

Information Technology — Capstone Project

Documentation - AssetPulse

Knight Foundation School of Computing and Information Sciences (KFSCIS) — Florida International University

Instructor: Masoud Sadjadi | Version: 2025 Fall | License: MIT

Team:

Anthony Yi - ayi002@fiu.edu

Erika Pita - epita007@fiu.edu

Tejash Pradhan - tprad007@fiu.edu

Victoria Riley - vbruc002@fiu.edu

Levis Vazquez - lvazq092@fiu.edu

1. Executive Summary

AssetPulse is a simple mobile IT asset management system created to assist small organizations track equipment. AssetPulse was created for small IT teams and users who are looking for a simple and light method of tracking their technological assets without having to be tied to a year subscription.

Key capabilities that AssetPulse provides are secure login, role-based permissions, asset creation and editing, metrics and export options. Key results shown by testing show stable performance, accurate data storage and correct access restrictions. Additionally, testing showed compatibility with both physical android devices and emulators.

2. Problem & Requirements

AssetPulse addresses a common major problem in small organizations: IT assets are often tracked through spreadsheets, word documents, paper logs or personal memory. This lack of a centralized tracking method of IT assets can lead to loss of equipment, inaccurate records, weak IT equipment lifecycle tracking and poor inventory tracking. AssetPulse aims to provide free alternative options to the expensive and complex asset management systems out in the market currently.

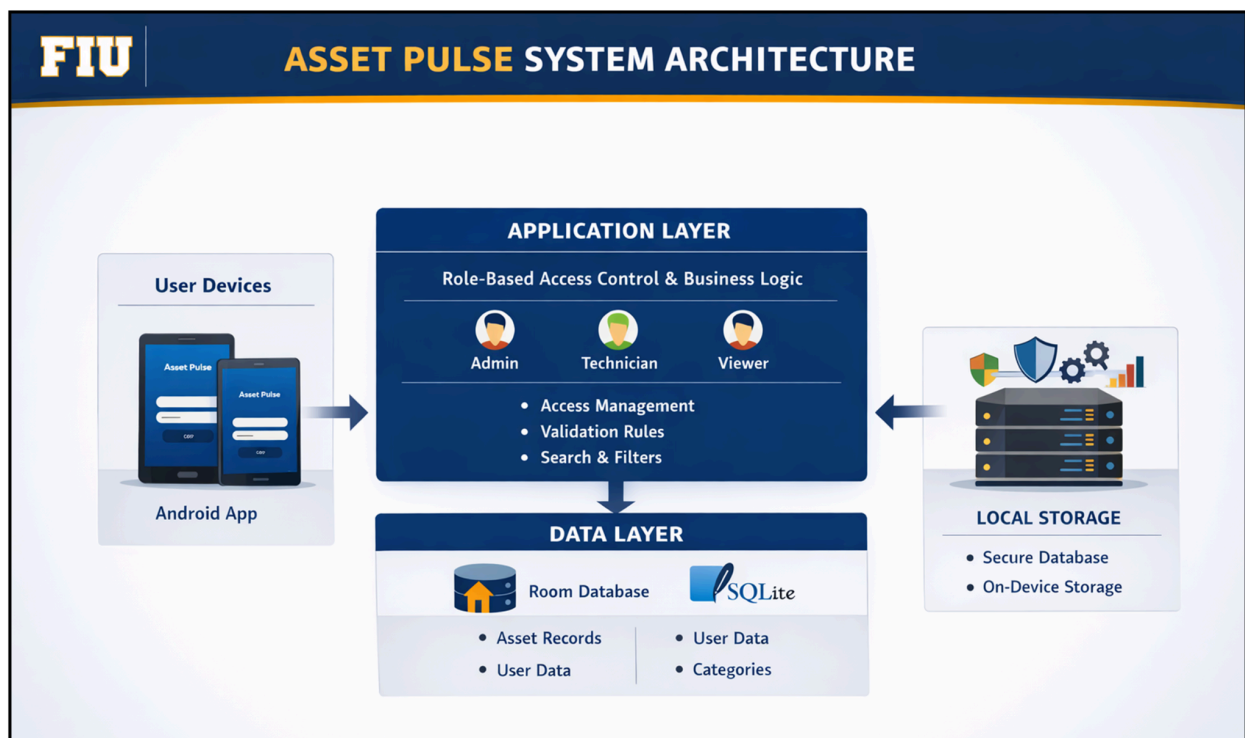
Primary stakeholders of AssetPulse are IT staff, administrators, or employees/managers who handle the IT equipment lifecycle in an organization.

An example of a user persona of AssetPulse is an IT administrator who is part of a small organization who needs to access information about his companies' IT assets in a fast, timely manner without a costly enterprise subscription.

Prioritized backlog:

1. Secure and accurate login flow
2. Role-based access control
3. Dashboard Metrics
4. CSV export feature
5. Mobile Friendly UI
6. Emulator compatibility
7. Physical device compatibility
8. Search and filter feature
9. Asset creation, deletion and editing
10. Installation documentation

3. System Overview & Architecture



AssetPulse uses a lightweight three-layer architecture:

- Frontend: Android mobile application
- Logic Layer: Access control, validation, workflows

- Data Layer: Local Room/SQLite database

4. Discipline-Specific Depth

- Architecture & Integration: AssetPulse utilizes an Android mobile frontend, a local logic/workflow layer, and a Room/SQLite layer. The application favors an offline, on-device deployment model. This helps reduce infrastructure needs and complexity.
- DevOps & SRE: AssetPulse creation shows the use of Gradle-based configuration and Android Studio usage.
- Operations & Security: Access management is addressed through secure login and role-based permissions. Configuration hardening is supported through SDK and Gradle settings. The application does not have a formal backup/restore feature.

5. Implementation Notes

Repository structure, coding conventions, notable patterns, and tradeoffs. Reference the README and module layout.

Repository structure:

The project follows a single-module Android architecture organized by functional layer. The root package is *com.levis.assetpulse* and the source tree is split into three main packages:

data/ - Contains the persistence layer, subdivided into *model/* (Room entity classes for Asset and User), *local/* (the AppDatabase singleton and DAO interfaces for AssetDao and UserDao) and *repository/* (the AssetRepository class that acts as the single access point for all database operations).

ui/ - Contains all presentation logic. The *screens/* sub-package holds every Composable screen (DashboardScreen, AssetsScreen, UsersScreen, UserDetailScreen, ProfileScreen, LoginScreen and MainScreen), while *theme/* sub-package holds Material 3 color, typography and theme definitions. The MainViewModel and Navigation sealed class live at the *ui/* package root.

MainActivity.kt - The single Activity entry point, which sets up the Compose content and the top-level NavHost.

The build system uses Gradle KTS with version catalog (libs.versions.toml) and KSP for Room annotation processing. The project targets SDK 36 with a minimum SDK of 30.

Coding Conventions:

The project is written entirely in Kotlin and uses Jetpack Compose for the user interface without any XML layouts. Each screen is a Composable function that receives a shared ViewModel and a NavController for navigation. Data from the database is observed using StateFlow and collected in each screen so the UI updates automatically when data changes. All database operations run on background threads through Kotlin coroutines.

Notable Patterns:

A single ViewModel is shared across all screens to keep the architecture simple and avoid additional setup with dependency injection frameworks. On first launch, the app seeds the database with sample users and over 50 assets with realistic dates, statuses and assignments so the app is ready to demo immediately. Navigation is handled through a sealed class with support for parameterized routes, allowing the Dashboard to link directly to a filtered view of assets by status or manufacturer. CSV export uses Android's FileProvider and share intent system to let users send or save asset data externally.

Tradeoffs:

- Room is configured with `fallbackToDestructiveMigration()`, which means any database schema change during development will wipe local data and reseed it. This is acceptable for a capstone project with seeded demo data but would not be appropriate for a production app that needs to preserve user data across updates.
- User passwords are stored as plain text to keep the scope manageable. A real deployment would require proper hashing.
- The single shared ViewModel approach keeps things straightforward but would become harder to maintain as the app grows. For a larger project, scoped ViewModels and a dependency injection framework would be a better choice.
- All data is stored locally with no cloud sync, which limits the app to single-device use.

6. Testing & Evaluation

Testing for AssetPulse was performed using physical Android devices and Android Emulators. The key idea explained to each tester of AssetPulse is that the application needs to be simple. Each quality assurance tester had their own knowledge and method of testing that helped identify potential issues or defects. Quality assurance testers helped validate the current implementations of ideas like login validation, role permissions, asset creation, deletion and editing, search and filter accuracy and check-in/check-out flow.

Evaluations results showed stable performance, accurate data storage, correct access restrictions and correct workflows.

Defects were noticed in the orientation of certain boxes in the UI, odd number formatting, and dashboard metrics not aligning with information and style of application. All defects were brought up during the development phase and corrected for final deployment.

7. Security, Privacy, Accessibility & Compliance

Summarize how the solution addresses ethical, legal, and societal impacts; include security, privacy, and accessibility considerations.

AssetPulse addresses several professional and societal concerns by improving accountability and record accuracy in IT asset tracking. From a privacy standpoint, the utilization of a local save database reduces the applications exposure to third party apps, websites and cloud services thus reducing the chances of privacy incidents. From a security standpoint, the application includes secure user login and role base permissions.

AssetPulse key idea of simplicity greatly helps with its usability and accessibility to users. This is supported through its mobile-friendly interface and straightforward workflows that are designed for everyday users. Export features allows individuals to both audit and verify information through a familiar format.

8. Deployment & Operations

Briefly describe how the system is deployed and operated; link to detailed guides in the appendices.

AssetPulse can be deployed with two different methods, both will utilize the Android operating system.

Method 1 - Windows Installation - The windows installation method allows one to deploy AssetPulse onto a Windows computer using the Android Studio as its emulator.

Method 2 - Deployment using APK - AssetPulse deployment using APK allows one to deploy the application with any physical or virtual device running the Android operating system.

Once deployment has been completed using one the methods listed above, users will not have to repeat the deployment methods. AssetPulse will run as a normal application in your Android environment and will require just a click to begin.

The application runs locally on the physical device or emulator. Data is stored locally in a local Local Room/SQLite database. Users will utilize the touchscreen of their device or mouse to select items. Users can also use their keyboard to input values such as price, date, name of product, etc.

9. Limitations & Future Work

Current limitations of AssetPulse are mainly from the application being both lightweight and simple. With these two key ideas in mind it greatly limited the operating system used, where data is stored and how the application can be used. Currently, AssetPulse is only available to use in an Android environment. This means individuals are only limited to either emulating an Android device or running the application on a physical Android device. Another limitation AssetPulse has is that all data is locally saved. This method of storage is great for security but can have its limitations when trying to have an active backup of data. As a result, AssetPulse works great for small teams but may not work as well for larger organizations.

Future work will prioritize looking into other data storing methods such as utilizing cloud platforms such as azure, amazon or locally hosted cloud storage. This approach is aimed to both expand the application's capabilities and improve backup method options. The next major prioritization for AssetPulse is to begin expansion into other systems such as the Windows operating system. This will allow more users to utilize AssetPulse in their company environment. With both these goals in mind for the future of AssetPulse, the team will also prioritize keeping it both simple and cost effective for all future implementations.

10. References

- Android Developers Documentation
- Kotlin Official Documentation
- Jetpack Compose Documentation
- SQLite Documentation
- Room Persistence Library Docs
- Course Materials / Senior Design Guidance

Appendices (Evidence & How-To)

A. Installation Guide -  AssetPulse Installation Guide.pdf

B. User Manual (task-oriented walkthroughs) -  AssetPulse - User Manual.pdf

C. No administrative task or operations are needed.

D. No API was utilized in this application.

E. Poster (36"×48" horizontal), Slides, and Video Links (Intro, Demo)

- Poster -  IT - AssetPulse.pdf
- Slides -
 - Introduction Slides -  AssetPulse Introduction Slides.pptx
 - Full Presentation -  AssetPulse - Presentation.pptx

- Intro - [AssetPulse Intro Video](#)
- Demo - [AssetPulse Demo Video](#)

F. Scrum Evidence Index (links to Sprint Planning, Dailies, Reviews, Retros)

- Scrum Repository - [Link](#)