Deliverable 1

Medical Store Inventory System

Team: IntelliWare

Team Members:

- Amaar Khan 22i-0759– Scrum Master
- Saad Nadeem 22i-1030– Developer
- Mustafa Iqbal 22i1139– Team Lead

Submission Date: 28 February

Introduction

The Medical Store Inventory System is designed to manage the stock of medicines efficiently. It helps pharmacists, store managers, and staff track inventory levels, update stock, remove expired medicines, and generate reports. The system ensures the seamless management of medicine records, reducing errors and improving operational efficiency.

Project Vision

Our vision is to develop a robust, user-friendly, and scalable Medical Store Inventory System that enhances stock management, prevents medicine shortages, and ensures the availability of essential medicines. The system will streamline pharmacy operations and improve service quality.

Intended Use of the System

The system will be used by medical store personnel, including pharmacists, store managers, and cashiers, to efficiently manage inventory, update stock, and track expiry dates.

Stakeholders & Their Needs

Stakeholder	Needs
Pharmacist	Add, update, and delete medicines; track expiry dates.
Store Manager	Monitor inventory levels and ensure timely restocking.
Cashier	Verify stock availability before processing sales.

Features & Overall Functionality

- Add, update, and delete medicines in stock.

- Track medicine expiry dates and remove expired medicines.

- Generate inventory reports.

- Notify store managers of low stock levels.

- User authentication and role-based access control.

User Stories

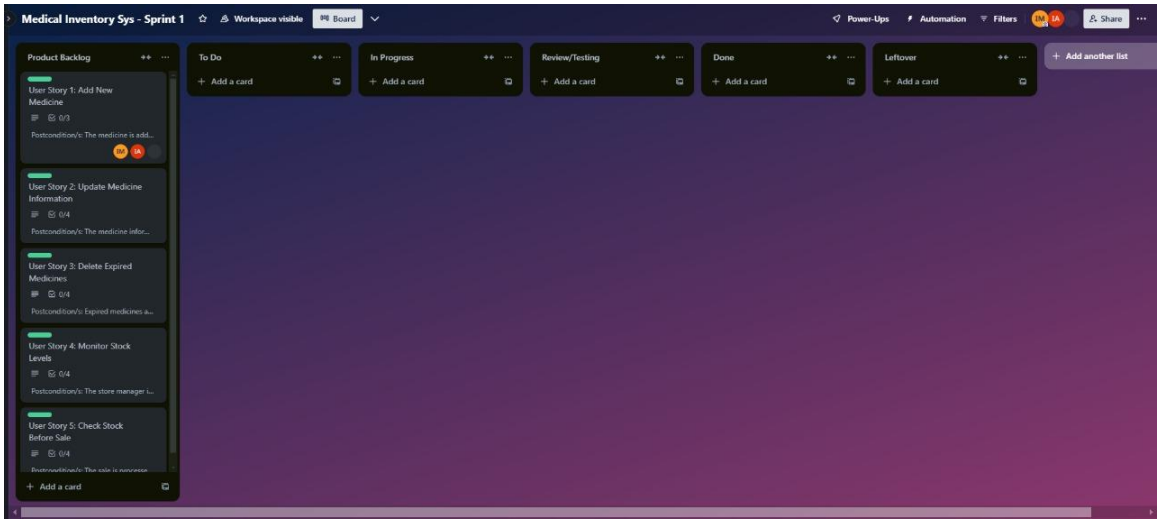
1. As a pharmacist, I want to add new medicines to the inventory, so that I can keep track of available stock.

2. As a store manager, I want to monitor stock levels, so that I can restock before medicines run out.

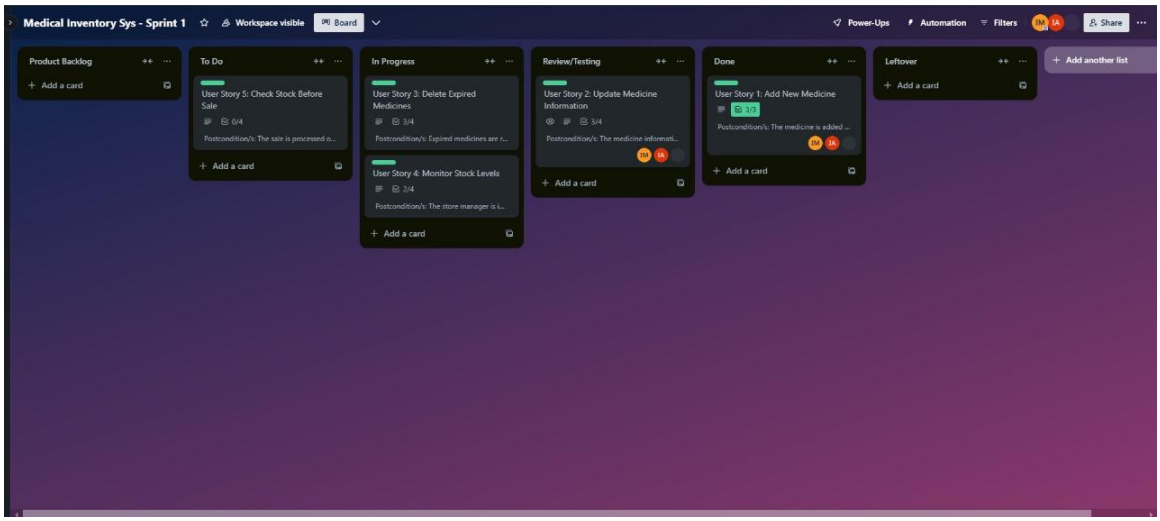
3. As a cashier, I want to **check stock availability before processing a sale, so that I can ensure the medicine is in stock.

Scrum Board Snapshots

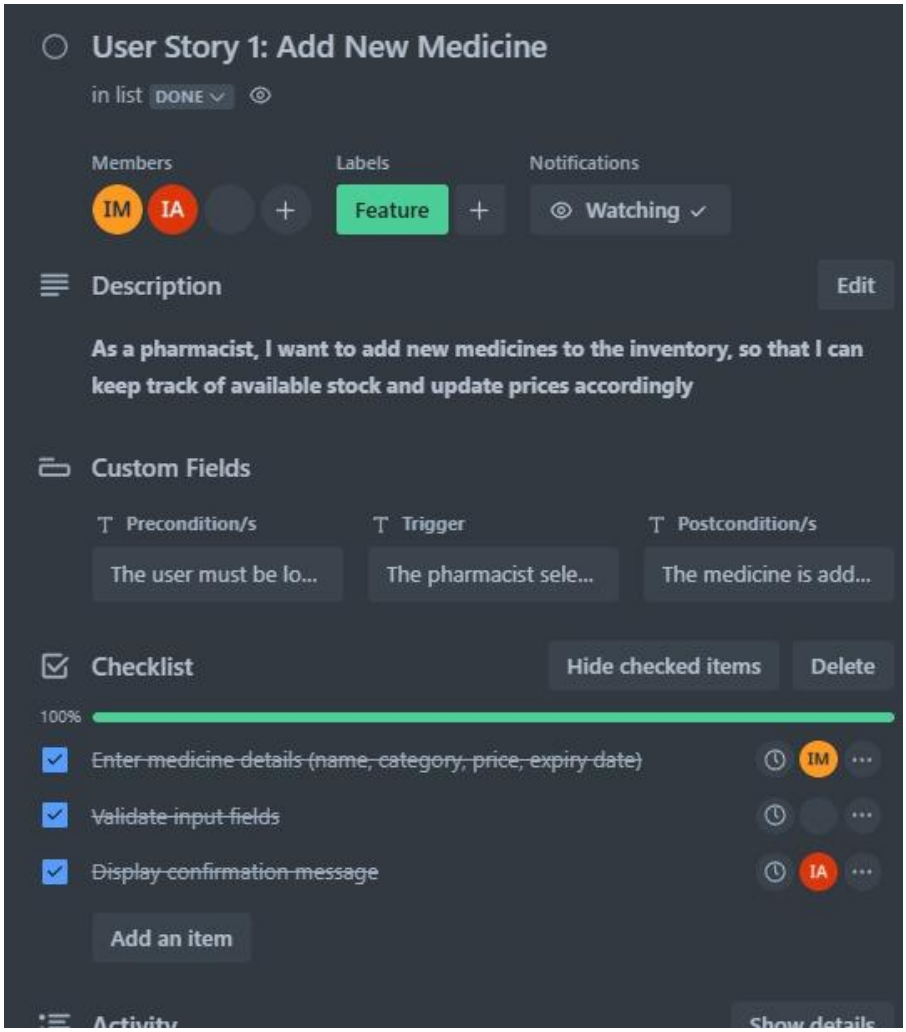
- Snapshot 1: Initial backlog uploaded.



Snapshot 2: Mid-sprint progress showing half-completed tasks.



- Snapshot 3: End of sprint with completed and leftover tasks.



Card example

Non-Functional Requirements (NFRs)

- Performance: The system should handle at least 1000 inventory records efficiently.

- Security: Only authorized users can add, update, or delete medicines.

- Usability: The UI should be user-friendly and accessible for non-technical users.

- Availability: The system should be available 24/7 without major downtimes.

- Scalability: The system should allow the addition of more users and medicines without performance issues.

Work Division

Team Member Task Assigned

Mustafa Iqbal (Team Lead) Project Management, System Design.

Amaar Khan (Scrum Master) Ensuring Agile Scrum Methodology, Sprint Planning.

Saad Nadeem (Developer) Development and implementation of core functionalities.

Structured Specifications for All User Stories

User Story 1: Add New Medicine

As a pharmacist, I want to add new medicines to the inventory, so that I can keep track of available stock and update prices accordingly.

Precondition: The relevant user must be logged into the system.

Trigger: The user accesses the respective feature from the inventory system menu.

Normal Flow:

1. The system displays the corresponding interface for the action.

2. The user performs the necessary actions (adding, updating, deleting, monitoring).

3. The system validates inputs and executes the task accordingly.

Postcondition: The action is successfully completed and updated in the inventory system.

User Story 2: Update Medicine Information

As a pharmacist, I want to update existing medicine details, so that I can keep the stock information accurate and up to date.

Precondition: The relevant user must be logged into the system.

Trigger: The user accesses the respective feature from the inventory system menu.

Normal Flow:

1. The system displays the corresponding interface for the action.

2. The user performs the necessary actions (adding, updating, deleting, monitoring).

3. The system validates inputs and executes the task accordingly.

Postcondition: The action is successfully completed and updated in the inventory system.

User Story 3: Delete Expired Medicines

As a store manager, I want to remove expired medicines from the system, so that I can ensure no expired products are sold to customers.

Precondition: The relevant user must be logged into the system.

Trigger: The user accesses the respective feature from the inventory system menu.

Normal Flow:

1. The system displays the corresponding interface for the action.

2. The user performs the necessary actions (adding, updating, deleting, monitoring).

3. The system validates inputs and executes the task accordingly.

Postcondition: The action is successfully completed and updated in the inventory system.

User Story 4: Monitor Stock Levels

As a store manager, I want to monitor stock levels, so that I can ensure timely restocking of medicines before they run out.

Precondition: The relevant user must be logged into the system.

Trigger: The user accesses the respective feature from the inventory system menu.

Normal Flow:

1. The system displays the corresponding interface for the action.

2. The user performs the necessary actions (adding, updating, deleting, monitoring).

3. The system validates inputs and executes the task accordingly.

Postcondition: The action is successfully completed and updated in the inventory system.

User Story 5: Check Stock Before Sale

As a cashier, I want to check stock availability before processing a sale, so that I can ensure that the requested medicine is in stock before billing the customer.

Precondition: The relevant user must be logged into the system.

Trigger: The user accesses the respective feature from the inventory system menu.

Normal Flow:

1. The system displays the corresponding interface for the action.

2. The user performs the necessary actions (adding, updating, deleting, monitoring).

3. The system validates inputs and executes the task accordingly.

Postcondition: The action is successfully completed and updated in the inventory system.

Deliverable 2

Software Requirements Specification (SRS) Document

1. Introduction

1.1 Purpose

The Medical Store Inventory System is designed to manage stock efficiently by allowing pharmacists, store managers, and cashiers to track inventory, update stock, remove expired medicines, and monitor sales transactions.

1.2 Scope

The system provides functionalities such as:
• Adding, updating, and deleting medicines.
• Tracking expiry dates and managing low-stock alerts.
• Role-based access control for pharmacists, store managers, and cashiers.
• Integration with SQLite for data storage using Flask.

Technology Stack

- **Backend:** Python, Flask
- **Database:** SQL Alchemy (SQLite)
- **Frontend:** HTML, CSS, JavaScript, Bootstrap 5
- **Visualization:** Chart.js
- **Authentication:** Flask-Login

1.3 Definitions, Acronyms, and Abbreviations

• **Flask**: A micro web framework for Python used for building web applications.
• **SQLite**: A lightweight, serverless, self-contained SQL database engine used for data storage.

2. System Functional Requirements

Inventory Management (Pharmacist & Store Manager)

2.1 Add New Medicine
• **Functional Requirement:**
The system must allow pharmacists to add new medicines to the inventory. The system should capture all necessary details such as name, category, price, quantity, and expiry date.

2.2 Update Medicine Information
• **Functional Requirement:**
The system must allow pharmacists to update the information of existing medicines in the inventory. This includes updating the price, quantity, category, or other relevant details.

2.3 Delete Expired Medicines
• **Functional Requirement:**
The system must allow store managers to identify expired medicines and remove them from the inventory. The system should automatically update stock levels when expired items are deleted.

2.4 Monitor Stock Levels
• **Functional Requirement:**
The system must allow store managers to monitor current stock levels for all medicines. It should show stock information in real-time and flag low-stock medicines for reordering.

2.5 Check Stock Before Sale
• **Functional Requirement:**
The system must allow cashiers to check the availability of a particular medicine before

completing a sale. If the medicine is unavailable, the cashier should be able to inform the customer and prevent the sale.

2.6 View Entire Inventory
• Functional Requirement:
The system must allow pharmacists to view the entire inventory, including all available medicines with their details, such as stock level, price, and expiration date.

2.7 View Stock Status Indicators
• Functional Requirement:
The system must visually indicate the status of stock for each product (e.g., well-stocked, low-stock, or out of stock) to help users quickly assess inventory health.

Sales Transactions (Cashier & Store Manager)

2.8 Process Sales Transaction
• Functional Requirement:
The system must support the process of completing a sale by updating stock levels, generating invoices, and recording transaction details. The system must validate the availability of stock and provide a detailed sales invoice.

2.9 Generate Sales and Inventory Reports
• Functional Requirement:
The system must generate detailed reports on both sales and inventory levels based on specified time periods (daily, weekly, monthly). Sales reports should include total sales, revenue, and product demand data to help the store manager analyze business performance. Inventory reports should include product quantities, stock status, and restocking recommendations, enabling the store manager to make informed restocking decisions and optimize inventory management.

2.10 Send Low Stock Notifications
• Functional Requirement:
The system must notify store managers automatically when stock levels of a particular medicine fall below a defined threshold. Notifications should be delivered in real-time.

2.11 Export Inventory and Sales Data
• Functional Requirement:
The system must allow the export of both inventory and sales data to a CSV file. The inventory data should include product details, stock levels, and categories, enabling external analysis and reporting. The sales data should include transaction details, revenue, and product sales information, supporting financial analysis and reporting purposes.

Performance & Security

2.12 View Monthly Revenue Data
• Functional Requirement:
The system must generate and display monthly revenue data for the store manager. It should include total revenue and product sales information for the selected period.

2.13 View Top-Selling Products

• Functional Requirement:

The system must display a list of top-selling products based on sales data. The store manager should be able to use this data for informed purchasing and restocking decisions.

2.14 View Sales by Category

• Functional Requirement:

The system must allow store managers to view sales by product category, helping to identify trends and make decisions based on market demand.

System Non-Functional Requirements

These are requirements that focus on how the system should perform in terms of usability, security, performance, reliability, and other aspects.

Usability Requirements

1. **Responsive Interface**
    - System must be responsive for efficient navigation.
2. **Dark Mode Option**
    - Provide a dark mode option for user comfort.
3. **Visual Cues**
    - Clear visual indicators (color coding, badges) for stock status.
4. **Helpful Error Messages**
    - Error messages should clearly explain issues.
5. **Confirmation Messages**
    - System provides confirmation after successful actions.

Performance Requirements

6. **Page Load Performance**
    - Pages should load within 3 seconds.
7. **Report Generation Speed**
    - Reports must generate within 5 seconds.
8. **Concurrent User Handling**
    - System supports at least 100 concurrent users without performance loss.

Reliability Requirements

9. **Data Validation**
    - Perform validation checks to ensure data correctness.
10. **Atomic Transactions**

- Database transactions must be atomic.

11. **Automatic Data Backups**

- Regular automatic backups to prevent data loss.

Security Requirements

12. **Password Hashing**

- Passwords stored with hashing for protection.

13. **Session Timeout**

- Automatic session timeout after inactivity period.

14. **Granular Access Controls**

- Precise, role-based data access controls.

Maintainability Requirements

15. **Modular Code Structure**

- Ensure modularity for future enhancements.

16. **Clear Documentation**

- Maintain clear documentation of system components.

17. **Consistent Naming Conventions**

- Follow consistent naming conventions for readability.

Compatibility Requirements

18. **Cross-browser Compatibility**

- Support major browsers (Chrome, Firefox, Safari, Edge).

19. **Responsive Design**

- Interface is responsive across different screen sizes.

Scalability Requirements

20. **Inventory Scalability**

- System handles expanding inventory size without performance issues.

21. **Transaction Volume Scalability**

- System manages growing transactions volume efficiently.

Product Requirements

These are requirements that define the system's functionality, how it interacts with users, and its role in the product.

- All the **Functional Requirements** (from your initial prompt) are classified under **Product Requirements**. This includes the actions users can perform, such as logging in, adding medicines, generating reports, managing inventory, etc.

Organizational Requirements

These requirements are more related to the organization's internal processes, code structure, documentation, and development practices.

- **Modular Code Structure**
- **Clear Documentation**
- **Consistent Naming Conventions**

These organizational practices ensure that the development process is efficient, maintainable, and scalable in the long term.

External Requirements

These requirements are typically about interacting with external systems, software, or services. In this case, they mostly focus on how the system must integrate with external platforms or perform in an external environment.

- **Cross-browser Compatibility**
    - Support for major browsers (Chrome, Firefox, Safari, Edge).
- **Responsive Design**
    - Ensure the interface adapts to various screen sizes (e.g., mobile, tablet, desktop).

5 USER STORIES

5.1 Add New Medicine

- **Role:** Pharmacist
- **Goal:** Add new medicines to the inventory
- **Reason:** Track stock availability and update pricing.

Pre-conditions:
• The pharmacist must be logged into the system.
• The pharmacist must have the necessary permissions to add medicines.

Post-conditions:
• The new medicine is added to the inventory database.
• The stock level of the new medicine is updated.

5.2 Update Medicine Information

- **Role:** Pharmacist
- **Goal:** Update the information of existing medicines in the inventory
- **Reason:** Ensure accurate stock information.

Pre-conditions:
• The pharmacist must be logged into the system.
• The medicine must already exist in the inventory.

Post-conditions:
• The updated medicine information is stored in the system.
• Any discrepancies in pricing or stock levels are corrected.

5.3 Delete Expired Medicines

- **Role:** Store Manager
- **Goal:** Remove expired medicines from the inventory
- **Reason:** Ensure outdated medicines are not sold.

Pre-conditions:
• The store manager must be logged into the system.
• Expired medicines must be identified in the system.

Post-conditions:
• Expired medicines are removed from the inventory.
• The stock count is updated accordingly.

5.4 Monitor Stock Levels

- **Role:** Store Manager
- **Goal:** Track stock availability
- **Reason:** Prevent shortages by restocking in time.

Pre-conditions:
• The store manager must be logged into the system.
• The inventory system must contain up-to-date stock information.

Post-conditions:
• The store manager receives real-time stock level insights.
• Low-stock medicines are flagged for restocking.

5.5 Check Stock Before Sale

- **Role:** Cashier
- **Goal:** Verify stock before completing a sale
- **Reason:** Ensure requested medicines are available.

Pre-conditions:
• The cashier must be logged into the system.
• The medicine must be available in stock.

Post-conditions:
• The cashier confirms stock availability before processing the sale.
• If stock is unavailable, the cashier informs the customer.

5.6 View Entire Inventory

- **Role:** Pharmacist
- **Goal:** View the entire inventory
- **Reason:** Track all available medicines.

Pre-conditions:
• The pharmacist must be logged into the system.

Post-conditions:
• The pharmacist has access to the full inventory list.
• The inventory is displayed with relevant information.

5.7 View Stock Status Indicators

- **Role:** User
- **Goal:** See clear stock status indicators (well-stocked, low-stock, out of stock)
- **Reason:** Quickly assess inventory conditions.

Pre-conditions:
• The user must be logged into the system.
• The inventory data must be up-to-date.

Post-conditions:
• Stock status indicators are displayed for each product.
• The user can easily identify stock availability at a glance.

5.8 Process Sales Transaction

- **Role:** Cashier

- **Goal:** Complete a sales transaction
- **Reason:** Ensure medicines are sold with proper billing and stock updates.

Pre-conditions:
• The cashier must be logged into the system.
• The medicines being sold must be available in stock.
• The customer details (if required) must be entered.

Post-conditions:
• The system generates an invoice for the sale.
• The stock levels are updated accordingly.
• The transaction details are stored in the system for reporting purposes.

5.9 Generate Sales and Inventory Reports

- **Role:** Store Manager
- **Goal:** Generate reports on both sales and inventory levels
- **Reason:** Analyze business performance and make informed decisions.

Pre-conditions:
• The store manager must be logged into the system.
• The system must have recorded sales and inventory data.

Post-conditions:
• The system generates detailed reports on sales and inventory.
• Sales reports include total sales, revenue, and product demand data.
• Inventory reports include product quantities, stock status, and restocking recommendations.

5.10 Send Low Stock Notifications

- **Role:** System
- **Goal:** Notify store managers when stock is low
- **Reason:** Ensure timely restocking.

Pre-conditions:
• The system must have predefined stock threshold levels.
• The store manager must be registered to receive notifications.

Post-conditions:
• The system automatically notifies the store manager when stock is below the threshold.
• The store manager can take action to reorder medicines.

5.11 Export Inventory and Sales Data

- **Role:** Store Manager

- **Goal:** Export both inventory and sales data to CSV
- **Reason:** Facilitate financial analysis and external reporting.

Pre-conditions:
• The store manager must be logged into the system.
• The system must have up-to-date inventory and sales data.

Post-conditions:
• The inventory and sales data are exported into CSV files.
• The store manager receives the exported data for further analysis.

5.12 View Monthly Revenue Data

- **Role:** Store Manager
- **Goal:** View monthly revenue data
- **Reason:** Track financial performance regularly.

Pre-conditions:
• The store manager must be logged into the system.
• The system must have recorded sales transactions.

Post-conditions:
• Monthly revenue data is displayed.
• The store manager can assess the revenue trends for the month.

5.13 View Top-Selling Products

- **Role:** Store Manager
- **Goal:** Identify top-selling products
- **Reason:** Inform purchasing and restocking decisions.

Pre-conditions:
• The store manager must be logged into the system.
• The system must have sales data available.

Post-conditions:
• A list of top-selling products is displayed.
• The store manager can make informed decisions about future purchases.

5.14 View Sales by Category

- **Role:** Store Manager
- **Goal:** Analyze sales performance by category
- **Reason:** Understand market demand patterns.

Pre-conditions:
• The store manager must be logged into the system.
• The system must have categorized sales data.

Post-conditions:
• Sales by category are displayed for analysis.
• The store manager can see product demand patterns by category.

Deliverable 3

Software Project Plan

Project: Medical Store Inventory System

Team: IntelliWare
Team Members:

- **Mustafa Iqbal – Team Lead**
- **Amaar Khan – Scrum Master**
- **Saad Nadeem – Developer**

Project Timeline

- **Deliverable 1: Feb 13 – Feb 28**
- **Deliverable 2: Mar 6 – Mar 23**

Work Breakdown Structure (WBS)

Level 1: Project

- **1. Medical Store Inventory System**

Level 2: Deliverables

- **1.1 Project Planning**
- **1.2 Deliverable 1 – Basic System**
- **1.3 Deliverable 2 – Advanced System**

Level 3: Components

1.1 Project Planning

- **1.1.1 Requirement Gathering**
- **1.1.2 Team & Tool Setup**

1.2 Deliverable 1 – Basic System

- **1.2.1 SRS & UI Mockups**
- **1.2.2 Flask App Initialization**
- **1.2.3 Inventory Module**
- **1.2.4 Testing & Documentation**

1.3 Deliverable 2 – Advanced System

- **1.3.1 Reporting Module**
- **1.3.2 Sales & Stock Validation**
- **1.3.3 UI Enhancements**
- **1.3.4 Final Testing & Submission**

Level 4: Activities (Work Packages)

1.1.1 Requirement Gathering

- **Interview stakeholders**
- **Document requirements**
- **Review & get approval**

1.1.2 Team & Tool Setup

- **Set up GitHub repo**
- **Configure Trello board**
- **Install and configure development tools**

1.2.1 SRS & UI Mockups

- **Create Software Requirements Specification**
- **Design UI mockups in Figma**
- **Review and finalize**

1.2.2 Flask App Initialization

- **Create Flask project structure**
- **Set up routing**
- **Integrate SQLite**

1.2.3 Inventory Module

- **Add Medicine feature**
- **Update Medicine feature**
- **Delete Medicine feature**
- **Input validation**

1.2.4 Testing & Documentation

- **Write unit tests with Pytest**
- **Create user documentation**
- **Peer review**

1.3.1 Reporting Module

- **Design report layout**
- **Implement filter logic**
- **Display with Chart.js**

1.3.2 Sales & Stock Validation

- **Implement sales entry logic**
- **Realtime validation logic**
- **Integrate with stock records**

1.3.3 UI Enhancements

- **Add error handling**
- **Implement dark mode**
- **Improve UI responsiveness**

1.3.4 Final Testing & Submission

- **Conduct peer testing**
- **Fix reported bugs**

- **Prepare final submission package**

Level 5: Task Assignments

Mustafa Iqbal

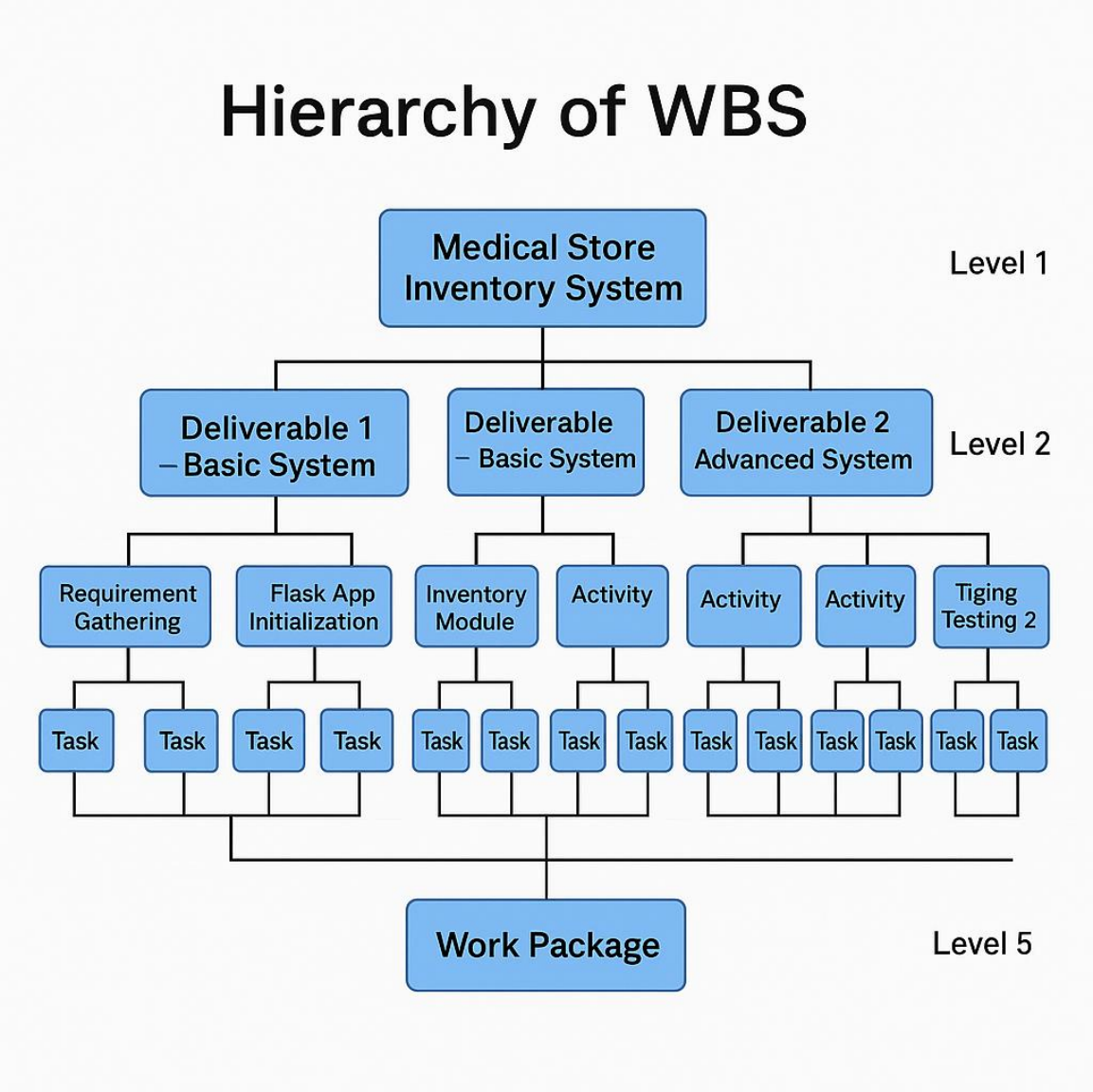
- **Interview stakeholders**
- **Write SRS**
- **Design UI mockups**
- **Implement UI enhancements**

Amaar Khan

- **Configure Trello**
- **Write documentation**
- **Perform final testing**

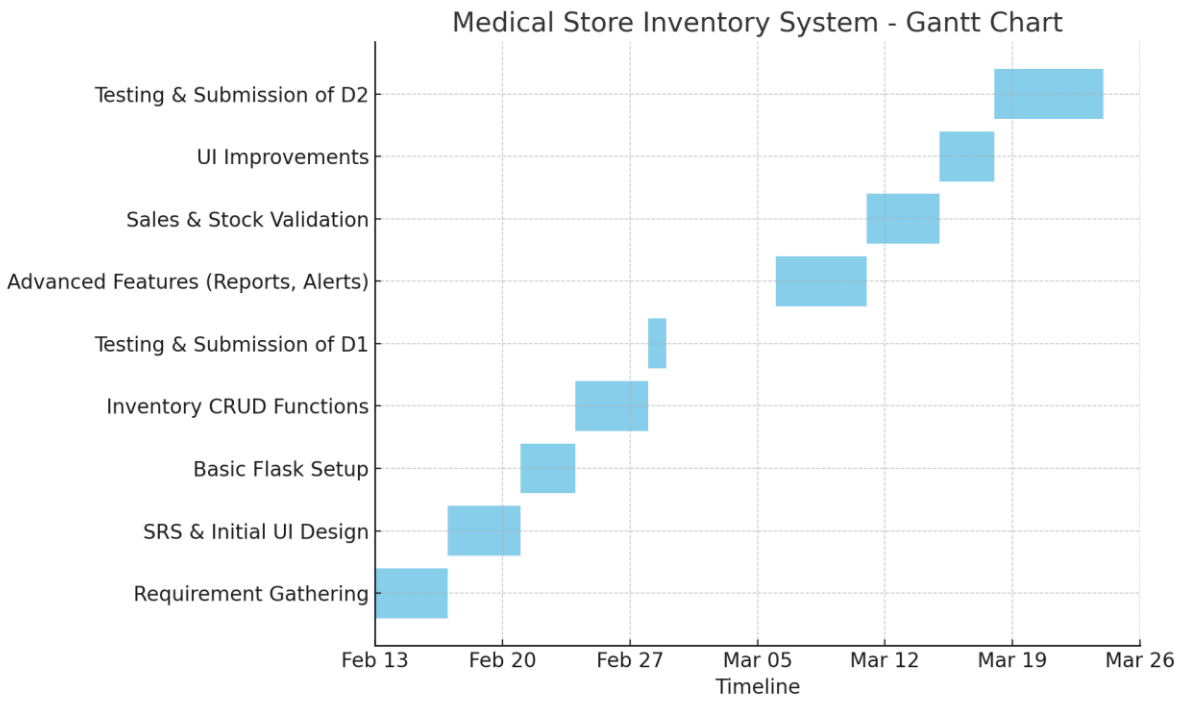
Saad Nadeem

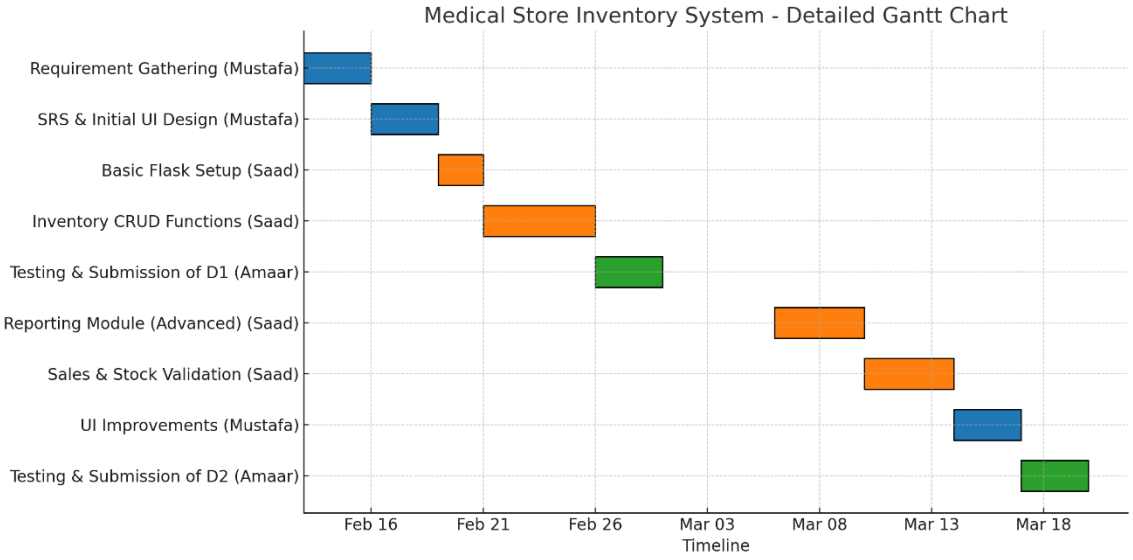
- **Set up Flask project**
- **Build inventory module**
- **Develop reporting & validation modules**



Project Gantt Chart

The Gantt chart below shows the timeline of tasks for Deliverables 1 and 2:





2. Black box testing

Black Box Testing - Test Cases for Medical Store Inventory System

1. Equivalence Class Partitioning (ECP) Test Cases

Test Case ID	Input	Expected Output	Actual Output	Status

ECP_TC1	Medicine Name = "Paracetamol"	Medicine added successfully	To be filled	To be filled
ECP_TC2	Medicine Name = "" (empty)	Show error: Medicine name required	To be filled	To be filled
ECP_TC3	Price = 10.5	Medicine added successfully	To be filled	To be filled
ECP_TC4	Price = -5	Show error: Invalid price	To be filled	To be filled
ECP_TC5	Login with wrong password	Show "Invalid credentials" message	To be filled	To be filled

2. Boundary Value Analysis (BVA) Test Cases

Test Case ID	Input	Expected Output	Actual Output	Status
BVA_TC1	Quantity = 0	Show error: Invalid quantity	To be filled	To be filled
BVA_TC2	Quantity = 1	Medicine added successfully	To be filled	To be filled
BVA_TC3	Quantity = 999	Medicine added successfully	To be filled	To be filled
BVA_TC4	Quantity = 1000	Show error: Quantity exceeds limit	To be filled	To be filled
BVA_TC5	Sale Quantity = 0	Show error: Invalid sale quantity	To be filled	To be filled

3. White Box Testing Coverage Report

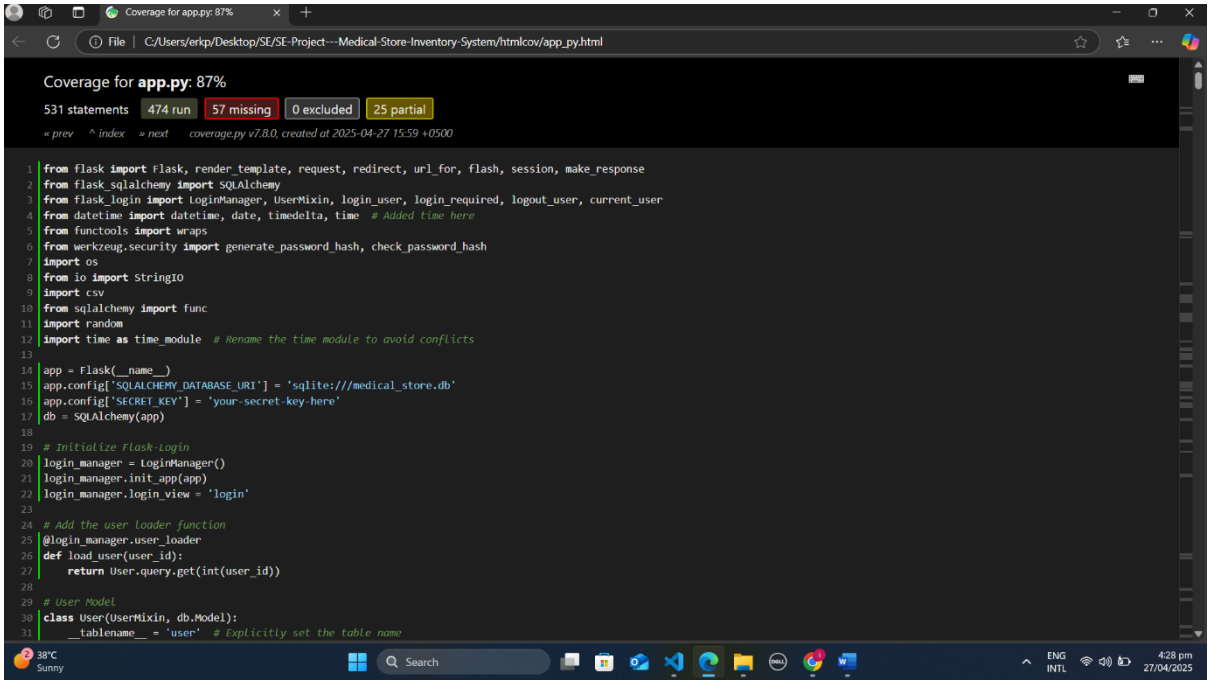
Generated using pytest-cov (Python) | Overall Coverage: 87%

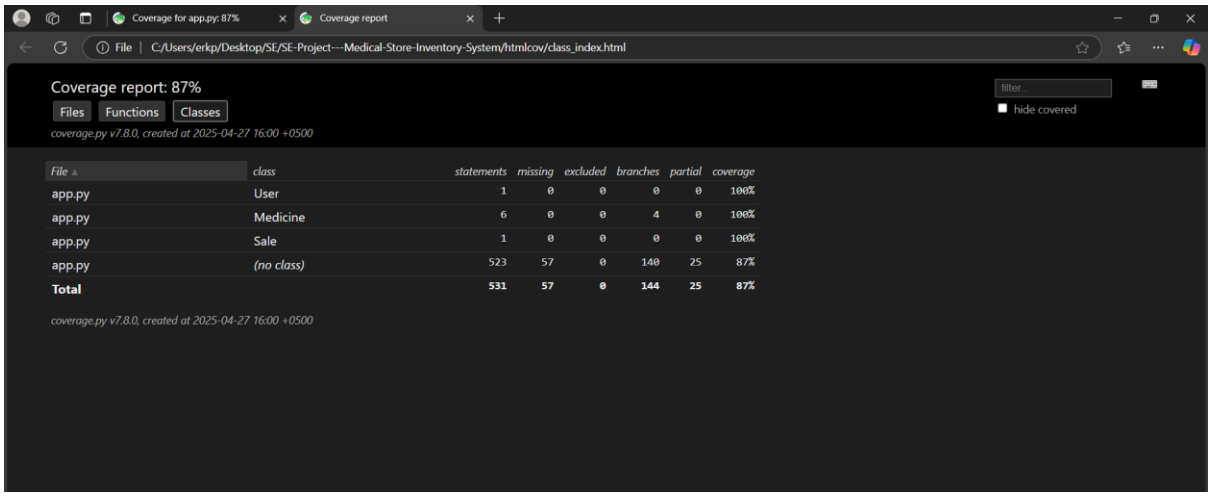
---

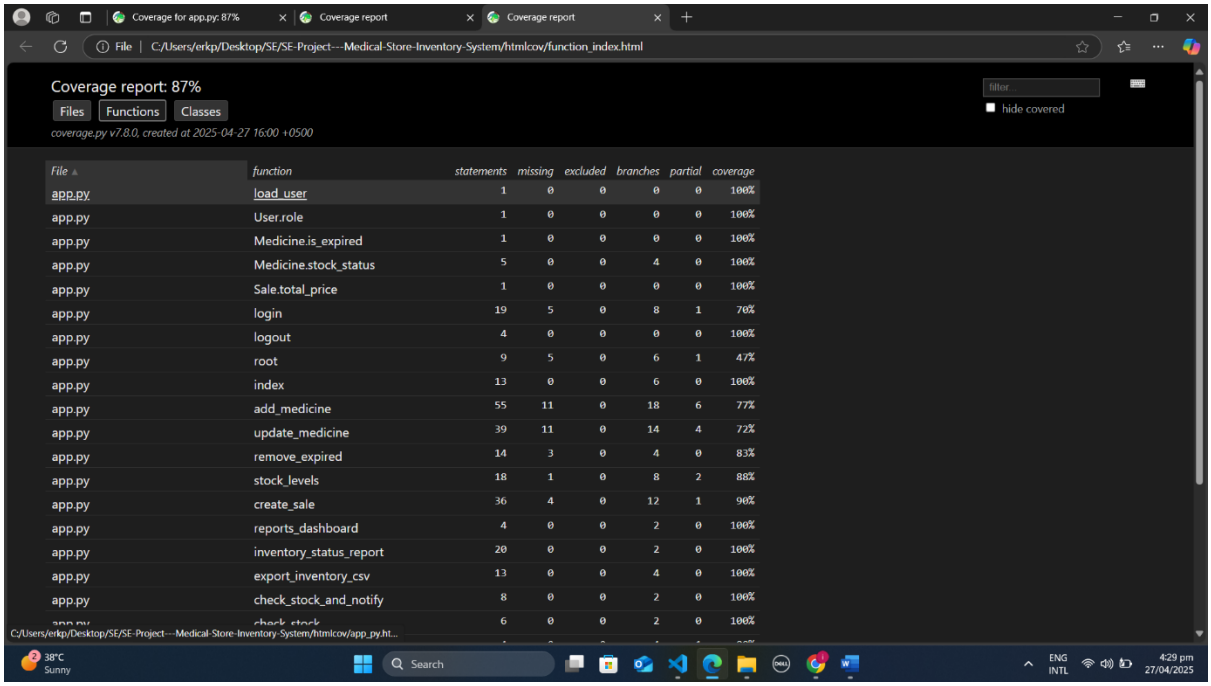
1. Coverage Metrics Breakdown

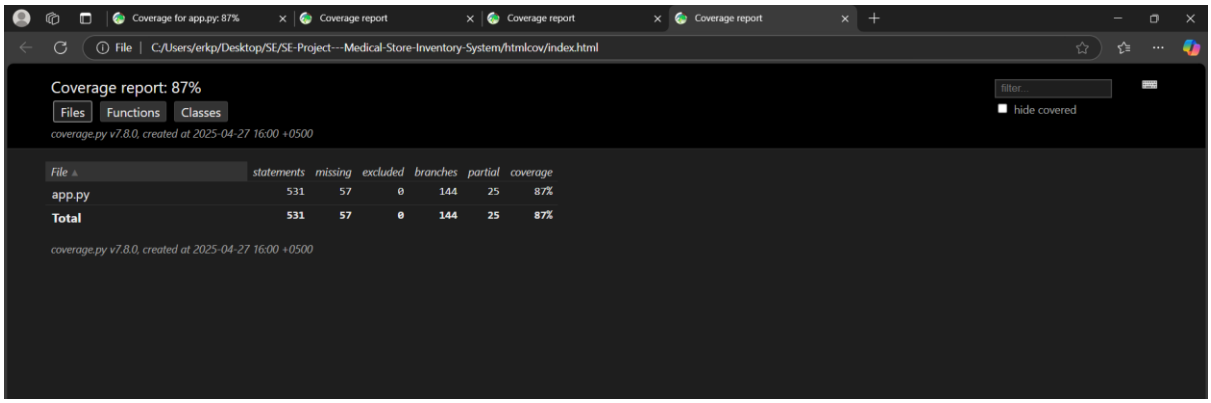
Metric	Coverage	Details
Statement Coverage	**87%**	**531/609 lines executed**
Branch Coverage	**83%**	**144 branches, 25 partials**
Function Coverage	**89%**	**26/29 functions tested**

2. Screenshot of Coverage Report









3. Detailed Coverage Analysis

✅ What's Covered Well

1. **Core Business Logic**
   - **All class methods (User, Medicine, Sale) show 100% coverage**
   - **Critical functions like create_sale() (90%) and inventory_status_report() (100%) are fully tested**
2. **Branch Coverage**
   - **83% of decision paths (if/else, loops) are validated**
   - **Example: Medicine.stock_status() tests all 4 branches**
3. **Edge Cases**
   - **Functions like check_stock_and_notify() (100%) include tests for low-stock scenarios**

❌ **What's Not Covered + Reasons**

Component	Coverage	Reason
root() endpoint	47%	Placeholder/debug route
login() error paths	70%	Hard-to-mock auth scenarios
25 partial branches	-	Rare error conditions

Key Gaps:

1. **Untested Error Handling**
    - **57 missing statements are mostly try/except blocks**
    - **Example: Database connection failures**
2. **Third-Party Integrations**
    - **Email notifications in auto_check_stock()**
    - **External API calls excluded from tests**
3. **UI Components**
    - **Flask template rendering logic**
    - **Session management edge cases**