Software Design Specifications

Stockwise Inventory Management System

Version: 1.1

Project Code	CS-3009
Supervisor	Fizza Aqeel
Co Supervisor	
Project Team	22K-4477 Syed Muqeet Ur Rehman 22K-4487 Muhammad Usman Sohail 22K-4658 Syed Hussamuddin
Submission Date	9th May, 2025

[Instructions]

- *No section of template should be deleted. You can write 'Not applicable' if a section is not applicable to your project. But all sections must exist in the final document.*
- *All comments/examples mentioned in square brackets ([]) are in the template for explanation purposes and must be replaced / removed in final document.*
- *This' Instruction' section should also be removed in final document.*
- *MS-Word Reviewing feature must be used to get the document reviewed by PMs or supervisors.*

Document History

[Revision history will be maintained to keep a track of changes done by anyone in the document.]

Version	Name of Person	Date	Description of change
1.1	Usman, Muqeet, Hussam	9.5.2025	Document Created

Distribution List

[Following table will contain list of people whom the document will be distributed after every sign-off]

Name	Role
Fizza Aqeel	Supervisor
	Co Supervisor

Document Sign-Off

[Following table will contain sign-off details of document. Once the document is prepared and revised, this should be signed-off by the sign-off authority.
Any subsequent changes in the document after the first sign-off should again get a formal sign-off by the authorities.]

Version	Sign-off Authority	Project Role	Signature	Sign-off Date

Document Information

Category	Information
Customer	FAST-NU
Project	Stockwise Inventory Management System
Document	Software Design Specification
Document Version	1.1
Status	Draft
Author(s)	Usman, Muqeet, Hussam
Approver(s)	Fizza Aqeel
Issue Date	9th May, 2025
Document Location	
Distribution	Advisor Project Coordinator's Office (through Advisor)

Definition of Terms, Acronyms and Abbreviations

Term	Description
SW	Stockwise
RESTful API	A web service design style that allows systems to communicate over HTTP using standard methods like GET, POST, PUT, DELETE
ACID	Atomicity, Consistency, Isolation, Durability
ReactJS	A JavaScript library for building user interfaces, especially dynamic single-page apps.
ORM	Object-Relational Mapping
Citus	An open-source extension to PostgreSQL that enables horizontal scaling of database workloads across multiple nodes.
CRUD	Create, Read, Update, Delete

Table of Contents

1 Introduction

1.1 Purpose of Document

This document specifies the software requirements for the "StockWise" (SW) version 1.0. This system is designed to streamline inventory tracking and management processes for medium to large-scale organizations. The scope of this document includes all the key functionalities and features of the SW, such as inventory monitoring, low-stock alert generation, supplier collaboration, order management, and sales reporting.
This SRS covers the complete system, including its subsystems like user management, inventory management, and alert generation, ensuring a cohesive and detailed understanding of the requirements for development, testing, and deployment. It serves as a foundation for communication between stakeholders, developers, and end users.

1.2 Intended Audience

This SRS is intended for developers, project managers, testers, end users, marketing staff, and documentation writers. Developers will use it to guide implementation, while testers will create test cases based on the system features and requirements. Project managers will use the document to align development with project goals, and marketing staff can ensure the product meets market needs. End users, such as admins and stock managers, will find relevant functionality outlined in the product functions and user interfaces sections. Documentation writers can reference it to create user manuals and help guides. Readers are encouraged to begin with the introduction and overall description, then focus on the sections most relevant to their role, such as system features, external interfaces, or nonfunctional requirements.

1.3 Document Convention

This document uses Arial font size 14 with bold and italic stylings for the headings, while Arial font size 10 is used for the main body.

1.4 Project Overview

StockWise is an enterprise-level inventory management system designed to streamline and automate stock tracking, order processing, and supplier collaboration for medium to large-scale retail businesses. Developed using a modern tech stack—ReactJS for the frontend, Node.js for the backend, and PostgreSQL for data storage—the system offers real-time inventory monitoring, low-stock alerts, order lifecycle management, role-based access control, and integration with external systems like POS, ERP, and payment gateways.

Key features include:

- Real-time stock updates and alerts to minimize wastage and stockouts.
- Sales reporting and analytics for informed decision-making.
- Seamless supplier interaction via automated purchase orders.
- Secure role-specific user access with two-factor authentication.
- High-performance scalability, supporting up to 10,000 concurrent users.

StockWise follows an Agile development process and aligns with GDPR and CCPA compliance, ensuring both efficiency and data security. The system aims to enhance operational transparency, reduce costs, and drive business growth through smart inventory control.

1.5 Scope

The StockWise (SW) is designed to streamline inventory tracking, order management, and stock monitoring for retail businesses. Its primary goal is to enhance operational efficiency by automating inventory processes, reducing stock discrepancies, and ensuring timely alerts for low-stock items. The system benefits businesses by minimizing inventory wastage, optimizing restocking decisions, and improving overall supply chain management. SW supports key business strategies such as cost reduction, operational transparency, and improved decision-making through data analytics. By integrating features like real-time stock updates, sales reporting, and user-specific functionalities for admins, stock managers, and suppliers, it aligns with the corporate goal of leveraging technology to drive business success and customer satisfaction.

2 Design Considerations

This section describes many of the issues which need to be addressed or resolved before attempting to devise a complete design solution. In other words, this section is used to formally set the groundwork for the system design.

2.1 Assumptions and Dependencies

While foundational assumptions and dependencies (e.g., PostgreSQL, external API compliance, stable internet connectivity) are detailed in the SRS, the following are design-specific assumptions that influence how the system architecture will be implemented:

- **Modular Architecture Assumption**: It is assumed that each major component (e.g., inventory, order management, reporting) can be modularized and developed independently using a microservice or loosely coupled design.
- **RESTful Integration Assumption**: External systems (POS, ERP, payment gateways) will reliably support RESTful APIs, enabling smooth integration without extensive custom adapters.
- **Component Reusability**: Common functionalities (e.g., user authentication, input validation, logging) will be encapsulated as reusable services or libraries.
- **UI Framework Compatibility**: The ReactJS frontend will remain compatible with third-party visualization libraries (e.g., Chart.js or D3.js) for rendering dynamic reports and dashboards.
- **Event-Driven Communication**: It is assumed that the backend can utilize event queues or message brokers (e.g., RabbitMQ or WebSockets) for future scalability in alerting and real-time updates.

2.2 Risks and Volatile Areas

The following are the most significant design-related risks and areas prone to change:

1. Integration Fragility

- **Risk**: External system APIs (e.g., supplier platforms, payment gateways) may change without notice or fail to meet expected standards.
- **Mitigation**: Use an **API Gateway with versioning support** and design wrapper modules around external services to absorb changes without affecting core logic.

2. Requirement Evolution

- **Risk**: Business needs may evolve (e.g., support for multiple warehouses, dynamic pricing, or region-specific compliance).
- **Mitigation**: Implement a **layered architecture** that cleanly separates business logic from data access and presentation layers, making it easier to accommodate requirement changes.

3. Performance Bottlenecks

- **Risk**: High concurrency (up to 10,000 users) and large-scale inventory datasets may stress system performance.
- **Mitigation**: Adopt **asynchronous processing**, database indexing strategies, and **horizontal scaling** via container orchestration (e.g., Docker + Kubernetes).

4. Security Enhancements

- **Risk**: Security threats may evolve, requiring rapid updates to authentication, encryption, or data validation mechanisms.
- **Mitigation**: Design a **pluggable security layer** using OAuth 2.0 and middleware-based validation, ensuring security can be patched without affecting business features.

5. Frontend Complexity Growth

- **Risk**: Adding more user roles or features may overcomplicate the UI and reduce usability.
- **Mitigation**: Follow **component-based UI development** using React hooks and context APIs, ensuring maintainable and scalable user interface logic.

3 System Architecture

The system architecture of StockWise is designed to ensure modularity, scalability, and maintainability. The system is divided into subsystems that work together to deliver features such as inventory tracking, order processing, reporting, and role-based access, while ensuring secure and efficient data flow.

3.1 System Level Architecture

At a high level, the StockWise system is decomposed into the following major elements:

- **Frontend Subsystem (Client App)**
  Developed using ReactJS, this provides the user interface for all roles: Admin, Stock Manager, Supplier, and Customer. It communicates with backend services via HTTP(S) requests.
- **Backend Subsystem (API Server)**
  Built with Node.js, this handles all business logic, request processing, security, and user authentication. It follows RESTful API design principles.
- **Database Subsystem**
  A PostgreSQL database stores persistent data including product information, user roles, order details, sales reports, and system logs.
- **External Integration Interfaces**
  Interfaces are established with third-party systems like POS platforms, ERP systems, and payment gateways via RESTful APIs. These allow real-time synchronization of inventory, order processing, and financial transactions.
- **Notification Subsystem**
  Responsible for sending alerts (e.g., low stock, order status changes) via email or system notifications.
- **Security Module**
  Enforces user authentication (including 2FA), authorization, encryption, and logging of sensitive operations.

Relationships Between Elements

- The frontend interacts solely with the backend via API calls.
- The backend accesses the database using a data access abstraction layer.
- External integrations are abstracted into service modules within the backend.
- All system-wide alerts and logs are passed through centralized services for consistency.

Physical Design Considerations

- Frontend executes on client machines (browsers/mobile).
- Backend and database components run on cloud-hosted Linux servers.
- Load balancers and API gateways will distribute requests and manage versioning.

Global Strategies

- **Error Handling**: Centralized exception handling with user-friendly messages on the frontend and detailed logs for developers.
- **Logging**: Audit and transaction logs for key events stored securely in the database.
- **Security**: Encryption of all sensitive data at rest and in transit; access restricted via RBAC and secure authentication.

3.2 Software Architecture

StockWise follows a **Three-Tier Architecture** pattern to clearly separate concerns and support maintainability and scalability:

1. User Interface Layer

- Built using **ReactJS**.
- Manages all user interactions: browsing products, managing inventory, generating reports.
- Sends requests and receives JSON responses via API endpoints.

2. Middle Tier (Business Logic Layer)

- Built in **Node.js/Express**.
- Acts as the control center — handling user authentication, request routing, inventory updates, reporting logic, and enforcement of business rules.
- Contains service modules for different subsystems (e.g., InventoryService, ReportService, UserService).

3. Data Access Layer

- Interacts with the **PostgreSQL database**.
- Handles all CRUD operations securely.
- Uses parameterized queries and ORM (e.g., Sequelize or Prisma) to reduce SQL injection risk.

4 Design Strategy

The design of StockWise is driven by the goals of scalability, maintainability, modularity, and user-centric performance. Key design strategies were selected to support long-term extensibility, promote reusability of components, and provide a secure, responsive experience across user roles. The following design considerations guided the high-level architectural and organizational decisions.

Future System Extension or Enhancement

To accommodate potential feature additions—such as multi-warehouse support, AI-powered demand forecasting, or mobile-first modules—the system follows a modular architecture. Each major functionality (e.g., inventory, orders, reporting) is encapsulated within its own service or module, allowing independent updates without affecting the rest of the system.

Strategy Used:

- Separation of concerns with well-defined interfaces between layers.
- Use of microservice-like patterns (even if deployed monolithically initially) to facilitate future decoupling.

Trade-off: Slight increase in initial development overhead due to the need for well-structured APIs and contracts between modules.

System Reuse

Code reusability is maximized through:

- **Component-based frontend**: Reusable React components (e.g., buttons, modals, input forms) are used across all user interfaces.
- **Shared middleware**: Authentication, logging, and error handling are centralized in the backend and reused across all routes.
- **Utility libraries**: Common logic (e.g., date formatting, report generation) is abstracted into helper modules.

Strategy Used:

- DRY (Don't Repeat Yourself) principle.
- Utility and shared service modules to isolate repeated logic.

Trade-off: Slight increase in codebase complexity, requiring good documentation and consistent module interfaces.

User Interface Paradigms

The user interface is built using modern Single Page Application (SPA) paradigms with ReactJS, ensuring:

- Fast, responsive interactions with minimal page reloads.
- Dynamic content updates for real-time stock levels and order statuses.
- Clear role-based views: Admins, Stock Managers, Suppliers, and Customers all get personalized UI experiences.

Strategy Used:

- Role-based component rendering and route protection.
- Client-side routing and state management (e.g., using Redux or React Context API).

Trade-off: Heavier frontend development workload; however, this pays off in a smoother UX and reduced server load.

Data Management

All core data (users, inventory, orders, etc.) is stored in a PostgreSQL relational database, chosen for its:

- ACID compliance for transaction safety.
- Support for complex queries needed for reporting.
- Horizontal scaling capability with tools like Citus (if needed in future).

Strategy Used:

- Centralized storage with normalized schema.
- Daily automated backups and role-based access at the DB level.

Trade-off: May need optimization for large datasets (e.g., indexing, caching) as the system scales.

Concurrency and Synchronization

The system supports concurrent access by thousands of users (e.g., stock managers updating items while customers place orders). Concurrency is handled through:

- Optimistic concurrency control for operations like stock updates to minimize lock contention.
- Asynchronous request handling in Node.js using non-blocking I/O.
- Real-time UI updates via WebSockets or polling for critical events (e.g., low stock alerts).

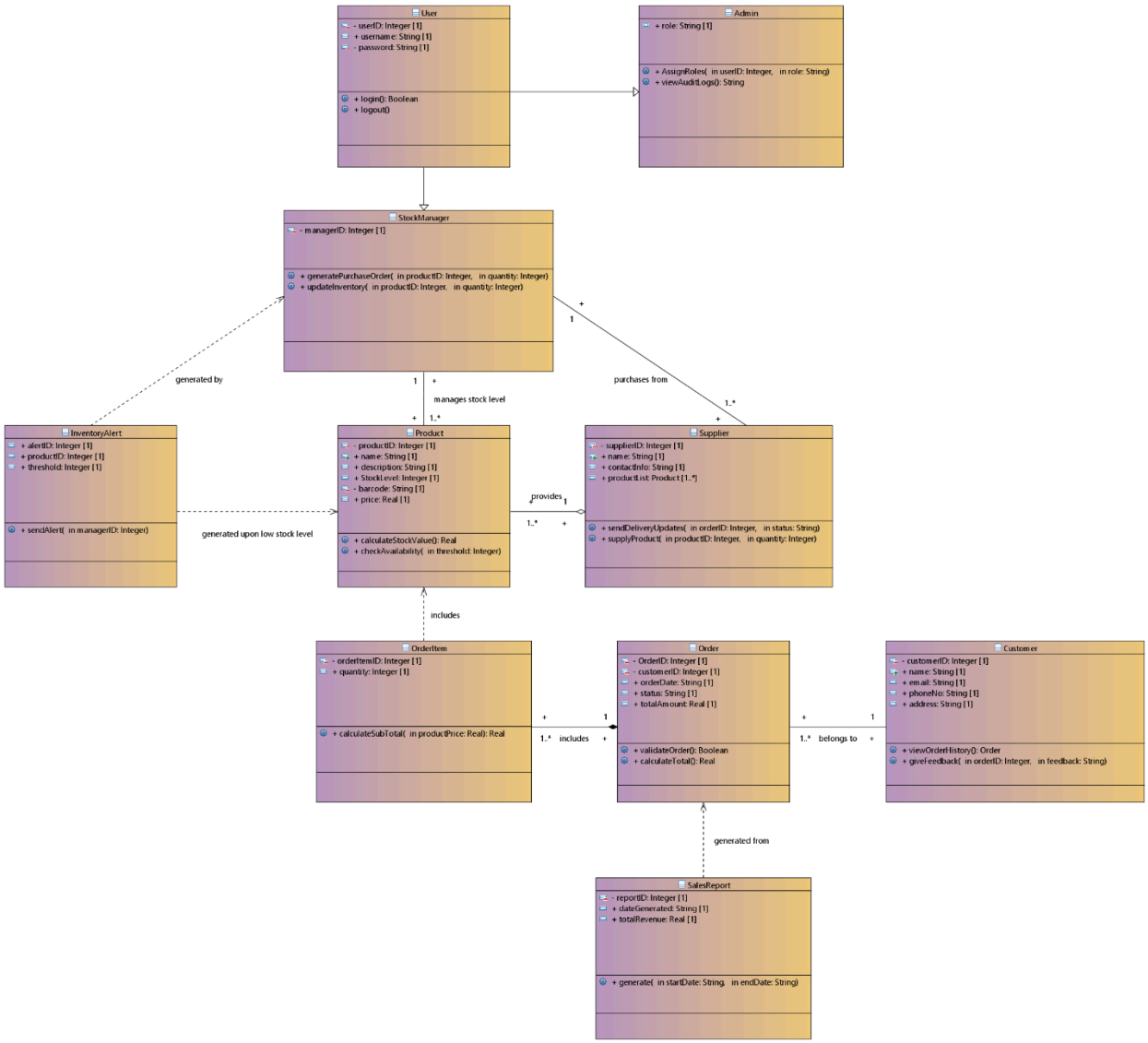
Strategy Used:

- Use of **transactional operations** in the database.
- Middleware queues for non-critical asynchronous tasks (e.g., email notifications).

Trade-off: Slightly more complex backend logic, but essential for reliability and scale.

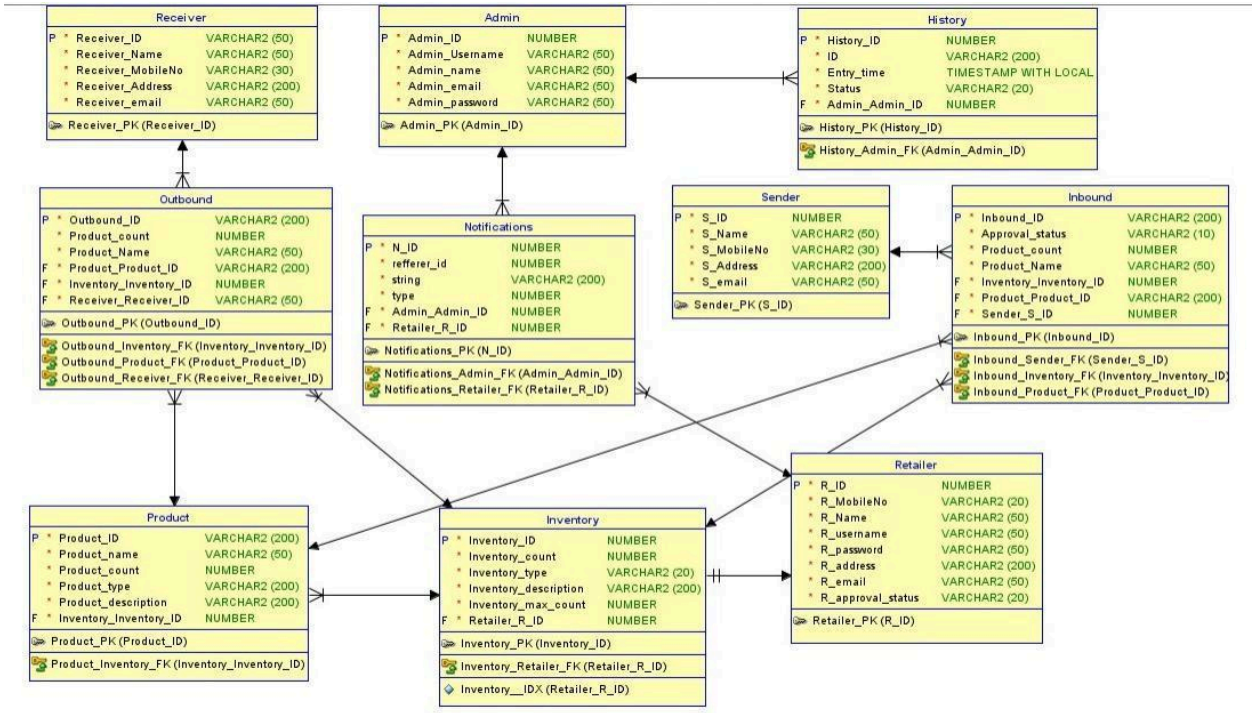
5 Detailed System Design

Class Diagram:



5.1 Database Design

5.1.1 ER Diagram



5.1.2 Data Dictionary

Receiver Table

Name: Receiver
Alias: None
Where-used/how-used: Stores receiver details for outbound deliveries
Content description: Notation for representing receiver content

Column Name	Description	Type	Length	Nullable	Default Value	Key Type
Receiver_ID	Unique Receiver ID	NUMBER		No		PK
Receiver_Name	Name of Receiver	VARCHAR2	200	Yes	NULL	
Receiver_CNIC	CNIC of Receiver	VARCHAR2	20	Yes	NULL	
Receiver_Phone	Phone Number	VARCHAR2	20	Yes	NULL	

Receiver_Address	Address	VARCHAR2	200	Yes	NULL

Admin Table

Name: Admin
Alias: None
Where-used/how-used: Manages users, notifications, history, etc.
Content description: Notation for representing admin content

Column Name	Description	Type	Length	Nullable	Default Value	Key Type
Admin_ID	Unique Admin ID	NUMBER		No		PK
Admin_Username	Admin Username	VARCHAR2	200	Yes	NULL	
Admin_email	Email of Admin	VARCHAR2	200	Yes	NULL	
Admin_name	Name of Admin	VARCHAR2	20	Yes	NULL	
Admin_password	Password	VARCHAR2	200	Yes	NULL	

History Table

Name: History
Alias: None
Where-used/how-used: Logs actions performed by admins
Content description: Notation for representing history content

Column Name	Description	Type	Length	Nullable	Default Value	Key Type
History_ID	Unique History ID	NUMBER		No		PK
History_data	Action Description	VARCHAR2	200	Yes	NULL	

Entry_time	Timestamp of Action	TIMESTAMP	Yes	NULL	
Admin_Admin_ID	Linked Admin ID	NUMBER	Yes	NULL	FK

Sender Table

Name: Sender
Alias: None
Where-used/how-used: Sender details for inbound deliveries
Content description: Notation for representing sender content

Column Name	Description	Type	Length	Nullable	Default Value	Key Type
S_ID	Unique Sender ID	NUMBER		No		PK
S_name	Sender Name	VARCHAR2	200	Yes	NULL	
S_CNIC	CNIC of Sender	VARCHAR2	20	Yes	NULL	
S_Phone	Phone Number	VARCHAR2	20	Yes	NULL	
S_Address	Address	VARCHAR2	200	Yes	NULL	

Inbound Table

Name: Inbound
Alias: None
Where-used/how-used: Details of products received from a sender
Content description: Notation for representing inbound content

Column Name	Description	Type	Length	Nullable	Default Value	Key Type
Inbound_ID	Unique Inbound ID	VARCHAR2	200	No		PK
Product_name	Name of Product	VARCHAR2	200	Yes	NULL	

Product_Quantity	Quantity of Product	NUMBER		Yes	NULL	
Product_Price	Price per Product	NUMBER		Yes	NULL	
Product_Manufacture	Manufacturing Date	VARCHAR2	20	Yes	NULL	
Inbound_S_ID	Sender ID	NUMBER		Yes	NULL	FK

Retailer Table

Name: Retailer
Alias: None
Where-used/how-used: Stores retailer information
Content description: Notation for representing retailer content

Column Name	Description	Type	Length	Nullable	Default Value	Key Type
Retailer_ID	Unique Retailer ID	NUMBER		No		PK
R_username	Retailer Username	VARCHAR2	200	Yes	NULL	
R_password	Retailer Password	VARCHAR2	200	Yes	NULL	
R_name	Retailer Name	VARCHAR2	200	Yes	NULL	
R_phone	Phone Number	VARCHAR2	20	Yes	NULL	
R_email	Email	VARCHAR2	200	Yes	NULL	
R_approval_status	Approval Status	VARCHAR2	20	Yes	NULL	

Product Table

Name: Product
Alias: None

Where-used/how-used: Stores product details
Content description: Notation for representing product content

Column Name	Description	Type	Length	Nullable	Default Value	Key Type
Product_ID	Unique Product ID	VARCHAR2	200	No		PK
Product_name	Name of Product	VARCHAR2	200	Yes	NULL	
Product_Price	Product Price	NUMBER		Yes	NULL	

Inventory Table

Name: Inventory
Alias: None
Where-used/how-used: Stores inventory stock records
Content description: Notation for representing inventory content

Column Name	Description	Type	Length	Nullable	Default Value	Key Type
Inventory_ID	Unique Inventory ID	NUMBER		No		PK
Inventory_Quantity	Quantity Available	NUMBER		Yes	NULL	
Inventory_Price	Price	NUMBER		Yes	NULL	
Product_Product_ID	Product ID	VARCHAR2	200	Yes	NULL	FK
Retailer_Retailer_ID	Retailer ID	NUMBER	Yes	NULL	FK	

Outbound Table

Name: Outbound
Alias: None

Where-used/how-used: Records dispatched items to receivers
Content description: Notation for representing outbound content

Column Name	Description	Type	Length	Nullable	Default Value	Key Type
Outbound_ID	Unique Outbound ID	VARCHAR2	200	No		PK
Product_name	Product Name	VARCHAR2	200	Yes	NULL	
Product_Quantity	Quantity	NUMBER		Yes	NULL	
Product_Price	Price	NUMBER		Yes	NULL	
Product_Manufacture	Manufacture Date	VARCHAR2	20	Yes	NULL	
Outbound_Inventory_ID	Inventory ID	NUMBER	Yes	NULL	FK	
Outbound_Product_ID	Product ID	VARCHAR2	200	Yes	NULL	FK
Outbound_Receiver_ID	Receiver ID	NUMBER	Yes	NULL	FK	

Notification Table

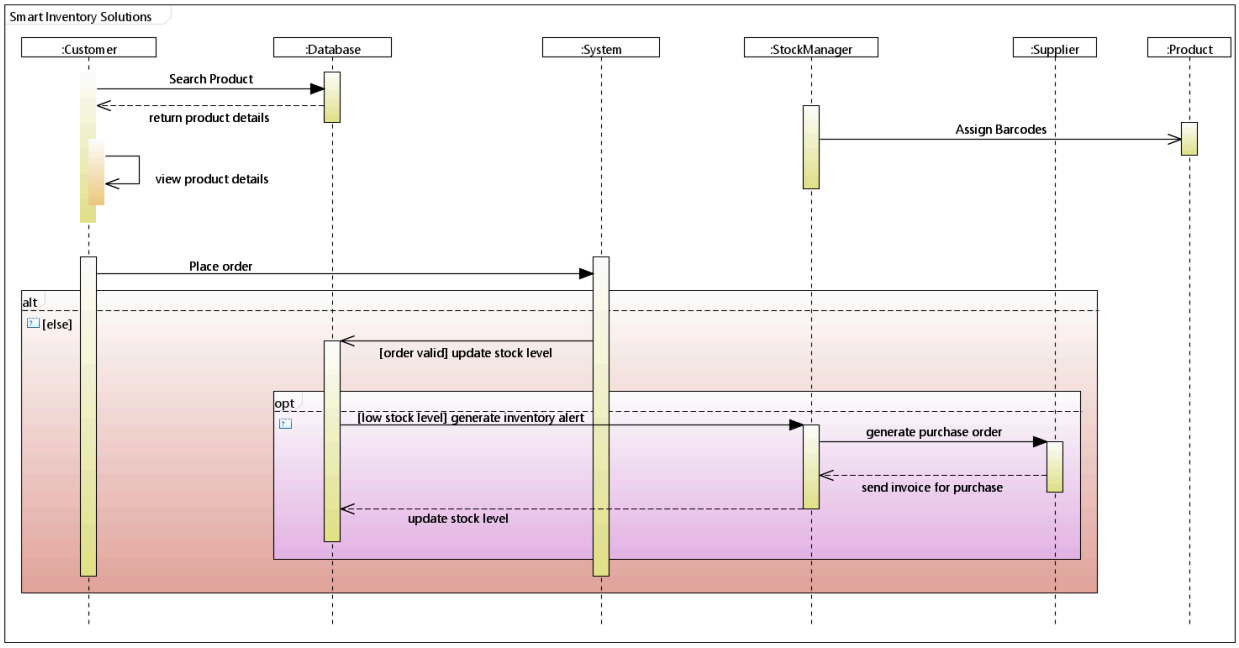
Name: Notification
Alias: None
Where-used/how-used: Alerts for admins and retailers
Content description: Notation for representing notification content

Column Name	Description	Type	Length	Nullable	Default Value	Key Type
N_ID	Unique Notification ID	NUMBER		No		PK
msg	Notification Text	VARCHAR2	200	Yes	NULL	
date	Notification Date	VARCHAR2	20	Yes	NULL	

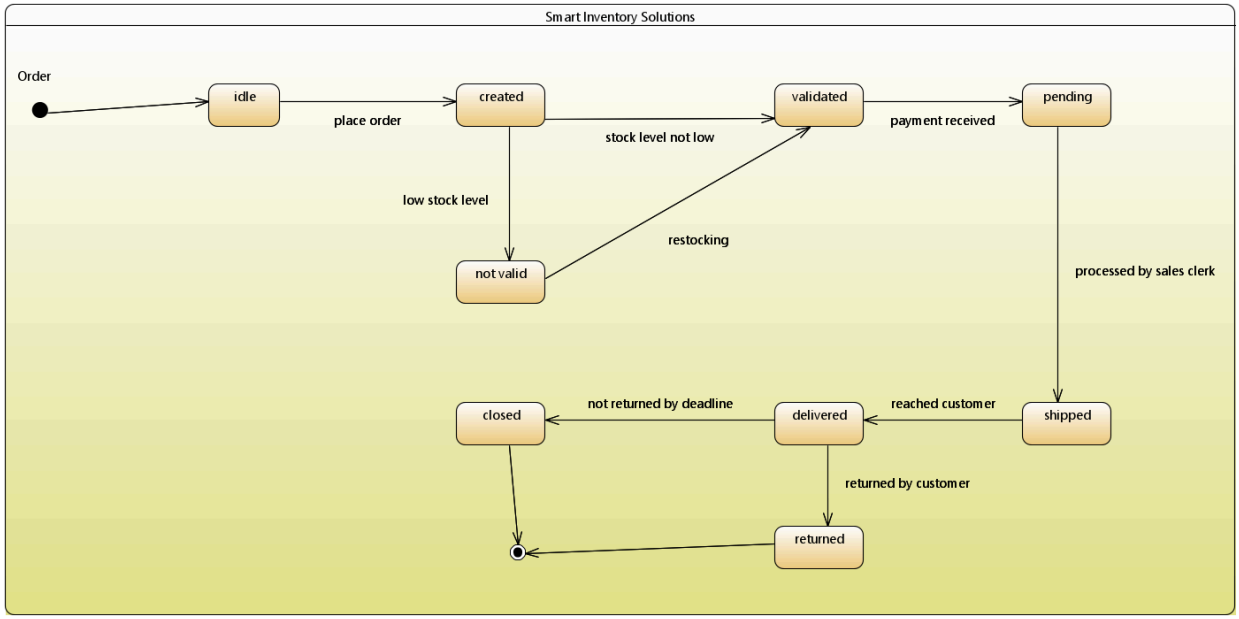
Admin_Admin_ID	Admin ID	NUMBER	Yes	NULL	FK
Retailer_R_ID	Retailer ID	NUMBER	Yes	NULL	FK

5.2 Application Design

Sequence Diagram



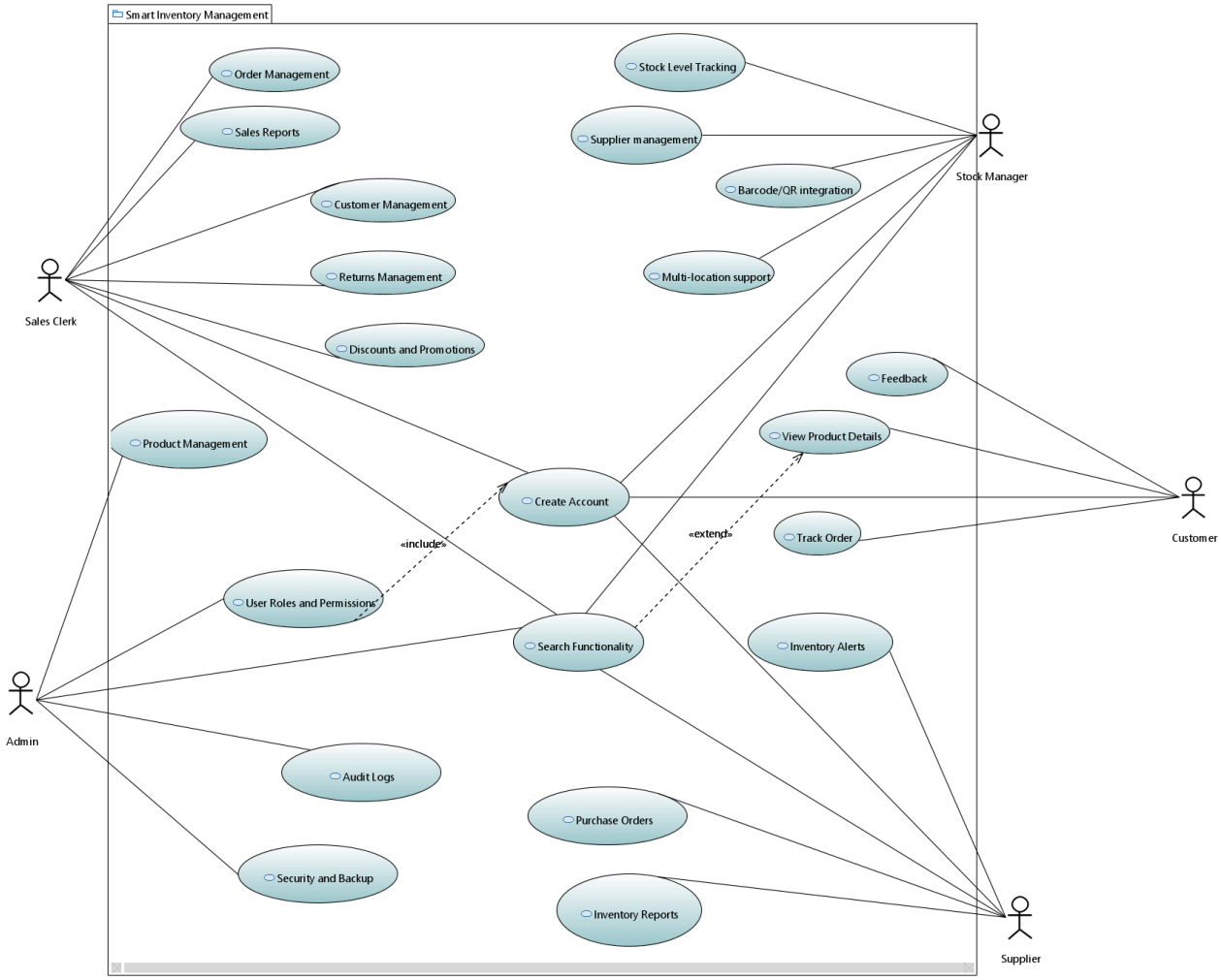
5.2.1 State Diagram



6 References

- **European Commission (2020).** ***General Data Protection Regulation (GDPR).***
  Retrieved from: https://gdpr-info.eu
- **Oracle (2020).** ***PostgreSQL Documentation (v13+).***
  Retrieved from: https://www.postgresql.org/docs/
- **Open Web Application Security Project (OWASP) (2021).** ***OWASP Top Ten Security Risks.***
  Retrieved from: https://owasp.org/www-project-top-ten/
- **Jovanovic, M., & Milinkovic, D. (2018).** ***Software Security: Threats and Vulnerabilities.***
  Springer Publishing.
- **International Organization for Standardization (ISO) (2018).** ***ISO/IEC 27001:2013 – Information Security Management Systems.***
  Retrieved from: https://www.iso.org/isoiec-27001-information-security.html
- **Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994).** ***Design Patterns: Elements of Reusable Object-Oriented Software.***
  Addison-Wesley Professional.
- **Pressman, R. S. (2014).** ***Software Engineering: A Practitioner's Approach (8th ed.).***
  McGraw-Hill Education.

7 Appendices



Appendix 1: Use Case Diagram

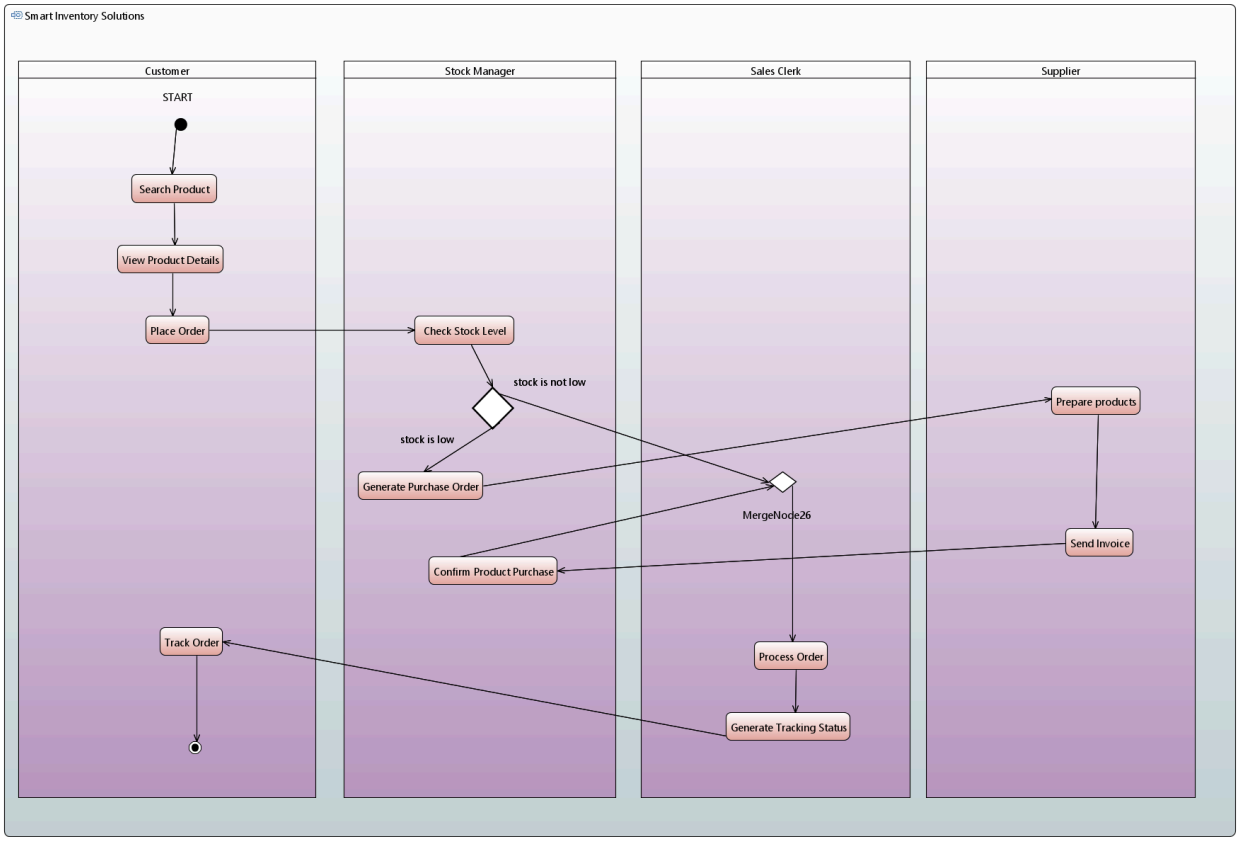


Diagram 2: Activity Diagram

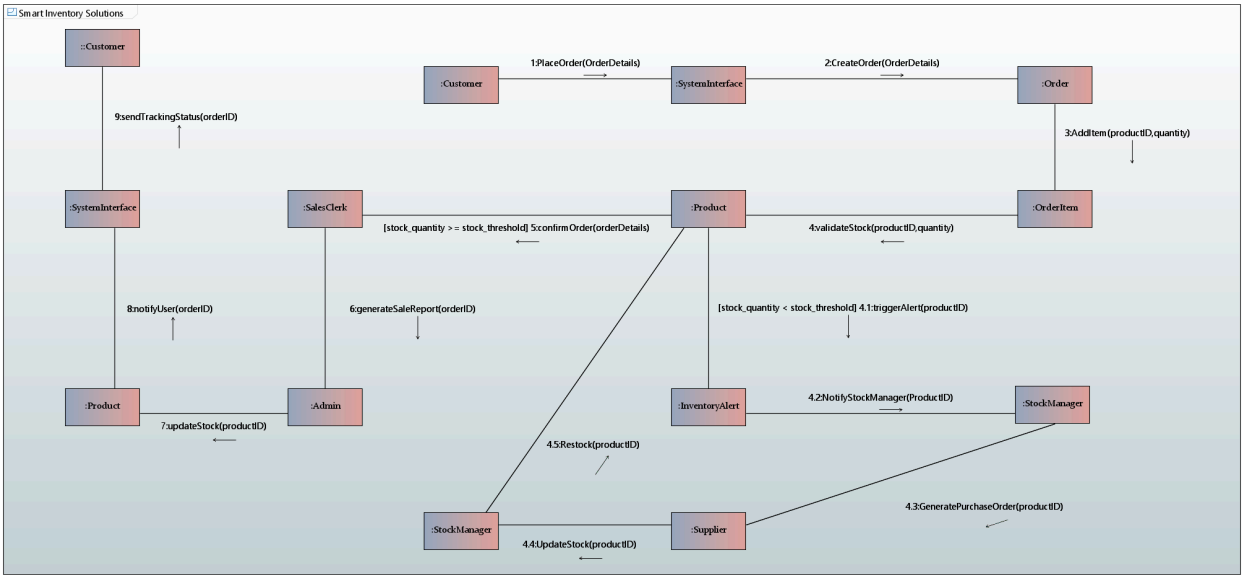


Diagram 3: Communication Diagram

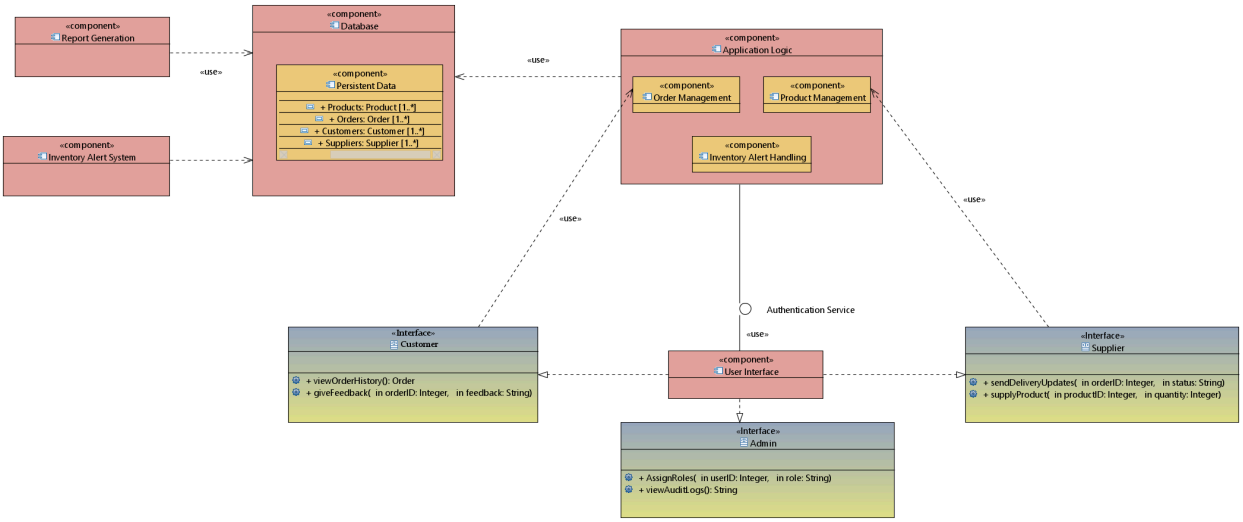


Diagram 4: Component Diagram

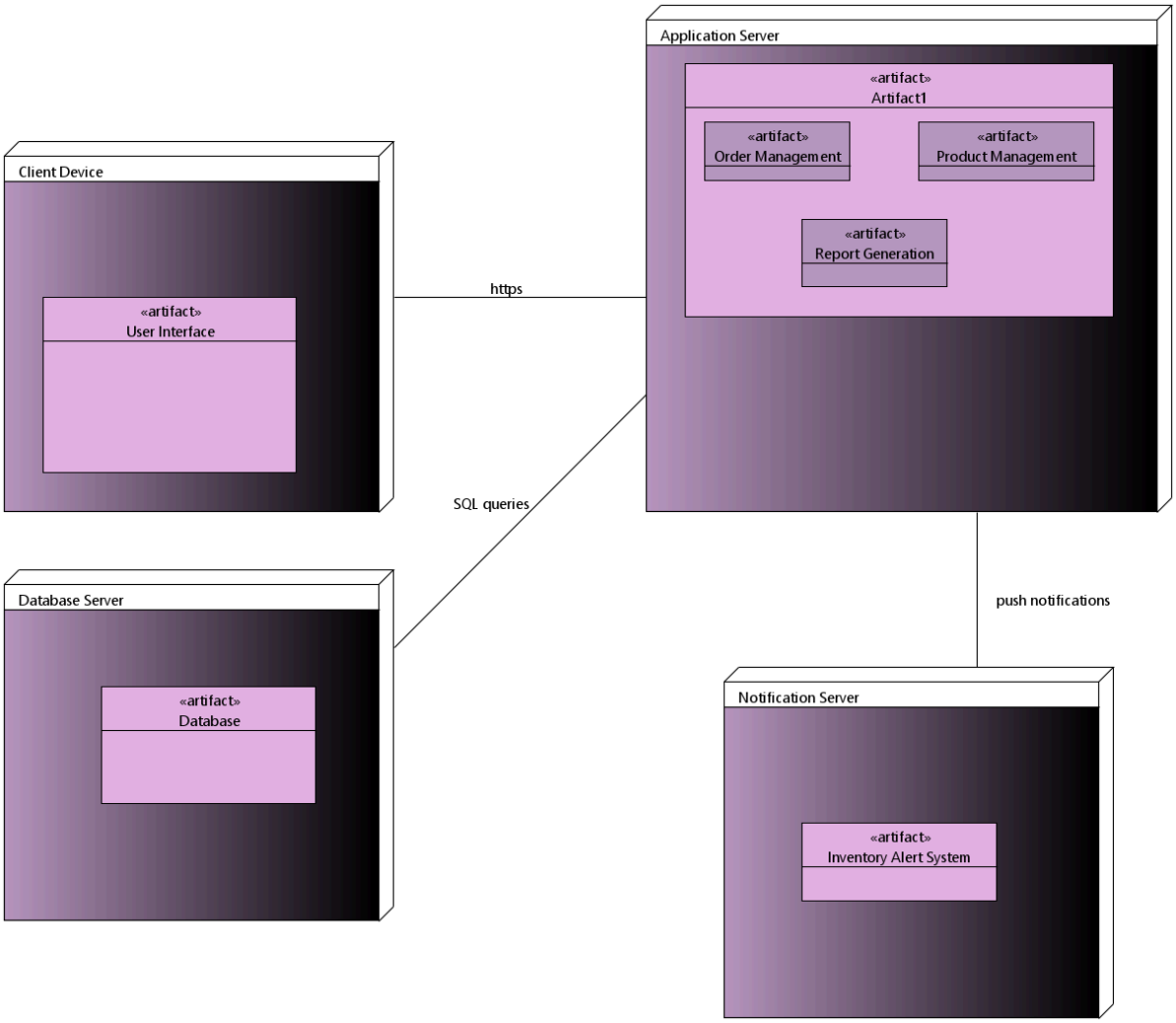


Diagram 5: Deployment Diagram