BSCS FINAL PROJECT

Requirements Specification

AssetIn: Asset Management Tool

Project Advisor

Sir.Asif Farooq

Presented by:

Group ID: F24BSCS042

Student Reg#	Student Name
L1F21BSCS1059	MUHAMMAD BURHAN
L1F21BSCS0485	AREEBA KHAN
L1F21BSCS0484	NOOR-UL-AAIN MAQBOOL

Faculty of Information Technology

University of Central Punjab

Software Requirements Specification

Version 01

AssetIn:

Asset Management Tool

Advisor: Sir. Asif Farooq

Group F23BSCS042

Member Name	Primary Responsibility
MUHAMMAD BURHAN	Back-end & Database
AREEBA KHAN	UI/UX & Front-end integration
NOOR-UL-AAIN-MAQBOOL	UI/UX & Front-end designing

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

Abstract

Asset management is crucial for organizations to maintain operational efficiency, yet many existing tools suffer from fragmented systems, poor usability, and limited scalability. These inefficiencies lead to increased costs, underutilization of assets, and difficulties in decision-making processes. Our project, **AssetIn**, aims to address these challenges by providing an intuitive, cloud-based asset management tool that integrates real-time data, vendor management, and predictive analytics.

Leveraging modern technologies such as **Angular**, **.NET**, and **MySQL**, the tool offers a comprehensive solution for tracking both fixed and variable assets, managing vendor relationships, and streamlining maintenance schedules. The tool also incorporates **cloud storage** for secure and scalable media handling, allowing organizations to store asset-related images, documents, and videos efficiently.

By addressing the shortcomings of existing tools such as Ralph3, SnipeIT, and Asset Tiger, **AssetIn** is expected to significantly improve asset management workflows. This solution will enhance user experience, optimize asset utilization, and reduce operational costs for organizations. Collaboration with **Nextonix** ensures industry-standard practices, and the final product will be tested in real-world scenarios to deliver a robust and practical asset management tool.

1. Introduction and Background

Effective asset management is a critical function for organizations aiming to optimize resources, reduce operational costs, and maintain long-term efficiency. Assets, including software licenses, development tools, intellectual property, and employee expertise, play a pivotal role in the functioning of any organization, particularly in the software industry. Despite the availability of several asset management tools, many organizations continue to face inefficiencies in tracking, maintaining, and utilizing their assets. These inefficiencies stem from overly complex systems, fragmented workflows, lack of real-time data integration, and limited customization options. The consequences include higher operational costs, underutilization of assets, and difficulties in making informed decisions.

Existing asset management solutions, such as **Ralph3**, **SnipeIT**, and **Asset Tiger**, provide specific functionalities but fail to deliver a comprehensive solution that balances usability, scalability, and efficiency. **Ralph3**, for example, is known for its robust features but is highly complex to install and configure, often requiring advanced technical knowledge. **SnipeIT**, while simplifying IT service management, lacks broader applicability for other types of assets and does not include vendor management features. Similarly, **Asset Tiger** offers ease of use but is not scalable for larger organizations. The fragmented nature of these systems hinders their effectiveness when applied across diverse organizational contexts.

AssetIn, our proposed solution, addresses these gaps by offering an intuitive, scalable, and cloud-based asset management tool that integrates multiple asset management functions into a single platform. Built using **Angular**, **.NET**, and **MySQL**, and leveraging **cloud storage**, AssetIn aims to provide a seamless experience for managing both fixed and variable assets. The tool will also incorporate predictive analytics, vendor management, and media handling features, ensuring organizations can optimize their asset utilization, streamline workflows, and make data-driven decisions.

The significance of this project lies in its potential to reduce the operational complexities faced by organizations in managing assets. By providing a user-centered, easy-to-use platform, AssetIn is set to enhance productivity, lower costs, and simplify asset management processes. The collaboration with **Nextonix** further ensures that the system adheres to industry standards, making it a practical and impactful solution for real-world applications.

1.1 Product (Problem Statement)

Organizations often face challenges in managing assets due to the complexity and limitations of existing tools. Current solutions lack essential features like real-time tracking, predictive analytics, and vendor management, resulting in fragmented workflows, underutilized resources, and higher operational costs. Additionally, the usability and scalability of these tools fail to meet the diverse needs of organizations, leading to low adoption rates.

AssetIn aims to solve these issues by providing a comprehensive, cloud-based solution that integrates all aspects of asset management into an intuitive and scalable platform. This system will reduce operational inefficiencies, enhance asset utilization, and support data-driven decision-making processes, ultimately simplifying asset management for businesses.

1.2 Background

Efficient asset management is essential for organizations to optimize resources, reduce operational costs, and ensure sustainable growth. Assets, ranging from physical resources to intellectual property and software licenses, form the backbone of operational efficiency. However, many organizations face challenges in managing these assets due to fragmented workflows, lack of real-time data integration, and limited customization options in existing tools.

Popular solutions like Ralph3, SnipeIT, and Asset Tiger address specific asset management needs but fall short in offering a holistic, scalable, and user-friendly approach. For instance, Ralph3 is feature-rich but overly complex to configure, SnipeIT simplifies IT asset management but lacks versatility for broader applications, and Asset Tiger, though easy to use, does not scale effectively for large organizations.

These shortcomings result in inefficiencies, including underutilization of assets, higher costs, and difficulties in making data-driven decisions. Recognizing this gap, the proposed solution, AssetIn, aims to deliver a comprehensive and intuitive cloud-based asset management tool. By integrating real-time tracking, vendor management, predictive analytics, and media handling capabilities, AssetIn seeks to streamline workflows, enhance decision-making, and optimize asset utilization across various industries.

1.3 Scope

AssetIn aims to develop a cloud-based asset management solution designed to address the limitations of current systems, providing organizations with a comprehensive platform for managing their assets. Key features include real-time tracking and monitoring of assets, with advanced search and filtering capabilities for quick access to critical data. The system also streamlines vendor management and procurement processes, facilitating vendor registration, maintenance, and sales management, all while ensuring smooth workflows throughout.

The platform will leverage predictive analytics to forecast maintenance needs, optimize asset usage, and generate actionable reports to support decision-making. Cloud storage integration will provide secure, scalable storage for managing documents, images, and videos related to assets. The user interface will be intuitive, offering dashboards for real-time monitoring and easy navigation, ensuring accessibility for different user roles, including organization owners, asset managers, and vendors.

AssetIn will be built on a robust MySQL database with secure, normalized structures to support scalability as the system grows. It will incorporate strong authentication mechanisms like JWT to safeguard user data and ensure role-based access controls. Extensive testing and quality assurance processes will validate the system's functionality and performance, ensuring the platform adapts to the needs of small, medium, and large organizations without compromising efficiency or reliability. By combining these features, AssetIn will streamline asset management workflows, reduce operational costs, and offer organizations a scalable, efficient solution for managing their assets.

1.4 Objective(s)/Aim(s)/Target(s)

The AssetIn project aims to develop a scalable and user-friendly asset management tool to improve operational efficiency and address the limitations of existing solutions. Key objectives include:

1. **User-Friendly Interface**: Design an intuitive platform for organization owners and asset managers to efficiently track, maintain, and manage both fixed and variable assets.
2. **Real-Time Asset Tracking and Vendor Management**: Implement real-time tracking of asset location, condition, and usage, while integrating vendor management for streamlined procurement, repairs, and sales.
3. **Predictive Analytics and Cloud Storage**: Utilize data-driven insights for asset optimization and lifecycle management, and provide secure cloud-based storage for asset-related media.
4. **Scalability and Security**: Build a scalable system with strong security features like JWT authentication to support growing asset and user volumes while ensuring data protection.

1.5 Challenges

The development of **AssetIn** involves several complex tasks that require deep understanding of technologies, problem-solving skills, and adaptability. Key challenges include:

1. **Scalability of Database Design**: Ensuring the **MySQL database** remains scalable and optimized as requirements evolve, while maintaining performance and flexibility.

2. **Implementation of Predictive Analytics**: Developing predictive models for asset optimization and integrating them with user-friendly reporting mechanisms.

3. **Ensuring Security and Data Protection**: Implementing robust security measures, including **JWT** for authentication, to protect sensitive data in a cloud environment.

4. **Performance Optimization**: Optimizing the system's performance as it scales, ensuring efficient handling of increasing assets, users, and media files.

5. **User Experience (UX) Design**: Creating an intuitive interface that supports multiple user roles without compromising usability in complex asset management workflows.

These challenges will require significant technical expertise and iterative problem-solving to ensure that the **AssetIn** system is robust, scalable, and user-friendly.

1.6 Learning Outcomes

Upon successful completion of the **AssetIn** project, students will gain valuable skills and knowledge across several key areas of computer science and software development. The specific learning outcomes include:

1. **Proficiency in Full-Stack Development:**

    - Students will develop skills in **frontend (Angular)** and **backend (.NET Core)**, building a scalable, real-time web application with integrated **MySQL** databases and **cloud storage**.

    - They will implement **APIs**, apply predictive analytics, and ensure **data security** using **JWT**.

    - Experience in **UI/UX design** for user-friendly interfaces and using **JIRA** for **Agile project management**.

2. **Deployment:**

    - Students will learn the deployment process, optimizing performance and ensuring smooth operation in real-world environments.

1.7 Nature of End Product

The AssetIn project will deliver a comprehensive asset management tool with the following key features:

- **Real-Time Asset Tracking**: Track asset location, condition, and status in real time for better decision-making and operational efficiency.

- **Predictive Analytics**: Use data-driven insights to forecast maintenance needs and optimize asset performance, reducing downtime and underutilization.
- **Integrated Vendor Management**: Streamline procurement, repairs, and asset sales within a unified platform.
- **Scalability and Security**: Ensure scalable performance to manage growing assets and users, with secure authentication using JWT for data protection.

1.8 Completeness Criteria

Here is a proposed completeness criteria table for **AssetIn**, based on the key components of your project. Each subpart is defined, with assigned weightage to reflect the importance of each feature in evaluating the project's success:

	Criteria	Weightage %
1	**User Interface Design (Web GUI):** Develop an intuitive, user-friendly interface for organization owners, asset managers, and vendors to easily manage and track assets. This will include real-time dashboards and essential navigation for core functionalities.	15
2	**Real-Time Asset Tracking and Monitoring:** Implementation of real-time tracking to monitor asset locations, status (assigned, in stock, damaged), and maintenance schedules, with advanced search and filter options.	15
3	**Vendor and Procurement Management:** Integration of vendor-related functionalities, allowing users to manage procurement, repairs, sales, and vendor registrations in a seamless manner.	10
4	**Cloud Storage Integration:** Securely manage asset-related media (documents, images, videos) through cloud storage solutions, ensuring scalability and accessibility of data.	10
5	**Predictive Analytics and Reporting:** Develop and implement predictive analytics for maintenance scheduling and asset performance optimization, as well as generating comprehensive reports for decision-making.	15
6	**Database Design and Iterative Development:** Ongoing design and adjustments of the database schema (using MySQL) to accommodate new requirements during the development process. The database design will be flexible and scalable, ensuring normalization, security, and efficiency throughout all phases of development.	20
7	**Security and Authentication:** Ensure robust security features such as **JWT-based authentication and authorization** to protect user data and asset information, especially in cloud storage environments.	5
8	**Testing:** Comprehensive functional and non-functional testing to ensure the system works effectively in real-world environments.	10

Total: 100%

This breakdown provides a clear structure for the evaluation of your project, with appropriate weightage for each critical component.

1.9 Business Goals

1. **Optimized Asset Utilization:** Track asset conditions and usage in real time to ensure efficient utilization and minimize wastage.
2. **Cost Efficiency:** Reduce operational costs through predictive maintenance, efficient asset tracking, and streamlined vendor management, leading to lower procurement and asset replacement costs.
3. **Data-Driven Decision-Making:** Leverage predictive analytics to make informed decisions on asset lifecycle, procurement, and maintenance.
4. **Streamlined Workflows:** Integrate asset tracking, vendor management, and maintenance scheduling into a unified platform to enhance workflow efficiency.
5. **Scalability and Security:** Ensure the system can scale with organizational growth while maintaining robust security with JWT authentication for data protection.
6. **Vendor and Compliance Management:** Improve vendor relationships and ensure compliance by centralizing vendor data and maintaining secure asset transaction records.
7. **Operational Efficiency and Sustainability:** Automate processes to reduce manual errors, improve productivity, and promote sustainability through optimized asset usage and reduced carbon footprint.

1.10 Related Work/ Literature Survey/ Literature Review

Asset management tools have been widely studied and developed, with a focus on improving usability, scalability, and performance. Several solutions already exist in the market, each addressing specific organizational needs. However, most existing tools suffer from complexity, limited scalability, or fragmented functionalities. This literature review explores popular asset management tools like:

1. **Ralph3**
2. **Snipe-IT**
3. **Asset Tiger**

While existing asset management tools like Ralph3, Snipe-IT, and Asset Tiger address certain aspects of asset tracking, they often fall short in providing a comprehensive, scalable, and user-friendly solution. **AssetIn** aims to fill this gap by offering a holistic asset management system that integrates **real-time tracking**, **vendor management**, **predictive analytics**, and **cloud storage** into one powerful and easy-to-use platform. Through this, AssetIn will streamline asset management workflows and improve decision-making processes for organizations across various industries.

1.11 Document Conventions

This Software Requirements Specification (SRS) follows these conventions for clarity and consistency:

1. **Font Style**: Times New Roman, 12pt for body text and 14pt bold for headings. Section titles are bold, and subsections are bold-italic.
2. **Font Color**: Standard black text with key terms highlighted in bold for emphasis.
3. **Lists**: Bullet points and numbered lists are used to simplify complex information.
4. **Tables**: Tables are formatted for clarity with bold headers and minimal borders.
5. **Acronyms/Abbreviations**: Defined upon first use (e.g., AssetIn as Asset Management Tool).

6. **Technical Terms**: Standard casing for software-related terms (e.g., Angular, .NET Core, MySQL).
7. **Layout**: Numbered sections and subsections, with an automatically updated table of contents.
8. **References**: External sources are cited in the References section using appropriate academic citation formats.

2. Overall Description

2.1 Product Features

AssetIn is a cloud-based asset management tool designed to simplify asset tracking, management, and maintenance. Key features include:

User and Organization Owner Panels: The User Panel allows asset purchase, repair, and transfer requests, while the Organization Owner Panel manages asset categories, user approvals, vendor interactions, and asset-related transactions.

Asset Manager Panel: Enables asset managers to update statuses, track locations, and handle tasks efficiently through an intuitive dashboard.

Vendor Dashboard: Provides organization owners with tools for managing vendor registrations and transactions related to asset sales, maintenance, and purchases.

Asset Tracking and Maintenance Scheduling: Real-time asset tracking monitors asset status, location, and condition, while maintenance scheduling ensures timely servicing to prevent downtime.

Cloud-Based Storage Integration: Secure cloud storage for storing and retrieving asset-related media (documents, images, videos), ensuring accessibility and protection.

Advanced Search and Filters: Enhanced search options allow users to quickly find and manage assets, improving efficiency in asset tracking and reporting.

Email Alerts: Configurable email notifications keep users informed about maintenance schedules, asset transfers, or status changes.

Barcode Scanning: Barcode scanning streamlines asset management by enabling quick, accurate tracking of assets.

Cloning Entity Feature: Allows users to duplicate asset records or configurations, saving time during bulk asset addition or category setup.

API Authentication and Security: Secure API access via JWT authentication protects sensitive data and ensures authorized interactions with the system.

Scalability and Adaptability: Designed to scale with growing organizations, the system supports future extensions like a Super Admin dashboard without compromising performance or security.

AssetIn combines these features to deliver a comprehensive, scalable, and secure asset management solution, enhancing workflows, reducing costs, and supporting data-driven decisions across various organizations.

2.2 User Classes and Characteristics

Organization Owner (Primary User Class)

- **Frequency of Use**: High (daily or multiple times per week)
- **Key Functions Used:**
    - Overseeing asset management tasks and categories.
    - Managing user roles, permissions, and asset-related requests. Handling vendor registration, sales, and maintenance.
    - Monitoring asset performance and generating reports.
    - Overseeing maintenance scheduling.
- **Technical Expertise**: Medium to High
- **Security/Privilege Levels:** Highest (full system access and admin rights)
- **Educational Level:** Varies (business owners or senior management with basic to intermediate technical knowledge)
- **Characteristics:**
    - Organization Owners have the broadest level of access and control.
    - They are responsible for strategic decision-making and approving financial or maintenance-related actions.
    - They require a comprehensive view of the organization's asset data, reports, and vendor activities, but may not need to be deeply involved in day-to-day operational tasks.

Asset Manager (Primary User Class)

- **Frequency of Use:** High (daily or multiple times per week)
- **Key Functions Used:**
    - Managing fixed and variable assets.
    - Performing asset status updates.
    - Handling tasks related to asset location tracking.
    - Scheduling and tracking maintenance.
    - Generating reports on asset conditions and usage.
- **Technical Expertise:** Medium (intermediate technical expertise required for asset tracking and maintenance processes)
- **Security/Privilege Levels:** Medium (can access asset-related data but cannot manage users or vendors)
- **Educational Level:** Likely holds a position in operations or asset management with practical knowledge in asset tracking.
- **Characteristics:**
    - Asset Managers are responsible for the day-to-day management of assets.
    - They use the system to update asset statuses, track locations, and schedule maintenance.
    - They may need a clear and easy-to-navigate dashboard to manage asset information efficiently.

Vendor (Primary User Class)

- **Frequency of Use:** Medium (dependent on the frequency of asset-related transactions)
- **Key Functions Used:**
    - Registering for asset-related transactions (sales, maintenance, and procurement).
    - Interacting with the organization's asset management team for asset servicing.
    - Viewing and responding to asset requests.
- **Technical Expertise:** Low to Medium (depends on the vendor's technical background, but primarily requires basic access to interact with the system)
- **Security/Privilege Levels:** Low (restricted to asset transactions and communication with the organization owner)
- **Educational Level:** Varies (likely basic technical knowledge or none needed, depending on the nature of the vendor interaction)
- **Characteristics:**
    - Vendors interact with the system mostly to manage their transactions and provide services for the organization's assets.

- They do not require full access to asset management functions, only those related to the sale, maintenance, or purchase of assets.

End Users (Secondary User Class)

- **Frequency of Use:** Medium to Low (depending on the frequency of asset requests)
- **Key Functions Used:**
  - Requesting asset purchases, repairs, or transfers through the User Request Panel.
  - Monitoring the status of their requested assets or pending actions.
- **Technical Expertise:** Low (basic user interface interactions for requesting assets)
- **Security/Privilege Levels:** Low (only limited access to asset requests and personal tracking data)
- **Educational Level:** Varies (general staff or employees with minimal technical skills)
- **Characteristics:**
  - These users interact with the system to request assets or check the status of their requests.
  - They are not involved in managing or tracking assets directly but rely on the system to track their asset-related requests.

System Admin (Technical User Class)

- **Frequency of Use:** Low to Medium (when system configurations, backups, or updates are required)
- **Key Functions Used:**
  - Performing system maintenance, such as data backups and updates.
  - Handling system configurations and ensuring smooth operation of the application.
  - Monitoring system performance and addressing any technical issues.
- **Technical Expertise:** High (requires a deep understanding of the software infrastructure)
- **Security/Privilege Levels:** High (full access to system configurations and server settings)
- **Educational Level:** Likely technical, with expertise in software or IT systems.
- **Characteristics:**
  - System Admins are responsible for the technical upkeep of the system and ensuring that it operates smoothly.
  - Their involvement is primarily in the background and less frequent than the main user classes.

2.3 Operating Environment

Hardware Requirements

For successful development and deployment of AssetIn, the following hardware is needed. Three laptops with:

- Multi-core processor (Intel i5 or equivalent) for development and testing.
- Minimum 8 GB RAM (16 GB recommended for better multitasking and performance).
- At least 256 GB SSD for storing software, frameworks, and local databases.
- Compatible OS: Windows 10+, macOS 10.12+, or Linux (e.g., Ubuntu).

Barcode Scanner

A scanner for testing barcode functionality, supporting USB or wireless connectivity. Must be compatible with supported barcode formats and offer fast, accurate scanning for smooth asset tracking integration.

These resources ensure a stable and efficient development environment for AssetIn.

Software and Application Components

- Nginx or Apache for handling HTTP requests and serving frontend/backend components.
- ASP.NET Core for scalable, secure request handling and database interactions.
- Angular 18 for a dynamic, responsive user interface.

- MySQL 8.0+ for storing asset, user, and vendor data, hosted locally or on cloud platforms (e.g., AWS RDS, Azure SQL).
- Cloudinary or similar for secure, scalable asset media storage.
- JWT for secure user authentication and data protection.
- Git & GitHub for collaborative development and version management.
- SMTP for sending alerts and notifications.
- RESTful APIs for system and external integrations.

API Testing Tools:

1. **Postman** will be used for comprehensive API testing, allowing developers and testers to create and execute requests to verify API functionality. This tool will ensure the integrity and security of the APIs used in **AssetIn** by testing different API endpoints, request types, and payloads. It supports automated tests and offers robust testing features, including data validation and response assertion.
2. **Sawger** will also be used for API testing, especially focusing on load testing and performance benchmarking. It will help test the system's ability to handle a high volume of simultaneous requests and ensure that **AssetIn** can perform efficiently under heavy usage scenarios.

Interoperability with Other Systems

AssetIn will need to coexist with vendor systems for asset purchase and maintenance, allowing vendors to interact with the platform for sales, repairs, and procurement processes.
APIs will be designed for seamless communication between **AssetIn** and third-party vendor management platforms.

Barcode Scanning Devices:
AssetIn will support external barcode scanning hardware (USB or wireless scanners) to help with efficient asset tracking. The software will interface with these devices to quickly register and update asset statuses.

Cloud Integration:

AssetIn will be hosted in a cloud environment (e.g., Amazon Web Services (AWS), Microsoft Azure, or Google Cloud), ensuring high availability, scalability, and data redundancy. The cloud platform will also be used for handling increased loads and maintaining the system's performance over time.

Software Dependencies:

- Node.js (for frontend build processes and backend communication)
- Microsoft .NET Core SDK (for the backend environment)
- MySQL Client (for interacting with the database)
- Angular CLI (for frontend development)

2.4 Design and Implementation Constraints

Hardware Limitations
The developers are restricted to using laptops with decent but limited hardware specifications (e.g., multi-core processors, 8–16 GB of RAM). This limits the ability to simulate large-scale, high-performance scenarios during local testing.

Interfaces to Other Applications
The system must integrate seamlessly with third-party vendor platforms for procurement, maintenance, and asset sales. This will require the development of APIs to communicate effectively with these systems.

Authentication Protocols
JWT is the mandated authentication method, requiring strict adherence to its implementation for user security and API authorization.

Design Conventions and Standards

- **Coding Standards:**
  Developers must follow best practices in C# and TypeScript for the backend and frontend, respectively, adhering to consistent coding conventions and patterns.
- **User Experience Guidelines:**
  The UI/UX must be designed to align with accessibility standards (e.g., WCAG 2.1), ensuring ease of use for all users, regardless of technical expertise.

Budget and Resources:

Development is limited by a fixed budget, which restricts the use of premium tools, services, or additional hardware resources.

2.5 Assumptions and Dependencies

2.5.1 Assumptions

1. **Cloud Integration and Infrastructure**:
   - The selected cloud storage provider **(e.g., Cloudinary)** is assumed to meet the system's needs for storing and retrieving media files (images, videos, documents) within usage limits and budget constraints. The provider's API is expected to remain stable and well-documented throughout the project lifecycle.
   - Development and production environments will have adequate resources, including laptops with compatible barcode scanners for testing, and servers with sufficient processing power, memory, and storage for managing asset data and user requests.
2. **Development and Usage Conditions**:
   - All required tools and technologies **(e.g., Angular, ASP.NET Core, MySQL, Postman, Swagger)** will be available in stable, compatible versions, with no major version changes or deprecations during the development lifecycle.
   - Users, including organization owners and asset managers, are assumed to have reliable internet access for seamless interaction with the platform. Initial usage will primarily target small to medium-sized organizations, with scalability needs increasing progressively over time.

2.5.2 Dependencies

- **Third-Party Services and Tools**:
  - The system relies on Cloudinary (or a similar service) for media storage, and any disruptions, service outages, or API changes could impact functionality. Similarly, tools like Postman and Swagger are critical for API testing and validation, and any limitations or unavailability could delay the development process.
  - Reusable software components (e.g., JWT for authentication, Angular CLI) are assumed to function reliably without major bugs or compatibility issues.
- **Resource and Network Stability**:
  - The stability of the development environment (IDEs, version control systems like Git, and CI/CD pipelines) and hosting infrastructure (e.g., AWS, Azure, or Google Cloud) is crucial. Any disruptions could affect timelines or performance.
  - The project assumes consistent team availability with no significant turnover or skill gaps, as well as stable and secure network connections for both development and deployment, ensuring the reliability of real-time features like asset tracking.

Risks if Assumptions Are Incorrect

- If the selected cloud storage provider experiences outages, performance issues, or fails to meet the system's requirements, media storage and retrieval functionality may be disrupted.

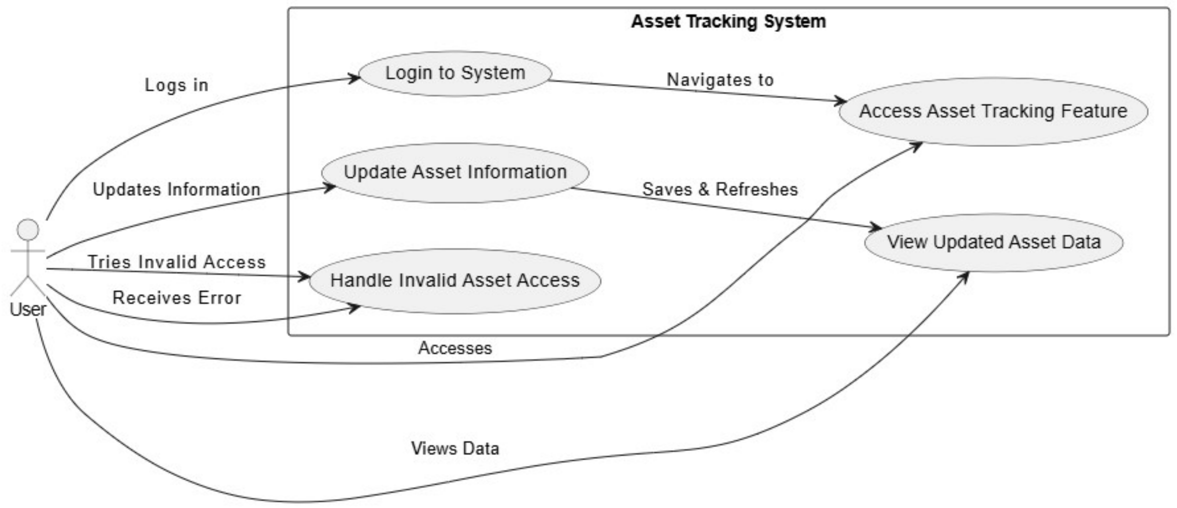
- If the system encounters larger-than-anticipated data volumes, it could lead to degraded performance, delayed processing, or challenges in scaling infrastructure effectively.

- Changes in data privacy or security regulations may require significant rework to ensure compliance, potentially delaying project timelines or increasing costs.

- If essential tools (e.g., Postman, Swagger) are deprecated, discontinued, or become unavailable, it could disrupt API testing and validation processes, causing delays in development.

3. Functional Requirements

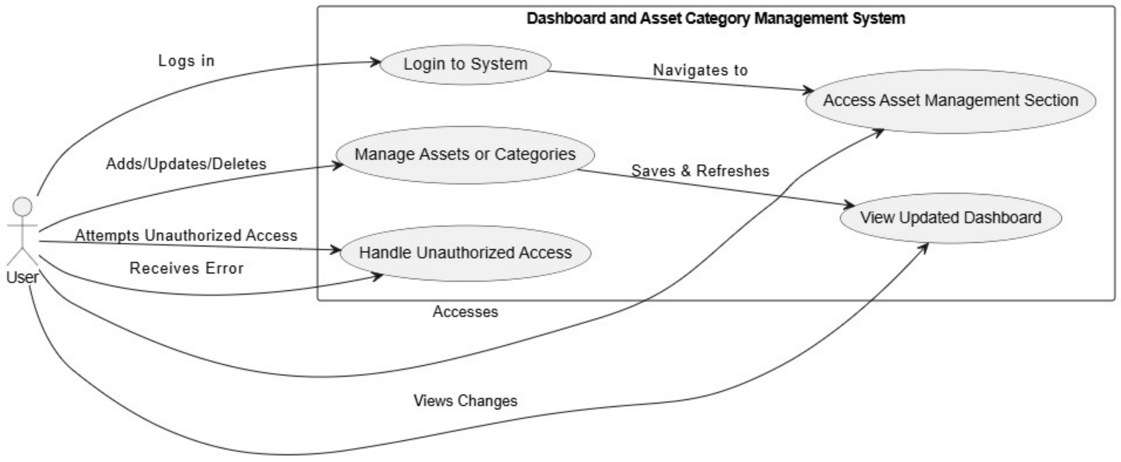
3.1 Track Asset Locations and Status



Identifier	UC-1	
Purpose	Allow users to track the location and status of assets to ensure accurate record-keeping and monitoring.	
Priority	High	
Pre-conditions	• User must be logged in with appropriate permissions. • Assets must already exist in the database.	
Post-conditions	• Asset location and status are updated and accessible to authorized users.	
Typical Course of Action		
S#	**Actor Action**	**System Response**
1	User logs into the system.	The system authenticates the user and displays the dashboard.
2	User accesses the asset tracking feature.	System displays asset details, including location and status.
3	User updates asset information (if required).	System validates input and updates the asset record.
4	User views updated data.	System refreshes and displays updated information in real-time.
Alternate Course of Action		
S#	**Actor Action**	**System Response**
1	User attempts to access an invalid or non-existent asset.	System displays a "No Results Found" message.

Table 1: UC-1

3.2 Access Dashboards and Manage Asset Categories


Dashboard and Asset Category Management System
Logs in
Login to System
Navigates to
Access Asset Management Section
Adds/Updates/Deletes
Manage Assets or Categories
Saves & Refreshes
View Updated Dashboard
Attempts Unauthorized Access
Handle Unauthorized Access
Receives Error
User
Accesses
Views Changes

Identifier	UC-2
Purpose	Allow users to access dashboards and manage asset categories effectively.
Priority	High
Pre-conditions	• User must be authenticated and authorized to manage assets or categories.
Post-conditions	• Changes to assets or categories are saved and reflected on the dashboard.

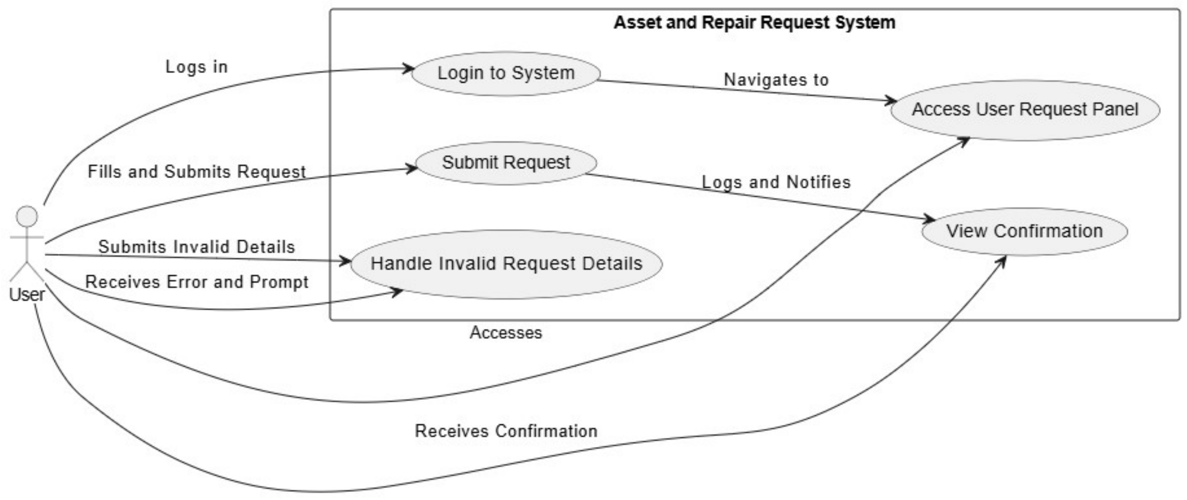
Typical Course of Action

S#	Actor Action	System Response
1	User logs into the system.	The system authenticates the user and displays the dashboard.
2	User accesses the asset management section.	System displays options to manage assets or categories.
3	User adds, updates, or deletes assets or categories.	System validates inputs and updates the database.
4	User views updated dashboard.	System refreshes and displays the changes made.

Alternate Course of Action

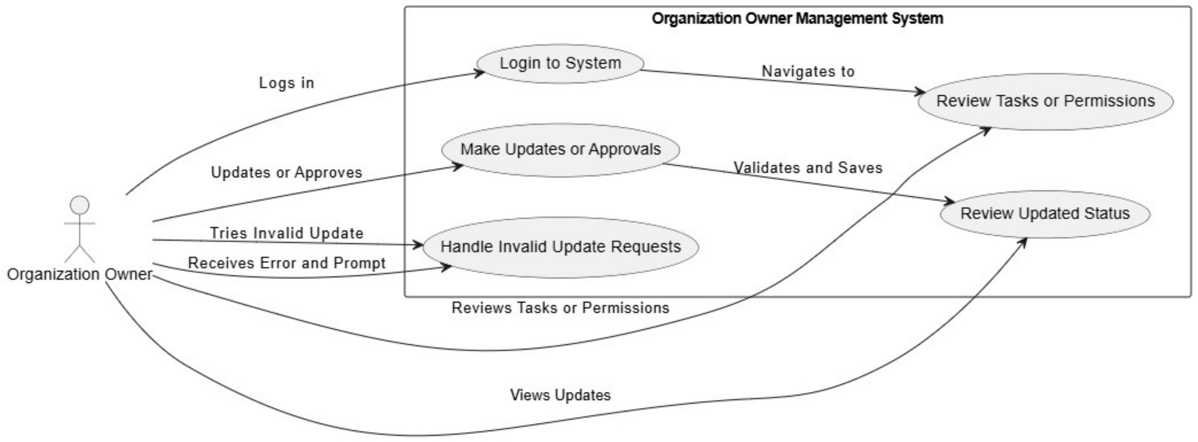
S#	Actor Action	System Response
1	User attempts to access without proper authorization.	System denies access and displays an error message.

3.3 Request Asset and Repairs



Identifier	UC-3		
Purpose	Allow users to submit requests for asset and repairs.		
Priority	High		
Pre-conditions	• User must be logged in and authorized to submit requests.		
Post-conditions	• Request is logged and routed to the appropriate manager for review.		
Typical Course of Action			
S#	**Actor Action**	**System Response**	
1	User logs into the system.	The system authenticates the user and displays the dashboard.	
2	User accesses the User Request Panel.	System displays options to request asset, and repairs.	
3	User fills in request details and submits.	System validates input and logs the request.	
4	User views confirmation.	System sends the request to the relevant manager and notifies the user.	
Alternate Course of Action			
S#	**Actor Action**	**System Response**	
1	User submits incomplete or invalid details.	System displays an error and prompts the user to correct the input.	

3.4 Oversee Asset Management Tasks


Organization Owner Management System
Login to System
Navigates to
Review Tasks or Permissions
Make Updates or Approvals
Validates and Saves
Review Updated Status
Handle Invalid Update Requests
Logs in
Updates or Approves
Tries Invalid Update
Receives Error and Prompt
Organization Owner
Reviews Tasks or Permissions
Views Updates

Identifier	UC-4
Purpose	Allow organization owners to oversee all asset management tasks, including user management and approvals.
Priority	High
Pre-conditions	• User must be logged in as an organization owner.
Post-conditions	• Updates to user permissions or request statuses are saved and reflected in the system.

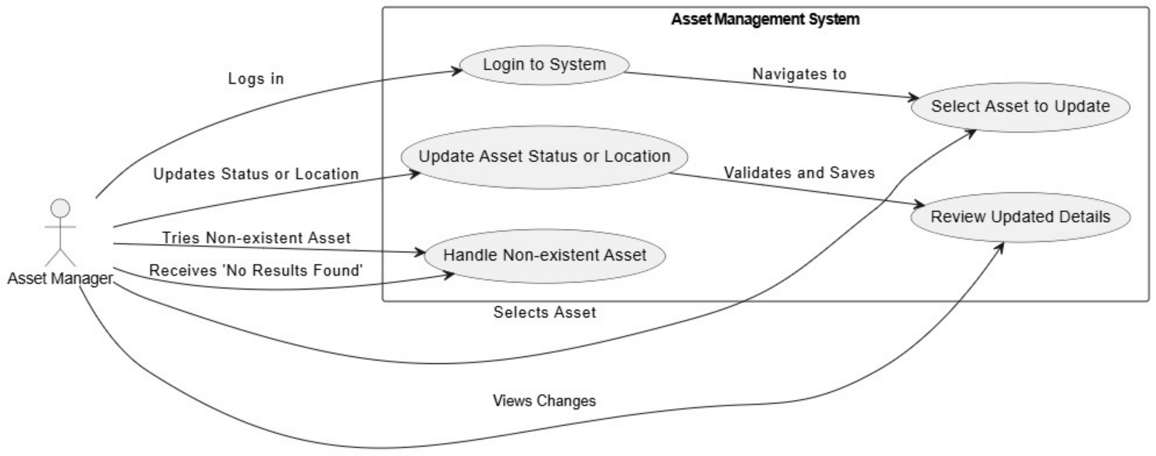
Typical Course of Action

S#	Actor Action	System Response
1	Organization owner logs in.	System authenticates the user and displays the Organization Owner Panel.
2	Owner reviews tasks or permissions.	System displays pending requests, user permissions, or asset categories.
3	Owner makes updates or approvals.	System validates changes and updates the database.
4	Owner reviews updated status.	System reflects the changes and sends relevant notifications.

Alternate Course of Action

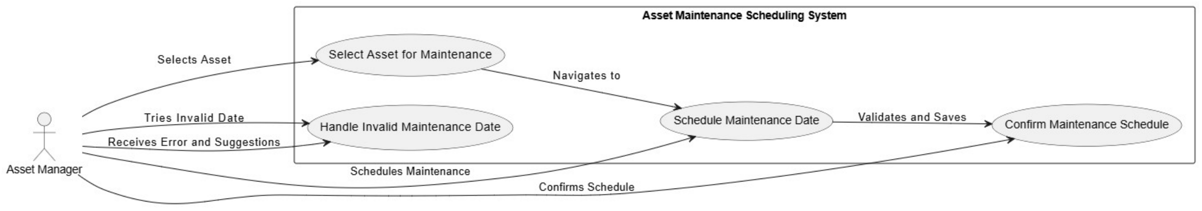
S#	Actor Action	System Response
1	Owner tries to update an invalid request.	System displays an error and prevents the action.

3.5 Manage Fixed and Variable Assets



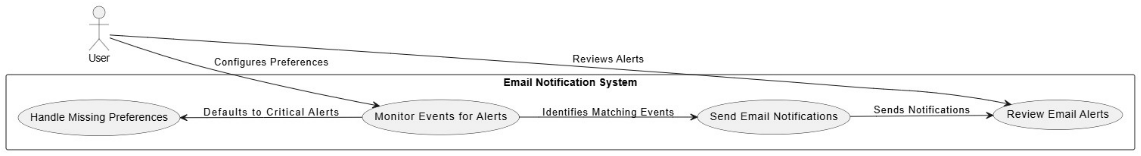
Identifier	UC-5	
Purpose	Allow asset managers to update statuses and manage fixed or variable assets.	
Priority	High	
Pre-conditions	• User must be logged in as an asset manager.	
Post-conditions	• Asset updates are reflected in real-time in the system.	
Typical Course of Action		
S#	**Actor Action**	**System Response**
1	Asset manager logs in.	System authenticates the user and displays the Asset Manager Panel.
2	Manager selects an asset to update.	System displays asset details and current status.
3	Manager updates the status or location.	System validates the input and updates the asset record.
4	Manager reviews changes.	System refreshes and displays updated details in real-time.
Alternate Course of Action		
S#	**Actor Action**	**System Response**
1	Manager tries to update a non-existent asset.	System displays a "No Results Found" message.

3.6 Schedule Maintenance for Assets


Asset Maintenance Scheduling System
Selects Asset
Select Asset for Maintenance
Navigates to
Tries Invalid Date
Receives Error and Suggestions
Handle Invalid Maintenance Date
Schedule Maintenance Date
Validates and Saves
Confirm Maintenance Schedule
Asset Manager
Schedules Maintenance
Confirms Schedule

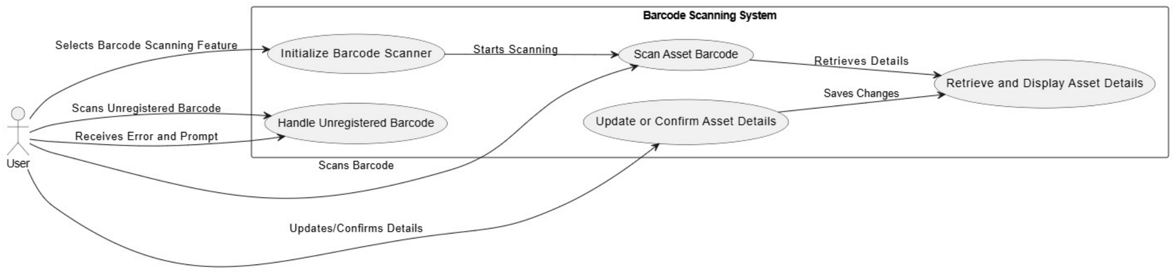
Identifier	UC-6	
Purpose	Allow asset managers to schedule asset maintenance and prevent downtime.	
Priority	Medium	
Pre-conditions	• User must be logged in as an asset manager.	
Post-conditions	• Maintenance schedules are saved, and notifications are sent to stakeholders.	
Typical Course of Action		
S#	**Actor Action**	**System Response**
1	Asset manager selects assets for maintenance.	System displays asset details and maintenance history.
2	Manager schedules a maintenance date.	System validates the input and saves the schedule.
3	Manager confirms the schedule.	System sends notifications for upcoming maintenance.
Alternate Course of Action		
S#	**Actor Action**	**System Response**
1	Manager tries to schedule maintenance for an invalid date.	System displays an error and suggests available slots.

3.7 Receive Configurable Email Alerts


User
Configures Preferences
Reviews Alerts
Email Notification System
Handle Missing Preferences
Defaults to Critical Alerts
Monitor Events for Alerts
Identifies Matching Events
Send Email Notifications
Sends Notifications
Review Email Alerts

Identifier	UC-7		
Purpose	Notify users of important events, such as maintenance schedules, asset transfers, or status changes, through email alerts.		
Priority	Medium		
Pre-conditions	• User must configure email alert preferences. • Relevant events must be logged in the system.		
Post-conditions	• The user receives email alerts for selected events.		
Typical Course of Action			
S#	**Actor Action**	**System Response**	
1	System monitors for events triggering alerts.	System identifies events that match user preferences.	
2	System sends email notifications.	User receives email alerts based on preferences.	
3	User reviews alerts.	System ensures timely delivery of accurate notifications.	
Alternate Course of Action			
S#	**Actor Action**	**System Response**	
1	User fails to configure preferences.	System defaults to sending alerts for critical events only.	

3.8 Utilize Barcode Scanning


Barcode Scanning System
Selects Barcode Scanning Feature
Initialize Barcode Scanner
Starts Scanning
Scan Asset Barcode
Retrieves Details
Retrieve and Display Asset Details
Saves Changes
Scans Unregistered Barcode
Handle Unregistered Barcode
Receives Error and Prompt
Update or Confirm Asset Details
User
Scans Barcode
Updates/Confirms Details

Identifier	UC-8
Purpose	Allow users to scan barcodes associated with assets for streamlined tracking and management.
Priority	Medium
Pre-conditions	• User must have access to a barcode scanning device. • Barcode data must be associated with assets in the system.
Post-conditions	• Scanned asset details are updated or retrieved in the system.

Typical Course of Action

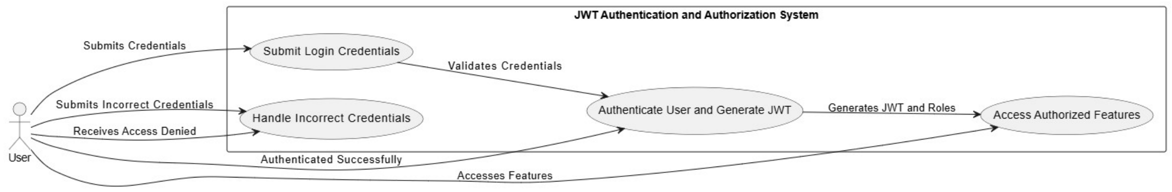
S#	Actor Action	System Response
1	User selects the barcode scanning feature.	System initializes the barcode scanner interface.
2	User scans an asset barcode.	System retrieves and displays asset details based on the scanned data.
3	User updates or confirms asset details.	System saves changes and updates the asset record.

Alternate Course of Action

S#	Actor Action	System Response
1	User scans an unregistered barcode.	System displays an error message and prompts the user to register the asset.

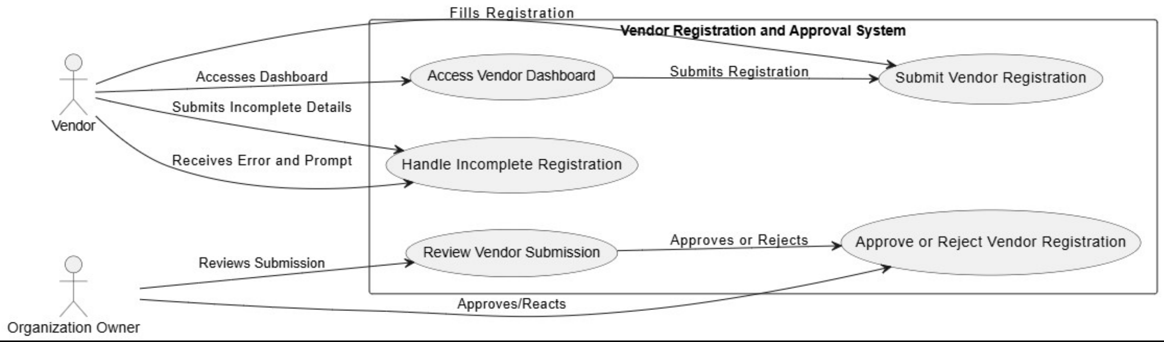
3.9 Authenticate and Authorize via JWT

e


JWT Authentication and Authorization System
Submits Credentials
Submit Login Credentials
Validates Credentials
Submits Incorrect Credentials
Handle Incorrect Credentials
Receives Access Denied
Authenticate User and Generate JWT
Generates JWT and Roles
Access Authorized Features
Authenticated Successfully
Accesses Features
User

Identifier	UC-9	
Purpose	Ensure secure user authentication and authorization using JSON Web Tokens (JWT).	
Priority	High	
Pre-conditions	• User must provide valid credentials.	
Post-conditions	• User is authenticated, and access is granted based on roles and permissions.	
Typical Course of Action		
S#	**Actor Action**	**System Response**
1	User submits login credentials.	System validates the credentials.
2	User is authenticated successfully.	System generates a JWT and assigns appropriate access roles.
3	User accesses authorized system features.	System enforces access based on the JWT permissions.
Alternate Course of Action		
S#	**Actor Action**	**System Response**
1	User submits incorrect credentials.	System denies access and prompts the user to retry.

3.10 Vendor Registration and Management


Vendor Registration and Approval System
Fills Registration
Vendor
Accesses Dashboard
Submits Incomplete Details
Receives Error and Prompt
Access Vendor Dashboard
Submits Registration
Submit Vendor Registration
Handle Incomplete Registration
Organization Owner
Reviews Submission
Review Vendor Submission
Approves or Rejects
Approve or Reject Vendor Registration
Approves/Reacts

Identifier	UC-10
Purpose	Allow vendors to register and participate in asset-related transactions, while enabling organization owners to manage vendor approvals.
Priority	Medium
Pre-conditions	• Vendor must be authenticated to access the system. • Organization owner must be logged in to manage vendor registrations.
Post-conditions	• Vendor registration is approved or rejected, and the vendor's status is updated in the system.

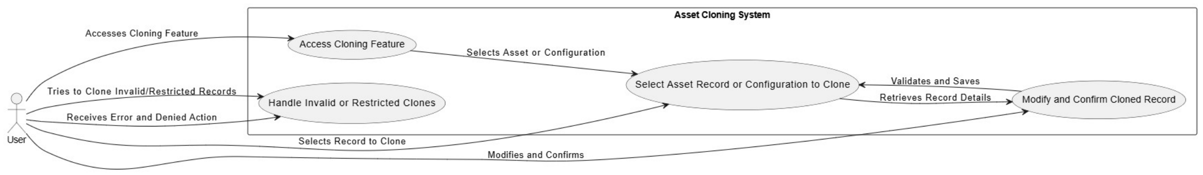
Typical Course of Action

S#	Actor Action	System Response
1	Vendor accesses the Vendor Dashboard.	System displays the registration form.
2	Vendor submits registration details.	System validates and saves the submission as "Pending Approval."
3	Organization owner reviews the submission.	System displays the vendor’s details for approval or rejection.
4	Organization owner approves or rejects the registration.	System updates the vendor's status and sends a notification.

Alternate Course of Action

S#	Actor Action	System Response
1	Vendor submits incomplete details.	System displays an error and prompts the vendor to complete the form.

3.11 Clone Asset Records or Configurations


Asset Cloning System
Accesses Cloning Feature
Access Cloning Feature
Selects Asset or Configuration
Select Asset Record or Configuration to Clone
Validates and Saves
Retrieves Record Details
Modify and Confirm Cloned Record
Tries to Clone Invalid/Restricted Records
Handle Invalid or Restricted Clones
Receives Error and Denied Action
User
Selects Record to Clone
Modifies and Confirms

Identifier	UC-11
Purpose	Allow users to duplicate asset records or configurations for efficient bulk asset setup.
Priority	Medium
Pre-conditions	• User must be logged in with appropriate permissions. • Existing asset records or configurations must be available.
Post-conditions	• New asset records or configurations are created and saved in the system.

Typical Course of Action

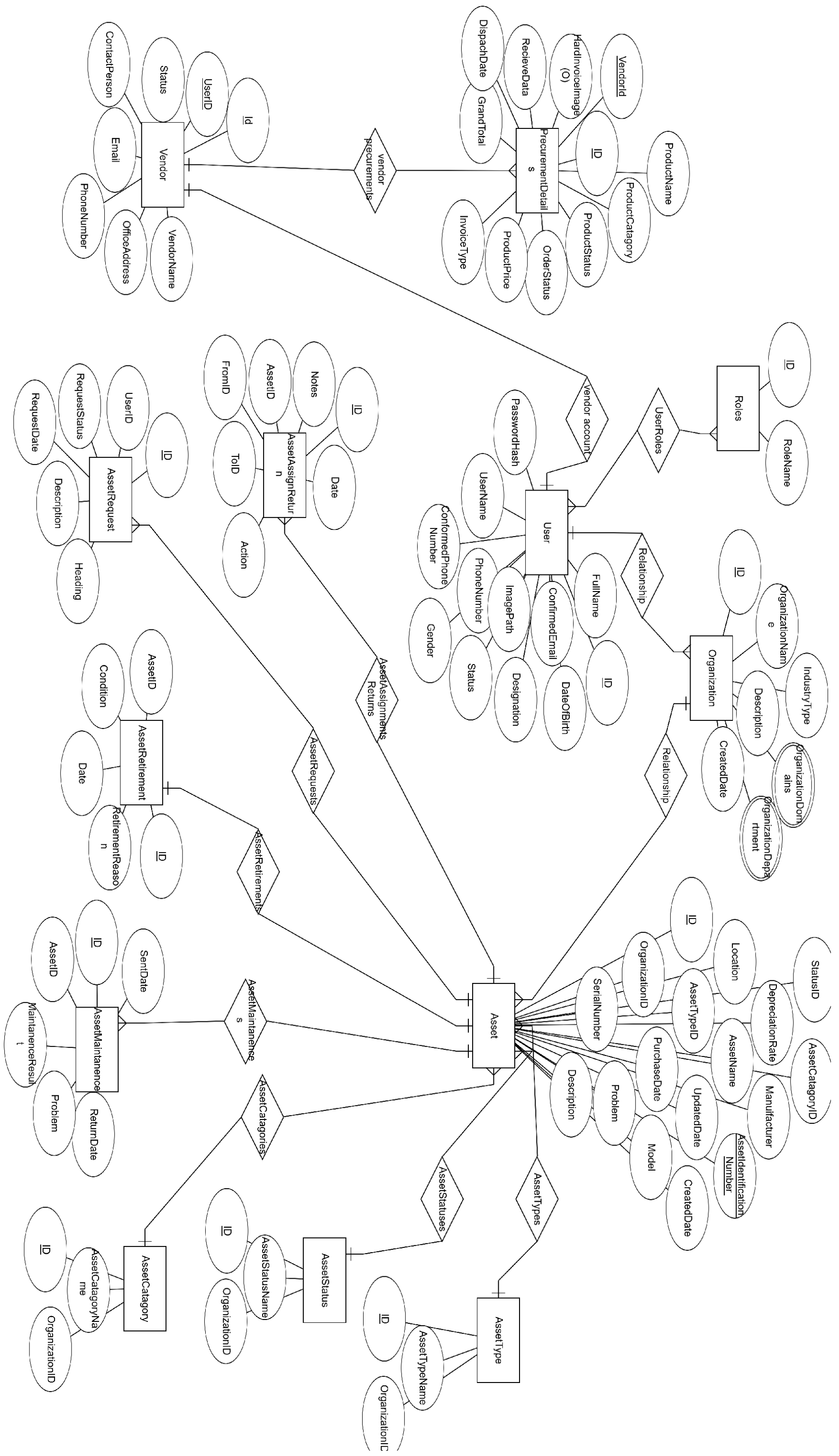
S#	Actor Action	System Response
1	User accesses the cloning feature.	System displays options to select asset records or configurations.
2	User selects an asset record or configuration to clone.	System retrieves the selected record details.
3	User modifies cloned details if needed and confirms.	System validates and saves the new record.

Alternate Course of Action

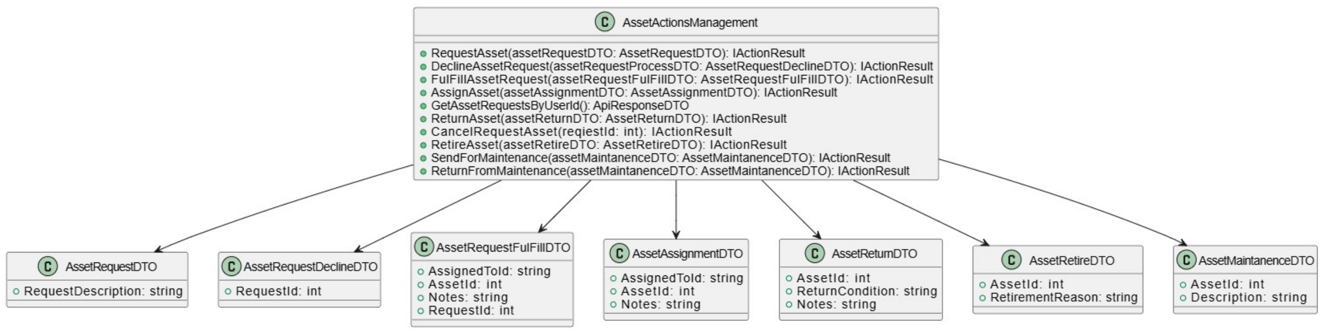
S#	Actor Action	System Response
1	User tries to clone invalid or restricted records.	System denies the action and displays an error message.

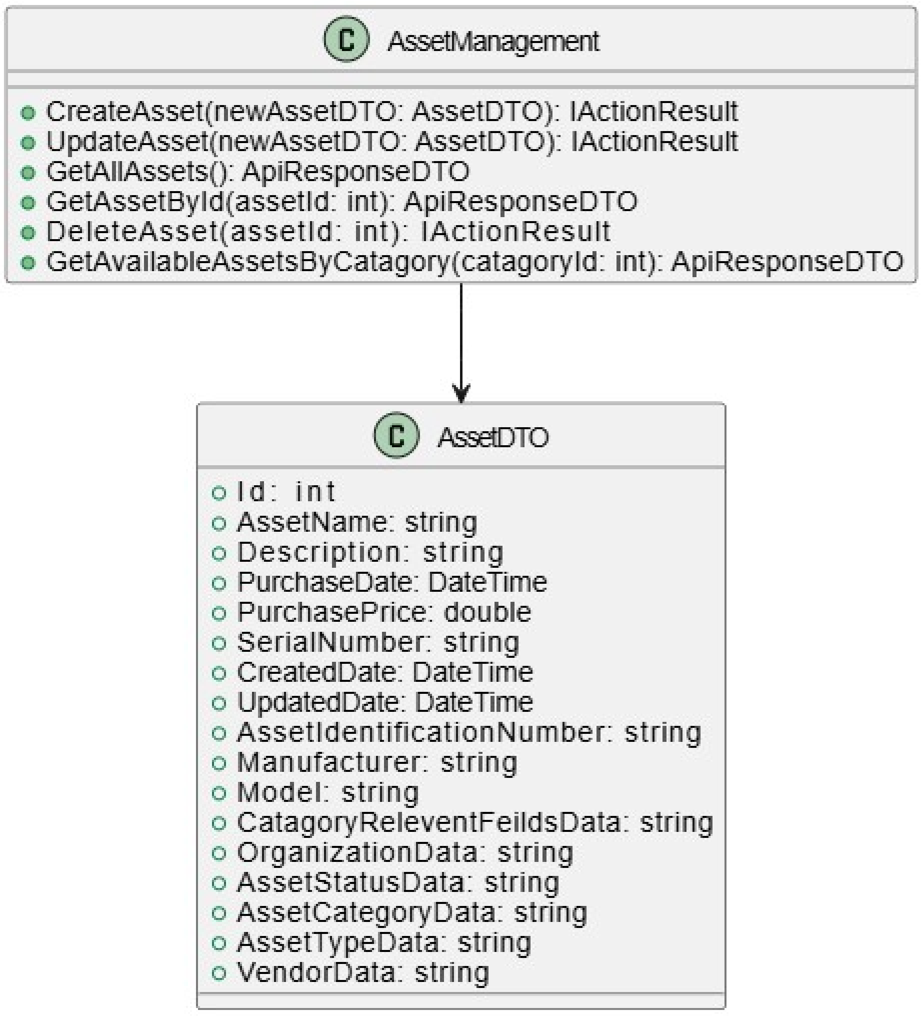
3.12 Requirements Analysis and Modeling

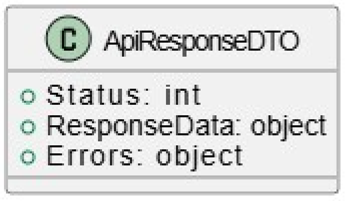
Entity-Relationship Diagram (ERD):

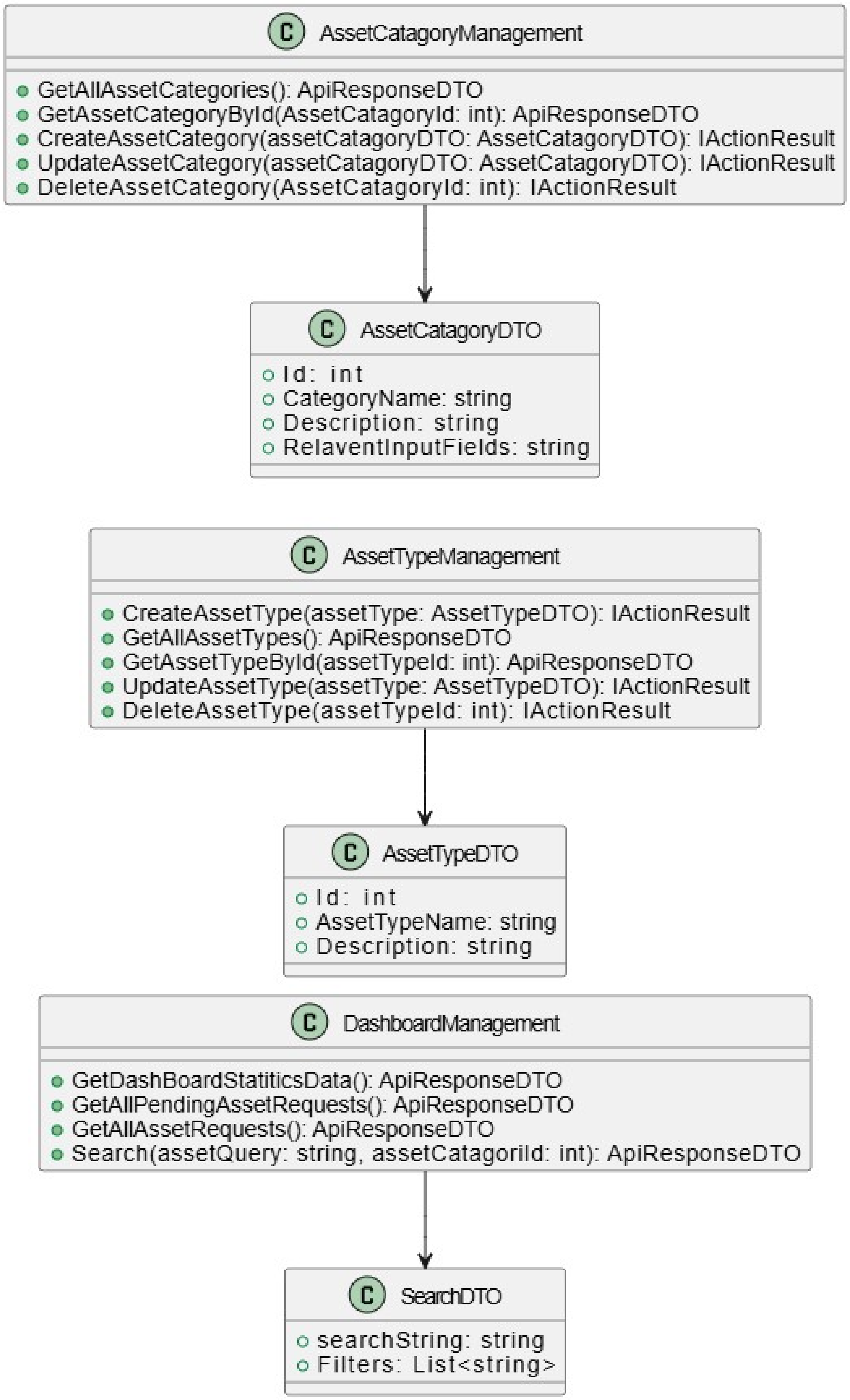


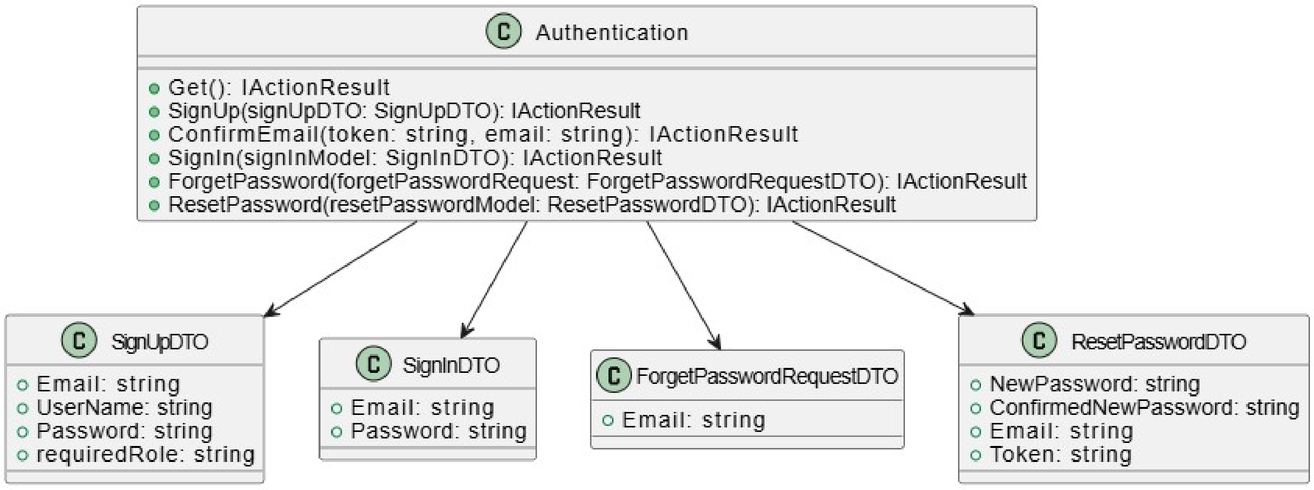
Class Diagrams:


AssetActionsManagement
RequestAsset(assetRequestDTO: AssetRequestDTO): IActionResult
DeclineAssetRequest(assetRequestProcessDTO: AssetRequestDeclineDTO): IActionResult
FulFillAssetRequest(assetRequestFulFillDTO: AssetRequestFulFillDTO): IActionResult
AssignAsset(assetAssignmentDTO: AssetAssignmentDTO): IActionResult
GetAssetRequestsByUserId(): ApiResponseDTO
ReturnAsset(assetReturnDTO: AssetReturnDTO): IActionResult
CancelRequestAsset(reqiestId: int): IActionResult
RetireAsset(assetRetireDTO: AssetRetireDTO): IActionResult
SendForMaintenance(assetMaintanenceDTO: AssetMaintanenceDTO): IActionResult
ReturnFromMaintenance(assetMaintanenceDTO: AssetMaintanenceDTO): IActionResult
AssetRequestDTO
RequestDescription: string
AssetRequestDeclineDTO
RequestId: int
AssetRequestFulFillDTO
AssignedToId: string
AssetId: int
Notes: string
RequestId: int
AssetAssignmentDTO
AssignedToId: string
AssetId: int
Notes: string
AssetReturnDTO
AssetId: int
ReturnCondition: string
Notes: string
AssetRetireDTO
AssetId: int
RetirementReason: string
AssetMaintanenceDTO
AssetId: int
Description: string

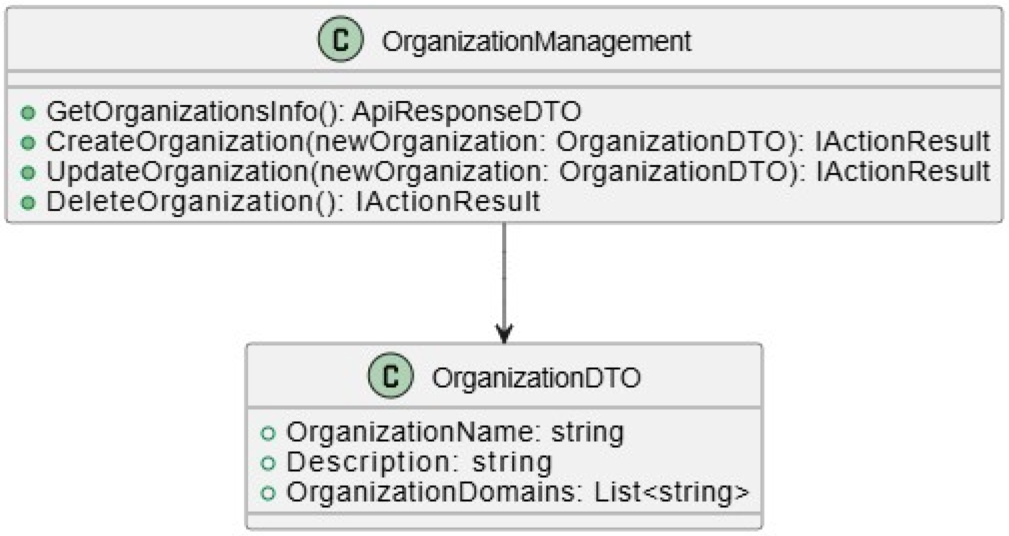

AssetManagement
CreateAsset(newAssetDTO: AssetDTO): IActionResult
UpdateAsset(newAssetDTO: AssetDTO): IActionResult
GetAllAssets(): ApiResponseDTO
GetAssetById(assetId: int): ApiResponseDTO
DeleteAsset(assetId: int): IActionResult
GetAvailableAssetsByCatagory(catagoryId: int): ApiResponseDTO
AssetDTO
Id: int
AssetName: string
Description: string
PurchaseDate: DateTime
PurchasePrice: double
SerialNumber: string
CreatedDate: DateTime
UpdatedDate: DateTime
AssetIdentificationNumber: string
Manufacturer: string
Model: string
CatagoryReleventFeildsData: string
OrganizationData: string
AssetStatusData: string
AssetCategoryData: string
AssetTypeData: string
VendorData: string



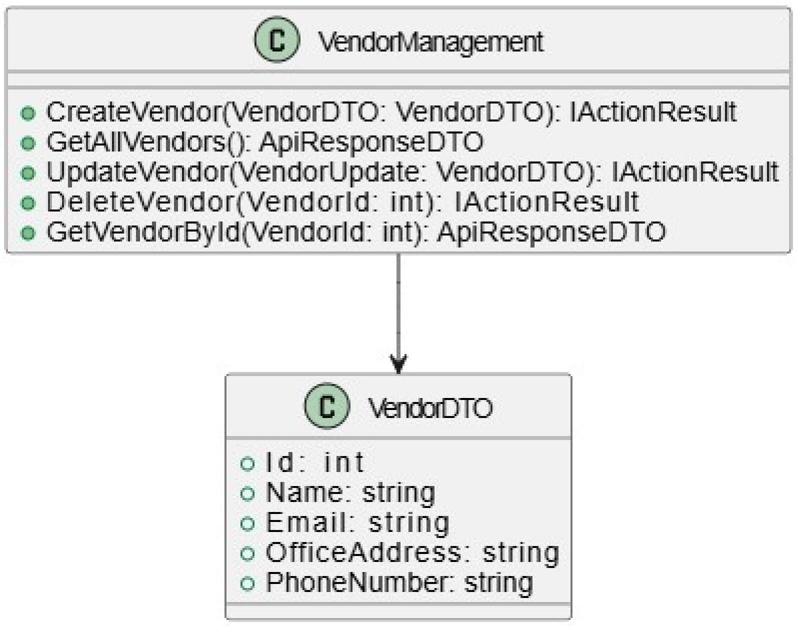

AssetCatagoryManagement
GetAllAssetCategories(): ApiResponseDTO
GetAssetCategoryById(AssetCatagoryId: int): ApiResponseDTO
CreateAssetCategory(assetCatagoryDTO: AssetCatagoryDTO): IActionResult
UpdateAssetCategory(assetCatagoryDTO: AssetCatagoryDTO): IActionResult
DeleteAssetCategory(AssetCatagoryId: int): IActionResult
AssetCatagoryDTO
Id: int
CategoryName: string
Description: string
RelaventInputFields: string
AssetTypeManagement
CreateAssetType(assetType: AssetTypeDTO): IActionResult
GetAllAssetTypes(): ApiResponseDTO
GetAssetTypeById(assetTypeId: int): ApiResponseDTO
UpdateAssetType(assetType: AssetTypeDTO): IActionResult
DeleteAssetType(assetTypeId: int): IActionResult
AssetTypeDTO
Id: int
AssetTypeName: string
Description: string
DashboardManagement
GetDashBoardStatiticsData(): ApiResponseDTO
GetAllPendingAssetRequests(): ApiResponseDTO
GetAllAssetRequests(): ApiResponseDTO
Search(assetQuery: string, assetCatagoriId: int): ApiResponseDTO
SearchDTO
searchString: string
Filters: List<string>



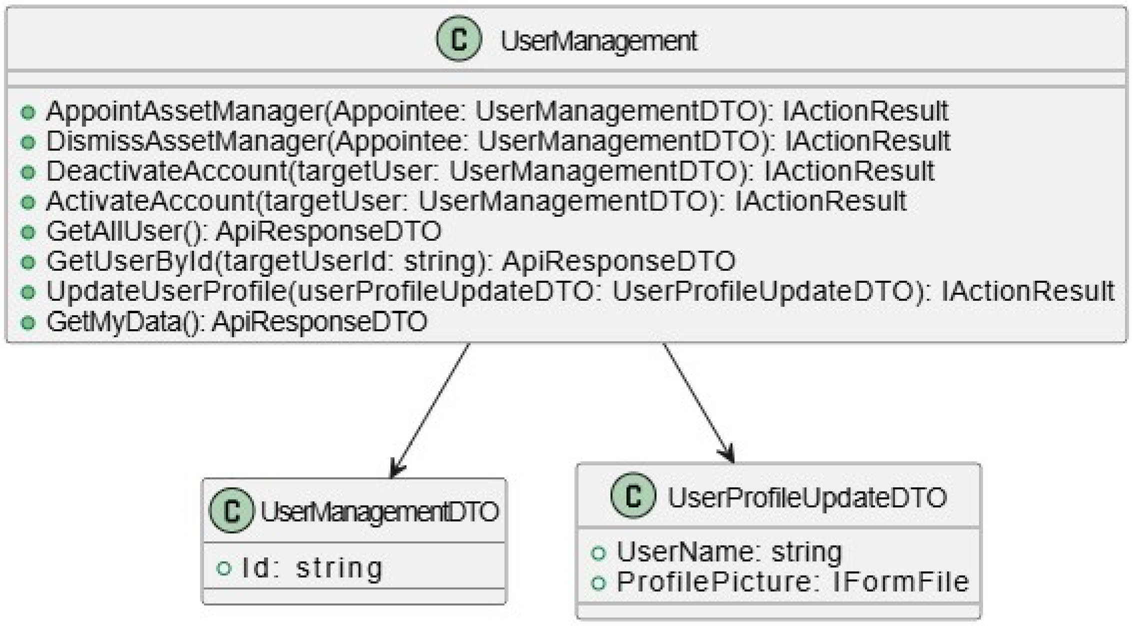
Authentication
Get(): IActionResult
SignUp(signUpDTO: SignUpDTO): IActionResult
ConfirmEmail(token: string, email: string): IActionResult
SignIn(signInModel: SignInDTO): IActionResult
ForgetPassword(forgetPasswordRequest: ForgetPasswordRequestDTO): IActionResult
ResetPassword(resetPasswordModel: ResetPasswordDTO): IActionResult
SignUpDTO
Email: string
UserName: string
Password: string
requiredRole: string
SignInDTO
Email: string
Password: string
ForgetPasswordRequestDTO
Email: string
ResetPasswordDTO
NewPassword: string
ConfirmedNewPassword: string
Email: string
Token: string



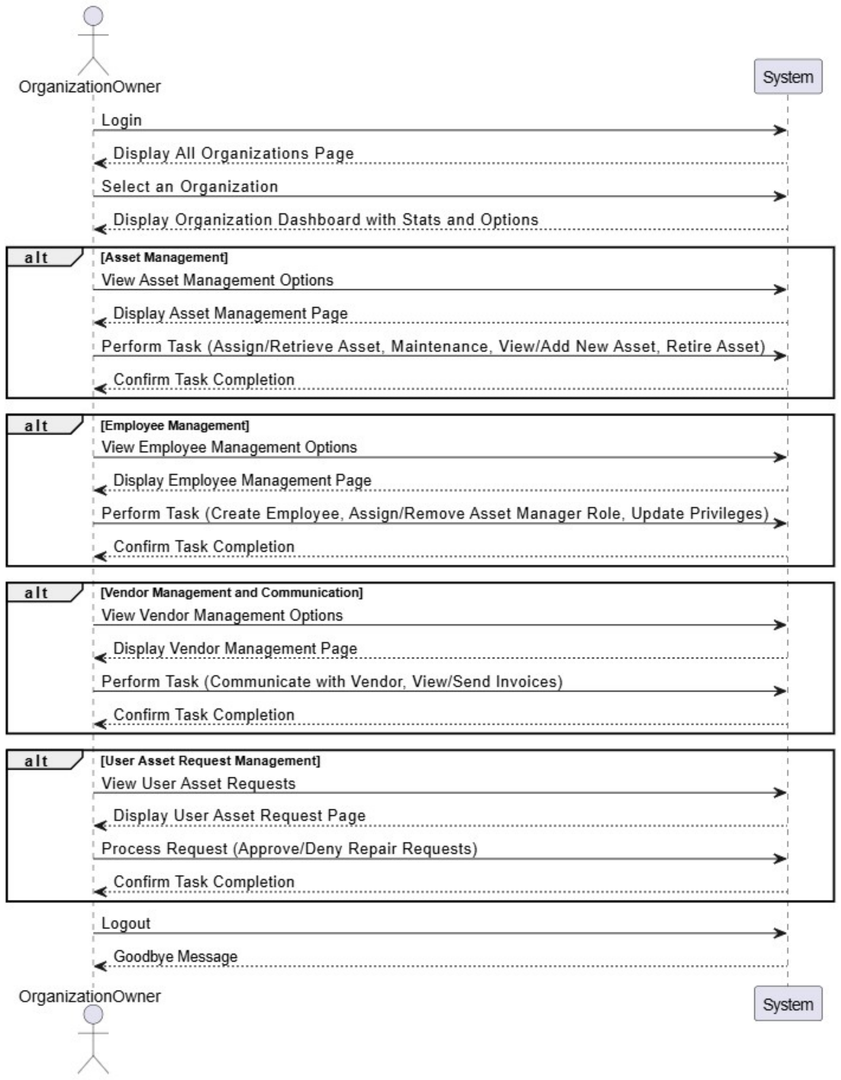
OrganizationManagement
GetOrganizationsInfo(): ApiResponseDTO
CreateOrganization(newOrganization: OrganizationDTO): IActionResult
UpdateOrganization(newOrganization: OrganizationDTO): IActionResult
DeleteOrganization(): IActionResult
OrganizationDTO
OrganizationName: string
Description: string
OrganizationDomains: List<string>

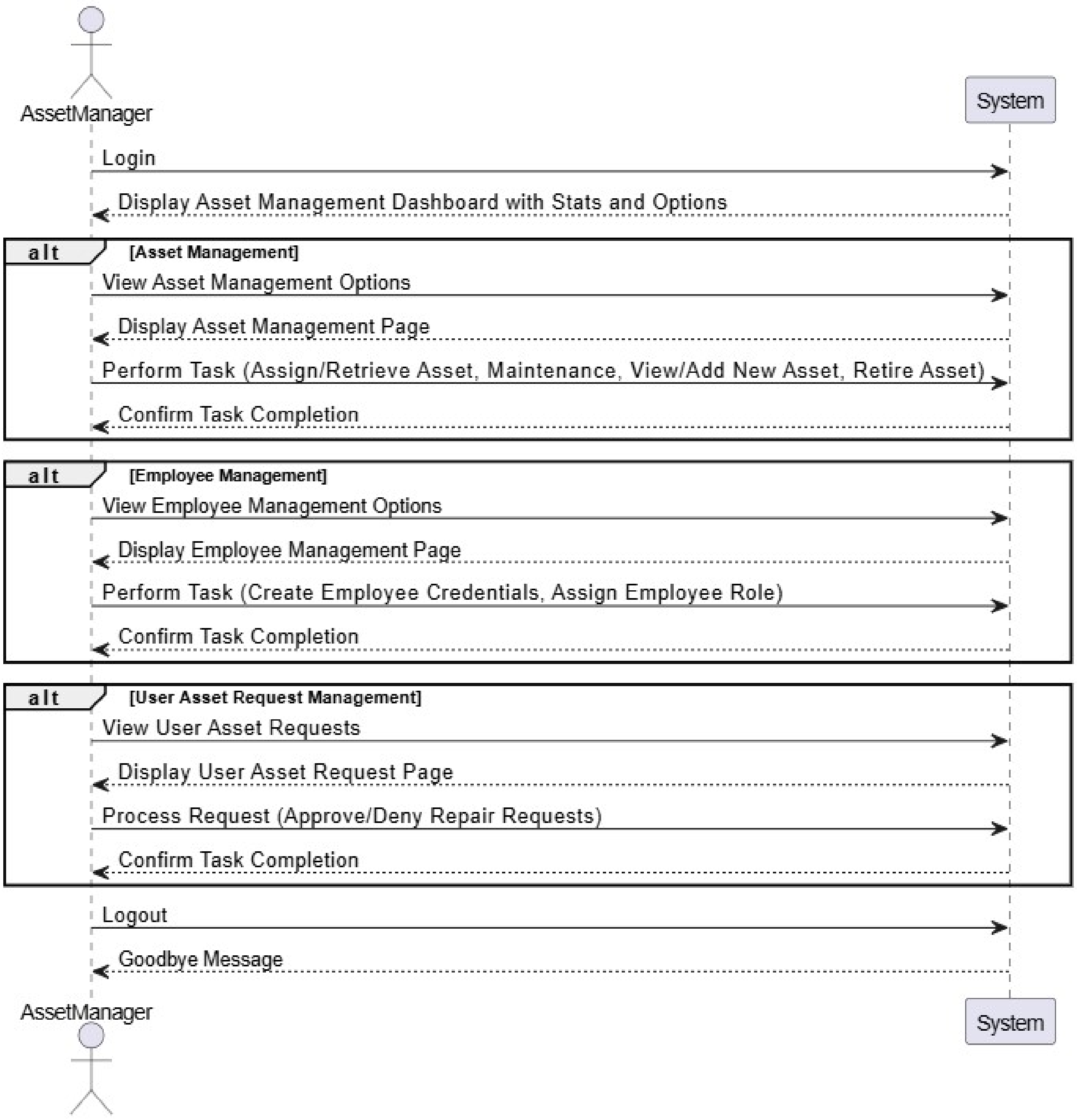


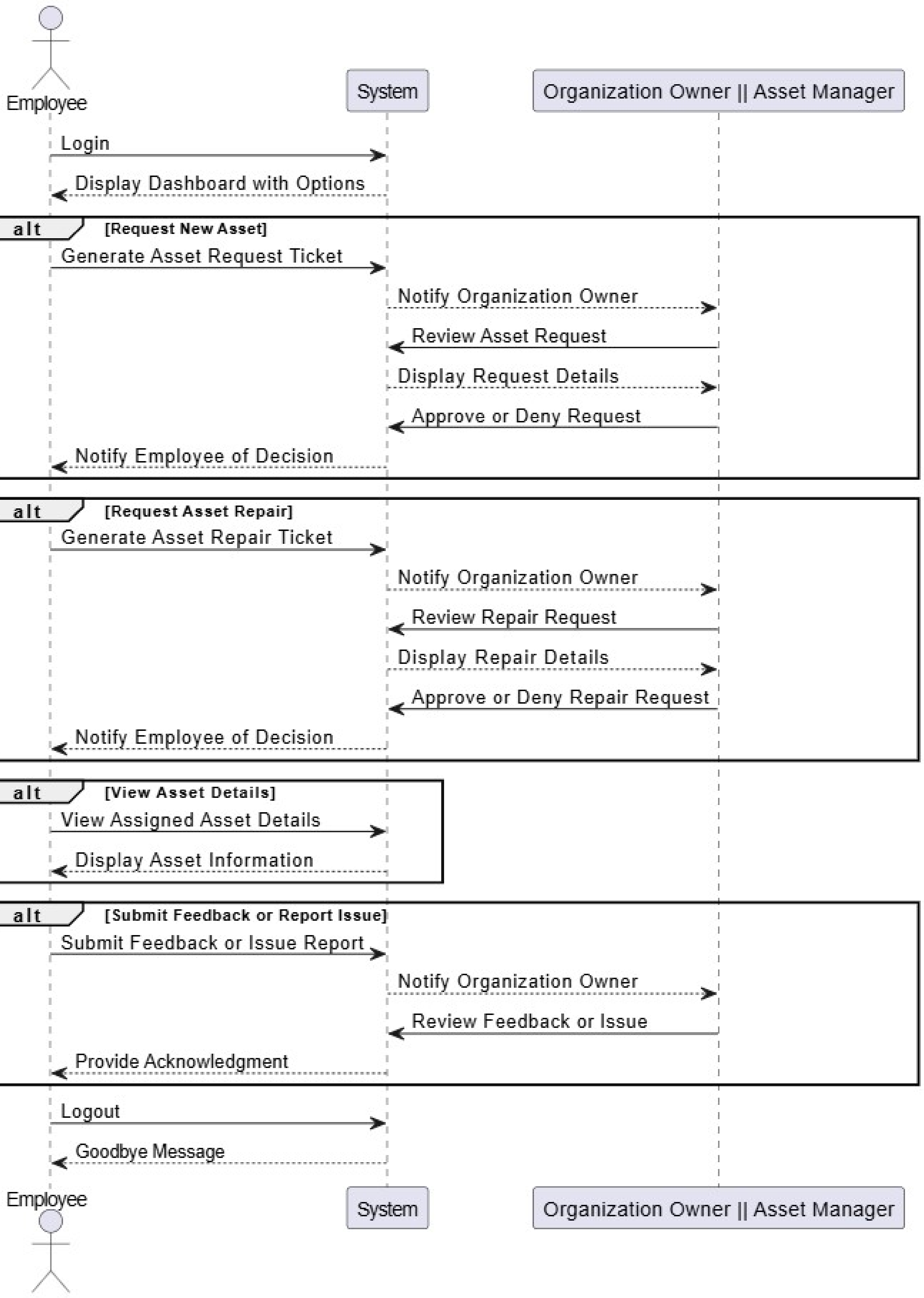
VendorManagement
CreateVendor(VendorDTO: VendorDTO): IActionResult
GetAllVendors(): ApiResponseDTO
UpdateVendor(VendorUpdate: VendorDTO): IActionResult
DeleteVendor(VendorId: int): IActionResult
GetVendorById(VendorId: int): ApiResponseDTO
VendorDTO
Id: int
Name: string
Email: string
OfficeAddress: string
PhoneNumber: string

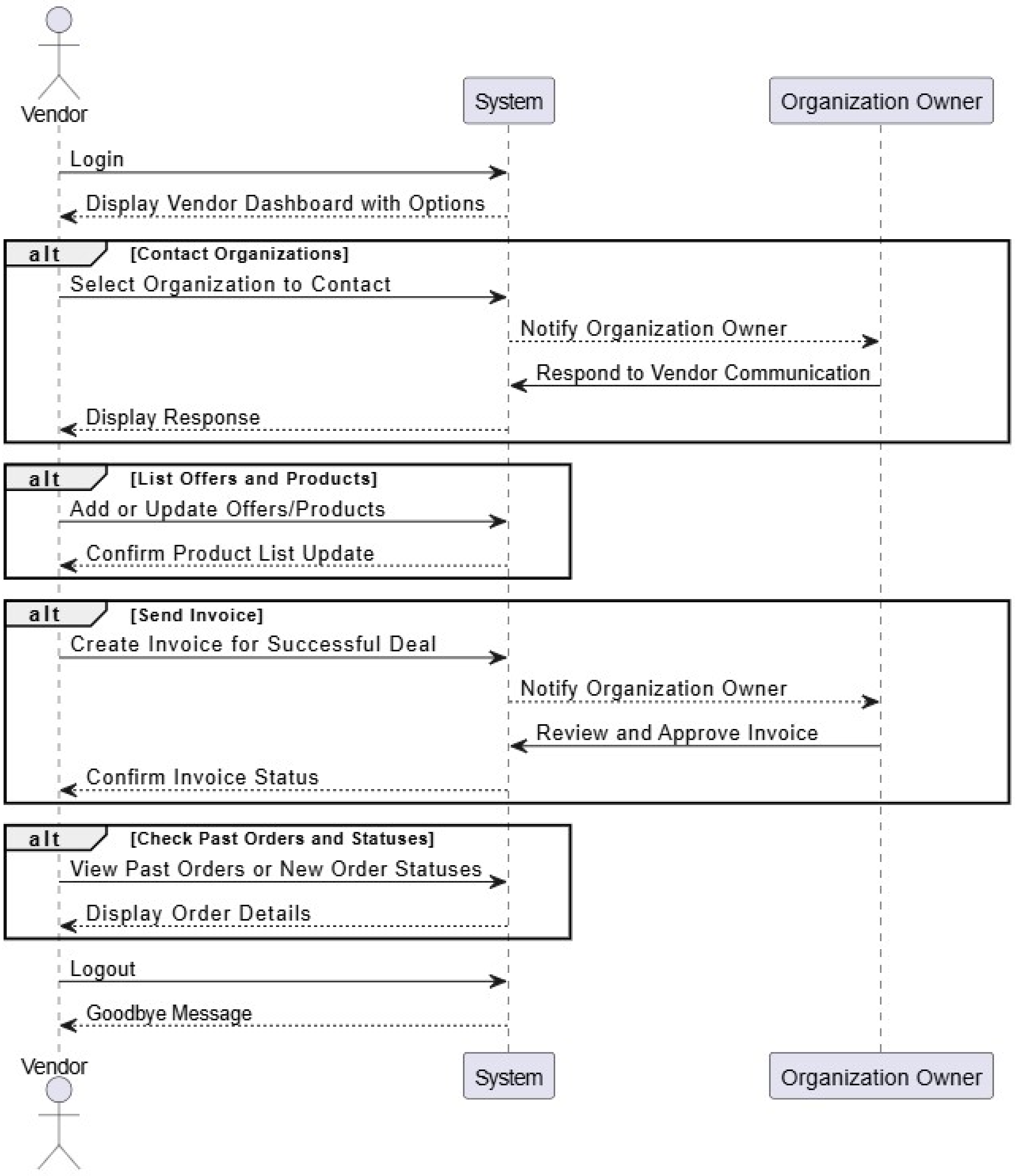

UserManagement
AppointAssetManager(Appointee: UserManagementDTO): IActionResult
DismissAssetManager(Appointee: UserManagementDTO): IActionResult
DeactivateAccount(targetUser: UserManagementDTO): IActionResult
ActivateAccount(targetUser: UserManagementDTO): IActionResult
GetAllUser(): ApiResponseDTO
GetUserById(targetUserId: string): ApiResponseDTO
UpdateUserProfile(userProfileUpdateDTO: UserProfileUpdateDTO): IActionResult
GetMyData(): ApiResponseDTO
UserManagementDTO
Id: string
UserProfileUpdateDTO
UserName: string
ProfilePicture: IFormFile

Sequence Diagrams:

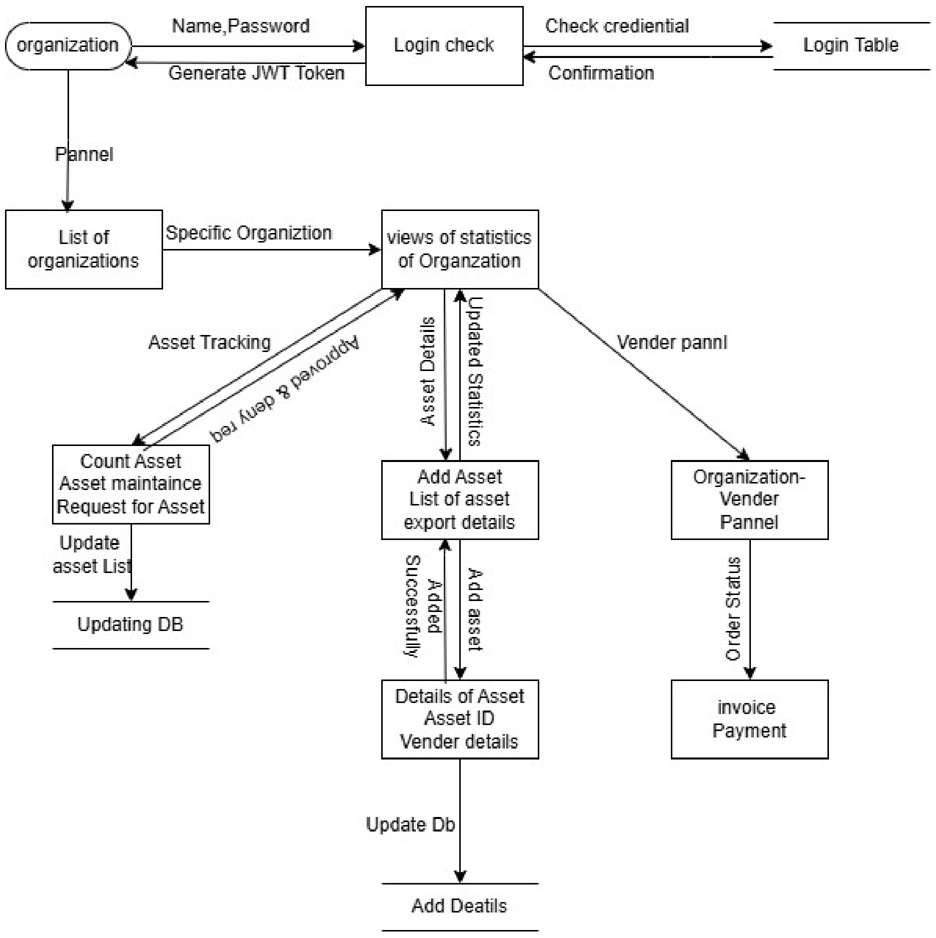

OrganizationOwner
System
Login
Display All Organizations Page
Select an Organization
Display Organization Dashboard with Stats and Options
alt
[Asset Management]
View Asset Management Options
Display Asset Management Page
Perform Task (Assign/Retrieve Asset, Maintenance, View/Add New Asset, Retire Asset)
Confirm Task Completion
alt
[Employee Management]
View Employee Management Options
Display Employee Management Page
Perform Task (Create Employee, Assign/Remove Asset Manager Role, Update Privileges)
Confirm Task Completion
alt
[Vendor Management and Communication]
View Vendor Management Options
Display Vendor Management Page
Perform Task (Communicate with Vendor, View/Send Invoices)
Confirm Task Completion
alt
[User Asset Request Management]
View User Asset Requests
Display User Asset Request Page
Process Request (Approve/Deny Repair Requests)
Confirm Task Completion
Logout
Goodbye Message
OrganizationOwner
System

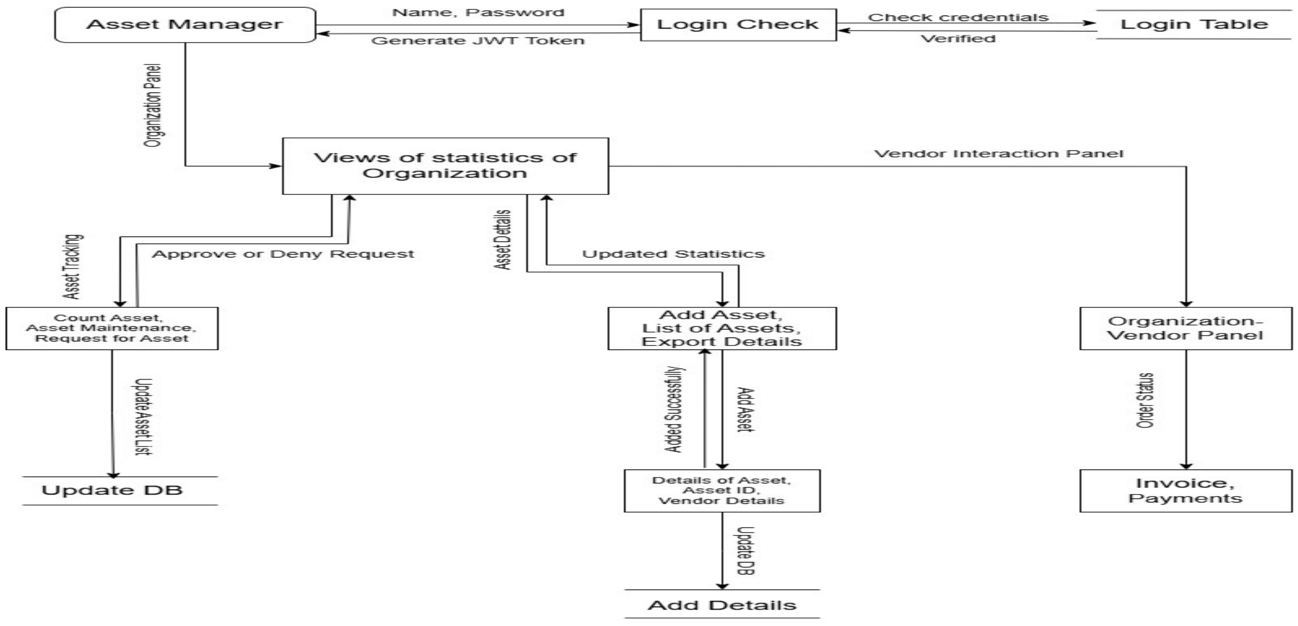

AssetManager
System
Login
Display Asset Management Dashboard with Stats and Options
alt [Asset Management]
View Asset Management Options
Display Asset Management Page
Perform Task (Assign/Retrieve Asset, Maintenance, View/Add New Asset, Retire Asset)
Confirm Task Completion
alt [Employee Management]
View Employee Management Options
Display Employee Management Page
Perform Task (Create Employee Credentials, Assign Employee Role)
Confirm Task Completion
alt [User Asset Request Management]
View User Asset Requests
Display User Asset Request Page
Process Request (Approve/Deny Repair Requests)
Confirm Task Completion
Logout
Goodbye Message
AssetManager
System

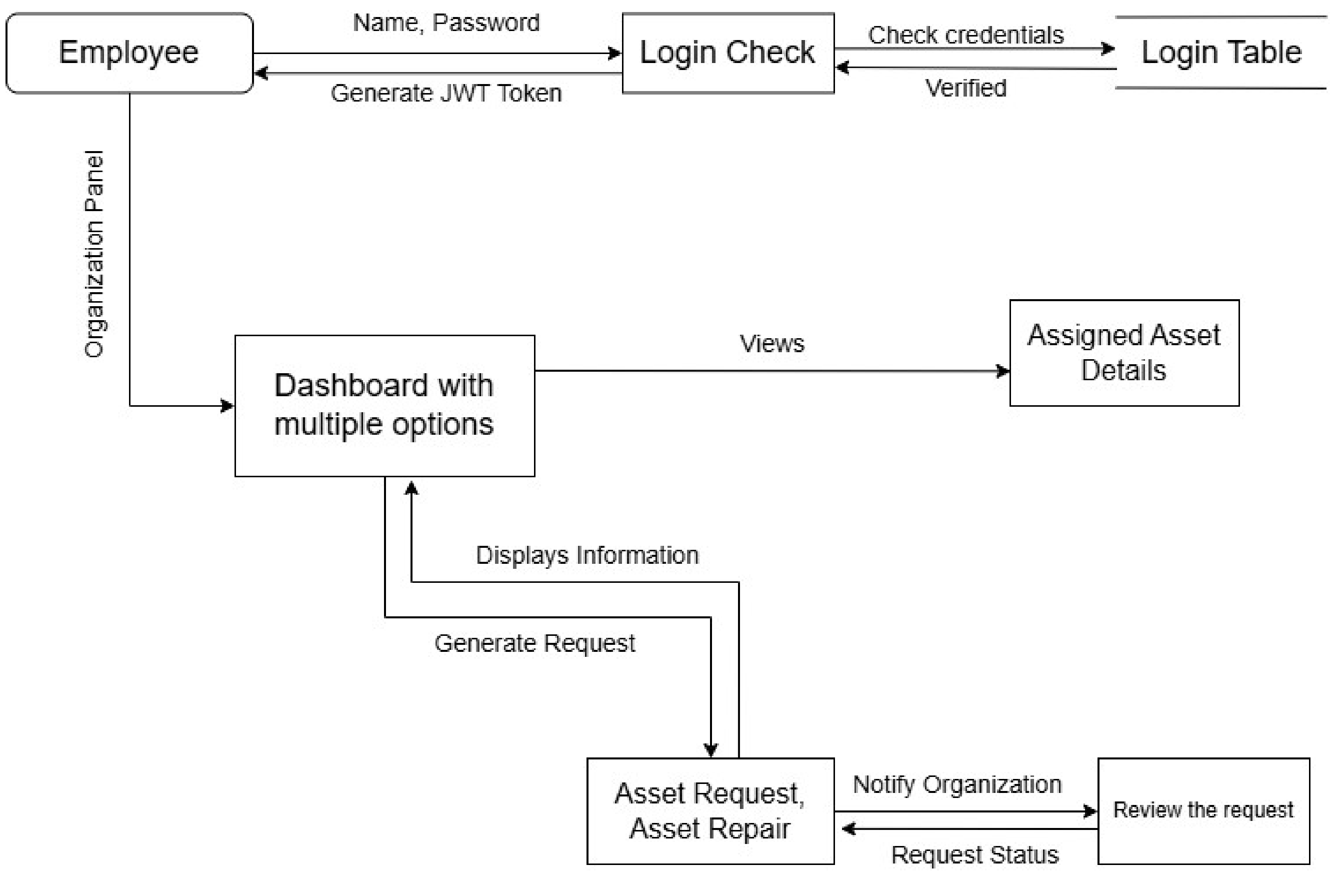

Employee
System
Organization Owner || Asset Manager
Login
Display Dashboard with Options
alt [Request New Asset]
Generate Asset Request Ticket
Notify Organization Owner
Review Asset Request
Display Request Details
Approve or Deny Request
Notify Employee of Decision
alt [Request Asset Repair]
Generate Asset Repair Ticket
Notify Organization Owner
Review Repair Request
Display Repair Details
Approve or Deny Repair Request
Notify Employee of Decision
alt [View Asset Details]
View Assigned Asset Details
Display Asset Information
alt [Submit Feedback or Report Issue]
Submit Feedback or Issue Report
Notify Organization Owner
Review Feedback or Issue
Provide Acknowledgment
Logout
Goodbye Message
Employee
System
Organization Owner || Asset Manager

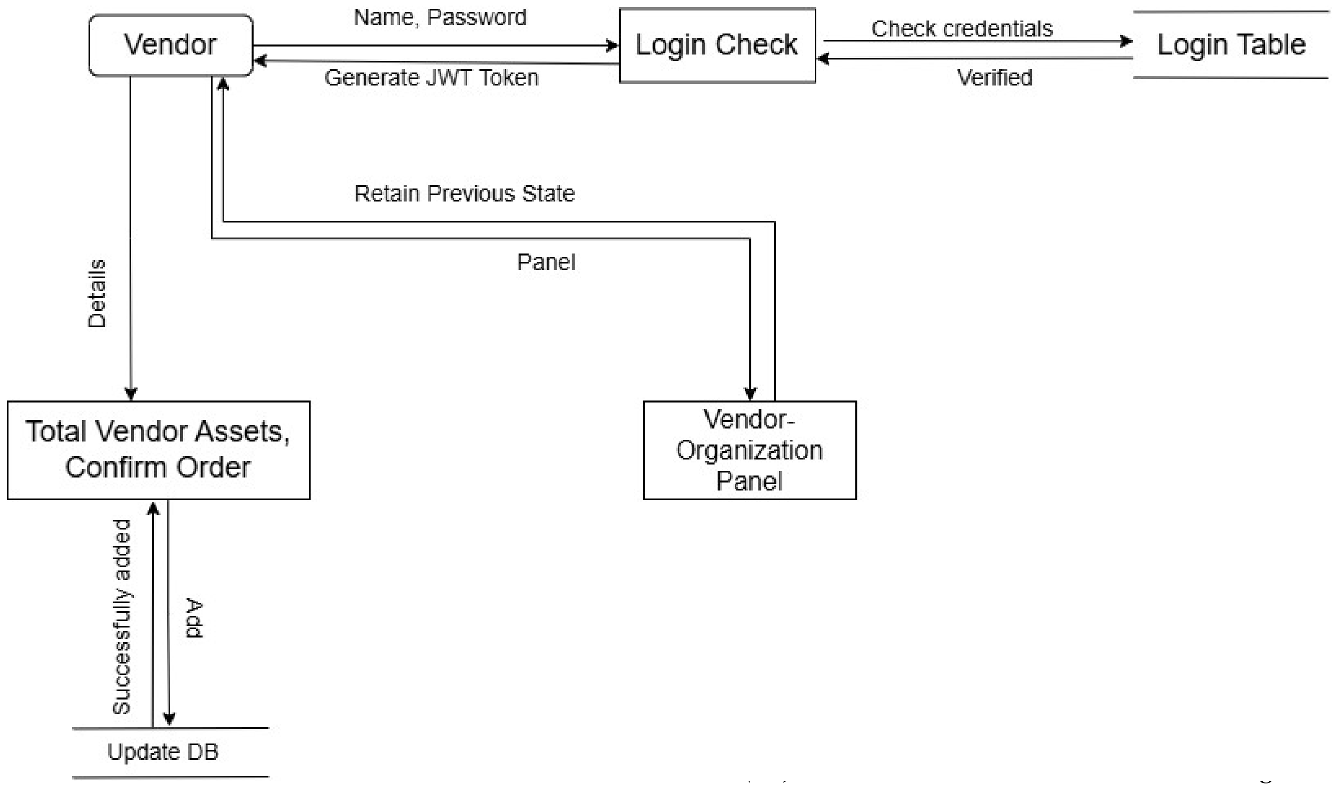

Vendor
System
Organization Owner
Login
Display Vendor Dashboard with Options
alt [Contact Organizations]
Select Organization to Contact
Notify Organization Owner
Respond to Vendor Communication
Display Response
alt [List Offers and Products]
Add or Update Offers/Products
Confirm Product List Update
alt [Send Invoice]
Create Invoice for Successful Deal
Notify Organization Owner
Review and Approve Invoice
Confirm Invoice Status
alt [Check Past Orders and Statuses]
View Past Orders or New Order Statuses
Display Order Details
Logout
Goodbye Message
Vendor
System
Organization Owner

Data Flow Diagram:






Employee
Name, Password
Generate JWT Token
Login Check
Check credentials
Verified
Login Table
Organization Panel
Dashboard with multiple options
Views
Assigned Asset Details
Displays Information
Generate Request
Asset Request, Asset Repair
Notify Organization
Request Status
Review the request


Vendor
Name, Password
Generate JWT Token
Login Check
Check credentials
Verified
Login Table
Retain Previous State
Panel
Details
Total Vendor Assets, Confirm Order
Vendor-Organization Panel
Successfully added
Add
Update DB

4. Nonfunctional Requirements

4.1 Performance Requirements

The performance requirements for **AssetIn** aim to ensure optimal functionality under various circumstances while providing a seamless experience for all users. These requirements address system responsiveness, scalability, and resource utilization to guide developers in making informed design choices.

System Responsiveness

- The system must respond to user actions (e.g., navigation, form submissions, and search queries) within 3 seconds under normal operating conditions.
- For resource-intensive tasks, such as generating reports or processing batch updates, operations should complete within 6 seconds for datasets under 10,000 records.

Scalability

- The platform must handle at least 500 concurrent users during peak usage without significant performance issues.
- It should dynamically scale to support up to 10,000 concurrent users using additional hardware or cloud resources as needed.

Database Query Efficiency

- Simple queries, such as retrieving asset details or generating reports, must execute within 2 seconds for datasets containing up to 50,000 records.
- More complex operations, like joins or filtered searches, should complete within 3 seconds for datasets of up to 100,000 records.

Media Retrieval

Media files (e.g., images or documents) stored in the cloud must be retrieved and displayed within 3 seconds for files smaller than 5 MB.

Real-Time Features

Real-time updates, such as asset tracking, must have a latency of no more than **2 seconds** for assets within the same network environment.

Functional Features Performance

- **Asset Tracking**: Asset statuses and locations must update in real time, with a maximum delay of 2 seconds for updates by managers.
- **Search and Filter**: Results for datasets up to 50,000 records should display within 3 seconds, while datasets up to 100,000 records must display results within 6 seconds.
- **Report Generation**: Detailed reports must generate within 10 seconds for datasets up to 100,000 records and optimize for download within 16 seconds for datasets as large as 1 million records.

- **Maintenance Scheduling**: Scheduling maintenance operations, including stakeholder notifications, should complete within 3 seconds.
- **Barcode Scanning**: Scanning and updating asset information via barcode must complete within 2 seconds after scanning.

4.2 Safety Requirements

The safety requirements for **AssetIn** aim to prevent harm, damage, or loss during its use, ensuring robust safeguards, regulatory compliance, and effective risk mitigation.

Data Integrity and Protection

The system will automatically back up critical asset data, such as location, status, media files, and user logs, at least once every 24 hours, securely storing the backups on a redundant cloud infrastructure. In the event of system updates or maintenance tasks, rollback mechanisms will be implemented to preserve data integrity, allowing the system to restore itself to a stable state in case of failure.

Access Control and Security

Role-based access control (RBAC) will ensure that only authorized roles can access sensitive actions and data. For instance, organization owners will be the only ones able to approve user requests, and asset managers will be responsible for updating asset statuses. Additionally, user authentication will be secured using JWT tokens to prevent unauthorized API usage or impersonation. To further enhance security, user sessions will automatically expire after 15 minutes of inactivity, reducing the risk of unauthorized access.

Error Handling and Recovery

The system will gracefully handle errors such as failed database queries or API calls, providing users with clear error messages without exposing sensitive system details. In case of failed operations, such as barcode scans or report generation, a recovery process will automatically retry the task or allow users to resume the operation without losing data.

Asset Management Safeguards

To avoid asset mismanagement, the system will prevent the deletion of active assets or categories that are linked to ongoing tasks, maintenance schedules, or requests. Before deletion can proceed, users will be prompted to resolve any dependencies. Additionally, input validation will be incorporated across all forms and data entry points to prevent SQL injection and cross-site scripting (XSS) attacks. All data transmissions, including login credentials and asset-related media, will be encrypted using SSL/TLS.

Compliance and Regulatory Standards

If the system is used in healthcare-related industries, it will comply with HIPAA (Health Insurance Portability and Accountability Act) to ensure the security of asset data tied to patient care. The cloud infrastructure used for storing media files will meet ISO/IEC 27001 standards for information security management and adhere to SOC 2 Type II standards for secure cloud operations. Asset-related notifications, such as those for maintenance or transfers, will be sent on time to prevent operational downtime or asset damage due to neglect.

Preventative Measures

The system will ensure that only authorized roles, such as organization owners or asset managers, can perform critical actions, including approving requests, updating asset statuses, or scheduling maintenance. Detailed audit logs will be maintained to track critical operations, enabling the monitoring and investigation of potential misuse. Scalability measures will be in place to dynamically allocate resources during peak usage, preventing system crashes or data loss. Additionally, rate limiting will be implemented to safeguard APIs from overuse or denial-of-service attacks. Critical system settings and configurations will be protected from changes during operational periods, such as ongoing maintenance or procurement cycles.

Safety-Critical Features

The system will include a disaster recovery plan to restore functionality within 4 hours in the event of a critical system failure or data breach. A secondary cloud region will be maintained for redundancy and failover support. The system will also warn users when performing potentially risky actions, such as deleting associated records or making significant configuration changes, requiring confirmation before proceeding. Real-time monitoring will detect unusual activity, such as multiple failed login attempts or unexpected data volume changes, and alert system administrators, allowing for quick responses to potential threats.

4.3 Security Requirements

The security and privacy requirements for **AssetIn** are designed to protect user data, maintain system integrity, and ensure compliance with relevant regulations. These requirements outline key measures to prevent unauthorized access, guarantee data confidentiality, and safeguard sensitive information.

User Authentication and Authorization

The system will implement Role-Based Access Control (RBAC) to restrict access to features based on user roles, such as organization owners and asset managers. JWT-based authentication will be used for secure session management, with sessions automatically expiring after 15 minutes of inactivity to prevent unauthorized access.

Data Encryption

To secure data transmission, all communications will be protected using SSL/TLS encryption. Additionally, user passwords will be stored as hashed values using bcrypt to prevent unauthorized access to sensitive credentials.

Audit Logging

Key actions within the system, such as logins, updates, and approvals, will be logged with timestamps. These logs will assist in debugging and monitoring system activity to ensure accountability.

Error Handling

The system will display generic error messages in the event of failures, ensuring that no sensitive system details are exposed to the users, thereby protecting the integrity of the application.

Data Collection and Control

Only necessary data will be collected, and users will have the ability to update their personal details (e.g., email addresses and passwords). Organization owners will be responsible for managing user accounts and organizational data to ensure proper access control.

Anonymization

To maintain user privacy, sensitive information will be anonymized in reports or analytics, ensuring that personally identifiable information is not exposed in non-essential contexts.

User Identity Authentication Requirements

The login system will validate user credentials against hashed records and limit login attempts to 5 failed attempts within 15 minutes to protect against brute force attacks. Additionally, a secure email-based password reset process will be provided for users to recover their accounts if necessary.

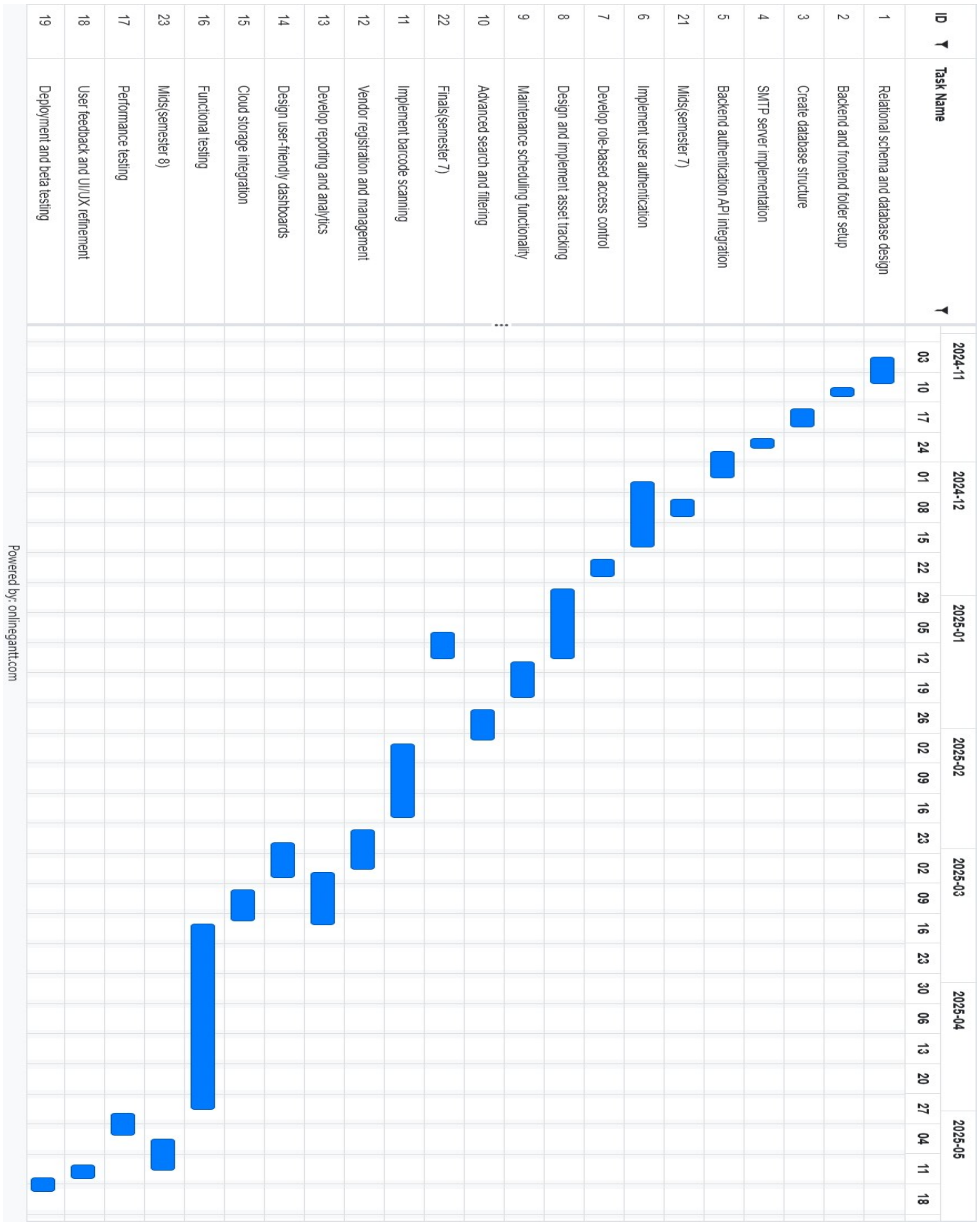
Backup and Recovery

The system will perform daily backups of critical data and provide recovery options in the event of data loss. This ensures business continuity and prevents data corruption or loss.

Validation and Prevention

The system will prevent unauthorized or risky actions, such as the deletion of active assets, by requiring user confirmation for critical tasks. This safeguard will help prevent accidental or malicious actions that could compromise data integrity or system functionality.

5. Revised Project Plan



ID	Task Name
1	Relational schema and database design
2	Backend and frontend folder setup
3	Create database structure
4	SMTP server implementation
5	Backend authentication API integration
21	Mids(semester 7)
6	Implement user authentication
7	Develop role-based access control
8	Design and implement asset tracking
9	Maintenance scheduling functionality
10	Advanced search and filtering
22	Finals(semester 7)
11	Implement barcode scanning
12	Vendor registration and management
13	Develop reporting and analytics
14	Design user-friendly dashboards
15	Cloud storage integration
16	Functional testing
23	Mids(semester 8)
17	Performance testing
18	User feedback and UI/UX refinement
19	Deployment and beta testing

Appendix A: Glossary

General Terms

Asset: Any tangible or intangible resource managed within the system, such as equipment, inventory, software licenses, or intellectual property.
Vendor: A third party or external organization that provides goods or services related to assets, such as repairs, maintenance, or sales.
Role-Based Access Control (RBAC): A security mechanism that restricts system access based on user roles (e.g., Organization Owner, Asset Manager).
Authentication: The process of verifying the identity of a user to grant access to the system.
Authorization: The process of granting or denying access to specific resources or actions based on the user's permissions.
Dashboard: A user interface that displays key metrics, statistics, and actionable insights for managing assets.
Cloud Integration: The use of cloud services to store, retrieve, and manage asset-related media files (e.g., images, videos).
Predictive Analytics: The use of data analysis techniques to forecast asset maintenance needs and optimize performance.
Audit Log: A chronological record of system activities, such as user actions or system changes, used for monitoring and debugging.
API (Application Programming Interface): A set of rules and tools for building software and enabling communication between the backend and frontend or external systems.

Abbreviations and Acronyms

SRS: Software Requirements Specification: A document detailing the functional and nonfunctional requirements of the project.
SMTP: Simple Mail Transfer Protocol: A protocol used to send emails within the system, such as notifications or password reset links.
JWT: JSON Web Token: A compact and secure method for transmitting information between the client and server for authentication and authorization.
SSL/TLS: Secure Sockets Layer / Transport Layer Security: Protocols used to encrypt communications between the client and server.
CRUD: Create, Read, Update, Delete: Basic operations performed on database records.
UI/UX: User Interface / User Experience: The design and experience of interacting with the system.
RBAC: Role-Based Access Control: A security feature that restricts system functionality based on user roles.
GDPR: General Data Protection Regulation: A regulation designed to protect user data privacy (used here for awareness).
AES: Advanced Encryption Standard: A method for securely encrypting sensitive data at rest.

6. References

[1] “Ralph3 Documentation,” *Ralph3 Official Site*. [Online]. Available: https://ralph3.readthedocs.io/. [Accessed: 20-Jan-2024].

[2] “Snipe-IT Asset Management,” *Snipe-IT Official Documentation*. [Online]. Available: https://snipe-it.readme.io/. [Accessed: 20-Jan-2024].

[3] “Asset Tiger Review,” *Asset Tiger Blog*. [Online]. Available: https://www.assettiger.com/blog/. [Accessed: 19-Jan-2024].

[4] Angular Team, “Angular Documentation,” *Angular*. [Online]. Available: https://angular.io/docs. [Accessed: 18-Jan-2024].

[5] Microsoft, “ASP.NET Core Overview,” *Microsoft Documentation*. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/. [Accessed: 18-Jan-2024].

[6] Oracle, “MySQL 8.0 Reference Manual,” *MySQL Official Documentation*. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/. [Accessed: 19-Jan-2024].

[7] T. Koller and M. Goedhart, *Valuation: Measuring and Managing the Value of Companies*, 7th ed. Wiley, 2020, pp. 45–67.

[8] J. Smith, “Predictive maintenance in asset management,” *Journal of Engineering Management*, vol. 23, no. 4, pp. 321–333, Dec. 2021.

[9] “General Data Protection Regulation (GDPR),” *European Union*. [Online]. Available: https://gdpr-info.eu/. [Accessed: 18-Jan-2024].

[10] “HIPAA Compliance,” *U.S. Department of Health & Human Services*. [Online]. Available: https://www.hhs.gov/hipaa/index.html. [Accessed: 17-Jan-2024].

[11] ISO/IEC 27001:2013, “Information Security Management Systems,” *International Organization for Standardization*. [Online]. Available: https://www.iso.org/standard/54534.html. [Accessed: 17-Jan-2024].

[12] Postman Team, “Postman API Testing,” *Postman*. [Online]. Available: https://learning.postman.com/docs/writing-scripts/. [Accessed: 19-Jan-2024].

13] Swagger, “Introduction to API Testing,” *Swagger.io*. [Online]. Available: https://swagger.io/docs/. [Accessed: 19-Jan-2024].

Appendix B: IV & V Report

(Independent verification & validation)
IV & V Resource

Name	Signature

S#	Defect Description	Origin Stage	Status	Fix Time	
				Hours	Minutes
1					
2					
3					
…					

Table 2: List of non-trivial defects