Visual Paradigm, StarUML, or Lucidchart for documentation and modeling purposes.

### 1.8.4. Deployment Environment

The deployment environment for the online food ordering and delivery project for Gamebella Hotel in PHP may consist of a web server, a database server, and a payment gateway. We may consider using popular web servers like Apache, and a relational database management system like MySQL.

## 1.9. Scope

The scope of the our project include designing and developing a user-friendly web application that allows customers to browse through the restaurant's menu, place orders, make payments, and track delivery status. The project may also include features like real-time order tracking, push notifications, customer feedback, and loyalty programs.

## 1.10. Risks, Assumptions, and Constraints

The risks, assumptions, and constraints may include factors like technical complexity, security risks, user acceptance, competition, regulatory compliance, and budget constraints. Some assumptions may include the availability of required resources, the acceptance of the solution by the stakeholders, and the timely delivery of the project.

## 1.11. Phases and Deliverables of Project

The phases and deliverables of our project may include:

**Planning Phase**

* This phase involves defining the project scope, objectives, and stakeholders. The deliverables of this phase include a project charter, requirements specification, and project plan.

**Analysis Phase**

* This phase involves gathering and analyzing user requirements to determine the functionality and features of the online food ordering system. The deliverables of this phase include use case diagrams, user stories, and functional requirements.

**Design Phase**

* This phase involves creating the system architecture, defining the data model, and designing the user interface. The deliverables of this phase include sequence diagrams, class diagrams, activity diagrams, and a prototype.

**Implementation Phase**

* This phase involves writing code, testing individual components, and integrating code modules into a functional system. The deliverables of this phase include code modules, unit tests, and integration tests.

**Testing Phase**

* This phase involves testing the system to ensure that it meets the user requirements and specifications. The deliverables of this phase include system tests and acceptance tests.

**Deployment Phase**

* This phase involves deploying the system to the production environment and providing user manuals, a deployment guide, and a system maintenance plan.

## 1.12. Work Breakdown Structure

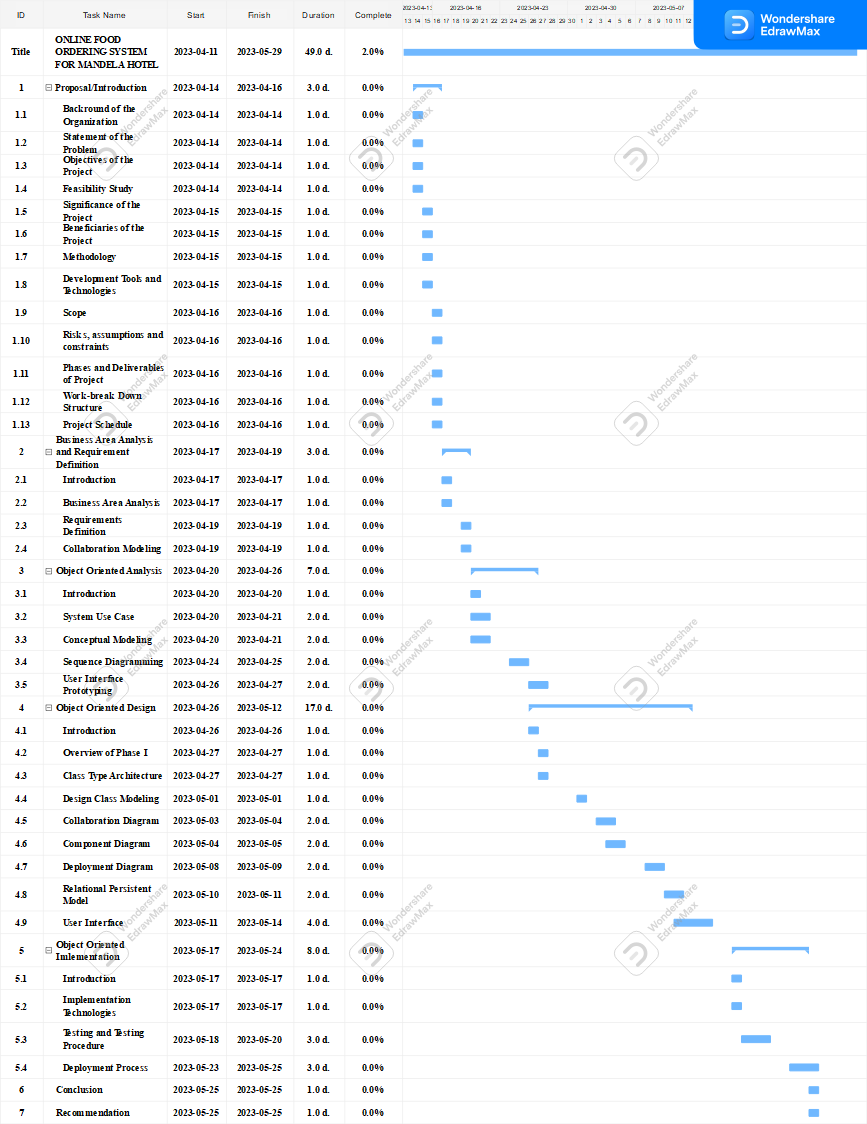
A Work Breakdown Structure (WBS) is a hierarchical decomposition of the project scope into smaller, more manageable components. It's a visual representation of the project's deliverables, tasks, and subtasks, organized in a logical and structured way.

The WBS typically starts with the project's main deliverables (or phases), then breaks them down into smaller, more manageable tasks, and finally into individual work packages. Each work package represents a discrete task that can be assigned to a specific team member, and it includes all the necessary information for completing the task, such as the scope, schedule, budget, and resources required.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Project: Online Food Ordering System for Gambela Hotel | | Prepared by: Adisu | | | Legend:  P = Primary  S = Support | |
| Manager: Adisu | | Page 1 of 1 | | |
| Responsibility Matrix | | |  | |
| Task ID | Task | Abeniezer | Adisu | Bereket | Mikias | Muluken |
| A | Project Plan/Proposal | P | P | S | S | S |
| B | Business Area Analysis | S | P | S | P | S |
| C | Requirement Definition | P | S | P | S | P |
| D | Collaboration Modeling | P | S | S | P | S |
| E | System Use Case | S | P | P | S | P |
| F | Conceptual Modeling | P | P | S | S | S |
| G | Sequence Diagramming | S | P | S | P | S |
| H | User Interface Prototyping | S | P | S | S | S |
| I | Design Class Modeling | P | P | P | S | P |
| J | Collaboration Diagram | S | S | S | P | P |
| K | Component Diagram | S | P | P | S | S |
| L | Deployment Diagram | P | S | S | P | P |
| M | Relational Persistence Model | S | P | S | S | S |
| N | User Interface | P | P | P | P | P |
| O | Testing and Testing Procedure | S | P | S | P | S |
| P | Deployment/Installation Process | P | P | P | P | P |
| Q | Conclusion and Recommendation | P | P | S | S | S |

## 1.13. Project Schedule

We scheduled our project schedule using a Gantt chart. The schedule may include timelines for each phase of the project, along with milestones and deliverables. The schedule may also include resource allocation, budget tracking, and risk management.



# CHAPTER 2

# BUSINESS AREA ANALYSIS AND REQUIREMENT DEFINITION

## 2.1 Introduction

Business area analysis is a process of evaluating the current business environment and identifying the needs and requirements of a specific business area.

Requirement definition, on the other hand, is the process of identifying and documenting the specific features and functionalities that the online food ordering system should have to meet the needs of Gambella Hotel. This includes identifying the key stakeholders, understanding their needs and requirements, and defining the system requirements that will meet those needs.

In generally, Business area analysis and requirement definition process will help ensure that the online food ordering system is tailored to meet the specific needs of the hotel and its customers, and is designed to improve efficiency and enhance the overall dining experience.

## 2.2 Business Area Analysis

### 2.2.1 Activities/Functions of the Organization

Gambella Hotel is located in Addis Ababa. The hotel offers a range of activities and functionalities to its guests, which include:

* **Accommodation:** It offers comfortable and spacious rooms for guests, including standard rooms, executive rooms, and suites. Each room is equipped with modern amenities such as air conditioning, TV, and Wi-Fi.
* **Dining:** It has a restaurant that offers a variety of local and international cuisine. The restaurant is open for breakfast, lunch, and dinner, and also offers room service. Overall, Gambella Hotel offers a range of activities and functionalities to its guests, making it a great choice for both business and leisure travelers.

### 2.2.2 Problems of the Current system

Some potential problems that the current system includes:

* **Manual Order Taking:** The manual system for food ordering requires a staff member to take orders manually, which can result in errors or miscommunications between the kitchen and the customer.
* **Limited Payment Options:** The manual system may only allow for cash payments, which can be inconvenient for customers who prefer to use other payment methods such as credit cards or mobile payments.
* **Lack of Real-Time Order Tracking:** The manual system may not provide real-time updates on the status of the order, which can lead to confusion and frustration among customers.
* **Inefficient Delivery Process**: The manual system may not have an efficient delivery process, which can result in delays or incorrect deliveries.
* **Limited Menu Options:** The manual system may not have the capability to offer a wide range of menu options, which can limit customer choices and satisfaction.

### 2.2.3 Forms and Reports of the current system

Forms and reports of the current systems include the following points:

**Forms**

* **Menu order form:** This form is used by customers to select the items they want to order from the restaurant's menu.
* **Table reservation form:** This form is used by customers to reserve a table in the restaurant.
* **Customer information form:** This form collects basic information about the customer, such as name, contact information, and any dietary restrictions or allergies.
* **Feedback form:** This form allows customers to provide feedback on their experience with the restaurant's food and service.

**Reports**

* **Sales report:** This report provides an overview of the restaurant's sales, including total revenue, number of orders, and average order value.
* **Inventory report:** This report tracks the restaurant's inventory of food and supplies to ensure that there is enough stock to meet demand.
* **Customer feedback report:** This report compiles feedback from customers to identify any issues or areas for improvement in the restaurant's food and service.

### 2.2.4 Players of the Existing System

The players or stakeholders of the existing manual food ordering system of Gambella Hotel include:

* Customers: The people who visit the hotel's restaurant to order food.
* Chef: The staff members who take orders, prepare food, and serve customers.
* Manager: The hotel's management team who oversee the restaurant operations and ensure that they are profitable and efficient.

## 2.3 Requirements Definitions

Requirement definition is the process of identifying and documenting the specific features and functionalities that a system should have to meet the needs of a business. In the case of our system requirement definition would involve identifying the specific features and functionalities that the system should have to improve the hotel's food ordering process and customer experience.

It includes Identify the business requirements, Gather information, Analyze the information, Define system requirements, Develop a project plan, Test and iterate.

### 2.3.1 Functional Requirement

Here are some functional requirements for our system:

* **Registration and Login:** The system should allow guests to register and create an account with their personal details such as name, email, and phone number. Once registered, guests should be able to log in to their account using their credentials.
* **Browse Menu:** The system should allow guests to browse the menu items with details such as name, description, image, and price. Guests should be able to filter and sort the menu items based on different criteria such as cuisine, dietary restrictions, and price range.
* **Place Order:** The system should allow guests to place an order by selecting the desired items from the menu and customizing them as per their preferences. Guests should be able to specify the delivery or pickup location, payment method, and any special instructions or comments.
* **Order Confirmation:** The system should confirm the order by sending a notification to the guest with details such as order number, estimated delivery time, and total cost. The system should also notify the hotel staff of the new order and provide them with the necessary details.
* **Order Tracking:** The system should allow guests to track the status of their order in real-time, such as "order received", "order confirmed", "order dispatched", and "order delivered". Guests should also be able to view the order history and track the delivery status using a map.
* **Payment Processing:** The system should process the payment securely and efficiently using a payment gateway that supports multiple payment methods such as credit/debit cards, mobile payments, and cash on delivery. The system should also generate an invoice or receipt for the guest and the hotel staff..
* **User Management:** The system should allow hotel staff to manage the guest accounts, such as creating new accounts, updating personal details, and resetting passwords. The system should also have the ability to handle guest complaints and resolve issues promptly.
* **Menu Management**: The system should allow hotel staff to manage the menu items, such as adding new items, updating prices, and removing out-of-stock items. The system should also have the ability to display different menus based on the time of day or day of the week.
* **Order Management:** The system should allow hotel staff to manage the incoming orders, such as accepting or rejecting orders, updating the status of orders, and communicating with guests. The system should also have the ability to generate reports and analytics on the order data.

#### 2.3.1.1 Essential Use case modeling

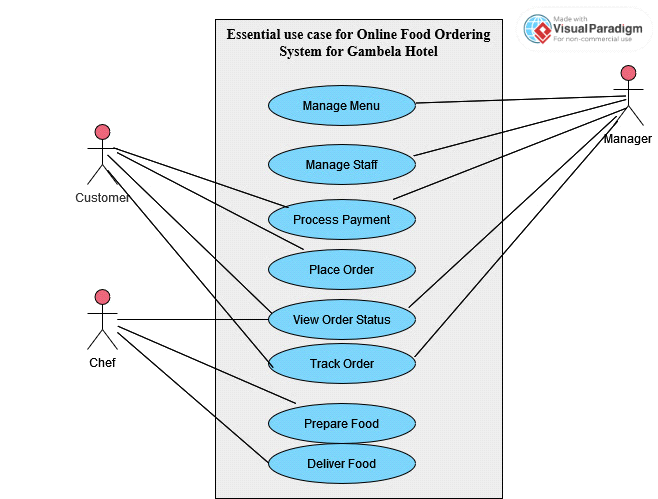
Essential use case modeling is a simplified form of use case modeling that focuses on the essential or core functionality of the system. It is used to identify the most important use cases that the system must support to meet the needs of its users.

The key steps in essential use case modeling include:

* **Identify the actors:** This involves identifying the different types of users or actors who will interact with the system, such as customers, Manager, and Chef.
* **Identify the goals:** This involves identifying the specific goals or objectives that each actor has when interacting with the system. For example, a customer may want to order food online, while a manager member may want to view and manage orders.
* **Identify the use cases:** This involves identifying the specific use cases or scenarios in which each actor interacts with the system to achieve their goals. For example, a customer use case may include "place an online order," while a manager use case may include "view and manage orders."
* **Prioritize the use cases:** This involves prioritizing the identified use cases based on their importance and feasibility. The most important and feasible use cases are then selected for further development and implementation.

Essential use case modeling helps to ensure that the system development process focuses on the most important and essential functionality of the system, which can help to improve the system's usability and overall effectiveness.

##### 2.3.1.1.1. Essential use case diagram



##### 2.3.1.1.2. Essential use case description

***Table 1.1 Description of Place order use case***

|  |  |
| --- | --- |
| Name | Place Order |
| Identifier | EUC 01 |
| Description | This use case describes the process of a customer placing an order for food online |
| Actor | Customer |
| Precondition | The customer has access to the internet and the system. |
| **Basic Course of Actions:**  1. The customer selects the desired food items from the online menu.  2. The customer specifies any special requests or instructions for the order, such as dietary restrictions or additional toppings.  3. The customer confirms the order and submits it for processing.  4. The order is sent to the restaurant's manager for review and verification of item availability.  5. The manager verifies the availability of the items and confirms the order with the customer.  6. The order is sent to the chef for preparation.  7. The chef prepares the order according to the customer's specifications.  8. The chef notifies the customer when the order is ready for pickup or delivery. | |
| **Alternative Course of Actions:**  1. If the items are not available, the restaurant's manager notifies the customer and prompts them to select a different item or cancel the order.  2. If the customer encounters any technical issues during the order placement process, they can contact customer support for assistance. | |
| Post Condition | The order is successfully placed and the customer receives a confirmation of their order. |

***Table 1.2 Description of view order status use case***

|  |  |
| --- | --- |
| Name | View Order Status |
| Identifier | EUC 02 |
| Description | It describes the process of an actor checking the status of an order. |
| Actor | Customer, Manager, chef |
| Precondition | The actor is logged into the system and has an order that is in progress. |
| **Basic Course of Actions:**  1. The actor selects the order they wish to check the status of.  2. The system displays the current status of the order, such as "Order received", "In progress", "Ready for delivery", or "Delivered". | |
| **Alternative Course of Actions:**  1. If the actor encounters any issues during the status check process, they can contact the waiter or the manager for assistance. | |
| Post Condition | The actor can view the current status of the order. |

***Table 1.3 Description of manage staff use case***

|  |  |
| --- | --- |
| Name | Manage Staff |
| Identifier | EUC 03 |
| Description | It describes the process of managing the staff members. |
| Actor | Manager |
| Precondition | The manager is authorized to manage the staff members and has access to the staff management system. |
| **Basic Course of Actions:**  1. The manager logs into the staff management system.  2. The manager views the list of staff members and their details, including their name, position, and schedule.  3. The manager assigns tasks and shifts to staff members based on their availability and skills.  4. The manager updates the staff member's details, including their contact information, position, and salary.  5. The manager creates new staff member profiles and deletes old ones as necessary. | |
| **Alternative Course of Actions:**  1. If a staff member is not available for their assigned shift, the manager can reassign the shift to another staff member.  2. If a staff member requests time off, the manager can approve or deny the request based on staffing needs and availability. | |
| Post Condition | The staff management system is updated with the new staff member details and schedules. The staff members are informed of their assigned tasks and shifts. The manager ensures that the restaurant is adequately staffed and is able to provide quality service to customers. |

***Table 1.4 Description of Process payment use case***

|  |  |
| --- | --- |
| Name | Process payment |
| Identifier | EUC 04 |
| Description | It involves the customer making a payment for their food order and the manager verifying the payment. |
| Actor | Manager, Customer |
| Precondition | The customer has placed an order through online and selected a payment method, such as credit card or cash on delivery. |
| Basic Course of Actions:  1. The customer selects a payment method and submits the payment information through the online.  2. The system verifies the payment information and processes the payment.  3. The system sends a notification to the manager to confirm the payment.  4. The manager verifies the payment and confirms the order with the customer.  5. The order is sent to the chef for preparation.  6. The chef prepares the order according to the customer's specifications.  7. The chef notifies the customer when the order is ready for pickup or delivery. | |
| Alternative Course of Actions:  1. If the payment information is invalid or cannot be processed, the customer is prompted to try again or use a different payment method.  2. If the manager cannot verify the payment, they will contact the customer to resolve the issue.  3. If the customer encounters any technical issues during the payment process, they can contact customer support for assistance. | |
| Post Condition | The payment is successfully processed. |

***Table 1.5 Description of manage menu use case***

|  |  |
| --- | --- |
| Name | Manage Menu |
| Identifier | EUC 05 |
| Description | It involves the manager adding, updating, or removing items from the online menu. |
| Actor | Manager |
| Precondition | The manager has access to the system and logged in to their account. |
| Basic Course of Actions:  1. The manager selects the "Manage Menu" option from the system's main menu.  2. The system displays the current menu items.  3. The manager selects an option to add, update, or remove an item.  4. If adding a new item, the manager enters the item information, such as name, description, price, and image (if applicable).  5. If updating an existing item, the manager selects the item to be updated and enters the new information.  6. If removing an item, the manager selects the item to be removed and confirms the removal.  7. The system verifies the changes and updates the online menu accordingly | |
| Alternative Course of Actions:  1. If there are technical issues with the system, the manager can contact IT support for assistance.  2. If the manager encounters any difficulties in adding, updating, or removing items, they can contact customer support for assistance. | |
| Post Condition | The menu is successfully updated. |

***Table 1.6 Description of track order use case***

|  |  |
| --- | --- |
| Name | Track order |
| Identifier | EUC 06 |
| Description | It involves the customer tracking the status of their food order and the manager providing updates on the order status. |
| Actor | Customer, Manager |
| Precondition | The customer has placed an order and the order has been confirmed and is being prepared. |
| Basic Course of Actions:  1. The customer selects the "Track Order" option from the system's main menu.  2. The system displays the current status of the order, such as "preparing," "out for delivery," or "delivered."  3. If the order is still being prepared, the system provides an estimated time for completion.  4. The manager updates the order status as it progresses through the preparation and delivery process.  5. If there are any delays or issues with the order, the manager notifies the customer and provides an explanation.  6. When the order is ready, the chef notifies the customer for pickup or delivery. | |
| Alternative Course of Actions:  1. If there are technical issues with the system, the customer can contact customer support for assistance.  2. If the order is delayed or cannot be fulfilled, the manager will notify the customer and work with them to resolve the issue. | |
| Post Condition | The customer is able to track the status of their order. |

***Table 1.7 Description of prepare food use case***

|  |  |
| --- | --- |
| Name | Prepare Food |
| Identifier | EUC 07 |
| Description | It describes the process of preparing food for the customer's order. |
| Actor | chef |
| Precondition | The order has been placed by the customer and received by the kitchen staff. |
| Basic Course of Actions:  1. The chef selects the "View Order Status" option from the system's main menu.  2. The system displays the current status of the orders, such as "preparing," "out for delivery," or "delivered."  3. The chef selects an order that is ready for preparation.  4. The system displays the details of the order, including the items, quantities, and any special requests or instructions.  5. The chef prepares the food according to the customer's specifications.  6. The chef ensures that the food is prepared accurately and in a timely manner.  7. The chef notifies the customer when the order is ready for pickup or delivery. | |
| Alternative Course of Actions:  - If the items are not available, the chef notifies the system and prompts the customer to select a different item or cancel the order.  - If the chef encounters any difficulties in preparing the food, they can contact the manager or customer support for assistance. | |
| Post Condition | The food is successfully prepared. |

***Table 1.8 Description of deliver food use case***

|  |  |
| --- | --- |
| Name | Deliver Food |
| Identifier | EUC 08 |
| Description | It involves the chef delivering the food to the customer's specified location. |
| Actor | chef |
| Precondition | The food has been prepared and is ready for delivery. |
| Basic Course of Actions:  1. The chef selects the "Deliver Food" option from the system's main menu.  2. The system displays the details of the order, including the customer's name, address, and phone number.  3. The chef checks the order to ensure that all items are included and accurate.  4. The chef packages the food for delivery and confirms the delivery address with the customer.  5. The chef delivers the food to the customer's specified location.  6. The chef confirms the delivery with the customer and ensures that they are satisfied with the order. | |
| Alternative Course of Actions:  1. If the customer is not available to receive the food, the chef will contact the customer to reschedule the delivery.  2. If there are any issues with the order, the chef will work with the customer to resolve the issue. | |
| Post Condition | The food is successfully delivered to the customer. |

#### 2.3.1.2 Essential user interface prototyping

Email (Input)

Password(Input)

**Login Form**

Total Price (Input)

Address (Input)

Customer ID (Input)

Name (Input)

Phone (Input)

Price (Input)

Quantity (Input)

FoodName (Input)

**Customer Place Order Form**

### 2.3.2 Non-functional Requirement

Non-functional requirement is a type of requirement that specifies the characteristics or qualities that a system or product must have, rather than describing a specific behavior or function.

Here are some non-functional requirements for our system:

* **Performance:** The system should be able to handle a large number of concurrent users and orders without any performance degradation. The system should also have fast response times for all user interactions, such as page loads, form submissions, and order confirmations.
* **Reliability:** The system should be highly reliable and available at all times, with minimal downtime or service disruptions. The system should also have appropriate backup and disaster recovery mechanisms in place to ensure business continuity in case of any unforeseen events.
* **Scalability:** The system should be designed to handle a growing number of users and orders over time, without requiring significant changes to the underlying architecture or infrastructure. The system should also have the ability to scale up or down based on the demand and usage patterns.
* **Security:** The system should also have appropriate measures in place to protect against cyber-attacks, data breaches, and unauthorized access. The system should use encryption to protect sensitive data such as passwords and payment information.
* **Usability:** The system should have a user-friendly and intuitive interface that is easy to use and navigate for guests and hotel staff. The system should also have appropriate help and support mechanisms in place to assist users in case of any issues or queries.
* **Accessibility:** The system should be accessible to users with different abilities and disabilities, such as visual impairments, hearing impairments, or mobility impairments.
* **Compatibility**: The system should be compatible with different web browsers, operating systems, and devices, such as desktops, laptops, tablets, and smartphones. The system should also support different languages and locales to cater to a diverse user base.

## **2.4 Collaboration Modeling**

Collaboration modeling is a technique used in software engineering to describe the interactions between different objects or components in a system.

CRC stands for Class-Responsibility-Collaboration. It is a technique used in object-oriented software design to identify and describe classes, their responsibilities, and their collaborations with other classes in a system.

In the CRC technique, each class is represented by an index card or a sticky note, on which the class name is written. The card is then divided into three sections: Class, Responsibilities, and Collaborations.

The Class section describes the basic information about the class, such as its name, its purpose, and its attributes. The Responsibilities section lists the tasks or functions that the class is responsible for performing. The Collaborations section identifies other classes or objects that the class interacts with to accomplish its responsibilities.

* List of classes are
* Manager,
* Chef,
* Customer,
* Order,
* Payment,
* Food and
* Delivery.

CRC for Manager CRC for Delivery

|  |  |
| --- | --- |
| **Manager** | |
| ManagerID  Name  Email  Password  PhoneNumber  addMenuItem()  updateMenuItem()  deleteMenuItem()  viewOrderHistory()  viewSalesReports() | Delivery  Order  Payment |

|  |  |
| --- | --- |
| **Delivery** | |
| DeliveryID  Name  Email  Password  PhoneNumber  VehicleType  updateDeliveryStatus()  viewDeliverSchedule()  viewDeliveryDetails() | Manager  Order |

|  |  |
| --- | --- |
| **Food** | |
| FoodID  Name  Description  Price  Ingredients  updateFoodDetails()  viewFoodDetails() | Order  Chef |

CRC for Food CRC for Payment

|  |  |
| --- | --- |
| **Payment** | |
| PaymentID  PaymentDate  PaymentAmount  PaymentMethod  makePayment()  updatePaymentStatus()  viewPaymentHistory() | Order  Manager  Customer |

CRC for Order CRC for Chef

|  |  |
| --- | --- |
| **Order** | |
| OrderID  OrderDate  DeliveryDate  addMenuItem()  updateMenuItem()  removeMenuItem()  calculateTotal()  confirmOrder()  cancelOrder() | Delivery  Payment  Food  Manager  Chef  Customer |

|  |  |
| --- | --- |
| **Chef** | |
| ChefID  Name  Email  Password  PhoneNumber  viewOrder()  updateOrderStatus()  viewOrderDetails() | Order  Food |

CRC for Customer

|  |  |
| --- | --- |
| **Customer** | |
| Customer ID  Name  Email  Password  PhoneNumber  DeliveryAddress  makePayment()  placeOrder()  viewMenu()  cancelOrder()  viewOrderStatus() | Order  Payment |

# CHAPTER 3

# OBJECT ORIENTED ANALYSIS

## 3.1 Introduction

Object-oriented analysis (OOA) is a software engineering technique used to analyze, design, and model a system based on object-oriented concepts. OOA involves identifying the objects in a system, their attributes, and their relationships with other objects. It is the first step in developing an object-oriented system and is followed by object-oriented design (OOD) and object-oriented programming (OOP).

## 3.2 System Use Case

In object-oriented analysis, a system use case is a description of the interactions between users (actors) and the system. It is used to identify the requirements of the system and to define its functionality.

A system use case in object-oriented analysis for an online food ordering system is a detailed description of how a customer can place an order for food through the system, including the steps involved, actors involved, preconditions, basic course of action, alternative courses of action, and post conditions. It helps to ensure that the system meets the needs of the customer and the restaurant, and that the order is accurately and efficiently processed from start to finish. The following are the steps involved in developing a system use case:

### 3.2.1 UI Identification

The first step in developing a system use case is to identify the user interface (UI) of the system. The UI is the part of the system that the user interacts with. It includes the screens, menus, and buttons that the user uses to input data and receive information. Here is the UI lists:

**Name: Registration Page**

**Identifier: UI01**

**Description:** The Registration Page is where new customers can create an account to access the online food ordering system.

**Name: Login Page**

**Identifier: UI02**

**Description:** This interface allows all actors involved in the food ordering system to securely access their accounts using a username or email and password.

**Name: Home Page**

**Identifier: UI03**

**Description:** It is the main interface for all actors involved in the online food ordering system, including customers, managers, and Chef.

**Name: Menu Management Page**

**Identifier: UI04**

**Description:** This interface allows the manager to manage the menu items, including adding new items, modifying existing items, and removing items. Customers can also browse the menu to view available items and their prices.

**Name: Staff Management Page**

**Identifier: UI05**

**Description:** This interface allows the manager to manage the staff, including adding new staff members, modifying existing staff information, and removing staff members.

**Name: My Orders Page**

**Identifier: UI06**

**Description:** The My Orders Page is a customer interface that allows customers to view their current and past orders, track the status of their orders, and submit complaints or feedback about their orders if necessary. This interface includes access to Cancel Order, Modify Order, and View Order Status features for all actors except delivery staff, and includes access to the Track Order Status feature for the manager.

### 3.2.2 Business Rules Identification

The next step is to identify the business rules of the system. Business rules are the rules that govern the behavior of the system. They define how the system should respond to different inputs and events. In the case of our system, some examples of business rules could include:

**ID: BR01**

**Name:** Menu Item Availability

**Description**: The system should only display menu items that are currently available for order. If a menu item is out of stock or unavailable for any reason, it should be marked as such in the system and should not be available for ordering.

**ID: BR02**

**Name:** Order Minimum and Maximum Limits

**Description:** The system should enforce minimum and maximum limits for order amounts.

**ID: BR03**

**Name:** Payment Options

**Description:** The system should allow users to pay for their orders using various payment options, such as credit/debit cards, or cash on delivery.

**ID: BR04**

**Name:** Delivery Time and Fees

**Description:** The system should allow users to select a delivery time that is convenient for them.

**ID: BR05**

**Name:** User Account Management

**Description:** The system should allow users to create and manage their accounts, including updating their personal information, changing their passwords, and viewing their order history.

**ID: BR06**

**Name:** Order Status Tracking

**Description:** The system should allow users to track the status of their orders in real-time. The system should also send notifications to users when their orders are confirmed, being prepared, out for delivery, or delivered.

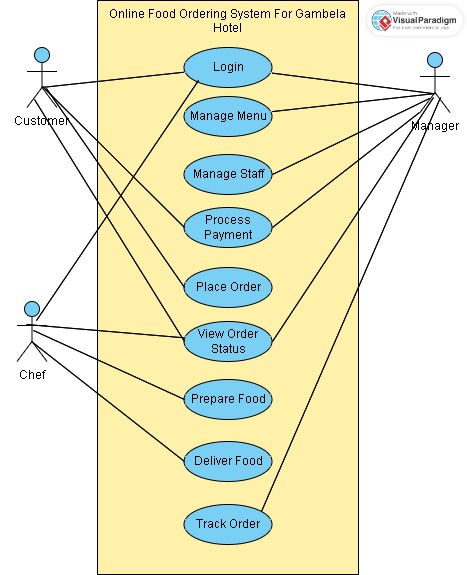
### 3.2.3 Actor Identification

The third step is to identify the actors in the system. Actors are the people or systems that interact with the system. We identified the following actors for our system.

* Customer: The person who places the order and pays for it.
* Manager: The person who manages the orders and oversees the restaurant operations.
* Chef: The person who cooks and prepares the food and delivers the food to the customer.

### 3.2.4 Designing the Use Case Diagram

The fourth step is to design the use case diagram. The use case diagram is a graphical representation of the system use cases and actors. It shows the interactions between the actors and the system.



### 3.2.5 Use Case Description

The final step is to develop a detailed description of each use case. The use case description includes the steps involved in each use case, the inputs and outputs, and any preconditions or post-conditions.

***Table 3.1 Description of Login use case***

|  |  |
| --- | --- |
| Name | Login |
| Identifier | SUC 01 |
| Description | This use case describes then process of the Manager, Chef, and Customer actors logging in to the system to access their respective accounts and perform their tasks. |
| Actor | Manager, Chef, Customer |
| Precondition | The system is available.  The actors have been registered and have valid credentials. |
| Extends | None |
| Includes | None |
| Basic Course of Actions:  1. The actor enters the URL for the Gambella Hotel's online food ordering system in their web browser.  2. The system displays the login page, prompting the actor to enter their username and password.  3. The actor enters their username and password and clicks the "Login" button.  4. The system verifies the credentials and logs the actor into their account.  5. The system displays the appropriate main menu for the actor, depending on their role (Manager, Chef, or Customer).  6. The actor can now access their account and perform their tasks. | |
| Alternative Course of Actions:   * If the actor enters an incorrect username or password, the system prompts the actor to re-enter their credentials. * If the actor encounters any technical issues, they can contact IT support for assistance. * If the actor has forgotten their credentials, they can use the "Forgot Password" feature to reset their password. | |
| Post Condition | The actor is successfully logged in to their account. |

***Table 3.2 Description of place order use case***

|  |  |
| --- | --- |
| Name | Place Order |
| Identifier | SUC 02 |
| Description | This use case involves the Customer placing an order for food, |
| Actor | Customer |
| Precondition | The Customer has access to a food ordering system and has selected the desired items from the menu. |
| Extends | None |
| Includes | None |
| Basic Course of Actions:  1. The Customer accesses the food ordering system online  2. The Customer browses the menu and selects the desired items, specifying any special requests or modifications as necessary.  3. The Customer provides any necessary information such as delivery address or pickup location.  4. The Customer confirms the order and submits it for processing.  5. The system processes the order and sends a confirmation to the Customer  6. If the Customer needs to modify or cancel the order, they can do so before it is prepared by following the appropriate steps.      a. To modify the order, the Customer can access the order details and make any necessary changes, such as adding or removing items or updating the delivery or pickup information.      b. To cancel the order, the Customer can access the order details and select the cancel option, which will remove the order from the system and notify the restaurant staff.  7. If the order is modified or cancelled, the system will update the order status and notify the Customer of the changes.  8. If the order is not modified or cancelled, the restaurant staff will prepare the order for pickup or delivery, depending on the Customer's preferences.  9. The Customer receives the food and checks that it matches the order details.  10. If there are any issues with the order, the Customer can contact the support team for assistance. | |
| Alternative Course of Actions:  1a. If the Customer encounters an issue or has a question during the ordering process, they can contact the support team for assistance. | |
| Post Condition | The order is successfully placed, modified or cancelled, and the Customer receives a confirmation of the order status. |

***Table 3.3 Description of view order status use case***

|  |  |
| --- | --- |
| Name | View Order Status |
| Identifier | SUC 03 |
| Description | This use case allows the Customer, Chef and Manager to view the status of an existing order that has been placed. The customer can see if the order has been processed, shipped, or delivered, while the manager, chef can see the status of orders they're responsible for. |
| Actor | Customer, Chef and Manager |
| Precondition | The order must have been placed and processed in the online ordering system. |
| Extends | None |
| Includes | None |
| Basic Course of Actions:  1. The actor logs in to their account on the online ordering system.  2. The actor navigates to the 'My Orders' page.  3. The actor selects the order they wish to track.  4. The system displays the order details, including the status of the order.  5. The actor can see if the order has been processed, shipped, or delivered.  6. If the order has been shipped, the customer can view the tracking information.  7. The system allows the actor to contact customer service if they have any questions or concerns about the order status. | |
| Alternative Course of Actions:  1. If the actor encounters any issues while viewing the order status, they can click on the 'Contact Us' button to reach out to customer service.  2. The system will provide the actor with the necessary contact information to get in touch with customer service. | |
| Post Condition | The actor is able to see the status of the order. |

***Table 3.4 Description of track order status use case***

|  |  |
| --- | --- |
| Name | Track Order Status |
| Identifier | SUC 04 |
| Description | This use case allows the customer and manager to track the real-time status of an order that has been shipped. The customer can view the current location of their order and estimated delivery time, while the manager can view the status of orders they're responsible for. |
| Actor | Customer, Manager |
| Precondition | The order must have been shipped and the actor must have access to the tracking information. |
| Extends | None |
| Includes | None |
| Basic Course of Actions:  1. The actor logs in to their account on the online ordering system.  2. The actor navigates to the 'My Orders' page.  3. The actor selects the order they wish to track.  4. The system displays the order details, including the tracking information.  5. The actor clicks on the 'Track Order' button.  6. The system displays the real-time status of the order, including the current location and estimated delivery time.  7. The actor can view the status of their order at any time by refreshing the tracking page. | |
| Alternative Course of Actions:  1. If the actor encounters any issues while tracking their order, they can click on the 'Contact Us' button to reach out to customer service.  2. The system will provide the actor with the necessary contact information to get in touch with customer service. | |
| Post Condition | The actor is able to view the real-time status of the order. |

***Table 3.5 Description of process payment use case***

|  |  |
| --- | --- |
| Name | Process Payment |
| Identifier | SUC 05 |
| Description | It involves the customer making a payment for their online food order, and the manager verifying the payment. |
| Actor | Customer and Manager |
| Precondition | The customer has placed an order through an online and the order has been confirmed and is ready for payment. |
| Extends | None |
| Includes | None |
| Basic Course of Actions:  1. The customer selects the "Process Payment" option from the system's main menu.  2. The system displays the payment options, such as credit card, debit card, or mobile payment.  3. The customer selects their preferred payment method.  4. The system processes the payment and confirms the payment has been made.  5. The manager verifies the payment and confirms the order for processing.  6. The chef prepares the food order for pickup or delivery.  7. The customer receives the food order and confirms that they are satisfied with the order. | |
| Alternative Course of Actions:   * If the customer encounters any issues with their payment (e.g. declined credit card), the system displays an error message and prompts the customer to try a different payment method or contact customer support. * If there are any issues with the order, the manager will work with the customer to resolve the issue. | |
| Post Condition | The customer's payment has been processed and the order has been confirmed. |

***Table 3.6 Description of Manage Menu use case***

|  |  |
| --- | --- |
| Name | Manage Menu |
| Identifier | SUC 06 |
| Description | This use case describes the process of manager updating and maintaining the menu items, prices, and availability in the system. |
| Actor | Manager |
| Precondition | The manager has access to the system and is logged in to their account. |
| Extends | None |
| Includes | None |
| Basic Course of Actions:  1. The manager selects the "Manage Menu" option from the system's main menu.  2. The system displays the menu items, prices, and availability.  3. The manager selects an item to update or add a new item to the menu.  4. The system prompts the manager to enter the item details, such as name, description, ingredients, and price.  5. The manager enters the item details and saves the changes.  6. The system updates the menu with the new item or the updated information.  7. The manager can also remove items from the menu if they are no longer available or desired.  8. The system updates the menu with the removed items. | |
| Alternative Course of Actions:  1. If the manager encounters any issues with saving their changes to the menu item, the system displays an error message and prompts the manager to try again  2. If there are technical issues with the system, the manager can contact IT support for assistance. | |
| Post Condition | The menu is updated and maintained. |

***Table 3.7 Description of Manage Staff use case***

|  |  |
| --- | --- |
| Name | Manage Staff |
| Identifier | SUC 07 |
| Description | This use case describes the process of managing staff |
| Actor | Manager |
| Precondition | The manager is logged into their account on the system and has appropriate permissions to manage staff. |
| Extends | None |
| Includes | None |
| Basic Course of Actions:  1. The manager selects the "Manage Staff" option on the online food ordering system.  2. The system displays a list of staff currently registered on the system.  3. The manager selects a specific staff member to edit or creates a new staff member to add to the system.  4. The manager edits the staff member details (e.g. name, role, contact information, etc.) as needed.  5. The manager saves the changes to the staff member.  6. The system updates the staff member details on the online food ordering system. | |
| Alternative Course of Actions:  1. If the manager encounters any issues with saving their changes to the staff member, the system displays an error message and prompts the manager to try again or contact customer support. | |
| Post Condition | The staff details on the online food ordering system have been updated according to the manager's changes. |

***Table 3.8 Description of Prepare food use case***

|  |  |
| --- | --- |
| Name | Prepare food |
| Identifier | SUC 08 |
| Description | This use case describes the process of chef viewing the status of online food orders, selecting an order that is ready for preparation, and preparing the food according to the customer's specifications. |
| Actor | Chef |
| Precondition | An order has been placed through the system and is ready for preparation. |
| Extends | None |
| Includes | None |
| Basic Course of Actions:  1. The chef selects the "View Order Status" option from the system's main menu.  2. The system displays the current status of the orders, such as "preparing," "out for delivery," or "delivered."  3. The chef selects an order that is ready for preparation.  4. The system displays the details of the order, including the items, quantities, and any special requests or instructions.  5. The chef prepares the food according to the customer's specifications.  6. The chef ensures that the food is prepared accurately and in a timely manner.  7. The chef notifies the customer when the order is ready for pickup or delivery. | |
| Alternative Course of Actions:  1. If the items are not available, the chef notifies the system and prompts the customer to select a different item or cancel the order.  2. If the chef encounters any difficulties in preparing the food, they can contact the manager or customer support for assistance. | |
| Post Condition | The food has been prepared according to the order specifications and is ready for delivery or pick-up. |

***Table 3.9 Description of Deliver food issues use case***

|  |  |
| --- | --- |
| Name | Deliver food |
| Identifier | SUC 09 |
| Description | This use case describes the process of delivering the food to the customer's specified location. |
| Actor | Chef |
| Precondition | The food order has been prepared and is ready for delivery. |
| Extends | None |
| Includes | None |
| Basic Course of Actions:  1. The chef selects the "Deliver Food" option from the system's main menu.  2. The system displays the details of the order, including the customer's name, address, and phone number.  3. The chef checks the order to ensure that all items are included and accurate.  4. The chef packages the food for delivery and confirms the delivery address with the customer.  5. The chef delivers the food to the customer's specified location.  6. The chef confirms the delivery with the customer and ensures that they are satisfied with the order. | |
| Alternative Course of Actions:  1. If the delivery staff encounters any issues or problems during the delivery process, such as difficulty finding the delivery address or a delay in traffic, they immediately notify the customer or the customer support team for assistance.  2. If the customer is not available to receive the delivery, the staff member contacts the customer or the customer support team to reschedule the delivery or make alternative arrangements. | |
| Post Condition | The food has been delivered to the customer and the order status has been updated in the system. |

## **3.3. Conceptual Modeling**

Conceptual modeling is the process of creating a high-level abstract representation of a system or a domain. It involves identifying the key concepts, entities, relationships, and constraints that are relevant to the system or domain being modeled, and representing them in a way that is easily understandable by stakeholders, such as end-users, developers, and managers.

The goal of conceptual modeling is to provide a common understanding of the system or domain being modeled, and to facilitate communication and collaboration among stakeholders. It is an important part of the software development process, as it helps to ensure that the requirements of the system are properly understood and documented before design and implementation begin. Conceptual modeling can be done using various techniques, such as entity-relationship modeling, object-oriented modeling, and UML modeling. These techniques provide different ways of representing the concepts, entities, and relationships of the system or domain being modeled, and are often used in combination to create a comprehensive and accurate representation.

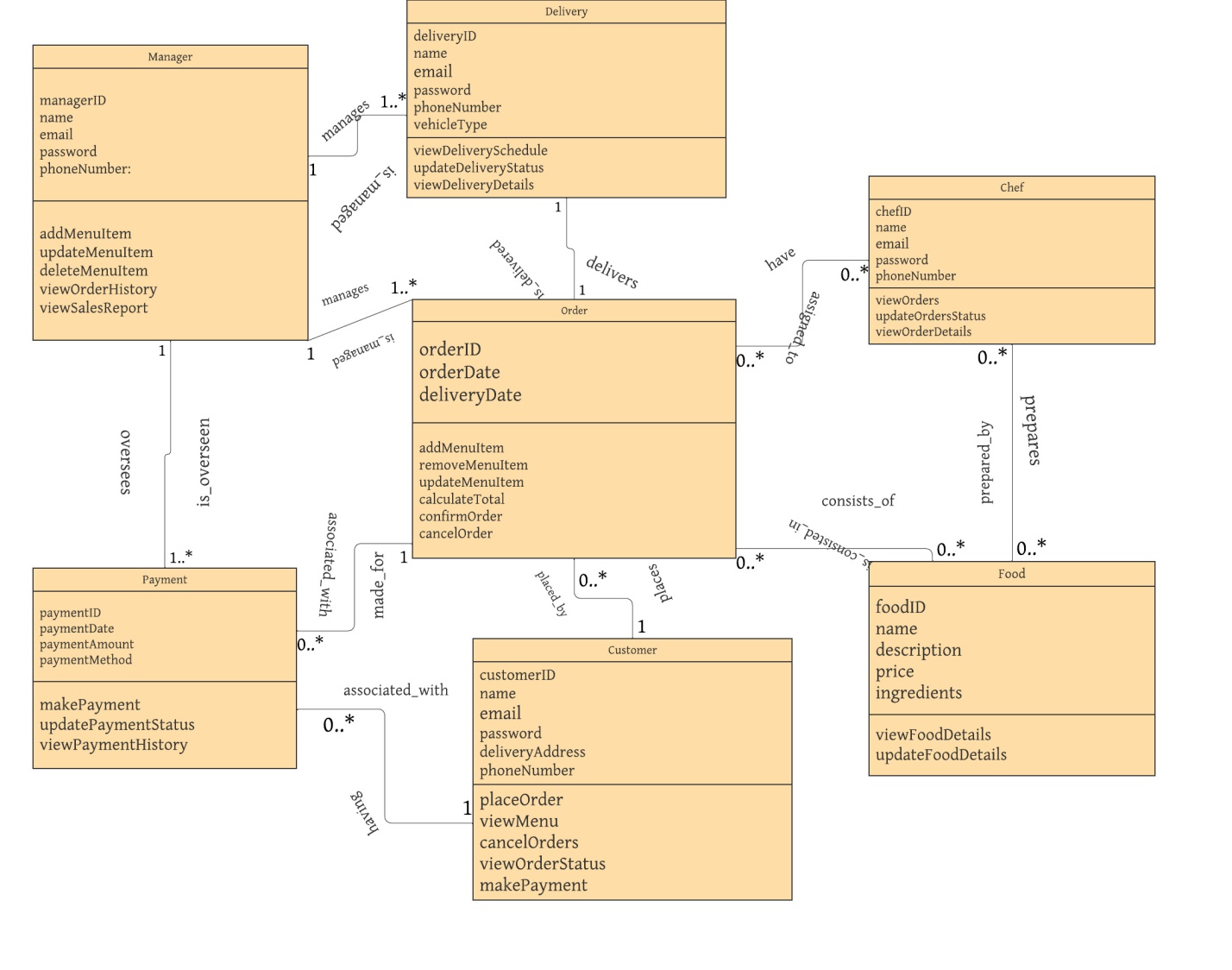
### 3.3.1. Class diagram

A class diagram is a type of UML (Unified Modeling Language) diagram that represents the static structure of a system by showing the classes, their attributes, methods, and the relationships between them. It is one of the most widely used diagrams in object-oriented modeling and provides a visual representation of the classes and objects that make up the system.

In a class diagram, each class is represented as a rectangle with three compartments:

* The top compartment contains the name of the class,
* The middle compartment contains the attributes (data members) of the class, and
* The bottom compartment contains the methods (operations) of the class.

The relationships between classes are represented by arrows that connect the classes.



### 3.3.2. Class description

**Class Name: Manager**

**Description:** It represents a manager who is responsible for managing the restaurant's menu, viewing order histories and sales reports, and managing customer accounts.

**Attributes:**

* managerID: It represents the unique identifier for the manager
* name: It represents the name of the manager
* email: It represents the email address of the manager
* phoneNumber: It represents the phone number of the manager
* password: It represents the password of the manager's account

**Methods:**

* addMenuItem: a method that adds a new menu item to the restaurant's menu
* deleteMenuItem: a method that removes a menu item from the restaurant's menu
* updateMenuItem: a method that updates an existing menu item in the restaurant's menu
* viewOrderHistory: a method that returns a list of all orders made by customers
* viewSalesReport: a method that returns a dictionary of sales reports for the restaurant

**Class Name: Customer**

**Description**: It represents a customer who can view the restaurant's menu, place orders, and make payments.

**Attributes:**

* customerID: It represents the unique identifier for the customer
* name: It represents the name of the customer
* email: It represents the email address of the customer
* phoneNumber: It represents the phone number of the customer
* password: It represents the password of the customer's account
* deliveryAddress: It represents the delivery address for the customer's orders

**Methods:**

* viewMenu: a method that returns a list of all menu items for the restaurant
* placeOrder: a method that places a new order for the customer
* cancelOrder: a method that cancels an existing order for the customer
* viewOrderStatus: a method that returns the status of an existing order for the customer
* makePayment: a method that makes a payment for an existing order

**Class Name: Chef**

**Description**: It represents a chef who is responsible for viewing and updating the status of orders.

**Attributes:**

* chefID: It represents the unique identifier for the chef
* name: It represents the name of the chef
* email: It represents the email address of the chef
* phoneNumber: It represents the phone number of the chef
* password: It represents the password of the chef's account

**Methods:**

* viewOrder: a method that returns a list of all orders that need to be prepared by the chef
* viewOrderDetails: a method that returns the details of a specific order
* updateOrderStatus: a method that updates the status of a specific order

**Class Name: Delivery**

**Description**: It represents a delivery person who is responsible for delivering orders to customers.

**Attributes:**

* deliveryId: It represents the unique identifier for the delivery person
* name: It represents the name of the delivery person
* email: It represents the email address of the delivery person
* phoneNumber: It represents the phone number of the delivery person
* password: It represents the password of the delivery person's account
* vehicleType: It represents the type of vehicle the delivery person will be using for deliveries

**Methods:**

* viewDeliverySchedule: a method that returns a list of all orders that need to be delivered by the delivery person
* viewDeliveryDetails: a method that returns the details of a specific order that needs to be delivered
* updateDeliveryStatus: a method that updates the delivery status of a specific order

**Class Name: Order**

**Description**: It represents an order made by a customer, containing information about the customer, menu items, payment, and delivery.

**Attributes:**

* orderID: It represents the unique identifier for the order
* customerID: It represents the unique identifier for the customer who placed the order
* chefID: It represents the unique identifier for the chef who will prepare the order
* deliveryID: It represents the unique identifier for the delivery person who will deliver the order
* orderDate: It represents the date and time the order was placed
* deliveryDate: It represents the date and time the order will be delivered
* orderStatus: It represents the status of the order
* paymentStatus: It represents the status of the payment for the order

**Methods:**

* addMenuItem: a method that adds a new menu item to the order
* removeMenuItem: a method that removes a menu item from the order
* updateMenuItem: a method that updates an existing menu item in the order
* calculateTotal: a method that calculates the total cost of the order
* confirmOrder: a method that confirms the order and updates the order status
* cancelOrder: a method that cancels the order and updates the order status

**Class Name: Payment**

**Description**: It represents a payment made by a customer for an order.

**Attributes:**

* paymentID: It represents the unique identifier for the payment
* customerID: It represents the unique identifier for the customer who made the payment
* orderID: It represents the unique identifier of the order for which the payment was made
* paymentDate: It represents the date and time the payment was made
* paymentAmount: It represents the amount of the payment
* paymentMethod: It represents the payment method used by the customer (e.g. credit card, PayPal, etc.)

**Methods:**

* makePayment: a method that makes a payment for an existing order
* updatePaymentStatus: a method that updates the payment status of an existing order
* viewPaymentHistory: a method that returns a list of all payments made by a specific customer

**Class Name: Food**

**Description**: It represents a food item on the restaurant's menu.

**Attributes:**

* foodID: It represents the unique identifier for the food item
* name: It represents the name of the food item
* description: It represents the description of the food item
* price: It represents the price of the food item
* ingredients: It represents the ingredients used in the food item

**Methods:**

* viewFoodDetails: a method that returns a string containing the details of the food item
* updateFoodDetails: a method that updates the details of the food item

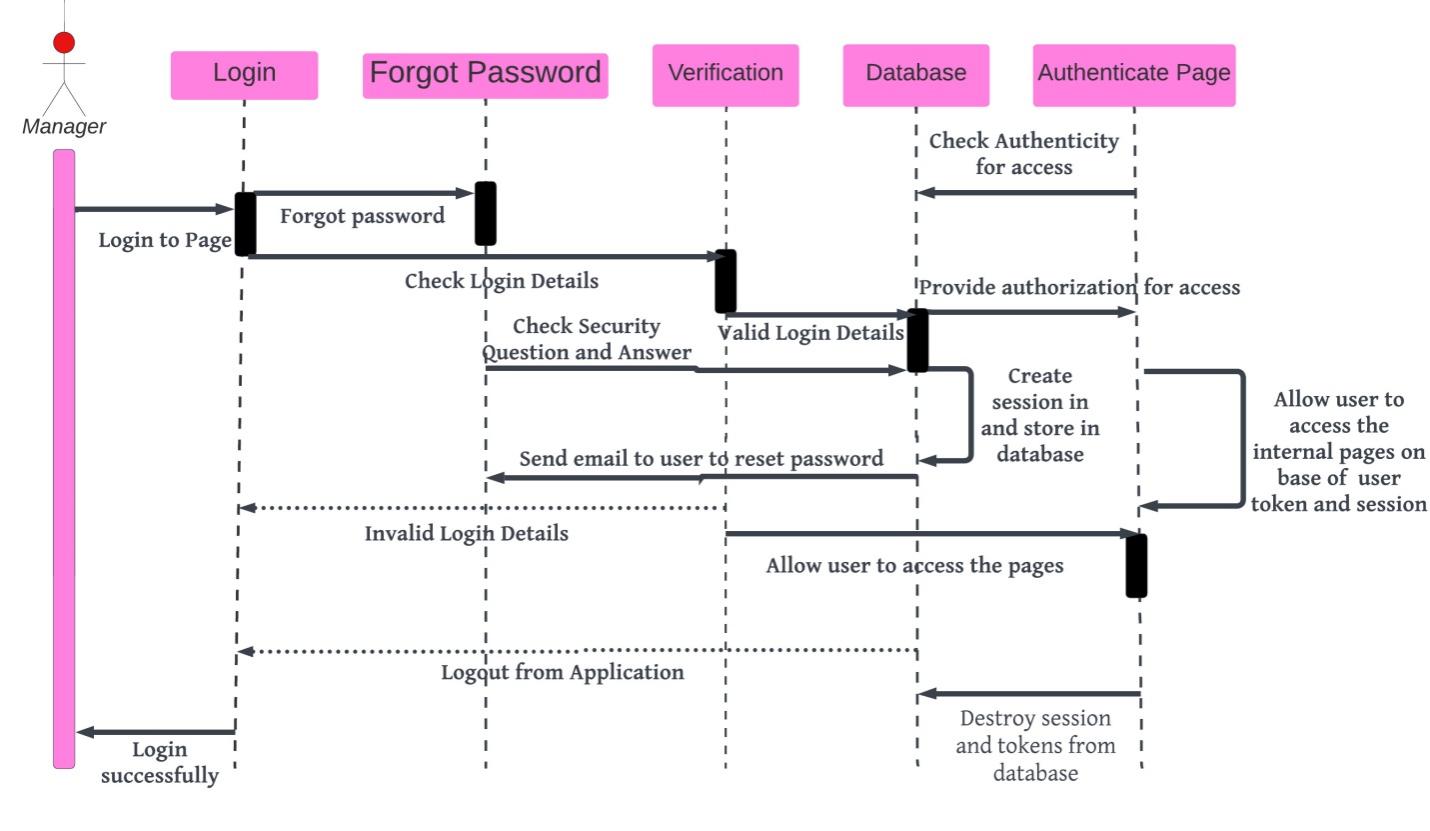
## **3.4. Sequence diagramming**

Sequence diagramming is a type of UML (Unified Modeling Language) diagram that shows the interactions between objects or components in a system over time. It is used to visualize the flow of messages or events between objects or components in a system.

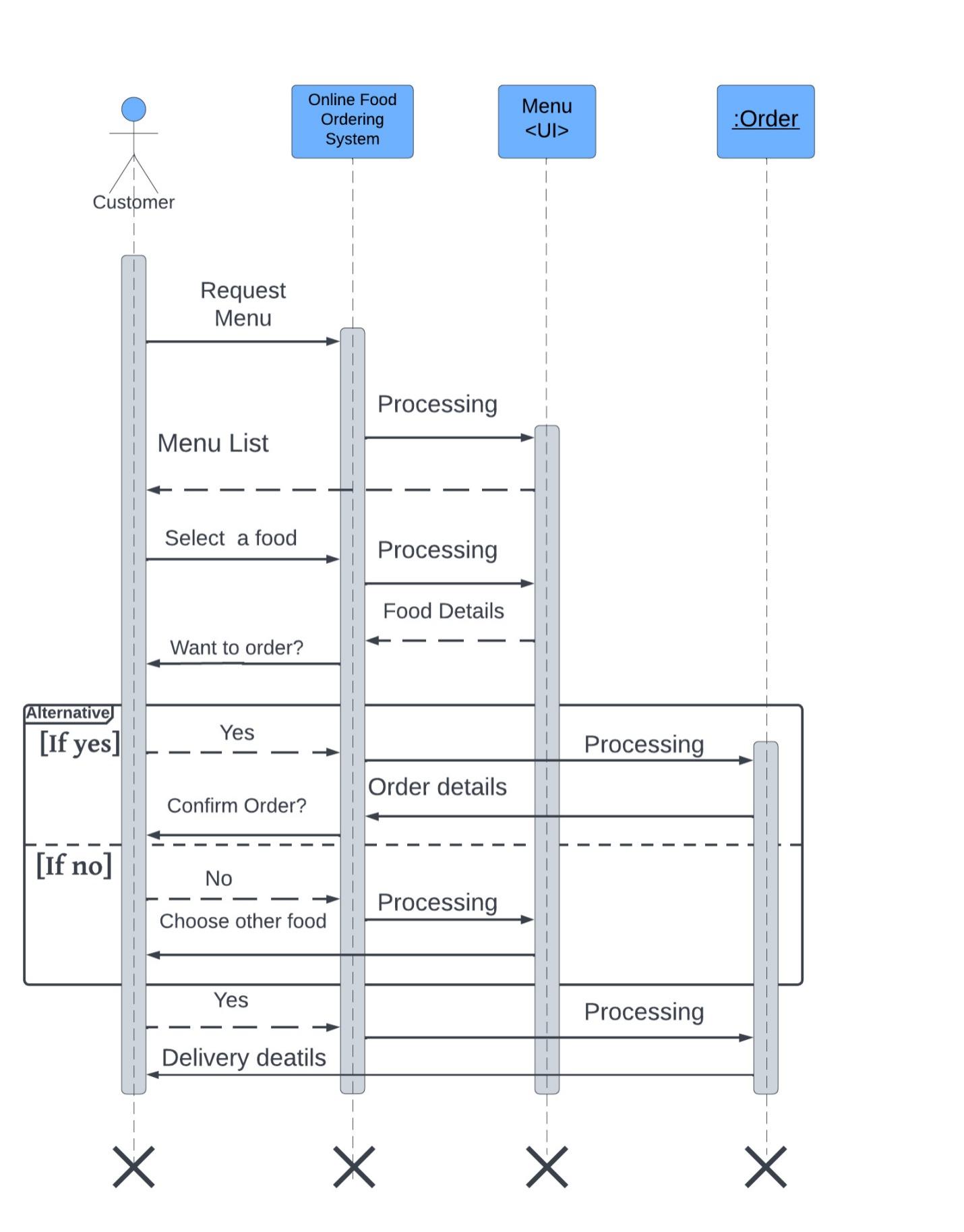
In a sequence diagram, time flows from top to bottom, and the interactions between objects are represented by arrows that connect the objects and the messages or events they exchange. The objects are represented as vertical lines called lifelines, and the messages or events are represented as horizontal arrows that traverse the lifelines.

We have drawn for to use cases, namely login and place order.

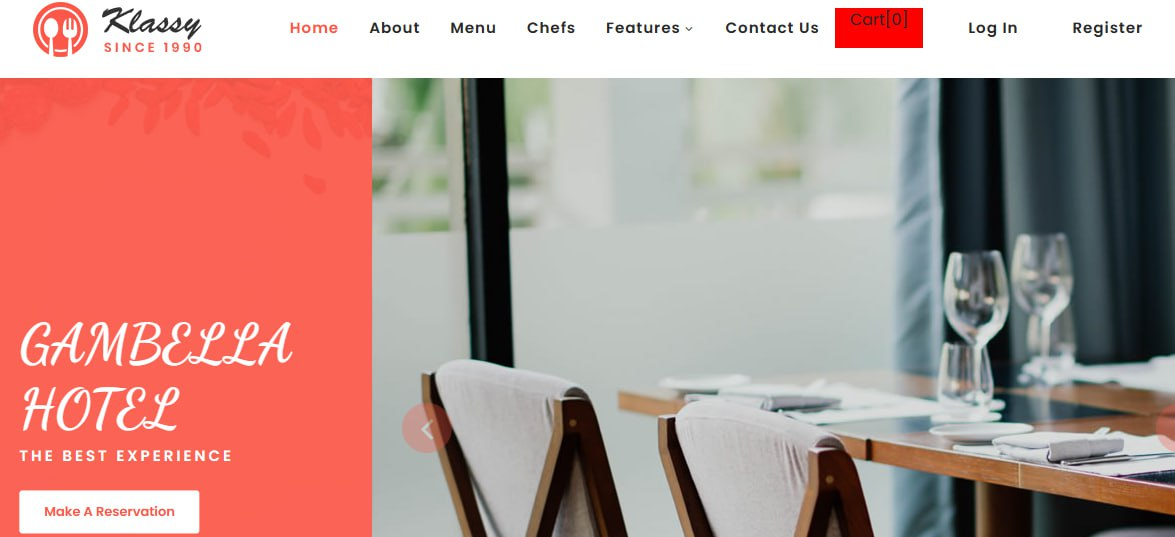
***Login sequence diagram***



***Place order sequence diagram***



## **3.5. User Interface Prototyping**



# CHAPTER 4

# OBJECT ORIENTED DESIGN

## 4.1. Introduction

Object-Oriented Design (OOD) is a process that involves developing a software system by using a set of concepts and principles that revolve around objects. In this design methodology, objects are the fundamental units of the system, and they interact with each other to achieve the system's goals.

In the context of our system, Object-Oriented Design can be used to create a software system that allows customers to place their food orders online. The system can be designed to include various objects such as the Manager, Customer, Chef, Delivery, Order, Payment, and Food classes, each with its unique set of attributes and methods.

The primary goal of Object-Oriented Design is to create a system that is modular, scalable, and easy to maintain. By using this design methodology, we can create a software system that is flexible and adaptable to changes in requirements. Additionally, OOD helps to improve the system's quality by ensuring that it is well-organized, easy to understand, and reusable.

## 4.2. Overview of the Phase I

Phase I of Object-Oriented Design would involve the following activities:

* **Requirements Gathering:** In this stage, we would identify the requirements for the system, such as the ability for customers to place food orders online, view menus, make payments, and track their orders.
* **Analysis:** Once we have gathered the requirements, we would analyze them to identify any potential issues or conflicts. This includes identifying any gaps in the requirements and resolving any ambiguities.
* **Design Objectives:** In this stage, we would define the design objectives for the system. This includes identifying the system's key features, such as the ability for customers to view menus, place orders, and track their orders. We would also identify the system's performance requirements, such as the need for the system to be fast and responsive, and the system's usability goals to ensure that the system is easy to use.
* **Use Case Modeling:** We would identify the actors, goals, and scenarios for the system. For example, the actors would include the Manager, Customer, Chef, and Delivery classes, and the goals would include placing orders, viewing menus, and making payments.
* **Class Modeling**: We would identify the classes required for the system, such as the Manager, Customer, Chef, Delivery, Order, Payment, and Food classes. We would define the attributes and methods for each class and define the relationships between them.
* **Architecture Design:** In this stage, we would define the overall architecture for the system. This includes identifying the subsystems and components required for the system, such as the user interface, database, and payment gateway, and defining their interactions.

## 4.2 Class Type Architecture

## (The UI Layer, The Controller Layer, Business/Domain Layer, Persistence Layer, System Classes)

Class Type Architecture is a software architecture pattern that organizes the classes in a system into different layers, each with a specific responsibility. These layers typically include the UI layer, the controller layer, the business/domain layer, the persistence layer, and the system classes. Here's a brief description of each layer:

1. **UI Layer:** This layer is responsible for handling user interactions and presenting information to the user. It typically consists of classes related to user interfaces, such as forms, screens, widgets, and views. The UI layer communicates with the other layers to request and display data, but it should not contain any business logic.

Example: Customer

2**. Controller Layer:** The controller layer acts as an intermediary between the UI layer and the business/domain layer. It receives user input from the UI layer, performs any necessary processing or validation, and coordinates the appropriate actions in the system. It often includes classes such as controllers, presenters, or view models.

1. **Business/Domain Layer:** This layer encapsulates the core business logic and rules of the application. It represents the concepts and behavior specific to the problem domain. Classes in this layer typically model entities, services, business rules, and workflows. The business/domain layer should be independent of any particular UI or persistence mechanism.

Example: Chef, manager, order, payment, delivery, food and customer

1. **Persistence Layer:** The persistence layer is responsible for data storage and retrieval. It includes classes and components that handle database access, file I/O, or any other means of persisting data. This layer abstracts the underlying data storage technology and provides a way for the other layers to interact with the data. Common classes in this layer are repositories, data access objects (DAOs), or data mappers.

Example: payment, food

**4. System Classes:** Apart from the specific layers mentioned above, there might be additional system classes that support the overall functioning of the application. These classes can include utility classes, configuration classes, logging or error handling classes, and other infrastructure components required by the system.

## 4.3 Design Class Modeling

Design class modeling is a process of creating a visual representation of the classes, their attributes, and their relationships in a software system. It is an important part of the design phase of software development, where the requirements are translated into a high-level design that can be used for implementation.

Design class modeling typically involves creating a class diagram, which is a graphical representation of the classes in the system and their relationships.

### 4.3.1. Class Diagram

****

### 4.3.2 Description of classes

**Class Name: Manager**

**Description:** It represents a manager who is responsible for managing the restaurant's menu, viewing order histories and sales reports, and managing customer accounts.

**Attributes:**

* managerID: It represents the unique identifier for the manager
* name: It represents the name of the manager
* email: It represents the email address of the manager
* phoneNumber: It represents the phone number of the manager
* password: It represents the password of the manager's account

**Methods:**

* addMenuItem(): a method that adds a new menu item to the restaurant's menu
* deleteMenuItem(): a method that removes a menu item from the restaurant's menu
* updateMenuItem(): a method that updates an existing menu item in the restaurant's menu
* viewOrderHistory(): a method that returns a list of all orders made by customers
* viewSalesReport(): a method that returns a dictionary of sales reports for the restaurant

**Class Name: Customer**

**Description**: It represents a customer who can view the restaurant's menu, place orders, and make payments.

**Attributes:**

* customerID: It represents the unique identifier for the customer
* name: It represents the name of the customer
* email: It represents the email address of the customer
* phoneNumber: It represents the phone number of the customer
* password: It represents the password of the customer's account
* deliveryAddress: It represents the delivery address for the customer's orders

**Methods:**

* viewMenu(): a method that returns a list of all menu items for the restaurant
* placeOrder(): a method that places a new order for the customer
* cancelOrder(): a method that cancels an existing order for the customer
* viewOrderStatus(): a method that returns the status of an existing order for the customer
* makePayment(): a method that makes a payment for an existing order

**Class Name: Chef**

**Description**: It represents a chef who is responsible for viewing and updating the status of orders.

**Attributes:**

* chefID: It represents the unique identifier for the chef
* name: It represents the name of the chef
* email: It represents the email address of the chef
* phoneNumber: It represents the phone number of the chef
* password: It represents the password of the chef's account

**Methods:**

* viewOrder(): a method that returns a list of all orders that need to be prepared by the chef
* viewOrderDetails(): a method that returns the details of a specific order
* updateOrderStatus(): a method that updates the status of a specific order

**Class Name: Delivery**

**Description**: It represents a delivery person who is responsible for delivering orders to customers.

**Attributes:**

* deliveryId: It represents the unique identifier for the delivery person
* name: It represents the name of the delivery person
* email: It represents the email address of the delivery person
* phoneNumber: It represents the phone number of the delivery person
* password: It represents the password of the delivery person's account
* vehicleType: It represents the type of vehicle the delivery person will be using for deliveries

**Methods:**

* viewDeliverySchedule(): a method that returns a list of all orders that need to be delivered by the delivery person
* viewDeliveryDetails(): a method that returns the details of a specific order that needs to be delivered
* updateDeliveryStatus(): a method that updates the delivery status of a specific order

**Class Name: Order**

**Description**: It represents an order made by a customer, containing information about the customer, menu items, payment, and delivery.

**Attributes:**

* orderID: It represents the unique identifier for the order
* customerID: It represents the unique identifier for the customer who placed the order
* chefID: It represents the unique identifier for the chef who will prepare the order
* deliveryID: It represents the unique identifier for the delivery person who will deliver the order
* orderDate: It represents the date and time the order was placed
* deliveryDate: It represents the date and time the order will be delivered
* orderStatus: It represents the status of the order
* paymentStatus: It represents the status of the payment for the order

**Methods:**

* addMenuItem(): a method that adds a new menu item to the order
* removeMenuItem(): a method that removes a menu item from the order
* updateMenuItem(): a method that updates an existing menu item in the order
* calculateTotal(): a method that calculates the total cost of the order
* confirmOrder(): a method that confirms the order and updates the order status
* cancelOrder(): a method that cancels the order and updates the order status

**Class Name: Payment**

**Description**: It represents a payment made by a customer for an order.

**Attributes:**

* paymentID: It represents the unique identifier for the payment
* customerID: It represents the unique identifier for the customer who made the payment
* orderID: It represents the unique identifier of the order for which the payment was made
* paymentDate: It represents the date and time the payment was made
* paymentAmount: It represents the amount of the payment
* paymentMethod: It represents the payment method used by the customer (e.g. credit card, PayPal, etc.)

**Methods:**

* makePayment(): a method that makes a payment for an existing order
* updatePaymentStatus(): a method that updates the payment status of an existing order
* viewPaymentHistory(): a method that returns a list of all payments made by a specific customer

**Class Name: Food**

**Description**: It represents a food item on the restaurant's menu.

**Attributes:**

* foodID: It represents the unique identifier for the food item
* name: It represents the name of the food item
* description: It represents the description of the food item
* price: It represents the price of the food item
* ingredients: It represents the ingredients used in the food item

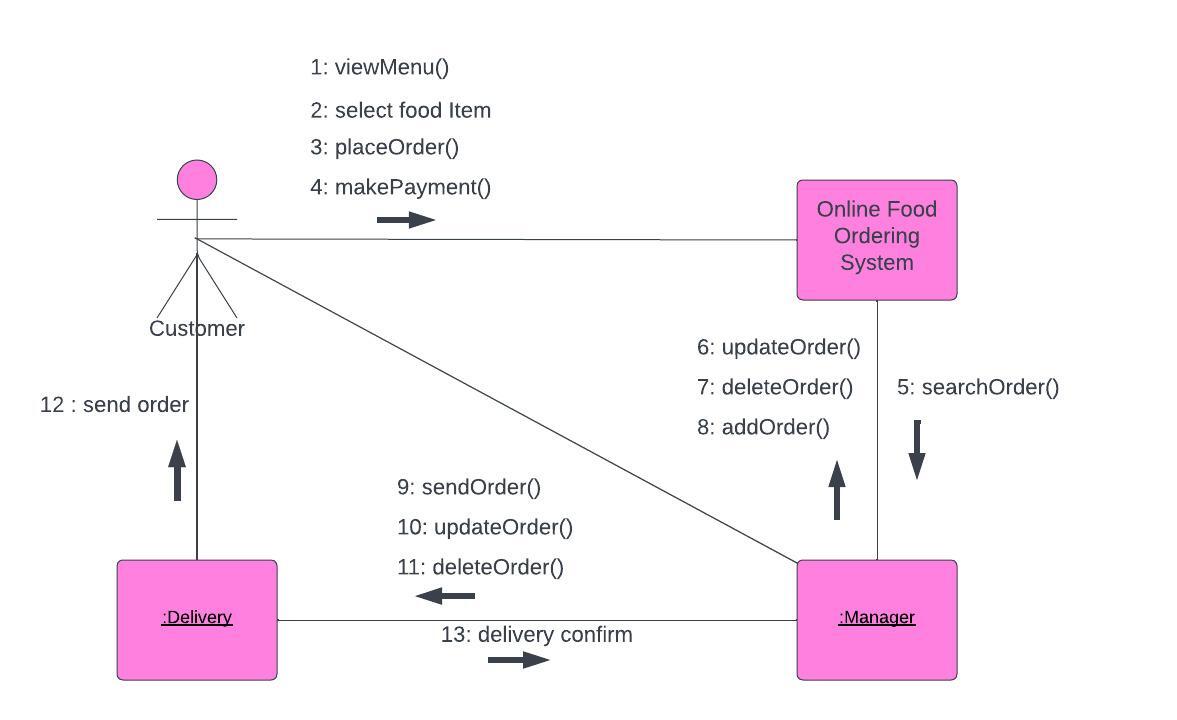
**Methods:**

* viewFoodDetails(): a method that returns a string containing the details of the food item
* updateFoodDetails(): a method that updates the details of the food item

## 4.4 Collaboration Diagram

A collaboration diagram is a visual representation of how different objects and components within the system interact with each other to achieve a specific goal.

In the case of our system, the diagram would show how the customer, the manager, and the delivery service work together to place and deliver an order.



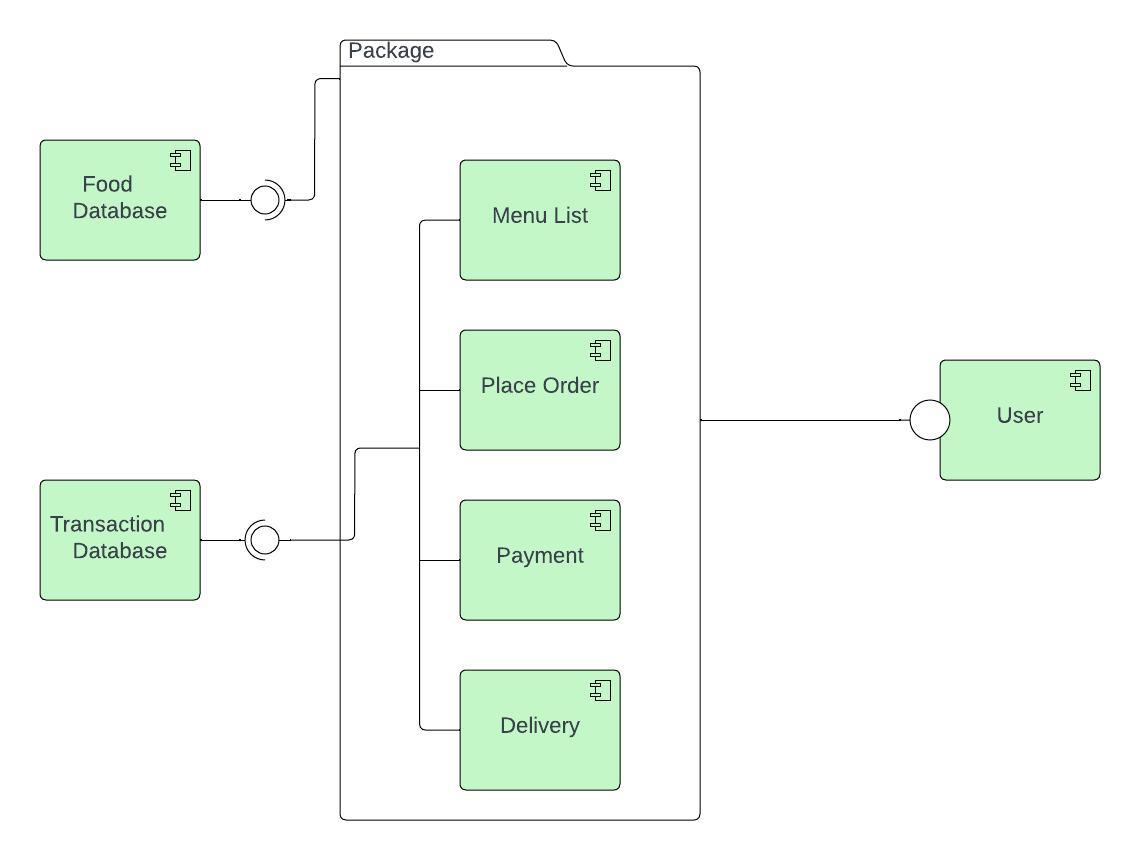
This collaboration diagram shows the interaction between customer, manager, system and delivery objects in online food ordering system.

## 4.5 Component Diagram / Sub-System Decomposition

A component diagram in the (UML) Unified Modeling Language shows how parts are wired together to create bigger parts of the food ordering system. They are used to show the structure of the food ordering system. The UML component diagram shows how a software system will be made up of a set of deployable components, such as dynamic-link library (DLL) files, executable files, or web services. Using well-defined interfaces, these parts communicate with each other and keep their internal details hidden from each other and the outside world.

Here are the benefits of designing the food ordering component diagram:

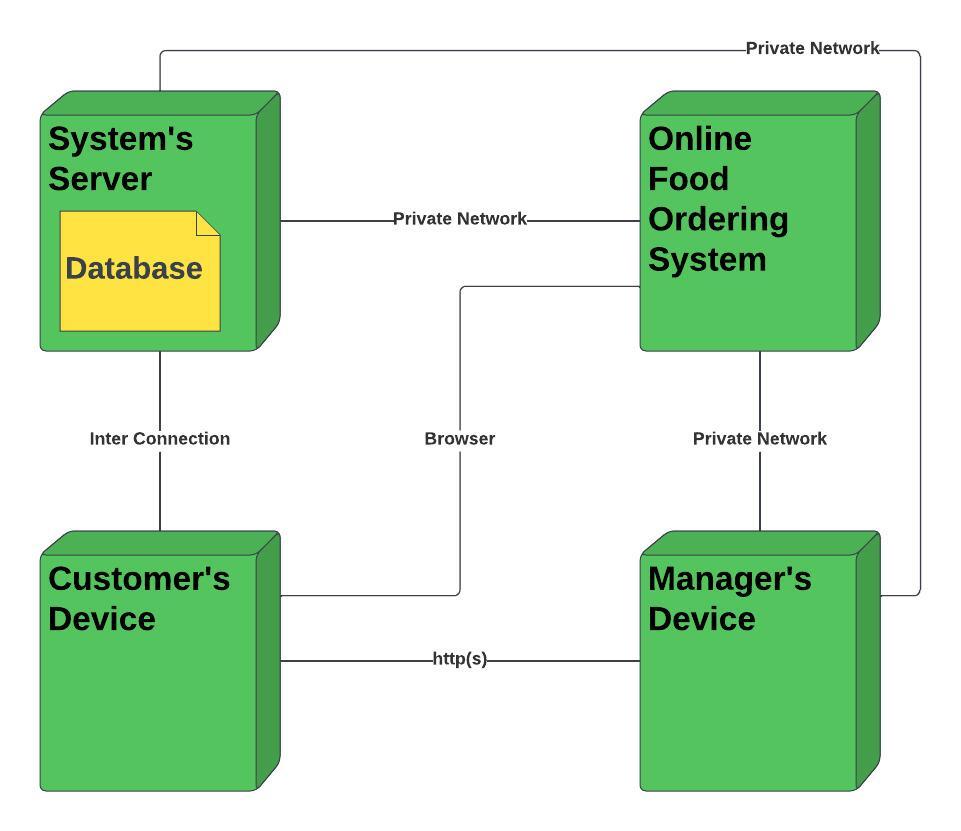
* Imagine how the system looks in real life.
* Pay attention to the system's parts and how they work together.
* Pay attention to how the service behaves when it comes to the interface.



This component diagram of food ordering system has 7 components which are food database, transaction database, menu list, place order, payment, delivery, and the user. The components inside the package are the required interfaces that are dependent on the provided food database component and vice versa for the transaction database. The included components were just based on the main function of the system.

## 4.6 Deployment Diagram

The Online Food Ordering System UML deployment diagram explains the sketch of the relationship between software and hardware. These hardware and software are labeled to clarify their part in the system's operation. They were represented by nodes and the connections were represented by labeled arrows.



This deployment diagram shows the scenario when the system is deployed. It has 4 nodes represented with boxes and relationship connections. The nodes are the online food ordering system, the customer's device, the manager's device, and the database (system server). The system server node contains a developed database that will hold the details of the system online. For the connection, the system is connected to the server database using a private network which enables it to pass a connection to the devices and enable users to access the system and database. The Manager and the customer then can communicate using an online or internet connection.

## 4.7 Relational Persistent Model

Relational persistence model refers to the way data is stored and retrieved in a relational database management system (RDBMS). In this model, data is organized into tables, with each table having a unique name and a predefined set of columns. The columns represent the attributes of the data, while the rows represent the individual records.

### 4.7.1 Mapping the Design Class into Relations

We map the design class into relation as follow:

* ***Mapping Manager Class***

There will be Manager table with ManagerID, Name, Email, Password, PhoneNumber being the columns.

**Manager**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ManagerID(PK) | Name | Email | Password | PhoneNumber |

* ***Mapping Delivery Class***

There will be Delivery table with DeliveryID, Name, Email, Password, VehicleType, PhoneNumber being the columns.

**Delivery**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| DeliveryID(PK) | Name | VehicleType | Password | PhoneNumber | Email |

* ***Mapping Chef Class***

There will be Chef table with ChefID, Name, Email, Password, PhoneNumber being the columns.

**Chef**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ChefID(PK) | Name | Email | Password | PhoneNumber |

* ***Mapping Food Class***

There will be Food table with FoodID, Name, Description, Price, Ingredients being the columns.

**Food**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FoodID(PK) | Name | Description | Price | Ingredients |

* ***Mapping Customer Class***

There will be Food table with CustomerID, Name, Email, Password, DeliverAddress, PhoneNumber being the columns.

**Customer**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CustomerID(PK) | Name | DeliverAddress | Email | Password | PhoneNumber |

* ***Mapping Payment Class***

There will be Food table with PaymentID, PaymentDate, PaymentMethod, PaymentAmount being the columns

**Payment**

|  |  |  |  |
| --- | --- | --- | --- |
| PaymentID(PK) | PaymentDate | PaymentMethod | PaymentAmount |

* ***Mapping Order Class***

There will be Food table with OrderID, OrderDate, DeliveredDate being the columns

**Order**

|  |  |  |
| --- | --- | --- |
| OrderID(PK) | OrderDate | DeliveredDate |

* ***Mapping manages relationship***

As the relationship is having one-to-many cardinality, the PK or FK of the one side (PK or FK of Manager table) should be posted to many side (Delivery table and Older table).

This will require adding the PK of Manager(ManagerID) in the Delivery table and Older table as a foreign key.

**Delivery**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| DeliveryID(PK) | Name | VehicleType | Email | Password | PhoneNumber | ManagerID(FK) |

**Order**

|  |  |  |  |
| --- | --- | --- | --- |
| OrderID(PK) | OrderDate | DeliveredDate | ManagerID(FK) |

* ***Mapping delivers relationship***

As the relationship is having one-to-one cardinality, the PK or FK of the either side should be posted to other side as foreign key.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| DeliveryID(PK) | Name | VehicleType | OrderID(FK) | Email | Password | PhoneNumber | ManagerID(FK) |

* ***Mapping oversees relationship***

As the relationship is having one-to-many cardinality, the PK or FK of the one side (PK or FK of Manager table) should be posted to many side (Payment table).

This will require adding the PK of Manager(ManagerID) in the Payment table as a foreign key.

**Payment**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PaymentID(PK) | PaymentDate | PaymentMethod | PaymentAmount | ManagerID(FK) |

* ***Mapping made\_for relationship***

As the relationship is having one-to-many cardinality, the PK or FK of the one side (PK or FK of Order table) should be posted to many side (Payment table).

This will require adding the PK of Order(OrderID) in the Payment table as a foreign key.

**Payment**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PaymentID(PK) | PaymentDate | PaymentMethod | PaymentAmount | ManagerID(FK) | OrderID(FK) |

* ***Mapping having relationship***

As the relationship is having one-to-many cardinality, the PK or FK of the one side (PK or FK of Customer table) should be posted to many side (Payment table).

This will require adding the PK of Customer(Customer ID) in the Payment table as a foreign key.

**Payment**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PaymentID(PK) | PaymentDate | PaymentMethod | PaymentAmount | ManagerID(FK) | OrderID(FK) | CustomerID(FK) |

* ***Mapping Places relationship***

As the relationship is having one-to-many cardinality, the PK or FK of the one side (PK or FK of Customer table) should be posted to many side (Order table).

This will require adding the PK of Customer(CustomerID) in the Order table as a foreign key.

**Order**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OrderID(PK) | OrderDate | DeliveredDate | ManagerID(FK) | CustomerID(FK) |

* ***Mapping assigned\_to relationship***

As the relationship is having many-to-many cardinality, we are supposed to create a table for associative entity where the PK of Chef and Order tables posted in the new table as foreign key. We can give the name to this associative entity as ChefOrder.

**ChefOrder**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ChefOrderID(PK) | OrderID(PK,FK) | ChefID(PK,FK) | StartTime | EndTime |

* ***Mapping Prepares relationship***

As the relationship is having many-to-many cardinality, we are supposed to create a table for associative entity where the PK of Chef and Food tables posted in the new table as foreign key. We can give the name to this associative entity as ChefFood.

**ChefFood**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ChefFoodID(PK) | ChefID(PK,FK) | FoodID(PK,FK) | Specialty | Experience |

* ***Mapping Consists\_of relationship***

As the relationship is having many-to-many cardinality, we are supposed to create a table for associative entity where the PK of Order and Food tables posted in the new table as foreign key. We can give the name to this associative entity as FoodOrder.

**FoodOrder**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FoodOrderID(PK) | OrderID(PK,FK) | FoodID(PK,FK) | Quantity | UnitPrice |

### 4.7.2 Normalizations

Normalization is the process of organizing data in a database in order to reduce redundancy and improve data integrity. It involves breaking up a larger table into smaller, more specific tables and establishing relationships between them.

There are several levels of normalization, each with specific guidelines and requirements:

**1. First Normal Form (1NF):** The table must have a primary key, and each column must contain atomic values (i.e. values that cannot be further decomposed).

**2. Second Normal Form (2NF**): The table must be in 1NF, and all non-key attributes must be fully dependent on the primary key.

**3. Third Normal Form (3NF):** The table must be in 2NF, and all non-key attributes must be independent of each other (i.e. there should be no transitive dependencies).

There are additional levels of normalization beyond 3NF, such as Boyce-Codd Normal Form (BCNF) and Fourth Normal Form (4NF), but they are less commonly used.

We have normalized our table and the normalized table written below:

**Manager**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ManagerID(PK) | Name | Email | Password | PhoneNumber |

**Delivery**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| DeliveryID(PK) | Name | OrderID(PK) | VehicleType | Email | Password | PhoneNumber | ManagerID(FK) |

**Chef**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ChefID(PK) | Name | Email | Password | PhoneNumber |

**Order**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OrderID(PK) | OrderDate | DeliveredDate | ManagerID(FK) | CustomerID(FK) |

**Payment**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PaymentID(PK) | PaymentDate | PaymentMethod | PaymentAmount | ManagerID(FK) | OrderID(FK) | CustomerID(FK) |

**Customer**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CustomerID(PK) | Name | DeliverAddress | Email | Password | PhoneNumber |

**Food**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FoodID(PK) | Name | Description | Price | IngredientsID(FK) |

**Ingredients**

|  |  |  |  |
| --- | --- | --- | --- |
| IngredientsID(PK) | Name | NutritionalInfo | Description |

**ChefOrder**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ChefOrderID(PK) | OrderID(PK,FK) | ChefID(PK,FK) | StartTime | EndTime |

**ChefFood**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ChefFoodID(PK) | ChefID(PK,FK) | FoodID(PK,FK) | Specialty | Experience |

**FoodOrder**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FoodOrderID(PK) | OrderID(PK,FK) | FoodID(PK,FK) | Quantity | UnitPrice |

### 4.7.3. Database Diagram

A database diagram is a visual representation of the structure and relationships within a database. It provides an overview of the tables, columns, and relationships between the tables in a database. Database diagrams are commonly used by database administrators, developers, and analysts to understand the organization of data and to design and document database systems.

****

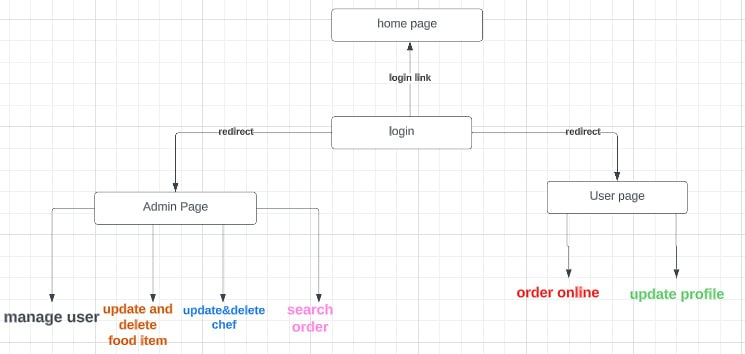
## 4.8. User Interface

The user interface (UI) refers to the graphical interface through which users interact with the system to place food orders.

The UI typically includes various components such as menus and buttons that allow users to browse through the available food items, select their desired items, customize their orders, and submit their orders for processing.

The UI should be designed in a user-friendly and intuitive manner to ensure that users can easily navigate through the system and place their orders without any confusion or frustration**.**

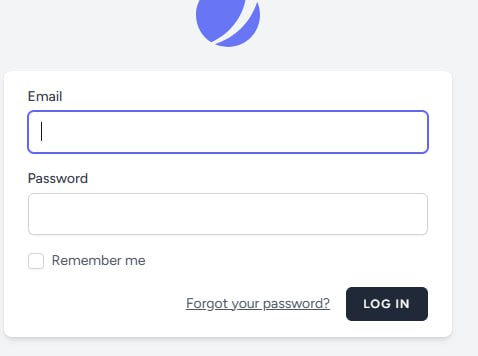
### 4.8.1. User Interface Flow Diagram



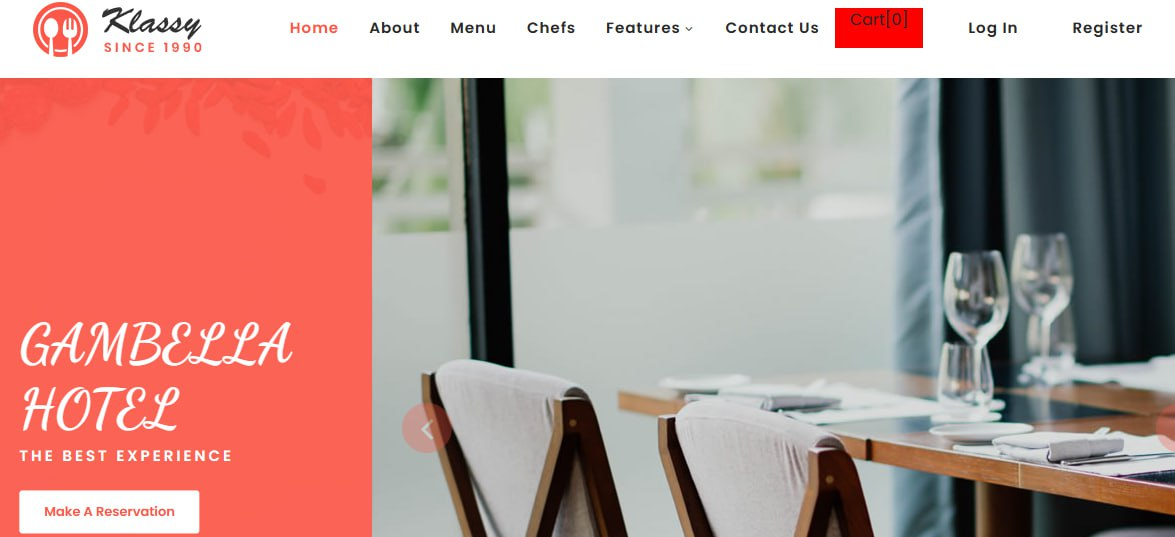
### 4.8.2 User Interface Design

Here are the UI.

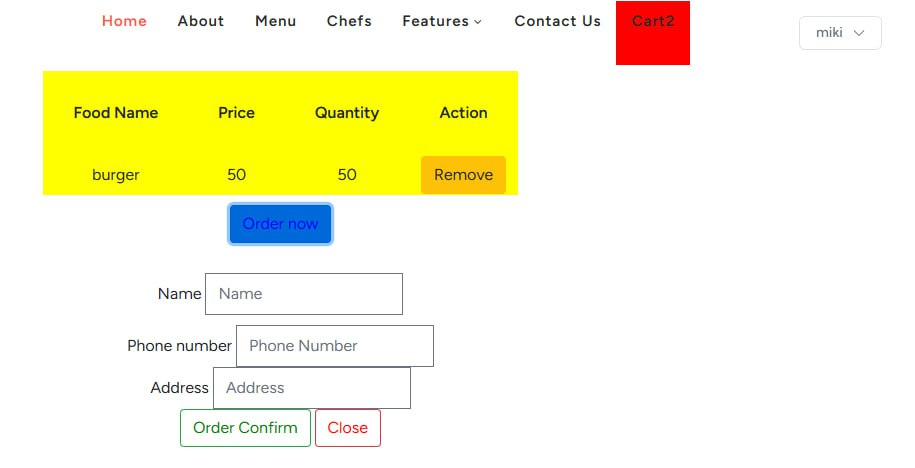
***UI for Login***



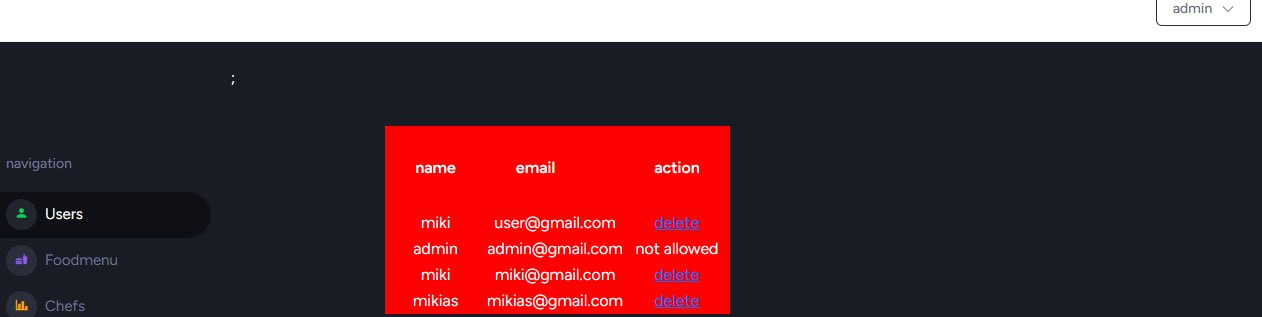
**UI for Homepage**



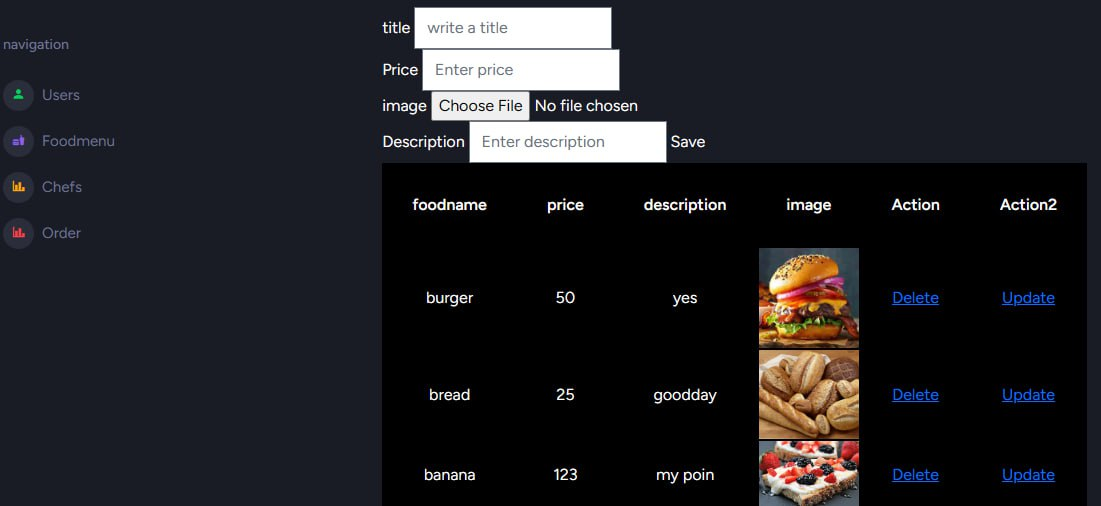
***UI for Order Page***



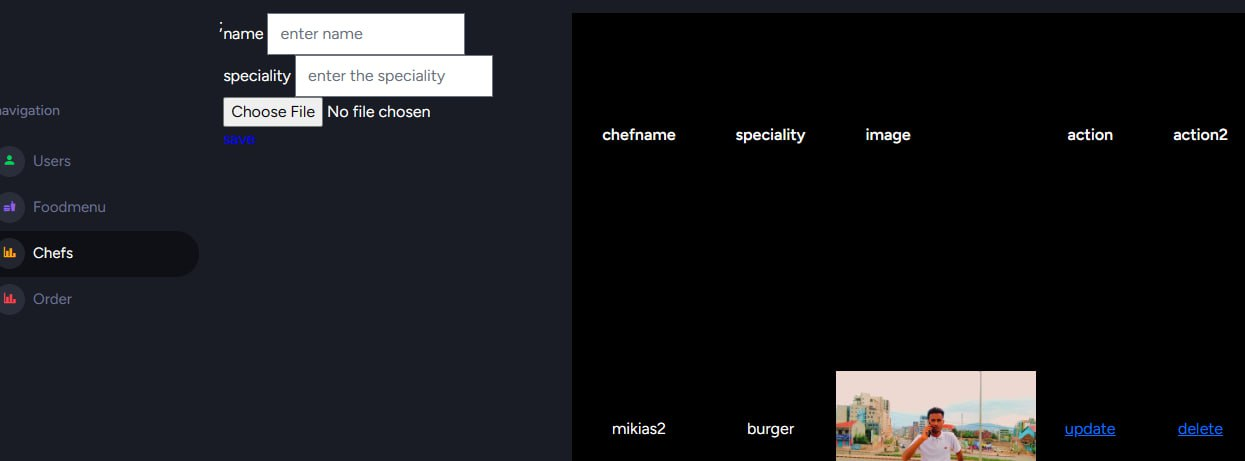
***UI for Manager User page***



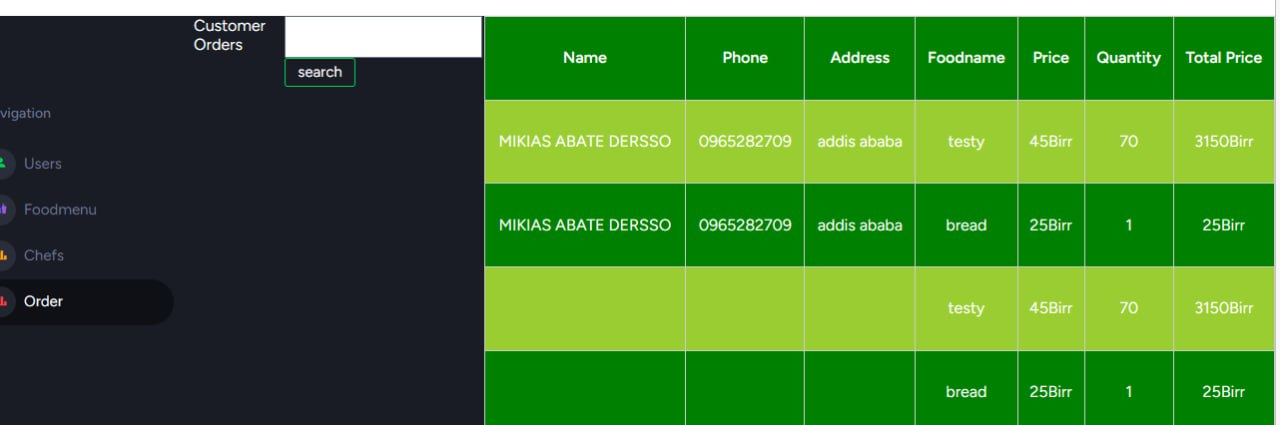
***UI for FoodMenu page***



***UI for Chef page***



***UI for OrderStatus Page***



# CHAPTER 5

# OBJECT ORIENTED IMPLEMENTATION

## 5.1 Introduction

Gambela hotel is located in Addis Ababa, Ethiopia, is a popular destination for guests seeking high-quality food and beverages. To improve the experience of its customers and streamline its operations, the hotel has decided to implement an online food ordering system. The system will enable customers to browse the menu, select their preferred items, place an order, and make a payment online. The hotel will also be able to manage orders efficiently, track inventory, and analyze customer data to improve the quality of service.

To implement the online food ordering system for Gambela hotel, we will use object-oriented programming (OOP) principles. OOP is a programming paradigm that involves creating objects that encapsulate data and behavior, and can communicate with other objects to accomplish tasks. We will use OOP to create several classes, including a manager class, a menu class, a customer class, a chef class, an order class, a food class, a delivery class, and a payment class.

Each class will have its methods and attributes that will enable it to perform its specific tasks.

To ensure that the system is scalable, efficient, and easy to maintain, we will use design patterns such as the Singleton pattern and the Factory pattern. We will also use the Model-View-Controller (MVC) architecture to separate the system's components into three layers: the model layer, which manages data and business logic; the view layer, which handles user interface elements; and the controller layer, which manages communication between the model and view layers.

To ensure that the system is reliable, secure, and user-friendly, we will perform unit testing, integration testing, and system testing. We will test each class and its methods to ensure that it works as expected and meets the system's requirements. We will also test the interactions between the classes to ensure that they work seamlessly together to provide a smooth and efficient user experience.

In conclusion, we used Internet programming language and OOP principles will enable the hotel to provide a seamless and convenient experience for its customers. The use of design patterns and the MVC architecture will ensure that the system is scalable, efficient, and easy to maintain. The testing procedures will ensure that the system is reliable, secure, and user-friendly.

## 5.2 Implementation Technology

We implemented our system using object-oriented programming (OOP) principles, has been developed using internet programming languages such as HTML, CSS, JavaScript, PHP, and the Laravel framework.

The user interface of the system has been developed using HTML to create the structure of the web pages, CSS to style the pages, and JavaScript to add interactivity and dynamic content. The use of these internet programming languages has enabled us to create a user-friendly interface that is responsive, intuitive, and visually appealing.

The system's backend has been developed using PHP as the server-side scripting language, which processes user requests and generates dynamic content. The Laravel framework has been used to develop the system's backend, which includes managing the database, processing payments, and handling the system's logic. The use of the Laravel framework has enabled us to develop a scalable and efficient system that is easy to maintain and update.

## 5.3 Testing and testing procedure

To ensure that our system is reliable, secure, and user-friendly, we will perform several types of testing, including unit testing, integration testing, and system testing.

### 5.3.1 Unit Testing

**Test case format**

**Feature to be tested | input data| Expected Output| Actual Output | pass/Fail**

Unit testing is a type of software testing that verifies the functionality of individual components or units of a software application. It is performed by testing each unit in isolation to ensure that it works as expected and meets the specified requirements. The goal of unit testing is to identify any defects or issues in the code early in the development process, which can save time and resources and improve the quality of the software.

In the context of our system, unit testing involves testing each class and its methods in isolation. For example, We would also test the order class and its methods, such as placeOrder() and updateOrderStatus(), to ensure that they work as expected.

Unit testing typically involves defining test cases that cover all possible scenarios and edge cases for each unit.

Each test case consists of:

* **Input data** represents the parameters passed to the unit, while
* **Expected output** represents the expected result of the unit based on the input data.
* **Actual output** represents the actual result of the unit after it has been executed with the input data.

During unit testing, we may encounter issues such as:

* **Logic errors** may arise if the code does not produce the expected output due to a mistake in the logic or algorithm.
* **Boundary errors** may arise if the code does not handle edge cases or boundary conditions correctly.
* **Exception errors** may arise if the code does not handle exceptions or error conditions correctly.

Here's an example of how unit testing could be performed for the addMenuItem() method of the Manager class:

Test case format:

**Feature to be tested:** Adding a menu item to the menu list

**Input data:**

- Item: Food(name="Chicken curry", description="Spicy chicken curry with rice", price=10.99, ingredients=["chicken", "rice", "curry powder"])

**Expected output:**

- The Food object should be added to the menu list.

**Actual output:**

- The Food object was added to the menu list.

**Pass/Fail: Pass**

Explanation: This unit test case tests the functionality of the Manager class' addMenuItem() method. The input data consists of a Food object that represents a new menu item to be added to the menu list. The expected output is that the Food object should be added to the menu list. The actual output is that the Food object was added to the menu list as expected. This test case passed, indicating that the addMenuItem() method works correctly.

### 5.3.2 Integration Testing

Integration testing is a type of software testing that verifies the interactions between different components of a software system. It is performed after unit testing and before system testing, and it aims to identify any defects or issues that may arise when the components are combined or integrated. Integration testing is essential in ensuring that the system is consistent and that all components work together as expected.

In the context of our system, integration testing involves testing the interactions between the different classes involved in the system. For example, We would also test the interaction between the order class and the payment class to ensure that the payment is processed correctly after the order is placed.

For Example:

**Test case format:**

**Feature to be tested:** Placing an order and updating its status

**Input data:**

- Customer: Customer(customer\_id=1, name="Adisu Wakeyo", email="adisuwakeyo@gmail.com", phone\_number="0979798622", password="1234", delivery\_address="FBE Campus")

- Order: Order(order\_id=1, customer\_id=1, chef\_id=2, delivery\_id=3, order\_date=datetime.datetime(2023, 5, 20, 10, 0), delivery\_date=datetime.datetime(2023, 5, 20, 13, 0)

- Menu item: Food(name="Chicken curry", description="Spicy chicken curry with rice", price=10.99, ingredients=["chicken", "rice", "curry powder"])

**Expected output:**

- The order should be placed successfully, and its status should be updated to "Delivered".

**Actual output:**

- The order was placed successfully, and its status was updated to "Delivered".

**Pass/Fail:** Pass

Explanation: This integration test case tests the interaction between the Customer class, the Order class, and the Chef class. The input data consists of a Customer object, an Order object, and a Food object that represents a menu item. The expected output is that the order should be placed successfully, and its status should be updated to "Delivered". The actual output is that the order was placed successfully, and its status was updated to "Delivered" as expected. This test case passed, indicating that the integration between the Customer, Order, and Chef classes works correctly.

## 5.3.3 System Testing

System testing is a type of software testing that verifies the functionality and performance of a complete software system or application. It is performed after integration testing and before user acceptance testing to ensure that the system meets all the specified requirements and is ready for release to the end-users. It is designed to ensure that the system is working correctly and meets all the functional and non-functional requirements, such as performance, reliability, security, and usability.

The primary goal of system testing is to identify any defects or issues in the system that were not identified during unit testing or integration testing.

In the context of our system, it would involve testing the entire system as a whole to ensure that it meets all the functional and non-functional requirements.

This would typically involve a range of testing techniques, including:

**Functional Testing:** would involve testing the system against the functional requirements to ensure that it behaves as expected.

**For example,**

* Test the ability of customers to view the menu, place orders, and make payments.
* Test the ability of managers to add, remove, and update menu items, view order history and sales report.
* Test the ability of chefs to view order queue and update order status.
* Test the ability of delivery personnel to view delivery schedule and update delivery status.

**Performance Testing:** would involve testing the system's performance under different load and stress conditions to ensure that it meets the performance requirements.

**For example,**

* Test the system's ability to handle a large number of orders during peak hours.
* Test the system's ability to process payments quickly and efficiently.

**Security Testing:** would involve testing the system's security features to ensure that it is secure and protected against potential threats.

**For example,**

* Test the system's login and authentication features.
* Test the system's protection against common security vulnerabilities such as cross-site scripting (XSS), SQL injection, and other common security issues.

**Usability Testing:** would involve testing the system's user interface and user experience to ensure that it is user-friendly and easy to use.

**For example,**

* Test the system's navigation, layout, and design.
* Test the system's user experience to ensure that it is user-friendly and easy to use.

To give a specific example, we could perform a system test case to verify the ability of a customer to place an order and make a payment:.

**Test case format:**

**Feature to be tested:** Placing an order and making a payment

**Input data:**

* Customer: Customer(customer\_id=1, name="Adisu Wakeyo", email="adisuwakeyo@gmail.com", phone\_number="0979798622", password="1234", delivery\_address="FBE Campus")
* Order: Order(order\_id=1, customer\_id=1, chef\_id=2, delivery\_id=3, order\_date=datetime.datetime(2023, 5, 20, 10, 0), delivery\_date=datetime.datetime(2023, 5, 20, 13, 0)
* Menu item: Food(name="Chicken curry", description="Spicy chicken curry with rice", price=10.99, ingredients=["chicken", "rice", "curry powder"])
* Payment method: "Credit card"
* Payment amount: 10.99

**Expected output:** The order should be placed successfully, and the payment should be processed and recorded in the system.

**Actual output:** The order was placed successfully, and the payment was processed and recorded in the system.

**Pass/Fail:** Pass

In this test case, we would simulate a customer placing an order for a menu item using a credit card payment method. We would verify that the order was placed successfully and that the payment was processed and recorded in the system.

To perform this test case, we would follow these steps:

1. The customer logs in to the system using their credentials.
2. The customer navigates to the menu and selects the desired menu item.
3. The customer places the order with the selected menu item.
4. The system generates an order ID and assigns it to the order.
5. The system sets the order status to "Pending".
6. The system calculates the total cost of the order.
7. The customer selects the credit card payment method and provides the necessary payment details.
8. The system processes the payment and records it in the system.
9. The system updates the order status to "Confirmed" and assigns the order to the chef and delivery personnel.
10. The system sends a confirmation email to the customer with the order details.

## 5.3 Deployment/Installation process

Here are the general steps for deploying and installing our system

**1. System Requirements:** Ensure that the system meets the necessary hardware and software requirements, including the operating system, web server, database server, and any other required software.

**2. Installation:** Install the necessary software components and configure the system according to the system requirements. This may include setting up the web server, database server, and any other required software components.

**3. Database Setup:** Set up the database and import any necessary data, such as menu items, customer details, and order history.

**4. Configuration:** Configure the system by updating any necessary configuration files, such as the database connection settings, email settings, and system settings.

**5. Data Migration:** If the system requires data migration from an existing system or database, migrate the data to the new system.

**6. Testing:** Perform system testing to ensure that the software is functioning correctly and meeting the needs of the customers, managers, chefs, and delivery personnel.

**7. Deployment:** Once the system has been tested and verified, deploy the system to the production environment. This may involve copying files to the web server, configuring the database connection, and updating any necessary configuration files.

**8. Monitoring:** After deployment, monitor the system to ensure that it is running smoothly and efficiently. This may involve monitoring server logs, database performance, and system usage patterns.

**9. Maintenance:** Perform regular maintenance tasks, including software updates, security patches, and backups, to ensure that the system remains secure and up-to-date.

**10. User Training:** Train the employees who will be using the system, such as the managers, chefs, and delivery personnel, to ensure that they are familiar with the system and can use it effectively. It may also be necessary to train the customers on how to use the system to place orders.

**11. Integration:** If the system requires integration with other systems, such as the hotel's inventory or accounting systems, ensure that the integration is working correctly and that the necessary configuration is in place.

**12. Scalability:** Consider the system's scalability and ensure that the system can handle the expected load and traffic. This may involve load testing and performance tuning.

**13. Disaster Recovery:** Develop and implement a disaster recovery plan to ensure that the system can recover from any unexpected failures or outages.

Overall, the deployment and installation process must be comprehensive, well-planned, and executed to ensure a successful deployment of the system. By following these steps and considering these key factors, we can ensure that the system is reliable, secure, and meets the needs of the users.

The installation process to install our system would involve the following steps:

1. Download the software package from the vendor's website or repository: You will need to download the software package for your online food ordering system from the vendor's website or repository. Make sure to download the correct version of the software that is compatible with your server's operating system and other software components.
2. Extract the software package to a directory on the server: Once you have downloaded the software package, extract it to a directory on the server where you want to install the system. This could be a directory in the web server's document root directory or any other directory that you choose.
3. Install the necessary software components: Before you begin the installation, make sure to install any necessary software components that are required for your system to run, such as the web server, database server, and other required software. You can use package managers or other installation methods to install the necessary software components.
4. Configure the application: Once you have installed the necessary software components, you will need to configure your online food ordering system by updating any necessary configuration files. This may include updating the database connection settings, email settings, and other system settings.
5. Migrate data: If you have an existing system or database, you may need to migrate data to the new system. This could involve exporting data from the old system or database and importing it into the new system.
6. Perform system testing: Before deploying the system to the production environment, make sure to perform system testing to ensure that the software is functioning correctly and meeting the needs of the users. This could involve testing the user interface, database connectivity, and other system features.
7. Deploy the system: Once you have tested the system and verified that it is functioning correctly, it's time to deploy the system to the production environment. This may involve copying files to the web server, configuring the database connection, and updating any necessary configuration files.
8. Monitor the system: After deployment, monitor the system to ensure that it is running smoothly and efficiently. This may involve monitoring server logs, database performance, and system usage patterns. Make sure to set up alerts and notifications to alert you of any issues or errors.
9. Perform regular maintenance tasks: Regular maintenance tasks, such as software updates, security patches, and backups, are essential to ensure that the system remains secure and up-to-date. Make sure to schedule regular maintenance tasks and perform them as needed.

# CHAPTER 6

# CONCLUSION AND RECOMMENDATION

## 6.1 Conclusion

We can conclude that the system would be a valuable addition to the hotel's services, providing customers with a convenient and efficient way to order food from the hotel's restaurant.

The system would consist of several classes, including Order, Payment, and Food, each with their own attributes and methods that would allow for the seamless processing of customer orders.

Overall, the online food ordering system would be a valuable asset for Gambela Hotel, providing customers with a modern and convenient way to order food from the hotel's restaurant, while also streamlining the ordering and payment process for the hotel staff.

## 6.2 Recommendation

We also recommend that the system should have the following features:

* **Customization options for customers**: The system should allow customers to customize their orders by adding or removing ingredients, selecting cooking preferences, and indicating any dietary restrictions or allergies.
* **Real-time order tracking**: The system should provide real-time updates on the status of each order, including when it is being prepared, when it is ready for pickup or delivery, and when it has been delivered.
* **Loyalty program:** The system should have a loyalty program that rewards customers for their repeat business, such as offering discounts or free items after a certain number of orders.
* **Feedback and reviews:** The system should allow customers to provide feedback and reviews on their orders, which can help the hotel improve its offerings and provide better service in the future.

By incorporating these features into the online food ordering system, Gambela Hotel can provide a more personalized and engaging experience for its customers, while also improving its overall service and reputation.