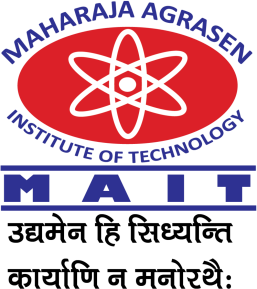
Software Engineering

Faculty Name – Dr. Jyoti Kaushik Student Name – Shivam Kumar Jha Branch – CSE (Shift-1) Semester – Fifth Semester

Enrollment no – 02614802722 Group – C2



Maharaja Agrasen Institute of Technology, PSP Area, Sector – 22, Rohini, New Delhi - 110083



# MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

**COMPUTER SCIENCE & ENGINEERING DEPARTMENT**

# VISION

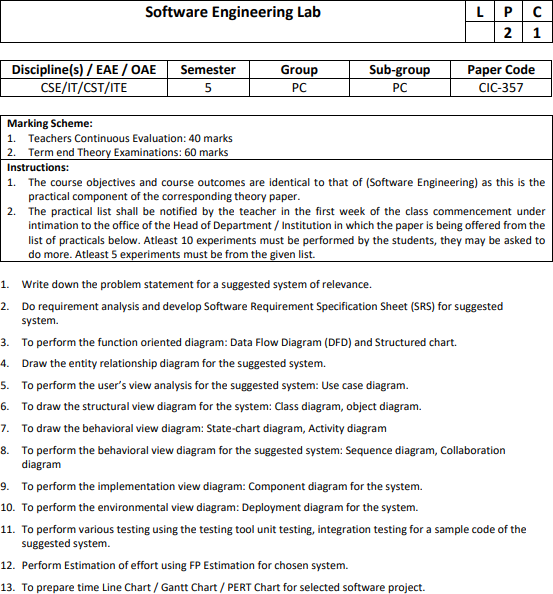
“To be center of excellence in education, research and technology transfer in the field of computer engineering and promote entrepreneurship and ethical values.”

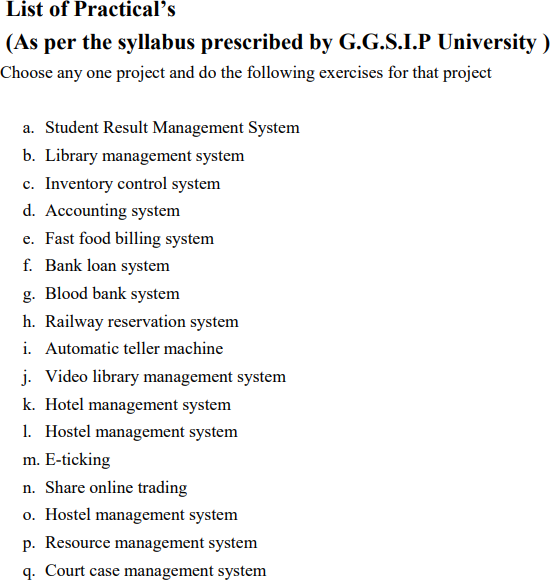
# MISSION

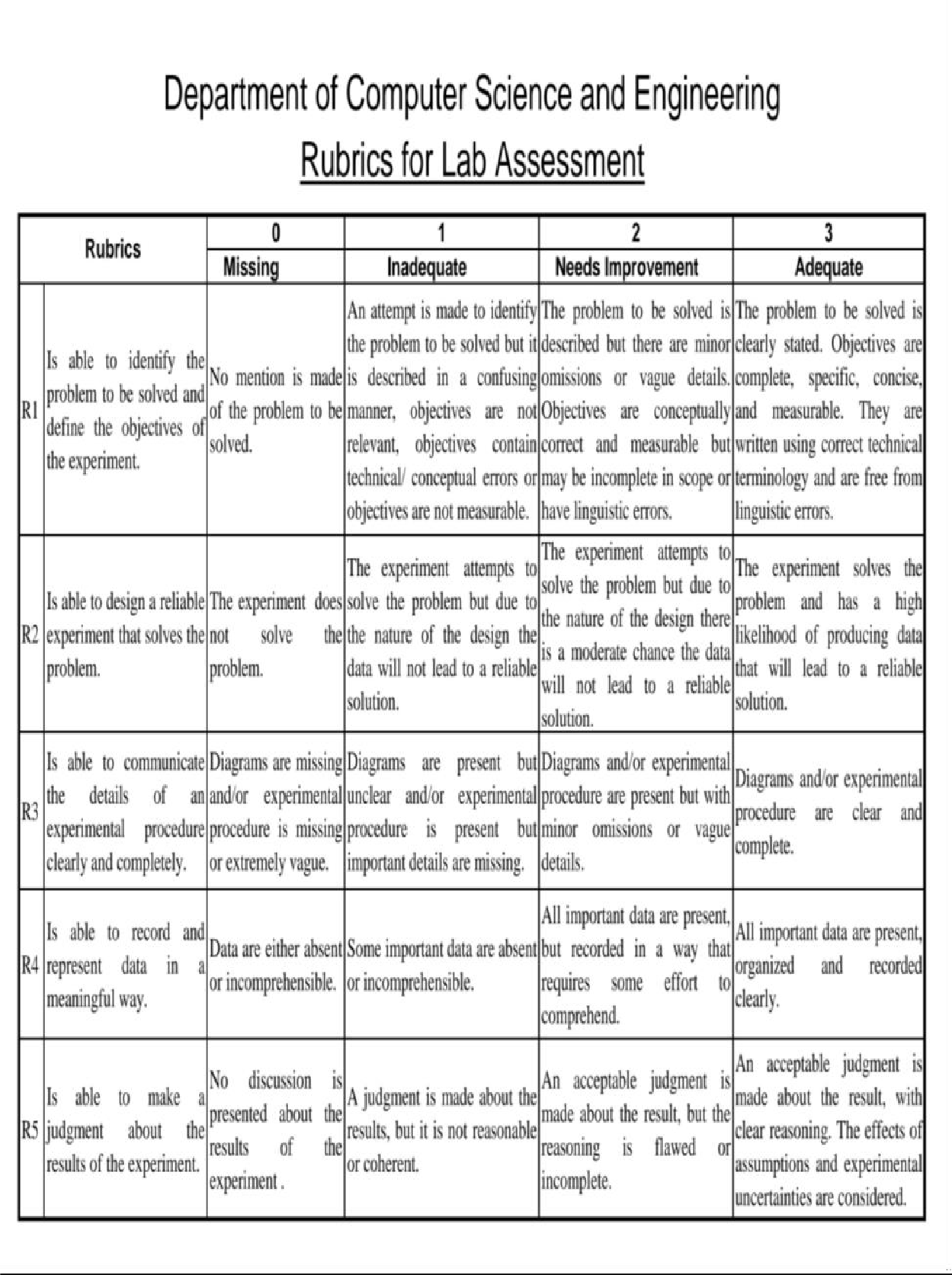
“To foster an open, multidisciplinary and highly collaborative research environment to produce world-class engineers capable of providing

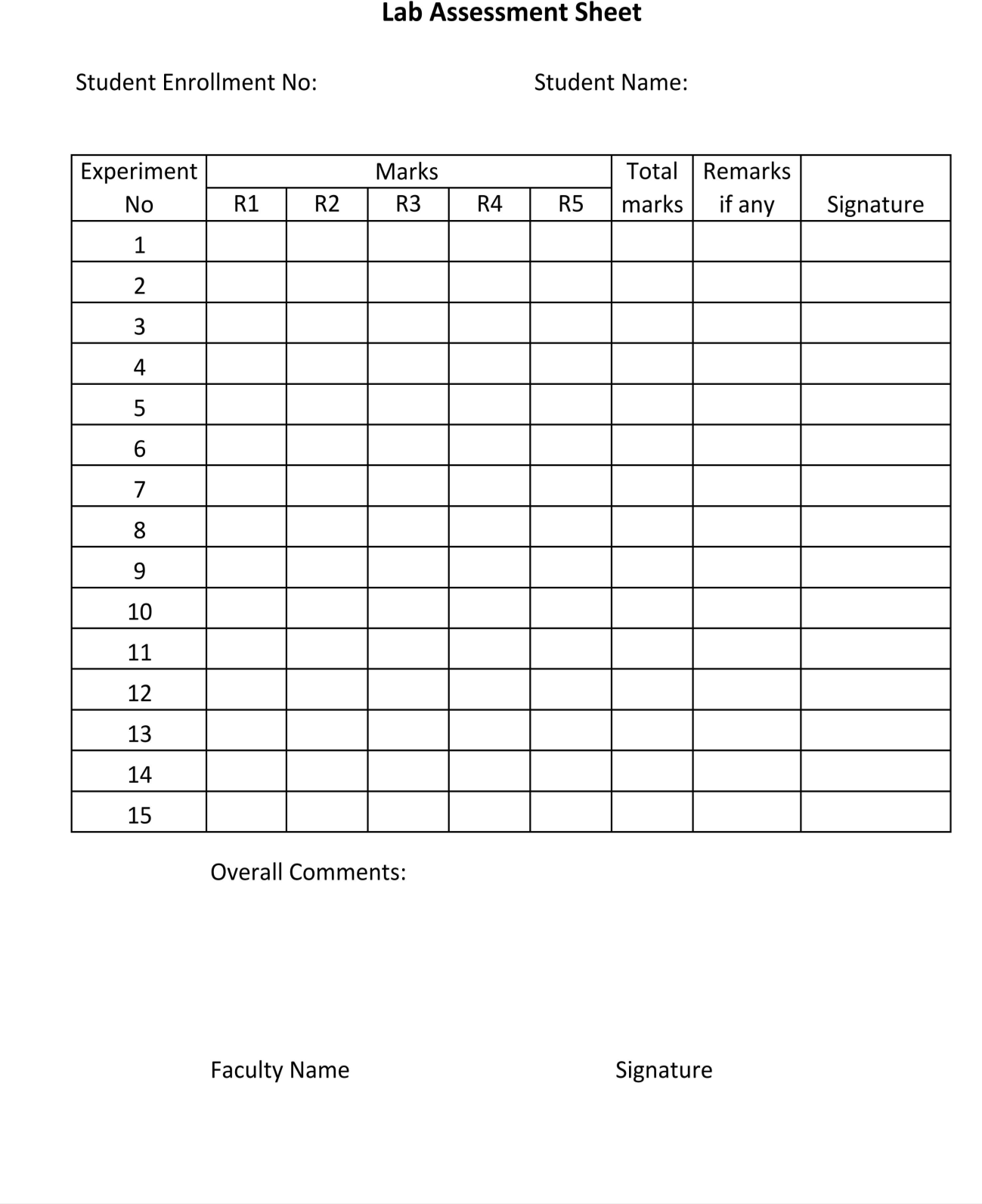
innovative solutions to real life problems and fulfill societal needs.”

**LIST OF EXPERIMENTS (As prescribed by GGSIPU)**

****





****

**02614802722**

**Shivam Kumar Jha**

**Experiment-1**

**AIM-** Write down the problem statement for a suggested system of relevance.

**Supermarket Billing System: Solutions Overview**

The **Supermarket Billing System** is designed to streamline and automate the billing process in a retail environment. This system will address key pain points in traditional billing processes, such as slow checkout times, manual errors, and inventory mismatches. By integrating real-time inventory updates, automated discount application, and efficient payment processing, the system aims to enhance the shopping experience for both customers and staff. The proposed solution focuses on accuracy, efficiency, and scalability, making it suitable for a wide range of supermarket sizes.

**---**

**# 1. Challenges**

**- Inventory Mismatches:** Managing a large inventory in real time can lead to discrepancies between stock levels and actual availability, resulting in delays and customer dissatisfaction.

- **Checkout Delays**: During peak hours, long queues can form, leading to a frustrating experience for customers and missed sales opportunities

**- Pricing Errors:** Manual price entry and frequent price changes in supermarkets increase the chances of pricing errors, which can affect customer trust.

**- Loyalty Integration:** Many supermarkets offer loyalty programs, and integrating these with the billing system can be complex, especially when calculating rewards or redeeming points.

**## 2. Security: A Top Priority**

**- Payment Data Security:** Handling sensitive payment information requires strong security measures. The system will implement encryption protocols for all transactions and adhere to PCI-DSS compliance to protect customer card data.

**- Employee Access Control:** Limiting access to certain system functionalities based on user roles, such as cashier, manager, and admin, will prevent unauthorized changes and reduce potential fraud.

**- Data Privacy:** With customer data being collected (like loyalty information), maintaining compliance with data protection regulations (e.g., GDPR) is crucial to prevent legal risks.

**## 3. Real-Time Inventory Tracking: A Game-Changer**

Real-time inventory tracking is a game-changing feature of this system. As items are scanned at checkout, the system will instantly update stock levels, ensuring that the supermarket has accurate data on item availability. This reduces the risk of stockouts, prevents over-ordering, and enables the management to make data-driven decisions for restocking. Additionally, real-time inventory tracking can integrate with e-commerce systems, allowing customers to view item availability online before visiting the store.

**## 4. Barcode Scanning and Price Calculation: A Critical Component**

The barcode scanning and price calculation feature is essential to the Supermarket Billing System. It ensures that each item is accurately identified and priced, which is critical for reducing errors at checkout. This component not only speeds up the checkout process but also provides the flexibility to apply discounts and promotions automatically. By eliminating manual price entry, it enhances accuracy, reduces cashier workload, and improves the overall customer experience.

**## 5. Logistics and Stock Replenishment: A Logistical Challenge**

Managing stock replenishment and restocking efficiently is a logistical challenge for supermarkets. Integrating the billing system with inventory management requires precise tracking of items sold and predictive algorithms to forecast demand based on sales trends. For instance, items in high demand need faster restocking to avoid running out, while slow-moving items should be ordered in smaller quantities. Real-time tracking also requires smooth communication between the billing and stock systems, so that low-stock alerts trigger timely restocks.

**## 6. System Integration and Customer Adoption: A Critical Issue**

One of the main challenges in implementing this system is integrating it with existing supermarket infrastructure, especially in large chains with legacy systems. Transitioning staff to a new system can lead to temporary operational challenges. Additionally, gaining customer acceptance, particularly for loyalty programs or digital receipts, may take time, especially with customers who are used to traditional billing methods. Smooth training programs for staff and clear communication with customers will be essential to ensure a seamless transition.

**## 7. Loyalty Programs and Promotions: A Key Opportunity**

Integrating loyalty programs and personalized promotions within the billing system represents a significant opportunity. By collecting data on customer purchasing behavior, the system can help the supermarket offer personalized discounts, rewards, and product recommendations. This not only encourages repeat business but also increases customer satisfaction by creating a tailored shopping experience. Furthermore, loyalty programs can drive additional revenue by incentivizing customers to spend more to earn rewards.

**---**

This Supermarket Billing System aims to create a more efficient, secure, and customer-centric shopping experience by addressing the above challenges and opportunities, ultimately benefiting both the supermarket and its customers.

**Experiment-2**

**AIM-** Do requirement analysis and develop Software Requirement Specification Sheet (SRS) for suggested System.

# **Software Requirements Specification (SRS)**

**## Supermarket Billing System**

**### 1. Introduction**

**#### 1.1 Purpose**

This Software Requirements Specification (SRS) document provides a detailed outline of the requirements for developing a Supermarket Billing System. It describes the functional and non-functional requirements, system constraints, and interface requirements necessary for successful implementation.

**#### 1.2 Scope**

**The Supermarket Billing System will encompass:**

- Point of Sale (POS) operations

- Inventory management

- Customer relationship management

- Payment processing

- Loyalty program management

- Reporting and analytics

- Employee management

- Security and access control

**#### 1.3 Definitions, Acronyms, and Abbreviations**

- SBS: Supermarket Billing System

- POS: Point of Sale

- CRM: Customer Relationship Management

- SKU: Stock Keeping Unit

- GDPR: General Data Protection Regulation

- PCI-DSS: Payment Card Industry Data Security Standard

- API: Application Programming Interface

- GUI: Graphical User Interface

- DBMS: Database Management System

**#### 1.4 References**

- ISO/IEC 27001:2013 - Information Security Management Systems

- PCI-DSS v4.0 - Payment Card Industry Data Security Standard

- GDPR - General Data Protection Regulation

- IEEE Std 830-1998 - Software Requirements Specifications

- Retail industry standards and guidelines

**#### 1.5 Overview**

This document outlines the complete requirements for the Supermarket Billing System, including system architecture, interfaces, features, and performance specifications. It serves as the primary reference for development and testing teams.

**### 2. Overall Description**

**#### 2.1 Product Perspective**

The Supermarket Billing System is designed to operate as a comprehensive solution for retail operations management. It will integrate with existing hardware infrastructure and third-party services while providing modern features for efficient retail management.

**##### 2.1.1 System Interfaces**

- POS Terminal Interface

- Inventory Management System

- Payment Gateway Integration

- Customer Display Interface

- Admin Dashboard

- Mobile Application Interface

- Reporting System

**##### 2.1.2 User Interfaces**

**1. Cashier Interface:**

- Quick billing screen

- Product lookup

- Payment processing

- Receipt generation

**2. Admin Interface:**

- Inventory management

- Employee management

- Reports and analytics

- System configuration

**3. Customer Interface:**

- Self-checkout kiosk

- Loyalty program portal

- Digital receipt access

**##### 2.1.3 Hardware Interfaces**

- Barcode Scanners

- Receipt Printers

- Cash Drawers

- Card Payment Terminals

- Display Screens

- Weight Scales

- Mobile Devices

- Servers and Networking Equipment

**##### 2.1.4 Software Interfaces**

- Operating Systems: Windows, Linux

- Database: SQL Server/MySQL/PostgreSQL

- Payment Gateway APIs

- Inventory Management System

- Backup and Recovery Systems

- Security and Authentication Systems

**##### 2.1.5 Communication Interfaces**

- TCP/IP for network communication

- HTTPS for secure data transmission

- WebSocket for real-time updates

- Email/SMS gateway for notifications

- API endpoints for third-party integration

**#### 2.2 Product Functions**

**1. Billing Operations:**

- Barcode scanning and product identification

- Price calculation with tax

- Discount application

- Multiple payment method processing

- Receipt generation

**2. Inventory Management:**

- Real-time stock tracking

- Automatic reorder notifications

- Stock reconciliation

- Product categorization

- Supplier management

**3. Customer Management:**

- Customer profile management

- Loyalty program tracking

- Purchase history

- Personalized offers

- Feedback management

**4. Reporting and Analytics:**

- Sales reports

- Inventory reports

- Customer analytics

- Financial reports

- Performance metrics

**#### 2.3 User Characteristics**

**1. Cashiers:**

- Basic computer literacy

- Training in POS operations

- Understanding of retail processes

**2. Managers:**

- Advanced system knowledge

- Business operation experience

- Report analysis capabilities

**3. Administrators:**

- Technical expertise

- System configuration knowledge

- Security protocol understanding

**4. Customers:**

- Varied technical proficiency

- Basic understanding of self-checkout

- Mobile app usage capability

**#### 2.4 Constraints**

- Hardware compatibility requirements

- Network bandwidth limitations

- Security compliance requirements

- Data privacy regulations

- Integration with legacy systems

- Budget constraints

- Training requirements

**#### 2.5 Assumptions and Dependencies**

- Reliable internet connectivity

- Adequate hardware infrastructure

- Staff training completion

- Third-party service availability

- Regular system maintenance

- Backup power systems

**### 3. Specific Requirements**

**#### 3.1 External Interface Requirements**

**##### 3.1.1 User Interfaces**

**1. POS Interface Requirements:**

- Intuitive layout

- Touch screen optimization

- Quick access buttons

- Clear error messages

- Responsive design

**2. Admin Dashboard Requirements:**

- Customizable views

- Real-time updates

- Interactive reports

- Search functionality

- Filter options

**##### 3.1.2 Hardware Interfaces**

**1. Barcode Scanner Requirements:**

- Multiple format support

- Quick response time

- Error detection

- Wireless connectivity option

**2. Payment Terminal Requirements:**

- Multiple card support

- Contactless payment

- Security encryption

- Receipt printing capability

**#### 3.2 Functional Requirements**

**##### 3.2.1 Billing Operations**

**1. Product Scanning:**

- Instant barcode recognition

- Manual entry option

- Quantity modification

- Price verification

**2. Payment Processing:**

- Multiple payment methods

- Split payment support

- Change calculation

- Receipt generation

- Refund processing

**##### 3.2.2 Inventory Management**

**1. Stock Tracking:**

- Real-time updates

- Low stock alerts

- Automatic reordering

- Stock reconciliation

- Supplier management

**2. Product Management:**

- Category organization

- Price updates

- Discount management

- Product information

- Image management

**#### 3.3 Performance Requirements**

- Transaction processing time < 2 seconds

- System availability > 99.9%

- Concurrent user support > 100

- Data backup frequency: Every 24 hours

- Report generation time < 30 seconds

**#### 3.4 Security Requirements**

**1. Authentication:**

- Role-based access control

- Multi-factor authentication

- Password policies

- Session management

**2. Data Protection:**

- Encryption at rest

- Secure transmission

- Audit logging

- Backup encryption

**#### 3.5 Software Quality Attributes**

**1. Reliability:**

- Error handling

- Data consistency

- Backup systems

- Recovery procedures

**2. Availability:**

- 24/7 operation

- Failover capability

- Maintenance windows

- Disaster recovery

**3. Maintainability:**

- Modular design

- Documentation

- Version control

- Update management

**4. Scalability:**

- Horizontal scaling

- Load balancing

- Resource optimization

- Performance monitoring

**### 4. Testing Requirements**

**#### 4.1 Unit Testing**

- Component level testing

- Function verification

- Error handling

- Performance validation

**#### 4.2 Integration Testing**

- Module interaction testing

- Interface testing

- Data flow verification

- System integration

**#### 4.3 System Testing**

- End-to-end testing

- Load testing

- Security testing

- User acceptance testing

**### 5. Implementation Requirements**

**#### 5.1 Development Environment**

- Development tools

- Testing frameworks

- Version control

- Documentation tools

**#### 5.2 Deployment Requirements**

- Installation procedures

- Configuration management

- Training requirements

- Support documentation

**### 6. Change Management Process**

- Change request procedures

- Impact analysis

- Approval workflow

- Implementation planning

- Testing requirements

- Documentation updates

**### 7. Document Approvals**

- Project Manager

- System Architect

- Development Lead

- Quality Assurance Lead

- Security Officer

- Business Stakeholders

**### 8. Supporting Information**

- System architecture diagrams

- Data flow diagrams

- User manuals

- Training materials

- Technical documentation

- Compliance certificates

**Experiment-3**

**AIM-** To perform the user’s view analysis for the suggested system: Use Case Diagram

**Use Case Diagram: Overview**

A use case diagram is a type of Unified Modeling Language (UML) diagram that describes the behavior of a system from the user's perspective. It shows the interaction between the system and users called "actors" and the different ways users can interact with the system. The diagram includes actors. Use cases and the relationship between those things Actors are represented by stick figures. Use cases with ellipses and relationships with lines that connect actors and use case.

**Use Case Diagram: Advantages**

There are several advantages of using Use Case Diagrams:

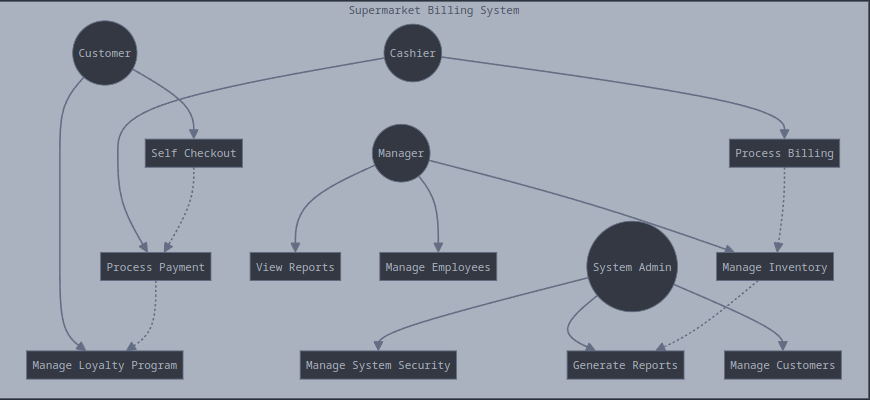
* **Improved communication**: Use Case Diagrams provide a common language and understanding of the system's functionality among stakeholders, including developers, users, and customers.
* **Identify functional requirements**: Use Case Diagrams help identify the functional requirements of the system, which is essential for developing a system that meets the user's needs.
* **Simplify complex systems**: Use Case Diagrams can simplify complex systems by breaking them down into smaller, more manageable pieces.
* **Enhance user experience**: By focusing on the user's perspective, Use Case Diagrams help ensure that the system is designed with the user's needs in mind, leading to a better user experience.
* **Reduce errors**: Use Case Diagrams can help reduce errors by identifying potential issues early in the development process.

**Use Case Diagram: Why use them in Software Engineering**

We use Use Case Diagrams in software engineering for several reasons:

* **Requirements gathering:** Use Case Diagrams are an effective tool for gathering requirements from stakeholders and users.
* **System design:** Use Case Diagrams help designers and developers understand the system's functionality and how it will be used.
* **Testing and validation:** Use Case Diagrams provide a basis for testing and validating the system's functionality.
* **Communication:** Use Case Diagrams facilitate communication among stakeholders, including developers, users, and customers.
* **Iterative development:** Use Case Diagrams support iterative development by allowing developers to refine the system's functionality based on user feedback.

**Use Case Diagram:**



Shivam Kumar Jha  
C2  
02614802722

**Experiment-4**

**AIM-** Draw the entity relationship diagram for the suggested system

**Entity Relationship Diagram: Overview**

An Entity-Relationship (ER) Diagram is a type of diagram that represents the structure of a database or a system by showing the relationships between entities, attributes, and relationships. ER diagrams are used to model the data and its relationships in a database, and they are an essential tool in database design. The diagram consists of entities (represented by rectangles), attributes (represented by ovals), and relationships (represented by lines connecting the entities).

**Entity Relationship Diagram: Advantages**

There are several advantages of using ER Diagrams:

* **Improved data modeling**: ER Diagrams provide a clear and concise way to model the data and its relationships in a database.
* **Better database design**: ER Diagrams help designers create a well-structured database that is easy to maintain and modify.
* **Reduced data redundancy**: ER Diagrams help identify and eliminate data redundancy, which can lead to data inconsistencies and errors.
* **Improved data integrity**: ER Diagrams help ensure data integrity by defining the relationships between entities and attributes.
* **Enhanced scalability**: ER Diagrams make it easier to scale the database as the system grows and evolves.

**Entity Relationship Diagram: Why use them in Software Engineering**

We use ER Diagrams in software engineering for several reasons:

* **Database design**: ER Diagrams are an essential tool in database design, as they help designers create a well-structured database that meets the system's requirements.
* **Data modeling**: ER Diagrams provide a clear and concise way to model the data and its relationships in a database
* **System analysis**: ER Diagrams help analysts understand the system's data and its relationships, which is essential for developing a system that meets the user's needs.
* **Communication**: ER Diagrams facilitate communication among stakeholders, including designers, developers, and users.
* **Data normalization**: ER Diagrams help designers normalize the data, which is essential for ensuring data integrity and reducing data redundancy.

**Entity Relationship Diagram:**

erDiagram

PRODUCT {

int product\_id PK

string barcode

string name

decimal price

int category\_id FK

int supplier\_id FK

int current\_stock

int reorder\_level

datetime created\_at

datetime updated\_at

}

CATEGORY {

int category\_id PK

string name

string description

datetime created\_at

}

SUPPLIER {

int supplier\_id PK

string name

string contact\_person

string phone

string email

string address

datetime created\_at

}

BILL {

int bill\_id PK

int customer\_id FK

int cashier\_id FK

decimal total\_amount

decimal tax\_amount

decimal discount\_amount

decimal final\_amount

datetime bill\_date

string payment\_method

string status

}

BILL\_ITEM {

int bill\_item\_id PK

int bill\_id FK

int product\_id FK

int quantity

decimal unit\_price

decimal total\_price

decimal discount

}

CUSTOMER {

int customer\_id PK

string name

string phone

string email

datetime registration\_date

int loyalty\_points

string status

}

EMPLOYEE {

int employee\_id PK

string name

string role

string username

string password\_hash

string email

string phone

datetime join\_date

string status

}

INVENTORY\_LOG {

int log\_id PK

int product\_id FK

string action\_type

int quantity

datetime log\_date

int employee\_id FK

string remarks

}

PAYMENT {

int payment\_id PK

int bill\_id FK

string payment\_method

decimal amount

string transaction\_id

datetime payment\_date

string status

}

PRODUCT ||--|| CATEGORY : "belongs\_to"

PRODUCT ||--|| SUPPLIER : "supplied\_by"

BILL ||--|{ BILL\_ITEM : "contains"

BILL ||--|| CUSTOMER : "belongs\_to"

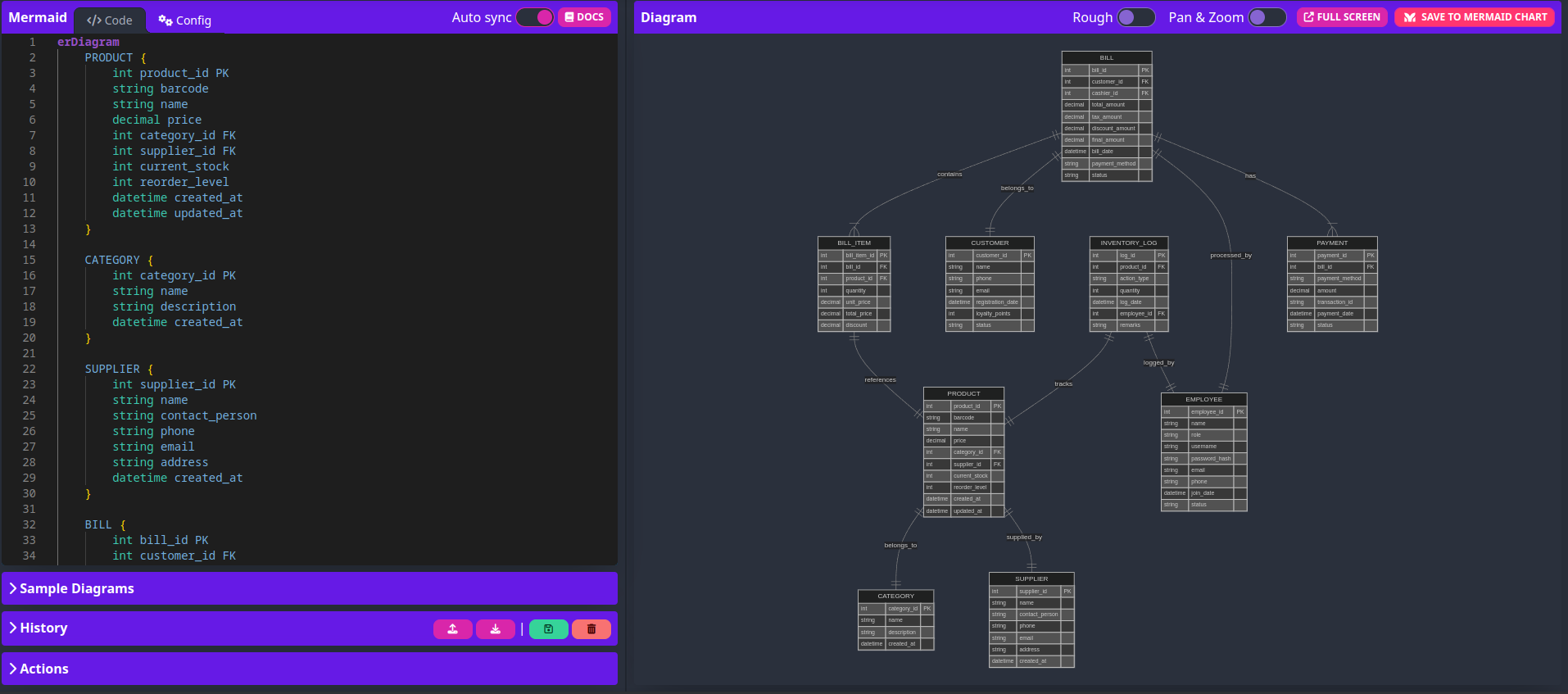
BILL ||--|| EMPLOYEE : "processed\_by"

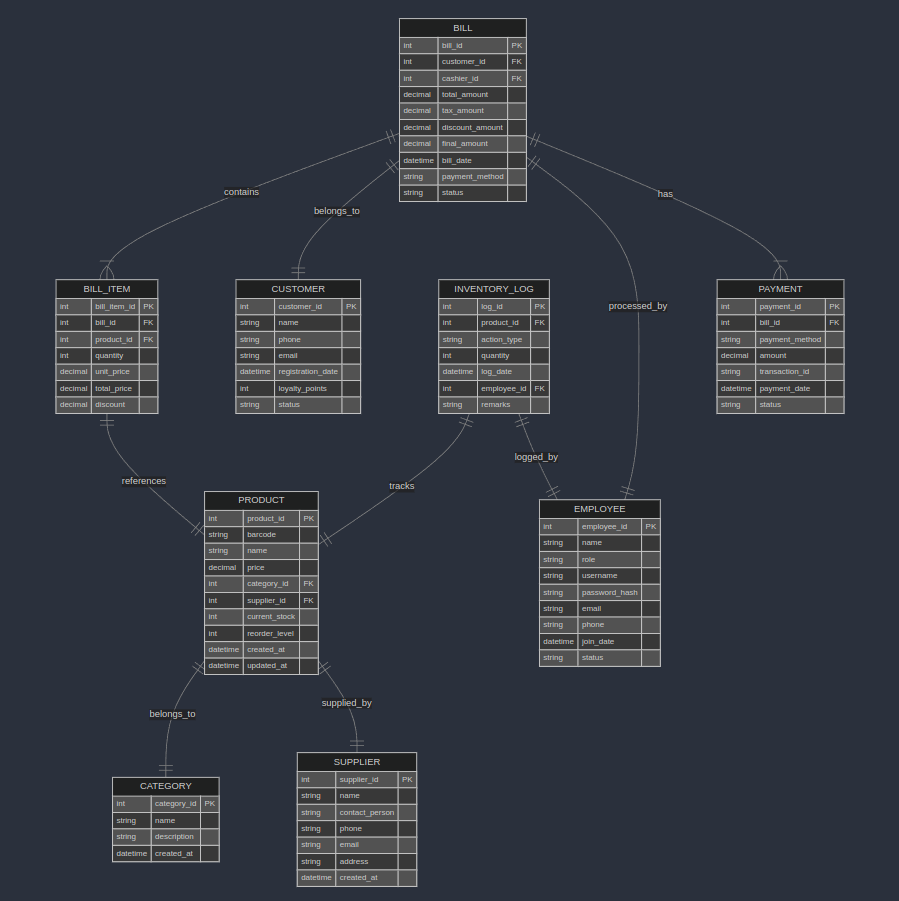
BILL ||--|{ PAYMENT : "has"

BILL\_ITEM ||--|| PRODUCT : "references"

INVENTORY\_LOG ||--|| PRODUCT : "tracks"

INVENTORY\_LOG ||--|| EMPLOYEE : "logged\_by"





**Experiment-5**

**AIM-** Draw the entity Data Flow Diagram for the suggested system

**Data Flow Diagram: Overview**

A Data Flow Diagram (DFD) is a visual tool that shows how data moves through an information system. It uses symbols to represent external entities, processes, data stores, and data flows. DFDs help analyze, design, and document systems by illustrating the flow and transformation of data. They range from high-level context diagrams to detailed lower-level diagrams, providing insights into system functionality and data handling.

**Data Flow Diagram: Advantages**

There are several advantages of using Data Flow Diagrams:

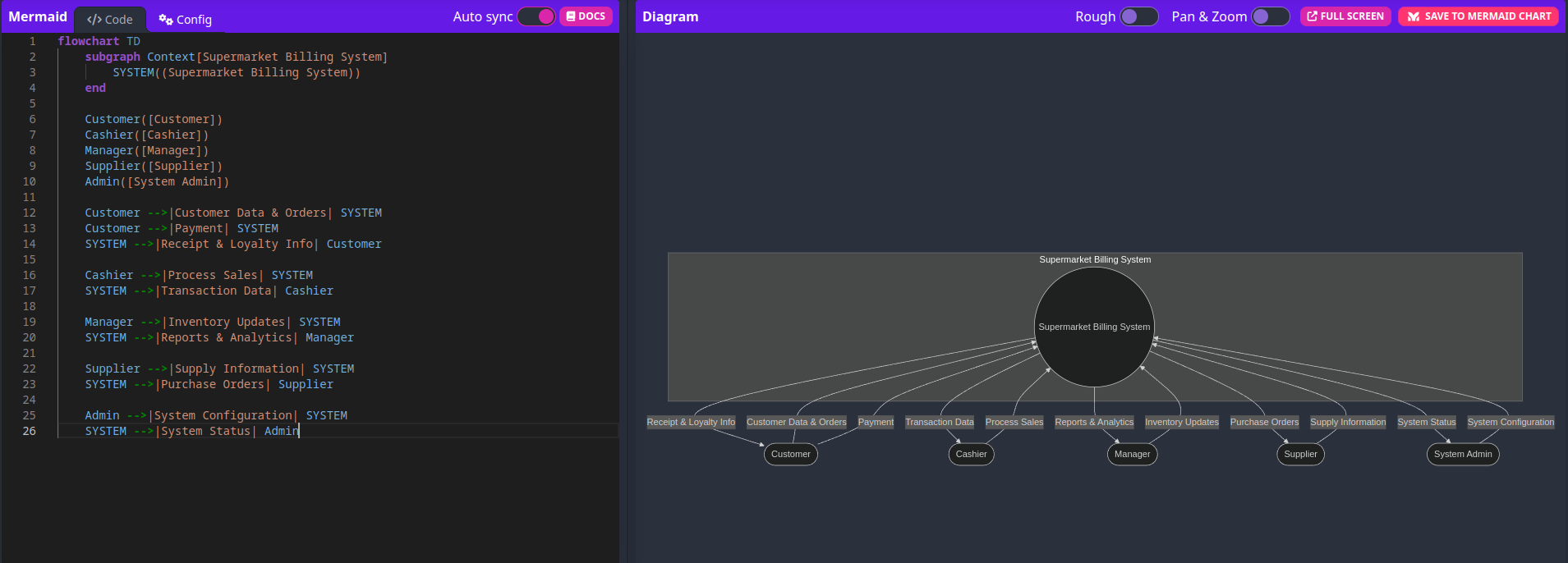
* Clear visualization of data processes and flows
* Improved system understanding for both technical and non-technical stakeholders
* Identification of inefficiencies, redundancies, or missing elements in a system
* Facilitation of communication between developers, analysts, and users
* Scalability from high-level overviews to detailed process breakdowns

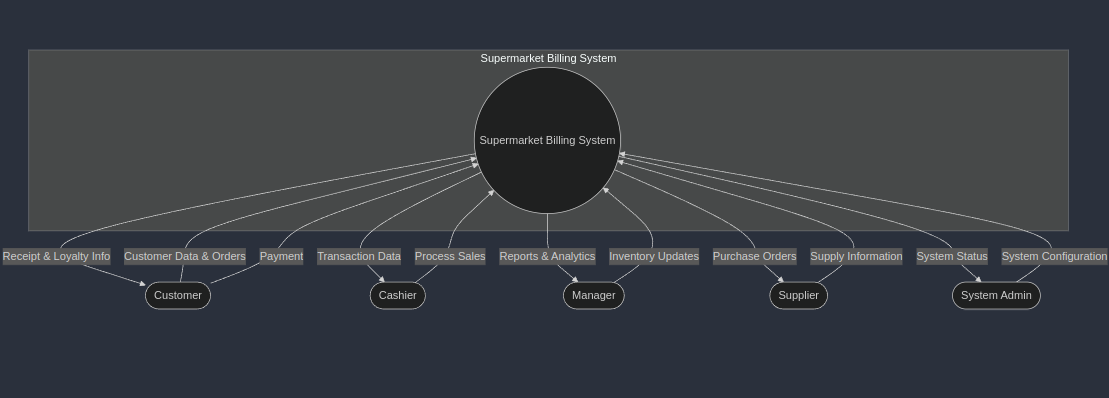
**Data Flow Diagram: Why use them in Software Engineering**

We use Data Flow Diagrams in software engineering for several reasons:

* **System visualization:** They provide a clear, graphical representation of how data moves through a system, making complex processes easier to understand.
* **Requirements analysis:** DFDs help identify and clarify system requirements by showing data inputs, processes, and outputs.
* **Communication:** They serve as an effective communication tool between developers, stakeholders, and non-technical team members.
* **Design aid:** DFDs assist in designing system architecture by highlighting data interactions and process dependencies.
* **Problem identification:** They can reveal inefficiencies, redundancies, or gaps in data flow, helping to optimize system design.

**Data Flow Diagram(Level 0):**





**Level 0 (Context) DFD:**

1. Shows the system as a single process

2. Identifies all external entities:

- Customer (makes purchases, provides data)

- Cashier (processes sales)

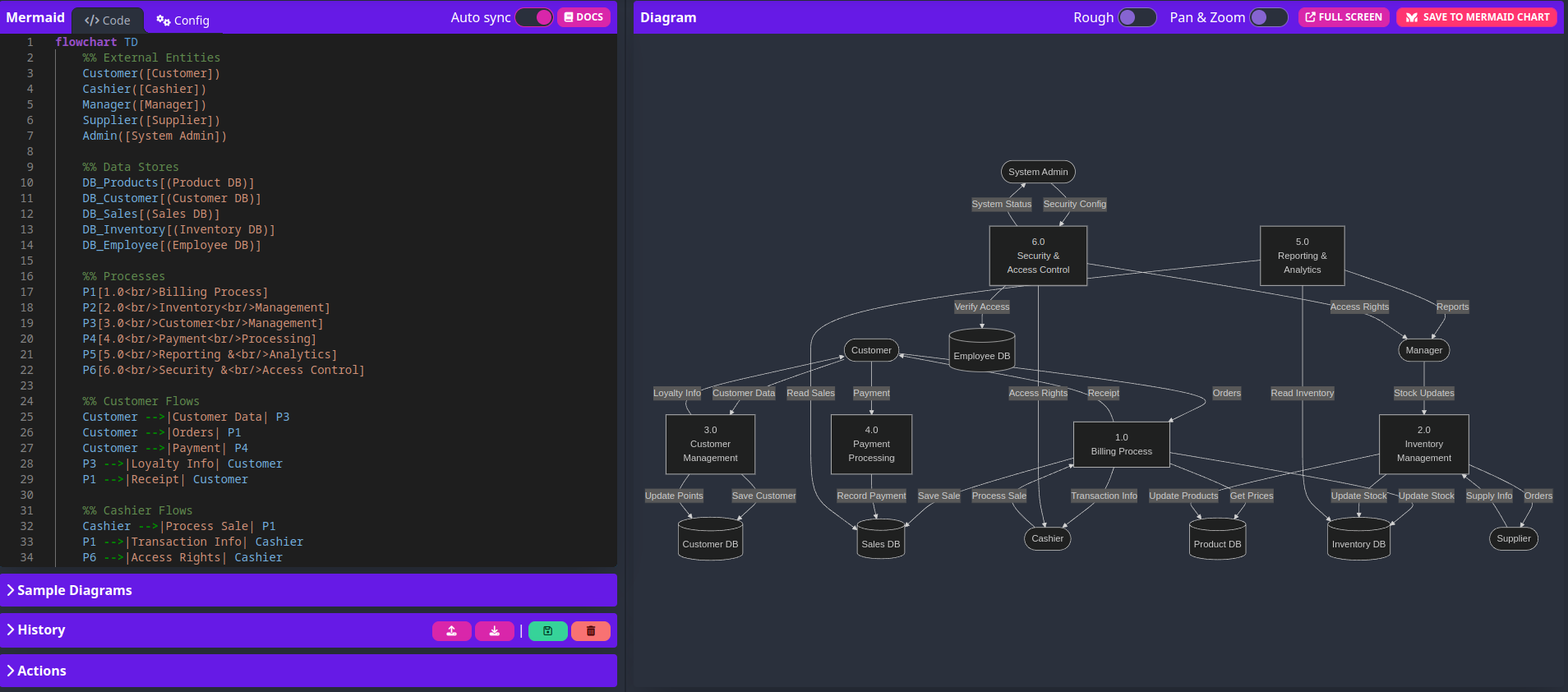
- Manager (handles inventory)

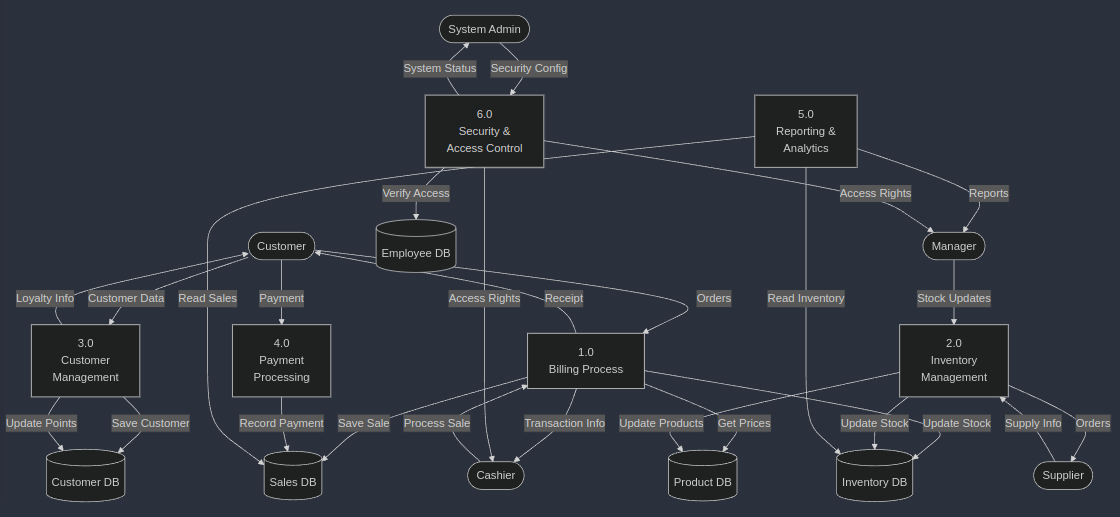
- Supplier (provides products)

- System Admin (manages system)

3. Shows basic information flows in and out of the system

**Data Flow Diagram(Level 1):**





Level 1 DFD:

1. Main Processes:

- 1.0 Billing Process

- 2.0 Inventory Management

- 3.0 Customer Management

- 4.0 Payment Processing

- 5.0 Reporting & Analytics

- 6.0 Security & Access Control

2. Data Stores:

- Product DB

- Customer DB

- Sales DB

- Inventory DB

- Employee DB

3. Key Data Flows:

- Customer interactions (orders, payments, receipts)

- Inventory updates and tracking

- Sales and transaction processing

- Security and access control

- Reporting and analytics

The Level 1 diagram shows how the processes interact with:

- External entities (shown at the edges)

- Each other (through data flows)

- Data stores (databases)

**Experiment-6**

**AIM: -** To draw the structural view diagram for the system: Class diagram, object diagram

**Description: -**

**Class Diagram**

This is a fundamental element in object-oriented software design. It illustrates the static structure of a system by displaying its classes, attributes, methods, and the relationships between them.

**Key Components of a Class Diagram**:

1. **Class**: The basic unit, represented as a rectangle divided into three sections. The top section shows the class name, the middle lists its attributes, and the bottom contains its methods.
2. **Attributes**: Characteristics or properties of a class (e.g., PassengerID, Name for the Passenger class).
3. **Methods**: Actions or behaviors a class can perform (e.g., reserveTicket(), cancelReservation() for the Reservation class).
4. **Relationships**:
   * **Association**: Defines relationships between classes, like how Passenger is associated with Reservation.
   * **Multiplicity**: Indicates how many instances of one class relate to another.
   * **Inheritance**: Represents a class deriving from another class.
   * **Aggregation/Composition**: Depicts "whole-part" relationships, where one class is made up of other classes.

**Class Diagram uses:**

* It is utilized during the design phase to define the system's structure.
* Helps developers grasp the overall organization of the system.
* Aids in communication between team members, ensuring shared understanding of the system’s architecture.

Object Diagram

This is a snapshot of objects in a system at a given point in time. It displays instances of the classes defined in the class diagram and the relationships between them.

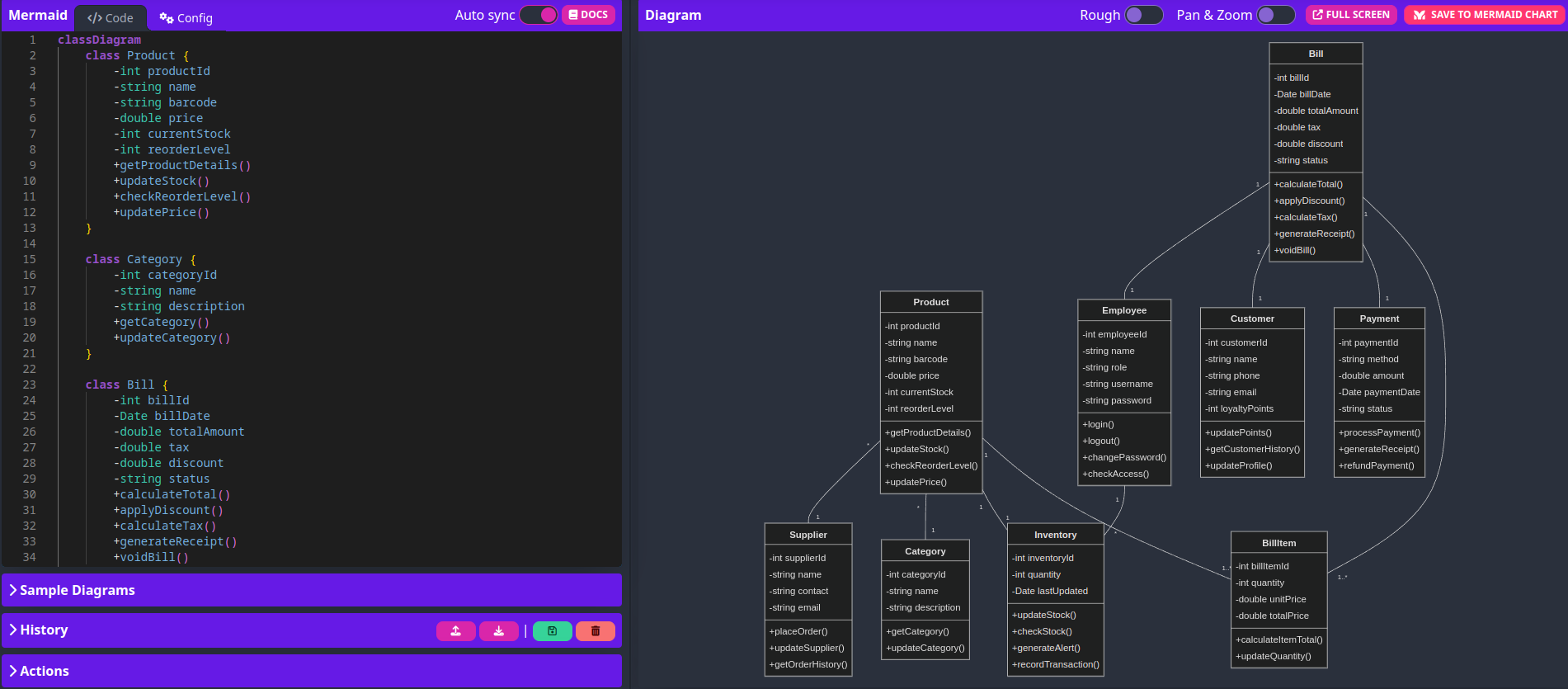
**Key Components of an Object Diagram**:

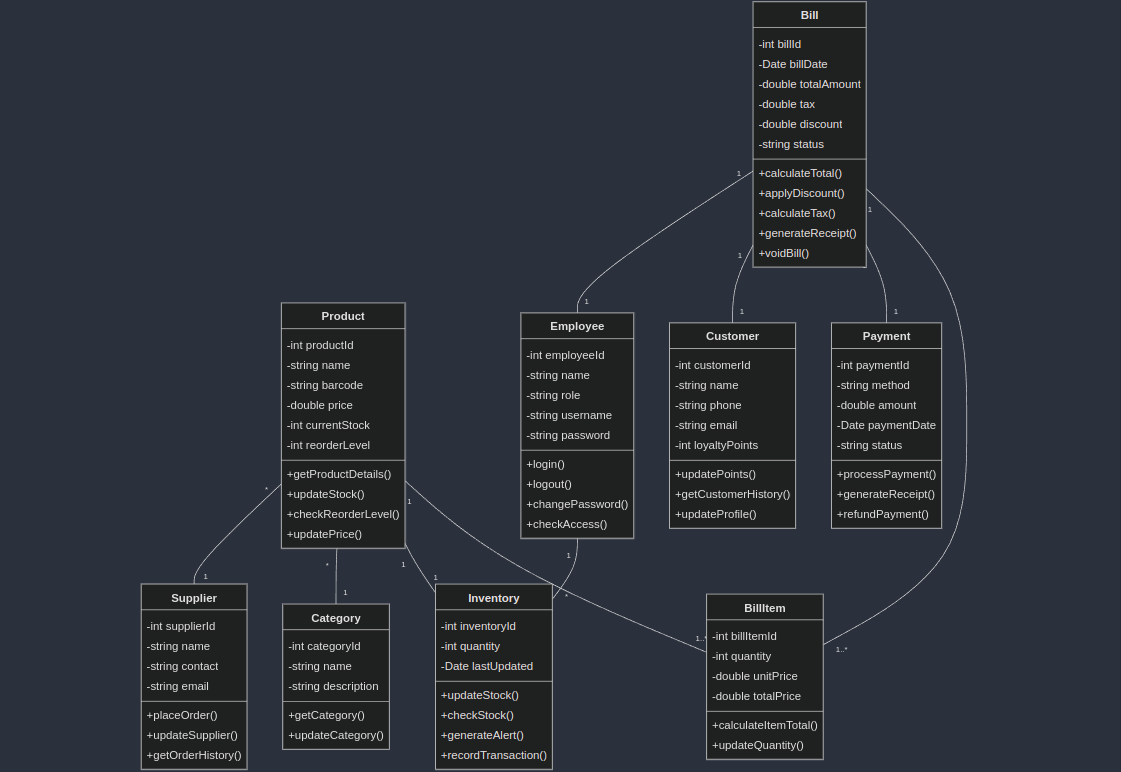
1. **Object**: A specific instance of a class, with assigned attribute values.
2. **Links**: Represents relationships between objects, akin to associations in a class diagram.
3. **Attributes with Values**: Unlike the class diagram, where attributes are only listed, an object diagram specifies their actual values.

**Object Diagram uses**:

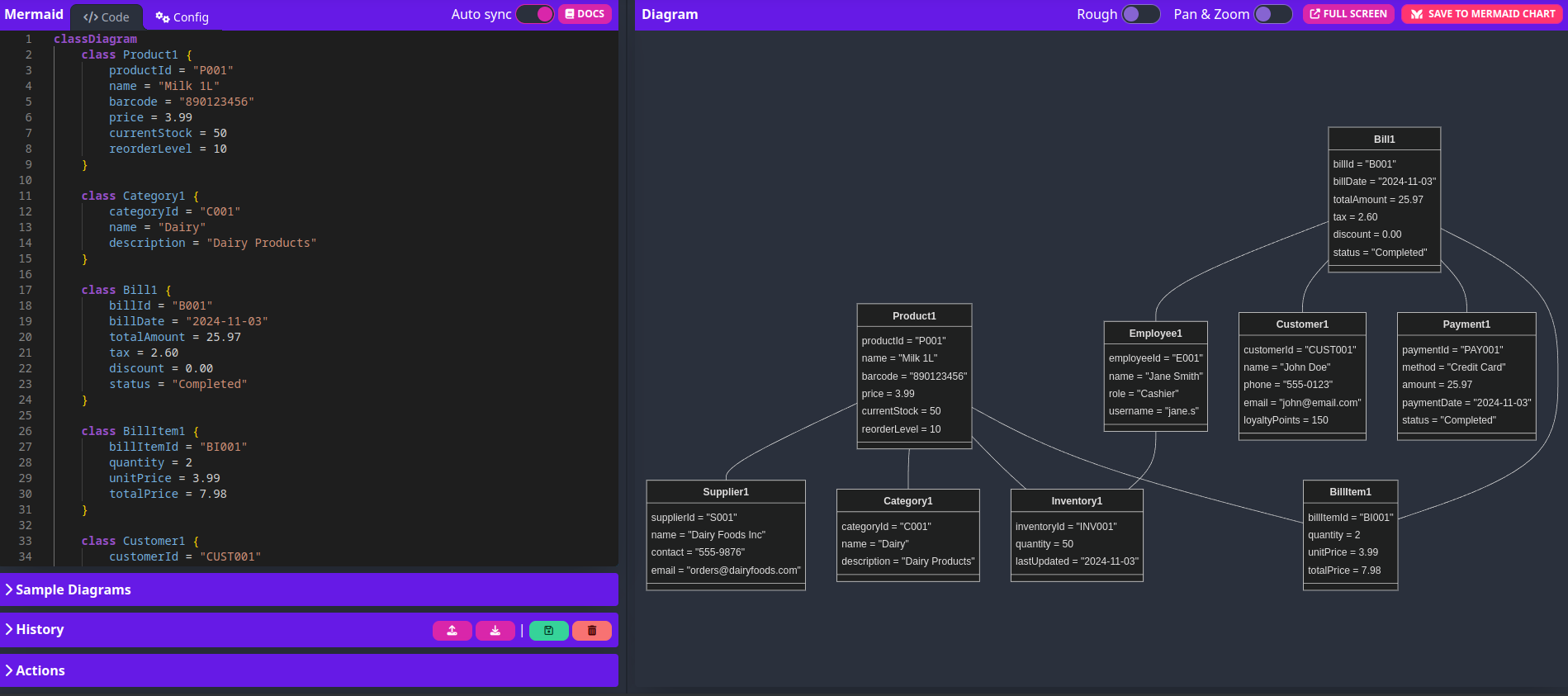
* Useful for illustrating a specific scenario or test case.
* Helps with debugging and validating relationships between instances.
* Offers a clear representation of system behavior at a particular moment in time.

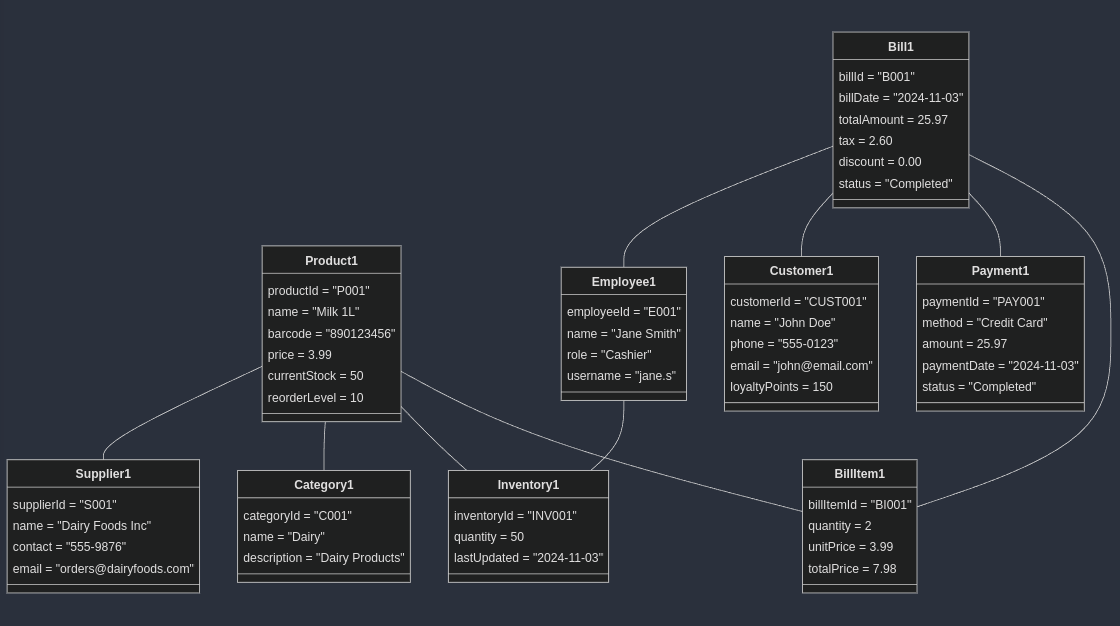
Class Diagram:





Object Diagram:





**Experiment 7**

**AIM: -** To draw the behavioral view diagram: State-chart diagram, Activity diagram.

**Description: -**

**State Chart Diagram**

A State Chart Diagram models the various states an object can occupy in a system and how it transitions from one state to another. This diagram is especially useful in systems where objects undergo changes over time in response to events or inputs.

**Key Components:**

**1. States (represented by rectangles):**

- Idle

- ScanningItems

- CalculatingTotal

- ProcessingPayment

- PrintingReceipt

- UpdatingInventory

**2. Transitions (arrows with labels):**

- System Start

- New Bill Started

- Cancel Transaction

- Scanning Complete

- Modify Items

- Total Confirmed

- Payment Failed/Successful

- Receipt Printed

**3. Annotations/Notes:**

- Updates running total

- Validates product

- Checks inventory

- Payment options (Cash, Card, Digital Payment)

**Overview:**

The state chart diagram illustrates the different states a POS (Point of Sale) system goes through during a transaction. It starts from an idle state, moves through scanning and calculation phases, handles payment processing, and ends with receipt printing and inventory updates. The diagram includes error handling (like payment failures) and allows for transaction cancellation at various points.

**Activity Diagram**

An Activity Diagram is a flowchart-like representation outlining the sequence of activities or tasks within a system. It captures the dynamic behavior of a system by showing the flow of control from one activity to another.

**Key Components:**

**1. Activities (rectangles):**

- Start New Bill

- Scan Item

- Add to Cart

- Calculate Total

- Apply Discounts

- Select Payment Method

- Print Receipt

- Update Inventory

- Update Loyalty Points

**2. Decision Nodes (diamonds):**

- Valid Item?

- More Items?

- Process Payment?

**3. Control Flow:**

- Start/End nodes

- Decision branches (Yes/No paths)

- Sequential flow indicators

**Overview:**

The activity diagram shows the step-by-step workflow of a retail transaction process. It begins with starting a new bill, follows through item scanning and validation, handles multiple item additions, processes payment, and concludes with receipt printing and system updates. The diagram includes error handling paths and shows the logical flow of activities required to complete a transaction, including the final steps of updating inventory and loyalty points.

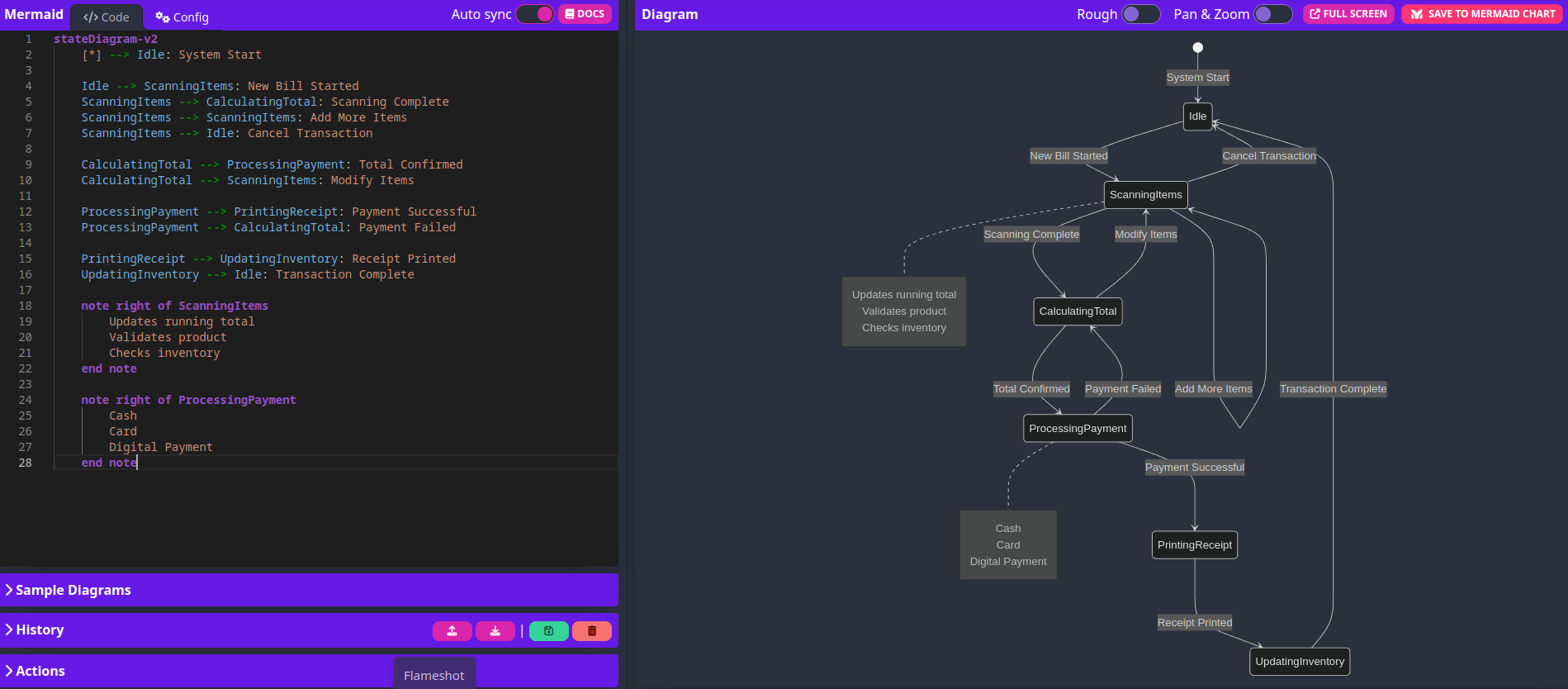
Both diagrams represent the same system from different perspectives:

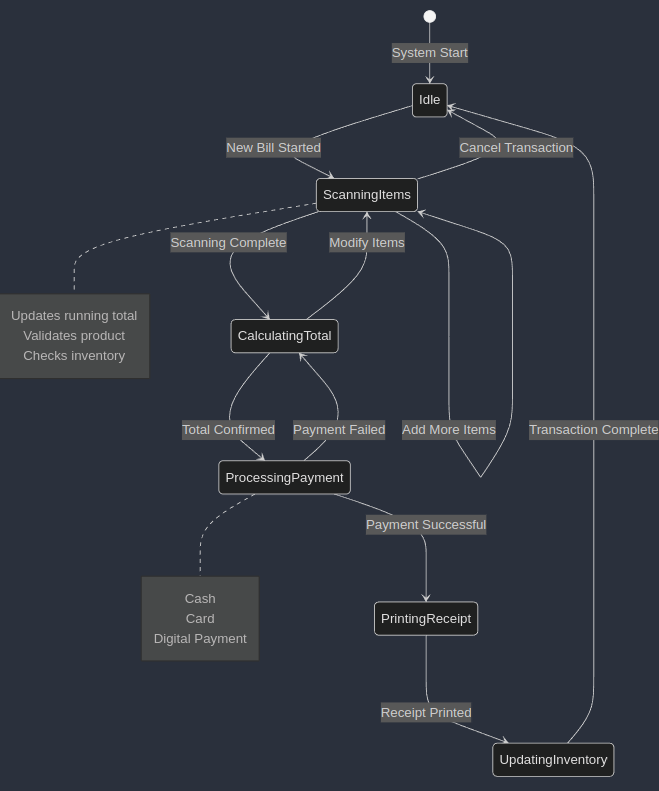
- The state chart focuses on the system's states and transitions between them

- The activity diagram emphasizes the sequential flow of actions and decisions in the process

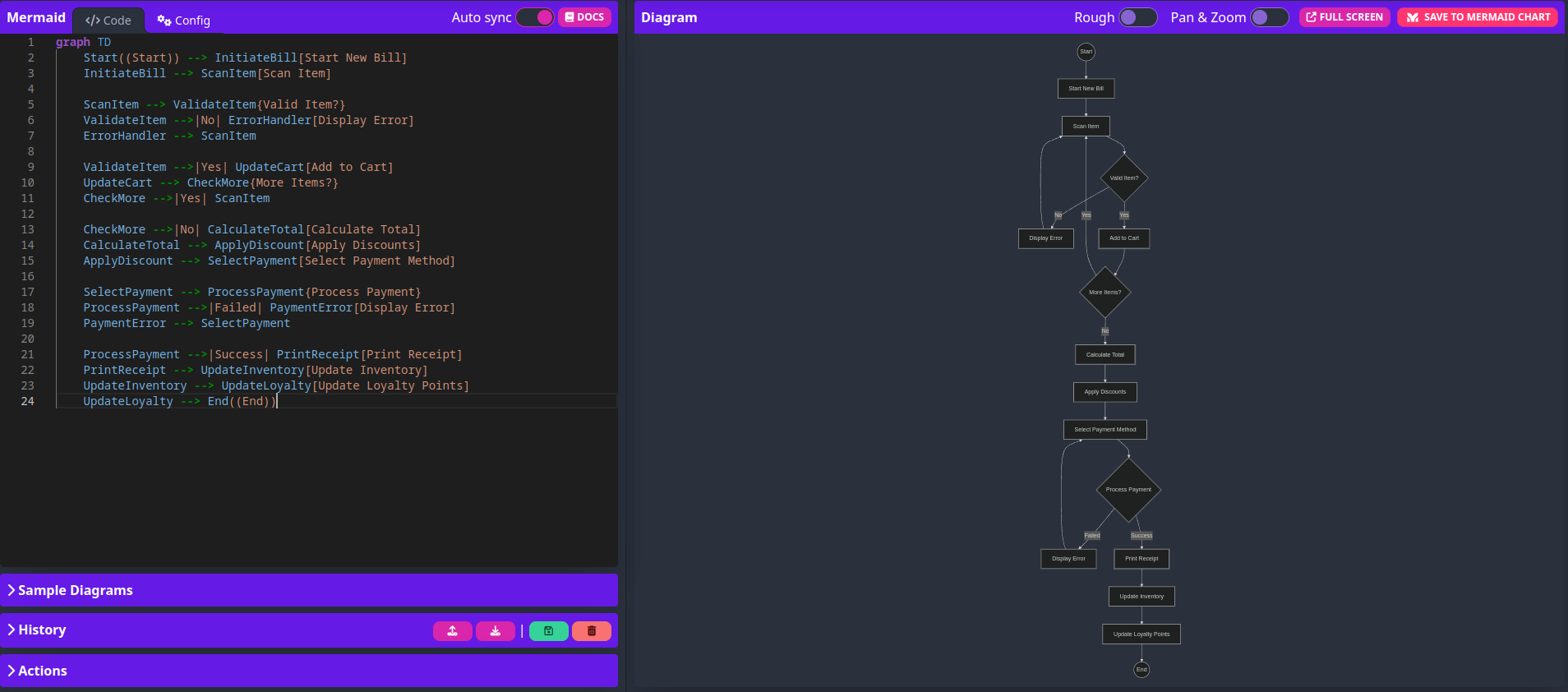
These complementary views help in understanding both the system's states and the detailed workflow of the transaction process.

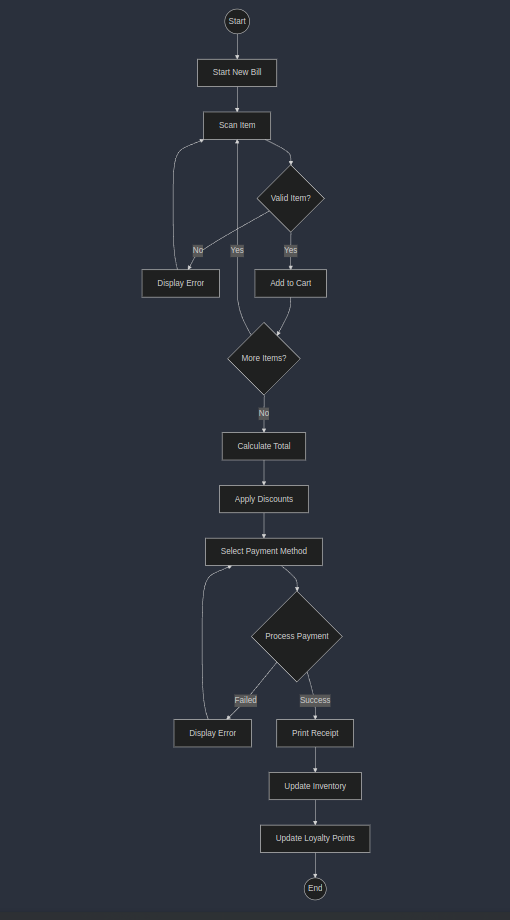
State-Chart Diagram:





Activity Diagram



**Experiment 8**

**AIM: -** To perform the implementation view diagram: Component diagram for the system.

**Description: -**

**Component Diagram for Supermarket Billing System(SBS)**

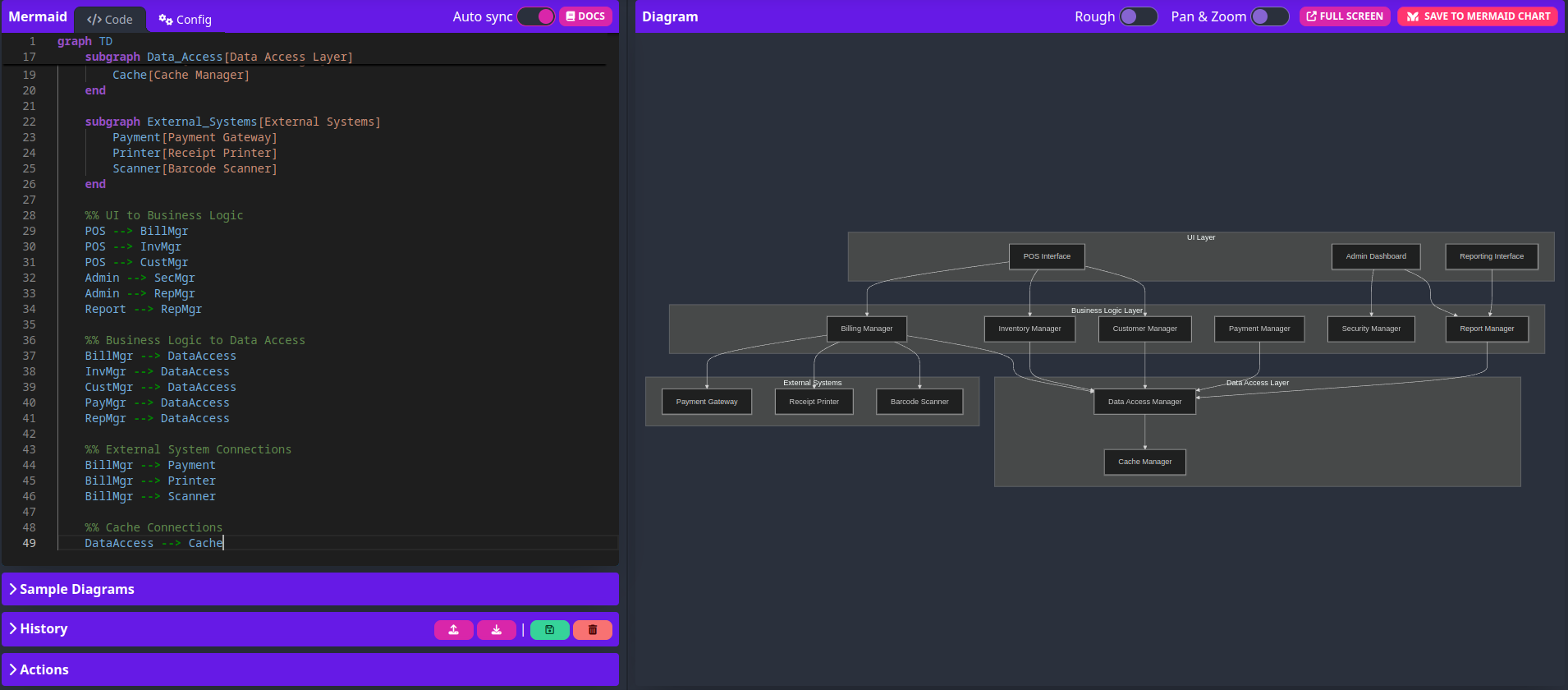
A Component Diagram provides a visual representation of the components and interfaces within the Supermarket Billing System(SBS). It depicts the system's architecture by showing how different modules interact with one another and the external interfaces they rely upon. This diagram is essential for understanding the system's modular structure and how various functionalities are encapsulated within specific components.

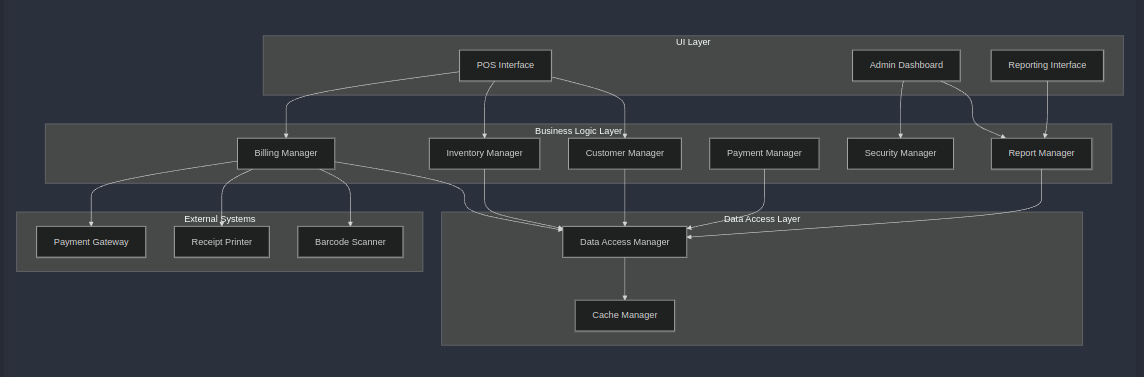
The primary purpose of the component diagram is to offer a well-organized and structured view of the system’s components, emphasizing their relationships and dependencies. It serves as a blueprint for implementation, helping developers visualize how modules interact and ensuring that all required functionalities are properly integrated.

**Significance of the Component Diagram:**

* Modularity: Ensures that each functionality is contained within its respective module, promoting a separation of concerns, which simplifies system design and future modifications.
* Reusability: Facilitates the reuse of interfaces across different modules, enabling easier integration and reducing redundancy within the system.
* Clarity: Provides a clear overview of the system architecture, aiding stakeholders in understanding the interactions between system components and ensuring a unified vision of how the system operates.

Component diagram:



****

**Experiment 9**

**AIM: -** To perform the environmental view diagram: Deployment diagram for the system.

**Description: -**

**Deployment Diagram**

A Deployment Diagram is a type of UML diagram that represents the physical deployment of software components (artifacts) on hardware or virtual environments (nodes). It shows how different components of a system are distributed across various nodes and interact within the system architecture.

**Key Components of a Deployment Diagram:**

**1. Store Frontend:**

- Barcode Scanner: For product identification

- Receipt Printer: For printing customer receipts

- POS Terminal: Main point-of-sale interface

- Payment Terminal: For processing card payments

**2. Network Infrastructure:**

- Network Switch: Central connection point

- Store Router: Manages network traffic

- Security Firewall: Protects against unauthorized access

**3. Store Server Room:**

- Application Server: Runs the main billing application

- Database Server: Stores all system data

- Cache Server: For improved performance

- Backup Server: Local data backup solution

**4. Cloud Infrastructure:**

- Load Balancer: Distributes network traffic

- Analytics Server: For business intelligence

- Cloud Backup: Remote backup solution

**Key Relationships:**

- Store Frontend components connect to Network Infrastructure

- Network Infrastructure manages communication between all components

- Store Server Room handles local processing and storage

- Cloud Infrastructure provides additional services and backup

**This architecture ensures:**

- High availability through redundancy

- Scalability through cloud integration

- Data security through multiple backup solutions

- Performance through caching and load balancing

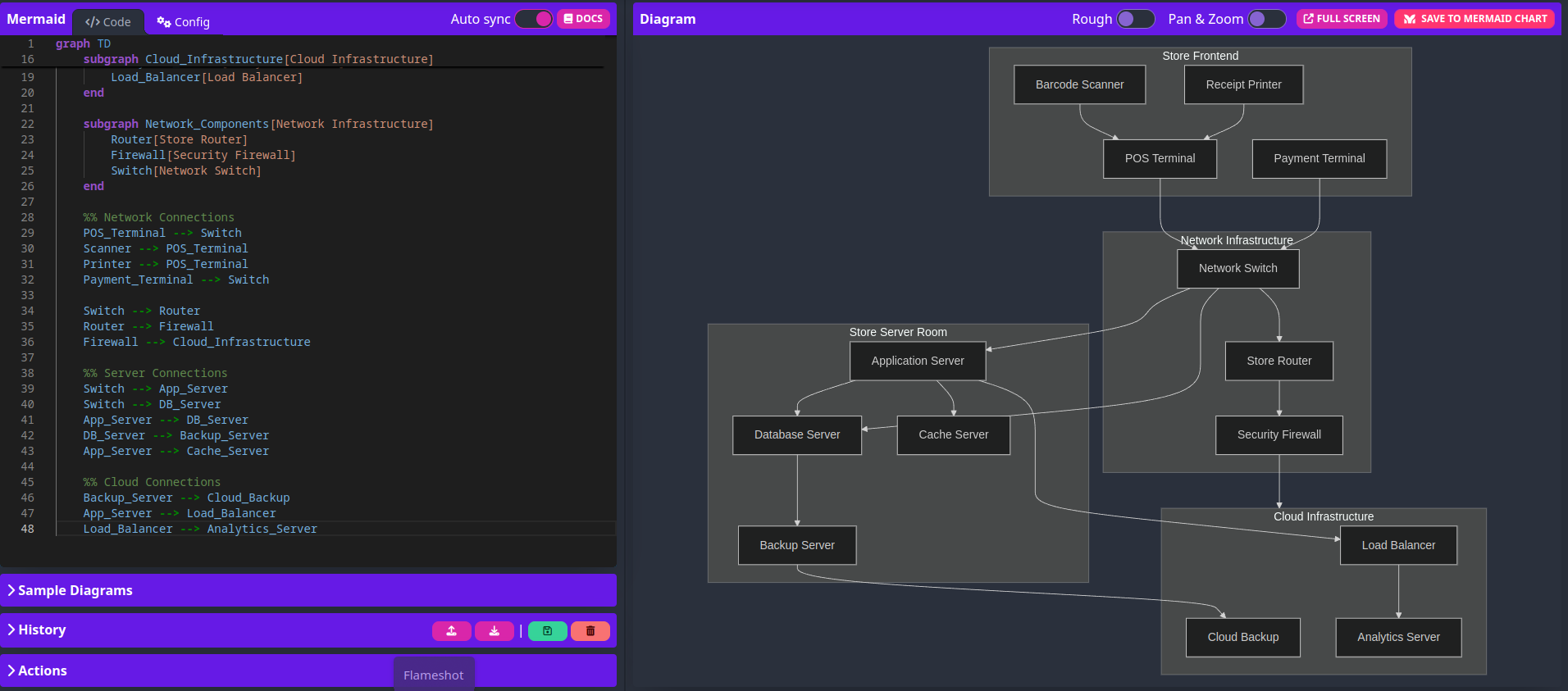
- Security through firewalls and network segregation

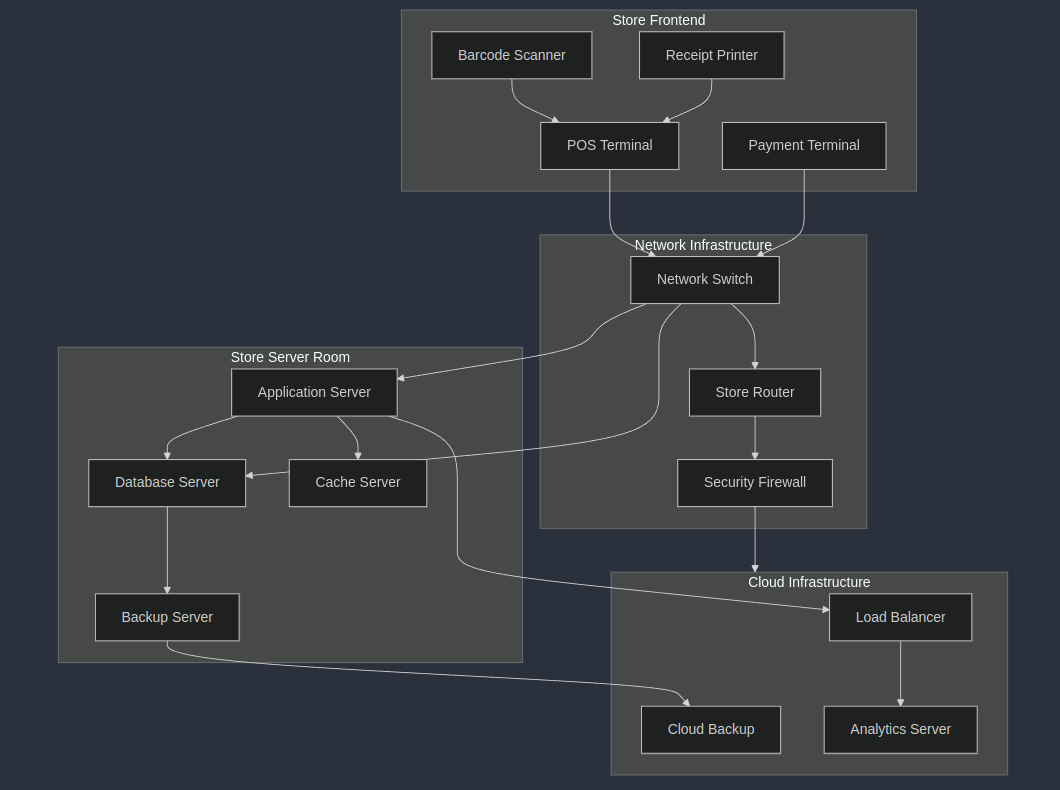
**Purpose of a Deployment Diagram:**

* Visualizing System Architecture: Illustrates how software components are distributed across hardware nodes and how they interact.
* Understanding System Requirements: Helps map components to specific hardware, enabling a clearer understanding of network infrastructure and hardware needs.
* Facilitating Deployment Planning: Assists in planning and executing the deployment process by providing a clear representation of where each component resides and how they communicate.
* Identifying Bottlenecks and Points of Failure: Highlights potential performance bottlenecks or critical points that could affect system reliability, leading to better design decisions.

In essence, a Deployment Diagram is vital for visualizing the physical deployment of a system’s components. It helps enhance understanding of the system’s architecture and aids in deployment planning. By mapping out relationships between nodes and artifacts, it ensures the system is optimized for performance, reliability, and scalability. This diagram is especially critical in complex environments like web applications, cloud services, or multi-tier architectures.

Deployment Diagram:





**Experiment 10**

**AIM: -** To perform various testing using the testing tool unit testing, integration testing for a sample code of the suggested system.

**Description:**

### Introduction to Software Testing for Supermarket Billing System

In the development of complex systems like the **Supermarket Billing System**, software testing is crucial to ensure accuracy, reliability, and quality. Testing guarantees that each component and the entire system function as expected. In this project, we use two main testing methodologies:

* **Unit Testing**: Tests individual components or functions independently to confirm that each performs correctly.
* **Integration Testing**: Verifies that different modules work together seamlessly to achieve the desired functionality.

Together, these testing methods ensure that the billing system operates reliably and integrates smoothly into the supermarket environment.

### Unit Testing

Unit testing involves testing specific components or units of code in isolation to ensure they perform correctly under various conditions. In the **Supermarket Billing System**, unit testing will include functions such as calculating the total price, applying discounts, and managing inventory.

#### Importance of Unit Testing:

* **Early Bug Detection**: Catches bugs within individual components early, reducing the cost and time required to resolve issues later.
* **Improved Code Quality**: Enhances code quality by thoroughly checking each function or module.
* **Refactoring**: Allows safe code refactoring by verifying that functionality remains intact after changes.
* **Modularity**: Ensures each module functions independently, supporting a modular design for easy maintenance.

### Integration Testing

Integration testing evaluates how different modules work together. After unit testing, modules are combined, and integration testing ensures that interactions function as intended. For the **Supermarket Billing System**, integration testing might include verifying that the discount calculation module interacts correctly with the billing module, and that the inventory system updates accurately with each purchase.

#### Importance of Integration Testing:

* **Interface Testing**: Helps detect errors in the way modules communicate.
* **System Reliability**: Ensures smooth operation of the system as a whole, verifying data and control flow between modules.
* **Dependency Testing**: Confirms that changes in one module do not negatively impact others.

### Tools for Unit and Integration Testing in C++

Several tools facilitate unit and integration testing in C++. These tools provide frameworks to write and validate test cases for code behavior.

* **Google Test (gTest)**: An open-source C++ testing framework supporting unit and integration testing, offering features like test fixtures, assertions, and test runners.
* **Boost.Test**: A part of the Boost C++ libraries, it supports comprehensive unit testing, with options for test case management and reporting.
* **CppUnit**: Another popular framework, providing extensive support for both unit and integration testing.

### Sample C++ Code with Unit Testing

Here’s a sample class for a supermarket billing system that includes functions for calculating the total price and applying discounts. The following unit tests ensure that these functions produce correct results.

**Supermarket Billing System Code (Billing class):**

***class Billing {***

***public:***

***double calculateTotalPrice(double unitPrice, int quantity) {***

***return unitPrice \* quantity;***

***}***

***double applyDiscount(double totalPrice, double discountRate) {***

***return totalPrice \* (1 - discountRate);***

***}***

***};***

**Unit Testing with Google Test:**

***#include <gtest/gtest.h>***

***TEST(BillingTest, CalculateTotalPriceTest) {***

***Billing bill;***

***EXPECT\_DOUBLE\_EQ(bill.calculateTotalPrice(10.0, 5), 50.0);***

***EXPECT\_DOUBLE\_EQ(bill.calculateTotalPrice(15.0, 0), 0.0);***

***}***

***TEST(BillingTest, ApplyDiscountTest) {***

***Billing bill;***

***EXPECT\_DOUBLE\_EQ(bill.applyDiscount(100.0, 0.1), 90.0);***

***EXPECT\_DOUBLE\_EQ(bill.applyDiscount(200.0, 0.2), 160.0);***

***}***

***int main(int argc, char \*\*argv) {***

***::testing::InitGoogleTest(&argc, argv);***

***return RUN\_ALL\_TESTS();***

***}***

### ***Sample C++ Code for Integration Testing***

Once unit testing is complete, integration testing ensures correct interactions between modules. Here, we test the interaction between the Billing and Inventory modules to verify that inventory updates accurately after a purchase and that total calculations are processed correctly.

#### Supermarket Billing System Code (Billing and Inventory classes):

***class Billing {***

***public:***

***double calculateTotalPrice(double unitPrice, int quantity) {***

***return unitPrice \* quantity;***

***}***

***double applyDiscount(double totalPrice, double discountRate) {***

***return totalPrice \* (1 - discountRate);***

***}***

***};***

***class Inventory {***

***public:***

***bool updateStock(int &stock, int quantityPurchased) {***

***if (stock >= quantityPurchased) {***

***stock -= quantityPurchased;***

***return true;***

***}***

***return false;***

***}***

***};***

#### **Integration Testing with Google Test:**

***#include <gtest/gtest.h>***

***TEST(IntegrationTest, BillingAndInventoryTest) {***

***Billing bill;***

***Inventory inv;***

***int stock = 10;***

***double unitPrice = 20.0;***

***int quantityPurchased = 3;***

***double totalPrice = bill.calculateTotalPrice(unitPrice, quantityPurchased);***

***bool stockUpdated = inv.updateStock(stock, quantityPurchased);***

***EXPECT\_DOUBLE\_EQ(totalPrice, 60.0);***

***EXPECT\_TRUE(stockUpdated);***

***EXPECT\_EQ(stock, 7); // Ensure stock has decreased by the quantity purchased***

***}***

***int main(int argc, char \*\*argv) {***

***::testing::InitGoogleTest(&argc, argv);***

***return RUN\_ALL\_TESTS();***

***}***

### ***Conclusion***

Testing, particularly unit and integration testing, is essential for ensuring the proper functioning of complex systems like the **Supermarket Billing System**. Unit testing verifies that each module behaves as expected individually, while integration testing confirms that modules interact correctly to form a complete, reliable system. By using tools like Google Test, we can efficiently validate the accuracy and robustness of the system, ensuring a high-quality and maintainable supermarket billing solution.