RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE

CB23332
SOFTWARE ENGINEERING LAB

Laboratory Record Note Book

Name : . . . . . .

Year / Branch / Section : . . . . . .

Register No. : . . . . . .

Semester : . . . . . .

Academic Year : . . . . . .

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)
RAJALAKSHMI NAGAR, THANDALAM – 602-105

BONAFIDE CERTIFICATE

NAME:_____**REGISTER NO.:** _____

ACADEMIC YEAR: 2024-25 **SEMESTER:** III **BRANCH:**_____B.E/B.Tech

This Certification is the bonafide record of work done by the above student in the

CB23332-SOFTWARE ENGINEERING - Laboratory during the year 2024 – 2025.

Signature of Faculty -in – Charge

Submitted for the Practical Examination held on _____

Internal Examiner

External Examiner

INDEX

S.No.	Name of the Experiment	Expt. Date	Faculty Sign
1.	Preparing Problem Statement		
2.	Software Requirement Specification (SRS)		
3.	Entity-Relational Diagram		
4.	Data Flow Diagram		
5.	Use Case Diagram		
6.	Activity Diagram		
7.	State Chart Diagram		
8.	Sequence Diagram		
9.	Collaboration Diagramt		
10.	Class Diagram		

EX NO:1	WRITE THE COMPLETE PROBLEM STATEMENT
DATE:	

AIM:

To prepare PROBLEM STATEMENT for Fast food billing system

ALGORITHM:

- The problem statement is the initial starting point for a project.
- A problem statement describes what needs to be done without describing how.
- It is basically a one-to-three-page statement that everyone on the project agrees with that describes what will be done at a high level.
- The problem statement is intended for a broad audience and should be written in non- technical terms.
- It helps the non-technical and technical personnel communicate by providing a description of a problem.
- It doesn't describe the solution to the problem.

INPUT:

- The input to requirement engineering is the problem statement prepared by customer.
- It may give an overview of the existing system along with broad expectations from the new system.
- The first phase of requirements engineering begins with requirements elicitation i.e. gathering of information about requirements.
- Here, requirements are identified with the help of customer and existing system processes.

SAMPLE OUTPUT

Stakeholder Problem Statement for Fast Food Billing System

Problem:

In the last six months, the average speed of order processing and bill generation at the fast food chain has declined by 15%, leading to longer wait times for customers. This delay in service is reflected in an increasing number of customer complaints and a decrease in customer satisfaction scores by 10%, compared to the same period last year.

The company aims to reduce the processing time and enhance the customer experience, but achieving this requires streamlining the billing system and upgrading key technology infrastructure.

Background:
The fast food chain has been experiencing delays in order processing, particularly during peak hours. The billing system, which relies on outdated software, has been identified as one of the key bottlenecks. As a result, the order-to-payment time has increased, leading to longer queues and frustrated customers. These delays are contributing to a decrease in overall customer satisfaction, as evidenced by recent feedback and complaints, as well as a drop in customer retention rates.

In order to address this problem, the company must overhaul the billing system and optimize its integration with the kitchen and order management system. This will require investing in more advanced point-of-sale (POS) technology and software upgrades, along with addressing underlying operational inefficiencies.

Relevance:
Fast food service is highly competitive, and customer experience plays a crucial role in maintaining customer loyalty and satisfaction. Long wait times, especially at the checkout stage, negatively impact the customer experience, leading to a reduction in repeat business and lower overall sales. Addressing the delays in billing and order processing will not only improve customer satisfaction but also help increase operational efficiency, reduce customer complaints, and ultimately boost sales.

Objectives:
The primary objective of this project is to improve the speed and accuracy of the billing system, aiming to reduce the processing time by 30% within the next six months. The specific objectives include:

1. Conducting a comprehensive analysis of the current billing system to identify inefficiencies and technical limitations, including integration challenges with the kitchen and order management systems.
2. Upgrading the point-of-sale (POS) technology and software, incorporating more modern systems that allow for faster order processing, quicker bill generation, and smoother payment transactions.
3. Implementing a streamlined billing process that reduces redundancies and enhances operational efficiency by enabling faster data transfer between the kitchen, the cashier, and the payment terminal.
4. Training staff on new technologies and optimized processes to ensure they can operate the new billing system efficiently and are familiar with best practices for speeding up service.

5. Developing and integrating customer-facing features such as mobile ordering and payment, which could reduce the need for in-store billing and further enhance customer convenience.
6. Improving communication between front-of-house and kitchen staff to better manage orders in the queue, ensuring orders are processed and billed without unnecessary delays.
7. Monitoring progress through customer feedback surveys, average order processing time metrics, and sales data, to track improvements in service speed and customer satisfaction.
8. Providing ongoing support and training to the staff to ensure continuous improvements in the speed of billing and order processing as part of the daily operations.
9. Establishing a feedback loop for ongoing evaluation of the system's performance, so that any new inefficiencies or areas for improvement can be identified and addressed promptly.

By addressing the delays in the billing process and upgrading technology, the company can reduce wait times, improve customer satisfaction, and enhance the overall efficiency of the restaurant operations.

Result:

The problem statement was written successfully by following the steps described above.

EX NO:2	WRITE THE SOFTWARE REQUIREMENT SPECIFICATION DOCUMENT
DATE:	

AIM:

To do requirement analysis and develop Software Requirement Specification Sheet(SRS) for Fast food billing system.

ALGORITHM:

SRS shall address are the following:
a) Functionality. What is the software supposed to do?
b) External interfaces. How does the software interact with people, the system‘s hardware, other hardware, and other software?
c) Performance. What is the speed, availability, response time, recovery time of various software functions, etc.?
d) Attributes. What is the portability, correctness, maintainability, security, etc. considerations?
e) Design constraints imposed on an implementation. Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

SAMPLE OUTPUT:

1. INTRODUCTION

1.1 PURPOSE

The purpose of this document is to define the software requirements for the Fast Food Billing System (FFBS) to automate the billing, order processing, and payment system for a fast food restaurant. The system aims to improve customer service by reducing wait times, managing orders efficiently, and ensuring accurate billing and payment processing.

1.2 DOCUMENT CONVENTIONS

The following conventions are used throughout this document:

- DB: Database
- POS: Point of Sale

1.3 INTENDED AUDIENCE AND READING SUGGESTIONS

This document is intended for:

- Developers: To understand the system design and functionality.
- Project Managers: For tracking the progress and ensuring all requirements are met.
- End-users (Restaurant Staff): To understand the functionalities of the billing system.
- Stakeholders: To ensure the system meets business needs.

This document should be read in its entirety to understand the functional and non-functional requirements of the Fast Food Billing System.

1.4 PROJECT SCOPE

The Fast Food Billing System will:

- Automate the billing process from order placement to payment.
- Enable staff to quickly and accurately generate bills for customers.
- Integrate with kitchen order management and inventory systems.
- Provide reporting features such as sales reports and customer payment history.

The system will be deployed in a restaurant setting with a point-of-sale (POS) interface for cashiers and customer-facing displays for payment processing.

1.5 REFERENCES

- POS Software Guidelines
- Fundamentals of Software Engineering by Pressman

---

2. OVERALL DESCRIPTION

2.1 PRODUCT PERSPECTIVE

The Fast Food Billing System is a stand-alone system integrated with the restaurant's existing point-of-sale hardware. It will interact with external systems, such as inventory management and payment gateways, to facilitate smooth order-to-payment processing.

2.2 PRODUCT FEATURES

The key features of the Fast Food Billing System include:

- Order Management: Enables the restaurant staff to place orders into the system.

- Payment Integration: Supports multiple payment methods, including cash, card payments, and mobile wallets.
- Reporting: Generates daily sales reports, order history, and customer receipts.
- Inventory Update: Automatically updates inventory levels when orders are processed.
- Order Customization: Allows customers to customize their orders (e.g., add extra toppings, change sides).

2.3 USER CLASS AND CHARACTERISTICS

The system supports two types of users:

1. Cashier: The cashier uses the system to enter orders, generate bills, process payments, and print receipts.
2. Manager: The manager has administrative access to generate reports, update menu items, view sales data, and manage employee accounts.
3. Customer: Customers interact with the system via digital menus (for self-ordering or cashier-assisted ordering) and payment methods.

2.4 OPERATING ENVIRONMENT

- Operating System: Windows 10/11 or Linux (for POS terminals)
- Database: MySQL or PostgreSQL (for storing orders, sales, and customer data)
- Hardware: POS terminals, barcode scanner, receipt printer, and payment gateway hardware.
- Platform: Java(for backend), HTML/CSS (for frontend interface on POS terminals)
- Internet: Required for payment processing and online reporting.

2.5 DESIGN AND IMPLEMENTATION CONSTRAINTS

- The system should work on a cloud-based environment for ease of updates and access from different locations.
- The database must support multiple concurrent transactions with consistency and reliability.
- The system must ensure that sensitive customer payment data is encrypted during transmission and storage.
- The application should be compatible with modern POS hardware, including receipt printers and barcode scanners.

2.6 ASSUMPTION DEPENDENCIES

- The system assumes that the restaurant uses a standard POS hardware setup.
- The system will rely on external payment gateways for processing electronic payments.

3. SYSTEM FEATURES

3.1 DESCRIPTION AND PRIORITY

The Fast Food Billing System is of high priority as it directly impacts the restaurant's revenue generation, operational efficiency, and customer satisfaction. The major functionalities are:

1. Order Entry: Order items and quantities are entered into the system, either manually by a cashier or automatically via a self-service kiosk.
2. Billing and Payment Processing: The system will calculate the total amount based on the items ordered, taxes, discounts, and any applicable promotions. The cashier or customer can process payment through various methods.
3. Receipt Generation: A printable receipt will be provided, including order details and payment status.

3.2 STIMULUS/RESPONSE SEQUENCES

- Order Placement: The cashier enters the items into the system.
  - Response: The system displays the order details and total price.
- Payment Processing: The cashier selects the payment method.
  - Response: The system processes the payment and updates the order status as "paid." A receipt is generated.
- Cancel Order: A manager can cancel an order before payment.
  - Response: The system updates the inventory and voids the transaction.

3.3 FUNCTIONAL REQUIREMENTS

1. Order Management:
   - Add items to the order.
   - Adjust order quantities and customization options.
   - Apply discounts or promotional offers.
2. Billing:
   - Calculate the total amount, including taxes.
   - Support multiple payment methods (cash, card, wallet).
   - Provide accurate change calculation for cash payments.
3. Payment:
   - Process payments securely via integration with payment gateways.
   - Provide payment confirmation to the customer.
4. Reports:
   - Generate daily/weekly/monthly sales reports.
   - Generate order history and item-level sales analytics.
   - Inventory Integration

---

4. EXTERNAL INTERFACE REQUIREMENTS

4.1 USER INTERFACES

- POS Interface: A graphical user interface (GUI) for cashiers to input orders, view prices, and process payments.
- Admin Interface: A web-based interface for managers to generate reports, manage users, and adjust menu items.

4.2 HARDWARE INTERFACES

- POS Terminal: Touchscreen terminal for cashiers to input data.
- Receipt Printer: To print customer receipts.
- Barcode Scanner: For item identification and pricing.

4.3 SOFTWARE INTERFACES

- Database: MySQL or PostgreSQL will store all order and transaction details.
- Payment Gateway: Integration with external payment services (e.g., Stripe, PayPal).

4.4 COMMUNICATION INTERFACES

- Wi-Fi: Required for data synchronization between POS terminals and central server.
- Cloud API: For accessing sales data, generating reports, and updating menus remotely.

---

5. NON-FUNCTIONAL REQUIREMENTS

5.1 PERFORMANCE REQUIREMENTS

- Response Time: The system should process each transaction within 5 seconds.
- Availability: The system should be available 24/7 with less than 1% downtime per year.
- Scalability: The system should handle up to 500 simultaneous transactions during peak hours.

5.2 SAFETY REQUIREMENTS

- Backup and Recovery: Daily backups of transaction and inventory data should be performed.
- Disaster Recovery: The system should have a plan for recovering from hardware or software failures.

5.3 SECURITY REQUIREMENTS

- Data Encryption: Sensitive data (e.g., customer payment information) should be encrypted using SSL/TLS.
- Access Control: Role-based access control (RBAC) to ensure that only authorized users can perform certain actions.

5.4 SOFTWARE QUALITY ATTRIBUTES

- Usability: The user interface should be intuitive for fast food restaurant staff, requiring minimal training.
- Maintainability: The system should support easy updates, including menu changes, price updates, and bug fixes.
- Portability: The system should be compatible with all major POS hardware and operating systems.

Result:

The SRS was made successfully by following the steps described above.

EX NO:3	DRAW THE ENTITY RELATIONSHIP DIAGRAM
DATE:	

AIM:

To Draw the Entity Relationship Diagram for Fast food billing system.

ALGORITHM:

Step 1: Mapping of Regular Entity Types
Step 2: Mapping of Weak Entity Types
Step 3: Mapping of Binary 1:1 Relation Types
Step 4: Mapping of Binary 1:N Relationship Types.
Step 5: Mapping of Binary M:N Relationship Types.
Step 6: Mapping of Multivalued attributes.

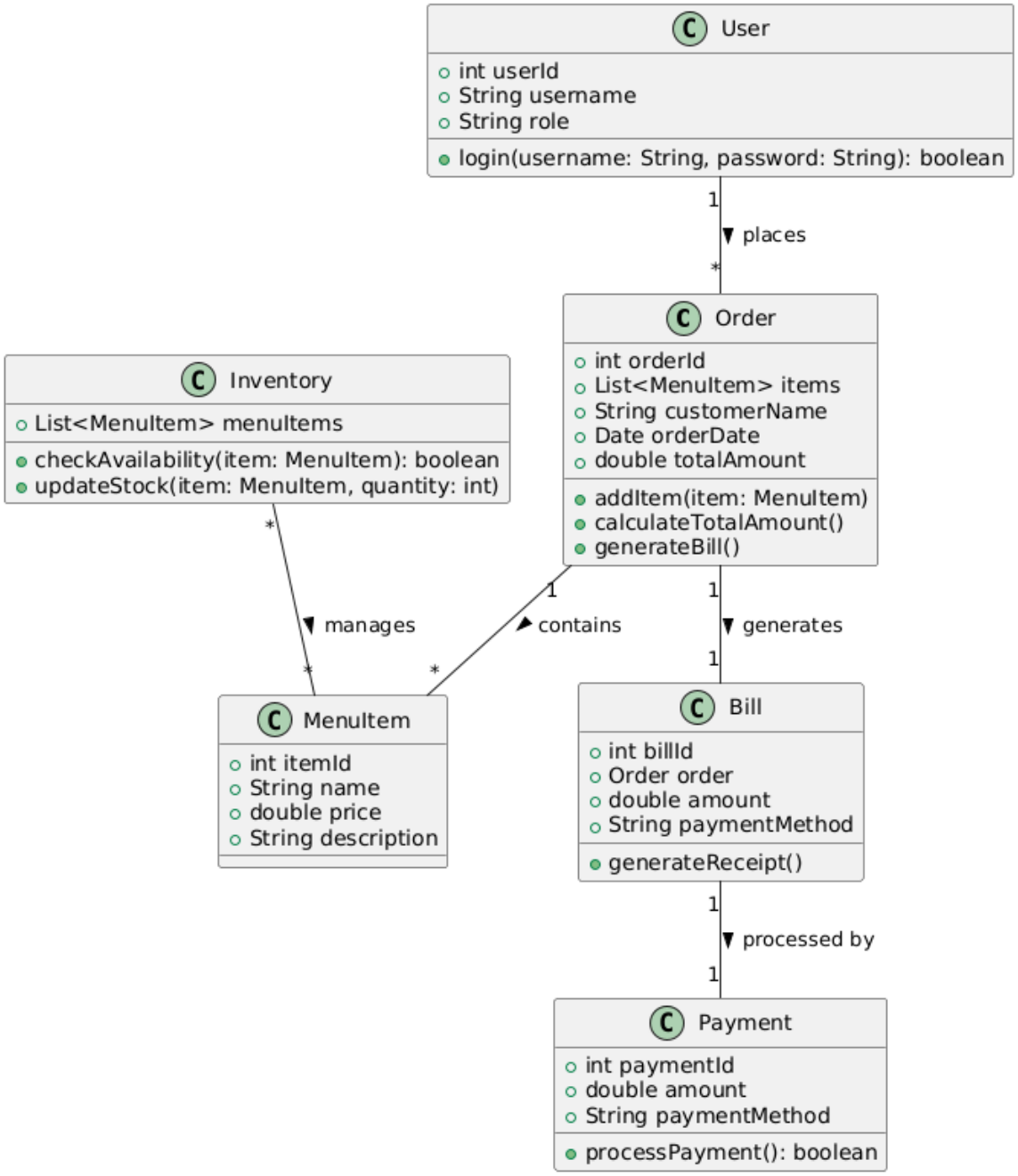
INPUT:

Entities
Entity Relationship Matrix
Primary Keys
Attributes
Mapping of Attributes with Entities

SAMPLE OUTPUT:
The Fully Attributed ERD

Result:

The entity relationship diagram was made successfully by following the steps described above.


User
int userId
String username
String role
login(username: String, password: String): boolean
1
places
*
Order
int orderId
List<MenuItem> items
String customerName
Date orderDate
double totalAmount
addItem(item: MenuItem)
calculateTotalAmount()
generateBill()
Inventory
List<MenuItem> menuItems
checkAvailability(item: MenuItem): boolean
updateStock(item: MenuItem, quantity: int)
*
manages
*
1
contains
*
1
generates
1
MenuItem
int itemId
String name
double price
String description
Bill
int billId
Order order
double amount
String paymentMethod
generateReceipt()
1
processed by
1
Payment
int paymentId
double amount
String paymentMethod
processPayment(): boolean

EX NO:4	DRAW THE DATA FLOW DIAGRAMS AT LEVEL 0 AND LEVEL 1
DATE:	

AIM:

To Draw the Data Flow Diagram for Fast food billing system
and List the Modules in the Application.

ALGORITHM:

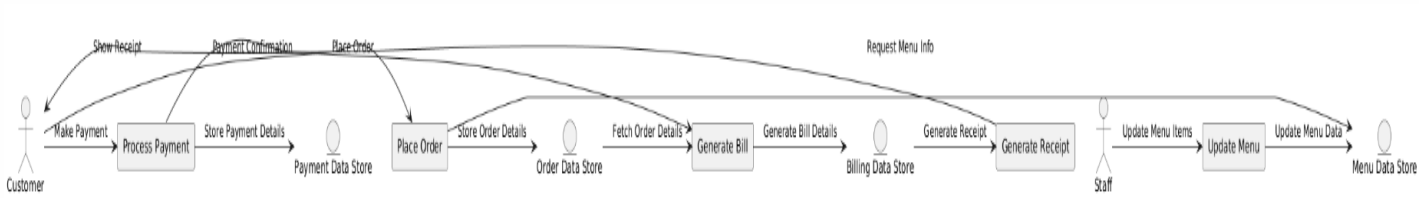
1. Open the Visual Paradigm to draw DFD (Ex.Lucidchart)
2. Select a data flow diagram template
3. Name the data flow diagram
4. Add an external entity that starts the process
5. Add a Process to the DFD
6. Add a data store to the diagram
7. Continue to add items to the DFD
8. Add data flow to the DFD
9. Name the data flow
10. Customize the DFD with colours and fonts
11. Add a title and share your data flow diagram

INPUT:

- Processes
- Datastores
- External Entities

Result:

The Data Flow diagram was made successfully by following the steps described above.


Show Receipt
Payment Confirmation
Place Order
Request Menu Info
Customer
Make Payment
Process Payment
Store Payment Details
Payment Data Store
Place Order
Store Order Details
Order Data Store
Fetch Order Details
Generate Bill
Generate Bill Details
Billing Data Store
Generate Receipt
Generate Receipt
Staff
Update Menu Items
Update Menu
Update Menu Data
Menu Data Store

EX NO:5	DRAW USE CASE DIAGRAM
DATE:	

AIM:

To Draw the Use Case Diagram for Fast food billing system

ALGORITHM:

Step 1: Identify Actors
Step 2: Identify Use Cases
Step 3: Connect Actors and Use Cases
Step 4: Add System Boundary
Step 5: Define Relationships
Step 6: Review and Refine
Step 7: Validate

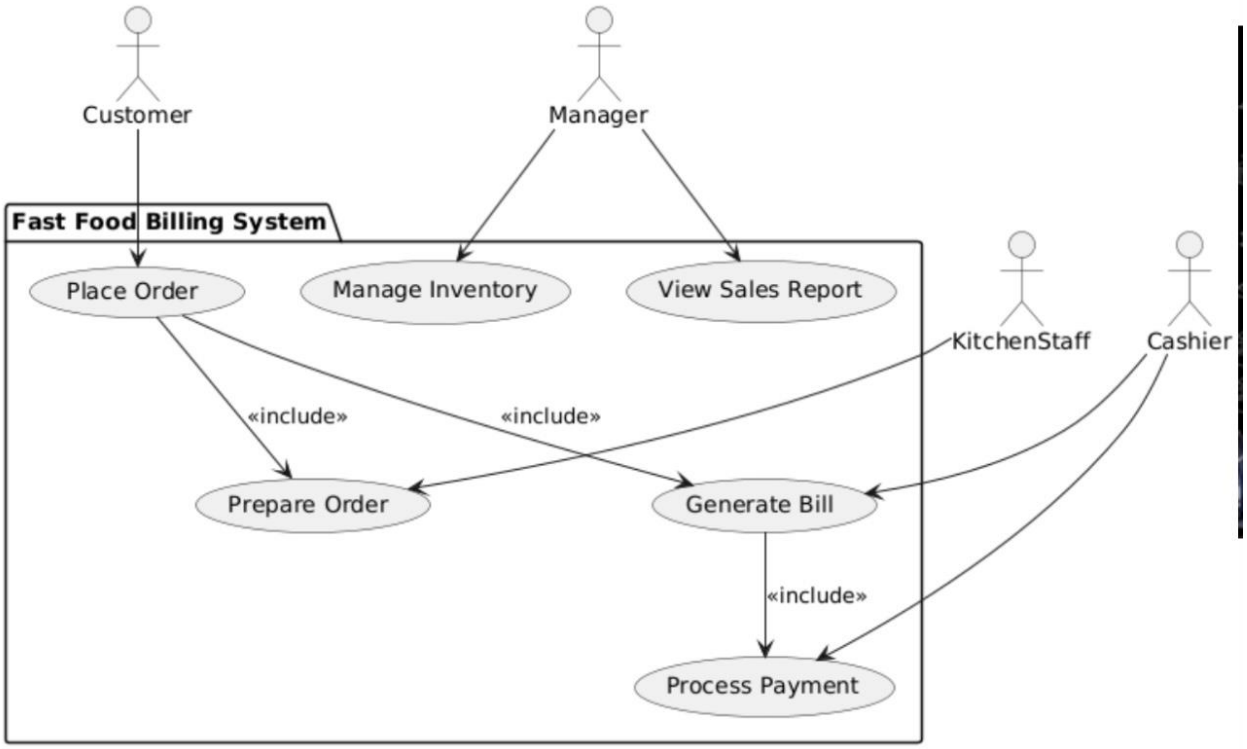
INPUTS:

The main inputs for the Use Case Diagram are as follows:

- Actors
- Use Cases
- Relations

Result:

The use case diagram has been created successfully by following the steps given.


Customer
Manager
Fast Food Billing System
Place Order
Manage Inventory
View Sales Report
KitchenStaff
Cashier
«include»
«include»
Prepare Order
Generate Bill
«include»
Process Payment

EX NO:6	DRAW ACTIVITY DIAGRAM OF ALL USE CASES.
DATE:	

AIM:

To Draw the activity Diagram for Fast food billing system.

ALGORITHM:

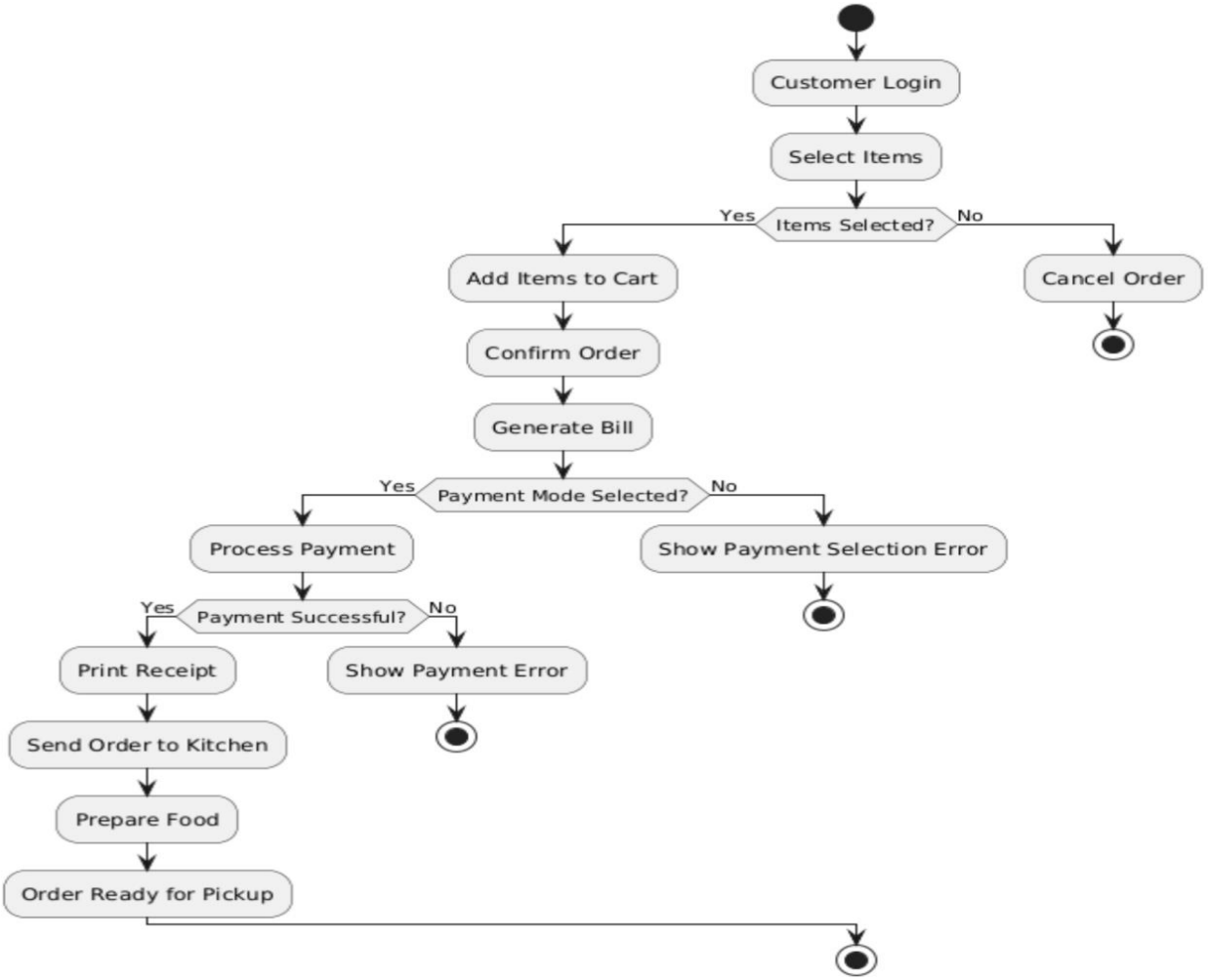
Step 1: Identify the Initial State and Final States
Step 2: Identify the Intermediate Activities Needed
Step 3: Identify the Conditions or Constraints
Step 4: Draw the Diagram with Appropriate Notations

INPUTS:

- Activities
- Decision Points
- Guards
- Parallel Activities
- Conditions

Result:

The Activity diagram has been created successfully by following the steps given.


Customer Login
Select Items
Items Selected?
Yes
No
Add Items to Cart
Cancel Order
Confirm Order
Generate Bill
Payment Mode Selected?
Yes
No
Process Payment
Show Payment Selection Error
Payment Successful?
Yes
No
Print Receipt
Show Payment Error
Send Order to Kitchen
Prepare Food
Order Ready for Pickup

EX NO:7	DRAW STATE CHART DIAGRAM OF ALL USE CASES.
DATE:	

AIM:

To Draw the State Chart Diagram for Fast food billing system

ALGORITHM:

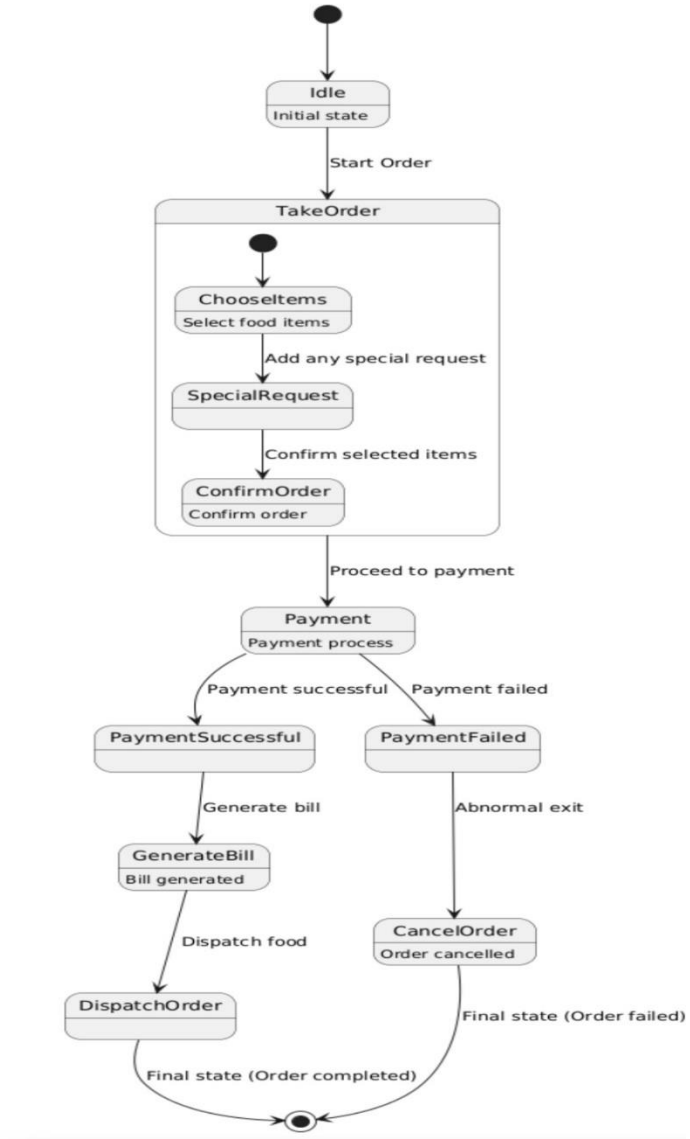
STEP-1: Identify the important objects to be analysed.
STEP-2: Identify the states.
STEP-3: Identify the events.
State chart diagram is used to describe the states of different objects in its life cycle.
Emphasis is placed on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately. State chart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.

INPUTS:

- Objects
- States
- Events

Result:

The State Chart diagram has been created successfully by following the steps given.


Idle
Initial state
Start Order
TakeOrder
ChooseItems
Select food items
Add any special request
SpecialRequest
Confirm selected items
ConfirmOrder
Confirm order
Proceed to payment
Payment
Payment process
Payment successful
Payment failed
PaymentSuccessful
PaymentFailed
Generate bill
Abnormal exit
GenerateBill
Bill generated
Dispatch food
CancelOrder
Order cancelled
DispatchOrder
Final state (Order failed)
Final state (Order completed)

EX NO:8	DRAW SEQUENCE DIAGRAM OF ALL USE CASES.
DATE:	

AIM:

To Draw the Sequence Diagram for Fast food billing system

ALGORITHM:

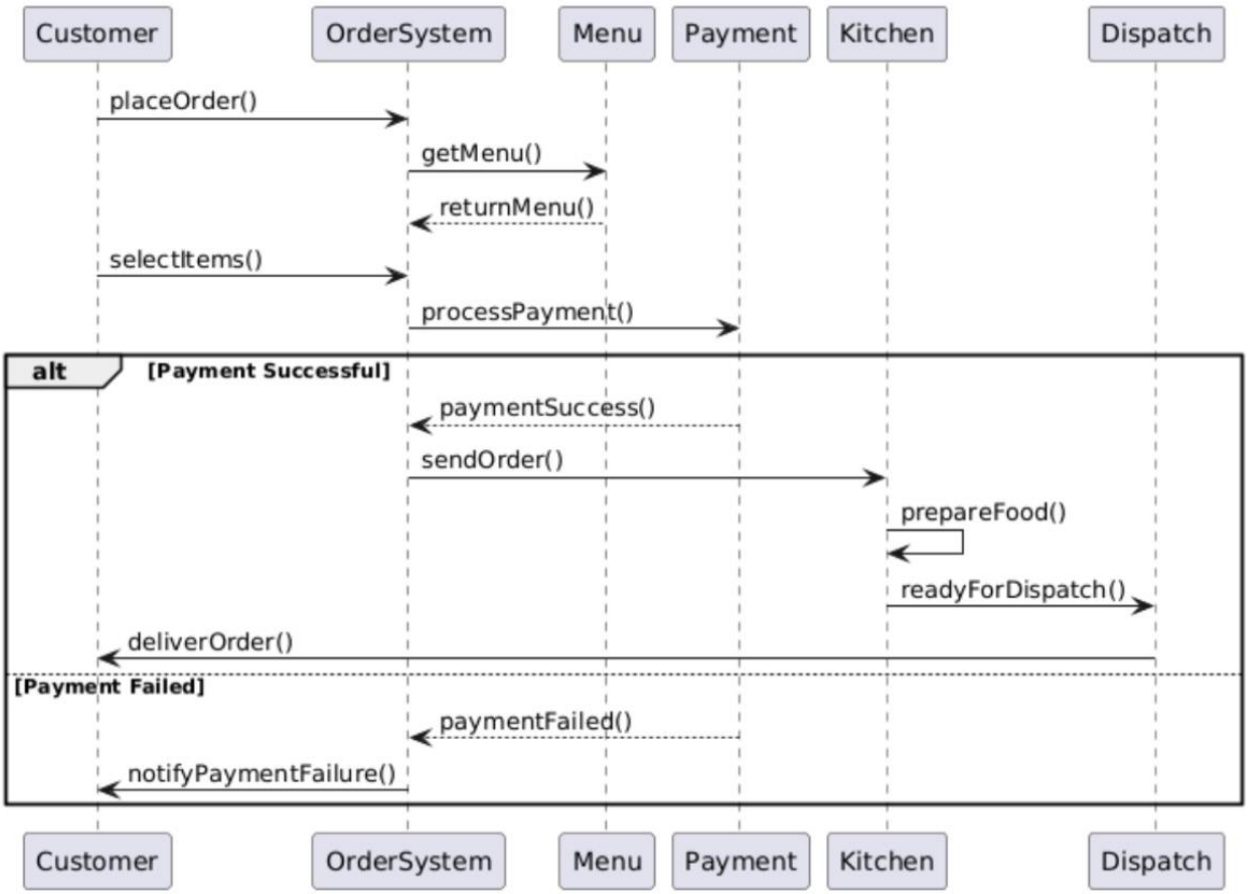
1. Identify the Scenario
2. List the Participants
3. Define Lifelines
4. Arrange Lifelines
5. Add Activation Bars
6. Draw Messages
7. Include Return Messages
8. Indicate Timing and Order
9. Include Conditions and Loops
10. Consider Parallel Execution
11. Review and Refine
12. Add Annotations and Comments
13. Document Assumptions and Constraints
14. Use a Tool to create a neat sequence diagram

Inputs:

- Objects taking part in the interaction.
- Message flows among the objects.
- The sequence in which the messages are flowing.
- Object organization.

Result:

The Sequence diagram has been created successfully by following the steps given.


Customer
OrderSystem
Menu
Payment
Kitchen
Dispatch
placeOrder()
getMenu()
returnMenu()
selectItems()
processPayment()
alt
[Payment Successful]
paymentSuccess()
sendOrder()
prepareFood()
readyForDispatch()
deliverOrder()
[Payment Failed]
paymentFailed()
notifyPaymentFailure()
Customer
OrderSystem
Menu
Payment
Kitchen
Dispatch

EX NO:9	DRAW COLLABORATION DIAGRAM OF ALL USE CASES
DATE:	

AIM:

To Draw the Collaboration Diagram for Fast food billing system

ALGORITHM:

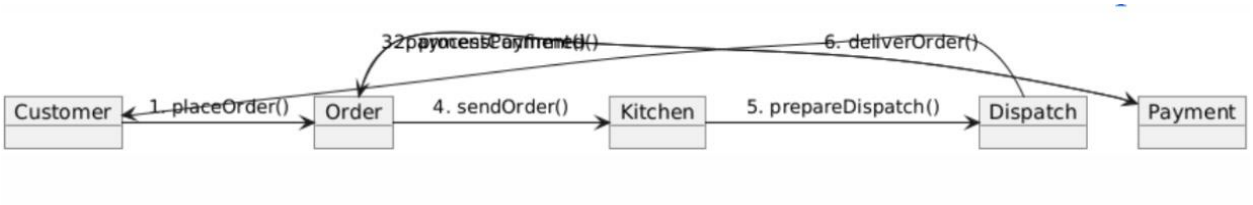
Step 1: Identify Objects/Participants
Step 2: Define Interactions
Step 3: Add Messages
Step 4: Consider Relationships
Step 5: Document the collaboration diagram along with any relevant explanations or annotations.

Inputs

- Objects taking part in the interaction.
- Message flows among the objects.
- The sequence in which the messages are flowing.
- Object organization.

Result:

The Collaboration diagram has been created successfully by following the steps given.


32processPayment()
6. deliverOrder()
Customer
1. placeOrder()
Order
4. sendOrder()
Kitchen
5. prepareDispatch()
Dispatch
Payment

EX NO:10	ASSIGN OBJECTS IN SEQUENCE DIAGRAM TO CLASSES AND MAKE CLASS DIAGRAM.
DATE:	

AIM:

To Draw the Class Diagram for Fast food billing system.

Algorithm:

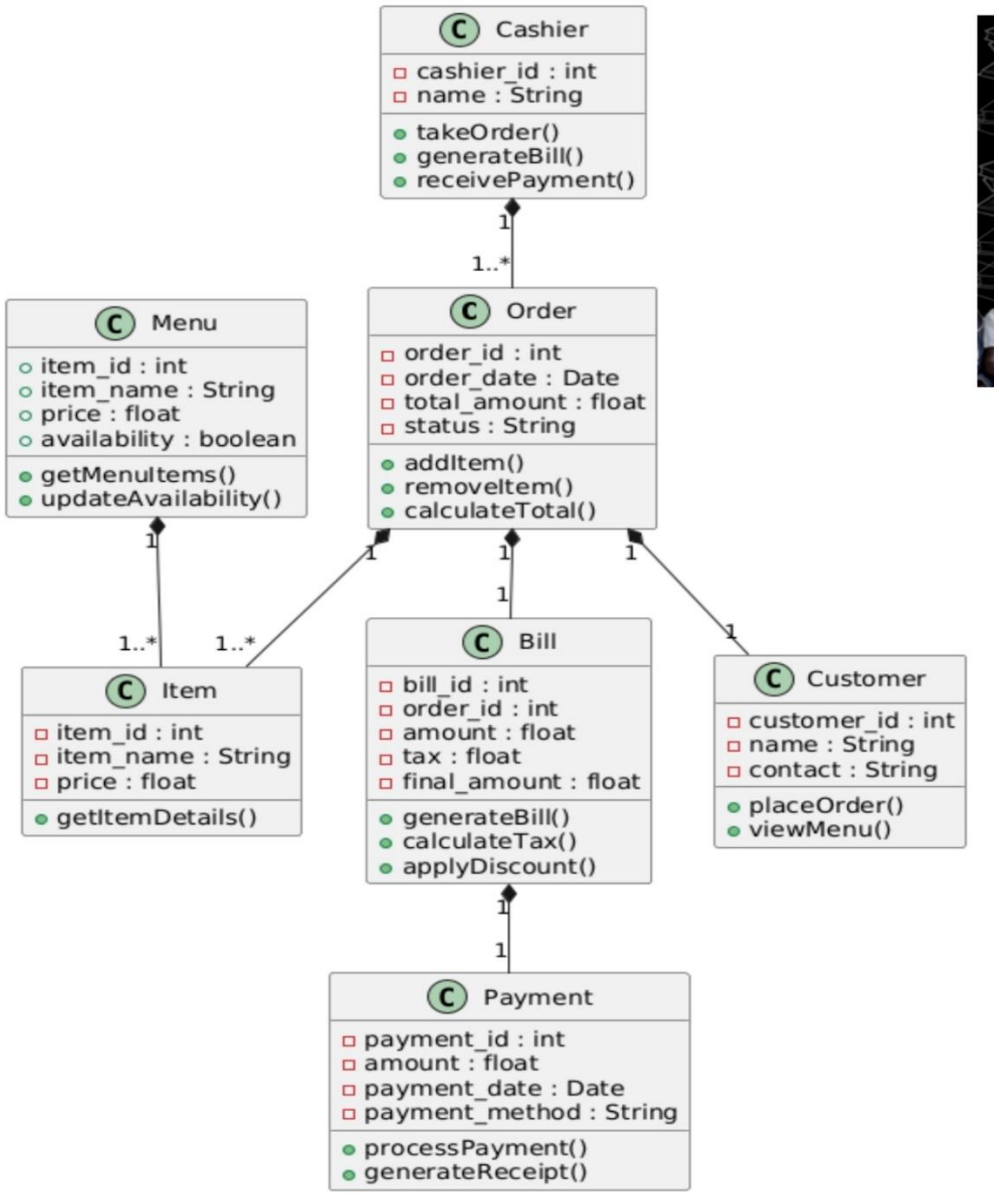
1. Identify Classes
2. List Attributes and Methods
3. Identify Relationships
4. Create Class Boxes
5. Add Attributes and Methods
6. Draw Relationships
7. Label Relationships
8. Review and Refine
9. Use Tools for Digital Drawing

INPUT:

1. Class Name
2. Attributes
3. Methods
4. Visibility Notation

RESULT:

The Class diagram has been created successfully by following the steps given.


Cashier
cashier_id : int
name : String
takeOrder()
generateBill()
receivePayment()
1
1..*
Menu
item_id : int
item_name : String
price : float
availability : boolean
getMenuItems()
updateAvailability()
Order
order_id : int
order_date : Date
total_amount : float
status : String
addItem()
removeItem()
calculateTotal()
1
1..*
1..*
1
1
1
1
1
Item
item_id : int
item_name : String
price : float
getItemDetails()
Bill
bill_id : int
order_id : int
amount : float
tax : float
final_amount : float
generateBill()
calculateTax()
applyDiscount()
Customer
customer_id : int
name : String
contact : String
placeOrder()
viewMenu()
1
1
Payment
payment_id : int
amount : float
payment_date : Date
payment_method : String
processPayment()
generateReceipt()

EX NO:11	
DATE:	**MINI PROJECT ON FASTFOOD BILLING SYSTEM**

Aim

To design and implement a Fast Food Billing System using Python and Streamlit, we will create a web-based application that allows users to select food items, calculate the total bill, apply discounts, and manage orders efficiently.

Algorithm:

1. Initialize the Application

- **Import Libraries**:
  - Use streamlit, sqlite3, pandas, and datetime for the web interface and data management.
  - Use folium and streamlit-folium for map integration (optional if you want location tracking).
  - Use requests for fetching real-time data (for GPS, for example, or mock data).
- **Database Initialization**:
  - Create SQLite tables to store:
    - **Items**: (item_id, item_name, description, price).
    - **Orders**: (order_id, customer_name, order_details, total_price, timestamp).
    - **Feedback**: (feedback_id, customer_name, order_id, feedback_text, timestamp).

2. Database Operations (CRUD)

- **add_item**: Add a new food item (name, description, price).
- **get_items**: Retrieve all food items.
- **add_order**: Add a new customer order (items ordered, quantities, total).
- **get_orders**: Retrieve all customer orders.
- **add_feedback**: Add feedback for an order.
- **get_feedback**: Retrieve all feedback.

3. Design the Streamlit User Interface

- Use st.sidebar.selectbox for navigation with options:
  - **Real-Time Billing** (Customer interaction for placing orders).
  - **Menu Management** (For adding food items).
  - **Order Management** (For viewing orders).
  - **Customer Feedback** (For viewing and submitting feedback).

4. Module Implementation

- Real-Time Billing (Customer Order Module):

- Use input fields to select food items and quantities.
- Calculate total cost and display the summary.
- Integrate a payment section to simulate payment and show the change.
- Store the order and payment details in the database.

- Menu Management:

- Add new food items to the menu.
- Display the list of available food items from the database.

- Order Management: Display all orders with details (items, customer name, total price).

- Feedback Module:

- Allow customers to submit feedback for their orders.
- Display all feedback received from customers.

5. Real-Time Integration

- Mock real-time GPS data or use a simple logic to simulate order progress or update status.
- Use folium.Map and folium.Marker to dynamically show any geographical details related to the order (if required).

6. Data Validation

- Validate user input (e.g., ensure the item name is valid, quantities are positive).
- Handle edge cases gracefully (e.g., insufficient payment, missing data).

7. Connect UI with Database Operations

- Map the UI actions to the database operations:
  - **Add Item** → add_item.
  - **Add Order** → add_order.
  - **Submit Feedback** → add_feedback.
  - **Fetch Data** → get_items, get_orders, get_feedback.

8. Display Data

- Use st.dataframe to display food items, orders, and feedback.
- Use folium for any location or interactive map updates (optional).

CODE:

```
import java.util.HashMap;
import java.util.Map;

public class Menu {
   private Map<String, Double> items;

   public Menu() {
      items = new HashMap<>();
      // Predefined Menu Items and Prices
      items.put("Burger", 5.99);
      items.put("Fries", 2.99);
      items.put("Soda", 1.50);
      items.put("Pizza", 8.99);
      items.put("Salad", 4.49);
   }
```

```
// Display menu
   public void displayMenu() {
      System.out.println("Welcome to the Fast Food Restaurant!");
      System.out.println("Here is the Menu:");
      for (Map.Entry<String, Double> entry : items.entrySet()) {
         System.out.printf("%-20s $%.2f\n", entry.getKey(), entry.getValue());
      }
   }

   // Get the price of an item
   public double getItemPrice(String item) {
      return items.getOrDefault(item, 0.0);
   }
}
import java.util.HashMap;
import java.util.Map;

public class Order {
   private Map<String, Integer> orderedItems;
   private Menu menu;

   public Order(Menu menu) {
      orderedItems = new HashMap<>();
      this.menu = menu;
   }

   // Add item to the order
   public void addItem(String item, int quantity) {
      orderedItems.put(item, orderedItems.getOrDefault(item, 0) + quantity);
   }

   // Display the order summary
   public void displayOrder() {
      double total = 0;
      System.out.println("\nYour Order:");
      for (Map.Entry<String, Integer> entry : orderedItems.entrySet()) {
         String item = entry.getKey();
         int quantity = entry.getValue();
         double price = menu.getItemPrice(item);
         total += price * quantity;
         System.out.printf("%-20s x%d = $%.2f\n", item, quantity, price * quantity);
      }
      System.out.printf("\nTotal: $%.2f\n", total);
   }

// Calculate total price
   public double calculateTotal() {
      double total = 0;
      for (Map.Entry<String, Integer> entry : orderedItems.entrySet()) {
         total += menu.getItemPrice(entry.getKey()) * entry.getValue();
      }
      return total;
   }
}
import java.util.Scanner;
```

```
public class BillingSystem {
    private Menu menu;
    private Order order;

    public BillingSystem() {
        menu = new Menu();
        order = new Order(menu);
    }

    // Method to handle user input and order process
    public void start() {
        Scanner scanner = new Scanner(System.in);

        while (true) {
            menu.displayMenu();
            System.out.println("\nEnter the item you want to order (or type 'done' to finish): ");
            String item = scanner.nextLine().trim();

            if (item.equalsIgnoreCase("done")) {
                break;
            }

            System.out.println("Enter the quantity: ");
            int quantity = scanner.nextInt();
            scanner.nextLine();  // consume newline character

            if (menu.getItemPrice(item) == 0) {
                System.out.println("Sorry, we don't have that item. Please select from the menu.");
            } else {
                order.addItem(item, quantity);
            }
        }

        order.displayOrder();
        processPayment(scanner);
    }

    // Method to handle the payment process
    private void processPayment(Scanner scanner) {
        double totalAmount = order.calculateTotal();

        if (totalAmount == 0) {
            System.out.println("No items in the order.");
            return;
        }

        System.out.printf("\nTotal amount to be paid: $%.2f\n", totalAmount);
        System.out.print("Enter payment amount: $");
        double payment = scanner.nextDouble();

        if (payment < totalAmount) {
            System.out.println("Insufficient payment. Please pay the full amount.");
        } else {
            double change = payment - totalAmount;
            System.out.printf("Payment successful! Your change is: $%.2f\n", change);
            System.out.println("Thank you for dining with us!");
        }
    }
```

```
public static void main(String[] args) {
      BillingSystem billingSystem = new BillingSystem();
      billingSystem.start();
   }
}
```

Conclusion:

In this implementation of the Fast Food Billing System using Streamlit and SQLite, we have successfully created a dynamic web-based application where users can place orders, view menu items, submit feedback, and manage orders. The system integrates a SQLite database for persistent data storage, ensuring that all menu items, orders, and feedback are stored and easily accessible