

Chapter 1

INTRODUCTION

Background of the Capstone Project

Organizations in the rapidly evolving field of information technology increasingly rely on accurate, centralized, and automated asset management systems to support smooth and reliable operations. As businesses expand, IT departments must handle growing inventories of devices and equipment, requiring tools that provide real-time visibility, lifecycle monitoring, and efficient auditing. Modern asset management practices emphasize features such as QR-based asset identification, automated reporting, and data-driven forecasting, which help reduce errors, strengthen accountability, and support informed decision-making. Without such technologies, organizations face inefficiencies, inconsistencies in records, and delays that negatively impact operational performance.

At MAP Active Philippines, the IT Department continues to depend on Excel-based spreadsheets to monitor its expanding inventory of IT assets. Although spreadsheets previously served as a workable solution, the growth of devices and operational demands has made manual tracking increasingly inefficient, prone to errors, and difficult to maintain. The absence of automation also limits the department's ability to verify accountability, track real-time asset status, and generate timely and accurate reports, resulting in delays and inconsistencies across daily IT operations. This challenge is further emphasized by the company's regional context. While the parent company in Indonesia utilizes a licensed enterprise-level asset management system, the high cost of this solution prevents MAP Active PH from adopting the same system. As a result, the Philippine branch is restricted to manual processes that no longer align with its digital transformation goals or the pace of organizational growth.

Additionally, the lack of a streamlined and cost-effective system directly affects the IT Department's capacity to conduct accurate asset operations such as maintain reliable inventory data, anticipate asset needs, and allocate resources efficiently. Without predictive tools and automated tracking, asset preparations become time-consuming, discrepancies become more likely, and opportunities for proactive maintenance are overlooked. This issue not only impacts MAP Active PH but also reflects a common challenge among organizations in emerging markets which is the need for affordable yet effective asset management solutions that align with global best practices. Implementing a localized, low-cost system with features such as QR scanning and forecasting can significantly enhance operational efficiency and support broader digital transformation efforts without incurring enterprise-level costs.

Problem Statement

The main problem is that MAP Active Philippines' IT Department continues to use manual asset monitoring via Excel spreadsheets because the enterprise-level technology used by its parent firm in Indonesia is too expensive to implement locally. As the organization's inventory grows, it becomes increasingly difficult to keep accurate and up-to-date records of asset allocation, condition, location, and status using spreadsheets.

This leads to time-consuming audits, higher risks of data errors, restricted insight into asset usage and accountability, and an inability to manage resources efficiently. It also limits the department's ability to support data-driven planning and align with the organization's evolving digital transformation and operational needs.

Project Vision and Scope

The goal of this project is to develop a cost-effective and centralized IT Asset Management System (MapAms) that will replace MAP Active PH's manual spreadsheet-based monitoring with a more accurate, efficient, and scalable digital solution. The system intends to

improve the IT Department's ability to track asset operations and maintenance activities, while also boosting operational efficiency and the organization's overall digital transformation goals. Designed specifically for MAP Active PH's scale and resource restrictions, the project aims to create a functional and integrated platform that improves accountability, data consistency, and asset-related procedures.

The scope of this project involves the creation of critical system modules such as authentication, account management, asset operation and maintenance, monitoring and dashboard, reports, context management, and other features. It also explains how to integrate asset requests and tracking with the Helpdesk and Ticket Management Systems and Ticket Tracking System to guarantee that asset-related procedures run smoothly. The system will provide automatic reporting functions to replace manual Excel-based operations, as well as reference modules for suppliers, manufacturers, and categories to ensure data consistency.

Activities and features that go beyond the set objectives and capstone constraints are not included in this project's scope. These include the license or procurement of third-party enterprise asset management solutions, full-scale deployment across departments other than the Rockwell IT office, and tracking of accessories or consumables, unless time allows. The project will also exclude Bring Your Own Device (BYOD) policy implementation, major rebranding or advanced system customization, long-term maintenance or post-deployment support, and architectural redesigns or prototype alterations that go beyond the approved capstone framework. These exclusions guarantee that the project remains focused, attainable, and on track with existing resources and schedules.

Objective and Goals

The main goal of this project is to create and implement an IT Asset Management System that substitutes Excel-based monitoring with a centralized, automated, and scalable

platform. The solution seeks to improve operational efficiency, tracking accuracy, and support MAP Active PH's digital transformation by combining key modules, forecasting capabilities, and smooth asset-related workflows according to the IT Department's requirements.

Specific Objectives

1. To design and implement core system modules that enable secure and centralized asset monitoring, where success is measured by at least 95% accuracy in asset registration and tracking compared to the current Excel-based method.
2. To integrate asset request and tracking processes with the Helpdesk and Ticket Tracking System, enabling unified handling of check-in and check-out activities. This objective will be considered successful if at least 80% of asset requests are processed and monitored through the system, reducing dependency on manual Excel tracking.
3. To develop a forecasting feature that analyzes historical asset usage and product demand to support proactive maintenance and data-driven procurement decisions. Success will be measured by achieving at least 85% forecasting accuracy based on available historical trends.
4. To deliver a fully functional prototype of the IT Asset Management System that is ready for User Acceptance Testing (UAT) and project evaluation. Success will be determined by the system's ability to pass UAT with minimal revisions, demonstrating usability, functionality, and alignment with project requirements.

Significance and Relevance

This study adds to the current body of knowledge on digital transformation and IT asset management by demonstrating how predictive forecasting may be integrated into a small-scale, cost-effective system. Through including forecasting for asset status, utilization patterns, and product demand, the project demonstrates how data-driven decision-making may be used even

in resource-constrained contexts. Additionally, the system demonstrates how regional IT solutions can mirror global best practices in current IT operations. Forecasting capabilities, underpinned by historical asset and utilization data, demonstrate predictive analytics' expanding academic importance in maintenance scheduling, lifecycle management, and procurement efficiency. This establishes the project as a practical case study of how emerging technologies might improve existing asset management approaches without requiring enterprise-level infrastructure.

For MAP Active Philippines, the solution provides a consolidated and automated platform to address major inefficiencies created by manual, Excel-based asset monitoring. Its forecasting capability provides significant operational benefits by enabling the IT Department to anticipate asset depreciation, forecast maintenance requirements, identify potential IT equipment shortages, predict product demand for frequently requested items, support budget planning and purchasing decisions, and reduce unplanned expenses while extending asset lifespan. The solution reinforces accountability, improves audit readiness, and optimizes overall resource allocation by leveraging data-driven insights.

Furthermore, the forecasting capability allows the department to transition from reactive maintenance to proactive planning, better aligning MAP Active's operations with modern digital transformation methods. This feature integrates smoothly with the integrated Helpdesk, Ticket Tracking, and Budget Management systems, resulting in a single and data-driven ecosystem for managing IT resources. Beyond MAP Active, the study proves that predictive technology can be used without incurring the large expenditures associated with enterprise systems. It provides a scalable and cost-effective blueprint for other firms looking to upgrade their IT processes through automation and intelligent forecasting.

Definition of Terms

Asset Management System (AMS). An AMS is a software solution that tracks, monitors, and manages organizational assets. It helps organizations ensure accountability, accurate record-keeping, and proper allocation of resources. The proposed MAP-AMS is designed to replace manual Excel monitoring with a more automated and efficient approach.

Audit. An audit is a systematic review conducted to verify the accuracy and condition of assets. In IT, it ensures that equipment and records are consistent and compliant with company policies. The project's QR-based audit feature will streamline this process by automating asset verification.

Authentication. Authentication is the process of confirming a user's identity before granting access to a system. It usually requires credentials like a username and password, but may also include tokens or biometrics. In this project, authentication ensures that only authorized IT staff can access sensitive asset information.

Authorization. Authorization determines what a verified user is allowed to do within a system. It defines access levels such as administrator, staff, or viewer roles. For MAP-AMS, this ensures that asset records are only updated or modified by users with the correct permissions.

Backend. The backend is the server-side component of an application where data is processed and stored. It manages system logic, database operations, and secure communication. In MAP-AMS, the backend is developed using Django to ensure reliable performance.

BPMN (Business Process Model and Notation). BPMN is a standardized graphical representation of business processes. It provides diagrams that make workflows easy to

understand for both technical developers and business users. This project uses BPMN to document asset management workflows and system interactions.

CI/CD (Continuous Integration / Continuous Deployment). CI/CD is a development practice that automates code testing, integration, and deployment. It allows faster and more reliable updates to software systems. Although not fully implemented in this project, CI/CD principles influence the way features are iteratively developed and tested.

Cloud Deployment. Cloud deployment refers to hosting an application on cloud servers accessible via the internet. It removes the need for physical infrastructure and allows scalability and flexibility. MAP-AMS is deployed on Railway's free cloud tier for accessibility during development and testing.

Database. A database is an organized collection of structured information stored electronically. It enables efficient storage, retrieval, and management of large volumes of data. In MAP-AMS, the database is used to keep track of IT assets, users, and audit logs.

Data Encryption. Data encryption is the process of converting information into a coded format that can only be accessed with a decryption key. It protects sensitive information from unauthorized access. Although the project uses basic encryption, it ensures user credentials and asset data are secured.

Dashboard. A dashboard is a visual interface that displays key data and performance metrics in real time. It provides users with quick insights into system operations. MAP-AMS includes a dashboard that summarizes asset statuses, audit progress, and requests.

Django. Django is a high-level Python web framework used for backend development. It offers built-in features like authentication, database integration, and security modules. MAP-AMS uses Django to manage asset records, user accounts, and system logic.

Encryption Key. An encryption key is a unique string of data used to encrypt and decrypt sensitive information. Without the correct key, encrypted data cannot be accessed. This ensures that even if data is intercepted, it remains unreadable to unauthorized parties.

Forecasting. Refers to the process of analyzing historical data to predict future trends, conditions, or events. In IT asset management, forecasting helps determine when equipment may require maintenance, when replacements will be needed, and how asset usage patterns may change over time. In this project, the forecasting feature supports proactive planning by generating data-driven predictions that help the IT Department prepare for future asset demands and lifecycle changes.

Frontend. The frontend is the user-facing part of an application. It is responsible for visuals, layouts, and interactions that users directly engage with. In this project, the frontend is developed using React to create a responsive and interactive interface.

Git. Git is a version control system that tracks changes in code during development. It allows developers to collaborate, merge updates, and revert to previous versions when needed. The project team uses Git for code management and collaboration.

GitHub. GitHub is a cloud-based platform built on Git that hosts code repositories. It provides tools for collaboration, issue tracking, and version control. MAP-AMS uses GitHub as the main repository for storing and managing its source code.

Helpdesk System. A helpdesk system manages support requests and IT-related issues. It organizes tickets so that issues can be tracked and resolved systematically. MAP-AMS integrates with a helpdesk system to handle asset-related service requests.

HTTPS (Hypertext Transfer Protocol Secure). HTTPS is a secure version of HTTP used for communication between a client and server. It encrypts the data exchanged to protect

sensitive information. When deployed, MAP-AMS will use HTTPS to safeguard asset data transmissions.

Login Credentials. Login credentials are information such as usernames and passwords used to authenticate a user. They serve as the first line of defense in system security. In MAP-AMS, credentials ensure that only authorized IT staff can access the system.

Middleware. Middleware is software that connects different applications, systems, or services. It acts as a bridge that allows smooth communication and integration. In MAP-AMS, middleware may handle requests between the frontend and backend.

Predictive Analytics. A data analysis approach that uses statistical techniques, algorithms, and historical records to forecast likely future outcomes. It goes beyond simple reporting by identifying patterns and relationships within data to generate actionable insights.

Prototype. A prototype is an early version of a system created to test ideas and functionality. It allows users and developers to evaluate features before the final product is released. The MAP-AMS prototype demonstrates asset monitoring, auditing, and reporting features.

QR Code (Quick Response Code). A QR code is a machine-readable code that stores digital information. When scanned, it quickly retrieves the stored data using a mobile device or scanner. MAP-AMS uses QR codes to identify IT assets.

Quality Assurance (QA). QA is the practice of testing and reviewing software to ensure it meets quality standards. It helps identify bugs, errors, and inefficiencies. The project team conducts QA to validate that MAP-AMS works as intended.

Railway. Railway is a cloud hosting platform used to deploy applications. It provides free and paid tiers, making it accessible to student developers. MAP-AMS uses Railway's free tier for cloud deployment and testing.

React. React is a JavaScript library used to build dynamic and interactive web interfaces. It allows developers to create reusable UI components. MAP-AMS uses React for its frontend to ensure a user-friendly interface.

Reports Module. A reports module generates structured outputs that summarize system data. It provides insights such as asset usage, maintenance history, and audit results. MAP-AMS includes a reports module to replace manual Excel reporting.

Role-Based Access Control (RBAC). RBAC is a method of restricting system access based on assigned user roles. It ensures that users only perform actions relevant to their responsibilities. In MAP-AMS, RBAC prevents unauthorized modifications to asset records.

Source Code Repository. A source code repository is a storage location for software code and related files. It supports collaboration, versioning, and backups. The MAP-AMS repository is maintained on GitHub.

Sprint. A sprint is a fixed period of time in Agile development where specific tasks or features are completed. It encourages iterative progress and constant feedback. The project follows sprints to manage time effectively within the capstone schedule.

Ticket Management System. A ticket management system organizes and tracks issues or requests submitted by users. Each issue is logged as a "ticket" until resolved. MAP-AMS integrates with a ticket system to process asset-related concerns.

User Acceptance Testing (UAT). UAT is the final testing stage conducted by end-users. It determines if the system meets requirements and is ready for deployment. MAP-AMS will undergo UAT to confirm usability and accuracy before final approval.

Version Control. Version control is the practice of tracking changes in code over time. It ensures collaboration without overwriting and allows rollback to previous versions. The project uses Git and GitHub for version control.

Structure of Document

This document is divided into five key parts, each representing a critical milestone of the project. The first section, Introduction, lays the groundwork for the study by discussing the background of manual IT asset monitoring and the need for a cost-effective IT Asset Management System with forecasting capabilities. It comprises the project's history and motivation, the issue description, the system's vision and scope, and the general and specific goals. This part also examines the project's theoretical and practical implications, defines key technical terms, and gives an overview of the document's structure.

The second part, entitled Related Literature, Studies, and Technical Background, examines current literature, theories, and previous studies on asset management, digital transformation, and forecasting technologies. It also provides a technical foundation for the study by examining approaches such as Agile Scrum, Microservices Architecture, DevOps practices, and enterprise integration, as well as the conceptual framework and theoretical paradigm that underpin the project.

The third section, Methodology and Project Management, discusses the Agile Scrum framework used in the development process, including team roles, responsibilities, sprint activities, and workflow organization. It also describes the architectural design decisions,

development procedures, quality assurance methodologies, and the innovation framework used to guide system implementation.

The fourth section, System Design, Development, and Implementation, discusses the system requirements, design specifications, and architectural model. It comprises UML diagrams, BPMN workflows, database structures, and wireframes, as well as documentation of the development stages, integration procedures, deployment and migration strategy, testing findings, and evaluation of the system's performance against its set objectives.

The final section, Summary, Conclusion, and Recommendations, summarizes the project's findings, draws conclusions based on the identified problem and objectives, and makes recommendations for improving the system, supporting its potential adoption within the organization, and guiding future research on asset management and forecasting technologies. The appendices include supplemental documents such as technical documentation, project management records, testing outputs, user manuals, and defense-related items that validate the project's processes and results.

Chapter 2

RELATED STUDIES AND LITERATURE REVIEW

Challenges in Manual Asset Tracking

One of the challenges in manual tracking of assets is inventory record inaccuracy. Occurs when the recorded number of assets does not match their real presence, can cause operational inefficiencies and financial losses in manual asset monitoring. According to (Destro et al., 2023) inventory record inaccuracy severely impairs warehouse efficiency, which is frequently made worse by the use of manual count techniques. This includes decreased picking productivity and missed sales. In addition to being labor-intensive and time-consuming, manual tracking results in poor data visibility and weaker audit trails due to frequent human data-entry errors and delays in record updates (Abdelmoti et al., 2025) Additionally, a quantitative investigation revealed that inconsistencies can continue at non-negligible rates (e.g., 0.5% or more) even in inventory systems that are primarily manual, hurting decision-making and confidence in asset data Linuwih and .Handayati (2025.)

To lower the need for manual tracking and minimizing human error, automation in IT or more general asset management systems greatly needed to improve real-time visibility and accuracy of asset inventories. According to (Alhadi et al., 2024) using digital twins in asset management, automation makes predictive maintenance possible, which reduces long-term maintenance costs and increases asset utilization. Through automating repetitive, rule-based processes, robotic process automation lowers operational risk and simplifies workflows for IT asset management lifecycle functions like discovery, retirement, and compliance monitoring (ResearchGate, 2025). Additionally, practitioner assessments demonstrate that automated IT asset management systems reduce costs through preemptive retirement of unused assets and

better resource allocation, while also assisting in the more reliable enforcement of security and license compliance (Cflow, 2025; Moveworks, n.d.).

Asset Management System with Forecasting

IT Asset Management Systems is an essential platform for comprehensive lifecycle governance of IT assets. Al-Shatri et al., (2024) define IT-AMS as a cross-functional process that unites inventory, contract management, financial, and risk controls for strategic oversight and efficiency. The researcher team underscores the importance of a “birth-to-death” lifecycle paradigm such as Installs, Moves, Adds, and Changes to ensure accurate inventory and cost accountability. Meanwhile, Giva (2022) and AssetLoom (2025) highlight how modern IT-AMS systems increasingly incorporate automation, sustainability priorities, and audit trails to improve decision-making and align with broader enterprise goals.

Adapting modern IT-AMS needs asset lifecycle management that coordinates procedures for planning, acquiring, operating, maintaining, and retiring assets in order to optimize value and lower risk across their lifetime. According to (Negri et al., 2021), digital twins, IoT sensors, and predictive analytics are important components of contemporary asset lifecycle management that increase the precision of maintenance forecasting and decision-making. Asset lifecycle management integration with corporate systems improves traceability, sustainability performance, and cost optimization across asset portfolios (Lee et al., 2022). Additionally, in order to guarantee compliance, resilience, and an efficient long-term asset strategy, companies are increasingly implementing lifecycle-centric governance frameworks (Mahmood & Bashir, 2023).

Companies and organizations are beginning to see digital transformation as a strategic necessity that combines data-driven processes, digital technology, and organizational change to improve value creation. According to (Jonathan et al., 2023), digital transformation effectiveness

depends on developing digital leadership, workforce capabilities, and flexible structures in addition to using new technologies. Cloud use, Automation, Predictive Analytics and AI-driven decision systems boost operational resilience and efficiency, especially in post-pandemic contexts, according to industry evaluations (World Economic Forum, 2022). Study from (Susanti et al., 2023), firms can link transformation projects with long-term business objectives by using digital maturity frameworks.

Predictive analytics is one of the digital transformations that has become a cornerstone in Information Technology applications, particularly within Asset Management Systems. Organizations seek to anticipate failures, optimize maintenance, and enhance operational efficiency. Adebisi et al., (2021) developed a conceptual model for predictive asset integrity management, showing how data analytics can improve reliability and safety in oil and gas operations while reducing costs. Similarly, Ojha and Kumar (2022) emphasized the role of scalable artificial intelligence models in predictive failure analysis for cloud-based AMS, highlighting the use of machine learning and deep learning to process real-time data streams and proactively identify potential breakdowns. Extending this perspective, Kashif and Chowdhury (2024) provided a comprehensive review of AMS technologies, situating predictive analytics within broader trends such as IoT, blockchain, and cloud computing, and underscoring its importance in enabling proactive decision-making and sustainable asset utilization.

Forecasting has become a critical component of modern Asset Management Systems. It enables organizations to anticipate equipment performance and optimize resource allocation. Iluore, Onose, and Emetere (2020) developed a real-time equipment monitoring model that integrates forecasting techniques to improve industrial asset reliability and reduce downtime. Additionally, according to (Ferrer et al., 2022) machine learning applications in asset management, highlighting predictive and forecasting models as essential for risk reduction and strategic planning. Bartram, Branke, and Motahari (2020) further emphasized the role of artificial

intelligence in forecasting within AMS, noting its capacity to enhance decision-making and long-term asset sustainability.

The analyzed literature collectively demonstrates how the need for automation and digitally empowered asset management systems is driven by the shortcomings of manual asset tracking, including inaccuracy, human error, and inadequate audit trails. Utilizing connection of asset lifecycle management with real-time monitoring, emerging technologies like digital twins, IoT sensors, and predictive analytics allow businesses to move beyond reactive maintenance to proactive forecasting and strategic decision-making. These developments are in line with researchers' implementation of digital transformation in asset management on MAP Active Philippines, where operational resilience and long-term asset value are strengthened by cloud usage, AI-driven models, and data-centric governance frameworks. Overall, the studies show that contemporary AMS solutions are integrated systems that improve productivity, compliance, sustainability, and organizational preparedness for future demands rather than merely being tools for inventory correctness.

SNIPE-IT Asset Management System

One of the widely used asset management systems is the Snipe-IT. It was created to assist businesses in keeping track of their hardware, software, licenses, and consumables over their entire lifecycle. To guarantee accurate and safe asset visibility, the platform offers role-based access control, automated audit logs, QR/barcode compatibility, and check-in/check-out procedures. It is appropriate for both small and big IT settings because of its API, which allows interaction with automation scripts and external systems (Snipe-IT, 2024). Additionally, it is widely used by businesses, academic institutions, and governmental organizations since it is web-based and open-source under the GNU AGPL license.

However, despite its powerful inventory and audit capabilities, Snipe-IT is limited in areas such as predictive lifecycle management and real-time device discovery. The system does not natively perform automated asset scanning, requiring third-party tools or manual entry to populate records. Studies and industry reports on TechRadar Pro (2023) suggest that many organizations using Snipe-IT rely on external discovery tools such as OCS Inventory or Lansweeper for automatic asset detection. This indicates that while Snipe-IT excels at asset tracking and accountability, it is less equipped to support advanced ITAM functions such as forecasting, total cost of ownership modeling, or AI-driven asset optimization.

GLPI Asset Management System

GLPI is an open-source platform for asset management and IT service management that offers features including inventory monitoring, help desk ticketing, license management, contract management, and service catalog operations. GLPI is frequently used in business environments that call for a complete ITSM suite because it supports plugins, REST APIs, and authentication systems like LDAP and Active Directory (Teclib, 2023). Also, it is one of the most adaptable open-source IT management systems because of its modular design, which enables enterprises to modify how assets, support operations, and maintenance tasks are recorded.

Nevertheless, higher complexity is a trade-off for GLPI's large feature set. When establishing inventory integrations or workflow automation, organizations frequently experience high learning curves and configuration needs (LinuxHint, 2024). GLPI still lacks sophisticated analytics and predictive modeling for asset lifespan forecasts, although having some automation through plugins, such as automatic device discovery. While GLPI excels at operational management, the literature on ITSM solutions notes that it mostly depends on third-party integrations to accomplish more complex ITAM functions, like automated lifecycle optimization or AI-assisted resource planning.

OCS Inventory NG

When it comes to hardware, software, and configuration data from networked devices are automatically scanned and gathered by OCS Inventory NG, an open-source asset discovery and inventory application. Large-scale deployments and centralized device monitoring are made possible by its use of lightweight agents for Windows, macOS, and Linux endpoints (OCS Inventory NG, 2024). It is often combined with GLPI to provide a comprehensive ITAM and ITSM architecture because of its robust automatic discovery capabilities. OCS Inventory NG is used by organizations to keep up-to-date device listings and guarantee precise network infrastructure visibility.

Although this asset management system is useful for inventory identification, OCS Inventory NG does not work as an independent full asset management system. It lacks lifecycle tracking, contract management, check-in/check-out workflows, and license governance, requiring pairing with external platforms for complete ITAM coverage (Fosnock, 2023). According to reports, its analytics capabilities are restricted to simple dashboards, and its user interface is less advanced than competing products. As a result, while OCS Inventory NG tackles the challenge of real-time device detection, it does not answer broader asset lifecycle challenges such as anticipating replacements, optimizing budgets, or automating compliance evaluations.

Comparative Matrix of Existing Asset Management Systems

Table 1

Comparative Matrix

Criteria	Snipe-IT	GLPI	OCS Inventory NG
Type / Category	IT Asset Management System	ITSM + Asset Management Platform	Asset Discovery & Inventory Tool
Core Functionality	Asset tracking, check-in/out, audit logs, QR/barcode labels	Asset inventory, helpdesk, contracts, license mgmt., ITSM modules	Automatic device discovery, software/hardware inventory
Automation Capabilities	Limited automation; relies on API integrations	Moderate automation through plugins; workflow rules possible	Strong automated discovery; minimal lifecycle automation
Lifecycle Management	Basic (assignments, status updates only)	Moderate (supports warranty, contracts; no predictive lifecycle)	None (inventory-only; no lifecycle tracking)
Predictive / Analytics Features	None; basic reporting	Limited; no predictive analytics	None; basic dashboards only
Scalability	Good for small to medium organizations	High scalability for enterprise environments	Scales well for large networks
Ease of Use	Very user-friendly, simple UI	Complex interface; steep learning curve	Technically oriented; UI less modern

Table 1
Comparative Matrix

Criteria	Snipe-IT	GLPI	OCS Inventory NG
Integration Support	Strong API; can integrate with discovery tools	Strong plugin ecosystem; supports OCS Inventory integration	Often used as backend discovery tool for GLPI
Strengths	Easy to deploy, open-source, good audit logs	Full ITSM suite, customizable, enterprise-ready	Excellent automated device discovery
Weaknesses / Limitations	No forecasting, no automated discovery	Requires complex setup, limited analytics	Lacks lifecycle mgmt., weak reporting
Gaps Not Solved	No predictive lifecycle analysis; no automated forecasting	Does not provide AI-driven lifecycle insights	Does not manage asset lifecycle or planning functions

The table shows a comparison matrix demonstrating that existing open-source asset management solutions differ greatly in capability. Snipe-IT has effective asset tracking and accountability tools, but it lacks automated discovery and lifetime forecasts. GLPI provides a comprehensive ITSM and asset management package with great customisation options, but its predictive analytics are restricted and complicated to configure. OCS Inventory NG excels in automatic hardware and software discovery, but it lacks end-to-end lifecycle management, requiring other systems for full IT asset management capability. These gaps highlight a common weakness across all platforms: (1) none provide automated lifecycle forecasting, (2) predictive analytics, or (3) fully integrated ITAM automation, indicating a significant opportunity for system improvement.

Map Active Philippines Asset Management System (MapAms)

Compared to other open-source systems like Snipe-IT, GLPI, and OCS Inventory NG, the MAP Active Philippines Asset Management System (MapAms) offers a more unified and intelligence-driven approach to asset lifecycle management. Existing tools mostly focus on tracking, ITSM workflows, or device identification, but none provide seamless coordination of asset operations, maintenance, and predictive analytics. MapAms fills these gaps by providing extensive modules for asset model management, asset auditing, maintenance scheduling, and component-level control, all backed by role-based access for administrators and operators.

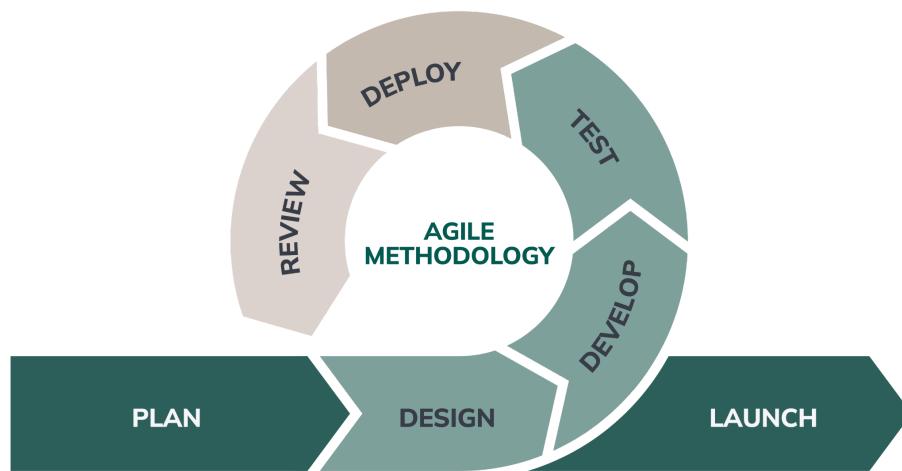
Unlike other systems, MapAms connects directly with the organization's independent Helpdesk and Ticket Tracking System, automating check-in and check-out workflows and maintaining consistency between asset modifications and ticket activity. In particular, MapAms includes forecasting features, such as asset status prediction and asset model demand prediction, which enable enterprises to anticipate asset requirements, replacements, and lifecycle hazards. This predictive and integrated architecture raises MapAms above standard asset management systems, making it a more proactive and comprehensive solution for organizational asset operations.

Agile Scrum Methodology Overview

The IT Asset Management System (MapAms) for MAP Active Philippines is being developed using the Agile Scrum approach, which is an iterative and adaptable framework that supports quick development and continuous improvement. Agile Scrum emphasizes short development cycles, known as sprints, that enable the team to deliver incremental features while responding to changing needs. This method encourages close communication among developers, product owners, and stakeholders, ensuring that input is quickly integrated throughout the project. Using this methodology shows that MapAms is created in a controlled

but adaptive manner utilizing Agile Scrum, resulting in faster delivery, higher quality, and greater alignment with business needs.

Figure 1. Agile Scrum Process



According to Behrens et al. (2021) agile methodology is an iterative and incremental approach to software development that prioritizes flexibility, frequent customer feedback, and collaboration among self-organizing teams. Rather than comprehensive prior planning, Agile operates in short cycles known as iterations or sprints that involve planning, design, execution, testing, and review, allowing for ongoing product refinement. This methodology emerged as a response to rigid, traditional development processes by emphasizing values such as individuals and interactions over processes and tools, working software over extensive documentation, and adapting to change over sticking to a plan (Hossain et al., 2021). Additionally, agile's flexible structure enables teams to deliver high-quality functioning software more frequently while pivoting in response to changing needs.

Agile scrum is the most appropriate technique for productive development since it is a lightweight, iterative framework inside the broader Agile methodology that organizes work into short, time-boxed cycles known as sprints to enable rapid value delivery. It focuses on

cooperation, continual feedback, and adaptability, enabling development teams to adjust swiftly to changing requirements. Serrador and Pinto (2015) found that Scrum procedures improve project performance by boosting team communication, increasing flexibility, and strengthening control over project deliverables. Their research indicated that Agile techniques, particularly Scrum, improve both efficiency and stakeholder satisfaction by encouraging regular inspection and transparent progress monitoring. As an Agile umbrella framework, Scrum provides standardized roles, ceremonies, and artifacts to help teams generate high-quality outputs iteratively.

Additionally, Agile Scrum is widely regarded as superior to more traditional, structured development approaches because of its significantly higher project success rates and lower risk of failure. According to Mishra and Alzoubi's (2023) comparison study, Agile efforts had a success rate of roughly 40%, while traditional waterfall projects had a completion rate of only 15%, with failure rates of 10% and 30%, respectively. This evidence demonstrates that Agile's iterative, feedback-driven cycles help teams negotiate uncertainty more effectively, adjust faster, and provide more value than inflexible, linear approaches.

In the development of the MAP Active Philippines IT Asset Management System, Scrum Artifacts play a critical role in ensuring structured progress and alignment with stakeholder needs. The product backlog lists all functional requirements, including asset operations and maintenance, checkin and checkout ticket integration, forecasts, and dashboard features. During each sprint, the sprint backlog organizes tasks such as asset module implementation and ticketing process integration. The increment symbolizes MapAms's functioning version at the end of each sprint, bringing the system closer to its ultimate objective of being completely functional and integrated. Using these artifacts, the project guarantees that each development cycle provides concrete improvements, meets end-user objectives, and follows a clear path to the final solution.

In terms of Scrum events, a sprint is a time-boxed iteration in which the development team works to fulfill a set of prioritized tasks from the sprint backlog. Each sprint produces an increment of the product that could be shipped or used, allowing stakeholders to monitor progress and provide input for continual improvement. Sprints for the MapAms are planned every two weeks, allowing the team to concentrate on small, achievable objectives like creating forecasting capabilities, integrating ticketing procedures, or building the asset module. The team can swiftly adjust to changing needs, deal with problems as they emerge, and make sure that each iteration of MapAms gradually advances toward a fully working and integrated system by breaking development up into several brief cycles. Additionally, the two-week sprint framework encourages frequent evaluation, cooperation, and improvement, guaranteeing that the project stays in line with organizational goals and stakeholder needs.

Next is the Sprint Review. This is a scrum event when the development team shows stakeholders the work finished during the sprint so they may comment, discuss, and validate the progress. This event guarantees openness and offers a chance to compare the product increment to the sprint objectives, assisting in determining areas that need to be improved or adjusted for future sprints. The team delivers finished modules or progress at the weekly Sprint Review with the Product Owner in the MapAms development process. Frequent evaluations enable ongoing cooperation and guarantee that the system meets stakeholder expectations. This enables the team to quickly integrate comments and continue moving forward with the delivery of a fully working, integrated asset management system.

For the Sprint Retrospective, it is a Scrum event that takes place at the conclusion of each sprint to evaluate the team's performance, procedures, and cooperation with the goal of ongoing improvement. The team determines what worked well, what difficulties or roadblocks were faced, and which parts of the process or communication need to be modified for subsequent sprints. Retrospectives are held at the conclusion of each two-week sprint in the

development of the MapAms to assess module progress. The team may adopt practical tactics, increase collaboration, and refine development procedures by methodically assessing areas for improvement and successes. This will ultimately ensure more efficient delivery of a fully functional and integrated asset management system.

Last but not least is the Daily Stand-up, a brief, time-limited Scrum meeting that takes place every day to coordinate the development team, assess progress, and pinpoint any potential roadblocks. Every team member usually presents what they have accomplished since the last meeting, what they intend to focus on next, and any difficulties or roadblocks they are encountering during the stand-up. Every workday, the MapAms team holds stand-up meetings where members can discuss their progress on projects. These frequent check-ins ensure that the project stays on course and that any necessary adjustments can be made quickly to continue progress toward providing a fully working and integrated asset management system. They also foster accountability, transparency, and early issue discovery.

Additionally in Scrum, there is an assigned position or role that directs the project's advancement. The product owner is in charge of establishing the product's vision, setting requirements priorities, and making sure the development team produces value that meets stakeholder expectations. Throughout the project, the Product Owner decides which features to prioritize, manages the product backlog, and clarifies requirements. Mr. Elvin Punzalan, the head of Map Active Philippines' IT department, is the product owner for the MAP Active Philippines IT Asset Management System (MapAms). In order to guarantee that the system satisfies organizational objectives, he leads the team by establishing priorities for modules and evaluating the team's progress during sprint reviews, and offering ongoing input.

Next is the Scrum Master, this role serves as a facilitator for the team, ensuring adherence to Scrum principles, removing obstacles, and promoting continuous improvement.

Also, the Development Team in Scrum consists of cross-functional members responsible for designing, building, testing, and delivering product increments, working collaboratively to meet sprint goals. In the development of the MapAms, Ma. Elisa Garrote acts as both the Scrum Master and Project Manager, guiding the development team through sprint planning, daily stand-ups, sprint reviews, and retrospectives. She ensures that tasks related to AMS modules are completed efficiently, supports team coordination, and addresses any impediments to progress. Next is Edzra Macas who acts as Lead backend developer. She is in charge of the development and administration of the asset modules. She makes sure that user authentication, effective data handling, and integration with external systems like the Helpdesk and Ticket Tracking System are supported by the backend infrastructure. Edzra guarantees that MapAms provides dependable speed, safe data management, and seamless operation across all modules by spearheading backend development. We also have Johanna Mae Sillano, who serves as the Lead Frontend Developer for the MapAms. She is responsible for designing and implementing the user interface, ensuring that the system is visually appealing, intuitive, and easy to navigate for both administrators and operators. Johanna Mae develops and integrates the frontend components for modules and ensures seamless interaction with the backend services. The Lead Frontend Developer makes sure that MapAms provides a user-friendly experience while adhering to the project's functional requirements and overall system objectives by concentrating on usability, responsiveness, and accessibility. Lastly, Bengie Villesco is responsible for Quality Assurance (QA) in the development of the MapAms. The QA role ensures that the system meets the required standards of functionality, reliability, and performance by conducting systematic testing, identifying bugs, and verifying that all modules perform as intended. Bengie oversees the testing of all modules as well as the integration with external systems like the Helpdesk and Ticket Tracking System. Through rigorous testing and validation, the Quality Assurance role guarantees that MapAms delivers a stable, high-quality, and fully

functional asset management system that aligns with organizational objectives and end-user expectations.

Agile Scrum is used in the MAP Active Philippines IT Asset Management System because of its intrinsic flexibility, which enables the development team to promptly adjust to evolving requirements and stakeholder input. All AMS modules are delivered progressively and continuously improved for enhanced functionality because of its iterative development. Additionally, by encouraging consistent communication and cooperation, Agile Scrum facilitates improved interaction with other groups, such as the Helpdesk, Ticket Tracking System, and Budget Management System. Agile Scrum is perfect for MapAms because of these characteristics, which guarantee that the system changes adaptably while being in line with project goals and organizational requirements.

Microservices Architecture

Microservices architecture is a software design technique that divides an application into a set of small, independently deployable services, each responsible for a certain business capability. According to Bigelow and Gillis (2023), these services communicate using lightweight APIs and keep their own data, allowing teams to create, launch, and grow them separately. This architecture promotes modularity, decentralized governance, and resilience because errors in one microservice are less likely to propagate throughout the system. Organizations that structure their systems in this manner gain higher agility, scalability, and maintainability than traditional monolithic architectures.

The concept of microservices originated in the mid-2000s, when Peter Rodgers developed Micro-Web-Services as a method for creating loosely linked components. This concept emerged over time through software architect workshops between 2011 and 2012, when the term Microservices was explicitly chosen to denote independently deployable,

fine-grained services. In the 2010s, large-scale users such as Netflix helped popularize the architecture, demonstrating how microservices could provide higher scalability and robustness than monolithic systems (Foote, 2021).

Core Principles of Microservices

Moving into the core principles, decentralization is a fundamental principle of microservices architecture, in which each service is self-governing and accountable for its own logic and data, hence decreasing shared dependencies and coupling. MapAms implements this approach by splitting the system into four autonomous services: (1) authentication, (2) asset management, (3) context, and (4) notification where each of which stores and manages its own data without relying on a central database. This is consistent with the Database-per-service paradigm, in which each microservice has its own dedicated datastore, allowing schema and storage technology to be adapted to its unique requirements (GeeksforGeeks, n.d.). MapAms implements this design to achieve strong service autonomy, fault isolation, and independent scalability, which improves resilience and enables continuous deployment.

Additionally, autonomy is a major principle in microservices design, which means that each service maintains its own logic, data, and deployment without being tightly coupled to others (Adarsh, 2022). In MapAms, this autonomy is represented by four independent services which are the authentication, asset, context, and notification, each of which runs and evolves independently and owns its own data. Communication between these services occurs via a central API Gateway, which sends external requests to the asset service while allowing that service to internally call the context service without disclosing internal dependencies. The usage of an API Gateway ensures loose coupling which is also one of the principles of microservices and service encapsulation, allowing each service to be created, launched, and scaled

independently. MapAms improves its modularity, resilience, and maintainability by combining autonomous services with API-based interactivity.

Finally, scalability in microservices refers to each service's capacity to scale independently in response to workload needs, allowing systems to remain efficient and responsive as consumption grows. According to Dragoni et al. (2017), microservices enable fine-grained scaling, allowing enterprises to dedicate resources to specific services rather than the entire program. In MapAms, this idea is utilized by dividing the system into independent services such as authentication, asset management, context, and notification so that heavily used modules, such as assets or context, can scale without harming others. This autonomous scaling, together with the API Gateway routing architecture, enables MapAms to handle additional organizational data, user traffic, and system complexity while preserving performance and dependability.

Benefits of Microservices

Microservices design provides considerable benefits in scalability, modularity, and maintainability by allowing services to be built, deployed, and scaled independently, as well as isolating failures to specific components (Domakonda, 2025). In particular, this architecture allows for autonomous deployment and resource allocation per service, lowering the likelihood that changes in one domain would influence others and simplifying system evolution. This means that MapAms' microservices such as authentication, asset service, context service, and notification service can all be grown and maintained without impacting the platform as a whole. Such decoupling improves agility by allowing teams to change or redesign a specific service without redeploying the entire system, which also simplifies maintenance and decreases operational risk.

Challenges in Microservices

One important problem of microservices is managing inter-service communication and ensuring consistent API interactions, especially as the number of services increases (Newman, 2015). In MapAms, this is important since the four services must share data efficiently via APIs. Failures or version incompatibilities in any service can disrupt workflows like asset updates or context enrichment, emphasizing the importance of proper API design, monitoring, and error management.

Microservices also add operational complexity by requiring several separate services to be deployed, monitored, and maintained (Richardson, 2018). For MapAms, each microservice has its own PostgreSQL database and is deployed on Railway, so the team must manage environment configurations, health checks, and CI/CD pipelines to keep modules like asset management, forecasting, and reporting synchronized and functional after updates.

DevOps Culture and CI/CD Practices

According to Suescún-Monsalve et al. (2021), DevOps is a cultural and organizational model that promotes collaboration between software development and IT operations teams to reduce development time and enhance service reliability. It encourages shared accountability, continuous feedback, and automation throughout the software delivery process by incorporating techniques like Continuous Integration (CI), Continuous Delivery (CD), and infrastructure-as-code. Through breaking down boundaries and fostering cross-functional collaboration, DevOps enables faster, more frequent, and higher-quality deployments. This allows enterprises to respond more flexibly to changing business needs, lower deployment risks, and expedite innovation.

In terms of its core principle, DevOps encourages collaboration and communication between development and operations, breaking down silos and ensuring both teams share

ownership for the software lifecycle (Moezz et al., 2024). In this model, teams work closely together, share goals, and provide constant feedback, resulting in a common culture dedicated to providing high-quality, stable releases. This approach applies to your MapAms capstone project, requiring that the development team and operations collaborate to construct and maintain services such as asset management, context, and authentication. By adopting a collaborative mentality, MapAms provides seamless delivery and issue resolution across functional boundaries.

DevOps also focuses on automating build, test, deployment, and infrastructure tasks to improve delivery and eliminate human error (Moezz et al., 2024). CI/CD pipelines, infrastructure-as-code, and automated testing are critical tools for ensuring that changes are continuously integrated and distributed reliably. Automated pipelines in the MapAms system may conduct microservice tests, check API contracts, and deploy new versions without requiring manual involvement. This automation improves system stability and accelerates iteration cycles.

Another key DevOps principle is continuous measurement and monitoring, providing constant visibility into performance, reliability, and process efficiency (Moezz et al., 2024). Using monitoring and feedback loops, teams may notice bottlenecks, failed deployments, and other issues early on and respond quickly. MapAms collects telemetry from microservices, tracks key indicators like response time and error rates, and adjusts resources or workflows as appropriate. This data-driven strategy enables preventive maintenance and continual improvement.

Small development teams benefit considerably from DevOps adoption, which improves delivery performance and operational collaboration. Korkmaz and Aydin's (2025) empirical study discovered that small teams structured with high collaboration between development and operations had significant gains in deployment frequency, incident recovery, and dependability. This methodology is perfectly aligned with MapAms' team structure, in which developers,

operations, and QA work closely together to provide continuous integration and delivery across microservices. Adopting DevOps methods allows your small team to boost release velocity, reduce failures, and retain better agility, thereby improving both development efficiency and system stability.

Continuous Integration (CI) is a DevOps strategy that involves merging code changes into a shared repository and automating builds and tests to find integration issues early (Raheem et al., 2024). This technique improves software quality by providing immediate feedback and decreasing the accumulation of errors that normally arise when integration is delayed. CI also supports tiny, incremental changes, which improves codebase stability and speeds up delivery cycles. MapAms uses CI to ensure that modifications to its microservices are evaluated on a continual basis, hence maintaining consistency and stability throughout development. On the other hand, Continuous Deployment (CD) is a DevOps strategy that automates code updates for production, ensuring reliable software deployment at any time (Raheem et al., 2024). Continuous Deployment goes a step further by automatically releasing verified changes to production without manual intervention, allowing for faster delivery of features and issue fixes. CD promotes modest, incremental releases, which improves system dependability, decreases deployment risk, and increases responsiveness to user feedback. In the context of MapAms, CD ensures that upgrades to microservices such as asset management, context, notification, and authentication are safely and effectively deployed, preserving system stability while accelerating feature delivery.

DevOps uses tools to automate, orchestrate, and streamline the software development lifecycle, increasing productivity and reliability. GitHub Actions automates processes, testing, and deployments straight from a repository, allowing development teams to quickly assess changes (Sharma et al., 2022). Docker containerizes microservices, allowing them to execute consistently across several environments while simplifying dependency management, scaling,

and deployment (Pahl, 2021). Railway functions as a cloud platform for deploying and managing applications and databases, providing infrastructure abstraction and monitoring capabilities. This is especially useful for MapAms, where microservices such as asset management, context, notification, and authentication require dependable deployment and runtime management.

Enterprise Architecture & System Integration

Enterprise Architecture is a structured discipline that establishes a consistent set of concepts, models, and methodologies for aligning an organization's business processes, information systems, and technology infrastructure with its strategic objectives (Alwadain, 2020). It provides a holistic blueprint for an enterprise's existing "as-is" and future "to-be" states, allowing for systematic planning and transformation. Enterprise Architecture serves as both a technological foundation and a governance mechanism, guiding decision-making, optimizing IT investments, and enabling digital transformation. Establishing this holistic architectural view can help firms increase agility, scalability, and coherence across business and IT domains.

The Open Group Architecture Framework (TOGAF) is one of the examples of an enterprise architecture framework that provides an organized approach to designing, planning, executing, and managing an organization's information systems and technological landscape. It is based on the Architecture Development Method, a step-by-step process for integrating business, data, application, and technological architectures (Ruiz, Araujo, & Alarcon, 2024). TOGAF enables enterprises to connect IT activities with business goals by providing standardized stages, artifacts, and best practices for architecture development. This makes it ideal for projects like MapAms, which involve the coordinated design of many services, interconnections, and supporting infrastructure.

Figure 2. Open Group Architecture Framework (TOGAF)

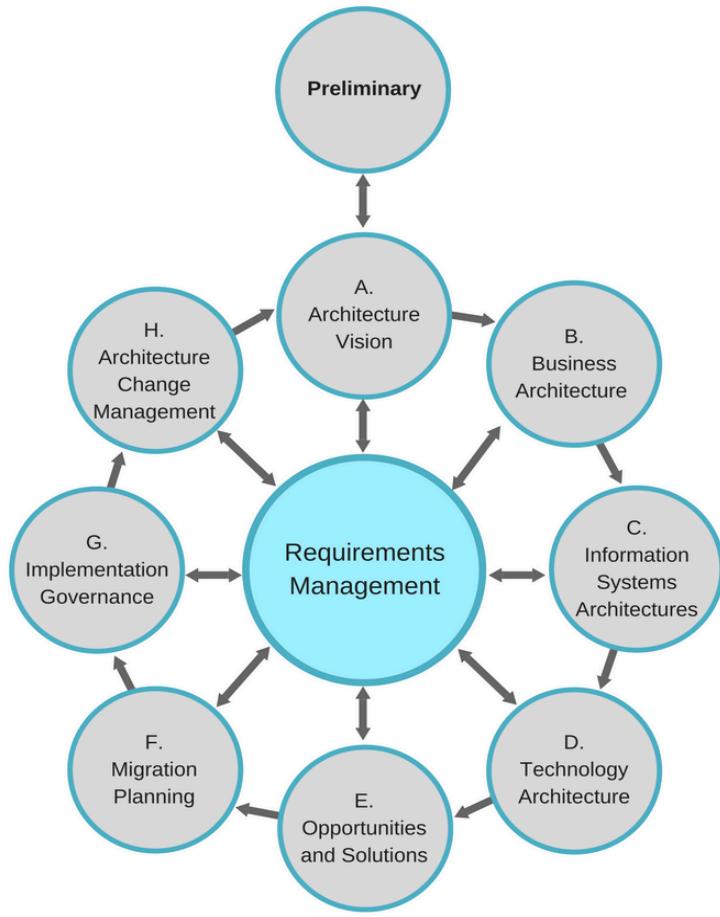


Figure 2 shows the TOGAF diagram that depicts the Architecture Development Method as a cyclical process, with each phase reflecting how MapAMS was conceptualized and developed. The Preliminary and Architecture Vision phases outline the project scope, and MapAMS was chosen as a solution to update asset tracking, increase audit accuracy, and enable forecasting for MAP Active Philippines. The Business Architecture section of the diagram represents the mapping of real-world business operations into structured workflows within the system. Moving on to the Data and Application Architecture sections of the model, the diagram depicts how information and applications are structured. For MapAms, this corresponds to designing microservices and the database-per-service pattern to ensure modularity and scalability. Finally, the Technology Architecture section focuses on the deployment and

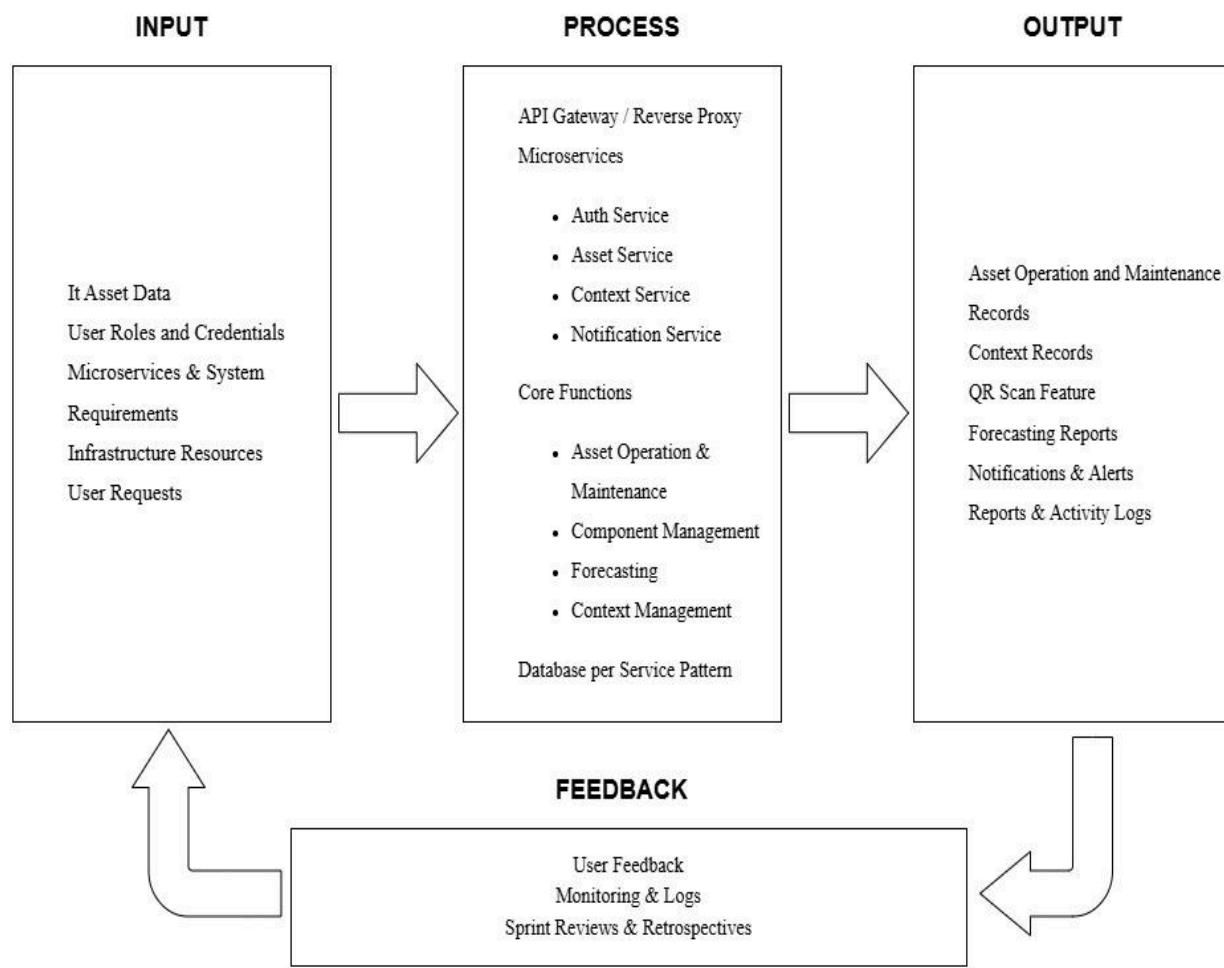
infrastructure components that MapAms provides through the use of Docker containers, Railway hosting, GitHub Actions CI/CD, and an API gateway for interservice communication. Overall, the TOGAF diagram depicts MapAms's structured architectural lifecycle, ensuring that each layer of the system is aligned with business objectives and technological best practices.

Integration patterns allow services to communicate in a variety of ways, from tight to loosely linked models. API-based integration is extensively utilized, allowing services to expose RESTful or other communication interfaces that promote modularity and scalability (Mylsamy & Shrivastav, 2022). Meanwhile, more complicated integration models, such as an Enterprise Service Bus, can be used to organize routing, transformation, and mediation in bigger systems (Kienle & Müller, 2023). API-based integration complements MapAms' microservices design, but an ESB-like method could be considered in future expansions or for interfacing with external corporate systems.

Conceptual Framework of MapAms

The conceptual framework of Map Active Philippines Asset Management System (MapAms) below illustrates how the system transforms organizational inputs into actionable outputs while maintaining a continuous feedback loop for improvement. The Input includes IT asset data, user roles and credentials (admins and operators), microservices and system requirements, infrastructure resources (Railway hosting, Docker containers, PostgreSQL databases), and user requests through the web interface or QR code scans. These inputs form the foundation for the system's core processes.

Figure 3. MapAms Conceptual Framework

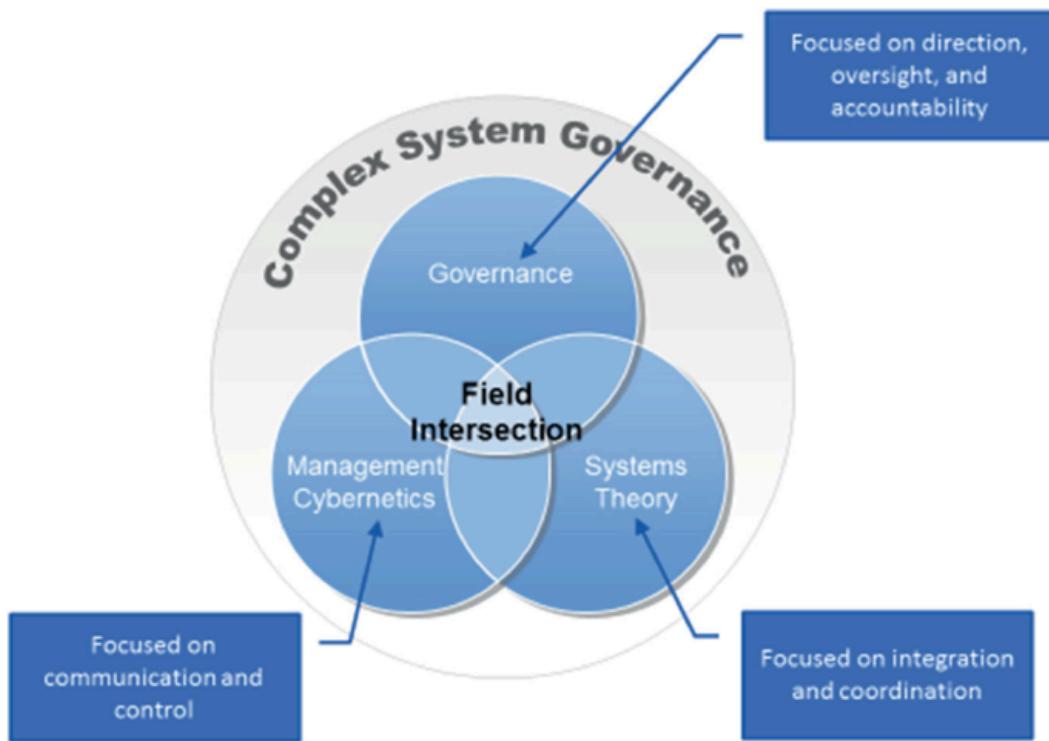


The Process step describes how MapAms processes and manages these inputs. Incoming requests are routed through the API Gateway, which sends them to the appropriate microservices. Each microservice serves a specific purpose: (1) Asset Service handles registration, auditing, and maintenance (2) Context Service enriches asset data with metadata such as category or supplier (3) Auth Service manages secure access and the (4) Notification Service sends out alerts for maintenance or updates. Functionalities such as component administration, reporting, and forecasting use a database-per-service design to ensure modularity, scalability, and independent deployment. These processes produce updated asset records, forecasting reports, notifications, and activity logs, which assist the business retain accurate and actionable data. Finally, the Feedback process, which includes user ideas,

monitoring logs, and sprint evaluations, guides ongoing improvement, ensuring that future MapAms iterations are consistent with operational demands, best practices, and ISO-aligned IT asset management concepts.

Theoretical Paradigm

Figure 4. Complex System Governance



MapAms theoretical paradigm is Complex System Governance (CSG), which states that big systems with numerous interacting subsystems require a metasystem layer to provide control, communication, coordination, and integration, providing both autonomy and stability. In the case of an asset management system like MapAms, CSG is extremely important since each microservice functions as a semi-independent subsystem, but the metasystem via components such as the API gateway, CI/CD pipeline, and monitoring governs them to ensure consistent performance and overall system viability. According to CSG principles, the metasystem imposes

relatively minimum constraints, allowing each microservice to expand independently while maintaining overall system alignment and resilience (Keating et al., 2022). This mix of autonomy and governance enables MapAMS to grow, adapt, and respond to environmental and operational changes without devolving into breakdowns.

Chapter 3

METHODOLOGY AND PROJECT MANAGEMENT

Agile Scrum Methodology in Project

The Agile Scrum framework was chosen by the team as the main approach for managing the Asset Management System's development. Scrum was chosen because it offers an adaptable and iterative methodology that enables the team to consistently produce functional software while successfully adapting to needs changes. In contrast to conventional linear approaches, Scrum places a strong emphasis on cooperation, feedback, and incremental improvement all of which are crucial for creating a system that is customized to the client's requirements.

The team makes sure that development is consistent and quantifiable by breaking up the work into sprints. Every sprint results in usable system increments that can be examined, tested, and verified against the client's requirements. This strategy lowers risks, encourages responsibility, and guarantees that the project develops in accordance with the specifications and workflows given by the MAP Active Philippines Retail Company's IT Department.

Adapting Scrum to the academic capstone setting also allows us to define clear responsibilities, maintain open communication, and practice industry-standard project management techniques. Through regular Scrum ceremonies and structured roles, the team is able to balance academic deliverables with practical software development practices, ensuring both process discipline and innovation throughout the project.

Team Roles and Responsibilities

Table No. Team Members

TEAM	ROLES
Ma. Elisa Johanna T. Garrote	Project Manager / Business Analyst / Documentation Analyst
Edzra C. Macas	Backend Developer / Frontend Developer / Quality Assurance
Johanna Mae L. Sillano	Frontend Developer / Quality Assurance / Backend Developer
Bengie B. Villesco	Quality Assurance / Backend Developer / Frontend Developer

The clarity of roles and responsibilities defined inside the Scrum framework is crucial to the project's success. Each team member has a unique role to play, yet they are also adaptable enough to help out when needed. Through outlining these responsibilities precisely, the team guarantee accountability while fostering cooperation across the various phases of development.

Elisa serves as the Scrum Master, Project Manager, Business Analyst, and Documentation Lead. She is responsible for guiding the team in applying Scrum practices, facilitating sprint planning and stand-up meetings, and ensuring that development activities align with the project's objectives. Elisa also oversees documentation, manages timelines, and communicates directly with the client to clarify requirements and validate progress. Her leadership ensures that the project maintains focus and direction while remaining adaptive to changes.

Edzra is the Lead Backend Developer and also contributes to frontend development when needed. She manages the backend architecture, implements the business logic, and sets up the database and integrations that form the system's foundation. Her work ensures that the backend remains reliable, scalable, and efficient, serving as the backbone of the application.

She plays a vital role in translating complex requirements into technical solutions that drive the system's functionality.

Johanna takes the lead as the Frontend Developer while also supporting backend tasks and documentation. She is responsible for designing and implementing the user interface, focusing on usability, responsiveness, and visual appeal. By ensuring smooth communication between the frontend and backend, Johanna bridges the technical and design aspects of the project. Her contribution ensures that the system is both functional and user-friendly, delivering a seamless experience for end users.

Bengie serves as Quality Assurance and Full-Stack Developer. He validates the system through testing, ensuring that features meet acceptance criteria and perform reliably. In addition to his QA role, he assists in both backend and frontend development, contributing to the system's robustness and stability. His detail-oriented approach ensures that errors are identified early, minimizing risks during deployment and maintaining overall product quality.

The Product Owner is the MAP Active Philippines Retail Company, specifically its IT Department, which serves as our client. They provide the workflows, system requirements, and expectations that guide development. Their role is to ensure that the features built by the team align with their operational needs and that the system delivers value to their department. They are also responsible for validating outputs and confirming that each sprint increment meets their standards.

Collaboration and accountability are achieved through the combination of defined lead roles and open team communication. While each member is accountable for their area of expertise, everyone contributes beyond their assigned role when needed. Regular Scrum ceremonies, progress monitoring, and continuous dialogue with the Product Owner ensure transparency, shared responsibility, and steady progress toward the project's goals.

Scrum Events

The Scrum Events provide structure to the team's workflow, ensuring that everyone remains aligned, adapts to challenges as they arise, and consistently delivers progress. These events guide the rhythm of the project, allowing the team to plan, monitor, evaluate, and improve performance throughout the duration of the capstone. Each ceremony is customized to fit the project's academic timeline and objectives, supporting collaboration among members while maintaining communication with the client.

Sprint Planning is conducted every two weeks at the beginning of each sprint. During this session, the team reviews the product backlog and identifies the specific tasks that can realistically be completed within the sprint timeline. These tasks are documented in the sprint backlog, which is structured with detailed columns such as User ID, Module, Sub-Module, Task, User Story, Acceptance Criteria, Priority, Assignee, Story Points, Status, Dates, Dependencies, and Remarks. This ensures that each sprint begins with clear objectives, responsibilities, and measurable outcomes, keeping progress consistent until the end of Capstone 2.

In Daily Stand-ups, these are conducted in short sessions, typically once a week or midweek, to monitor progress and identify obstacles. In these meetings, each member provides updates on their completed work, current assignments, and potential blockers. The information helps determine whether adjustments are needed, such as extending timelines, reallocating tasks, or providing additional support, ensuring that the sprint stays on track.

For Sprint Reviews, these are carried out both through direct meetings and by monitoring the sprint backlog maintained in Excel. Completed tasks are presented, validated, and compared against acceptance criteria to confirm whether they meet the requirements set by the Product Owner. This process not only ensures transparency in what has been delivered but

also allows the client to provide timely feedback for alignment before proceeding to the next sprint.

Lastly, Sprint Retrospectives take place at the end of each sprint to reflect on the team's overall performance. The discussion begins with identifying what challenges or issues were encountered, followed by highlighting the practices that went well. From there, the team develops solutions for the problems faced and agrees on adjustments to improve efficiency and collaboration. Through analyzing task durations, obstacles, and outcomes, the retrospective enables continuous learning and process improvement, strengthening the team's workflow for subsequent sprints.

Scrum Artifacts

In the Scrum framework, several key artifacts play a vital role in ensuring transparency, enabling efficient tracking of progress, and fostering effective communication throughout the project. One of the most important artifacts is the Product Backlog, which serves as a comprehensive and prioritized list of all modules, features, enhancements, and other tasks to be completed. In our case, the Product Backlog includes core modules such as Authentication, Asset Management, Asset Model, Asset Maintenance, Asset Audit, Ticket Integration, Dashboard with predictive analytics, Components, Accessories, Consumables, and Context such as supplier, manufacturer, etc.

It also covers specific features like QR scanning for asset information, forecasting, ticket tracking integration, and budget management. While the backlog is refined with inputs from the company as the client, its maintenance and prioritization largely fall under the responsibility of the Project Manager, ensuring alignment with both academic requirements and business needs. This dynamic backlog evolves throughout the project to reflect the most critical and valuable

work, supporting the team in delivering consistent progress and preparing for both client and academic defenses.

Another essential artifact is the Sprint Backlog, which represents the selected subset of tasks from the Product Backlog that the team commits to deliver within a sprint. In our project, the Sprint Backlog is documented in detail, structured with columns such as User ID, Module, Sub-Module, Features, Functions, Test Cases, and corresponding Status. This structure provides clarity on task ownership, scope, and progress while allowing for monitoring of items that are completed, blocked, or encountering issues. The Sprint Backlog is updated continuously by the team during sprint execution to reflect real-time progress, ensuring that both the company and academic panelists can clearly track development. This artifact not only guides the team's daily activities but also ensures that priorities remain aligned with the sprint's objectives and the overall project goals.

The third key artifact is the Increment, which refers to the cumulative result of completed work at the end of each sprint. For our team, a "Done Increment" is defined as a potentially shippable product that has passed functional testing, fulfilled requirements in the Requirements Traceability Matrix (RTM), and successfully undergone User Acceptance Testing (UAT) as well as review from academic panelists. Each increment demonstrates tangible progress, incorporating all tested and validated features that are ready for use or further demonstration. This artifact is crucial not only for ensuring system readiness but also for providing stakeholders and panelists with clear evidence of the team's deliverables and value creation after every sprint cycle.

The Product Backlog, Sprint Backlog, and Increment are Scrum artefacts that work together to create an open and organised framework that facilitates efficient communication, directs decision-making, and guarantees alignment between the development team, the client

business, and academic needs. The project team can clearly monitor progress, adjust to adjustments, and produce a system that satisfies academic criteria and real-world business needs by skilfully utilising these artefacts.

Sprint Cycle

In the Scrum framework, the sprint cycle defines the iterative rhythm of development, allowing the team to consistently deliver increments of value while maintaining transparency and alignment with project goals. Our project is divided into seven sprints, each lasting two weeks, with a focused objective that contributes to the overall development of the Asset Management System.

Sprint 1 focused on planning, ideation, and design. The team created the initial wireframes, developed prototypes, and presented the system concept to the company for alignment. This sprint established the foundation for future development and ensured that stakeholder expectations were properly addressed before coding began. Sprint 2 concentrated on the development of the frontend interface. The team translated wireframes into functional UI components, implementing core layouts and user navigation flows. This sprint established the system's visual identity and prepared the frontend for integration with backend services in future sprints.

Sprint 3 marked the beginning of backend development and integration. Core APIs and services were built to support asset registration, supplier management, and related modules. The sprint emphasized establishing a stable backend foundation to connect with the frontend developed in Sprint 2. Sprint 4 continued backend development with a stronger focus on integration and data flow between modules. At this stage, asset-related modules, ticket tracking integration, and context registration were refined and connected, ensuring a cohesive system architecture.

Sprint 5 shifted attention back to the frontend for refinement and revisions. Based on feedback from company stakeholders and academic panelists, the team applied improvements to UI/UX design, adjusted navigation elements, and polished frontend functionality to ensure alignment with user expectations. Sprint 6 will focus on backend refinement and cleaning of integrations. The team resolved technical issues, improved performance, and ensured seamless communication across system modules. This sprint served as a stabilization phase, preparing the system for its final stages of development. Sprint 7 will emphasize the integration of emerging technologies within the system, particularly the forecasting module under predictive analytics. In addition, the team finalized refinements to both frontend and backend, preparing the system for User Acceptance Testing (UAT) and defense presentation.

After combined, these sprints offer an organised and step-by-step method for developing systems. Every sprint produced measurable results that added to the overall system, guaranteeing consistent, trackable development that could be adjusted in response to input from academic and business assessors. The team was able to successfully deploy the MAP-AMS - Asset Management System by striking a balance between innovation, functionality, and quality thanks to this sprint cycle.

Tool Stack

The success of the project depends not only on the development process but also on the effective use of tools that support collaboration, documentation, version control, and communication. The team has carefully selected tools that balance practicality and accessibility, ensuring both the project's technical needs and academic requirements are met.

One of the primary tools used is Microsoft Excel, which serves as the team's main project management tracker. Excel is utilized for sprint planning, project timelines, the Requirements Traceability Matrix (RTM), and test cases. Its flexibility allows the Project

Manager to monitor progress, track dependencies, and update statuses in a format that is both transparent and easy to align with academic documentation requirements. Additionally, Microsoft Word and Google Docs are used for creating and maintaining documentation, ensuring that project artifacts remain organized and readily available.

For version control, the team uses GitHub, following a structured branching strategy to maintain code quality and minimize conflicts. The Develop Branch acts as the integration point for completed features before preparing a release. The Release Branch is used for final testing, bug fixing, and versioning before merging into the production-ready Main Branch. Experimental work is conducted in the Test Branch, while specific issues are handled in Issue Branches using the format issue/<ticket-id>-<short-description>. Urgent fixes are managed through Hotfix Branches, ensuring critical bugs in production can be resolved quickly without disrupting ongoing development. This branching model allows the team to separate stable, production-ready code from ongoing development and testing activities.

For team communication, the group uses Microsoft Teams, which provides an accessible platform for discussions, file sharing, and quick updates. Teams serve as the primary channel for coordinating schedules, discussing blockers, and conducting remote meetings. Its integration with other Microsoft tools also supports seamless collaboration with documentation and project files. Overall, the team has created a useful and effective toolstack that supports project transparency, collaboration, and version integrity throughout the whole development cycle by integrating Teams for communication, GitHub for version control, and Excel for project management.

Architectural Design Decisions

The architecture of the Asset Management System plays a central role in defining how the different modules function, interact, and adapt to both current and future requirements. The

team recognized early in the project that the system would not be limited to a single set of features but would grow in scope as new demands emerged, such as asset forecasting, asset management and multiple system integration. With these considerations, the architecture needed to be secure, modular, and capable of handling large-scale operations without compromising speed or reliability.

To meet these needs, the project adopts a Microservices Architecture rather than a Monolithic structure. This decision is not only influenced by technical advantages but also by the practical requirements of the client, MAP Active Philippines IT Department. Since the system is intended to streamline the IT team's management of assets, maintenance, and auditing processes, the architecture must allow flexibility for future expansion and integration with other company systems. Through decoupling the system into multiple services, the architecture ensures that changes or issues in one area will not disrupt the entire system, an important factor for IT operations that must remain stable and secure.

Microservices

The Asset Management System was designed using Microservices Architecture because it offers significant advantages in terms of performance, security, and reliability compared to a traditional Monolithic approach. In a monolithic system, all modules share the same codebase and database, making the application easier to develop initially but harder to maintain and scale over time. A failure in one part of the system could bring the entire application down, and updates often require redeploying the whole system. This approach does not align with the complexity and critical nature of an enterprise-level asset management platform.

Additionally, Microservices separate the system into smaller, separate services, each of which is in charge of particular modules such Dashboard Analytics, Asset Models, Asset

Registration, Maintenance, Audit, and Context Management. The design is more robust against system failures and security breaches because each service can operate on its own process, many of which use distinct databases. Operations can proceed uninterrupted even if one service experiences an error because the others continue to function. Because of this separation, even when a certain module is being fixed or upgraded, the IT department may still rely on the system.

On the side of security, it is further strengthened by distributing data across multiple services. In a monolithic system, a single database breach could expose all information. With microservices, even if one database is compromised, the rest of the system remains secure. This is critical for asset management, where sensitive company data such as supplier contracts, depreciation records, and audit results must be safeguarded. Also, performance is another important factor. Microservices allow each service to run independently, avoiding the overhead of running the entire application for every request. For example, scanning a QR code to retrieve asset information should not require the system to interact with unrelated modules like forecasting or ticket integration. This independence results in faster response times and smoother user experience, especially as the system grows in scale.

The expected benefits of microservices extend beyond immediate functionality. The architecture supports scalability, allowing individual services to be scaled up or optimized based on demand. For example, forecasting analytics may require more resources compared to supplier registration, so the system can allocate resources accordingly. Fault isolation ensures that issues remain contained, improving reliability, while maintainability is enhanced because updates and improvements can be applied to one module without affecting the rest of the system. These qualities make the system adaptable to future expansions and easier for the IT department to manage in the long term.

Despite these advantages, microservices also introduce challenges. Deployment is more complex compared to a monolithic setup, as each service must be managed separately. The system requires additional effort to ensure consistency across environments, and coordination is needed when services interact. To address this, the team employs containerization, with each service packaged into its own Docker container. This ensures that services run consistently in both local development and production environments, simplifying deployment and reducing compatibility issues.

Overall, monolithic architecture would not have supported the system's long-term vision, even if it would have been easier to construct, especially given the capstone project's short timeline. The team's decision to use microservices results in a more contemporary, safe, and expandable solution that demonstrates both academic rigour and practical relevance. This architecture offers the MAP Active Philippines IT Department a system that is not only operational right now but also flexible enough to accommodate future requirements, integrations, and developments in technology.

Core Architectural Patterns

No Provided for Now

Development, Operations, and QA Methodology

The development and deployment of the Asset Management System are supported by a structured DevOps toolchain designed to ensure consistency, maintainability, and transparency across environments. The selected tools balance academic requirements and industry standards, allowing the team to manage code effectively, containerize services, and deploy the system in a reliable manner.

DevOps Toolchain

Version Control is managed through GitHub, where the team follows a structured branching model. The Develop Branch acts as the main integration branch for completed work, while the Release Branch is used for final testing, bug fixing, and versioning before merging into the production-ready Main Branch. Additional branches support experimentation and issue resolution: the Test Branch for experimental changes, Issue Branches for bug fixes and improvements, and Hotfix Branches for urgent production fixes. This branching guide ensures clarity in workflow, minimizes conflicts, and enables traceability across all development activities.

Containerization is achieved through Docker, with each backend microservice (Authentication, Assets, Contexts) and the frontend application running inside its own container. Dockerfiles define how each service is built and configured, while docker-compose.dev.yml and docker-compose.yml orchestrate multi-service environments for development and production respectively. This setup provides consistency across local and production environments, simplifies service management, and isolates dependencies to reduce compatibility issues. The React frontend is containerized using a multi-stage Dockerfile that builds optimized static files and serves them through Nginx, while Django backend services run with Gunicorn in production mode.

Automation in the pipeline is partially implemented. While GitHub Actions can be used for automated builds and validation, the team currently relies primarily on manual validation with Docker builds and documented test cases. Issues identified during testing are tracked in GitHub Issues and resolved in dedicated feature or hotfix branches before reintegration. Full CI/CD automation has not yet been implemented but is considered for future iterations.

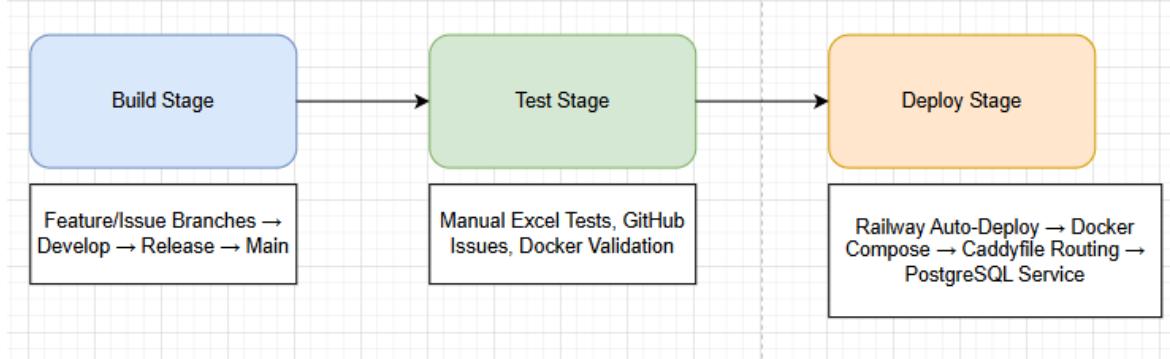
Deployment is handled through Railway, which monitors the Main Branch and triggers deployments upon new commits. Each service is built and run based on its Dockerfile, while docker-compose.yml orchestrates multi-service deployments. The Caddyfile acts as a reverse proxy to route requests between the frontend and backend services. In production, PostgreSQL is managed as an external service in Railway, ensuring persistent data storage. Environment variables are maintained through .env.dev and .env.prod files, with production values securely stored in Railway's environment settings.

The project's workflow is dependable and well-organised according to its DevOps toolchain. Docker ensures consistency across environments, Railway simplifies deployment with little manual involvement, and GitHub maintains branching discipline and version control. The foundation allows for the future adoption of GitHub Actions or other automation tools as the project develops, even though automated CI/CD integration is not yet fully implemented.

CI/CD Pipeline Design

The Continuous Integration and Continuous Deployment (CI/CD) pipeline was designed to establish a standardized process for managing development, testing, and deployment of the Asset Management System. Structuring the workflow into defined stages, the pipeline ensures that changes are validated before release, deployments remain consistent across environments, and the production environment remains stable and reliable.

Figure 1. CI/CD Diagram



The pipeline consists of three primary stages: Build, Test, and Deploy. In the Build stage, development begins in feature or issue-specific branches that follow strict naming conventions. Completed work is merged into the develop branch for integration, and pre-production validation takes place in release branches before being merged into the main branch. Each service, including the React frontend and Django-based backend microservices, is built through its own Dockerfile. Multi-service orchestration is handled through docker-compose.yml, guaranteeing consistency across development and production environments.

The Test stage validates that services are functioning as expected. While automated testing via GitHub Actions is not fully implemented, the team performs thorough manual testing based on Excel-documented test cases. These cover both functional and non-functional requirements, including CRUD workflows, authentication, and usability checks. Defects identified are logged into GitHub Issues and resolved through issue or hotfix branches before being reintegrated into the main workflow. Docker builds also serve as validation checkpoints to ensure containers compile and run correctly before progressing further.

The Deploy stage begins once the release branch is validated and merged into the main branch. Railway continuously monitors the main branch and automatically initiates deployment whenever new commits are pushed. During deployment, Docker images are built for each

service, migrations and static file collections are executed, and services are launched with environment variables securely managed through Railway. The Caddyfile serves as a reverse proxy, directing incoming requests appropriately between the frontend and backend services, while PostgreSQL is managed as an external Railway service to ensure persistent storage.

Failures within the pipeline are handled systematically. Build failures prevent merging, requiring developers to review logs and correct issues. Testing failures are documented, tracked in GitHub, and resolved through dedicated branches. Deployment errors in Railway are addressed by reviewing logs and rolling back to the last stable tagged version in GitHub. For critical issues in production, urgent fixes are developed in hotfix branches, validated, and deployed immediately to restore service stability. Overall, the team is able to maintain development discipline, guarantee release transparency, and produce a dependable, scalable system that satisfies customer and academic standards thanks to this organised CI/CD pipeline.

Testing Strategy

No Provided for Now

Innovation Framework

No Provided for Now

Chapter 4

System Design, Development, and Implementation

Requirements Engineering

Stakeholder Analysis

Stakeholder analysis serves as a critical component in the development of the MAP Active IT Asset Management System with Forecasting (MapAms), facilitating the systematic identification of individuals and groups with either direct or indirect interests in the project. This analytical process seeks to delineate stakeholder roles, responsibilities, degrees of influence, and their relational dynamics within the project framework. Through this mapping, the analysis supports the alignment of the system's design and implementation with both technical requirements and academic standards, while also addressing the operational needs of the client. Stakeholders are generally classified into two main categories, with internal stakeholders referring to the project development team, and external stakeholders including the client organization and academic advisers.

Table _

Stakeholder Analysis

Stakeholder	Role	Type	Interest in System	Level of Influence	Responsibility
Mr. Elvin Punzalan	IT Department Head, MAP Active Philippines	External	High	High	Serves as the Product Owner representing the client organization. Provides workflow details, identifies process inefficiencies,

					and defines the required functionalities of the Asset Management System. Validates deliverables and ensures that the system aligns with the IT department's operational needs.
Mrs. Rosicar Escobar	Capstone Adviser	External (Academic)	Medium	High	Provides academic guidance by setting deliverables, defining timelines, and assessing sprint progress. Reviews documentation and ensures that outputs comply with institutional standards and capstone requirements.
Ms. Keziah Monzon	Capstone Adviser	External (Academic)	Medium	High	Collaborates with the project team during consultations and documentation reviews. Offers insights on technical modeling, report writing, and presentation structure to ensure academic quality.
Mr. Estong Odpaga	System Consultant Adviser	External (Technical)	Medium	Medium	Provides technical consultation and validation of development decisions. Advises the team on best practices in software design, system architecture, and implementation strategies.
Ms. Ma. Elisa Johanna Garrote	Project Manager / Scrum Master / Business Analyst	Internal	High	High	Leads project planning and coordination. Oversees sprint schedules, assigns tasks, and ensures communication between the client, advisers, and development team. Manages progress

					tracking and sprint retrospectives.
Ms. Edzra Macas	Lead Backend and Frontend Developer	Internal	High	High	Handles backend system architecture and supports frontend integration. Ensures proper implementation of business logic, database integrity, and API performance. Provides technical recommendations during sprint reviews.
Ms. Johanna Mae Sillano	Lead Frontend and Backend Developer	Internal	High	Medium	Focuses on frontend user interface and user experience. Ensures design consistency and usability, while also contributing to backend development and API integration.
Mr. Bengie Villesco	Quality Assurance / Full Stack Developer	Internal	High	Medium	Conducts quality assurance testing based on established acceptance criteria. Identifies bugs and validates that each feature performs according to user stories and requirements.
End Users (IT Staff)	System Users	External	High	Medium	Represent the operational users of the system. Responsible for utilizing the AMS in asset registration, tracking, and auditing. Provide feedback during system demonstrations and user acceptance testing.

The table shows the classification and involvement of both internal and external stakeholders who play vital roles in the development of the MAP Active IT Asset Management System with Forecasting (MapAms). As presented, the external stakeholders include

representatives from the client organization, academic advisers, and technical consultants, while the internal stakeholders comprise the project development team. This distribution ensures that both the academic and industrial aspects of the system are addressed throughout the project lifecycle. The inclusion of the client organization, MAP Active Philippines, represented by Mr. Elvin Punzalan, reflects the project's real-world application and operational purpose. His active participation as the Product Owner provides the development team with essential insights regarding workflow inefficiencies, required functionalities, and feature prioritization. Through continuous validation and feedback, he influences the project's direction to ensure that the resulting system aligns with the IT Department's operational needs and long-term objectives.

The table also outlines the contributions of academic advisers who serve as guiding authorities throughout the project's execution. Both Ms. Rosicar Escobar and Ms. Keziah Monzon act as Capstone Advisers responsible for monitoring deliverables, reviewing documentation, and ensuring adherence to institutional guidelines. Their influence extends beyond administrative oversight, as they also assist the team in refining technical documentation, system models, and sprint outputs. Likewise, the System Consultant Adviser, Mr. Estong Odpaga, provides technical supervision to maintain the system's alignment with current industry standards and software engineering practices. His role supports the team in addressing architectural and implementation challenges, bridging theoretical frameworks with practical development strategies. The inclusion of these advisers strengthens the project's academic foundation and ensures that all technical and methodological standards are achieved.

Lastly, the table highlights the participation of the internal stakeholders, composed of the project's development team and the end users. The internal stakeholders; namely Ms. Ma. Elisa Johanna Garrote, Ms. Edzra Macas, Ms. Johanna Mae Sillano, and Mr. Bengie Villesco—play distinct but interconnected roles throughout the development cycle. Elisa functions as the

Project Manager and Scrum Master, overseeing sprint planning, communication, and task coordination. Edzra and Johanna serve as the lead developers for the backend and frontend, respectively, ensuring the technical accuracy and efficiency of each module. Bengie, on the other hand, focuses on quality assurance and validation of features according to the acceptance criteria and also contributed a huge part in development as a full stack developer. In addition, the MAP Active IT staff, as the intended end users, are identified as key participants during the User Acceptance Testing (UAT) phase, where they will provide practical feedback on the system's usability and performance. Their input will be instrumental in refining the final output, ensuring that the AMS effectively supports asset registration, maintenance, and auditing functions. Overall, the table demonstrates how each stakeholder contributes to the project's collaborative and iterative development approach, ensuring balance between academic rigor and real-world applicability.

Elicitation Techniques

The requirement-gathering process for the Asset Management System (AMS) involved the application of both survey and interview methods to ensure a comprehensive understanding of the client's needs. The survey served as the preliminary phase of data collection and was administered to the IT Department Head of MAP Active Philippines during the early stage of development in March. It consisted of a structured set of questions designed to identify existing challenges in asset and inventory management, inefficiencies in workflow, and expectations for system functionality. This technique enabled the team to collect organized and objective responses, which provided a foundational perspective on the operational environment of the IT department.

Following the survey, an in-depth interview was conducted with the IT Department Head to obtain more detailed insights and clarify the responses provided in the initial survey. All four

members of the project team participated in this session, which was carried out face-to-face at the client's premises. The interview focused on exploring pain points, validation of workflow processes, and identification of features that could enhance efficiency in asset monitoring. Through open-ended questions and discussion, the team was able to understand the daily challenges experienced by IT personnel in managing asset information, maintenance schedules, and accountability procedures. This approach allowed the team to gather qualitative data that complemented the structured information from the survey, resulting in a more holistic view of the client's operational needs.

Furthermore, the combination of survey and interview techniques proved effective in establishing an accurate representation of the client's requirements. The survey provided an initial framework that allowed the stakeholder to reflect on the organization's processes, while the interview encouraged interactive discussion and deeper exploration of issues. This sequence of methods was particularly appropriate for the stakeholder, as it facilitated both reflection and collaboration before technical development began. The general insight derived from these elicitation activities revealed that the organization required a comprehensive system to manage its assets and inventory more efficiently, promoting proper monitoring, accountability, and transparency. The information obtained from these techniques served as the foundation for identifying the system's functional and non-functional requirements, which guided the design and development of the Asset Management System.

Functional Requirements

The functional requirements of the MAP Active IT Asset Management System with Forecasting (MapAms) were derived from the results of the requirements elicitation activities, specifically the survey and interview conducted with the IT Department of MAP Active Philippines. These requirements represent the key functionalities necessary to address the

identified workflow pain points and operational inefficiencies. To ensure systematic prioritization, the team utilized the MoSCoW method, which categorizes each function into four levels: Must Have, Should Have, Could Have (Nice to Have), and Won't Have. This prioritization strategy allowed the project team to manage development resources efficiently and ensure that the most essential features were implemented first.

The prioritization process was guided by the feedback of the IT Department Head, technical feasibility, and the project's timeline. The Must Have requirements were considered critical to the operation of the system and were therefore prioritized during the early development sprints. These include modules such as authentication, context management, asset registration, repair, and audit core functionalities that form the backbone of the MAP-AMS. The Should Have requirements were identified as secondary but valuable enhancements that improve usability and system performance, such as dashboard and reporting capabilities. Meanwhile, the Nice to Have requirements, including the forecasting feature and QR code scanning for asset information viewing, were recognized as innovative additions that could be developed if time and resources permit. The Won't Have requirements, such as accessories and consumables management, were excluded from the current scope due to time and resource limitations but may be considered for future iterations.

Table _

MoSCow Prioritization of Functional Requirements

ID	Priority	Module/Feature	Description
FR-01	Must Have	Authentication (Integrated)	Enables secure user login and role-based access control.
FR-02	Must Have	Profile Management	Allows users to manage personal and account

			information.
FR-03	Must Have	Assets Module (Includes Asset Model/Product Registration)	Handles asset registration, tracking, and detailed product data management.
FR-04	Must Have	Asset Repair (Maintenance)	Records and manages repair or maintenance activities for assets.
FR-05	Must Have	Asset Audit	Supports auditing processes and asset verification.
FR-06	Must Have	Context (Category, Supplier, Manufacturer, Depreciation, Status, Recycle Bin)	Manages contextual data required across modules for efficient asset tracking.
FR-07	Should Have	Dashboard	Displays summarized system analytics and key performance indicators.
FR-08	Should Have	Reports	Generates detailed reports such as asset depreciation, end-of-life, warranty, and audit results.
FR-09	Should Have	Components	Manages the relationship of asset parts and their configurations.
FR-10	Nice to Have	Notification	Sends system alerts or reminders regarding asset activities or maintenance schedules.
FR-11	Nice to Have	Forecasting	Provides predictive analytics for asset utilization and replacement cycles.
FR-12	Nice to Have	QR Scanned (for asset info viewing only)	Allows quick retrieval of asset information through QR code scanning.
FR-13	Won't Have	Accessories and Consumables	Excluded from current scope but may be considered for

		future development.
--	--	---------------------

The functional requirements presented in this section are aligned with both the Product Backlog and the Requirements Traceability Matrix (RTM). In the Product Backlog, the development process follows the MoSCoW prioritization approach, where must-have functionalities are implemented in the early and refinement sprints to establish the system's core operations. Meanwhile, should-have and nice-to-have features are integrated in later stages to enhance usability and innovation. In the RTM, this alignment is reflected by documenting the version and completion status of each module. The initial iterations of the RTM primarily included the must-have and should-have requirements, which correspond to completed and tested modules. Subsequent refinements introduced minor features and enhancements. This structured approach ensures that every functional requirement identified during the elicitation process is traceable, implemented, and validated throughout the system's development lifecycle.

Non-Functional Requirements

Table _

Non-Functional Requirements

ID	Category	Requirement Description	MoSCoW Priority
NFR-1	Maintainability	Supplier updates must preserve historical data through versioning or logs, ensuring traceability and compliance with data retention policies.	Must
NFR-2	Performance	Category list pages must load within 2 seconds for up to 1000 records, utilizing pagination with responsive list numbers and scrollable pages to handle large datasets efficiently.	Must

NFR-3		Manufacturer updates must trigger real-time notifications to linked assets if changes affect depreciation or status, maintaining system responsiveness under 1 second.	Must
NFR-4		Status pages must load with filters applied by default (e.g., alphabetical), supporting up to 5000 records with efficient querying for performance.	Must
NFR-5		Depreciation calculations must be accurate to two decimal places for currency fields, with formulas executed in under 100ms per record.	Must
NFR-6		Asset Model view pages must load details and tabs (Details/About, Assets) in under 1 second, with responsive design for mobile views.	Must
NFR-7		Advanced search for Asset Models must handle multiple filters (e.g., category, manufacturer) efficiently, returning results in under 500ms.	Should
NFR-8		Asset list views must support pagination and sorting, handling up to 5000 assets with load times under 3 seconds.	Must
NFR-9		Advanced asset searches must support multi-field filtering, optimized for quick responses even with large datasets.	Should
NFR-10		Asset check-out processes must integrate with tickets securely, completing in under 2 seconds with real-time status updates.	Must
NFR-11		Component views and operations must be performant, with tables supporting filtering and pagination for scalability.	Should
NFR-12		Status operations must be efficient, with updates propagating to linked entities in real-time.	Must
NFR-13		Depreciation operations must maintain numerical precision and support batch processing for performance.	Must
NFR-14	Reliability	Updates to categories must be processed atomically to maintain data consistency, with the system handling concurrent edits	Must

		gracefully and logging changes for audit purposes.	
NFR-15		Deletion of categories must check for dependencies (e.g., linked assets or components) and prevent removal if in use, returning error messages within 1 second to ensure system reliability.	Must
NFR-16		Deletion of suppliers must be reversible via recycle bin within 30 days, maintaining data availability and recoverability.	Should
NFR-17		Status updates must be backward-compatible, ensuring no disruption to existing asset linkages.	Should
NFR-18		Deletion of statuses must include dependency checks and soft-delete mechanisms for recoverability.	Should
NFR-19		Deletion of depreciations must prevent removal if linked to assets, with error responses in JSON format for API integrations.	Should
NFR-20		Asset Model updates must maintain referential integrity with linked assets, logging changes for audit trails.	Must
NFR-21		Deletion of Asset Models must be restricted if in use, with soft-delete and recycle bin integration for data recovery.	Should
NFR-22		Asset updates must be atomic and trigger recalculations for depreciation or EoL, ensuring data consistency.	Must
NFR-23		Asset check-in must validate against existing requests, ensuring reliability and preventing unauthorized actions.	Must
NFR-24		Recycle bin recovery must be reliable, with operations logged and limited to admins for security.	Should
NFR-25		The system must handle errors gracefully, displaying user-friendly messages and preventing exposure of sensitive data.	Must
NFR-26	Scalability	Manufacturer registration must handle high	Should

		concurrency (up to 10 simultaneous users) without data loss, ensuring scalability.	
NFR-27		Depreciation updates must recalculate affected asset values automatically, ensuring computational efficiency for batches up to 1000 assets.	Should
NFR-28		Recycle bin must retain deleted items for at least 90 days, with storage optimized to handle up to 10,000 items without performance degradation.	Should
NFR-29		The system must be scalable to support up to 100 concurrent users without exceeding 80% CPU utilization on standard hardware.	Must
NFR-30	Security	The system must ensure that access to the context module (category, supplier, manufacturer, status, depreciation, recycle bin) is restricted to IT Head/Admin roles only, with role-based authentication enforced at all times.	Must
NFR-31		Supplier registration must enforce security by validating file uploads against malware and limiting to .xlsx for imports, with all operations completing in under 3 seconds.	Must
NFR-32		Deletion of manufacturers must log events for security auditing, with operations restricted to admins and completed securely.	Must
NFR-33		Status registration must validate alphanumeric inputs and enforce uniqueness, with error handling that prevents system crashes.	Must
NFR-34		Depreciation registration must support integer months and float currencies, with input sanitization to prevent injection attacks.	Must
NFR-35		Asset registration must enforce admin-only access and validate inputs (e.g., dates not in future), with secure file handling.	Must
NFR-36		Asset deletions must use soft-delete for recoverability, with admin confirmation and logging.	Must
NFR-37		Asset view pages must display full details securely, with tabs loading asynchronously for	Must

		better performance.	
NFR-38		Manufacturer operations must ensure data security through input sanitization and access controls.	Must
NFR-39		API requests (e.g., for assets, models) must be secure with HTTPS, rate-limited to 100 requests/minute per user to prevent abuse.	Must
NFR-40		Authentication must use secure protocols (e.g., JWT), with sessions expiring after 30 minutes of inactivity for security.	Must
NFR-41		Ticket, repair, and audit modules must ensure data privacy, with operations audited and retained for 1 year.	Must
NFR-42	Usability	Category registration forms must validate inputs in real-time, supporting maximum input lengths (e.g., 100 characters for names), file uploads limited to JPEG/PNG under 5MB, and provide immediate feedback for duplicates or missing required fields, ensuring data integrity and usability.	Must
NFR-43		Supplier list pages must support alphabetical sorting and filtering by type, with search bars responding in under 500ms for improved usability.	Should
NFR-44		Manufacturer pages must be responsive across devices, with hover effects and clickable elements providing intuitive navigation and accessibility compliance (e.g., WCAG 2.1 Level AA).	Must
NFR-45		Recycle bin pages must be searchable and filterable across tabs (assets, components), with pagination supporting 50 items per page for usability.	Should
NFR-46		Asset Model creation must validate all fields (e.g., positive integers for life months, doubles for costs) and support file uploads under 5MB, with the process completing in under 2 seconds.	Must
NFR-47		Reports generation (e.g., asset, depreciation) must export in multiple formats (Excel, PDF, CSV) within 5 seconds for up to 1000 records,	Should

		with toast notifications for usability.	
NFR-48		Supplier operations must validate alphanumeric inputs (up to 100 chars for names, 500 for notes) and handle concurrency without errors.	Should
NFR-49		All pages must include a consistent MAP AMS Footer and be scrollable, ensuring cross-browser compatibility (Chrome, Firefox, Edge) and accessibility features like keyboard navigation.	Must

The table represents a set of non-functional requirements (NFRs) for the MAP Asset Management System (AMS), designed to ensure the system's performance, security, usability, and reliability across its operational context. Each NFR defines a specific quality attribute or constraint from the perspective of the IT Admin and the system's overall architecture. These requirements are intended to support the IT Admin in maintaining a robust and efficient asset management process by establishing standards such as load times under two seconds for category lists, secure authentication with JWT and session expiry, and malware-checked file uploads limited to 5MB. For instance, several NFRs focus on performance metrics, including computations completing in under 100ms for depreciation calculations and report exports finishing within five seconds for 1000 records, ensuring smooth operation with large datasets. Other NFRs emphasize security and compliance, such as admin-only access with role-based authentication, data privacy through auditing of tickets and repairs, and WCAG 2.1 Level AA accessibility for manufacturer pages, alongside requirements for atomic updates and real-time notifications. Additionally, the table addresses scalability and recovery, with provisions for handling 10,000 items in the recycle bin for 90 days and supporting 10 concurrent users without data loss, all aimed at delivering a dependable and user-friendly system for asset management tasks.

User Stories and Use Cases

Table _

User Stories and Use Cases

User Story Number	User Story
US-001	As an IT Admin, I want to create, update, and delete categories with validation and dependency checks so that I can maintain an organized asset categorization system.
US-002	As an IT Admin, I want to view category lists with pagination and quick load times so that I can manage large datasets easily.
US-003	As an IT Admin, I want to register suppliers with malware-checked .xlsx uploads so that I can ensure data security and integrity.
US-004	As an IT Admin, I want to update supplier details and track changes so that I can maintain accurate supplier records.
US-005	As an IT Admin, I want to delete suppliers and recover them within 30 days so that I can manage supplier data with flexibility.
US-006	As an IT Admin, I want to manage manufacturer data with real-time notifications so that I can keep asset-related manufacturer info current.
US-007	As an IT Admin, I want to view manufacturer pages across devices so that I can manage data on any device.
US-008	As an IT Admin, I want to manage status records with uniqueness checks so that I can track asset statuses accurately.
US-009	As an IT Admin, I want to calculate depreciation with precision so that I can assess asset financial value correctly.
US-010	As an IT Admin, I want to manage asset models with validation and file uploads so that I can define asset templates effectively.
US-011	As an IT Admin, I want to view asset model details with tabs so that I can review all relevant model data.
US-012	As an IT Admin, I want to manage assets with secure access and recalculations so that I can maintain an accurate asset inventory.
US-013	As an IT Admin, I want to view asset details securely so that I can monitor asset status and history.

US-014	As an IT Admin, I want to check out assets with ticket integration so that I can assign assets to users efficiently.
US-015	As an IT Admin, I want to check in assets with validation so that I can ensure proper asset return.
US-016	As an IT Admin, I want to generate reports in multiple formats so that I can analyze asset and depreciation data.
US-017	As an IT Admin, I want to manage recycle bin items for 90 days so that I can recover deleted data as needed.
US-018	As an IT Admin, I want to create tickets for asset-related issues so that I can track and resolve problems.
US-019	As an IT Admin, I want to manage repair workflows so that I can restore asset functionality.
US-020	As an IT Admin, I want to conduct audits with data retention so that I can ensure compliance and accuracy.

The table provides a list of user stories for the MAP Asset Management System (AMS), which captures asset based processes in an organizational context. Each user story describes a feature or functionality through the perspective of an IT Admin who will interact with the system. User stories have been designed for the IT Admin to manage a wide range of activities including aggregating areas defined as categories, registering and updating suppliers, managing manufacturers, defining status for assets, calculating depreciations, create and view asset models, registering assets and tracking them, to managing check outs and check ins, generating reports, and recovering deleted records from the recycle bin, creating tickets, conducting repairs, and performing audits. For example, user stories focus on creating and managing structured data forms, like categories and asset models, which identify required fields being validated, and various file upload and attachments to efficiently manage asset name data. User stories also consider functionality like the ability to securely update details of suppliers with history tracking, the ability to view paginated lists of asset data with reasonable load times, and description of a ticket system to check out an asset, and the ability to export a detailed report to

analyze aspects of reported and defined events, all of which adheres to varying security and performance demands. There is additionally an emphasis on monitoring the status of assets, receiving alerts immediately, and controlling workflows, which will include capability of soft-deletes or recovery during a 90-day recycle bin period, all developed to ensure the integrity and compliance of asset inventory.

Figure _ User Use Case Diagram

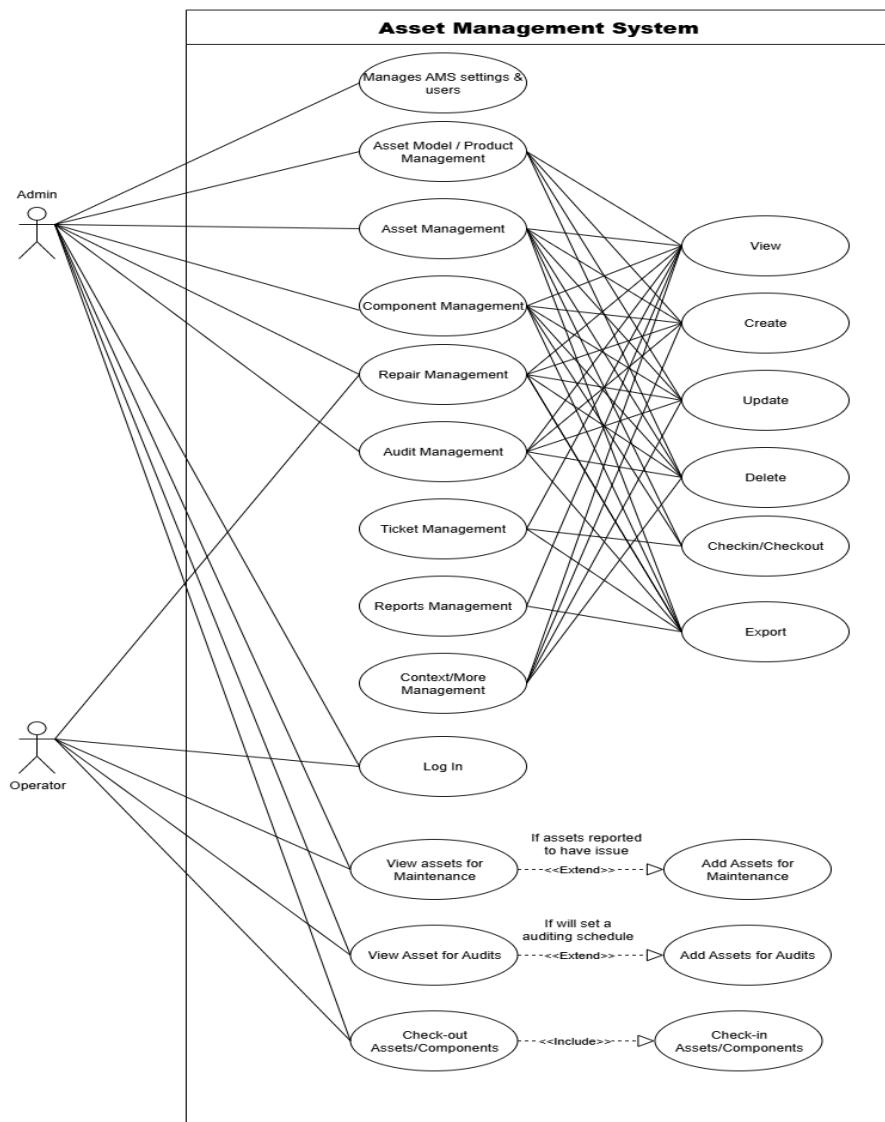


Figure <number> illustrates the use case from the user perspective in the Asset Management System (AMS). It describes a user's ability to perform any of the asset specific use cases available to them based on their roles. The AMS defines two different kinds of users, Admin and Operator, with each role having specific access rights and duties to effectively and securely manage the assets of the organization.

The Admin has full access to all modules and functionalities within the system. This includes the ability to view, create, update, delete, check in and check out, and export data across multiple management modules such as Asset Model/Product Management, Asset Management, Component Management, Repair Management, Audit Management, Ticket Management, Reports Management, and Context/More Management. In addition, the Admin exclusively manages the "Manages AMS Settings & Users" module, which allows configuration of system settings, user account management, and overall administrative control. Through these permissions, the Admin can maintain data consistency, oversee asset operations, and ensure that all records within the system are accurate and up to date.

Meanwhile, the Operator has limited access focused primarily on operational and monitoring tasks. Operators can view records and perform check-in and check-out activities for assets and components. They have access to modules such as Asset Model/Product Management, Asset Management, Component Management, Repair Management, Audit Management, Ticket Management, and Reports Management for viewing purposes. Additionally, Operators can view assets for maintenance and audits and are able to add assets for these processes when issues are reported or when audit schedules are created. However, they do not have access to the "Manages AMS Settings & Users" and "Context/More Management" modules to maintain system security and prevent unauthorized configuration changes.

Acceptance Criteria

Table _

Acceptance Criteria

User Story ID	Acceptance Criteria
US-001	<ul style="list-style-type: none">- Category creation validates inputs (e.g., 100 char limit) and prevents duplicates- Updates are atomic with change logging- Deletion checks dependencies and prevents removal if in use
US-002	<ul style="list-style-type: none">- Category list loads in < 2 seconds for 1000 records- Pagination with responsive numbers and scrollable pages works
US-003	<ul style="list-style-type: none">- Supplier registration completes in < 3 seconds- .xlsx uploads are malware-checked and limited to 5MB
US-004	<ul style="list-style-type: none">- Supplier updates preserve historical data via versioning- Concurrent edits are handled gracefully
US-005	<ul style="list-style-type: none">- Supplier deletion is reversible via recycle bin for 30 days- Error message returns in < 1 second if dependencies exist
US-006	<ul style="list-style-type: none">- Manufacturer registration handles 10 concurrent users without data loss- Updates trigger real-time notifications in < 1 second
US-007	<ul style="list-style-type: none">- Manufacturer pages are responsive with hover effects- Meets WCAG 2.1 Level AA accessibility
US-008	<ul style="list-style-type: none">- Status registration enforces alphanumeric uniqueness- Updates are backward-compatible with existing links- Deletion includes dependency checks and soft-delete
US-009	<ul style="list-style-type: none">- Depreciation calculations are accurate to 2 decimal places- Computations complete in < 100ms per record- Updates recalculate affected assets for batches up to 1000
US-010	<ul style="list-style-type: none">- Asset Model creation validates fields (e.g., positive integers) and limits file uploads to 5MB- Process completes in < 2 seconds- Updates maintain referential integrity
US-011	<ul style="list-style-type: none">- Asset Model view loads tabs in < 1 second- Responsive design works on mobile- Advanced search handles multiple filters in < 500ms

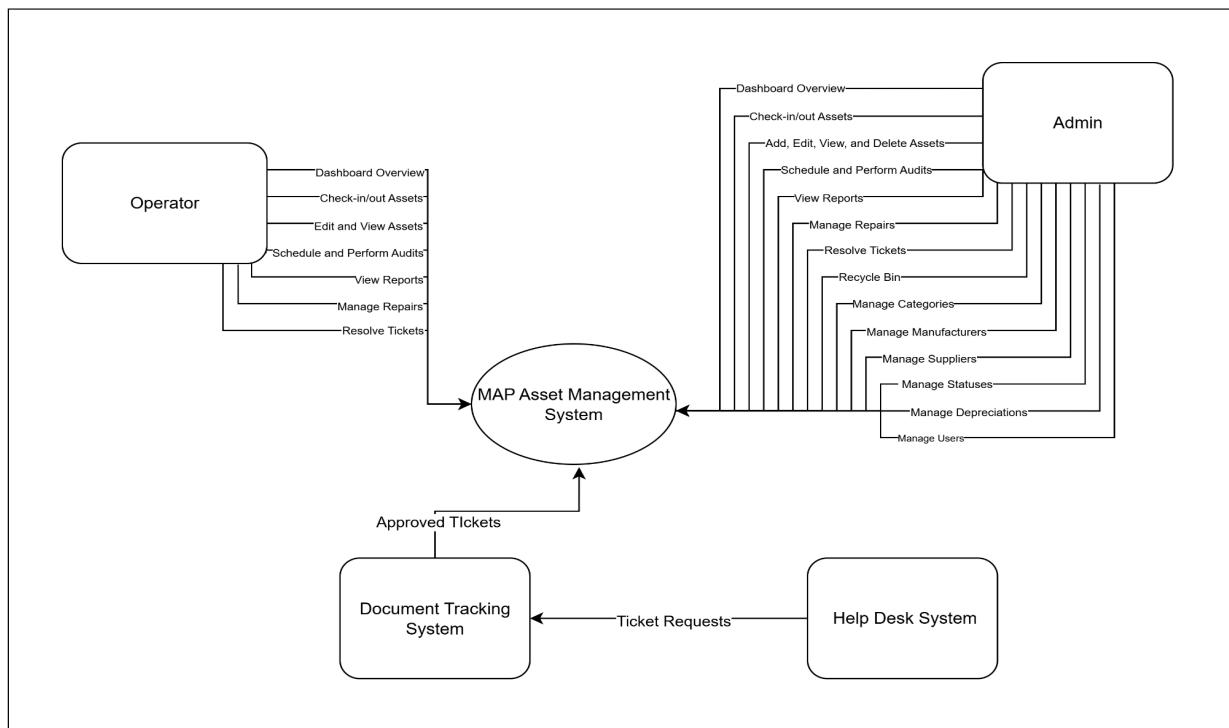
US-012	<ul style="list-style-type: none"> - Asset registration is admin-only with date validation (no future dates) - Updates trigger depreciation recalculations atomically - Deletions use soft-delete with recycle bin
US-013	<ul style="list-style-type: none"> - Asset view displays details securely with async tab loading - No sensitive data exposure - Advanced search supports multi-field filtering
US-014	<ul style="list-style-type: none"> - Asset check-out integrates with tickets and completes in < 2 seconds - Real-time status updates provided
US-015	<ul style="list-style-type: none"> - Asset check-in validates against existing requests - Prevents unauthorized actions
US-016	<ul style="list-style-type: none"> - Reports export in Excel, PDF, CSV within 5 seconds for 1000 records - Toast notifications appear - Asset and depreciation reports generated
US-017	<ul style="list-style-type: none"> - Recycle bin retains items for 90 days, handling 10,000 items - Search and filter across tabs with 50 items/page
US-018	<ul style="list-style-type: none"> - Tickets are created with secure access - Linked to assets and audited for 1 year
US-019	<ul style="list-style-type: none"> - Repairs are managed with workflow tracking - Data privacy ensured with auditing
US-020	<ul style="list-style-type: none"> - Audits are conducted with data retention - Ensures compliance with security protocols

The table represents a set of acceptance criteria for the MAP Asset Management System (AMS), designed to validate the system's functionality, performance, and compliance with specified requirements. Each criterion is associated with a unique "User Story ID" such as US-001 to US-020 and defines specific conditions that must be satisfied to confirm the successful implementation of corresponding user stories from the IT Admin's perspective. These criteria ensure that key processes, such as category creation with input validation like 100-character limit and duplicate prevention, supplier registration completing in under three seconds with malware-checked 5MB uploads, and depreciation calculations achieving accuracy to two decimal places within 100ms, meet defined standards. For instance, several criteria focus on performance, including category lists loading in less than two seconds for 1000 records with

responsive pagination, asset model views loading tabs in under one second, and reports exporting in Excel, PDF, or CSV within five seconds for 1000 records with toast notifications. Other criteria emphasize security and data integrity, such as admin-only asset registration with no future date validation, supplier updates preserving historical data via versioning, and recycle bin retention handling 10,000 items for 90 days with searchable filters. Additionally, the table addresses usability and compliance, with requirements like WCAG 2.1 Level AA accessibility for manufacturer pages, real-time notifications for manufacturer updates, and audit data retention to ensure security protocols, providing a robust framework for system validation and quality assurance.

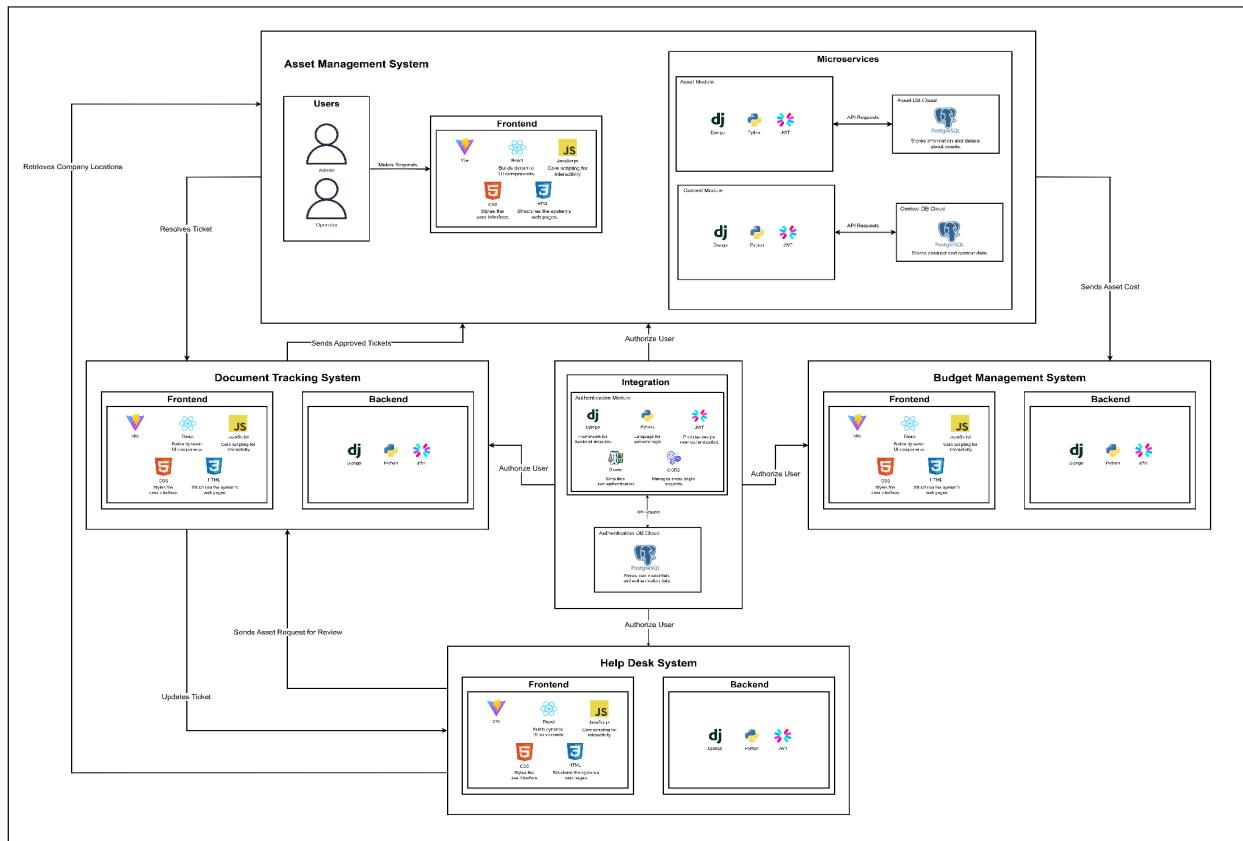
High-Level System Design

Figure __. System Context Diagram



The System Context Diagram illustrates the MAP Asset Management System as the central platform that interacts with both human users and external systems. The Admin and Operator users perform core asset management functions such as check-in/check-out, auditing, and repair tracking, with the Admin having additional privileges to manage users, categories, and depreciation settings. The system also communicates with the Help Desk System to receive ticket requests and with the Document Tracking System to send approved tickets. This diagram provides an overview of all external interactions that define the system's operational boundaries.

Figure _ . Overview of the Integrated System

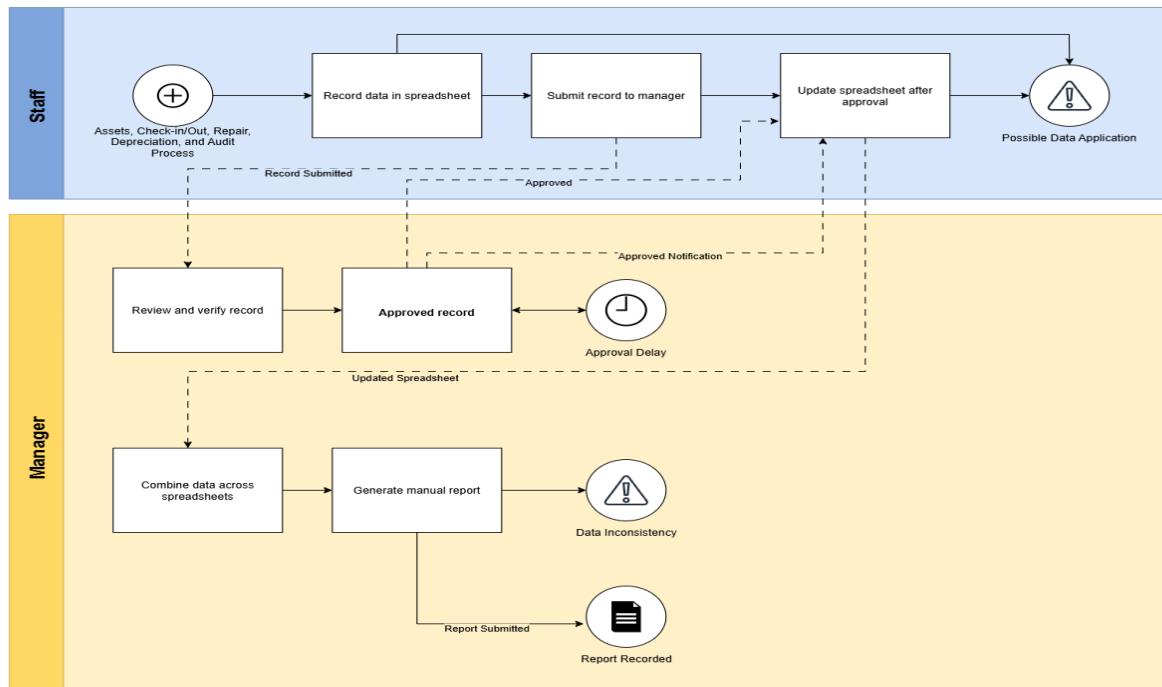


The figure above illustrates the integrated architecture of the MAP Asset Management System with other organizational systems. All systems, including the Asset Management, Budget Management, Document Tracking, and Help Desk systems, use a single centralized Authentication Service that manages user authorization and access control. The Asset

Management System serves as the core platform that handles asset tracking, repairs, audits, and check-in and check-out operations. It communicates with the Budget Management System, which retrieves asset cost information and displays this data in its financial dashboard. The Help Desk System provides location data and sends asset-related ticket requests to the Document Tracking System for review and approval. Once approved, the Document Tracking System forwards the validated tickets to the Asset Management System. The Asset Management System then resolves or updates the ticket status based on the actions required by the Budget Management System. This integration ensures that asset, budget, and document processes remain synchronized across departments, enabling efficient tracking, accountability, and workflow automation.

Business Process Architecture

Figure ___. As-Is Business Process Model of MAP Asset Management System



The As-Is Business Process Model shows the current manual process used by MAP Active Philippines in managing its IT assets. It includes activities such as asset recording, check-in/check-out, repair, depreciation, and audit. The diagram also shows how staff and managers handle records using spreadsheets and how the process often leads to delays and data problems.

Roles and Responsibilities

Staff

- Record asset data in spreadsheets.
- Submit the record to the manager for checking and approval.
- Update the spreadsheet after the manager's approval.
- Handle any errors found in the data.

Manager

- Review and verify the asset record.
- Approve or return records for correction.
- Combine all spreadsheets into one file.
- Generate manual reports.
- Submit the final report for recording.

Table _

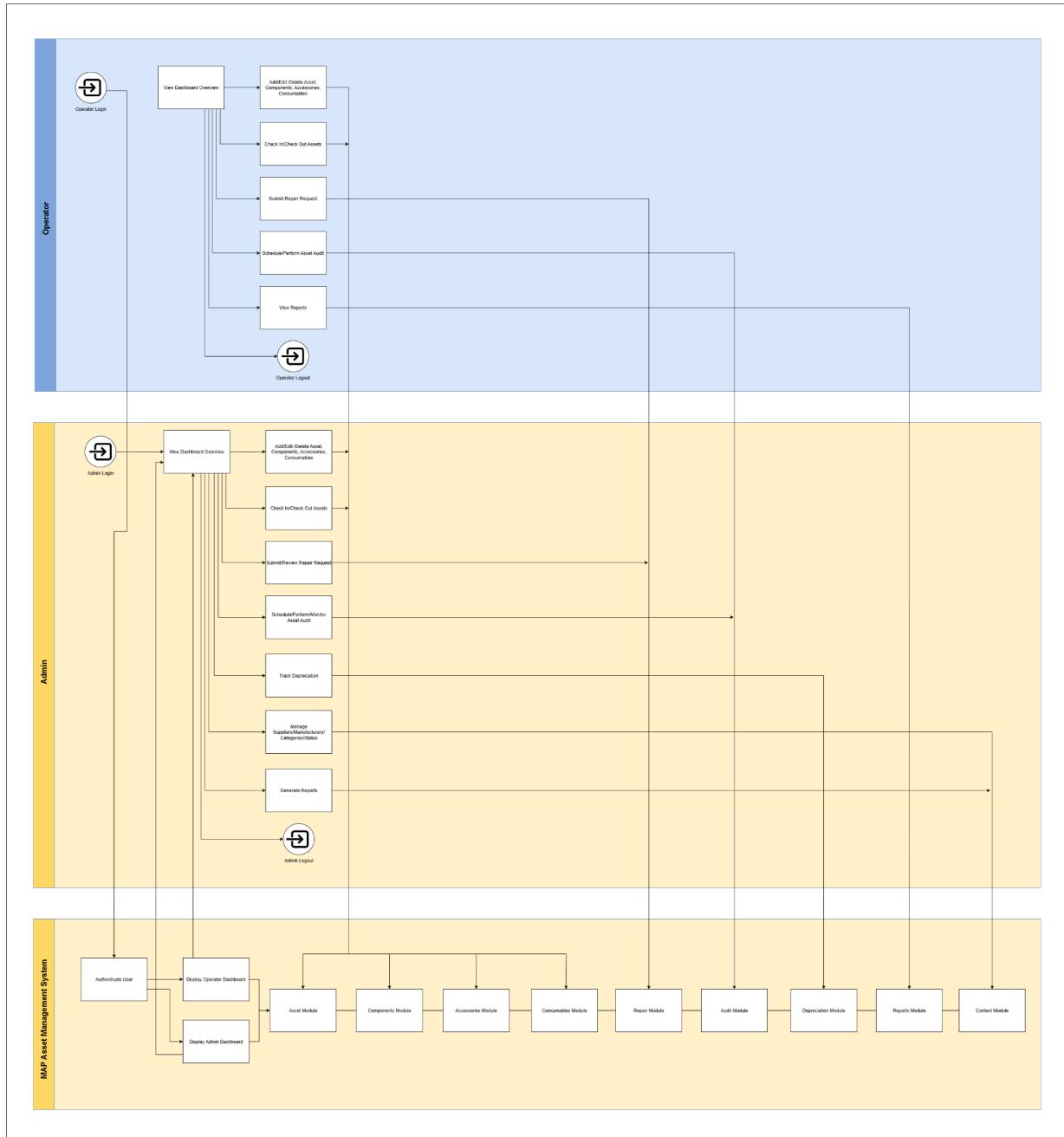
Process Flow and Identified Pain Points

Process Description	Pain Point / Observation
Staff record asset information in a spreadsheet.	Manual data entry can lead to errors and incomplete information.
The record is submitted to the manager for	Submitting through email or shared files may

checking and approval.	cause version errors or lost files.
The manager reviews it for accuracy.	Checking each record takes time and often causes approval delays.
After approval, staff update the spreadsheet.	Repeated changes can cause confusion or overwritten data.
Manager consolidates all approved data from various spreadsheets.	Manual consolidation increases the risk of duplication and inconsistencies.
A manual report is then generated from the consolidated data.	Report preparation is time-consuming and may not always reflect the most recent updates.
Finally, the report is submitted and recorded, marking the end of the process.	The process is fully manual and takes a long time to complete.

The As-Is Business Process Model presents the existing manual process used by MAP Active Philippines in managing its IT assets. The process mainly depends on spreadsheets for recording, checking, and reporting asset information. Staff members are responsible for encoding and updating asset data, while managers review, approve, and combine these records to prepare reports. However, this manual process often results in several issues such as data errors, missing files, and slow approval. The use of multiple spreadsheets makes it difficult to track changes and maintain accurate records. Preparing reports also takes time and may not reflect the latest information. Overall, the current process is time-consuming, prone to human error, and lacks efficiency, which affects the company's ability to manage its IT assets properly.

Figure __. To-Be Business Process Model of MAP Asset Management System

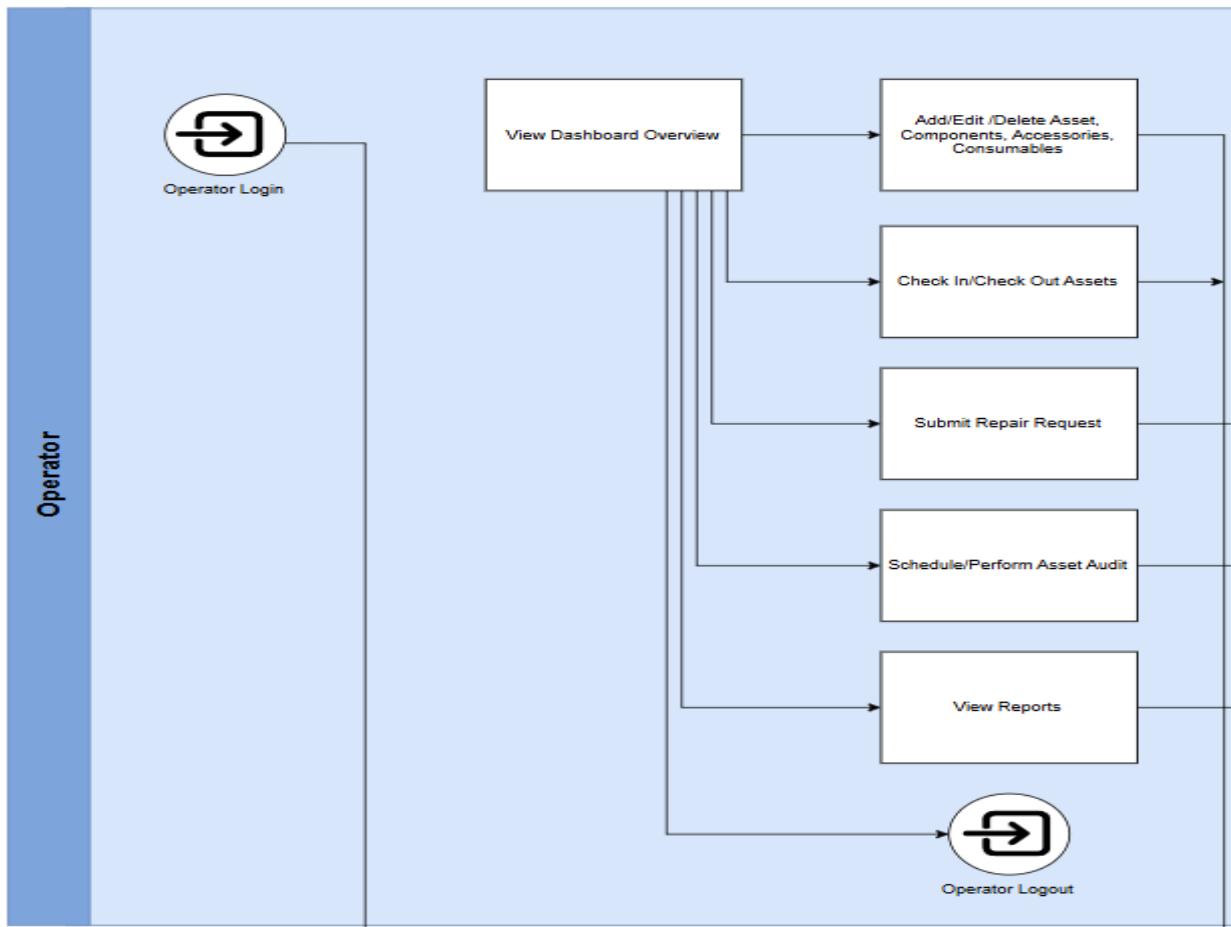


The To-Be Business Process Model shows the improved process of managing IT assets using the MAP Asset Management System. This new system replaces the manual and paper-based process with a computerized one. It allows both the Admin and Operator to

perform tasks such as adding assets, checking in and checking out items, sending repair requests, tracking depreciation, and creating reports. With this system, data is stored in one place, making asset management faster, more accurate, and easier to monitor.

Roles and Responsibilities

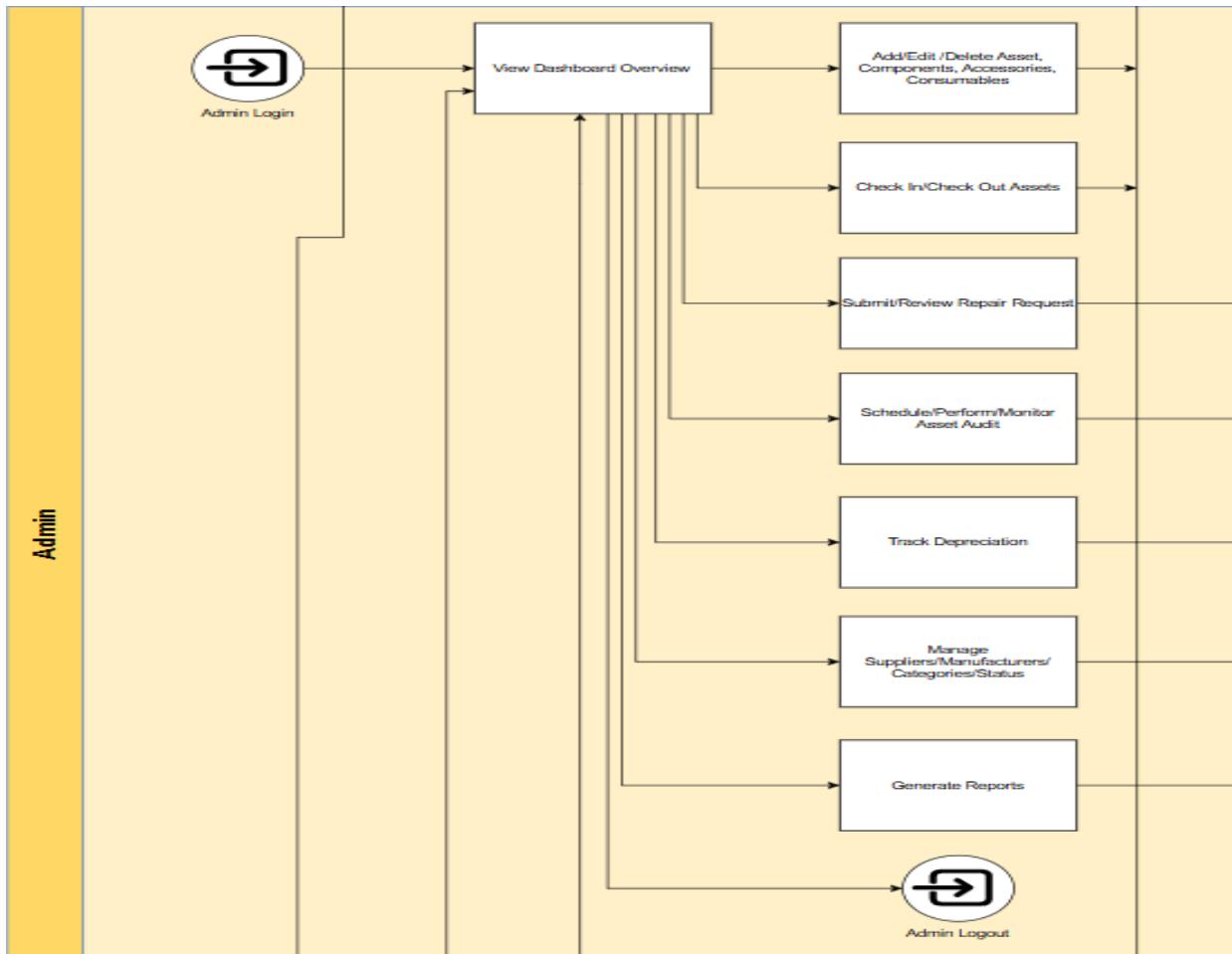
Figure __. Operator



- Log in to the system through the Operator Dashboard.
- Add, edit, or delete asset information, including components, accessories, and consumables.
- Check in and check out assets when they are borrowed or returned.
- Submit repair requests for damaged or faulty equipment.

- Schedule and perform asset audits.
- View reports to check the status and condition of assets.
- Log out securely after finishing all tasks

Figure _ Admin



- Log in to the system through the Admin Dashboard.
- Add, edit, or delete asset records, components, accessories, and consumables.
- Review and approve repair requests from operators.
- Schedule, perform, and monitor asset audits.
- Track the depreciation or value decrease of assets.
- Manage suppliers, categories, and asset statuses.

- Generate reports for monitoring and documentation.
- Log out securely after finishing system activities.

Figure _ . MAP Asset Management System

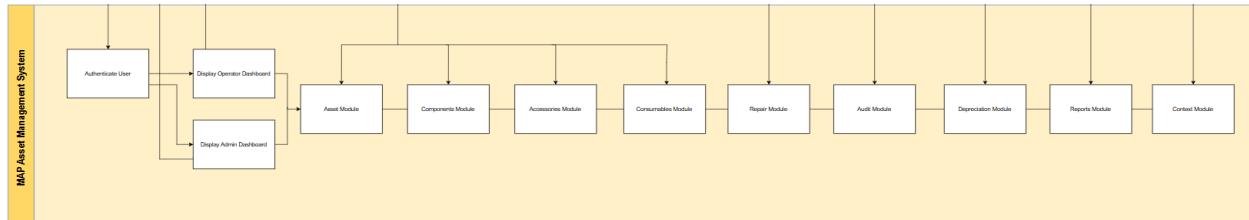


Table _

Process Flow and Improvements

Process Description	System Improvement / Benefit
User authentication through the login page	Provides secure access and separates admin and operator functions
Asset information is recorded and stored in the database	Data is managed through connected microservices for accurate and consistent records.
Asset movement (check-in/check-out) is monitored automatically	Updates records in real time and ensures accurate asset tracking.
Repair requests are created and reviewed within the system	Keeps a clear record of maintenance history and status.
Audit schedules and results are managed through the system	Makes audits easier to organize and follow up.
Depreciation values are computed by the system	Ensures consistent and reliable calculation of asset value.
Supplier, category, and status information are maintained	Keeps all records organized and up to date.
Reports are generated from stored data	Produces accurate summaries for decision-making.
Users log out after completing their activities	Protects data and maintains system security.

The To-Be Business Process Model presents the improved process of managing IT assets through the MAP Asset Management System. It replaces the manual and spreadsheet-based process with a computerized system that allows the Admin and Operator to perform their tasks more efficiently. These include adding and updating asset information, checking in and checking out items, sending repair requests, monitoring depreciation, and generating reports.

The system uses microservices to support automation, data accuracy, and real-time updates. It ensures that all records are well-organized, accessible, and secure. Overall, the To-Be Business Process Model provides a faster, safer, and more reliable way of managing the company's IT assets.

Analysis of Improvements

The MAP Asset Management System shows clear improvements compared to the old manual process that relied on spreadsheets and emails. The new system helps reduce the time needed for asset recording, checking, and report preparation. In the previous setup, the process usually took around five days to complete because all data were entered and reviewed using spreadsheets. With the new system, these tasks can now be done within a shorter time depending on the availability of the Admin.

The system also helps reduce human error since all data are stored and updated in one place. This prevents missing, repeated, or incorrect information that often occurred when using spreadsheets. The automatic calculation of depreciation and real-time tracking of assets also make the records more accurate and reliable. In addition, the audit and report features can now be completed in just a few minutes instead of several hours.

Overall, the new system makes the work of both the Admin and Operator more organized and efficient. It helps them save time, lessen errors, and maintain updated records.

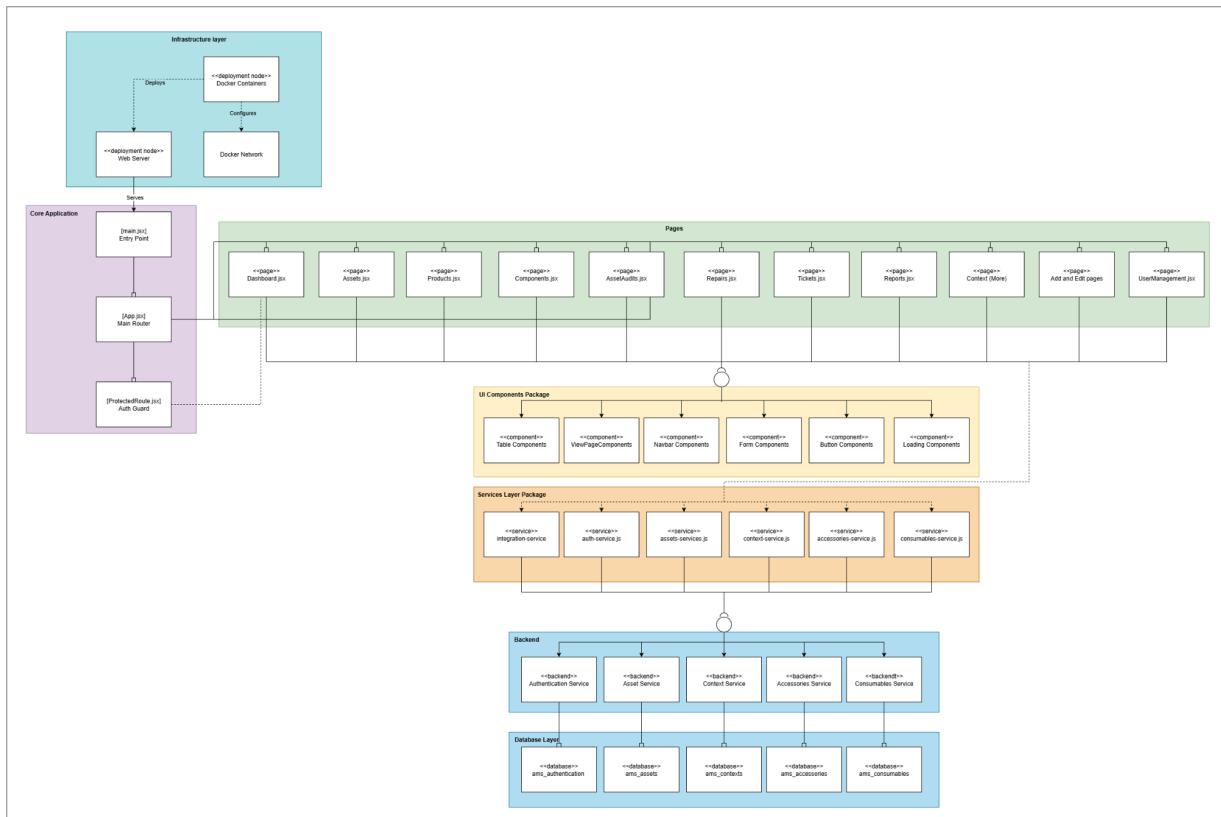
The MAP Asset Management System also supports faster decision-making and smoother operations in managing the company's IT assets.

Application Architecture

The Application Architecture of the MAP Active IT Asset Management System with Forecasting (MapAms) explains how the system is organized and how its parts work together. It provides a safe, stable, and flexible structure that supports managing and keeping track of assets. The design keeps the system running smoothly, protects data, and makes it easy to maintain and update.

Components of Application Architecture

Figure _ Components Application Diagram



This component diagram shows the structure of the system and how each part connects and interacts with the others. At the bottom part is the Infrastructure Layer, which includes the web server hosted inside Docker containers. These containers are linked through a Docker network, allowing the system to be deployed in a stable and flexible environment. This layer ensures that the system runs smoothly and can easily be configured or updated when needed. It serves as the foundation that supports all the other layers above it, making sure the application is properly hosted and accessible.

Above it is the Core Application, which serves as the central part of the entire system. The main.jsx file acts as the entry point where the application starts, while the App.jsx file functions as the main router that connects all pages and handles navigation between them. The ProtectedRoute.jsx file plays an important role in maintaining security by allowing only authorized users to access certain parts of the system. Together, these components make sure that users can move around the system easily and securely, while keeping the structure organized and efficient.

The Pages Layer contains all the main sections that users interact with. It includes pages such as Dashboard, Assets, Products, Repairs, Tickets, Reports, and User Management. Each page has its own specific function, helping users view, manage, and update different kinds of information. These pages are connected to the UI Components Package, which contains reusable interface elements like tables, buttons, forms, loading indicators, and navigation bars. This setup allows the system to maintain a consistent design and makes it easier to manage or update user interface parts without changing the whole system.

The Services Layer Package acts as the bridge between the frontend and the backend. It manages data communication and processing by using files such as auth-service.js, assets-service.js, contact-service.js, and accessories-service.js. These services handle the

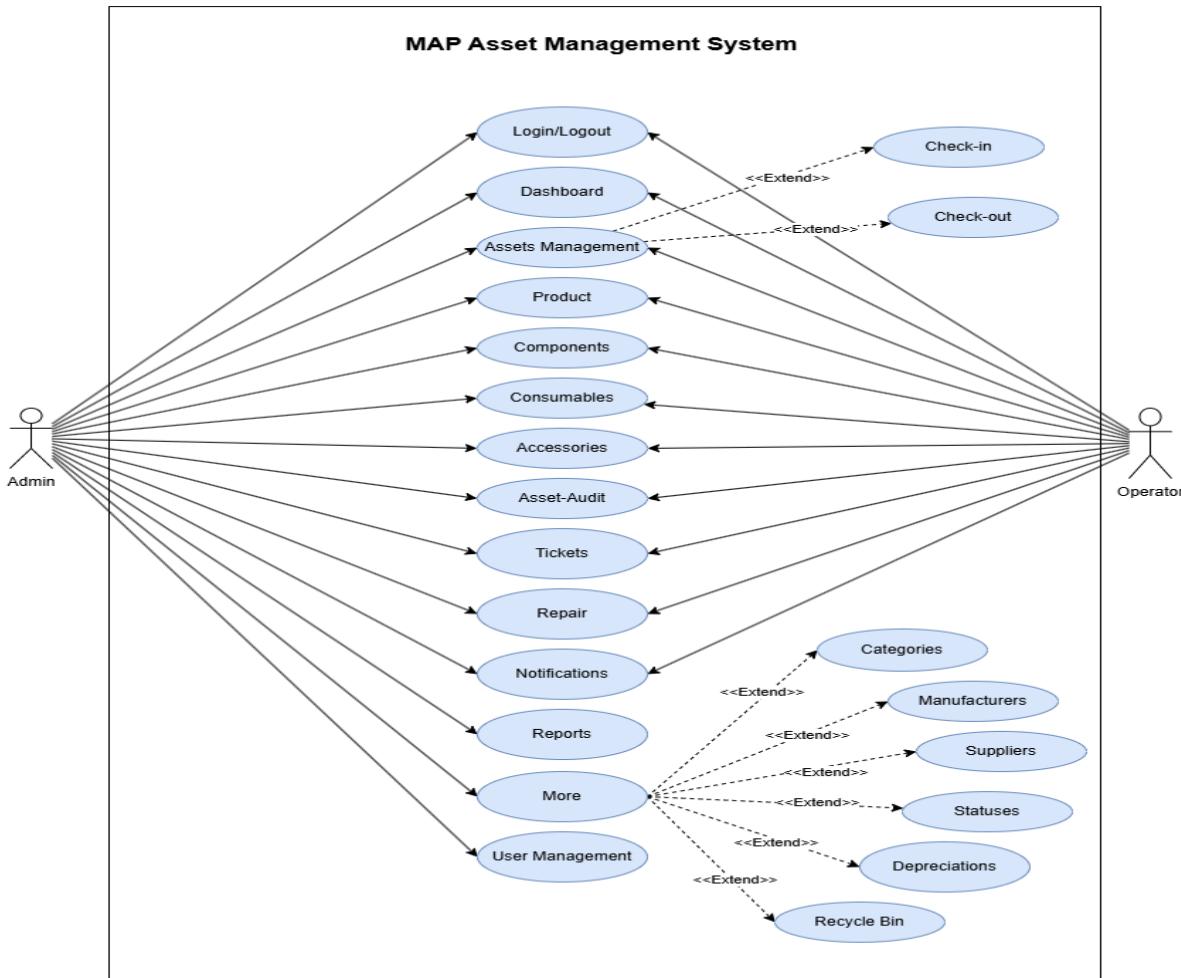
requests and responses between the user interface and the backend, ensuring that the system's operations run correctly. Below it is the Backend Layer, which performs the main logic and processing for different features such as authentication, asset management, contact management, and accessory tracking.

At the lowest level is the Database Layer, which is responsible for storing all data used by the system. Each database handles a specific area of information, such as ams_authentication for user accounts, ams_assets for asset records, ams_contacts for contact details, and ams_accessories for accessory data. By organizing the data into separate databases, the system can easily retrieve and update information without affecting other parts.

Overall, this diagram shows the structure of the system where every layer has a clear and specific role. The infrastructure provides the environment, the core application handles routing and security, the pages present the interface, the services manage communication, and the backend and database handle processing and storage. This layered design helps the system stay organized, efficient, and easy to maintain while ensuring all components work together smoothly.

Detailed UML Diagrams for Key Services

Figure __. Use Case Diagram



The diagram presents the Use Case Diagram of the MAP Asset Management System, which shows how the two main users, the Admin and the Operator, interact with the system and its different functions. Both users are required to log in to access the system and can log out after completing their tasks. They can also use the dashboard to view important information and perform asset management activities. These include adding, updating, or removing asset details to maintain accurate records. The system ensures that both users can perform their roles efficiently within a controlled and organized environment.

The Admin has wider access to the system and is responsible for managing most of its core functions. These include handling products, components, consumables, accessories, and conducting asset audits. The Admin can also process tickets, manage repair requests, send notifications, and generate reports for monitoring and documentation. Another important role of the Admin is user management, which involves adding, editing, or removing users who have access to the system. The “More” section of the system allows the Admin to manage categories, manufacturers, suppliers, statuses, depreciations, and the recycle bin, which help in keeping asset data organized and updated.

The Operator, on the other hand, has a more specific role focused on daily operations involving assets. The Operator can check in and check out assets, which are connected to the asset management and dashboard functions. This feature helps track the movement and availability of assets in real time. By providing these functions, the MAP Asset Management System supports accurate record keeping and efficient asset monitoring.

Sequence Diagram

Figure #. Sequence Diagram 1

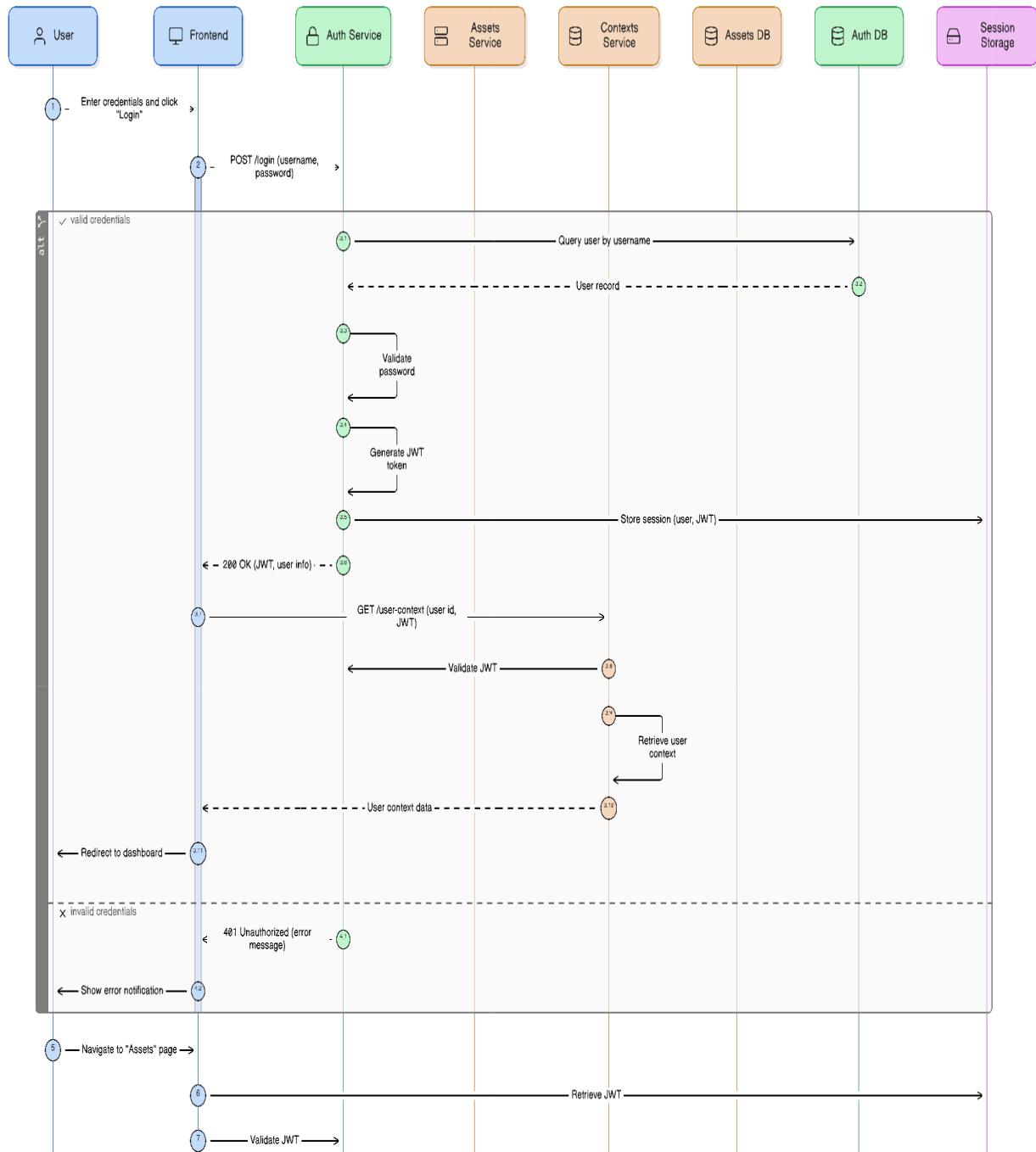


Figure __. Sequence Diagram

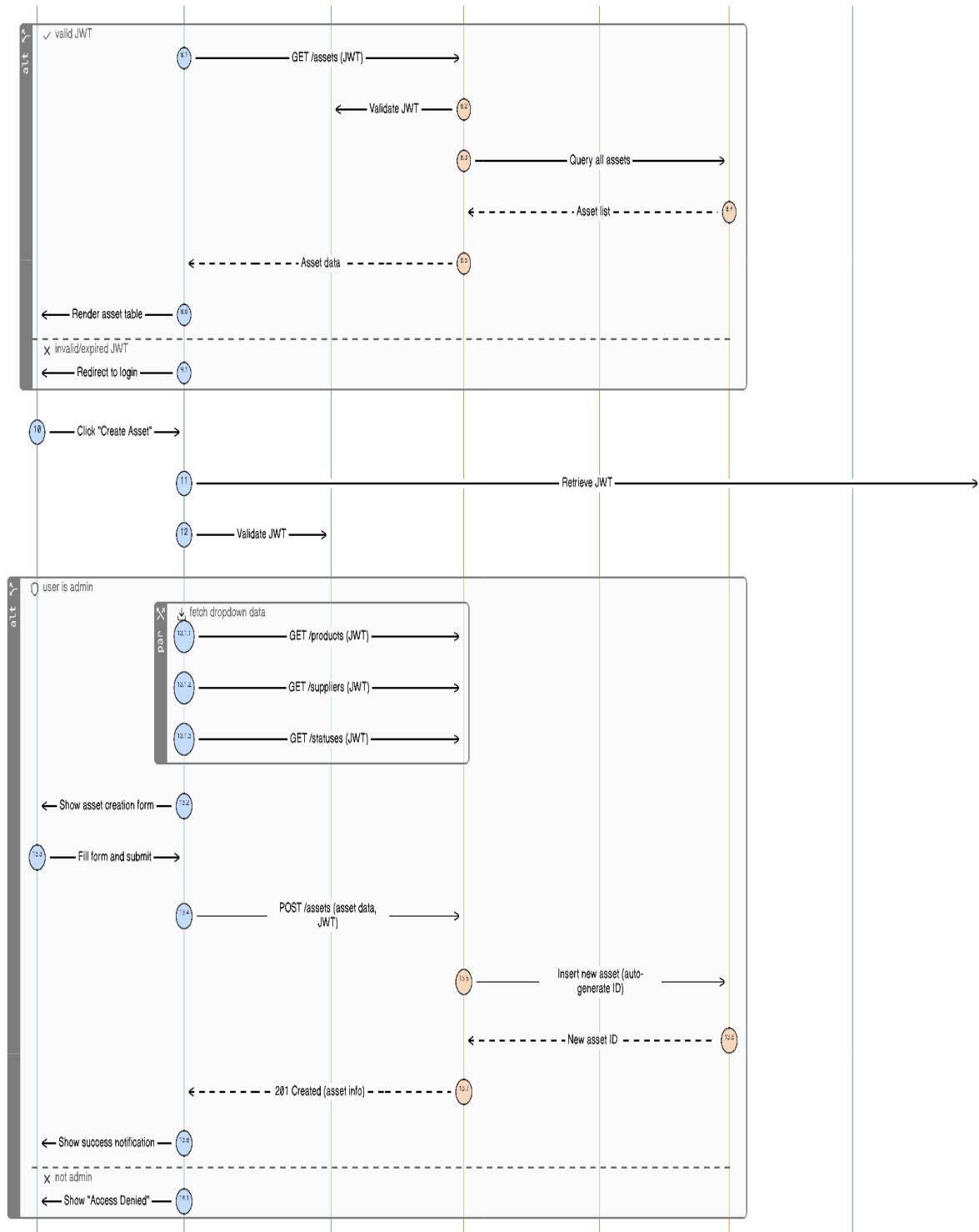
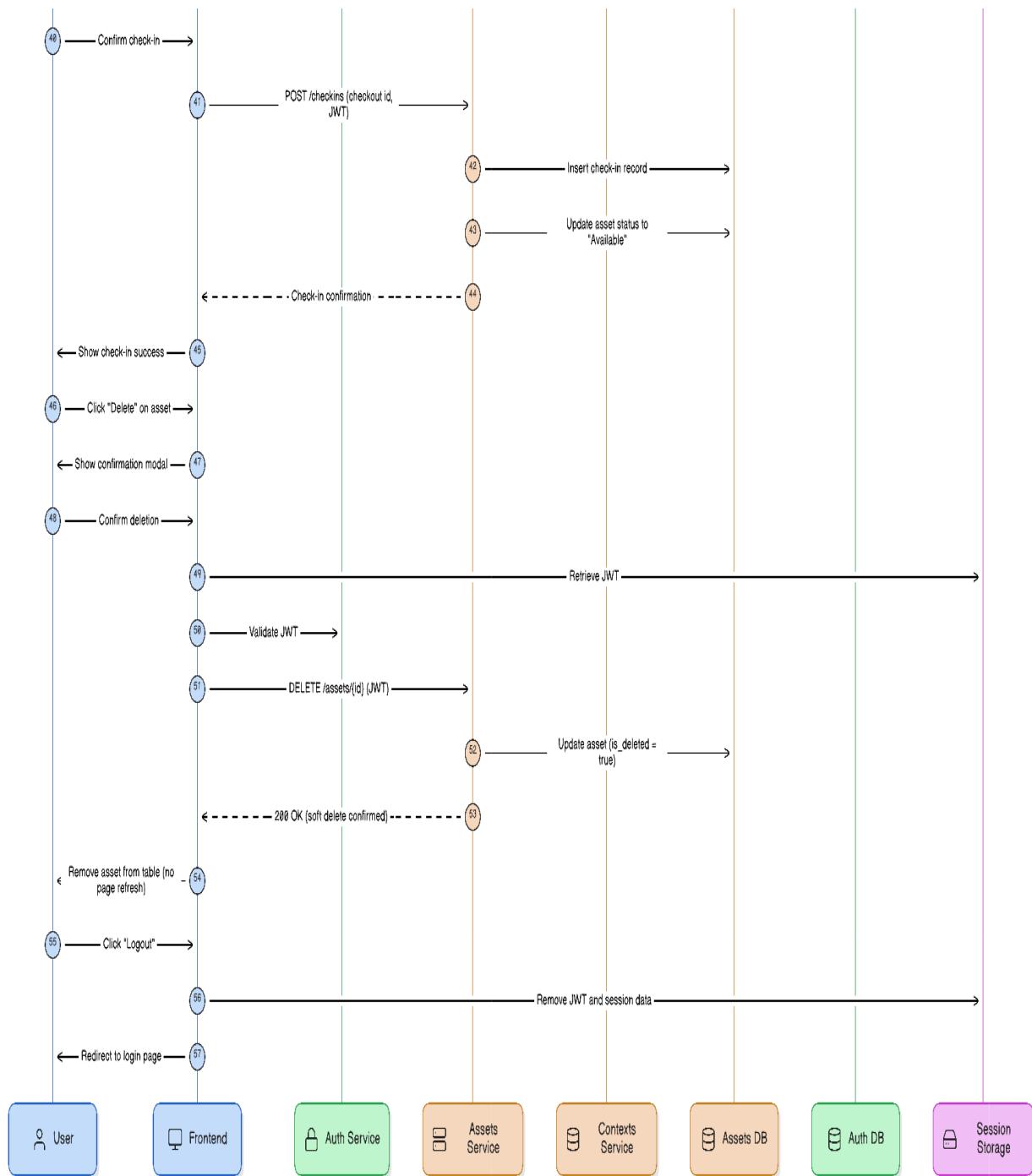


Figure __. Sequence Diagram



The System Sequence Diagrams above explain how the system handles each user process from start to finish. They focus on the flow of actions and the interaction between the user, frontend, backend services, and databases for key operations such as logging in, viewing assets, creating new assets, checking assets in and out, deleting assets, and logging out. Each step is illustrated in order, making it easier to understand how the system responds to user actions and how multiple processes occur behind the scenes. User requests can result in data retrieval, updates, or the creation of new records, and all of these steps are clearly represented to show the logical sequence of operations.

These diagrams also provide a detailed view of how data moves through the system, including the addition, updating, retrieval, and soft deletion of information. The system keeps track of asset status, checkouts, and check-ins over time, ensuring that records remain accurate and up to date. Connections between different components show how one action can affect multiple parts of the system, such as updating an asset's record while recording a checkout history and generating logs. This detailed representation helps in understanding the full flow of operations, how data integrity is maintained, and how internal processes are organized.

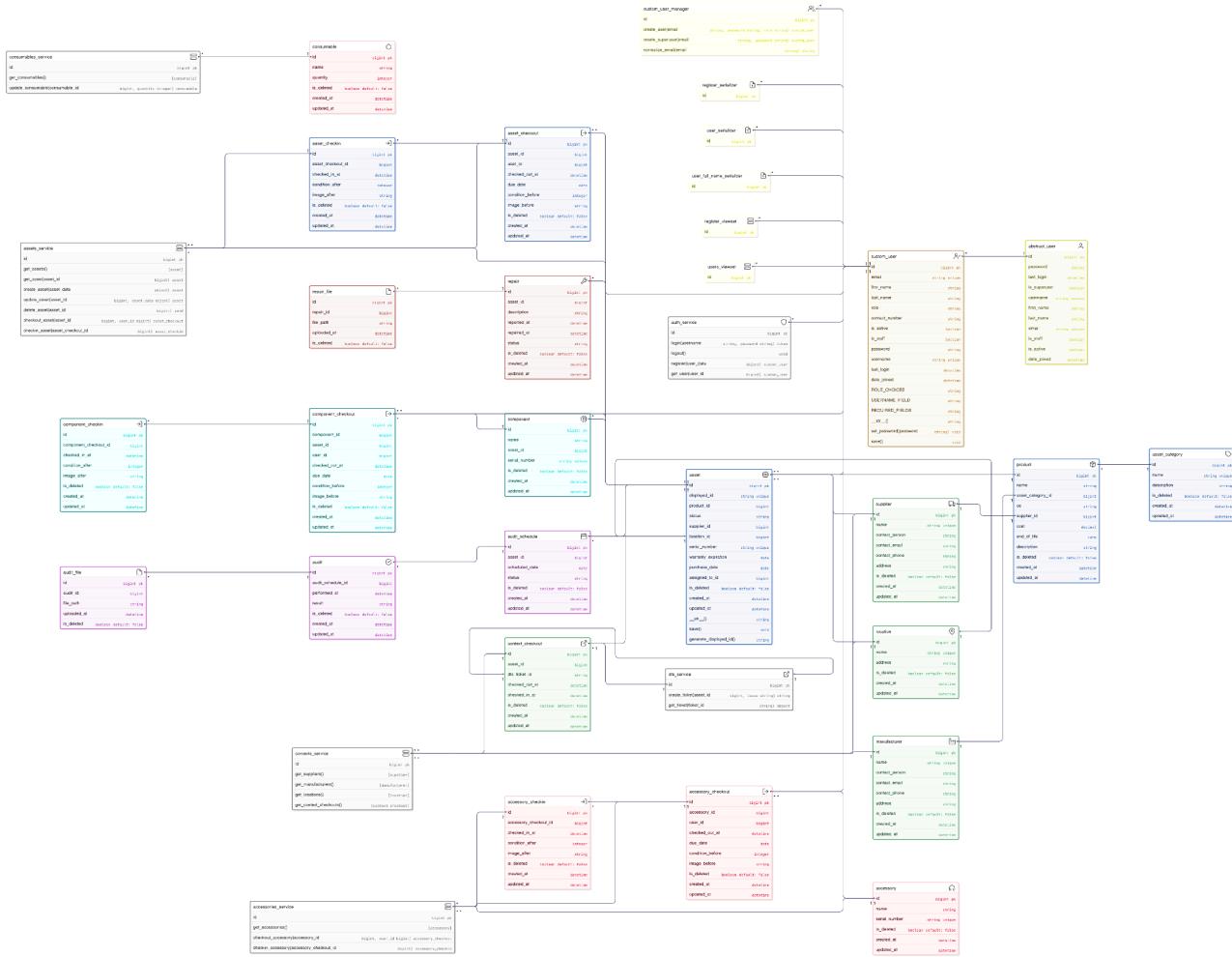
Security and access control are included in the flow. The system checks JWT tokens, confirms user roles, and makes sure that only authorized users can perform certain actions. Session management is represented to show how secure access is kept throughout each process. Error handling is included to illustrate how the system responds to invalid inputs or failed operations, keeping data safe and guiding users properly when issues occur.

Interactions between different microservices, API calls, and database operations are also illustrated. Each service's role is shown, including how data dependencies are managed and how tasks are separated between components. External integrations, such as DTS help desk tickets, are included to show how requests from other systems are handled and tracked.

These representations make the system structure clear, showing how complex operations are managed efficiently.

Class Diagrams

Figure _ Class Diagram



The figure above shows the class diagram, which presents the overall structure and connections of the system's main parts. It explains how different components work together to manage, track, and maintain assets. The diagram includes sections for important tasks such as

recording assets, organizing them into categories, managing supplier information, tracking repairs, handling help desk tickets (DTS tickets), and managing inventory.

At the center of the diagram is the Asset class, which is linked to Asset Category, Supplier, Manufacturer, and Product to describe the type and source of each asset. The Asset class is also connected to records like Asset Checkout, Asset Check-in, Asset Repair, and DTS Tickets. DTS tickets represent help desk requests for maintenance or problem solving. These links allow the system to keep complete and up-to-date information about each asset and any help desk requests related to it.

Other items such as Components, Accessories, and Consumables are included to manage smaller items used with the main assets. Each has its own check-in and checkout records for proper monitoring. Reference classes such as Supplier, Manufacturer, and Location help keep records accurate and consistent. The diagram shows different types of relationships, including one-to-many and many-to-one, meaning one asset can have many repair, movement, or ticket records while still being linked to a single category or supplier.

The diagram also shows how the backend and frontend parts interact through API calls, allowing data to flow between them. Relationships such as inheritance, composition, aggregation, and association are used to organize the information. Rules like soft deletion, unique IDs, role-based access, and integration with DTS tickets are included to keep the system secure and organized.

Data Architecture and Design

Logical Data Model (ERD)

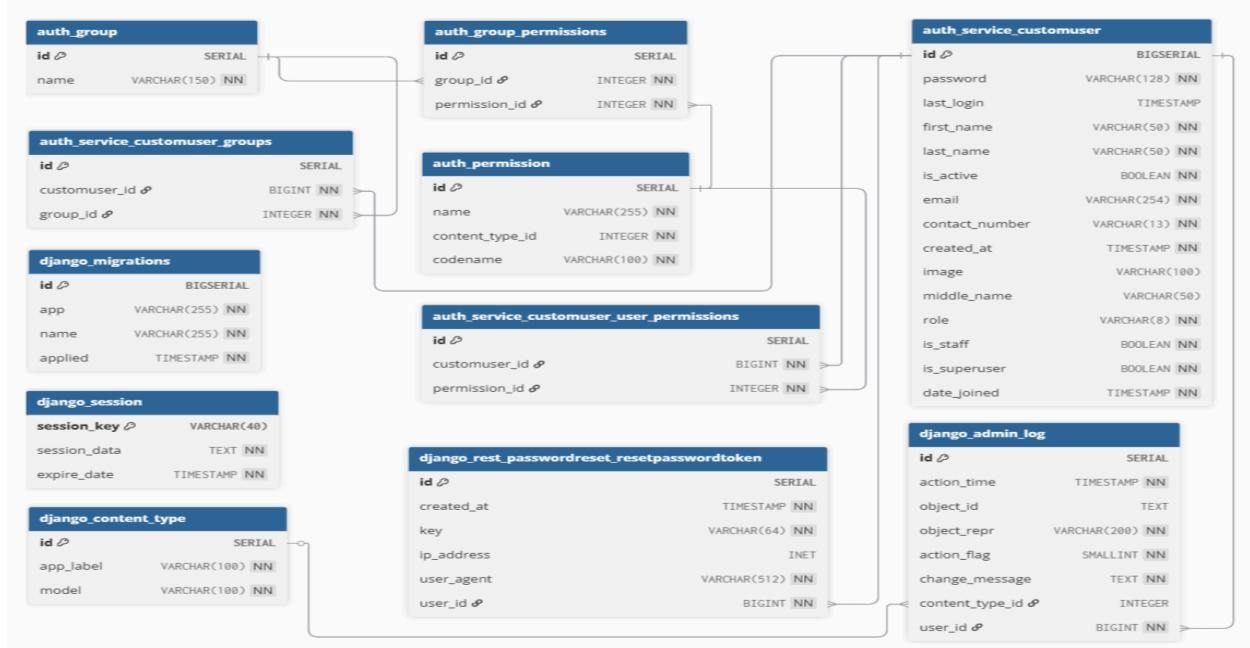
The Logical Data Model represents the conceptual structure of the system's data. It identifies the main entities, their attributes, and the relationships between them. In this system,

each microservice maintains its own logical model to ensure data separation, modularity, and independent scalability. The following subsections describe the Entity Relationship Diagrams (ERDs) and key relationships for each microservice.

Authentication Service

The Authentication Service manages user accounts, login credentials, group memberships, and permissions. It follows Django's built-in authentication structure, extended with a custom user model for additional fields. This service ensures secure access, role-based authorization, and session tracking.

Figure _ Authentication Service



Key Relationships in the Service

In the context of this database model, the primary entities include Custom Users, Groups, Permissions, Sessions, and related Django system tables. The following relationships define how these entities interact:

1. Custom Users and Groups (M:M Relationship)

A Custom User can belong to multiple Groups, and each Group can include multiple Users. This relationship is managed through the auth_service_customuser_groups junction table.

2. Custom Users and Permissions (M:M Relationship)

A Custom User can be assigned multiple Permissions directly, while a Permission can be granted to several Users. This association is maintained through the auth_service_customuser_permissions table.

3. Groups and Permissions (M:M Relationship)

Each Group can have multiple Permissions that define access rights, and each Permission can be linked to multiple Groups. This relationship is implemented through the auth_group_permissions table.

4. Permissions and Content Types (M:1 Relationship)

Each Permission corresponds to one Content Type, representing a specific Django model. The django_content_type table defines these mappings.

5. Custom Users and Sessions (1:M Relationship)

A Custom User can have multiple active sessions. Each record in the django_session table represents one active session per user.

6. Custom Users and Admin Logs (1:M Relationship)

Each django_admin_log entry is linked to one Custom User, who may have multiple log records of their activities.

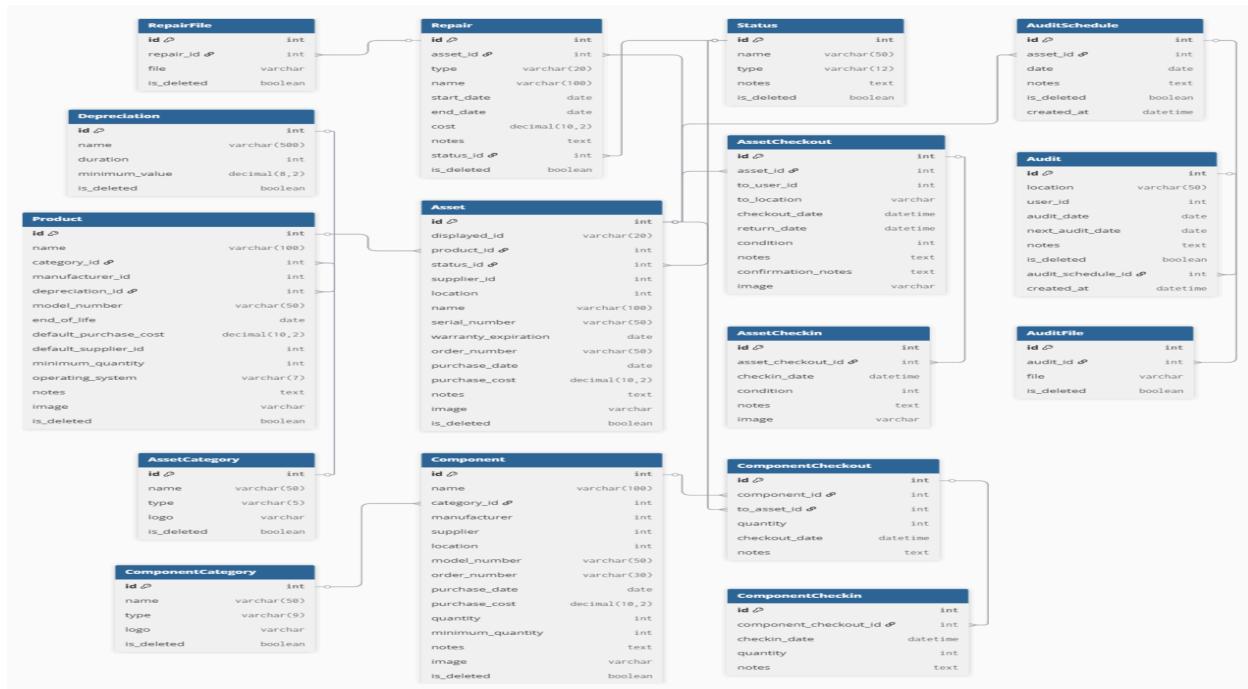
7. Custom Users and Password Reset Tokens (1:1 Relationship)

Each Custom User can have one active Password Reset Token for secure password recovery, managed through the django_rest_passwordreset_resetpasswordtoken table.

Assets Service

The Assets Service oversees the entire lifecycle of physical and component assets, including registration, depreciation, checkout, repair, and audits. It integrates categories, manufacturers, suppliers, and depreciation schedules to ensure efficient tracking, accountability, and traceability of all assets.

Figure __. Assets Service



Key Relationships in the Service

In the Assets Service, assets interact with categories, manufacturers, suppliers, and depreciation methods. They also link to checkouts, repairs, and audits, enabling efficient tracking and management.

1. AssetCategory and Product (1:M Relationship)

Each AssetCategory can include multiple Products, but each Product belongs to one AssetCategory.

2. Depreciation and Product (1:M Relationship)

A single Depreciation method can apply to multiple Products, defining how their value decreases over time. Each Product is linked to one Depreciation record.

3. Product and Asset (1:M Relationship)

A Product can have multiple Assets registered under it, representing the physical units derived from that product type. Each Asset is associated with one Product.

4. Status and Asset (1:M Relationship)

Each Status record defines an operational state (e.g., Deployed, Deployable, Archived) and can be assigned to multiple Assets. Each Asset is linked to one Status.

5. Asset and AssetCheckout (1:M Relationship)

An Asset can have multiple Checkout transactions over time, representing various deployments to users or locations. Each AssetCheckout belongs to a single Asset.

6. AssetCheckout and AssetCheckin (1:1 Relationship)

Each AssetCheckout transaction can have one corresponding AssetCheckin, representing the return of the same asset.

7. ComponentCategory and Component (1:M Relationship)

Each ComponentCategory can include multiple Components, but each Component belongs to a single ComponentCategory.

8. Component and ComponentCheckout (1:M Relationship)

A Component can be checked out multiple times for use with different assets. Each ComponentCheckout record belongs to one Component.

9. ComponentCheckout and ComponentCheckin (1:M Relationship)

Each ComponentCheckout transaction can have multiple ComponentCheckin records to track partial returns of components.

10. ComponentCheckout and Asset (M:1 Relationship)

Each ComponentCheckout links a Component to the Asset it is installed in, meaning many component checkouts can relate to a single asset.

11. Asset and Repair (1:M Relationship)

An Asset can undergo multiple Repair or Maintenance activities over time. Each Repair record belongs to one Asset.

12. Repair and RepairFile (1:M Relationship)

A Repair can have multiple RepairFile attachments documenting maintenance work or supporting evidence. Each RepairFile belongs to one Repair.

13. Asset and AuditSchedule (1:M Relationship)

An Asset can have several scheduled audits to verify its condition and existence.

Each AuditSchedule is linked to a single Asset.

14. AuditSchedule and Audit (1:1 Relationship)

Each AuditSchedule generates exactly one Audit upon execution, while each Audit corresponds to one schedule.

15. Audit and AuditFile (1:M Relationship)

An Audit can include multiple AuditFile attachments for documentation or verification evidence. Each AuditFile belongs to one Audit.

Contexts Service

The Contexts Service manages reference data for suppliers and manufacturers, serving as a shared source of truth for vendor and production details. Other microservices, such as the Assets Service, reference this data through API communication rather than direct database relationships.

Figure _ . Contexts Service

Manufacturer		Supplier	
id ↗	integer	id ↗	integer
name	varchar(50)	name	varchar(50)
manu_url	varchar(200)	address	varchar(100)
support_url	varchar(200)	city	varchar(50)
support_phone	varchar(13)	zip	varchar(4)
support_email	varchar(254)	contact_name	varchar(100)
notes	text	phone_number	varchar(13)
logo	varchar(100)	email	varchar(254)
is_deleted	boolean	URL	varchar(200)

Key Relationships in the Service

In the context of this database model, the primary entities include Supplier and Manufacturer. The following relationships describe how these entities interact conceptually within the system:

1. Supplier (Independent Entity)

The Supplier entity stores detailed information about asset vendors, including company name, address, contact details, website URL, and logo. Suppliers are independent records but can be referenced by assets in the Asset Service through their unique identifiers.

2. Manufacturer (Independent Entity)

The Manufacturer entity holds information about companies that produce assets, including URLs for general and support pages, contact details, and logos. Like suppliers, manufacturers are standalone entries that can be logically associated with assets in other services.

3. Supplier and Asset (Logical Relationship)

A Supplier can provide multiple Assets, but each Asset is linked to one Supplier. This connection is logical, as cross-database relationships are managed through service-level communication rather than database foreign keys.

4. Manufacturer and Asset (Logical Relationship)

A Manufacturer can produce multiple Assets, and each Asset corresponds to a single Manufacturer. This relationship is also logical, handled through inter-service API calls instead of direct relational constraints.

Physical Data Model (Schema)

Authentication Service

The database schema for the Auth Service is designed to manage user authentication, authorization, and administrative operations. It ensures data integrity, supports role-based access control, and integrates with Django's built-in authentication framework.

Table #

Authentication Service

Table Name	Columns	Description
auth_service_customuser	id (PK), email, password, first_name, last_name, middle_name, contact_number, image, role, is_active, is_staff, is_superuser, created_at, date_joined, last_login	Stores user profile information and login credentials. Each user has a role (e.g., Admin, User) with corresponding permissions.
auth_group	id (PK), name	Defines groups used for role-based access control. Users can be assigned to groups to inherit permissions.
auth_permission	id (PK), name, content_type_id (FK to django_content_type), codename	Contains permission details that determine access rights for models or views.
auth_group_permissions	id (PK), group_id (FK to auth_group), permission_id (FK to auth_permission)	Junction table linking groups to assigned permissions. Enables many-to-many relationship between groups and permissions.
auth_service_customuser_groups	id (PK), customuser_id (FK to auth_service_customuser), group_id (FK to auth_group)	Junction table linking users to their assigned groups. Supports inheritance of permissions from groups.

auth_service_customuser_permissions	id (PK), customuser_id (FK to auth_service_customuser), permission_id (FK to auth_permission)	Junction table linking users to their individual permissions outside group assignments.
django_session	session_key (PK), session_data, expire_date	Stores session data for authenticated users. Each record represents an active login session.
django_content_type	id (PK), app_label, model	Internal Django table mapping applications and models to their content types for permission management.
django_admin_log	id (PK), action_time, object_id, object_repr, action_flag, change_message, content_type_id (FK to django_content_type), user_id (FK to auth_service_customuser)	Records all actions performed in the Django Admin interface, including timestamps and affected objects.
django_migrations	id (PK), app, name, applied	Tracks migration history to maintain consistency between models and database structure.
django_rest_passwordreset_resetpasswordtoken	id (PK), created_at, key, ip_address, user_agent, user_id (FK to auth_service_customuser)	Handles password reset requests and temporary token management for users.

The Auth Service database schema manages all authentication and authorization processes in the system. It centers on a custom user model (auth_service_customuser) storing user information, credentials, and roles. Role-based access control is supported through tables like auth_group, auth_permission, and their junctions for user-group and permission assignments. Django system tables (django_session, django_content_type, django_admin_log, django_migrations) handle sessions, content-type mapping, admin activity logging, and migrations. The django_rest_passwordreset_resetpasswordtoken table manages password reset requests and tokens. Overall, the schema provides a structured, secure, and flexible foundation for authentication, access control, and administrative operations.

Assets Service

The database schema for the Asset Service is designed to manage all asset-related records and lifecycle operations within the organization. It provides structured data management for asset registration, categorization, movement, maintenance, depreciation, and auditing. The schema supports traceability, accountability, and integration with other microservices such as Contexts and Authentication.

Table #

Asset Service

Table Name	Columns	Descriptions
Depreciation	id (PK), name, duration, minimum_value, is_deleted	Defines depreciation methods applied to assets for accounting and financial purposes, including the duration and minimum value threshold.
AssetCategory	id (PK), name, type, logo, is_deleted	Contains categories used to classify assets (e.g., IT equipment, furniture). Helps organize and filter assets in the system.
Product	id (PK), name, category_id (FK to assetcategory), manufacturer_id, depreciation_id (FK to depreciation), model_number, end_of_life, default_purchase_cost, default_supplier_id, minimum_quantity, operating_system, notes, image, is_deleted	Stores predefined product templates used for registering new assets, including manufacturer, depreciation, and supplier references.
Status	id (PK), name, type, notes, is_deleted	Maintains standardized asset status labels such as <i>Deployable</i> , <i>Deployed</i> , <i>Pending</i> , <i>Under Repair</i> , or <i>Archived</i> .

Asset	id (PK), displayed_id, product_id (FK to product), status_id (FK to status), supplier_id, location, name, serial_number, warranty_expiration, order_number, purchase_date, purchase_cost, notes, image, is_deleted	Holds core asset information including identity, condition, supplier, and location. Each asset has a unique displayed_id generated upon creation.
AssetCheckout	id (PK), asset_id (FK to asset), to_user_id, to_location, checkout_date, return_date, condition, notes, confirmation_notes, image	Logs asset checkout transactions, including who checked out the asset, when, and under what condition.
AssetCheckin	id (PK), asset_checkout_id (OneToOne FK to assetcheckout), checkin_date, condition, notes, image	Records asset return details, including date, condition upon return, and any supporting documentation.
ComponentCategory	id (PK), name, type, logo, is_deleted	Defines classifications for components, allowing efficient tracking of replaceable or detachable parts associated with assets.
Component	id (PK), name, category_id (FK to componentcategory), manufacturer, supplier, location, model_number, order_number, purchase_date, purchase_cost, quantity, minimum_quantity, notes, image, is_deleted	Stores data about asset components or consumables, including supplier and manufacturer links, cost, and quantity tracking.
ComponentCheckout	id (PK), component_id (FK to component), to_asset_id (FK to asset), quantity, checkout_date, notes	Records the issuance of components to specific assets, allowing detailed tracking of component movements.

ComponentCheckin	id (PK), component_checkout_id (FK to componentcheckout), checkin_date, quantity, notes	Tracks the return or reinstallation of components previously checked out to an asset.
Repair	id (PK), asset_id (FK to asset), type, name, start_date, end_date, cost, notes, status_id (FK to status), is_deleted	Logs asset repairs, maintenance, or upgrades, including repair type, cost, and associated status.
RepairFile	id (PK), repair_id (FK to repair), file, is_deleted	Stores supporting documents or images related to a specific repair activity.
AuditSchedule	id (PK), asset_id (FK to asset), date, notes, is_deleted, created_at	Contains planned audit schedules for assets, ensuring regular verification of asset presence and condition.
Audit	id (PK), location, user_id, audit_date, next_audit_date, notes, is_deleted, audit_schedule_id (OneToOne FK to auditschedule), created_at	Logs completed audits, including responsible personnel, location, and findings.
AuditFile	id (PK), audit_id (FK to audit), file, is_deleted	Stores evidence or related documentation for completed audit records.

The Asset Service database schema manages the complete lifecycle of physical and digital assets within the organization. It integrates asset categorization, registration, movement tracking, repair logging, and audit management in a unified structure. Core entities such as Asset, Product, and Depreciation define asset properties and financial values, while operational entities like AssetCheckout, AssetCheckin, and Repair record day-to-day activities. Component and ComponentCategory tables support detailed management of asset parts and consumables. Finally, the Audit and AuditSchedule tables ensure accountability through regular verification of

asset status and documentation. Together, this schema provides a reliable and scalable foundation for asset tracking, maintenance, and compliance management across the enterprise.

Contexts Service

The database schema for the Contexts Service is designed to manage supporting reference data used across other services, such as supplier and manufacturer information. It provides structured and reusable context data that helps maintain data consistency and normalization throughout the system.

Table #

Contexts Service

Table Name	Columns	Descriptions
Supplier	id (PK), name, address, city, zip, contact_name, phone_number, email, URL, notes, logo, is_deleted	Stores information about suppliers, including contact details, address, and company logo. The is_deleted flag supports soft deletion to retain record history without removing data from the database.
Manufacturer	id (PK), name, manu_url, support_url, support_phone, support_email, notes, logo, is_deleted	Contains manufacturer information, including website and technical support contact details. The is_deleted flag enables soft deletion for data preservation.

The Contexts Service database schema manages all supporting and reference data used across the system. It is designed to provide standardized information for suppliers and manufacturers, ensuring consistency and reducing redundancy between interconnected services. The contexts_supplier table stores supplier details, including address, contact information, and company logo, supporting procurement and maintenance processes. The contexts_manufacturer table contains manufacturer information such as website links, technical support details, and contact references, enabling effective tracking of product sources and support channels. Overall, the schema offers a well-structured and reusable foundation for managing contextual information, promoting data integrity, and supporting seamless integration across all system services.

Data Dictionary

The Data Dictionary defines the tables, fields, and constraints used in each microservice. It provides an overview of the data structures, key attributes, and relationships to ensure consistency, integrity, and clarity across all services.

Authentication Service

Table #

Authentication Service

Table Name	Field Name	Data Type	Constraints
auth_group	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(150)	Unique, Not Null
auth_permission	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(255)	Not Null
	content_type_id	INTEGER	Not Null

	codename	VARCHAR(100)	Not Null
	(content_type_id, codename)		Unique Index
auth_group_permissions	id	SERIAL	Primary Key, Auto Increment
	group_id	INTEGER	Not Null
	permission_id	INTEGER	Not Null
	(group_id, permission_id)		Unique Index
auth_service_customuser	id	BIGSERIAL	Primary Key, Auto Increment
	password	VARCHAR(128)	Not Null
	last_login	TIMESTAMP	Not Null
	first_name	VARCHAR(50)	Not Null
	last_name	VARCHAR(50)	Not Null
	is_active	BOOLEAN	Not Null
	email	VARCHAR(254)	Unique, Not Null
	contact_number	VARCHAR(13)	Not Null
	created_at	TIMESTAMP	Not Null
	image	VARCHAR(100)	Nullable
	middle_name	VARCHAR(50)	Nullable
	role	VARCHAR(8)	Not Null
	is_staff	BOOLEAN	Not Null
	is_superuser	BOOLEAN	Not Null
	date_joined	TIMESTAMP	Not Null
auth_service_customuser_groups	id	SERIAL	Primary Key, Auto Increment
	customuser_id	BIGINT	Not Null
	group_id	INTEGER	Not Null
	(customuser_id, group_id)		Unique Index

auth_service_customuser_user_permissions	id	SERIAL	Primary Key, Auto Increment
	customuser_id	BIGINT	Not Null
	permission_id	INTEGER	Not Null
	(customuser_id, permission_id)		Unique Index
django_content_type	id	SERIAL	Primary Key, Auto Increment
	app_label	VARCHAR(100)	Not Null
	model	VARCHAR(100)	Not Null
	(app_label, model)		Unique Index
django_admin_log	id	SERIAL	Primary Key, Auto Increment
	action_time	TIMESTAMP	Not Null
	object_id	TEXT	Nullable
	object_repr	VARCHAR(200)	Not Null
	action_flag	SMALLINT	Not Null
	change_message	TEXT	Not Null
	content_type_id	INTEGER	Nullable
	user_id	BIGINT	Not Null
django_migrations	id	BIGSERIAL	Primary Key, Auto Increment
	app	VARCHAR(255)	Not Null
	name	VARCHAR(255)	Not Null
	applied	TIMESTAMP	Not Null
django_rest_passwordreset_resetpasswordtoken	id	SERIAL	Primary Key, Auto Increment
	created_at	TIMESTAMP	Not Null
	key	VARCHAR(64)	Unique, Not Null
	ip_address	INET	Nullable
	user_agent	VARCHAR(512)	Not Null
	user_id	BIGINT	Not Null

django_session	session_key	VARCHAR(40)	Primary Key
	session_data	TEXT	Not Null
	expire_date	TIMESTAMP	Not Null

Assets Service

Table #

Assets Service

Table Name	Field Name	Data Type	Constraints
Product	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(100)	Not Null
	category_id	INTEGER	Foreign Key → asset_category(id), On Delete CASCADE
	manufacturer_id	INTEGER	Positive integer
	depreciation_id	INTEGER	Foreign Key → depreciation(id), On Delete PROTECT
	model_number	VARCHAR(50)	Nullable
	end_of_life	DATE	Nullable
	default_purchase_cost	DECIMAL(10,2)	Nullable
	default_supplier_id	INTEGER	Nullable
	minimum_quantity	INTEGER	Default 1
operating_system		VARCHAR(7)	Nullable, Choices: linux, windows, macos, ubuntu, centos, debian, fedora, other
notes		TEXT	Nullable

	image	VARCHAR(100)	Nullable
	is_deleted	BOOLEAN	Default False
Asset	id	SERIAL	Primary Key, Auto Increment
	displayed_id	VARCHAR(20)	Unique, Auto-generated
	product_id	INTEGER	Foreign Key → product(id), On Delete CASCADE
	status_id	INTEGER	Foreign Key → status(id), On Delete CASCADE
	supplier_id	INTEGER	Not Null
	location	INTEGER	Not Null
	name	VARCHAR(100)	Not Null
	serial_number	VARCHAR(50)	Nullable
	warranty_expiration	DATE	Nullable
	order_number	VARCHAR(50)	Nullable
	purchase_date	DATE	Nullable
	purchase_cost	DECIMAL(10,2)	Nullable
	notes	TEXT	Nullable
	image	VARCHAR(100)	Nullable
	is_deleted	BOOLEAN	Default False
AssetCheckout	id	SERIAL	Primary Key, Auto Increment
	asset_id	INTEGER	Foreign Key → asset(id), On Delete CASCADE
	to_user_id	INTEGER	Not Null
	to_location	VARCHAR(255)	Not Null
	checkout_date	TIMESTAMP	Auto Now Add
	return_date	TIMESTAMP	Not Null
	condition	SMALLINT	Not Null, 1–10

	notes	TEXT	Nullable
	confirmation_notes	TEXT	Nullable
	image	VARCHAR(100)	Nullable
AssetCheckin	id	SERIAL	Primary Key, Auto Increment
	asset_checkout_id	INTEGER	One-to-One → asset_checkout(id)
	checkin_date	TIMESTAMP	Not Null
	condition	SMALLINT	Not Null, 1–10
	notes	TEXT	Nullable
	image	VARCHAR(100)	Nullable
ComponentCategory	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(50)	Not Null
	type	VARCHAR(5)	Default "Component"
	logo	VARCHAR(100)	Nullable
	is_deleted	BOOLEAN	Default False
Component	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(100)	Not Null
	category_id	INTEGER	Foreign Key → component_category(id), On Delete CASCADE
	manufacturer	INTEGER	Foreign Key → status(id), On Delete CASCADE
	supplier	INTEGER	Not Null
	location	INTEGER	Not Null
	model_number	VARCHAR(50)	Nullable
	order_number	VARCHAR(30)	Nullable
	purchase_date	DATE	Auto Now Add
	purchase_cost	DECIMAL(10,2)	Not Null

	quantity	INTEGER	Default 1
	minimum_quantity	INTEGER	Default 0
	notes	TEXT	Nullable
	image	VARCHAR(100)	Nullable
	is_deleted	BOOLEAN	Default False
ComponentCheckout	id	SERIAL	Primary Key, Auto Increment
	component_id	INTEGER	Foreign Key → component(id)
	to_asset_id	INTEGER	Foreign Key → asset(id)
	quantity	INTEGER	Default 1
	checkout_date	TIMESTAMP	Auto Now Add
	notes	TEXT	Nullable
ComponentCheckin	id	SERIAL	Primary Key, Auto Increment
	component_checkout_id	INTEGER	Foreign Key → component_checkout(id)
	checkin_date	TIMESTAMP	Auto Now Add
	quantity	INTEGER	Default 1
	notes	TEXT	Nullable
Repair	id	SERIAL	Primary Key, Auto Increment
	asset_id	INTEGER	Foreign Key → asset(id)
	type	VARCHAR(20)	Not Null, Choices: maintenance, repair, upgrade, test, hardware, software
	name	VARCHAR(100)	Not Null
	start_date	DATE	Default Now
	end_date	DATE	Nullable
	cost	DECIMAL(10,2)	Not Null
	notes	TEXT	Nullable

	status_id	INTEGER	Foreign Key → status(id)
	is_deleted	BOOLEAN	Default False
RepairFile	id	SERIAL	Primary Key, Auto Increment
	repair_id	INTEGER	Foreign Key → repair(id)
	file	VARCHAR(100)	Not Null
	is_deleted	BOOLEAN	Default False
AuditSchedule	id	SERIAL	Primary Key, Auto Increment
	asset_id	INTEGER	Foreign Key → asset(id)
	date	DATE	Not Null
	notes	TEXT	Nullable
	is_deleted	BOOLEAN	Default False
	created_at	TIMESTAMP	Default Now, Not Editable
Audit	id	SERIAL	Primary Key, Auto Increment
	location	VARCHAR(50)	Not Null
	user_id	INTEGER	Not Null
	audit_date	DATE	Default Now
	next_audit_date	DATE	Default Now
	notes	TEXT	Nullable
	is_deleted	BOOLEAN	Default False
	audit_schedule_id	INTEGER	One-to-One → audit_schedule(id)
	created_at	TIMESTAMP	Default Now, Not Editable
AuditFile	id	SERIAL	Primary Key, Auto Increment
	audit_id	INTEGER	Foreign Key → audit(id)
	file	VARCHAR(100)	Not Null
	is_deleted	BOOLEAN	Default False

Contexts Services

Table Name	Field Name	Data Type	Constraints
Supplier	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(50)	Not Null
	address	VARCHAR(100)	Nullable
	city	VARCHAR(50)	Nullable
	zip	VARCHAR(4)	Nullable
	contact_name	VARCHAR(100)	Nullable
	phone_number	VARCHAR(13)	Nullable
	email	VARCHAR(254)	Nullable
	URL	VARCHAR(200)	Nullable
	notes	TEXT	Nullable
	logo	VARCHAR(100)	Nullable
	is_deleted	BOOLEAN	Default False
Manufacturer	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(50)	Not Null
	manu_url	VARCHAR(200)	Nullable
	support_url	VARCHAR(200)	Nullable
	support_phone	VARCHAR(13)	Nullable
	support_email	VARCHAR(254)	Nullable
	notes	TEXT	Nullable
	logo	VARCHAR(100)	Nullable

	is_deleted	BOOLEAN	Default False
Depreciation	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(500)	Not Null
	duration	INTEGER	Not Null
	minimum_value	DECIMAL(8,2)	Not Null
	is_deleted	BOOLEAN	Default False
AssetCategory	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(50)	Not Null
	type	VARCHAR(5)	Default "Asset"
	logo	VARCHAR(100)	Nullable
	is_deleted	BOOLEAN	Default False
Status	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(50)	Not Null
	type	VARCHAR(12)	Not Null, Choices: deployable, deployed, undeployable, pending, archived
	notes	TEXT	Nullable
	is_deleted	BOOLEAN	Default False

Data Flow Diagrams (DFD) (Level 0, Level 1)

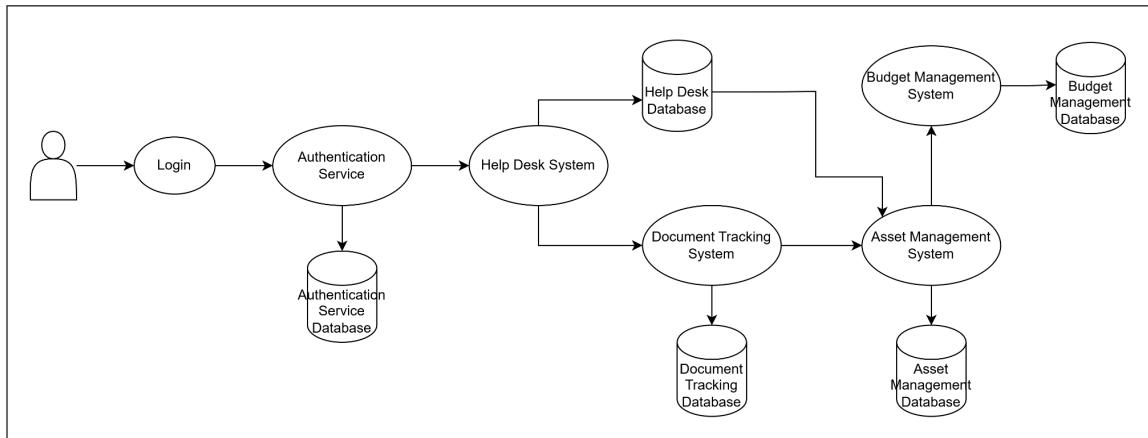
The Data Flow Diagrams illustrate how data moves between the system's processes, data stores, and external entities.

a. Level 0 DFD

The Level 0 Data Flow Diagram illustrates the overall interaction between the Asset Management System (AMS), its associated cloud databases, and external systems such as the Help Desk System, Document Tracking System, Budget

Management System, and the Authentication Service. The AMS serves as the central process, handling asset-related requests, updates, and approvals while ensuring that all communications between systems are securely authorized.

Figure #. Contexts Service

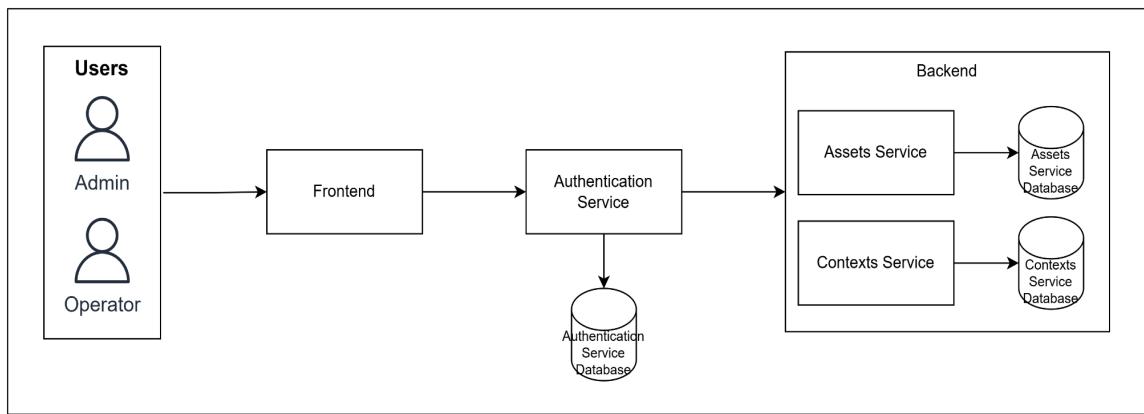


The Level 0 Data Flow Diagram shows how the main systems work together, including the Authentication Service, Help Desk System, Asset Management System, Budget Management System, and Document Tracking System. The process starts when the user logs in through the Authentication Service, which checks their credentials and allows access to the connected systems. After authentication, the user can perform tasks such as submitting and tracking help desk tickets, managing assets, handling budgets, or processing documents based on their permissions. Each system has its own database but exchanges important information with the others through the Authentication Service to ensure data consistency and secure communication across all modules.

b. Level 1 DFD

The Level 1 Data Flow Diagram provides a more detailed view of the internal processes within the Asset Management System (AMS). It breaks down the main process from the Level 0 DFD into smaller sub-processes, showing how data moves between users, the frontend, and backend services.

Figure #. Contexts Service



The Level 1 Data Flow Diagram shows how users such as the Admin and Operator interact with the system through the Frontend. The Frontend connects to the Authentication Service, which verifies user credentials using the Authentication Service Database. Once authenticated, users can access the Backend, where the Assets Service and Contexts Service perform core functions. The Assets Service manages asset-related operations and stores data in the Assets Service Database, while the Contexts Service handles supporting records such as suppliers and manufacturers, saving them in the Contexts Service Database.

Requirements Engineering

Table #

Technology Stack Justification

Technology/Component	Purpose/Justification	Key Features/Benefits
React (Frontend)	Builds the user interface, making it easy to use and interactive.	Uses a component-based structure for easier updates and maintenance. Updates the interface quickly without reloading the whole page. Works well on different devices.
Django (Backend)	Manages data and handles system operations securely and efficiently.	Has a clear structure (Model-View-Template) for organization. Includes tools for user login and data management. Protects against common security issues. Provides APIs for the frontend. Supports secure communication between client and server.
PostgreSQL (Database)	Stores and manages system data reliably and safely.	Keeps data accurate and consistent. Handles complex searches and queries. Free and cost-effective. Works well with large amounts of data.
Overall Stack	Combines React, Django, and PostgreSQL to create a strong and reliable system.	Easy to use, fast, and secure. Handles monitoring and reporting of assets. Adapts to future organizational needs.

Deployment Architecture

Figure __. Deployment Architecture

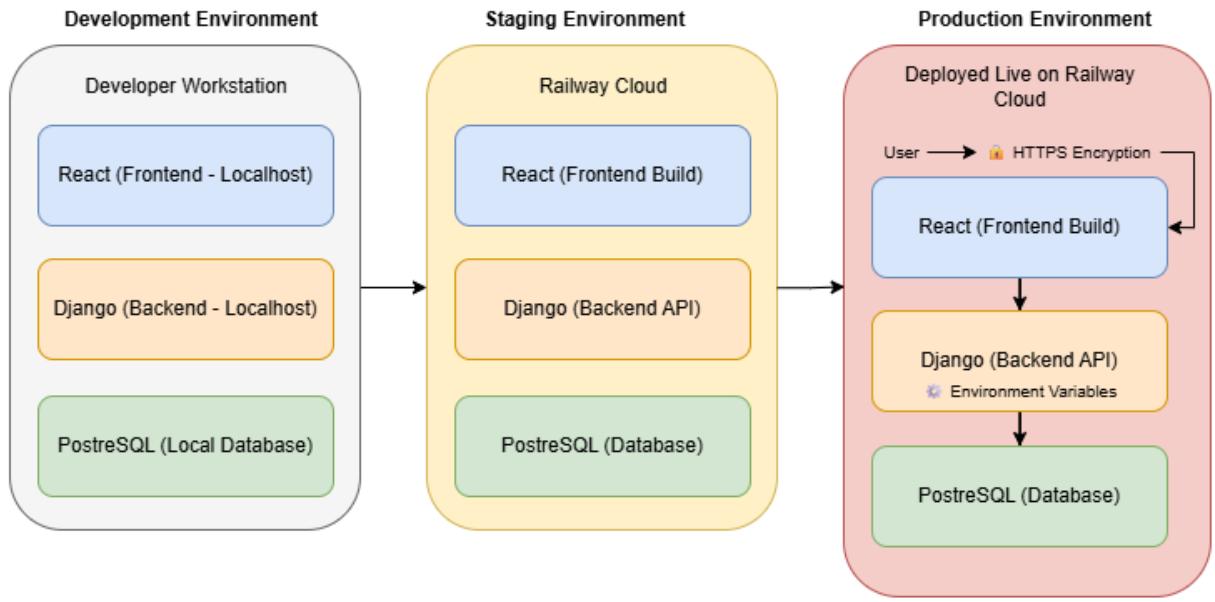


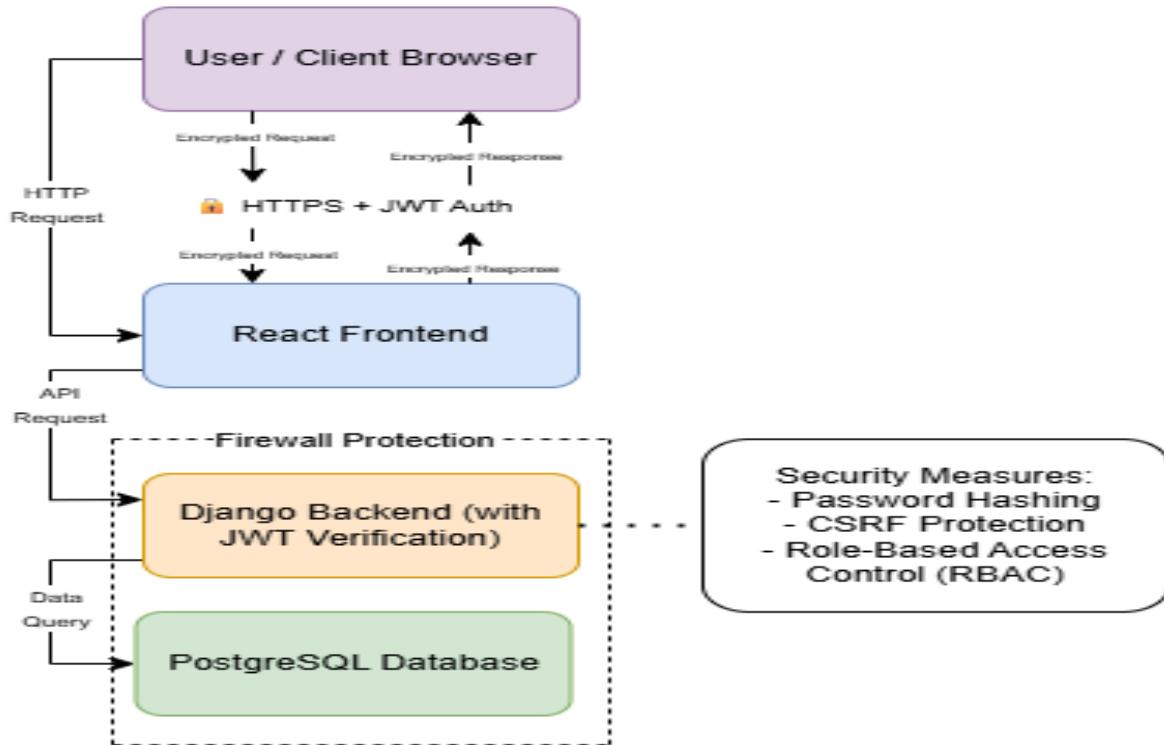
Figure shows the deployment architecture of the MAP-AMS, which is divided into three main environments: Development, Staging, and Production. This structured deployment approach ensures a systematic and controlled process for building, testing, and releasing updates while maintaining performance, security, and consistency across all stages. Each environment plays a specific role in preparing the system for live deployment and ensuring that all modules perform as intended. The Development Environment focuses on creating and testing features, the Staging Environment validates integration and functionality, and the Production Environment serves as the live platform accessible to users. This layered setup helps minimize deployment errors and ensures the system is stable before public use. It also allows the development team to monitor and refine each stage of deployment effectively. Overall, this architecture ensures that MAP-AMS delivers a secure, efficient, and reliable asset management experience for organizations.

In the Development Environment, developers work locally to build and test individual components of MAP-AMS. The React frontend is executed using Node.js, allowing developers to test user interfaces for functions such as asset registration, asset repair, and asset audit visualization. The Django backend runs on a local server to process requests and manage communication with a PostgreSQL database configured locally. This setup allows developers to test and validate system features in a safe environment before integration. Version control through Git and GitHub ensures proper synchronization of updates and collaboration among team members. Once individual components are verified, they are deployed to the Staging Environment hosted on Railway Cloud for full integration and system testing. This transition ensures that the system's behavior under cloud infrastructure is consistent and reliable before going live.

The Staging and Production Environments represent the final phases of deployment where the complete system is tested and then released to users. In the Staging Environment, the React frontend, Django backend, and PostgreSQL database are deployed on Railway Cloud to simulate the production setup. This environment allows comprehensive testing, including functionality, performance, and User Acceptance Testing (UAT), ensuring that modules such as asset tracking, depreciation, and audit reporting operate correctly. Once validated, the system is promoted to the Production Environment, where it becomes fully accessible to users. In this environment, the React frontend is deployed as a static web build served securely over HTTPS, while the Django backend and PostgreSQL database run as managed services. All sensitive credentials, including database keys and JWT secrets, are managed using environment variables to maintain security. This cloud-based architecture ensures scalability, availability, and data protection, providing users with a stable and secure experience when accessing MAP-AMS online.

Network and Security Design

Figure #. Network and Security



The Figure shows the network and security design of the MAP-AMS, which emphasizes the protection of data integrity, confidentiality, and availability throughout all communication layers. The system architecture is designed to ensure that every data transaction between the client, frontend, backend, and database is securely transmitted and verified. It utilizes Railway Cloud's built-in network security mechanisms, including firewall configurations and virtual network isolation, to prevent unauthorized access to backend and database services. The layered structure separates user interaction, business logic, and data management components, minimizing exposure to potential attacks. This design ensures that external users can only communicate through the React frontend, while direct access to the Django backend and PostgreSQL database is restricted. By isolating network traffic and enforcing controlled

entry points, MAP-AMS maintains a secure and resilient infrastructure. This comprehensive design contributes to the system's ability to deliver asset management functionalities safely and efficiently.

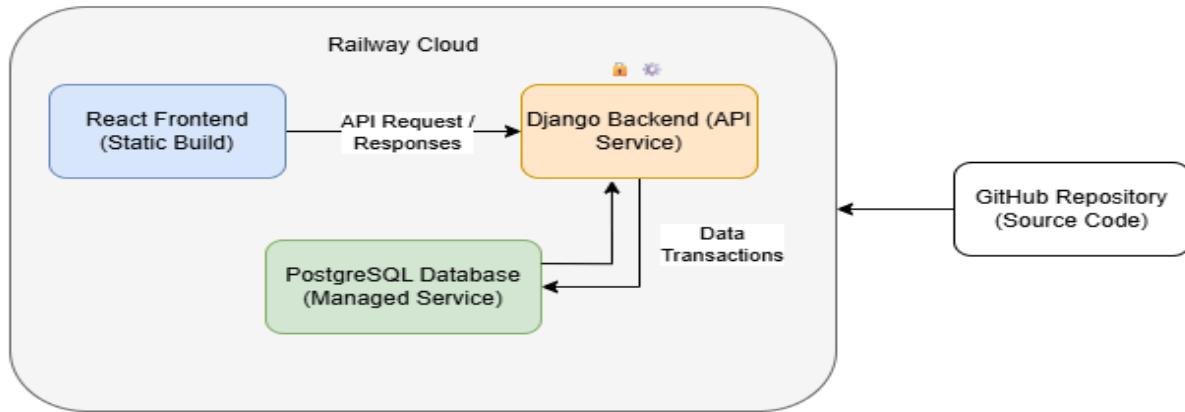
The system employs HTTPS encryption and JSON Web Token (JWT) authentication to secure all client-server communications. When a user accesses MAP-AMS through a web browser, requests are encrypted using HTTPS, protecting data from interception and tampering during transmission. Upon successful authentication, the system generates a JWT containing encrypted credentials, which is stored on the client side. This token is attached to each subsequent API request, allowing the Django backend to verify user identity before granting access to protected resources. Because JWT authentication is stateless, it enhances scalability and reduces server-side session management overhead. This method ensures that only authorized users can access confidential modules such as asset records, supplier details, and audit reports. By combining encryption and token-based validation, the system maintains secure and authenticated interactions across all components.

Additional security mechanisms are integrated within the backend and database layers to reinforce protection against threats and unauthorized actions. The Django backend implements password hashing, Cross-Site Request Forgery (CSRF) protection, and Role-Based Access Control (RBAC) to manage user privileges effectively. These measures ensure that users can only perform operations according to their designated roles, reducing the risk of privilege escalation or data manipulation. The PostgreSQL database further enforces access control by accepting queries only from authenticated backend connections, preventing direct external access. Moreover, Railway Cloud's infrastructure provides firewall protection that isolates database traffic and blocks malicious network activity. Collectively, these layered defenses maintain a high level of data confidentiality and operational reliability. Using this

network and security design, MAP-AMS ensures that all asset management transactions are handled in a secure, authenticated, and controlled environment.

Cloud Infrastructure

Figure #. Cloud Infrastructure Diagram



The cloud infrastructure design of the MAP-AMS illustrates how the system is fully hosted on Railway Cloud, which provides managed hosting, deployment, and database services for both the frontend and backend components. This cloud platform ensures seamless deployment, scalability, and continuous availability for end-users accessing the system. The Django backend operates as a managed API service within Railway's containerized environment, allowing automatic rebuilds and restarts whenever code updates are pushed from the connected GitHub repository. The PostgreSQL database is also managed by Railway Cloud, ensuring secure data transactions, SSL encryption, automated backups, and high availability. Meanwhile, the React frontend is deployed as a static build within the same cloud environment and delivered over HTTPS for optimized speed and security. This fully managed infrastructure provides a stable foundation for the asset management system, minimizing manual configuration and system downtime.

The system's deployment workflow follows a Continuous Integration and Continuous Deployment (CI/CD) model integrated with GitHub, ensuring that development and production remain synchronized. Every new commit or update to the source code triggers an automated build and deployment process within Railway Cloud. This automation minimizes human error and ensures that the latest stable version of the system is consistently deployed. The CI/CD pipeline also performs validation checks and dependency installations to maintain version control integrity. Also, Developers can focus on iterative improvements and feature enhancements rather than manual server maintenance.

Furthermore, Railway Cloud provides built-in system monitoring tools that track application performance, resource utilization, and service uptime. These monitoring capabilities enable proactive detection of performance issues and facilitate prompt troubleshooting. The platform's firewall protection and network isolation ensure that backend and database services remain secure from unauthorized access. Moreover, Railway's scalable hosting options allow MAP-AMS to dynamically adjust resources based on usage demand, ensuring both cost efficiency and operational flexibility. Together, these features establish a secure, reliable, and easily maintainable environment for hosting the MAP-AMS application. Overall, the cloud infrastructure ensures operational continuity and long-term sustainability for asset management operations.

Development Journey

Sprint Summaries

The main goal of Sprint 1 was to finalize and complete all user interface revisions across the system to ensure a visually consistent and user-friendly design. This phase primarily focused on improving the overall appearance and usability of each module, including the Asset Model or Products, Assets, Asset Repair, Asset Audit, Reports, Context, and Tickets. The team

aimed to create a cohesive structure that aligned with the project's overall functionality and design standards. During this sprint, the developers also reviewed the layout flow and ensured that each module was properly connected to its intended process for future backend integration. Since the emerging technology component of the system required both UI and functionality development, it was intentionally postponed to Sprint 3 to allow the team to concentrate on completing all revisions without compromise. The sprint also stressed teamwork to synchronize front-end outputs and minimize duplicating design efforts. Towards the end of this phase, the team had created a uniform and well-structured interface across all modules, laying the groundwork for the next sprint's development work.

Several issues arose during Sprint 1 that had an impact on workflow and scheduling. Some of the most prevalent challenges included design inconsistencies between modules, initial setup issues, and delays in completing the specified timetable. These obstacles were met through effective collaboration among members, continuous communication, and the leadership of team leaders and mentors. The team held regular meetings to monitor progress, modify priorities, and identify areas that needed to be reworked or refined. Timeline modifications were made to address these concerns while maintaining the overall quality of the UI. Despite small setbacks, the team displayed accountability and teamwork by finishing all tasks within the amended timetable. Sprint 1 ended well, with all modules polished and ready for backend integration in the next sprint.

The primary objective of Sprint 2 was to re-implement the backend to align with the revised user interface completed in the previous sprint. This phase emphasized connecting the backend logic with the redesigned front-end modules to ensure full system functionality and data consistency. The team focused on integrating all major modules, including assets, repairs, audits, and reports, while also developing system interconnections that support communication between different components. Another critical goal was to implement the new authentication

system to improve security and user management, although this task experienced some delays due to dependencies from earlier revisions. During this sprint, the team made significant improvements in backend integration, ensuring that each module could process data effectively and display up-to-date information on the user interface. To ensure system reliability, developers responsible for API endpoints, database migrations, and testing routines have to work closely together. Despite small failures, the sprint resulted in a fully operating system with essential features successfully linked and tested for basic functionality.

Additionally, the team encountered numerous technical issues during sprint 2 that necessitated urgent attention and cautious planning. Common challenges included trouble establishing APIs, handling database migrations, and resolving testing errors that occurred during integration. To solve these concerns, the team held collaborative debugging sessions, adhered to correct planning rules, and made swift changes to their process when unforeseen issues occurred. When a problem took longer to resolve and threatened to stall progress, the team provided alternate solutions and temporary workarounds to ensure that development proceeded smoothly. Furthermore, the sprint was extended by one week beyond its planned deadline to accommodate necessary changes and retesting, allowing the team to complete all critical backend functionality. The team met its targets while preserving quality and system performance thanks to good communication and problem-solving skills. After the end of Sprint 2, the project had reached a stable state, laying the groundwork for the following sprint, which would focus on final testing, emerging technology integration, and system optimization.

Key Technical Challenges and Solutions

This section discusses the primary technical challenges encountered during the development of the system and the corresponding solutions implemented to address them. Throughout the development process, the team faced various issues related to data

consistency, inter-service communication, deployment, frontend integration, and user interface reliability. These challenges were expected given the project's microservices-based architecture and the integration of Django (backend) with React (frontend). The subsections below present the key challenges identified in both backend and frontend development, along with the practical solutions adopted to ensure the system's stability, scalability, and usability.

Backend and System Architecture Challenges

This subsection discusses the major backend challenges encountered during the development of the system, focusing on the Django microservices, database management, and containerized deployment setup. The backend architecture aimed to achieve modularity, scalability, and maintainability, but several technical complexities emerged during integration and deployment.

A. Data Consistency Across Microservices

Challenge: The system was designed using multiple Django-based microservices, including the Authentication Service, Assets Service, and Contexts Service. Each service maintained its own database, which introduced challenges in preserving data consistency, particularly when operations required coordination across multiple services. For instance, when an employee checked out an asset, both the status of the asset record in the Assets Service and the corresponding employee record in the User Service needed to be updated simultaneously. Without proper synchronization, discrepancies could arise, such as assets appearing checked out in one service but not reflected in another.

Solution: To address this challenge, the team implemented a reliable data synchronization approach through RESTful API communication between services. Each microservice exposed well-defined API endpoints that could be called by other services

whenever a shared operation occurred. For example, when an asset was checked out, the Asset Service sent a request to update the corresponding employee record in the User Service, ensuring that both databases reflected the same transaction. To maintain data integrity during multi-step operations, the team applied a simplified version of the Saga pattern. This approach ensured that if one operation failed, compensating actions were executed to revert previous steps, preventing partial or inconsistent updates. Additionally, each API request included validation and status checks to confirm successful completion before committing data changes. Through these measures, the system maintained consistent and synchronized data across all microservices while reducing the risk of transactional errors.

B. Inter-Service Communication

Challenge: Establishing reliable communication between independent microservices was another significant challenge during backend development. Since each Django service operated as a standalone application with its own runtime and database, coordinating operations between them required secure and standardized communication. Early in development, the team encountered issues such as endpoint mismatches, inconsistent request formats, and authentication errors. These problems caused delays in data exchange between services like Authentication, Assets, and Contexts, which affected overall system integration.

Solution: The team standardized inter-service communication using RESTful APIs with JSON-based payloads to ensure consistent data exchange. Each service was designed with clear and documented endpoint structures to minimize confusion and errors during integration. To enhance security, JWT (JSON Web Token) authentication was implemented for all inter-service requests, allowing services to verify the authenticity of

incoming requests before processing them. This approach ensured that only authorized services could communicate with one another. Consistent endpoint conventions, standardized response formats, and centralized API documentation improved reliability and made service interactions predictable and easier to maintain.

C. Containerization and Deployment

Challenge: Managing multiple Django microservices, along with a PostgreSQL database, presented deployment and environment consistency challenges. Differences in local configurations, package versions, and dependencies often led to runtime errors when deploying or testing the system on different machines. The manual setup process for each service was time-consuming and prone to human error, which made development and testing less efficient.

Solution: To overcome deployment and environment consistency issues, the team adopted Docker and Docker Compose for containerization and orchestration. Each Django microservice was containerized with its own dependencies, ensuring that the application would run identically across different environments. Docker Compose was configured to manage multiple containers simultaneously, including the PostgreSQL database, allowing all services to start and communicate with a single command. Environment variables were stored in a dedicated .env file to separate configuration details, such as database credentials and API keys, from the source code. This improved security and made configuration management more flexible. By using Docker, the team eliminated version conflicts and simplified the process of setting up the development environment. The containerized setup also made it easier to scale individual services in the future and prepared the system for potential cloud deployment.

Frontend and Integration Challenges

This subsection discusses the key challenges encountered in the frontend development and integration process. The main issues involved connecting the React and Vite application with multiple Django microservices, maintaining design consistency, and ensuring proper display across different screen sizes. These challenges affected data handling, user interface behavior, and overall usability. The following parts explain how the development team addressed these challenges to improve performance, reliability, and user experience.

A. API Integration and Data Handling

Challenge: Integrating the React + Vite frontend with multiple Django microservices initially led to issues with data synchronization and authentication. Since the system relied on various APIs for assets, authentication, and contexts, asynchronous requests sometimes caused components to render before complete data retrieval. In addition, JWT tokens were not consistently attached to requests, leading to failed authentication and restricted access to protected endpoints.

Solution: A centralized API service was implemented to manage all communication between the React + Vite frontend and the backend microservices. Using Axios as the primary HTTP client, the team configured interceptors to automatically include JWT tokens in all outgoing requests. This ensured secure and consistent authentication across the system. To enhance data reliability and performance, React Query was adopted for fetching, caching, and synchronizing server data. It helped prevent redundant requests and allowed real-time updates when backend data changed. Loading indicators and conditional rendering were also introduced to ensure components only appeared once valid data was fully retrieved, improving the overall user experience.

B. User Interface Consistency

Challenge: The frontend system faced challenges in maintaining a consistent appearance across different modules of the application. Several pages had variations in colors, layouts, and typography, which made the system look uneven and less professional. These inconsistencies affected the overall user experience and made navigation confusing for users. The issue occurred mainly because developers designed components independently without a shared reference or unified style guide, leading to repetitive and mismatched designs in different parts of the system.

Solution: The team addressed this problem by developing a shared component library that followed a single and well-defined design standard. This library included reusable elements such as buttons, forms, tables, navigation bars, and dialog boxes. Each component was designed to maintain the same color scheme, spacing, and layout structure throughout the application. Implementing these reusable components not only ensured visual consistency but also simplified future updates and maintenance. The unified design helped create a cleaner interface, improved navigation flow, and provided users with a more seamless and cohesive experience across all pages of the React and Vite application.

C. Responsiveness

Challenge: Some parts of the frontend did not properly adjust to different screen sizes, which caused layout distortion and uneven alignment when viewed on tablets or mobile devices. Certain elements, such as tables and buttons, appeared too large or too small, making the system difficult to use on smaller screens. This lack of adaptability reduced the system's accessibility and negatively affected the user experience, especially for users accessing the system through mobile devices.

Solution: The team enhanced responsiveness by implementing flexible grid layouts and using reusable components that could automatically adjust to different screen sizes. Media queries and responsive design principles were applied to ensure that all interface elements, including forms, buttons, and tables, displayed correctly on both desktop and mobile devices. The use of adaptive containers and dynamic resizing improved readability and usability across platforms. These improvements made the system more accessible, user-friendly, and visually balanced regardless of the device being used.

Implementation Strategy

The Implementation Strategy for the MAP Active IT Asset Management System with Forecasting (MapAms) explains how the system will be deployed, migrated, and adopted within MAP Active Philippines. It ensures that the transition from manual IT asset tracking to a digital and secure management process is properly planned and executed. The implementation focuses on three key components: the Deployment Plan, Migration Plan, and User Training and Change Management.

Deployment Plan

The deployment of the MAP Asset Management System will use a blue-green deployment setup to ensure continuous system availability while minimizing downtime. In this setup, two identical environments, called Blue and Green, are maintained. The Blue environment represents the current live version of the system that users access, while the Green environment contains the new version prepared for deployment. Users continue working in the Blue environment while the Green version is tested and validated for stability and performance. Once the Green version is confirmed to be working properly, user access is switched from Blue to Green, making the Green version live. The Blue environment remains

available as a backup copy, allowing the team to quickly revert if issues occur. This setup ensures a seamless transition between versions and prevents major service interruptions.

Within each deployment environment, a phased rollout approach will be applied. This approach means that the system will be released in stages rather than all at once. Each phase focuses on a specific module or group of users, allowing the development team to test, evaluate, and resolve issues before moving to the next phase. The goal is to minimize risks, reduce disruption, and allow users to gradually adapt to the new system.

The MAP-AMS consists of two major modules: the Contexts Module and the Assets Module. The Contexts Module will be deployed first because it contains the foundational data needed by the entire system. This module manages categories, suppliers, manufacturers, depreciation, and status records that serve as references for all IT asset operations. After the successful deployment of the Contexts Module, the Assets Module will be implemented. This module handles IT asset operations such as registration, check-in and check-out, repairs, and audits. During this phase, the system will also be connected to other organizational systems:

- a. The Help Desk System, which sends ticket requests for IT asset check-in and check-out.
- b. The Document Tracking System (DTS), which processes the approval or denial of requests and returns the result to the Asset Management System.
- c. The Budget Management System, which the Asset Management System provides cost information to.

All these systems share a centralized Authentication Service, which manages user accounts, credentials, and access control across all systems. This shared authentication ensures secure login, consistent user data, and proper role-based access management throughout the organization.

Additionally, each phase of the rollout will include validation checkpoints, system and integration testing, and user acceptance testing to confirm readiness. After every deployment, system monitoring will be carried out using Docker container logs and performance metrics to identify and resolve any technical issues immediately, ensuring that the system remains stable and reliable.

Migration Plan

The Migration Plan ensures that existing IT asset data from spreadsheets and records is accurately and securely transferred into the MAP-AMS database. This plan focuses on maintaining data consistency and eliminating errors that commonly occur in manual tracking systems. The process is divided into four major steps.

1. Data Profiling and Cleansing

This step involves reviewing all existing data to remove duplicate entries and correct inconsistencies that may affect the accuracy of the new system. It also includes standardizing data formats for asset names, serial numbers, supplier details, purchase dates, and depreciation values. Through this process, only verified and reliable data will be used during the migration to ensure the quality and integrity of the system's database.

2. Data Structuring

After cleaning, the data will be organized into structured templates that match the MAP-AMS database schema. This ensures that all records conform to the system's required format and relationships before being imported. Each field will be checked to follow the correct data type, constraints, and relational dependencies so that data integrity is preserved once the importation process begins.

3. Data Importation

The cleaned and structured data will then be uploaded into the system using a data import utility. The migration will begin with context data such as categories, suppliers, manufacturers, and depreciation methods since these serve as the foundation for the entire system. Once the context data is successfully migrated, the IT asset records will be imported and linked to their corresponding references. Referential integrity checks will be performed to ensure that all assets are properly connected to their related context information.

4. Verification and Validation

The final step of migration focuses on ensuring that all imported data is accurate and complete. Small-scale testing will first be conducted to confirm that the import tools and formats are functioning correctly. After the full migration, a final cross-check will be performed between the system's database and the original records to verify completeness and correctness. Backups will also be created before and after each stage to prevent data loss and allow quick restoration if any issue arises.

User Training and Change Management

The User Training and Change Management plan ensures that all personnel involved in IT asset management are equipped with the knowledge and skills to use the new system effectively. This stage focuses on preparing employees for the transition from manual processes to an integrated, digital platform and ensuring that the adoption of the MAP Active Asset Management System is smooth and sustainable.

1. Orientation Sessions

The training phase will begin with orientation sessions to introduce the overall purpose, scope, and benefits of the MAP Active Asset Management System. These

sessions will explain how the system enhances existing workflows, supports accurate record-keeping, and integrates with other organizational systems such as the Help Desk, Document Tracking, and Budget Management Systems. The goal is to build awareness and understanding of how the system contributes to improved operational efficiency and transparency.

2. Hands-on Training

After orientation, hands-on training sessions will be conducted to familiarize users with the system's core functionalities. Participants will learn how to perform key operations such as registering IT assets, managing context data, checking assets in and out, recording repairs, and generating reports. These sessions will also simulate real-world workflows to help users apply what they learn to their daily tasks and gain confidence in navigating the system.

3. Training Materials

To support continued learning, comprehensive training materials will be provided. These will include user manuals, visual guides, and short video tutorials that illustrate system procedures in detail. The materials will serve as ongoing references for employees to review after the formal training sessions, ensuring consistent understanding and proper system use across all departments.

4. Train-the-Trainer Approach

A train-the-trainer strategy will be implemented to establish in-house expertise within MAP Active Philippines. Selected IT personnel will receive advanced training that enables them to serve as internal trainers and support staff after deployment. This

approach reduces dependency on the development team and ensures that knowledge sharing continues even after the initial implementation phase.

5. Change Management and Feedback

The change management component will focus on guiding users through the transition and encouraging acceptance of the new system. Communication activities will emphasize the benefits of automation, such as reduced manual workload, faster processing, and greater accuracy in IT asset tracking. Feedback sessions and surveys will be conducted to gather user experiences, identify potential issues, and determine where additional support may be needed. Regular follow-up evaluations will also be performed to measure user satisfaction and ensure ongoing improvement in system utilization.

Test Plan

The testing phase for the Asset Management System (AMS) was conducted using an Agile Sprint-based approach to ensure iterative validation of core functionalities. Testing was organized into three sprints: Sprint 1 focused on module and UI-based validation, Sprint 2 emphasized backend integration to UI, and Sprint 3 targeted advanced features such as dashboard analytics, real-time notifications, and QR code generation for asset tracking. The test plan was structured to verify Role-Based Access Control (RBAC), data integrity, user interface consistency, input validation, navigation accuracy, and system responsiveness.

Objectives

The objectives of the test plan were to ensure accurate management of IT assets across their lifecycle, including registration, update, deletion, checkout/check-in, and status tracking. It also aimed to validate UI components such as tables, filters, modals, breadcrumbs, buttons, and

footers. Additionally, the plan confirmed backend integration for data persistence, duplicate detection, alert mechanisms, and dependency handling, such as preventing deletion of categories linked to assets. Comprehensive functional coverage was targeted, with emphasis on edge cases, validation rules, and real-time data accuracy.

Scope

The scope included functional testing of all core modules such as Categories, Suppliers, Manufacturers, Statuses, Asset Models, Assets, Components, Accessories, and Consumables, along with UI validation, backend logic such as import/export and alerts, and advanced features like dashboard cards, forecasting, and QR generation. Load testing, security penetration testing beyond RBAC, and mobile responsiveness were excluded from the scope, though future mobile support is recommended.

Test Types

Test types consisted of Sprint 1 (Module & UI Based), which covered UI layout, navigation, button functionality, form fields, pagination, scrolling, and footer presence. Sprint 2 (Backend Integration to UI) focused on data validation, duplicate checks, import/export behavior, success/danger alerts, linked entity constraints, and routing. Sprint 3 addressed real-time dashboard metrics such as Due for Return and Low Stock, modal content accuracy, forecasting, and QR code generation/redirection.

Test Environment

The test environment was a web-based interface tested on modern browsers, with pre-configured test data including IT assets such as laptops, monitors, and peripherals, categories like Hardware and Software, suppliers, and sample checkout histories. Manual test execution was performed with detailed logging in Excel, capturing steps, input, expected result, actual result, status, and comments.

Test Schedule

The test schedule included multiple execution cycles in Sprint 1 with defect re-testing, such as asset checkout/check-in and category/supplier updates. Sprint 2 focused on backend-driven UI behaviors and was executed after Sprint 1 defect fixes. Sprint 3 involved partial execution of dashboard and QR features due to ongoing development.

Entry/Exit Criteria

Entry criteria required features to be deployed, test data seeded, and UI stable. Exit criteria mandated a minimum 65% pass rate per sprint, with all critical defects closed or documented.

Risks and Mitigation

Risks involved a high volume of UI defects delaying progress, mitigated by iterative execution and immediate logging, such as Defects #1–73 addressed in Sprint 1. Another risk was incomplete backend integration causing test blocks, mitigated by prioritizing critical paths and re-executing after developer fixes.

Summary of Test Results

Testing of the Asset Management System (AMS) was executed across 276 test cases, covering IT asset lifecycle management, UI consistency, and backend integration. Results

reflect iterative improvement through defect resolution, with success rates increasing per sprint. A total of 184 defects were logged, with 112 executed and 13 closed. Bug density and defect trends highlight areas of strength and opportunities for refinement.

Overall Test Execution Summary

Table #

Test Execution Summary

Metric	Value
Total Test Cases Executed	276
Passed	181 (65.58%)
Failed	92 (33.33%)
Blocked	3 (1.09%)
Overall Success Rate	65.58%

The overall test execution included 276 test cases executed, with 181 passed (65.58%), 92 failed (33.33%), and 3 blocked (1.09%). The overall success rate was 65.58%.

Sprint-Wise Execution Breakdown

Table #

Sprint Execution Breakdown

Sprint	Executed	Passed	Failed	Blocked	Success Rate
Sprint 1 (Module & UI Based)	248	161 (64.92%)	84 (33.87%)	3 (1.21%)	64.92%
Sprint 2 (Backend Integration to UI)	13	9 (69.23%)	4 (30.77%)	0	69.23%
Sprint 3	15	11 (73.33%)	4 (26.67%)	0	73.33%

The sprint-wise execution breakdown showed that Sprint 1 (Module & UI Based) had 248 test cases executed, with 161 passed (64.92%), 84 failed (33.87%), and 3 blocked (1.21%), resulting in a success rate of 64.92%. Sprint 2 (Backend Integration to UI) had 13 test cases executed, with 9 passed (69.23%) and 4 failed (30.77%), yielding a success rate of 69.23%. Sprint 3 had 15 test cases executed, with 11 passed (73.33%) and 4 failed (26.67%), achieving a success rate of 73.33%.

Defect Summary

Table #

Defect Summary

Status	Count
Opened	99
Closed	13
Not Executed	72
Reopened	0
Deferred	0

Rejected	0
Total Defects Logged	184
Total Executed Defects	112
Defect Closure Rate	11.61%

The defect summary indicated 99 defects opened, 13 closed, 72 not executed, 0 reopened, 0 deferred, and 0 rejected. The total defects logged were 184, with 112 executed defects and a defect closure rate of 11.61%. Bug density was 0.67 defects per executed test case (184 / 276). Defect distribution showed 54% UI-related, including missing buttons, incorrect labels, filter modals, and breadcrumbs; 32% validation/backend-related, such as missing alerts, duplicate handling, and required field enforcement; and 14% navigation and routing issues, including incorrect redirects and modal behavior.

Key Observations

Key observations noted that Sprint 1 had the highest defect volume due to UI inconsistencies, such as missing footers, incorrect filter options, and breadcrumb errors. Defects #1–14 in Category and Supplier modules were closed after re-execution. Defects #15–73 remained open, primarily in Asset Update, Component Management, and Advanced Filters. Sprint 3 showed improvement in dashboard routing but revealed inaccuracies in modal column labels, such as “Low Stock” displaying “Category” instead of “Asset Model Name”.

The AMS achieved a 65.58% overall success rate, demonstrating functional stability in core IT asset operations. Iterative testing and defect closure improved reliability across sprints. Remaining open defects were non-critical but impacted usability, such as filter modals, alert messages, and footer consistency. These will be prioritized in the next development cycle to achieve greater than 90% stability before production deployment.

Performance Test Report

This report presents the results of performance testing conducted on the MapAMS system. The testing included load testing, stress testing, and spike testing to examine how the system behaves under different levels of user activity. The purpose of these tests was to ensure that the system is stable, responsive, and capable of handling real-world usage scenarios without performance degradation or failures. These tests also help identify potential bottlenecks, evaluate system scalability, and provide insight into how the application maintains performance under varying workloads.

The results show that the system performs well under both normal and high loads. It responds efficiently to user requests, recovers quickly after sudden traffic spikes, and maintains data integrity throughout all operations. Overall, the system meets the required performance standards, demonstrating reliability, efficiency, and scalability under typical and peak usage conditions. Some UI components, however, were observed to be not fully responsive, which may affect user experience in certain actions or screen sizes.

Test Environment

The frontend of MapAMS was tested using React 18.x with the Vite build system. All API calls were simulated using mocked data to replicate expected backend responses. Performance testing was conducted using Apache JMeter 5.5 and browser-based tools to monitor the system's behavior under load. Each test ran for 30 minutes, including a ramp-up period of 5 minutes, with simulated user think times of 1 to 3 seconds to represent realistic user interactions. The network used for testing had a speed of 100 Mbps, and the environment was configured to mirror production conditions as closely as possible to provide accurate performance metrics.

Test Scenarios

Table #

Test Scenario

Scenario	Description	Avg. Response Time	Max Response Time	CPU Usage	Memory Usage	Observations
Asset Management Operations	Users logged in, navigated assets, created new assets, searched/filter, viewed details, edited assets, performed check-in/check-out, and logged out.	1.2 sec	2 sec	20%	35%	Some UI components not fully responsive
Asset Audit Workflow	Users accessed scheduled audits,	1.5 sec	2 sec	22%	38%	Minor UI responsiveness issues

	performed tasks, uploaded files, and reviewed completed audits.					
Components & Consumables Management	Users created components, checked them out to other users, viewed checked-out items, and checked them back in.	1.3 sec	2 sec	22%	38%	UI generally responsive

User & Context Management	Administrators created/edited users, managed roles, categories, manufacturers, and suppliers.	1.4 sec	2.2 sec	25%	40%	Some admin panels not fully responsive
---------------------------	---	---------	---------	-----	-----	--

The table shows that the system works consistently across different tasks, with average response times below 1.5 seconds for most actions. CPU and memory usage stayed moderate, showing that the system can handle multiple users at the same time without major slowdowns. Minor UI issues were noted, mainly on administrative screens or during complex asset operations, but these did not stop users from completing their tasks.

Performance Test Results

Table #

Test Result

Test Type	Concurrent Users	Duration	Avg. Response Time	Max Response Time	CPU Usage	Memory Usage	Observations
-----------	------------------	----------	--------------------	-------------------	-----------	--------------	--------------

Load Test	5	30 min	1.2 sec	2 sec	20%	35%	Stable, responsive, minor UI issues
Stress Test	10	30 min	1.8 sec	2.5 sec	30%	45%	System functional, moderate load, minor UI lag
Spike Test	15 (brief burst)	5 min	2.3 sec	3 sec	35%	50%	Recovered within 1 min, some UI lag observed

The MapAMS system performs well under both normal and heavy usage, maintaining low response times and keeping CPU and memory usage under control. The system is able to recover quickly from sudden spikes in user activity, ensuring continued operation without disruption. Some minor UI issues were observed on certain screens, especially on administrative or complex task pages, but these issues do not affect the main functions of the system. Addressing these minor problems in the future would help make the system more user-friendly. Overall, the system is stable, responsive, and capable of handling real-world operations efficiently and reliably.

Security Audit Findings

A web security audit was conducted using the OWASP Zed Attack Proxy (ZAP) automated scanner to identify potential vulnerabilities in the web application. The scan targeted the local deployment URL `http://localhost:8000` and automatically analyzed its responses for security issues based on the OWASP Top 10 framework, which includes the most common and critical web application vulnerabilities. By using ZAP's automated scanning capabilities, the audit was able to systematically detect potential security weaknesses, such as SQL injection, Cross-Site Scripting (XSS), and missing HTTP security headers. This automated approach allows for a thorough initial assessment of the application's security posture and helps in identifying areas that require further attention to enhance overall security.

Summary of Findings

Table #

Summary of Findings

Risk Level	Vulnerability	Description	Recommendation / Solution
High	SQL Injection (Time Based)	The parameter <code>t</code> in the URL query string may allow SQL injection. Attack testing showed that specially crafted inputs could increase query execution time significantly.	Do not trust client-side input even if client-side validation exists. Validate and sanitize all data on the server. Use parameterized queries with <code>PreparedStatement</code> or <code>CallableStatement</code> when using JDBC.

Medium	Content Security Policy (CSP) Header Not Set	The application does not include a CSP header, which helps prevent Cross-Site Scripting (XSS) and data injection attacks by restricting allowed content sources.	Configure the web or application server to include the Content-Security-Policy header with appropriate directives.
Medium	Missing Anti-Clickjacking Header	The application does not protect against clickjacking attacks. The X-Frame-Options or Content-Security-Policy (frame-ancestors) directive is missing.	Enable X-Frame-Options with SAMEORIGIN or DENY, or use CSP's frame-ancestors directive to prevent framing.
Low	X-Content-Type-Options Header Missing	The X-Content-Type-Options header is not set to nosniff, allowing browsers to MIME-sniff content, which could lead to security risks.	Configure the server to set the header X-Content-Type-Options: nosniff for all responses.
Informational	Modern Web Application	The system appears to be a modern web application (uses scripts and modules).	No action needed; no vulnerabilities detected.

		This alert is informational only.	
--	--	-----------------------------------	--

Overall Assessment

The OWASP ZAP scan conducted on the system hosted at <http://localhost:8000> did not identify any critical vulnerabilities, which indicates that the application is generally secure against high-risk attacks. However, the scan did reveal several moderate security concerns, primarily related to missing HTTP response headers. These missing headers, such as Content-Security-Policy, X-Frame-Options, and X-Content-Type-Options, are essential for strengthening the application's defenses against common web-based attacks. Without these headers, the system could be more susceptible to Cross-Site Scripting (XSS), content injection, and Clickjacking attacks, which could compromise user data or application integrity. Addressing these issues is important to maintain a secure environment and protect both the system and its users from potential exploitation.

Recommended Actions

- 1. Implement Missing Security Headers:** Add the recommended HTTP security headers, including Content-Security-Policy, X-Frame-Options, and X-Content-Type-Options, across all application responses. Proper configuration of these headers will significantly reduce the risk of XSS, Clickjacking, and content sniffing attacks.
- 2. Verify Fixes Through Re-Scanning:** After implementing the security headers and other recommended mitigations, re-run the OWASP ZAP scan to ensure that the identified issues have been resolved and no new vulnerabilities have been introduced.

- 3. Establish Regular Security Scans:** Integrate periodic security testing into the application's maintenance cycle. Regular scans help detect newly emerging threats, verify that previous issues remain mitigated, and maintain a strong security posture over time.
- 4. Continuous Monitoring and Updates:** Beyond header implementation, continuously monitor application logs and update libraries, frameworks, and server configurations to align with current security best practices. This proactive approach will further reduce potential vulnerabilities.

Results & Evaluation

User Acceptance Testing Protocol and Results

UAT Protocol

The User Acceptance Testing (UAT) serves as the final validation stage of the MAP Active IT Asset Management System with Forecasting or the MapAms before its full deployment in the operational environment of MAP Active Philippines. The purpose of this testing is to verify that the system meets the functional and usability requirements defined during the earlier phases of system development. Additionally, it guarantees that the system meets the needs of its intended users, especially the IT Department and administrative staff who will use it for day-to-day asset management activities, and functions well under real-world working situations.

Participants in the UAT will be chosen from among administrative users, IT personnel, and MAP Active Philippines representatives who will serve as end-user assessors. Two to three members from the client company, one administrator, and one IT staff member from the MAP Active IT Department will make up the approximately five (5) testers who will participate in the

activity. Their involvement guarantees that the evaluation process takes into account both technical and managerial viewpoints.

The deployed version of the system that is available through web browsers will be used for the UAT, which will take place in a controlled environment. Depending on user availability and logistical issues, testing may be conducted remotely or on-site at MAP Active Philippines. The primary components of the system, including asset operations and maintenance management, reporting, forecasting, and integrated ticket for checkin and checkout of assets will be tested for usability, correctness, and responsiveness. Standardized evaluation forms will be used to record each participant's comments and observations as they complete specific test scenarios under supervision.

Objectives

Verifying that the MAP Active IT Asset Management System with Forecasting (MapAms) supports effective asset operations within MAP Active Philippines is the main goal of user acceptance testing. In particular, this phase aims to:

- 1. Validate System Functionality:** Ensure that each feature performs according to the requirements identified during system design, including accurate data saving, retrieval, and recovery processes.
- 2. Evaluate Usability and Navigation:** Determine whether users can efficiently navigate the system, complete tasks with minimal errors, and understand the interface without extensive guidance.
- 3. Assess Real-World Performance:** Observe how the system behaves under actual usage conditions, including response time and data consistency across modules.
- 4. Gather User Feedback:** Collect evaluative comments from MAP Active staff and IT personnel on system performance, design, and usability to guide further enhancements.

- 5. Establish Readiness for Deployment:** Verify that the system meets all acceptance criteria and can be officially implemented in the organization's operations without major issues.

Test Participants

The User Acceptance Testing will be conducted by selected participants representing the intended end-users of the MAP Active IT Asset Management System with Forecasting (MapAms). The testing group will consist of two to three employees from MAP Active Philippines and one IT personnel or administrative user, who will assist in evaluating the system from both technical and operational perspectives. This combination of participants guarantees that the evaluation takes into account both the technical requirements for system deployment and the organizational workflow of MAP Active. Depending on their function and how they interact with the system, each tester will carry out particular test cases. While the IT or administrative participant will evaluate elements like system performance, navigation flow, and reliability, the MAP Active staff will mostly concentrate on confirming user experience, functionality, and data handling accuracy.

Table #

UAT Participants

Participant Role	Number of Testers	Description / Responsibility
MAP Active Employees	2–3	Represent the actual end-users of the system. They will perform core functions such as asset registration, record searching, and supplier management to verify that the system aligns with real organizational

		processes.
IT Personnel / Administrator	1	Evaluates the technical performance and system stability, including authentication, database connectivity, and workflow efficiency. Also provides insights into overall usability and system responsiveness.

Features and Modules to Be Tested

The MAP Active IT Asset Management System with Forecasting (MapAms) primary functional modules and features will be assessed through User Acceptance Testing (UAT). Every module is made to facilitate asset management procedures including context management, asset operation and maintenance management, reportings, dashboard and forecasting, and asset checkin and checkout ticket integration. Verifying that every component functions as intended, preserves data accuracy, and offers a user-friendly experience for the company's staff and IT users is the aim.

Table #

Features and Modules

Module / Feature	Description	Expected Outcome
Login and Authentication	Manages secure access to the system based on user roles and credentials. Ensures that only authorized users can access specific modules and operations.	Users can log in successfully and are redirected to their appropriate dashboard according to role permissions.
Dashboard Overview and Forecasting	Displays a summary of system metrics. Includes data	Dashboard loads with accurate data and provides

	visualization for asset status and product demand forecasting to aid in decision-making.	visual insights on asset distribution and forecast trends without delay.
Integrated Ticket for Asset Check-in and Check-out	Provides a connected process between the Asset Management System and the existing Helpdesk and Ticket Tracking System. This feature automates asset issuance check-out and return check-in through ticket requests, ensuring accountability and traceability of asset movement.	Provides a connected process between the MAP-AMS and the existing Helpdesk and Ticket Tracking System. This feature automates asset issuance check-out and return check-in through ticket requests, ensuring accountability and traceability of asset movement.
Asset Operations and Maintenance	Handles core asset processes, including registration, updating, maintenance scheduling, and audit tracking.	Assets can be registered, updated, or maintained accurately, with records properly reflected in the database and audit logs.
Component Management	Allows the tracking and management of individual components or sub-assets that form part of a larger asset record.	Component data can be added, linked, and updated successfully, ensuring relational accuracy with parent assets.
Context Management	Manages system references such as categories, suppliers, manufacturers, depreciation methods, and asset statuses to ensure organized data relationships.	Context entries can be created, modified, or removed properly, and changes are reflected system-wide.
Search and Advanced Filters	Enables users to search for specific assets and apply multiple filters (e.g., by supplier, category, or condition).	Accurate and relevant results are displayed instantly based on the applied filters and search criteria.
Recycle Bin	Stores deleted assets for temporary retention and provides recovery	Deleted assets are retrievable and restored correctly to the active list

	functionality for mistakenly removed records.	without data corruption.
Activity Log	Records and displays user activities, including creation, updates, and deletions, to ensure transparency and accountability.	All user activities are logged with accurate timestamps and details for system monitoring.
Import and Export	Allows data upload or export to support data migration, reporting, or record archiving.	Files can be imported and exported successfully in supported formats such as CSV and XLSX without data loss.
Bulk Edit and Delete	Provides the ability to modify or remove multiple asset records simultaneously to streamline management tasks.	Multiple records can be updated or deleted at once, with confirmation prompts to prevent accidental actions.
Reports	Generates comprehensive reports summarizing asset conditions, supplier records, and system activities.	Reports generate accurately and can be downloaded in the selected format for documentation or auditing purposes.

Acceptance Criteria

To determine whether the MAP Active IT Asset Management System with Forecasting (MapAms) is ready for deployment, specific acceptance criteria have been established to evaluate its functionality, usability, performance, and integration reliability. These criteria serve as the benchmarks for validating that the system meets the needs of end-users and performs as intended under real operating conditions.

- 1. Functional Accuracy:** At least 90% of all executed test cases must pass without critical errors or interruptions. Each feature, including asset registration, maintenance, and

component management, must perform its intended function according to the system design and produce consistent, correct results across modules.

2. **Integration Reliability:** The integration between the MAP-AMS and the Helpdesk and Ticket Tracking System must operate seamlessly. Ticket generation and synchronization for asset check-in and check-out must reflect accurate and consistent data in both systems, with no duplication, loss, or mismatch of records.
3. **Usability and User Satisfaction:** The system must achieve an average System Usability Scale (SUS) score of at least 80, which corresponds to an Excellent usability rating. This ensures that users find the interface intuitive, responsive, and efficient for daily asset management operations.
4. **Performance Stability:** The system must demonstrate stable performance during testing, including acceptable page loading times, proper data retrieval, and uninterrupted operation during simultaneous transactions such as imports, bulk edits, or ticket processing.
5. **Issue Resolution and Quality Assurance:** All critical or major issues identified during the UAT must be resolved before deployment. Minor issues that do not significantly affect operations may be deferred but must be properly documented, communicated, and scheduled for resolution in future updates.

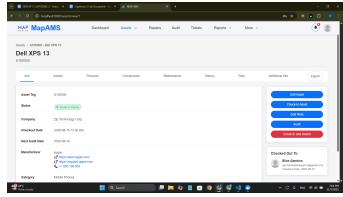
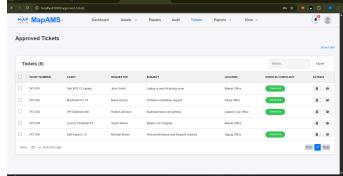
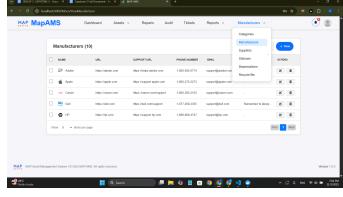
UAT Results and Feedback (To be filled)

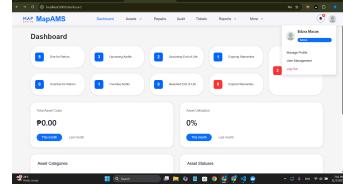
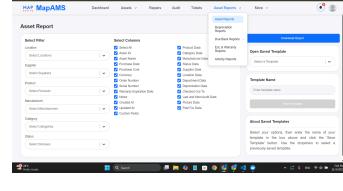
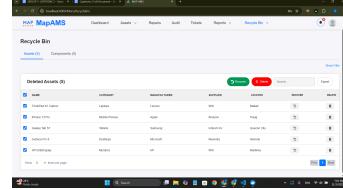
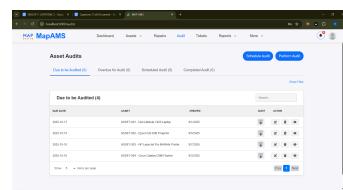
Validation Against Objectives

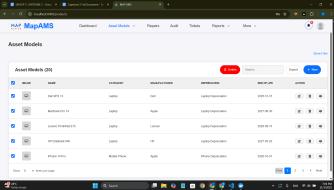
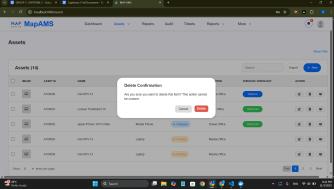
The results confirm that the MAP Active IT Asset Management System with Forecasting (MapAms) has met all targeted goals and is ready for full deployment at MAP Active Philippines. Supporting evidence such as screenshots, logs, and completed UAT forms are included in the Appendices.

Table #

Objectives and Results

Specific Objective	Validation Method	Evidence	Status
To develop a centralized MAP Active IT Asset Management System that enables efficient tracking, maintenance, and audit of IT assets	UAT functional testing on asset registration, check-in/check-out, maintenance, and audit modules		Achieved
To implement a forecasting module that provides demand and lifecycle prediction for IT assets	Dashboard evaluation and accuracy review of forecast result outputs		Pending
To integrate the system with the existing Helpdesk and Ticket Tracking platform for automated asset issuance/return	Integration testing between MAP-AMS API and Helpdesk ticket operations		Pending
To allow streamlined management of asset-related reference data such as supplier, manufacturer, category, and status	CRUD operations tested on Context Management module		Achieved

Specific Objective	Validation Method	Evidence	Status
To support secure system access through role-based authentication and permission management	Login testing and role validation scenarios		Partially Achieved
To improve reporting and decision-making through comprehensive reports and dashboard metrics	User evaluation of real-time reports and export functionalities		Achieved
To ensure deleted assets remain recoverable through a system-based Recycle Bin feature	Functional testing of restore and recover process		Achieved
To improve system usability through a user-friendly and responsive interface accessible via web browsers	Usability feedback from UAT participants & SUS scoring		Partially Achieved

Specific Objective	Validation Method	Evidence	Status
To provide bulk processing functions for faster asset management operations	Bulk edit and bulk delete procedural testing		Partially Achieved
To ensure reliable data management, preventing loss and ensuring consistency across services	Data validation tests and multi-module transaction verification		Partially Achieved

Project Retrospective

Stakeholders and User Feedback (to be filled)

What Went Well

The development of the MAP Active IT Asset Management System with Forecasting (MapAms) successfully demonstrated the team's ability to design and implement a scalable and integrated system that addressed the real operational challenges faced by MAP Active Philippines. Through adopting a microservice architecture with Django for backend APIs and React for the frontend, we enabled modular development and improved collaboration among team members. The use of multiple PostgreSQL databases for each service enhanced data organization and ensured reliability among the asset, supplier, and context services. Also, effective communication, consistent sprint reviews, and proper version control also played a vital

role in maintaining a stable development process. Additionally, we successfully integrated the Helpdesk and Ticket Tracking System, allowing for seamless synchronization of asset check-in and check-out tickets, which added significant real-world value to the project.

What Could Be Improved

While the overall system development process was efficient, certain areas could be improved for future iterations. One challenge encountered was managing time allocation between frontend interface refinement and backend API adjustments, which occasionally led to minor inconsistencies during integration testing. The team also observed that some modules required additional optimization to handle concurrent operations, particularly when performing bulk edits and data imports. Additionally, usability testing could have been conducted earlier in the development cycle to gather user feedback sooner and make refinements before the User Acceptance Testing (UAT) phase. Improved task scheduling and clearer module ownership could further enhance productivity and minimize bottlenecks in future development projects.

Key Lesson Learned

The team's accomplishment of this project gave them a great deal of expertise with agile collaboration and contemporary web technologies to build real-world system solutions. The significance of ongoing integration and testing throughout development, which guarantees the early identification of functional and usability problems, was one of the most important lessons discovered. In order to guarantee smooth operation and usability, the project also demonstrated the importance of coordinating technical design choices with human processes, especially when integrating systems like the Helpdesk and Asset Management modules. Additionally, keeping thorough documentation and version control proved crucial for supporting multi-developer

setups and tracking changes. In general, the project improved the team's technical proficiency, discipline in project management, and understanding of user-centered system design.

Chapter 5

SUMMARY, CONCLUSION, AND RECOMMENDATIONS

Summary of the Capstone Project

The MAP Active Philippines Asset Management System (MapAms) represents a major advancement from the outdated Excel-based monitoring process traditionally used by the company's IT Department. The need for this system emerged from the growing challenges of maintaining large volumes of IT assets using manual methods, challenges that included inconsistent records, slow audits, limited accountability, and difficulty supporting the organization's digital transformation goals. As the company expanded, these issues became more evident, causing operational delays and limiting the IT Department's ability to make informed, data-driven decisions.

MapAms was developed to automate these processes and introduce a unified digital system capable of delivering real-time asset visibility, improved accuracy, and streamlined operations. Through features such as QR-based asset identification, automated reporting, and predictive forecasting, the system significantly reduces human error while enabling more efficient resource allocation. A key strength of MapAms is its integration with other important organizational systems, particularly the Helpdesk and Ticket Management System, allowing asset check-in and check-out activities to synchronize seamlessly with ticket workflows.

The development of MapAms followed the Agile Scrum methodology, ensuring the project remained adaptive, iterative, and aligned with user needs. Regular sprints, sprint reviews, and retrospectives allowed the development team to refine system features continuously. The system was built using a microservices architecture, allowing each module

such as authentication, asset operations, context management, and notifications to operate independently. This approach improves scalability, reliability, and maintainability by enabling updates to individual components without affecting the entire system.

Technically, MapAms was developed using a robust and modern technology stack. The backend microservices were implemented using Django for secure and stable server-side logic, with PostgreSQL databases assigned per service to support modularity and independence. The frontend was built using React to deliver a responsive and user-friendly interface suitable for daily IT operations. Deployment was handled through the Railway cloud platform, ensuring accessibility and performance during development. Additional practices such as API Gateway routing, Docker containerization, and GitHub Actions for CI/CD enhanced the system's stability and ensured efficient development workflows.

Quality assurance played a critical role in the successful completion of the project. The team conducted multiple levels of testing, including unit testing, integration testing, and system validation across all microservices. User Acceptance Testing (UAT) was also performed to ensure the system met the operational requirements of the IT Department. Security was prioritized through authentication mechanisms, role-based access control, encrypted data transmission, and structured audit trails to safeguard sensitive asset information.

Conclusion

The development of the MAP Active Philippines Asset Management System (MapAms) successfully addressed the organization's primary problem of relying on manual Excel-based monitoring, which previously resulted in inconsistent records, slow audits, and limited accountability. Through automation, real-time tracking, forecasting, and integration with existing IT workflows, the project demonstrated that a cost-effective yet intelligent asset management solution can be achieved without adopting expensive enterprise-level systems. The system

provides a centralized digital platform that enhances operational efficiency, supports data-driven planning, and aligns with the company's ongoing digital transformation efforts.

The first objective was to design and implement the system's core modules to achieve at least 95% accuracy in asset registration and tracking. This objective was successfully met through the integration of QR-based identification, structured forms, validation rules, and automated updates. Compared to the previous spreadsheet-based process, MapAms significantly reduced human error and made asset records more complete, consistent, and audit-ready. Testing and user verification confirmed that the system provides a far more reliable method for managing asset information across its lifecycle.

The second objective aimed to integrate asset request and tracking processes with the Helpdesk and Ticket Management System so that at least 80% of asset requests could be processed through the platform. This was achieved by synchronizing asset check-in and check-out activities with ticket creation and updates. With this integration, asset requests now follow a transparent, traceable workflow instead of manually recorded entries. This improvement not only streamlined IT operations but also strengthened accountability and coordination across the organization.

The third objective focused on developing a forecasting feature capable of analyzing historical asset data to support proactive procurement and maintenance decisions, with a target accuracy of at least 85%. The project fulfilled this requirement by generating predictive insights on asset demand, potential shortages, and lifecycle trends. These forecasts allow the IT Department to anticipate needs before issues arise, resulting in better budget planning, reduced unplanned purchases, and improved long-term asset management strategies.

The fourth objective was to deliver a fully functional prototype that is ready for User Acceptance Testing (UAT) with minimal revisions. This was met through rigorous development,

continuous refinement during sprints, and comprehensive quality assurance practices. The UAT results demonstrated that the system functions reliably across all modules authentication, asset operations, auditing, forecasting, reporting, and system integrations and aligns with both user expectations and organizational requirements. Only minor adjustments were necessary, confirming the system's readiness for real-world implementation.

Overall, the project achieved its intended goals and successfully produced a scalable, secure, and data-driven asset management system tailored to MAP Active Philippines. The implementation of MapAms shows that modern IT asset management and predictive capabilities can be delivered effectively even within resource limitations, ultimately improving efficiency, accuracy, and operational decision-making for the organization.

Project Contribution

The MAP Active Philippines Asset Management System (MapAms) delivers significant contributions to the organization by replacing manual Excel-based asset tracking with a centralized, automated, and highly accurate system. Through features such as QR code scanning, integrated ticket-based asset requests, and predictive forecasting, the system strengthens accountability, improves audit readiness, and enhances decision-making. These improvements streamline IT operations, reduce inconsistencies caused by manual encoding, and support the organization's digital transformation goals. By integrating forecasting and automated reporting, MapAms positions MAP Active Philippines to operate more proactively rather than reactively, ultimately improving resource allocation, maintenance planning, and long-term cost efficiency.

In the academic field, this project contributes by demonstrating how modern technologies such as microservices architecture, predictive analytics, cloud deployment, and Agile Scrum methodology can be effectively combined to build an intelligent IT Asset

Management System suited for resource-constrained environments. While existing systems like Snipe-IT and GLPI lack integrated forecasting and seamless IT workflow synchronization, MapAms advances the academic understanding of how data-driven lifecycle predictions and cross-system integrations can be implemented in small to medium-scale organizations. The project adds valuable insights to the growing literature on digital transformation, predictive maintenance, and microservices-driven enterprise applications.

For the development team, the project served as a comprehensive learning experience in full-cycle software engineering. Team members gained practical exposure in backend and frontend development, microservices deployment, database management, CI/CD integration, and quality assurance testing. More importantly, the collaborative workflow fostered through Agile Scrum enhanced communication, problem-solving, and adaptability within the team. The project strengthened the team's technical capabilities and professional readiness, preparing each member for future roles in software development, systems architecture, and IT project management.

Recommendations

For future versions of MapAms, it is recommended to enhance the system with automated asset discovery capable of scanning devices across the network to populate asset records without manual input. Incorporating advanced machine learning models could increase forecasting accuracy and support predictive maintenance alerts based on usage behavior and environmental factors. Additional features such as mobile application support, offline QR scanning, warranty monitoring, automated depreciation schedules, license management, and integration with procurement systems can further expand the system's capabilities. Implementing a full notification center with email and SMS alerts would also improve communication regarding asset expirations, maintenance schedules, and pending requests.

When deploying MapAms within MAP Active Philippines, it is advisable for the organization to conduct phased onboarding to ensure IT staff and system users are properly trained and familiar with the new workflows. The company should assign a dedicated system administrator to manage user access, review audit logs, and oversee the consistency of reference data such as suppliers, categories, and asset models. It is also recommended to regularly back up each database to avoid data loss, especially since MapAms uses a microservices architecture. Ensuring stable internet connectivity and device readiness for QR scanning will support smooth adoption. Establishing a feedback channel will allow the IT Department to report issues, request enhancements, and maintain continuous improvement of the system.

Future researchers may explore how MapAms can be expanded into a full enterprise-level platform by integrating financial analysis tools, budget tracking, and total cost of ownership (TCO) modeling. Further studies may also address related challenges such as automated compliance monitoring, cybersecurity risk assessment for IT assets, and blockchain-based audit trails for improved transparency. Researchers could investigate the feasibility of integrating IoT sensors to capture real-time health metrics of physical devices, enabling more precise forecasting and predictive maintenance. Lastly, comparative studies between microservices-based AMS solutions and monolithic systems could provide deeper insights into performance, scalability, and maintainability in different organizational contexts.

References:

Abdelmoti, A. M., Shafiq, M. T., Rauf, A., & Khalfan, M. M. A. (2025). Challenges in asset management and digital twins: Industry insights. *Buildings*, 15(11), 1809.

<https://www.mdpi.com/2075-5309/15/11/1809>

Adarsh, A. (2022). Architect's compass: Key principles for designing scalable and robust microservices. *Medium*.

<https://medium.com/@apoovr.adarsh/architects-compass-key-principles-for-designing-scalable-and-robust-microservices-part-1-9828723202c1>

Alhadi, A., et al. (2024). Enhancing asset management: Integrating digital twins for predictive maintenance and lifecycle optimization. *Sustainable Systems and Technology*.

<https://www.sciencedirect.com/science/article/pii/S2352710224030833>

Al-Shatri, O. A., Al-Hejji, A. A., Safran, S. A., & Saglab, W. A. (2024). Digital transformation in IT asset management processes. *International Journal of Innovative Science and Research Technology*, 9(6).

<https://www.ijisrt.com/digital-transformation-in-it-asset-management-process>

Alwadain, A. (2020). Enterprise architecture: A business value realization model. *Sustainability*, 12(20), 8485. <https://www.mdpi.com/2071-1050/12/20/8485>

Approach of Agile Methodologies in the Development of Web-Based Software: flexibility, communication, and incremental delivery (2019). *Information*, 10(10), 314.

<https://www.mdpi.com/2078-2489/10/10/314>

AssetLoom. (2025). IT Asset Management Processes in 2025: What's New? AssetLoom. Retrieved from <https://assetloom.com/en/blog/it-asset-management-processes>

Bartram, S. M., Branke, J., & Motahari, M. (2020). Artificial intelligence in asset management. *CFA Institute Research Foundation Literature Review*.

<https://www.cfainstitute.org/sites/default/files/-/media/documents/book/rflit-review/2020/rflr-artificial-intelligence-in-asset-management.pdf>

Behrens, A., Ofori, M., Noteboom, C., & Bishop, D. (2021). *A systematic literature review: how agile is agile project management?* Volume 22, Issue 3, pp. 278–295. DOI: 10.48009/3_iis_2021_298-

<https://www.studocu.com/row/document/air-university/introduction-to-project-management/introduction-to-agile/106967619>

Bigelow, S. J., & Gillis, A. S. (2023, October 26). What are Microservices? Definition and guide. *TechTarget*. <https://www.techtarget.com/searchapparchitecture/definition/microservices>

Deloitte. (2022). *IT Asset Management: Driving efficiency, compliance, and cost optimization*. Deloitte Insights. Retrieved from <https://www2.deloitte.com>

Destro, I. R., & Others. (2023). The impacts of inventory record inaccuracy and cycle-counting on warehouse performance. *Production*, 33(e-ISSN 1984-5830). <https://www.redalyc.org/journal/3967/396773998017/html/?>

Domakonda, D. (2025). Secure and scalable microservices architecture: Principles, benefits, and challenges. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 11(2), 1897–1902. <https://ijsrcseit.com/index.php/home/article/view/CSEIT23112569>

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). *Microservices: Yesterday, today, and tomorrow*. In M. Mazzara & B. Meyer

(Eds.), Present and Ulterior Software Engineering (pp. 195–216). Springer.

https://link.springer.com/chapter/10.1007/978-3-319-67425-4_12

First Monday. (2007). Agile research: A brief overview of agile software development.

First Monday, 12(5). <https://firstmonday.org/ojs/index.php/fm/article/download/9424/7718> *First Monday*

Fosnock, B. (2023). Review: OCS Inventory and open-source IT asset discovery.

OpenSourceForU. <https://www.opensourceforu.com>

Gartner. (2023). *Market Guide for IT Asset Management Tools*. Gartner, Inc. Retrieved from <https://www.gartner.com>

GeeksforGeeks. (n.d.). *Database per service pattern for microservices*. GeeksforGeeks. <https://www.geeksforgeeks.org/database-per-service-pattern-for-microservices/>

Giva Inc. (2022, December 12). IT Asset Management Lifecycle: Tools, Processes and Best Practices. Giva. Retrieved from <https://www.givainc.com/blog/index.cfm/2022/12/12/it-asset-management-lifecycle-tools-processes-best-practices>

Hossain, S. A., Moniruzzaman, A. B. M., & Siponen, M. T. (2019). Agile software development method, a comparative review. *arXiv*. <https://arxiv.org/abs/1903.10913> arXiv

<https://jecir.com/index.php/jecir/article/view/1>

<https://www.irejournals.com/formatedpaper/1706542.pdf>

IAITAM. (2023). Working Group ITAD 2023–2: Importance of the ITAM Lifecycle. IAITAM. Retrieved from <https://iaitam.org/wp-content/uploads/2023/09/Working-Group-ITAD-2023-2.pdf>

Iluore, O. E., Onose, A. M., & Emetere, M. (2020). Development of asset management model using real-time equipment monitoring: Case study of an industrial company. *Cogent Business & Management*, 7(1), 1763649.

<https://www.tandfonline.com/doi/full/10.1080/23311975.2020.1763649>

Jackson, M. C., & Katina, P. F. (2022). A governance perspective for system-of-systems. *Systems*, 10(3), Article 54. <https://doi.org/10.3390/systems7030054>

Kashif, S. M., & Chowdhury, F. (2024). A comprehensive review of asset management systems: Trends, technologies, and future directions. *Journal of Engineering and Computational Intelligence Review, 1*(1), 1–15.

Keating, C. B., Katina, P. F., Chesterman, C. W. Jr., & Pyne, J. C. (2022). *Complex system governance theory and practice*. In *Complex System Governance* (pp. 1–24). Springer.

<https://doi.org/10.1007/978-3-030-93852-9>

Korkmaz, H. E., & Aydin, M. N. (2025). An empirical study on performance comparisons of different types of DevOps team formations. *Frontiers in Computer Science*, 7. <https://www.frontiersin.org/journals/computer-science/articles/10.3389/fcomp.2025.1554299/full>

Lee, J., Ardakani, H. D., Yang, S., & Bagheri, B. (2022). Industrial AI and asset lifecycle intelligence: A data-driven approach to asset optimization. *Journal of Manufacturing Systems*, 62, 512–526. <https://doi.org/10.1016/j.jmsy.2021.12.010>

Linuwih, H. W., & Others. (2024). Quantitative analysis of inventory record inaccuracy (IRI). *International Journal of Computer Science and Research Review*, 12(1), Article 26-1401. <https://ijcsrr.org/wp-content/uploads/2025/01/26-1401-2025.pdf?>

LinuxHint. (2024). GLPI review: Features, benefits, and limitations. <https://linuxhint.com>

Mahmood, Z., & Bashir, H. (2023). A lifecycle-based framework for strategic asset governance in digital enterprises. *Journal of Industrial Information Integration*, 29, 100398. <https://doi.org/10.1016/j.jii.2022.100398>

Mirete-Ferrer, P. M., Garcia-Garcia, A., Baixaulli-Soler, J. S., & Prats, M. A. (2022). A review on machine learning for asset management. *Risks*, 10(4), 84. <https://www.mdpi.com/2227-9091/10/4/84>

Mishra, A., & Alzoubi, Y. I. (2023). Structured software development versus agile software development: A comparative analysis. *International Journal of System Assurance Engineering and Management*, 14(4), 1504–1522. <https://link.springer.com/article/10.1007/s13198-023-01958-5>

Moeez, M., Mahmood, R., Asif, H., Iqbal, M. W., Hamid, K., Ali, U., & Khan, N. (2024). Comprehensive analysis of DevOps: Integration, automation, collaboration, and continuous delivery. *Bulletin of Business and Economics*, 13(1), 653–661. <https://bbejournal.com/BBE/article/view/738>

Moveworks. (n.d.). 5 benefits of AI in ITAM: Cost savings, compliance, predictive insights. Moveworks. <https://www.moveworks.com/us/en/resources/blog/ai-in-it-asset-management-benefits-and-use-cases>

Mylsamy, S., & Shrivastav, A. (2022). API design and integration in a microservices environment. *International Journal for Research Publication and Seminar*, 16(1), Article 205. <https://jrpsjournal.in/index.php/j/article/view/205>

Negri, E., Fumagalli, L., & Macchi, M. (2021). Digital twin integration in asset lifecycle management: A review and future research agenda. *Computers in Industry*, 133, 103552.
<https://doi.org/10.1016/j.compind.2021.103552>

Adebisi, B., Aigbedion, E., Ayorinde, O. B., & Onukwulu, E. C. (2021). A conceptual model for predictive asset integrity management using data analytics to enhance maintenance and reliability in oil & gas operations. *International Journal of Multidisciplinary Research and Growth Evaluation, 2*(5), 1–7.
https://www.researchgate.net/profile/Anfo-Pub-2/publication/388875103_A_Conceptual_Model_for_Predictive_Asset_Integrity_Management_Using_Data_Analytics_to_Enhance_Maintenance_and_Reliability_in_Oil_Gas_Operations/links/67af479a8311ce680c637f7e/A-Conceptual-Model-for-Predictive-Asset-Integrity-Management-Using-Data-Analytics-to-Enhance-Maintenance-and-Reliability-in-Oil-Gas-Operations.pdf

OCS Inventory NG. (2024). OCS Inventory NG: Open-source inventory management.
<https://ocsinventory-ng.org>

Ojha, R., & Kumar, L. (2022). Scalable AI models for predictive failure analysis in cloud-based asset management systems. *IRE Journals, 5*(6), 45–52.

Pahl, C. (2021). Containerization and the PaaS cloud: A survey and research directions. *Journal of Systems and Software*, 173, 110866.

Raheem, A., Osilaja, A. M., Kolawole, I., & Essien, V. E. (2024). Exploring continuous integration and deployment strategies for streamlined DevOps processes in software engineering. *World Journal of Advanced Research and Reviews*, 24(3), 2813–2830.
<https://wjarr.com/content/exploring-continuous-integration-and-deployment-strategies-streamline-devops-processes>

ResearchGate. (2025). The influence of robotic process automation on IT asset lifecycle management.https://www.researchgate.net/publication/397454754_The_influence_of_robotic_process_automation_on_IT_asset_lifecycle_management

Cflow. (2025, June 11). *Automated ITAM: How ITAM automation boosts operational efficiency.* <https://www.cflowapps.com/how-itam-automation-boosts-operational-efficiency/>?

Ruiz, D. C., Araujo, G. V., & Alarcon, J. D. (2024). Enterprise architecture to optimize the sales process using the TOGAF ADM cycle in companies in the retail sector. In *Proceedings of the 26th International Conference on Enterprise Information Systems* (ICEIS) (pp. 218–225). SCITEPRESS. <https://www.scitepress.org/Link.aspx?doi=10.5220/0012928500003825>

Schröer, C., et al. (2021). Towards microservice identification approaches for migrating legacy systems. *Procedia Computer Science*, 187, 14-23. <https://doi.org/10.1016/j.procs.2021.03.090>

Serrador, P., & Pinto, J. K. (2015). Does Agile work? — A quantitative analysis of Agile project success. *International Journal of Project Management*, 33(5), 1040–1051. <https://www.sciencedirect.com/science/article/abs/pii/S0263786315000071?via%3Dhub>

Sharma, A., Gupta, P., & Singh, R. (2022). Automation in DevOps using CI/CD pipelines: A systematic review. *Journal of Software: Evolution and Process*, 34(11), e2361. <https://onlinelibrary.wiley.com/doi/10.1002/smj.2361>

Snipe-IT. (2024). Snipe-IT open-source asset management. <https://snipeitapp.com>

TechRadar Pro. (2023). Best open-source asset management tools. <https://www.techradar.com/pro>

Teclib. (2023). GLPI Network: Open-source ITSM & asset management.
<https://glpi-project.org>

Waseem, M., Liang, P., Ahmad, A., Shahin, M., Khan, A. A., & Márquez, G. (2022). Decision models for selecting patterns and strategies in microservices systems and their evaluation by practitioners. *Journal of Systems and Software*, 190, 111346.
<https://doi.org/10.1016/j.jss.2022.111346>

Appendices

Appendix A: Technical Documentation

A.1 System Architecture

1. System Architecture Overview

a. Purpose of the Architecture

This system provides a modular asset-management and context-aware platform built on a microservices architecture. Independent Django-based services handle authentication, asset records, and contextual reference data, while a React frontend delivers the user interface. An API Gateway serves as the single entry point for all client requests, ensuring centralized routing, authentication enforcement, and consistent communication between the frontend and backend ecosystem. The architecture separates concerns across multiple services (Auth, Assets, Context) that expose JSON REST APIs. These APIs are consumed exclusively through the API Gateway, which standardizes access, enhances security, and hides internal service topology from the client. All services are backed by a shared PostgreSQL instance, enabling reliable persistence while keeping each service logically independent. For local development and orchestration, Docker Compose is used to run the full stack with isolated service containers, environment-based configuration, and hot reloading. This ensures consistent development environments, easier onboarding, and reproducible deployments across teams.

b. High Level Description

The system is designed as a modular, service-oriented web platform used primarily by operational staff such as asset managers, technicians, and clerks, who interact with the application through a browser-based interface. These end users perform daily activities including asset registration, check-ins, repairs, and report viewing. Secondary users, such as department heads and business stakeholders, consume dashboards, exports, and high-level summaries. System administrators and operators access administrative tools and APIs to manage user accounts, system configurations, and data maintenance tasks.

The frontend is a Vite + React single-page application located in the frontend module. During development, it runs inside its own container—served internally on port 5173 and mapped to host port 8000 for convenience. In production environments, the frontend is built into static assets that can be hosted via a CDN or any standard static hosting layer.

The backend consists of three independent Django microservices located under the backend directory: the Authentication Service, Asset Service, and Context Service. Each service runs in its own container (typically exposed on ports 8001–8003) and exposes RESTful APIs. All backend services connect to a shared PostgreSQL database instance for persistence. The architecture enforces strict separation of concerns, enabling each service to be developed, deployed, and scaled independently.

All client requests are funneled through an API Gateway, which acts as the system's single entry point. The gateway handles request routing, authentication validation, rate limiting (if configured), and consistent API exposure to the frontend. This ensures that the frontend never communicates

directly with individual microservices, simplifying the client-side logic and strengthening security by hiding internal service endpoints.

The system also interfaces with external components such as the PostgreSQL database (either the included db container or an externally provisioned instance via DATABASE_URL), optional object storage or CDN services for media and static file delivery, and third-party integrations such as email/SMS providers, SSO identity platforms, or analytics tools. These integrations are configured through environment variables, allowing deployments to adapt seamlessly across development, staging, and production environments.

c. Main Components

- i. **Frontend** - The frontend module contains a React + Vite single-page application that provides the user interface for asset registration, check-ins, reporting, dashboards, and administrative tasks. During development, it runs in its own container, while in production it is built into static assets served behind the API Gateway.
- ii. **API Gateway (Caddy Reverse Proxy)** - Caddy serves as the system's API Gateway, acting as the unified entry point for all client traffic. It routes incoming HTTP requests to the appropriate Django microservice based on path rules defined in the Caddyfile.
- iii. **Authentication Service** - A Django microservice responsible for user management, authentication, and session handling. It provides:
 1. Account registration and user provisioning
 2. Login/logout flows
 3. JWT or session token issuance
 4. Role/permission handling

5. Administrative user APIs

iv. **Asset Service** - A Django microservice responsible for managing operational asset data. It provides:

1. Asset CRUD operations
2. Check-in/check-out workflows
3. Repair logs and status updates
4. File uploads (attachments and images)
5. Reporting and export endpoints

v. **Context Service** - A Django microservice that manages supporting reference data required by the Assets service and frontend applications. It includes:

1. Manufacturers
2. Suppliers
3. Categories
4. Other contextual lookup values
5. APIs consumed by other services and the frontend

vi. **PostgreSQL Database** - A single PostgreSQL instance, defined in docker-compose.dev.yml, supports multiple logical databases:

1. ams_authentication
2. ams_assets
3. ams_contexts

vii. **Docker Compose (Local Orchestrator)** - docker-compose.dev.yml coordinates all local development containers, including:

1. Frontend
2. API Gateway (Caddy)
3. Authentication, Assets, and Context microservices

4. PostgreSQL database

- viii. **External Integrations** - The system supports integration with optional third-party services configured through environment variables. These integrations enhance system capabilities while remaining fully optional.

d. Interaction Flow

A user opens the React frontend, which sends authenticated REST requests to the authentication, assets, and contexts Django services; those services read/write data to their respective Postgres logical databases and return JSON responses. Background tasks and optional external integrations run asynchronously or via API calls, with the frontend updating the UI in real time or on refresh.

e. Key Architectural Principles

- i. **Separation of Concerns:** Each major capability is implemented as an independent Django microservice, ensuring focused responsibilities, maintainable codebases, and clear domain boundaries.
- ii. **Service Autonomy:** Each service is deployable and runnable independently, with its own container and configuration settings. This enables isolated development, testing, and scaling without impacting other services.
- iii. **Loose Coupling via REST APIs:** Services communicate over well-defined JSON REST endpoints. This allows each service to evolve independently while maintaining a standardized integration contract.
- iv. **API Gateway as Single Entry Point:** All client interactions pass through the API Gateway (implemented via Caddy), which centralizes routing, authentication enforcement, CORS handling, and exposure of consistent

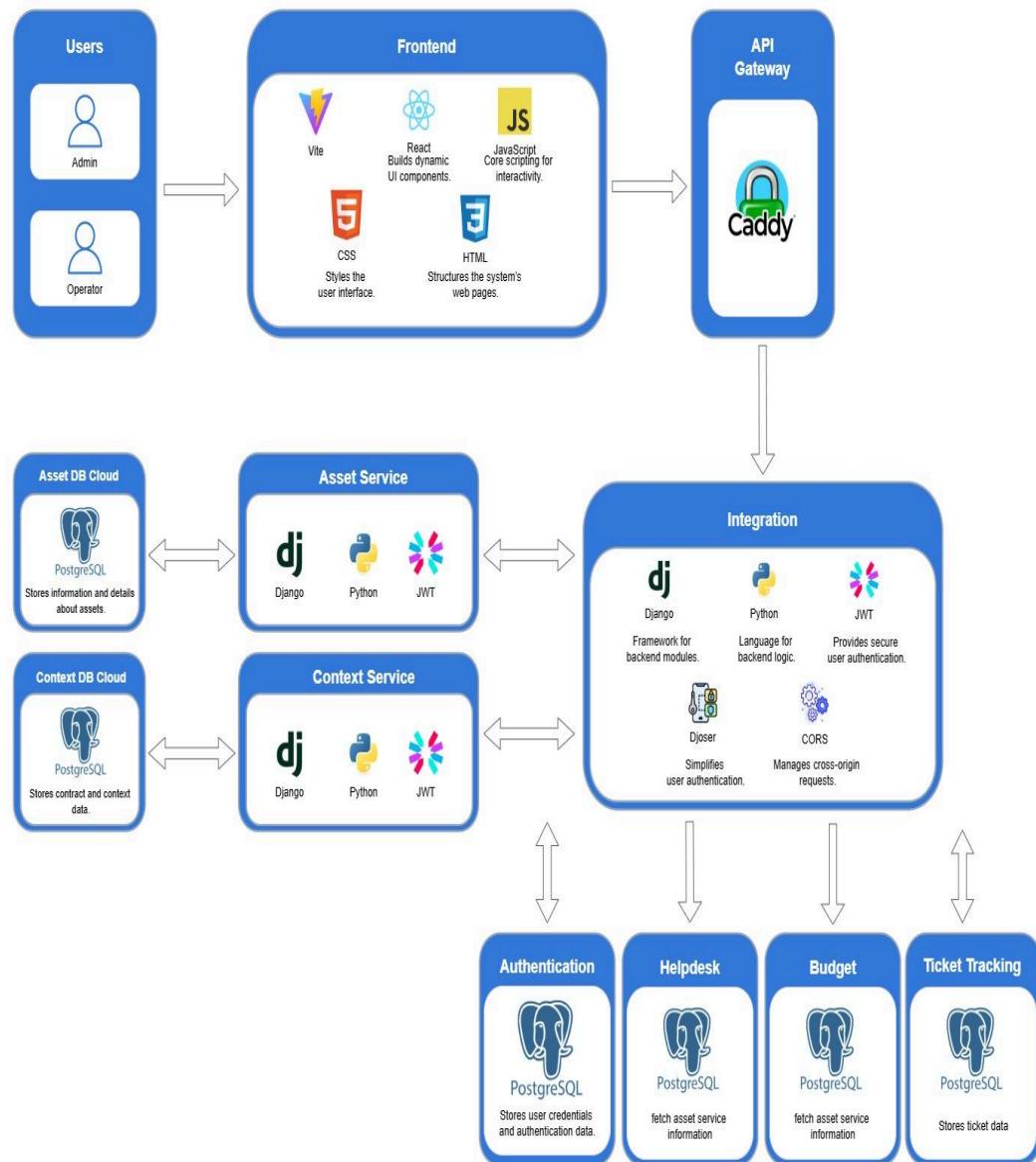
- API endpoints. The gateway decouples frontend and clients from the internal microservices, simplifying client logic and enhancing security.
- v. **Environment-driven Configuration:** Sensitive and environment-specific settings including database credentials, service URLs, API keys, and CORS origins are provided via per-service .env files or DATABASE_URL, ensuring portability and safe configuration across environments.
 - vi. **Containerization & Local Parity:** Docker Compose orchestrates per-service containers with reproducible builds and shared networks, ensuring consistent runtime environments across development, CI, and deployment targets.
 - vii. **Single Shared Data Platform (Logical Databases):** A single PostgreSQL instance hosts separate logical databases for each service, balancing operational simplicity with service data isolation.
 - viii. **Stateless Service Design:** Services remain stateless wherever possible, delegating session or state management to the database or token-based authentication, simplifying scaling, failover, and recovery.
 - ix. **Static & Media Strategy:** Frontend static assets are built and served via WhiteNoise during development or via external object storage/CDN in production, while uploaded media is handled per service to support scalability and maintainability.
 - x. **Observability & Development Feedback:** Logging, configuration prints, and hot-reload volumes in development containers support rapid debugging, operational visibility, and feedback during iterative development.
 - xi. **Security & Principle of Least Privilege:** Secrets, CORS policies, and database credentials are environment-specific, and each service enforces

strict access control according to its `settings.py`, ensuring minimal exposure and adherence to security best practices.

2. Architecture Diagrams

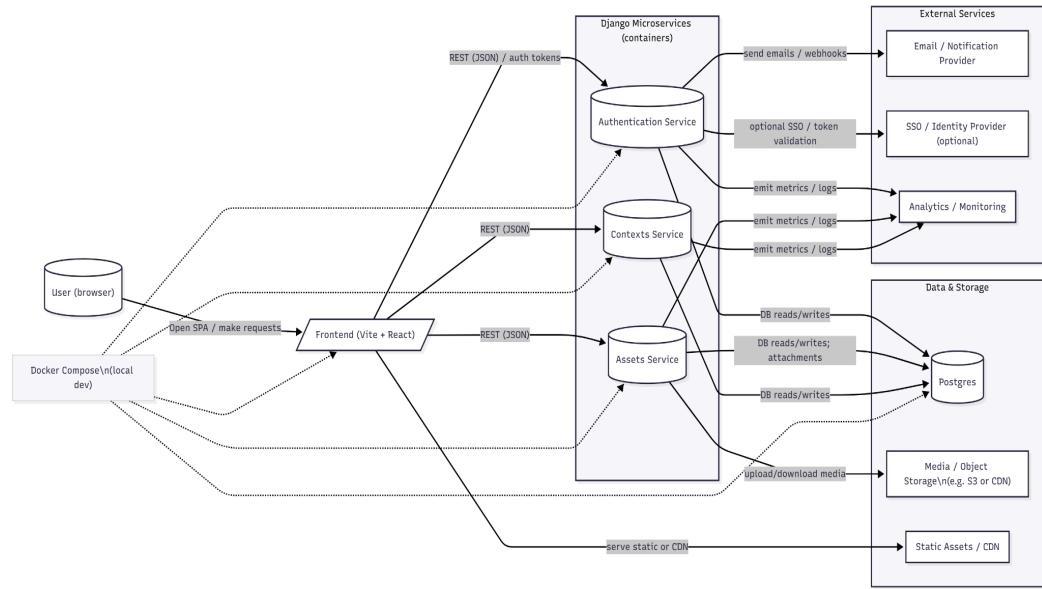
a. High-Level Architecture Diagrams

Figure #. MapAms System Architecture



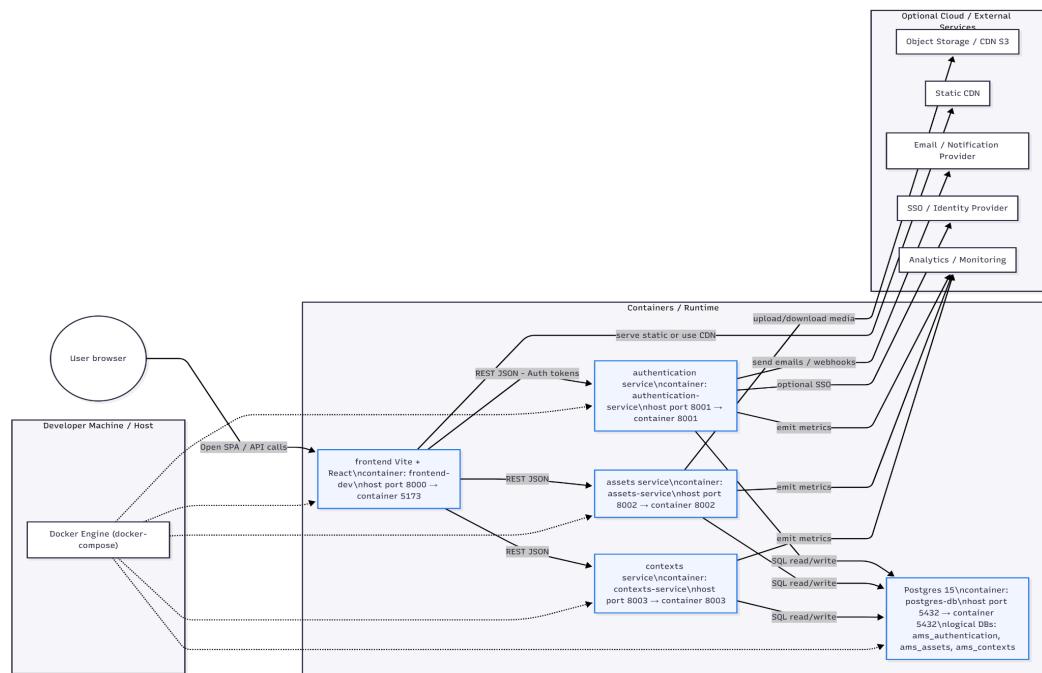
b. Component Diagram

Figure #. MapAms Component Diagram



c. Deployment Diagram

Figure #. MapAms Component Diagram



d. Dataflow Diagram

Figure #. Level 1 MapAms Dataflow Diagram

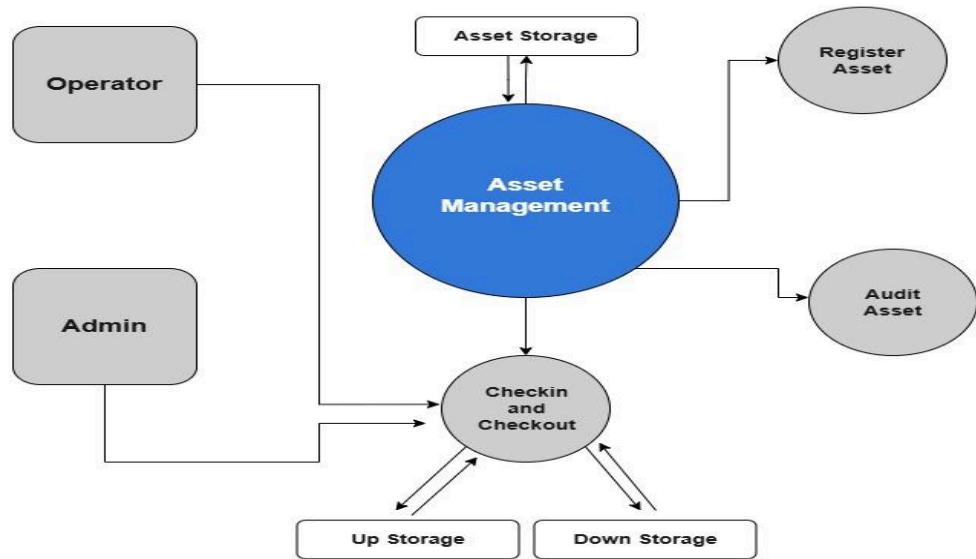


Figure #. Level 2+ MapAms Dataflow Diagram

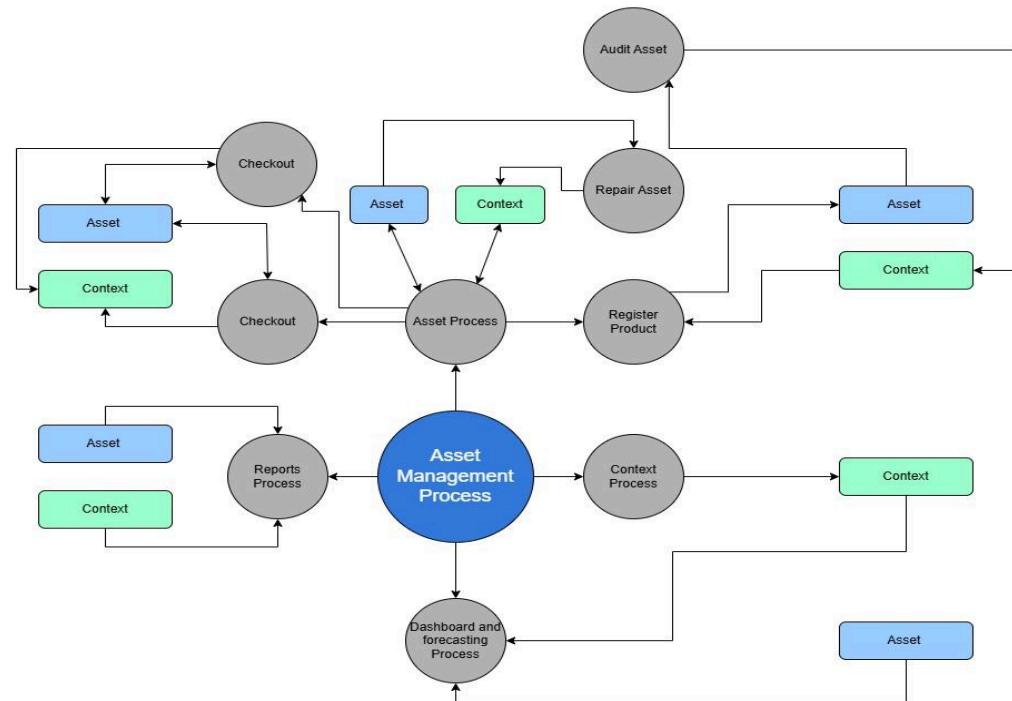
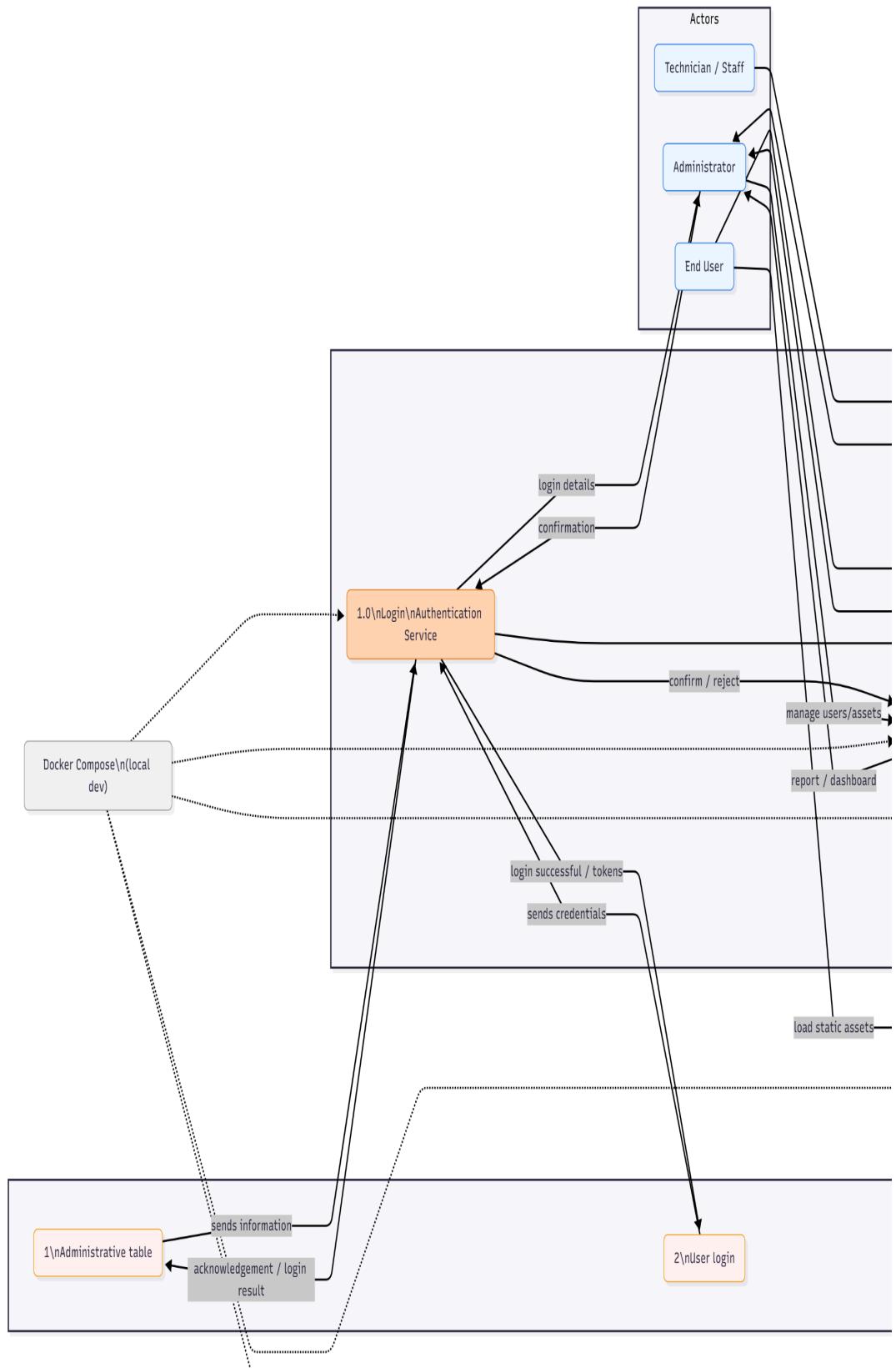
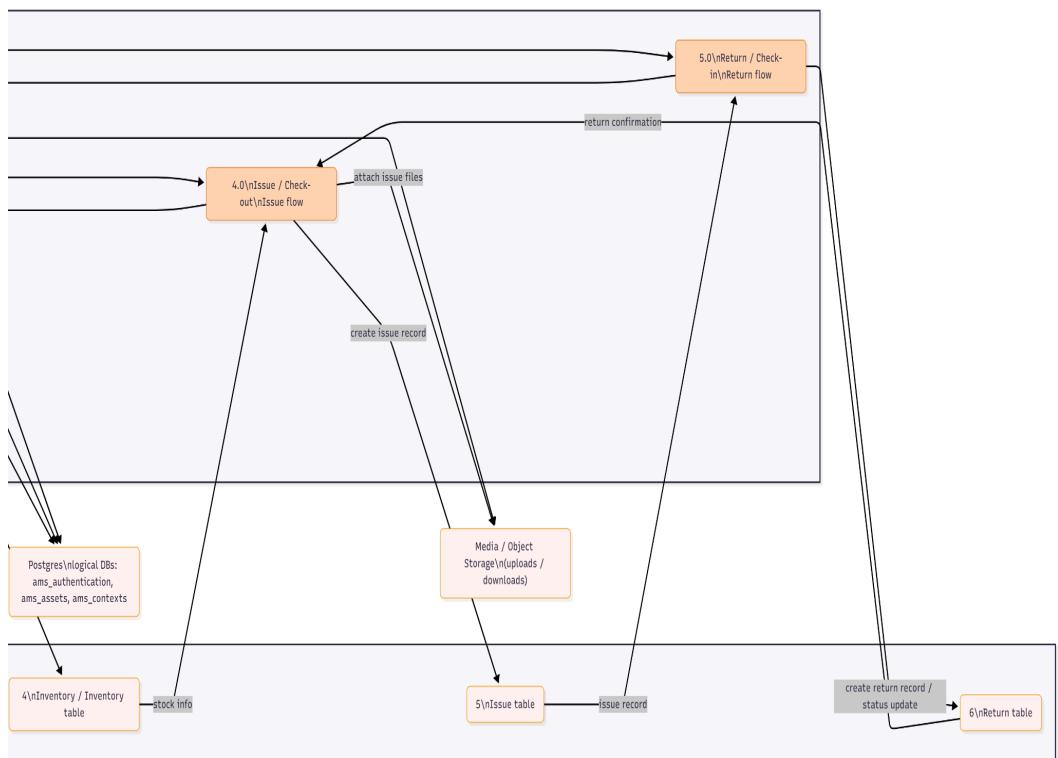
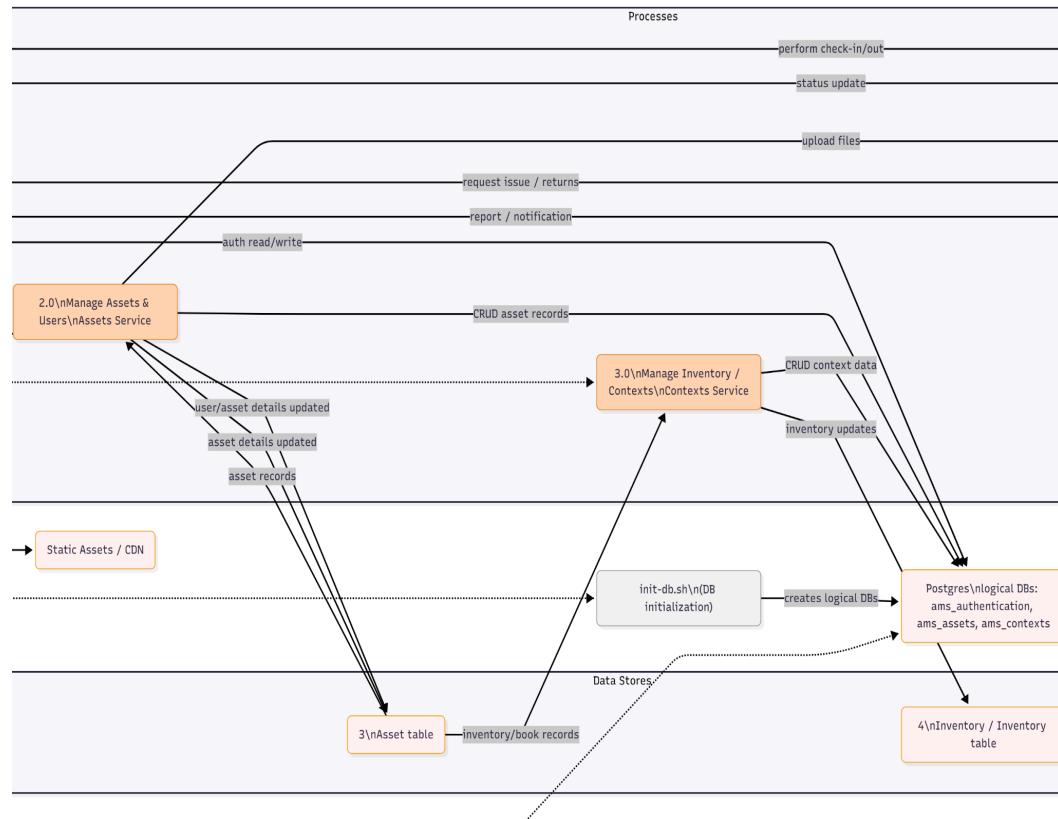


Figure #. Level 3 Dataflow Diagram





A.2 Information Systems Integrations

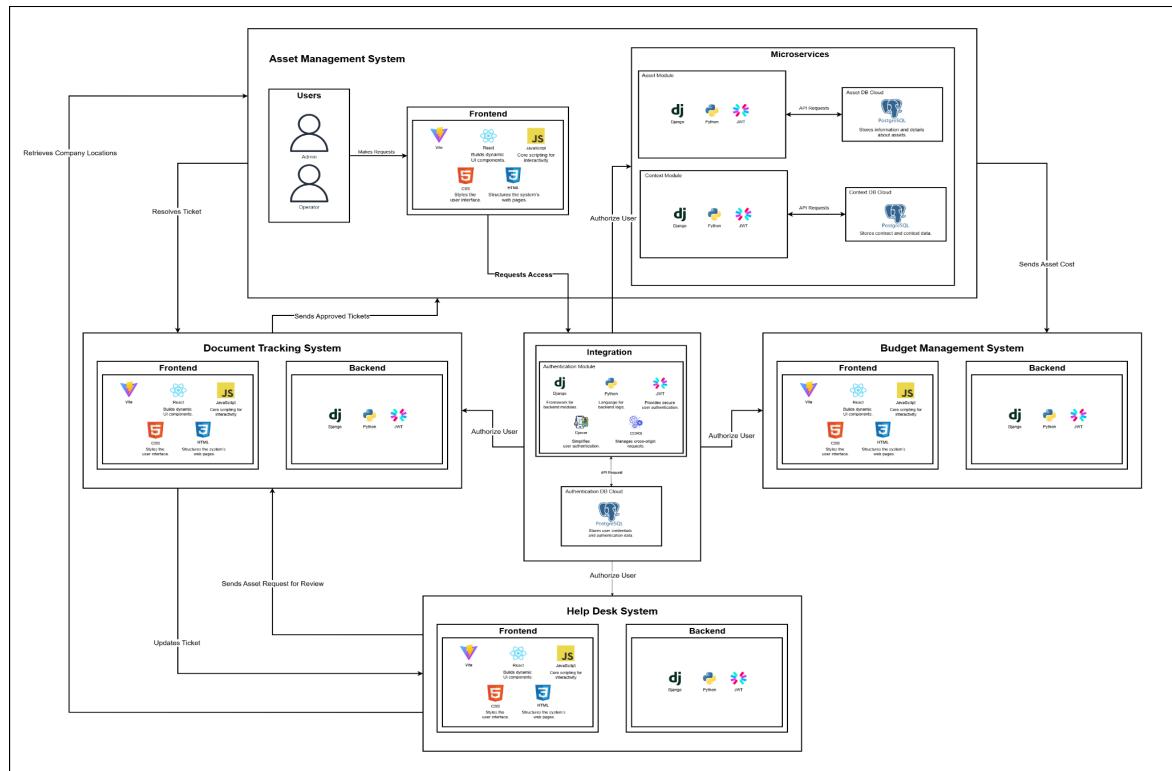
1. Overview

The MAP-AMS forms part of a larger enterprise ecosystem composed of multiple interconnected systems. These include:

- a. Centralized Authentication Service (CAS) - Shared login and authorization for all systems
- b. Help Desk System - Sends asset-related tickets to AMS
- c. Document Tracking System (DTS) - Approves and forwards ticket requests
- d. Budget Management System - Retrieves asset cost and depreciation data
- e. Microservice-Based Backend - Individual Django services for assets, contexts, authentication, contacts, and accessories

Integration is accomplished through secure REST APIs, shared authentication, and standardized JSON data formats.

2. Integration Architecture Diagram



The architecture shows AMS at the center, communicating with the following systems:

a. Central Authentication Service

The Central Authentication Service provides JWT tokens that allow secure user access across the organization's systems. It also maintains consistent role-based authorization for all platforms connected to the ecosystem. This service is used by the AMS, the Budget Management System, the Help Desk System, and the Document Tracking System (DTS) to ensure that all users operate under verified and unified credentials.

b. Help Desk System to DTS to AMS Workflow

In this workflow, the Help Desk System creates asset-related tickets that contain details about requests or issues. These tickets are forwarded to the Document Tracking System, which validates and approves the entries before they are processed. Once approved, the AMS automatically receives the ticket information and performs the necessary actions such as check-in, check-out, or repair updates. After processing, the AMS sends the updated status back to both the DTS and the Help Desk System to ensure that all systems maintain consistent and synchronized records.

c. Budget Management System

The AMS provides the Budget Management System with key financial information, including asset purchase cost, depreciation values, and end-of-life (EoL) estimates. The Budget Management System retrieves this information through designated GET endpoints, allowing it to generate financial reports and projections accurately.

d. Microservices (Internal Integration)

The AMS communicates with several internal Django-based microservices that include authentication, assets, contacts, context, and accessories. These microservices operate within an internal Docker network that allows secure and efficient communication between services. Each service exposes its own REST API, allowing a service-to-service model using standardized endpoint structures.

3. List of System Integration Points / APIs

a. Authentication Service

Purpose: Central login and RBAC

Integration Points:

Endpoint	Method	Description
/auth/login	POST	Validates user credentials and issues JWT
/auth/verify-token	GET	Checks token validity for service-to-service calls
/auth/users/<id>	GET	Retrieves user profile and role

b. Asset Management System (AMS)

Purpose: Provides asset, model, status, depreciation, and audit data used by external departments.

Endpoint	Method	Consumed by	Description

/assets/	GET	Help Desk, Budget System	Retrieves asset list and cost data
/assets/<id>	GET	Help Desk, DTS	Retrieves asset details for ticket validation

c. Document Tracking System (DTS)

Purpose: Ticket approval workflow

Outgoing Data	Description
AMS (POST /tickets/receive)	Sends validated ticket with asset ID, request type, requestor, location
Help Desk	Returns updated status after AMS resolves ticket

d. Help Desk System

Purpose: Sends maintenance or asset-movement tickets

Outgoing Data	Description
DTS	Ticket for approval
AMS	Approved asset tickets (ID, location, purpose, requester)

e. Budget Management System

Purpose: Fetches cost-related asset information

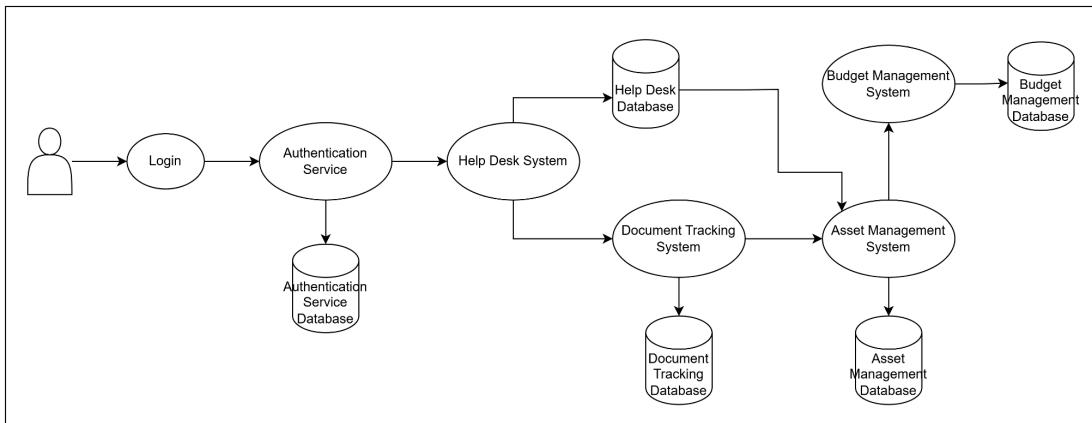
Endpoint	Method	Description
/assets/costs	GET	Retrieves purchase cost, depreciation, EoL
/reports/depreciation-summary	GET	Retrieves EoL forecasting and financial valuation

f. Internal Microservice APIs

Each microservice exposes its own endpoints:

- i. Context Service: categories, manufacturers, suppliers, depreciations, statuses
- ii. Authentication Service: user management, RBAC
- iii. Assets Service: models, components, repair, audit, check-in/out

4. Data Flow Diagrams (DFDs)



Key Flows:

- a. Users authenticate through CAS → CAS returns JWT → system grants access
- b. Help Desk → DTS → AMS ticket workflow
- c. AMS ↔ Budget Management for asset cost data
- d. Microservices exchange internal JSON data through Docker network

5. Data Mapping and Transformation Logic

- a. Ticket Data Mapping (Help Desk → DTS → AMS)

Field	Help Desk	DTS	AMS
Ticket ID	/	/	/
Asset ID	/	/	/
Request Type	/	/	/
Location	/	/	/
Approver		/	/
Status	Pending	Approved	In Progress → Completed

Transformation:

- i. DTS appends approval fields
- ii. AMS enriches with asset data, timestamps, checker information
- b. Asset Financial Data (AMS → Budget System)

AMS Field	Budget Mapping	Transformation
purchase_cost	cost_value	No transformation
depreciation_value	monthly_dep_rate	Rounds to 2 decimal

		places
expected_EOL	eol_date	Computed using depreciations model
category	cost_category	Lookup from Context Service

c. Internal Microservice Data Mapping

All services communicate through standardized API responses:

```
{
  "id": <integer>,
  "name": <string>,
  "created_at": "2025-01-01",
  "updated_at": "2025-01-01",
  "is_deleted": false
}
```

Transformations:

- i. Soft-deleted items are flagged but not removed
- ii. Pagination wrappers (page, total_pages, results[])
- iii. File uploads (images, documents) are converted to Base64 or stored with database references

6. Summary

The MAP-AMS integrates seamlessly with the organization's enterprise systems by leveraging centralized authentication, REST APIs, microservices, data transformation rules, and real-time workflow exchanges. This combination enables the system to

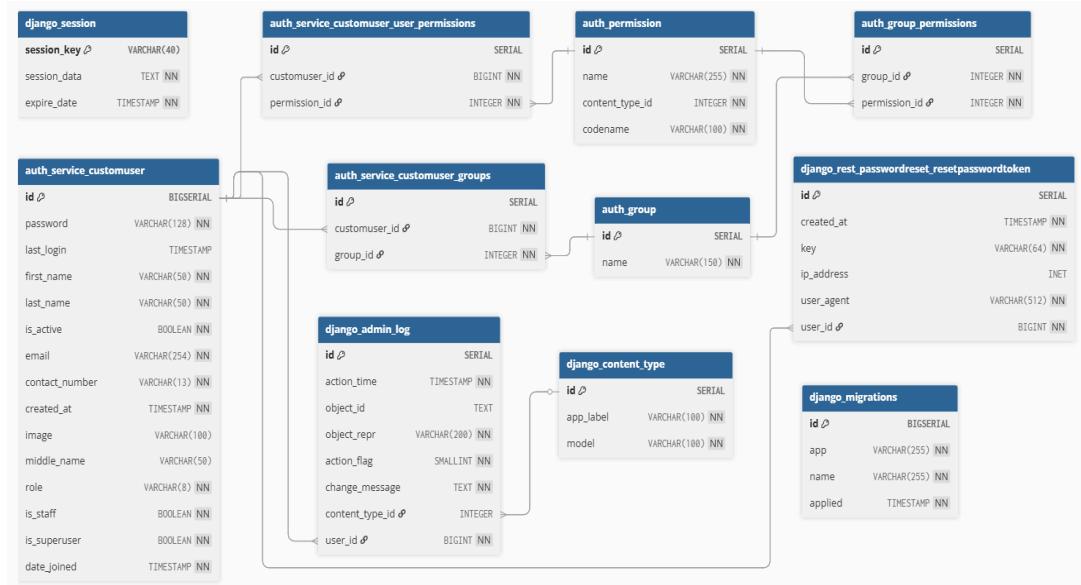
maintain consistent, secure, and reliable asset management processes across all departments, ensuring efficient coordination and data integrity throughout the organization.

A.3 - To be filled

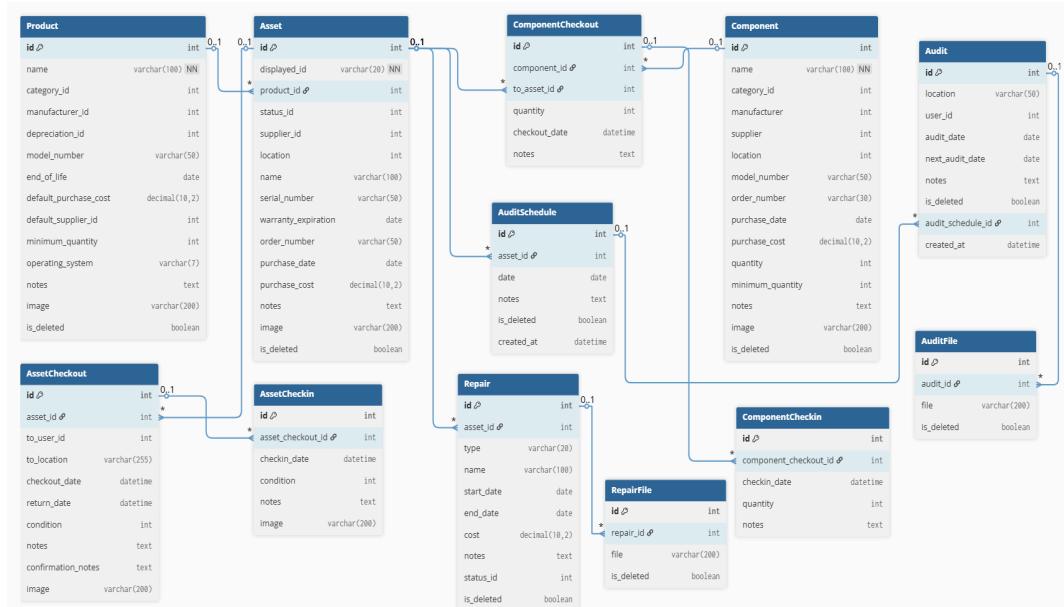
A.4 Database Schema and Data Management

1. Entity-Relationship Diagrams (ERDs)

a. Authentication Service



b. Assets Service



c. Contexts Service

Category	Status	Deprecation
id Ø int name varchar(50) NN type varchar(9) logo varchar(200) is_deleted boolean	id Ø int name varchar(50) NN type varchar(12) notes text is_deleted boolean	id Ø int name varchar(500) NN duration int minimum_value decimal(8,2) is_deleted boolean
Manufacturer	Supplier	
id Ø int name varchar(50) NN manu_url varchar(200) support_url varchar(200) support_phone varchar(13) support_email varchar(100) notes text logo varchar(200) is_deleted boolean	id Ø int name varchar(50) NN address varchar(100) city varchar(50) zip varchar(4) contact_name varchar(100) phone_number varchar(13) email varchar(100) URL varchar(200) notes text logo varchar(200) is_deleted boolean	

2. Data Schema

a. Authentication Service

Table	Key Columns / Relationships
auth_service_customuser	id (PK), email, password, role, is_active, is_staff, is_superuser, created_at, last_login
auth_group	id (PK), name
auth_permission	id (PK), name, content_type_id (FK), codename
auth_group_permissions	group_id (FK → auth_group), permission_id (FK → auth_permission)
auth_service_customuser_groups	customuser_id (FK → auth_service_customuser), group_id (FK → auth_group)
auth_service_customuser_permissions	customuser_id (FK → auth_service_customuser), permission_id (FK → auth_permission)
django_session	session_key (PK), session_data, expire_date

django_content_type	id (PK), app_label, model
django_admin_log	id (PK), user_id (FK → customuser), content_type_id (FK → content_type)
django_rest_passwordreset_resetpaswordtoken	id (PK), user_id (FK → customuser), key, created_at

b. Assets Service

Table	Key Columns / Relationships
Product	id (PK), name, category_id, manufacturer_id, depreciation_id, default_purchase_cost
Asset	id (PK), displayed_id, product_id (FK → Product), status_id, supplier_id, location, serial_number, purchase_date
AssetCheckout	id (PK), asset_id (FK → Asset), to_user_id, checkout_date, return_date
AssetCheckin	id (PK), asset_checkout_id (1:1 FK → AssetCheckout), checkin_date
Component	id (PK), name, category_id, supplier, quantity
ComponentCheckout	id (PK), component_id (FK → Component), to_asset_id (FK → Asset), quantity
ComponentCheckin	id (PK), component_checkout_id (FK → ComponentCheckout), quantity
Repair	id (PK), asset_id (FK → Asset), type, name, start_date, cost, status_id
RepairFile	id (PK), repair_id (FK → Repair), file
AuditSchedule	id (PK), asset_id (FK → Asset), date
Audit	id (PK), location, user_id, audit_schedule_id (1:1 FK → AuditSchedule), audit_date

AuditFile	id (PK), audit_id (FK → Audit), file
-----------	--------------------------------------

c. Contexts Service

Table	Key Columns / Relationships
Category	id (PK), name, type, logo, is_deleted
Supplier	id (PK), name, address, city, zip, contact_name, phone_number, email, url, notes, logo, is_deleted
Manufacturer	id (PK), name, manu_url, support_url, support_phone, support_email, notes, logo, is_deleted
Status	id (PK), name, type, notes, is_deleted
Depreciation	id (PK), name, duration, minimum_value, is_deleted

3. Data Dictionary

a. Authentication Service

Table Name	Field Name	Data Type	Constraints
auth_group	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(150)	Unique, Not Null
auth_permission	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(255)	Not Null
	content_type_id	INTEGER	Not Null
	codename	VARCHAR(100)	Not Null
	(content_type_id, codename)		Unique Index
auth_group_permissions	id	SERIAL	Primary Key, Auto Increment

	group_id	INTEGER	Not Null
	permission_id	INTEGER	Not Null
	(group_id, permission_id)		Unique Index
auth_service_customuser	id	BIGSERIAL	Primary Key, Auto Increment
	password	VARCHAR(128)	Not Null
	last_login	TIMESTAMP	Not Null
	first_name	VARCHAR(50)	Not Null
	last_name	VARCHAR(50)	Not Null
	is_active	BOOLEAN	Not Null
	email	VARCHAR(254)	Unique, Not Null
	contact_number	VARCHAR(13)	Not Null
	created_at	TIMESTAMP	Not Null
	image	VARCHAR(100)	Nullable
	middle_name	VARCHAR(50)	Nullable
	role	VARCHAR(8)	Not Null
	is_staff	BOOLEAN	Not Null
	is_superuser	BOOLEAN	Not Null
	date_joined	TIMESTAMP	Not Null
auth_service_customuser_groups	id	SERIAL	Primary Key, Auto Increment
	customuser_id	BIGINT	Not Null
	group_id	INTEGER	Not Null
	(customuser_id, group_id)		Unique Index
auth_service_customuser_user_permissions	id	SERIAL	Primary Key, Auto Increment

	customuser_id	BIGINT	Not Null
	permission_id	INTEGER	Not Null
	customuser_id, permission_id)		Unique Index
django_content_t ype	id	SERIAL	Primary Key, Auto Increment
	app_label	VARCHAR(100)	Not Null
	model	VARCHAR(100)	Not Null
	(app_label, model)		Unique Index
django_admin_lo g	id	SERIAL	Primary Key, Auto Increment
	action_time	TIMESTAMP	Not Null
	object_id	TEXT	Nullable
	object_repr	VARCHAR(200)	Not Null
	action_flag	SMALLINT	Not Null
	change_message	TEXT	Not Null
	content_type_id	INTEGER	Nullable
	user_id	BIGINT	Not Null
django_migratio ns	id	BIGSERIAL	Primary Key, Auto Increment
	app	VARCHAR(255)	Not Null
	name	VARCHAR(255)	Not Null
	applied	TIMESTAMP	Not Null
django_rest_pass wordreset_resetp asswordtoken	id	SERIAL	Primary Key, Auto Increment
	created_at	TIMESTAMP	Not Null
	key	VARCHAR(64)	Unique, Not Null
	ip_address	INET	Nullable

	user_agent	VARCHAR(512)	Not Null
	user_id	BIGINT	Not Null
django_session	session_key	VARCHAR(40)	Primary Key
	session_data	TEXT	Not Null
	expire_date	TIMESTAMP	Not Null

b. Assets Service

Table Name	Field Name	Data Type	Constraints
Depreciation	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(500)	Not Null
	duration	INTEGER	Not Null
	minimum_value	DECIMAL(8,2)	Not Null
	is_deleted	BOOLEAN	Default False
AssetCategory	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(50)	Not Null
	type	VARCHAR(5)	Default "Asset"
	logo	VARCHAR(100)	Nullable
	is_deleted	BOOLEAN	Default False
Product	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(100)	Not Null
	category_id	INTEGER	Foreign Key → asset_category(id), On Delete CASCADE
	manufacturer_id	INTEGER	Positive integer
	depreciation_id	INTEGER	Foreign Key → depreciation(id),

			On Delete PROTECT
	model_number	VARCHAR(50)	Nullable
	end_of_life	DATE	Nullable
	default_purchase_cost	DECIMAL(10,2)	Nullable
	default_supplier_id	INTEGER	Nullable
	minimum_quantity	INTEGER	Default 1
	operating_system	VARCHAR(7)	Nullable, Choices: linux, windows, macos, ubuntu, centos, debian, fedora, other
	notes	TEXT	Nullable
	image	VARCHAR(100)	Nullable
	is_deleted	BOOLEAN	Default False
Status	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(50)	Not Null
	type	VARCHAR(12)	Not Null, Choices: deployable, deployed, undeployable, pending, archived
	notes	TEXT	Nullable
	is_deleted	BOOLEAN	Default False
Asset	id	SERIAL	Primary Key, Auto Increment
	displayed_id	VARCHAR(20)	Unique, Auto-generated
	product_id	INTEGER	Foreign Key → product(id), On

			Delete CASCADE
	status_id	INTEGER	Foreign Key → status(id), On Delete CASCADE
	supplier_id	INTEGER	Not Null
	location	INTEGER	Not Null
	name	VARCHAR(100)	Not Null
	serial_number	VARCHAR(50)	Nullable
	warranty_expiration	DATE	Nullable
	order_number	VARCHAR(50)	Nullable
	purchase_date	DATE	Nullable
	purchase_cost	DECIMAL(10,2)	Nullable
	notes	TEXT	Nullable
	image	VARCHAR(100)	Nullable
	is_deleted	BOOLEAN	Default False
AssetCheckout	id	SERIAL	Primary Key, Auto Increment
	asset_id	INTEGER	Foreign Key → asset(id), On Delete CASCADE
	to_user_id	INTEGER	Not Null
	to_location	VARCHAR(255)	Not Null
	checkout_date	TIMESTAMP	Auto Now Add
	return_date	TIMESTAMP	Not Null
	condition	SMALLINT	Not Null, 1–10
	notes	TEXT	Nullable
	confirmation_notes	TEXT	Nullable
	image	VARCHAR(100)	Nullable

AssetCheckin	id	SERIAL	Primary Key, Auto Increment
	asset_checkout_id	INTEGER	One-to-One → asset_checkout(id)
	checkin_date	TIMESTAMP	Not Null
	condition	SMALLINT	Not Null, 1–10
	notes	TEXT	Nullable
	image	VARCHAR(100)	Nullable
ComponentCategory	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(50)	Not Null
	type	VARCHAR(5)	Default "Component"
	logo	VARCHAR(100)	Nullable
	is_deleted	BOOLEAN	Default False
Component	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(100)	Not Null
	category_id	INTEGER	Foreign Key → component_category(id), On Delete CASCADE
	manufacturer	INTEGER	Foreign Key → status(id), On Delete CASCADE
	supplier	INTEGER	Not Null
	location	INTEGER	Not Null
	model_number	VARCHAR(50)	Nullable
	order_number	VARCHAR(30)	Nullable
	purchase_date	DATE	Auto Now Add
	purchase_cost	DECIMAL(10,2)	Not Null

	quantity	INTEGER	Default 1
	minimum_quantity	INTEGER	Default 0
	notes	TEXT	Nullable
	image	VARCHAR(100)	Nullable
	is_deleted	BOOLEAN	Default False
ComponentCheckout	id	SERIAL	Primary Key, Auto Increment
	component_id	INTEGER	Foreign Key → component(id)
	to_asset_id	INTEGER	Foreign Key → asset(id)
	quantity	INTEGER	Default 1
	checkout_date	TIMESTAMP	Auto Now Add
	notes	TEXT	Nullable
ComponentCheckin	id	SERIAL	Primary Key, Auto Increment
	component_checkout_id	INTEGER	Foreign Key → component_checkout(id)
	checkin_date	TIMESTAMP	Auto Now Add
	quantity	INTEGER	Default 1
	notes	TEXT	Nullable
Repair	id	SERIAL	Primary Key, Auto Increment
	asset_id	INTEGER	Foreign Key → asset(id)
	type	VARCHAR(20)	Not Null, Choices: maintenance, repair, upgrade, test, hardware, software
	name	VARCHAR(100)	Not Null

	start_date	DATE	Default Now
	end_date	DATE	Nullable
	cost	DECIMAL(10,2)	Not Null
	notes	TEXT	Nullable
	status_id	INTEGER	Foreign Key → status(id)
	is_deleted	BOOLEAN	Default False
RepairFile	id	SERIAL	Primary Key, Auto Increment
	repair_id	INTEGER	Foreign Key → repair(id)
	file	VARCHAR(100)	Not Null
	is_deleted	BOOLEAN	Default False
AuditSchedule	id	SERIAL	Primary Key, Auto Increment
	asset_id	INTEGER	Foreign Key → asset(id)
	date	DATE	Not Null
	notes	TEXT	Nullable
	is_deleted	BOOLEAN	Default False
	created_at	TIMESTAMP	Default Now, Not Editable
Audit	id	SERIAL	Primary Key, Auto Increment
	location	VARCHAR(50)	Not Null
	user_id	INTEGER	Not Null
	audit_date	DATE	Default Now
	next_audit_date	DATE	Default Now
	notes	TEXT	Nullable
	is_deleted	BOOLEAN	Default False

	audit_schedule_id	INTEGER	One-to-One → audit_schedule(id)
	created_at	TIMESTAMP	Default Now, Not Editable
AuditFile	id	SERIAL	Primary Key, Auto Increment
	udit_id	INTEGER	Foreign Key → audit(id)
	file	VARCHAR(100)	Not Null
	is_deleted	BOOLEAN	Default False

c. Contexts Service

Table Name	Field Name	Data Type	Constraints
Supplier	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(50)	Not Null
	address	VARCHAR(100)	Nullable
	city	VARCHAR(50)	Nullable
	zip	VARCHAR(4)	Nullable
	contact_name	VARCHAR(100)	Nullable
	phone_number	VARCHAR(13)	Nullable
	email	VARCHAR(254)	Nullable
	URL	VARCHAR(200)	Nullable
	notes	TEXT	Nullable
	logo	VARCHAR(100)	Nullable

	is_deleted	BOOLEAN	Default False
Manufacturer	id	SERIAL	Primary Key, Auto Increment
	name	VARCHAR(50)	Not Null
	manu_url	VARCHAR(200)	Nullable
	support_url	VARCHAR(200)	Nullable
	support_phone	VARCHAR(13)	Nullable
	support_email	VARCHAR(254)	Nullable
	notes	TEXT	Nullable
	logo	VARCHAR(100)	Nullable
	is_deleted	BOOLEAN	Default False

4. CRUD Operations

a. Create ©

Creating data in the system involves adding new records to the respective tables for each microservice. For example, new users can be added to the Authentication Service by creating entries in the auth_service_customuser table with required fields such as email, password, and role. In the Assets Service, new products or assets are created by inserting records into the Product and Asset tables, respectively, including references to categories, manufacturers, depreciation methods, suppliers, and initial status. Similarly, the Contexts Service allows the creation of reference data such as Supplier, Manufacturer, and Category entries, which can later be referenced by other services. The Django ORM handles validation and integrity checks automatically during record creation.

b. Read ®

Reading or retrieving data is performed whenever the system or users need to access existing records. For instance, the Authentication Service allows fetching user profiles, permissions, and group memberships through queries on the auth_service_customuser, auth_group, and related tables. In the Assets Service, assets, components, checkouts, repairs, and audits can be queried based on various criteria such as asset status, location, or category. The Contexts Service provides read access to suppliers, manufacturers, and categories, which are referenced by other microservices via API calls. These read operations ensure that the system can efficiently present accurate and up-to-date information for reporting, dashboards, and transactional processing.

c. Update (U)

Updating data modifies existing records while preserving their unique identifiers. In the Authentication Service, user information such as name, contact details, or role can be updated as needed, while permissions and group memberships are adjusted through junction tables. The Assets Service allows updating product details, asset status, component quantities, repair logs, and audit outcomes to reflect real-world changes in asset condition, deployment, or maintenance history. The Contexts Service supports updates to supplier, manufacturer, or category information, ensuring that reference data remains current. Django ORM ensures that updates respect foreign key relationships and constraints to maintain data integrity across tables.

d. Delete (D)

Deleting data can be performed as either a soft deletion or hard deletion, depending on the table and business rules. Most tables use a `is_deleted` flag for soft deletion, preserving historical records for auditing or rollback purposes.

For example, assets, components, categories, suppliers, and manufacturers can be marked as deleted without physically removing them from the database. In contrast, some temporary records such as password reset tokens or session data may be deleted permanently once they expire or are no longer needed. This combination of soft and hard deletion ensures both operational flexibility and compliance with traceability requirements.

A.5 Network Configuration

1. Overview

The network configuration of MAP-AMS describes how the system is hosted and secured, emphasizing connectivity, traffic flow, and foundational security mechanisms. The design ensures that all data transactions between users, frontend, backend, and databases are encrypted, authenticated, and controlled to prevent unauthorized access and maintain data integrity, confidentiality, and availability.

2. Network Topology Diagram

MAP-AMS uses a layered architecture separating the React frontend, Django backend microservices (Authentication, Assets, Contexts), and PostgreSQL databases. External users access the system exclusively through the frontend, which communicates with backend services over HTTPS. Backend services interact internally with the database through isolated network channels. In production, Railway Cloud provides network isolation, firewall rules, and traffic monitoring, while CI/CD pipelines ensure secure deployment and consistent configuration across all services.

3. IP Address Schemes and VLANs

- a. Each backend microservice and database is assigned internal addresses within Railway Cloud's virtual network.
- b. The frontend uses public IPs for client access over HTTPS.

- c. Internal communication is logically segmented to prevent direct exposure of databases to external traffic, supporting VLAN-style isolation for backend and database traffic.

4. Firewall Rules

- a. Only the frontend's public endpoints are exposed externally.
- b. Backend services accept connections only from authenticated frontend requests or other authorized services.
- c. Database connections are restricted to backend microservices, blocking direct access from external networks.
- d. Railway Cloud firewall monitors and blocks unauthorized access attempts and suspicious traffic patterns.

5. Communication Protocols

- a. HTTPS is used for all client-to-frontend and frontend-to-backend requests.
- b. JWT authentication secures and validates each user request.
- c. Internal service-to-service communication uses HTTP within the isolated network, and database connections use SSL encryption.
- d. SSH is enabled for secure administrative access to services when needed.
- e. Messaging or asynchronous tasks, if implemented, can use protocols like AMQP or similar message brokers.

6. Additional Security Measures

- a. Backend Security: Django implements password hashing, Cross-Site Request Forgery (CSRF) protection, and Role-Based Access Control (RBAC) to limit user privileges
- b. Database Security: PostgreSQL enforces access control and accepts connections only from authenticated backend services

- c. Cloud Security: Railway Cloud provides network isolation, firewall protection, automated monitoring, and scalable hosting to ensure operational continuity
- d. CI/CD Pipeline: GitHub-triggered deployment ensures secure automated updates and reduces human error

7. Summary

This network configuration ensures that MAP-AMS operates in a secure, isolated, and resilient environment. External users interact only with the frontend. Backend services and databases are protected through controlled access, firewall rules, and secure protocols. Layered defenses, combined with Railway Cloud infrastructure and JWT authentication, provide a reliable foundation for safe and efficient asset management operations.

A.6 - To be filled

A.9 System Monitoring and Maintenance

1. Overview

The MAP Active IT Asset Management System with Forecasting (MapAms) is designed for long-term operational stability at MAP Active Philippines. After deployment, the system will follow a structured monitoring, logging, backup, and disaster recovery plan to ensure high availability, quick issue resolution, and minimal downtime.

2. Monitoring Tools and Strategies

Component	Tool/Method	Purpose
Error & Exception Tracking	Sentry (integrated with Laravel)	Real-time error alerts, stack traces, performance bottlenecks, and release tracking
Application Performance	Laravel Telescope (production mode) + Laravel Pulse	Monitors requests, slow queries, cache hits, queue performance, and custom metrics
Server Health	UptimeRobot + custom cron health check	HTTP ping every 5 minutes, SSL expiry alerts, and server resource monitoring (CPU, RAM, Disk)
Queue & Scheduled Tasks	Laravel Horizon	Monitors queue workers, failed jobs, and job throughput
Custom Alerts	Mail/Slack notifications via Laravel	Immediate alerts for critical events (failed login attempts, backup failure, high error rate)

3. Key Performance Metrics and Alert Thresholds

Metric	Alert Threshold	Notification Channel
CPU Usage	> 80% for 5 minutes	Slack + Email
Memory Usage	> 85%	Slack + Email

Disk Usage	> 90%	Slack + Email
HTTP 5xx Error Rate	> 5 errors/minute	Slack + Email
Average Response Time	> 2 seconds	Slack
Database Slow Queries	> 1 second	Sentry + Email
Failed Queue Jobs	> 3 failed jobs	Horizon + Slack
Backup Failure	Any failure	Email (urgent)

4. Logging Strategy

- a. All logs are handled by Laravel Monolog with daily rotation.
- b. Log channels: daily (default), slack (critical errors), sentry (exceptions).
- c. Logs are stored in /storage/logs and retained for 60 days.
- d. Sensitive data (passwords, tokens) are automatically filtered.
- e. Audit logs (user actions on assets, repairs, audits) are stored permanently in the database (audit_logs table).

5. Backup Procedures

- a. Automated daily backups at 2:00 AM (Philippine time) using spatie/laravel-backup.
- b. Backup scope: MySQL database + uploaded files (asset photos, documents).
 - i. Backup destinations:
 - ii. Local server (/backups)
 - iii. Google Drive (via spatie/laravel-google-drive-adapter)
 - iv. External HDD (weekly manual copy by IT staff)
- c. Backup retention: 30 daily, 12 monthly, 2 yearly.
- d. Backup verification: Weekly cleanup job verifies the latest backup can be restored successfully.

6. Disaster Recovery Plan

- a. Detection - Automated alert via monitoring tools or user report.
- b. Assessment - IT Head (Mr. Elvin Punzalan) or designated admin assesses severity.
- c. Recovery Steps:
 - i. Minor issue (bug, failed job) → Deploy fix from Git within 4 hours.
 - ii. Database corruption → Restore from latest healthy backup (estimated downtime: 30-60 minutes).
 - iii. Server failure → Switch to backup server or restore on new instance using latest backup + Git pull (estimated RTO: 4 hours).
 - iv. Complete data center loss → Restore from Google Drive backups to new hosting (estimated RTO: 12–24 hours).
- d. Post-recovery - Full system test, user notification, and incident report.
- e. Recovery Time Objective (RTO): 4 hours (critical functions)
- f. Recovery Point Objective (RPO): Maximum 24 hours data loss

A.10 APIs and Integration Points

1. Overview

The MAP Active IT Asset Management System with Forecasting (MapAms) is built on a microservices architecture using Django REST Framework (DRF). Each domain is deployed as an independent service with its own PostgreSQL database. All services expose RESTful JSON APIs directly at the root path. Interactive OpenAPI 3 documentation generated by drf-spectacular is available at the root URL of every service.

2. Production Base URLs

Service	URL	Port (Development)	Purpose
Frontend	https://mapams.up.railway.app	http://localhost:8000	Main user interface

Authentication Service	https://auth-mapams.up.railway.app	http://localhost:8001	Authentication, JWT issuance, user management
Assets Service	https://assets-mapams.up.railway.app	http://localhost:8002	Core asset, checkout/checkin, repair, audit
Contexts Service	https://contexts-mapams.up.railway.app	http://localhost:8003	Categories, suppliers, manufacturers, statuses, depreciation

3. Authentication Flow

- a. Only the Auth Service exposes login endpoints:
 - i. POST <https://auth-mapams.up.railway.app/token/> → returns JWT access and refresh tokens
 - ii. POST <https://auth-mapams.up.railway.app/token/refresh/>
- b. All subsequent requests to Assets and Contexts services must include:
 - i. Authorization: Bearer <access_token>
 - ii. Accept: application/json

4. Endpoints

a. Authentication Service

Endpoint	HTTP Methods	Description
/users/	GET	List all users (Admin only)
/users/<id>/	GET	Retrieve user profile & role

b. Assets Service

Endpoint	HTTP Methods	Description	Required Role

/assets/	GET, POST, PUT, PATCH, DELETE	Full asset management, filtering, search, pagination	Authenticated
/asset-checkout/	GET, POST	Assign asset to user/location	IT Staff

c. Contexts Service

Endpoint	HTTP Methods	Description
/categories/	GET, POST, PUT, PATCH, DELETE	Full asset management, filtering, search, pagination
/suppliers/	GET, POST, PUT, PATCH, DELETE	Supplier master data

5. Error

a. Error Handling

Follows standard HTTP status codes with detailed DRF messages (400, 401, 403, 404, 422, 500).

b. Error Codes

- i. 200/201 - Success
- ii. 400 - Bad Request
- iii. 401 - Unauthorized
- iv. 403 - Forbidden
- v. 404 - Not Found
- vi. 422 - Validation Error (returns field errors)
- vii. 500 - Server Error

6. Deployment and Maintenance on Railway

a. Each service is linked to its own GitHub repository branch

- b. Automatic redeploy on push to main
- c. Environment variables managed via Railway dashboard (SECRET_KEY, DATABASE_URL, etc.)
- d. Zero-downtime deploys enabled
- e. Railway provides built-in metrics, logs, and usage-based billing

7. Future Integration Points

- a. Custom domain mapping (mapams.mapactive.ph → mapams.up.railway.app)
- b. Mobile app consuming the same root-level APIs
- c. Webhook notifications to external helpdesk/ticketing system
- d. Power BI direct link via PostgreSQL read-replica (future)

A.11 System User Manual

This manual provides instructions on how to use the main modules of the system, presented in the following order: Login, Dashboard, Assets, Asset Models, Components, Repairs, Tickets, Reports, Context, Profile, and User Management.

1. Login

1.1 Accessing the Login Page

1. Open your browser and navigate to
2. the system URL provided by your
3. administrator.
4. You will see the login form with fields
5. for Email/Username and Password.

1.2 Logging In

1. Enter your registered email/username.
2. Enter your password.
3. Click the Login button.
4. If your credentials are correct, you will be redirected to the Dashboard.
5. If login fails:
 - Check for typos in email and password.
 - If you still cannot log in, contact your system administrator.

The image shows a laptop on a desk with a white potted plant. The laptop screen displays the 'ASSET MANAGEMENT SYSTEM' dashboard. The dashboard features a table of asset data, a pie chart showing the distribution of assets, a bar chart for asset value, and a status summary. To the right of the laptop is a separate window showing the login interface for 'MapAMS'. The interface includes fields for Email and Password, a 'Log In' button, and a 'Forgot Password?' link.

ASSET MANAGEMENT SYSTEM

ID	Name	Category	Status
A1001	Laptop	Laptop	Active
A1002	Smart TV	Printer	Active
A1003	Office Chair	Printer	Active
A1002	Projector	Projector	Inactive
A1008	Projector	Laptop	Inactive

ASSETS
\$1,25000

STATUS

MAP active MapAMS

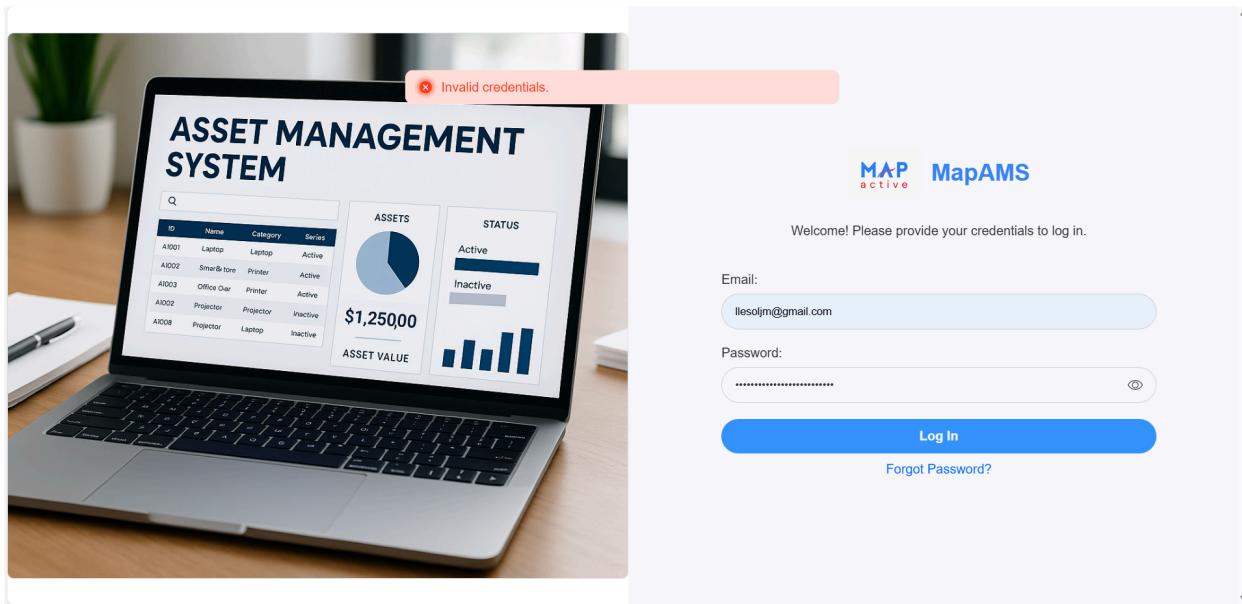
Welcome! Please provide your credentials to log in.

Email:

Password:

Log In

[Forgot Password?](#)



2. Dashboard Module

2.1 Overview

The Dashboard is the first screen after login. It gives a quick summary of system status and key metrics using cards, charts, and tables.

Category	Count	Details
Due for Return	5	
Upcoming Audits	3	
Upcoming End of Life	2	
Expiring Warranties	1	
Overdue for Return	0	
Overdue Audits	1	
Reached End of Life	0	
Expired Warranties	0	
Low Stock	2	

2.2 Dashboard Page

1. Navigation bar at the top:

- Links to modules such as Assets, Components, Repairs, Tickets, Reports, Context, Profile, and User Management.

The screenshot shows the MapAMS application interface. At the top left is the logo 'MapAMS' with 'active' written below it. To the right are navigation links: Dashboard, Assets (with a dropdown arrow), Repairs, Audit, Tickets, Reports (with a dropdown arrow), More (with a dropdown arrow), and a user profile icon. A notification badge with the number '1' is visible above the user profile icon.

2. Dashboard cards

The screenshot shows the 'Dashboard' cards section. It features ten rounded rectangular cards arranged in two rows of five. Each card contains a small blue or red square with a white number (e.g., 5, 3, 2, 1, 0, 2) and a corresponding status message. The cards are: Due for Return (5), Upcoming Audits (3), Upcoming End of Life (2), Expiring Warranties (1), Low Stock (2), Overdue for Return (0), Overdue Audits (1), Reached End of Life (0), Expired Warranties (0).

3. KPI cards:

- Show counts such as total assets, in-use assets, items under repair, open tickets, etc.

The screenshot shows the 'Forecast Insights' section. It contains three large rounded rectangular cards: 'FORECASTED TOTAL DEMAND' (138 units, +12.0%), 'MOST REQUESTED ASSET MODEL' (MacBook Pro model, +8.0%), and 'EXPECTED SHORTAGE RISK' (Low status, -5.0%).

4. Forecasting & Charts (Admin):

- Asset status forecast (historical + forecast).
- Product or component demand forecast.
- Hover over data points to see tooltips.

5. Total Asset Cost and Asset Utilization

- Shows cost-related insights and how assets are being used across the organization.

6. Pie Charts

- Visual summaries showing Asset categories and Asset Statuses.

3. Assets Module

3.1 Navigating to Assets

1. From the navigation bar, click Assets and it will show the main Assets table.

3.2 Assets List View

1. Main components:

- Table header: title, search box, filter button, export button.
- Assets table: rows of assets with columns such as Image, Asset Tag, Name, Category, Status, Location, etc.
- Row actions: View, Edit, Check In/Check Out, Delete (soft delete).
- Pagination at the bottom: change page and items per page.

3.3 Searching and Filtering

1. Search:

- Use the Search field to find assets by name, category, or other key fields.
- Results update to show only matching assets.

2. Filter:

- Click the Filter button to open the Asset Filter modal.
- Select criteria such as category, status, location, date ranges, etc.

- Click Apply to filter the list.
- Use Reset to clear all filters.

3.4 Creating a New Asset

1. Click the New Asset button.
2. Fill up the form, especially the required fields (indicated by red asterisks).
3. Press the upload button to add an image (respecting the max file size).
4. Click Save to create the asset.
5. If the form is incomplete, the Save button may be disabled until required fields are filled.

3.5 Viewing an Asset

1. In the Assets table, click the View icon in the Actions column.
2. You'll be taken to the Asset View page, which uses a tabbed layout. The Asset View page includes the following tabs:
 - Asset Details
 - Additional Fields
 - History
 - Checkout History
 - Components
 - Repair
 - Audits

- Attachments
3. Use the tabs to navigate between sections.

3.6 Editing an Asset

1. From the Assets table, click the Edit icon.
2. Update the fields as needed.
3. Click Save to apply changes.
4. You may see a success message indicating that the asset was updated.

3.7 Check In / Check Out

1. In the Assets table:
 - Use the Check Out button to assign an asset to a user or location.
 - Use the Check In button to return it to inventory.
2. When performing a **Check In**, you will be directed to the Check-In page, where you need to provide the following information:

3. Check-In Details Form

- Check-In Date
- Status
- Condition
- Location
- Notes

3.8 Deleting Assets (Soft Delete)

1. Click the Delete icon in the Actions column.
2. A confirmation modal will appear.
3. Confirm to soft-delete the asset (it will not be permanently removed)
4. A success message will confirm the delete action.

4. Asset Model Modules

4.1 Purpose

Asset Models define reusable templates for physical assets (e.g. "Dell Laptop 14", "HP Laser Printer").

4.2 Accessing Asset Models

1. From the navigation bar or a "More/Settings" area, choose Asset Models (or equivalent).

4.3 Asset Models List

1. The table consist of:
 - Columns: Model Name, Category, Manufacturer, Depreciation, etc.
 - Actions: View, Edit, Delete.

4.4 Searching and Filtering

3. Search:

- Use the Search field to find asset models by name or other key fields.
- Results update to show only matching assets.

4. Filter:

- Click the Filter button to open the Asset Modal Filter modal.
- Select criteria
- Click Apply to filter the list.
- Use Reset to clear all filters.

4.5 Creating a New Asset Model

6. Click the New button.
7. Fill up the form, especially the required fields (indicated by red asterisks).
8. Press the upload button to add an image (respecting the max file size).
9. Click Save to create the asset.
10. If the form is incomplete, the Save button may be disabled until required fields are filled.

4.6 Viewing an Asset

1. In the Asset Model table, click the View icon in the Actions column.
2. You'll be taken to the Asset Model View page, which uses a tabbed layout that includes the following tabs:
 - Asset Model Details
 - Additional Fields

- Assets
3. Use the tabs to navigate between sections.

4.7 Editing an Asset

1. From the Assets Model table, click the Edit icon.
2. Update the fields as needed.
3. Click Save to apply changes.
4. You may see a success message indicating that the asset model was updated.

4.8 Deleting Asset Model (Soft Delete)

1. Click the Delete icon in the Actions column.
2. A confirmation modal will appear.
3. Confirm to soft-delete the asset (it will not be permanently removed)
4. A success message will confirm the delete action.

A.13 Version Control and Source Code Repository

Repository URL: <https://github.com/ElisaGarrote/Capstone1.git>

1. Branching Guide:

Table #

Repository Branching

Branch Type	Purpose	Format / Example
Develop	Active development before release	develop
Release	Final testing and merging into production	release/<version> e.g., release/1.0.0
Test	Experimental changes or QA testing	test/<description>
Issue	Bug/defect fixes or improvements assigned by QA/PM	issue/<ticket-id>-<short-description> e.g., issue/123-fix-login-redirect
Hotfix	Urgent production fixes	hotfix/<ticket-id>-<short-description> e.g., hotfix/301-payment-failure

2. Tagging / Releases

- Semantic versioning is used: vMAJOR.MINOR.PATCH

Example: v1.2.0, v2.0.3

- Tags are applied to commits merged into main after successful testing.

3. Commit Guidelines

- Commit messages should be short, descriptive, and optionally scoped.

Example formats:

- i. feat(auth): add JWT refresh token

- ii. fix(assets): correct depreciation calculation
 - iii. chore(contexts): update supplier fixtures
- b. Maintain consistent style to facilitate history readability.

4. Code Review

- a. PR Summary & Scope: Confirm the PR description clearly states purpose, linked issue/ticket, and which services/files are affected (e.g., assets, authentication, frontend).
- b. Small & Focused: Ensure changes are small and limited to the stated scope; large refactors should be split or justified.
- c. Correctness / Business Logic: Verify the change implements the intended behavior (API shape, validations, state transitions) and that edge cases are handled.
- d. Django Patterns & Conventions: For backend code, check that Django patterns are followed: use of models, serializers, query optimization (avoid N+1), appropriate model managers, and correct usage of settings.py and env variables.
- e. Migrations: Confirm DB migrations are included when models change, migration names are clear, and migrations run cleanly (no accidental destructive operations).
- f. Database Access Safety: Look for raw SQL or unparameterized queries; prefer Django ORM or parameterized DB calls; verify transaction use where needed.
- g. API Contract / Schema: Verify REST endpoints' request/response formats, status codes, and authentication/authorization requirements — update API docs if contract changes.

- h. Authentication & Authorization: Ensure endpoints enforce correct permissions (DRF permission classes), token handling is secure, and no endpoints unintentionally allow AllowAny unless intended.
- i. Input Validation & Error Handling: Check serializers/forms validate input and errors are returned consistently (no 500s for client errors).
- j. Security Checks: Validate secrets are not hard-coded, .env usage is correct, no sensitive logs printed, CSRF/CORS settings are safe for production, and dependency CVEs are addressed.
- k. Frontend Practices: For frontend, check component structure, state handling (avoid unnecessary re-renders), proper use of react-router, form validation, and safe handling of tokens (no storing tokens in localStorage if not intended).
- l. Network & CORS: Confirm CORS and CSRF trusted origins are configured appropriately (see settings.py), and cross-service calls use secure channels.
- m. Static & Media Handling: Verify static files are built by frontend and either served by WhiteNoise or configured for CDN/object storage; check any media upload handling for size/type checks.
- n. Docker & Dev Config: Check Dockerfile and docker-compose.dev.yml updates volumes, ports, env files and ensure they don't expose secrets or dev-only settings in production images.
- o. Init & Startup Scripts: Review init-db.sh or other init scripts for idempotency, safety, and correct DB creation; confirm they won't re-create or drop production data.

- p. Tests: Confirm unit and integration tests cover the change; new code has tests and existing tests still pass; prioritize backend API tests, serializer tests, and critical frontend flows.
- q. Logging & Observability: Ensure meaningful logs are added (no sensitive data), and any new metrics/events are emitted to analytics/monitoring.
- r. Performance Considerations: Check for expensive operations in request path (heavy loops, unbounded queries, large file handling) and suggest pagination, caching, or async tasks where appropriate.
- s. Async & Background Work: If reports or long tasks are added, verify they run asynchronously (Celery / background job) and don't block request/response cycles.
- t. Dependency Changes: Verify third-party updates are necessary, check for known CVEs, and update requirements.txt (or package.json) with exact versions.
- u. Error & Edge Case Paths: Ensure error states (timeouts, DB unavailability) are handled gracefully and meaningful messages are returned to the caller.
- v. Documentation & README: Confirm README, API docs, and any developer notes (backend/<service>/.env.example, README.md) are updated for setup, migration steps, or new env vars.
- w. Backward Compatibility & Migration Plan: For breaking changes, confirm migration plan and communication are included (deprecated endpoints, DB migrations that require downtime, or data transformations).

- x. Lint & Formatting: Ensure code passes linters/formatting (eslint, Python linters) and follows repo style conventions.
- y. PR Checklist Completed: Ensure PR includes a small checklist of items verified (tests added, docs updated, migrations included, security reviewed).
- z. Reviewer Notes: Add suggested reviewers (service owners), and mention if the change requires additional manual testing (e.g., uploading large media, running docker-compose up).

5. Repository Structure

```
docker-compose.dev.yml  
docker-compose.yml  
init-db.sh  
package.json  
run_unit_tests.py  
backend/  
  assets/  
    Dockerfile  
    entrypoint.sh  
    manage.py  
    railway.toml  
    requirements.txt  
  assets/      # Django project files (__init__.py, settings.py, urls.py, wsgi.py, etc.)  
  assets_ms/    # Django app code (models.py, serializers.py, views.py, tests.py,  
  services/, api/, migrations/)  
  media/       # audit_files/, product_images/, repair_files/
```

```
staticfiles/
templates/
user_images/
authentication/
Dockerfile
entrypoint.sh
manage.py
railway.toml
requirements.txt
auth_service/ # App code (models.py, serializers.py, views.py, tests.py, migrations/)
authentication/ # Django project files (asgi.py, settings.py, urls.py, wsgi.py)
staticfiles/
templates/
user_images/
contexts/
Dockerfile
entrypoint.sh
manage.py
railway.toml
requirements.txt
contexts_ms/ # App code (models.py, serializers.py, views.py, tests.py, services/, migrations/, utils/)
media/      # manufacturer_logos/, supplier_logos/
staticfiles/
templates/
```

```
env/          # Local Python virtual environment snapshot  
frontend/  
    Caddyfile  
    Dockerfile  
    eslint.config.js  
    index.html  
    package.json  
    README.md  
    vite.config.js  
public/       # ExternalEmployee.json  
src/          # api.js, App.jsx, main.jsx, constants.js, plus folders: api/, assets/,  
              components/, data/, hooks/, pages/, services/, styles/, utils/
```

A.14

A.15

Appendix B: Project Management Artifacts

B.1 Project Charter / Proposal

Part I: Project Overview

Project Name	MapAms (IT Asset Management System Forecasting)		
Project Charter Author	200Push Team		
Creation Date	September 17, 2025	Last Revision	November 13, 2025
Project Requestor	MAP Active PH IT Head	Project Manager	Ma. Elisa Johanna T. Garrote
Project Charter Status	Approved		
Project Sponsor Signature	MAP Active PH IT Head	Date of Project Approval	September 17, 2025
Proposed Project Start and End	February 2025 - November 2025		

Part II: Project Details

Project Goals and Outcome	The project aims to deliver a modern and cost-effective IT Asset Management System with a QR scanner for asset information and a forecasting feature for predicting asset status and product demand. It seeks to replace the Excel-based tracking method, enhancing accuracy, accountability, and operational efficiency for MAP Active PH's IT Department.
----------------------------------	---

Project Scope	<ol style="list-style-type: none"> 1. Core system modules including Authentication, Account Management, Asset Operation and Maintenance, Monitoring and Dashboard, Context, and Reports Module. 2. Integration of asset requests and tracking with the Helpdesk and Ticket Tracking Systems for seamless monitoring. 3. QR code generation and scanning for quick asset retrieval and audit verification. 4. Forecasting feature to analyze usage patterns and predict future status and product demand. 5. Automated report generation to replace manual Excel tracking. 6. Functional prototype deployment for User Acceptance Testing (UAT) and evaluation. 7. System documentation (user manual and technical package)
Project Deliverables	<ol style="list-style-type: none"> 1. Functional AMS prototype integrated with forecasting and QR features. 2. Audit, maintenance, and reporting automation modules. 3. System documentation package. 4. UAT results and final presentation materials.

Benefits	<ol style="list-style-type: none"> 1. 95%+ asset tracking accuracy compared to Excel. 2. Up to 40% faster audit preparation and reporting. 3. Improved accountability and real-time monitoring of assets. 4. Supports MAP Active PH's digital transformation and cost-efficiency initiatives.
Stakeholders	<p>Primary Stakeholders:</p> <ul style="list-style-type: none"> • Sir Elvin Punzalan – MAP Active PH IT Head (approves requirements and integration alignment) • Group 7 – Asset Management System developers • Group 5 – Helpdesk Team (integration for ticketing requests) • Group 1 – Ticket Tracking Team (links asset issues with AMS) • Group 7 – Budget Management Team (integrates asset cost data) <p>Secondary Stakeholders:</p> <ul style="list-style-type: none"> • Prof. Rosicar Escobar – Capstone Adviser • Prof. Keziah Monzon – Capstone Adviser • Prof. Estong Odpaga – Industry Adviser

Constraints/Risk	<ol style="list-style-type: none"> 1. Limited timeline (September – November 2025 Capstone schedule). 2. Small development team (4 members). 3. No budget for paid enterprise tools – must use open-source technologies (Django, React, Railway, GitHub Student Pack). 4. Scope limited to MAP Active PH Rockwell IT Department. 5. Dependence on integration delivery from collaborating teams.
Assumption	<ol style="list-style-type: none"> 1. MAP Active IT Department provides accurate asset data for migration. 2. Users have devices capable of QR scanning. 3. Stable internet connectivity for cloud access. 4. Collaborating teams (Helpdesk, Ticket, Budget) meet their delivery timelines. 5. Forecasting accuracy depends on available historical data. 6. At least 70% adoption rate by IT Department staff upon deployment
Project Team	<p>Ma. Elisa Johanna T. Garrote – Project Manager, Business Analyst, Documentation Analyst</p> <p>Edzra C. Macas – Back-end Developer, Front-end Developer</p> <p>Bengie B. Villesco – Back-end Developer, Front-end Developer, Quality Assurance</p> <p>Johanna Mae L. Sillano – Front-end Developer, Quality Assurance</p>

Budget Requirements	No formal budget allocation; the team uses free and open-source tools
----------------------------	---

We, the undersigned, acknowledge and approve the project charter for the MAP Active IT Asset Management System (MapAMS) Capstone Project.

Team Members

1. Garrote, Ma. Elisa Johanna T. 
2. Macas, Edzra C. 
3. Sillano, Johanna Mae L. 
4. Villesco, Bengie B. 

Instructor Approval

Approved

Revisions Required (Comments: _____)

B.2 Sprint Backlogs and Burndown Chart

Part I: Sprint backlogs

Sprint 1: System Preparation

Project Name:	Asset Management System with Forecasting (MapAms)	Total Task	46
Team	Group 7: 200Push	Overall Progress	100%

Sprint 1: Prototype					
US-ID	Description	Priority	Story Points	Assignee	Status
US-01 to US-22	Context Module Prototype creations	High	7	Johanna Sillano	Done
US-23 to US-56	Asset Module Prototype creations	High	7	Bengie Villesco	Done
US-57 to US-61	Reports Module Prototype creations	Medium	5	Bengie Villesco	Done
US-62 to US-75	Dashboard Module Prototype creations	Medium	5	Johanna Sillano	Done
US-76 to US-83	Integrated Module Prototype creations	High	7	Edzra Macas	Done
US-84	Notification Module Prototype creations	Medium	5	Johanna Sillano	Done
	BRD Creation	High		Elisa Garrote	Done
	Backend and Frontend Tools Preparation	Medium		Edzra Macas	Done

Sprint 2: Frontend and Backend Development

Project Name:	Asset Management System with Forecasting (MapAms)	Total Task	77
Team	Group 7: 200Push	Overall Progress	100%

Sprint 2: Frontend and Backend Development					
US-ID	Description	Priority	Story Points	Assignee	Status
US-01 to US-22	Context Module Frontend	High	7	BengieVillesco	Done
US-23 to US-56	Asset Module Frontend	High	7	Johanna Sillano / Bengie Villesco	Done
US-57 to US-61	Reports Module Frontend	Medium	5	Bengie Villesco	Done
US-62 to US-75	Dashboard Module Frontend	Medium	5	Johanna Sillano	Done
US-76 to US-83	Integrated Module Frontend	High	7	Edzra Macas	Done
US-84	Notification Module Frontend	Low	3	Johanna Sillano	Done
	Testing UI/UX			Bengie Villesco	Done
	Revision on Models			Edzra Macas	Done

Sprint 3: Frontend and Backend Development

Project Name:	Asset Management System with Forecasting (MapAms)	Total Task	20
Team	Group 7: 200Push	Overall Progress	100%

Sprint 3: Frontend and Backend Development					
US-ID	Description	Priority	Story Points	Assignee	Status
US-01 to US-22	Context Module Backend	High	7	BengieVillesco	Done
US-23 to US-56	Asset Module Backend	High	7	Johanna Sillano / Bengie Villesco	Done
	Testing Backend			Bengie Villesco	Done

Sprint 4: Frontend and Backend Development

Project Name:	Asset Management System with Forecasting (MapAms)	Total Task	57
Team	Group 7: 200Push	Overall Progress	45%

Sprint 4: Frontend and Backend Development					
US-ID	Description	Priority	Story Points	Assignee	Status
US-57 to US-61	Reports Module Backend	High	7		Not Executed
US-62 to US-75	Dashboard Module Backend	Medium	5	Edzra Macas	In Progress
US-76 to US-83	Integrated Module Backend	High	7	Edzra Macas	Done

US-84	Notification Module Backend	Low	3		Not Executed
	Testing Backend			Bengie Villesco	Done

Sprint 5: Frontend Revisions

Project Name: Asset Management System with Forecasting (MapAms) **Total Task** 44
Team Group 7: 200Push **Overall Progress** 100%

Sprint 5: Frontend Revisions					
US-ID	Description	Priority	Story Points	Assignee	Status
US-01 to US-22	Context Module Frontend Revisions	High	7	Bengie Villesco	Done
US-23 to US-56	Asset Module Frontend Revisions	High	7	Johanna Sillano / Bengie Villesco	Done
US-57 to US-61	Reports Module Frontend Revisions	Medium	5	Bengie Villesco	Done
US-62 to US-75	Dashboard and Forecasting Module Frontend Revisions	Medium	5	Johanna Sillano	Done
US-76 to US-83	Integrated Module Frontend Revisions	High	7	Edzra Macas	Done
	Testing UI/UX			Bengie Villesco	Done
	Revision on Models			Edzra Macas	Done

Sprint 6: Backend Revisions

Project Name:	Asset Management System with Forecasting (MapAms)	Total Task	52
Team	Group 7: 200Push	Overall Progress	29%

Sprint 6: Backend Revisions					
US-ID	Description	Priority	Story Points	Assignee	Status
US-01 to US-22	Context Module Backend Revisions	High	7	Elisa Garrote	Done
US-23 to US-56	Asset Module Backend Revisions	High	7	Edzra Macas	In Progress
US-57 to US-61	Reports Module Backend Revisions	Medium	5	Elisa Garrote / Bengie Villesco	In Progress
US-76 to US-83	Integrated Module Backend Revisions	Medium	5	Edzra Macas	In Progress
	Testing Backend endpoints	High		Bengie Villesco	Done
	Forecasting UI/UX Development	High		Johanna Sillano	Done
	QR Code UI/UX Development	High		Johanna Sillano	Done

Sprint 7: Backend Revisions

Project Name:	Asset Management System with Forecasting (MapAms)	Total Task	29
Team	Group 7: 200Push	Overall Progress	75%

Sprint 6: Backend Revisions					
US-ID	Description	Priority	Story Points	Assignee	Status
US-23 to US-56	Asset Module Backend Revisions	High	7	Edzra Macas	Done
US-57 to US-61	Reports Module Backend Revisions	Medium	5	Elisa Garrote / Bengie Villesco	In Progress
US-62 to US-75	Dashboard and Forecasting Module Backend Revisions	Medium	5	Elisa Garrote / Edzra Macas	In Progress
US-76 to US-83	Integrated Module Backend Revisions	High	7	Edzra Macas	Done
	Testing Backend endpoints			Bengie Villesco	Done

Sprint 7: Additional Sprint

Project Name:	Asset Management System with Forecasting (MapAms)	Total Task	15
Team	Group 7: 200Push	Overall Progress	20%

Sprint 6: Backend Revisions					
US-ID	Description	Priority	Story Points	Assignee	Status
US-30 to	Asset Module QR Code Development	High	7	Johanna Sillano	Done

US-36					
US-74 and US-75	Forecasting development	Medium	5	Edzra Macas	In Progress
US-57 to US-61	Reports Module Backend Revisions	Medium	5	Elisa Garrote / Bengie Villesco	In Progress
US-76 to US-83	Integrated Module Backend Revisions (AUTH)	High	7	Bengie Villesco	In Progress
US-84	Notification Development	Low	3	Edzra Macas	Not Executed
	Testing Overall Performance	High		Bengie Villesco	Not Executed

Part II: Burndown Charts

Sprint 1:

B.3 Meeting Minutes

Part I: Team Meetings

Kick-off Meeting			
Date	Feb 23, 2025	Attendees	All members
Agenda			
Role Distribution, Task Assignment, and Role Sharing			
Key Decisions			
Roles are distributed and the possible task that will be handled. Also, the team discusses the task that can be shared with other member incase the assign member in role is not available or need assistance			
Action Items with Owners and Deadline			
Task	Description	Owner	Deadline
Research	Technology Stack Research	Edzra and Bengie	Next Meeting
	System Research Planning	Elisa and Johanna	Next Meeting

Technology Stack Meet			
Date	Feb 25, 2025	Attendees	All members
Agenda			
Decision on Technology Stack			

Key Decisions

The team decided to use react+vite, django, and Github in accordance with other industry groups.

Action Items with Owners and Deadline

Task	Description	Owner	Deadline
Prepare Questionnaire	Preparing questions for company about their workflow painpoints	Elisa	March 11
Company Meeting	Team will meet the client in MapActive PH together with other industry groups	All	March 11

Technology Stack Meet

Date Feb 25, 2025 Attendees All members

Agenda

Decision on Technology Stack

Key Decisions

The team decided to use react+vite, django, and Github in accordance with other industry groups.

Action Items with Owners and Deadline

Task	Description	Owner	Deadline
Prepare Questionnaire	Preparing questions for company about their workflow painpoints	Elisa	March 11
Company Meeting	Team will meet the client in MapActive PH together with other industry groups	All	March 11

Business Requirements and Wireframes Meet

Date	Mar 20, 2025	Attendees	All members
Agenda			
Discuss the created business requirements and wireframes. Check if wireframe is suitable on requirements			
Key Decisions			
The team will revise some wireframes to match the business requirements			
Action Items with Owners and Deadline			
Task	Description	Owner	Deadline
Create Sprint	PM will create sprint backlogs	Elisa	March 27
Wireframe revisions	Team will revise the prototype according into the company feedbacks	Bengie, Edzra, and Johanna	March 27

Sprint 1 Meet

Date	Apr 6, 2025	Attendees	All members
Agenda			
The team will discuss the task on Sprint 1			
Key Decisions			
The team discusses the task on sprint 1 which includes creating prototype for the map asset management system			
Action Items with Owners and Deadline			
Task	Description	Owner	Deadline
Stand-up	The team will conduct stand-up		

	meeting between weeks		
Sprint Retrospective	The team will meet next week about sprint retrospective		Next Meeting

Sprint 1 Retrospective			
Date	Apr 13, 2025	Attendees	All members
Agenda			
Check the flow and updates of team			
Key Decisions			
The team discusses what went well and not during sprint 1. Also, discusses what can be improved in sprint 2			
Action Items with Owners and Deadline			
Task	Description	Owner	Deadline
Sprint 2 Meet	The team will meet on next meeting before start of sprint 2		Next Meeting

Sprint 2 Meet			
Date	Apr 15, 2025	Attendees	All members
Agenda			
The team will discuss the task on Sprint 2			
Key Decisions			
The team discusses the task on sprint 2 which includes developing the UI and UX of the map asset management system			

Action Items with Owners and Deadline

Task	Description	Owner	Deadline
Stand-up	The team will conduct stand-up meeting between weeks		
Sprint Retrospective	The team will meet next week about sprint retrospective		Next Meeting

Sprint 2 Retrospective

Date	Apr 24, 2025	Attendees	All members
Agenda			
Check the flow and updates of team			
Key Decisions			
The team discusses what went well and not during sprint 2. Also, discuss what can be improved in sprint 2 and adjust the system according to company feedback every weekly meeting.			
Action Items with Owners and Deadline			
Task	Description	Owner	Deadline
Sprint 2 Meet	The team will meet on next meeting before start of sprint 3		Next Meeting

Sprint 3 Meet

Date	Apr 15, 2025	Attendees	All members
Agenda			
The team will discuss the task on Sprint 3			

Key Decisions

The team discusses the task on sprint 3 which includes developing the backend of the map asset management system

Action Items with Owners and Deadline

Task	Description	Owner	Deadline
Stand-up	The team will conduct stand-up meeting between weeks		
Sprint Retrospective	The team will meet next week about sprint retrospective		Next Meeting

Sprint 3 Retrospective

Date	Apr 24, 2025	Attendees	All members
------	--------------	-----------	-------------

Agenda

Check the flow and updates of team

Key Decisions

The team discusses what went well and not during sprint 3. Also, discuss what can be improved in sprint 3 and adjust the system according to company feedback every weekly meeting.

Action Items with Owners and Deadline

Task	Description	Owner	Deadline
Sprint 2 Meet	The team will meet on next meeting before start of sprint 3		Next Meeting

Sprint 4 Meet

Date	Apr 15, 2025	Attendees	All members
Agenda			
The team will discuss the task on Sprint 4			
Key Decisions			
The team discusses the task on sprint 4 which includes continuation of development of the backend of the map asset management system.			

Sprint 4 Retrospective and Sprint 5 (1) Meet			
Date	Sep 11, 2025	Attendees	All members
Agenda			
Review the system and determine improvements needed from Capstone 1.			
Key Decisions			
<ul style="list-style-type: none"> The team discussed what went well and what did not during the sprints in Capstone 1. New approaches will be applied to maximize development time for Capstone 2. Weekly sprints and Wednesday stand-up meetings are confirmed, with retrospectives at the end of each sprint. A two-week sprint cycle will be followed for major outputs and reviews. Beta version with fully integrated modules is targeted for November 12, with a possible extension to November 20–26. Basic functionalities must be completed by last week of October (around Oct 22). Emerging technologies will be incorporated by Sprint 3. Final revisions must focus on polishing modules, integration, and defense preparation. 			
Action Items with Owners and Deadline			
Task	Description	Owner	Deadline
Finalize Sprint Setup	Implement weekly sprints, Wednesday stand-ups, and retrospectives	Team	Before Sprint 5
Prepare Beta	Complete module integration and	Team	Nov 12 (possible)

Version	core features		extension to Nov 20–26)
Finish Basic Functions	Core features completed before integration	Team	Oct 22
Integrate Emerging Tech	Add new technologies starting Sprint 3	Assigned Developers	Sprint 3
System Revisions	Polish system components for defense	Team	Before Mock & Final Defense
Help Desk & Automation Updates	Ensure proper integration and feature completion	Assigned Developers	Before Final Version
Defense Materials	Prepare presentation, video, and documentation	Documentation & Presentation Team	Before Mock Defense

Sprint 5 (1) Retrospective			
Date	Apr 29, 2025	Attendees	All members
Agenda			
Review Sprint 1(5) performance, identify what went well, what didn't, and outline improvements for Sprint 2(6).			
Key Decisions			
<ul style="list-style-type: none"> UI customization early in the sprint helped align the system with project goals. Mid-sprint retrospectives will continue because they allowed timely adjustments. Responsibilities will be balanced better to avoid uneven workload among members. Reusable components will remain part of the development approach to speed up future modules. Proven UI libraries (e.g., React Select) will be used for complex components to reduce rework. Standardized component and responsiveness design will be established early in each sprint. Environment setup issues (Docker, PostgreSQL) will be handled collectively to avoid delays. 			

- Realistic time estimation will be prioritized in Sprint 2 planning.
- Members returning from breaks will undergo short catch-up sessions to restore momentum.

Action Items with Owners and Deadline

Task	Description	Owner	Deadline
Sprint 2 Meet	The team will meet on next meeting before start of sprint 3		Next Meeting
Balanced Task Distribution	Divide dev & documentation tasks evenly	Team Lead	Start of Sprint 2
Stand-up Progress Reporting	Members present actual status during stand-ups	All Members	Every Wednesday
Realistic Time Estimates	Improve estimation in sprint planning	Team	Sprint 2 Planning
Use Proven Libraries	Apply reliable UI libraries for dropdowns and complex components	Frontend Devs	Sprint 2
Standardized Components	Set unified component/respondiveness guidelines	UI/UX + Frontend	Early Sprint 2
Fix Environment Setup	Review Docker & PostgreSQL setup as a team	Backend + DevOps	Sprint 2
Mid-sprint Reviews	Continue mid-sprint checks and raise blockers early	Team	Mid-Sprint 2
Catch-up Sessions	Assist returning members with quick sync sessions	Team Lead	As needed
Improve Communication	Encourage earlier reporting of difficulties	Entire Team	Continuous

Part I: Company Meetings

B.4 Project Timeline

TASK	ASSIGNED TO	PROGRESS	START	END
I. Revision / Draft Week				
1. Final list of AMS Modules	Elisa	100%	9/5/25	9/10/25
2. Cleaning GitHub Branches	Bengie	100%	9/5/25	9/10/25
3. Creating Docker (Local / Deployment)	Edzra	100%	9/5/25	9/10/25
4. UI Structure Revision Plan (Document)	Johanna	100%	9/6/25	9/10/25
1.1 Chapter 1 Document Draft				
1. Project Charter	Elisa	100%	9/15/25	9/18/25
2. Version Control & CI/CD Pipeline Document	Elisa	100%	9/15/25	9/18/25
3. RRL	Elisa & Edzra	100%	9/15/25	9/18/25
4. Project Background and Motivation	Elisa	100%	9/18/25	9/20/25
5. Problem Statement	Elisa	100%	9/18/25	9/20/25
1.1 Chapter 1 Document Draft				
6. Project Vision and Scope	Elisa	100%	9/18/25	9/20/25
7. Objectives and Goals	Elisa	100%	9/20/25	9/24/25
8. Significance and Relevance	Elisa	100%	9/20/25	9/24/25

9. Definition of Terms	Elisa	100%	9/20/25	9/24/25
10. Structure of the Document	Elisa	100%	9/20/25	9/24/25
1.2 Document Revision				
1. Business Requirements (Simple for UI)	Elisa	100%	9/15/25	9/16/25
2. RTM (End of Sprint 1)	Elisa	100%	9/17/25	9/18/25
3. Test Cases (End of Sprint 1)	Elisa	100%	9/19/25	9/21/25
4. Testing (UI Only)	Elisa	100%	9/24/25	9/27/25
		0%		
II. Development Sprint 1 Backlogs (Frontend Development & Customization)				
1. Reports Module Developmet	Bengie	100%	9/17/25	9/19/25
2. Asset Model/Product Customization	Johanna	100%	9/17/25	9/19/25
3. Asset Customuzation	Johanna	100%	9/19/25	9/21/25
4. Asset Maintenance	Edzra	100%	9/19/25	9/21/25
5. Asset Audit	Edzra	100%	9/19/25	9/21/25
2.1 Development Sprint 1 Backlogs (Frontend Development & Customization)				
6. Category Customization (Context)	Bengie	100%	9/19/25	9/21/25
7. Manufacturer Customize (Context)	Bengie	100%	9/19/25	9/21/25

8. Supplier Customize	Bengie	100%	9/21/25	9/23/25
9. Status Customize	Bengie	100%	9/21/25	9/23/25
10. Depreciation Customize	Edzra	100%	9/21/25	9/23/25
2.2 Development Sprint 1 Backlogs (Frontend Development & Customization)				
11. Recycle Bin Customize	Edzra	100%	9/21/25	9/23/25
12. Tickets Customization	Johanna	100%	9/21/25	9/23/25
13. Dashboard Customize	Johanna	40%	9/21/25	9/23/25
14. Component Customization	Edzra	100%	9/24/25	9/26/25
15. Notification Customization	Johanna	0%	9/24/25	9/26/25
2.3 Development Sprint 1 Backlogs (Frontend Development & Customization)				
16. Component Customization	Edzra	100%	9/24/25	9/26/25
17. Consumables Customization	Bengie	0%	9/24/25	9/26/25
18. Accessories Customization	Johanna	0%	9/24/25	9/26/25
		0%		
		0%		
III. Sprint 2 Development and Documents				

1. Sprint 1 Retrospective Summary	Elisa	100%	9/29/25	9/29/25
2. Updated RTM	Elisa	100%	9/30/25	10/1/25
3. Methodology Draft	Elisa	100%	9/30/25	10/1/25
4. Test Result for Sprint 1	Elisa	100%	10/1/25	10/1/25
5. Architecture Diagrams	All	100%	9/30/25	9/30/25

3.1. Sprint 2 Refining Backend

1. Implementation of Auth(Move to next Sprint)	Bengie	0%		
2. Finalization of Models	Edzra	100%	10/1/25	10/4/25
3. Supplier Backend	Elisa	100%	10/2/25	10/5/25
4. Manufacturer Backend	Johanna	0%	10/2/25	10/5/25
5. Status Backend	Bengie	100%	10/4/25	10/7/25
6. Category Backend	Edzra	0%	10/4/25	10/7/25
7. Depreciation Backend	Elisa	100%	10/4/25	10/6/25
8. Recycle Bin	Johanna	0%	10/4/25	10/6/25
		0%		
		0%		

3.2 Documents

1. Progress Report of Sprint 2	Elisa	100%	10/8/25	10/8/25
2. Updated RTM	Bengie / Elisa	100%	10/7/25	10/8/25
3. Test Cases and Result	Bengie	80%	10/7/25	10/8/25
4. Documentation: Part 4.1 - 4.3 (Req. Analysis, Sys. Design, BPMN)	Johanna / Macas	100%	10/7/25	10/8/25
		0%		

Mid Sprint 2 Backend Refine

Asset Model	Bengie	0%	10/9/25	10/12/25
Assets	Bengie	0%	10/9/25	10/12/25
Asset Check-in and Check-out	Edzra	0%	10/9/25	10/12/25
Tickets	Edzra	0%	10/9/25	10/12/25
Asset Repair	Elisa	30%	10/9/25	10/12/25

Mid Sprint 2 Frontend Refine

NavBar	Johanna	0%	10/9/25	10/12/25
Page Layout (Focus on scrollbar and margins)	Johanna	0%	10/9/25	10/12/25
Profile Management	Johanna	0%	10/9/25	10/12/25
Status Display	Johanna	0%	10/9/25	10/12/25
		0%		

Mid Sprint 2 Documentation

System Architecture	Edzra	100%	10/13/25	10/14/25
RTM v2.1	Elisa	100%	10/13/25	10/14/25
(App/Data/Tech Architecture)	Bengie, Edzra, Johanna	100%	10/13/25	10/14/25
		0%		
		0%		

End Sprint 2 Backend Refine

Asset Audit	Bengie	0%	10/15/25	10/22/25
Component	Edzra	0%	10/15/25	10/22/25
Component Check-in/out	Edzra	0%	10/15/25	10/22/25
Check-in Due Reports	Elisa	0%	10/15/25	10/22/25
Depreciation Reports	Elisa	0%	10/15/25	10/22/25

End Sprint 2 Backend Refine

All Asset Reports	Bengie	0%	10/22/25	10/25/25
EoL Reports	Elisa	0%	10/22/25	10/25/25
Activity Reports	Edzra	0%	10/22/25	10/25/25
		0%		
		0%		

End Sprint 2 Frontend Refine

Advance Filter UI (Assets, Products, and Components)	Johanna	0%	10/15/25	10/19/25
Login UI Refine	Johanna	0%	10/15/25	10/19/25
Register User UI Refine	Johanna	0%	10/15/25	10/19/25

Sprint 3 Start Documentation

Sprint 2 Retrospective	Elisa	100%
4.7.1 Development Journey (Sprint Summary)	Elisa	100%
4.7.2 Development Journey (Sprint Summary)	Edzra / Johanna	100%
RTM v2.2	Bengie	100%

Appendix C: Research and Requirements Artifacts

C.1 Approved Adviser Acceptance Form

March 13, 2025

Faculty, ROSICAR E. ESCOBER

Sir Royette Vergara:

This is to inform you that you have been chosen to be the thesis adviser of:

Section and Group Members: **3-1: Ma. Elisa Johanna T. Garrote, Edzra C. Macas, Johanna Mae L. Sillano, Bengie B. Villesco**

Program: **Bachelor of Science in Information Technology**

Capstone Project: **Asset Management System with AI as Emerging Technology (Tentative name)**

Please sign below if you accept the above-mentioned candidate for discipleship.

Thank you very much for your cooperation.

Demelyn E. Monzon, Ph.D

CaPECom Chairman

This confirms my acceptance to be the thesis adviser:

Faculty: _____

Signature: _____

Date: _____

C.2 Company Endorsement / Memorandum of Agreement

Polytechnic University of the Philippines - Quezon City Campus/Branch
Bachelor of Science in Information Technology
Don Fabian St., Brgy. Commonwealth.

March 03, 2025

MAP Active Philippines
1 Proscenium at Rockwell
Estrella Street, Makati, Metro Manila

Subject: Request for Collaboration and Data Gathering for Capstone Project Development

Dear [Recipient's Name or "Sir/Madam"],

We are a group of Bachelor of Science in Information Technology (BSIT) students from Polytechnic University of the Philippines - Quezon City Campus/Branch, currently working on our Capstone Project. Our project involves developing an integrated information system tailored to address specific business needs. We are reaching out to your esteemed organization to request your collaboration in providing the necessary data and insights to ensure the success of our project.

About Our Capstone Project

We aim to design and develop a system that addresses real-world challenges and improves operational efficiency. To achieve this, we strive to:

1. Understand your current workflows, processes, and pain points.
2. Identify the technologies and systems currently in use.
3. Gather requirements for the proposed system, including features and functionalities that would benefit your organization.
4. Develop an **integrated solution** that aligns with your business needs and industry best practices.

How Your Organization Can Help

We kindly request your assistance in the following areas:

1. **Interviews or Meetings** – To understand how things are done in your organization, including workflows, challenges, and expectations from the proposed system.
2. **Data Gathering** – To collect relevant data (e.g., sample datasets, process flows) necessary for system development.
3. **Feedback** – To validate our proposed solution and ensure it meets your requirements.

Assurance of Data Security

We understand the sensitivity of the information you may share with us. Please rest assured that:

- All data gathered will be used solely for academic purposes and system development.
- We will adhere to **strict data privacy and security protocols** to ensure the confidentiality of your information.
- Any data shared will be securely stored and accessible only to the project team.
- At the end of the project, all data will be securely deleted or returned to your organization, as you prefer.

Next Steps

If you are open to collaborating with us, we would appreciate the opportunity to discuss this further. We will accommodate your interview schedule, meetings, or data-sharing sessions. Please let us know a convenient time for you, and we will make the necessary arrangements.

We believe that this collaboration will benefit our academic growth and provide your organization with valuable insights and potential solutions to enhance your operations.

Thank you for considering our request. We look forward to your positive response. Should you have any questions or require additional information, please feel free to contact us at garratemaelisajohanna@gmail.com or in our contact number 0961- 451-1157.

Sincerely,

Ma. Elisa Johanna T. Garrote
Project Manager
Polytechnic University of the Philippines
garrotmaelisajohanna@gmail.com / 0961 451 1157

On behalf of the Capstone Project Team:

1. Bengie B. Villesco - bvillesco@gmail.com - 0938 878 4663
2. Johanna Mae L. Sillano - sillanojohannamae@gmail.com - 0999 181 2219
3. Edzra C. Macas - edzram7@gmail.com - 0936 980 8474

Noted by:

Kezaiah M. Cruz, MSIT

C.3 Survey Questionnaires and Raw Data

Survey Questionnaire

Company Process Assessment Questionnaire

This questionnaire is designed to gather comprehensive information about the organization, including how its workflows are structured, the current systems and tools in use, the challenges faced by various departments, and potential areas where processes and systems could be improved to enhance efficiency and effectiveness.

Section 1: General Information

This section aims to gather information about your role, the people involved in system development, and the key IT systems used in your department.

1. What is your role in the organization?
(Describe your responsibilities and involvement in system processes.)
2. Who are the people that should be involved in the development of the system?
(Provide details based on your department's structure.)
3. What key IT systems or software does your department currently use?
(Include tools such as databases, POS systems, document trackers, spreadsheets, and others.)

Section 2: Current Processes

This section explores how your department handles its workflows, identifies process breakpoints, and examines the tools and data used.

4. Describe your typical workflow for your main tasks.
(Explain the step-by-step process.)
5. How does your department handle help desk processes?
6. How does your department handle the asset management process?
7. How does your department handle budgeting processes?
8. How does your department handle approval workflows?

Section 3: Department and System Breakpoints

9. What are the process breakpoints in the Planning Department or System?
10. What are the process breakpoints in the Brands Department or System?
11. What are the process breakpoints in the Supply Chain Management Department or System?
12. What are the process breakpoints in the Finance Department or System?
13. What are the breakpoints in your current reporting capability?

Section 4: Documentation, Tools, and Data Handling

14. How is information recorded and tracked throughout the process?
(Indicate whether steps are manual or automated.)
15. What tools, systems, or documents are used during the process?
16. What types of data are collected and stored during the process?
17. Which steps rely heavily on manual input?
18. Are there approval steps or dependencies that cause delays?

Section 5: Pain Points

This section identifies challenges, delays, and frustrations in your department's workflows.

19. What are the most common challenges you experience in your daily workflow?
20. Which points in the process often result in delays or mistakes?
21. Have you experienced data loss, duplication, or system crashes?
22. What security concerns do you have with the current system?
23. How does your team handle unexpected issues like system downtime or technical problems?
24. Which steps are most frustrating or time-consuming for your team?
25. Have there been complaints from other departments or clients regarding your system? If yes, what were they?

Section 6: Suggestions for Improvement

This section gathers input on potential improvements to current workflows and system features.

26. If you could improve one part of your workflow, what would it be?
27. What system features do you wish existed to make your work easier?
28. Have you considered other tools or solutions in the past? If so, what were they?

Section 7: Final Insights

This section collects any additional feedback or willingness to participate in follow-up discussions.

29. Is there anything else you would like to share about your processes or system challenges?
30. Would you be open to a follow-up interview if needed?

Raw Data

The respondent, Sir Elvin Punzalan, is a stakeholder and manager overseeing IT workflows. The people who should be involved in the system development include the IT team, the Finance head, and the Operations manager. The key IT systems currently used by the department are SAP, ETP (Enterprise Transaction Processing), and the POS system.

In terms of current processes, employees raise IT-related issues through a ticketing system, although parts of the tracking are still manual. The IT department manages asset tracking for hardware, software, and devices, with some steps performed manually. Budgeting processes are tracked manually, with existing approval workflows in place. Document tracking is integrated with the helpdesk ticketing system.

The main challenges identified include the complexity of the POS system, delays caused by manual approvals, and repetitive data entry that often results in errors. Suggestions for improvement include focusing system development on helpdesk and workflow management, automating approval steps to reduce delays, and ensuring the system is cost-effective.

The company expects a web-based solution with cloud deployment, prioritizing budgeting and IT asset management. The preferred technology stack includes a backend developed in .NET, a SQL Server database, and deployment on Azure or AWS. Additional notes from the interview indicate that the company expects the team to propose solutions rather than only gather requirements. The development project should be simple and achievable within 18 weeks. The next steps involve preparing for stakeholder interviews and refining the proposal.

C.4 Interview Transcripts and Guides (Anonymized)

Interview Transcripts and Guides

This document presents anonymized interview transcripts collected from company stakeholders during the Capstone 1 company visit. The transcripts are written in third person to reflect the observations recorded during the interviews. The structure follows the survey questionnaire and is divided into sections for clarity. Each section summarizes responses to specific topics, including general information, current processes, department and system breakpoints, documentation and data handling, pain points, suggestions for improvement, and final insights. The transcripts serve as a raw record of interview observations for analysis and project planning while ensuring the confidentiality of participants.

Section 1: General Information

Question 1: What is the respondent's role in the organization?

The respondent is a manager overseeing IT workflows, responsible for system maintenance, database management, and workflow coordination.

Question 2: Who should be involved in the development of the system?

The respondent indicated that the IT team, the Finance head, and the Operations manager should be involved in the system development process.

Question 3: What key IT systems or software does the department currently use?

The respondent reported that the department uses SAP, ETP (Enterprise Transaction Processing), and a POS system.

Section 2: Current Processes

Question 4: Describe the typical workflow for the respondent's main tasks.

The respondent explained that requests are received via email, the system is updated accordingly, issues are tracked in spreadsheets, and weekly reports are submitted to management.

Question 5: How does the department handle help desk processes?

Employees log tickets in a ticketing system. Tickets are assigned to IT staff and are typically resolved within 24 hours.

Question 6: How does the department handle the asset management process?

The IT department tracks hardware, software, and devices. Some steps, such as logging new devices, are performed manually in spreadsheets.

Question 7: How does the department handle budgeting processes?

Department spending is tracked manually. Approval workflows exist but occasionally cause delays.

Question 8: How does the department handle approval workflows?

Approval steps require manager signatures. Delays occur when managers are unavailable.

Section 3: Department and System Breakpoints

Question 9: What are the process breakpoints in the Planning Department or System?

The respondent noted that manual data entry and delays in approval signatures often create bottlenecks.

Question 10: What are the process breakpoints in the Brands Department or System?

The respondent indicated that delays are caused by slow reporting and manual tracking.

Question 11: What are the process breakpoints in the Supply Chain Management Department or System

The respondent reported that repetitive data entry and duplicated records are the main challenges.

Question 12: What are the process breakpoints in the Finance Department or System?

The respondent identified manual approvals and slow report consolidation as significant breakpoints.

Question 13: What are the breakpoints in the current reporting capability?

The respondent explained that current reporting is hindered by delayed data inputs and manual tracking processes.

Section 4: Documentation, Tools, and Data Handling

Question 14: How is information recorded and tracked throughout the process?

The respondent reported that some steps are automated in the ticketing system, while others are still tracked manually in spreadsheets.

Question 15: What tools, systems, or documents are used during the process?

The respondent indicated the use of Excel files, logs, custom software, and ticketing systems.

Question 16: What types of data are collected and stored during the process?

The respondent noted that user information, transactions, certificates, and operational records are collected and stored.

Question 17: Which steps rely heavily on manual input?

The respondent stated that asset logging, budgeting approvals, and certain reporting steps are heavily manual.

Question 18: Are there approval steps or dependencies that cause delays?

The respondent confirmed that waiting for manager signatures and slow data entry are common sources of delays.

Section 5: Pain Points

Question 19: What are the most common challenges experienced in daily workflow?

The respondent identified delays due to manual approvals, repetitive data entry errors, and occasional system downtime.

Question 20: Which points in the process often result in delays or mistakes?

Manual tracking, waiting for approvals, and duplicating entries in multiple systems frequently cause errors and delays.

Question 21: Have there been issues with data loss, duplication, or system crashes?

The respondent reported occasional loss or duplication of spreadsheets and minor system downtime.

Question 22: What security concerns exist with the current system?

Limited access control and potential data breaches were highlighted as concerns.

Question 23: How does the team handle unexpected issues like system downtime or technical problems?

The respondent indicated that manual workarounds are followed, tickets are escalated internally, and backup files are used to continue operations.

Question 24: Which steps are most frustrating or time-consuming for the team?

Manual approvals, repeated data entry, and slow reporting processes were identified as most time-consuming.

Question 25: Have there been complaints from other departments or clients regarding the system? If yes, what were they?

The respondent mentioned complaints related to delays in approvals, slow report generation, and difficulties accessing certain systems.

Section 6: Suggestions for Improvement

Question 26: If one part of the workflow could be improved, what would it be?

The respondent suggested automating approvals and reducing manual tracking to improve efficiency.

Question 27: What system features are desired to make work easier?

The respondent recommended a centralized web-based workflow and automated report generation.

Question 28: Have other tools or solutions been considered in the past? If so, what were they?

The respondent noted consideration of ERP add-ons and cloud-based ticketing systems.

Section 7: Final Insights

Question 29: Is there anything else to share about processes or system challenges?

The respondent emphasized the need for a solution that is simple, cost-effective, and achievable within the project timeline.

Question 30: Would the respondent be open to a follow-up interview if needed?

The respondent confirmed openness to follow-up interviews for clarification and further discussion.

C.5 UAT Forms and Signed Off Sheets

User Acceptance Testing

Document Information	
Project Name	MapAMS (IT Asset Management System Forecasting)
Author	200Push Team
Issue Date	

Document History

Version	Date	Summary of change	Author
1.0	26/11/2025	Release version	200Push Team

Approvals

Name	Title	Signature	Escalation
Ma. Elisa Johanna T. Garrote	Project Manager		

DOCUMENT CIRCULATION

Version	Date	Person/Group	Author
1.0	26/11/2025	200Push Team	

This document may include information that is **CONFIDENTIAL** and shall not be disclosed outside UCL and shall not be duplicated, used, or disclosed in whole or in part for any purpose other than to evaluate and implement procedures defined within this document.

1. ACCEPTANCE TESTING SCOPE

This section defines the scope of User Acceptance Testing (UAT) for the MapAMS system. The testing will focus on verifying that the system meets the functional and operational requirements as specified in the project documentation. It covers all key modules and features, ensuring proper workflow, user access control, data accuracy, and overall system behaviour before deployment. The criteria listed in the following tables will guide the testing and determine if the system is acceptable for production use.

Testing and Acceptance Criteria – Stage 1

Test #	Steps / Condition	Acceptance Criteria
Context		
1	Verify RBAC	Only user with Admin role can access context and see the "More" menu in the navigation bar.
2	Check if the navigation bar has a "More" menu.	"More" menu show as dropdown.
3	Verify Context menu when "More" tab is click or hover.	The following choices must be seen, has hover, and clickable: <ul style="list-style-type: none">- Category- Supplier- Manufacturer- Status- Depreciation- Recycle Bin

4	Click each menus in "More" tab to verify the routing of Context menu .	The user must redirect to the correct page when click menus such as category, supplier, manufacturer, status, depreciation, and recycle bin.
Category Context		
1	Click "More" tab then click "Categories" to verify the routing page.	The user must redirect to the "Category" page.
2	Verify Category Page .	<p>1. Category Page must show tables that has the following columns:</p> <ul style="list-style-type: none"> - Checkbox - Image - Name - Type - Quantity - Edit - Delete <p>2. Above columns, table must have headers, buttons, and filters, like the following:</p> <ul style="list-style-type: none"> - Categories (Quantity Register) - Search Bar - Filters (Type and list must be sorted Alphabetically) - "+ New" button to register new category. It must be clickable and has hover. - Shows bulk-delete when checkbox is clicked. <p>3. The table must have pagination and the list must be responsive to paginationlist number.</p> <p>4. While typing for search category (all column values are searchable), user must</p>

		<p>be able to see all the result of the search value. If no record found, it must return a message of "No categories found." in the table.</p> <p>5. The list must update based on the selected filters when the user uses the filter feature (still sorted alphabetically.)</p>
3	Check if the Category Page supports scrolling.	User can scroll to view all content without layout issues or cutoffs
4	Check if the Category Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Category Registration		

1	Click "+ New" button to verify if category registration page exist.	User must redirect to the category registration page.
2	<p>Verify Category Registration Page.</p>	<ol style="list-style-type: none"> 1. The Category Registration page must have header that contains the following: <ul style="list-style-type: none"> - Link "Categories" to navigate on list of Category. Must be seen as Category / New Category . - Import button with Icon and must be clickable and has hover. The Import button must allowed xlsx file only, validate the column structure and data types per column, return error messages for any invalid entries, and populate the fields with the file's values if no errors are found. - Has line separator to seperate headers and form. 2. Import button must allowed xlsx file only. 3. Form must have the following fields: <ul style="list-style-type: none"> - Category Name. Required and has (*.) Also. must be textfield and max input. - Category Type. Required and has (*.) Also, a dropdown. - Icon. Must have button to choose file. Once file choosen, the file name must be seen and clickable to view the Icon. Icon must accept jpeg and png file only max of 5mb. 4. Save button. Must be enable only if required field are filled. 5. There must be validation if category already existed (backend works), missing value on required fields, and successful category registration. 6. Once Supplier successfully registered, user must automatically navigate to list of categories with success alert message of "Succesfully register new category". If the save is unsuccessful, return an error

		message using danger alert indicating the reason.
5	Click "Save" button to verify saved changes.	Once successfully saved, the saved category must display in the table of the Category List page (Backend works). The system should then navigate the user to the category list page with a success alert. If the save is unsuccessful, return an error message using danger alert such as 'Category name already exists' or a similar notice.
3	Check if the Category Registration Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.

Category Update		
1	Click "edit" icon in the action column seen in the table to verify if the page for updating category is exist.	User must redirect to the page of updating category details.
2	Verify the original value of the fields.	The user must see all fields populated with the values from the originally registered category.
3	Verify fields that can be change.	Only Category name and Icon can be change. Dropdown for category type must be disable.
4	Verify validation for missing value on required fields.	User must not be able to click the "Save" button and error message must display "<field name> is required" in its field.
5	Click "Save" button to verify saved changes.	Once successfully saved, the category details must be automatically updated in the tables and on the individual detail page. The system should then navigate the user to the category list page with a success alert message of "Category updated successfully". If the save is unsuccessful, return an error message using danger alert such as 'Category name already exists' or a similar notice.
5	Check if the Category Update Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.

Category Delete		
1	Click "delete" icon in the action column seen in the table to verify if modal for deletion confirmation will show.	User must see Delete Confirmation Modal.
2	Verify all items linked to the category before deletion.	<ul style="list-style-type: none"> 1. If category is link into asset, components, accessories, and consumables, deletion must not be successful. Return a message why unsuccesful. 2. If nothing is link to category, system must return successful deletion of category and automatically update the category table.
3	Check if the Category Delete Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Supplier Context		
1	Click "More" tab then click "Suppliers" to verify the routing page.	The user must redirect to the "Supplier" page.

2	Verify Supplier Page.	<p>1. Category Page must show tables that has the following columns:</p> <ul style="list-style-type: none"> - Checkbox - Picture - Name - Address - City - State - Zip - Country - Contact - Phone Number - Edit Icon - View Icon - Delete Icon <p>2. Above columns, table must have headers, buttons, and filters like the following:</p> <p>Suppliers(Quantity Register)</p> <ul style="list-style-type: none"> - Search Bar - Filters (city, state, country, contact) - "+ New" button to register new supplier. It must be clickable and has hover. - There must be export button. It must be clickable and has hover. - Once export button is click, it must download xlsx format of file to user file manager <p>3. The table must have pagination and the list must be responsive to pagination list number.</p> <p>4. While typing for search supplier (all column values are searchable), user must be able to see all the result of the search value. If no record found, it must return a message of "No suppliers found." in the table.</p> <p>5. The list must update based on the selected filters when the user uses the filter feature (still sorted alphabetically.)</p>
---	------------------------------	---

4	Click "export" button to verify if it will download xlsx format of file.	System must automatically download xlsx format of file with the records of suppliers.
3	Check if the Supplier Page supports scrolling.	User can scroll to view all content without layout issues or cutoffs
4	Check if the Supplier Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.

Supplier Registration

1	Click "+ New" button to verify if supplier registration page exist.	User must redirect to the supplier registration page.
2	Verify Supplier Registration Page.	<p>1. The Supplier Registration page must have header that contains the following:</p> <ul style="list-style-type: none"> - Link "Supplier" to navigate on list of Supplier. Must be seen as Supplier / New Supplier - Import button with Icon and must be clickable and has hover - Has line separator to separate headers and form <p>2. Import button must allowed xlsx file only</p> <p>3. The page must have supplier registration form that contains the following:</p> <ul style="list-style-type: none"> - Supplier Name. Textfield and required - Address. Optional - City - Zip - State - Country

		<ul style="list-style-type: none"> - Contact Name - Phone Number - Fax - Email - URL - Notes - Logo. Must accept png and jpeg only. Max of 5 mb - Save Button. Must be enable only if required fields are filled. <p>4. There must be validation if supplier already existed, missing value on required fields, and successful supplier registration.</p> <p>5. Once Supplier successfully registered, user must automatically navigate to list of supplier with success alert message of "Successfully register new supplier". If the save is unsuccessful, return an error message using danger alert indicating the reason.</p>
3	<p>Check if the Category Registration Page have MAP AMS Footer.</p>	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Supplier Update		
1	Click "edit" icon in the action column seen in the table to verify if the page for updating supplier is exist.	User must redirect to the page of updating supplier details.

2	Verify the original value of the fields.	The user must see all fields populated with the values from the originally registered supplier.
3	Verify header.	The header of edit supplier must be same of register having - Supplier / Update Supplier - Name of Supplier. - Delete Button
4	Verify validation for missing value on required fields.	User must not be able to click the "Save" button and error message must display "<field name> is required" in its field.
5	Click "Save" button to verify saved changes.	Once successfully saved, the supplier details must be automatically updated in the tables and on the individual detail page. The system should then navigate the user to the supplier list page with a success alert message of "Supplier updated successfully". If the save is unsuccessful, return an error message using danger alert such as 'Supplier name already exists' or a similar notice.
5	Check if the Category Update Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Supplier Delete		
1	Click "delete" icon in the action column seen in the table to verify if	User must see Delete Confirmation Modal.

	modal for deletion confirmation will show.	
2	Click "delete" button to verify deletion status.	<p>1. For a successful deletion: The user must see a success alert stating, "Supplier deleted successfully!"</p> <p>2. For failed deletion: The user must see a danger alert explaining the reason why the deletion failed.</p>
3	Verify all items linked to the category before deletion.	<p>1. If supplier is link into asset, components, accessories, and consumables, deletion must not be successful. Return a message why unsuccesful.</p> <p>2. If nothing is link to category, system must return successful deletion of category and automatically update the category table.</p>

Supplier View Full Details

1	Click "view" button/icon in the action column seen in the table to verify the page of full details.	User must redirect to the supplier detail page.
2	Verify Supplier Details Page.	<p>1. Supplier detail page must have header that contains:</p> <ul style="list-style-type: none"> - Picture (If applicable) - Supplier(link) / View Supplier - Name of Supplier - Edit Button - Delete Button

		<p>2. Supplier details must have tab contains:</p> <ul style="list-style-type: none"> - Details in form - Assets (Table contains assets that has this supplier) <ul style="list-style-type: none"> - Components (Table contains assets that has this supplier) - Accessories (Table contains assets that has this supplier) - Consumables (Table contains assets that has this supplier)
3	Check if the Supplier Details Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.

Manufacturer Context		
1	Click "More" tab then check if the "Manufacturers" menu is exist in the dropdown context.	User must see the "Manufacturers" menu in the dropdown choices.
2	Click "More" tab then click "Manufacturers" to verify the routing page.	User must navigate to the "Manufacturers" page.

3	Verify Manufactuerer Page.	<p>1. Manufactuerer Page must show tables that has the following columns:</p> <ul style="list-style-type: none"> - Checkbox - Picture - Name - URL - Support URL - Phone Number - Email - Notes - Edit Icon - Delete Icon <p>2. Above columns, table must have headers, buttons, and filters like the following:</p> <ul style="list-style-type: none"> - Manufacturers (Quantity Register) - Search Bar - Filters (Name and must be sorted alphabetically) - "+ New" button to register new manufacturer. It must be clickable and has hover. - There must be export button. It must be clickable and has hover. <p>3. The table must have pagination and the list must be responsive to paginationlist number.</p> <p>4. While typing for search manufacturer (all column values are searchable), user must be able to see all the result of the search value. If no record found, it must return a message of "No manufacturer found." in the table.</p> <p>5. The list must update based on the selected filters when the user uses the filter feature (still sorted alphabetically.)</p>
4	Click "export" button to verify if it will download xlsx format of file.	System must automatically download xlsx format of file with the records of manufacturers.

5	Check if the Manufacturer Page supports scrolling.	User can scroll to view all content without layout issues or cutoffs
6	Check if the Manufacturer Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Manufacturer Registration		
1	Click "+ New" button to verify if supplier registration page exist.	User must navigate to the manufacturer registration page.
2	Verify Manufacturer Registration Page.	<p>1. The Manufacturer Registration page must have header that contains the following:</p> <ul style="list-style-type: none"> - Link "Manufacturer" to navigate on list of Manufacturer. Must be seen as Manufacturers / New Manufacturer - Import button with Icon and must be clickable and has hover. The Import button must allowed xlsx file only, validate the column structure and data types per column, return error messages for any invalid entries, and populate the fields with the file's values if no errors are found. - Has line separator to seperate headers and form <p>2. Import button must allowed xlsx file only</p> <p>3. The page must have manufacturer registration form that contains the following:</p> <ul style="list-style-type: none"> - Manufacturer Name. Textfield and required - URL - Support URL - Phone Number - Support Email - Notes

		<ul style="list-style-type: none"> - Logo. Must accept png and jpeg only. Max of 5 mb - Save Button. Must be enable only if required fields are filled. 4. There must be validation if manufacturer already existed, missing value on required fields, and successful manufacturer registration. 5. Once Manufacturer successfully registered, user must automatically navigate to list of manufacturer with success alert message of "Successfully register new manufacturer". If the save is unsuccessful, return an error message using danger alert indicating the reason.
3	Check if the Manufacturer Registration Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Manufacturer Update		
1	Click "edit" icon in the action column seen in the table to verify if the page for updating manufacturer is exist.	User must navigate to the page of updating manufacturer details.

2	Verify the original value of the fields.	The user must see all fields populated with the values from the originally registered manufacturer.
3	Verify header.	The header of edit manufacturer must be same of register having - Manufacturers / Update Manufacturer - Name of Manufacturer. - Delete Button
4	Verify validation for missing value on required fields.	User must not be able to click the "Save" button and error message must display "<field name> is required" in its field.
5	Click "Save" button to verify saved changes.	Once successfully saved, the manufacturer details must be automatically updated in the tables and on the individual detail page. The system should then navigate the user to the manufacturer list page with a success alert message of "Manufacturer updated successfully. If the save is unsuccessful, return an error message using danger alert such as 'Manufacturer name already exists' or a similar notice.
6	Check if the Category Update Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Manufacturer Delete		
1	Click "delete" icon in the action column seen in the table to verify if modal for deletion confirmation will show.	User must see Delete Confirmation Modal.

2	Click "delete" button to verify deletion status.	<p>1. For a successful deletion: The user must see a success alert stating, "Manufacturer deleted successfully!"</p> <p>2. For failed deletion: The user must see a danger alert explaining the reason why the deletion failed.</p>
3	Verify all items linked to the manufacturer before deletion.	<p>1. If manufacturer is link into asset, components, accessories, and consumables, deletion must not be successful. Return a message why unsuccesful.</p> <p>2. If nothing is link to manufacturer, system must return successful deletion of manufacturer and automatically update the manufacturer table.</p>
Status Context		
1	Click "More" tab then check if the "Stuses" menu is exist in the dropdown context.	User must see the "Statuses" menu in the dropdown choices.
2	Click "More" tab then click "Statuses" to verify the routing page.	User must navigate to the "Statuses" page.
3	Verify Statuses Page.	<p>1. Statuses Page must show tables that has the following columns:</p> <ul style="list-style-type: none"> - Checkbox - Name

		<ul style="list-style-type: none"> - Type - Notes - Assets (how many uses) - Edit Icon - Delete Icon <p>2. Above columns, table must have headers, buttons, and filters like the following:</p> <ul style="list-style-type: none"> - Statuses (Quantity Register) - Search Bar - Filters (Sort by greatest to least used) - "+ New" button to register new statuses. It must be clickable and has hover. - There must be export button. It must be clickable and has hover. <p>3. The table must have pagination and the list must be responsive to paginationlist number.</p> <p>4. While typing for search status (all column values are searchable), user must be able to see all the result of the search value. If no record found, it must return a message of "No status found." in the table.</p> <p>5. The list must update based on the selected filters when the user uses the filter feature.</p>
4	Click "export" button to verify if it will download xlsx format of file.	System must automatically download xlsx format of file with the records of statuses.
5	Check if the Statuses Page supports scrolling.	User can scroll to view all content without layout issues or cutoffs
6	Check if the Statuses Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.

Statuses Registration		
1	Click "+ New" button to verify if status registration page exist.	User must navigate to the status registration page.
2	Verify Status Registration Page.	<p>1. The Manufacturer Registration page must have header that contains the following:</p> <ul style="list-style-type: none"> - Link "Statuses" to navigate on list of Statuses. Must be seen as Statuses / New Status - Import button with Icon and must be clickable and has hover. The Import button must allowed xlsx file only, validate the column structure and data types per column, return error messages for any invalid entries, and populate the fields with the file's values if no errors are found. - Has line separator to seperate headers and form <p>2. Import button must allowed xlsx file only</p> <p>3. The page must have status registration form that contains the following:</p> <ul style="list-style-type: none"> - Status Name. Textfield and required - Status Type. Dropdown and required - Notes - Save Button. Must be enable only if required fields are filled. <p>4. There must be validation-if Status already existed, missing value on required fields, and successful status registration.</p> <p>5. Once the status is successfully registered, the system must automatically navigate to the list of statuses and display a success alert. If the save is unsuccessful, return an error message using danger alert indicating the reason.</p>

3	Check if the Status Registration Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Status Update		
1	Click "edit" icon for status that is not yet use in the action column seen in the table to verify if the page for updating status is exist.	User must navigate to the page of updating status details.
2	Verify the original value of the fields.	The user must see all fields populated with the values from the originally registered status.
3	Verify header.	The header of edit status must be same of register having <ul style="list-style-type: none"> - Statuses / Update Status - Name of Status. - Delete Button
4	Verify validation for missing value on required fields.	User must not be able to click the "Save" button and error message must display "<field name> is required" in its field.
5	Verify save button behavior.	Save Button. Must be enable only if required fields are filled.
6	Click "Save" button to verify saved changes.	Once successfully saved, the status details must be automatically updated in the tables and on the individual detail page. The system should then navigate the user to the status list page with a

		success alert message of "Category updated successfully". If the save is unsuccessful, return an error message using danger alert such as 'Status name already exists' or a similar notice.
7	Check if the Status Update Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
8	Verify "edit" icon for used status.	Edit icon must be disabled and has a tooltip of "This status is currently in use and cannot be edited." when hover.
9	Verify "edit" icon for system default status.	Edit icon must be disabled and has a tooltip of "This is a system default status label, and cannot be edited." when hover.
Status Delete		
1	Click "delete" icon for status that is not yet use in the action column seen in the table to verify if modal for deletion confirmation will show.	User must see Delete Confirmation Modal.
2	Click "delete" button to verify deletion status.	<ol style="list-style-type: none"> For a successful deletion: The user must see a success alert stating, "Status deleted successfully!" For failed deletion: The user must see a danger alert explaining the reason why the deletion failed.

3	Verify the "delete" icon for used status.	Delete icon must be disabled and has a tooltip of "This status is currently in use and cannot be deleted." when hover.
4	Verify the "delete" icon for system default status.	Delete icon must be disabled and has a tooltip of "This is a system default status label, and cannot be deleted." when hover.

Depreciation Context

1	Click "More" tab then check if the "Depreciations" menu is exist in the dropdown context.	User must see the "Depreciations" menu in the dropdown choices.
2	Click "More" tab then click "Depreciations" to verify the routing page.	User must navigate to the "Depreciations" page.
3	Verify Depreciations Page.	<ul style="list-style-type: none"> 1. Depreciations Page must show tables that has the following columns: <ul style="list-style-type: none"> - Checkbox - Name - Duration - Minimum Value - Edit Icon - Delete Icon 2. Above columns, table must have headers, buttons, and filters like the following: <ul style="list-style-type: none"> - Depreciation (Quantity Register) - Search Bar - Filters (Sort by greatest to least value) - "+ New" button to register new depreciation. It must be clickable and has hover. - There must be export button. It must be clickable and has hover.

		<p>3. The table must have pagination and the list must be responsive to paginationlist number.</p> <p>4. While typing for search depreciation (all column values are searchable), user must be able to see all the result of the search value. If no record found, it must return a message of "No depreciation found." in the table.</p> <p>5. The list must update based on the selected filters when the user uses the filter feature.</p>
4	Click "export" button to verify if it will download xlsx format of file.	System must automatically download xlsx format of file with the records of statuses.
5	Check if the Depreciations Page supports scrolling.	User can scroll to view all content without layout issues or cutoffs
6	Check if the Depreciations Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.

Depreciation Registration		
1	Click "+ New" button to verify if depreciation registration page exist.	User must navigate to the depreciation registration page.
2	Verify Depreciation Registration Page.	<p>1. The Depreciation Registration page must have header that contains the following:</p> <ul style="list-style-type: none"> - Link "Depreciations" to navigate on list of Depreciations. Must be seen as Depreciations / New Depreciation - Import button with Icon and must be clickable and has hover - Has line separator to seperate headers and form <p>2. Import button must allowed xlsx file only. The Import button must allowed xlsx file only, validate the column structure and data types per column, return error messages for any invalid entries, and populate the fields with the file's values if no errors are found.</p> <p>3. The page must have depreciation registration form that contains the following:</p> <ul style="list-style-type: none"> - Depreciation Name. Textfield and required - Duration. Numeric in months and required - Minimum Value. Numeric and has currency - Save Button. Must be enable only if required fields are filled. <p>4. There must be validation if Depreciation already existed, missing value on required fields, and successful</p>

		<p>status registration.</p> <p>5. Once the depreciationn is successfully registered, the system must automatically navigate to the list of deprecations and display a success alert. If the save is unsuccessful, return an error message using danger alert indicating the reason.</p>
3	Check if the Depreciations Registration Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Depreciation Update		
1	Click "edit" icon in the action column seen in the table to verify if the page for updating depreciation is exist.	User must navigate to the page of updating depreciation details.

2	Verify the original value of the fields.	The user must see all fields populated with the values from the originally registered depreciation.
3	Verify header.	The header of edit depreciation must be same of register having - Depreciations / Update Depreciation - Name of Depreciation. - Delete Button
4	Verify validation for missing value on required fields.	User must not be able to click the "Save" button and error message must display "<field name> is required" in its field.
5	Click "Save" button to verify saved changes.	Once successfully saved, the depreciation details must be automatically updated in the tables and on the individual detail page. The system should then navigate the user to the depreciation list page with a success alert. If the save is unsuccessful, return an error message using danger alert such as 'Depreciation name already exists' or a similar notice.
6	Check if the Depreciation Update Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Depreciation Delete		

1	Click "delete" icon in the action column seen in the table to verify if modal for deletion confirmation will show.	User must see Delete Confirmation Modal.
2	Click "delete" button to verify deletion status.	<ul style="list-style-type: none"> 1. For a successful deletion: The user must see a success alert stating, "Depreciation deleted successfully!" 2. For failed deletion: The user must see a danger alert explaining the reason why the deletion failed.
3	Verify all items linked to the depreciation before deletion.	<ul style="list-style-type: none"> 1. If depreciation is being used, deletion must not be successful. Return an error message stating that the depreciation is used. 2. If nothing is link to depreciation, system must return successful deletion of depreciation and automatically update the depreciation table.

Recycle Bin Context

1	Click "More" tab then check if the "Recycle Bin" menu is exist in the dropdown context.	User must see the "Recycle Bin" menu in the dropdown choices.
2	Click "More" tab then click "Recycle Bin" to verify the routing page.	User must navigate to the "Recycle Bin" page.
3	Verify Recycle Bin Page.	<ul style="list-style-type: none"> 1. Recycle Bin page must show tables that has all columns of deleted item

		<p>2. Above columns, table must have headers, buttons, and filters like the following:</p> <ul style="list-style-type: none"> - Recycle Bin (Quantity Register) - Search Bar - Filters: filters for overall columns <p>3. There must be tab that has assets, accessories, components, and consumables and each tab show table and all columns of deleted item.</p> <p>4. The table must have pagination and the list must be responsive to pagination list number.</p>
4	Click "export" button to verify if it will download xlsx format of file.	System must automatically download xlsx format of file with the records of active tab of the recycle bin.
5	Check if the Recycle Bin Page supports scrolling.	User can scroll to view all content without layout issues or cutoffs
6	Check if the Recycle Bin Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Asset Model / Products		
1	Verify RBAC	Users with the Admin or Operator role can access the page containing records of Asset Models / Products.

2	Click "Asset" menu in the navigation bar then check if the "Asset Model" is exist in the dropdown.	User must see the "Asset Model" menu in the dropdown choices.
3	Click "Asset" menu in the navigation bar then click "Asset Model" to verify the routing page.	User must navigate to the "Asset Model" page.
4	Verify Asset Model / Products Page.	<p>1. Asset Model Page must show tables that has the following columns:</p> <ul style="list-style-type: none"> - Checkbox - Asset Model Picture - Asset Model Name - Category - Model Number - End of Life - Manufacturer - Depreciation - Default Cost - Minimum Quantity - View Details Icon/button - Edit Icon/button - Delete Icon/button <p>2. Above columns, table must have headers, buttons, and filters like the following:</p> <ul style="list-style-type: none"> - Asset Model / Products (Quantity Register) - Search Bar - Filters (Show Modal for multiple filtering) - "+ New" button to register new asset model/products. It must be clickable and has hover. - There must be export button. It must be clickable and has hover. <p>3. The table must have pagination and the list must be responsive to pagination list number.</p> <p>4. While typing for search asset model (all column values are searchable), user must</p>

		<p>be able to see all the result of the search value. If no record found, it must return a message of "No asset model found." in the table.</p> <p>5. The list must update based on the selected filters when the user uses the filter feature.</p>
5	Click "export" button to verify if it will download xlsx format of file.	System must automatically download xlsx format of file with the records of asset model / products.
6	Check if the Asset Model / Products Page supports scrolling.	User can scroll to view all content without layout issues or cutoffs

7	Check if the Asset Model / Products Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Asset Model / Product Registration		
1	Verify RBAC	Users with the Admin role can access the registration page for Asset Models and should be the only ones who can see the '+ New' button.
2	Click "+ New" button to verify if Aset Model / Product registration page exist.	User must navigate to the Asset Model / Product registration page.

3	<p>Verify Asset Model / Product Registration Page.</p>	<ol style="list-style-type: none"> 1. The Depreciation Registration page must have header that contains the following: <ul style="list-style-type: none"> - Link "Asset Model" to navigate on list of Asset Models. Must be seen as Asset Model / New Asset Model - Import button with Icon and must be clickable and has hover. The Import button must allowed xlsx file only, validate the column structure and data types per column, return error messages for any invalid entries, and populate the fields with the file's values if no errors are found. - Has line separator to separate headers and form 2. Import button must allowed xlsx file only 3. The page must have Asset Model / Product registration form that contains the following: <ul style="list-style-type: none"> - Product Name. Required, with maximum input length. - Category. Required, dropdown, with a "+" button to immediately add a new category. - Manufacturer. Dropdown, with a "+" button to immediately add a new manufacturer. - Depreciation. Dropdown, with a "+" button to immediately add a new depreciation value. - Model Number. Text field, with maximum input length. - End of Life. Field, only accepts positive integer input, with "months" displayed at the end. - Default Purchase Cost. Field, only accepts positive double values, must show "PHP" before field. - Minimum Quantity. Field, only accepts positive integer input. - CPU. Dropdown. - GPU. Dropdown.
---	---	--

		<ul style="list-style-type: none"> - Operating System. Dropdown. - RAM. Dropdown. - Screen Size. Dropdown. - Storage Size. Dropdown. - Archive Model. Switch Button. - Notes. Text field, with maximum input length. - Image Upload. Must accept JPEG and PNG formats, maximum 5 MB. - Save Button. Must be enable only if required fields are filled. <p>4. There must be validation if Asset Model already existed, missing value on required fields, and successful Asset Model registration.</p> <p>5. Once the Asset Model is successfully registered, the system must automatically navigate to the list of Asset Model and display a success alert. If the save is unsuccessful, return an error message using danger alert indicating the reason.</p>
4	Check if the Asset Model / Product Registration Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Asset Model / Product Update		
1	Verify RBAC	Users with the Admin role can access the page for updating Asset Models / Products details and should be the only ones who can see the 'Edit' icon/button in the table.

2	Click "edit" icon in the action column seen in the table to verify if the page for updating asset model / product is exist.	User must navigate to the page of updating Asset Model / Product details.
3	Verify the original value of the fields.	The user must see all fields populated with the values from the originally registered asset model / product.
4	Verify header.	The header of edit Asset Model / Product must be same of register having <ul style="list-style-type: none"> - Asset Model / Update Asset Model - Name of Asset Model / Product. - Clone Button - Delete Button
5	Verify validation for missing value on required fields.	User must not be able to click the "Save" button and error message must display "<field name> is required" in its field.
6	Click "Save" button to verify saved changes.	Once successfully saved, the asset model/product details must be automatically updated in the tables and on the individual detail page. The system should then navigate the user to the Asset Model / Products page with a success alert message of "Product updated successfully". If the save is unsuccessful, return an error message, such as 'Model name already exists,' or similar notice.
7	Click 'Clone' button to verify navigation.	User must navigate to the "Asset Model Registration" page with "Asset Model name (cloned)" as Asset model name.

8	Check if the Asset Model / Product Update Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Asset Model / Product Delete		
1	Verify RBAC	Only user with Admin role can delete existing Asset Model.
2	Click "delete" icon in the action column seen in the table to verify if modal for deletion confirmation will show.	User must see Delete Confirmation Modal.
2	Click "delete" button to verify deletion status.	<ul style="list-style-type: none"> 1. For a successful deletion: The user must see a success alert stating, "Asset Model / Product deleted successfully!" 2. For failed deletion: The user must see a danger alert explaining the reason why the deletion failed.
4	Verify all items linked to the depreciation before deletion.	<ul style="list-style-type: none"> 1. If Asset Model / Product is being used such link into an asset, deletion must not be successful. Return an error message stating that the Asset Model is used. 2. If nothing is link to Asset Model / Product, system must return successful deletion of Asset Model / Product and automatically update the Asset Model / Product table.

View Page Asset Model		
1	Verify RBAC	Users with the Admin or Operator role can access the page containing records of Asset Models / Products and view full details of each Asset Model.
2	Click "Asset" menu in the navigation bar then check if the "Asset Model" is exist in the dropdown.	User must see the "Asset Model" menu in the dropdown choices.
3	Click "Asset" menu in the navigation bar then click "Asset Model" to verify the routing page.	User must navigate to the "Asset Model" List Page.
4	Verify if Action Button for viewing an Asset Model is exist in the page.	User must see 'Eye' Icon/button for viewing an asset model.
5	Click 'Eye' Icon/button in the action column seen in the table to verify navigation.	User must navigate to the "Asset Model Detail" Page.
6	Verify Asset Model Detail Page .	<p>1. The Asset Model Detial Page must have a header that contains:</p> <ul style="list-style-type: none"> - Picture. (If applicable. Use default image for non-picture.) - Link "Asset Model" to navigate on list of Asset Models. Must be seen as Asset Model / Show Asset Model - Name of Asset Model - Clone Button - Edit Button

		<ul style="list-style-type: none"> - Delete Button <p>2. Asset Model Details section must have tabs containing:</p> <ul style="list-style-type: none"> - Details / About (in form view). - Assets (table containing list of assets linked to this Asset Model). <p>3. Form Details must include the following:</p> <ul style="list-style-type: none"> - Header: "Asset Model Details". - Product Name. - Model Number. - Category. - Manufacturer. - Depreciation. - End of Life. - Default Purchase Cost. - Minimum Quantity. - CPU. - GPU. - Operating System. - RAM. - Screen Size. - Storage Size. - Notes. - Created At. - Updated At.
7	Check if the Asset Model Detail Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Asset Model Advance Search/Filter		
1	Verify RBAC	Users with the Admin or Operator role can perform advance search through multiple search filters.

2	Click "Asset" menu in the navigation bar then check if the "Asset Model" is exist in the dropdown.	User must see the "Asset Model" menu in the dropdown choices.
3	Click "Asset" menu in the navigation bar then click "Asset Model" to verify the routing page.	User must navigate to the "Asset Model" List Page.
4	Verify if "Filters" button for advance filtering exist in the Asset Model / Product Table.	User must see "filters" button in the Asset Model / Product for advance filtering.
5	Click "Filter" button to verify modal.	<p>1. User must see modal that has the following:</p> <ul style="list-style-type: none"> - Category (Dropdown) - Manufacturer (Dropdown) - Depreciation (Dropdown) - Achieved (Dropdown y/s) - In used by Asset (Dropdown y/s) - Created At (From) - Created At (To) - Updated At (From) - Updated At (To) - Connectivity (Dropdown) - CPU (Dropdown) - GPU (Dropdown) - RAM (Dropdown) - Operating System (Dropdown) - Screen Size (Dropdown) - Storage Size (Dropdown) - Reset Filter Button - Apply Filter Button <p>2. If Reset filter is clicked, fields must remove all data entered.</p> <p>3. If apply filter is clicked, modal must close and list of data on a table must follow filtered data.</p>

Asset		
1	Verify RBAC	Users with the Admin or Operator role can access the page containing records of Assets
2	Check if the navigation bar has an Asset menu.	User must see the "Asset" menu in the dropdown choices.
3	Click "Asset" menu in the navigation bar	User must navigate to the "Asset" page.

4	Verify Asset Page.	<p>1. Asset Model Page must show tables that has the following columns:</p> <ul style="list-style-type: none"> - Checkbox + Bulk action for multiple records - Image - ID - Name - Category - Status - Checkin and Checkout buttons - View Details Icon/button - Edit Icon/button - Delete Icon/button <p>2. Above columns, table must have headers, buttons, and filters like the following:</p> <ul style="list-style-type: none"> - Asset (Quantity Register) - Search Bar - Filters (Show Modal for multiple filtering) - "+ New" button to register new asset. It must be clickable and has hover. - There must be export button. It must be clickable and has hover. <p>3. The table must have pagination and the list must be responsive to pagination list number.</p> <p>4. While typing for search asset model (all column values are searchable), user must be able to see all the result of the search value. If no record found, it must return a message of "No asset found." in the table.</p> <p>5. The list must update based on the selected filters when the user uses the filter feature.</p>
5	Click "export" button to verify if it will download xlsx format of file.	System must automatically download xlsx format of file with the records of asset.

6	Check if the Asset Page supports scrolling.	User can scroll to view all content without layout issues or cutoffs
7	Check if the Asset Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Asset Registration		
1	Verify RBAC	Users with the Admin role can access the registration page for Asset and should be the only ones who can see the '+ New' button.
2	Click "+ New" button to verify if Asset registration page exist.	User must navigate to the Asset registration page.

3	Verify Asset Registration Page.	<ol style="list-style-type: none"> 1. The Asset Registration page must have header that contains the following: <ul style="list-style-type: none"> - Link "Asset" to navigate on list of Asset Models. Must be seen as Assets / New Asset - Import button with Icon and must be clickable and has hover. The Import button must allowed xlsx file only, validate the column structure and data types per column, return error messages for any invalid entries, and populate the fields with the file's values if no errors are found. - Has line separator to separate headers and form 2. Import button must allowed xlsx file only 3. The page must have Asset registration form that contains the following: <ul style="list-style-type: none"> - Asset ID. Required, autogenerate, and has max input - Product Dropdown. Required - Status Dropdown. Required with a "+" button to immediately add a new status. - Supplier Dropdown with a "+" button to immediately add a new supplier. - Location Dropdown with a "+" button to immediately add a new location - Asset Name, Field and has max input - Serial Number. Field, alphanumeric input and has max input - Warranty Expiration Date. - Order Number. Field and has max input - Purchase date. Date picker - Purchase Cost. Field and float numeric input - Disposal Status. Dropdown - Schedule Audit Date. Date picker - Notes. Field and has max input - Image Upload. Maximum of 5mb and only accept jpeg and png. - Save Button. Must be enable only if required fields are filled. 4. There must be validation if Asset
---	--	--

		<p>already existed, missing value on required fields, and successful Asset registration.</p> <p>5. Once the Asset is successfully registered, the system must automatically navigate to the list of assets and display a success alert. If the save is unsuccessful, return an error message using danger alert indicating the reason.</p>
4	Check if the Asset Registration Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Asset Update		
1	Verify RBAC	Users with the Admin role can access the page for updating Asset details and should be the only ones who can see the 'Edit' icon/button in the table.
2	Click "edit" icon in the action column seen in the table to verify if the page for updating asset is exist.	User must navigate to the page of updating Asset details.

3	Verify the original value of the fields.	The user must see all fields populated with the values from the originally registered asset.
4	Verify header.	The header of edit Asset must be same of register having <ul style="list-style-type: none"> - Assets / Update Asset - Name of Asset. - Clone Button - Delete Button
5	Verify validation for missing value on required fields.	User must not be able to click the "Save" button and error message must display "<field name> is required" in its field.
6	Click "Save" button to verify saved changes.	Once successfully saved, the asset details must be automatically updated in the tables and on the individual detail page. The system should then navigate the user to the asset list page with a success alert. If the save is unsuccessful, return an error message using danger alert such as 'Asset name already exists' or a similar notice.
7	Click 'Clone' button to verify navigation.	User must navigate to the "Asset Registration" page with "Asset name (cloned)" as Asset name.
8	Check if the Asset Update Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.

Asset Delete		
1	Verify RBAC	Only user with Admin role can delete existing Asset.
2	Click "delete" icon in the action column seen in the table to verify if modal for deletion confirmation will show.	User must see Delete Confirmation Modal.
2	Click "delete" button to verify deletion status.	<ul style="list-style-type: none"> 1. For a successful deletion: The user must see a success alert stating, "Asset deleted successfully!" 2. For failed deletion: The user must see a danger alert explaining the reason why the deletion failed.
4	Verify all items linked to the depreciation before deletion.	<ul style="list-style-type: none"> 1. If Asset is being used such link into a check-in and check-out or simply deployed status, deletion must not be successful. Return an error message stating that the Asset is used. 2. If nothing is link to Asset, system must return successful deletion of Asset and automatically update the Asset table.
View Page Asset		
1	Verify RBAC	Users with the Admin or Operator role can access the page containing records of Asset and view full details of each Asset.

2	Check if the navigation bar has an Asset menu.	User must see the "Asset" menu in the dropdown choices.
3	Click "Asset" menu in the navigation bar to verify the routing page.	User must navigate to the "Asset" List Page.
4	Verify if Action Button for viewing an Asset is exist in the page.	User must see 'Eye' Icon/button for viewing an asset details.
5	Click 'Eye' Icon/button in the action column seen in the table to verify navigation.	User must navigate to the "Asset Detail" Page.

6	Verify Asset Detail Page.	<p>1. The Asset Detial Page must have a header that contains:</p> <ul style="list-style-type: none"> - Picture. (If applicable) - Link "Asset" to navigate on list of Assets. <p>Must be seen as Assets / Show Asset</p> <ul style="list-style-type: none"> - Name of Asset - Clone Button - Edit Button - Delete Button <p>2. Asset Details section must have tabs containing:</p> <ul style="list-style-type: none"> - Details / About (in form view). - History (table containing list of check-in and check-out history of asset) <ul style="list-style-type: none"> = Data must in the table: <ul style="list-style-type: none"> = Checkout Date, Expected Return Date, Condition, Photos, Notes, User, Confirmation Email Sent, Confirmation Receive of Asset - Components (A table contains list of components check-out to asset) - Maintenance / Repair (A table consist history of maintenance of asset) - Audits (Same tab and tables that can be seen on audits) - Additional Files (A table that has additional attachment related to asset, can add additional file there) <p>3. Form Details must include the following:</p> <ul style="list-style-type: none"> - QR Code for Asset Information Scanning - Beside QR, there must be the following Information: <ul style="list-style-type: none"> = Serial Number = Asset ID = Property of MAP Active Philippines - There must be a print button of QR, it must navigate / open page to printing asset qrcode - Asset ID - Asset Serial Number - Asset Model / Product Name
---	---------------------------	--

	<ul style="list-style-type: none">- Category- Supplier- Manufacturer- Depreciation Type- Fully Depreciated (Date, How many depreciation year, months, and weeks left)- Location- Warranty (Date, How many warranty year, months, and weeks left)- End of life (Date, How many EoL year, months, and weeks left)- Order Number- Purchase Date- Purchase Cost- If smartphone<ul style="list-style-type: none">= IMEI Number= Connectivity (Product)= Operating System (Product)= Storage Size (Product)- If Laptop<ul style="list-style-type: none">= SSD Encryption Status= CPU (Product)= GPU (Product)= Operating System (Product)= RAM (Product)= Screen Size (Product)= Storage Size (Product)- Notes- Created At- Updated At.
--	--

7	Check if the Asset Detail Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Asset Advance Search/Filter		
1	Verify RBAC	Users with the Admin or Operator role can perform advance search through multiple search filters.
2	Check if the navigation bar has an Asset menu.	User must see the "Asset" menu in the dropdown choices.
3	Click "Asset" menu in the navigation bar to verify the routing page.	User must navigate to the "Asset" List Page.
4	Verify if "Filters" button for advance filtering exist in the Asset Table.	User must see "filters" button in the Asset for advance filtering.

5	Click "Filter" button to verify modal.	<p>1. User must see modal that has the following:</p> <ul style="list-style-type: none"> - Asset ID. Numeric - Asset Model. Textfield - Status. Dropdown - Supplier. Dropdown - Location. Searchable dropdown - Asset Name. Textfield - Serial Number. Numeric - Warranty Expiration. Date - Order No. Numeric - Purchase Date. Date - Purchase Cost. Numeric - Reset Filter Button - Apply Filter Button <p>2. If Reset filter is clicked, fields must remove all data entered.</p> <p>3. If apply filter is clicked, modal must close and list of data on a table must follow filtered data.</p>

Asset Check-Out

1	Verify RBAC	Users can perform asset check-out once there is existing ticket request for asset check-out
2	Check if the navigation bar has an Asset menu.	User must see the "Asset" menu in the dropdown choices.
3	Click "Asset" menu in the navigation bar	User must navigate to the "Asset" page.

4	Check if the navigation bar has a Tickets Tabs menu.	User must see the "Tickets" tabs in the navigation bar.
5	Click "Tickets" menu in the navigation bar	User must navigate to the "Tickets" page.
6	Check if the Asset Page and Tickets Page have a "Check-out" button to perform asset checkout.	User must see "Check-out" button in both the Asset and Ticket tables on the page.
7	Click "Check-out" button seen in the table of both asset and ticket page to verify the routing.	User must navigate to the "Asset Check-out" Page.
8	Verify Asset Check-Out Page.	<p>1. The Asset Check-Out page must have header that contains the following:</p> <ul style="list-style-type: none"> - Link "Asset" to navigate on list of Assets. Must be seen as Assets / Checkout Asset - Below must be the AssetID - Has line separator to seperate headers and form <p>2. The page must have Asset Check-out form that contains the following fields:</p> <ul style="list-style-type: none"> - Checkout To as form name/header - Employee Field (Prefilled by ticket) - Location Field (Prefilled by Ticket) - Checkout Date (Prefilled by Ticket) - Expected Return Date (Prefilled by Ticket) - Condition Dropdown - Notes - Image Upload (Max of 5 mb notes and must accept only jpeg and png) - Save Button. Must be enable only if

		<p>required fields are filled.</p> <p>4. There must be validation if the ticket is expired or has an issue, missing value on required fields, and successful Asset Check-out.</p> <p>5. Once a checkout is successful, the system must automatically navigate to the list of Asset Checkout records/history of the specific asset and the records there must be updated.</p>
9	Check if the Asset Check-Out Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Asset Check-In		
1	Verify RBAC	Users can perform asset check-in once there is existing ticket request for asset check-in
2	Check if the navigation bar has an Asset menu.	User must see the "Asset" menu in the dropdown choices.
3	Click "Asset" menu in the navigation bar	User must navigate to the "Asset" page.
4	Check if the navigation bar has a Tickets Tabs menu.	User must see the "Tickets" tabs in the navigation bar.
5	Click "Tickets" menu in the navigation bar	User must navigate to the "Tickets" page.

6	Check if the Asset Page and Tickets Page have a "Check-in" button to perform asset checkout.	User must see "Check-in" button in both the Asset and Ticket tables on the page.
7	Click "Check-in" button seen in the table of both asset and ticket page to verify the routing.	User must navigate to the "Asset Check-in" Page.
8	Verify Asset Check-InPage .	<ul style="list-style-type: none"> 1. The Asset Check-In page must have header that contains the following: <ul style="list-style-type: none"> - Link "Asset" to navigate on list of Assets. Must be seen as Assets / Checkin Asset - Below must be the AssetID - Has line separator to seperate headers and form 2. The page must have Asset Check-in form that contains the following fields: <ul style="list-style-type: none"> - Check-in Date (Prefilled by ticket) - Status Dropdown - Condition Dropdown - Location - Notes Field - Image Upload (Max of 5 mb notes and must accept only jpeg and png) - Save Button. Must be enable only if required fields are filled. 4. There must be validation if the ticket is expired or has an issue, missing value on required fields, and successful Asset Check-in. 5. Once a checkin is successful, the system must automatically navigate to the list of Asset Checkin records/history of the specific asset and the records there must be updated.
9	Check if the Asset Check-In Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.

Tickets		
1	Verify RBAC	Users can perform action base on ticket subject (i.e asset checkout and asset checkin).
2	Click "Tickets" tab in the navigation bar	User must navigate to the "Tickets" page.
3	Check if the Tickets Page have a "Check-in" and "Check-out" button to perform base on ticket subject.	User must see "Check-in" button in both the Asset and Ticket tables on the page.
5	Verify Ticket Page and its behavior.	User must be able to view and perform actions based on the orginal criteria as of the moment.
6	Check if the Tickets Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Asset Repair		
1	Verify RBAC	Users with the Admin or Operator role can access the page containing records of Asset Repair.
2	Check if the navigation bar has a Repairs tab.	User must see the "Repairs" tab in the navigation bar.

3	Click "Repairs" tab in the navigation bar	User must navigate to the "Repairs" page.
4	Verify Asset Repair Page.	<p>1. Asset Repair Page must show tables that has the following columns:</p> <ul style="list-style-type: none"> - Checkbox (for multiple selection) - Asset ID - Asset Name - Repair Type - Repair Name - Start Date - End Date - Cost - Supplier - Notes - Attachment - Update Icon - Delete Icon <p>2. Above columns, table must have headers, buttons, and filters like the following:</p> <ul style="list-style-type: none"> - Asset Repairs (Quantity Register) - Search Bar - Filters (by type, status, date range, asset) - "+ New" button to register new asset repair. It must be clickable and has hover. - There must be export button. It must be clickable and has hover. <p>3. The table must have pagination and the list must be responsive to paginationlist number.</p> <p>4. While typing for search asset model (all column values are searchable), user must be able to see all the result of the search value. If no record found, it must return a message of "No asset repair found." in the table.</p> <p>5. The list must update based on the selected filters when the user uses the filter feature.</p>

5	Click "export" button to verify if it will download xlsx format of file.	System must automatically download xlsx format of file with the records of asset repairs.
6	Check if the Asset Repair Page supports scrolling.	User can scroll to view all content without layout issues or cutoffs
7	Check if the Asset Repair Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Asset Repair Registration		
1	Verify RBAC	Both the Admin and Operator users can access and record entries on the Asset Repair page.
2	Click "+ New" button to verify if Asset Repair registration page exist.	User must navigate to the Asset Repair registration page.

3	<p>Verify Asset Repair Registration Page.</p>	<ol style="list-style-type: none"> 1. The Asset Repair Registration page must have header that contains the following: - Link "Repair" to navigate on list of Asset Repair. Must be seen as Repairs / New Repair - Import button with Icon and must be clickable and has hover. The Import button must allowed xlsx file only, validate the column structure and data types per column, return error messages for any invalid entries, and populate the fields with the file's values if no errors are found. - Has line separator to separate headers and form 2. Import button must allowed xlsx file only 3. The page must have Asset Repair registration form that contains the following: <ul style="list-style-type: none"> - Asset. Dropdown and Required - Supplier. Dropdown - Maintenance Type. Dropdown and Required - Maintenance Name. Field and Required - Start Date. Datepicker and Required - End Date. Date Picker - Cost. Field and has php before the field. - Notes. Field and has max input - Upload file. Button for choosing file and note of maximum file upload: 300mb - Save Button. Must be enable only if required fields are filled. 4. There must be validation if Asset Repair already existed, missing value on required fields, and successful Asset Repair registration. 5. Once the Asset Repair is successfully registered, the system must automatically navigate to the list of Asset Repairs and display a success alert. If the save is unsuccessful, return an error message using danger alert indicating the reason.
---	--	--

4	Check if the Asset Repair Registration Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Asset Repair Update		
1	Verify RBAC	Both users admin and operator can update asset repair details. Both the Admin and Operator users can update asset repair details.
2	Click "edit" icon in the action column seen in the table to verify if the page for updating asset repair is exist.	User must navigate to the page of updating Asset Repair details.
3	Verify the original value of the fields.	The user must see all fields populated with the values from the originally registered asset repair.
4	Verify header.	The header of edit Asset must be same of register having <ul style="list-style-type: none"> - Repairs / Update Repair - Name of Repair Type - A line separator to separate headers and form.
5	Verify validation for missing value on required fields.	User must not be able to click the "Save" button and error message must display "<field name> is required" in its field.
5	Click "Save" button to verify saved changes.	Once successfully saved, the asset repair details must be automatically

		updated in the tables and on the individual detail page. The system should then navigate the user to the asset repair list page with a success alert. If the save is unsuccessful, return an error message using danger alert such as 'Repair name already exists' or a similar notice.
7	Check if the Asset Repair Update Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Asset Repair Delete		
1	Verify RBAC	Both users admin and operator can delete asset repair. Both the Admin and Operator users can delete asset repair.
2	Click "delete" icon in the action column seen in the table to verify if modal for deletion confirmation will show.	User must see Delete Confirmation Modal.
2	Click "delete" button to verify deletion status.	<ol style="list-style-type: none"> For a successful deletion: The user must see a success alert stating, "Repair deleted successfully!" and automatically update the list of Asset Repair Table. For failed deletion: The user must see a danger alert explaining the reason why the deletion failed.
Schedule Audit Registration		

1	Verify RBAC	Both the Admin and Operator users can access and record entries on the Schedule Audit page.
2	Click "Audit" tab in the navigation bar	User must navigate to the "Audit" page.
3	Click "Schedule Audit" button.	User must navigate to the "Schedule Audit" Page.
5	Verify Schedule Audit Page and its behavior.	User must be able to view and perform actions based on the orginal criteria from capstone 1.
6	Check if the Schedule Audit Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Schedule Audit Update		
1	Verify RBAC	Both the Admin and Operator users can access and update the record of Schedule Audit.
2	Click "Audit" tab in the navigation bar	User must navigate to the "Audit" page.

3	Click "Edit" icon seen in the table.	User must navigate to the "Schedule Audit" Page with populated registered values.
5	Verify Schedule Audit Page and its behavior.	User must be able to view and perform actions based on the orginal criteria from capstone 1.
6	Check if the Schedule Audit Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Schedule Audit Delete		
1	Verify RBAC	Both the Admin and Operator users can delete schedule audit.
2	Click "delete" icon in the action column seen in the table to verify if modal for deletion confirmation will show.	User must see Delete Confirmation Modal.
3	Click "delete" button to verify deletion status.	<ul style="list-style-type: none"> 1. For a successful deletion: The user must see a success alert stating, "Schedule Audit deleted successfully!" 2. For failed deletion: The user must see a danger alert explaining the reason why the deletion failed.
Perform Scheduled Audit		

1	Verify RBAC	Both the Admin and Operator users can access and perform Asset Audit on existing scheduled asset audit.
2	Click "Audit" tab in the navigation bar	User must navigate to the "Audit" page.
3	Click "Scheduled Audit" to verify the routing.	User must navigate to the "Scheduled Audit" Page containing record of schedule audit.
5	Click "Audit" icon seen in the table.	User must navigate to the "Perform Audit" Page where the registered schedule audit is already populated.
6	Check if the Perform Audit Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Audit		
1	Verify RBAC	Both the Admin and Operator users can access and perform actions on the Audit Page.
2	Click "Audit" tab in the navigation bar	User must navigate to the "Audit" page.
5	Verify Audit Page and its behavior.	User must be able to view and perform actions based on the original criteria from capstone 1.
6	Check if the Schedule Audit Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.

Components		
1	Verify RBAC	Users with the Admin or Operator role can access the page containing records of Components
2	Check if the navigation bar has a Components menu.	User must see the "Components" menu in the dropdown choices when Asset Menu clicked.
3	Click "Components" menu in the navigation bar	User must navigate to the "Component" page.
4	Verify Component Page.	<p>1. Component Page must show tables that has the following columns:</p> <ul style="list-style-type: none"> - Checkbox. For multiple for bulk delete and update - Name (Component Name) - Available (How many component available for check-out) - Model Number - Minimum Quantity - Purchase Date - Purchase Cost - Order Number - Category - Check-out Button - Check-in Button - View Button - Edit Button - Delete Button <p>2. Above columns, table must have headers, buttons, and filters like the following:</p> <ul style="list-style-type: none"> - Components (Quantity Register) - Search Bar

		<ul style="list-style-type: none"> - Filters (by name, category, and manufacturer) - "+ New" button to register new asset. It must be clickable and has hover. - There must be export button. It must be clickable and has hover. 3. The table must have pagination and the list must be responsive to paginationlist number. 4. While typing for search component (all column values are searchable), user must be able to see all the result of the search value. If no record found, it must return a message of "No component found." in the table. 5. The list must update based on the selected filters when the user uses the filter feature.
5	Click "export" button to verify if it will download xlsx format of file.	System must automatically download xlsx format of file with the records of asset.
6	Check if the Comonent Page supports scrolling.	User can scroll to view all content without layout issues or cutoffs

7	Check if the Component Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Component Registration		
1	Verify RBAC	Users with the Admin role can access the registration page for Component and should be the only ones who can see the '+ New' button.
2	Click "+ New" button to verify if Asset registration page exist.	User must navigate to the Component registration page.
3	Verify Component Registration Page.	<p>1. The Component Registration page must have header that contains the following:</p> <ul style="list-style-type: none"> - Link "Component" to navigate on list of Asset Models. Must be seen as Components / New Component - Import button with Icon and must be clickable and has hover. The Import button must allowed xlsx file only, validate the column structure and data types per column, return error messages for any invalid entries, and populate the fields with the file's values if no errors are found. - Has line separator to separate headers and form <p>2. Import button must allowed xlsx file only</p> <p>3. The page must have Asset registration form that contains the following:</p> <ul style="list-style-type: none"> - Component Name. Field and Required - Category. Required and Dropdown (must have button for quick registration of category) - Manufacturer. Dropdown (must have button for quick registration of

		<p>Manufacturer)</p> <ul style="list-style-type: none"> - Supplier. Dropdown (must have button for quick registration of supplier) - Location. Dropdown (must have button for quick registration of location) - Model Number. Field - Order Number. Field - Purchased Date - Purchased Cost. Field and must have php before the field - Quantity. Field and Required - Minimum Quantity. Field. - Notes. Field and must have max input. - Image Upload. Only accepts png and jpeg. Maximum of 5mb - Save Button. Must be enable only if required fields are filled. <p>4. There must be validation if Component already existed, missing value on required fields, and successful Component registration.</p> <p>5. Once the Component is successfully registered, the system must automatically navigate to the list of Components and display a success alert. If the save is unsuccessful, return an error message using danger alert indicating the reason.</p>
4	<p>Check if the Component Registration Page have MAP AMS Footer.</p>	<p>User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.</p>
Component Update		
1	Verify RBAC	<p>Users with the Admin role can access the page for updating Component details and should be the only ones who can see the 'Edit' icon/button in the table.</p>

2	Click "edit" icon in the action column seen in the table to verify if the page for updating asset is exist.	User must navigate to the page of updating Component details.
3	Verify the original value of the fields.	The user must see all fields populated with the values from the originally registered component.
4	Verify header.	The header of edit Asset must be same of register having <ul style="list-style-type: none"> - Components / Update Component - Name of Component. - Clone Button - Delete Button
5	Verify validation for missing value on required fields.	User must not be able to click the "Save" button and error message must display "<field name> is required" in its field.
6	Click "Save" button to verify saved changes.	Once successfully saved, the component details must be automatically updated in the tables and on the individual detail page. The system should then navigate the user to the component list page with a success alert. If the save is unsuccessful, return an error message using danger alert such as 'Component name already exists' or a similar notice.
7	Click 'Clone' button to verify navigation.	User must navigate to the "Component Registration" page with "Component name (cloned)" as Component name.

8	Check if the Component Update Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Component Delete		
1	Verify RBAC	Only user with Admin role can delete existing Asset.
2	Click "delete" icon in the action column seen in the table to verify if modal for deletion confirmation will show.	User must see Delete Confirmation Modal.
2	Click "delete" button to verify deletion status.	<ul style="list-style-type: none"> 1. For a successful deletion: The user must see a success alert stating, "Component deleted successfully!" 2. For failed deletion: The user must see a danger alert explaining the reason why the deletion failed.
4	Verify all items linked to the component before deletion.	<ul style="list-style-type: none"> 1. If Component is being used such link into a check-in and check-out or simply deployed status, deletion must not be successful. Return an error message stating that the Component is used. 2. If nothing is link to Component, system must return successful deletion of Component and automatically update the Component table.
View Page Component		

1	Verify RBAC	Users with the Admin or Operator role can access the page containing records of Component and view full details of each Component.
2	Check if the navigation bar has a Components menu.	User must see the "Components" menu in the dropdown choices when Asset Menu clicked.
3	Click "Components" menu in the navigation bar	User must navigate to the "Component" page.
4	Verify if Action Button for viewing an Component is exist in the page.	User must see 'Eye' Icon/button for viewing an component details.
5	Click 'Eye' Icon/button in the action column seen in the table to verify navigation.	User must navigate to the "Component Detail" Page.

6	Verify Component Detail Page.	<p>1. The Component Detial Page must have a header that contains:</p> <ul style="list-style-type: none"> - Picture. (If applicable) - Link "Component" to navigate on list of Assets. Must be seen as Components / Show Component - Name of Component - Edit Button - Delete Button - Information of Remaining components. How many components are there and how many components are deployed. - A line separator to separate headers and form. <p>2. Component Details section must have tabs containing:</p> <p>a. Details / About (in form view same as snipe-it).</p> <ul style="list-style-type: none"> = Informations: - Name - Model Number - Category - Manufacturer - Supplier - Location - Minimum Quantity - Purchase Date - Purchased Cost - Notes - Created At - Updated At <p>b. Active Checkouts</p> <ul style="list-style-type: none"> = Table view: - Table Header: Active Checkouts - Checkout and Export Button <p>c. Columns on Table</p> <ul style="list-style-type: none"> - Checkout Date - User - Checkout to - Notes - Checkin Button <p>d. History</p> <ul style="list-style-type: none"> = Table View:
---	--------------------------------------	---

		<ul style="list-style-type: none"> - Date - Action - User - Component Name - Asset Name (Check-out to) - Notes <p>d. Additional Files (A table that has additional attachment related to asset, can add additional file there)</p>
7	Check if the Component Detail Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Component Check-Out		
1	Verify RBAC	Both Admin and Operator users can perform component checkout.

2	Check if the Component Page have a "Check-out" button.	User must see "Check-out" button in the Component Page.
3	Click "Check-out" button seen in the table of Component Page.	User must navigate to the "Component Check-out" Page.
4	Verify Component Check-Out Page.	<p>1. The Component Check-Out page must have header that contains the following:</p> <ul style="list-style-type: none"> - Link "Component" to navigate on list of Components. Must be seen as Components/ Checkout Component - Below must be the Component Name - Has line separator to separate headers and form <p>2. The page must have Component Check-out form that contains the following fields:</p> <ul style="list-style-type: none"> - Checkout To. Dropdown of list of assets. Required - Quantity. Field and Required. Must show how many quantity remaining - Checkout Date - Notes - Save Button. Must be enable only if required fields are filled. <p>4. There must be validation if have invalid quantity, missing value on required fields, and successful Component Check-out.</p> <p>5. Once a checkout is successful, the system must automatically navigate to the list of Component Checkout records/history of the specific component and the records there must be updated.</p>
5	Check if the Component Check-Out Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.

Component Check-In			
1	Verify RBAC	Both Admin and Operator users can perform component checkin.	
2	Check if the Component Page have a "Check-in" button.	User must see "Check-in" button in the Component Page.	
3	Click "Check-in" button seen in the table of Component Page.	User must navigate to the "Component Check-in" Page.	
4	Verify Component Check-In Page .	<p>1. The Component Check-In page must have header that contains the following:</p> <ul style="list-style-type: none"> - Link "Component" to navigate on list of Components. Must be seen as Components/ Checkin Component - Below must be the Component Name - Has line separator to separate headers and form <p>2. The page must have Component Check-in form that contains the following fields:</p> <ul style="list-style-type: none"> - Checkin From (Asset Name) - Quantity. Field. Must be equal or less than the quantity checkout - Notes - Save Button. Must be enable only if required fields are filled. <p>4. There must be validation if have invalid quantity input, missing value on required fields, and successful Component Check-in.</p> <p>5. Once a checkin is successful, the system must automatically navigate to the list of Component Checkin records/history of the specific asset and the records there must be updated.</p>	

5	Check if the Component Check-In Page have MAP AMS Footer.	User must see the "© 2025 MAP AMS. All rights reserved." in the footer section.
Component Advance Search/Filter		
1	Verify RBAC	Users with the Admin or Operator role can perform advance search through multiple search filters.

2	Verify if "Filters" button for advance filtering exist in the Component Table on the Component Page.	User must see "filters" button in the Component Table on Component page for advance filtering.
3	Click "Filter" button to verify modal.	<p>1. User must see modal that has the following:</p> <ul style="list-style-type: none"> - Category (Dropdown) - Manufacturer (Dropdown) - Supplier (Dropdown) - Location(Dropdown) - Purchase Date From - Purchase Date To - Due for Check-in From - Due for Check-in To - Created At(From) - Created At (To) - Updated At (From) - Updated At (To) - Reset Filter Button - Apply Filter Button <p>2. If Reset filter is clicked, fields must remove all data entered.</p> <p>3. If apply filter is clicked, modal must close and list of data on a table must follow filtered data.</p>

Dashboard | Due for Return

1	Verify real-time count of assets and components with check-on due	The user must be able to see "Due for Return" card and must show total number of assets that are due for return/check-in
2	Verify Notice	Notice must be a week early before the expected return date.

3	Click "Due for Return" Card to verify its content	User must see a modal that contains a table list of asset and component due for return/checkin.
4	Verify Data Accuracy.	<ul style="list-style-type: none"> - The system retrieves only records where <code>expected_return_date <= current_date</code>. - The count displayed matches the number of qualifying records in the database. - Any updates to the database such as new <code>expected_return_date</code> entries that is <code><= current_date</code> must be reflected in the count.
Dashboard Upcoming Audits		
1	Verify real-time count of upcoming audits	The user must be able to see "Upcoming Audits" card and must show total upcoming audits.
2	Click "Due for Checkin" Card to verify its routing.	User must navigate to the page of due to be audited.
3	Verify Data Accuracy.	<ul style="list-style-type: none"> - The count displayed matches total number of assets that are due for checkin. - Any updates to the database such as new due date entries that is <code><= current_date</code> must be reflected in the count.

Dashboard Upcoming End of Life		
1	Verify real-time count of assets with Upcoming End of Life	The user must be able to see the "Upcoming End of Life" card, which should display the total number of assets approaching their end of life.
2	Click "Upcoming End of Life" Card to verify its routing.	User must navigate to the page of Upcoming End of Life using asset table filter with date of near end of life.
3	Verify Data Accuracy.	<ul style="list-style-type: none"> - The count displayed matches total number of assets approaching their end of life. - Any updates to the database such as new asset approaching end of life must reflected in the count.
Dashboard Expiring Warranties		
1	Verify real-time count of assets with Expiring Warranty.	The user must be able to see the "Expiring Warranties" card, which should display the total number of assets approaching their expiring warranties.
2	Click "Expiring Warranties" Card to verify its routing.	User must navigate to the page of Expiring Warranty using asset table filter with date of near expiring warranty.

3	Verify Data Accuracy.	<ul style="list-style-type: none"> - The count displayed matches total number of assets approaching their expiring warranty. - Any updates to the database such as new asset approaching expiring warranty must reflected in the count.
---	-----------------------	---

Dashboard | Low Stock

1	Verify real-time count of asset model with lowstock or below the minimum quantity.	The user must be able to see the "Low Stock" card, which should display the total number of asset model with lowstock or below the minimum quantity.
2	Click "Low Stock" Card to verify its content.	User must see a modal that contain table list of asset model with lowstock (Asset Model Name, Minimum Quantity, and Available Quantity)
3	Verify Data Accuracy.	<ul style="list-style-type: none"> - The count displayed matches total number of asset model with lowerstock or below the minimum quantity. - Any updates to the database such as new asset model with lowstock or below the minimum quantity must reflected in the count.

Dashboard | Overdue Audits

1	Verify real-time count of asset Due for Audit	The user must be able to see the "Overdue Audits" card, which should
---	---	--

		display the total number of asset that are overdue for audits.
2	Click "Overdue Audits" Card to verify its routing.	User must navigate to the page of Overdue for Audits.
3	Verify Data Accuracy.	<ul style="list-style-type: none"> - The count displayed matches total number of asset that are overdue for audits. - Any updates to the database such as new asset overdue for audit must reflected in the count.

Dashboard | Overdue for Return

1	Verify real-time count of asset and component overdue for return.	The user must be able to see the "Overdue for Return" card, which should display the total number of asset and component that are overdue for return.
2	Click "Overdue for Return" Card to verify its content.	User must see a modal that contain table list of asset and component overdue for return/checkin (Asset/Component Type, Asset/Component Name, Checkout to, and Expected Checkin Date)
3	Verify Data Accuracy.	<ul style="list-style-type: none"> - The count displayed matches total number of asset and component that are

		<p>overdue for return.</p> <ul style="list-style-type: none"> - Any updates to the database such as new asset and component overdue for return must reflected in the count.
Dashboard Expired Warranties		
1	Verify real-time count of asset with expired warranty.	The user must be able to see the "Expired Warranty" card, which should display the total number of asset and component that are overdue for return.
2	Click "Expired Warranties" Card to verify its content.	User must navigate to the page of Expiring Warranty using asset table filter with data of beyond expiring warranty.
3	Verify Data Accuracy.	<ul style="list-style-type: none"> - The count displayed matches total number of assets that beyond expiring warranty. - Any updates to the database such as new asset beyond its expiring warranty must reflected in the count.
Dashboard Total Asset Costs		
1	Verify if "Total Asset Costs" Card exist.	User must be able to see a card for "Total Asset Costs".

2	Verify "Total Asset Costs" Card content.	<ul style="list-style-type: none"> - User must be able to see a card title of "Total Asset Costs" - User must be able to see total value (2 decimal places) of all assets with php currency before the value.
---	--	---

Dashboard | Asset Utilization

1	Verify if "Asset Utilization" Card exist.	User must be able to see a card for "Asset Utilization".
2	Verify "Asset Utilization" Card content.	<ul style="list-style-type: none"> - User must be able to see a card title of "Asset Utilization". - User must be able to see percentage of utilization of all assets.

Dashboard | Asset Category

1	Verify if "Asset Categories" Card exist.	User must be able to see a card for "Asset Categories".
2	Verify "Asset Categories" Card content.	<ul style="list-style-type: none"> - User must be able to see a card title of "Asset Categories". - User must be able to see a pie chart

		<p>corresponding on all category asset have.</p> <ul style="list-style-type: none"> - User must be able to see and click the "Browse All" button.
3	Click "Browse All" button to verify its routing.	User must automatically navigate to the "Category" page.
Dashboard Asset Statuses		
1	Verify if "Asset Statuses" Card exist.	User must be able to see a card for "Asset Statuses".
2	Verify "Asset Statuses" Card content.	<ul style="list-style-type: none"> - User must be able to see a card with the title of "Asset Statuses". - User must be able to see a pie chart corresponding on all statuses asset have. - User must be able to see and click the "Browse All" button.
3	Click "Browse All" button to verify its routing.	User must automatically navigate to the "Statuses" page.
Forecasting Asset Status Forecast		

1	Verify if "Asset Status Forecast" Card exist.	User must be able to see a card for "Asset Status Forecast".
2	Verify "Asset Status Forecast" Card content.	<ul style="list-style-type: none"> - User must be able to see a card with the title of "Asset Status Forecast". - User must be able to see a chart with predicted counts of each status (Active, Under Repair, Lost, Stolen, Write-Off)
Forecasting Product Demand Forecast		
1	Verify if "Product Demand Forecast" Card exist.	User must be able to see a card for "Product Demand Forecast".
2	Verify "Product Demand Forecast" Card content.	<ul style="list-style-type: none"> - User must be able to see a card with the title of "Product Demand Forecast". - User must be able to see a graph with predicted check-out counts per product model (in demand next month).
QR QR Code Generation Upon Asset Registration		
1	Click "Asset" menu in the navigation bar	User must navigate to the "Asset" page.

2	Click "+ New" button to verify if Asset registration page exist.	User must navigate to the Asset registration page.
3	Fill all required asset details	All mandatory fields are completed.
4	Click the "Save" button.	System validates input and registers the asset.
5	System automatically generates a unique QR code linked to that specific asset's details page.	A unique QR code image is generated and stored in the database, associated with the asset.
6	View the asset record from the asset list.	The asset record displays a visible QR code image.
7	Scan the QR code using a QR scanner or mobile device.	The scanned QR code redirect user to the correct asset details page the system.
QR Viewing Asset Information view QR Scan		

1	Open the system or mobile device camera with QR scanning capability.	QR scanner is ready for use.
2	Scan the QR code attached to or displayed for the asset.	QR scanner reads the QR code successfully.
3	Verify the QR code redirects to the asset's details page like /assets/view/<asset_id>	Browser or system opens the asset's information page instantly.
4	Verify the displayed asset information.	The correct asset details are shown accurately.
5	Attempt scanning a QR code of another asset.	The system displays a different asset's details page correctly.
6	Attempt scanning a modified/invalid QR code.	System displays an error message or "Asset not found."

1.1 Sign-off Criteria

Sign-off Criteria		
Item	Acceptance Criteria	Responsibility
1.	All stage 1 Acceptance Criteria signed off as complete	

2. MILESTONES

Date	Milestone

3. RISK AND CONTINGENCIES

Identifier	Risk	Contingency

4. SIGN OFF

Required signatures for sign off UAT Test Plan (this document).

Name	Title/Responsibility	Signature
Sir Elvin Punzalan	MAP Active PH IT Head	
Ma. Elisa Johanna T. Garrote	Project Manager	

Appendix D: Defense and Validation

D.1 Panel Evaluation Sheets (Signed)

PROPOSAL TITLE:					
CLIENT'S NAME:					
MEMBERS:	1	Name	Grade		
	2			SECTION:	
	3			FACULTY-IN-CHAR GE:	
	4			ADVISER: (if any)	
	5			DATE OF PRESENTATION:	
	6				

CRITERIA	SUCCESS INDICATORS	SCORE	COMMENTS
1. PROBLEM DEFINITION (20 points)	<ul style="list-style-type: none"> Creates clear and unambiguous problem statements in the project. Articulates how this project will be approved and of benefit Explains the contribution of the project/study Identify project scope, objectives, benefits, weaknesses and potential risks. 		
	TOTAL SCORE:		
	RATING (Total Score / 20) * 100:		
2. PROBLEM ANALYSIS (5 points)	<ul style="list-style-type: none"> Identify and describe the cause-and-effect relationships in the problem domain. 		
	TOTAL SCORE:		
	RATING (Total Score / 5) * 100:		
3. REVIEW OF RELATED LITERATURE (25 points)	<ul style="list-style-type: none"> Information is gathered from multiple, research-based sources. Well organized, demonstrates logical sequencing and structure. Detailed conclusions are reached from the evidence offered. Information is cited properly and in APA format. 		

	<ul style="list-style-type: none"> Students are able to grasp the concepts presented and be able to relate the concepts with the project proposal. 		
	TOTAL SCORE:		
	RATING (Total Score / 25) * 100:		

CRITERIA	SUCCESS INDICATORS	SCORE	COMMENTS
4. REQUIREMENTS ANALYSIS (15 points)	<ul style="list-style-type: none"> Explains the overall design objectives. Defines functional and non-functional requirements. Analyzes the constraints and limitations of the project. 		
	TOTAL SCORE:		
	RATING (Total Score / 15) * 100:		
5. SOLUTION ANALYSIS (15 points)	<ul style="list-style-type: none"> Solutions offered by the project addresses the problem stated. The design carries out the system requirements entirely. The solution offered in the project is correct 		
	TOTAL SCORE:		
	RATING (Total Score / 15) * 100:		
6. DESIGN SPECIFICATIONS (25 points)	<ul style="list-style-type: none"> Design models follow correct notation. All specified requirements are reflected in the design models. The design of the interface is intuitive and user-friendly The expectations, behaviors and reactions before, during and after use of the interface are mostly positive. All system entities and respective attributes and relationships are correctly modelled. 		
	TOTAL SCORE:		
	RATING (Total Score / 25) * 100:		
7. TECHNICAL REPORT (30 points)	<ul style="list-style-type: none"> Objectives are very clearly stated and the report has a strong central focus. Clear structure. The report is very clear and very cohesive. All Tables/Figures link to text. All sources are identified and referenced appropriately in the body. 		

	<ul style="list-style-type: none"> Excellent appearance of Report, Figures/Tables/ References 		
	TOTAL SCORE:		
	RATING (Total Score / 30) * 100:		

CRITERIA	SUCCESS INDICATORS	SCORE	COMMENTS
8. PRESENTATION (30 points)	<ul style="list-style-type: none"> Students are dressed professionally. Information presented is logically arranged and visual aids are readable and organized Maintains eye contact with the audience, seldom returning to notes. Uses clear voice and correct, precise pronunciation of terms so that all audience can hear the presentation Members can easily grasp question content and answers to questions are clear, precise, consistent, and complete The group shows unity and coordination in answering questions 		
	TOTAL SCORE:		
	RATING (Total Score / 30) * 100:		

SUMMARY				OVERALL COMMENTS/SUGGESTIONS	
Criteria			Rating	Weight	
1. Project Grade 70%					
	Sub-Criteria	Rating	Weight		
	1.1. Problem Definition 10%				
	1.2. Problem Analysis 10%				
	1.3. Requirements Analysis 20%				
	1.4. Solution Analysis 20%				
	1.5. Design Specifications 40%				
	TOTAL				
2. Technical Report 10%					

3. Presentation 10%			
4. Review of Related Literature 10%			Evaluated by:
GRADE			

D.2 Photos from Defense and Pilot Testing

Defense Photos

Figure __. <insert title>

<insert image>

Pilot Testing Photos

Figure __. <insert title>

<insert image>

D.3 Certificates of Copyright / Patent Application (If any)

Certificates of Copyright

Appendix E: Access Links (All valid for 3 years)

E.1 Live System URL

The system is deployed on Railway using a multi-service architecture where the frontend and each Django microservice run as separate services. Each backend microservice has its own public domain and its own PostgreSQL database. These URLs provide direct access to the live, production-ready system.

Links:

- Frontend Web App: <https://<frontend>.railway.app>
- Auth Microservice API: <https://<auth-service>.railway.app>
- Asset Microservice API: <https://<asset-service>.railway.app>
- Context Microservice API: <https://<context-service>.railway.app>

E.2 Source Code Repository (e.g., GitHub)

All project components are stored in a single GitHub repository, including the React frontend and all three Django microservices. Each microservice is structured as an independent project with its own environment, dependencies, and database, while being centralised in one repository for easier review and version control.

Link:

- GitHub Repository: <https://github.com/ElisaGarrote/Capstone1>

E.3 Comprehensive Online Documentation (e.g., ReadTheDocs, GitBook)

The system's full technical and functional documentation is provided through an online documentation platform or through a local `/docs` directory. It includes the architecture overview,

database designs, API references, installation instructions, deployment steps, and the user manual.

Links:

- Documentation Portal: <https://<your-docs>.gitbook.io/>
- GitHub Docs Folder: /docs in the repository

E.4 System Demo Video (YouTube/Vimeo)

A complete demo video is provided to showcase system features such as authentication, asset and component management, check-in/check-out workflows, dashboard analytics, and real-use scenarios. It serves as a visual guide for evaluators and future developers.

Link:

- Demo Video: <https://youtu.be/EvEdyV-bNx4?si=a3K88JDZoR5hFxwP>

E.5 CI/CD Pipeline Dashboard (if publicly accessible)

If CI/CD is configured, build and deployment logs may be publicly accessible through GitHub Actions or Railway. These dashboards provide insight into automated builds, tests, and deployments. If the pipeline is private, access details are omitted.

Links:

- GitHub Actions: <https://github.com/ElisaGarrote/Capstone1/actions>
- Railway Project Dashboard: <https://railway.app/project/<project-id>>
- If private: CI/CD pipelines are configured internally and not publicly accessible.