Designing Document

Anthony Colley

Joshua Swilling

Jose Vazquez

Kennedy Stokes-Sutton

# Introduction

The scope of this project is to create a tool that can be useful for small and medium-sized businesses to track their inventory. As we know, it can be difficult for small businesses to keep their inventory up to date. This tool will also provide a sales tracking feature, combining both functions to help business owners stay at ease with their transactions.

This program is designed to run on any system capable of supporting Python 3.12.0 and MySQL. We expect the program to be executable on any processor with at least dual-core capabilities, such as an Intel Dual-Core processor.

Regarding the user interface (UI) design, we aim to provide the best user experience. Therefore, we recommend that the owner use a screen with at least 720p resolution, although a 1080p display will offer a better experience.

This document will provide information about all the tools that will be used to develop this tool. It will also outline the backend structure with simple diagrams and explain what is expected from the owner in terms of system requirements and usage.

# Architecture

The architecture selected by the team is layered architecture. This approach allows us to separate the responsibilities of the system, simplifying maintenance. It also makes it easier to scale the system in the future if necessary. The layered architecture enables us to add any special features that the owner might request.

## Presentation Layer

* View

The view will be created using the Python extension CustomTkinter, which allows us to build simple buttons and fields. This approach helps us maintain an easy interaction between the user and the system. CustomTkinter is one of the most popular UI libraries for Python, enabling us to work with the backend directly without the need for additional integration, keeping the code simple. As the primary goal of the UI is to facilitate user interaction, it is essential that this interaction is as easy and intuitive as possible to ensure user satisfaction.

## Business Layer

* Main

In the business layer, we manage the processing of all the information gathered in the presentation layer. When a user interacts with the UI, Python will translate that information into the business layer and communicate with the database.

* Scope

The business layer will include all necessary logic, such as calculating sales totals, checking stock levels, and managing other business operations.

## Database Layer

* Inventory

Using Python libraries like sqlite3, we will maintain a connection between the Python program, the backend, and the database. This ensures that all data is stored securely and prevents data loss.

* Database

The database architecture will use SQLite, as it is sufficient for our needs and does not require a larger, more complex database system. We will create five tables: Inventory, Users, Customers, Sales, and Sales Items. Each table will hold specific data, and the Sales\_Items table will interact with the Users, Sales, and Inventory tables, enabling us to track each transaction effectively.

Database: Relational Model

The database will consist of the following tables:

|  |  |
| --- | --- |
|  | Users Table   user\_ID INT PRIMARY KEY username VARCHAR(50) NOT NULL password VARCHAR(50) NOT NULL role VARCHAR(20) NOT NULL date\_created DATETIME DEFAULT CURRENT\_TIMESTAMP Item Table item\_ID INTEGER PRIMARY KEY product TEXT NOT NULL size TEXT NOT NULL quantity INTEGER price DECIMAL(10, 2) NOT NULL date\_added DATETIME DEFAULT CURRENT\_TIMESTAMP Customers Table   customer\_id INTEGER PRIMARY KEY customer\_name TEXT NOT NULL customer\_address TEXT NOT NULL customer\_status INTEGER customer\_notes TEXT date\_order DATETIME DEFAULT CURRENT\_TIMESTAMP Sales Table   sale\_id INT PRIMARY KEY user\_id INT NOT NULL customer\_id INT NOT NULL grand\_total DECIMAL(10, 2) date\_sale DATETIME DEFAULT CURRENT\_TIMESTAMP FOREIGN KEY (user\_id) REFERENCES users(user\_ID) FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id) Sales\_Items Table   sale\_item\_id INTEGER PRIMARY KEY sale\_id INT NOT NULL item\_id INT NOT NULL quantity INTEGER NOT NULL price DECIMAL(10, 2) NOT NULL FOREIGN KEY (sale\_id) REFERENCES sales(sale\_id) FOREIGN KEY (item\_id) REFERENCES items(item\_ID) |

Interaction with the Layers (Presentation, Business, Database)

As mentioned in the presentation layer, all data will be stored in the database to ensure we do not lose critical information, as keeping this data in memory within Python could be risky in the event of a system failure.

The frontend will interact with functions such as viewing items, checking stock levels, and processing sales. When a sale is made, the system will remove the corresponding item from the inventory and record the transaction in the database. A transaction ID (stored as sale\_item\_id in the database) will be generated for each sale, enabling the owner to track all sales.

For example, in the backend class, a function like this:

|  |
| --- |
| def query\_inventory(self, item\_id):  self.cursor.execute('SELECT \* FROM inventory WHERE item\_id = ?', (item\_id,))  row = self.cursor.fetchone() will query the inventory and return the relevant data to the frontend. |

# Entity Relationship Diagram(ERD)

As shown in the entity-relationship diagram, the layers will interact as follows:

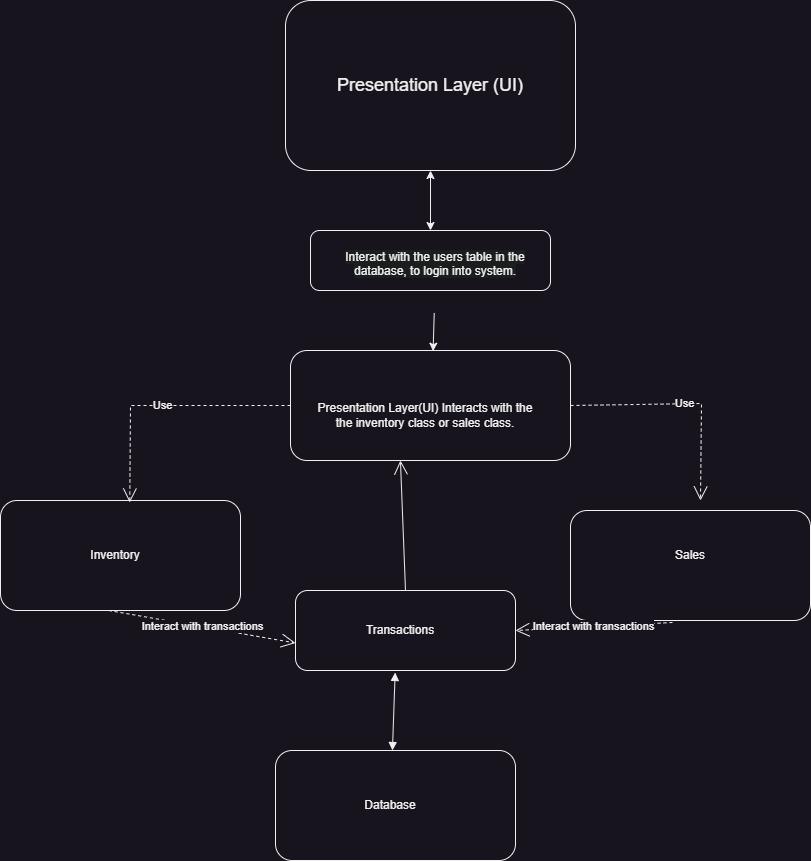
The first layer, which is also the one the user interacts with, is the presentation layer. This layer collects information from the user to assign permissions based on their access level (Level 1, 2, or 3). Each level corresponds to specific actions the user can perform in relation to the database:

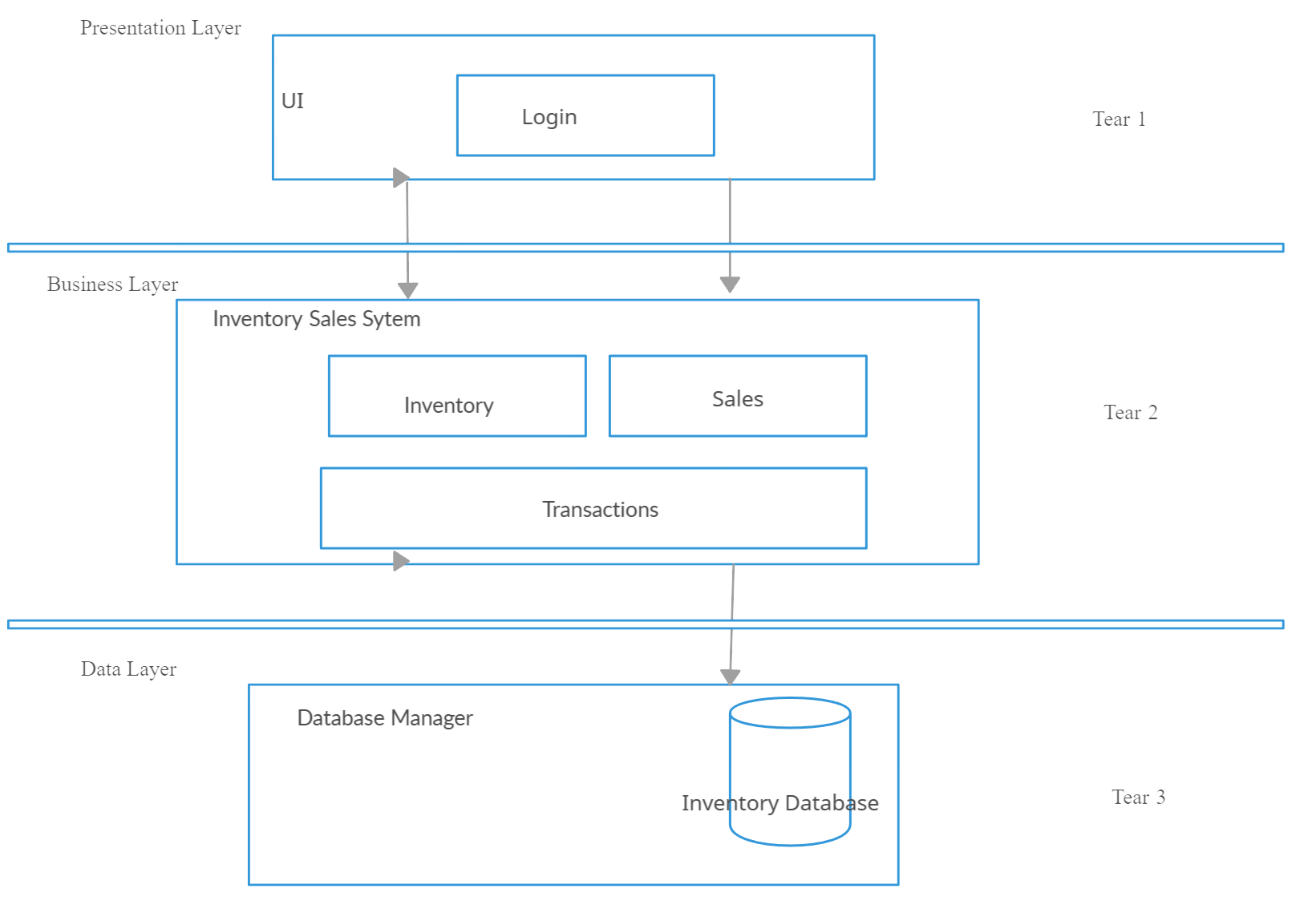
Level 1 users can view items but cannot delete or add new ones.

Level 2 users can interact with the database itself, allowing them to perform actions such as modifying or deleting items.

Level 3, or the owner's level, allows users to add new users and assign them appropriate permission levels.

In the UI, users can choose whether to access the inventory or sales section. Each time there is an interaction (a transaction), the system will record it in the database. The database will then return transaction information, which will be displayed in the UI, allowing the user to view the transaction details.





# Class Diagram

As mentioned in the architecture section, Python will be used as the main programming language to interact with the database. For each instance, such as Login, Inventory, Sales, and Transactions, there will be a defined class. Additionally, there will be hidden classes that do not appear in the UI but are crucial to the backend, as they manage the connections to the database.

Sales Inventory System Class

This is the main class of the program. In this class, you will have access to various options, such as viewing the inventory, seeing customers, reviewing transactions, generating reports, and processing sales.

Customer Class

The Customer Class will hold all customer information which is stored in case they wish to sign up with the store. It will also store transaction details for each customer. Information such as the customer's name, contact details, and purchase history will be stored here.

Inventory Class

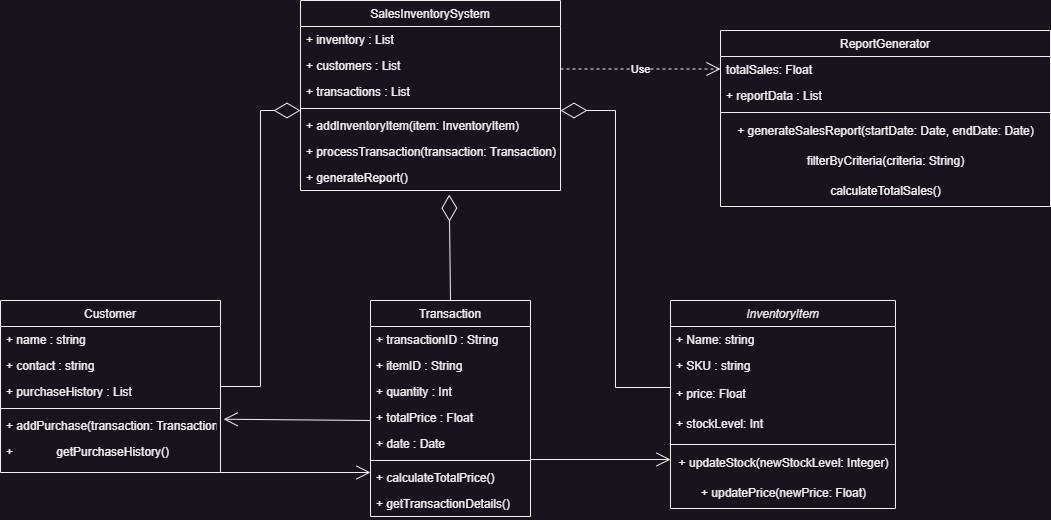
The Inventory Class will display the items in the inventory, including the name, SKU (which will serve as the ID in the database), price, and stock level. This class will also manage stock updates when sales occur and will manage price adjustments.

Report Generator Class

The Report Generator Class will create a transaction report each time a sale is made. This ensures that all sales are recorded in the system’s history, allowing the owner to track sales performance and other metrics.

Backend

The Backend is responsible for interacting with the database. Each class will be connected to the database to ensure that all data is safely stored and updated.



# Database diagram (Schema)

# Database Model

The database model we are going to use is the relational model, as it is the best option that fits the requirements of our program.

We will be dealing with inventory as the central table in the database, followed by tables for sales, users, customers, and sales\_items.

The ID for the inventory will be the SKU, as it allows us to easily identify which product it corresponds to. If an item does not have an SKU, we can manually add it, and the system will first check the database to see if that SKU is available.

Next, we have the users table, which does not have any foreign keys and serves as the root table where all user information will be stored, along with their access levels. Similarly, the customers table does not contain any foreign keys because it is not necessary. Initially, we considered storing all customer information on the items table, but this could lead to incorrect queries.

The sales table has two foreign keys: one for user\_id, which connects to the users table, and another for customer\_id, which connects to the customer’s table.

The sales\_items table will store all transaction history. It also has two foreign keys: sale\_id (connecting to the sales table) and item\_id (connecting to the items table).

Relationships between Tables:

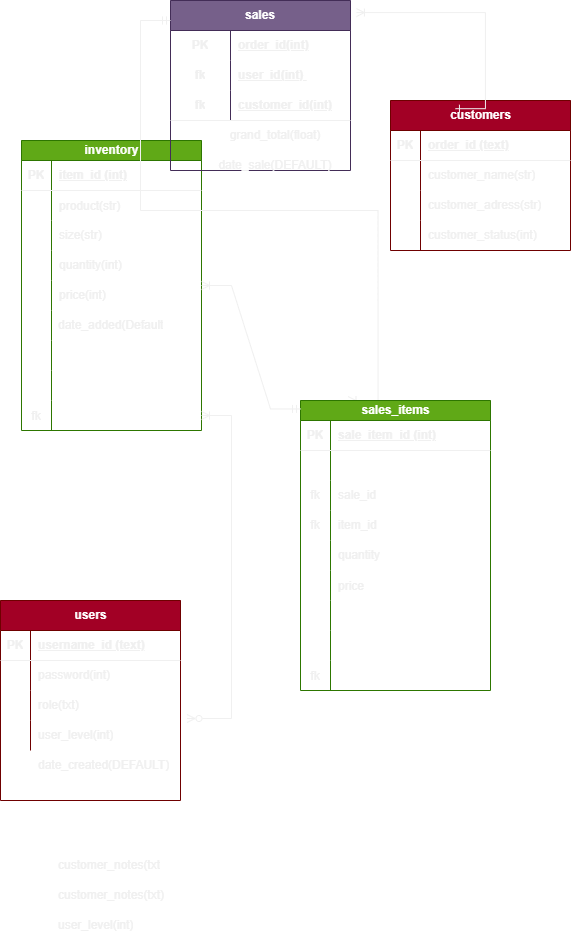
Users and Sales: There is a relationship between users and sales to ensure that each user can make as many sales as needed. This is a one-to-many (1) relationship, meaning one user can make multiple sales, but each sale is linked to a single user.

Customers and Sales: The customers table is connected to the sales table so that each sale is associated with a specific customer. This is also a one-to-many (1) relationship, meaning one customer can make multiple purchases, but each sale is linked to a single customer.

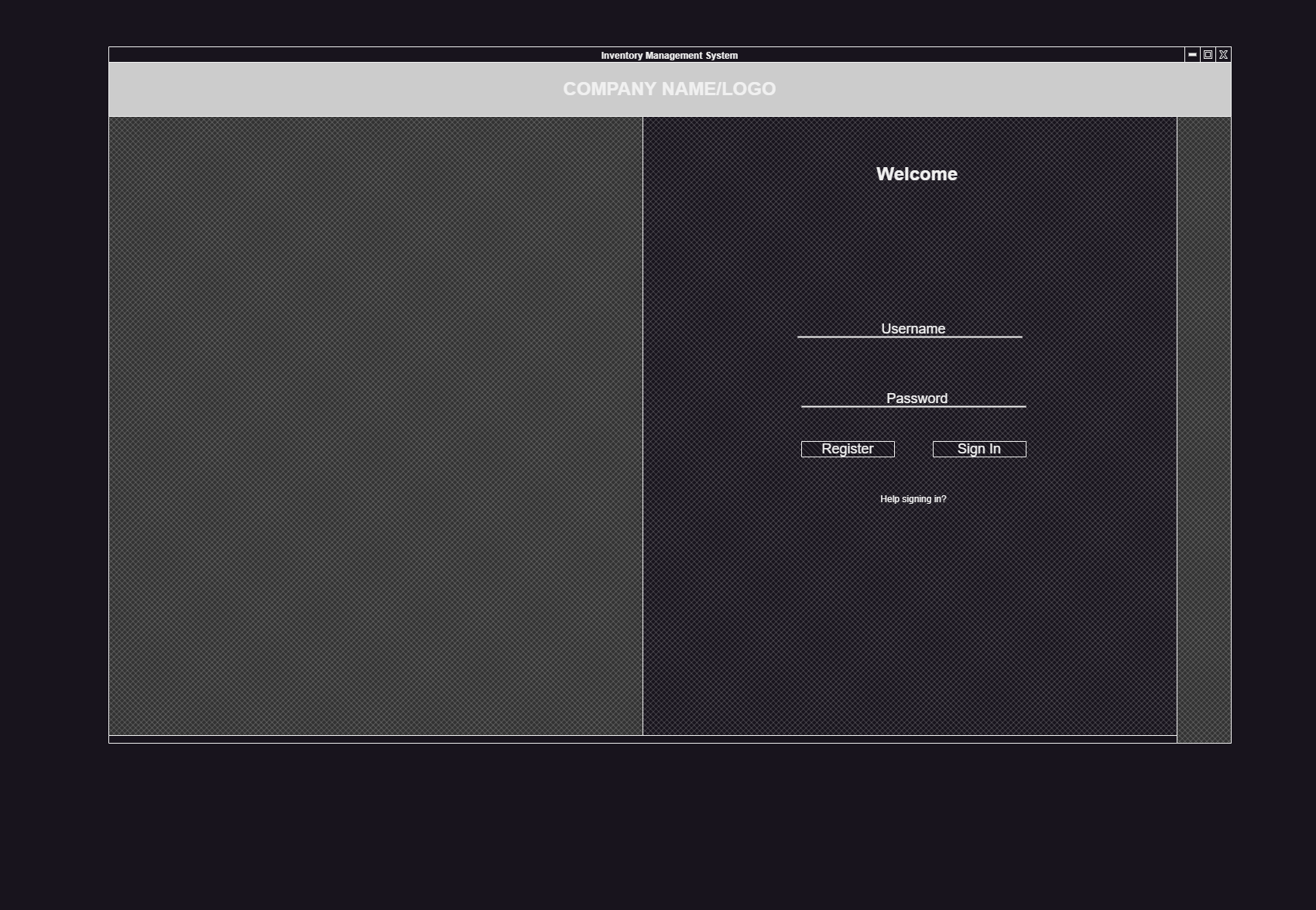
Items and Sales\_Items: The items table is connected to the sales\_items table, which stores records of items sold. This is a one-to-many (1) relationship, where one item can appear in multiple transactions, but each record in the sales\_items table represents one specific item from a sale.

Sales and Sales\_Items: The sales table is connected to the sales\_items table to track the items sold in each sale. This is another one-to-many (1) relationship, where one sale can have multiple items, and each item recorded in the sales\_items table corresponds to a specific sale.

Items and Sales\_Items: The items table is also connected to the sales\_items table. A single item can be sold multiple times in different transactions, until it runs out of stock. This is a one-to-many (1) Relationships, where one item can be sold many times.



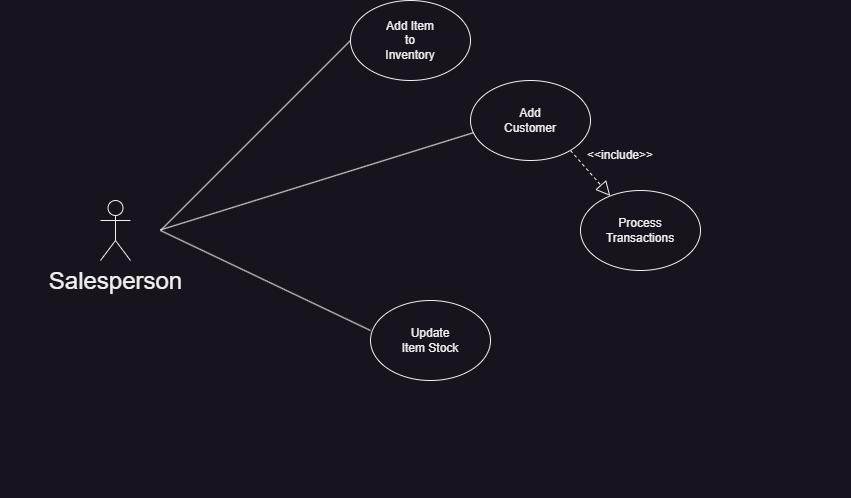
# Interface design

As we know, interface design is a crucial part of this project. Since maintaining data integrity is equally important, we must prioritize the needs of our users. Our goal is to create an interface that is as user-friendly as possible, minimizing unnecessary complexity and focusing on essential features. This approach will facilitate a streamlined training process. A screenshot of a computer

Description automatically generatedScreens screenshot of a computer

Description automatically generated

# Use Cases

In the use cases we have identified three primary user roles: the salesperson, the admin/owner, and the customer. The salesperson will manage all sales transactions, add new customers, and, if authorized, add new items and restock inventory. The admin/owner will have a comprehensive view of all transactions, sales, inventory, and items. Customers will benefit from faster processing if they are already in our database, allowing for quicker purchases. A diagram of a company

Description automatically generatedA black background with a black background and a black circle with white text

Description automatically generated

**Test Cases**

**Login Window**

* Attempt to log in with correct credentials.
* Verify database encryption to prevent unauthorized access.
* Evaluate login functionality with new and deleted users.

**Inventory**

* Check the correct display of all inventory items.
* Evaluate item queries using SKU.
* Verify accurate stock levels.
* Attempt to add a duplicate item (should fail).
* Attempt to add stock to an existing item.
* Attempt to delete an item.
* Attempt to add a new item.
* Ensure basic users cannot modify the database, only view inventory items. Level 2 and 3 users should have modification privileges.

**Sales**

* Verify inclusion of customers, correct items, and grand total in sales transactions.
* Ensure stock is reduced after each sale.
* Verify transaction history for each sale.
* Attempt to sell an out-of-stock item.
* Attempt to sell an unregistered item.
* Attempt to remove an item and add a new one.
* Attempt to sell more quantities than available.

**Transactions**

* Verify that every transaction is recorded.
* Ensure accuracy of information in each transaction.

**Users**

* Evaluate adding, deleting, and modifying users.
* Attempt to log in with deleted user credentials (should fail).

**Database**

* Verify database integrity.

# Summary

Team 4 is making significant progress on the program, prioritizing user-friendliness as the primary goal. As stated, everyone agrees with the project's design. The UML and database schema outline the initial scope of the program. Additional features can be added by coders, but they should inform the team and avoid spending excessive time on them. Everything is progressing as planned.