Cloud Based Multi Outlet System

Report

Developed by:

Akib Ibna Sunny(072)

Sanzida Hossain(161)

Sumaia Afrose Jyoti(191)

Ozifa Rahman(200)

Mehedi Hasan Joy(380)

Supervised by

Rubaiya Reza Sohana

Lecturer

Department of Computer Science & Engineering

Software Engineering Project

. **Table of contents**

Title: Cloud-Based Multi-Outlet Management System

Overview:

This project aims to develop a comprehensive cloud-based solution tailored for businesses that operate multiple retail or service outlets. With this system, business owners and administrators can oversee and manage every branch from a central platform. The system will include modules for inventory management, employee supervision, sales tracking, and analytics—all accessible through a secure web interface.

Objective:

- To centralize and simplify the operational processes of businesses with multiple outlets.
- To provide real-time visibility into sales and inventory data.
- To improve overall productivity and reduce manual coordination between outlets.
- To ensure secure, role-based access and enhance decision-making using data analytics.

Motivation:

Managing a growing network of outlets presents unique challenges—disparate systems, lack of synchronization, communication delays, and inconsistent reporting. This project is motivated by the need for a unified, cloud-driven platform that eliminates these inefficiencies and helps businesses scale effortlessly while maintaining control and consistency.

As businesses scale across various geographic locations, managing them individually becomes inefficient. A centralized cloud-based system offers improved productivity, better visibility, and reduced operational overhead.

Technologies Used:

Cloud Based Multi Outlet System is a web based website integrated with **Cloud Computing** Utilized to host the application on scalable and flexible cloud infrastructure (Google Cloud). This ensures high availability, data redundancy, and remote access.

RESTful APIs APIs are used for communication between the frontend and backend, allowing for a modular and scalable system structure. This also makes the system extensible for future mobile or third-party integrations.

Database Management System (DBMS) A relational database like MySQL is used for efficient data storage, retrieval, and management. The database stores structured data for businesses, outlets, products, sales and users.

Version Control GitHub will be used to manage source code versions and enable collaboration among developers.

• **Technology**

– MySQL for database management

– Google Cloud

– RESTful APIs

– XAMPP for running project on the server-side machine

• **Language**

– React.js

– Node.js

– SQL

Features:

1. **Multi-Outlet Dashboard:** A centralized dashboard to monitor and manage all outlets. Offers a consolidated view of performance metrics, sales trends, inventory levels, and employee activities.
2. **Real-Time Inventory Tracking:** Track stock levels across multiple outlets in real time. Receive notifications for low-stock or overstocked items and automate restocking procedures.
3. **Employee Management Per Outlet:** Manage employees assigned to specific outlets. Assign roles, monitor work hours, access performance metrics, and manage permissions on a per-outlet basis.
4. **Role-Based Access Control:** Define different access levels for Admins, Managers, and Employees to ensure data security and streamline operations by limiting access to relevant features only.
5. **Sales Analytics and Reporting:** Generate sales reports filtered by outlet, product, or employee. Visualize key metrics such as daily sales, best-selling products, and profit margins through interactive charts and graphs.
6. **Cloud Data Synchronization:** Ensure that all data—sales, inventory, employee updates—is automatically synchronized across outlets via a secure cloud server, minimizing data loss and inconsistencies.
7. **Notifications and Alerts:** Get real-time alerts for critical events like low inventory, security breaches, sales anomalies, or employee shift updates. Notifications can be sent via email, SMS, or in-app messages.

Proposed System

The proposed system is a cloud-based software platform designed to centralize and streamline the management of multiple retail or service outlets under one organization. The system enables real-time coordination between various locations while providing oversight and control to administrative users.

Key Proposals:

- **Central Admin Panel:** A secure interface for business owners and administrators to manage outlets, view analytics, and configure system-wide settings.
- **User Authentication & Role Management:** Role-based access for employees, managers, and administrators.
- **Inventory & Sales Management:** Modules for adding, updating, and tracking inventory items and sales transactions at each outlet.
- **Cloud Integration:** Hosted on a cloud infrastructure for high availability, scalability, and global access.
- **Analytics Engine:** Provides visualized reports and KPIs to assist in business decisions.

System Architecture

The system follows a three-tier architecture to promote modularity and scalability:

1. Presentation Layer:

- Built using React.js for a dynamic, responsive user interface.
- Responsible for displaying data to users and collecting input.

2. Application Layer:

- Implemented with Python Django, this layer handles business logic, API endpoints, and user sessions.
- It mediates between the presentation layer and the data layer, ensuring data integrity and workflow logic.

3. Data Layer:

- Uses MySQL for structured storage of users, inventory, transactions, and logs.
- This layer manages data persistence and supports querying for reports and analytics.

This architecture ensures loose coupling, better maintainability, and ease of deployment and scaling.

Goal

The primary goal of the project is to develop a robust and scalable cloud-based system that streamlines the operations of businesses with multiple outlets by:

- Reducing manual dependency and error-prone coordination
- Providing real-time insights into operations across all branches
- Ensuring secure and efficient inventory, sales, and staff management
- Offering a cost-effective, user-friendly solution adaptable to different business domains

ER-Diagram:

An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database.

Tables and Attributes

- ❖ Business

Attributes:

- business_id (Primary Key)
- name
- industry_type
- owner_info

- ❖ Outlet

Attributes:

- outlet_id (Primary Key)
- business_id (Foreign Key referencing Business)
- location
- manager_id

- ❖ Employee

Attributes:

- employee_id (Primary Key)
- outlet_id (Foreign Key referencing Outlet)
- name
- role
- contact_info

❖ Product

Attributes:

- product_id (Primary Key)
- name
- category
- price

❖ Inventory

Attributes:

- inventory_id (Primary Key)
- outlet_id (Foreign Key referencing Outlet)
- product_id (Foreign Key referencing Product)
- stock_quantity

❖ Sales

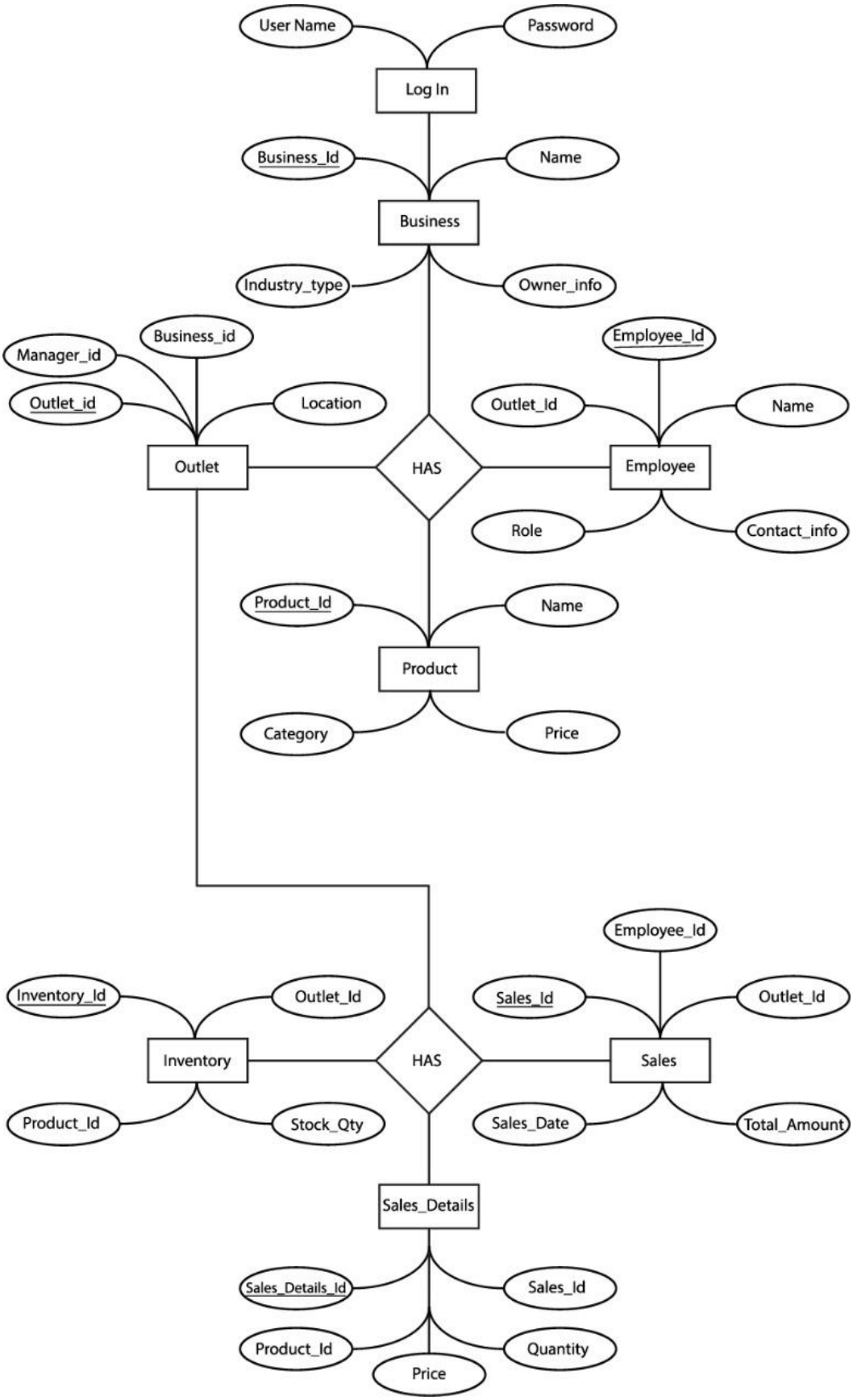
Attributes:

- sales_id (Primary Key)
- outlet_id (Foreign Key referencing Outlet)
- employee_id (Foreign Key referencing Employee)
- sale_date
- total_amount

❖ Sales_Detail

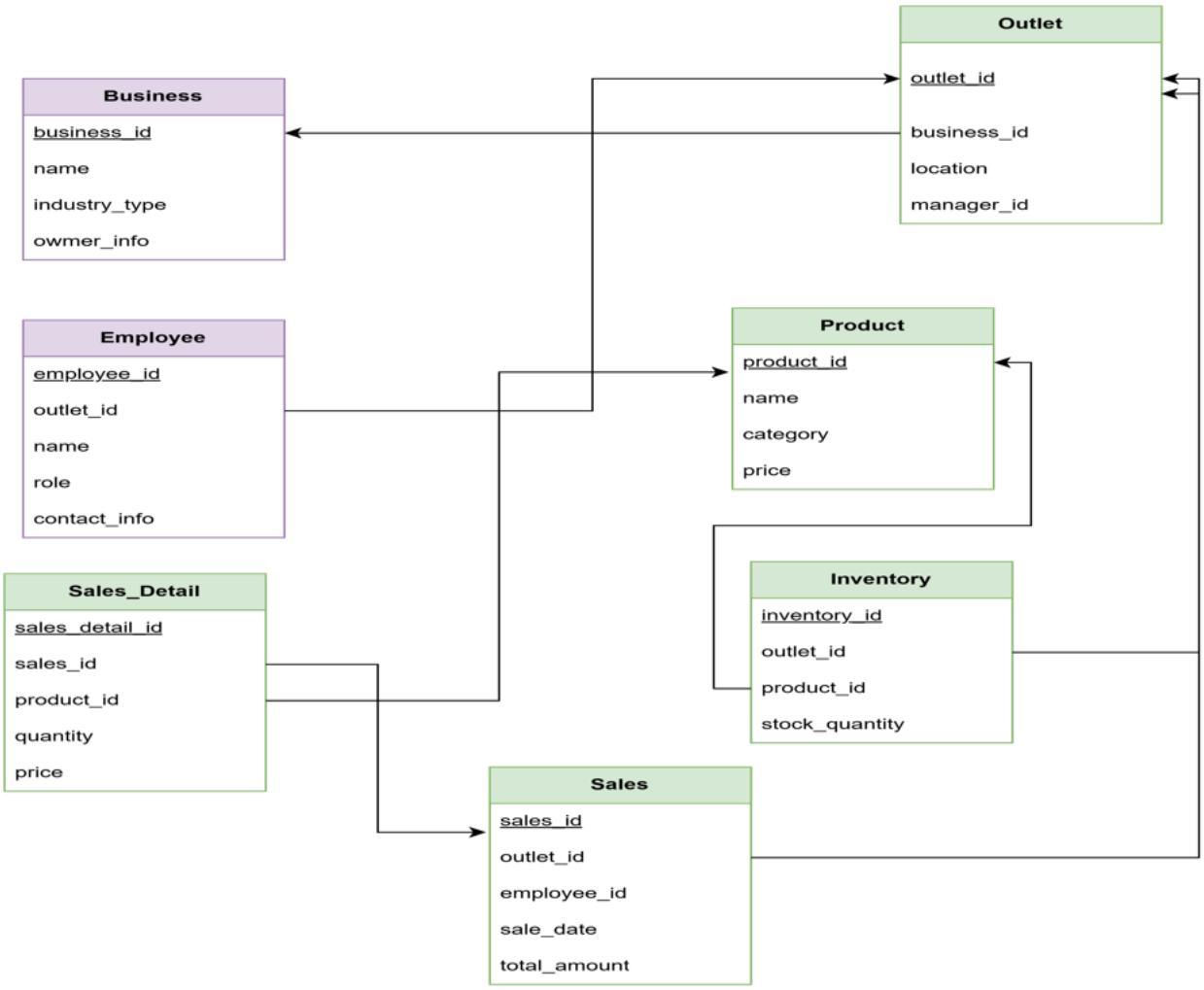
Attributes:

- Sales_detail_id (Primary Key)
- sales_id (Foreign Key referencing Sales)
- product_id (Foreign Key referencing Product)
- quantity
- price


User Name
Password
Log In
Business_Id
Name
Business
Industry_type
Owner_info
Business_id
Manager_id
Outlet_id
Location
Outlet
HAS
Employee_Id
Outlet_Id
Name
Employee
Role
Contact_info
Product_Id
Name
Product
Category
Price
Employee_Id
Inventory_Id
Outlet_Id
Inventory
HAS
Sales_Id
Outlet_Id
Sales
Product_Id
Stock_Qty
Sales_Date
Total_Amount
Sales_Details
Sales_Details_Id
Sales_Id
Product_Id
Quantity
Price

Schema Diagram:

This schema diagram (Part of ER Diagram) represents a relational database structure for Cloud Based Multi Outlet system.



Use Case Diagram:

The Use Case Diagram, which is a type of UML (Unified Modeling Language) diagram used to represent the interactions between users (actors) and a system. It outlines the functionalities (use cases) that the system provides to its users.

Components of the Diagram:

System Boundary:

The diagram is enclosed within a rectangular boundary labeled "System," which represents the scope of the system being modeled. All use cases inside this boundary are functionalities provided by the system.

Actors:

Actors are entities (users or external systems) that interact with the system. They are represented by stick figures and are placed outside the system boundary. The actors in this diagram are:

- **Admin:** Likely a system administrator with high-level access to manage the system.
- **Manager:** A user responsible for overseeing operations, such as inventory and sales.
- **Employee:** A user who handles day-to-day operations like processing sales.
- **Customer:** An end-user who interacts with the system to make purchases or view their order history.

Use Cases:

Use cases are the specific functionalities or actions that the system allows the actors to perform. They are represented by ovals inside the system boundary. The use cases in this diagram are:

- **Manage Employees:** Likely involves adding, updating, or removing employee records.
- **Manage Outlets:** Involves managing the details of different business outlets.
- **View Sales Reports:** Allows users to generate and view reports on sales performance.
- **Track Inventory:** Involves monitoring the stock levels of products.
- **Manage Inventory:** Involves updating inventory, such as adding or removing stock.
- **Process Sale:** Refers to handling a sales transaction, likely involving recording the sale and updating inventory.
- **View Order History:** Allows users to see past orders or purchases.
- **Make Purchase:** Refers to a customer buying products through the system.
- **Login & Register:** Allows users to log into the system or create an account.

Relationships:

The lines connecting actors to use cases represent the interactions between the actors and the system. Each line indicates which actor can perform which use case:

Admin:

- Can "Manage Employees."
- Can "Manage Outlets."
- Can "View Sales Reports."

Manager:

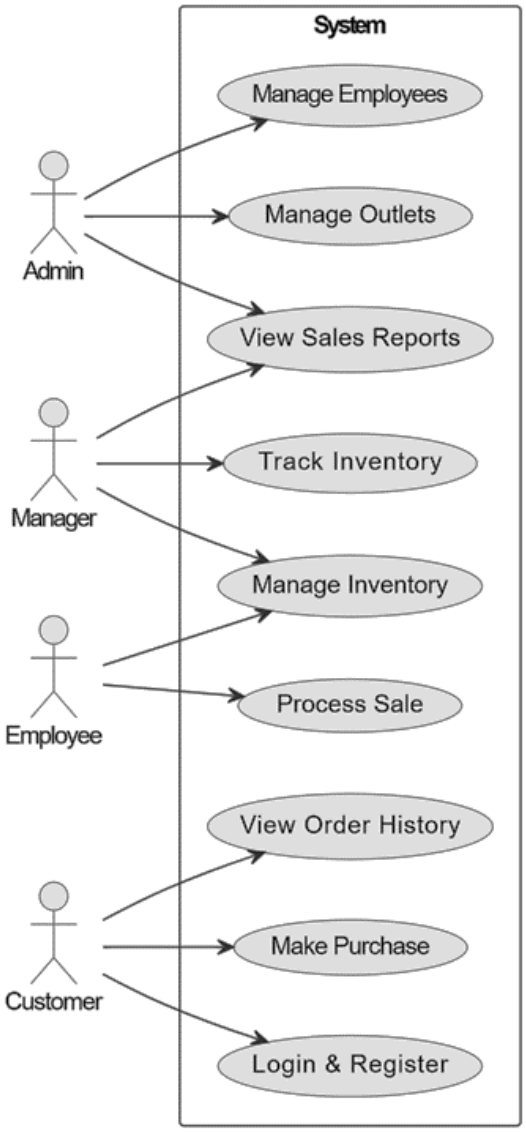
- Can "Track Inventory."
- Can "Manage Inventory."
- Can "View Sales Reports."

Employee:

- Can "Process Sale."

Customer:

- Can "View Order History."
- Can "Make Purchase."
- Can "Login & Register."



Class Diagram:

The Class Diagram, which is another type of UML (Unified Modeling Language) diagram used to represent the static structure of a system by showing its classes, their attributes, operations (methods), and the relationships among them.

Components of the Diagram:

Classes:

Each class is represented by a rectangle divided into three sections:

- **Top Section:** The name of the class.
- **Middle Section:** The attributes (data fields) of the class, with their data types.
- **Bottom Section:** The operations (methods) of the class, with their return types.

The classes in this diagram are:

- Admin
- Outlet
- Employee
- Customer
- Product
- Inventory
- Sale
- SaleDetails

Attributes:

Attributes are the data fields of each class, shown in the middle section with their data types. Attributes in red are likely primary or key identifiers (e.g., IDs). Here's a breakdown:

Admin:

- adminID: int (primary key)
- name: String
- email: String
- password: String

Outlet:

- outletID: int (primary key)
- name: String
- location: String
- contactInfo: String

Employee:

- employeeID: int (primary key)
- name: String
- role: String
- contact: String

Customer:

- customerID: int (primary key)
- name: String
- phone: String
- email: String

Product:

- productID: int (primary key)
- name: String
- category: String
- price: float

Inventory:

- inventoryID: int (primary key)
- quantity: int
- product: Product (association with Product class)
- outlet: Outlet (association with Outlet class)

Sale:

- saleID: int (primary key)
- date: Date
- totalAmount: float
- customer: Customer (association with Customer class)
- employee: Employee (association with Employee class)

SaleDetail:

- saleDetailID: int (primary key)
- quantity: int
- subTotal: float
- product: Product (association with Product class)
- sale: Sale (association with Sale class)

Operations (Methods):

Operations are the methods or functions that the class can perform, shown in the bottom section in green. Here's a breakdown:

Admin:

- manageOutlet(): void (manages outlets)
- assignEmployee(): void (assigns employees to outlets)

Outlet:

- viewSalesReport(): void (generates sales reports for the outlet)

Employee:

- processSale(): void (handles the sale process)

Customer:

- placeOrder(): void (places an order)

Product:

- getProductInfo(): String (returns product information)

Inventory:

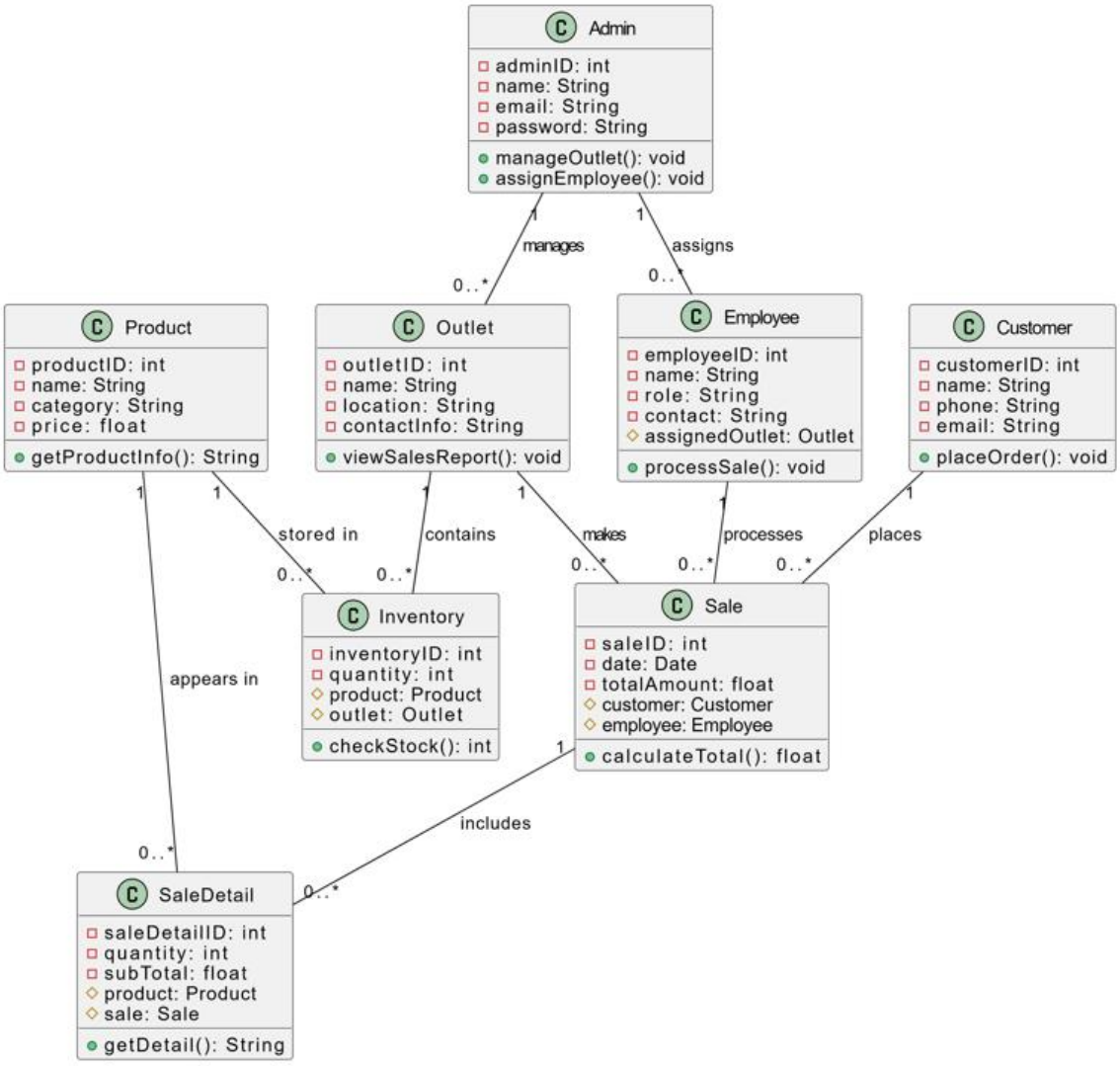
- checkStock(): int (checks the stock quantity)

Sale:

- calculateTotal(): float (calculates the total amount for the sale)

SaleDetail:

- getDetail(): String (returns details of the sale)



Challenges:

- Ensuring secure cloud data communication
- Real-time synchronization of data across outlets
- Scalability for high data volume
- Designing a simple yet powerful UI

Methodology

The development of the Cloud-Based Multi-Outlet Management System follows the Agile Software Development Methodology, which emphasizes iterative development, customer collaboration, and responsiveness to change.

This methodology ensures high adaptability, collaboration and continuous improvement throughout the software development lifecycle. Future improvements include mobile app integration, AI-based sales forecasting, third-party API integrations, offline sync features, and multilingual interface support for broader reach.

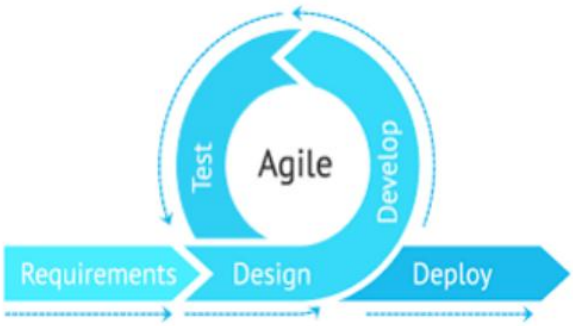


Figure 3.1: Agile Development model

Reasons for choosing Agile Development model:

- Clients are happier with the end product. This is possible due to improvements and involving clients in development decisions throughout the process;
- Lower cost; Fast releases
- More open communication between the teams and clients
- Greater project transparency. It is achieved due to regular meetings with the clients and systems. That allows all the involved parties to access the project data and progress
- Competitive advantage to the team thanks to spotting defects and making changes throughout the development process rather than at the end
- There are some high risk features and goals.

Current Limitations

- No offline mode for operations during internet failure
- Limited multilingual support
- Does not yet support third-party app integrations

Conclusion:

The Cloud-Based Multi-Outlet Management System successfully addresses the complex needs of growing, geographically distributed businesses. Through centralized control, real-time updates, and comprehensive analytics, the system enhances operational visibility and decision-making. With its modular architecture and future-ready technology stack, it is designed for scalability, flexibility, and integration with other enterprise solutions.

The project lays a strong foundation for further innovations such as AI-driven forecasting, mobile access, and deeper integration with financial and logistical platforms.

Project Outlook