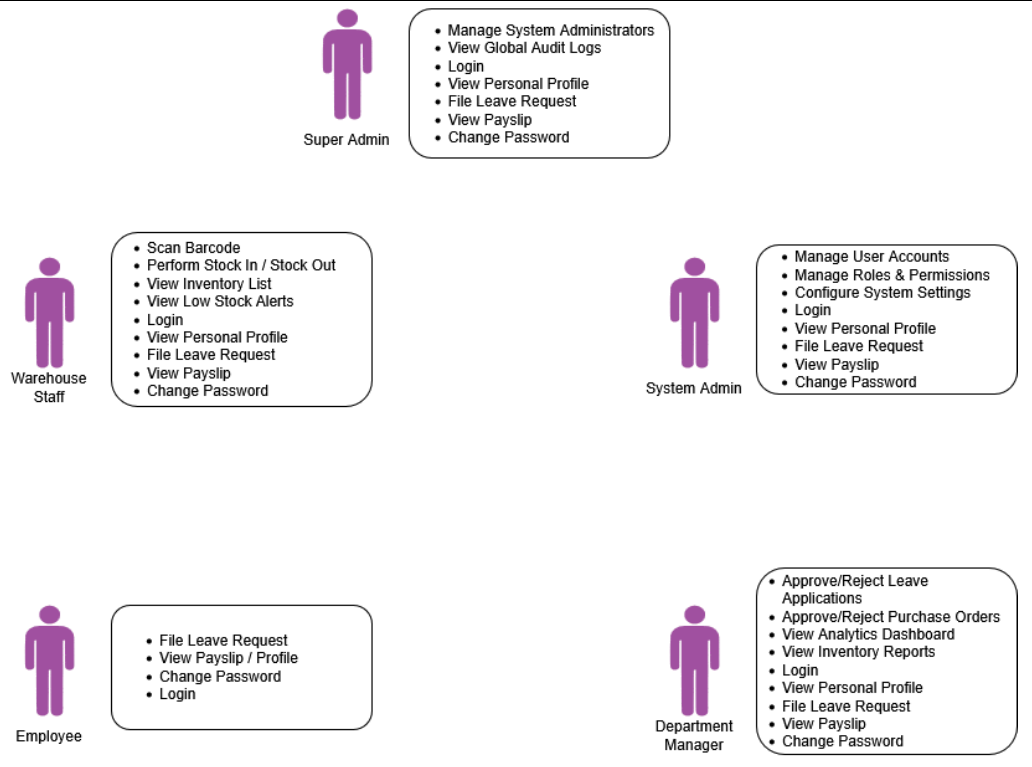
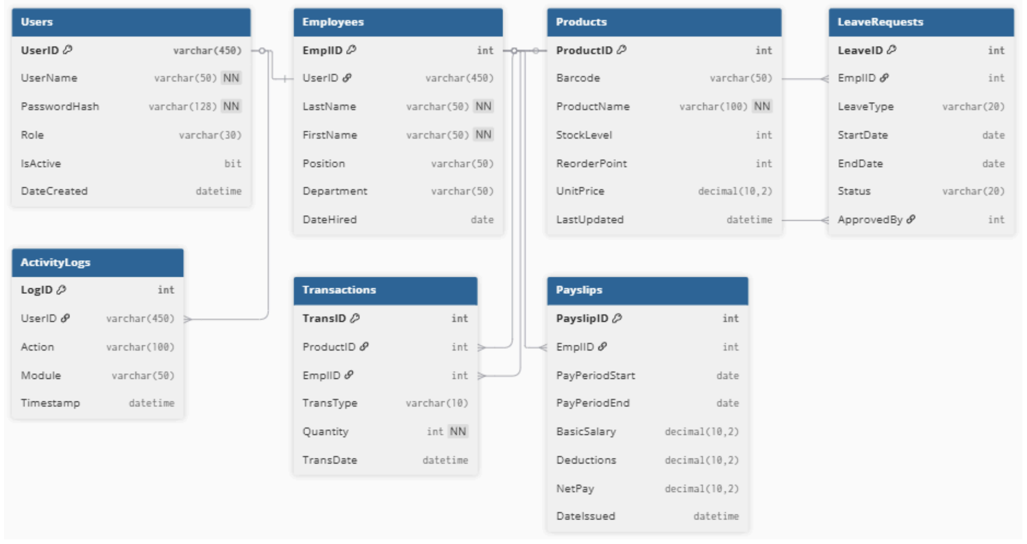
USE_CASE_ERD_DATA_DICTIONARY

NAME	**EVANDER HAROLD S. AMORCILLO**
PROJECT TITLE	**MobileOps Connect**
SUBJECT: CODE: TIME:	**IT15/L Integrative Programming and Technologies** **8441** **10:00 AM – 12:00 PM**
TOPIC (Type of Business Process)	**#33 Mobile Business Systems**
Products/Services	**Mobile-Integrated ERP Solution**
Agile Model - SDLC	MobileOps Connect is designed to modernize hardware store operations by integrating inventory management and HR workflows into a unified mobile-responsive ERP. Requirement identification was conducted by analyzing legacy paper-based processes, ensuring the system addresses the pain points of Warehouse Staff, Managers, and Employees. Core Requirements include: ● Real-time inventory tracking via barcode scanning (ZXing.Net). ● Remote leave application and approval workflow. ● Role-Based Access Control (RBAC) for data security. ● Automated low-stock alerts and push notifications. ● Centralized audit logging for system accountability. Functional Requirements focused on user roles: ● Super Admin (The Developer) maintains full system control, managing System Administrator accounts, performing database backups, and overseeing global security logs. ● System Admins can manage standard user accounts (Staff/Managers), configure system settings, and reviewing standard audit trails. ● Warehouse Staff can scan barcodes to perform Stock In/Out transactions and view real-time inventory levels. ● Department Managers can remotely approve leave requests, view analytics dashboards, and monitor stock movements. ● Employees can independently file leave requests, view payslips, and check leave balances via mobile. ● System performs automated checks for reorder points and sends email/push alerts to managers. Non-functional Requirements ensure enterprise-grade reliability: ● Security: ASP.NET Core Identity for authentication and encrypted password storage. ● Performance: SignalR for real-time data updates without page reloads. ● Usability: Bootstrap 5 mobile-first design for seamless access on smartphones.

	● Reliability: Offline capabilities using PWA Service Workers. The Sprint Planning Strategy (Agile Scrum): ● Sprint 1-2: System setup, Database creation, and Authentication (RBAC) module. ● Sprint 3-4: Development of Inventory Module (Barcode scanning, CRUD operations) and Products database. ● Sprint 5-6: Implementation of HR Module (Leave requests, Payslip view) and Notification logic. ● Final Sprint: Integration testing, bug fixing, and deployment of Audit Logs and Dashboards. Planning Outcome: The adoption of the Agile methodology ensures that MobileOps Connect evolves through iterative feedback. By delivering functional modules (Inventory first, then HR) in two-week sprints, the developer can rapidly adjust to changes, ensuring the final deliverable is a robust, secure, and user-centric solution that effectively eliminates manual bottlenecks in business operations.
Use Case Diagram (Diagram Only)	 Super Admin • Manage System Administrators • View Global Audit Logs • Login • View Personal Profile • File Leave Request • View Payslip • Change Password Warehouse Staff • Scan Barcode • Perform Stock In / Stock Out • View Inventory List • View Low Stock Alerts • Login • View Personal Profile • File Leave Request • View Payslip • Change Password System Admin • Manage User Accounts • Manage Roles & Permissions • Configure System Settings • Login • View Personal Profile • File Leave Request • View Payslip • Change Password Employee • File Leave Request • View Payslip / Profile • Change Password • Login Department Manager • Approve/Reject Leave Applications • Approve/Reject Purchase Orders • View Analytics Dashboard • View Inventory Reports • Login • View Personal Profile • File Leave Request • View Payslip • Change Password

Entity Relational Diagram



Data Dictionary

MobileOps Connect

Users Table

Field Names	**Datatype**	**Length**	**Description**
UserID-PK	Varchar	450	Unique System ID (ASP.NET Identity GUID)
Username	Varchar	50	Login Name / Email
PasswordHash	Varchar	128	Encrypted Password
Role	Varchar	30	(Super Admin, Manager, Staff, Employee)
IsActive	Boolean	1	Account Status
DateCreated	DateTime	-	Date Registered

Employees Table

Field Names	**Datatype**	**Length**	**Description**
EmplID-PK	Int-AI	9	Unique Employee ID
UserID-FK	Varchar	450	Link to Users Table
LastName	Varchar	50	Employee Last Name
FirstName	Varchar	50	Employee First Name
Position	Varchar	50	Job Title (e.g., Warehouse Staff)
Department	Varchar	50	Assigned Dept
DateHired	Date	-	Employment Date

Products Table

Field Names	Datatype	Length	Description
ProductID-PK	Int-AI	9	Unique Product ID
Barcode	Varchar	50	Scanned Barcode Value
ProductName	Varchar	100	Name of Item
StockLevel	Int	9	Current Qty in Warehouse
ReorderPoint	Int	9	Low Stock Alert Level
UnitPrice	Decimal	10,2	Cost per Item
LastUpdated	DateTime	-	Date of last movement

Transactions Table

Field Names	Datatype	Length	Description
TransID-PK	Int-AI	9	Unique Transaction ID
ProductID-FK	Int	9	Link to Products
EmplID-FK	Int	9	Who scanned the item
TransType	Varchar	10	"IN" or "OUT"
Quantity	Int	9	Qty Adjusted
TransDate	DateTime	-	When it happened

LeaveRequests Table

Field Names	Datatype	Length	Description
LeaveID-PK	Int-AI	9	Unique Request ID
EmplID-FK	Int	9	Requesting Employee
LeaveType	Varchar	20	(Sick, Vacation, Emergency)
StartDate	Date	-	Start of Leave
EndDate	Date	-	End of Leave
Status	Varchar	20	Pending / Approved / Rejected
ApprovedBy-FK	Int	9	Manager who approved

ActivityLogs Table

Field Names	**Datatype**	**Length**	**Description**
LogID-PK	Int-AI	9	Log ID
UserID-FK	Varchar	450	User who did the action
Action	Varchar	100	What they did (e.g., "Deleted User")
Module	Varchar	50	System Area (e.g., "Inventory")
Timestamp	DateTime	-	Exact time of event

Accounting Table

Field Name	Datatype	Length	Description
PayslipID	int	11	Primary Key, Auto-increment. Unique identifier for the payslip.
EmpID	int	11	Foreign Key referencing Employees(EmpID).
PayPeriodStart	date	-	The start date of the payroll cycle.
PayPeriodEnd	date	-	The end date of the payroll cycle.
BasicSalary	decimal	10,2	The gross basic salary for the period.
Deductions	decimal	10,2	Total deductions (taxes, benefits).
NetPay	decimal	10,2	The final take-home pay amount.
DateIssued	datetime	-	The timestamp when the payslip was generated.

Date Submitted:	February 16, 2026
Teacher's Feedback	____________________ ____________________ ____________________ ____________________ ____________________

	____ ______________________________ ______________________________ Student's Signature (after feedbacking) Teacher's Signature

1. Users Table

- Primary Owner: System Administrator
- Why: The System Admin is the one who "Manages User Accounts" (creating logins, resetting passwords).
- Secondary Owner: Super Admin (Can manage the System Admins).

2. Employees Table

- Primary Owner: Employee (Everyone)
- Why: Every single user (Manager, Staff, Admin) has a profile here.
- Action: The Employee views their own profile here. The Department Manager views the records of their staff.

3. Products Table

- Primary Owner: Warehouse Staff
- Why: The Warehouse Staff is the one scanning barcodes and checking StockLevel every day.
- Secondary Owner: Department Manager (Views Inventory Reports based on this table).

4. Transactions Table

- Primary Owner: Warehouse Staff
- Why: This table records "Stock In" and "Stock Out." Only the Warehouse Staff performs these physical actions.
- Secondary Owner: Department Manager (Views the history/reports to see where items went).

5. LeaveRequests Table

- Primary Owner: Employee
- Why: The Employee *creates* the row when they "File Leave Request".
- Secondary Owner: Department Manager
- Why: The Manager *updates* the row when they click "Approve" or "Reject".

6. ActivityLogs Table

- Primary Owner: System (Automated)
- Viewer: Super Admin & System Administrator
- Why: Regular users never touch this table. It is exclusively for the Admins to "View Audit Logs" to see who did what.

Summary Table for Your Mental Model

Table Name	Who Inputs Data? (Write)	Who Checks Data? (Read)

Users	System Admin	System / Super Admin
Employees	HR / System Admin	Employee (Self) / Manager
Products	Warehouse Staff	Manager
Transactions	Warehouse Staff	Manager / Admin
LeaveRequests	Employee	Department Manager
ActivityLogs	System (Auto)	Super Admin

The Basic Rule of Crow's Foot Notation

- Single Line (| or -): Represents "ONE". This is the Parent.
- Crow's Foot (< or ∈): Represents "MANY". This is the Child.

---

1. Users ↔ Employees (1-to-1 Relationship)

The Connection: Users.UserID —— Employees.UserID

- Visual: A single line on both ends (no crow's foot).
- The Logic: "One User Account belongs to exactly One Employee Profile."
- Why?
  - You don't want one person (Evander) to have two different login accounts.
  - You don't want one login account to be shared by five different people (security risk).
  - Technical Detail: The Employees table has a Foreign Key UserID that is also unique. This strictly links 1 login to 1 physical person.

2. Products ↔ InventoryTransactions (1-to-Many)

The Connection: Products.ProductID —< InventoryTransactions.ProductID

- Visual: Single line at Products, Crow's foot at Transactions.
- The Logic: "One Product can appear in Many Transactions."
- Why?
  - Imagine a Hammer (Product ID 101).
  - Today, you stock in 50 Hammers (Transaction A).
  - Tomorrow, you sell 2 Hammers (Transaction B).
  - Next week, you buy 10 more (Transaction C).
  - Result: The "Hammer" is listed in the Products table once, but it appears in the Transactions table three times (or infinitely).

3. Users ↔ InventoryTransactions (1-to-Many)

The Connection: Users.UserID —< InventoryTransactions.EmpID

- Visual: Single line at Users, Crow's foot at Transactions.
- The Logic: "One Warehouse Staff (User) can perform Many Stock Transactions."
- Why?
  - Your warehouse staff member, John, works all day.
  - At 8:00 AM, he scans a delivery (Transaction 1).
  - At 9:00 AM, he scans an outgoing shipment (Transaction 2).
  - Result: John's User ID appears in the InventoryTransactions table every time he scans something.

4. Employees ↔ LeaveRequests (1-to-Many)

The Connection: Employees.EmpID —< LeaveRequests.EmpID

- Visual: Single line at Employees, Crow's foot at LeaveRequests.
- The Logic: "One Employee can file Many Leave Requests."
- Why?
  - Mary (Employee) gets sick in January (Request #1).
  - Mary goes on vacation in May (Request #2).
  - Mary has an emergency in December (Request #3).
  - Result: Mary exists once in the Employee table, but she has multiple rows in the Leave Requests table over time.

5. Users ↔ LeaveRequests (1-to-Many)

The Connection: Users.UserID —< LeaveRequests.ApprovedBy

- Visual: Single line at Users, Crow's foot at LeaveRequests.
- The Logic: "One Manager (User) can Approve Many Leave Requests."
- Why?
  - This is a specific relationship for the "ApprovedBy" column.

    - The HR Manager logs in. They see 10 pending requests from different staff.
    - They click "Approve" on all 10.
    - Result: The HR Manager's ID is stamped on all 10 leave request rows as the person who approved them.

6. Users ↔ ActivityLogs (1-to-Many)

The Connection: Users.UserID —< ActivityLogs.UserID

- Visual: Single line at Users, Crow's foot at ActivityLogs.
- The Logic: "One User generates Many Activity Logs."
- Why?
    - This is your Audit Trail.
    - When the Super Admin logs in, that's 1 row in the log.
    - When they create a new user, that's another row.
    - When they change a password, that's another row.
    - Result: A single user creates a long history of actions ("Many" logs) over the lifetime of the system.

Summary for Your Professor:

If asked, simply say:

> "The Users and Products tables are the 'Reference Data' (the One side), while tables like Transactions, Logs, and Requests are the 'Transactional Data' (the Many side) because they grow indefinitely as the system is used."

1. The "Why" (Introduction)

> *"MobileOps Connect is designed to modernize hardware store operations..."*

- **The Logic:** You are establishing the **Problem Statement** immediately. The hardware store currently uses "legacy paper-based processes" (clipboards, physical forms).
- **The Solution:** You are proposing a "Unified Mobile-First ERP." This means you aren't just building an app; you are building a *system* that connects two different worlds: **Inventory** (Warehouse) and **HR** (People).
- **Why Agile?** You mentioned "addressing pain points." Agile is chosen because you can show the warehouse staff a prototype, get their feedback ("This button is too small"), and fix it in the next Sprint.

2. Core Requirements (The "Big Pillars")

This section lists the non-negotiable features your system *must* have.

- **Real-time tracking (ZXing.Net):** You are using a specific library for barcode scanning. This proves you have researched the technology.

- **RBAC (Role-Based Access Control):** This is your security backbone. It ensures a Warehouse Staff member cannot accidentally approve a Manager's leave request.
- **Audit Logging:** This is critical for accountability. If 50 cement bags go missing, the Audit Log tells you exactly *who* logged in and modified the stock level.

3. Functional Requirements (The "Who Does What")

This breakdown proves you understand the **hierarchy** of your users.

- **Super Admin (You):** You added this role to show that *someone* needs to maintain the database and fix the servers. That person is you (the Developer).
- **System Admin vs. Super Admin:** We separated these to show a "Chain of Command." A System Admin helps users reset passwords, but they can't delete the whole database—only the Super Admin can.
- **Warehouse Staff & Managers:** You clearly separated their duties. Staff = *Physical Work* (Scanning). Managers = *Decision Work* (Approving).

4. Non-Functional Requirements (The "Quality" Specs)

These are the "invisible" features that make the system professional.

- **Security (ASP.NET Core Identity):** You aren't writing your own login code (which is dangerous). You are using Microsoft's industry-standard security framework.
- **Performance (SignalR):** This is your "Wow Factor." Usually, web pages need to be refreshed to see new data. SignalR makes the screen update *instantly* (like a chat app) when a new order comes in.
- **Reliability (PWA/Service Workers):** This is your "Offline Mode." If the internet cuts out in the warehouse, the app still works. This is crucial for a "Mobile Business System."

5. Sprint Planning (The "Timeline")

Agile Scrum divides work into 2-week "Sprints." This schedule shows your professor you have a realistic plan to finish by the deadline.

- **Sprint 1-2 (The Foundation):** You can't build the house without the concrete. This is setting up the Database and Login system.
- **Sprint 3-4 (The "Meat"):** This is the Inventory Module. It's the hardest part, so you do it early.
- **Sprint 5-6 (The "Side Dish"):** The HR Module is easier (mostly forms), so you do it later.
- **Final Sprint (Polish):** Testing and bug fixing. This shows maturity—you know code never works perfectly the first time.

6. Planning Outcome (The "Goal")

> *"...the developer can rapidly adjust to changes..."*

- **The Key Phrase:** "Iterative Feedback."

- **Translation:** You are admitting that you might not get everything right on Day 1. Agile allows you to build a small part, test it, fix it, and then build the next part. This is much safer than the old "Waterfall" method where you build the whole thing and hope it works at the end.

🛡️ Professor Defense Q&A

If he asks these questions, here are your answers:

Q: Why did you use Agile Scrum for a solo project?

A: "Because Agile allows me to break a massive ERP system into manageable 2-week chunks (Sprints). It keeps me focused on finishing one module (like Inventory) before starting the next (HR), ensuring I always have working software to show."

Q: Why SignalR?

A: "For a warehouse, speed is critical. If item stock drops to zero, the Manager needs to see that alert *instantly* without refreshing their browser. SignalR provides that real-time push capability."

Q: What is the difference between Super Admin and System Admin?

A: "The Super Admin is the 'Developer' role that manages the server and database backups. The System Admin is an 'Operations' role that handles day-to-day user support like password resets."