Software Architecture Group Project Report

1. Functional and Non-Functional Requirements for Each Component

Inventory Management (B. Krishanth - 11212)

Functional Requirements:

1. Add, view, update, and delete inventory items.
2. Track stock levels and notify when stock is low.
3. Manage supplier details and relationships.
4. Generate inventory reports to minimize food wastage.

Non-Functional Requirements:

1. Ensure system availability during operational hours.
2. Maintain high data integrity for inventory records.
3. Provide secure access for authorized staff only.

Order Management (Fathima Thahzeen- 11485)

Functional Requirements:

1. Create, update, and delete customer orders.
2. Process payments securely.
3. Generate detailed bills for completed orders.

Non-Functional Requirements:

1. Ensure payment system reliability.
2. Provide responsive and fast system interactions.
3. Ensure compatibility with multiple payment gateways.

Profile Management (P. Arigaran-11121)

Functional Requirements:

1. Create, update, and delete customer profiles.
2. Manage loyalty program details.
3. Offer personalized recommendations to customers.

Non-Functional Requirements:

1. Ensure secure storage of personal data.
2. Provide fast retrieval of customer profiles.
3. Ensure scalability for a growing customer base.

Service Management (R.M.P.M.B. Rathnayaka-14627)

Functional Requirements:

1. Manage staff schedules and update availability.
2. Assign tables dynamically during peak hours.
3. Oversee day-to-day restaurant operations.

Non-Functional Requirements:

1. Maintain high system performance during peak hours.
2. Ensure seamless staff interaction with the system.
3. Provide real-time updates for table and staff statuses.

Feedback Management (M.N. Fathima Nusra - 11554)

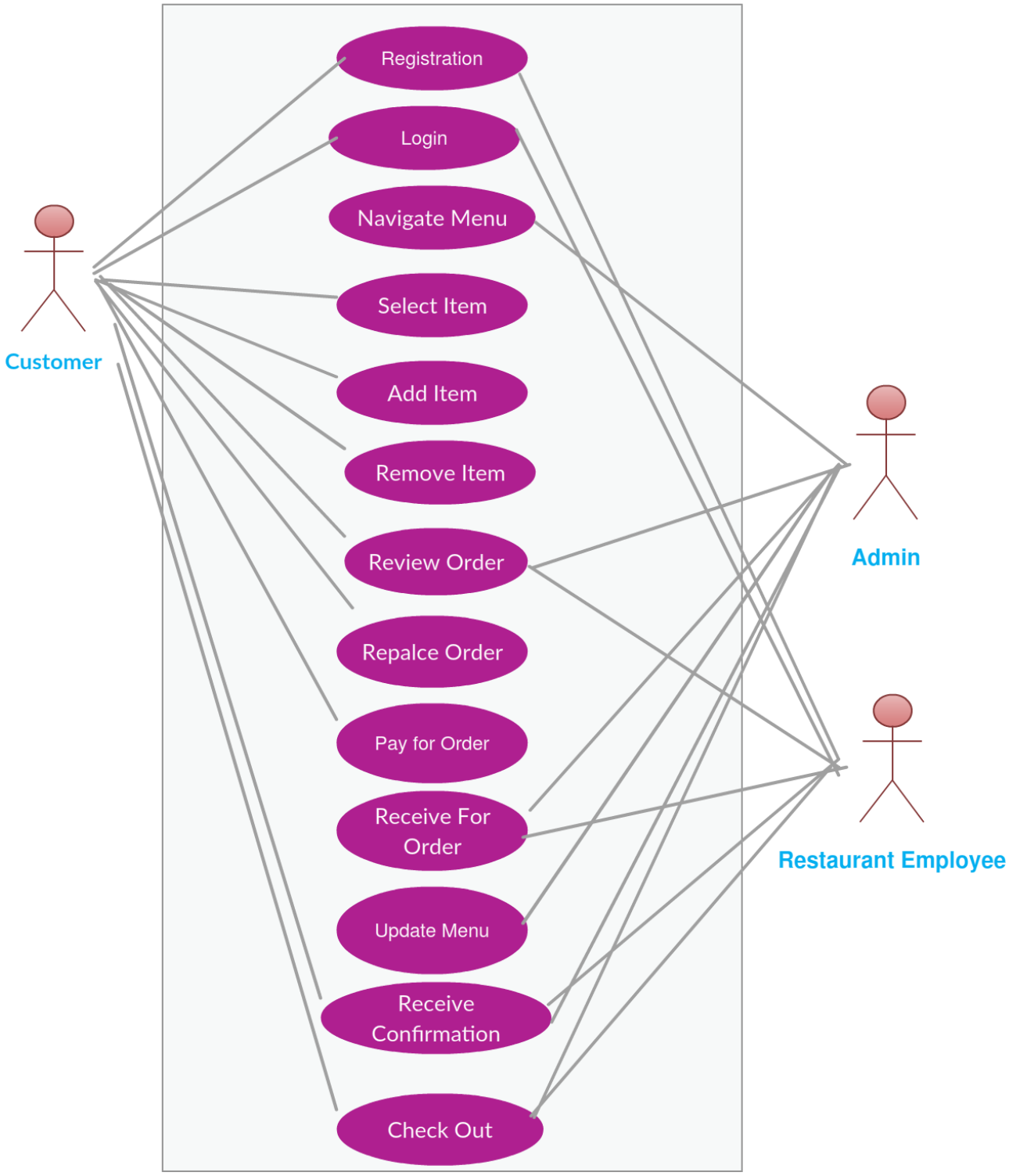
Functional Requirements:

1. Collect, update, and delete customer feedback.
2. Analyze feedback to identify trends.
3. Generate actionable insights for improvement.

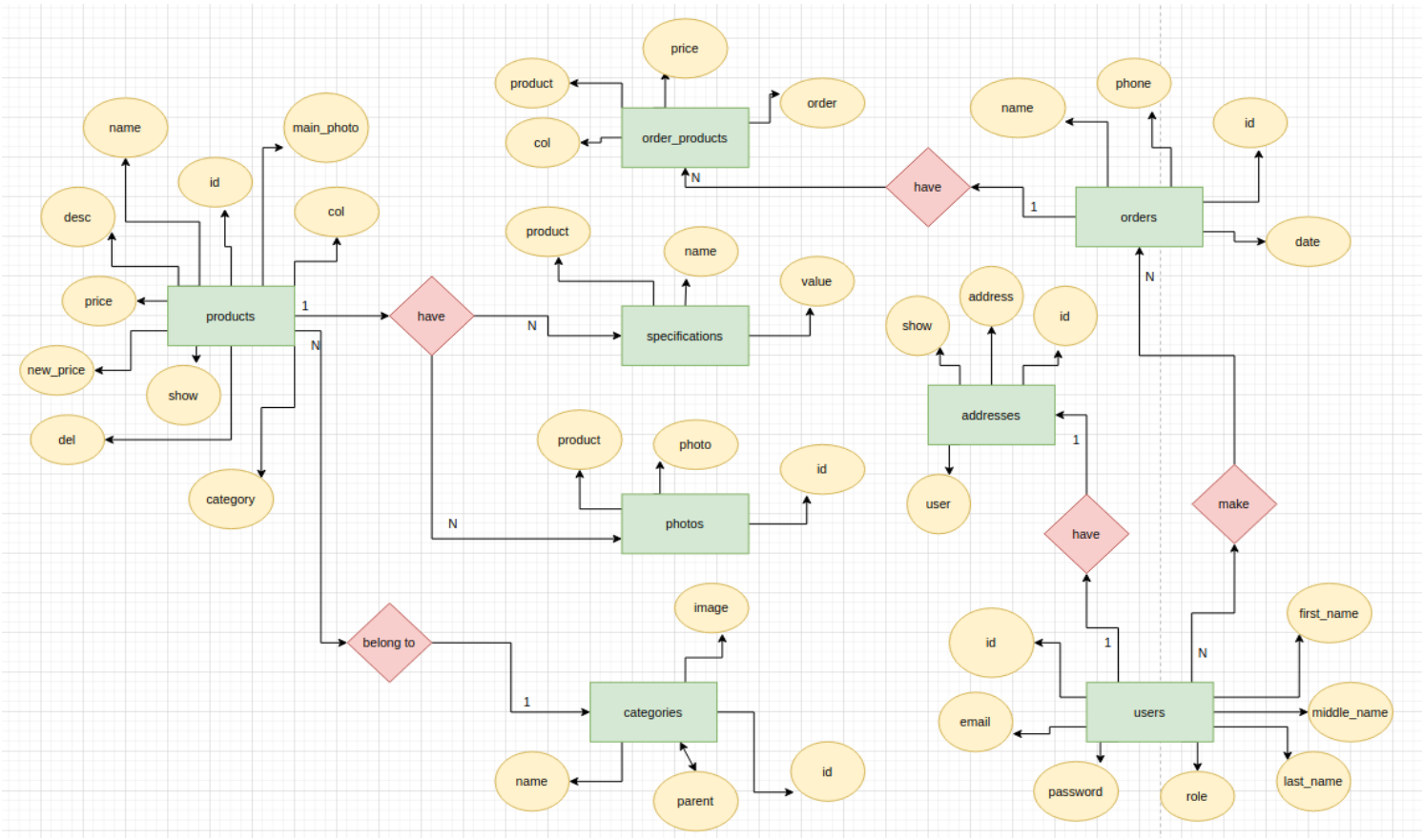
Non-Functional Requirements:

1. Ensure secure and anonymous feedback collection.
2. Provide insightful visual reports.
3. Maintain high uptime for feedback submission.

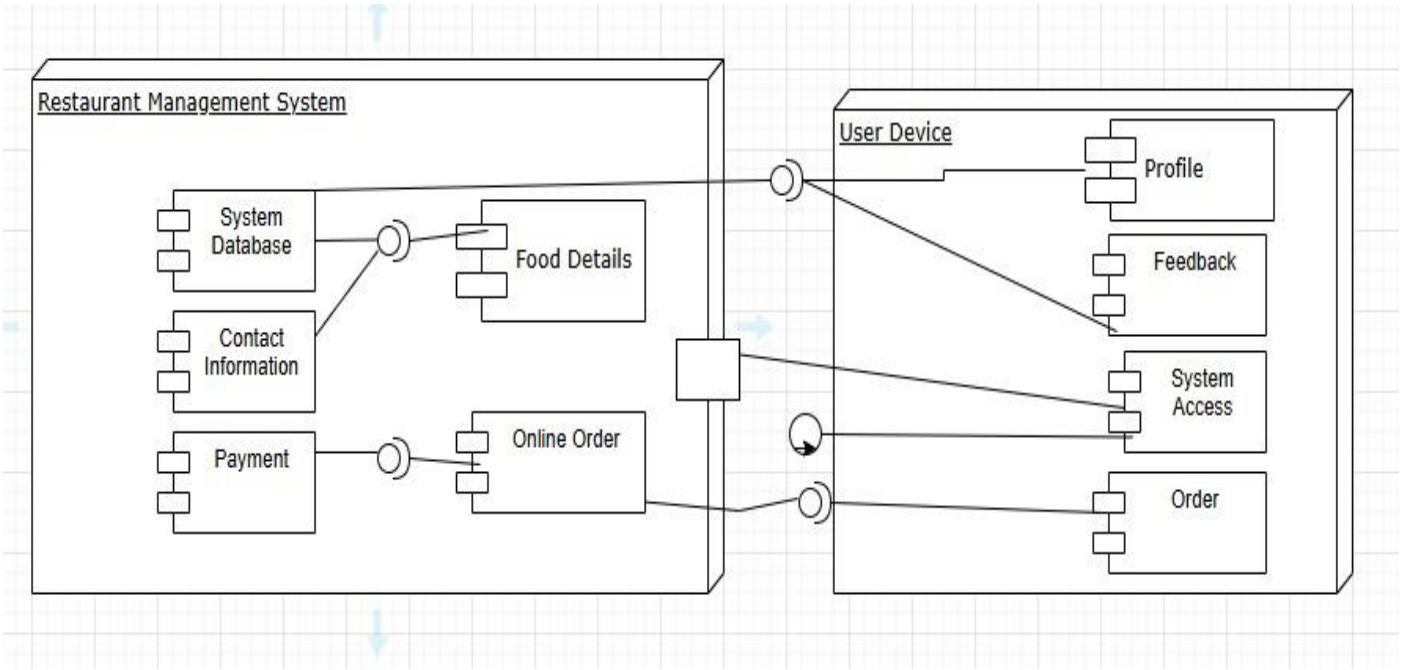
2. Use Case Diagram for the Entire System



3. Entity Diagram for the Entire System



4. High-Level Component Diagram for the Entire System



5. Architectural Pattern Explanation

The **Model-View-Controller (MVC)** pattern is a widely used architectural design pattern in software development. It separates the application logic into three interconnected components: **Model**, **View**, and **Controller**. This separation makes the system easier to manage, scale, and maintain.

Components of MVC in Your System

1. **Model (Data Layer)**
    - **Purpose**: Represents and manages the data and business logic of the system.
    - **Examples in Your System**:
        - Inventory: Tracks available items, stock levels, and updates when items are sold.
        - Orders: Stores details about customer orders, such as status (pending, completed), item lists, and totals.
        - Profiles: Maintains customer and staff information, such as names, roles, and permissions.
    - **Key Characteristics**:
        - Handles database operations (CRUD: Create, Read, Update, Delete).
        - Does not know anything about how data is displayed (View) or controlled (Controller).
2. **View (Presentation Layer)**
    - **Purpose**: Manages everything the user sees and interacts with.
    - **Examples in Your System**:
        - Customer Interface: Displays menus, order statuses, and payment options.
        - Staff Interface: Shows tools for managing inventory, processing orders, and user profiles.
    - **Key Characteristics**:
        - Retrieves data from the **Model** via the **Controller**.

        - Responsible for rendering the user interface (UI), including HTML, CSS, and JavaScript in web-based systems.
        - Provides feedback to the user (e.g., confirmation messages, error notifications).

3. **Controller (Logic Layer)**
    - **Purpose**: Acts as the intermediary between the **Model** and **View**.
    - **Examples in Your System**:
        - When a customer places an order, the **Controller** validates input, updates the **Model** (order details), and informs the **View** to display a confirmation.
        - When staff update inventory, the **Controller** processes the input, updates the **Model** (inventory data), and refreshes the **View** to reflect the changes.
    - **Key Characteristics**:
        - Handles user inputs and converts them into operations on the **Model**.
        - Contains application logic, such as validation, processing, and decision-making.

Benefits of Using MVC in Your System

1. **Separation of Concerns**:
    - Each component has a distinct responsibility:
        - **Model**: Data and business logic.
        - **View**: UI and user experience.
        - **Controller**: Application logic and data flow.
2. **Scalability**:
    - Adding new features becomes easier. For example:
        - A new payment gateway requires changes only in the **Controller** and minor updates in the **View**. The **Model** remains unaffected.

3. **Maintainability**:
    - Code is more organized and easier to debug:
        - Bugs in data retrieval? Check the **Model**.
        - Issues with UI? Focus on the **View**.
        - Incorrect logic flow? Investigate the **Controller**.
4. **Reusability**:
    - Components can be reused across the application:
        - The **Model** for managing orders can be shared between customer and staff interfaces.
5. **Testability**:
    - Each component can be tested independently, making it easier to write unit tests and integration tests.

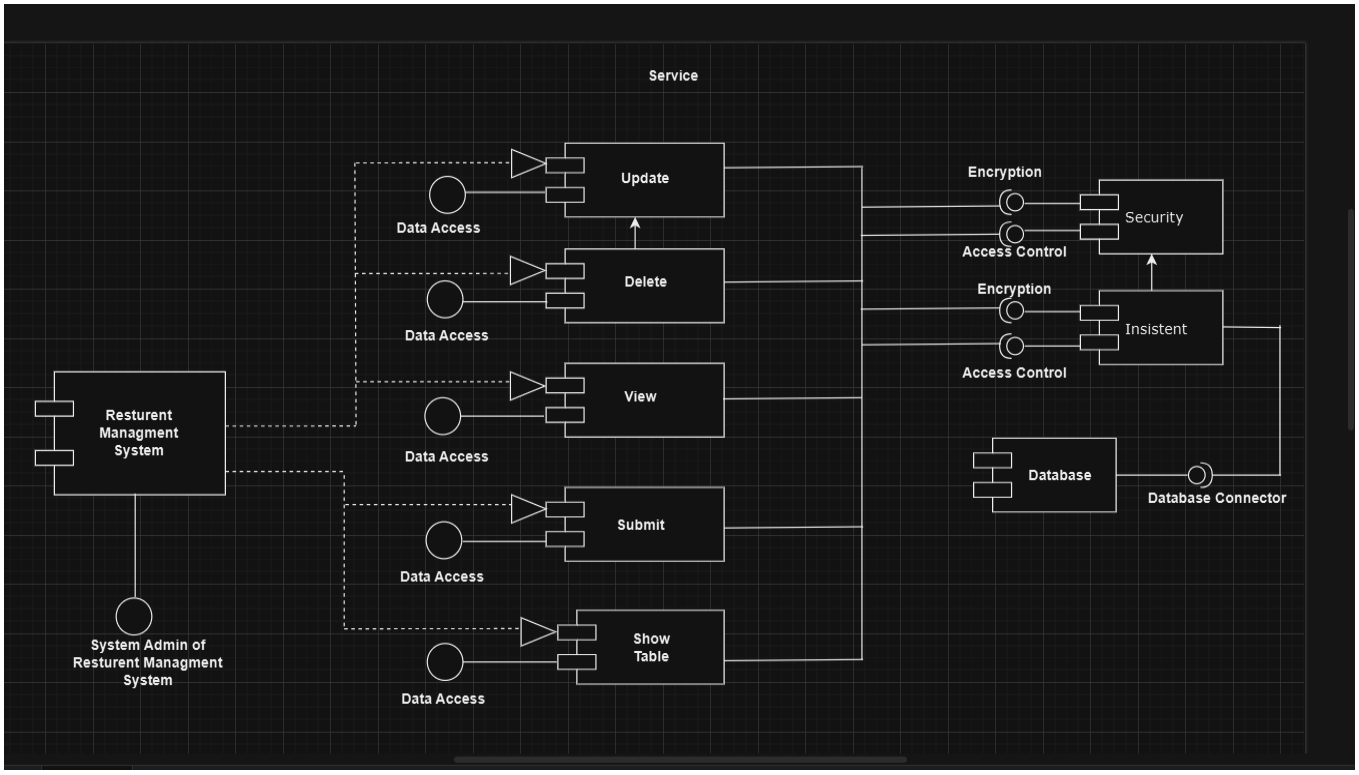
Example Flow in Your System

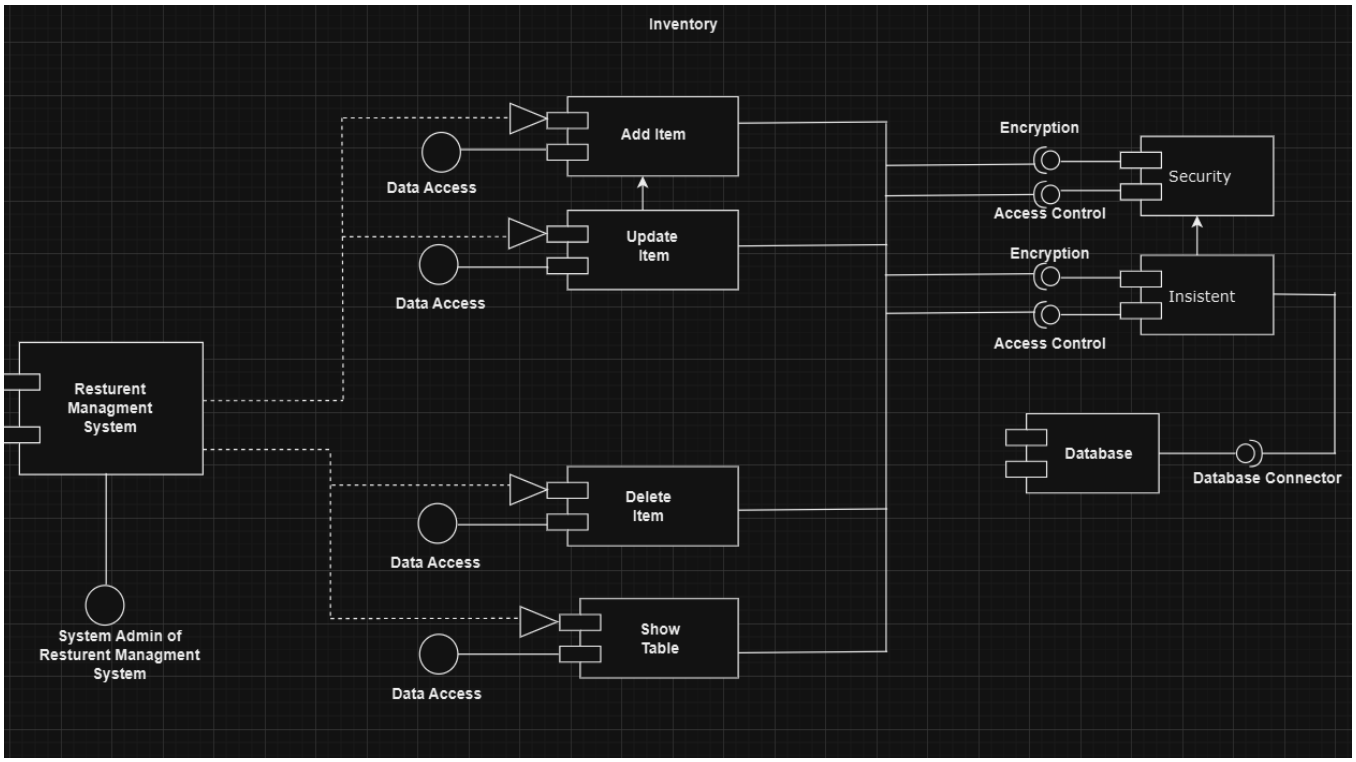
Let's look at how the MVC components interact during a typical operation:

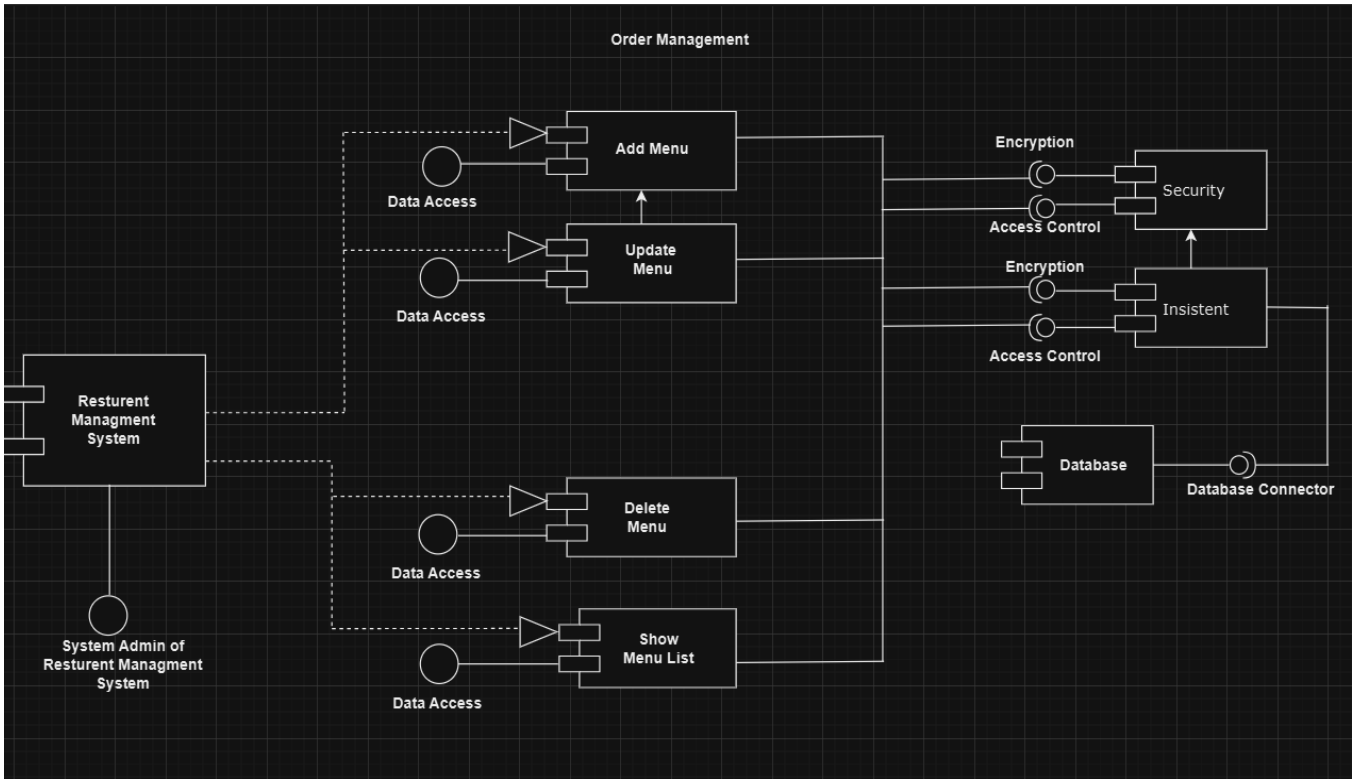
Scenario: A customer places an order.

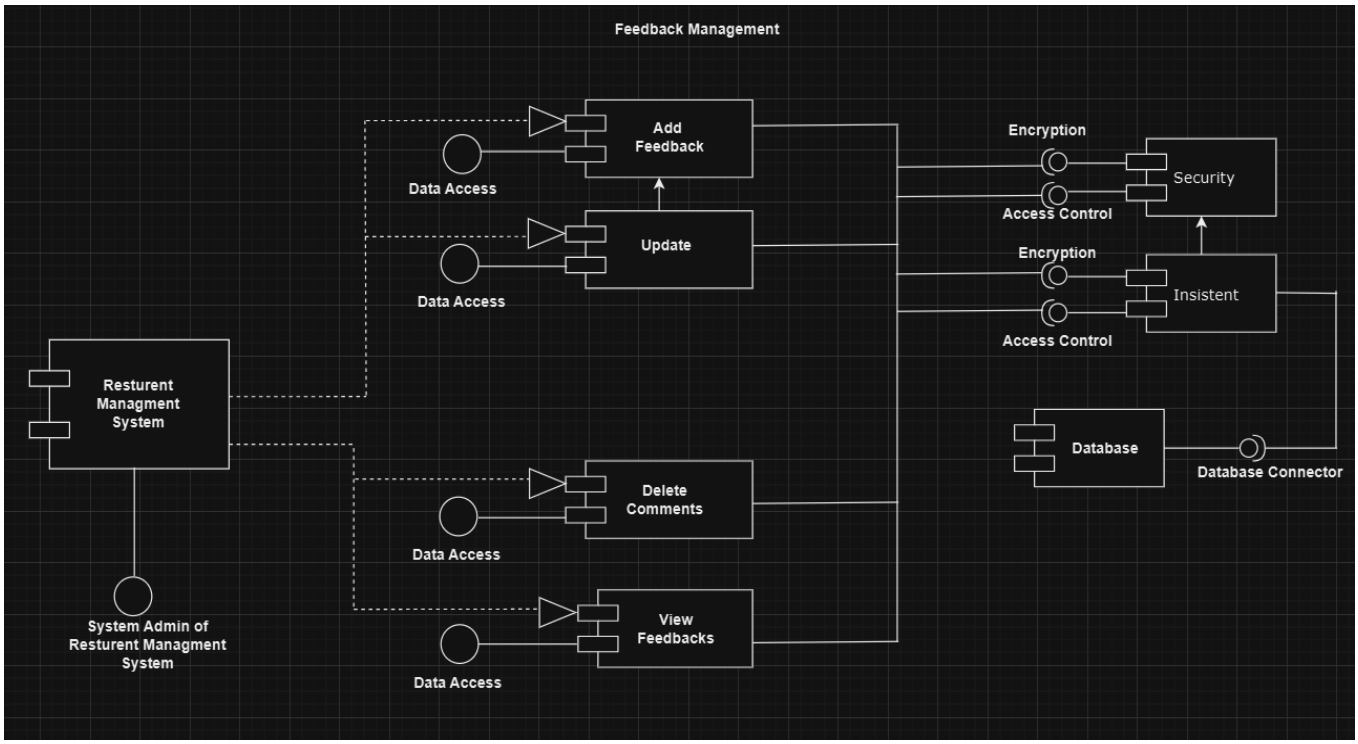
1. **View**:
    - The customer uses the UI to select menu items and clicks "Place Order."
2. **Controller**:
    - Captures the user input, validates it, and sends the order details to the **Model**.
3. **Model**:
    - Updates the database with the new order details and adjusts inventory accordingly.
4. **Controller**:
    - Retrieves the updated inventory data from the **Model** and instructs the **View** to update the UI.
5. **View**:
    - Displays a confirmation message and the updated menu with adjusted stock levels.

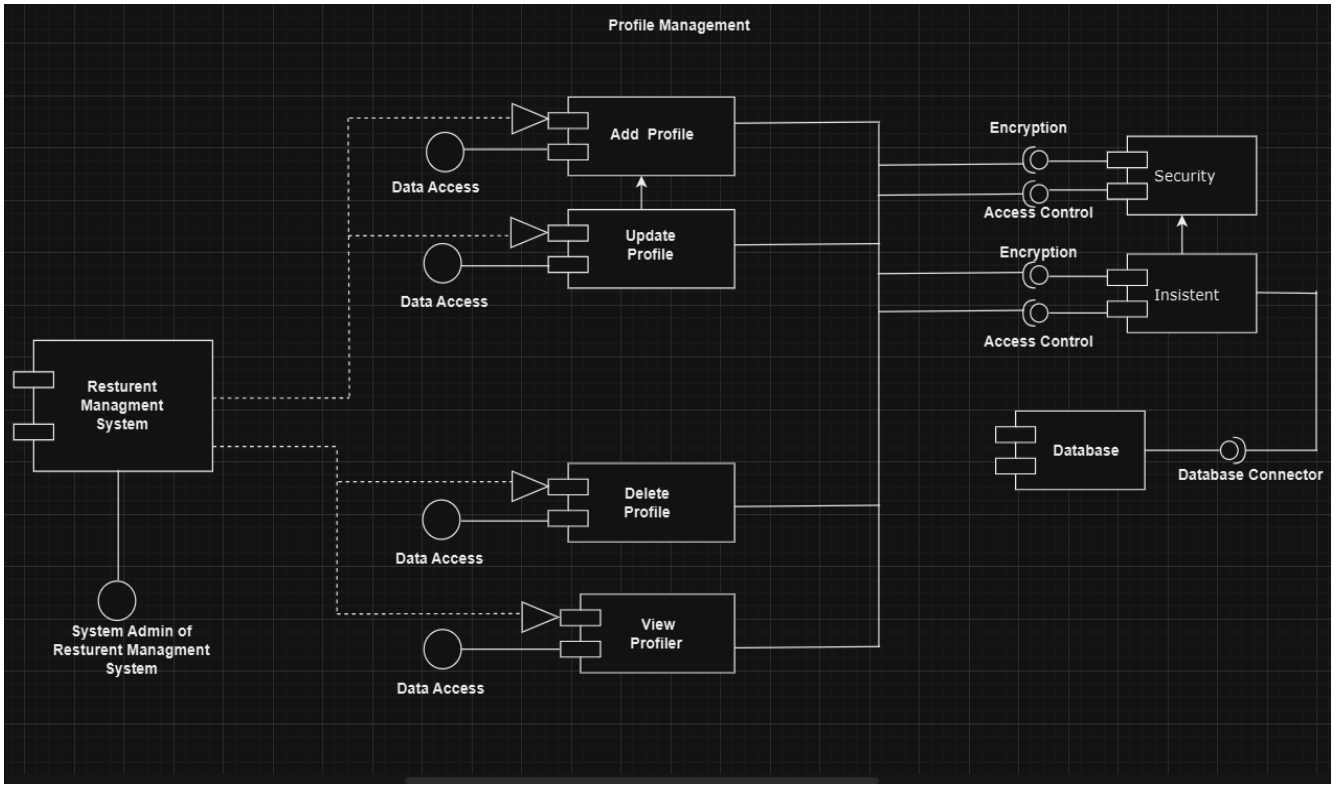
6. Component Diagram for Assigned Component



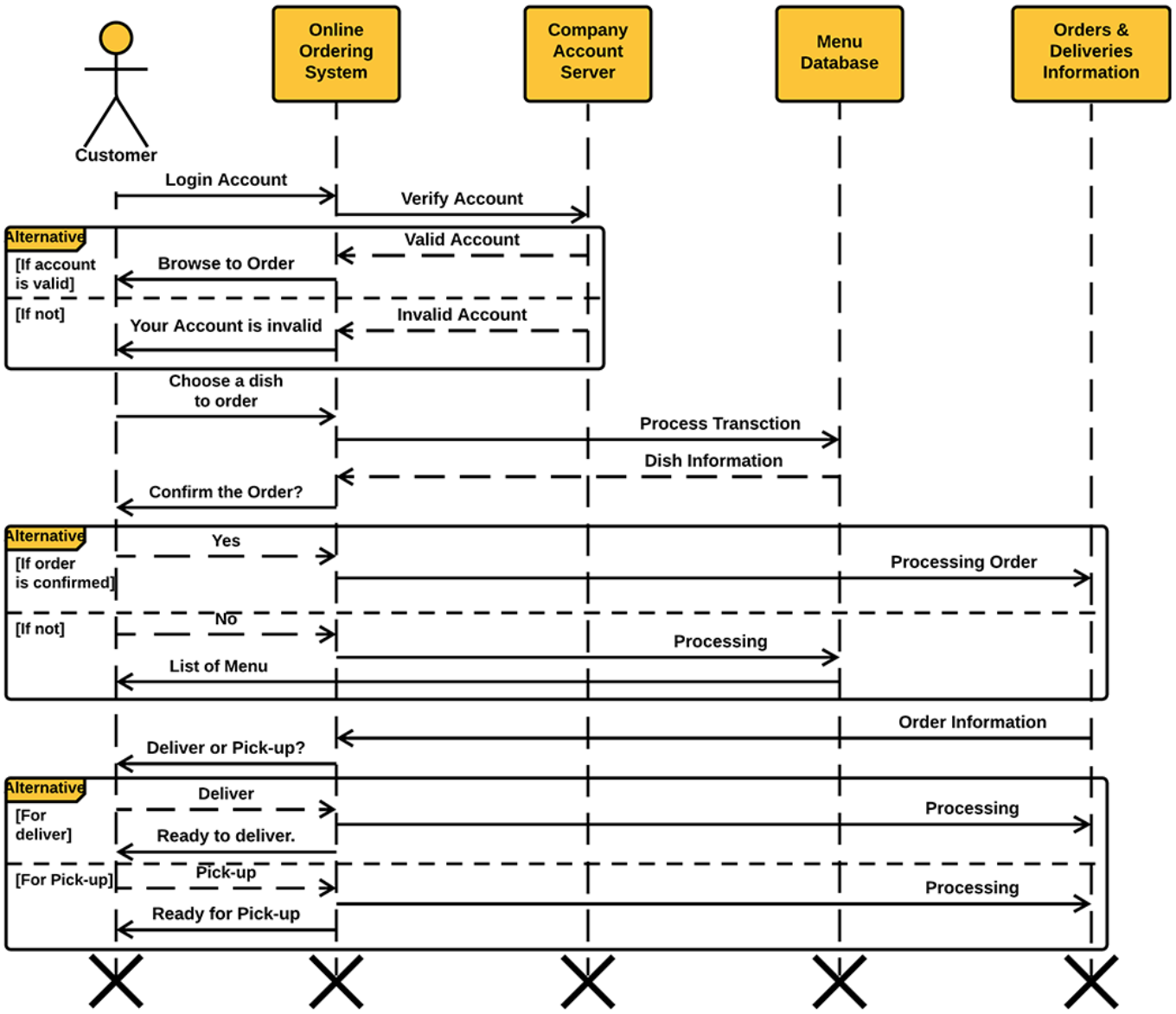



Order Management
Add Menu
Data Access
Update Menu
Data Access
Delete Menu
Data Access
Show Menu List
Data Access
Resturent Managment System
System Admin of Resturent Managment System
Encryption
Security
Access Control
Encryption
Insistent
Access Control
Database
Database Connector


Feedback Management
Add Feedback
Data Access
Update
Data Access
Delete Comments
Data Access
View Feedbacks
Data Access
Resturent Managment System
System Admin of Resturent Managment System
Encryption
Security
Access Control
Encryption
Insistent
Access Control
Database
Database Connector


Profile Management
Add Profile
Data Access
Update Profile
Data Access
Delete Profile
Data Access
View Profiler
Data Access
Resturent Managment System
System Admin of Resturent Managment System
Encryption
Security
Access Control
Encryption
Insistent
Access Control
Database
Database Connector

7. Sequence Diagram for Assigned Component



8. Design Pattern Utilization

Each member will:

1. **Identify the design pattern used:** For example, Feedback Management uses the **Template Pattern** to define steps for feedback processing while allowing flexibility for analysis.
2. **Explain the choice:** Template Pattern was chosen to standardize the workflow while allowing customizable analytics.

9. Quality Attributes

Quality attributes describe non-functional requirements that a system should meet to deliver a high standard of performance and user experience. Here are the attributes mentioned, explained in detail:

1. **Availability**
    - **What It Means**: The system is designed to minimize downtime and ensure that it is accessible to users whenever needed.
    - **How It's Ensured**:
        - Redundant database connections: If one database connection fails, another is available to ensure continuous service.
        - Load balancing: Helps distribute traffic to avoid overload on a single server.
2. **Interoperability**
    - **What It Means**: The ability of the system to communicate with other systems seamlessly.
    - **How It's Ensured**:
        - REST APIs are used as a standardized way to exchange data. This makes integration with third-party systems and services efficient.
3. **Modifiability**
    - **What It Means**: The system is built in a way that it can be easily updated or changed without major restructuring.
    - **How It's Ensured**:
        - Modular design: The system is divided into self-contained modules. For example, if you update the payment gateway module, it doesn't affect the rest of the system.
4. **Performance**
    - **What It Means**: The system handles requests efficiently to provide fast responses to users.
    - **How It's Ensured**:
        - Optimized database queries reduce execution time and prevent delays.
        - Use of caching mechanisms for frequently accessed data.

5. **Security**
    - **What It Means**: The system safeguards user data and prevents unauthorized access.
    - **How It's Ensured**:
        - Secure authentication methods, such as OAuth2 or multi-factor authentication, to validate user identities.
        - Encrypted storage for sensitive data, like passwords and personal information.

10. SonarQube Test Coverage Report

11. GitHub Repository Link

https://github.com/zapped767/Reserve.git

12. Youtubelink

11212 https://youtu.be/giIGyClEwoE?si=l3afLZ6934cy-PQc

11485 https://www.youtube.com/watch?v=h9dzOZ3wdmU

11554 (47) 25 November 2024 - YouTube

14627 https://youtu.be/n9coaucHEQA

11121 https://youtu.be/DffvVykUnn8?si=5_P09aelrhhmssMD