BIS 698 – Information Systems Project

Asset Management System

Project Report

By Group 1:

Sri Charan Ravva
Thrivikram Kotharu
Ayushi Tripathi
Sai Abhiram Koneru
Chinnu Muraoka

Instructor: Prof. Wilfred Owobu

Central Michigan University

Table of Contents

1. Background

Central Michigan University Student Activity Centre is a crowded building where students attend to play, exercise, and engage in group activities. On its list are the facilities, one of which is a gym that has some sport and recreation equipment, such as basketballs, footballs, volleyballs, and other equipment that can be checked out by students for personal or organized purposes. Nowadays, it is completed by hand and with return of gear in paper logs or word-of-mouth sign-out. Although years of use with such an ad hoc system by the centre do exist, such is unable to keep up with demand and usage by students as they grow. As more students take the equipment out of the gym, it becomes hard for staff to know which units are loaned, who loaned them, and whether they have been returned in good working condition.

Such inefficiency has led to prevalent issues like lost equipment, mysterious damages, and slow check-out processes. Students are made to wait during equipment check-out, whereas staff members get to spend hours on redundant paperwork instead of utilizing those resources on more valuable endeavours which can propel the student experience forward. Additionally, without computer monitoring of equipment usage, the centre is not better able to provide meaningful statistics to inform budgeting, identify trends in student usage, or make suggestions regarding new or replacement equipment to university administration. These are problems that call for a contemporary, centralized system. A specifically designed asset management program for the Student Activity Centre would not only streamline equipment tracking but also improve accountability, reduce losses, and provide valuable insights data in assisting with day-to-day operations as well as long-term planning.

2. Business Problem

The sheer number of daily checkouts to dozens of students and hundreds of groups generates perpetual scheduling conflict, lost materials, and damage. Because it is impossible to account for a broken-up game system or lost projector without having a single, unifying point, unbudgeted replacement expenses occur continually. Staff spend a significant amount of time tediously completing forms manually and going around searching for equipment rather than working with students. This adds to waiting times and frustration for administrators when trying to organize events, taking away from the original purpose of keeping campus life active. The personnel have no idea how the assets are utilized, demand over time, or condition. It cannot enable them to justify in the budget proposals for the equipment, plan for the maintenance, or deplete the inventory levels depending on what they are utilized by the students, resulting in wasteful expenditure and inefficient resource utilization.

3. Project Description

The Asset Management System for the CMU Student Activity Center is developed to replace the present manual, paper-based process of tracking sports and recreational equipment. Currently, employees manually record check-in and check-out information, which frequently results in mistakes, hold-ups, and accountability problems. All assets may be registered, tracked, and

controlled in real time on the unified digital platform that our suggested system offers. Digital equipment check-in and check-out, role-based staff access, maintenance history monitoring, and straightforward asset availability and usage reporting are among the essential features. The technology ensures seamless daily operations by automating asset tracking**.**

System Purpose:

The purpose of this system is to provide a centralized, effective, and user-friendly interface for organizational asset management. It allows users to track check-in and check-out processes, track asset data, and confirm availability. The solution converts asset management into digital form so that it provides accurate data, increases accountability, and reduces human mistakes. This system provides safe storage of information, efficient operations, and quick access to asset data using the backend technology of MySQL, and Tainter as the desktop GUI. The overall aims are to provide maximum transparency, utilize available resources optimally, and support informed decision-making on organizational assets.

Core functionalities

User Authentication and Access

- Users can log in and log out securely. New administrators can register for an account.
- Change password options are available.
- Different access levels for Admins, Users/Staff.

2. **User and Role Management**

- View lists of all registered users, admins, and managers
- Add new users, admins, or managers through the system

3. **Dashboard**

- Monthly calendar display
- Quick view of asset status: broken, under repair, available, and missing
- A checkout summary table shows asset ID, description, checkout date, return date, and who the asset is assigned to

4. **Asset Management**

- Table view of all assets showing asset ID, description, status, site, location, and assigned person The system will display all assets in a table that includes details such as the asset ID, description, current status, site, location, and person it is assigned to.
- New assets can be added by filling out information like asset ID, description, a photo, site, and location. The asset list can also be downloaded or exported when needed.
- Clicking on an asset description will bring up a pop-up with more detailed information. Assets can be flagged as “under maintenance.”

5. **Check-Out/ Check-In Processes**

- Users can select one or more assets from a list to check out. Selected assets are added to a checkout list. During checkout, details such as the checkout date, return date, site, and location need to be entered. The asset must be assigned to a person by entering their name, employee ID, title, phone number, and email
- Once assigned, that person is automatically added to the system's user list. A table will then show the checked-out assets with their status, site, location, and the person responsible. Select assets to check back in.

6. **Lists Functionality**

- View a complete list of all assets and their availability, under maintenance.
- View a list of assets that have been disposed (kept as archived records).

7. **Reports & Analytics**

Visual reports showing asset usage trends, Identify the most frequently used assets, Reports on which users check out assets the most, Option to filter and export reports.

8. **Account Settings**

Users can change their password, Update personal information such as phone number, email, or title. And Set preferences for system notifications.

Out-of-Scope Features:

- Mobile Application (iOS/Android): Mobile application for asset tracking and check-in/check-out will not be included in this project.
- QR/Barcode Integration: There will be no scanning technology in this stage for automated asset tracking.
- High-End Security Features: There is no biometric authentication, enterprise-grade encryption in this system.
- Support for Multiple Languages: As of now, this system will be support English.

4. Project Feasibility:

Economic Feasibility:

The total development expense is estimated to be around $8,264 (5 students × 20 hrs/week × 8 weeks × $10.33/hr). The Student Activity Centre spends an average of $1,000–$5,000 yearly in insurance claims for stolen or misplaced equipment. The Asset Management System would be able to recover a huge portion of this cost every year by reducing these losses.

ROI (low savings $1,000): –87.9%
ROI (average savings $3,000): –63.7%
ROI (high savings $5,000): –39.5%

Technical Feasibility:

The system will make use of free programs like Python, MySQL, and Tkinter that are compatible with the PCs already in the Student Activity Center. Neither expensive software nor new hardware are required. Because the project team is capable of building and maintaining the system, there is little technical risk.

Schedule Feasibility:

The project can be completed in 8 weeks, with each phase carefully planned:

- Weeks 1–2: Requirements and database design
- Weeks 3–5: Python + MySQL development
- Weeks 6–7: Tkinter interface and integration
- Week 8: Testing and final deployment

Operational Feasibility:

The system will feature a simple-to-use Tkinter interface with menus that are easy to comprehend for borrowing, returning. staff and volunteers would need just a short training session. The system reduces paperwork, speeds up check-outs.

Student Developer	Hourly Wage ($)	Hours/Week	Weeks	Total Pay ($)
Student 1	10.33	20	8	1,652.80
Student 2	10.33	20	8	1,652.80
Student 3	10.33	20	8	1,652.80
Student 4	10.33	20	8	1,652.80
Student 5	10.33	20	8	1,652.80
Total	-	-	-	**8,264.00**

5. Use Cases:

User Registration

Use Case Name:	**User Registration**	**ID:** UC-0	**Priority: High**
Actor:	User (Staff)		
Description:	A new staff user created an account to access the Asset Management System. Admins cannot register themselves; only Staff can use the Sign-Up form.		
Trigger:	User clicks the "Sign Up" button on the landing page.		
Type:	External		
Preconditions:	1. The system is online and accessible. 2. Users have valid registration details. 3. Admin role cannot be selected through registration (system forces User role).		
Normal Course:	1. Staff opens the Asset Management System. 2. The system loads the Landing Page. 3. Staff clicks the "Sign Up" button on the landing page. 4. System redirects Staff to the Registration Page. 5. System displays a registration form with fields (Email, First Name, Last Name, Password, Re-type Password). 6. Staff clicks into the Email field and enters a valid email address. 7. Staff clicks into the First Name field and enters their First Name. 8. Staff clicks into the Last Name field and enters their Last Name. 9. Staff clicks into the Password field and enters a valid password. 10. Staff clicks into the Re-type password field and enters the password that matches the field above. 11. Staff reviews all entered information. 12. Staff clicks the "Submit" button. 13. System performs field validation (mandatory fields, unique email, password rules). 14. If validation succeeds, the system saves the new user record into the User database. 15. System displays a “User Created” pop-up. 16. Staff click the ‘Ok’ button and is redirected to the ‘Log In’ Page.		
Postconditions:	1. New user account is stored in User database. 2. User account created with Staff role. 3. Staff can now log in.		

User Login

Use Case Name:	**User Login**	**ID:** UC-1A	**Priority: High**
Actor:	Staff		

Description:	A Staff securely logs into the Asset Management System using their credentials.
Trigger:	Staff logs into the system using registered credentials either through redirection or by clicking on the ‘Log In’ button on the landing page.
Type:	External
Preconditions:	1. The system is online and accessible. 2. The user has a registered account with valid credentials.
Normal Course:	1. Staff navigate to the landing page. 2. Staff click on ‘Log In’. 3. System displays fields for email and password. 4. Staff click Email field and enter their email. 5. Staff click Password field and enter password. 6. Staff clicks the “Login” button. 7. System checks fields for empty values. 8. System validates credentials in User database. 9. System confirms that role = Staff. a. If valid → proceed to Step 10. b. If invalid → display error message and return to Step 2. 10. System grants access and directs Staff to their assigned dashboard with limited menu options.
Postconditions:	1. Staff sessions are created and stored in the system. 2. Staff gain access to the system’s functionalities according to their role.

User Login

Use Case Name:	**User Login**	**ID:** UC-1A	**Priority: High**
Actor:	Admin		
Description:	A Staff securely logs into the Asset Management System using their credentials.		
Trigger:	Staff logs into the system using registered credentials either through redirection or by clicking on the ‘Log In’ button on the landing page.		
Type:	External		
Preconditions:	1. The system is online and accessible. 2. The user has a registered account with valid credentials.		
Normal Course:	1. Staff navigate to the landing page. 2. Staff click on ‘Log In’. 3. System displays fields for email and password. 4. Staff click Email field and enter their email. 5. Staff click Password field and enter password. 6. Staff clicks the “Login” button. 7. System checks fields for empty values. 8. System validates credentials in User database.		

	9. System confirms that role = Staff. a. If valid → proceed to Step 10. b. If invalid → display error message and return to Step 3. 10. System grants access and directs Staff to their assigned dashboard with limited menu options.
Postconditions:	1. Staff sessions are created and stored in the system. 2. Staff gain access to the system's functionalities according to their role.

User Login

Use Case Name:	**User Login**	**ID:** UC-1B	**Priority: High**
Actor:	Admin		
Description:	Admin logs into the system. Admin accounts cannot be created via registration; only assigned by existing Admin.		
Trigger:	Admin clicks "Login" button on the landing page.		
Type:	External		
Preconditions:	1. The system is online and accessible. 2. The admin valid credentials. 3. Admin role is assigned by another Admin.		
Normal Course:	1. Admin navigates to the landing page. 2. Admin click on 'Log In'. 3. Admin displays fields for email and password. 4. Admin clicks Email field and enter their email. 5. Admin clicks Password field and enter password. 6. Admin clicks the "Login" button. 7. System checks fields for empty values. 8. System validates credentials. 9. System confirms that role = Admin. a. If valid → proceed to Step 10. b. If invalid → display error message and return to Step 3. 10. System grants access and directs Admin to their assigned dashboard.		
Postconditions:	1. Admin sessions are created and stored in the system. 2. Admin gain access to the system's functionalities according to their role.		

Add New Asset

Use Case Name:	**Add New Asset**	**ID:** UC-2	**Priority: High**
Actor:	Admin, Staff		
Description:	Admin or Staff adds a new asset with details such as name, category, and quantity.		

Trigger:	Admin/Staff chooses to register a new asset in the List of Assets screen. (e.g., click the "Add New Asset" button).
Type:	External
Preconditions:	1. Admin/Staff is logged into the system. 2. Asset details are available to be entered.
Normal Course:	1. Admin/Staff selects 'Add Asset' option from the List of Assets screen. 2. System displays add new asset pop-up with fields (Asset Name, Category, Quantity). 3. Admin/Staff enters asset information. 4. Admin/Staff clicks "Save Asset". 5. System validates input. a. If valid → proceed to Step 6. b. If invalid → show error and return to Step 3. 6. System saves asset details in Asset datastore. 7. System confirms successful addition of new asset.
Postconditions:	1. Asset record is stored in the system database. 2. New assets appear in the list of Asset view.

Check-Out Asset

Use Case Name:	**Check-Out Asset**	**ID:** UC-3	**Priority:** High
Actor:	Admin, Staff		
Description:	A user processes a request for a student/staff to check out one or more available assets, and the system records the transaction.		
Trigger:	User selects Check-Out option on the sidebar of the user dashboard.		
Type:	External		
Preconditions:	1. Admin/Staff is logged into the system. 2. Student will walk into the gym and request to check-out an asset. 3. The asset is available to Check-out (i.e. Status = Available)		
Normal Course:	1. Admin/Staff navigates to the "Check-Out" screen. 2. Admin/Staff enters the Student ID to search for the borrower's entry in the system. 3. System displays the student's name, student’s status (active or inactive). 4. Admin/Staff selects the asset(s) and quantity to be checked out from the list. 5. Admin/Staff clicks "Check Out Selected". 6. System verifies asset availability using the Assets store. 7. System updates the asset Status to 'Checked Out'.		

	8. System records the transaction in the Transactions data store, linking the asset and borrower (Student ID) with a Clock-Out Time. 9. System returns confirmation pop up to the Admin/Staff with a success message stating the student's name and Asset ID checked-out.
Postconditions:	1. Asset status is updated to 'Checked Out'. 2. The inventory will be updated. 3. The Transaction record is created.

Check-In Asset

Use Case Name:	**Check-In Asset**	**ID:** UC-4	**Priority: High**
Actor:	Admin, Staff		
Description:	The system processes the return of previously checked-out assets by a student.		
Trigger:	Admin/Staff initiates return of an asset by clicking on the 'Check-In Asset' option on the dashboard sidebar.		
Type:	External		
Preconditions:	1. Staff/Admin is logged into the system. 2. Student will walk into the gym and request to check-in an asset. 3. The asset is currently 'Clocked Out'.		
Normal Course:	1. Admin/Staff navigates to the "Check-In" screen by clicking on the "Check-In Asset" option on the sidebar. 2. Admin/Staff asks the student for their ID. 3. Admin/Staff enters the Student ID and verifies their credentials. 4. System displays the list of assets, asset tag ID and the count of assets currently held by the student. 5. Admin/Staff selects the asset(s) to be checked in. 6. Admin/Staff changes the status of all the assets to be checked-in to 'Available' from 'Checked-out'. 6. Admin/Staff clicks "Check In Selected". 7. System updates the asset Status to 'Available' in Assets. 8. System updates the corresponding Transaction record with the Clock-In Time. 9. System returns confirmation to the Admin/Staff with a success message in green.		
Postconditions:	1.Assert will be marked as check-in. 2. Then in the inventory, the assert number will be updated.		

Generate Asset Report

Use Case Name:	**Generate Asset Report**	UC-5	**Medium**
Actor:	Admin		
Description:	Admin generates reports about transaction history, asset details, asset usage, and user activity.		
Trigger:	Admin requests a report by clicking on “Reports” option on the sidebar of the admin dashboard		
Type:	External		
Preconditions:	1. User is authenticated as Admin. 2. Asset and transaction data exist in the system.		
Normal Course:	1. Admin selects "Reports" option. 2. System displays report filters (From/To date range). 3. Admin/Manager selects filters and clicks "Search". 4. System retrieves relevant data from Assets and Transactions. 5. System generates and displays the report output (e.g., transaction list). 6. Admin may click "Export to Excel" to download the above report. 7. If exported, the system generates the report file in .xls format. 8. Admin may also download an analytical asset detailed report directly by clicking on the “Asset Report” button on the right side of the screen.		
Postconditions:	1. Report is displayed on screen. 2. Report is stored or exported if requested.		

Manage User Accounts

Use Case Name:	**Manage User Accounts**	**ID:** UC-6	**Priority: High**
Actor:	Admin		
Description:	Admin manages user accounts by adding, updating, or assigning roles.		
Trigger:	Admin initiates User management action.		
Type:	External		
Preconditions:	1. Admin is logged into the system. 2. Admin has the required permissions.		
Normal Course:	1. Admin selects 'User Management' from the dashboard sidebar. 2. System displays list of existing users, their status and role. 3. Admin chooses an action (e.g., changing a user's role). 4. Admin updates user information (e.g., changes Role from 'User' to 'Admin'). 5. Admin clicks "Save".		

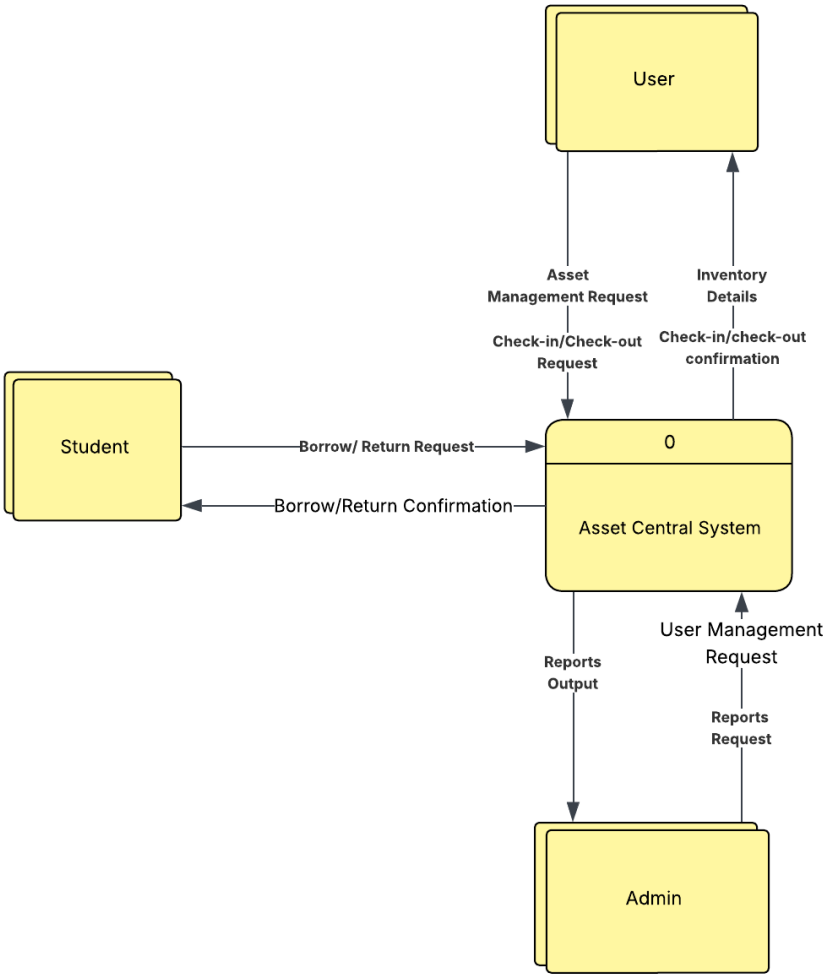
	6. System validates inputs. 7. System saves changes to the User data store. 8. System confirms update.
Postconditions:	1. User database is updated with changes. 2. Updated access rights are effective immediately.

Update Account Settings

Use Case Name:	**Update Account Settings**	**ID:** UC-7	**Priority: Low**
Actor:	Admin, Staff		
Description:	A user updates their personal details, password, or notification preferences.		
Trigger:	User selects 'Account Settings'.		
Type:	External		
Preconditions:	1. Admin/Staff is authenticated and logged in.		
Normal Course:	1. Admin/Staff navigates to 'Account Settings' from dashboard sidebar. 2. System displays editable fields (change password, first name, last name). 3. Admin/Staff updates information. 4. System validates changes. 5. System saves new information to User. 6. System confirms update.		
Postconditions:	1. Updated information is stored in the system. 2. Changes take effect immediately.		

6. DFD Diagrams

6.1 Context Level DFD



Context Level DFD

The Context-Level Data Flow Diagram (DFD Level-0) provides a high-level overview of the *Asset Central System* and illustrates how external entities interact with the system. At this level, the entire application is represented as a single process, and only the major data flows between the system and external actors are shown.

In this system, three primary external entities interact with Asset Central: Students, Users, and Admins. Students initiate Borrow/Return Requests when they want to take or return sports equipment. The system processes these requests and sends back a Borrow/Return Confirmation, ensuring that inventory and transaction records remain accurate.

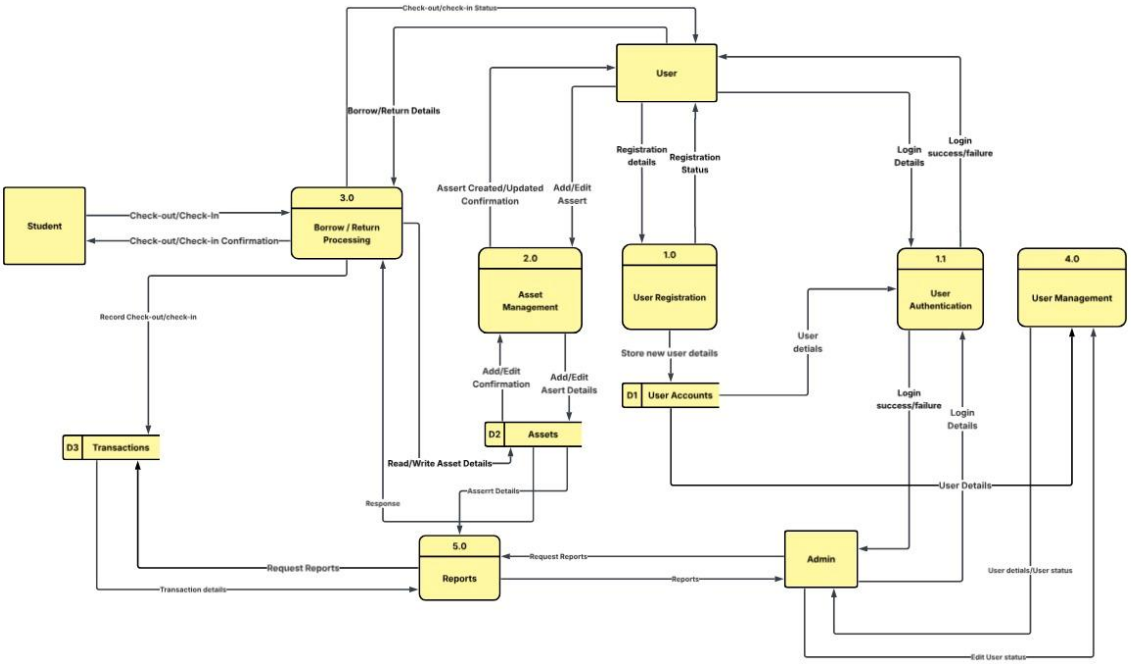
Users, which typically include staff members responsible for managing day-to-day operations, communicate with the system through two main functions: Asset Management and Check-in/Check-out. Users submit Asset Management Requests to add, update, or manage asset information. They also perform Check-in/Check-out Requests when assets are issued or returned

on behalf of students. In response, the system provides updated Inventory Details and Check-in/Check-out Confirmations, supporting smooth operational workflows.

Admins interact with the system at a supervisory level. They submit User Management Requests to create, modify, or manage user accounts within the application. Additionally, admins can send Report Requests to retrieve system-generated reports. In return, the system provides Reports Output containing summarized or detailed information about assets, usage history, and user activity to support decision-making and oversight.

Overall, the context diagram demonstrates how the Asset Central System integrates student borrowing, staff operations, and administrative oversight within a unified platform. This high-level model clearly outlines the boundaries of the system and highlights the essential flow of information between external stakeholders and the system.

6.2 Zero Level



Zero Level Diagram- Lucid Chart

The Level-0 Data decomposes the Asset Central System into its major internal processes and illustrates how data moves between system components, external entities, and data stores. This level of detail shows the internal functionality behind the single Process 0 from the context diagram, including user registration, authentication, asset management, borrowing/return processing, user administration, and reporting.

Process 1.0 – User Registration

This process allows new users to register for access to the system. When a user submits their registration details, the process validates the information and stores the new user record in the User Accounts data store (D1). The system then returns a registration status message to the user. Only regular users can register themselves; admin accounts are not created through this process.

Process 1.1 – User Authentication

This process handles login requests from both Users and Admins. It receives login credentials, verifies them against the stored records in User Accounts (D1), and returns either a successful login response or an error message. Once authenticated, the user's role (User or Admin) determines their access to other processes in the system.

Process 2.0 – Asset Management

This process manages the creation, modification, and viewing of asset information. Users can add new assets or edit existing records, and these updates are written to the Assets data store (D2). The process may also fetch asset details from D2 to display updated inventory information to the user and to support other processes such as borrowing and reporting. Process 3.0 – Borrow / Return Processing This process handles the operational workflow in which a student borrows (checks out) or returns (checks in) an asset. When a staff user enters borrow/return details, the system verifies asset availability using the Assets store (D2), updates the asset status, and records the transaction in the Transactions data store (D3). The process returns confirmation messages to both the student and the staff user. This process captures both the physical movement of assets and the audit trail for accountability.

Process 4.0 – User Management (Admin)

This process allows administrators to manage user accounts within the system. Admins can update user information, activate or deactivate users, and change user roles (such as promoting a regular user to admin). These updates are written back to the User Accounts data store (D1), and confirmation responses are returned to the admin.

Process 5.0 – Reports

This process generates asset reports, transaction summaries, usage statistics, and other analytical outputs for Users and Admins. Upon receiving a report request, the process retrieves relevant data from both the Assets (D2) and Transactions (D3) data stores, formats the results, and delivers the report to the requesting entity. Reports support decision-making for asset availability, maintenance needs, and usage trends.

External Entities

- Student (E1): Initiates check-out and check-in requests and receives confirmations.

- User (E2): Performs registration, login, asset management activities, and handles borrow/return operations.
- Admin (E3): Logs in, manages system users, and requests reports. Overall, the Level-0 DFD clearly illustrates how the system processes user actions, manages asset data, records transactions, and supports administrative oversight through structured data flows and well-defined processes.

7. Task List:

Task Mode	Task Name	Duration	Start	Finish	Predecessors
	Phase 1: Database design	**10 days**	**Mon 9/29/25**	**Fri 10/10/25**	
	Create ER diagram	5 days	Mon 9/29/25	Fri 10/3/25	
	Create users/employees, List of assets, List of assets,	3 days	Fri 10/3/25	Tue 10/7/25	
	Insert sample data and testing.	4 days	Tue 10/7/25	Fri 10/10/25	
	Phase 2: Backend Development	**11 days**	**Fri 10/10/25**	**Fri 10/24/25**	**1**
	User Authentication & Access	**3 days**	**Fri 10/10/25**	**Tue 10/14/25**	**4**
	User & Role Management	**4 days**	**Tue 10/14/25**	**Fri 10/17/25**	**4**
	Dashboard	**2 days**	**Thu 10/16/25**	**Fri 10/17/25**	**4**
	Asset Management	**2 days**	**Fri 10/17/25**	**Mon 10/20/25**	
	Check-In / Check-Out	**4 days**	**Fri 10/17/25**	**Wed 10/22/25**	
	Lists Functionality	**2 days**	**Wed 10/22/25**	**Thu 10/23/25**	
	Reports & Analytics	**2 days**	**Wed 10/22/25**	**Thu 10/23/25**	
	Account Settings	**2 days**	**Thu 10/23/25**	**Fri 10/24/25**	
	Phase 3: Frontend development (User Interface)	**10 days**	**Fri 10/24/25**	**Thu 11/6/25**	**38**
	User Authentication	**2 days**	**Fri 10/24/25**	**Mon 10/27/25**	**38**
	User & Role Management	**2 days**	**Mon 10/27/25**	**Tue 10/28/25**	**38**
	Dashboard	**1 day**	Tue 10/28/25	**Tue 10/28/25**	**38**
	Asset Management	**3 days**	**Tue 10/28/25**	**Thu 10/30/25**	**38**
	Check-In / Check-Out	**2 days**	**Thu 10/30/25**	**Fri 10/31/25**	**38**
	Lists Functionality	**2 days**	**Fri 10/31/25**	**Mon 11/3/25**	**38**
	Reports & Analytics	**2 days**	**Mon 11/3/25**	**Tue 11/4/25**	**38**
	Account Settings	**2 days**	**Tue 11/4/25**	**Wed 11/5/25**	**38**
	Phase 4: Integration, Testing & Deployment	**22 days?**	**Thu 11/6/25**	**Fri 12/5/25**	**75**
	Integration Tasks	**7 days?**	**Thu 11/6/25**	**Fri 11/14/25**	**75**
	Testing Tasks	**7 days**	**Fri 11/14/25**	**Mon 11/24/25**	**75**
	Deployment Tasks	**7 days**	**Mon 11/24/25**	**Tue 12/2/25**	**75**

Phase 1: Design of the Database (10 days)
Create the ER diagram to show the relationships and entities.
Make tables for assets, staff, and users.
Add sample data and use test runs to verify the database.

Phase 2: Development of the Backend (11 days)
Put access control and user authentication into practice.
Create asset management, dashboard, and role management modules.
Include lists and check-in/check-out features.
Create features for account settings, analytics, and reports.

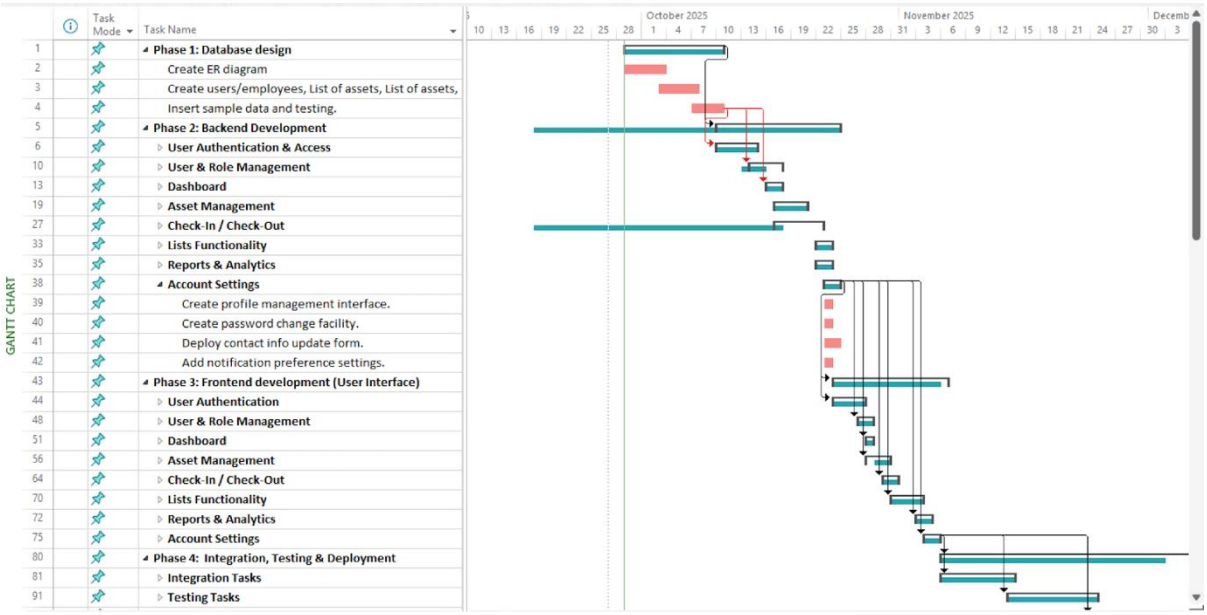
Phase 3: Development of the Frontend (10 days)
Create dashboard, role management, and authentication user interfaces.
Create a user-friendly interface for asset management and check-in/check-out.

Make lists, reports, analytics, and account settings pages accessible to users.

Phase 4: Deployment, Testing, and Integration (22 days)
Combine the backend and frontend modules.
Perform user acceptability, performance, and functional testing.
Set up the system for use in production.

8. Critical path model:

The Critical Path is the longest sequence of dependent tasks that determines the total project duration. If any critical task is delayed, the entire project is delayed. Tasks on the Critical Path are shown in red in MS Project. Non-critical tasks have float/slack, meaning they can shift without affecting the end date.



Schedule & Critical Path

Phase 1: Database Design (Sep 29 – Oct 10)

- ER diagram → MySQL schema → sample/seed data.
- Critical: unlocks all backend work.

Phase 2: Backend Development (Oct 10 – Oct 24)

- Authentication (login/reset/roles), Asset module, Check-in/Check-out, Reports, Account Settings.
- **Critical focus:** Auth, Asset, Account Settings (drive UI work).

Phase 3: Frontend UI (Oct 24 – Nov 6)

- Tkinter screens for Auth, Dashboard, Assets, Reports.
- Depends on corresponding backend endpoints.
- Critical screens: Auth & Assets (feed Integration).

Phase 4: Integration, Testing & Deployment (Nov 6 – Dec 2)

- Integrate all modules → Unit/Integration/System testing → UAT → Deployment & docs.
- **Critical:** Integration → Testing → Deployment chain.

Critical Path (0 slack):
Database Design → Key Backend (Auth/Asset/Account Settings) → Tkinter UI (Auth/Assets) → Integration → Testing → Deployment.
(Tasks on this chain are tracked in **red** in MS Project.)

Milestones:

Project Start (Sep 29)

Phase-1 Complete (Oct 10)

Phase-2 Complete (Oct 24)

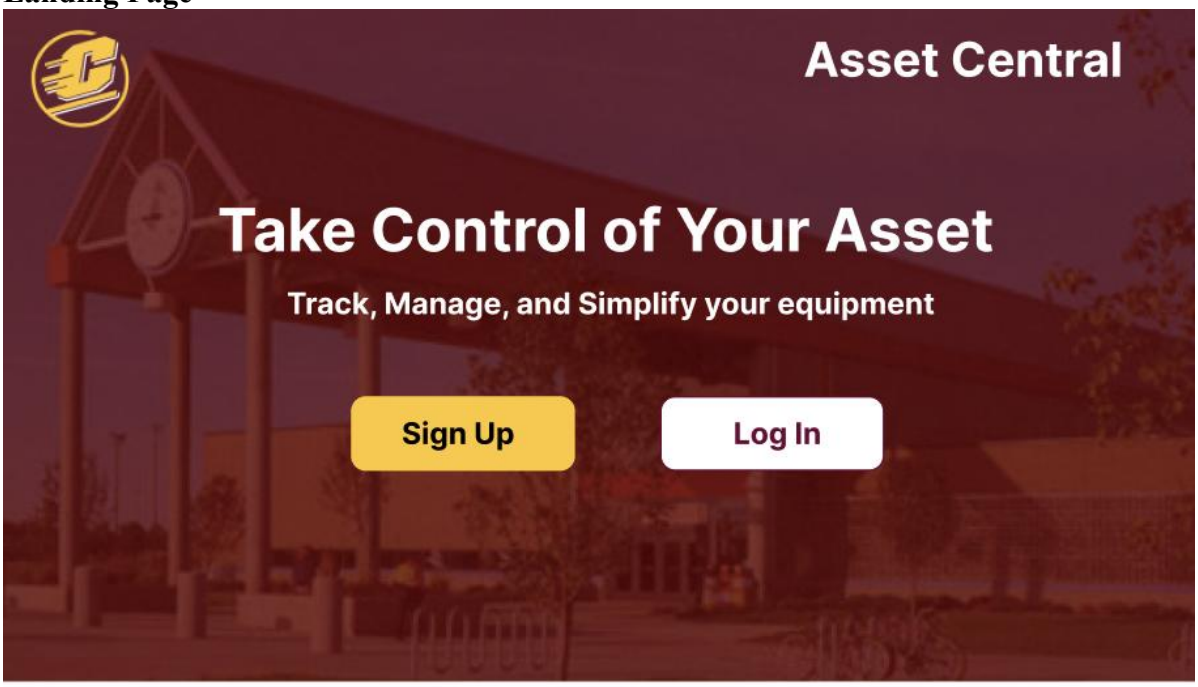
Phase-3 Complete (Nov 6)

Phase-4 Integration Complete (late Nov)

Project Finish (Dec 5, 2025)

9. UI Designs (Screens):

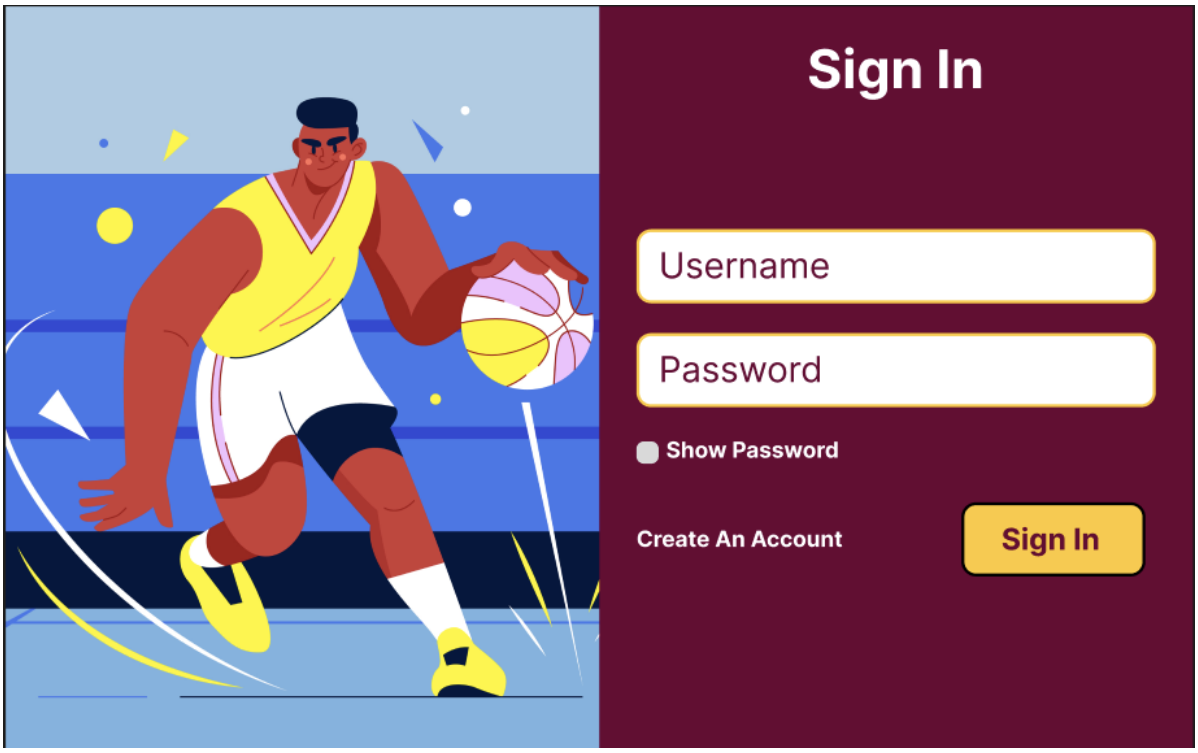
1. **Landing Page**



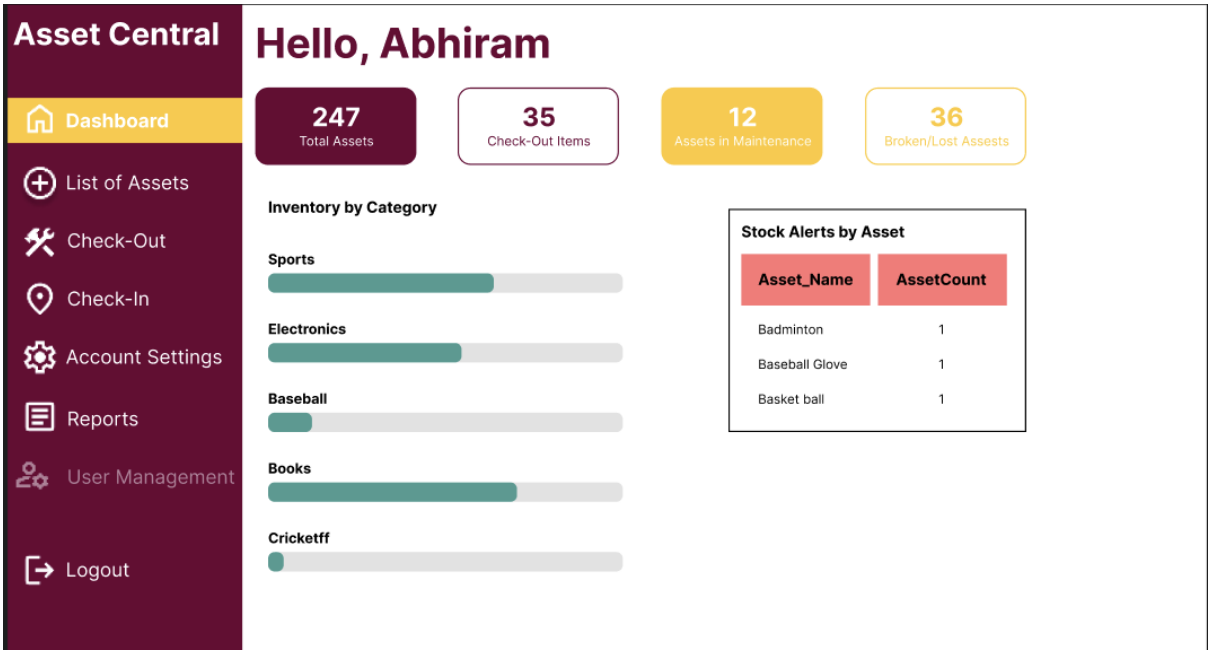
Save time and take full control of your equipment inventory

2. **Sign Up**

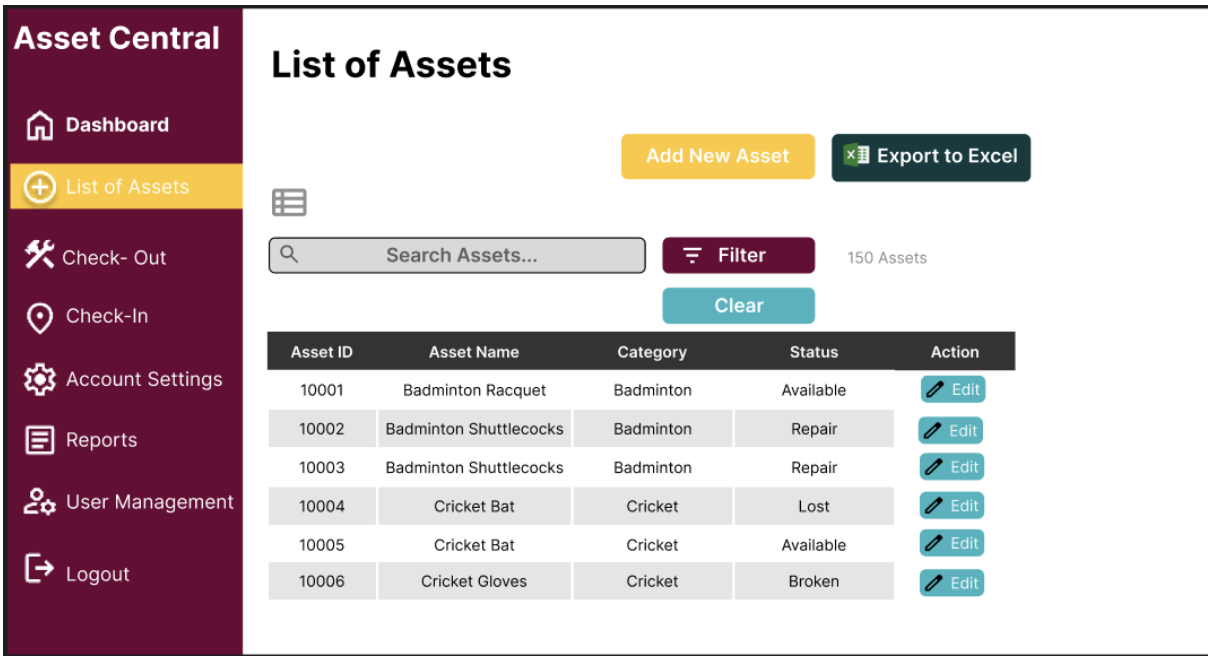
3. Sign In



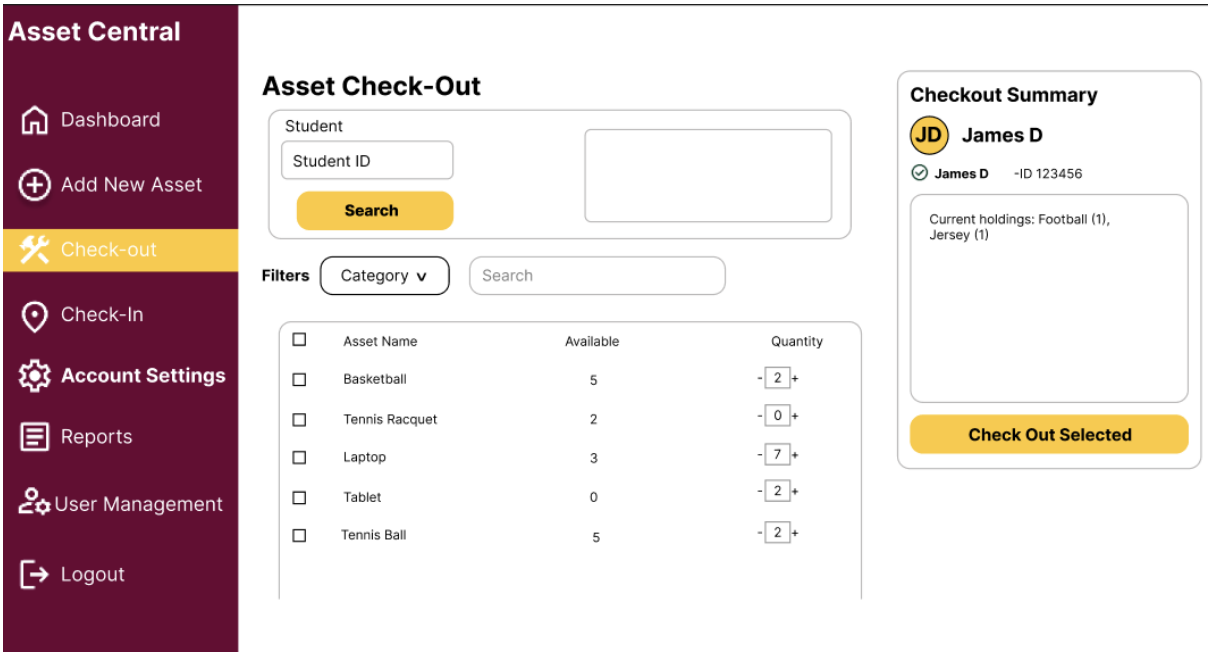
4. Dashboard



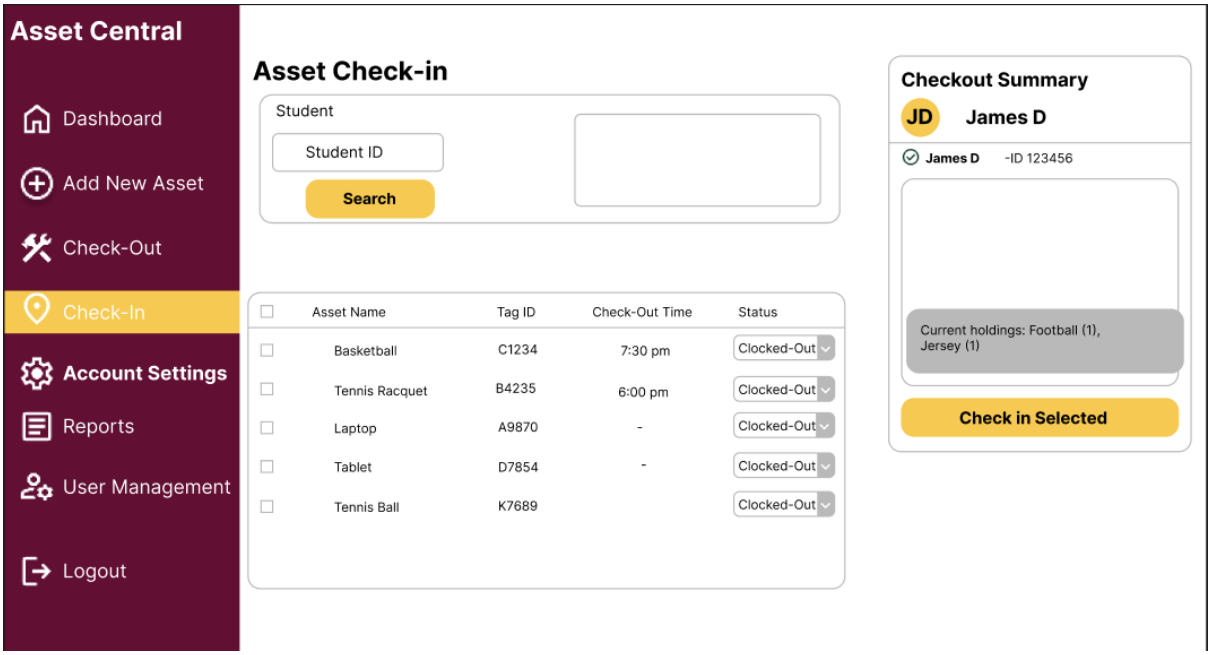
5. List of Assets



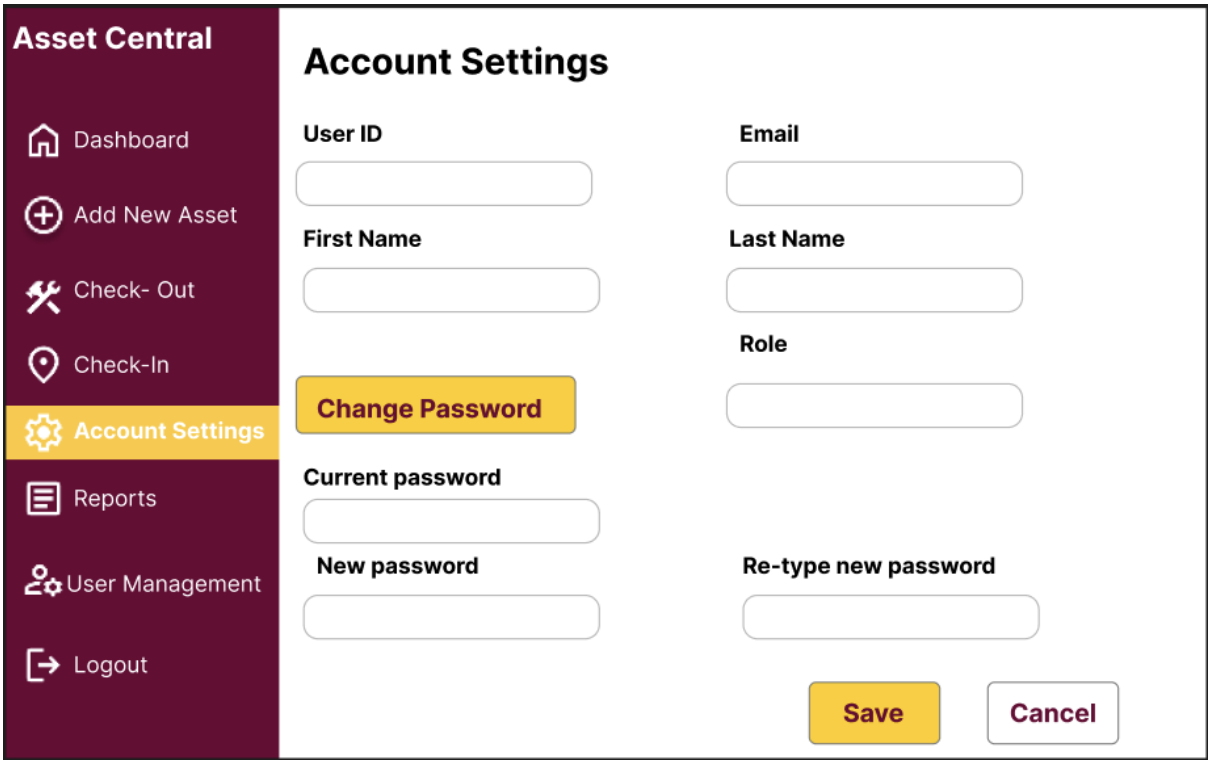
6. Check-Out



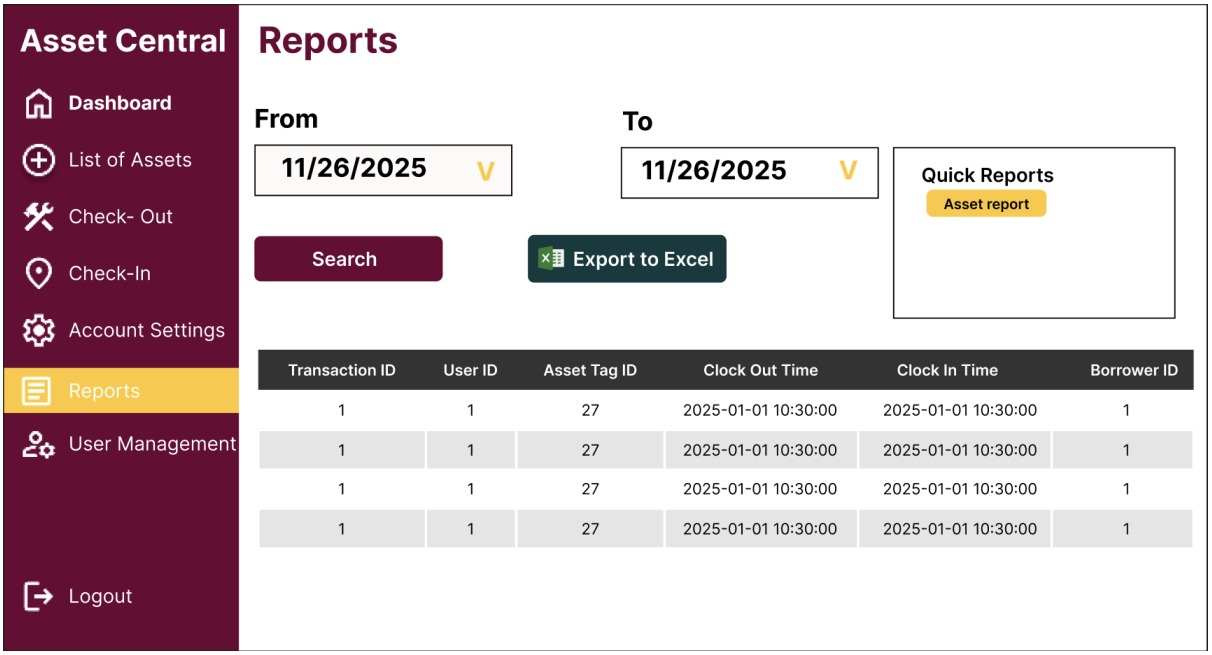
7. Check-In



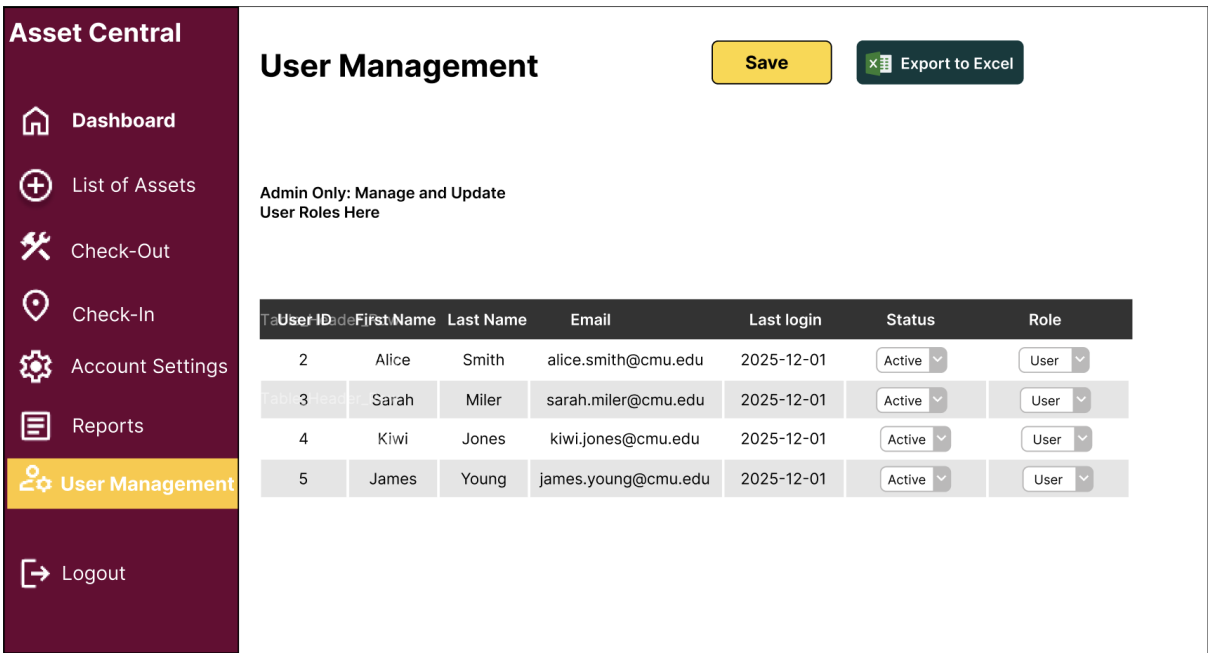
8. Account Settings



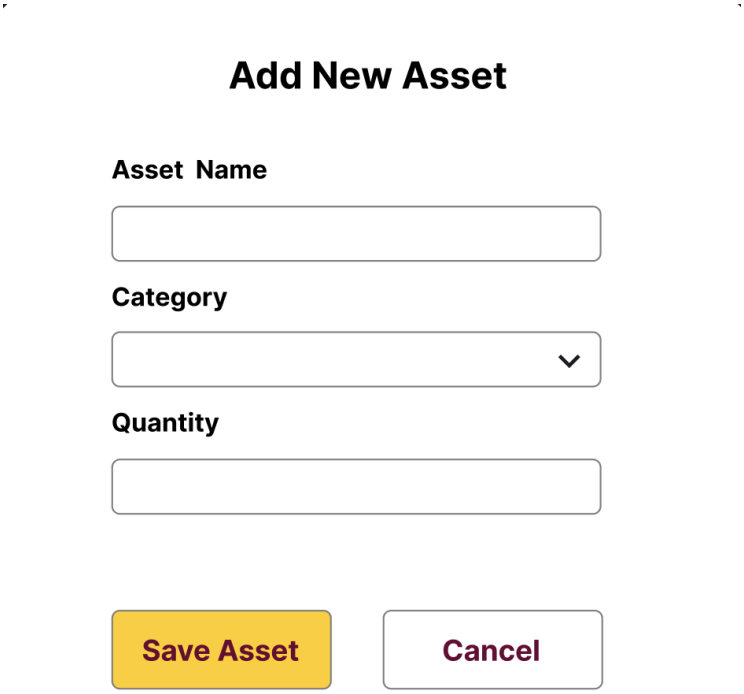
9. Reports



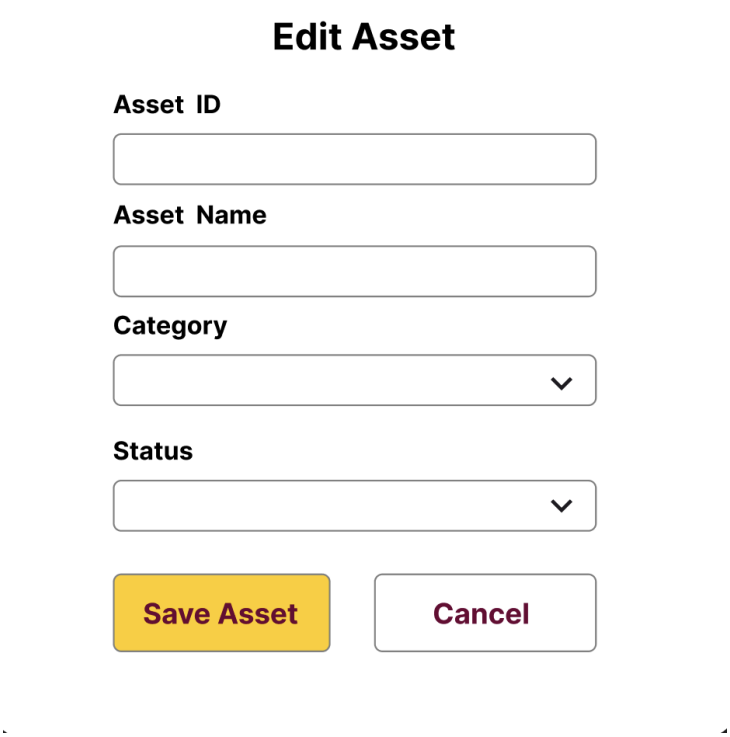
10. User Management



11. Add an Asset

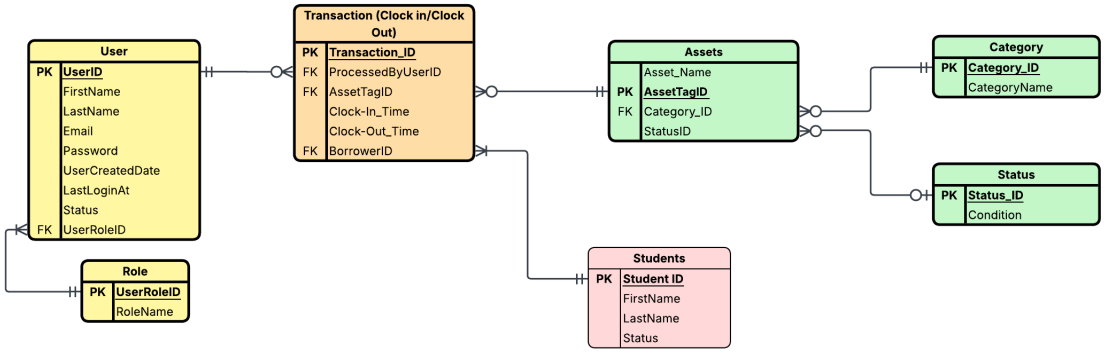


12. Edit an Asset



10. Entity–Relationship Diagram (ERD):

The Entity–Relationship (ER) diagram represents the structure of the Sports Asset Management System and illustrates how key data entities interact within the database. The design follows relational database principles and ensures referential integrity across all functional modules such as asset tracking, user management, borrower identification, and check-in/check-out transactions.



ERD Diagram

The system is composed of seven core entities: *User, Role, Assets, Category, Status, Students,* and *Transaction*. Each table includes a primary key (PK) to uniquely identify records, as well as foreign keys (FK) that establish logical connections between related entities.

User

The User entity stores information about staff members who operate the system. Each user is assigned a role, and this relationship is enforced through the foreign key User Role ID, which references the Role table. Key attributes include the user's name, email, password, account status, and timestamps for creation and last login.

PK: UserI D

FK: User Role ID → Role.UserRoleID

This ensures that every system user is mapped to a valid role such as Admin or Staff.

Role

The Role entity defines authorization levels within the system. It contains role identifiers and role names (e.g., Admin, Student Worker).

PK: UserRoleID

The Role table is referenced by the User table, supporting structured access control.

Assets

The Assets entity stores detailed information about individual equipment items available for checkout. Each asset is linked to a Category (e.g., basketball, cricket bat) and a Status (e.g., Available, Broken, Clocked-Out).

PK: AssetTagID

FK1: Category_ID → Category.Category_ID

FK2: StatusID → Status.Status_ID

This enables the system to classify inventory and track the condition of each item.

Category
The Category entity groups assets based on type. It provides a hierarchical structure that improves searchability and reporting.
PK: Category_ID
Assets reference categories to maintain standardized organization across inventory records.

Status
The Status entity represents the condition or availability of each asset. Examples include *Available*, *Clocked-Out*, *Lost,* or *Under Repair*.
PK: Status_ID
This table supports real-time status updates and ensures accurate reporting of asset conditions.

Students
The Students entity stores information about individuals who borrow equipment. Attributes include first name, last name, and student status.
PK: StudentID
This table is referenced by the Transaction table to identify the borrower associated with each check-out event.

Transaction (Clock-In/Clock-Out)
The Transaction entity captures both check-out and check-in activities. It links together the user processing the transaction, the borrower, and the asset being issued or returned. Timestamp fields document when equipment is checked out and when it is returned.
PK: Transaction_ID
FK1: ProcessedByUserID → User.UserID
FK2: AssetTagID → Assets.AssetTagID
FK3: BorrowerID → Students.StudentID
This structure ensures that every transaction is fully accountable and tied to legitimate users, assets, and borrowers.

Summary
Role → User: One-to-many (one role can have many users).
Category → Assets: One-to-many (each category contains multiple assets).
Status → Assets: One-to-many (each asset has one status).
User → Transaction: One-to-many (staff process many transactions).
Assets → Transaction: One-to-many (assets may appear in many transactions over time).
Students → Transaction: One-to-many (students may borrow multiple assets).
The ER diagram ensures data consistency, supports efficient inventory tracking, and provides a scalable structure for future system enhancements such as analytics dashboards, automated notifications, and audit trails.

11. SQL Script and Procedures:

```sql
USE BIS698Fall25_GRP1;

SET FOREIGN_KEY_CHECKS = 0;
```

This initial script prepares the database environment by executing the USE command to select the target database, BIS698Fall25_GRP1, ensuring all subsequent table creation and data manipulation commands are applied to the correct location. Crucially, it then uses the command SET FOREIGN_KEY_CHECKS = 0; to temporarily disable constraint validation. This practice is often employed at the beginning of a schema deployment to allow interdependent tables to be created or data to be loaded without the system immediately enforcing relationship rules, which is necessary when tables that reference each other are created out of order.

```sql
CREATE TABLE IF NOT EXISTS `Role` (
    `UserRoleID` INT AUTO_INCREMENT PRIMARY KEY,
    `RoleName`   VARCHAR(50) NOT NULL
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8mb4;

CREATE TABLE IF NOT EXISTS `Category` (
    `Category_ID`  INT AUTO_INCREMENT PRIMARY KEY,
    `CategoryName` VARCHAR(100) NOT NULL
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8mb4;

CREATE TABLE IF NOT EXISTS `Status` (
    `Status_ID` INT AUTO_INCREMENT PRIMARY KEY,
    `Condition` VARCHAR(50) NOT NULL
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8mb4;
```

This script defines three essential **lookup tables** used for data normalization and integrity across the system: Role, Category, and Status. The Role table establishes user permissions (e.g., 'Admin'); the Category table classifies the type of assets being tracked (e.g., 'Laptop' or 'Projector'); and the Status table defines the condition of an asset (e.g., 'Available' or 'Damaged'). Each of these tables features a simple structure with an auto-incrementing primary key (UserRoleID, Category_ID, Status_ID) and a single descriptive text field (RoleName, CategoryName, Condition), with all having the **InnoDB** engine and **utf8mb4** character set.

```sql
CREATE TABLE IF NOT EXISTS `User` (
    `UserID`          INT AUTO_INCREMENT PRIMARY KEY,
    `FirstName`       VARCHAR(100) NOT NULL,
    `LastName`        VARCHAR(100) NOT NULL,
    `Email`           VARCHAR(255) NOT NULL UNIQUE,
    `Password`        VARCHAR(255) NOT NULL,
    `UserCreatedDate` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    `Status`          TINYINT(1) NOT NULL DEFAULT 1,
    `UserRoleID`      INT NOT NULL,
    CONSTRAINT `fk_user_role`
        FOREIGN KEY (`UserRoleID`)
        REFERENCES `Role`(`UserRoleID`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8mb4;

CREATE TABLE IF NOT EXISTS `Students` (
    `StudentID` INT AUTO_INCREMENT PRIMARY KEY,
    `FirstName` VARCHAR(100) NOT NULL,
    `LastName`  VARCHAR(100) NOT NULL,
    `Status`    VARCHAR(50)  NOT NULL
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8mb4;
```

This script defines two distinct entity tables: User and Students. The **User** table stores system administrators or staff, including their name, a unique Email, Password, and a foreign key to the Role table (UserRoleID) to define their access level. The **Students** table is a separate, simpler entity designed to hold information about the individuals who will be borrowing the assets, containing basic fields like FirstName, LastName, and a Status field. Separating these two entities ensures a clear distinction between system operators and asset borrowers, while the User table implements an ON DELETE RESTRICT constraint on the role relationship to prevent the deletion of a role that is still assigned to an active user.

```sql
CREATE TABLE IF NOT EXISTS `Assets` (
    `AssetTagID`  INT AUTO_INCREMENT PRIMARY KEY,
    `Asset_Name`  VARCHAR(150) NOT NULL,
    `Category_ID` INT NOT NULL,
    `StatusID`    INT NOT NULL,
    CONSTRAINT `fk_assets_category`
        FOREIGN KEY (`Category_ID`)
        REFERENCES `Category`(`Category_ID`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT `fk_assets_status`
        FOREIGN KEY (`StatusID`)
        REFERENCES `Status`(`Status_ID`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8mb4;
```

This script defines the core **Assets** table, which catalogs every item managed by the system. The table includes the asset's primary key, AssetTagID, its Asset_Name, and two foreign keys to classify the item: Category_ID links to the Category lookup table, and StatusID links to the Status lookup table to describe its current condition. Both foreign key constraints (fk_assets_category and fk_assets_status) are set up with **ON UPDATE CASCADE** to automatically update asset records if a category or status ID changes, and **ON DELETE RESTRICT** to prevent the accidental deletion of a category or status that is currently assigned to any asset.

```
66  CREATE TABLE IF NOT EXISTS `Transaction` (
67      `Transaction_ID`    INT AUTO_INCREMENT PRIMARY KEY,
68      `ProcessedByUserID` INT NOT NULL,
69      `AssetTagID`        INT NOT NULL,
70      `Clock_In_Time`     DATETIME NOT NULL,
71      `Clock_Out_Time`    DATETIME NULL,
72      `BorrowerID`        INT NOT NULL,
73      CONSTRAINT `fk_trx_user`
74          FOREIGN KEY (`ProcessedByUserID`)
75          REFERENCES `User`(`UserID`)
76          ON UPDATE CASCADE
77          ON DELETE RESTRICT,
78      CONSTRAINT `fk_trx_asset`
79          FOREIGN KEY (`AssetTagID`)
80          REFERENCES `Assets`(`AssetTagID`)
81          ON UPDATE CASCADE
82          ON DELETE RESTRICT,
83      CONSTRAINT `fk_trx_student`
84          FOREIGN KEY (`BorrowerID`)
85          REFERENCES `Students`(`StudentID`)
86          ON UPDATE CASCADE
87          ON DELETE RESTRICT
88  ) ENGINE = InnoDB
89    DEFAULT CHARSET = utf8mb4;
90
91  SET FOREIGN KEY CHECKS = 1;
```

This script initiates the creation of the **Transaction** table, the central table for recording all asset movements. It logs when an asset is checked in or out using the Clock_In_Time and Clock_Out_Time fields, and records three crucial foreign keys: ProcessedByUserID (linked to the User table), AssetTagID (linked to the Assets table), and BorrowerID (linked to the Students table). This structure ensures that every transaction is fully traceable, documenting which staff member processed it, which specific asset was involved, and which student borrowed the item, with all relationships utilizing ON UPDATE CASCADE and ON DELETE RESTRICT for referential integrity.

Procedures:

Note: Only two procedures are described below as examples. The full list of stored procedures is included in the submitted SQL file.

```
CREATE DEFINER=`kotha2t`@`%` PROCEDURE `get_all_assets`()
BEGIN
    SELECT
        a.AssetTagID AS `Asset ID`,
        a.Asset_Name AS `Asset Name`,
        c.CategoryName AS `Category`,
        s.Condition AS `Status`
    FROM Assets a
    INNER JOIN Category c
        ON a.Category_ID = c.Category_ID
    INNER JOIN Status s
        ON a.StatusID = s.Status_ID
    ORDER BY a.AssetTagID ASC;
END
```

This procedure retrieves a complete list of all assets along with their associated category and status information. It joins the Assets, Category, and Status tables to produce a unified dataset containing the asset ID, name, category type, and condition. The query ensures that the inventory view remains accurate and organized by ordering the result set by the asset tag ID in ascending order.

This procedure is primarily used to populate the “List of Assets” screen in the application, support asset audits, and assist administrators in managing the full equipment inventory. By centralizing asset data retrieval into a single procedure, it ensures consistency and improves the efficiency of backend operations.

```
CREATE DEFINER=`kotha2t`@`%` PROCEDURE `get_available_category_counts`()
BEGIN
    SELECT
        c.CategoryName,
        COUNT(a.AssetTagID) AS quantity
    FROM Assets a
    INNER JOIN Status s
        ON a.StatusID = s.Status_ID
    INNER JOIN Category c
        ON a.Category_ID = c.Category_ID
    WHERE s.`Condition` = 'Available'
    GROUP BY c.CategoryName
    ORDER BY c.CategoryName;
END
```

This procedure generates a summarized breakdown of all assets that are currently marked as “Available” across different categories. By filtering assets based on their status and grouping them by category, the procedure returns the total count of available items within each classification, allowing teams to quickly assess equipment availability.

It is used within dashboard analytics, reporting views, and inventory monitoring features to help staff identify shortages, plan equipment allocation, and maintain smooth operational workflows. The categorized summary supports decision-making by offering a clear snapshot of inventory distribution across the facility.