

Software Engineering

B7IS127



07 August 2023

10608881

KASSIM NOMAAN SAYAD

**Contents**

[**Requirements specification** 2](#_Toc142201632)

[**Evidence of Research and Analysis for Life-Cycle Model:** 4](#_Toc142201633)

[**System Architecture/Design Specification:** 5](#_Toc142201634)

[**UML Diagrams:** 6](#_Toc142201635)

# **Requirements specification**

Functional Requirements:

1. User Registration and Authentication:
   * Users should be able to create accounts and log in securely.
   * User roles should include customers, restaurant owners, delivery staff, and administrators.
2. Restaurant Management:
   * Restaurant owners should be able to register their restaurants and manage their menu items.
   * Each menu item should have attributes like name, description, price, and category.
3. Order Placement and Tracking:
   * Customers should be able to browse restaurants and their menus.
   * Customers should be able to place orders and customize them (e.g., toppings, quantity).
   * The system must support real-time order tracking for customers and delivery staff.
4. Payment Processing:
   * The system should support various payment methods for customers.
   * Secure payment processing should be implemented to protect user data.
5. Delivery Management:
   * Delivery staff should receive real-time order notifications and delivery details.
   * Delivery staff should update order statuses (e.g., picked up, delivered) as they progress.
6. Reviews and Ratings:
   * Customers should be able to rate and leave reviews for restaurants and their items.
   * The system should display average ratings and reviews for each restaurant.

Non-Functional Requirements:

1. Performance:
   * The system must handle a high number of concurrent users and orders efficiently.
   * Response times for critical operations (e.g., order placement, payment processing) should be low.
2. Security:
   * User data should be securely stored and encrypted.
   * The system should have measures to prevent common web vulnerabilities (e.g., SQL injection, cross-site scripting).
3. Scalability:
   * The architecture should be scalable to accommodate future growth and increased user load.

Stakeholders:

1. Customers: Who will be placing orders and using the application.
2. Restaurant Owners: Who will be managing their menus and accepting orders.
3. Delivery Staff: Who will be responsible for delivering the orders.
4. Administrators: Who will oversee the entire system and manage restaurants and users.

Constraints:

1. Budgetary Constraints: The development budget for the system is limited.
2. Time Constraints: The system needs to be developed and deployed within a specific timeframe.
3. Technology Constraints: The system must be compatible with existing restaurant POS systems and payment gateways.

# **Evidence of Research and Analysis for Life-Cycle Model:**

After thorough research and analysis of the food ordering software development requirements, it is evident that an Agile development life-cycle model, particularly the Scrum framework, is the most appropriate choice. Agile is well-suited for this project due to its following advantages:

* Iterative Development: The food ordering software involves continuous feedback and evolving requirements. Agile's iterative approach allows the development team to adapt quickly to changes, ensuring that customer feedback is incorporated in each sprint.
* Customer Collaboration: Agile emphasizes continuous collaboration with stakeholders. For a customer-centric application like food ordering software, involving customers throughout the development process is crucial to meeting their needs effectively.
* Quick Time-to-Market: The food industry is highly competitive, and early delivery of core functionalities is important. Agile's incremental delivery allows for the software to be deployed in usable increments, providing early value to end-users.
* Risk Management: The uncertain nature of software development can lead to unforeseen challenges. Agile mitigates these risks by breaking down the project into smaller, manageable iterations, reducing the impact of potential issues.
* Continuous Improvement: Quality is essential in the food ordering software. Agile promotes continuous improvement through regular testing and feedback, resulting in a high-quality product.

Considering these factors, Agile, particularly the Scrum framework, provides the flexibility, adaptability, and customer focus needed for the successful development of the food ordering software.

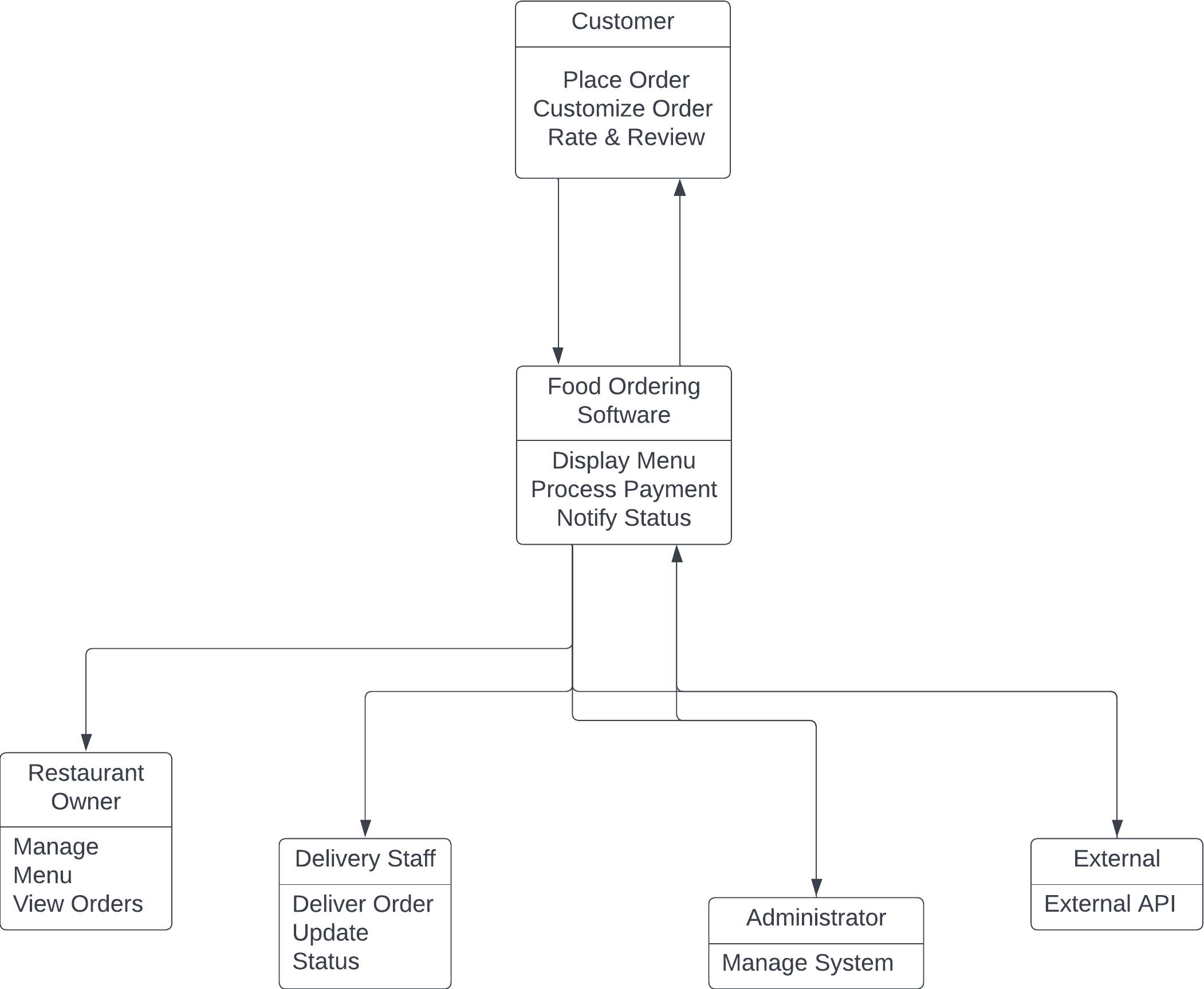
# **System Architecture/Design Specification:**

The system architecture of the food ordering software will follow a microservices architecture, which allows for a modular and scalable system. The architecture will consist of the following main components:

1. User Management Microservice: Responsible for user registration, authentication, and managing user roles.
2. Restaurant Management Microservice: Handles restaurant registration, menu management, and reviews/ratings.
3. Order Management Microservice: Manages order placement, tracking, and payment processing.
4. Delivery Management Microservice: Facilitates delivery staff operations, order notifications, and status updates.
5. Review/Rating Microservice: Handles customer reviews and restaurant ratings.

# **UML Diagrams:**

1. **Use Case Diagram:**



**Explanation:**

Actors:

1. Customer: Represents the users who want to place orders and interact with the food ordering software. They have use cases to "Place Order," "Customize Order," and "Rate & Review" restaurants.
2. Food Ordering Software: Represents the core system itself. It provides functionalities to "Display Menu" to customers, "Process Payment" for orders, and "Notify Status" to customers about the order status.
3. Restaurant Owner: Represents the users who manage restaurants and their menus. They have use cases to "Manage Menu" (add, edit, remove menu items) and "View Orders" received from customers.
4. Delivery Staff: Represents the users responsible for delivering orders to customers. They have a use case to "Deliver Order" and can "Update Status" during the delivery process.
5. Administrator: Represents the users who oversee and manage the entire food ordering system. They have a use case to "Manage System," which might involve tasks like monitoring system performance and handling system-related configurations.
6. External: Represents an external system or API that interacts with the food ordering software. It can have specific use cases, such as "External API" to integrate with external services.

Use Cases:

1. Place Order: Allows customers to place new food orders through the food ordering software.
2. Customize Order: Allows customers to customize their food orders, for example, by adding specific toppings or making special requests.
3. Rate & Review: Allows customers to rate and leave reviews for restaurants they have ordered from.
4. Display Menu: The food ordering software displays the menu items of the restaurants to customers.
5. Process Payment: The food ordering software handles the payment processing for placed orders.
6. Notify Status: The food ordering software notifies customers about the status of their orders (e.g., preparing, out for delivery).
7. Manage Menu: Allows restaurant owners to manage their restaurant's menu by adding, editing, or removing menu items.
8. View Orders: Allows restaurant owners to view the orders they have received from customers.
9. Deliver Order: Enables delivery staff to deliver orders to customers.
10. Update Status: Allows delivery staff to update the status of an order during the delivery process.
11. Manage System: Allows administrators to manage and oversee the food ordering system.
12. External API: Represents interactions with an external system or API to integrate additional services into the food ordering software.

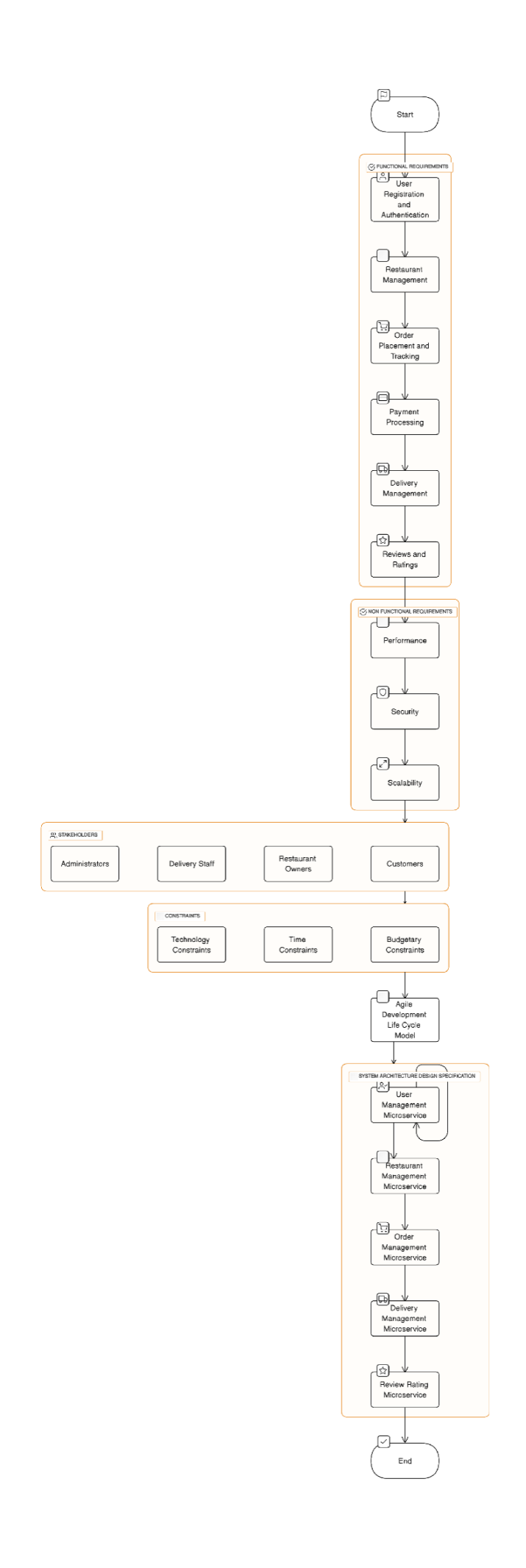
The Actor-Use Case diagram visually represents the various actors (users and external systems) and the interactions they have with the food ordering software. It outlines the main functionalities provided by the system for each actor and illustrates the relationships between actors and use cases.

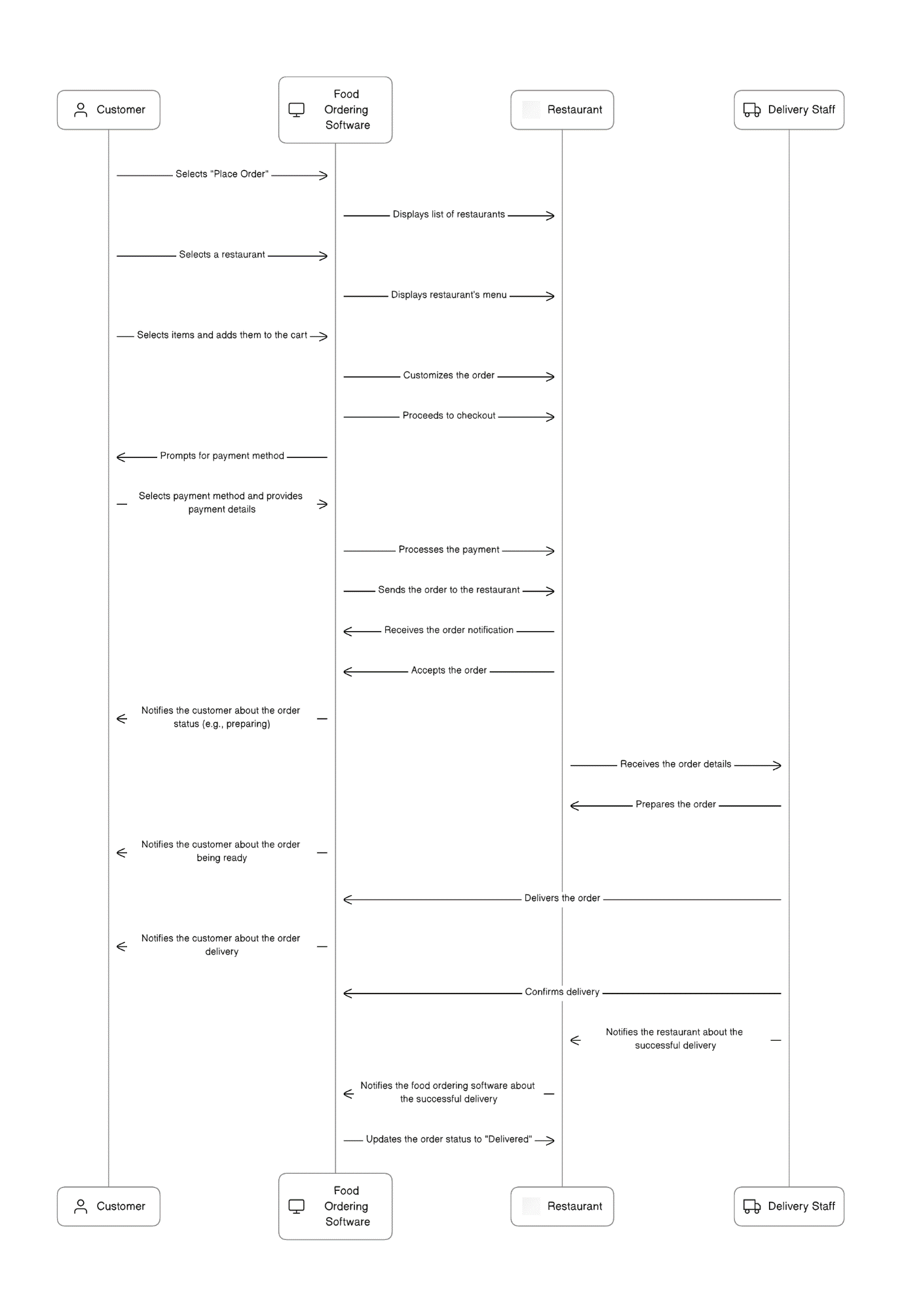
Customers can perform essential actions like placing orders, customizing orders, and leaving reviews. The food ordering software acts as the central entity that facilitates the entire food ordering process, including displaying menus, processing payments, and notifying customers about order status.

Restaurant owners can manage their menus and view orders, while delivery staff is responsible for delivering orders and updating order statuses. Administrators have control over the entire system, and external systems or APIs can integrate with the food ordering software to provide additional functionality.

Overall, the Actor-Use Case diagram provides a clear and concise overview of the various actors and their interactions with the food ordering software, outlining the key functionalities offered by the system.

1. **Flow Chart:**



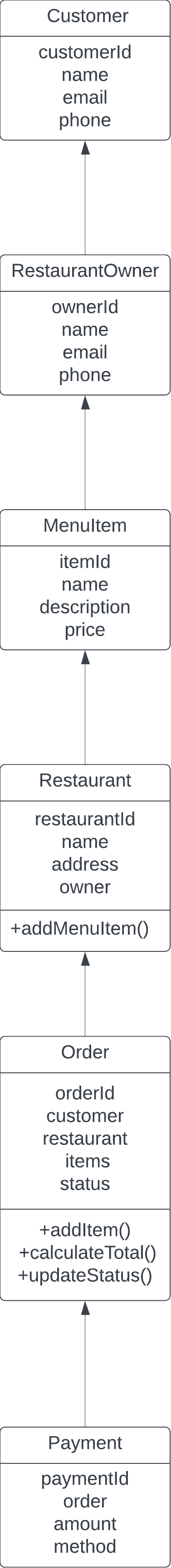
1. **Sequence Diagram:**

Explanation:

In this sequence diagram, the interactions between the Customer, Food Ordering Software, Restaurant, and Delivery Staff are illustrated during the food ordering process.

1. The Customer selects "Place Order" and browses through the list of available restaurants. The Customer selects a specific restaurant and views its menu.
2. The Customer selects the desired food items and adds them to the cart. The Customer customizes the order if needed (e.g., toppings, quantity).
3. The Customer proceeds to checkout and selects the payment method. Payment details are provided and processed securely by the Food Ordering Software.
4. Once the payment is processed, the Food Ordering Software sends the order to the selected restaurant, which then accepts the order.
5. The Food Ordering Software notifies the Customer about the order status (e.g., preparing) and sends the order details to the restaurant.
6. The restaurant prepares the order and notifies the Customer when it is ready.
7. The Delivery Staff receives the order notification and proceeds to pick up the order from the restaurant.
8. The Delivery Staff delivers the order to the Customer's location and confirms the delivery.
9. The restaurant and the Food Ordering Software are notified about the successful delivery, and the order status is updated to "Delivered."

The sequence diagram provides a clear and visual representation of the sequence of interactions and messages exchanged between the actors involved in the food ordering process. It helps in understanding the flow of actions and communications.

1. **Class Diagram**

Explanation:

1. Customer: Represents a customer who can place orders. It contains attributes like customerId, name, email, and phone. Customers can place orders.
2. RestaurantOwner: Represents the owner of a restaurant. It contains attributes like ownerId, name, email, and phone. Restaurant owners can manage their restaurants and menu items.
3. MenuItem: Represents individual items available on the restaurant's menu. It contains attributes like itemId, name, description, and price.
4. Restaurant: Represents a restaurant. It contains attributes like restaurantId, name, address, and the owner who owns the restaurant. Restaurant owners can add menu items to their restaurant.
5. Order: Represents an order placed by a customer. It contains attributes like orderId, customer (who placed the order), restaurant (where the order is placed), items (the list of menu items ordered), and status (e.g., pending, delivered). The Order class has methods to add items to the order, calculate the total amount, and update the order status.
6. Payment: Represents a payment made for an order. It contains attributes like paymentId, order (for which the payment is made), amount, and method (the payment method used).

Relationships:

* Customer has a one-to-many relationship with Order, as a customer can place multiple orders.
* RestaurantOwner has a one-to-many relationship with Restaurant, as an owner can own multiple restaurants.
* Restaurant has a one-to-many relationship with MenuItem, as a restaurant can have multiple menu items.
* Order has a one-to-many relationship with Payment, as an order can have multiple payments (e.g., partial payments, split payments).

This class diagram provides an overview of the main classes and their relationships.