EEC 626 Software Engineering Project

Inventory Management Web Application

Prepared by
Wongpiwat Sa Ngiam 2883857
Biswadeep Mazumder 2860920
Uday Shankar Boddupalli 2868122

CONTENTS

6. Test Management

6.1. A complete list of system test cases
6.2. Traceability of test cases to use cases
6.3. Techniques used for test case generation
6.4. Test results and assessments
6.5. Defects reports

7. Conclusions

7.1. Outcomes of the project
7.2. Lessons learned
7.3. Future development

8. References

8.1 Project Repository
8.2 Additional Resources

----------xxxxxxxxxx----------

1. Introduction

1.1 Purpose and Scope

The purpose of this document is to provide a comprehensive and detailed overview of the **Inventory Management System** developed for **SR Business Group**. This document outlines the system architecture, database schema, workflows, and the development methodologies employed during the project lifecycle. It serves as a guide for stakeholders, future developers, and system administrators to understand, maintain, and enhance the application.

The system is designed to streamline inventory operations, including order management, product listing, and insights into sales performance, with a focus on scalability and reliability.

1.2 Product Overview

The **Inventory Management System** is a **cloud-based web application** tailored for small to medium-sized businesses to efficiently manage their inventory. It simplifies **inventory tracking**, **order processing**, and **supplier management** while providing actionable insights through data visualization.

Key Capabilities:

- **Order Management**: Create, modify, and track orders.
- **Product Management**: Add, update, or remove products from inventory.
- **Dashboard Analytics**: Visualize sales trends and monitor low-stock items via interactive charts.
- **Role-Based Access**: Different access levels for business owners, managers, and workers.
- **Automated Processes**: Future enhancements include low-stock alerts and shipment tracking.

Typical Scenarios of Use:

- A manager accesses the dashboard to view pending orders and inventory status.
- A worker adds newly received products to the inventory database.
- A business owner reviews sales trends to make data-driven restocking decisions.

1.4 Terms, Acronyms, and Abbreviations

Term/Acronym	Definition
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
CI/CD	Continuous Integration/Continuous Deployment
EF Core	Entity Framework Core
HTTP	Hypertext Transfer Protocol
OAuth	Open Authorization (Authentication Mechanism)
SQL	Structured Query Language
UAT	User Acceptance Testing
DTO	Data Transfer Object
MVC	Model-View-Controller (architecture pattern)
JWT	JSON Web Token (for authentication)
ORM	Object-Relational Mapping
DBMS	Database Management System
Azure App Service	Microsoft's cloud-based hosting platform
Swagger	An interactive API documentation tool
LINQ	Language Integrated Query (used in .NET for querying data)
EF Migrations	Entity Framework process for updating the database schema
Postman	A tool for API development and testing
SQL Server Management Studio (SSMS)	A tool for managing SQL Server databases
Azure DevOps	A Microsoft platform for CI/CD and project management
Azure Monitor	Tool for monitoring Azure applications and infrastructure
Azure Storage	Scalable cloud storage service by Azure
Entity	Represents a table in the database
Azure CLI	Command-line interface for managing Azure resources

React	A JavaScript library for building user interfaces
Next.js	A React framework for building server-side rendered applications
Axios	A JavaScript library for making HTTP requests
REST	Representational State Transfer (architectural style for APIs)
DNS	Domain Name System
CORS	Cross-Origin Resource Sharing (browser security feature)
VS Code	Visual Studio Code, a code editor
Azure SQL Database	A cloud-based relational database service by Microsoft
Geist Font	Font family used in Next.js projects
GitHub	A platform for version control and collaboration
Firewall	A network security system

2. Project Management Plan

2.1 Project Organization

The project is stored in a GitHub repository: GitHub Repository - Inventory Management.

- **Branch Structure**:
  - **Backend Branch**: Contains production-ready code for the backend developed using .NET Core.
  - **Frontend Branch**: Contains production-ready code for the frontend built with React (Next.js).

- **Project Management Tool**:
  Trello was used to manage the project development lifecycle. The project breakdown and detailed task assignments were established during the third week of development.

  - **Weekly Progress Meetings**:
    - **Monday Meetings**: Team discussions to plan weekly tasks.
    - **Friday Updates**: Progress demonstrations and updates to the professor.

- **Tracking Progress**:
  A comprehensive project tracker file was maintained to monitor progress, task completion, and pending activities.

2.2 Lifecycle Model Used

The project followed the **Kanban Agile Methodology**:

- Tasks were prepared and listed in a **Trello dashboard**, ensuring clear visibility and tracking of progress.

- Continuous updates and feedback loops were maintained to adapt to evolving requirements.

- Focused on iterative development with regular deliveries.

2.3 Risk Analysis

Identified Risks and Mitigation Strategies:

Risk	Impact	Likelihood	Mitigation Strategy
Scope Creep	Delays in deliverables	Medium	Strictly adhere to the finalized requirements and weekly goals.
Deployment Failures	Service interruptions	Low	Set up a staging environment for testing before production deployment.
Resource Unavailability	Missed milestones	High	Maintain backups and plan for resource allocation redundancy.
Technical Integration Challenges	Inconsistent workflows	Medium	Conduct regular integration testing and resolve blockers early.
Data Loss or Corruption	Major delays	Low	Regular backups and Azure SQL Database recovery mechanisms.
Team Communication Gaps	Misalignment of goals	Medium	Weekly meetings and Trello updates to ensure synchronization.

2.4 Hardware and Software Resource Requirements

Hardware Requirements:

Development machines (Windows/Mac/Linux),
minimum 8GB RAM, 256GB SSD.

Software Requirements:

1. **Backend**:
   - .NET 8 Framework.
   - Entity Framework Core 7.
   - SQL Server Management Studio for database management.

2. **Frontend**:
   - Next.js
   - React
   - Node.js for running the frontend.

3. **Development Tools**:
   - Visual Studio Code and WebStorm for coding.
   - Postman for API testing.
   - Git for version control.

4. **Testing Environment**:
   - Supported browsers: Chrome, Firefox, Safari.
   - Manual testing setup with test cases documented in spreadsheets.

2.5 Deliverables and Schedule

Sprint Number	Start Date	End Date	Sprint Goal	Tasks	Status
0	September 2, 2024	September 8, 2024	Planning and Setup	Finalize requirements, Set up development environment	Completed
1	September 9, 2024	September 22, 2024	Basic Framework and UI Design	Implement basic structure, Design dashboard layout, Set up database	Completed
2	September 23, 2024	October 6, 2024	Backend Development and Data Integration	Develop backend logic, Integrate frontend with APIs, Render dynamic cards	Completed
3	October 7, 2024	October 20, 2024	Chart Implementation and API Enhancements	Implement charts, Enhance APIs, Test data flow	Completed
4	October 21, 2024	November 3, 2024	Detailed Pages and Routing	Implement routing, Develop detailed views	Completed
5	November 4, 2024	November 17, 2024	Testing, Refinement, and User Feedback	Conduct testing, Gather feedback, Refine features	Completed
6	November 18, 2024	December 1, 2024	Final Adjustments and Deployment	Make final adjustments, Prepare documentation, Deploy to production	Completed

3.Requirement Specifications

3.1 Stakeholders for the System

The stakeholders for the Inventory Management System include:

 Business Owners

- **Responsibilities:** Oversee the inventory system, approve major orders, and analyze sales trends using the dashboard.
- **Risks:** Dependence on accurate data and timely alerts; any system downtime could lead to operational delays.

 Managers

- **Responsibilities:** Create and manage orders, ensure inventory stock levels are maintained, and resolve product discrepancies.
- **Risks:** Incorrect or incomplete data entry may result in order fulfillment errors or stock shortages.

 Workers

- **Responsibilities:** Update inventory lists, record incoming stock, and report low-stock alerts.
- **Risks: High dependency on system usability;** complex interfaces may lead to data entry mistakes.

 System Administrators

- **Responsibilities:** Maintain the backend system, manage database operations, and handle server deployments.
- **Risks:** Security vulnerabilities or database misconfigurations may disrupt the system's operations.

 Clients/Users

- **Responsibilities:** Use the system for order fulfillment and inventory tracking.
- **Risks:** A user-friendly interface is critical; poor UX may impact adoption.

3.2 Use Cases

1. **Place an Order**
   - A manager creates a new order for inventory stock replenishment.

2. **Track Inventory**
   - A worker views current stock levels and identifies low-stock items.

3. **Modify Product Details**
   - A manager updates item details, such as price or stock level.

4. **Generate Order Report**
   - The business owner generates a report of order trends over the past month.

5. **Cancel an Order**
   - The system administrator performs routine updates and checks on the database.

6. **System Maintenance**
   - The system administrator performs routine updates and checks on the database.

3.2.2 Graphic Use Case Model

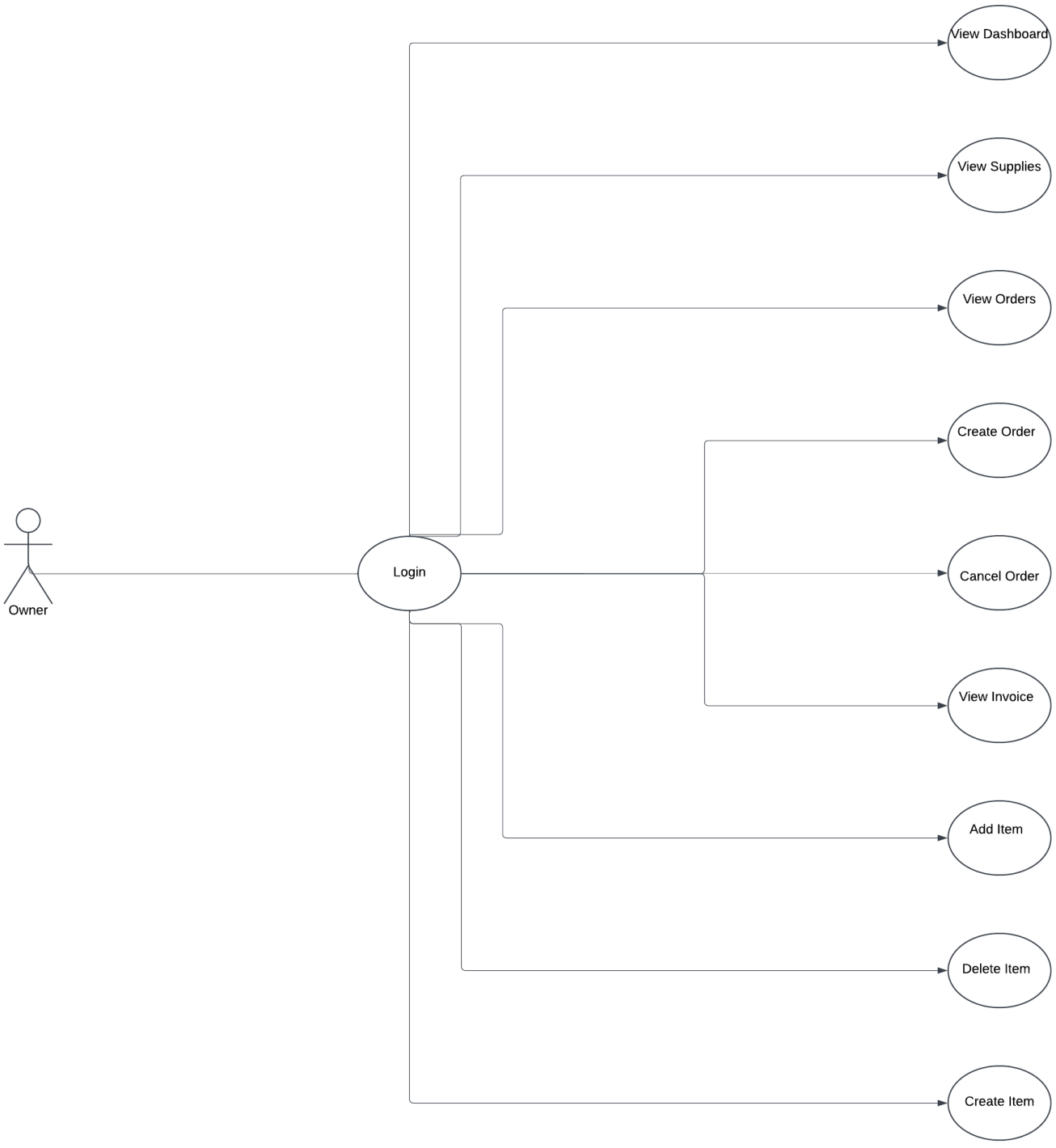


Figure 1: Use case Diagrams

3.2.3 Textual Description for Each Use Case

1. Place an Order

Actors: Manager
Preconditions: The manager is logged into the system, and inventory data is accessible.
Description: The manager initiates the order creation process by navigating to the order management module. They select items from the inventory list, specify the required quantities, and confirm the order. The system validates stock availability and processes the order by updating the inventory and adding the order to the database. If the stock is insufficient or the database is inaccessible, the system provides appropriate error messages to the manager.
Postconditions: The new order is created, updated in the system, and visible on the dashboard.

2. Track Inventory

Actors: Worker
Preconditions: The worker is logged into the system and has permissions to access inventory data.
Description: The worker navigates to the inventory tracking module to view the current stock levels. The system highlights items flagged as low-stock and generates alerts for critical shortages. The worker uses this information to plan and prioritize restocking efforts. If there are synchronization issues or data-loading failures, the system alerts the worker and logs the error for resolution.
Postconditions: The worker obtains accurate, up-to-date inventory information, with alerts generated for low-stock items as needed.

3. Modify Product Details

Actors: Manager
Preconditions: The manager is logged into the system and has permissions to edit product details.
Description: The manager accesses the product management module to edit item attributes such as price, description, or stock level. After making changes, they save the updates, prompting the system to validate the inputs and commit the changes to the database. In cases of invalid input (e.g., incorrect price format) or database update failures, the system displays an error message for corrective action.
Postconditions: The updated product details are saved and reflected in the system, ensuring accurate and consistent inventory data.

4. Generate Order Report

Actors: Business Owner

Preconditions: The business owner is logged into the system and has access to the reporting module.

Description: The owner selects a date range from the reporting module to generate a detailed report of order trends. The system processes the request, retrieves relevant data, and generates the report, which includes metrics like order volumes, patterns, and sales performance. The report is made available for download in CSV format. If the selected date range lacks sufficient data, the system notifies the owner.

Postconditions: A report is successfully generated and made available for review and download.

5. Cancel an Order

Actors: Manager, System Administrator

Preconditions: The user is logged into the system and has the necessary permissions to cancel orders.

Description: The user identifies an order that needs cancellation, navigates to the order management module, and selects the specific order. After providing a reason for the cancellation, the system processes the request, updates the order status to "Canceled," and adjusts inventory if necessary. If the order is invalid or already completed, the system prevents the cancellation and informs the user.

Postconditions: The order is canceled, with its status updated and the cancellation reason logged.

7. System Maintenance

Actors: System Administrator

Preconditions: The administrator has access to the backend tools and permissions to perform maintenance tasks.

Description: The administrator performs regular maintenance activities such as applying updates, creating database backups, and monitoring server health. Any identified issues are resolved or logged for follow-up. If system errors occur during maintenance (e.g., server downtime or backup failures), the administrator implements contingency measures and restores system functionality.

Postconditions: The system remains secure, operational, and optimized for performance.

3.3 Rationale for Use Case Model

The use case model is designed to align with the real-world tasks and responsibilities of stakeholders while addressing potential system requirements. By focusing on key user interactions, the model ensures that the system provides meaningful functionality, fosters efficient workflows, and supports critical decision-making processes.

3.4 Non-Functional Requirements

- **Performance**
  1. The system must handle up to 200 concurrent users without any noticeable latency.
  2. API response time for CRUD operations should not exceed 500 milliseconds under normal load.
  3. Dashboards and charts should load in under 2 seconds with real-time data updates.
- **Scalability**
  1. The system must scale horizontally to support increased user demands and data growth.
  2. The database should be able to handle up to 10 million rows without performance degradation.
  3. The CI/CD pipeline must support deployment to multiple environments (e.g., staging, production).
- **Usability**
  1. The user interface should adhere to WCAG 2.1 standards for accessibility.
  2. The system must be responsive and functional on devices of varying screen sizes (desktop, tablet, mobile).
  3. Tooltips and error messages should be provided for all user inputs to improve usability.
- **Security**
  1. All data exchanges between the frontend and backend must use HTTPS and be encrypted.
  2. Role-based access control must restrict sensitive operations to authorized users only.
  3. Database access must be limited to specific IPs through firewall rules.
- **Reliability**
  1. The system must achieve 99.9% uptime over any 30-day period.
  2. Automated error logging and monitoring tools (e.g., Azure Monitor) must be in place to detect and resolve issues.
  3. Backup processes should run daily, with retention of backups for 30 days.
- **Maintainability**
  1. The codebase must follow a modular architecture (for easy debugging and extension.

2. Documentation must include clear API endpoints, database schema, and deployment steps.

- **Portability**
  1. The system must be deployable on any cloud service provider if migration is required.
  2. Configuration files (e.g., connection strings) must be environment-agnostic and parameterized.
- **Interoperability**
  1. The backend API must adhere to REST standards, allowing integration with third-party systems.
  2. Exported reports and data must be in common formats (e.g., CSV).
- **Auditability**
  1. All user actions (e.g., login, order creation) must be logged with timestamps for audit purposes.
  2. Change history for critical entities (e.g., orders, users) must be tracked and accessible.
- **Extensibility**
  1. The system must be designed to accommodate future features, such as integrating shipment tracking or sales systems.
  2. Additional modules should be addable without major refactoring of existing code.

4. Architecture

4.1 Architectural Styles Used

The Inventory Management System adopts the Model-View-Controller (MVC) architectural style for the backend and follows a component-based architecture for the frontend.

- **Backend (MVC Architecture):**

  The MVC style is designed to separate concerns within the application:
  - **Model**: Represents the database schema and handles data interaction through Entity Framework Core. This ensures that the database logic is decoupled from the application logic.
  - **View**: In this case, the frontend serves as the "View," consuming API responses to render user interfaces.
  - **Controller**: Processes incoming HTTP requests, invokes appropriate business logic, and returns responses to the frontend.

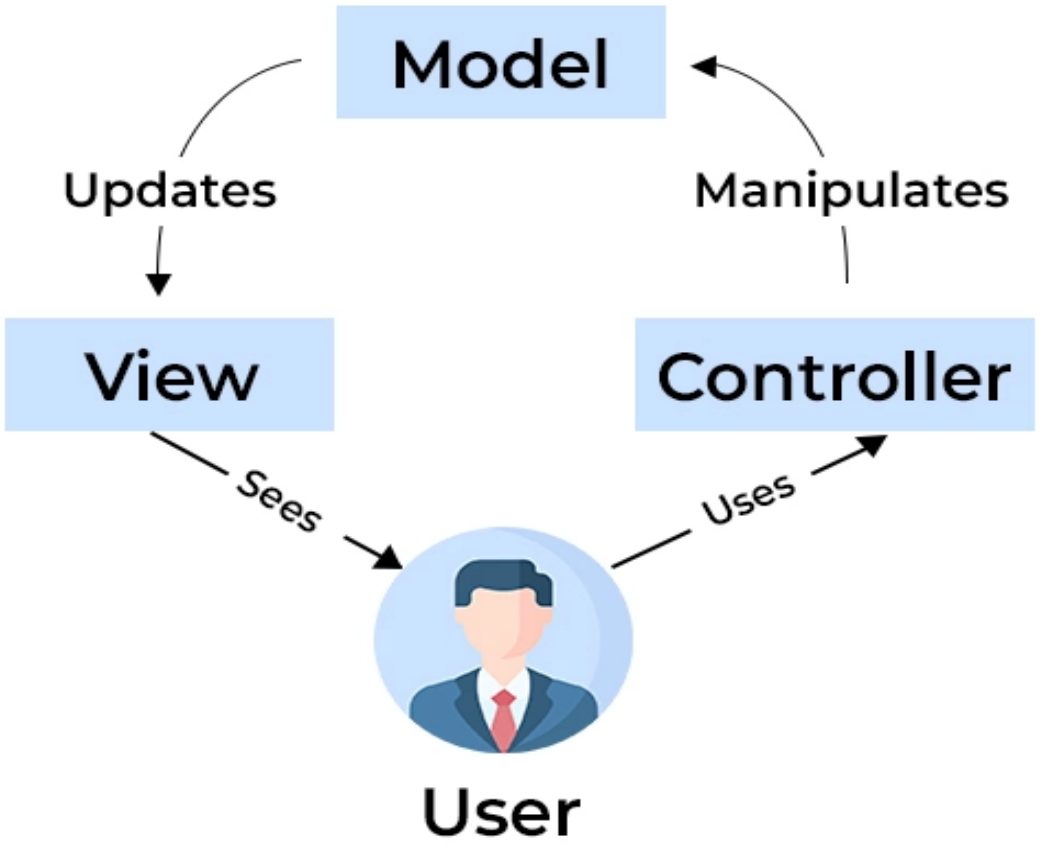


Figure 2: MVC simple diagram

- **Benefits**:
  - Modularity: Each layer can be developed, tested, and maintained independently.
  - Scalability: The separation of concerns ensures that the system can handle increased complexity without significant refactoring.
  - Testability: Business logic can be tested in isolation, improving the reliability of the system.

- **Frontend (Component-Based Architecture):**
  The frontend, built with React (Next.js), is designed using reusable UI components. Each component encapsulates logic and presentation for specific functionalities such as displaying a dashboard card, managing inventory lists, or handling order interactions.

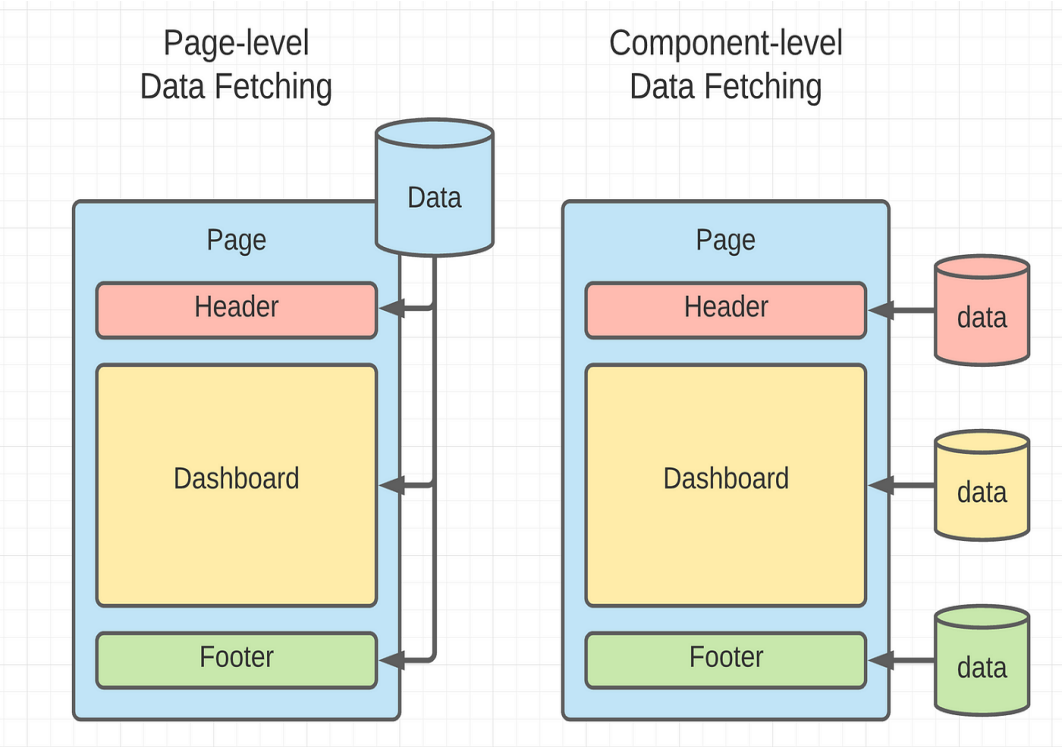


Figure 3: page vs component level data fetching simple diagram

- **Benefits**:
  - **Reusability**: Components can be reused across the application, reducing development effort.
  - **Responsiveness**: The architecture ensures that the application is adaptive across various devices.
  - **Maintainability**: Components are modular, making it easier to debug and enhance specific features.

4.2 Architectural Model

The architecture consists of interconnected components that work together to deliver the system's functionality.

- **Frontend to Backend Interaction**:
  - The React app uses **Axios** to send HTTP requests to the backend API, which is hosted on an Azure App Service.
  - The backend processes these requests and interacts with the Azure SQL Database via **Entity Framework Core** to fetch or update data.

- **Component Interactions**:
  Frontend (Next.js App):
  - Handles user interactions and sends requests to the backend.
  - Consumes API responses to display data dynamically.

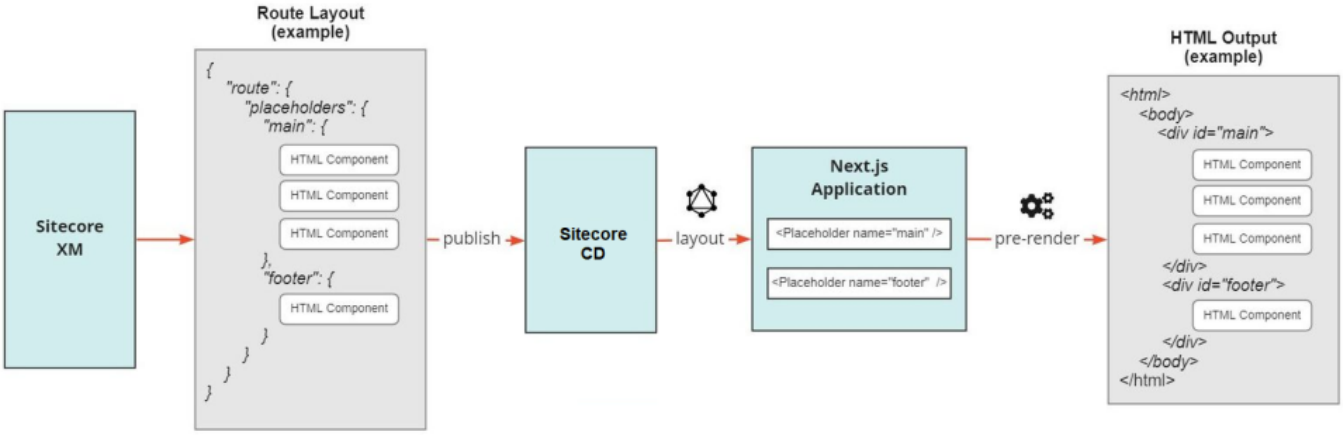


Figure 4: NEXT.js working diagram

Backend (ASP.NET Core API):

- Controllers handle incoming requests and route them to the appropriate services.
- Services execute business logic and call repositories for database interactions.
- Repositories abstract Entity Framework operations, ensuring database access is efficient and secure.

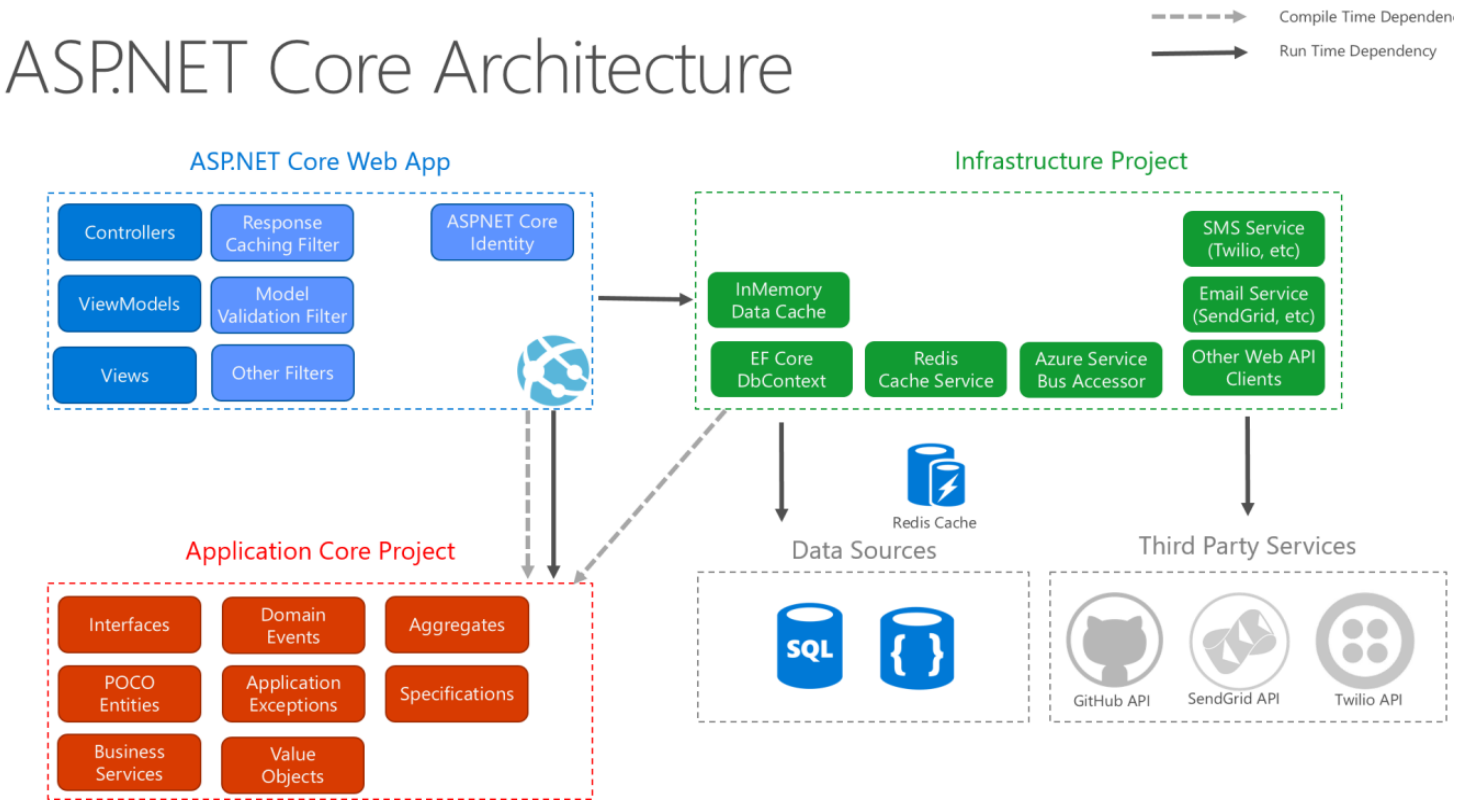


Figure 5: backend architecture diagram

Database (Azure SQL Database):

- Stores persistent data such as orders, inventory items, users, and suppliers.
- Ensures data integrity through foreign key relationships and constraints.

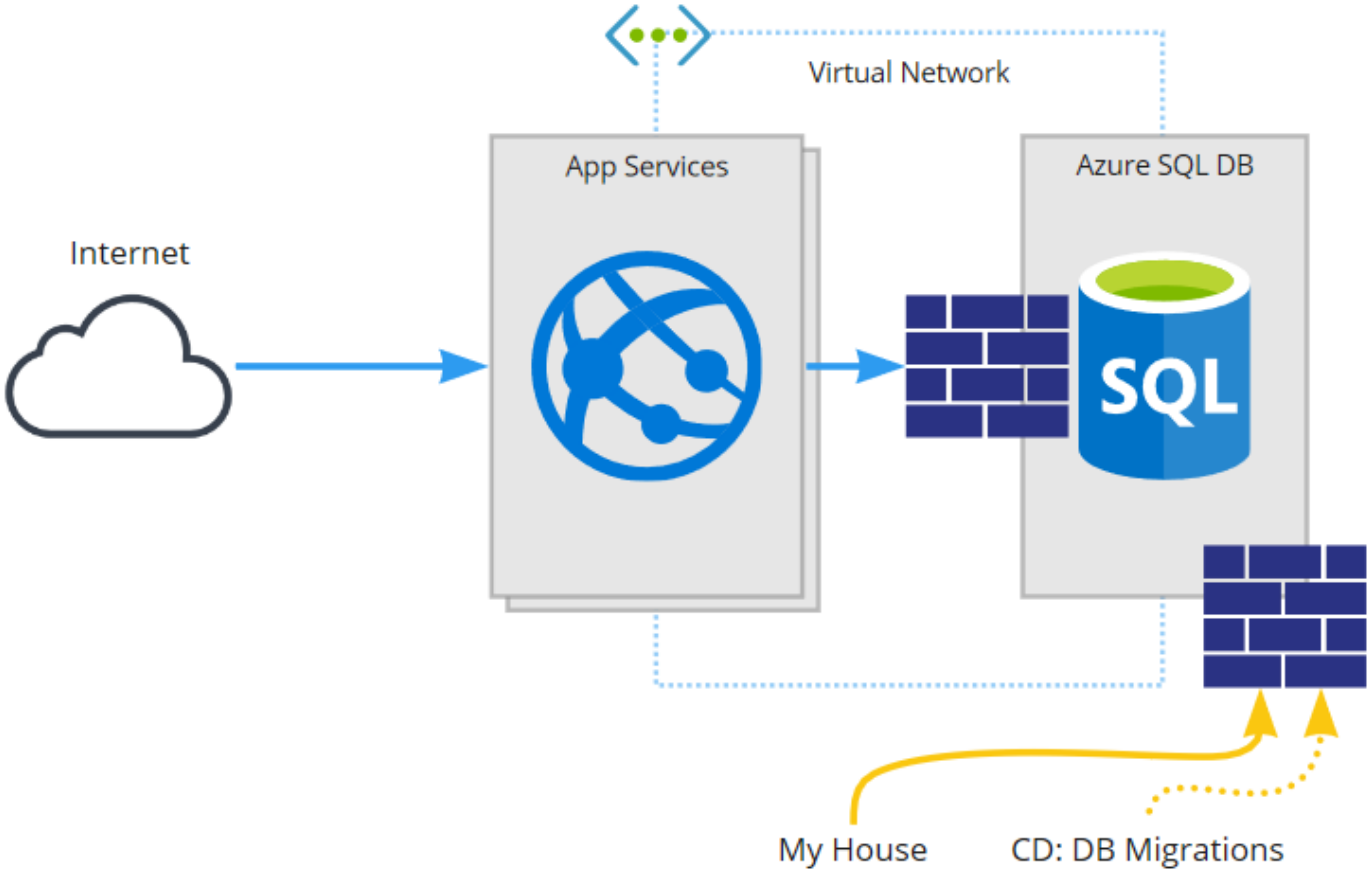


Figure 6: Azure SQL DB access level data fetching simple diagram

CI/CD Pipeline:

- Managed through **Azure DevOps**, the pipeline automates the building, testing, and deployment of both the frontend and backend.
- Staging environments are used to validate updates before they are deployed to production.

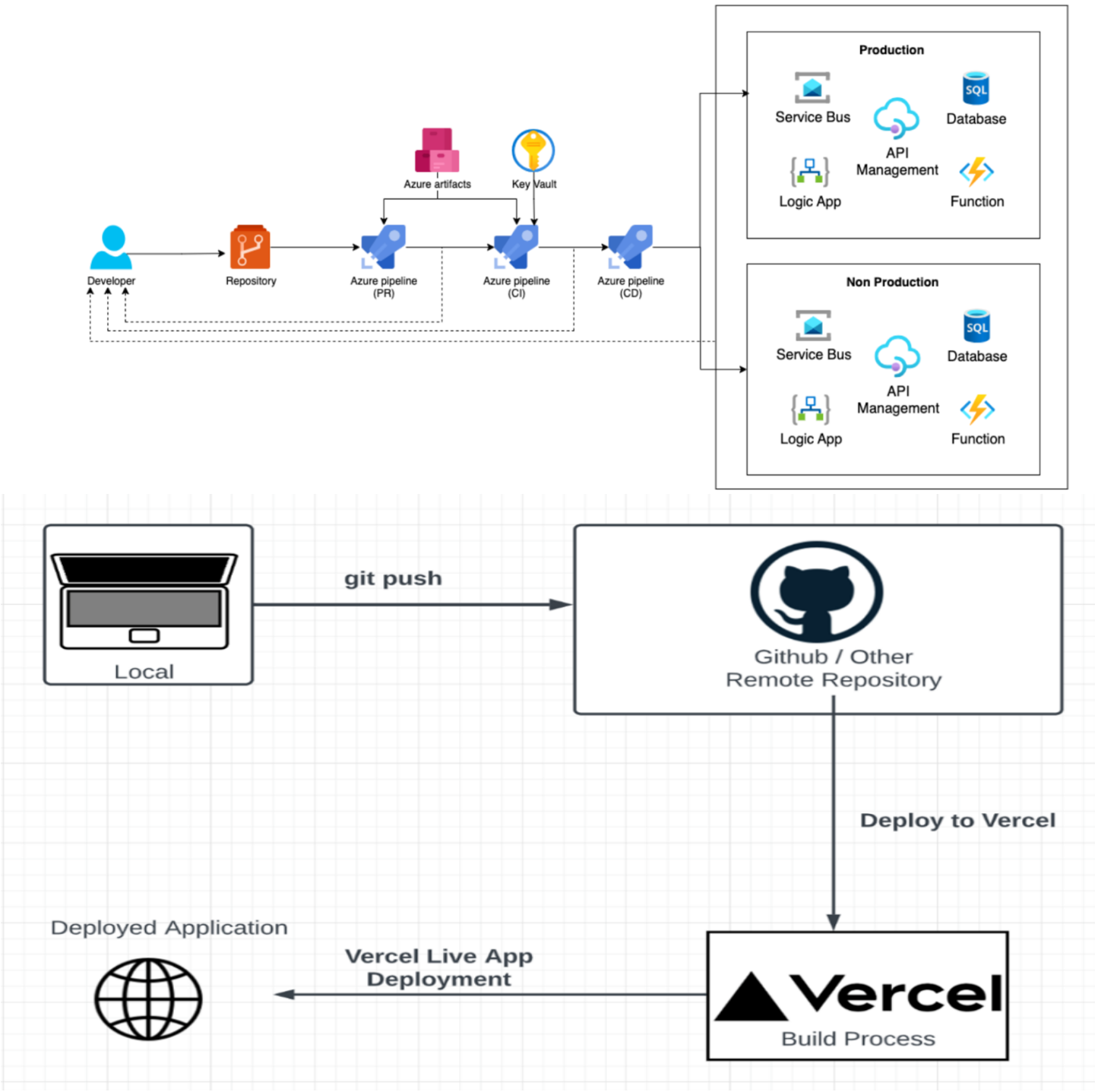


Figure 7: Deployment Process for both frontend and backend

Workflow Example:

a. A manager logs into the React app and creates a new order.
Axios sends a POST request with order details to the backend API.

b. The API Controller routes the request to
the OrderService, which validates the input and
calls the repository to save the order.

c. The repository uses Entity Framework to insert the order
into the database.

d. A success response is sent back to the frontend,
were the new order is displayed.

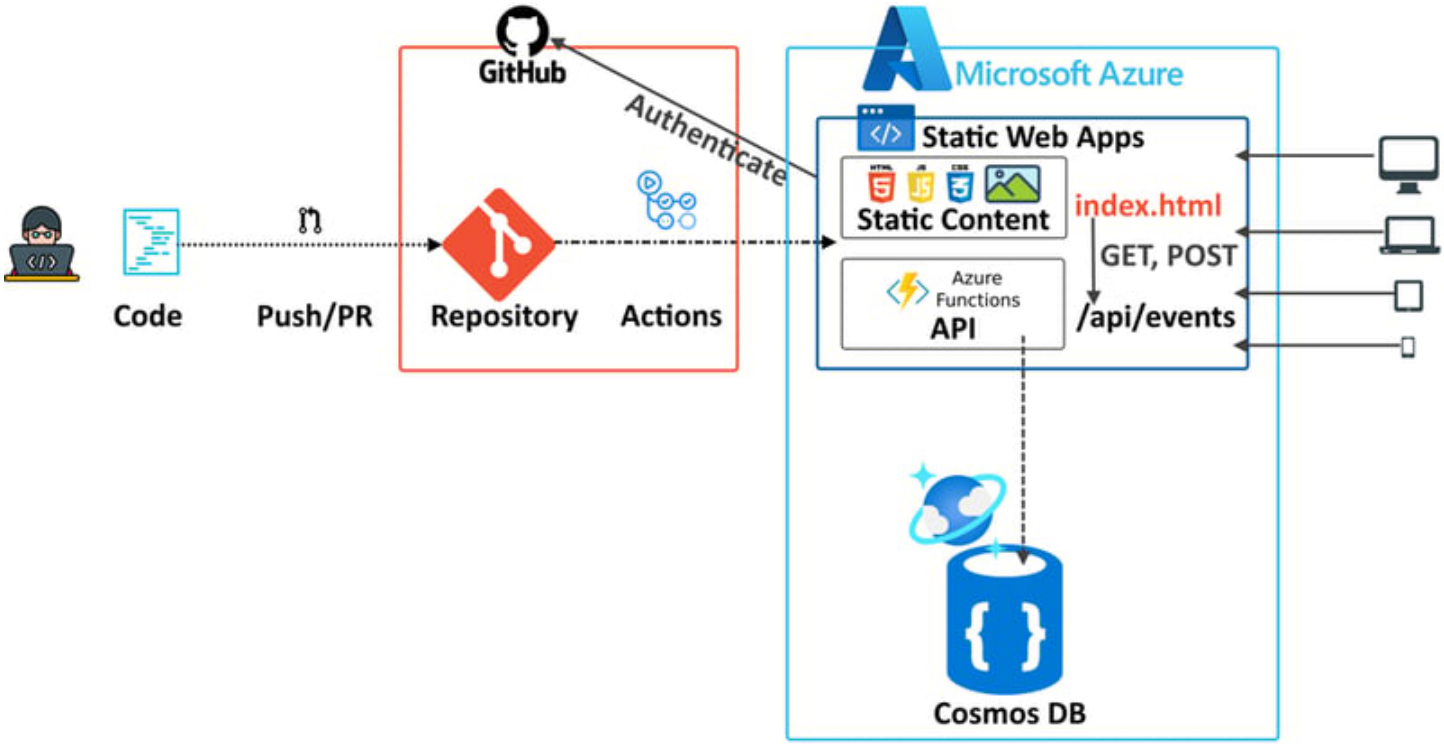


Figure 8 : Application Flow Diagram

4.3 Technology, Software, and Hardware Used

- **Technology and Software**:
  - **Backend**: ASP.NET Core 8.0, Entity Framework Core, Azure SQL Database.
  - **Frontend**: React, Next.js Framework, Axios for HTTP requests.
  - **CI/CD**: Azure DevOps for automated builds and deployments.
  - **Testing Tools**: Swagger for API testing, Postman for manual endpoint validation.
  - **Monitoring**: Azure Monitor and Application Insights for performance tracking and error detection.

- **Hardware**:
  - Development machines with minimum configurations: 8GB RAM, 256GB SSD.
  - Azure-hosted infrastructure: Vercel deployment for frontend and backend, scalable to handle traffic surges.

4.4 Rationale for Your Architectural Style and Model

The chosen architectural styles and model were selected to address the core requirements of modularity, scalability, and maintainability:

1. **MVC Architecture**:
   - Ensures that the backend logic is cleanly separated into layers, making it easier to debug and extend.
   - Supports scalability by allowing each layer to scale independently.

2. **Component-Based Frontend**:
   - Provides a modular approach to UI development, enabling faster development and consistent design patterns.
   - Ensures responsiveness and usability across devices.

3. **Integration with Azure Services**:
   - Leveraging Azure App Service and Azure SQL Database provides a reliable, scalable, and secure hosting environment.
   - Azure DevOps facilitates seamless CI/CD, reducing manual effort and deployment risks.

4. **Alignment with Modern Development Practices**:
    - By adopting industry-standard tools and methodologies, the system is future-proofed for integration, scaling, and enhancements.
    - Testability is improved at every layer, from unit testing backend services to integration testing APIs.

This architectural design ensures the system meets both current operational needs and future scalability requirements while maintaining reliability and performance.

5.Design

5.1 User Interface Design

The user interface (UI) of the Inventory Management System is designed to be user-friendly, responsive, and accessible across various devices. The design prioritizes clarity, efficiency, and role-specific functionality.

Key Features:

- **Dashboard**: Displays order summaries, low-stock alerts, and sales trends using cards and charts.

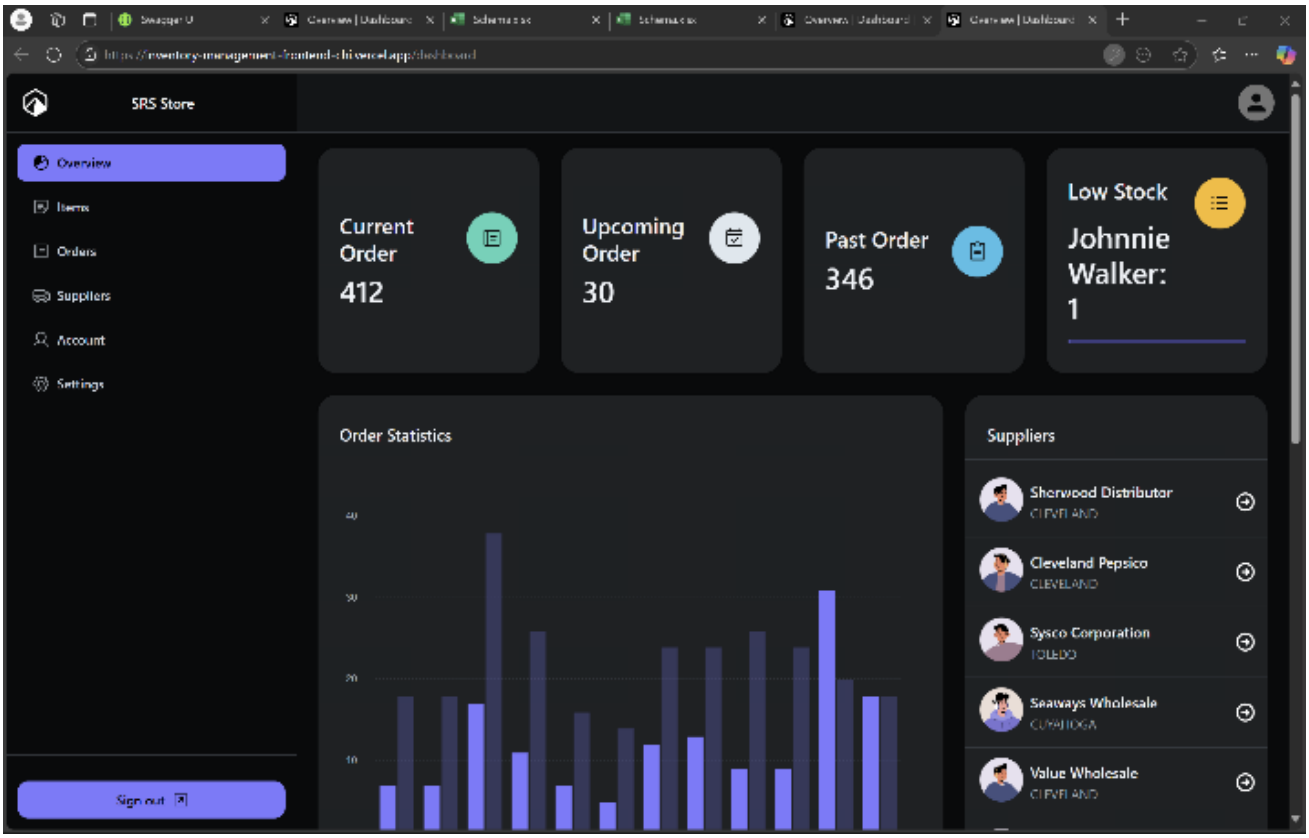


Figure 9 : Application Dashboard

- **Forms and Tables**: Interactive forms for data entry (e.g., creating orders) and dynamic tables for data visualization (e.g., inventory and order lists).

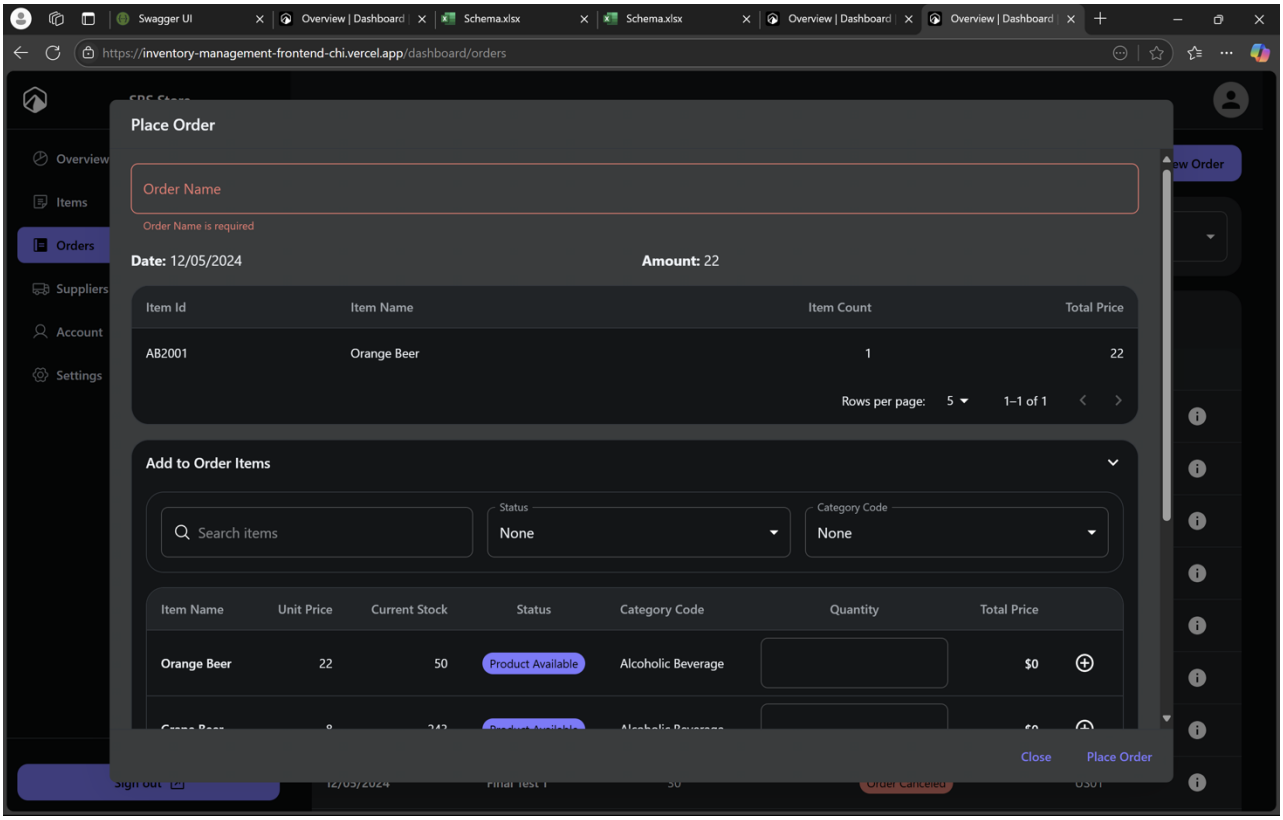


Figure 9 : Order Screen

- **Responsiveness**: The UI dynamically adjusts to different screen sizes, ensuring usability on desktops, tablets, and mobile devices.
- **Accessibility**: Adheres to WCAG 2.1 standards, providing an inclusive experience for all users.

5.2 Components Design

Static Models:

- **Frontend Components**:
  - **Header Component**: Includes branding, user profile access, and logout functionality.
  - **Dashboard Cards**: Displays summarized metrics for orders, inventory, and sales trends.
  - **Inventory List Component**: A searchable and sortable table displaying current inventory levels.
  - **Forms Component**: Handles user input for tasks like creating or modifying orders.

- **Backend Components**:
  - **Controllers**: Handle API requests and responses for different modules (e.g., OrdersController, InventoryController).
  - **Services**: Implement business logic and validate input data.
  - **Repositories**: Abstract database interactions using Entity Framework Core.

Dynamic Models:

- **User Interaction Workflow**:
  - A user logs into the system → Navigates to a specific module → Performs an action (e.g., creates an order) → Receives feedback (e.g., success message or error notification).

- **Component Interaction Flow**:
  - **Frontend (Next.js)**: A dashboard card triggers a request to fetch low-stock items.
  - **Backend (API)**: The InventoryController receives the request, invokes InventoryService, and retrieves data from the database via the repository.
  - **Database**: Returns the requested data, which is processed and sent back to the frontend for display.

5.3 Database Design

Database Schema:

The system uses an Azure SQL Database, with tables designed to ensure data integrity and optimized performance.

- **Key Tables and Relationships**:
  - **Category Table**: Stores product categories and relates to the Items table.
  - **Items Table**: Contains product details and relates to categories and orders via foreign keys.
  - **Orders Table**: Tracks order details, linked to users and items through the OrderItems table.
  - **Users Table**: Maintains user data, with roles determining access levels.
  - **OrderItems Table**: A junction table linking orders and items, tracking quantities and order statuses.
  - **Supplier Table**: Maintains supplier information, linked to categories through the SupplierToCategory table.

Normalization:

The schema follows third normal form (3NF) to eliminate redundancy and maintain data integrity.

ER Diagram:

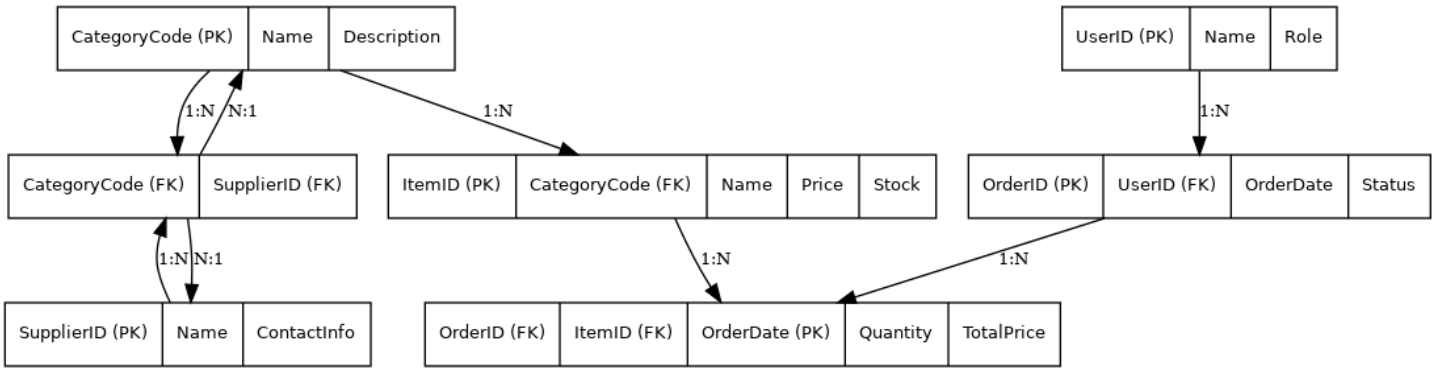


Figure 10 : Entity Relation Diagram

5.4 Rationale for Your Detailed Design Models

The detailed design ensures that the system adheres to industry best practices for modularity, scalability, and maintainability:

1. **Separation of Concerns**:
    - The frontend design isolates UI components for ease of modification and testing.
    - The backend structure (controllers, services, repositories) ensures clean separation between logic and data access.

2. **Efficiency**:
    - Database normalization minimizes redundancy, ensuring efficient data storage and retrieval.
    - Use of indexes and optimized queries improves application performance under high load.

3. **Scalability**:
    - Component-based frontend architecture and modular backend logic allow for easy integration of new features.
    - The database schema supports expansion, such as adding new user roles or product attributes.

4. **Usability and Accessibility**:
    - The UI design accommodates diverse user needs, ensuring ease of use for non-technical stakeholders.

5.5 Traceability from Requirements to Detailed Design

Models

Each design decision directly maps to the functional and non-functional requirements defined earlier in the project:

1. **Place an Order**
    - **UI Design**: Order creation form.
    - **Backend Design**: OrderController, OrderService, and OrderRepository handle order creation and validation.
    - **Database Design**: Orders table and OrderItems table store order data.

2. **Track Inventory**
    - **UI Design**: Inventory tracking table with low-stock alerts.
    - **Backend Design**: InventoryController and InventoryService manage stock-level queries.
    - **Database Design**: Items table stores stock levels, indexed for fast retrieval.

3. **Modify Product Details**
    - **UI Design**: Editable fields for product attributes.
    - **Backend Design**: ItemController and ItemService validate and save updates.
    - **Database Design**: Items table stores updated product details.

4. **Generate Order Report**
    - **UI Design**: Report generation module with filters.
    - **Backend Design**: ReportController and services generate aggregated data.
    - **Database Design**: SQL queries fetch data from Orders and OrderItems tables.

5. **System Maintenance**
    - **Backend Design**: Admin-level access to database management tools.
    - **Database Design**: Scheduled backups and audit logs support maintenance tasks.

By aligning requirements with detailed design models, the system delivers functionality and performance tailored to user needs.

6.Test Management

6.1 List of Test Cases

Test Case 1

ID	TC01
Test Input	Attempt to log in with valid credentials.
Steps	1) Go to login page.
	2) Enter valid credentials.
	3) Click "Login."
Expected Output	User logs in successfully and is redirected to the home page. Login time is recorded.
Description	Confirm that a user can log in with valid credentials and access the system.
Pass/Fail	Pass
Test Images	Test Image 1

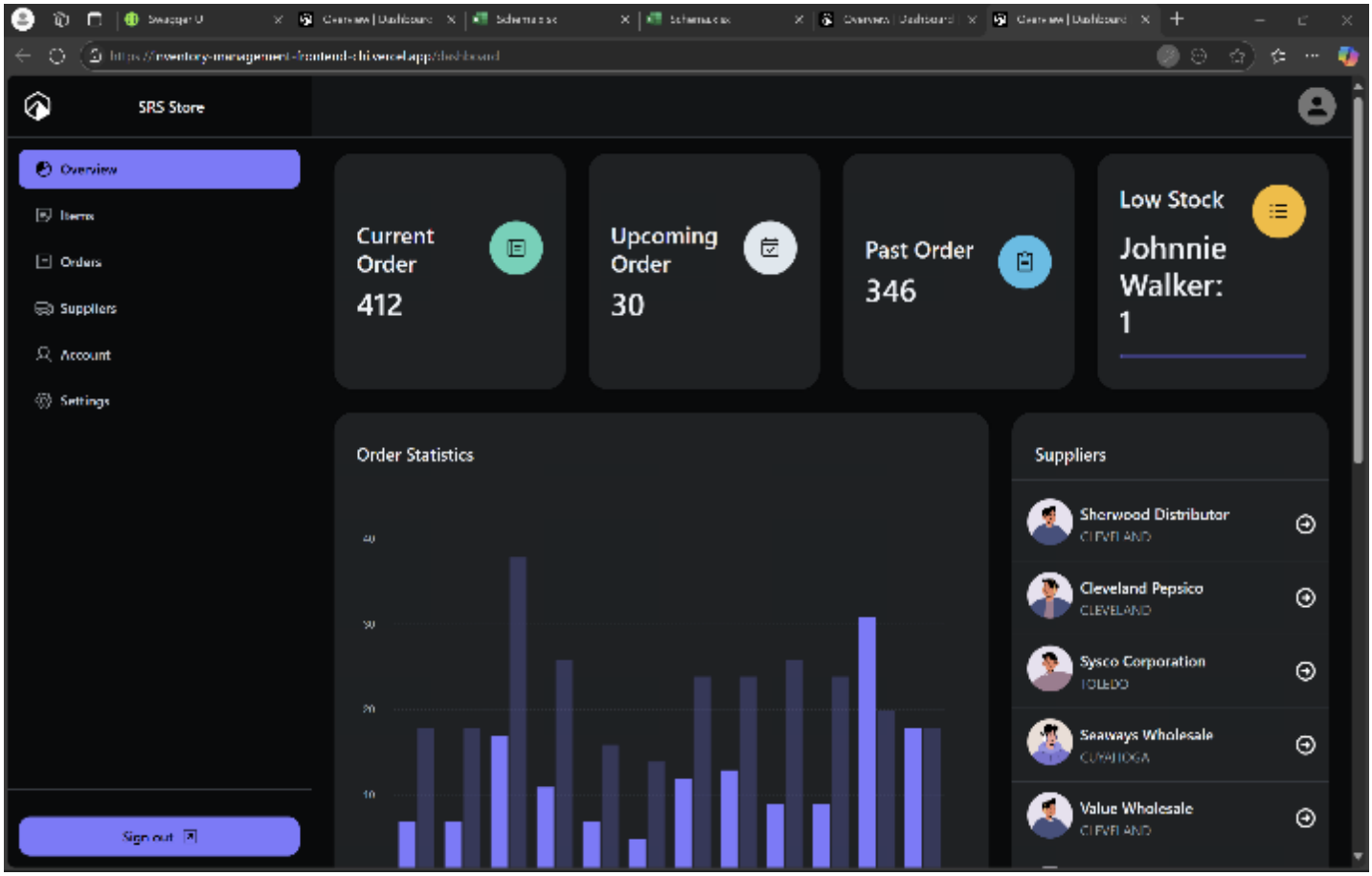


Figure 11 : Test Image 1

Test Case 2

ID	TC02
Test Input	Attempt to log in with an incorrect password.
Steps	1) Enter valid email.
	2) Enter incorrect password.
	3) Click "Login."
Expected Output	System displays an error message for incorrect credentials, and login fails.
Description	Confirm that the system prevents login with incorrect credentials.
Pass/Fail	*Pass*
Test Images	Test Image 2

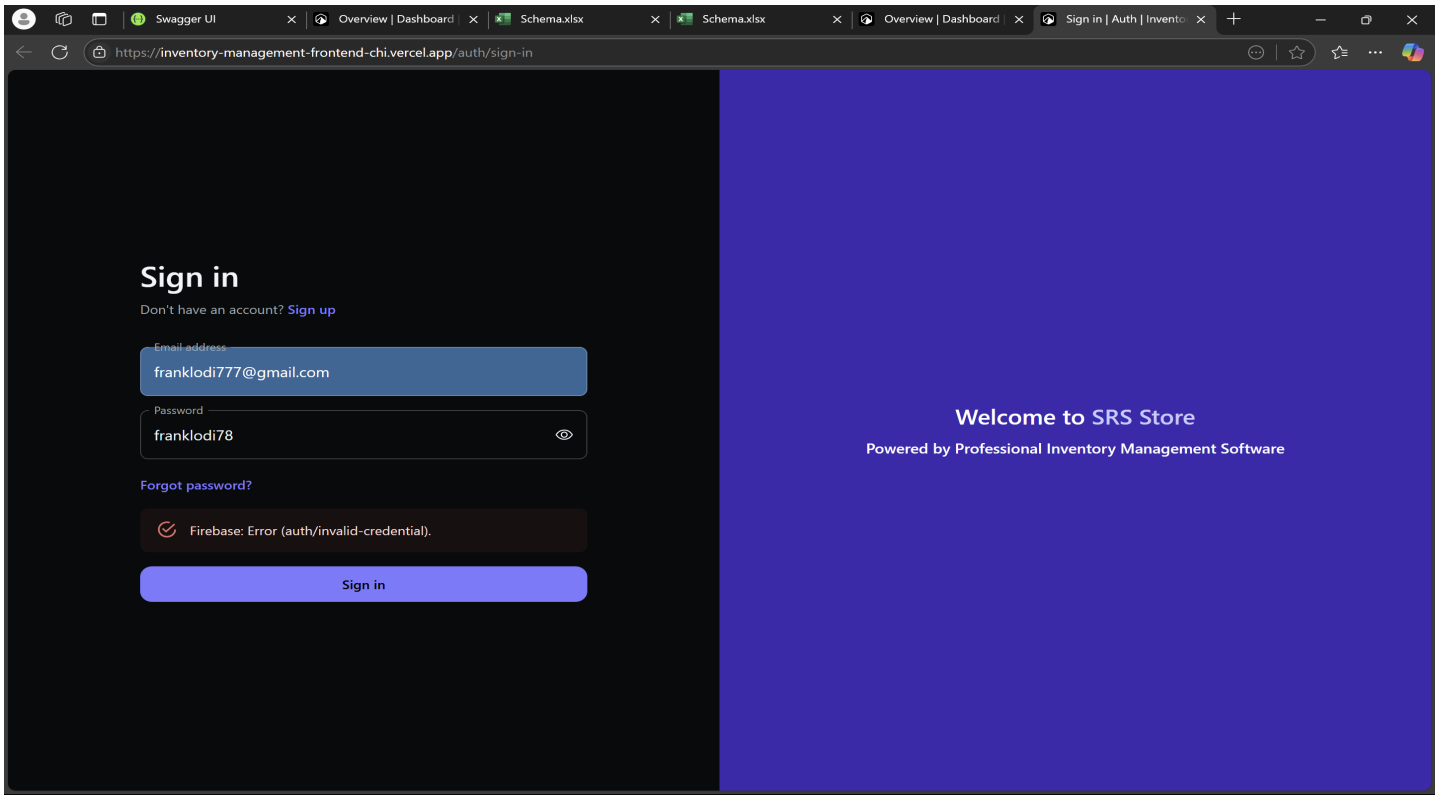


Figure 12 : Test Image 2

Test Case 3

ID	TC03
Test Input	Attempt to add a new product to the inventory.
Steps	1) Go to the "Product List" page.
	2) Click "Add Product."
	3) Fill in product details.
	4) Click "Save."
Expected Output	Product is successfully added to the inventory, and the product list is updated.
Description	Verify that a user can add a new product to the inventory successfully.
Pass/Fail	*Pass*
Test Images	Test Image 3, Test Image 4

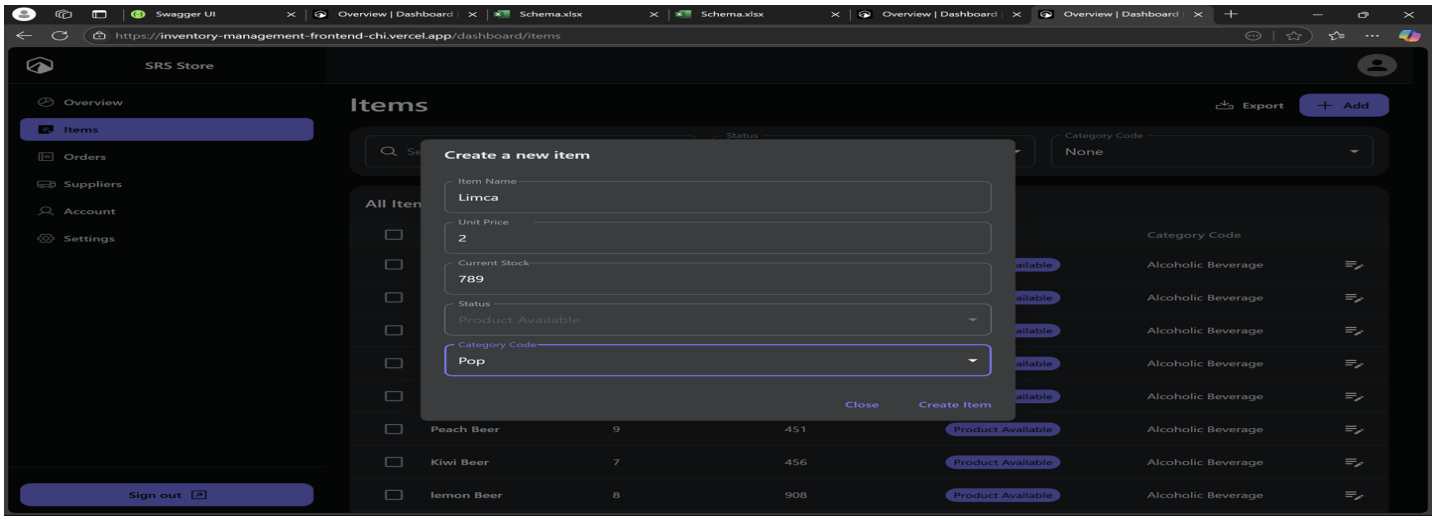
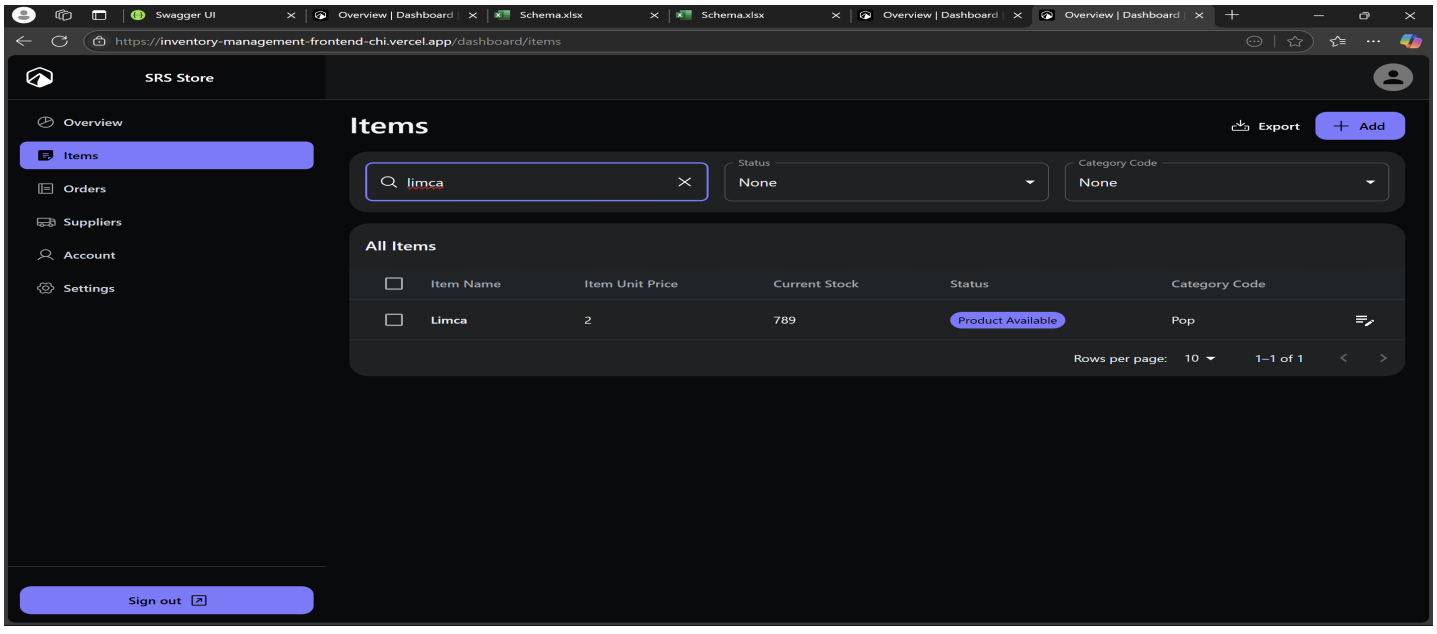


Figure 13 : Test Image 3 , Test Image 4



Test Case 4

ID	TC04
Test Input	Attempt to modify an existing order.
Steps	1) Go to the "Order Management" page.
	2) Select an existing order.
	3) Click "Edit."
	4) Update order details and click "Save."
Expected Output	Order details are successfully updated in the database.
Description	Our system lets you cancel an orderand re order again. As per Business requirements
Pass/Fail	Pass - Out of Scope
Test Images	No Image

Test Case 5

ID	TC05
Test Input	Attempt to delete an existing order.
Steps	1) Go to the "Order Management" page.
	2) Select an existing order.
	3) Click "Delete" and confirm the action.
Expected Output	Order is successfully deleted, and the order list is updated.
Description	Or system don’t allow you to delete an order. You can cancel the order
Pass/Fail	Pass - Out of Scope
Test Images	No image

Test Case 6

ID	TC06
Test Input	Attempt to view order details as a Worker.
Steps	1) Log in as a Worker.
	2) Go to the "Order Details" page.
	3) Select an order to view the details.
Expected Output	Worker can view the details of the selected order.
Description	Verify that a Worker can view the details of both previous and upcoming orders.
Pass/Fail	Pass. (We are still awaiting confirmation from Business users)
Test Images	Test Image 6

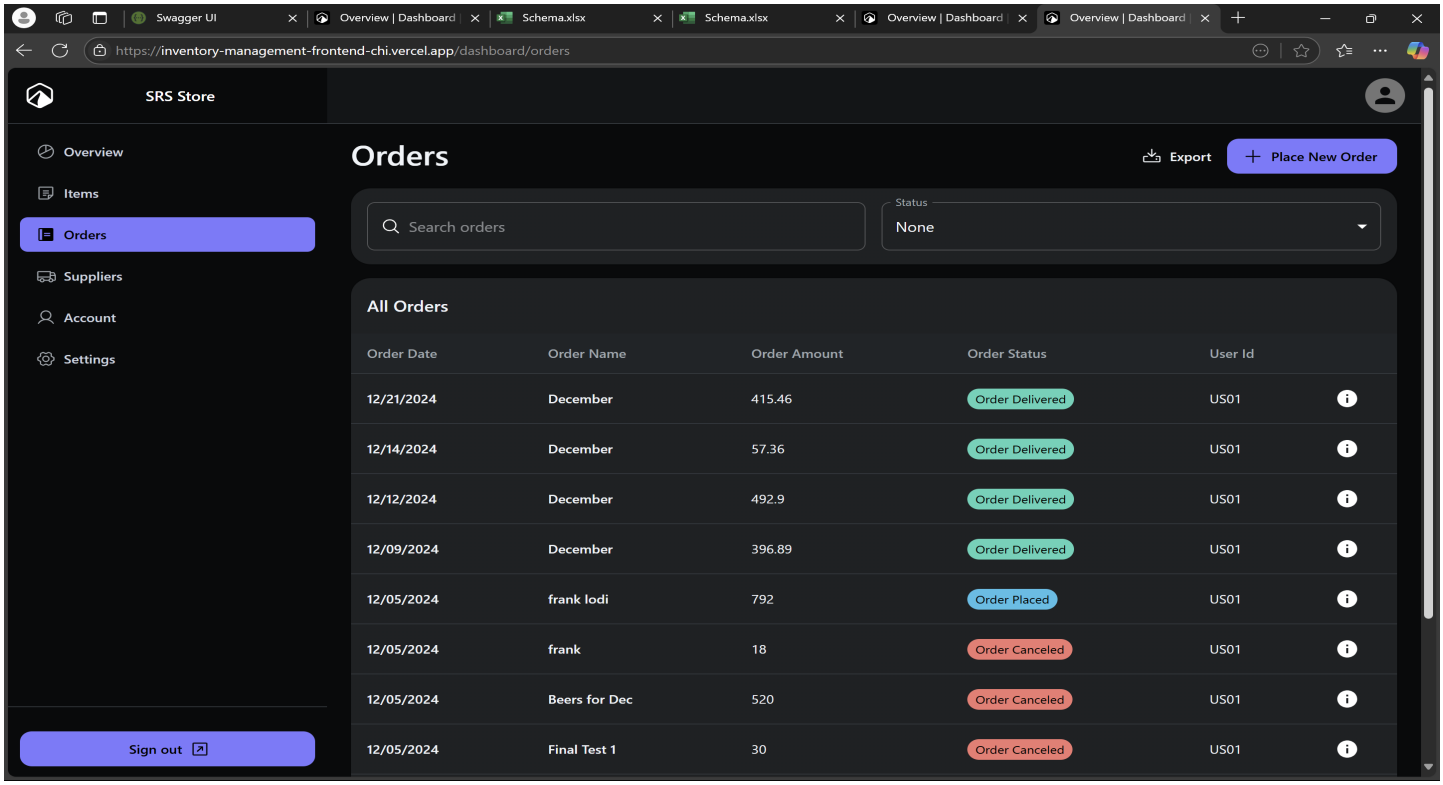


Figure 14 : Test Image 6

Test Case 7

ID	TC07
Test Input	Attempt to view low stock items on the dashboard.
Steps	1) Log in as a Manager.
	2) Go to the dashboard page.
	3) View the "Low Stock Items" card.
Expected Output	The dashboard displays products with low stock accurately.
Description	Verify that the low stock items card shows accurate information.
Pass/Fail	*Pass*
Test Images	*Test Imag7.i*

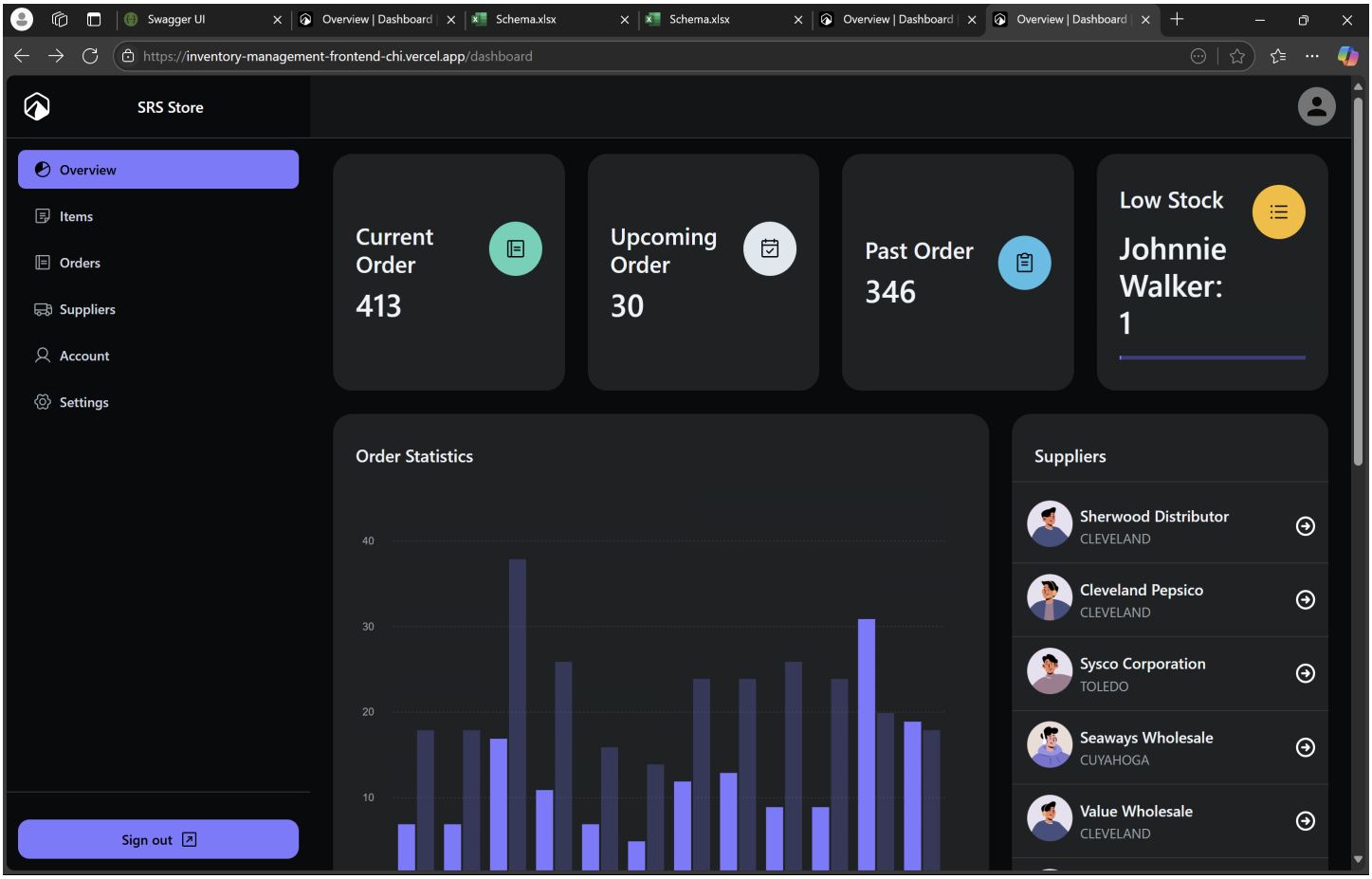


Fig 15: Test Image 7

Test Case 8

ID	TC08
Test Input	Attempt to create an order without providing mandatory fields.
Steps	1) Go to "Order Management".
	2) Click "Create Order".
	3) Leave mandatory fields empty.
	4) Click "Save".
Expected Output	System displays an error message indicating mandatory fields are required.
Description	Verify that mandatory fields are validated when creating an order.
Pass/Fail	*Pass*
Test Images	Test Image 8

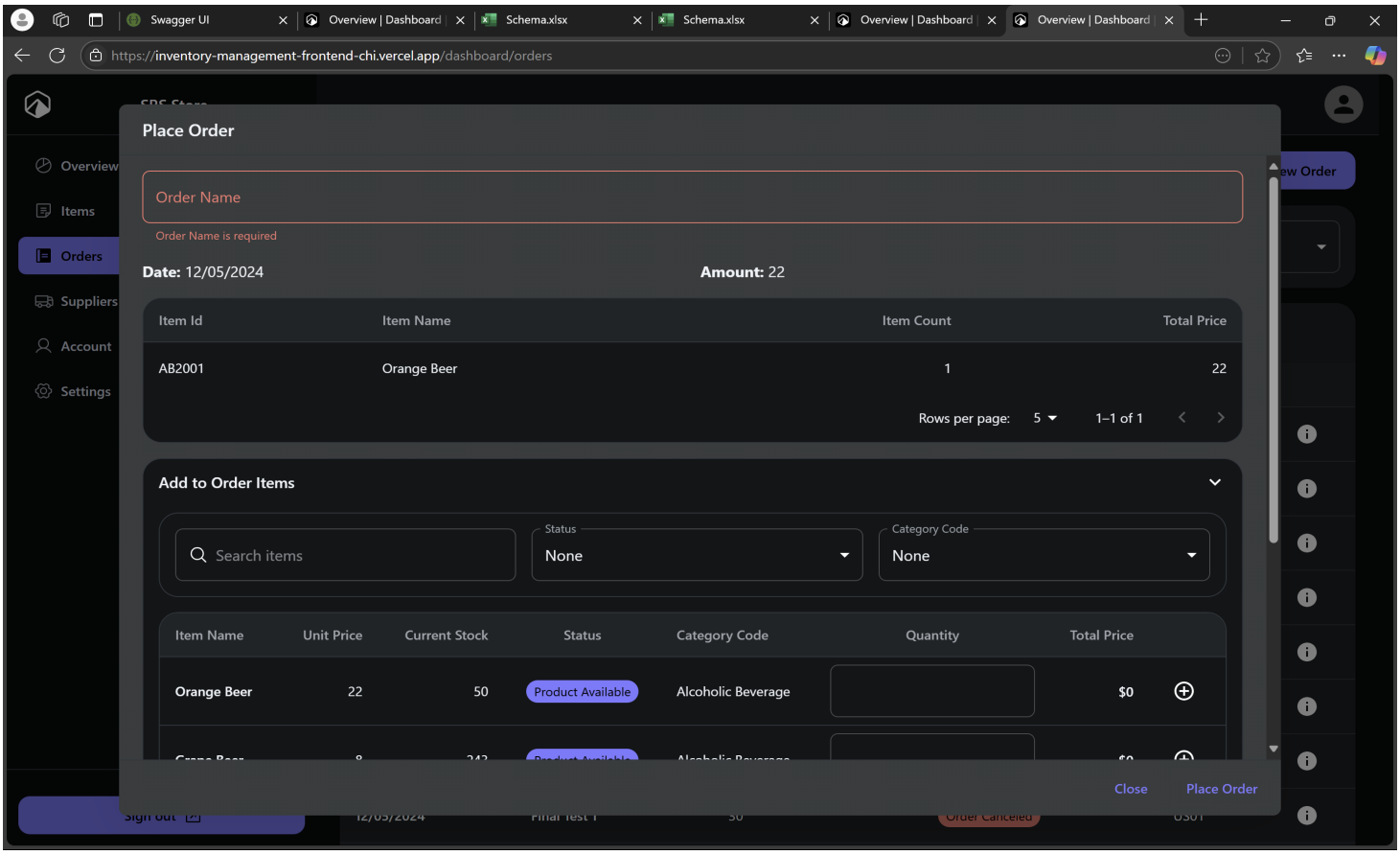


Fig 16: Test Image 8

Test Case 9

ID	TC09
Test Input	Attempt to create an order with invalid quantity (negative value).
Steps	1) Go to "Order Management".
	2) Click "Create Order".
	3) Enter negative value for quantity.
	4) Click "Save".
Expected Output	System displays an error message for invalid quantity input.
Description	Verify that the system does not allow negative quantities for orders.
Pass/Fail	*Pass*
Test Images	Test Image 9

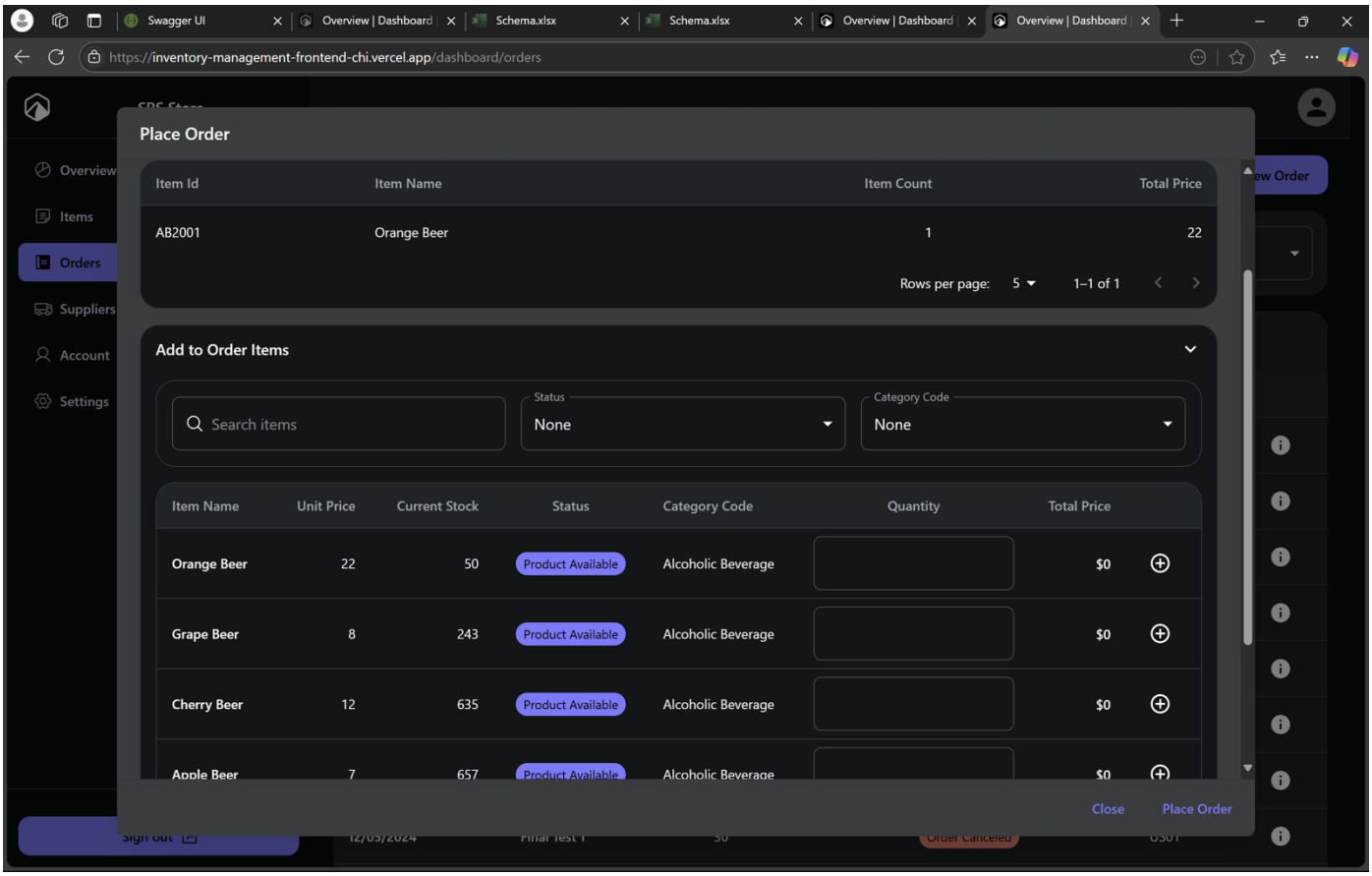


Fig 17: Test Image 9

Test Case 10

ID	TC10
Test Input	Attempt to edit an inventory item with valid details.
Steps	1) Go to "Inventory Management".
	2) Select an item.
	3) Click "Edit".
	4) Update the fields and click "Save".
Expected Output	Inventory item details are successfully updated.
Description	Verify that users can update inventory item details correctly.
Pass/Fail	*Pass*
Test Images	*Test Image 10.i, 10.ii, 10.iii*

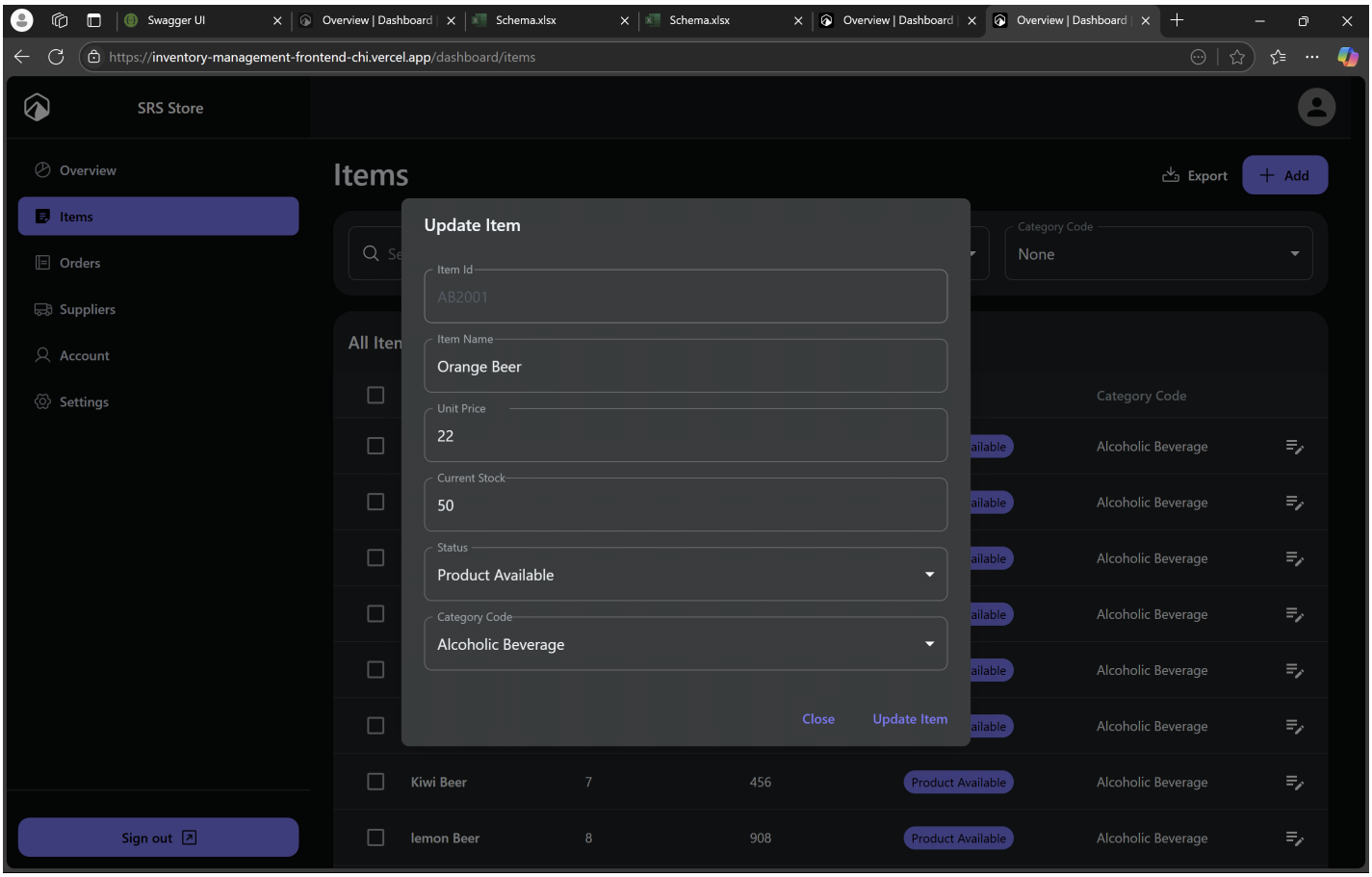


Fig 18: Test Image 10.i

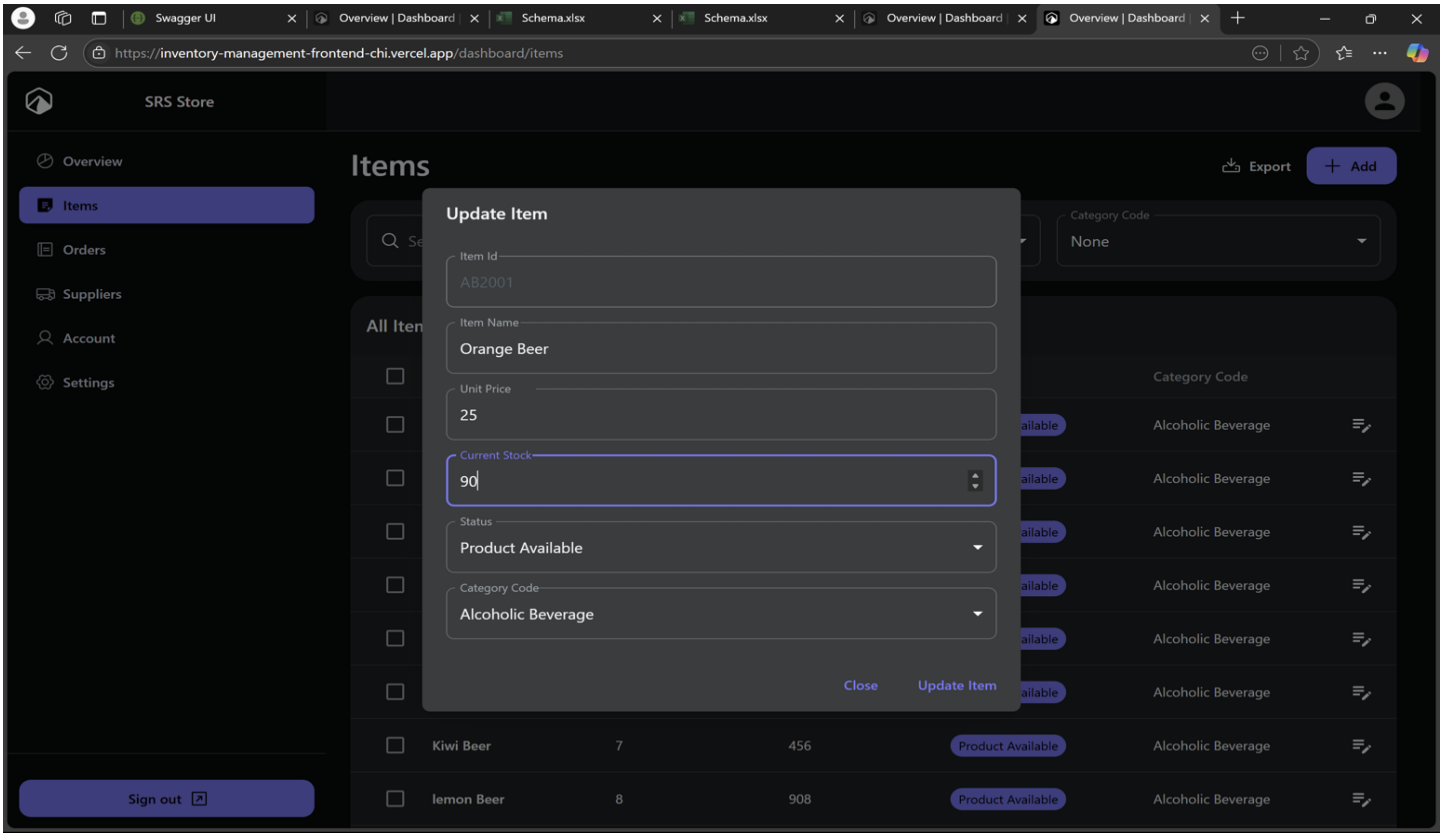


Fig 19: Test Image 10.ii

Test Case 11

ID	TC11
Test Input	Attempt to delete a product that is linked to an existing order.
Steps	1) Go to "Product List". 2) Select a product. 3) Click "Delete".
Expected Output	System displays an error message preventing deletion due to linked orders.
Description	Verify that the system prevents deletion of products linked to existing orders.
Pass/Fail	*Pass*
Test Images	*Test Image 11.i, 11.ii*

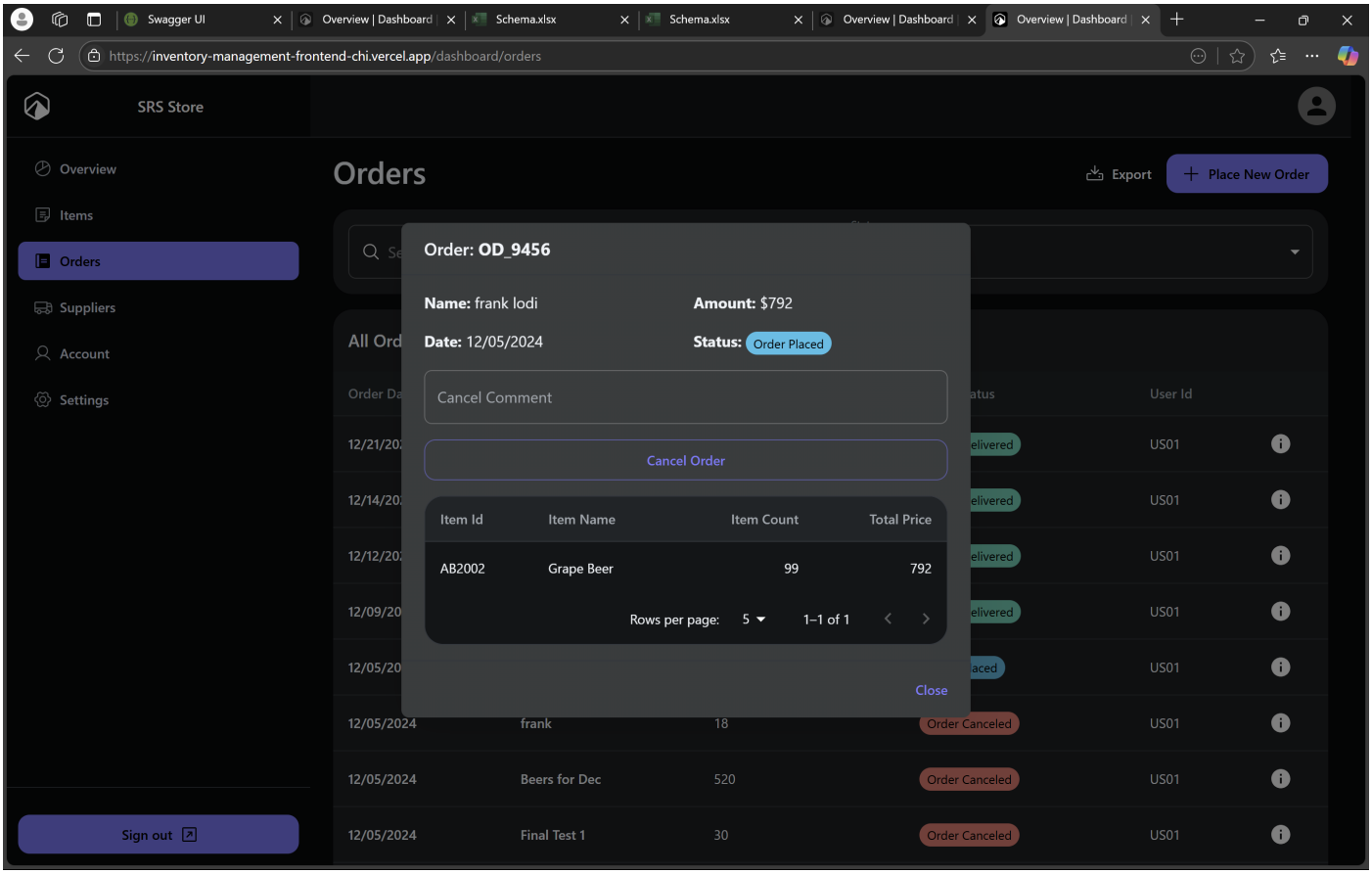


Fig 20: Test Image 11.i

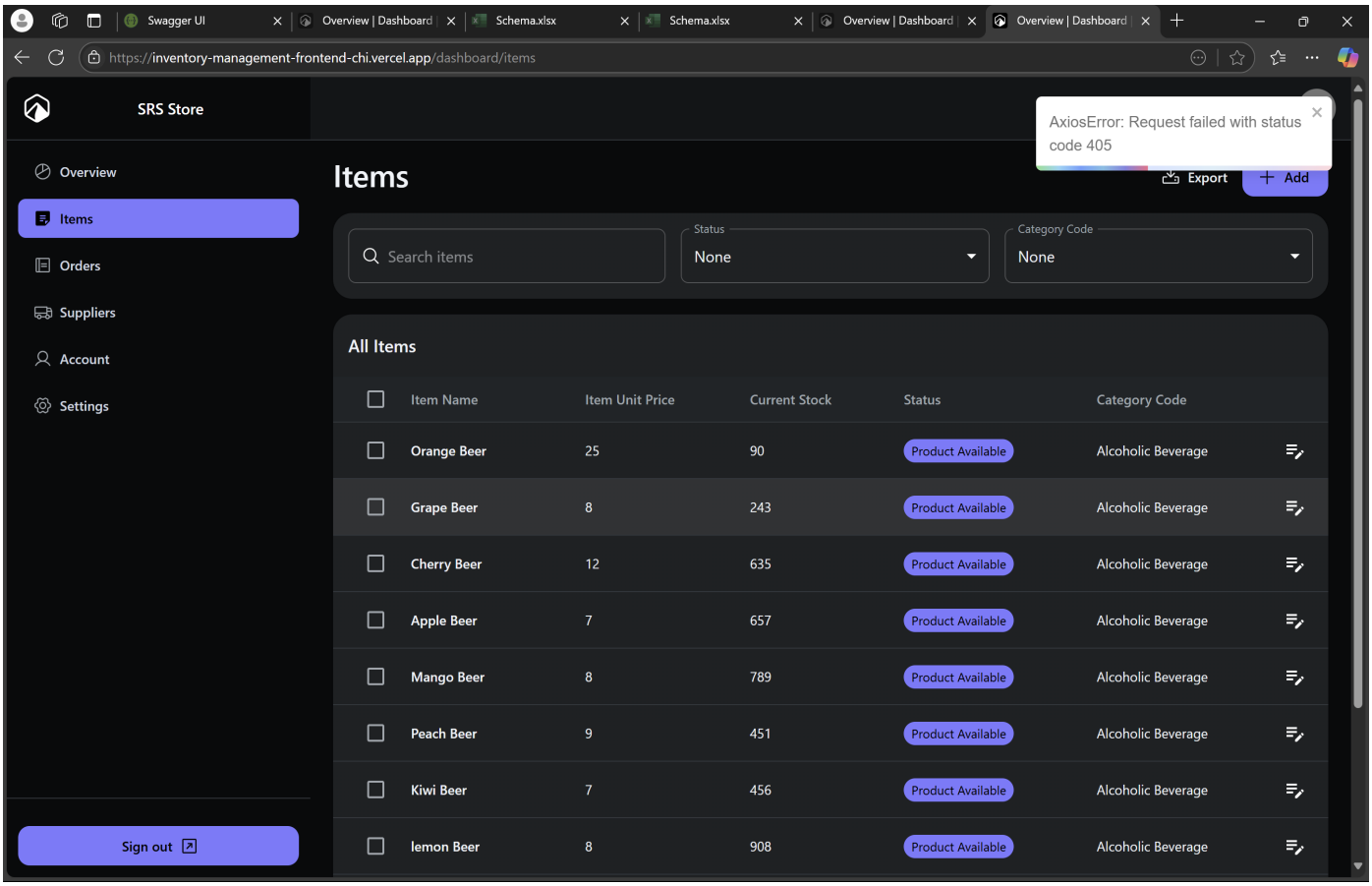


Fig 21: Test Image 11.ii

Test Case 12

ID	TC12
Test Input	Attempt to view the highest-selling products chart.
Steps	1) Go to the dashboard. 2) Locate the highest-selling products chart.
Expected Output	The highest-selling products chart displays accurate data.
Description	Verify that the chart shows correct information on highest-selling products.
Pass/Fail	*Fail (Out of Scope)*
Test Images	*Attach relevant screenshots*

Test Case 13

ID	TC13
Test Input	Attempt to create an order with special characters in the order name.
Steps	1) Go to "Order Management". 2) Click "Create Order". 3) Enter special characters (e.g., "@#$%") in the order name. 4) Click "Save".
Expected Output	System validates input and either accepts or displays a relevant error message.
Description	Verify how the system handles special characters in text fields during order creation.
Pass/Fail	*To be filled during testing*
Test Images	*Test Image 13*

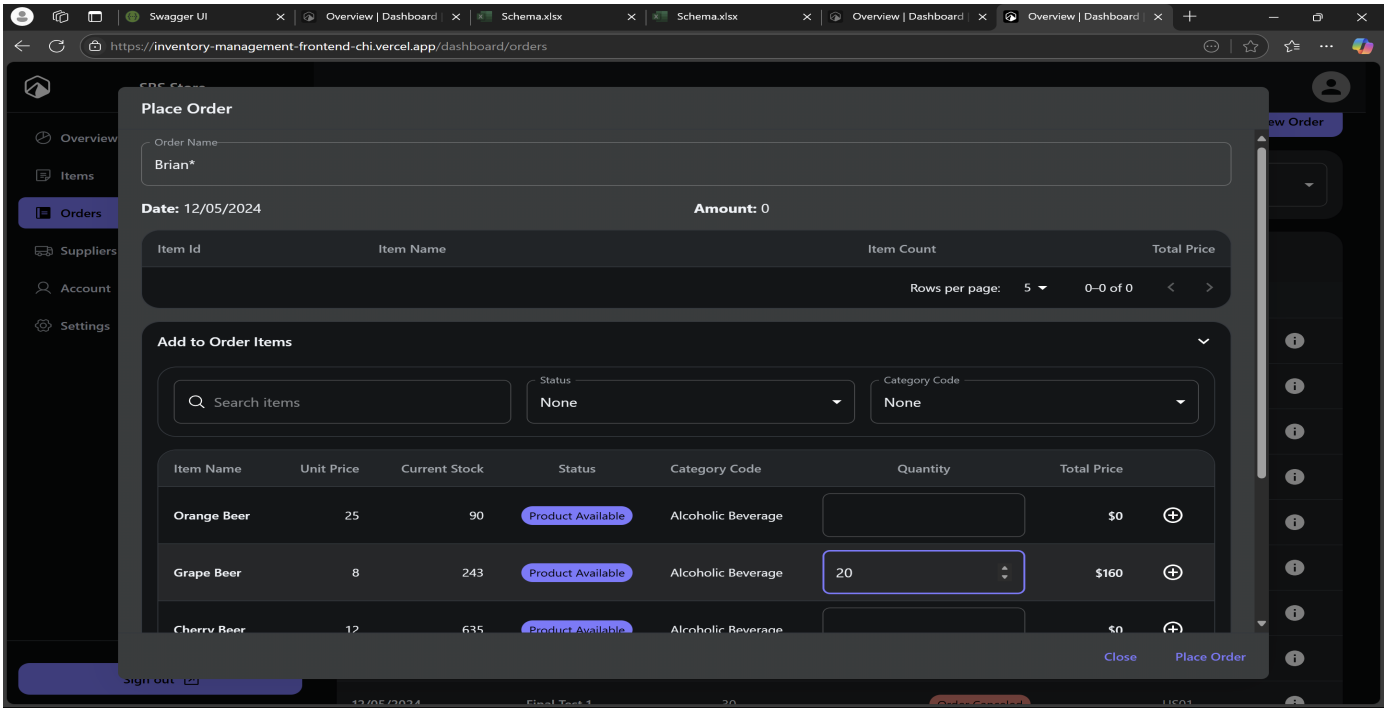


Fig 22: Test Image 13.i

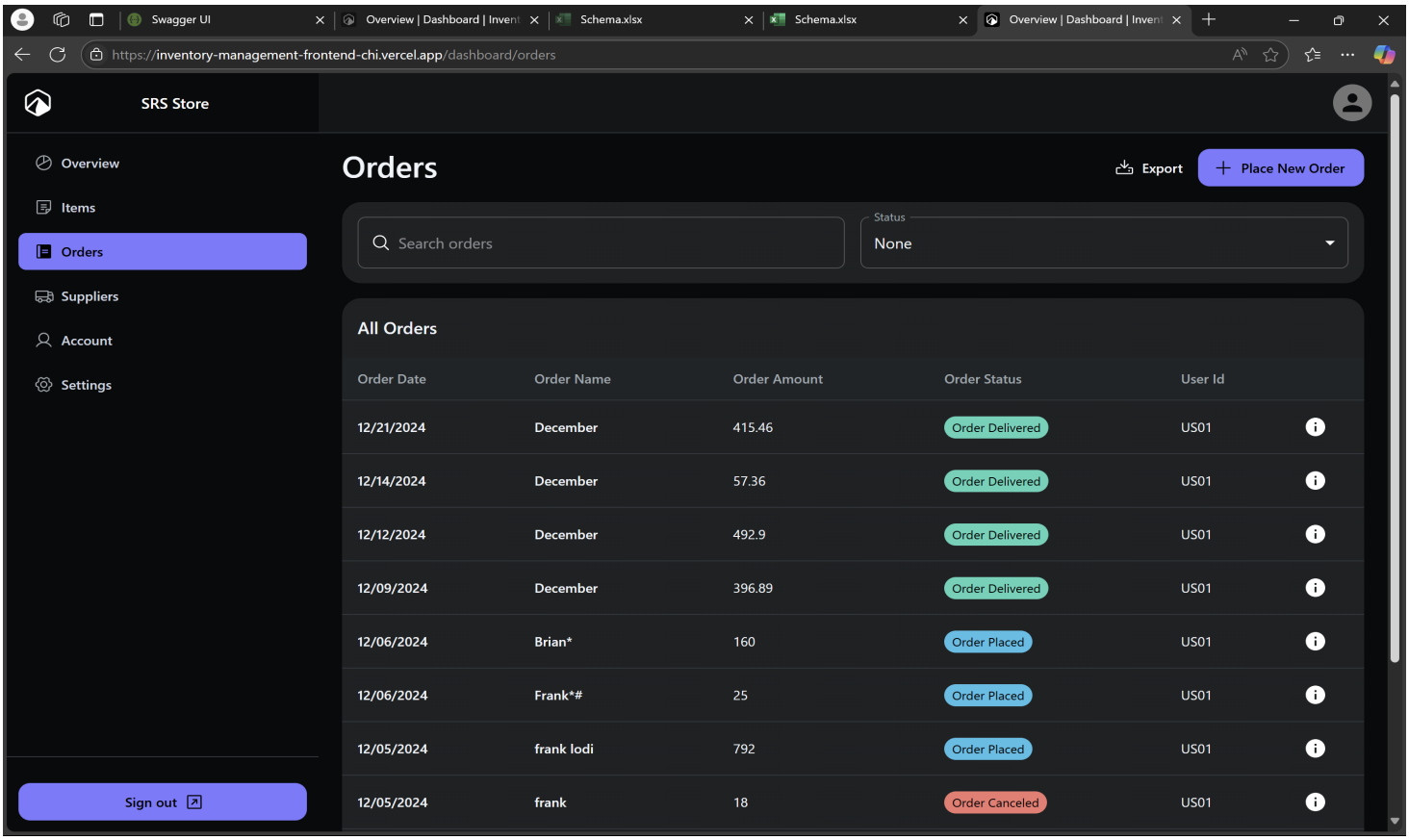


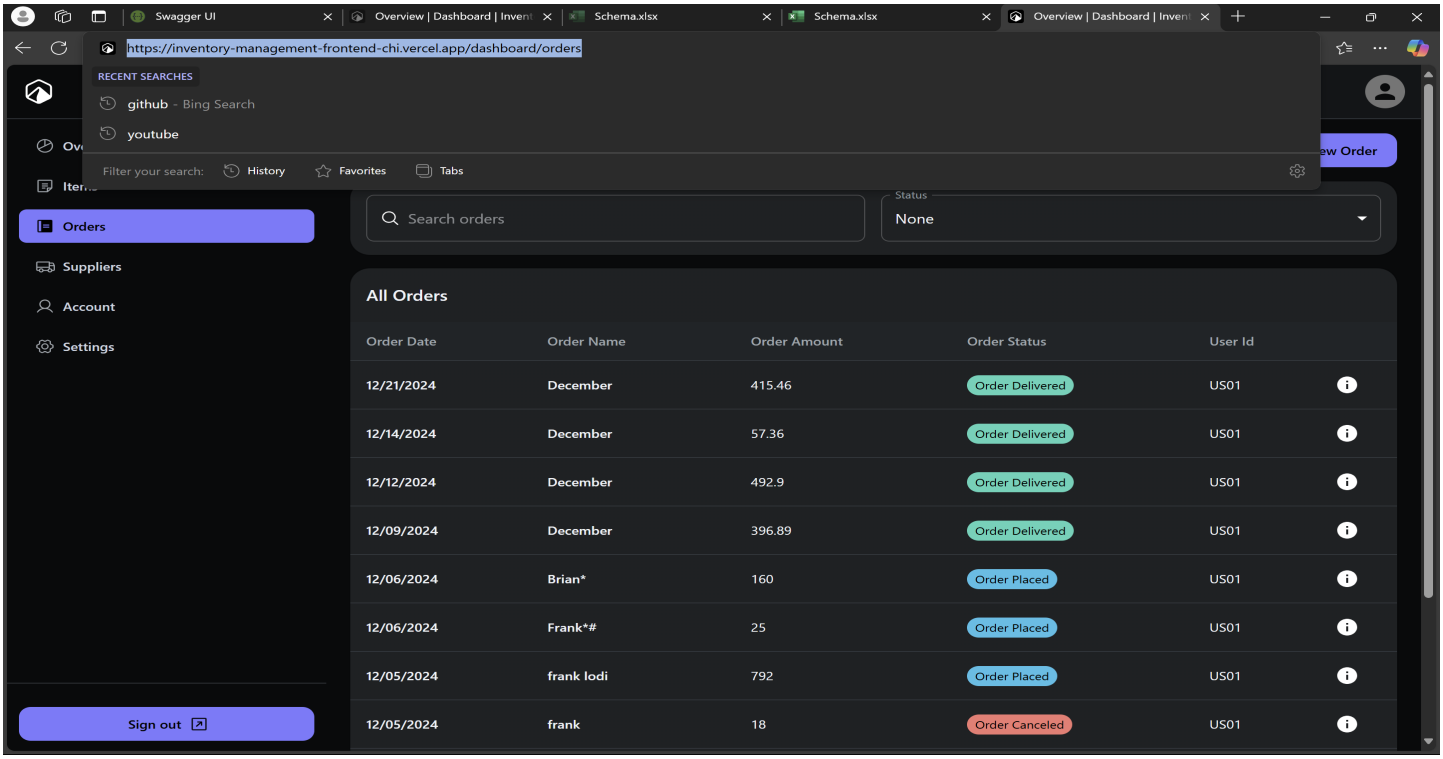
Fig 23: Test Image 13.ii

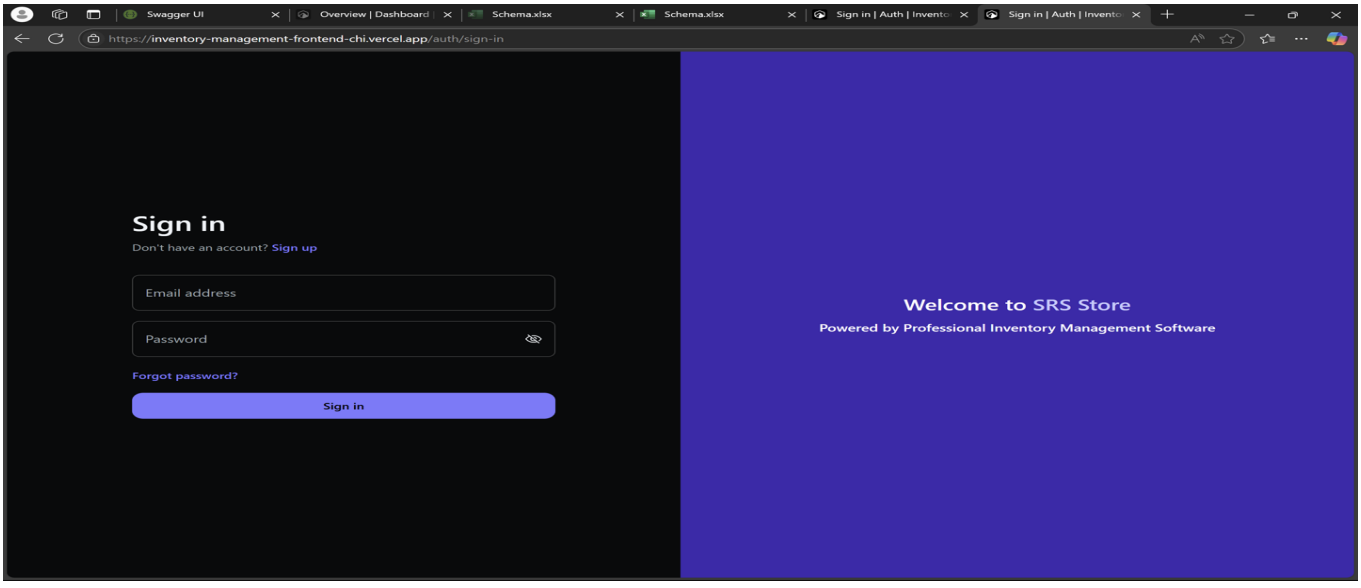
Test Case 14

ID	TC14
Test Input	Attempt to log in multiple times concurrently from different devices with the same credentials.
Steps	1) Log in with valid credentials on Device A. 2) Attempt to log in again with the same credentials on Device B.
Expected Output	System either allows concurrent login or automatically logs out the first session, depending on security settings.
Description	Verify that the system handles multiple concurrent logins as per design.
Pass/Fail	*To be filled during testing*
Test Images	*Attach relevant screenshots*

Test Case 15

ID	TC15
Test Input	Attempt to access an order using a direct URL without proper authentication.
Steps	1) Copy the URL of an order details page. 2) Log out. 3) Paste the URL directly in the browser.
Expected Output	System should redirect to the login page if the user is not authenticated.
Description	Verify that unauthorized users cannot directly access restricted pages using URLs.
Pass/Fail	*Pass*
Test Images	*Attach relevant screenshots*





6.2 Traceability of Test Cases to Use Cases

Use Cases	Relevant Test Cases and their features
Place an Order	TC01 (Login), TC03 (Add product), TC08 (Mandatory fields validation), TC09 (Negative quantity), TC16 (Stock validation), TC22 (Authorization).
Track Inventory	TC07 (Low stock items on dashboard), TC10 (Edit inventory item), TC21 (No data for charts).
Modify Product Details	TC03 (Add product), TC10 (Edit inventory item), TC20 (Negative stock value).
Generate Order Report	TC12 (Highest-selling products chart), TC21 (Empty data for charts).
Cancel an Order	TC04 (Modify order), TC05 (Delete order – validation).
System Maintenance	TC14 (Concurrent login), TC15 (Unauthorized URL access), TC17 (Delete inventory), TC18 (Concurrent order creation).

6.3 Techniques Used for Test Case Generation

Technique	Description	Examples
Boundary Value Analysis	Tested numeric inputs like quantities and stock levels at boundary values.	TC09, TC20
Equivalence Partitioning	Categorized valid and invalid inputs for fields like order name and login credentials.	TC02, TC13
Error Guessing	Identified edge cases based on experience, such as special character handling.	TC13, TC14
Role-Based Testing	Tested permissions and restrictions based on user roles.	TC06, TC22
Scenario-Based Testing	Simulated real-world usage scenarios.	TC18, TC23
Negative Testing	Ensured proper handling of invalid inputs and actions.	TC08, TC09, TC20

6.4 Test Results and Assessments

The test results demonstrate that the system meets the functional requirements with a few exceptions:

Overall Assessment:

- **Pass Rate:** Most test cases passed, indicating that the core functionalities, such as order management, inventory tracking, and role-based access, are robust and reliable.

- **Fail Rate:** Failures occurred in edge cases, such as handling special characters in text fields (TC13) and implementing user role restrictions (TC22).

Strengths:

- The system successfully validates inputs and prevents incorrect operations, such as creating orders with invalid quantities or editing past orders.
- Data consistency is maintained during concurrent operations, ensuring no data corruption or loss.

Improvements Needed:

- **Role-Based Access Control:** Currently, all users can create orders, irrespective of their roles (TC22).
- **Special Character Validation:** The system needs to restrict the use of special characters in text fields (TC13).

Software Quality:

- **Functionality**: The software meets most of its functional requirements, with minor improvements required for edge cases.
- **Reliability:** The system is reliable under normal and concurrent use, with no significant performance issues reported.
- **Usability:** The UI is intuitive and ensures that users are guided with error messages for invalid actions.

6.5 Defects Reports

Defect ID	Defect Description	Priority	Resolution Plan	Status
DEF01	Special characters allowed in order name (TC13).	Medium	Implement input validation to restrict special characters.	Work in Progress
DEF02	Role-based access control not implemented (TC22).	High	Update the role-based access control mechanism to enforce proper permissions.	Work in Progress
DEF03	No data handling for charts (TC21).	Low	Add appropriate messages or placeholders for charts with no data.	Work in Progress
DEF04	Concurrent logins not restricted or managed (TC14).	Low	Add session management to handle multiple logins based on business requirements.	Work in Progress

6.6 General Steps for Improving Failed Test Cases

1. **Root Cause Analysis**: Identify the underlying issues contributing to test case failures through debugging and error logging.
2. **Validation Enhancements:** Strengthen input validation mechanisms and role checks at both frontend and backend levels.
3. **UI/UX Improvements:** Ensure that user interfaces provide clear feedback or alternatives for scenarios like empty data or restricted actions.
4. **Testing Framework Updates:** Expand test cases to cover edge cases and scenarios that may have been previously overlooked.
5. **Retesting**: After implementing fixes, rerun the failed test cases to confirm that the issues have been resolved.
6. **Documentation Updates:** Update the system's documentation to reflect the fixes and their impact on functionality.

7.Conclusions

7.1 Outcomes of the Project

The Inventory Management System successfully delivered its core objectives, meeting the business requirements outlined at the project's inception. The following outcomes were achieved:

1. **Core Functionalities Delivered:**
    - **Order Management:** Users can create, modify, and cancel orders, with seamless updates to the inventory.
    - **Inventory Tracking:** Real-time updates and alerts for low-stock items ensure efficient stock management.
    - **Role-Based Features:** Managers and workers have differentiated access, enabling tailored interactions with the system.
    - **Dashboard Analytics:** The dashboard provides actionable insights with visual representations of sales trends and inventory status.

2. **System Reliability and Performance:**
    - The system demonstrated high reliability during testing, with a majority of test cases passing successfully.
    - The system handles concurrent operations and maintains data integrity, ensuring smooth multi-user workflows.

3. **Scalability and Maintainability:**
    - The modular architecture (**MVC for backend, component-based frontend**) supports future enhancements with minimal disruption to existing functionality.
    - **CI/CD** integration ensures continuous deployment and testing, allowing seamless updates.

4. **Partial Goals:**
    - **Role-Based Access Control:** While the feature is implemented, further refinements are needed to restrict certain actions effectively.
    - **Special Character Validation:** This area requires immediate attention to prevent data inconsistencies.

7.2 Lessons Learned

The project provided valuable insights and learning experiences across technical, process, and team collaboration aspects:

1. **Technical Insights**:
    - **Strengths**:
        - The combination of **Next.js** and **.NET Core** proved highly effective in delivering a scalable and responsive system.
        - **Azure DevOps** streamlined the **CI/CD** process, ensuring smooth deployments and consistent testing.

    - **Challenges**:
        - Managing role-based access and edge cases like special character handling revealed gaps in initial planning.
        - Concurrent operations required additional focus to ensure data consistency.

2. **Process Improvements**:
    - **Agile** methodology, combined with **Trello and MS Teams** for task management, facilitated iterative progress and regular stakeholder feedback.
    - Weekly meetings with the professor helped identify and address potential issues early in the development cycle.

3. **Collaboration and Communication**:
    - Clear communication among team members and stakeholders was vital for aligning goals and resolving bottlenecks.
    - Real-time updates in Trello ensured transparency and accountability for task completion.

4. **Testing and Validation**:
    - Early investment in comprehensive test case generation paid off by identifying key issues before deployment.
    - A greater emphasis on negative testing helped uncover edge cases that could impact user experience.

7.3 Future Development

The Inventory Management System serves as a foundation for future enhancements to meet evolving business needs. Key areas for future development include:

1. **Sales System Integration**:
    - Integrate the system with sales platforms to provide a unified solution for managing both inventory and sales.
    - Automate financial tracking by linking sales data to inventory usage, streamlining reconciliation processes.
2. **AI-Driven Features**:
    - Develop an AI-based recommendation system that analyzes past trends and automatically prepares order lists for restocking.
    - Use machine learning algorithms to predict sales patterns, ensuring optimal inventory levels and reducing stockouts or overstocking.
3. **Enhanced Reporting and Analytics**:
    - Expand the dashboard to include more detailed analytics, such as supplier performance and product profitability.
    - Allow customizable reporting options, enabling users to generate insights tailored to their needs.
4. **Mobile Application Development**:
    - Create a mobile-friendly version of the application or a dedicated mobile app for on-the-go inventory and order management.
5. **User Experience Improvements**:
    - Introduce role-specific dashboards to improve usability and efficiency.
    - Add multi-language support to cater to a diverse user base.

These enhancements will not only expand the system's functionality but also align it with emerging technologies and user expectations, making it a robust and forward-looking solution.

8. References

8.1 Project Repository

1. **GitHub Repository**:
    - Inventory Management System
    - **Backend Branch**: Contains the production-ready code for the backend, developed using .NET Core Web API.
    - **Frontend Branch**: Contains the production-ready code for the frontend, built with React (Next.js).

2. **Development Tools and Frameworks**
    - **.NET Core Web API**:
        1. Description: The backend framework used for developing RESTful APIs with robust, scalable architecture.
        2. Download Link: .NET SDK 8.0
    - **Next.js**:
        1. Description: The frontend framework used for creating reusable components and delivering server-rendered web applications.
        2. Website (Next.js Official Website)
    - **React**:
        1. Description: React is the library for web and native user interfaces. Build user interfaces out of individual pieces called components written in JavaScript.
        2. Website (React Official Website)
    - **Entity Framework Core**:
        1. Description: Used for database interactions, providing an ORM for Azure SQL Database.
        2. Documentation: EF Core Documentation
    - **Axios**:
        1. Description: A JavaScript library used in the frontend to handle HTTP requests to the backend API.
        2. Documentation: Axios GitHub

3. **Testing Tools**
    1. **Swagger**:
        - Description: Used for API validation and interactive documentation of endpoints.
        - Documentation: Swagger OpenAPI Documentation
    2. **Postman**:
        - Description: A platform for API development, testing, and debugging.
        - Download Link: Postman
    3. **Jest and React Testing Library**:
        - Description: Tools for unit and integration testing of React components.
        - Documentation: Jest | React Testing Library
    4. **NUnit**:
        - Description: A unit testing framework for .NET used to validate backend functionality.
        - Download Link: NUnit

4. **Project Management and Communication**
    1. **Trello**:
       Used for sprint planning, task management, and tracking project progress.
       Access Link: Trello
    2. **Microsoft Teams**:
       Collaboration platform used for team communication and file sharing throughout the project.
       Download Link: Microsoft Teams

5. **Cloud Hosting and CI/CD Pipeline**
    1. **Azure App Service**:
       Used for hosting the backend and frontend services in a scalable and secure environment.
       Documentation: Azure App Service
    2. **Azure SQL Database**:
       Cloud database for storing inventory data, orders, and user information.
       Documentation: Azure SQL Database
    3. **Azure DevOps**:
       Used for CI/CD pipeline creation, ensuring automated builds, testing, and deployments.
       Documentation: Azure DevOps

6. **Design and Accessibility Standards**
    1. **WCAG 2.1 (Web Content Accessibility Guidelines)**:
       Standards followed to ensure the application is accessible to all users, including those with disabilities.
       Access Link: WCAG 2.1 Guidelines
    2. **Clean Code Principles**:
       Ensured modular and maintainable code by adhering to industry-standard practices.
       Reference Book: Clean Code by Robert C. Martin

8.2 Additional Resources

1. **Visual Studio Code**:
    - Code editor used for development.
    - Download Link: Visual Studio Code
2. **WebStorm**:
    - Code editor used for development.
    - Download Link: WebStorm
3. **Azure Monitor**:
    - Description: Used for tracking application performance and error logging.
    - Documentation: Azure Monitor

This comprehensive set of references provides all the tools, frameworks, and platforms that contributed to the successful development and delivery of the project.