**CHAPTER 1**

**PROJECT BACKGROUND**

**Rationale**

In the active and fast-paced world of fashion retail, efficient inventory management is vital for sustaining smooth operations and ensuring customer satisfaction. The Queen’z Inventory System is designed for inventory tracking and management for clothing businesses, offering a complete solution to handle the difficulties of stock control.

This system is specifically tailored for clothing retailers to help them maintain accurate and up-to-date records of their inventory. It provides the business owner with real-time insights into the availability of various clothing items, including details such as dress styles, colors, designs, and sizes. With this system, owners can easily monitor the quantity of stocks and make informed decisions to ensure that popular items are sufficiently stocked.

In the Queen’z Inventory System, staff play a vital role in maintaining the accuracy and efficiency of the inventory. They are able to view, add, and edit item information, as well as update the status of items whether available, sold out, or phase out. This capability ensures that the data about the products such as quantity remains up to date. Additionally, staff can track monthly sales by updating the quantity of items sold and keeping the inventory accurate. If needed, they also have the option to print receipts for each transaction, improving the overall customer experience.

On the administrative side, the system allows administrators to efficiently manage the inventory. They can update the quantity of stocks and item availability, track activity logs, and manage accounts. This ensures that the inventory reflected is always accurate and up-to-date, helping to maintain operational efficiency and preventing discrepancies that could result in customer dissatisfaction or loss sales.

Staff can quickly search the inventory for specific items and check availability, enabling them to respond swiftly to customer inquiries and fulfill orders without delay. To ensure accountability, all staff actions are logged, providing a clear record of changes made to the inventory. With these features, both staff and administrators contribute to a highly efficient inventory system, ensuring accurate, reliable data that supports smooth business operations and high customer satisfaction.

By integrating these functionalities into a unified system, the Clothing Business Inventory System enhances inventory management, reduces manual errors, and supports better decision-making. Finally, this system aims to improve the overall efficiency of clothing businesses, enabling them to respond quickly to market demands and trends while keeping their inventory well-organized and optimized.

**Statement of the Problem**

The current inventory management system at Queen'z is inefficient and lacks the ability to track stock levels in real time. This has resulted in discrepancies between physical stock and recorded stock, leading to overstocking or understocking of products, delayed order processing, and reduced customer satisfaction. Additionally, the manual tracking process is time-consuming and prone to human error. Therefore, a more robust, automated inventory system is needed to streamline operations and improve accuracy in stock management.

1. What specific problems does Queen'z face with the current inventory system?

1.1 real-time tracking capabilities;

1.2 stock discrepancies;

1.3 overstocking;

1.4 understocking;

1.5 manual errors

2. How does the lack of automation affect inventory management at Queen'z in terms of the following aspect?

2.1 Manual processes;

2.2 Workload on staff;

2.3 Stock records;

2.4 Customer dissatisfaction.

3. What are the goals of implementing a new inventory system for Queen'z?

4. How will the new system improve customer satisfaction?

5. What kind of data should the new inventory system track?

6. What are the potential risks if the system is not upgraded?

**Significance of the Study**

The implementation of an automated inventory management system for Queen'z is crucial in addressing the current challenges faced by the business.

The following groups stand to benefit from this study:

1. Queen'z Management - The management team will benefit from improved decision-making capabilities, as the system will provide real-time data on the quantity of stocks and sales trends. This will allow for better planning, and cost reduction.

2. Employees - The staff is responsible for inventory management and will benefit from a more efficient and optimized workflow. Automation reduces manual tasks and minimizes the risk of errors, enabling staff to focus on higher-value activities like customer service and sales.

3. Customers- Customers will benefit through fast transactions. The system will help ensure that popular products are always in stock, improving customer satisfaction by reducing instances of delayed or unfulfilled orders.

4. Suppliers - This will improve supplier relationships, as Queen'z will be able to communicate more effectively about stock and reduce last-minute orders.

5. Business Analysts and Researchers - This study provides valuable insights for other businesses looking to adopt automated inventory management systems. It contributes to the growing field of inventory management technology by showcasing the practical benefits of such systems in a retail environment.

6. Owners/Investors - The owners or investors of Queen'z will benefit from increased profitability due to the reduction of overstocking and understocking, improved operational efficiency, and better overall business performance.

**Objectives of the Study**

To develop and implement a streamlined, automated inventory management system for Queen'z that improves the accuracy, efficiency, and overall management of stock levels to support business operations.

**Specific Objectives:**

1. To provide real-time tracking of inventory levels – Ensure that stock levels are updated automatically as products are sold or received, minimizing discrepancies between physical and recorded inventory.

2. To lessen manual errors in stock management – Automate stock entry and monitoring processes to lessen human errors associated with manual tracking.

3. To optimize stock reordering processes – Implement a system that generates notification when stock levels reach a predefined threshold, ensuring timely reordering to avoid stock outs or overstocking.

4. To improve order processing time – Streamline the process of fulfilling customer orders by integrating the inventory system with sales and purchase operations, reducing delays and improving customer satisfaction.

6. To provide a user-friendly interface for staff – Design an inventory system that is easy to use for Queen'z employees, minimizing training time and ensuring efficient operation of the system.

**Definition of Terms**

Admin: The system user with the highest privileges who is responsible for overseeing and managing inventory operations, including monitoring activity logs.

Accountability: The system's ability to track user actions and ensure that all changes to the inventory are recorded, contributing to transparency and security.

Activity Logs: A system feature that records the actions performed by users (e.g., adding or updating stock), which helps in monitoring and accountability.

Customer Engagement: Interaction between the business and customers, often enhanced by providing accurate product availability and a smooth shopping experience.

Database: A structured collection of data stored and managed electronically, typically on a server.

Inventory System: A software tool designed to manage and track stock levels, product availability, and stock movements within a business. In this study, it refers to the Queen’z Clothing Inventory System.

Operational Efficiency: The effectiveness with which a system or process functions in managing tasks, reducing errors, and improving performance in day-to-day operations.

Overstocking: The accumulation of excess stock, which ties up resources and can lead to losses if items are not sold in a timely manner.

Real-Time Inventory Management: A feature of the system that allows for instantaneous updates on stock levels and product availability, ensuring that inventory data is current and accurate.

Stock Levels: The quantity of each item available in the inventory. Maintaining proper stock levels is crucial to preventing overstocking or stock.

**CHAPTER 2**

**REVIEW OF RELATED LITERATURE**

Inventory management systems play a critical role in ensuring operational efficiency and customer satisfaction in retail businesses. Several studies highlight the importance of automated systems in overcoming the limitations of manual inventory processes.

Research by Helo and Szekely (2005) underscores the impact of inventory management on customer satisfaction. Their study revealed that businesses with automated inventory systems experienced fewer stockouts and delays in fulfilling customer orders. This is vital for Queen'z, where maintaining the availability of popular products directly affects customer loyalty and sales.

A study by Jonsson and Mattsson (2008) found that technological advancements in inventory management, such as real-time tracking and predictive analytics, help businesses stay competitive in the marketplace. For Queen'z, adopting such technology can help anticipate stock requirements, minimize waste, and reduce unnecessary stockpiling, all of which are common issues in manual systems.

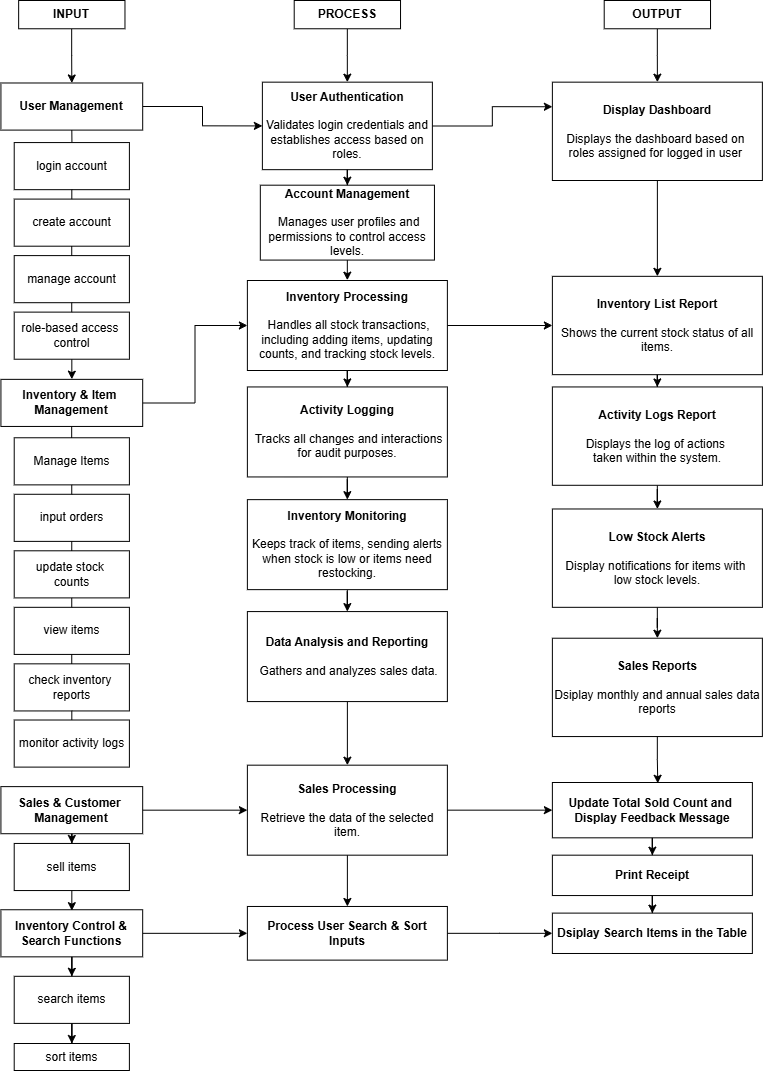
In their research, Waller and Fawcett (2012) examined how automated systems reduce operational costs. They found that companies using inventory management software experienced a significant reduction in manual labor and warehouse costs. This is highly relevant for Queen'z, where the current manual system not only consumes time but also leads to errors that cost the business financially.

Dela Cruz (2021) emphasized the role of technology in helping small businesses grow. The study focused on how automated inventory systems can aid in streamlining business operations, allowing SMEs to compete with larger corporations. For Queen'z, the adoption of an automated inventory system could result in better business scalability, allowing it to grow without being hindered by outdated manual processes.

According to Gonzales (2022), small businesses in the Philippines have reduced operational costs by adopting automated inventory management systems. The study showed that businesses experienced a significant drop in expenses related to overstocking and manual labor. For Queen'z, transitioning to an automated system could lead to similar cost savings, allowing for more efficient allocation of resources.

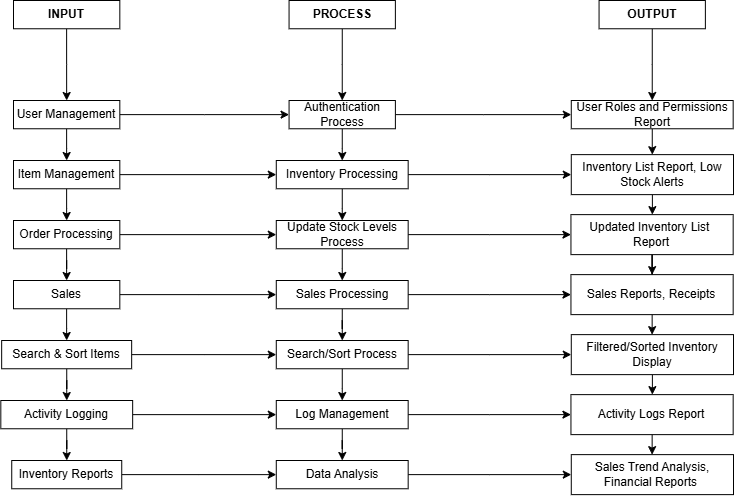
According to Dutta and Sahu (2021), effective inventory systems assist in tracking product quantities, managing reorder points, and minimizing stock outs. These systems can significantly reduce excess inventory costs while ensuring that customer demands are met promptly. Queen'z Inventory System embodies these features, providing users with a friendly interface that facilitates ease of use and quick decision-making. As outlined by Johnson et al. (2022), companies that implement these systems experience improved accuracy in inventory records, leading to better financial forecasting and planning.

According to Reyes and Cruz (2022), effective inventory systems allow companies to maintain optimal stock levels, reducing both excess inventory costs and the risk of stockouts. Queen'z Inventory System incorporates these principles, offering features designed to enhance inventory control and visibility. Villanueva (2021) identifies that systems which integrate these features lead to more efficient operations and better decision-making among local businesses. Queen'z Inventory System reflects these findings by providing an interface that allows users to monitor inventory in real time.

**Theoretical Conceptual Framework**

### 

### **Input, Process and Output Diagram Outline**



**CHAPTER 3**

**METHODOLOGIES**

**Environment**

The Queen’z Inventory System will be implemented within a fashion retail business setting, where efficient inventory management is critical to day-to-day operations. The system is designed for use by both Admin such as business owners and managers as well as Staff, which includes employees, who are handling sales, inventory, and customer service. It will operate on a desktop-based platform accessible through desktop, allowing real-time inventory updates and monitoring. The system will be deployed in an environment with high volumes of stock, frequent updates, and multiple user roles working concurrently, ensuring seamless integration with the store’s sales, inventory, and reporting processes.

The technical environment includes:

* Operating System: The system will be compatible with widely-used operating systems such as Windows, macOS, and Android/iOS for mobile devices.
* Database: A centralized database, such as MySQL will store all inventory data, user logs, sales transactions, and reports.
* Network Requirements: A stable internet connection will be required for real-time synchronization of data between users and the system.
* Security: Data encryption and user authentication protocols will be implemented to ensure secure access and prevent unauthorized system use.

#### Respondents:

The target respondents for the Queen’z Inventory System include:

* **Administrators (Owners, Managers):** These are individuals responsible for overseeing business operations, managing inventory, and handling reports. They will test features such as account management, stock monitoring, and report generation to ensure the system meets their needs for decision-making and operational efficiency.
* **Staff (Sales Personnel, Inventory Managers):** These employees are responsible for daily inventory management tasks, including updating stock status, tracking sales, and assisting customers. They will interact with features like adding/editing item information, checking stock availability, updating sold quantities, and printing receipts.

During the development and testing phases, both **Admin** and **Staff** users will participate in system trials, providing feedback on usability, efficiency, and potential areas for improvement. Respondents will represent a diverse range of experience levels to ensure the system is intuitive for all users.

| Respondents | No. of R |  |
| --- | --- | --- |
| Owner |  |  |
| Employee |  |  |

#### Procedure:

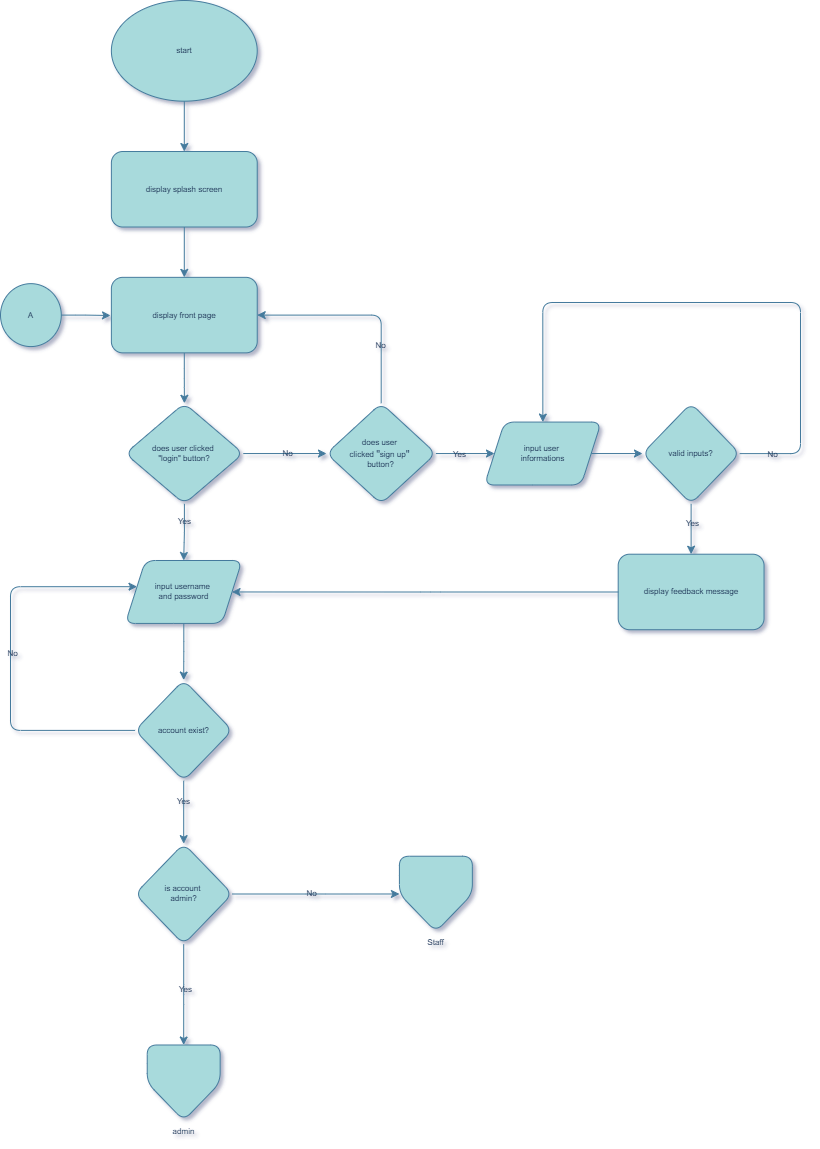
The development and deployment of the Queen’z Inventory System will follow these steps:

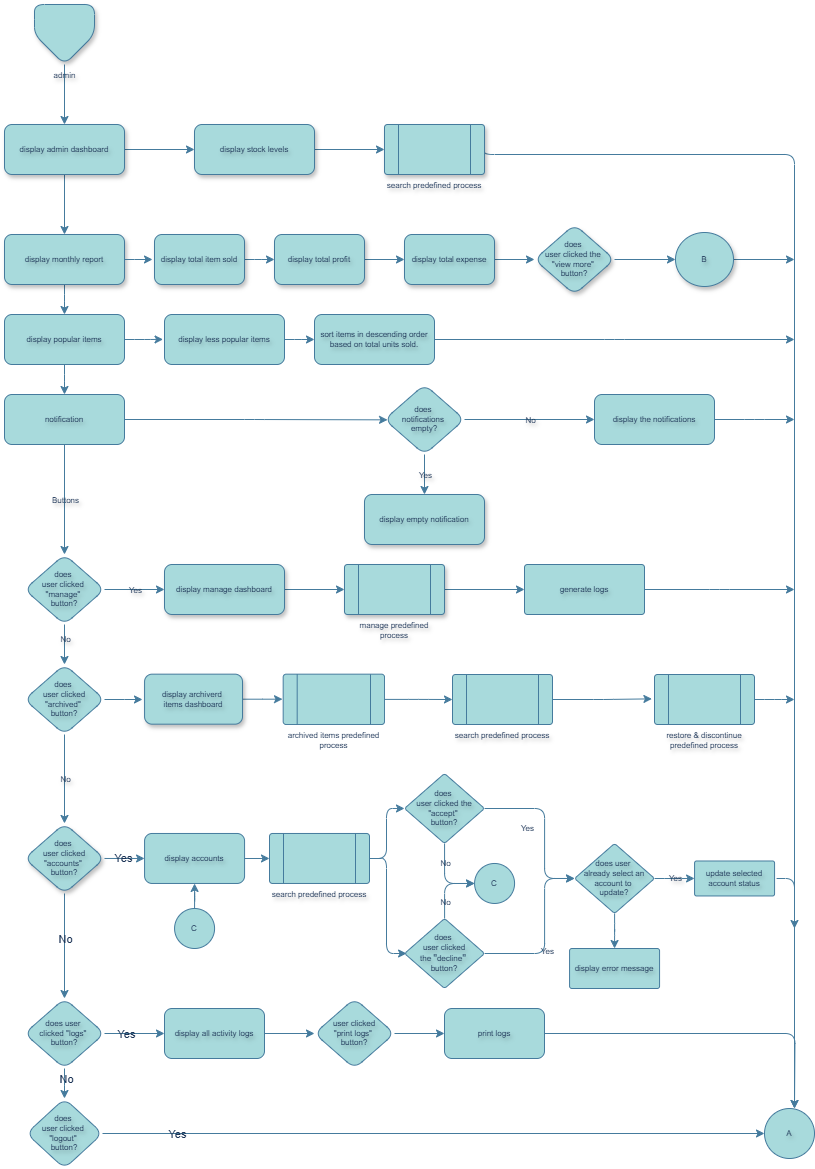
1. **Requirement Analysis:**
   * Initial meetings with the business owner and key staff members will be conducted to gather detailed requirements for the system. This phase will involve understanding the inventory processes, user roles, reporting needs, and system functionalities necessary for daily operations.
2. **System Design:**
   * Based on the requirements, the system architecture will be designed, detailing how the system will handle inventory updates, user roles, and reporting. Wireframes and workflows will be created to illustrate key features like inventory tracking, sales reporting, and notifications.
3. **Development:**
   * The system will be developed iteratively, starting with core features like inventory management, sales tracking, and reporting. Automation features, user authentication, and role-based access will be incorporated to ensure streamlined functionality.
4. **Testing:**
   * User acceptance testing (UAT) will be conducted with both Admin and Staff users. Respondents will test the system’s performance in a real-world scenario, including adding inventory, tracking stock levels, and generating reports. Bugs and usability issues will be documented and addressed.
5. **Deployment:**
   * Once testing is complete, the system will be deployed across the business. Admins will receive training on how to manage user accounts, track inventory trends, and generate reports. Staff will be trained on daily operations, such as updating stock, printing receipts, and handling customer requests.
6. **Evaluation and Feedback:**
   * After deployment, regular feedback from both Admin and Staff users will be collected to assess the system’s performance. System updates and improvements will be made based on this feedback to ensure it continues to meet the evolving needs of the business.

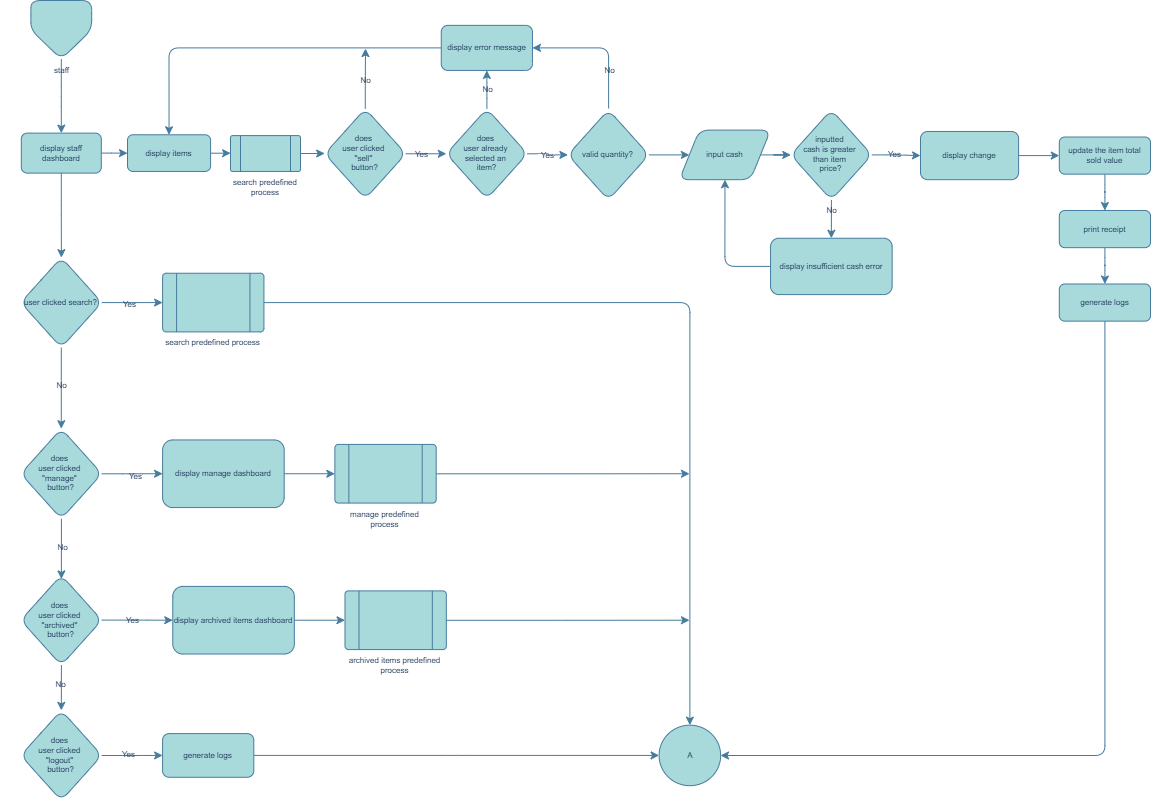
**Diagrams**

**Use Case**

**Flowchart**







**Tools and Software**

To ensure the efficient development, deployment, and maintenance of the Queen’z Inventory System, a variety of tools and software will be employed. These include technologies for system design, development, testing, and security to ensure that the platform operates smoothly across all user roles and devices.

#### Development Tools:

* **Visual Studio Code:** A powerful, open-source code editor that will be used for writing and editing code. It supports various programming languages like HTML, CSS, JavaScript, and PHP for frontend and backend development.
* **Git:** A version control system used to manage and track changes in the source code during the development process. Git ensures that all code revisions are logged, and developers can collaborate efficiently on updates and fixes.

#### Database Management:

* **MySQL / PHPMyAdmin:** A robust relational database management system (RDBMS) will be used to store and manage inventory data, user logs, sales information, and reports. These databases provide secure, scalable storage solutions that can handle the demands of a growing inventory system.

#### Design Tools:

* **Figma**: This tool will be utilized for designing the user interface (UI) and user experience (UX) of the Queen’z Inventory System. Figma enables the design team to create interactive mockups and workflows that illustrate how users (Admins and Staff) will interact with the system, ensuring an intuitive and user-friendly interface.
* **Canva**: Used for creating graphics and presentations, Canva will help in visually communicating the system’s features and benefits. It allows the team to design marketing materials, user guides, and training documents that are visually appealing and easy to understand.
* **Edraw Max and Draw.io**: These diagramming tools will be employed to create flowcharts, use case diagrams, and system architecture visualizations. They facilitate the clear communication of the design process to the development team and stakeholders, ensuring everyone is aligned on the project’s structure and functionalities.

#### Collaboration Tools:

* **Google Meet:** This platform will facilitate virtual meetings among team members, allowing developers, designers, and stakeholders to discuss progress, share updates, and collaborate effectively in real time. It ensures that all team members can connect easily, regardless of their location.
* **GitHub:** Used for version control and collaboration, GitHub will enable the development team to manage code changes, track issues, and collaborate on features. This platform supports code review processes and provides a central repository for all project files, ensuring organized development and easy rollback of changes if needed.

**References**

Helo, P., & Szekely, B. (2005). Logistics information systems: An analysis of software solutions for supply chain co-ordination. Industrial Management & Data Systems, 105(1), 5-18.

Jonsson, P., & Mattsson, S.-A. (2008). Inventory management practices and their implications on perceived planning performance. International Journal of Production Research, 46(7), 1787-1812.

Waller, M. A., & Fawcett, S. E. (2012). Data science, predictive analytics, and big data: A revolution that will transform supply chain design and management. Journal of Business Logistics, 33(2), 77-84.

Dela Cruz, A. (2021). Technology adoption among SMEs in the Philippines: Implications for business growth and innovation. Philippine Journal of Business and Management, 14(2), 45-60.

Gonzales, P. L. (2022). Cost-saving strategies through inventory management automation in small businesses. Philippine Journal of Entrepreneurship, 8(3), 120-130.

Dutta, A., & Sahu, S. (2021). The impact of inventory management systems on business efficiency. *International Journal of Business Research*, 12(4), 123-134.

.

Johnson, R., Smith, T., & Lee, P. (2022). Benefits of adopting modern inventory systems. *Operations Management Review*, 29(2), 45-58.

Delgado, M., & Mendoza, J. (2022). Overcoming challenges in inventory management system implementation among local businesses. *Philippine Journal of Business Studies*, 18(2), 45-58.

**Appendices**

Appendix A – Transmittal Letter

Appendix B – Questionnaire

### **Queen'z Inventory System Questionnaire**

**Section 1: General Information**

1. **Name:**
   * [Your Name]
2. **Position:**
   * [Your Position]
3. **Shop Name:**
   * [Your Shop]
4. **Email:**
   * [Your Email]

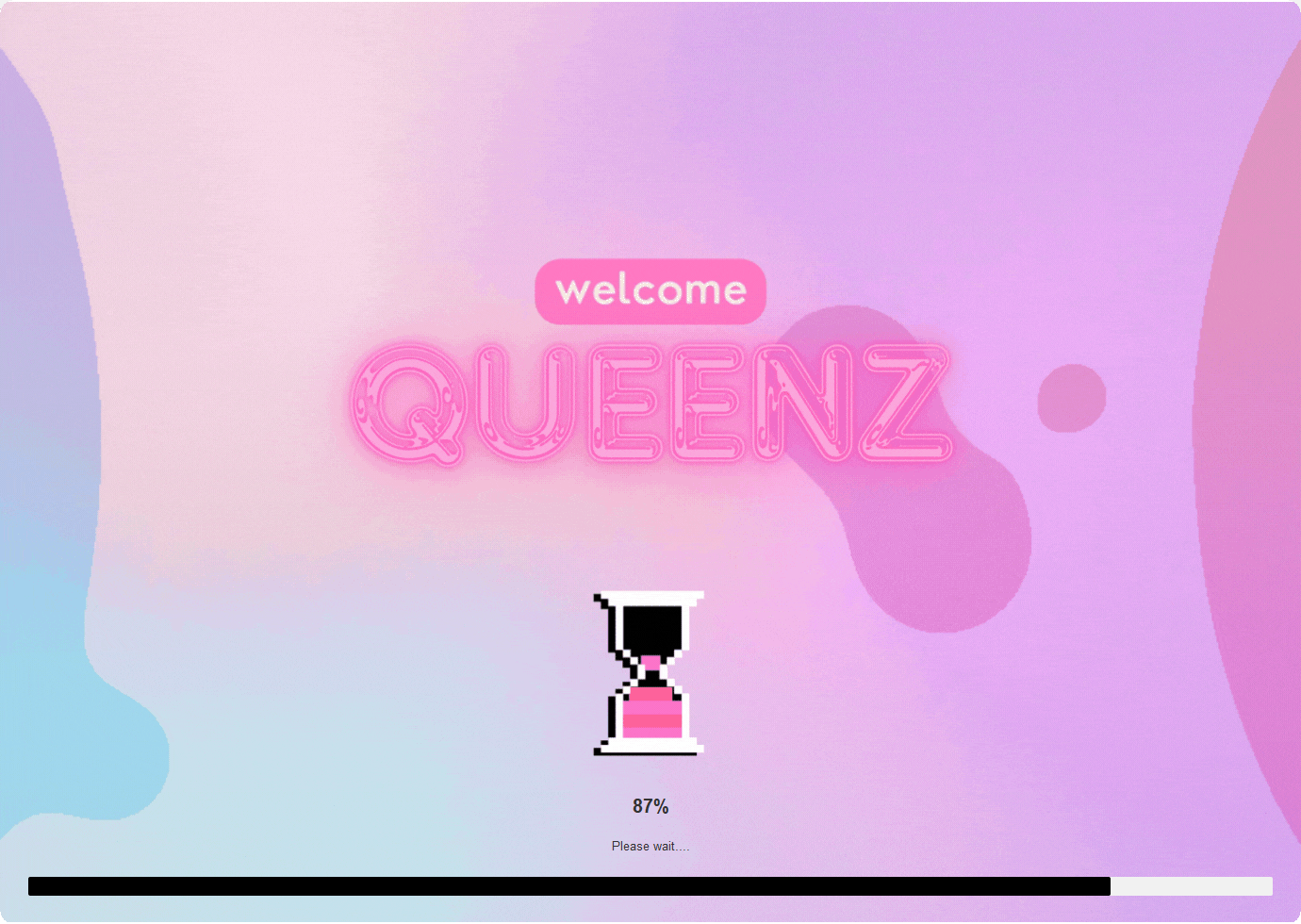
**Section 2: Usage Experience**

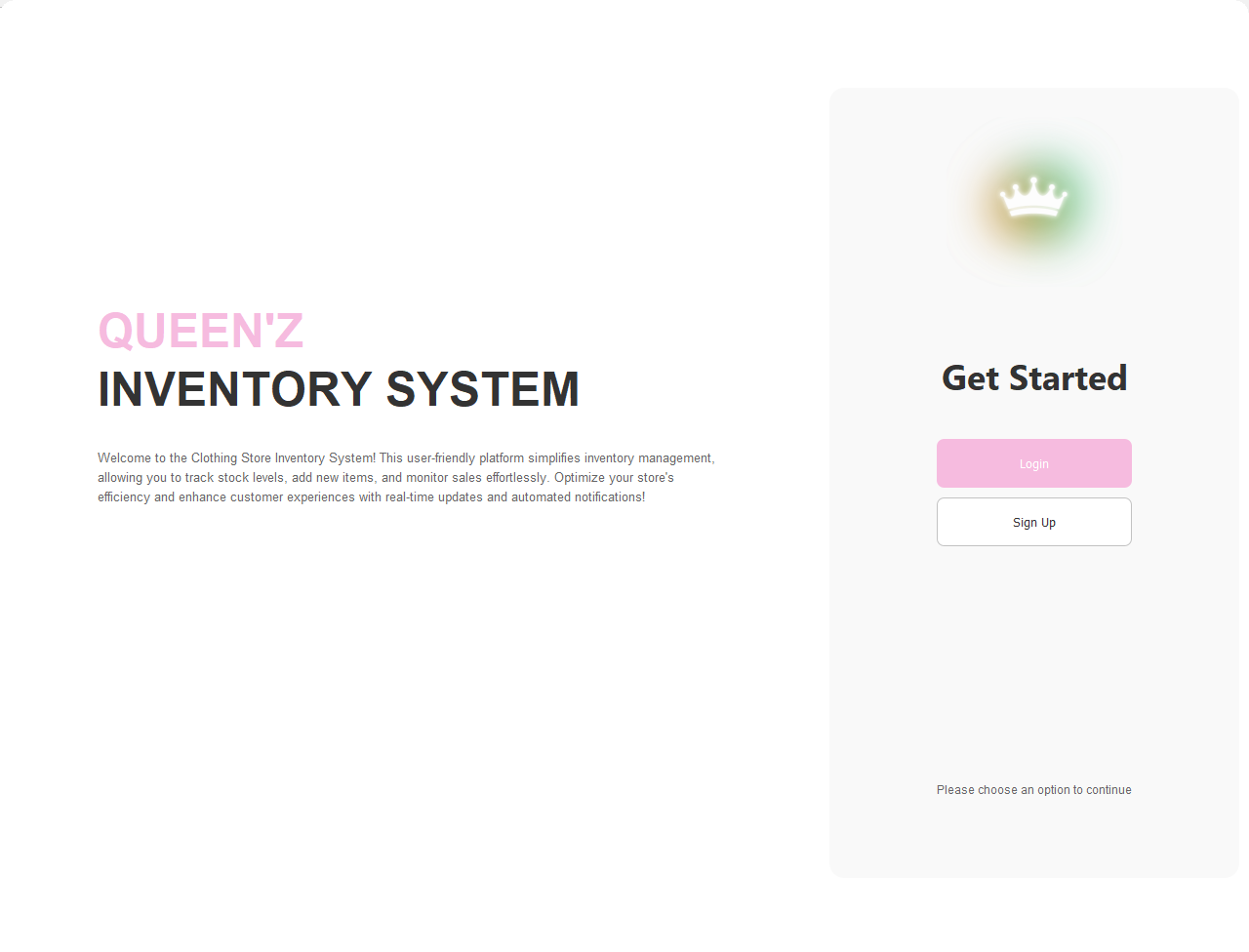
1. **How long have you been using the Queen'z Inventory System?**
   * Less than 1 month
   * 1-3 months
   * 3-6 months
   * More than 6 months
2. **How frequently do you use the system?**
   * Daily
   * Weekly
   * Monthly
   * Occasionally

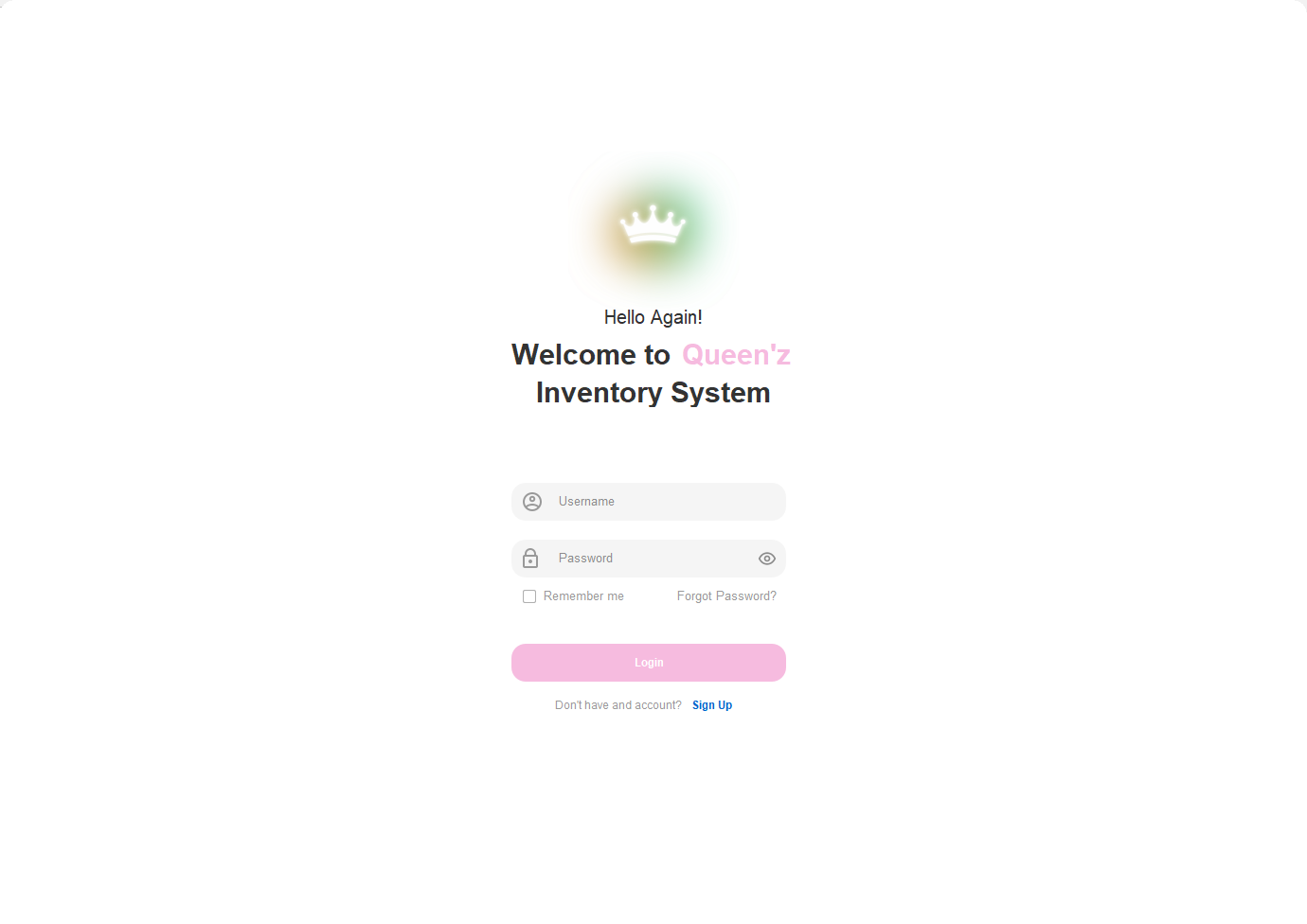
**Section 3: System Features**

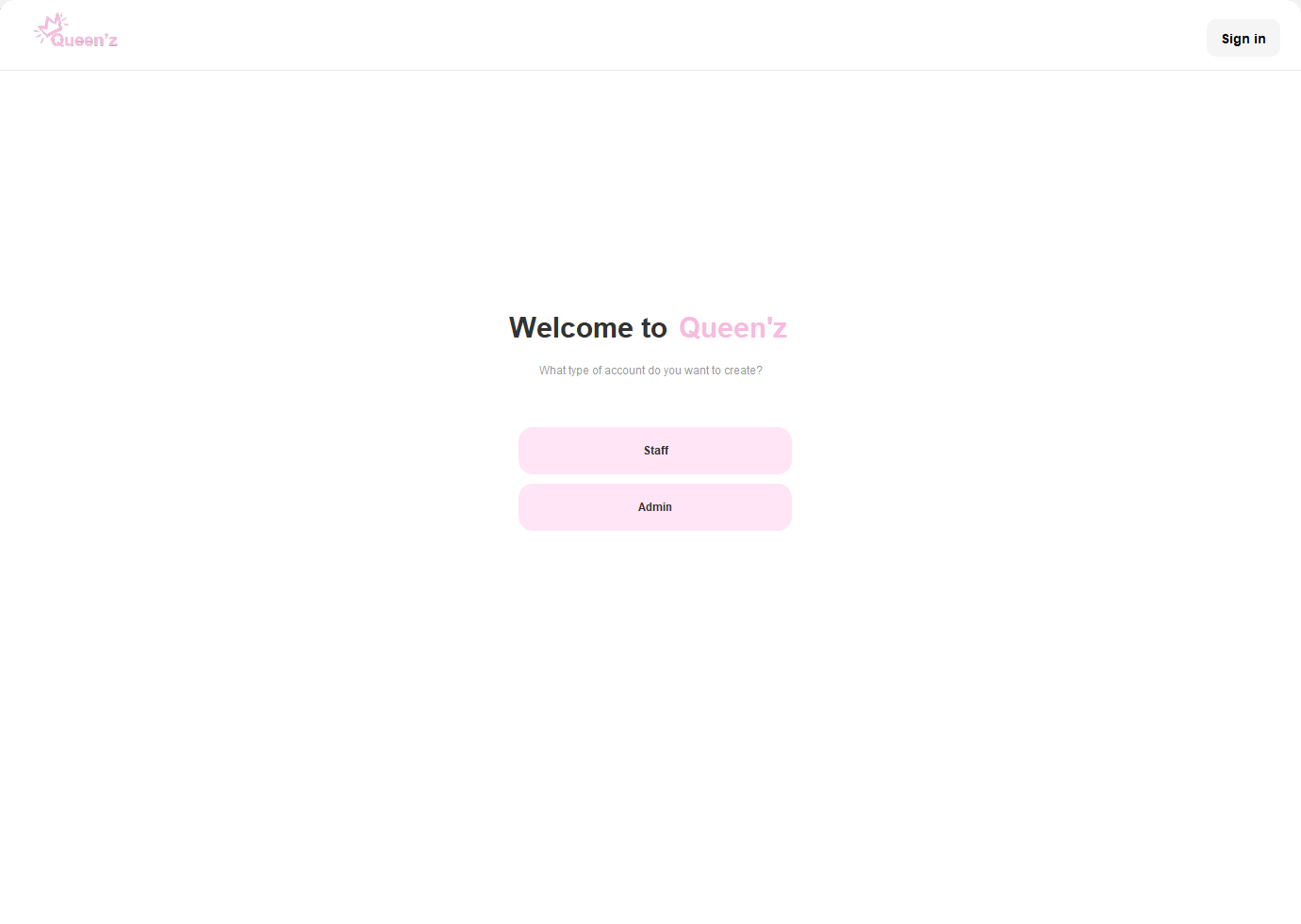
1. **Which features do you find most beneficial?** (Select all that apply)
   * Inventory Tracking
   * Reporting and Analytics
   * User Management
   * Barcode Scanning
   * Integration with Other Systems
   * Other: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. **How would you rate the ease of use of the system?**
   * Very easy
   * Easy
   * Neutral

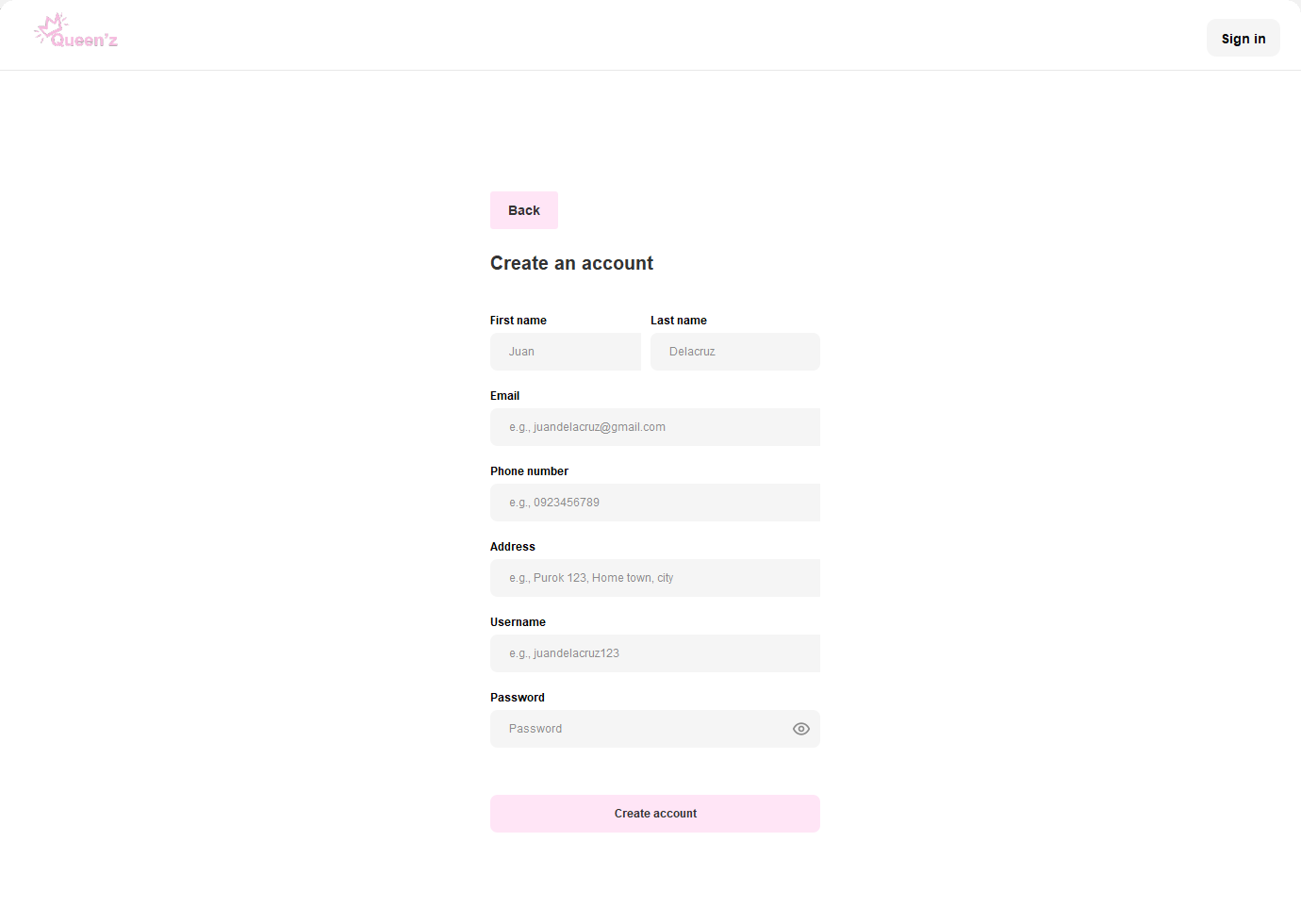
Appendix C – Screenshot of GUI

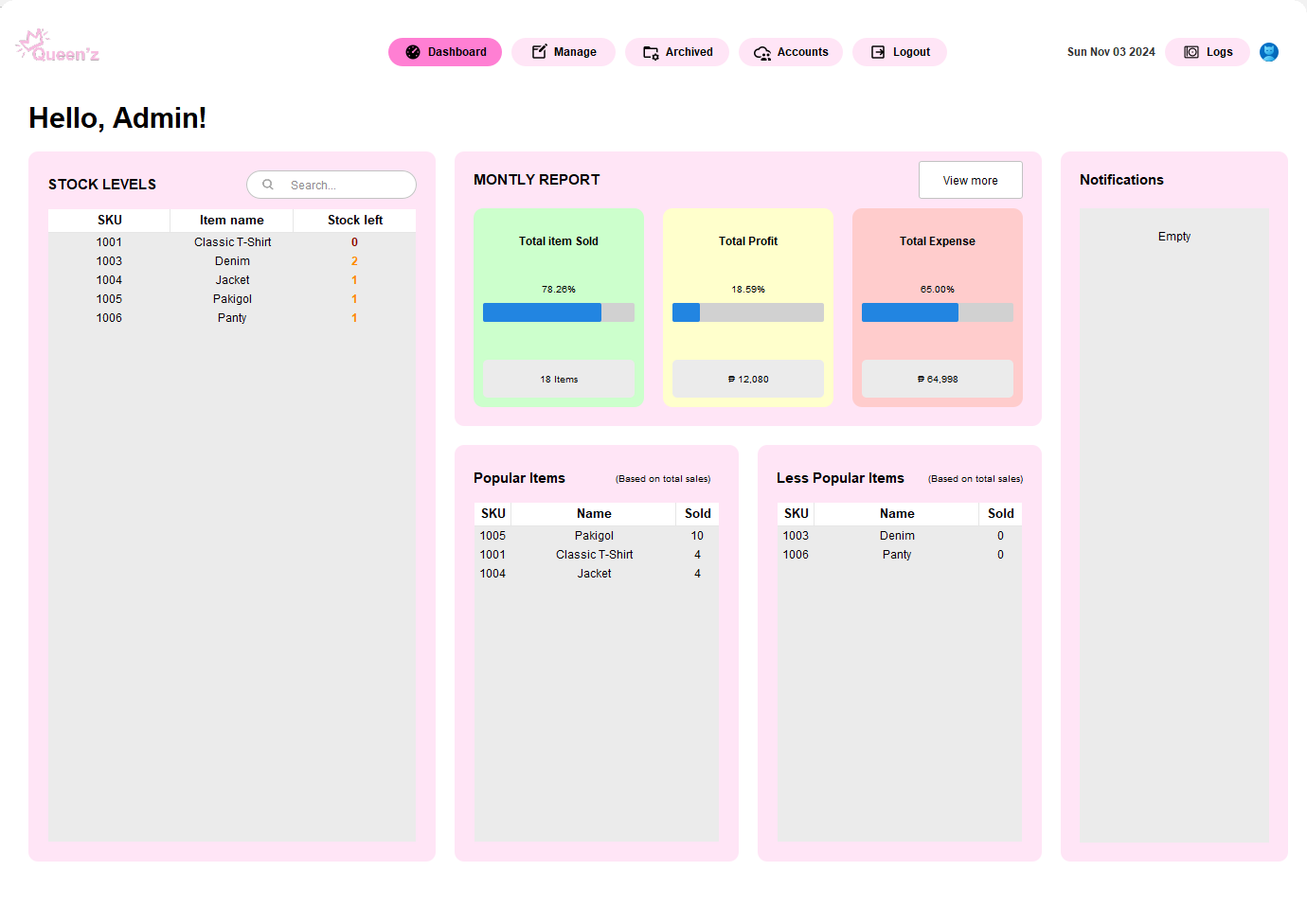


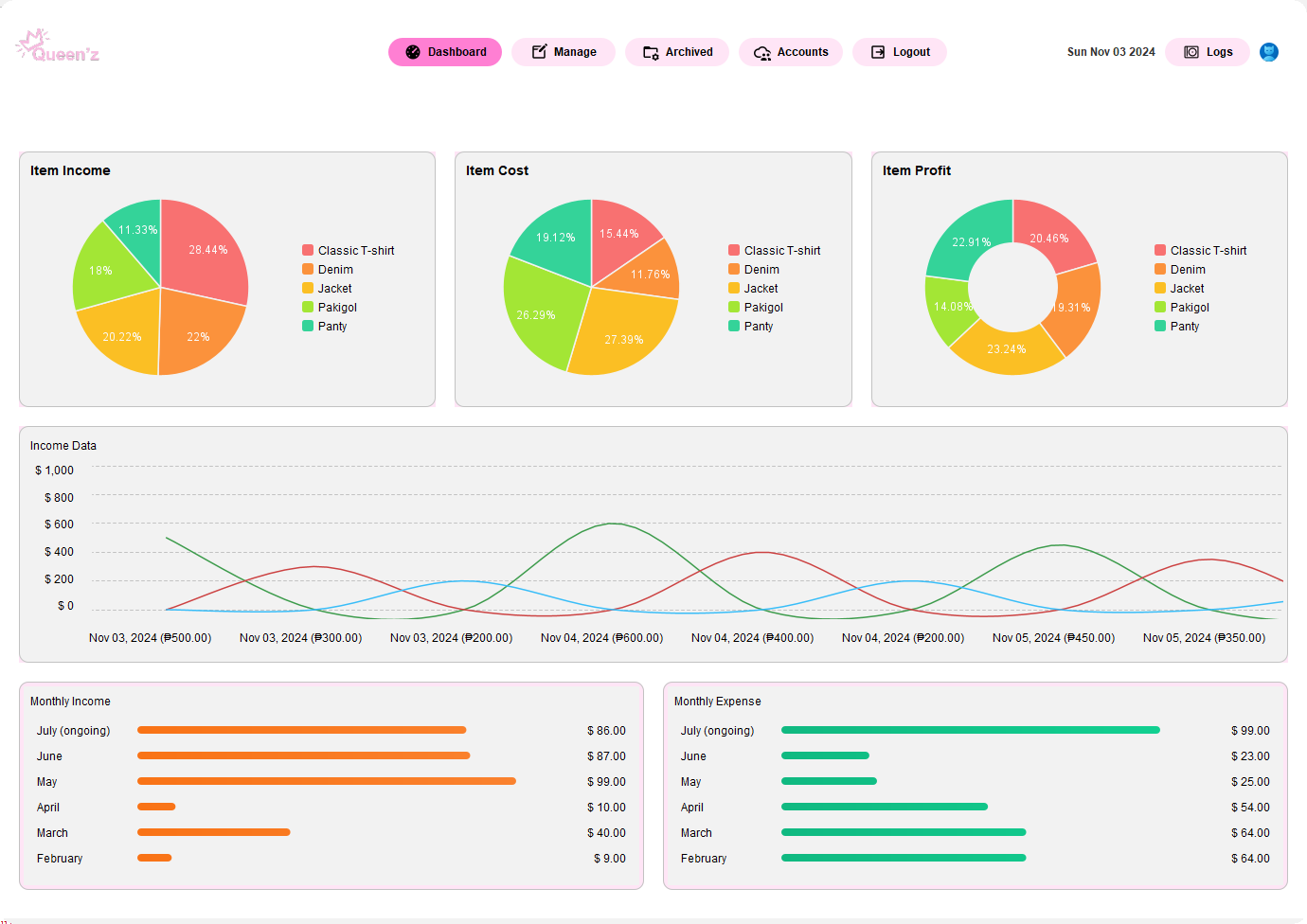


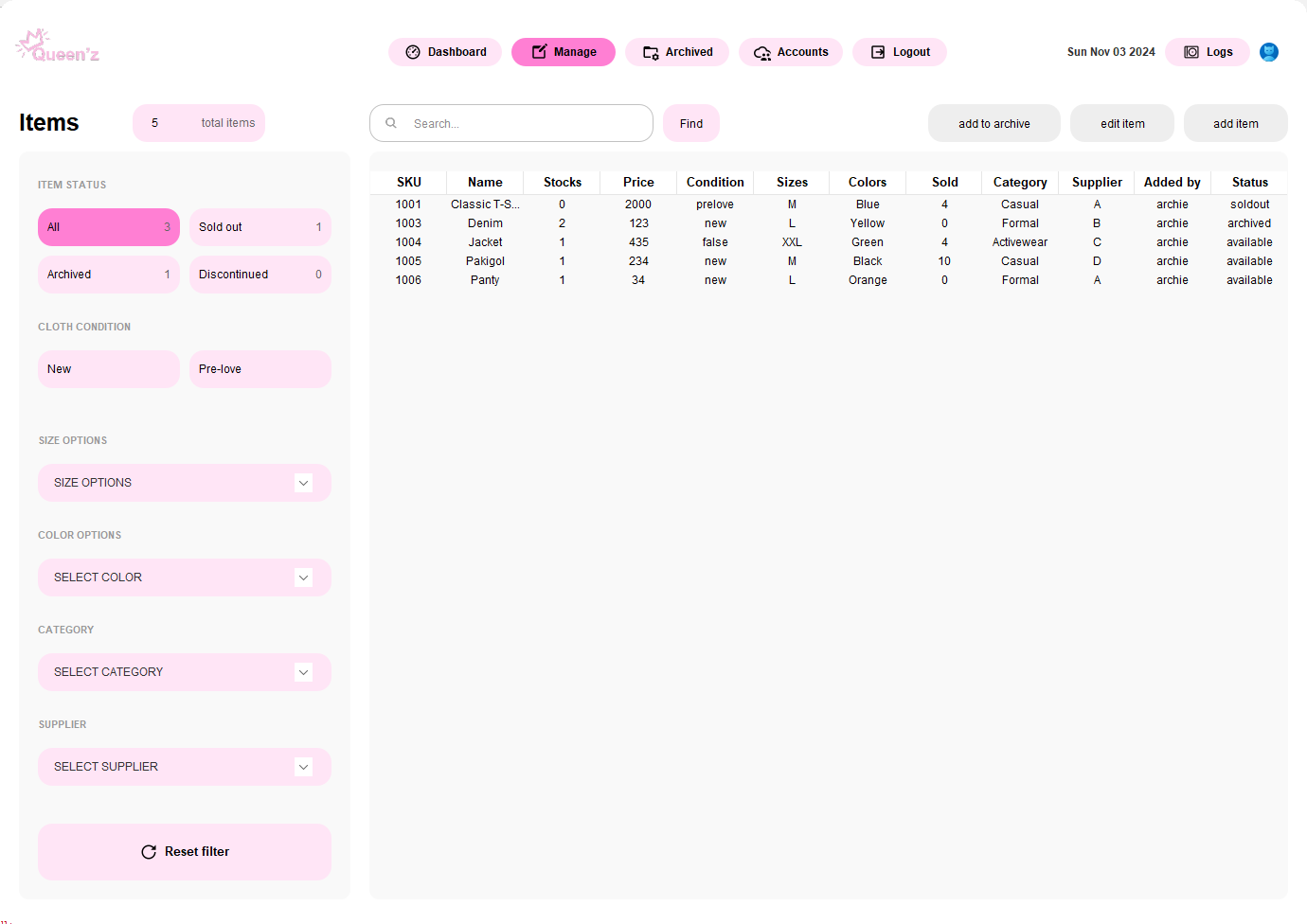


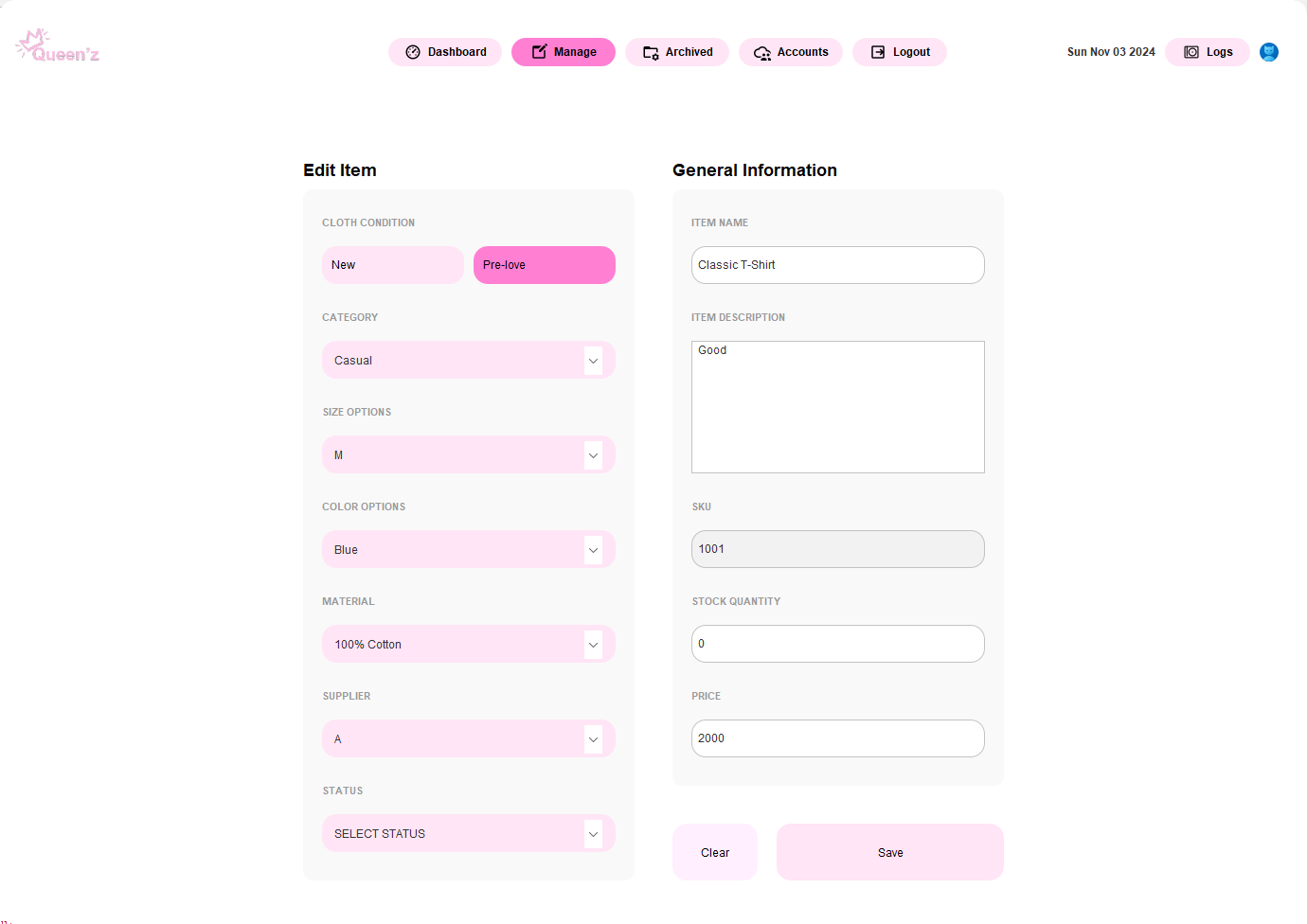


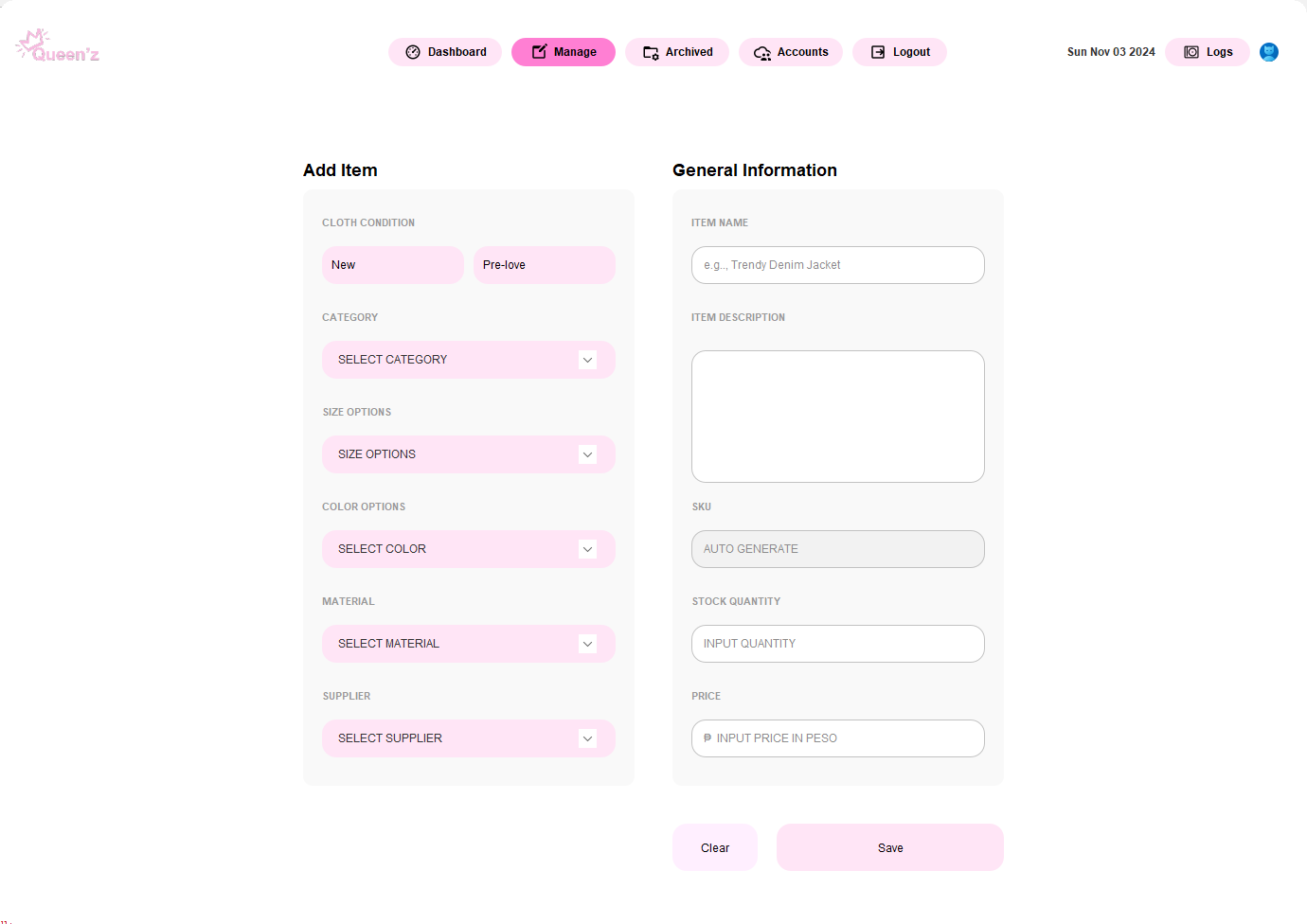


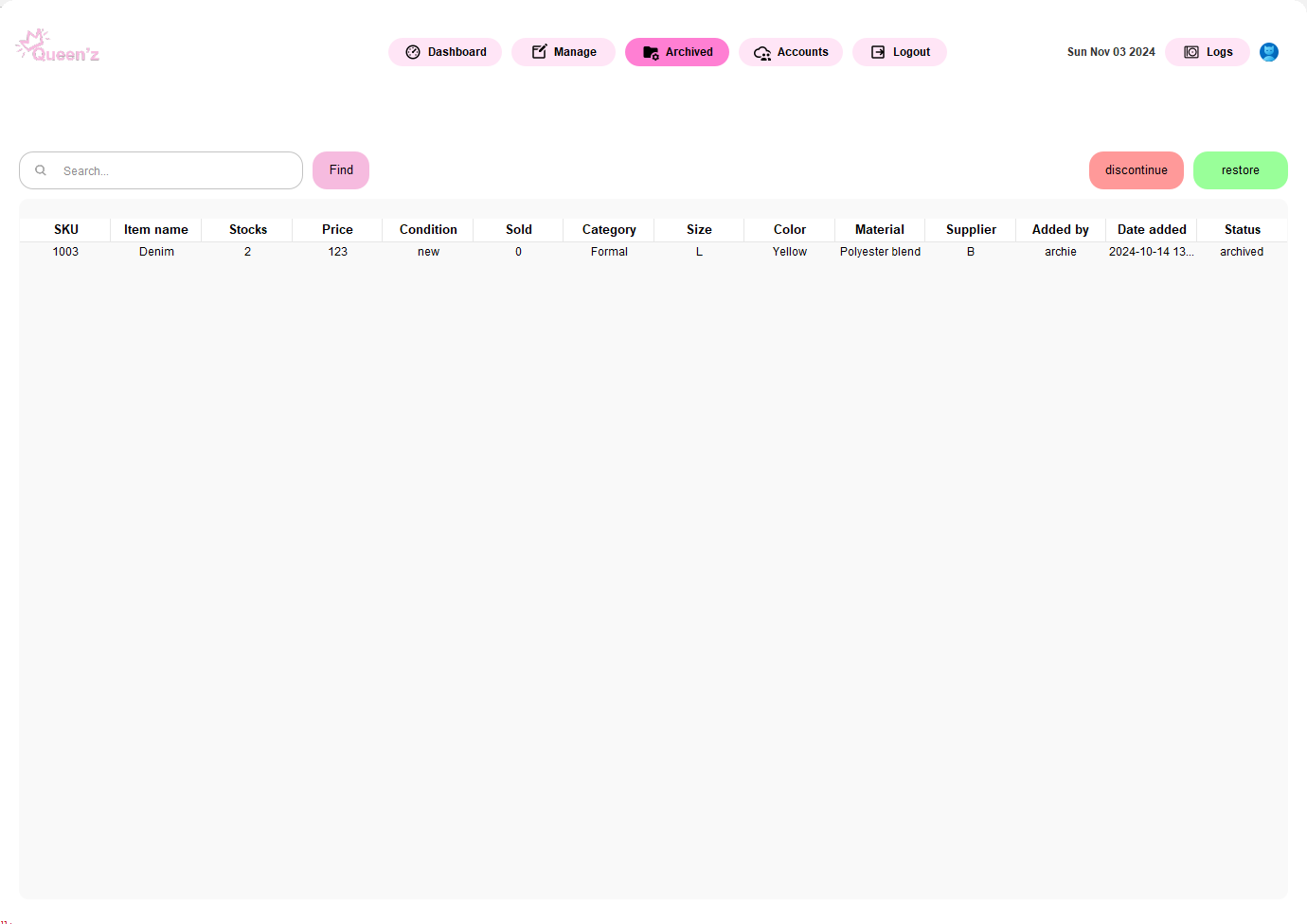


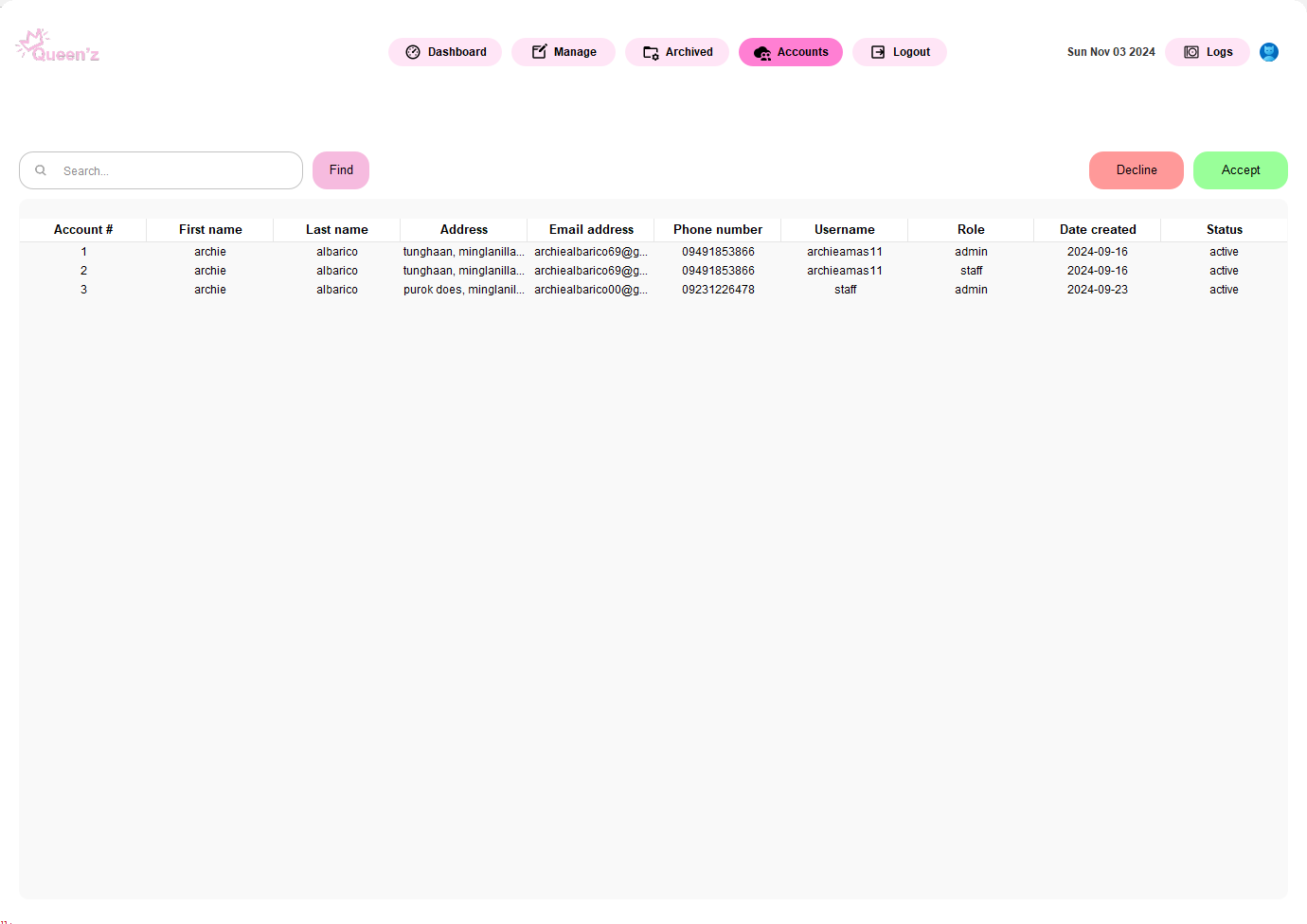


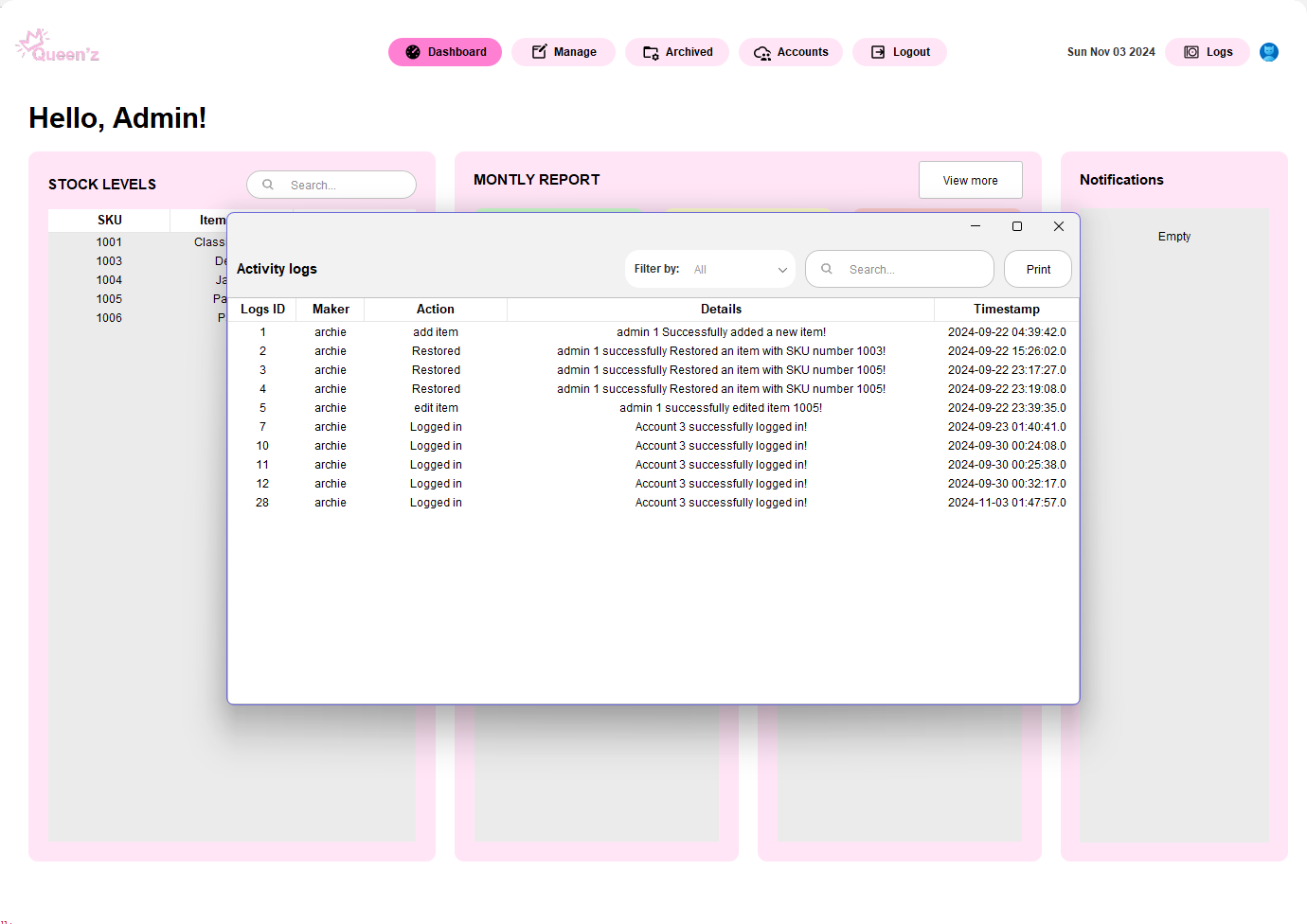


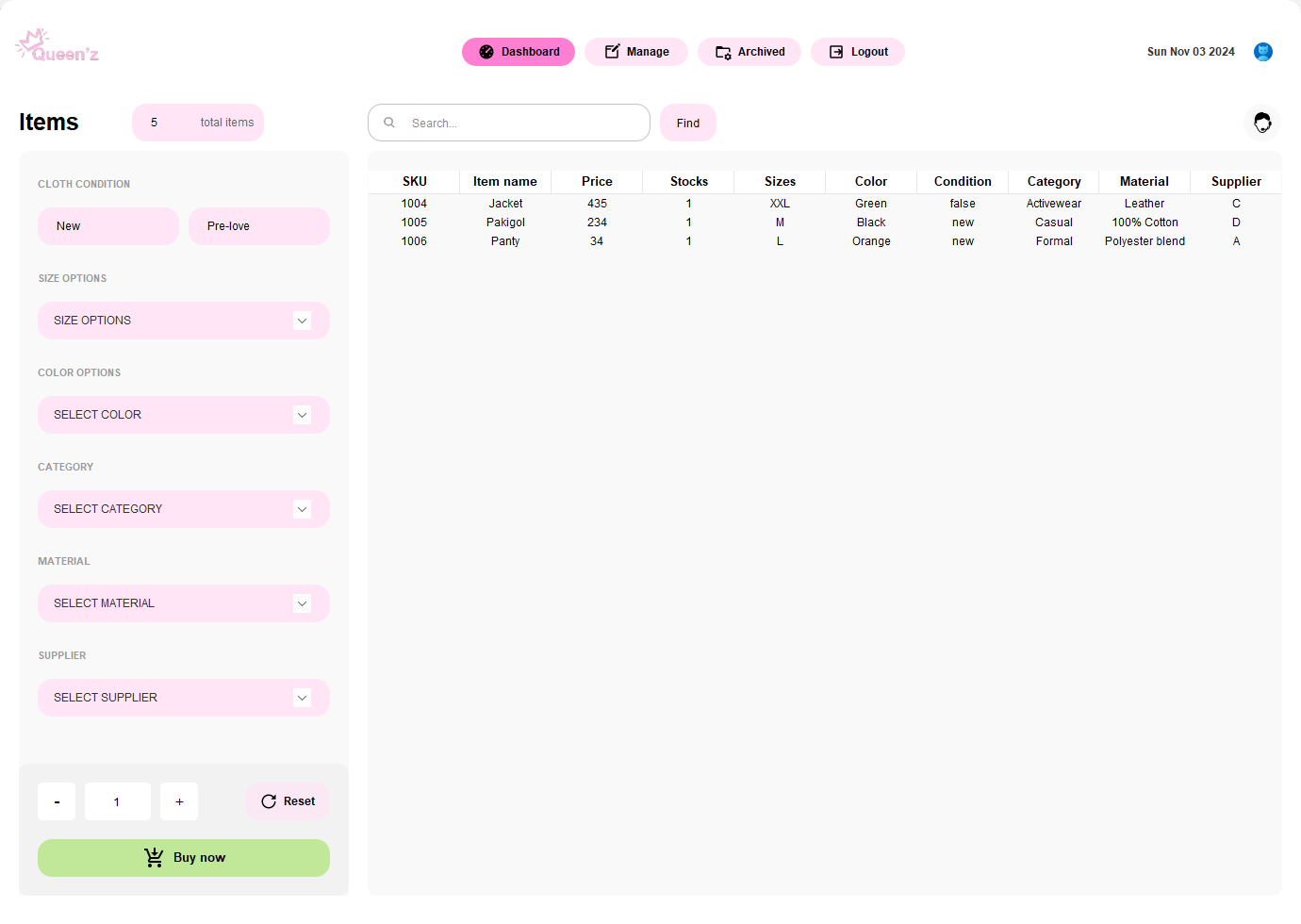




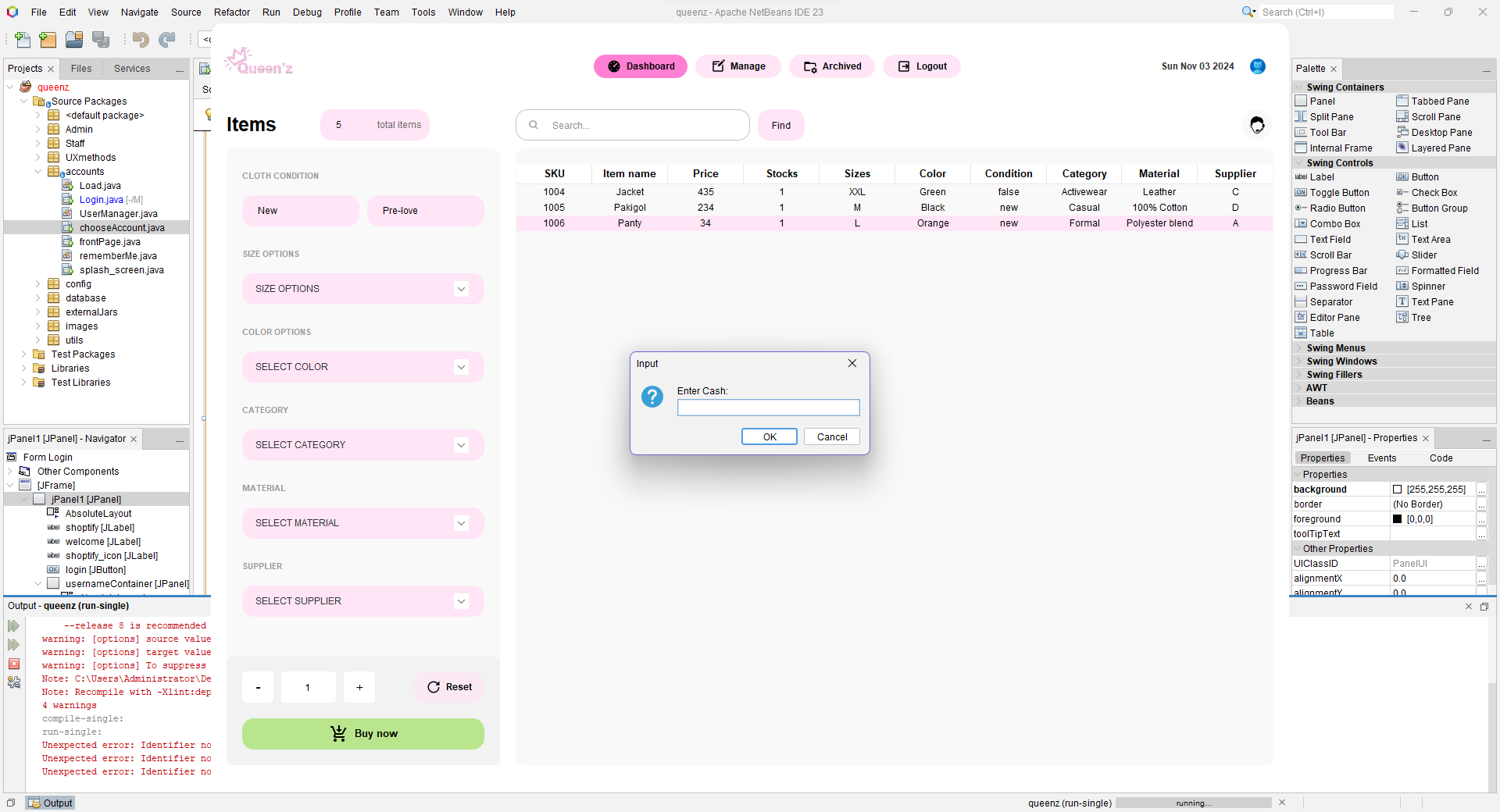


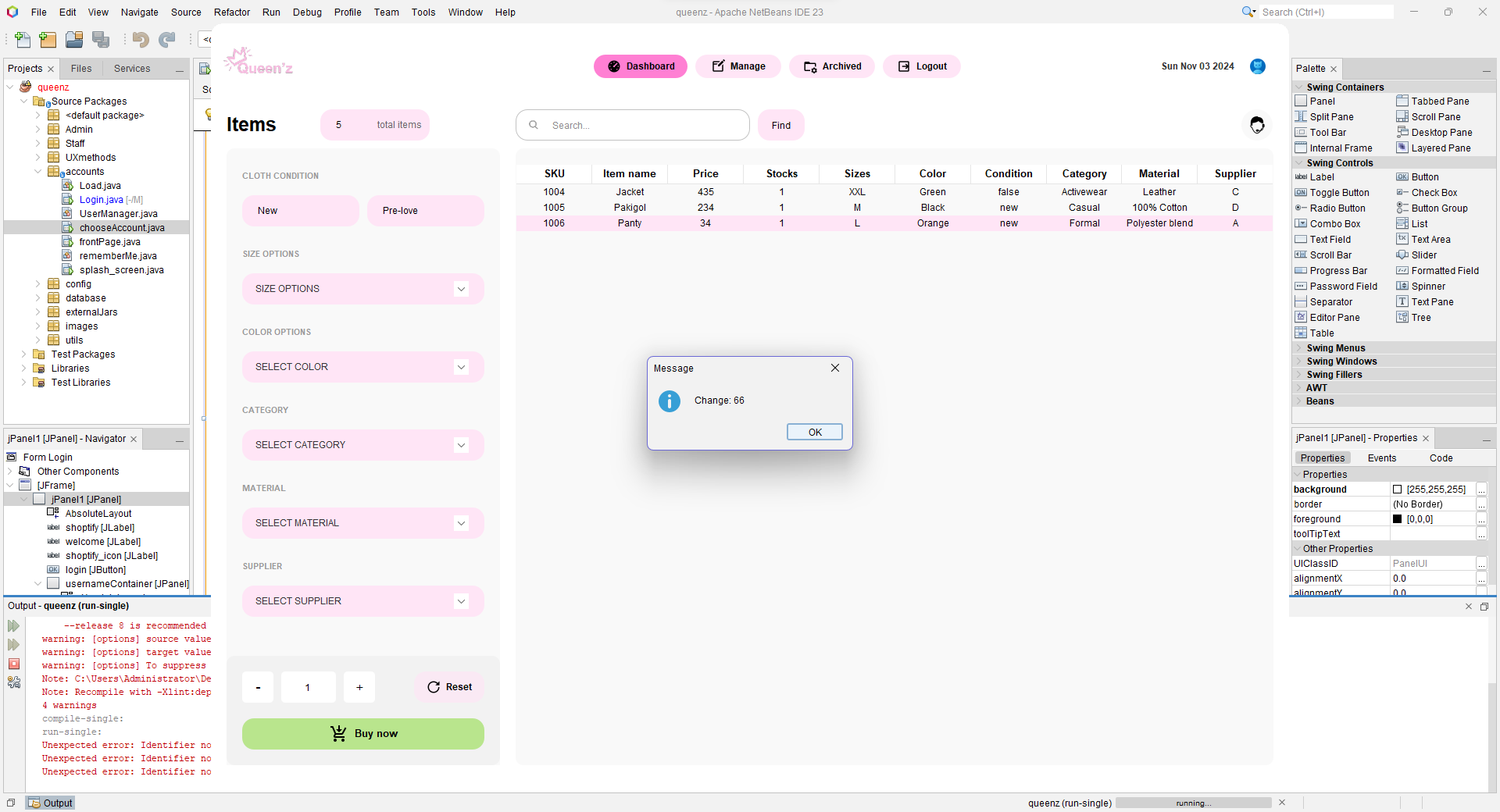


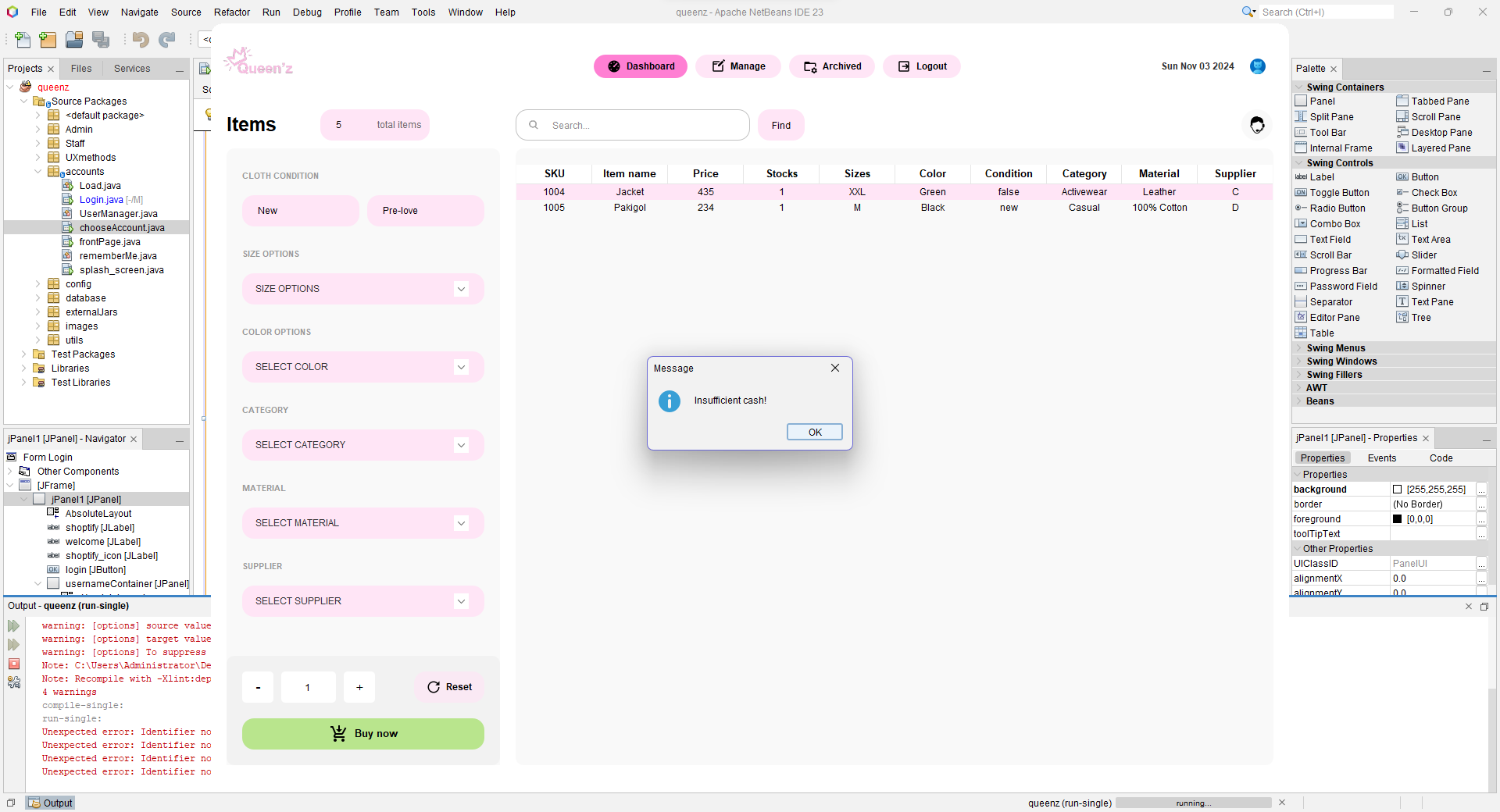




‘







Appendix D – Codes

// This method is triggered when the login action is performed, like when a login button is clicked.

private void loginActionPerformed(java.awt.event.ActionEvent evt) {

// Retrieve the entered username and password from the input fields

String user = username.getText();

String pass = password.getText();

// Check if either the username or password fields are empty or if default placeholder text is present

if (user.isEmpty() || pass.isEmpty() || user.equals("Username") || pass.equals("Password")) {

// Show an error notification if the input fields are not correctly filled

Notifications.getInstance().show(Notifications.Type.ERROR, "Login Failed: Please enter both a username and a password.");

} else {

// Check the login type by calling a method that verifies the username and password

String loginType = checkLogin(user, pass);

// If loginType is not null, it indicates that login was successful

if (loginType != null) {

// Show a success notification if the login was successful

Notifications.getInstance().show(Notifications.Type.SUCCESS, "Welcome back, " + user + "! You have logged in successfully.");

// Set the logged-in user's ID in the UserManager

UserManager.setLoggedInUserId(accountId);

// Determine which dashboard to open based on the login type

switch (loginType) {

case "staff":

// Open the staff dashboard if the login type is "staff"

openDashboard(new staff\_dashboard());

// Record the login for admin/staff tracking purposes

recordAdminLogin();

break;

case "admin":

// Open the admin dashboard if the login type is "admin"

openDashboard(new admin\_dashboard());

// Record the login for admin tracking

recordAdminLogin();

break;

}

} else {

// Show an error notification if loginType is null (indicating invalid credentials)

Notifications.getInstance().show(Notifications.Type.ERROR, "Login Failed: The entered username or password is incorrect. Please double-check your credentials and try again.");

// Clear the password field for security after a failed login attempt

password.setText("");

}

}

}

// This method is triggered when the "Submit" action is performed, such as clicking the "Submit" button in a registration form.

private void submitActionPerformed(java.awt.event.ActionEvent evt) {

// Retrieve input values from each form field

String first\_name = fname.getText(); // First name input

String last\_name = lname.getText(); // Last name input

String em = email.getText(); // Email address input

String phone = number.getText(); // Phone number input

String user = username.getText(); // Username input

String pass = password.getText(); // Password input

String add = address.getText(); // Address input

String selectedRole = role; // Role selected for user

// Check if any of the required fields are empty

if (em.isEmpty() || first\_name.isEmpty() || last\_name.isEmpty() || selectedRole.isEmpty() || user.isEmpty() || pass.isEmpty() || phone.isEmpty() || add.isEmpty()) {

// Show error notification if any field is empty

Notifications.getInstance().show(Notifications.Type.ERROR, "Please fill in all fields!");

return;

}

// Check if the email address exceeds the character limit

if (em.length() > 35) {

Notifications.getInstance().show(Notifications.Type.ERROR, "The Email Address is too long. The limit is 35 characters.");

return;

}

// Check if the first name exceeds the character limit

if (first\_name.length() > 20) {

Notifications.getInstance().show(Notifications.Type.ERROR, "First Name is too long. The limit is 20 characters.");

return;

}

// Check if the last name exceeds the character limit

if (last\_name.length() > 20) {

Notifications.getInstance().show(Notifications.Type.ERROR, "Last Name is too long. The limit is 20 characters.");

return;

}

// Check if the username exceeds the character limit

if (user.length() > 20) {

Notifications.getInstance().show(Notifications.Type.ERROR, "Username is too long. The limit is 35 characters.");

return;

}

// Check password length constraints

if (pass.length() < 8) {

Notifications.getInstance().show(Notifications.Type.ERROR, "Password must be at least 8 characters long.");

password.setText(""); // Clear the password field if the password is too short

return;

}

if (pass.length() > 20) {

Notifications.getInstance().show(Notifications.Type.ERROR, "Password is too long. The limit is 35 characters.");

password.setText(""); // Clear the password field if the password is too long

return;

}

// Validate phone number format (must be 11-12 digits, only numeric)

if (phone.length() < 11 || phone.length() > 12 || !phone.matches("\\d+")) {

Notifications.getInstance().show(Notifications.Type.ERROR, "Invalid Phone Number! It must be 11-12 digits long.");

number.setText(""); // Clear the phone field for invalid phone numbers

return;

}

// Create a database connector instance to interact with the database

databaseConnector dbc = new databaseConnector();

try {

// Check if the email is already registered

if (isAccountExist.checkEmail(em)) {

Notifications.getInstance().show(Notifications.Type.ERROR, "Email address already registered.");

return;

}

// Check if the username is already taken

if (isAccountExist.checkUsername(user)) {

Notifications.getInstance().show(Notifications.Type.ERROR, "Username already taken.");

return;

}

// Encrypt the password using BCrypt for security

String hashedPass = BCrypt.hashpw(pass, BCrypt.gensalt());

// Prepare SQL query to insert a new user record into the database

PreparedStatement pst;

String sql = "INSERT INTO `tbl\_accounts`(`email\_address`, `first\_name`, `last\_name`, `phone\_number`, `username`,`password`, `role`, `status`, `address`, `date\_created`) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, CURDATE())";

pst = dbc.getConnection().prepareStatement(sql);

// Set each placeholder in the SQL statement with corresponding values

pst.setString(1, em); // Set email address

pst.setString(2, first\_name); // Set first name

pst.setString(3, last\_name); // Set last name

pst.setString(4, phone); // Set phone number

pst.setString(5, user); // Set username

pst.setString(6, hashedPass); // Set hashed password

pst.setString(7, selectedRole); // Set user role

pst.setString(8, "Pending"); // Set account status to "Pending"

pst.setString(9, add); // Set address

// Execute the insert operation

pst.executeUpdate();

pst.close();

// Show success notification upon successful account creation

Notifications.getInstance().show(Notifications.Type.SUCCESS, "Account created successfully.");

// Reset role and navigate back to the login screen

role = null;

Login back = new Login();

back.setVisible(true);

this.dispose(); // Close the current window after account creation

} catch (HeadlessException | SQLException e) {

// Handle any exceptions that may occur during database interaction

}

}

// This method retrieves archived items from the database and displays them in a provided JTable.

public static void archive(JTable archiveTable) {

// SQL query to select item details for archived items

String query = "SELECT "

+ "i.item\_SKU as `SKU`, " // SKU (Stock Keeping Unit) of the item

+ "i.item\_name as `Item name`, " // Name of the item

+ "i.item\_stocks as `Stocks`, " // Number of stocks available

+ "i.item\_price as `Price`, " // Price of the item

+ "i.item\_condition as `Condition`, " // Condition of the item (e.g., new, used)

+ "i.total\_sold as `Sold`, " // Total number of items sold

+ "i.item\_category as `Category`, " // Category of the item (e.g., electronics, clothing)

+ "i.item\_size as `Size`, " // Size of the item, if applicable

+ "i.item\_color as `Color`, " // Color of the item, if applicable

+ "i.item\_material as `Material`, " // Material of the item, if applicable

+ "i.item\_supplier as `Supplier`, " // Supplier of the item

+ "a.first\_name as `Added by`, " // Name of the person who added the item

+ "i.date\_added as `Date added`, " // Date when the item was added to the system

+ "i.item\_status as `Status` " // Status of the item (e.g., archived)

+ "FROM tbl\_items i " // Main table: items

+ "JOIN tbl\_accounts a ON a.account\_id = i.added\_by " // Join with accounts to get the name of the person who added the item

+ "WHERE i.item\_status = 'archived'"; // Filter only archived items

// Call a helper method to execute the query and display the results in the JTable

display\_query(archiveTable, query);

}

// This method retrieves and displays items from the database in a provided JTable, based on their status (e.g., active, archived).

public static void manage(JTable table, String status) {

// SQL query to select item details from the database

String query = "SELECT "

+ "i.item\_SKU as `SKU`, " // SKU (Stock Keeping Unit) of the item

+ "i.item\_name as `Name`, " // Name of the item

+ "i.item\_stocks as `Stocks`, " // Number of stocks available

+ "i.item\_price as `Price`, " // Price of the item

+ "i.item\_condition as `Condition`, " // Condition of the item (e.g., new, used)

+ "i.item\_size as `Sizes`, " // Available sizes for the item

+ "i.item\_color as `Colors`, " // Available colors for the item

+ "i.total\_sold as `Sold`, " // Total number of items sold

+ "i.item\_category as `Category`, " // Category of the item (e.g., electronics, clothing)

+ "i.item\_supplier as `Supplier`, " // Supplier of the item

+ "a.first\_name as `Added by`, " // Name of the person who added the item

+ "i.item\_status as `Status` " // Status of the item (e.g., active, archived)

+ "FROM tbl\_items i " // Main table: items table (tbl\_items)

+ "JOIN tbl\_accounts a ON a.account\_id = i.added\_by "; // Join with accounts table to get 'Added by' info

// If the status parameter is not "all", filter items based on the specified status

if (!"all".equalsIgnoreCase(status)) {

query += "WHERE i.item\_status = '" + status + "'"; // Add a WHERE clause for specific item status

}

// Execute the query and display the results in the specified JTable

display\_query(table, query);

}

// Retrieves and displays basic item details (SKU, name, stock) on the dashboard

public static void dashboard(JTable table) {

// SQL query to select SKU, name, and remaining stock from the items table

String query = "SELECT "

+ "item\_SKU as `SKU`, " // SKU (Stock Keeping Unit) of the item

+ "item\_name as `Item name`, " // Name of the item

+ "item\_stocks as `Stock left` " // Quantity of stock left for each item

+ "FROM tbl\_items "; // Source table: tbl\_items

// Executes the query and displays the results in the provided JTable

display\_query(table, query);

}

// Retrieves and displays user account details in a JTable

public static void accounts(JTable table) {

// SQL query to select various account details from the accounts table

String query = "SELECT "

+ "account\_id as `Account #`, " // Unique ID of the account

+ "first\_name as `First name`, " // First name of the account holder

+ "last\_name as `Last name`, " // Last name of the account holder

+ "address as `Address`, " // Address of the account holder

+ "email\_address as `Email address`, "// Email address of the account holder

+ "phone\_number as `Phone number`, " // Phone number of the account holder

+ "username as `Username`, " // Username of the account holder

+ "role as `Role`, " // Role of the user (e.g., admin, staff)

+ "date\_created as `Date created`, " // Date the account was created

+ "status as `Status` " // Status of the account (e.g., active, suspended)

+ "FROM tbl\_accounts "; // Source table: tbl\_accounts

// Executes the query and displays the results in the provided JTable

display\_query(table, query);

}

// Retrieves and displays item details available to staff, filtered to only show available items

public static void to\_staff\_table(JTable table) {

// SQL query to select item details that are available for staff to view

String query = "SELECT "

+ "item\_SKU as `SKU`, " // SKU of the item

+ "item\_name as `Item name`, " // Name of the item

+ "item\_price as `Price`, " // Price of the item

+ "item\_stocks as `Stocks`, " // Number of stocks available

+ "item\_size as `Sizes`, " // Available sizes for the item

+ "item\_color as `Color`, " // Available colors for the item

+ "item\_condition as `Condition`, " // Condition of the item (e.g., new, used)

+ "item\_category as `Category`, " // Category of the item (e.g., electronics, clothing)

+ "item\_material as `Material`, " // Material of the item, if applicable

+ "item\_supplier as `Supplier` " // Supplier of the item

+ "FROM tbl\_items " // Source table: tbl\_items

+ "WHERE item\_status = 'available'"; // Filter to show only items with status 'available'

// Executes the query and displays the results in the provided JTable

display\_query(table, query);

}

// Updates the total items count on a JLabel

public static void updateTotalItems(JLabel totalItemsLabel) {

try {

// Create a new database connection

databaseConnector dbc = new databaseConnector();

// SQL query to count the total items in tbl\_items

String query = "SELECT COUNT(\*) as TotalItems FROM tbl\_items";

// Prepare and execute the query, store the result in ResultSet

try (PreparedStatement pst = dbc.getConnection().prepareStatement(query);

ResultSet rs = pst.executeQuery()) {

if (rs.next()) {

// Retrieve the total items count and display it in the JLabel

int totalItems = rs.getInt("TotalItems");

totalItemsLabel.setText(String.format("%d", totalItems));

} else {

// If no result, set the label to "0"

totalItemsLabel.setText("0");

}

}

} catch (SQLException ex) {

// Print an error message if there is an exception

System.out.println("Error while updating total items: " + ex.getMessage());

}

}

// Displays the count of items with a specific status on a JLabel

public static void displayStatus(JLabel totalItemsLabel, String status) {

try {

// Create a new database connection

databaseConnector dbc = new databaseConnector();

// SQL query to count items based on the provided status

String query = "SELECT COUNT(\*) as TotalItems FROM tbl\_items WHERE item\_status = '" + status + "'";

// Prepare and execute the query, store the result in ResultSet

try (PreparedStatement pst = dbc.getConnection().prepareStatement(query);

ResultSet rs = pst.executeQuery()) {

if (rs.next()) {

// Retrieve the count of items with the specified status and display it in the JLabel

int totalItems = rs.getInt("TotalItems");

totalItemsLabel.setText(String.format("%d", totalItems));

} else {

// If no result, set the label to "0"

totalItemsLabel.setText("0");

}

}

} catch (SQLException ex) {

// Print an error message if there is an exception

System.out.println("Error while updating total items: " + ex.getMessage());

}

}

// Displays log entries in a JTable, including setting column widths

public static void logs(JTable table) {

try {

// SQL query to retrieve logs with the account holder who made each log entry

String query = "SELECT "

+ "l.logs\_id as `Logs ID`, " // Log ID

+ "a.first\_name as `Maker`, " // The user who made the log entry

+ "l.logs\_action as `Action`, " // Action description of the log

+ "l.logs\_details as `Details`, " // Detailed description of the log

+ "l.logs\_timestamp as `Timestamp` " // Timestamp of the log entry

+ "FROM tbl\_logs l " // Source table: tbl\_logs

+ "JOIN tbl\_accounts a ON a.account\_id = l.account\_id"; // Join with tbl\_accounts to get user info

// Execute the query and display the results in the provided JTable

display\_query(table, query);

// Set preferred column widths for each column, if available

if (table.getColumnCount() > 0) {

TableColumn column;

column = table.getColumnModel().getColumn(0);

column.setPreferredWidth(20); // Logs ID

column = table.getColumnModel().getColumn(1);

column.setPreferredWidth(20); // Maker

column = table.getColumnModel().getColumn(2);

column.setPreferredWidth(100); // Action

column = table.getColumnModel().getColumn(3);

column.setPreferredWidth(400); // Details

column = table.getColumnModel().getColumn(4);

column.setPreferredWidth(100); // Timestamp

} else {

System.out.println("No columns available to set widths.");

}

} catch (Exception ex) {

// Print an error message if there is an exception

System.out.println("Errors: " + ex.getMessage());

}

}

// Displays the best-selling items in a JTable

public static void best\_item(JTable table) {

// SQL query to fetch items with total\_sold > 0, ordered by highest sales

String query = "SELECT item\_SKU AS `SKU`, "

+ "item\_name AS `Name`, "

+ "total\_sold AS `Sold` "

+ "FROM tbl\_items "

+ "WHERE total\_sold > 0 "

+ "ORDER BY total\_sold DESC";

// Display the query results in the table

display\_query(table, query);

// Set preferred column widths for better table layout

if (table.getColumnCount() > 0) {

TableColumn column;

column = table.getColumnModel().getColumn(0);

column.setPreferredWidth(10); // SKU column

column = table.getColumnModel().getColumn(1);

column.setPreferredWidth(150); // Name column

column = table.getColumnModel().getColumn(2);

column.setPreferredWidth(20); // Sold column

} else {

System.out.println("No columns available to set widths.");

}

}

// Displays items with no sales in a JTable

public static void not\_best\_item(JTable table) {

// SQL query to fetch items with total\_sold < 1, ordered by lowest sales

String query = "SELECT item\_SKU AS `SKU`, "

+ "item\_name AS `Name`, "

+ "total\_sold AS `Sold` "

+ "FROM tbl\_items "

+ "WHERE total\_sold < 1 "

+ "ORDER BY total\_sold ASC";

// Display the query results in the table

display\_query(table, query);

// Set preferred column widths for better table layout

if (table.getColumnCount() > 0) {

TableColumn column;

column = table.getColumnModel().getColumn(0);

column.setPreferredWidth(10); // SKU column

column = table.getColumnModel().getColumn(1);

column.setPreferredWidth(150); // Name column

column = table.getColumnModel().getColumn(2);

column.setPreferredWidth(20); // Sold column

} else {

System.out.println("No columns available to set widths.");

}

}

// Updates the JLabel to display the total sold items

public static int total\_solds;

public static void total\_sold(JLabel total\_sold) {

try {

// Establish database connection

databaseConnector dbc = new databaseConnector();

// SQL query to calculate the sum of total\_sold

String query = "SELECT SUM(total\_sold) as total\_profit FROM tbl\_items WHERE total\_sold > 0";

try (PreparedStatement pst = dbc.getConnection().prepareStatement(query)) {

try (ResultSet rs = pst.executeQuery()) {

if (rs.next()) {

total\_solds = rs.getInt("total\_profit");

// Format and set the label with the total sold items

NumberFormat numberFormat = NumberFormat.getNumberInstance();

String formattedSales = numberFormat.format(total\_solds);

total\_sold.setText(String.format("%s Items", formattedSales));

} else {

total\_sold.setText("0 Items");

}

}

}

} catch (SQLException ex) {

System.out.println("Error: " + ex.getMessage());

}

}

// Updates the JLabel to display total sales in currency format

public static int total\_profits;

public static void total\_sales(JLabel total\_profit) {

try {

// Establish database connection

databaseConnector dbc = new databaseConnector();

// SQL query to calculate total sales by multiplying item price with items sold

String query = "SELECT SUM(item\_price \* total\_sold) as total\_profit FROM tbl\_items WHERE total\_sold > 0";

try (PreparedStatement pst = dbc.getConnection().prepareStatement(query)) {

try (ResultSet rs = pst.executeQuery()) {

if (rs.next()) {

total\_profits = rs.getInt("total\_profit");

// Format and set the label with the total profit in currency format

NumberFormat numberFormat = NumberFormat.getNumberInstance();

String formattedSales = numberFormat.format(total\_profits);

total\_profit.setText(String.format("₱ %s", formattedSales));

} else {

total\_profit.setText("₱ 0");

}

}

}

} catch (SQLException ex) {

System.out.println("Error: " + ex.getMessage());

}

}

// Updates the JLabel to display total expense estimation

public static int total\_expenses;

public static void total\_expense(JLabel total\_expense) {

try {

// Establish database connection

databaseConnector dbc = new databaseConnector();

// SQL query to estimate total expenses as the sum of the item price and stock sold

String query = "SELECT SUM(item\_price) \* SUM(item\_stocks + total\_sold) as total\_profit FROM tbl\_items";

try (PreparedStatement pst = dbc.getConnection().prepareStatement(query)) {

try (ResultSet rs = pst.executeQuery()) {

if (rs.next()) {

total\_expenses = rs.getInt("total\_profit");

// Format and set the label with the total expense in currency format

NumberFormat numberFormat = NumberFormat.getNumberInstance();

String formattedSales = numberFormat.format(total\_expenses);

total\_expense.setText(String.format("₱ %s", formattedSales));

} else {

total\_expense.setText("₱ 0");

}

}

}

} catch (SQLException ex) {

System.out.println("Error: " + ex.getMessage());

}

}

// Sets up a progress bar to show the percentage of profit relative to total expenses

public static void precentage\_bar\_for\_total\_profit(JProgressBar progressBar, JLabel percentage) {

if (total\_expenses == 0) {

progressBar.setValue(0); // No expenses means 0% profit

percentage.setText("0.00%"); // Display 0% for clarity

return;

}

// Calculate profit as a percentage of expenses

double profitPercentage = ((double) total\_profits / total\_expenses) \* 100;

if (profitPercentage < 0) {

profitPercentage = 0; // Avoid negative percentages

} else if (profitPercentage > 100) {

profitPercentage = 100; // Cap at 100% if it exceeds

}

// Update the JProgressBar and JLabel

progressBar.setValue((int) profitPercentage);

progressBar.repaint(); // Refresh the bar visually

percentage.setText(String.format("%.2f%%", profitPercentage)); // Display formatted percentage

}

// Sets up a progress bar to show the percentage of total expenses against a maximum budget

public static void percentage\_bar\_for\_total\_expense(JProgressBar progressBar, JLabel percentage) {

if (total\_expenses == 0) {

progressBar.setValue(0); // No expenses means 0%

percentage.setText("0.00%"); // Display 0% for clarity

return;

}

// Define maximum budget or expense target

double maxExpenses = 100000; // Adjust as needed

double expensePercentage = ((double) total\_expenses / maxExpenses) \* 100;

// Ensure percentage is within bounds

if (expensePercentage < 0) {

expensePercentage = 0;

} else if (expensePercentage > 100) {

expensePercentage = 100;

}

// Update JProgressBar and JLabel

progressBar.setValue((int) expensePercentage);

progressBar.repaint();

percentage.setText(String.format("%.2f%%", expensePercentage));

}

// Sets up a progress bar to show the percentage of total items sold out of total items in stock

public static void percentage\_bar\_for\_total\_items(JProgressBar progressBar, JLabel percentage) {

try {

databaseConnector dbc = new databaseConnector();

// SQL query to calculate total available items (items in stock + items sold)

String query = "SELECT SUM(item\_stocks + total\_sold) as total\_items FROM tbl\_items";

try (PreparedStatement pst = dbc.getConnection().prepareStatement(query); ResultSet rs = pst.executeQuery()) {

if (rs.next()) {

int total\_items = rs.getInt("total\_items");

if (total\_items == 0 || total\_solds == 0) {

progressBar.setValue(0); // No items or no sold items means 0%

percentage.setText("0.00%");

return;

}

// Calculate percentage of items sold out of total items

double soldPercentage = ((double) total\_solds / total\_items) \* 100;

// Ensure percentage is within bounds

if (soldPercentage < 0) {

soldPercentage = 0;

} else if (soldPercentage > 100) {

soldPercentage = 100;

}

// Update the JProgressBar and JLabel

progressBar.setValue((int) soldPercentage);

progressBar.repaint();

percentage.setText(String.format("%.2f%%", soldPercentage));

} else {

percentage.setText("0.00%"); // Handle cases with no result from query

}

}

} catch (SQLException ex) {

System.out.println("Error: " + ex.getMessage());

}

}

// Updates the item status and logs action details

public static void status(String status, String successMessage, String actionType, int get\_item\_id, int admin\_id, JTable dashboard\_table, JTable archive\_table, JTable manage\_table, JLabel total\_items) {

try {

databaseConnector dbc = new databaseConnector();

String sql = "UPDATE tbl\_items SET `item\_status`=? WHERE `item\_SKU`=?";

try (PreparedStatement pst = dbc.getConnection().prepareStatement(sql)) {

// Bind parameters

pst.setString(1, status);

pst.setInt(2, get\_item\_id);

// Execute update and check rows updated

int rowsUpdated = pst.executeUpdate();

if (rowsUpdated > 0) {

// Show success notification

Notifications.getInstance().show(Notifications.Type.SUCCESS, successMessage);

// Refresh display tables and update total items

displayAll(dashboard\_table, manage\_table, archive\_table, total\_items, "available");

// Record log entry

String details = "admin " + admin\_id + " successfully " + actionType + " an item with SKU number " + get\_item\_id + "!";

actionLogs.recordAdminLogs(admin\_id, actionType, details);

} else {

JOptionPane.showMessageDialog(null, "Failed to update item!");

}

}

} catch (SQLException e) {

JOptionPane.showMessageDialog(null, "SQL Error updating data: " + e.getMessage());

e.printStackTrace();

}

}

// Updates account status and logs the action

public static void accounts\_status(String status, int id, JTable table, int selected\_account) {

try {

databaseConnector dbc = new databaseConnector();

String sql = "UPDATE tbl\_accounts SET `status`=? WHERE `account\_id`=?";

try (PreparedStatement pst = dbc.getConnection().prepareStatement(sql)) {

// Bind parameters

pst.setString(1, status);

pst.setInt(2, selected\_account);

// Execute update and check rows updated

int rowsUpdated = pst.executeUpdate();

if (rowsUpdated > 0) {

// Show success notification

Notifications.getInstance().show(Notifications.Type.SUCCESS, "Successfully updated account information");

// Refresh account table

display\_items.accounts(table);

// Record log entry

String details = "admin " + id + " successfully updated account " + selected\_account + "!";

actionLogs.recordAdminLogs(id, "Update account", details);

} else {

JOptionPane.showMessageDialog(null, "Failed to update account!");

}

}

} catch (SQLException e) {

JOptionPane.showMessageDialog(null, "SQL Error updating data: " + e.getMessage());

e.printStackTrace();

}

}

// Class to handle chart display in the application

public class display\_chart {

// Pie chart objects for displaying different pie charts

private static PieChart pieChart1;

private static PieChart pieChart2;

private static PieChart pieChart3;

// Line chart object for displaying line chart

private static LineChart lineChart;

// Horizontal bar chart objects for displaying bar charts

private static HorizontalBarChart barChart1;

private static HorizontalBarChart barChart2;

// Method to create and display the first pie chart in a specified panel

public static void createPieChart1(JPanel panel) {

// Initialize PieChart object

pieChart1 = new PieChart();

// Set header label for the pie chart

JLabel header1 = new JLabel("Item Income");

header1.putClientProperty(FlatClientProperties.STYLE, "font:Arial bold +2");

// Attach header to pie chart

pieChart1.setHeader(header1);

// Add color scheme to pie chart sections

pieChart1.getChartColor().addColor(

Color.decode("#f87171"), // Red

Color.decode("#fb923c"), // Orange

Color.decode("#fbbf24"), // Yellow

Color.decode("#a3e635"), // Light green

Color.decode("#34d399"), // Green

Color.decode("#22d3ee"), // Cyan

Color.decode("#818cf8"), // Blue

Color.decode("#c084fc") // Purple

);

// Set pie chart background to null and add border styling

pieChart1.putClientProperty(FlatClientProperties.STYLE, "background:null");

pieChart1.putClientProperty(FlatClientProperties.STYLE, "border:5,5,5,5,$Component.borderColor,,20");

// Set dataset for the pie chart

pieChart1.setDataset(createPieData());

// Add pie chart to panel and update panel display

panel.removeAll();

panel.add(pieChart1);

panel.validate();

panel.repaint();

}

// Similar method to create and display the second pie chart

public static void createPieChart2(JPanel panel) {

pieChart2 = new PieChart();

JLabel header1 = new JLabel("Item Cost");

header1.putClientProperty(FlatClientProperties.STYLE, "font:Arial bold +2");

pieChart2.setHeader(header1);

pieChart2.getChartColor().addColor(

Color.decode("#f87171"), // Red

Color.decode("#fb923c"), // Orange

Color.decode("#fbbf24"), // Yellow

Color.decode("#a3e635"), // Light green

Color.decode("#34d399"), // Green

Color.decode("#22d3ee"), // Cyan

Color.decode("#818cf8"), // Blue

Color.decode("#c084fc") // Purple

);

pieChart2.putClientProperty(FlatClientProperties.STYLE, "background:null");

pieChart2.putClientProperty(FlatClientProperties.STYLE, "border:5,5,5,5,$Component.borderColor,,20");

pieChart2.setDataset(createPieData());

panel.removeAll();

panel.add(pieChart2);

panel.validate();

panel.repaint();

}

// Similar method to create and display the third pie chart, with DONUT style

public static void createPieChart3(JPanel panel) {

pieChart3 = new PieChart();

JLabel header1 = new JLabel("Item Profit");

header1.putClientProperty(FlatClientProperties.STYLE, "font:Arial bold +2");

pieChart3.setHeader(header1);

pieChart3.getChartColor().addColor(

Color.decode("#f87171"), // Red

Color.decode("#fb923c"), // Orange

Color.decode("#fbbf24"), // Yellow

Color.decode("#a3e635"), // Light green

Color.decode("#34d399"), // Green

Color.decode("#22d3ee"), // Cyan

Color.decode("#818cf8"), // Blue

Color.decode("#c084fc") // Purple

);

pieChart3.putClientProperty(FlatClientProperties.STYLE, "background:null");

pieChart3.putClientProperty(FlatClientProperties.STYLE, "border:5,5,5,5,$Component.borderColor,,20");

// Set chart to donut type

pieChart3.setChartType(PieChart.ChartType.DONUT\_CHART);

pieChart3.setDataset(createPieData());

panel.removeAll();

panel.add(pieChart3);

panel.validate();

panel.repaint();

}

// Creates dataset for pie charts with sample item categories and random values

public static DefaultPieDataset createPieData() {

DefaultPieDataset<String> dataset = new DefaultPieDataset<>();

Random random = new Random();

// Adds entries with random values

dataset.addValue("Classic T-shirt", random.nextInt(100) + 50);

dataset.addValue("Denim", random.nextInt(100) + 50);

dataset.addValue("Jacket", random.nextInt(100) + 50);

dataset.addValue("Pakigol", random.nextInt(100) + 50);

dataset.addValue("Panty", random.nextInt(100) + 50);

return dataset;

}

// Method to create and display a line chart

public static void createLineChart(JPanel panel) {

lineChart = new LineChart();

// Sets line chart type to CURVE

lineChart.setChartType(LineChart.ChartType.CURVE);

// Sets chart border style

lineChart.putClientProperty(FlatClientProperties.STYLE, "border:5,5,5,5,$Component.borderColor,,20");

// Generates data for line chart

createLineChartData();

// Adds line chart to panel and updates display

panel.removeAll();

panel.add(lineChart);

panel.validate();

panel.repaint();

}

// Generates dataset for line chart with income, expense, and profit data

public static void createLineChartData() {

DefaultCategoryDataset<String, String> categoryDataset = new DefaultCategoryDataset<>();

NumberFormat currencyFormat = NumberFormat.getCurrencyInstance(new Locale("en", "PH"));

Calendar cal = Calendar.getInstance();

SimpleDateFormat df = new SimpleDateFormat("MMM dd, yyyy");

int[] incomeData = {500, 600, 450, 700, 620, 580, 610};

int[] expenseData = {300, 400, 350, 500, 450, 420, 390};

int[] profitData = {200, 200, 100, 200, 170, 160, 220};

// Loop to add values to the dataset

for (int i = 0; i < incomeData.length; i++) {

String date = df.format(cal.getTime());

categoryDataset.addValue(incomeData[i], "Income", date);

categoryDataset.addValue(expenseData[i], "Expense", date);

categoryDataset.addValue(profitData[i], "Profit", date);

// Move calendar to the next date

cal.add(Calendar.DATE, 1);

}

lineChart.setCategoryDataset(categoryDataset);

lineChart.getChartColor().addColor(Color.decode("#38bdf8"), Color.decode("#fb7185"), Color.decode("#34d399"));

}

// Method to create and display bar charts for income and expense

public static void createBarChart(JPanel panel1, JPanel panel2) {

// First bar chart for monthly income

barChart1 = new HorizontalBarChart();

JLabel header1 = new JLabel("Monthly Income");

header1.putClientProperty(FlatClientProperties.STYLE, "font:Arial bold +2");

barChart1.setHeader(header1);

barChart1.setBarColor(Color.decode("#f97316"));

barChart1.setDataset(createData());

// Add first bar chart to panel1

panel1.putClientProperty(FlatClientProperties.STYLE, "border:5,5,5,5,$Component.borderColor,,20");

panel1.add(barChart1);

panel1.removeAll();

panel1.add(barChart1);

panel1.validate();

panel1.repaint();

// Second bar chart for monthly expense

barChart2 = new HorizontalBarChart();

JLabel header2 = new JLabel("Monthly Expense");

header2.putClientProperty(FlatClientProperties.STYLE, "font:Arial bold +2");

barChart2.setHeader(header2);

barChart2.setBarColor(Color.decode("#10b981"));

barChart2.setDataset(createData());

// Add second bar chart to panel2

panel2.putClientProperty(FlatClientProperties.STYLE, "border:5,5,5,5,$Component.borderColor,,20");

panel2.add(barChart2);

panel2.removeAll();

panel2.add(barChart2);

panel2.validate();

panel2.repaint();

}

Appendix E – Software and Hardware Specifications

The Queenz Inventory System requires specific hardware and software components for optimal performance. These specifications are generally outlined based on the needs of the inventory system's complexity, database handling, and scalability. Below is an outline of common hardware and software specifications required for an inventory management system like Queenz:

**Hardware Specifications:**

• Processor: Intel i3 10th gen / AMD Ryzen 3 3200G, or equivalent (Quad-core or better)

• RAM: Minimum 8GB DDR4 RAM • Storage: SSD storage with at least 256GB capacity (scalable based on the database size)

• Network Interface: 10Mbps network adapter or higher for fast communication

• Backup: External backup drive or cloud-based backup support

• Display: Standard Full HD (1920x1080) resolution

• Wireless connectivity (Wi-Fi or 4G/5G)

**Software Specifications:**

• Operating System: • Client: Windows 10/11

• Database Management System (DBMS):

• MySQL for handling inventory records

• Programming Languages:

• Backend: Java

• Frontend: Java Maven Flatflaf

• Security Software:

• Firewalls and anti-virus software for malware protection

Appendix G – Gantt chart (Schedule)