Software Engineering Major Project Task 3

Cooper Truong

Client: Stephen Truong

1.0 Identifying and Defining

1.1 Introduction

In today's competitive retail environment, efficient inventory management and stock analytics are crucial for a business to succeed. In knowing this I am attempting to produce a solution to help aid my client's retail shop.

The Retail Pro+ will be targeted to and used mainly by retail shop owners, alongside the employees of the retail shop. The software, therefore should be suitable enough for people with minimal levels of computer experience use. My client will be my Dad who owns a retail shop of his own.

Currently, my client uses a rather old and ineffective system to manage his inventory. This approach is susceptible to a variety of bugs and missing important features such as real-time stock tracking, automated data insights, and reporting tools. Consequently, my client frequently grapples with challenges relating to overstocking, understocking, and inaccurate inventory records, and often struggles to identify effective strategies to improve business operations. All of these complications not only interfere with day-to-day operations but also result in forgone sales, excess storage costs, and poor customer experience.

Additionally, the existing system provides little to no visibility of sale trends, making it difficult to identify top sellers or predict demand. This limitation hinders informed decision-making, especially when it comes to restocking or promotional planning.

To mitigate these difficulties, Retail Pro+ offers an up-to-date and more streamlined approach. It will feature real-time inventory/stock tracking, sale analytics, and a user-friendly interface. Ultimately, Retail Pro+ will reduce my client's operational inefficiencies, reduce costly waste, and allow smarter decisions to be made which will greatly benefit them in a competitive environment.

1.2 Functional Requirements

1. **Inventory Management System**
    a. Add/Edit/Delete Products

    Users must be able to add new products to the inventory with fields like name, retail price, and supplier price.

- Technical Details: A backend SQLite database stores product data in an Inventory table. The deactivation is handled by an is_active boolean flag, implementing a soft-delete pattern. Product images are handled using the Pillow (PIL) library.

b. Manual stock updates

Employees will be able to manually update stock levels through the system interface. This includes adjusting quantities when new stock arrives, correcting discrepancies, or updating after returns.

- Technical Details: The product edit form includes an adjustable stock field. Form validation is used to prevent negative stock values.

c. Inventory search and filtering

Users must be able to search and filter inventory by price, retail price, and stock count. Inventory will appear in a scrollable table. A search bar will be available to search for a specific item.

- Technical Details: Filtering is implemented via parameterised SQLAlchemy queries against the SQLite database, which are enhanced by database indexes for performance.

2. **Sales Analytics**

a. Sales data tracking

Log every sale, including details like date/time, item(s) sold, quantity and total sale price

- Technical Details: Sales are stored in normalised Sale and SaleItem tables, linked by foreign keys to the Inventory and User tables to maintain data integrity.

b. Performance reports

Generate daily/weekly/monthly reports highlighting best-selling products, sales trend, and revenue

- Technical Details: Data is aggregated using SQLAlchemy and loaded into Pandas DataFrames for analysis. Visualisations are rendered directly in the UI using the Plotly library to create interactive charts.

c. Forecasting

Forecast future sales in the upcoming day, week, year

- Technical Details: This is achieved using the Prophet library, which is specifically designed for time-series forecasting. The data is pre-processed to account for days with no sales to improve model accuracy.

3. **User and Workspace Management**

   a. Login System

Users must authenticate using a unique email and a secure password before accessing the application. The system supports multiple user roles ('Admin' and 'Employee') to control access to features and data.

- Technical Details: Passwords are never stored in plain text; they are securely hashed using the bcrypt algorithm. Authentication is managed through Streamlit's session state, and access control is enforced through conditional logic in the backend

   b. Two-Factor Authentication (2FA)

To enhance security, users are required to provide a second form of verification when logging in. A time-sensitive, 6-digit code is sent to their registered email address to confirm their identity

- Technical Details: A secure random number is generated for the one-time password. The code is sent using the smtplib and ssl libraries over a secure connection to the email provider.

   c. Workspace Collaboration

The system is built around collaborative workspaces. Users can create multiple workspaces, and the 'Admin' of a workspace can invite other users to join via a unique email link, as well as remove existing members.

- Technical details: Workspaces, members, and pending invitations are managed across several related tables in the database (Workspace, WorkspaceMember).

   d. Team Chat:

Each workspace contains a dedicated real-time chat feature, allowing team members to communicate within the application.

- *Technical Detail:* Chat messages are stored in the WorkspaceMessage table, linked to the relevant user and workspace. The chat interface uses Streamlit's auto-refresh capability to display new messages.

e. Two-Factor Authentication (2FA)

To enhance security, users must be required to provide a second form of verification when logging in to confirm their identity.

- Technical Details: Implemented as an email-based One Time Password (OTP) system. Secure 6-digit authentication code generated using python’s random library. This code is sent to the user’s registered email address using the smtplib and ssl libraries over a secure connection.

1.3 Non-Functional Requirements

1. **Performance**

   The system/software must feel fast and responsive. The retail employees should be able check stocks, add products and generate reports quickly and efficiently without delay, especially during busy periods

   - Technical Details: Performance is achieved through the use of indexed database fields for quick lookups, efficient SQLAlchemy queries to minimise database load, and a responsive Streamlit front-end.

2. **Scalability**

   The system must be able to handle growing amounts of data. As when more items are added to the database/inventory, the system must cope and remain fast and efficient rather than slow.

   - Technical Details: The database schema is fully normalised to reduce redundancy and allow for efficient growth. The use of the SQLAlchemy ORM allows for flexible and scalable database interactions.

3. **Usability**

   Accounting for the fact that my client for my software has limited tech experience, the system must be easy to learn and use. A confusing interface could lead to mistakes like incorrect stock updates.

   - Technical Details: The UI is built with Streamlit, providing a clean and minimalistic interface. A consistent layout, clear navigation sidebar, descriptive labels, and confirmation prompts are used throughout the application to enhance usability and prevent user error.

4. **Reliability**

The system needs to be stable and trustworthy, errors or crashes could result in incorrect data affecting the business.

- Technical Details: Data integrity in critical operations, like processing a sale, is ensured by using database transactions, which guarantees that multi-step database changes either complete successfully or fail together, preventing data corruption. Pre-check loops are also used to validate data before it is committed to the database.

5. **Security**

The system must protect sensitive data, including user credentials and sale information, from unauthorised access and cyber threats

- Technical Details: Security is addressed through multiple layers: bcrypt password hashing , two-factor authentication , parameterised queries to prevent SQL injection , a brute-force login attempt lockout mechanism, and HTML escaping on all user-generated content to prevent Cross-Site Scripting (XSS) attacks.

1.4 Feasibility Study

Technical Feasibility

After looking into the requirements of this project, I am satisfied as it is possible to complete in a given period. I would like to implement the application’s core logic as well as data operations with Python, and use the UI interface Streamlit to display the information. As it is mostly centred around data, the Steamlit framework will be able to provide a fast, intuitive and modern interface. I have worked with these technologies before and therefore, they should not pose a challenge from an implementation point of view.

Time Feasibility

The final submission for this project is due in Term 2, Week 9A, on Tuesday, 24th June 2025. Since I am currently still in the planning phase of my major project, I have plenty of time to work on it. Nonetheless, I do not intend to take the entire timeline for development. Time management is critical for any project. In this regard, my plan is to wrap up important design tasks like coding and implementing many parts of the system long before the deadline. This approach will ensure there is a dedicated and significant period remaining for the crucial final stages of the project.

Financial Feasibility

The project is economically feasible. The main cost of this project will be my own development time. The software I plan to use, including Python, Streamlit, and the SQLite database, is open-source and free of charge.

1.5 Defining Boundaries

To ensure the project remains focused and achievable within the specified timeframe, I have defined the following boundaries to clarify what the Retail Pro+ system will and will not do.

In Scope

- *User and Workspace Management:* The system will offer secure user login with password requirements, two-factor authentication, and role-based access control (Admin/Employee).

- *Inventory Control:* I will implement features to allow users to add, edit, and delete products, manually update stock levels, and use search and filtering tools to view the inventory.

- *Sales Processing:* The software will allow users to log every sale, including the items sold and total price, which will automatically adjust inventory levels.

- *Reporting and Analytics:* I will build a reporting module to generate daily, weekly, and monthly reports on key metrics like best-selling products and overall sales trends.

- *Predictive forecasting*: I will implement an advanced sales forecasting tool. To improve accuracy, the model will be trained on a complete historical timeline that accounts for periods of no sales.

- *AI Insights:* I will integrate the Google Gemini AI model as a business assistant. It will generate performance reports based on a context-aware snapshot of live business data and will also feature an interactive chat for the client to ask questions and receive data-driven answers.

Out of Scope

- *Direct Payment Processing:* The application will record sales data, however, it will not integrate with payment gateways or process any financial transactions such as credit card payments.
- *Hardware Integration:* The software will not be designed to integrate with specialised peripherals like barcode scanners or receipt printers.

- *E-Commerce Integration: This will be an internal management tool for a physical shop and will not connect to any online sales platforms.*

1.6 Social and Ethical Issues

As the developer of Retail Pro+, I have identified and addressed several key social and ethical implications during the project's design. The primary ethical issue is safeguarding the security and privacy of sensitive business and system user information. Regarding this aspect, my design includes critical security features to mitigate these risks such as automated bcrypt hashing for all user passwords, two-factor authentication (2FA), and use of parameterised SQL queries to mitigate database injection attacks. Moreover, although the software performs several manual functions, its role in my client's small business specifically is not to eliminate employment but rather to assist staff by reducing effort for tedious data entry work so enhancing operational processes helps improve the workflows and customer interactions. Finally, with regard to forecasting and report generation, AI bias or errors could introduce inaccuracies or unfair assumptions, so I ensure that all AI-generated materials clearly indicate they are meant as drafts tailored for client analysis, ensuring that a human with business context always makes the final decision.

1.7 Tools and Resources

To successfully develop Retail Pro+, I have selected the following hardware, software, and key resources.

Hardware

I will be using my own laptop (Yoga Pro 7 14ARP8) as it is capable of handling and running such code. The final application will be available to run on on a standard computer in my client's retail shop.

Software

- *Programming Language (Python):* I have selected Python because its libraries are vital for this project's requirements, especially in data analysis and web development.

- *Web Framework (Streamlit)*: I will use Streamlit to build the user interface. This framework will allow me to create a clean, responsive, and easy-to-use application suitable for a client with limited technical expertise.

- *Database (SQLite):* I chose SQLite as it is a serverless, lightweight database that is ideal for this application. It will be embedded within the application, thus making maintenance and deployment easier.

Core Python Libraries

- *Pandas*: I will use this library for data manipulation, which is essential for creating the sales and performance reports.
- *Sckit-learn and Prophet:* Will be used to implement the prediction/forecasting of sales
- *Bcrypt:* I will use this to hash user passwords, providing an important layer of security
- *Smtplib and ssl:* will be used to send secure emails for two factor authentication and password resets
- *Pillow (PIL):* I will use this library to handle the uploading and processing of images (in this case the product images)

Diagram Tools

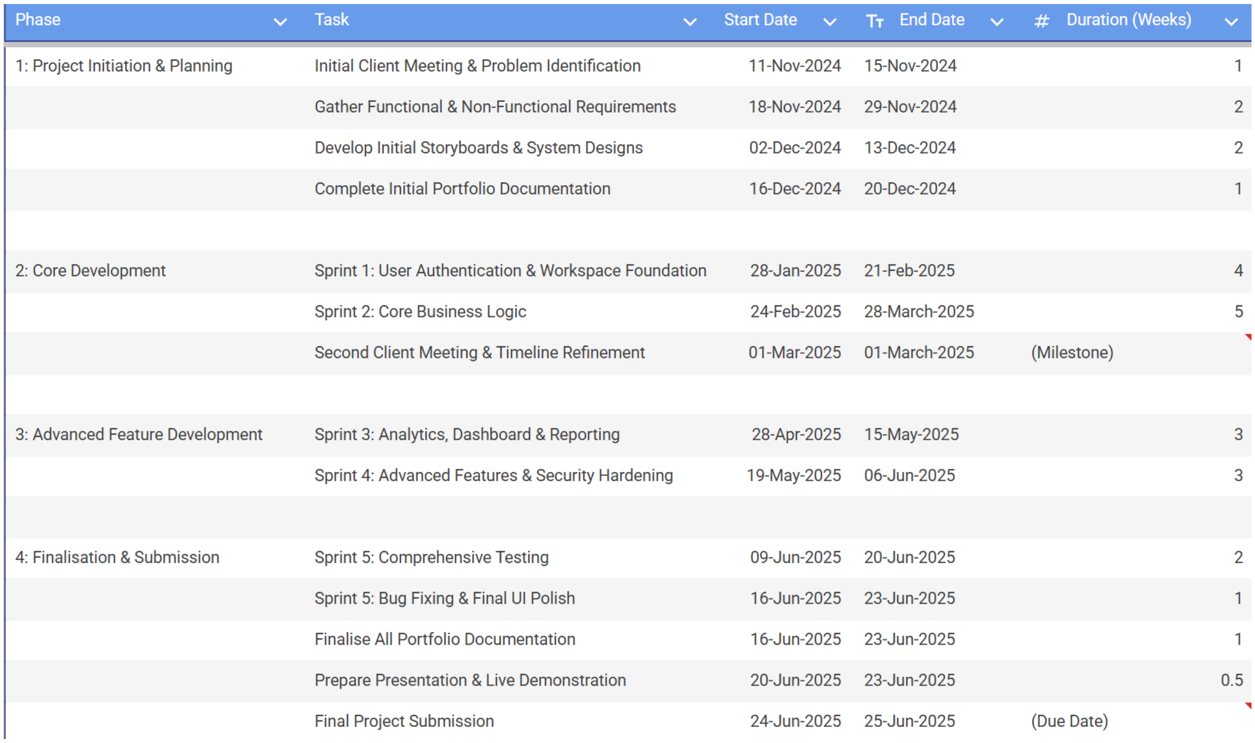
I am using LucidChart to create the system diagrams.

2.0 Research and Planning

2.1 GANTT Chart

	Nov (24)	Dec (24)	Jan (25)	Feb (25)	Mar (25)	April (25)	May (25)	Jun (25)
Problem Identification	🟦							
Gather Requirements	🟦							
Storyboards & System Design		🟦						
Initial Portfolio Documentation		🟦						

Sprint 1			🟦	🟦				
Sprint 2				🟦	🟦			
Client Meeting & Timeline Review					🟦			
Sprint 3						🟦	🟦	
Sprint 4							🟦	🟦
Sprint 5								🟦
Final Portfolio Documentation								🟦
Prepare Presentation & Live Demo								🟦
Final Project Submission								🟦



Phase	Task	Start Date	End Date	Duration (Weeks)
1: Project Initiation & Planning	Initial Client Meeting & Problem Identification	11-Nov-2024	15-Nov-2024	1
	Gather Functional & Non-Functional Requirements	18-Nov-2024	29-Nov-2024	2
	Develop Initial Storyboards & System Designs	02-Dec-2024	13-Dec-2024	2
	Complete Initial Portfolio Documentation	16-Dec-2024	20-Dec-2024	1
2: Core Development	Sprint 1: User Authentication & Workspace Foundation	28-Jan-2025	21-Feb-2025	4
	Sprint 2: Core Business Logic	24-Feb-2025	28-March-2025	5
	Second Client Meeting & Timeline Refinement	01-Mar-2025	01-March-2025	(Milestone)
3: Advanced Feature Development	Sprint 3: Analytics, Dashboard & Reporting	28-Apr-2025	15-May-2025	3
	Sprint 4: Advanced Features & Security Hardening	19-May-2025	06-Jun-2025	3
4: Finalisation & Submission	Sprint 5: Comprehensive Testing	09-Jun-2025	20-Jun-2025	2
	Sprint 5: Bug Fixing & Final UI Polish	16-Jun-2025	23-Jun-2025	1
	Finalise All Portfolio Documentation	16-Jun-2025	23-Jun-2025	1
	Prepare Presentation & Live Demonstration	20-Jun-2025	23-Jun-2025	0.5
	Final Project Submission	24-Jun-2025	25-Jun-2025	(Due Date)

2.2 Agile Sprint/Iteration Plan

My development will follow an Agile methodology, broken down into five distinct sprints. Each sprint is designed to produce a tangible and testable part of the final application, allowing for continuous feedback and refinement throughout the project lifecycle.

Sprint 1 - User Authentication and Workspaces
The main focus of this sprint is to create the basic framework for user management as well as data separation. I will start with designing and initialising an SQLite database schema, which includes the creation of users, inventory, and sales tables. Following that, I will make the first registration page where created passwords must follow a certain requirement (minimum length, capital letters, etc). I will then develop the secure login system, ensuring all passwords are hashed using the bcrypt library. Finally, I will establish the basic application structure using Streamlit and create the functionality for a user to have an active workspace, which will form the foundation for all subsequent features.

Sprint 2 - Inventory and Sales
This sprint focuses on the main value proposition of the application: managing inventory and processing sales. My initial focus will be on the inventory management page, where users can add, edit, or delete products and perform manual stock updates when new stock arrives or discrepancies are found. After completing this step, I will proceed to develop the sales processing page. With this feature, a user can search for items, add them to an ongoing order

and record the sale including details such as date of transaction and sold items with their respective quantities. To uphold data consistency, automatic adjustments to inventory levels will be done by the system when returns are processed and during finalisation of sales.

Sprint 3 - Analytics and Reporting

With the core data being captured, the goal of this sprint is to build the tools for the client to gain insights from that data. As portrayed in the storyboards, I will create the main dashboard that shows sales figures for today, this week and this year. I will also create a specific reports page which will automatically compile performance reports including the most sold items and sales trends over time. For reporting data visually, I am going to apply current charting libraries such as Plotly and make interactive graphs that convey relevant and important information easily.

Sprint 4 - Advanced Features and Security Hardening

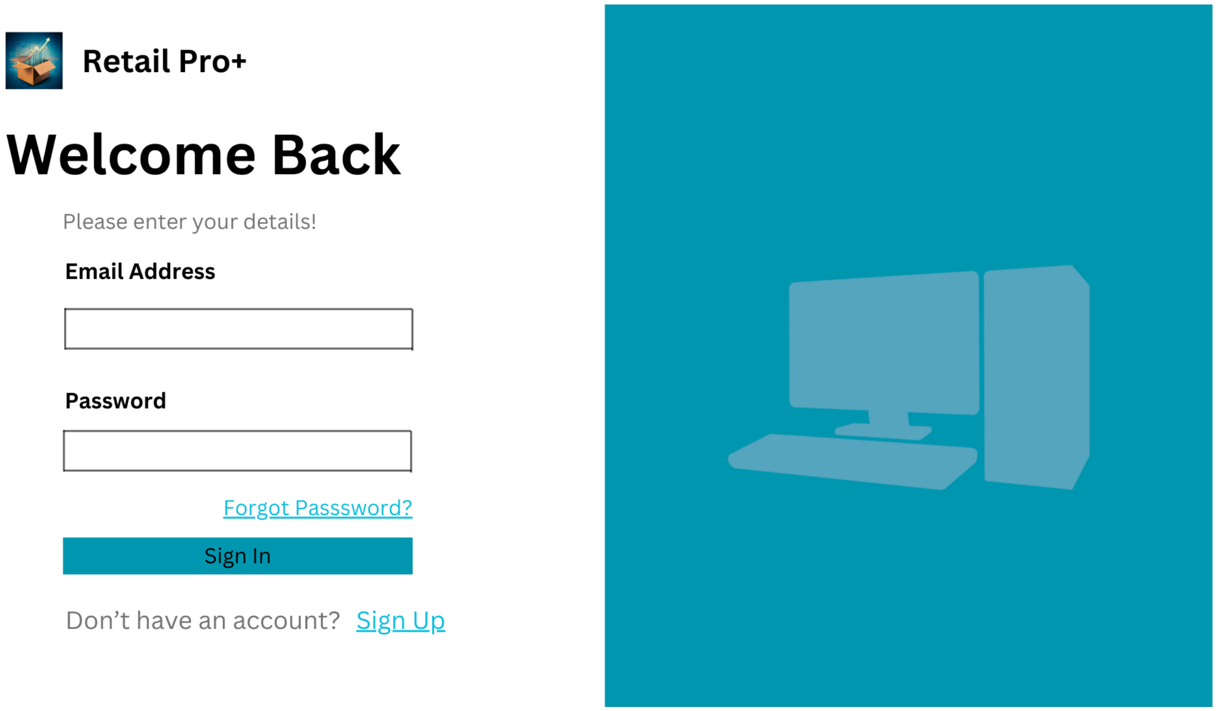
This sprint aims to refine the application by adding innovative and complex features while strengthening security. I will start by implementing the sales forecasting feature that aids in predicting sales for restocking purposes. As per my plan, this will use a time-series model to allow for advanced analytics. I will then integrate the Google Gemini AI service, which is a core component of my system design, to generate business reports and provide a chat interface. To improve security measures, I intend to implement two-factor authentication (2FA) where users receive verification through email. Finally, I will extend the system by setting up role-based access control so employees have restricted access relative to administrators.

Sprint 5 - Finalisation, Testing and Handover

The final sprint is dedicated to polishing the application and preparing for the final submission and client handover. I will conduct comprehensive testing across all features to define the effectiveness of the software and evaluate its performance. This will be followed by a round of bug fixing and adjustments to the user interface that will stem from my tests as well as additional feedback from the client. Finally, I will develop presentation slides and rehearse the live software demonstration.

2.3 System Modelling

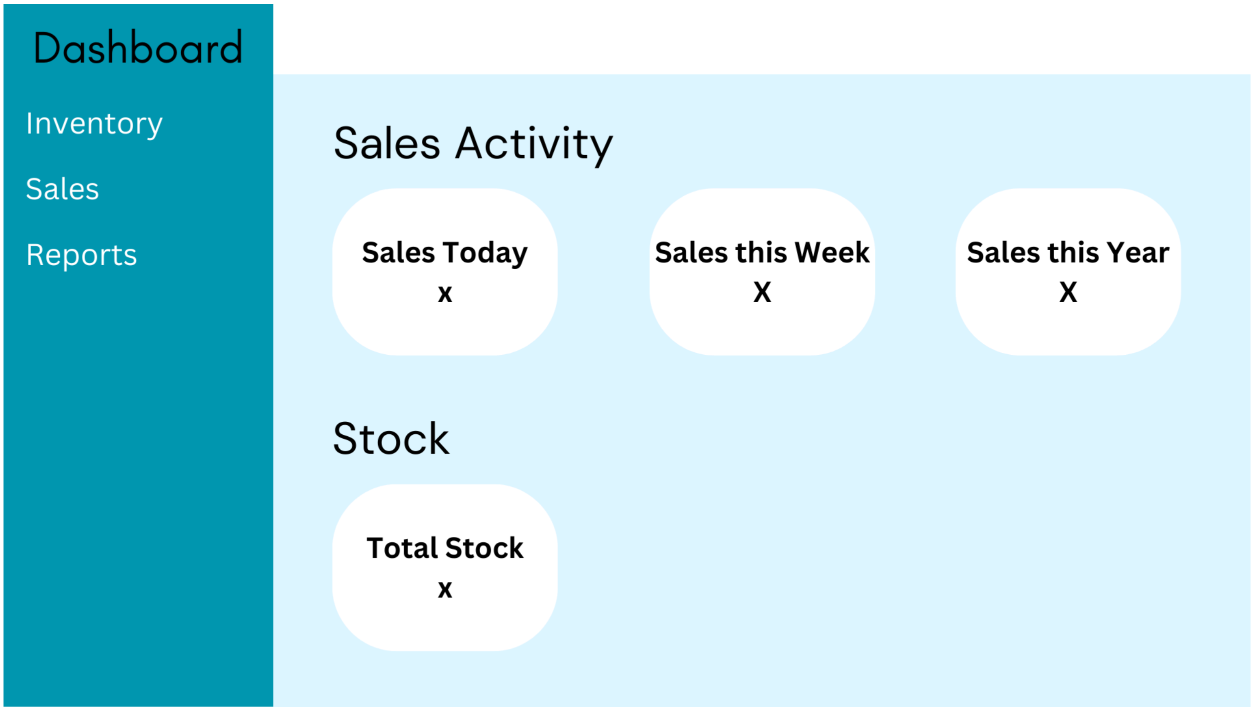
Storyboard

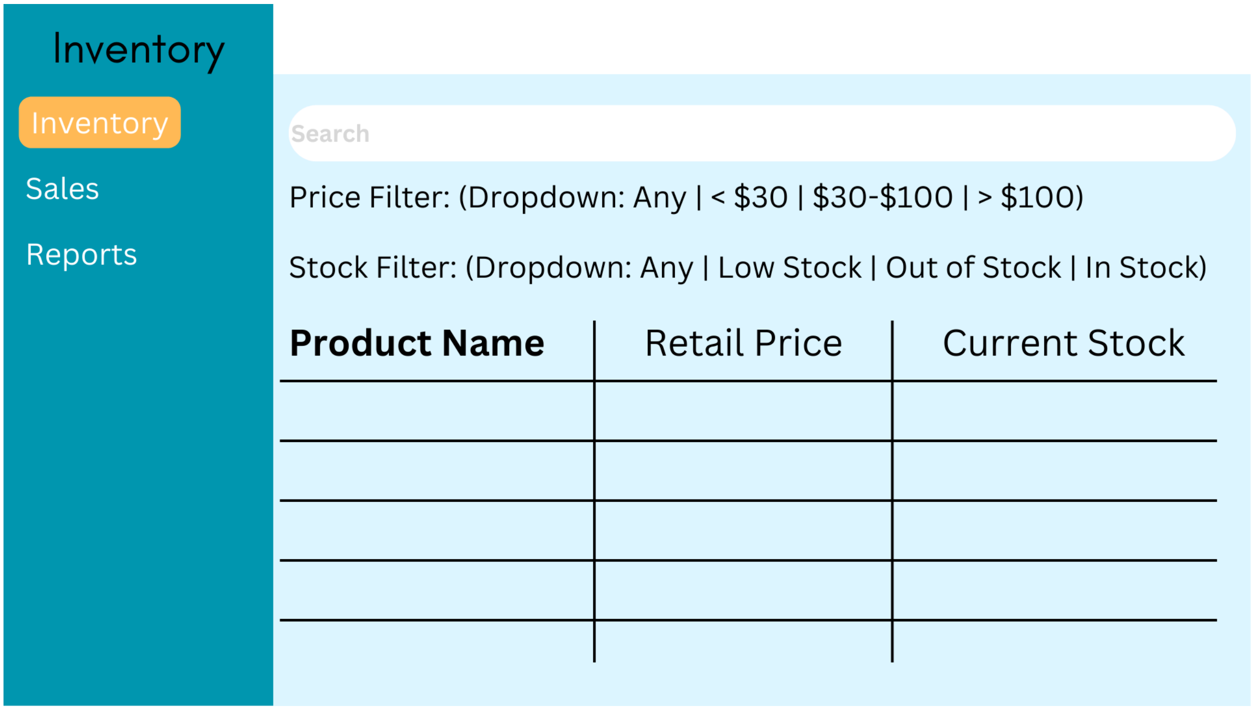


Welcome Back

Please enter your details!

Don’t have an account? Sign Up



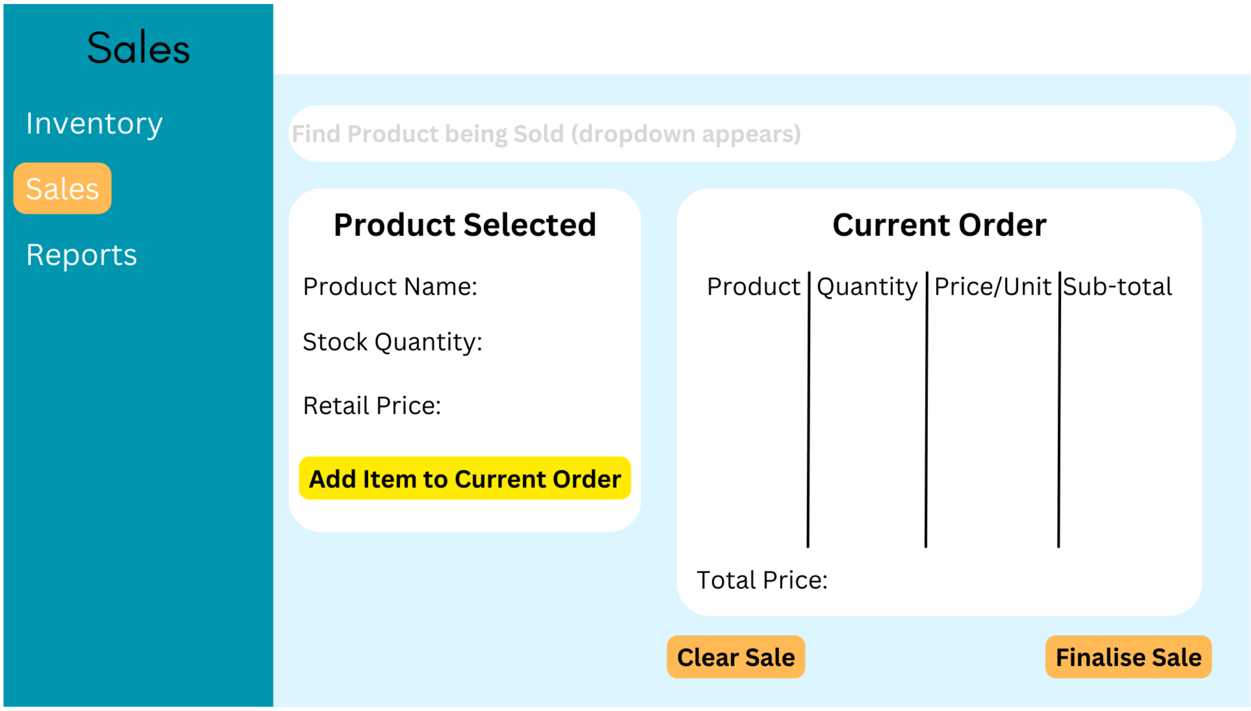


Search

Price Filter: (Dropdown: Any | < $30 | $30-$100 | > $100)

Stock Filter: (Dropdown: Any | Low Stock | Out of Stock | In Stock)

Product Name	Retail Price	Current Stock



Find Product being Sold (dropdown appears)

Product Selected

Product Name:

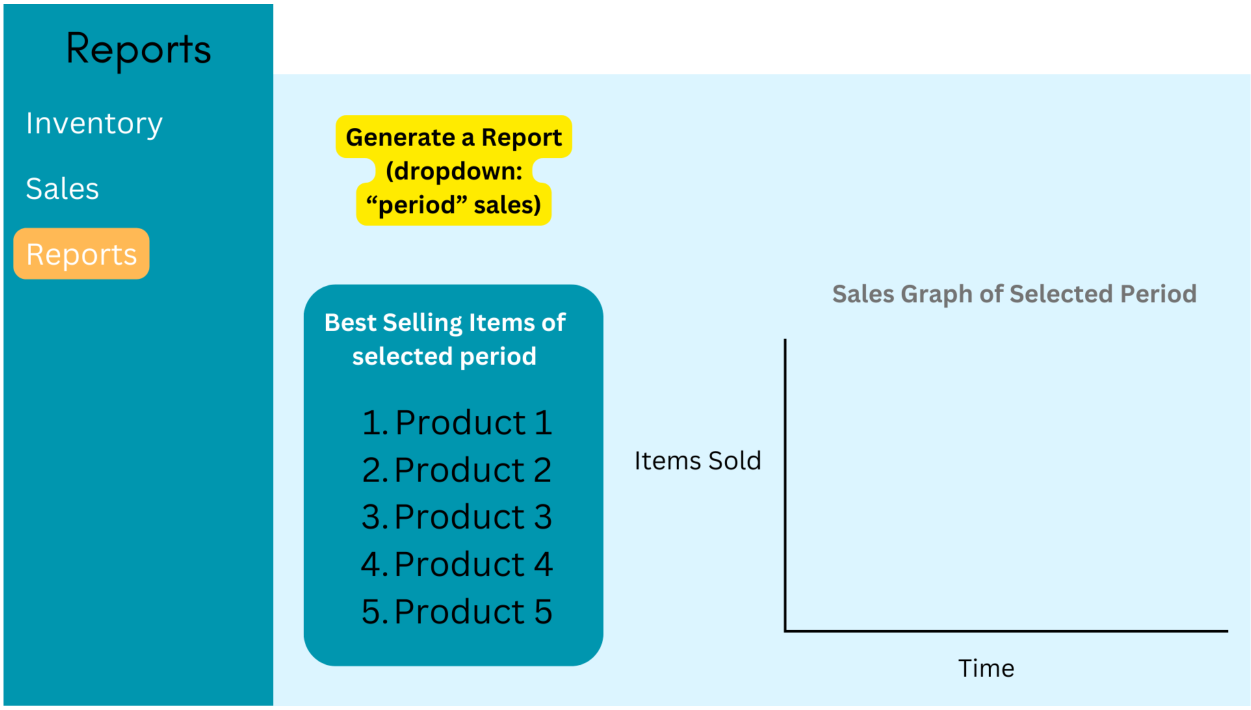
Stock Quantity:

Retail Price:

Current Order

Product	Quantity	Price/Unit	Sub-total

Total Price:


Reports
Inventory
Sales
Reports
Generate a Report (dropdown: “period” sales)
Best Selling Items of selected period
1. Product 1
2. Product 2
3. Product 3
4. Product 4
5. Product 5
Sales Graph of Selected Period
Items Sold
Time

Pseudocode

Outline: The pseudocode acts as a detailed, language-agnostic blueprint for the application's logic, outlining every function from user registration with password hashing to generating AI-powered reports. It defines the step-by-step algorithms for all core processes, including database interactions and the main application loop.

```
BEGIN

IMPORT all necessary libraries (Streamlit, SQLAlchemy, bcrypt, etc.)

DEFINE Global Constants:
DATABASE_FILE = "retail_pro_plus_v3.db"
IMAGE_DIRECTORY = "inventory_images"
EMAIL_SENDER_ADDRESS
EMAIL_SENDER_PASSWORD
SMTP_SERVER_DETAILS

INITIALISE SQLAlchemy Database Engine and Session
DEFINE Database Table Models (Schema):
TABLE User(id, email, password_hash, name)
TABLE Workspace(id, name, owner_id)
TABLE WorkspaceMember(id, workspace_id, user_id, role, status, invite_token)
TABLE WorkspaceMessage(id, workspace_id, user_id, content, timestamp)
TABLE Inventory(id, workspace_id, name, price, stock, image_path, is_active)
TABLE Sale(id, workspace_id, user_id, datetime, total_amount)
TABLE SaleItem(id, sale_id, inventory_item_id, quantity, price_at_sale)

FUNCTION register_new_user(email, password, name)
RECEIVE user details.
HASH the provided password using bcrypt.
CREATE a new User record in the database.
IF user creation is successful:
CREATE a default "My Workspace" for the new user.
LINK the user to their new workspace as the owner.
RETURN success.
ELSE (e.g., email already exists):
DISPLAY error message.
RETURN failure.
END FUNCTION

FUNCTION find_user_by_email(email)
QUERY database for a user with the matching email.
RETURN user data if found, otherwise return NULL.
END FUNCTION

FUNCTION check_user_password(plain_password, hashed_password)
COMPARE the plain password with the hashed password using bcrypt.
RETURN true if they match, otherwise false.
END FUNCTION

FUNCTION send_email(recipient, subject, body)
CONNECT to SMTP server using predefined credentials.
CREATE email message.
SEND email.
RETURN success or failure.
END FUNCTION

FUNCTION get_user_workspaces(user_id)
QUERY database for all workspaces the user is an 'accepted' member of.
RETURN list of workspaces.
END FUNCTION
```

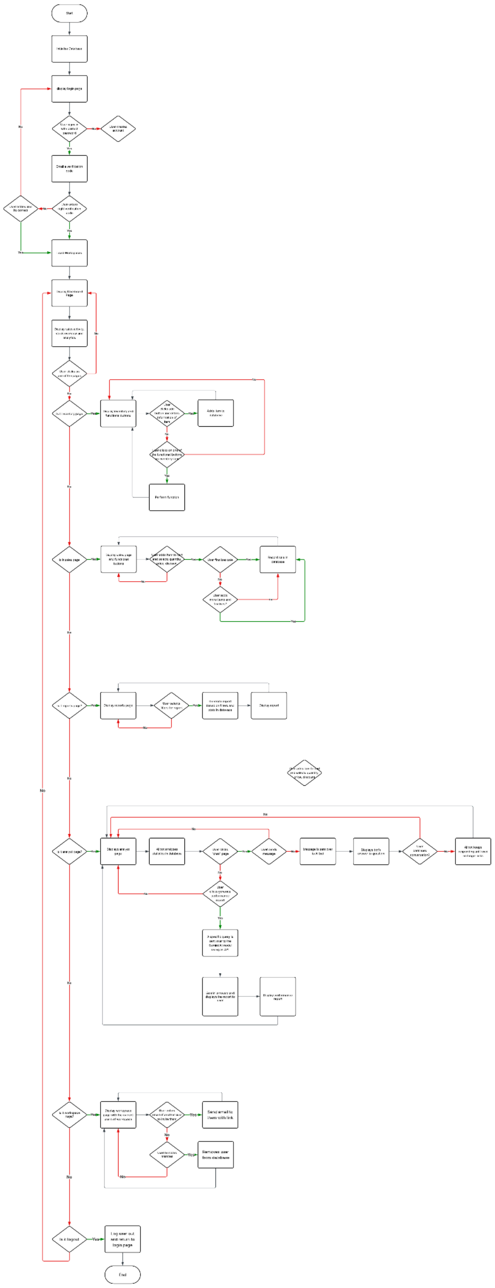
```
FUNCTION invite_workspace_member(workspace_id, invitee_email, inviter_name)
    GENERATE a unique invitation token.
    CREATE an invitation link with the token.
    CREATE a 'pending' record in the WorkspaceMember table.
    SEND an invitation email to the invitee with the link.
    RETURN success or failure.
END FUNCTION

FUNCTION process_workspace_invitation(token, user_id)
    FIND the pending invitation in the database using the token.
    IF token is valid and invitation is for the current user:
        UPDATE the WorkspaceMember record status to 'accepted'.
        ASSIGN the user_id to the record.
        RETURN success message.
    ELSE:
        RETURN error message.
    END IF
END FUNCTION

FUNCTION remove_workspace_member(workspace_id, member_id, current_user_id)
    VERIFY the current user is the owner of the workspace.
    IF authorised:
        DELETE the member's record from the WorkspaceMember table.
        RETURN success.
    ELSE:
        DISPLAY "Not authorised" error.
        RETURN failure.
    END IF
END FUNCTION

FUNCTION add_product(workspace_id, name, price, stock, image)
    VALIDATE inputs (e.g., name not empty, price/stock not negative).
    IF an image is uploaded, SAVE it to the server and get the path.
    CREATE a new Inventory record in the database for the current workspace.
    RETURN success or failure.
END FUNCTION

FUNCTION get_products(workspace_id, filters)
    QUERY the Inventory table for items in the current workspace.
    APPLY any filters provided (search term, price range, stock status).
    RETURN a list of matching products.
END FUNCTION

FUNCTION update_product(item_id, new_data)
    FIND the product in the database by its ID.
    UPDATE its record with the new data (name, price, stock).
    RETURN success or failure.
END FUNCTION

FUNCTION deactivate_product(item_id)
    FIND the product in the database.
    SET its 'is_active' flag to false (soft delete).
    RETURN success or failure.
END FUNCTION

FUNCTION record_new_sale(workspace_id, user_id, cart_items, total_amount)
    CREATE a new Sale record.
    FOR EACH item in the cart:
        CREATE a new SaleItem record linked to the Sale.
        DECREMENT the stock level of the corresponding item in the Inventory table.
    END FOR
    RETURN success or failure.
END FUNCTION

FUNCTION get_sales_summary(workspace_id)
    QUERY all sales for the workspace.
    CALCULATE total sales for today, this week, and this year.
    RETURN a dictionary of the calculated totals.
END FUNCTION

FUNCTION get_best_sellers(workspace_id)
    QUERY and join Sales, SaleItems, and Inventory tables.
    GROUP by product to sum quantities sold.
    ORDER by total quantity sold in descending order.
    RETURN a list of the top-selling products.
END FUNCTION

FUNCTION generate_sales_forecast(item_id)
    FETCH the sales history for a specific product.
    PREPARE the data for the Prophet forecasting model (ds, y columns).
    TRAIN the Prophet model on the historical data.
    PREDICT sales for the next day, week, and 30 days.
    RETURN the predictions.
END FUNCTION

FUNCTION generate_ai_performance_report(workspace_data)
    CONNECT to the Google Generative AI API.
    CREATE a detailed prompt containing the workspace's sales and inventory data.
    INSTRUCT the AI to act as a business analyst and generate a report with highlights and suggestions.
    SEND the prompt to the AI model.
    RETURN the AI-generated text report.
END FUNCTION

PROCEDURE start_application
    INITIALISE Streamlit page configuration and session state variables (logged_in_user, current_page, cart, etc.).

    IF user IS NOT logged in:
        DISPLAY the Retail Pro+ logo and title.
        GET the current authentication flow step (e.g., 'login', 'signup', 'forgot_password').
        IF step is 'login', CALL show_login_page.
        IF step is 'signup', CALL show_signup_page.
        IF step is 'enter_2fa', CALL show_two_factor_auth_page.
        IF step is related to password reset, CALL the relevant password reset page.
        EXIT PROCEDURE.
    END IF


    REFRESH the user's workspace list from the database to catch any changes.


    WITHIN the sidebar:
        DISPLAY the app logo and a welcome message.
        DISPLAY a dropdown/selectbox to choose the active workspace.
        ON CHANGE, update session state and reset relevant data (like the sales cart).
        DISPLAY navigation buttons for each page (Dashboard, Inventory, Sales, etc.).
        ON CLICK, update the 'current_page' in session state.
        DISPLAY a Logout button.
```

```
ON CLICK, clear the entire session state and redirect to the login page.
END SIDEBAR


GET the 'current_page' from session state.
CHECK if a workspace is required for the current page.
IF required and no workspace is selected, DISPLAY an error.

CASE of current_page:
WHEN "Dashboard": CALL show_dashboard_page.
WHEN "Inventory": CALL show_inventory_page.
WHEN "Sales": CALL show_sales_page.
WHEN "Reports": CALL show_reports_page.
WHEN "AI Analyst": CALL show_performance_report_page.
WHEN "Workspace": CALL show_workspace_management_page.
WHEN "Chat": CALL show_workspace_chat_page.
WHEN "Accept Invite": CALL show_accept_invite_page.
OTHERWISE: Default to Dashboard.
END CASE
END PROCEDURE

PROCEDURE show_login_page
DISPLAY form for email and password.
WHEN "Sign In" is clicked:
VALIDATE inputs.
FIND user in database. CHECK password.
IF credentials are valid:
GENERATE and SEND a 2-factor authentication code via email.
REDIRECT to 2FA verification page.
ELSE:
IMPLEMENT login attempt lockout.
DISPLAY "Invalid credentials" error.
END IF
DISPLAY links for "Forgot Password?" and "Sign Up".
END PROCEDURE

PROCEDURE show_dashboard_page
FETCH and DISPLAY sales summary metrics (Today, Week, Year).
FETCH and DISPLAY stock overview metrics (Total Units, Low Stock, Out of Stock).
FETCH and DISPLAY a list of the Top 5 Best Sellers.
FETCH data and DISPLAY pie charts for "Top Products by Quantity Sold" and "Inventory Status".
END PROCEDURE

PROCEDURE show_inventory_page
DISPLAY search and filter controls.
DISPLAY a button to "Add New Item", which shows/hides an input form.
FETCH and DISPLAY a list of all inventory items based on filters.
FOR EACH item:
DISPLAY item image, name, price, and stock status.
PROVIDE buttons for "Edit/Restock", "Deactivate", "Predict Sales", and "View Image".
IF an action button is clicked, display the relevant form/modal (e.g., the edit form with pre-filled data).
END FOR
END PROCEDURE

PROCEDURE show_sales_page
CREATE a two-column layout.
IN the left column:
DISPLAY a dropdown of available products.
WHEN a product is selected, show its details and a form to add quantity and discount.
WHEN "Add to Order" is clicked, add the item to the session state 'cart'.
IN the right column:
DISPLAY the current order (the 'cart').
FOR EACH item in the cart, show details and a "Remove" button.
CALCULATE and DISPLAY the total order price.
PROVIDE "Clear Order" and "Finalise Sale" buttons.
WHEN "Finalise Sale" is clicked, CALL record_new_sale function, clear the cart, and show a success message.
END layout
END PROCEDURE

ON SCRIPT RUN:
CREATE image directories if they do not exist.
CALL start_database() to ensure all tables are created.
CALL start_application() to begin the web app loop.
END ON

END
```

Flowchart

https://lucid.app/lucidchart/77025335-5303-4b34-a1da-236fae5f7bda/edit?invitationId=inv_6f8926de-17f9-47a0-b20f-0669dbc4d61c

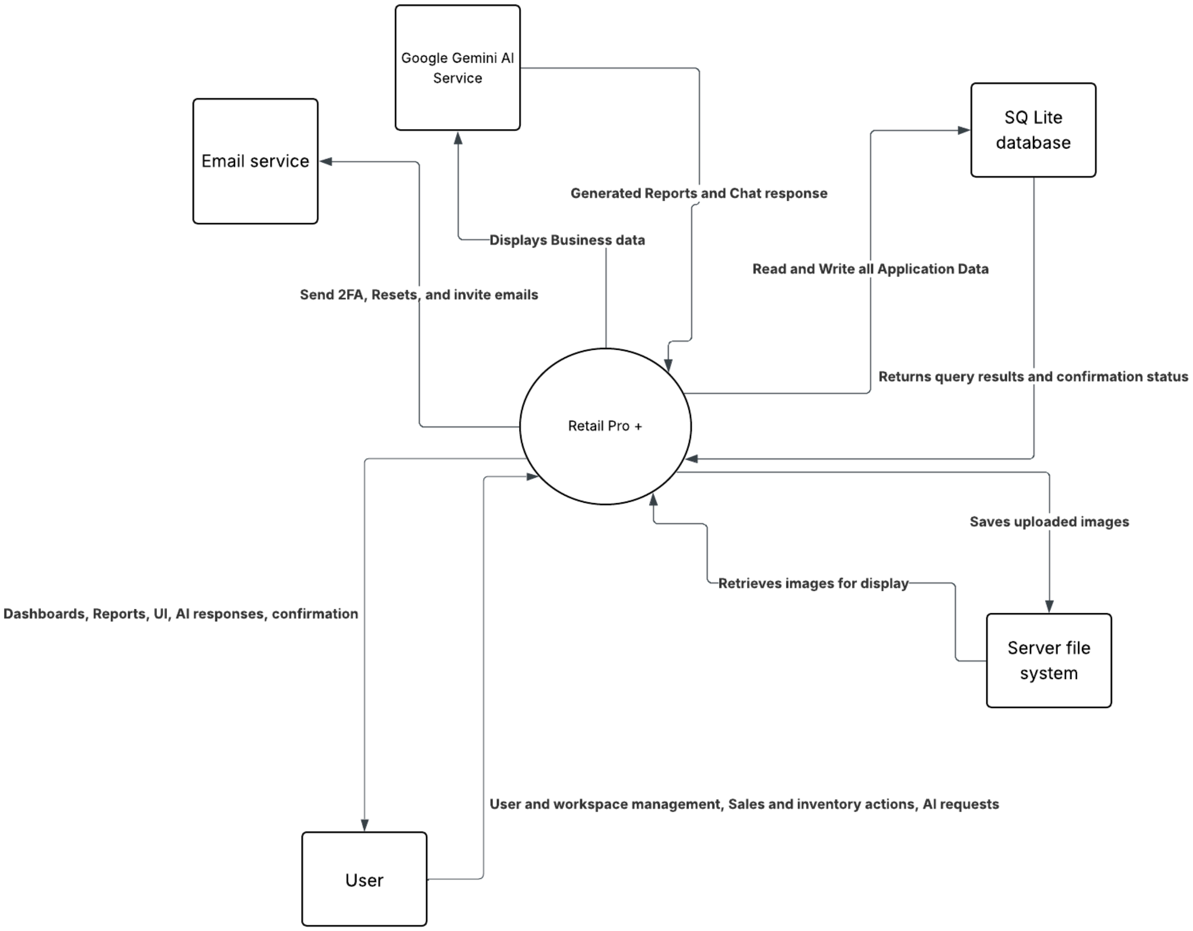
Outline: The flowchart provides a visual representation of the application's sequence and logic, detailing the primary user journey from the initial login check to navigating between different pages like the Dashboard or Sales module.


Start
Initialise Database
display login page
User signs in with correct password
User creates account
Email a verification code
User enters right verification code
User retries and its correct
Load Workspaces
Display Dashboard Page
Display sales activity, stock overview and analytics
User clicks on one of the pages
Is it inventory page
Display inventory and functional buttons
User clicks add button and enters information of item
Adds item to database
User clicks on one of the functional buttons on inventory page
Perform function
Is it sales page
Display sales page and functional buttons
User adds item to cart and selects quantity, price, discount
User finalises sale
Record sale in database
User adds more items and finalises?
Is it reports page?
Display reports page
User selects filters for report
Generate report based on filters and stats in database
Display report
User adds item to cart and selects quantity, price, discount
Is it analyst page?
Displays analyst page
AI bot analyses statistics in database
User clicks "chat" page
User sends message
Message is sent over to AI bot
Displays bot's answer to question
User continues conversation?
AI bot keeps responding until user no longer asks
User clicks generate performance report
A specific query is sent over to the Gemini AI model using an API
Gemini answers and displays the report to user
Display performance report
Is it workspace page?
Display workspace page with the current users of workspace
User enters email of another user to invite them
Send email to them with link
User removes member
Removes user from database
Is it logout
Log user out and return to login page
End
Yes
No

Context Diagram

https://lucid.app/lucidchart/69e60f23-b45f-40a8-90b1-af41967fe032/edit?invitationId=inv_c7e900c3-4a13-47ef-b615-7f7d0c50b2fb

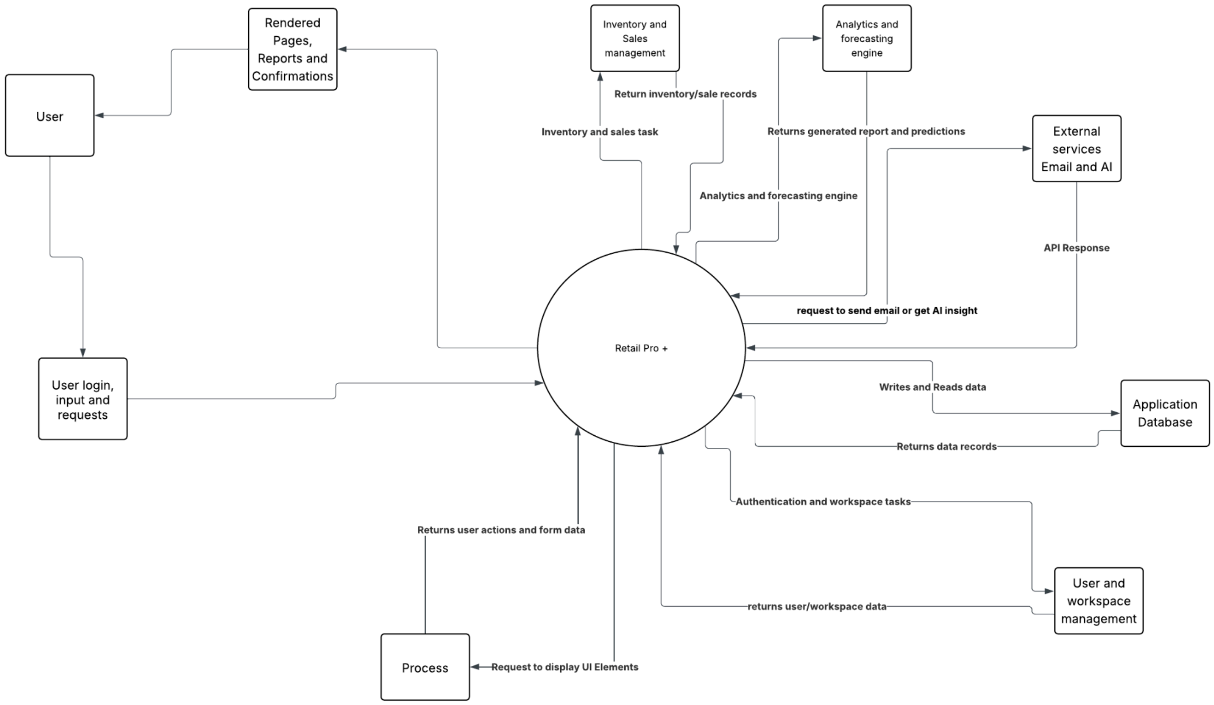
Outline: The Context Diagram presents a high-level overview of the entire system, illustrating how Retail Pro+ interfaces with external entities such as the User, the Google Gemini AI Service, an Email Service, and its database. It defines the system's boundaries by showing the primary data flows between the application and the outside world.


Google Gemini AI Service
Email service
SQ Lite database
Generated Reports and Chat response
Displays Business data
Read and Write all Application Data
Send 2FA, Resets, and invite emails
Returns query results and confirmation status
Retail Pro +
Saves uploaded images
Retrieves images for display
Dashboards, Reports, UI, AI responses, confirmation
Server file system
User and workspace management, Sales and inventory actions, AI requests
User

Data Flow Diagram

https://lucid.app/lucidchart/c16edb90-91d9-40a4-a9b5-f1fb4f7a9c56/edit?invitationId=inv_40e23f1d-8b7b-4280-af8f-6d8eaf57fddf

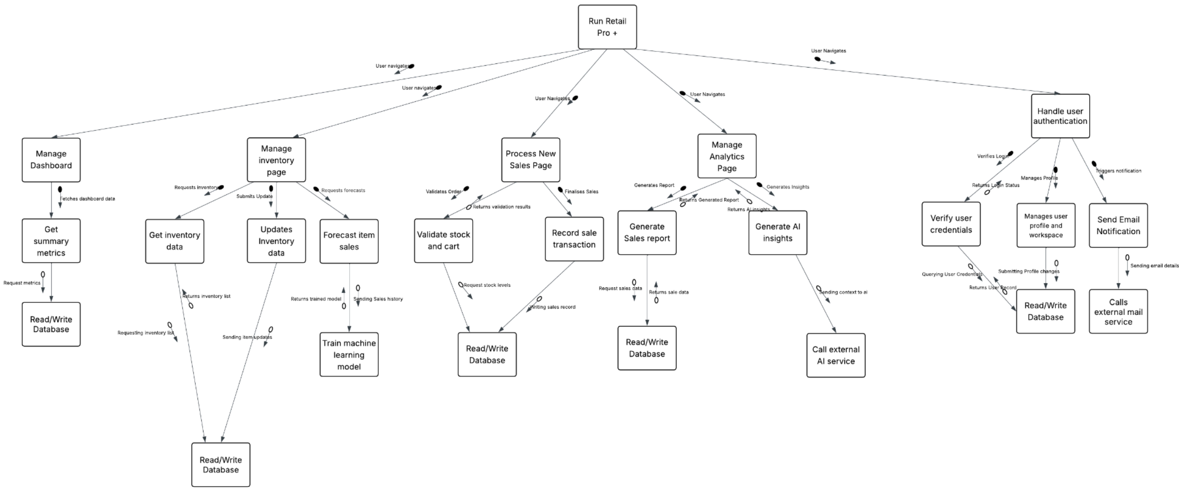
Outline: The Level 1 Data Flow Diagram breaks down the system into its main internal processes, such as inventory management, the analytics and forecasting engine, and user authentication. It visualises how data is transformed as it moves between these functions, from raw user input to generated reports and predictions.


Rendered Pages, Reports and Confirmations
User
Inventory and Sales management
Analytics and forecasting engine
Return inventory/sale records
Inventory and sales task
Returns generated report and predictions
External services Email and AI
Analytics and forecasting engine
API Response
request to send email or get AI insight
Retail Pro +
User login, input and requests
Writes and Reads data
Application Database
Returns data records
Authentication and workspace tasks
Returns user actions and form data
User and workspace management
returns user/workspace data
Process
Request to display UI Elements

Structure Chart

https://lucid.app/lucidchart/6d1afb73-1e99-439b-8d5c-220ba2454994/edit?invitationId=inv_7b22eb22-9f9a-4e09-9d5a-ead5b63348b4

Outline: The Structure Chart illustrates the application's top-down, modular design, showing how the main program is broken down into major functions like 'Manage Inventory page' and 'Handle user authentication'. This hierarchy visualises the control flow, demonstrating how high-level modules call upon lower-level sub-functions to perform specific tasks.



Data Dictionary

Global Variables

These variables are defined in the global scope and are accessible throughout the application.

Variable	**Data Type**	**Description**	**Scope**
DATABASE_FILE	String	The filename for the SQLite database.	Global
INVENTORY_IMAGE_DIRECTORY	String	The name of the directory where uploaded inventory images are stored.	Global
SENDER_EMAIL	String	The email address used by the application to send emails (invites, 2FA codes).	Global
SENDER_APP_PASSWORD	String	The application-specific password for the SENDER_EMAIL account.	Global
SMTP_SERVER	String	The SMTP server address for sending emails (e.g.,	Global

		"smtp.gmail.com").	
SMTP_PORT	Integer	The port number for the SMTP server	Global
DATABASE_URL	String	The fully formatted connection string for the SQLAlchemy database engine.	Global
engine	SQLAlchemy Engine	The core SQLAlchemy interface to the database.	Global
SessionLocal	SQLAlchemy sessionmaker	A factory for creating new database session objects.	Global
Base	DeclarativeMeta	The base class for all SQLAlchemy ORM models in the application.	Global

Database

These classes define the tables and their relationships in the application's database.

Table/Class	**Column/Attribute**	**Data Type**	**Description**
User		SQLAlchemy Model	Represents a registered user in the system.
	id	Integer	Primary key for the user
	email	String	The user's email address. Must be unique.
	password_hash	LargeBinary	The user's securely hashed password using bcrypt.
	name	string	The full name of the

			user.
Workspace		SQLAlchemy Model	Represents a collaborative workspace containing inventory, sales, etc.
	Id	Integer	Primary key for the workspace.
	Name	String	The name of the workspace
	Owner_user_id	Integer	Foreign key referencing the id of the user who owns the workspace.
	created_at	String	ISO 8601 formatted timestamp of when the workspace was created.
WorkspaceMessage		SQLAlchemy Model	Represents a single chat message within a workspace.
	id	Integer	Primary key for the message.
	Workspace_id	Integer	Foreign key referencing the id of the workspace this message belongs to.
	user_id	Integer	Foreign key referencing the id of the user who sent the message
	content	TEXT	The text content of the chat message
	timestamp	String	ISO 8601 formatted timestamp of when

			the message was sent.
WorkspaceMember		SQLAlchemy Model	Links users to workspaces, defining their membership and role.
	Id	Integer	Primary key for the membership record
	workspace_id	Integer	Foreign key referencing the id of the associated workspace.
	user_id	Integer	Foreign key referencing the id of the associated user.
	role	String	The role of the user in the workspace (e.g., ‘owner’, ‘member’)
	inivited_by_user_id	Integer	Foreign key referencing the id of the user who sent the invitation.
	invite_email	String	The email address the invitation was sent to. Used for non-registered users.
	Invite_token	String	A unique token for accepting an invitation.
	status	String	The status of the membership (e.g., 'pending', 'accepted').
	joined_at	string	ISO 8601 formatted timestamp of when the user joined.

Inventory		SQLAlchemy Model	Represent a single product or item in a workplace’s inventory
	id	Integer	Primary key for the inventory item.
	workspace_id	Integer	Foreign key referencing the id of the workspace this item belongs to.
	name	String	The name of the product. Uniquely constrained per workspace for active items.
	retail_price	REAL	The selling price of one unit of the item.
	stock_level	Integer	The current quantity of the item in stock.
	image_path	String	The local file path to the item's saved image.
	is_active	Boolean	A flag indicating if the product is active (True) or deactivated (False).
sale		SQLAlchemy Mode	Represents a single sales transaction
	Id	Integer	Primary key for the sale.
	workspace_id	Integer	Foreign key referencing the id of the workspace where the sale occured.
	recorded_by_user_id	Integer	Foreign key

			referencing the id of the user who recorded the sale
	sale_datetime	String	ISO 8601 formatted timestamp of when the sale was recorded.
	total_amount	REAL	The total value of the sale after discounts.
SaleItem		SQLAlchemy Model	An association table representing one line item within a Sale.
	Id	Integer	Primary key for the sale item line.
	Sale_id	Integer	Foreign key referencing the id of the parent Sale record.
	Inventory_item_id	Integer	Foreign key referencing the id of the Inventory item that was sold.
	Quantity_sold	Integer	The number of units of the item sold in this transaction.
	Price_per_unit_at_sale	REAL	The price of the item at the time of sale.
	Discount_percentage	REAL	The discount percentage applied to this line item.
	subtotal	REAL	The final calculated price

Application Function

These functions encapsulate the business logic of the application

Function	Return Data Type	Description	Scope
row_to_dict()	Dictionary or None	Converts a SQLAlchemy model object into a dictionary.	Global
create_database_connection()	SQLAlchemy Session or None	Creates and returns a new database session object for queries.	Global
start_database()	None	Initialises the database and creates all tables defined by the models if they don't exist.	Global
post_workspace_message()	Boolean	Saves a new chat message to the database for a given workspace.	Global
get_workspace_messages()	List of Dictionaries	Retrieves recent chat messages for a workspace	Global
get_sales_by_item()	List of Dictionaries	Gathers sales performance data for each product over a specified period.	Global
clear_workspace_chat()	Boolean	Deletes all chat messages for a specified workspace	Global
rename_workspace()	Boolean	Renames a workspace	Global
refresh_user_workspace_state()	None	Updates the user's workspace list in st.session_state and handles changes.	Global
create_new_workspa	Integer	Creates a new	Global

ce()		workspace and assigns ownership.	
add_workspace_team_member()	Boolean	Adds a new member or a pending invitation to a workspace.	Global
remove_workspace_member()	Boolean	Removes a member from a workspace	Global
cancel_pending_invite()	Boolean	Cancels a pending invitation to a workspace	Global
get_user_workspaces_from_db()	List of Dictionaries	Fetches all workspaces a given user is an accepted member of.	Global
find_workspace_in_db()	Dictionary	Retrieves the details of a single workspace by its ID.	Global
get_workspace_member_details()	List of Dictionaries	Fetches details of all members (accepted and pending) for a workspace.	Global
process_workspace_invitation_token()	Tuple	Validates an invitation token and adds the user to the workspace.	Global
is_user_a_member_of_workspace()	Boolean	Checks if a user is an accepted member of a specific workspace.	Global
get_workspace_owner_user_id()	Integer	Retrieves the owner's user ID for a workspace	Global
find_user_by_email_in_db()	Dictionary	Finds a user by their email address	Global
find_user_by_id_in_d	Dictionary	Finds a user by their	Global

b()		unique user ID.	
register_new_user()	Tuple	Registers a new user, hashes their password, and creates a default workspace.	Global
update_user_password_in_db()	Boolean	Updates a user's password in the database.	Global
add_product()	Boolean	Adds a new product to the inventory for a specific workspace.	Global
get_products()	List of Dictionaries	Retrieves a list of inventory products based on filters.	Global
get_product_by_id()	Dictionary	Fetches a single product's details by its ID.	Global
update_product()	Boolean	Updates the details of an existing inventory product.	Global
deactivate_product()	Boolean	Sets an inventory item’s flag	Global
record_new_sale()	Boolean	Records a new sale	Global
get_sales_summary_data()	Dictionary	Calculates total sales	Global
get_total_units_sold()	Integer	Calculates the total number of all units sold in a workspace.	Global
get_chart_sales_data()	Pandas DataFrame	Collects sales data and prepares it for charts	Global
get_best_sellers()	List of Dictionaries	Retrieves the top-selling products based on quantity sold.	Global

get_product_sales_history()	Pandas DataFrame	Fetches the complete sales history for a single product.	Global
prepare_forecasting_data()	Pandas DataFrame	Prepares and cleans sales history data for the Prophet Model	Global
train_sales_forecasting_model()	Prophet Model	Trains a Prophet forecasting model on the prepared sales data.	Global
generate_sales_forecast()	Dictionary	Uses a trained model to predict future sales quantities.	Global
hash_user_password()	bytes	Hashes a plain-text password using bcrypt.	Global
check_user_password()	Boolean	Verifies a plain-text password against a stored hash.	Global
password_meet_req()	Tuple	Checks if a password meets the defined security requirements.	Global
send_application_email()	Boolean	Sends an email using the configured SMTP settings.	Global
save_uploaded_inventory_image()	String	Saves an uploaded image file	Global
generate_ai_performance_report()	String	Generates a business performance summary	Global
show_login_page()	None	Displays login page	Global
show_dashboard_page()	None	Displays dashboard page	Global
show_inventory_page()	None	Displays inventory page	Global

show_sales_page()	None	Displays sales page	Global
show_reports_page()	None	Displays reports page	Global
show_workspace_management_page()	None	Displays workspace management page	Global
show_accept_invite_page()	None	Displays accepted invite page	Global
show_workspace_chat_page()	None	Displays workspace chat page	Global
show_performance_report_page()	None	Displays performance reports page	Global
start_application()	None	Start the website	Global

Streamlit Session State

These keys are used on the st.session_state object to maintain the application's state across user interactions and page reruns.

Variable	**Data Type**	**Description**	**Scope**
logged_in_user	Dictionary	Stores the authenticated user's data	Session
current_page	String	Controls which main page is currently displayed to the user	Session
auth_flow_page	String	Manages the sub-state of the authentication process	Session
user_workspaces	List of Dictionaries	A cache of the workspaces the current user is a member of.	Session
current_workspace_id	Integer	The ID of the currently active	Session

		workspace selected by the user.	
current_workspace_name	String	The name of the currently active workspace.	Session
cart	List of Dictionaries	A temporary list of all items added to the current order on the "Sales" page.	Session
active_action	String	The action a user wants to perform on an item	Session
active_item_id	Integer	The ID of the specific item being edited or viewed.	Session
login_attempts	Dictionary	Counts how many times a user has tried to log in with the wrong password.	Session
auth_user_email	String	Temporarily holds the user's email while verifying their 2-factor code.	Session
auth_expected_code	String	The secret 2-factor code sent to the user that they must enter to log in.	Session
reset_email	String	Temporarily holds a user's email during the forgot password process	Session
reset_expected_code	String	The secret code sent to a user's email to let them reset their password.	Session
pending_invite_token_after_login	String	Stores a workspace invitation link if a user	Session

		clicks it before they are logged in.	
generated_report	String	Stores the AI-generated business report text after it's been created.	Session
chat_session	GenerativeModel Chat	The active chat conversation with the AI Business Assistant.	Session
messages	List of Dictionaries	The stored message history for the current AI chat.	Session

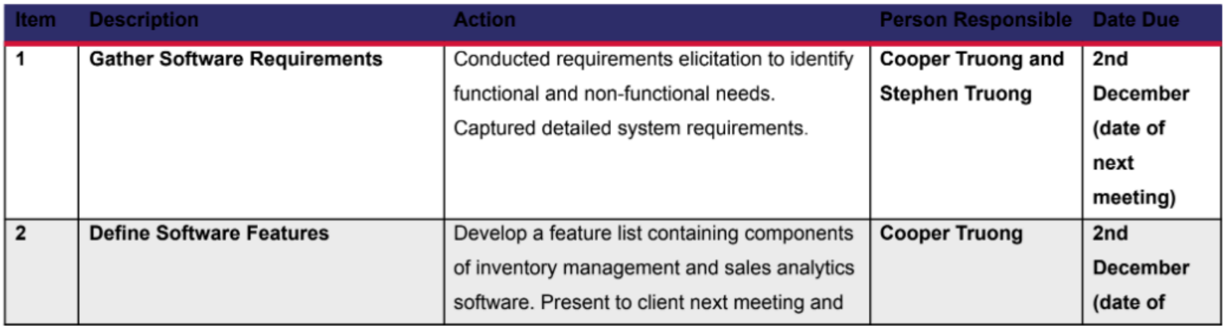
2.4 Client Communication and Feedback

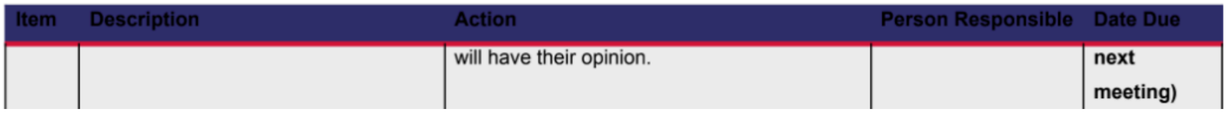
To ensure the final software solution is successful and aligns with the client's vision, I have maintained a log of all formal meetings and correspondence.

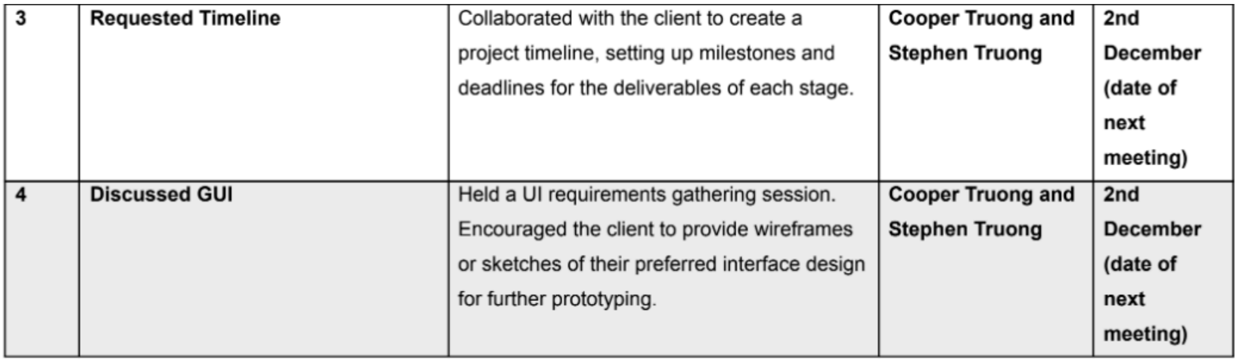
Communication Log Entry #1

- Date: 13th November 2024
- Topic: Project scoping and requirements elicitation

The first official meeting was held with my client, Stephen Truong, to establish the foundation for the project. As part of this meeting, I did a requirements elicitation to capture his main functional and non-functional needs towards the new software. We explored together what was wrong with his current system, and based on that, I designed and offered a preliminary feature list concentrating on inventory management and sales analytics. We also collaborated on creating a high-level project timeline, setting up initial milestones for the project deliverables. Finally, we conducted a UI requirements gathering session, during which I asked my client to share his ideas on the interface design. He indicated the need for a clean, simple layout with clear, easy to read text, given the varied technical skill levels of his staff.





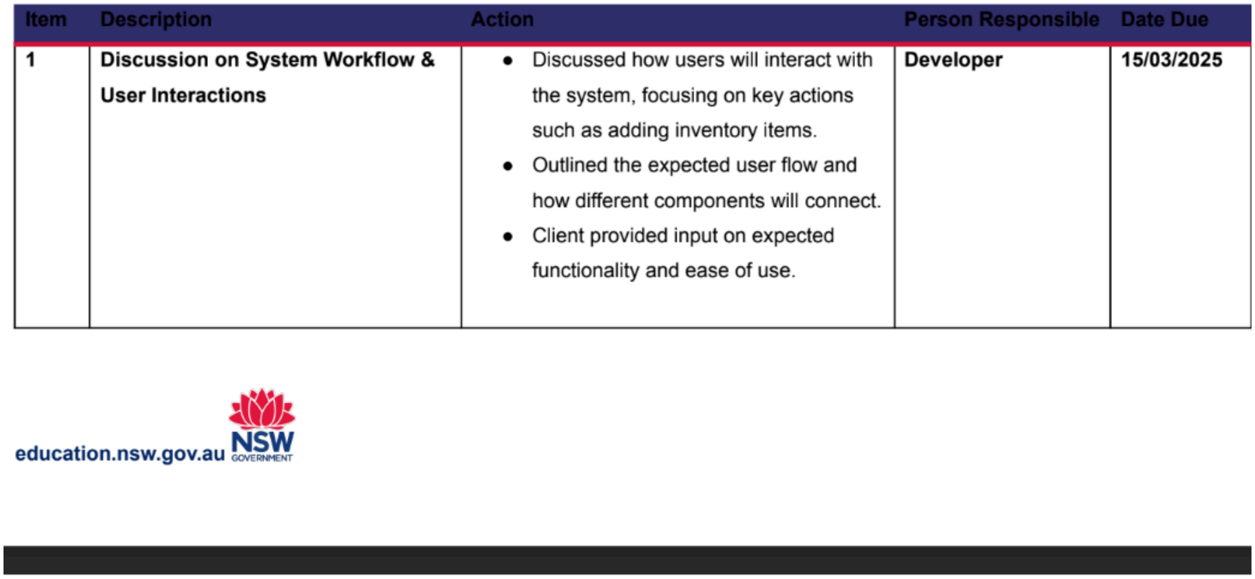


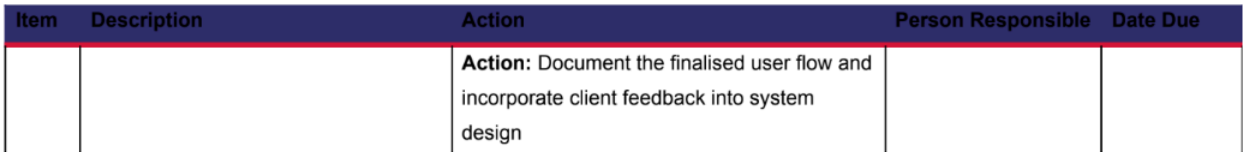
Item	Description	Action	Person Responsible	Date Due
1	**Gather Software Requirements**	Conducted requirements elicitation to identify functional and non-functional needs. Captured detailed system requirements.	**Cooper Truong and Stephen Truong**	**2nd December (date of next meeting)**
2	**Define Software Features**	Develop a feature list containing components of inventory management and sales analytics software. Present to client next meeting and will have their opinion.	**Cooper Truong**	**2nd December (date of next meeting)**
3	**Requested Timeline**	Collaborated with the client to create a project timeline, setting up milestones and deadlines for the deliverables of each stage.	**Cooper Truong and Stephen Truong**	**2nd December (date of next meeting)**
4	**Discussed GUI**	Held a UI requirements gathering session. Encouraged the client to provide wireframes or sketches of their preferred interface design for further prototyping.	**Cooper Truong and Stephen Truong**	**2nd December (date of next meeting)**

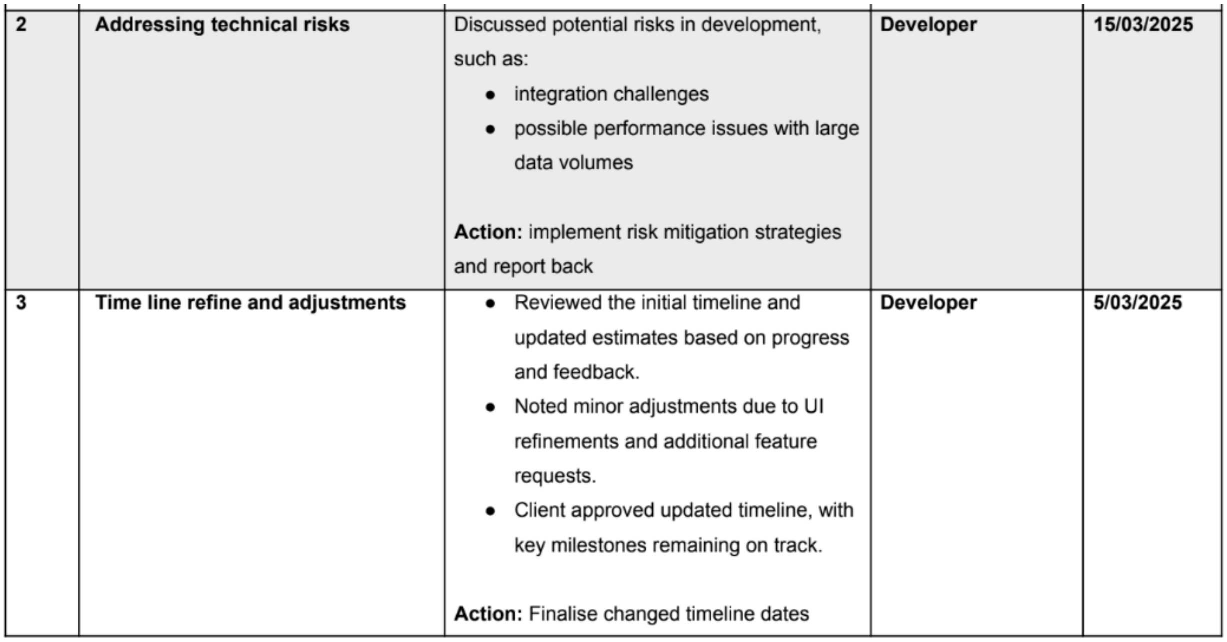
Communication Log Entry #2

- Date: 1st of March 2025
- Topic: System Workflow, Risk Assessment, and Timeline Refinement

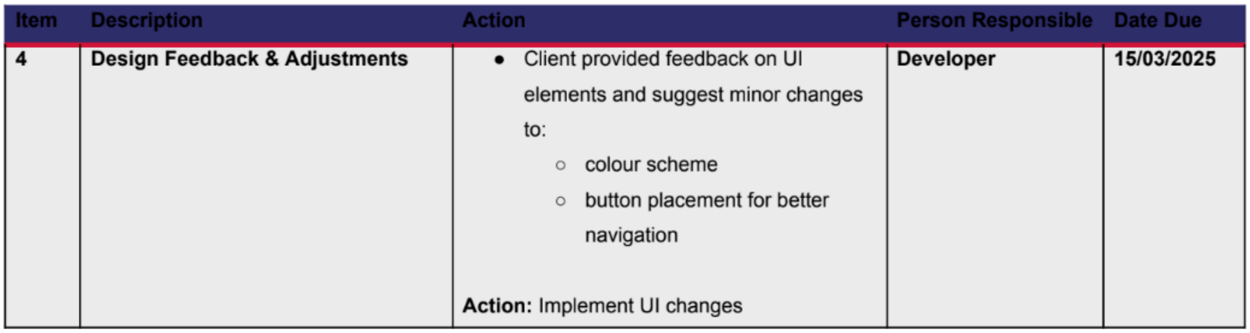
In our second meeting, we focused on refining the project plan and system design. Our discussion started with describing the application's system workflows, user interactions, adding new inventory items, and processing a sale. My client provided valuable input on the expected functionality, focusing on ensuring the system would be easy to use during busy periods in the shop. We also reviewed integration with other systems, potential performance bottlenecks over time with incremental sales and inventory data, and any other possible technical concerns. With this in mind, we made adjustments to the project timeline. I applied his suggestions from our workflow discussion, and the client agreed to the revised timeline as long as major timeline milestones were preserved. Outcomes for this meeting included finalised user flow documentation as well as moving forward with documented strategies for identified risks.







Item	Description	Action	Person Responsible	Date Due
1	**Discussion on System Workflow & User Interactions**	• Discussed how users will interact with the system, focusing on key actions such as adding inventory items. • Outlined the expected user flow and how different components will connect. • Client provided input on expected functionality and ease of use. **Action:** Document the finalised user flow and incorporate client feedback into system design	**Developer**	**15/03/2025**
2	**Addressing technical risks**	Discussed potential risks in development, such as: • integration challenges • possible performance issues with large data volumes **Action:** implement risk mitigation strategies and report back	**Developer**	**15/03/2025**
3	**Time line refine and adjustments**	• Reviewed the initial timeline and updated estimates based on progress and feedback. • Noted minor adjustments due to UI refinements and additional feature requests. • Client approved updated timeline, with key milestones remaining on track. **Action:** Finalise changed timeline dates	**Developer**	**5/03/2025**



Item	Description	Action	Person Responsible	Date Due
4	**Design Feedback & Adjustments**	• Client provided feedback on UI elements and suggest minor changes to: ○ colour scheme ○ button placement for better navigation **Action:** Implement UI changes	**Developer**	**15/03/2025**

3.0 Producing and Implementing

3.1 Development sprints/iterations

The following is a development log of the work completed during each sprint, adhering to my initial Agile plan. While the plan in Section 2.2 outlined my intentions, this section details the actual implementation process, challenges faced, and the final outcomes.

Sprint 1 - User Authentication and Workspaces
The first sprint was dedicated to building the application's core architecture. Then, I worked on the user authentication system which consisted of a login page that uses bcrypt hashing for secure credential storage and a registration page with password requirements tailored to ensure stronger security measures. I also implemented the fundamental logic for creating a user's primary workspace upon registration. By the end of this sprint, crucial portions of the application were ready including account creation and login functionalities based on robust security mechanisms.

Sprint 2 - Inventory and Sales
This sprint focused on implementing the primary business functions of the application. I designed the inventory management page where users can add new products, update old ones, and view all of the items in a scrollable table. One of the important features implemented is “soft-delete,” which prevents permanent deletion of data by marking an item as inactive instead of removing it entirely. Next I developed the sales processing module which allows for adding items to a cart and completing a sale. To ensure data integrity, the function that finalises a sale was written to use a database transaction, guaranteeing that the creation of the sales record and the decrementation of the stock level occur together as a single, reliable operation. During this sprint, I discovered and resolved bugs such as one caused by double-clicking on an inventory item’s image, resulting in errors due to how the image data was accessed within the function.

Sprint 3 - Analytics and Reporting
With the core data being captured, the goal of this sprint was to build the tools for the client to gain insights from that data. I created the main dashboard page which displays critical sales figures like the current day, week, and year sales as per the initial storyboards. I then built the dedicated reports page, which allows the client to generate performance reports that highlight the best-selling products and overall sales trends. For data visualisation, I used Plotly library to build interactive donut charts that clearly show inventory status and top products sales enabling clients to get sharp insights at a glance.

Sprint 4 - Advanced Features and Security Hardening
This sprint was focused on implementing the more complex, innovative features and strengthening the application's security. I completed the sales forecasting component and integrated Google Gemini AI services for automated business report generation alongside an interactive chat interface. In addition, I used 2FA email code verification for hardening security on user accounts. During this phase, I addressed an issue where the sales prediction model's results would not change even after a long period without sales. I fixed this by modifying the data preparation function to generate a complete date range, forcing the model to learn from periods of inactivity.

Sprint 5 - Finalisation, Testing and Handover
The final sprint was dedicated to polishing the application and preparing it for handover. No new features were added; instead, the focus shifted entirely to rigorous testing and stabilisation. I conducted two distinct testing cycles: Normal Usage Testing, where I performed everyday tasks to identify workflow issues, and Unit Testing, using the pytest framework to test individual functions. This strategy was very useful and revealed several bugs present in the system. These bugs will be detailed along with their fixes in the “Testing and Evaluation” section of this portfolio. The rest of the sprint was allocated to resolving these bugs for better application stability as well as completing the documentation for submission.

3.2 Innovative Solution Proposal

This section outlines a proposal for an additional innovative feature that could be integrated into Retail Pro+ in the future. This proposal includes a description of the solution and a user interface (UI) prototype design.

The Problem/Opportunity
Setting the correct price for products is one of the biggest challenges for a retail business. Under all conditions, a static price may not be optimal. During peak season, a price that is profitable during peak season might be too high during a slow period, leading to missed sales.

The client currently relies on experience and estimates to set prices, lacking a tool to make data driven pricing decisions to maximise profit and efficiently manage stock turnover.

The Proposed Solution

I recommend the addition of a “AI-Powered Dynamic Pricing Suggestions” module which would utilise the integration of Google Gemini AI in a more evaluative manner. Rather than changing prices at will, this functionality would suggest potential pricing improvements for the clients to modify as they see fit.

The system would periodically analyse a combination of factors for each product:

- *Sales Velocity:* how quickly an item is selling compared to its historical average
- *Current Stock Level:* a high volume of stock for a slow moving item might trigger a discount suggestion
- *Time-Based Trends:* Using data from the sales forecaster, the AI could identify if a product sells better on weekends and suggest a small strategic discount on a weekday to encourage sales
- *Product Age:* For items that have been in inventory for a long time, the AI could suggest a clearance price to free up space

Based on this analysis, the system would generate a clear, actionable suggestion, such as: "Sales for 'Product X' are low this week, and stock is high. Consider a 15% discount to boost sales."

Benefits to Client

- *Revenue Optimisation:* Will aid the client to find the best optimal price to maximise both profit margins and sale volumes
- *Proactive Stock Management:* Provides a strategic tool to clear out old or slow-moving inventory before it becomes a liability
- *Data Driven Strategy:* Eliminates guesswork when it comes to setting a price by replacing it with practical suggestions based on the company’s data trends.
- *Competitive Advantage:* Gives an opportunity to small businesses to take advantage of advanced pricing policies, which are usually available only to big companies.

User Interface (UI) Prototype Design

To accommodate this feature, I would design a new page accessible from the main sidebar.

- A new “Pricing Advisor” page would be added to the application’s main navigation menu
- Each card would include the product Name, current price, AI’s suggested new price, and about two or three brief reasons for the suggestions, like “High stock, low recent sales” or similar.

- On every card, there would be a green button saying “Accept & Update Price” and a grey “Dismiss Suggestion” button.
- If the client clicks "Accept," the system would show a confirmation pop-up, and upon approval, would automatically update the item's retail price in the inventory database. This way we ensure that client will always have full control over their pricing decisions regardless of how automated everything works behind the scenes.

4.0 Testing and Evaluation

4.1 Testing Cycle

Testing Cycle 1: Normal Usage Testing

Objective: To simulate a typical user's journey through the application, identifying usability issues, logical flaws, and unexpected behaviour in a standard workflow.

	What is the Bug	**How I found this**	**How I fixed it**
Bug #1	Discount applies incorrectly to subsequent items in the cart. When adding multiple different products to a sales order, applying a discount to one item causes that same percentage discount to be pre-filled and applied to the next item added, even if the user doesn't want a discount on the second item. The user has to manually reset the discount field to 0.	Navigated to the "Sales" page. Selected the first product and added it to the order with a 10% discount. Selected a second, different product. The "Discount (%)" field was already filled with "10.0". Added the second product to the order without changing the discount field, and the 10% discount was incorrectly applied.	I realised the issue was that the state of the discount input field wasn't resetting after I added an item to the cart. Even though I set clear_on_submit=True on my form, the sales_discount_input value persisted between additions. To fix this, I made sure to call st.rerun() immediately after a product is successfully added to the st.session_state.cart. This forces a complete reload of the page elements, which reliably clears the form state and

			resets the discount input back to its default value for the next item.
Bug #2	Renaming a workspace to a name another user already has causes an unhandled error. The application should allow two different users to have workspaces with the same name (e.g., "My Store"), but a bug was causing a generic SQLAlchemyError when this was attempted.	While testing multi-user functionality, I logged in as User A and saw their workspace was named "Cooper's Workspace". I then logged in as User B and navigated to the "Workspace" page to rename their own workspace. When I tried to rename it to "Cooper's Workspace", the application hung and showed a generic error instead of succeeding.	I investigated the rename_workspace function and found a faulty check I had initially placed. I was incorrectly trying to prevent any two workspaces from having the same name across the entire application. My code already correctly checks if workspace.owner_user_id matches the current user_id, which is the only check that matters. I removed the unnecessary global name-uniqueness logic, which fixed the error.
Bug #3	Deactivated products can still be sold. Products that are marked as "deactivated" from the Inventory page still appear in the product dropdown on the Sales page, allowing them to be added to a cart and sold.	I was testing the inventory lifecycle. I went to the "Inventory" page and used the "Deactivate" button on a product that had stock. Afterwards, I went to the "Sales" page to process a new order and was surprised to see that the product I had just deactivated was still available for selection. I was able to add it to the cart	I traced this to my show_sales_page function. I realised that when I was fetching items for the sales dropdown, my call to get_products wasn't explicitly filtering out inactive items. To fix this, I adjusted the query inside my get_products function. I made sure the line query = query.filter(Inventory.is_active == True) is

		and finalise the sale.	always included unless I specifically request inactive products, ensuring only active items appear on the sales page.
Bug #4	Inventory search is case-sensitive. The search bar on the Inventory page was case-sensitive.	I added a new product named 'Black Boots'. After adding it, I tried to find it again using the search bar on the Inventory page. When I typed 'black boots' in all lowercase, the item disappeared from the list. It only reappeared if I typed it exactly as 'Black Boots' with the correct capitalisation.	I looked into my get_products function and found the source of the problem. My SQLAlchemy query was using the .like() method for filtering, which is case-sensitive in SQLite. I fixed this by changing the method to .ilike(), which is SQLAlchemy's operator for a case-insensitive LIKE operation. The line in my code now reads Inventory.name.ilike(f "%{search_term}%"), so the search works regardless of capitalisation.

Testing Cycle 2: Unit Testing

Objective: To test individual functions and components of the application under stress and with invalid inputs. This cycle focuses on security, data integrity, and error handling.

Bug Found #1: record_new_sale - Sale Allowed with Insufficient Stock

- Function under testing: record_new_sale(workspace_id, recorded_by_user_id, cart_items, total_sale_amount)

- Bug Description: An earlier version of the function did not check for sufficient stock before creating Sale and SaleItem records. This could lead to a situation where a sale is recorded for more items than are available, causing the inventory level in the database to become negative.

- How This Was Found: I wrote a unit test that created a product with a stock level of 5. The test then called the record_new_sale function with a cart attempting to sell 10 units of that product. When I inspected the database after the test ran, the product's stock level was -5, and a sale had been recorded for an impossible number of items.

- How I Fixed It: I came to the conclusion that it was imperative to validate every item against the database prior to generating the main sale record. In doing this, I implemented what is now known as pre-check loops at the start of my record_new_sale function. As the code now shows, it iterates through the cart_items, queries the database for the current stock level of each item, and raises a ValueError if stock_info.stock_level < item_in_cart['quantity']. This aborts the entire transaction using a session.rollback() and prevents the stock from ever going into the negative.

Bug Found #2: get_chart_sales_data - Incorrect Report Aggregation Across Years

- Function Under Test: get_chart_sales_data(workspace_id, period="Week")

- Bug Description: The weekly sales report would become inaccurate in cases where the week stretched across two different months or years (e.g., the last week of December when January has just started). The performance of the function was suboptimal due to incorrect data grouping.

- How I Found This: My unit test for the weekly report was failing at the beginning of a new year. I created test sales data spanning the last week of December and the first week of January. When my test ran on January 1st, the report was incorrectly including sales from the previous year in the current week's total, leading to inflated numbers.

- How I FIxed This: When I looked at my get_chart_sales_data function, I found the problem was in how I was comparing dates. I was initially using basic string manipulation on the ISO format timestamp, which was unreliable across year boundaries. I fixed this by properly converting the sale_datetime string from the database into a true Python datetime object using datetime.datetime.fromisoformat(). Then, I could reliably get the .date() and .isocalendar() information to correctly aggregate the sales.

Bug Found #3: process_workspace_invitation_token - Case-Sensitive Email Matching

- Function Under Test: process_workspace_invitation_token(invite_token, accepting_user_id)

- Bug Description: When users accepted an invitation, the email verification process was case-sensitive. For instance, if the user was invited as User@example.com but they joined as user@example.com, then the system would not consider them as matched.

- How I Found This: To cover edge cases of user invitations, I created a unit test. Specifically, I placed a pending invitation for ‘Test.User@email.com’ and created a new account ‘test.user@email.com’. My function call with new user's ID and invoked using the updated token did not work which indicated that I was not processing email case-insensitivity logic.

- How I Fixed It: I went into my process_workspace_invitation_token function and confirmed the email comparison was the issue. I fixed this by modifying the conditional check to convert both the invite_email from the database and the email from the accepting user's object to lowercase using .lower() before comparing them. This is now reflected in the line ... invite_details_dict['invite_email'].lower() == accepting_user_obj['email'].lower() and ensures that 'User@email.com' and 'user@email.com' are correctly matched.

Bug Found #4: remove_workspace_member - Owner Can Remove Themselves

- Function under test: remove_workspace_member(workspace_id_to_modify, member_user_id_to_remove, current_user_id_acting)

- Bug Description: There was no safeguard to prevent a workspace owner from accidentally removing themselves from their own workspace.

- How I Found This: My unit test was designed to check for invalid removal scenarios. I configured a test where the current_user_id_acting and the member_user_id_to_remove were the same, and both corresponded to the workspace owner. I expected the function to fail, but my test showed that the function proceeded with the deletion.

- How I Fixed It: I realised my remove_workspace_member function was missing a critical safety check. While I had a check to prevent other users from removing the owner, I didn't stop the owner from self-removal. I fixed this by adding the if member_user_id_to_remove == current_user_id_acting: condition.

4.2 Evaluation

Personal

Overall, I am extremely proud of the final Retail Pro+ application and consider the project a success. The primary goal was to produce a solution to aid my client's retail shop by replacing his old and ineffective inventory system. I believe the final product comprehensively achieves this by addressing the key challenges of overstocking, understocking, and inaccurate inventory records that were identified at the start of the project.

This project provided me with useful insight from a technical standpoint. I was able to implement the entire technology stack which included Python for the core logic, Streamlit web framework for the user interface, and SQLite for the database. The challenge of integrating advanced libraries like Prophet to forecast or Google’s Gemini AI for report generation proved rewarding. The development process, particularly the discovery and resolution of bugs during the testing phase, reinforced the importance of robust testing.

Following the Agile sprint plan I created was critical to managing the project's complexity. Organising the work into five separate sprints enabled me to concentrate on delivering functionally complete parts progressively. The decision to set aside an entire sprint for thorough testing and stabilisation proved exceptionally beneficial in enhancing the application’s overall quality and reliability. If I were to undertake this project again, I would develop unit tests with pytest alongside feature development instead of leaving it all to the final sprint as this might have identified some errors sooner.

Reflecting on the project's execution, one area I could have improved is time management. Although the final deadline was not missed, some parts of the work like integrating advanced AI and forecasting capabilities during Sprint 4 took longer than expected based on my GANTT chart. This added some stress in the closing weeks of the project. Were this project offered to me again, I would leave extra buffer time between sprints to manage other unplanned difficulties and intricate tasks. This experience has been a valuable lesson in the importance of creating more conservative time estimates and building flexibility into a project schedule.

Completing this project has profoundly refined my skills within planning and design through to implementation and evaluation.

Client

Upon completion of the final sprint, I presented the finished Retail Pro+ application to my client, Stephen Truong. His feedback was overwhelmingly positive, and he was extremely pleased with the final product.

His words were that the software relieved the main issues that had been a headache for him. Definitely, one of his concerns was the inventory system because he suffered from inaccurate inventory records. Now with our solution, adding, editing and viewing stock in real time is done through a simple system which makes it effortless.

An important factor was that it had to be designed for users with almost no experience in computers. My client confirmed this requirement was met as well. He said that Streamlit gave a modern look to the interface which made it easy to use and navigate . Most importantly the dashboard enabled him see all his sales figures daily, weekly, and yearly at a glance.

The client was also enthusiastic about the sales analytics features. He pointed out that his business would greatly benefit from the automation that generates reports on “best-selling products” and “sales trends” as it would significantly improve decision making for restocking and promotional offers. He found the AI-generated reports to be an innovative and surprisingly useful feature for getting a quick summary of the business's health.

Overall, the client expressed his full satisfaction with the software solution. He strongly believes that Retail Pro+ will help get rid of a lot of operational inefficiencies and enable more sophisticated decisions, which was the ultimate goal of the project.

4.3 Terms and Conditions

Welcome to Retail Pro+. By creating an account and using our software (the "Service"), you agree to these simple terms.

1. **Your Account and Conduct**

You are responsible for all activity that occurs under your account and for keeping your password secure. You agree to use the Service only for lawful business purposes and not to misuse the Service or infringe on the rights of others.

2. **Data and Ownership**

You are the sole owner of the business information data you input into Retail Pro+ like stock and sales data. We own the Service itself, including but not limited to, the software, branding and design. You may not copy, modify or distribute the software without our permission.

3. **AI Powered Features**

Our Service utilises Artificial Intelligence to generate sale forecasts and prepare business analytics formal reports. This information is presented for the purpose of providing guidance only and should not be considered a replacement for professional insight into business decisions. We are not responsible for the accuracy of content that has been produced by AI tools.

4. **Termination**

We reserve the right to suspend or terminate your access to the Service at any time, without notice, if you violate these terms.

5. **Disclaimer**

The Service is provided on an "as is" basis without any warranties. We are not liable for any direct or indirect damages, including any loss of data or profits, arising from your use of the Service.

6. **General Terms**

These terms are governed by the laws of New South Wales, Australia. We may update these terms from time to time by posting the new version within the application.

4.4 End User License Agreement

This End User License Agreement ("EULA") outlines the legal agreement between the user (either individual or a single entity) and Cooper Truong, the developer of Retail Pro+ software application including any documentation associated with it.

By installing, copying, or otherwise using the Software, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not install or use the Software.

1. **Grant of License**

The developer grants you a revocable, non-exclusive, non-transferable, limited license to install and use the Software on a device owned and controlled by you, solely for your internal business purposes within your retail establishment, in accordance with the terms of this EULA.

2. **License Restrictions**

You confirm that you shall not undertake action or engage third parties to perform actions for:

a) Commercial exploitation whether through licensing, selling, renting, leasing, assigning or distribution of the Software.
b) Modify, decompile, reverse engineer, or create derivative works of the Software.
c) Removal or alteration of trademark logos regarding copyrights along proprietary marks placed on Software.

3. **Intellectual Property**

The Software is licensed, not sold. You acknowledge that the developer, Cooper Truong, retains all right, title, and interest in and to the Software, including all copyrights, trademarks, and other intellectual property rights.

4. **Termination**

If you breach any of the terms outlined above, this EULA will terminate automatically and without notice. Upon termination of the EULA, you are no longer bound by its terms and as such stop using the software.

5. **Disclaimer of Warranty**

The software is provided to you 'as is' and 'as available', with all faults and defects without warranty of any kind. the developer expressly disclaims all warranties, whether express, implied, statutory, or otherwise.

6. **Limitation of Liability**

Thus being said the developer shall not be held liable for any special incidental or consequential damages incured such as a loss of profits, data or business disruption arising to the inability to use the software.

Logbook

This logbook provides a chronological record of the key activities and milestones.

November 2024

- 11/11/24: Project initiated. Began preliminary research into retail management systems.
- 13/11/24: [Milestone] Held initial meeting with client, Stephen Truong. Discussed project scope and identified core problems with the current system.
- 18/11/24: Started gathering and documenting functional and non-functional requirements based on client feedback.
- 25/11/24: Finalised the initial list of requirements for the software.

December 2024

- 02/12/24: Began designing system models using Lucidchart, including the Context Diagram and Data Flow Diagrams.
- 09/12/24: Developed initial UI storyboards for the main application pages (Dashboard, Inventory, Sales).
- 16/12/24: Commenced work on the initial portfolio documentation, focusing on the "Identifying and Defining" section.

January 2025

- 28/01/25: [Sprint 1 Started] Began development. Set up the Python environment, installed necessary libraries (Streamlit, SQLAlchemy), and initialised the Git repository.
- 31/01/25: Designed and created the initial SQLite database schema (init_db function).

February 2025

- 07/02/25: Implemented the user registration and login system with bcrypt password hashing.
- 14/02/25: Developed the core logic for creating and managing user workspaces.
- 21/02/25: [Sprint 1 Completed]
- 24/02/25: [Sprint 2 Started] Began work on the inventory management module.
- 28/02/25: Implemented "Add" and "Edit" functionality for inventory items.

March 2025

- 01/03/25: [Milestone] Held second meeting with client. Discussed system workflow, addressed technical risks, and refined project timeline.
- 07/03/25: Developed the sales processing page and the logic for adding items to the cart.
- 14/03/25: Implemented the record_sale function with database transactions to ensure data integrity.
- 28/03/25: [Sprint 2 Completed]

April 2025

- 28/04/25: [Sprint 3 Started] Began development of the main dashboard page.

May 2025

- 02/05/25: Implemented sales summary metrics (Today, Week, Year).
- 09/05/25: Developed the reports page and integrated Plotly for data visualisation charts.
- 16/05/25: [Sprint 3 Completed]
- 19/05/25: [Sprint 4 Started] Began research and implementation of the Prophet library for sales forecasting.
- 26/05/25: Integrated the Google Gemini AI for generating business performance reports.
- 30/05/25: Implemented the two-factor authentication (2FA) system via email.

June 2025

- 06/06/25: [Sprint 4 Completed]
- 09/06/25: [Sprint 5 Started] Commenced the final testing phase.

- 10/06/25: Conducted "Normal Usage Testing" to identify workflow issues and usability bugs.
- 13/06/25: Began writing unit tests using the pytest framework to test individual functions.
- 16/06/25: Focused on fixing the bugs identified during both testing cycles.
- 20/06/25: [Sprint 5 Completed] All major bugs resolved and code frozen.
- 23/06/25: Finalised all portfolio documentation and prepared the presentation for the client handover.
- 25/06/25: [Milestone] Final project and portfolio submitted.

Bibliography

ArjanCodes (2024). *SQLAlchemy: The BEST SQL Database Library in Python*. [online] YouTube. Available at: https://www.youtube.com/watch?v=aAy-B6KPld8.

Charming Data (2023). *Push Code to your GitHub Account - Under 3 Minutes*. [online] YouTube. Available at: https://www.youtube.com/watch?v=vpRkAoCqX3o [Accessed 25 Jun. 2025].

Emojipedia (2019). *📙 Emojipedia — 😃 Home of Emoji Meanings 💁👌🎍😍*. [online] Emojipedia.org. Available at: https://emojipedia.org/.

Greg Hogg (2022). *Plotly Tutorial - Basics in 7 Minutes!* [online] YouTube. Available at: https://www.youtube.com/watch?v=PqUaDvbczbI [Accessed 25 Jun. 2025].

Lucidchart (2025). *Online Diagram Software & Visual Solution*. [online] Lucidchart. Available at: https://www.lucidchart.com/pages.

Patrick Loeber (2025). *Gemini API with Python - Getting Started Tutorial*. [online] YouTube. Available at: https://www.youtube.com/watch?v=qfWpPEgea2A [Accessed 25 Jun. 2025].

Rob Mulla (2023). *Learning Pandas for Data Analysis? Start Here.* [online] YouTube. Available at: https://www.youtube.com/watch?v=DkjCaAMBGWM [Accessed 24 Oct. 2024].

The Noun Project. (2017). *Notification Bell Icons at Noun Project*. [online] Available at: https://thenounproject.com/browse/icons/term/notification-bell/ [Accessed 25 Jun. 2025].

Unsplash (2024). *550+ Gray Background Pictures | Download Free Images on Unsplash*. [online] Unsplash.com. Available at: https://unsplash.com/s/photos/gray-background.

Vexels (2023). *Blue Computer Pc Icon PNG & SVG Design For T-Shirts*. [online] Vexels.com. Available at: https://www.vexels.com/png-svg/preview/325936/blue-computer-pc-icon [Accessed 25 Jun. 2025].