RAMAIAH INSTITUTE OF TECHNOLOGY, BANGALORE – 560054

(Autonomous Institute, Affiliated to VTU)

Department of Computer Science & Engineering

CS51: Software Engineering and Modelling

Report On
Cafe Management System

Rifah Balquees	1MS22CS114
Shrinidhi Pawar	1MS22CS141

Under the Guidance

Mrs. Shilpa Chaudhari
Assistant Professor

Ramaiah Institute of Technology

(Autonomous Institute, Affiliated to VTU)

MSR Nagar, MSRIT Post, Bangalore-560054

RAMAIAH INSTITUTE OF TECHNOLOGY, BANGALORE – 560054
(Autonomous Institute, Affiliated to VTU)

Department of Computer Science & Engineering

Evaluation Report

Team Member Details		
Sl No	**USN**	**NAME**
1.	1MS22CS114	Rifah Balquees
2.	1MS22CS141	Shrinidhi Pawar

SL No.	Component	Maximum Marks	Marks Obtained
1	Requirement Engineering & Use Case Model	10	
2	High and Low Level design	10	
	Total Marks	20	

Signature of the Student **Signature of the Faculty**

TABLE OF CONTENTS

Sl No.	Title	Page No
1	Abstract	1
2	Introduction	2-3
2	IEEE SRS Specification	4-8
3	System Model-Use Case Diagram	9-12
4	Preview Section	13
5	Class diagram(High level design)	14-15
6	State diagram(Low level design)	16-23
7	Sequence Diagram	24-25

Abstract

Running a cafe involves managing various operations such as taking customer orders, tracking order status, billing, and generating reports. Manual handling of these tasks often leads to inefficiencies, delays, and errors, negatively impacting customer satisfaction and operational efficiency. The **Cafe Management System** addresses these challenges by providing a software solution that automates and streamlines core processes.

This report outlines the Software Requirements Specification (SRS) and the system's architectural design, including both High-Level Design (HLD) and Low-Level Design (LLD). The SRS defines the functional and non-functional requirements of the system, offering a clear understanding of what the software aims to achieve. The HLD provides an overview of the system architecture, illustrating the interaction between modules such as Menu, Order Manager, Billing, and Report Manager. The LLD details the internal workings of these components, including class diagrams, database schemas, and method-level implementation specifics.

By addressing the problem of inefficient manual operations in cafes, this report presents a robust, scalable, and user-friendly solution. The Cafe Management System aims to enhance customer experience, optimize order processing, and provide accurate financial insights, making it a valuable tool for modern cafe management.

Introduction to Problem Statement

In today’s fast-paced world, managing cafe operations efficiently is a significant challenge. Many cafes still rely on manual systems for order-taking, billing, and record-keeping. These outdated methods are time-consuming, error-prone, and inefficient, resulting in delays and inaccuracies in order processing, billing, and inventory management. This leads to customer dissatisfaction and operational inefficiencies.

The proposed Cafe Management System (CMS) aims to automate key operations such as order management, billing, and reporting. By replacing manual systems with a centralized digital platform, the CMS will enhance operational efficiency, reduce human errors, and improve customer satisfaction. The key benefits include:

- **Order Management**: Digital order entry minimizes errors and enhances efficiency.
- **Billing Accuracy**: Automated billing ensures accurate tax calculations, discounts, and totals.
- **Performance Monitoring**: Real-time insights into sales, customer preferences, and inventory allow managers to make informed decisions.

The CMS will streamline cafe operations, ensuring both seamless customer experiences and efficient backend management.

Process Model

The **Agile Model** is selected for the development of the Cafe Management System due to its iterative, flexible, and collaborative nature. Agile is ideal for projects where requirements evolve and stakeholder feedback is critical.

In Agile, development is divided into short cycles called **sprints**, each lasting 2-4 weeks. After every sprint, the CMS is reviewed by stakeholders for feedback. This allows for continuous refinement based on real-world usage, ensuring the system meets the dynamic needs of the cafe.

Stages of Agile Development for CMS:

- **Requirement Gathering and Planning (Sprint 0)**: The CMS's core features are defined in collaboration with cafe managers and staff. Initial user stories are created.
- **Sprint Development**: Features like order management, automated billing, and reporting are developed incrementally, prioritizing essential functionalities.
- **Testing**: Each sprint includes functional and usability testing to ensure the system works as intended.
- **Feedback and Refinement**: Ongoing feedback helps refine the system and prioritize further features.

- **Deployment and Maintenance**: After deployment, the system can be updated or enhanced in future sprints based on feedback.

Why Agile is Suitable

Agile is ideal for the CMS due to its flexibility. In the dynamic cafe environment, operational needs and customer preferences can change rapidly. Agile accommodates these evolving requirements, ensuring that the system can be adjusted as needed.

Agile's iterative nature allows for the early delivery of functional parts of the CMS. For example, features like order management and billing can be deployed early, offering immediate benefits to the cafe. Additionally, continuous feedback and testing ensure the system meets the real-world needs of cafe operations.

Advantages of Agile for CMS:

- **Flexibility**: Agile adapts to changing requirements, which is essential in a fast-paced environment like a cafe.
- **Customer Collaboration**: Regular communication with stakeholders ensures that the system aligns with their needs.
- **Faster Delivery**: Functional parts of the system can be used early, improving daily operations.
- **Continuous Improvement**: Features can be enhanced and refined based on feedback.
- **Early Problem Detection**: Frequent testing identifies issues early, ensuring a higher-quality product.
- **Transparency**: Regular updates ensure that stakeholders have clear visibility into the project's progress.

Adopting Agile will enable the development of a flexible, responsive, and collaborative Cafe Management System, ensuring it meets evolving needs and improves cafe operations and customer satisfaction.

1.Cafe Management System Requirement Document

1.1. Preface

This document is intended for developers, project managers, and stakeholders involved in creating and deploying the Cafe Management System. It outlines the system's requirements and provides a blueprint for its design and implementation.

The document includes a version history, describing the changes made in each revision to track the evolution of the requirements.

1.2. Introduction

1.2.1 Purpose

The purpose of this document is to define the functional and non-functional requirements for the Cafe Management System. This system aims to enhance operational efficiency by automating order management, billing, and employee monitoring.

1.2.2 Scope

The Cafe Management System will:

- Streamline cafe operations through a user-friendly POS interface.
- Provide a secure platform for managing customer payments and employee records.
- Enhance the overall customer experience and reduce operational overheads.

1.2.3 Overview

This document includes the system's requirements, architecture, and use case models, detailing how the Cafe Management System will operate. It defines the services provided, system constraints, and future enhancements.

1.3. Glossary

Term	**Definition**
POS	Point of Sale system used for managing customer orders and transactions.
Admin	The cafe owner or manager responsible for overseeing operations.
Employee Module	A feature for tracking employee work schedules and performance metrics.

1.4. User Requirements Definition

1.4.1 Overview

The Cafe Management System is designed for the following user groups:

- **Customers**: Use the system to view menus, place orders, and make payments.
- **Staff**: Operate the POS to manage orders and generate bills.
- **Admin**: Oversee operations, manage employees, and track financial performance.

1.4.2 Key User Requirements

- **Customers should be able to**:
  - View the menu and customize orders.
  - Receive an order confirmation and digital receipt.
- **Staff should be able to**:
  - Mark orders as completed and update order statuses.
  - Generate customer bills efficiently.
- **Admins should be able to**:
  - View detailed sales and performance reports.
  - Manage employee records, including attendance and performance.

1.5. System Requirements Specification

1.5.1 Functional Requirements

Order Management

The Order Management feature allows customers to place orders through a user-friendly POS system with options for customization, such as add-ons or special instructions, and automatically calculates the total. Staff can track these orders in real-time, update their status (pending, in progress, completed), and manage order flow efficiently, ensuring a smooth process even during peak times.

Generate Bill

Generate Bill generates an itemized bill that accurately lists the ordered items, their quantities, prices, applicable taxes, and any relevant discounts, ensuring transparent and efficient billing.

View Report

View Report allows managers to generate and view reports related to sales, inventory, and staff performance. This functionality helps admins analyze business trends, make informed decisions, and ensure smooth operations.

Relationship Between Functional Requirements

The Order Management feature seamlessly integrates with the Generate Bill and View Report functionalities to enhance the overall operational efficiency of the system. When customers place orders through the user-friendly POS system, they can customize their orders with add-ons or special instructions, with the total automatically calculated. As orders progress, staff can track their status in real-time, ensuring smooth order flow, even during peak times. Once an order is completed, the Generate Bill feature automatically produces an itemized bill, accurately listing the ordered items, quantities, prices, taxes, and discounts. This ensures transparency and helps customers understand their total charges. Additionally, the View Report functionality allows managers and admins to generate detailed reports on sales, inventory, and staff performance, providing insights into business trends and enabling informed decision-making. Together, these features create a cohesive system that streamlines both customer ordering and internal operations, supporting effective business management.

1.5.2 Non-Functional Requirements

Performance:

- **Concurrent Users Handling**: The system must be designed to handle at least 50 concurrent users without experiencing performance degradation. The system should scale horizontally if required to ensure that it remains fast and responsive even as traffic or user count increases. This is particularly important for ensuring smooth operations during peak hours when many customers place orders simultaneously.
- **Fast Transaction Processing**: The time taken to process customer orders, update statuses, generate bills, and complete payments must be minimized. A delay in these areas can negatively impact the customer experience, leading to dissatisfaction and increased waiting times.

Usability:

- **User-Friendly Interface**: The POS system interface should be simple, intuitive, and easy for non-technical staff to use. This is critical to minimize training time and reduce the chance of errors. Staff should be able to navigate the system quickly, from placing orders to updating statuses, without requiring technical expertise.
- **Responsive Design**: The system must be responsive and easily accessible on various devices, such as tablets, smartphones, and desktop computers. This ensures that staff can operate the POS system seamlessly, whether they are working from a central station or moving around the café.

Security:

- **Data Encryption**: The system must implement robust encryption mechanisms to protect sensitive customer and payment data, both during transaction processing and when stored in databases. Payment details, personal customer information, and staff records must be protected by end-to-end encryption and other security protocols to avoid any data breaches.
- **POS System Security**: The system should be secured against unauthorized access by implementing role-based authentication. Admins, staff, and customers must each have distinct access levels to ensure that sensitive data and critical operations are only accessible to authorized personnel.
- **Compliance**: The system must comply with global security standards, such as PCI DSS for payment security and GDPR for customer data protection, to ensure that it meets legal requirements and industry best practices.

Scalability:

- **System Expansion**: The system must be designed with scalability in mind, allowing future upgrades and expansions with minimal disruption to ongoing operations. This includes supporting the addition of new functionalities, such as new payment gateways, employee management features, or inventory management tools.
- **Cloud Integration**: For long-term scalability, the system should be designed with cloud compatibility. This will allow the system to scale horizontally, handle more concurrent users, and expand without requiring costly and complex hardware upgrades.

1.6. Requirements Addressing Key Challenges

The **Cafe Management System** is designed to address several key challenges that cafes typically face in their operations, such as order management inefficiencies, errors in billing, difficulties in tracking employee attendance, and managing menu and employee data manually. These challenges can lead to operational bottlenecks, poor customer experiences, and financial discrepancies. The system's requirements are specifically designed to solve these challenges by automating workflows, enhancing transparency, and providing real-time insights into various aspects of cafe management.

Order Management:

- **Problem Addressed**: Manual tracking of orders often leads to delays and errors in fulfilling customer orders, especially during peak hours when staff are overwhelmed.
- **Requirement**: The POS system allows customers to place orders directly into the system, reducing the chance of manual input errors and ensuring that orders are processed promptly. The system also enables staff to track the status of each order in real-time, allowing for better coordination and quicker service.

Billing and Payment:

- **Problem Addressed**: Manual billing can often result in human errors, such as incorrect totals, failure to apply discounts, or mistakes in processing payments, which can lead to customer dissatisfaction and financial discrepancies.
- **Requirement**: The system generates automatic, itemized bills that are accurate and transparent, eliminating the possibility of errors in manual calculations. Integration with digital payment providers allows for seamless and secure payment processing, reducing the likelihood of payment fraud or errors in transaction handling.

Employee Management:

- **Problem Addressed**: Managing employee attendance, work schedules, and performance manually is often time-consuming, prone to mistakes, and difficult to track efficiently.
- **Requirement**: The system provides an automated method for managing employee schedules, attendance, and shift changes, improving administrative efficiency and reducing manual errors. Admins can also track staff performance based on system-generated reports, making it easier to evaluate staff efficiency and make data-driven decisions regarding staffing and training.

Admin Management:

- **Problem Addressed**: Manual management of tasks such as monitoring sales and managing employee records often results in inefficiencies and missed opportunities for operational improvement.
- **Requirement**: The admin role in the system allows managers to easily track employee performance, and generate financial reports. These features streamline the management process, enabling quick decision-making, identifying areas for improvement, and enhancing overall operational control.

Non-Functional Requirements:

- **Performance**: Ensuring the system can handle peak loads without slowdowns, even with 50+ concurrent users, guarantees a smooth user experience during busy periods.
- **Usability**: A user-friendly interface ensures that even non-technical staff can quickly learn how to use the system, reducing training time and potential operational errors.
- **Security**: Secure payment processing and data protection ensure that customer and business information is safeguarded, protecting the business from potential security breaches and legal issues.
- **Scalability**: The system is designed to grow with the business, allowing easy integration of new payment methods or employee management features as needed, ensuring long-term viability.

1.7. System Models

1.7.1 Use Case Diagram

The **Use Case Diagram** identifies key actors and interactions within the Cafe Management System. Below is a simplified version that uses standard UML notation for clarity.

- **Actors**:
  - **Customer**: Interacts with the POS to place orders and make payments.
  - **Staff**: Manages orders and generates bills.
  - **Admin**: Manages the system, including menu items and employee records.

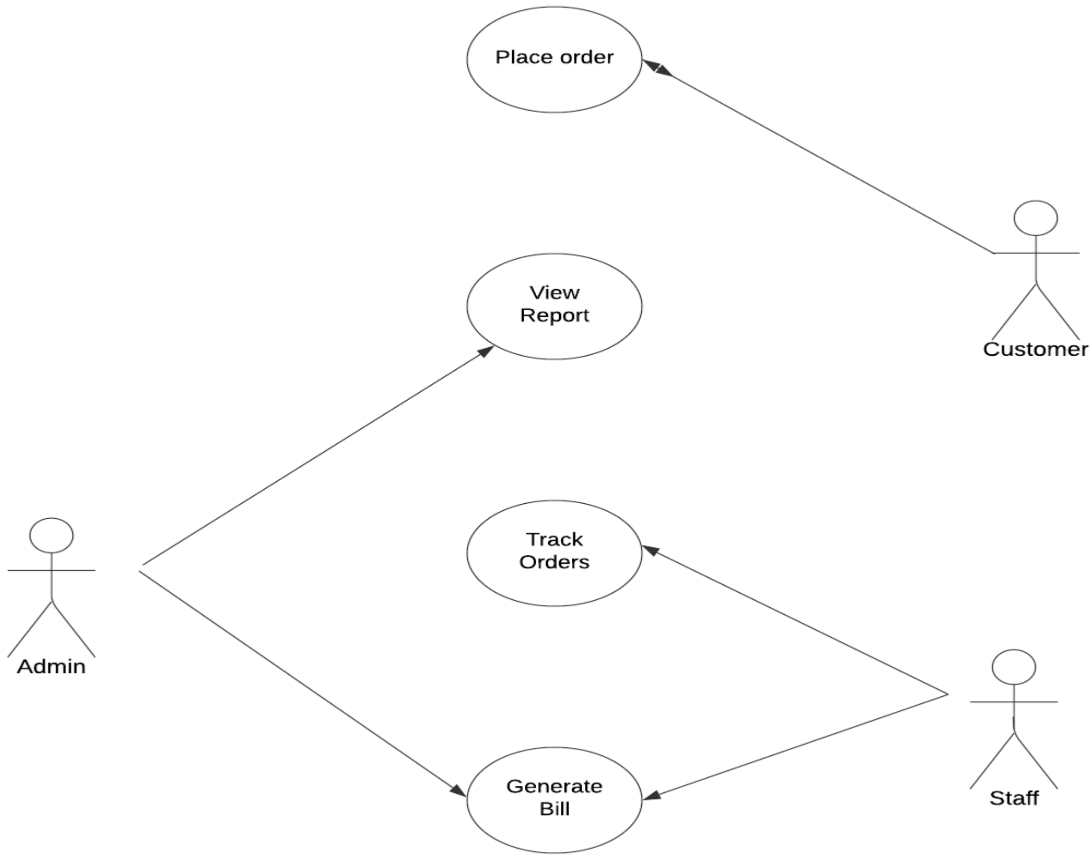

Fig 1.1 Use case diagram

1.7.2 Use Case Description for Place Order

Actors	Customer: The individual who places an order.
Description	The Place Order use case enables customers to browse the menu, select items, customize their order (e.g., add-ons, special requests), and submit the order for processing. The system displays the total price, including taxes and any discounts, before finalizing the order
Functional Requirements	Order Management,Generate Bill The Place Order use case triggers the Order Management functionality to process the customer's order and initiate the billing process once the order is confirmed
Data	Menu Items,Special Instructions,Customer Detail,Order Total
Stimulus	Customer selects items,submits order
Response	The system confirms the order and displays the order summary (items, total cost, payment options). A unique order ID is generated for tracking purposes.
Comments	The system should be designed for ease of navigation, especially for customers unfamiliar with the menu.

1.7.3 Use Case Description for View Report

Actors	Manager/Admin: The person responsible for managing the operations of the cafe and analyzing reports.
Description	The View Report use case allows managers to generate and view reports related to various operational aspects, such as sales, inventory, and staff performance. The manager can customize the reports based on time, categories, or specific data points.
Functional Requirements	View Report,Order Management The View Report use case interacts with the Order Management feature to extract order data for generating reports and insights, supporting informed decision-making.
Data	Sales Data,Inventory Data,Employee Performance Data

Stimulus	Manager requests a report,Manager selects filters
Response	The system generates and displays the requested report. The report can be exported or printed for further analysis.
Comments	The system should allow reports to be filtered by time (daily, weekly, monthly) and specific criteria (item categories, employee performance). Visual representations (charts/graphs) could enhance report readability

1.7.4 Use Case Description for Track Orders

Actors	Staff Member: The employee responsible for managing and fulfilling customer orders.
Description	The Track Orders use case allows staff to view the status of customer orders in real-time, update the order status (e.g., pending, in progress, completed), and send the orders to the kitchen or barista for preparation. Staff can notify customers once their orders are ready.
Functional Requirements	Order Management,Generate Bill The Track Orders use case interacts with the Order Management system to monitor order status and updates. Once an order reaches the "completed" status, it links to the Generate Bill functionality to finalize the transaction.
Data	Order ID ,Order Status,Customer Information,Order Details
Stimulus	Staff members receive a new order,Staff updates the order status,Order is completed and marked as ready for customer pickup.
Response	The system updates the order status and shows real-time information about the order’s progress. The customer is notified when the order is ready for pickup or delivery.
Comments	The system should provide notifications to staff members to ensure timely processing, especially during busy hours. Clear status indicators should be displayed to avoid confusion.

1.7.5 Use Case Description for Generate Bill

Actors	Staff Member: The employee responsible for generating and finalizing customer bills.
Description	The Generate Bill use case allows the staff to create an itemized bill for customers once their order is finalized. The bill includes all ordered items, quantities, prices, taxes, and any applicable discounts. The total amount due is calculated, and the bill is provided to the customer for payment.
Functional Requirements	Generate Bill,Order Management The Generate Bill use case is a continuation of the Order Management process. After the order is placed and processed, the bill is generated using the order details tracked by the Order Management system
Data	Order Details ,Taxes,Discounts ,Total Amount
Stimulus	Staff finalizes the order and requests a bill. The system retrieves the order details and applies any necessary calculations (taxes, discounts). The customer is ready for payment.
Response	The system generates a detailed, itemized bill and displays it for review. The bill is printed or sent digitally to the customer for payment.
Comments	The bill should be clear and readable, with no ambiguities about the prices or discounts.

Preview

Purpose and Structure

1. **Purpose:**
    - To detail the functional and non-functional requirements of the system.
    - To serve as a reference for developers, stakeholders, and testers throughout the development lifecycle.
2. **Structure:**
    - The document begins with a **Preface** and an **Introduction**, defining the scope and purpose of the system.
    - A **Glossary** is provided to clarify technical terms.
    - The **User Requirements Definition** highlights the key functionalities from a user's perspective.
    - Detailed **System Requirements** (functional and non-functional) are outlined in subsequent sections.
    - A **Use Case Diagram** visualizes the interactions within the system.
    - Finally, an **Index** organizes key terms, diagrams, and functions for ease of reference.

Chapter 2

2.1 Class diagram for Cafe Management System

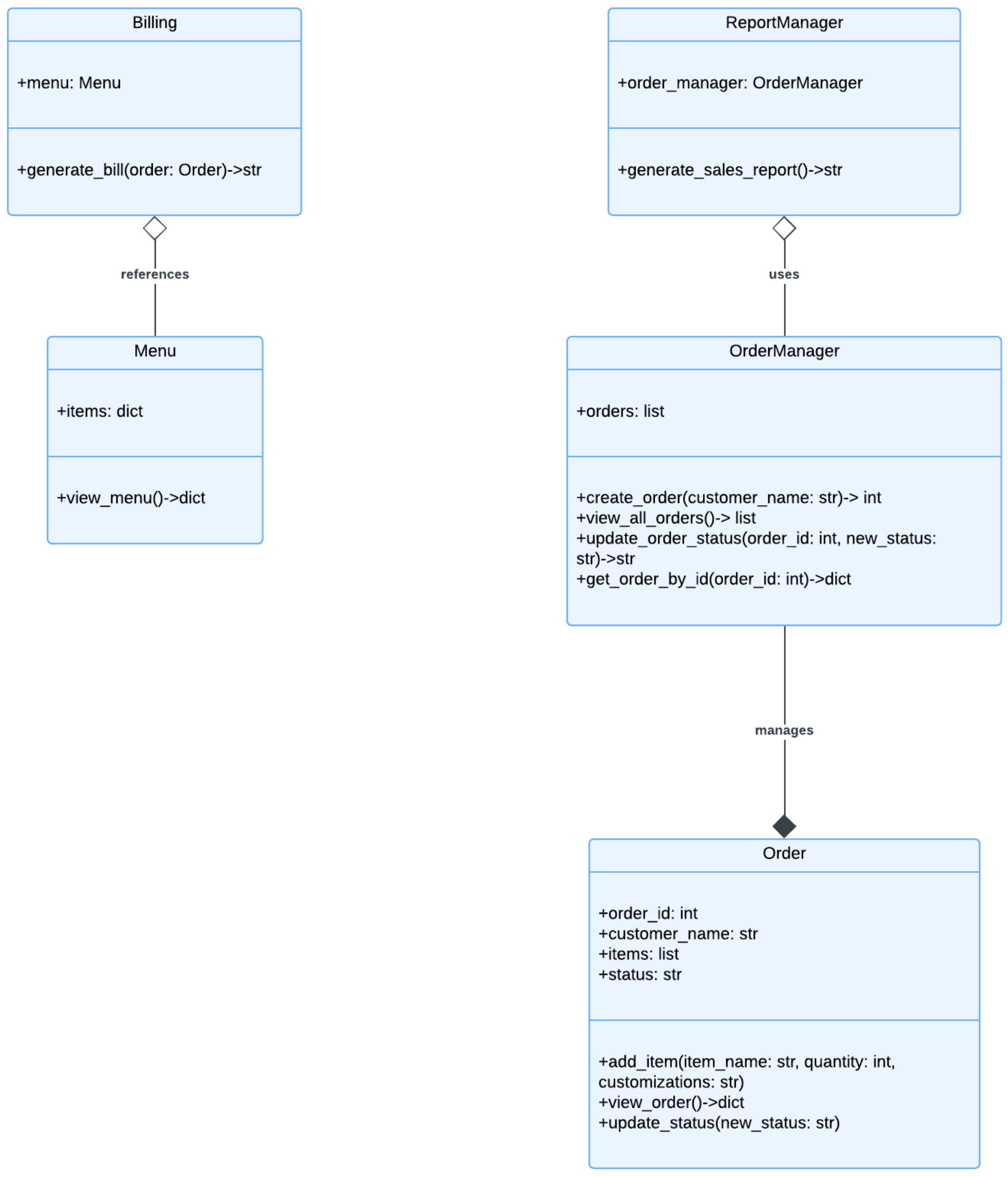
Billing
+menu: Menu
+generate_bill(order: Order)->str
references
Menu
+items: dict
+view_menu()->dict
ReportManager
+order_manager: OrderManager
+generate_sales_report()->str
uses
OrderManager
+orders: list
+create_order(customer_name: str)-> int
+view_all_orders()-> list
+update_order_status(order_id: int, new_status: str)->str
+get_order_by_id(order_id: int)->dict
manages
Order
+order_id: int
+customer_name: str
+items: list
+status: str
+add_item(item_name: str, quantity: int, customizations: str)
+view_order()->dict
+update_status(new_status: str)

Fig 2.1 Class diagram for Cafe Management System

State	**Description**
Billing	Manages the process of accessing the menu and generating a bill for an order.
Menu	Handles menu initialization, item loading, and displaying the menu.
ReportManager	Generates sales reports by interacting with the OrderManager.
OrderManager	Manages the creation, retrieval and updation of orders.
Order	Tracks order details, items, and status updates.

Stimulus	**Description**
Create Order	A new order is created with a customer_name. Returns an order ID
View all Orders	The user requests to view a list of all existing orders.
Update Order Status	The status of a specific order is updated (e.g., "Preparing," "Ready," or "Completed").
Get Order by ID	The user retrieves order details using a specific order_id.
Add Item	An item is added to an order, specifying the item name, quantity, and any customizations.
View Order	The user views the details of a specific order, such as items, quantities, and order status.
Generate Bill	A bill is generated for a specific order, including itemized details and total cost.
View Menu	The user requests to view the list of available menu items and their prices.
Generate Sales Report	A sales report is generated, summarizing orders and revenue managed by the OrderManger.

2.2 State Diagrams

2.2.1 Billing Class State Diagram

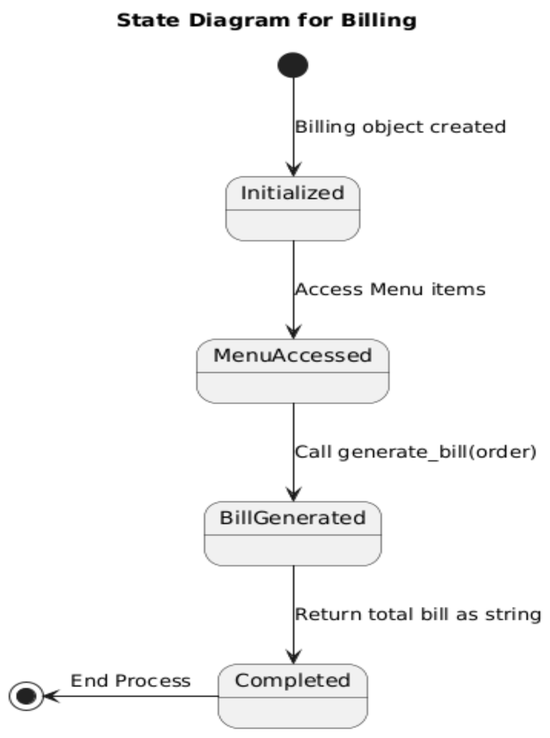

Fig 2.2 State diagram for Billing Class

The Billing class manages the process of generating a bill by interacting with the Menu class.

States:

- **Initialized**: The Billing object is created and ready for use.
- **MenuAccessed**: The Menu items are accessed to retrieve available options.
- **BillGenerated**: A bill is generated for the given Order object using the generate_bill() method.
- **Completed**: The bill is returned as a string, and the process ends.

Transitions:

- Transition from **Initialized** to **MenuAccessed** occurs when the Menu is accessed.
- Transition to **BillGenerated** happens when the generate_bill(order) method is called.
- The process ends at **Completed** after the bill is successfully generated and returned.

2.2.2 Menu State Diagram

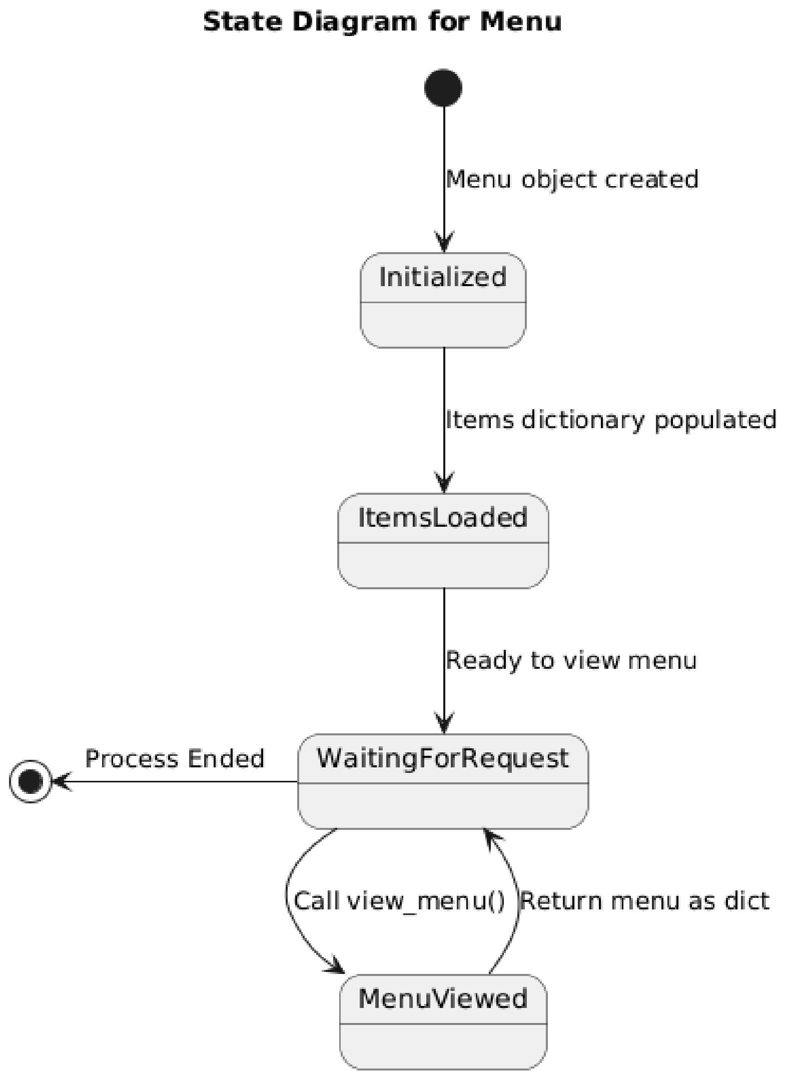

Fig 2.3 State diagram for Menu Class

The Menu class handles loading and displaying menu items.

States:

- **Initialized**: The Menu object is created.
- **ItemsLoaded**: The items dictionary is populated with available menu items.
- **WaitingForRequest**: The class is idle and ready to process a request.
- **MenuViewed**: The view_menu() method is invoked to retrieve the menu details.

Transitions:

- Transition from **Initialized** to **ItemsLoaded** occurs when menu items are added.
- Transition to **WaitingForRequest** indicates the class is ready to process requests.
- Transition to **MenuViewed** happens when the view_menu() method is called.
- The process loops back to **WaitingForRequest** after displaying the menu.

2.2.3 ReportManager Class State Diagram

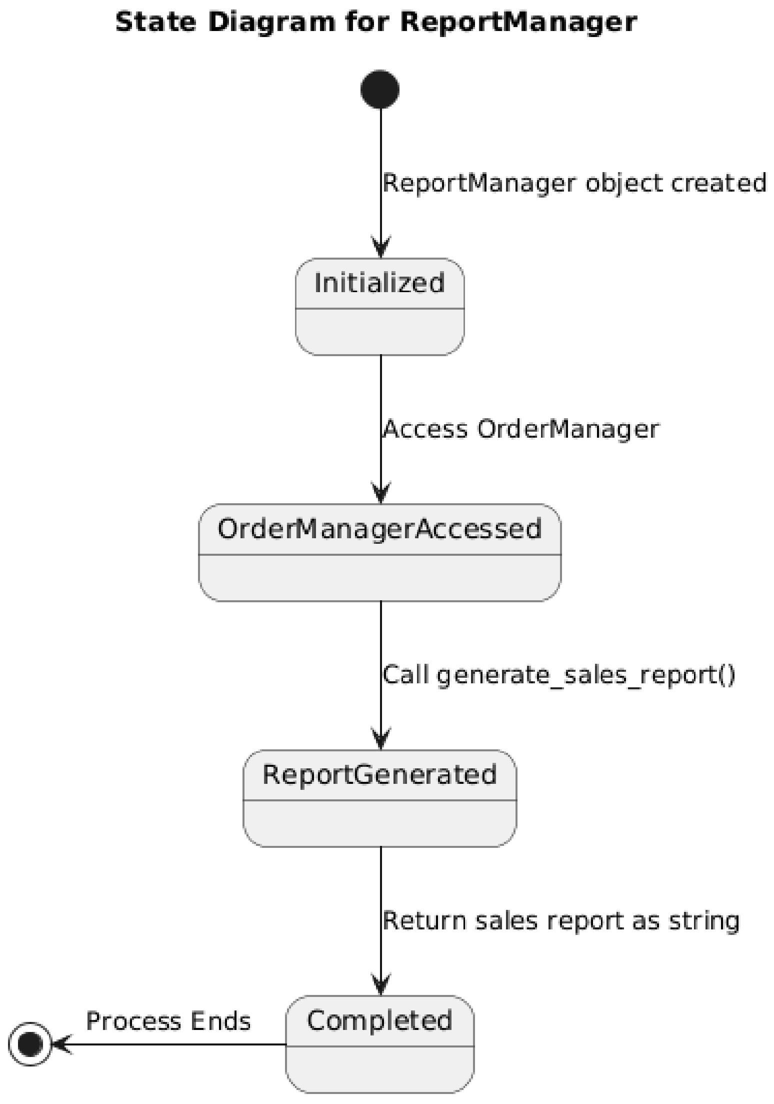
State Diagram for ReportManager
ReportManager object created
Initialized
Access OrderManager
OrderManagerAccessed
Call generate_sales_report()
ReportGenerated
Return sales report as string
Completed
Process Ends

Fig 2.4 State diagram for ReportManager Class

The ReportManager class generates a sales report by interacting with the OrderManager.

States:

- **Initialized**: The ReportManager object is created.
- **OrderManagerAccessed**: The OrderManager is accessed to retrieve order details.
- **ReportGenerated**: The generate_sales_report() method generates a consolidated sales report.
- **Completed**: The report is returned as a string, and the process ends.

Transitions:

- Transition from **Initialized** to **OrderManagerAccessed** happens when the OrderManager is used.
- Transition to **ReportGenerated** occurs when generate_sales_report() is called.
- The process ends at **Completed** after generating and returning the report.

2.2.4 OrderManager Class State Diagram

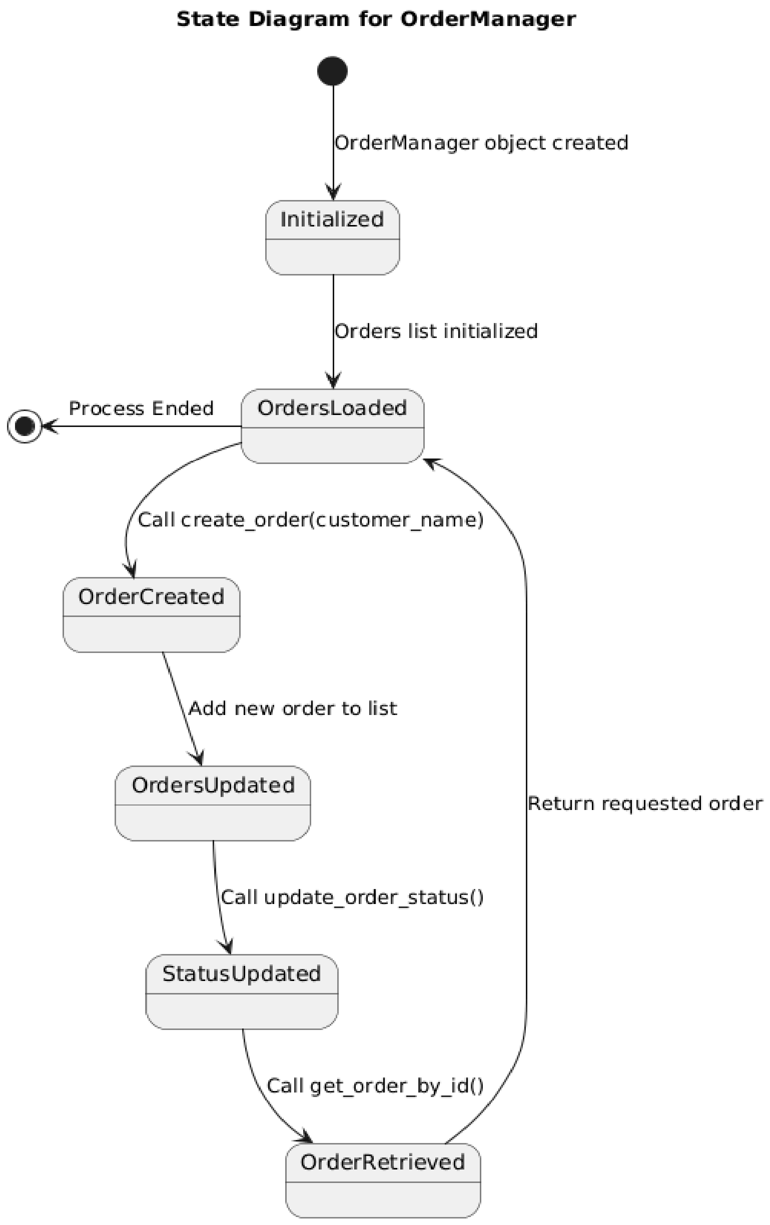

Fig 2.5 State diagram for OrderManager Class

The OrderManager class handles the creation, retrieval, and updating of orders.

States:

- **Initialized**: The OrderManager object is created, and the orders list is initialized.
- **OrdersLoaded**: Existing orders are loaded or the order list is initialized.
- **OrderCreated**: A new order is created using the create_order() method.
- **OrdersUpdated**: The newly created order is added to the orders list.
- **StatusUpdated**: The status of a specific order is updated using update_order_status().
- **OrderRetrieved**: A specific order is retrieved using the get_order_by_id() method.

Transitions:

- Transition from **Initialized** to **OrdersLoaded** occurs when the orders list is initialized.
- Transition to **OrderCreated** happens when create_order(customer_name) is invoked.
- Transition to **OrdersUpdated** follows the creation of a new order.
- Transition to **StatusUpdated** occurs when the order’s status is updated.
- Transition to **OrderRetrieved** happens when retrieving a specific order by ID.
- The process loops back to **OrdersLoaded** after completing the above operations

2.2.5 Order Class State Diagram

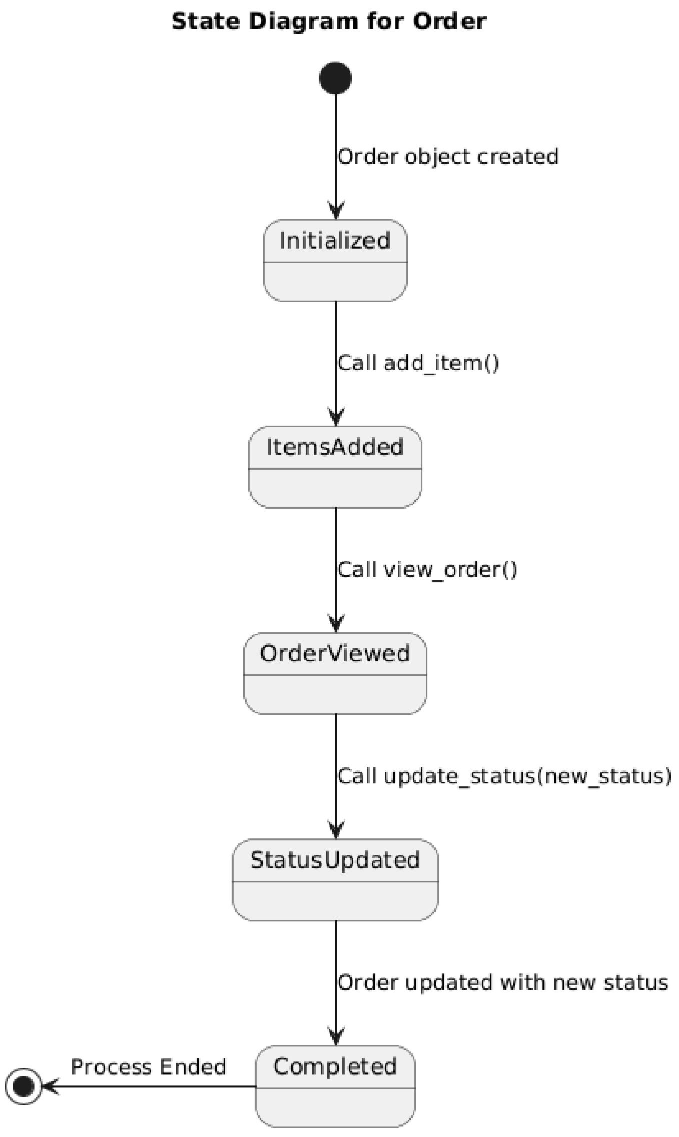

Fig 2.6 State diagram for Order Class

The Order class manages the details of an order, such as items, status, and order information.

States:

- **Initialized**: The Order object is created with attributes like order_id, customer_name, and items.
- **ItemsAdded**: Items are added to the order using the add_item() method.
- **OrderViewed**: The current order details are retrieved using the view_order() method.
- **StatusUpdated**: The order status is updated (e.g., from "pending" to "completed") using update_status(new_status).
- **Completed**: The process ends after successfully updating the order status.

Transitions:

- Transition from **Initialized** to **ItemsAdded** occurs when an item is added to the order.
- Transition to **OrderViewed** happens when order details are viewed.
- Transition to **StatusUpdated** occurs when the order status is updated.
- The process ends at **Completed** after all changes are made.

2.2.6 Sequence Diagram

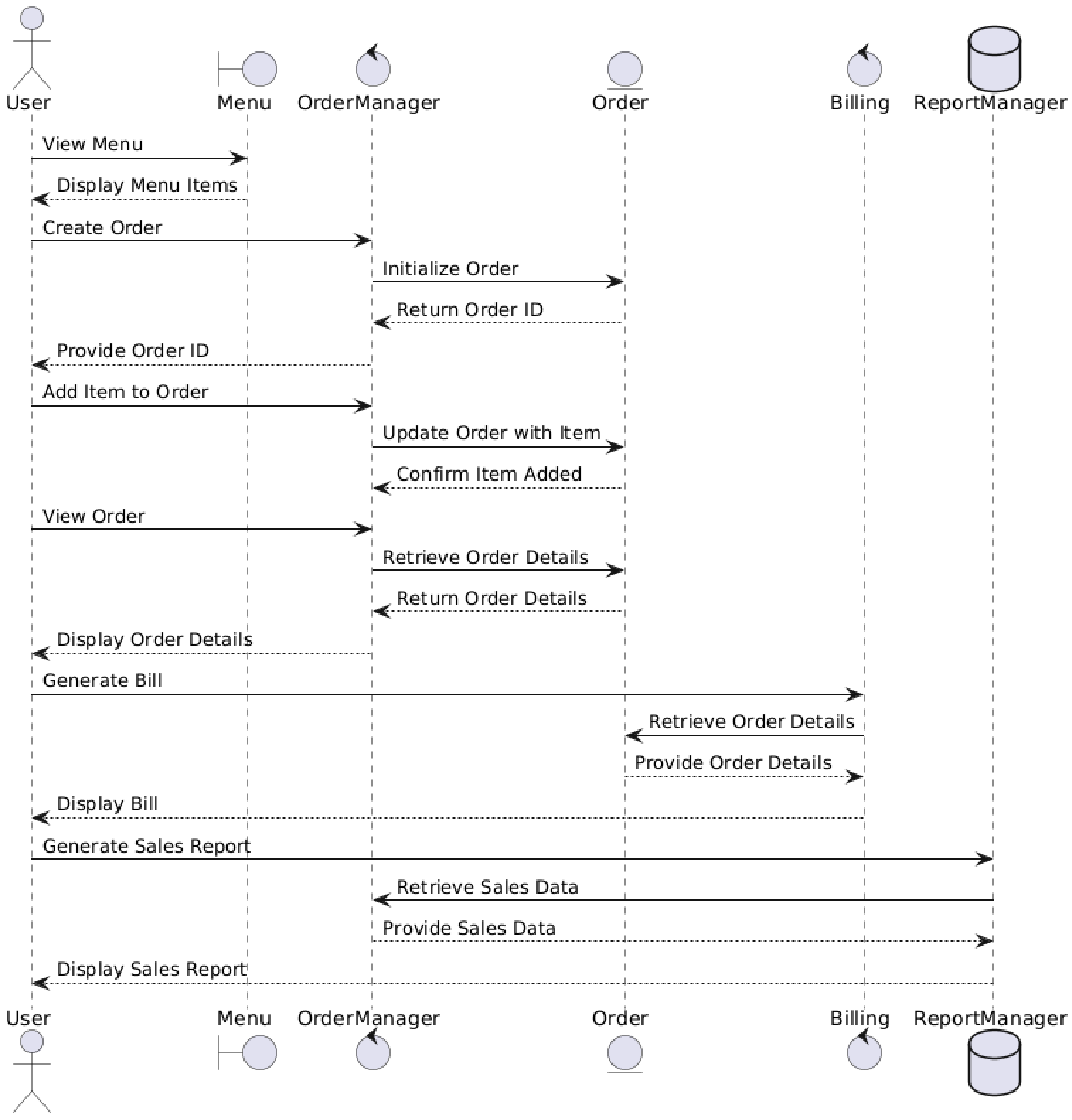

Fig 2.7 Sequence diagram

This sequence diagram depicts the **Cafe Management System**, showing interactions between the **User** and system components (**Menu**, **OrderManager**, **Order**, **Billing**, and **ReportManager**):

1. **View Menu**:
    - User requests the menu.
    - **Menu** displays available items.
2. **Create Order**:
    - User creates an order.
    - **OrderManager** initializes it via **Order** and returns an **Order ID**.
3. **Add Item to Order**:
    - User adds items.
    - **OrderManager** updates the order in **Order**.
4. **View Order**:
    - User requests order details.
    - **OrderManager** retrieves details from **Order**.
5. **Generate Bill**:
    - User requests a bill.
    - **Billing** fetches order details from **Order** and generates the bill.
6. **Generate Sales Report**:
    - User requests a sales report.
    - **ReportManager** retrieves data from **OrderManager** and generates the report.

This flow highlights how the system handles requests and processes them through its components.