EXPERIMENT - 1

AIM - To design, develop, and test an automated Inventory Management System (IMS)

THEORY -

An Inventory Management System (IMS) is a software application designed to track and control the flow of goods within an organization. It maintains information about items, stock levels, suppliers, purchase orders, and sales. The system ensures accurate stock updates, reduces human errors, and enables informed decision-making through timely and structured data.

IMS typically includes modules such as item registration, stock-in/stock-out operations, supplier management, report generation, and user authentication. Modern inventory systems are essential for businesses because they automate manual tasks, increase operational efficiency, and provide real-time visibility into inventory status.

Problem Statement -

Many organizations still depend on manual or semi-automated methods for tracking inventory, such as paper records or spreadsheets. These methods often lead to issues like inaccurate stock counts, delayed updates, stockouts, overstocking, mismanagement of supplier information, and poor decision-making due to lack of real-time data. Human errors and inconsistent record-keeping further complicate inventory control.

There is a need for a centralized and automated Inventory Management System (IMS) that can maintain accurate stock information, track item movement, generate necessary reports, manage suppliers, and provide real-time insights. Such a system must reduce manual effort, improve accuracy, streamline operations, and support smooth functioning of inventory processes within an organization.

Experiment-2

Software Requirements Specification

for

Inventory Management System

Version 1.0 approved

Prepared by <Sameer Sharma and Vishesh Verma>

<Maharaja Agrasen Institute of Technology>

<20 August 2025>

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) documents the functional and nonfunctional requirements for the Inventory Management System (IMS). The IMS will automate inventory tracking (inflow/outflow), produce reports, send low-stock alerts, and provide a user-friendly interface to reduce manual errors, improve decision making, and increase customer satisfaction for managing stocks.

1.2 Document Conventions

Standard IEEE conventions have been followed in this document. Each requirement is uniquely tagged and clearly described. Important terms are written in bold.

1.3 Intended Audience and Reading Suggestions

- Project stakeholders / business owners — read Section 1 (overview), Section 2 (product functions) and Section 5 (business rules).
- Developers / architects — read the entire document, especially Sections 3 and 4 (interfaces and system features).
- Testers / QA — use functional requirements in Section 4 and nonfunctional items in Section 5 for test cases.
- Documentation writers / trainers — refer to Section 2.6 (user documentation) and UI descriptions in Section 3.1.

1.4 Product Scope

IMS is a web-based (and optional mobile) application that:

- Tracks product inventory in real-time (incoming receipts, stock movements, sales/dispatch, returns).
- Generates reports (stock levels, stock valuation, movement history, low-stock items, reorder suggestions).
- Sends alerts via email/SMS/in-app for low stock and other events.
- Manages users and roles (admin, manager, stock clerk, viewer).

Primary benefits: reduced manual error, timely restocking, faster operations, data-driven purchasing decisions, improved customer satisfaction.

1.5 References

- IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications.
- Entity-Relationship Diagram Draft (provided separately).

2. Overall Description

2.1 Product Perspective

IMS is a new, self-contained product intended to be deployed as a SaaS or on-premises web application. It may optionally integrate with existing POS/accounting systems.
Logical components:

- Web Client (browser-based UI)
- Backend REST API
- Database (relational DB)
- Notification service (email/SMS)
- Barcode/QR scanner hardware interface

2.2 Product Functions

- User authentication & authorization
- Product catalog management (create/update/delete)
- Stock receipts (purchase receipts, returns-in)
- Stock issues (sales dispatch, returns-out, transfers)
- Stock adjustments (corrections, write-offs)
- Real-time stock level calculation
- Low-stock alerts & reorder suggestions
- Reporting & dashboards
- Audit trail / activity log
- Integration endpoints (REST API / webhooks)

2.3 User Classes and Characteristics

- System Administrator
  - Full privileges: system configuration, user management, backups.
  - Technical competence: high.
- Inventory Manager
  - Manage products, suppliers, ordering, view reports.
  - Technical competence: moderate.
- Stock Clerk
  - Record receipts and issues, perform adjustments, scan items.
  - Technical competence: basic-to-moderate.
- Sales / POS Operator
  - Create sales/dispatch records; view product availability.
  - Technical competence: basic.
- Auditor / Viewer
  - Read-only access to reports and logs.

2.4 Operating Environment

- Backend: Linux server (Ubuntu/CentOS) or containerized (Docker), REST API service (Node.js).
- Database: PostgreSQL (or MySQL) with daily backups.
- Clients: Modern browsers (Chrome, Firefox, Edge, Safari) - latest two major versions; optional Android/iOS mobile app.
- Communication: HTTPS (TLS 1.2+), SMTP for emails, gateway for SMS (third-party).

2.5 Design and Implementation Constraints

- Must use HTTPS and secure credential storage (hashed passwords).
- Data retention and backup schedule to meet business needs.
- Support for multi-tenant deployment (optional) — must be planned if SaaS.
- Integration via REST API using JSON.
- Conform to company UI style guide (TBD).
- Performance constraint: API response times should meet nonfunctional requirements (Section 5.1).

2.6 User Documentation

Delivered artifacts:

- User manual (PDF/HTML) with step-by-step workflows for each user class.
- Quick-start guide for Stock Clerks.
- Admin guide (setup, backup, restore, configuration).
- API documentation.
- In-app contextual help and tooltips.

2.7 Assumptions and Dependencies

- Reliable internet for cloud deployments; local network for on-premise.
- Third-party email/SMS gateways available and configured.
- POS/accounting systems that require integration expose APIs or CSV export.
- Barcode scanners emulate keyboard input (HID) or have SDKs supported.
- Compliance requirements (tax/legal) may vary by region, customer responsibility to configure.

3. External Interface Requirements

3.1 User Interfaces

- Dashboard: KPIs (low-stock count, stock value, recent moves).
- Product list and product detail pages.
- Stock operations screens: Receive Stock, Issue Stock, Transfer, Adjust.
- Alerts page for low-stock & critical events.
- User & role management screens.
- Standard UI elements: header, navigation, breadcrumbs, modals, confirmation dialogs, form validation messages, accessible keyboard shortcuts (where appropriate).

3.2 Hardware Interfaces

- Barcode/QR scanners (USB HID / Bluetooth): system shall accept SKU/serial via scanner input.
- Label printers: support printing of barcode labels (via client-side printing or server-side label generation PDF).

3.3 Software Interfaces

- Database: PostgreSQL (primary), schema described in design docs.
- External systems: REST API endpoints for:
  - POS integration (send stock updates / receive sales).
  - Supplier procurement systems (optional).
- Authentication: OAuth 2.0 / JWT tokens for API clients (implementation detail).
- Third-party services: SMTP (email), SMS gateway (Twilio/alternative), optional SSO provider.

3.4 Communications Interfaces

- All client-server communication over HTTPS (TLS 1.2+).
- API format: JSON (UTF-8).
- Webhooks: Outgoing webhooks for events (e.g., low-stock, new order).
- Email templates for alerts and reports.
- Rate limits on API usage.

4. System Features

Each feature below includes description, priority, stimulus/response sequence, and functional requirements.

4.1 Product & Catalog Management

4.1.1 Description & Priority

Maintain a master list of products. Priority: High.

4.1.2 Stimulus / Response

User creates/edits/deletes a product → system validates input and persists record → UI shows updated list and confirmation.

4.1.3 Functional Requirements

- REQ-1: The system shall allow creation of a product with fields: SKU (unique), name, description, category, unit of measure, supplier(s), purchase price, selling price, reorder level, reorder quantity, barcode, batch/lot support (optional). Priority: High
- REQ-2: The system shall enforce SKU uniqueness. Priority: High
- REQ-3: The system shall allow editing and soft-deleting (archiving) of product records. Priority: High
- REQ-4: The system shall support product categories and tagging for filtering. Priority: Medium
- REQ-5: The system shall allow attaching images to products (size limits defined). Priority: Low

4.2 Inventory Transactions (Receive / Issue / Transfer)

4.2.1 Description & Priority

Record inflow/outflow/transfer of stock. Priority: High.

4.2.2 Stimulus / Response

Clerk scans product and records quantity for receipt → system validates, updates stock, creates transaction record, updates audit log.

4.2.3 Functional Requirements

- REQ-6: The system shall record stock receipts (purchase receipt, return-in) with transaction ID, date/time, product SKU, quantity, unit cost, supplier, batch/lot (if applicable), and destination location. Priority: High
- REQ-7: The system shall record stock issues (sales dispatch, return-out) with transaction details and decrement stock appropriately. Priority: High
- REQ-8: The system shall allow internal transfers between locations/warehouses and update source and destination stock levels atomically. Priority: High
- REQ-9: The system shall support stock adjustments (positive/negative) with mandatory reason and user remarks; these adjustments shall be logged. Priority: High
- REQ-10: The system shall validate that stock issue operations do not produce negative available stock unless a user with override permission performs the action. Priority: High

4.3 Real-time Stock Level Calculation & Availability

4.3.1 Priority: High

4.3.3 Functional Requirements

- REQ-11: The system shall present current stock level (available, reserved, on-order, in-transit) per SKU and per location. Priority: High
- REQ-12: The system shall recalculate available stock immediately after any confirmed inventory transaction. Priority: High

4.4 Alerts and Reorder Suggestions

4.4.1 Priority: High

4.4.3 Functional Requirements

- REQ-13: The system shall generate low-stock alerts when product stock ≤ reorder level. Priority: High
- REQ-14: The system shall send alerts via in-app notifications, email, and SMS (configurable) to specified users/roles. Priority: High
- REQ-15: The system shall suggest reorder quantities based on reorder quantity field and configurable safety stock / lead time (optional). Priority: Medium

4.5 Reporting & Dashboards

4.5.1 Priority: High

4.5.3 Functional Requirements

- REQ-16: The system shall provide the following reports with filtering and export to CSV/PDF: Current Stock Report, Stock Movement History, Stock Valuation (FIFO and weighted average as options), Low Stock Report, Top-moving Products, Purchase History, Audit Log. Priority: High
- REQ-17: The system shall provide a dashboard summarizing KPIs (total SKUs, total stock value, low-stock count, stock aging) with date-range selection. Priority: High
- REQ-18: The system shall allow scheduled reports to be emailed to specified recipients. Priority: Medium

4.6 User Management & Security

4.6.1 Priority: High

4.6.3 Functional Requirements

- REQ-19: The system shall implement role-based access control (RBAC) with roles: Admin, Manager, Clerk, Viewer; custom roles shall be supported. Priority: High
- REQ-20: The system shall require strong password policy (configurable) and store passwords hashed (bcrypt/argon2). Priority: High

- REQ-21: The system shall support 2FA (optional) via authenticator apps or SMS. Priority: Medium
- REQ-22: The system shall record all user actions related to inventory changes in an audit log (who, what, when, old value, new value). Priority: High

4.7 Audit & Reports Retention

4.8.1 Priority: Medium

4.8.3 Functional Requirements

- REQ-25: The system shall expose REST API endpoints for key operations: fetch stock levels, push transactions, fetch product catalog. API shall be documented and protected using API keys/JWT. Priority: Medium
- REQ-26: The system shall provide webhooks for configurable events (e.g., low-stock, stock received). Priority: Low

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- NFR-1: The system shall respond to UI actions (page load, search, single API call) within 2 seconds under normal load (< 100 concurrent users).
- NFR-2: API endpoints shall return results for standard queries (e.g., fetch product list page) within 500 ms under typical load.
- NFR-3: The system shall support at least 500 concurrent users in the baseline configuration (scaleable via horizontal scaling).

5.2 Safety Requirements

- SAF-1: The system shall prevent accidental destructive operations with confirmation dialogs and permission checks (e.g., deleting product or bulk destructive import).
- SAF-2: Sensitive operations (stock adjustments, deletion) must require a reason and be recorded in the audit log.

5.3 Security Requirements

- SEC-1: All communications shall be encrypted (HTTPS/TLS).
- SEC-2: Passwords shall be stored hashed using a secure algorithm (bcrypt/argon2). No plaintext storage.
- SEC-3: Role-based access control (RBAC) must be enforced on both UI and API.
- SEC-4: System shall implement rate limiting on public APIs to prevent abuse.
- SEC-5: Data backups must be encrypted at rest.
- SEC-6: Personal data (if any) shall be stored and processed per applicable privacy regulations.

5.4 Software Quality Attributes

- Usability: Common tasks (receive stock, issue stock) must be performable by trained staff within 60 seconds.
- Availability: System shall target 99.5% uptime (SLA to be defined for SaaS customers).
- Maintainability: Codebase shall have automated tests (unit and integration) with target coverage (TBD).
- Portability: Support common relational DBs (Postgres, MySQL) where feasible.
- Reliability: Transactions that modify stock shall be ACID-compliant to avoid inconsistent state.

5.5 Business Rules

- BR-1: Stock valuation method default is FIFO, configurable to weighted average.
- BR-2: A product can have multiple suppliers; preferred suppliers may be specified.
- BR-3: Reorder is suggested when physical stock ≤ reorder level; actual purchase orders remain a separate flow.
- BR-4: Negative stock is not allowed except when an authorized user performs an override with a mandatory reason.

6. Other Requirements

- DB Schema: Entity-relationship for Products, Transactions, Locations, Users, Suppliers.
- Internationalization: Support for multiple currencies and locales; primary locale initially set to English (en-IN default).
- Legal/Compliance: Tax calculation is out of scope for core IMS; an integration point shall be provided for tax modules.
- Backup & Restore: Daily automated backups; ability for admin to run manual backup and restore (with role restrictions).
- Data Migration: Provide migration tools for importing from spreadsheets or legacy systems.
- Scalability: Support horizontal scaling of API services and read-replicas for DB.

Appendix A: Glossary

- **SKU**: Stock Keeping Unit — unique product identifier.
- **Stock Receipt**: Operation recording incoming stock.
- **Stock Issue**: Operation recording outgoing stock (e.g., sales).
- **Reorder Level**: Configured threshold which triggers low-stock alert.
- **FIFO**: First-In-First-Out valuation method.
- **RBAC**: Role-Based Access Control.

Appendix B: Analysis Models

- ER Diagram: Entities: Product, ProductBatch, Location, InventoryRecord, Transaction, User, Supplier, AuditLog.
- Sequence Diagrams: Stock Receive, Stock Issue, Stock Transfer flows (to be created in design phase).

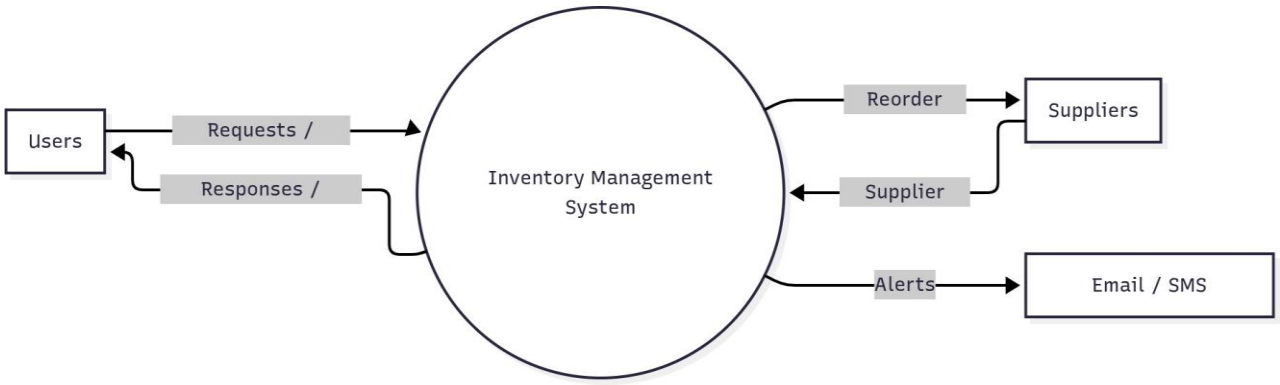
Appendix C: To Be Determined List

- Detailed UI mockups TBD
- API specification TBD

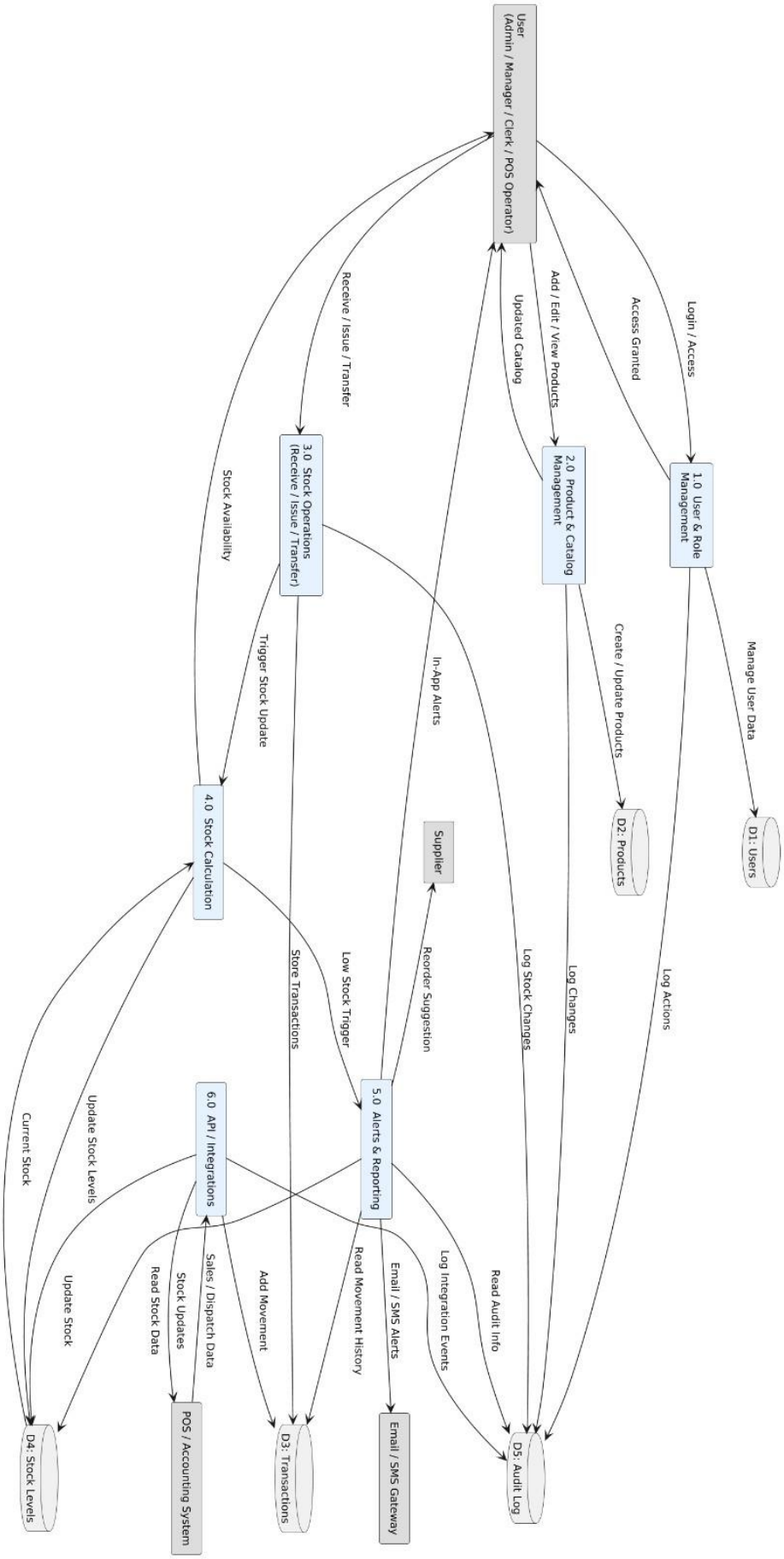
EXPERIMENT - 3

AIM - To perform the function oriented diagram: Data Flow Diagram(DFD) andStructured chart.

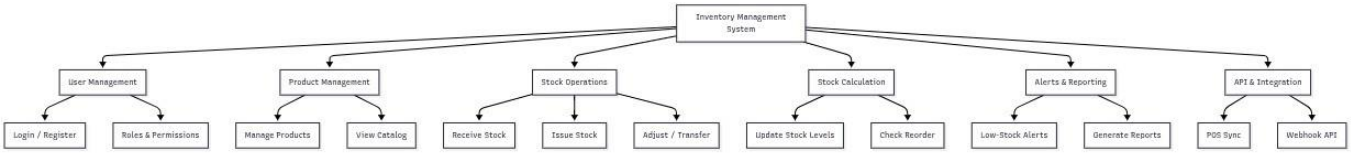
LEVEL 0 Data Flow Diagram



LEVEL 1 Data Flow Diagram



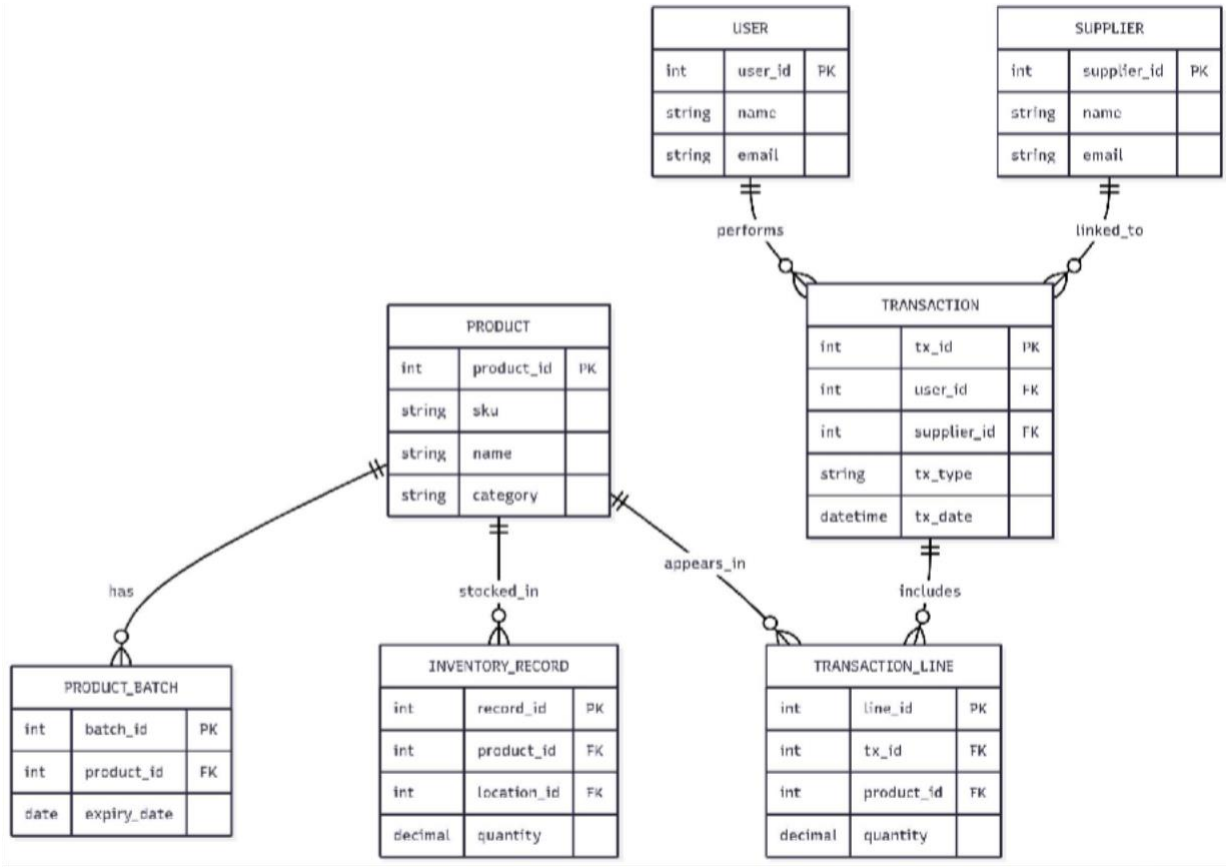
Structured Chart



EXPERIMENT - 4

AIM - Draw the entity relationship diagram for the suggested system (Parking Management System).

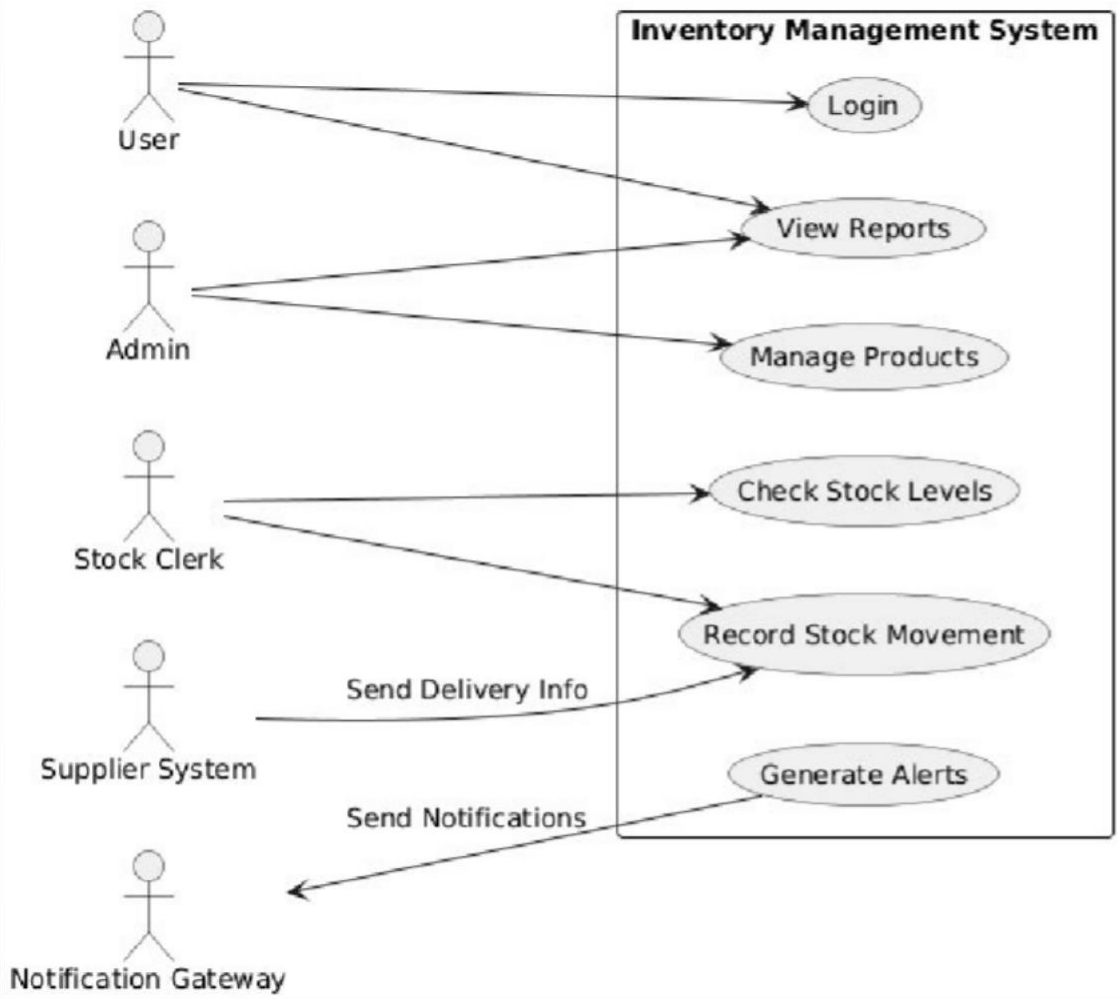
Entity Relationship Diagram



EXPERIMENT - 5

AIM - To perform the user's view analysis for the suggested system (Parking Management System) i.e Use case diagram.

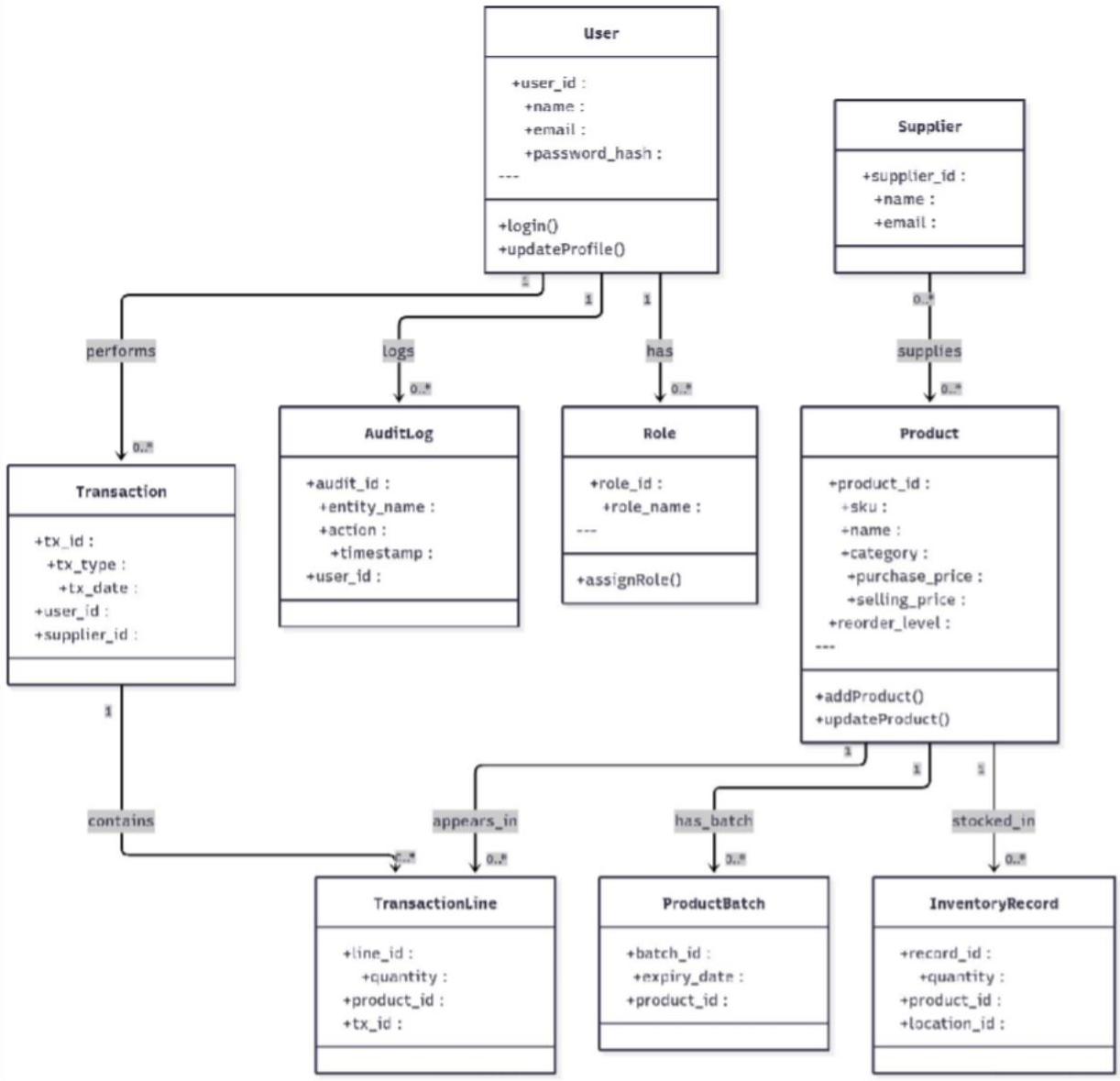
Use case diagram



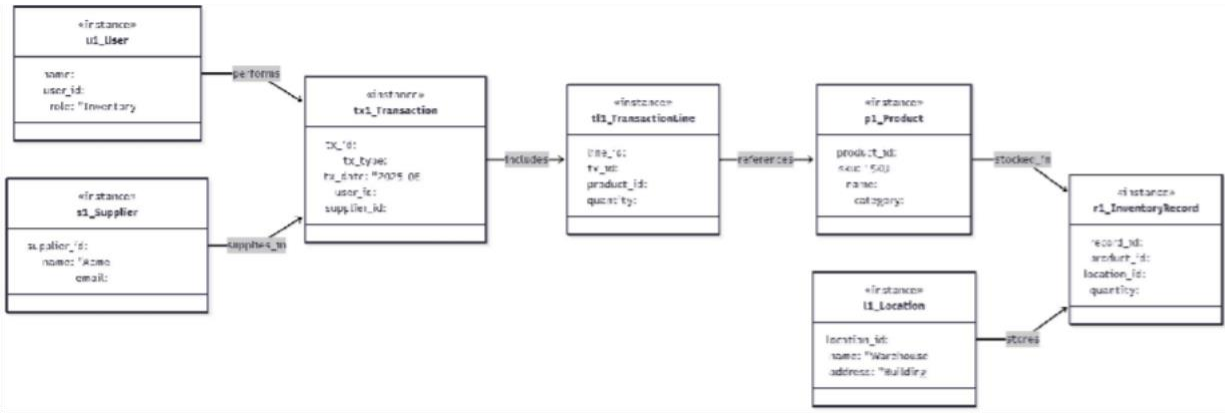
EXPERIMENT - 6

AIM - To draw the structural view diagram for the system: Class diagram, Object diagram.

Class diagram



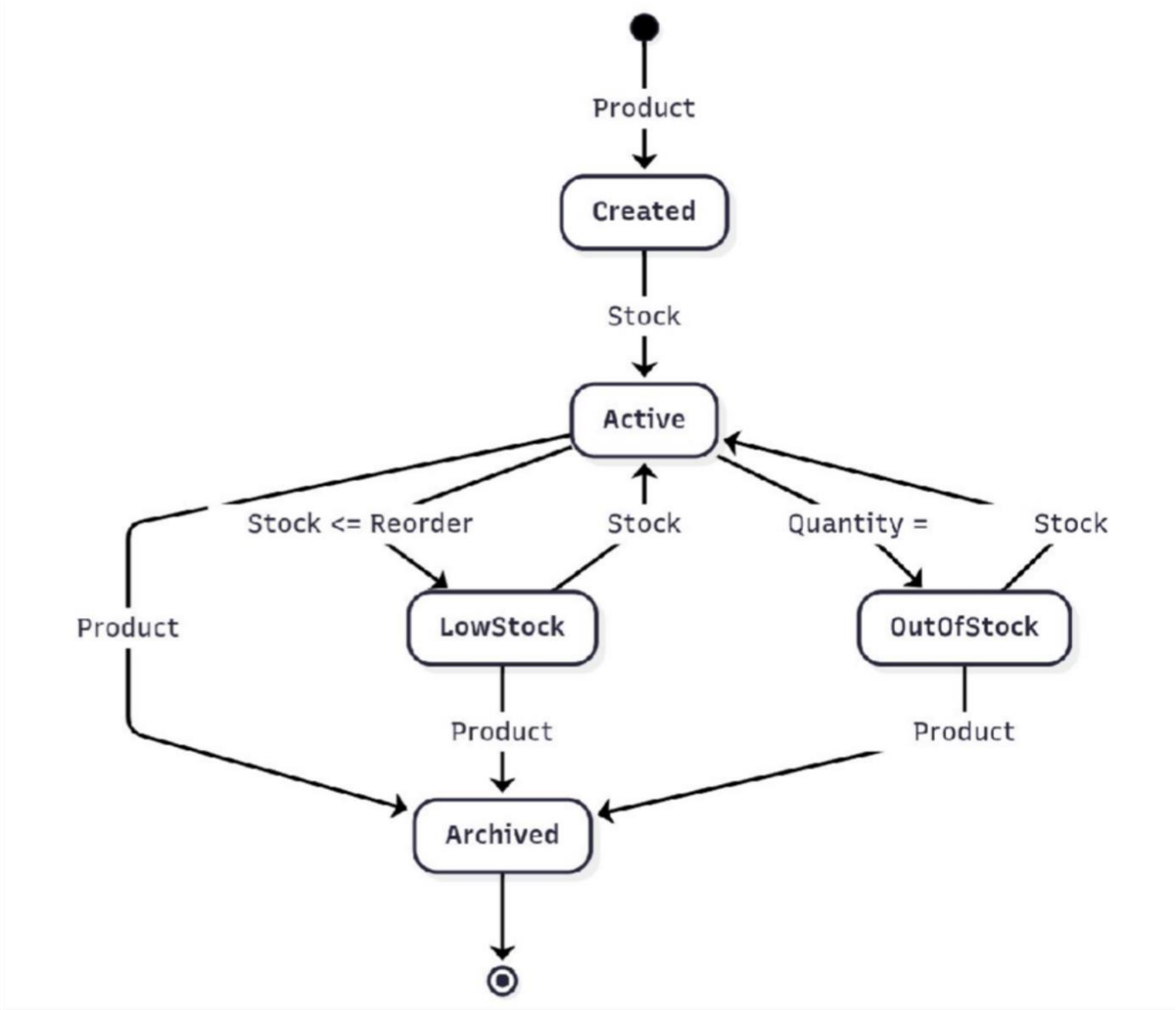
Object diagram



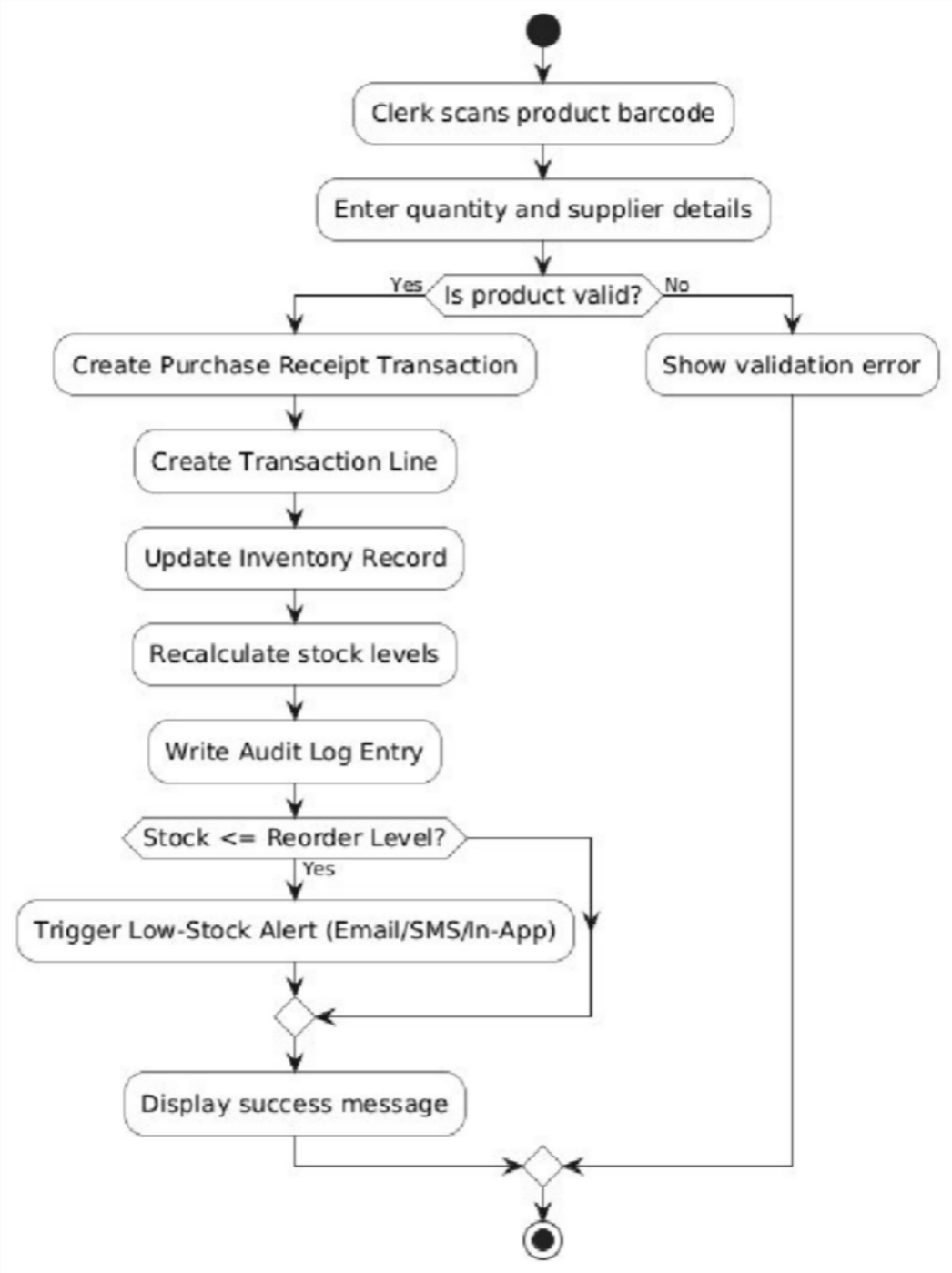
EXPERIMENT - 7

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

State-chart diagram



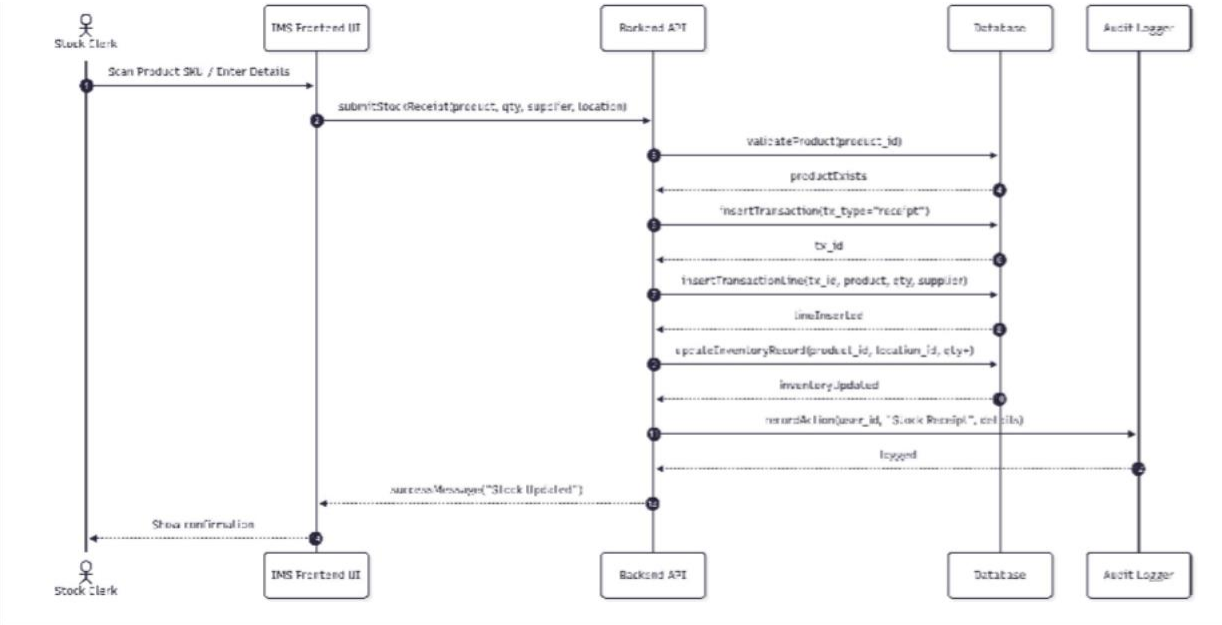
Activity diagram



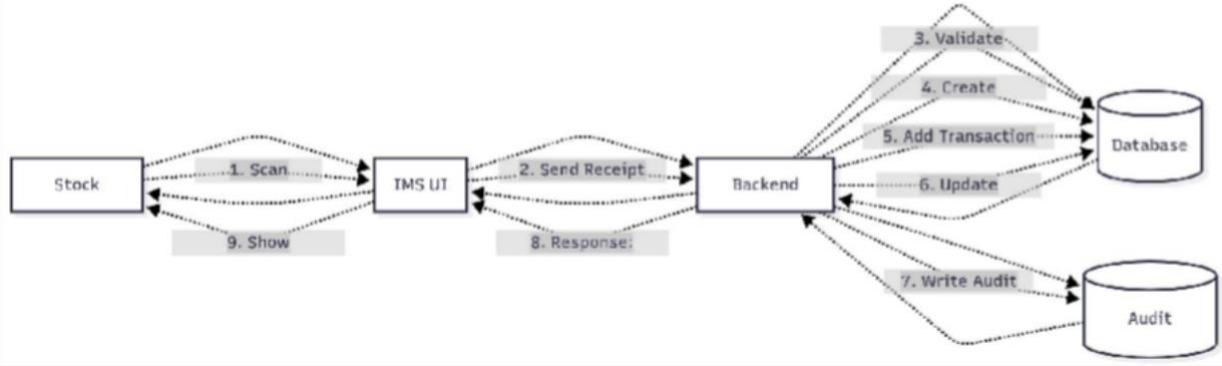
EXPERIMENT - 8

AIM - To perform the behavioral view diagram for the suggested system (Parking Management System) : Sequence diagram, Collaboration diagram.

Sequence diagram



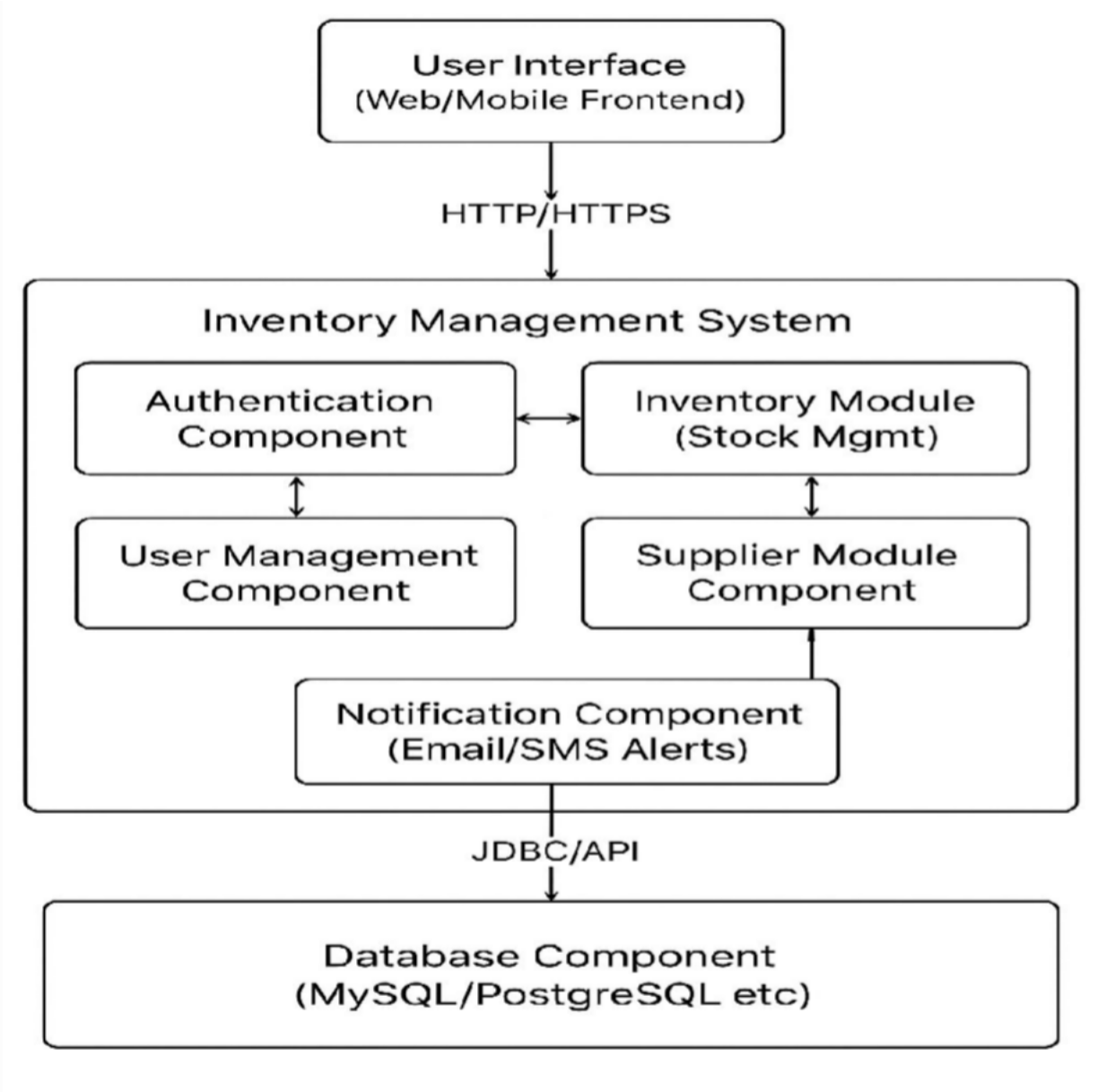
Collaboration diagram



EXPERIMENT - 9

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

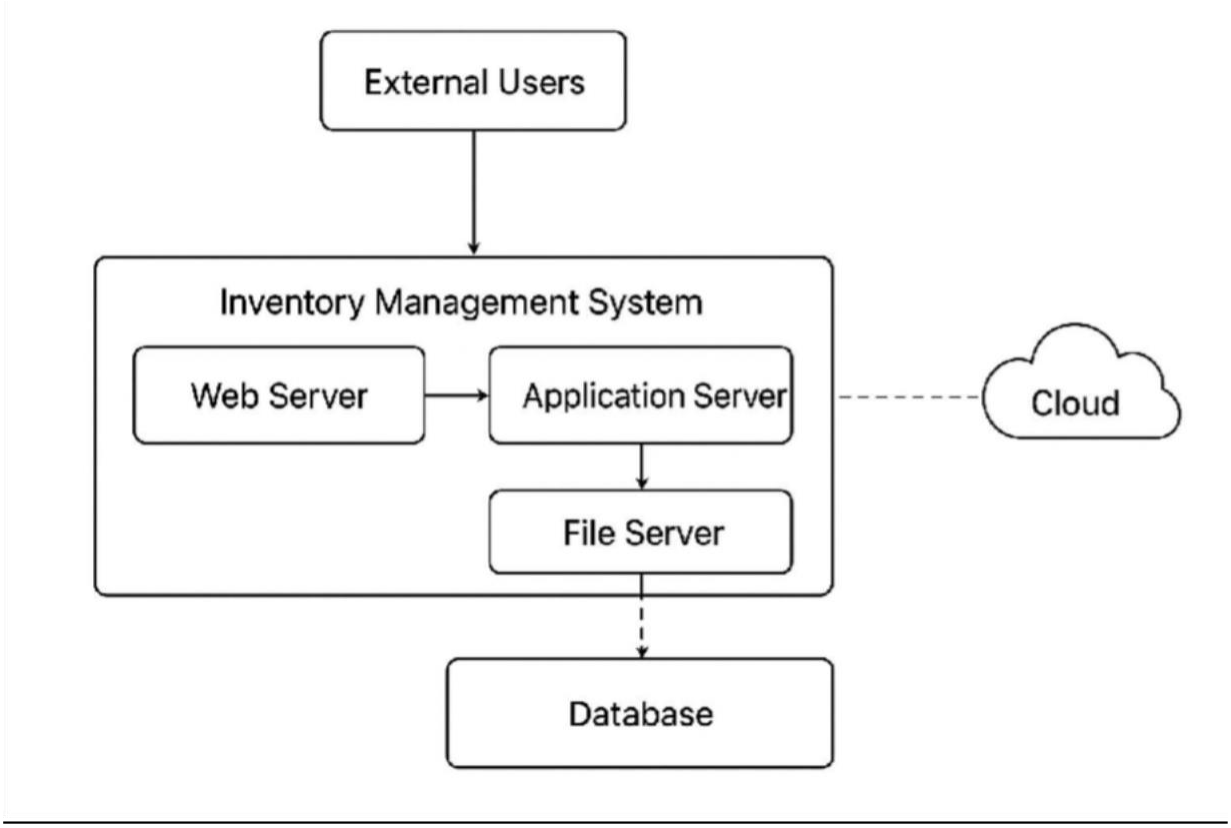
Component diagram



EXPERIMENT - 10

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

Deployment diagram



EXPERIMENT - 11

AIM - To perform various types of software testing such as unit testing, integration testing, and system testing on the Inventory Management System (IMS)

THEORY

Software testing is the process of evaluating software to detect differences between expected and actual results. It is done to improve quality, reliability, and performance.

Below is the theory for all major testing types used in the IMS project.

1. Unit Testing Purpose of Unit Testing :-

- Validate each function works correctly
- Catch bugs early
- Ensure isolated correctness
- Characteristics
- Performed by developers
- Uses mock data

Example in IMS

Testing individual functions like:

- add_item()
- update_stock()
- get_stock()

2. Integration Testing

Integration Testing checks how multiple modules work together

Purpose-

- Ensure modules interact correctly
- Detect interface defects
- Validate data flow between modules

Examples in IMS

- Add Item → Update Stock → Generate Report
- Supplier Module + Inventory Module
- Login System + Access Control

TEST CASES FOR PMS

(A) Unit Test Cases

Tc No.	Test scenario	Input	Expected Output
U1	Add valid item	101, “mouse” 10	Item added
U2	Add duplicate item	101, “mouse” 1	Item exists
U3	Negative quantity	102, “keyboard” -5	Invalid quantity

CONCLUSION:

Testing of the Inventory Management System (IMS) was performed using unit testing, integration testing, and system testing. Each module was tested individually and then combined to verify inter-module communication. The complete system was validated against SRS requirements.

EXPERIMENT - 12

AIM - To perform estimation of effort using Function Point (FP) Estimation for the Inventory Management System (IMS).

THEORY

FP Estimation is used to estimate the size and effort required to develop a software system. It is based on the number of external inputs, outputs, files and user interactions.

1. **Identifying Function Types for IMS**

   - Identify and count function types
   - Internal Logical Files (ILF) — persistent internal data

   Items
   - Users / Roles
   - Suppliers
   - Purchase Orders
   - Stock Transactions (stock-in / stock-out logs)

   Categories (item categories)
   ILF count = 6

2. **External Interface Files (EIF) — referenced external files / systems**

- External ERP / Billing system (integration)
- Email/SMS gateway (for notifications)

EIF count = 2

3. **Value Adjustment Factor (VAF)**

GSC	Rating (0–5)	Justification
1 Data communications	2	Web UI + possible ERP/email integ-
2 Distributed functions	1	Mostly centralized web app
3 Performance	3	Real-time stock updates required
4 Heavily used configu-	2	Some configurable thresholds & roles
5 Transaction rate	3	Moderate transaction volume expe-
6 On-line data entry	4	Many online CRUD operations
7 End-user efficiency	3	Dashboards, quick lookup required
8 On-line update	4	Real-time updates to stock
9 Complex processing	2	Moderate — stock math, PO linking
10 Reusability	1	Limited reuse requirements
11 Installation ease	1	Standard web deployment
12 Operational ease	2	Notifications, monitoring
13 Multiple sites	1	Single/limited sites
14 Facilitate change	3	Expected evolution — categorres.roles

4. Final FP Calculation

FP = UFP x VAF

UFP = 145

VAF = 0.98

Adjusted Function Points (FP) = UFP × VAF = 145 × 0.98 = 142.1 ≈ 142 FP

RESULT:

Function Point estimation for IMS is 142 FP, which helps in effort and cost estimation of the project.

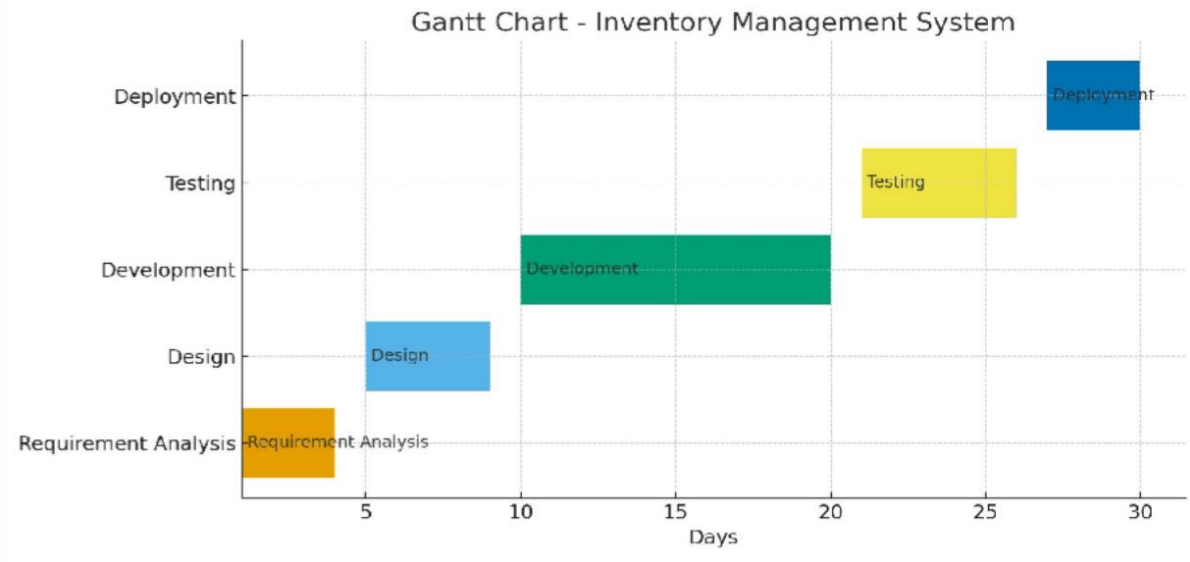
EXPERIMENT – 13

AIM - To prepare a time-line chart/Gantt Chart/PERT Chart for the Inventory Management System (IMS).

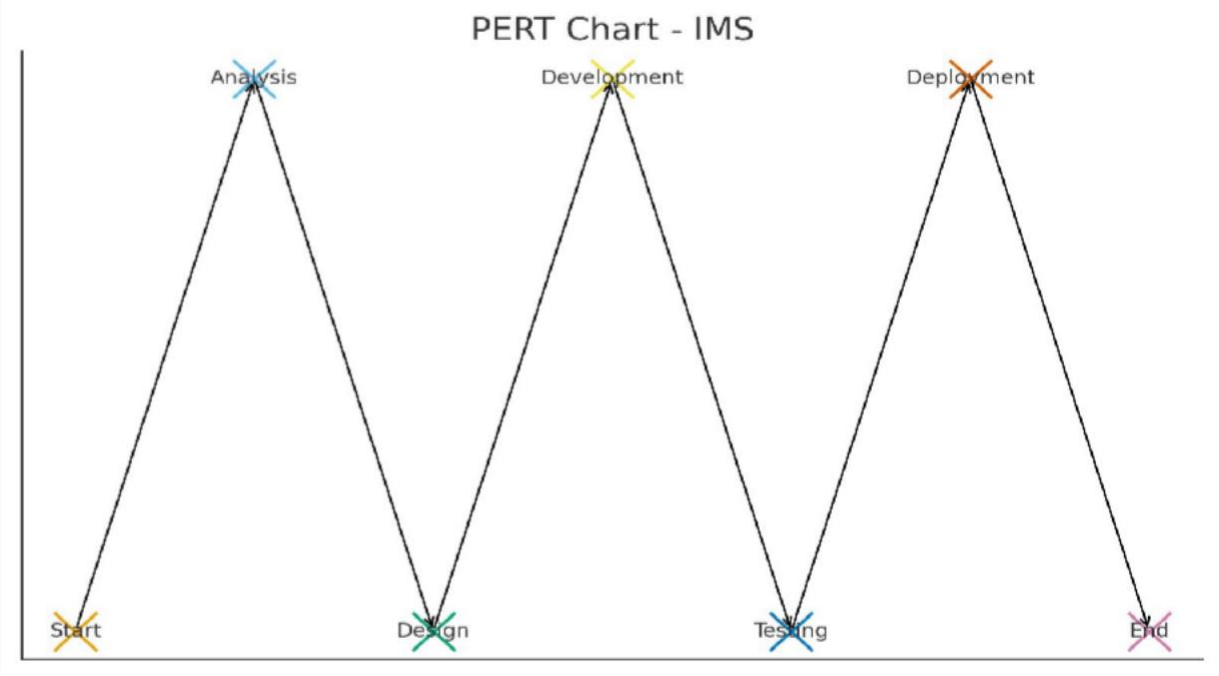
THEORY

Project scheduling helps visualize tasks, duration, dependencies and the critical path.

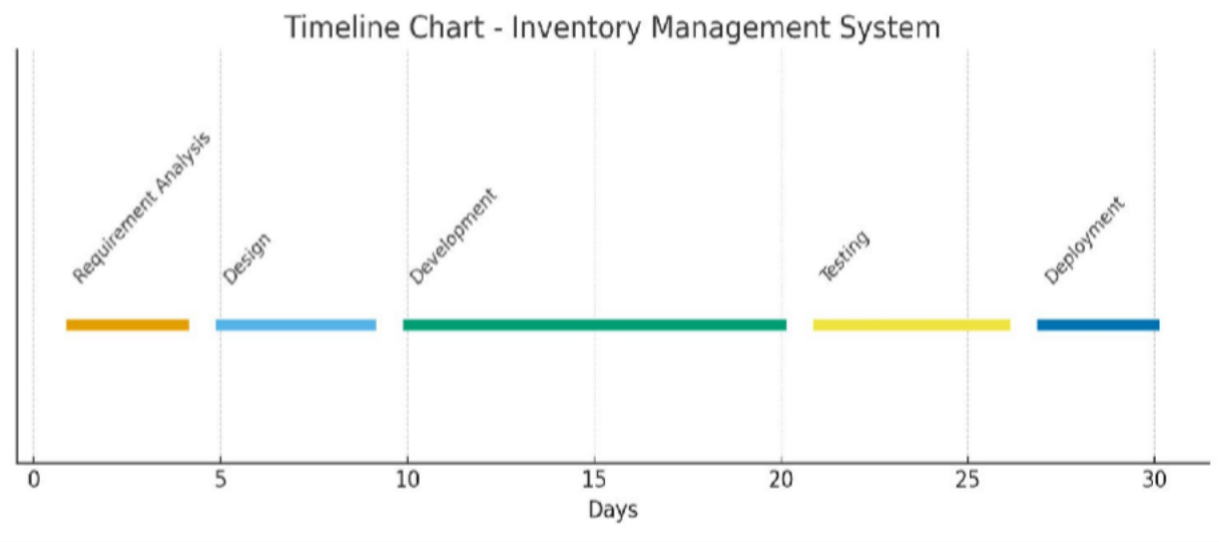
Gantt Chart for IMS



PERT Chart (Major Stages)



Timeline Chart



CONCLUSION: The Inventory Management System (IMS) project schedule and workflow were visually represented using three key charts: the Gantt Chart, Timeline Chart, and PERT Chart.