Advanced Software Engineering

Group members:

Xhesjano Halla

Anjeza Xhelilaj

Ema Elezi

Laura Dule

Department: Computer Engineering

Course: Advanced Software Engineering

Project: Inventory system

Accepted by: Dr. Igli Hakrama

Table of Contents

1. Summary of the project

1.1 Project overview

To keep track of all the stocks of the university throughout the whole supply chain can be rather challenging and tiring if done by hand. Writing everything in a piece of paper can be tiring and introduce a lot of mistakes that can lead to many situations that the university does not want to be in. In order to keep track of all the goods and prevent mistakes the university should have a system that sorts everything in classes, offices of the staff and in the storage rooms. The inventory system web application is the best solution to sort all of these problems. The information will be stored in a protected server which will be flexible in the way of modifying it.

1.2 Purpose and scope of this specifications

The focus of our program will be **STOCKS**. Everything will be related to the stocks of the university.

Stocks will be divided into **Classes, Offices, Storages** and **Laboratories.**

This program will help to keep track of the stock of the university via the web-application which is easier to manage rather than doing everything by hand. This web-application will save time from all the parties, and it will make the inventory process more efficient, and the users will be informed of the stocks the university has, what stock should be replaced and what is need of.

Scope:

The University Inventory Management System will provide a structured approach to organizing and maintaining inventory records. The system will include:

- **Categorization of Goods** – Items will be classified based on their location (classrooms, staff offices, and storage rooms).

- **Real-Time Inventory Tracking** – Ensuring up-to-date records of available assets, including quantity and condition.
- **Automated Alerts and Reports** – Notifications for maintenance requirements and low stock.
- **User Roles and Access Control** – Allowing different levels of access for staff, administrators, and maintenance teams.

2. System description

Our proposed platform is designed to streamline the management of inventory within a university, ensuring efficient tracking, organization, and accessibility of resources. Universities often struggle with maintaining a well-structured inventory system, leading to misplaced assets, inefficient resource allocation, and increased operational costs.

Our system aims to provide an efficient and user-friendly solution for managing equipment, supplies, and other university assets across various departments. By integrating real-time tracking, automated restocking alerts, and detailed usage reports, our platform enhances resource management while reducing manual workload.

Furthermore, the platform is designed to be intuitive and adaptable to the daily operations of university staff, ensuring seamless inventory control without requiring extensive training.

2.1 Product Context

Our platform is independent and self-contained. While other university inventory management systems primarily focus on internal operations within specific departments, our platform extends its functionality to provide a centralized solution for managing inventory across the entire institution. It enables universities to

efficiently track, allocate, and maintain their assets while ensuring transparency and accessibility for all relevant departments.

All inventory data, including asset records, and stock levels, will be created, stored, and managed exclusively within our platform, without reliance on external systems. This ensures data security, consistency, and ease of use for university staff.

A detailed view of how the system operates can be found in our GitHub repository: https://github.com/XhHalla/Inventory-Management-System .

2.1 User characteristics

This program will be used by these users:

- University Staff
- Finance
- IT
- Club Coordinators
- Service Department

a) Finance

Finance will be an important administrator which will decide which stock will be removed, where the stock will be placed and in which storeroom it will be held. This administrator will manage the inventory system, also will keep track of inventory status to buy new items if needed and "predict" the items that they would need to buy soon. The Finance department will also take care of the user privileges and roles. They can also accept or decline requests from University Staff, IT, Handyman, or Club Coordinators.

b) University Staff

The University Staff user would be able to request a quality check for their office or class. They can make this request for an upcoming

event so the IT department would make sure that the status of devices in that class have not changed and the handyman can make sure that everything else on that environment is in a good condition. University staff would be responsible for editing the status of items in its office and they also can edit the status of items in a class that they are teaching (Professor).

c) IT

The IT department will get requests from different user roles upon checking the status of a device and updating it, changing a device if it is not in a working condition. IT will be aware if any stock is needed that is related to them.
This administrator will decide where the stock will be placed, and it will also be updated in the inventory web-application system. If the IT department wants to add a device or needs a device it will make a request which will go to the Finance department.

d) Service Department

The Service Department will get requests from different user roles upon on checking the status of classes and updating it, changing or fixing the materials that are not in working condition. They will be aware if any stock is needed.

The Service Department will decide where the stock that he is managing will be placed and it will also be responsible for updating the inventory. They will also be responsible for forwarding the requests to Finance whenever a new item is needed.

2.2 Assumptions

- It is assumed that the data generated and registered will be fully confidential and it will be available only for the specific departments of the university.

- It is assumed that every user has the appropriate equipment (computer or mobile device, internet access) to use the program.
- It is assumed that the users of this program have general knowledge on how to use the internet and their respective devices effectively.
- It is assumed that every change that happens should be updated in the system for better organization and better usage of this system.
- It is assumed that every user of the program has his/her university email activated and working properly.
- It is assumed that the stock is already registered by the service department.
- It is assumed that the users will update the inventory for every object that is missing, damaged or broken.
- It is assumed that we have all the barcodes for each product registered.

2.3 Constraints

The system may be constrained by:

- Having every user to understand how this system works and making sure they do not do any mistakes.
- Our system is a standalone platform that does not integrate with external inventory management systems.
- Problems with access management or security.

- The system does not allow the user to make customizations regarding the system name components as it may cause anomalies.
- Inventory requests must be approved by designated Inventory Managers before items can be issued.
- Other constraints can be found on the way.

2.4 Dependencies

List of dependencies that affect the requirements:

- The user should always have internet access.
- The user should have an electronic device.
- The need for at least moderate internet speed.
- There should be a user from the Finance or IT department to make changes such as adding or deleting a user, adding or deleting stocks etc.
- The response time depends on the volume of requests and the number of users that are currently using the system.

3.1.1 Functional Requirements

1. **Item Management**: The system should allow specific users to add, modify or delete items in the inventory database. This includes the ability to enter item descriptions, unit of measure (such as pieces, etc.), and stock levels. The item management feature should also allow users to categorize items into groups, such as product type which will be separatable in two main groups. One of the groups will be managed by the IT department and the other will be managed by the Service Department. The Item Management system will also be able to move the items in different classes, offices or storage rooms.

2. **Stock Management:** The inventory management system should track the current stock levels of each item and provide notifications for low stock levels. This helps ensure that there is enough inventory to fulfill the demands and to prevent stockouts. The system should also allow users to set reorder points for each item, which triggers a notification when the stock level drops below a certain threshold.

3. **Order Management:** The system should provide a way to manage and track purchase orders. This includes the ability to create, approve, and fulfill purchase orders. The order management feature should also allow users to view order history.

4. **Reporting:** The inventory management system should provide a variety of reports that allow users to view information about their inventory and purchasing activities. This may include reports on stock levels, purchase history and states of each item. The system should also allow users to customize reports and export data to other formats, such as Excel or PDF.

5. **Barcode and Scanning:** The system should provide support for barcode scanning and QR code scanning. This makes it easy for users to scan items for accurate and efficient inventory management. The barcode and scanning feature should also allow users to print barcodes for items that do not already have them.

6. **Multi-Location Support:** The system should allow users to manage inventory across multiple locations. This may include the ability to transfer inventory between locations and to view inventory levels for each location.

7. **User Management:** The system should allow administrators to manage user accounts, assign roles and permissions, and control access to the system. This helps ensure that only authorized users have access to sensitive information, such as financial data or storage room locations with their respective items.

8. **Mobile Accessibility:** The system should be accessible from mobile devices, allowing users to manage or view their inventory, make requests on the go. This includes the ability to access inventory levels, add or modify items and view reports.

9. **Data Security:** The system should provide robust data security to protect sensitive information, such as financial data, customer information, and inventory details. This includes secure storage of data, secure transmission of data, and access controls to prevent unauthorized access. The system should also have backup and recovery features to ensure that data can be restored in the event of a disaster.

10. **Real-time updates:** The system should provide real-time updates on inventory levels, status of the items in the inventory and purchasing activities. This helps users stay up to date with the latest information and make informed decisions.

13. **Stock valuation:** The system should provide a way to value inventory, such as using first-in, first-out (FIFO) or last-in, first-out (LIFO) methods. This helps users determine the value of their inventory for financial reporting purposes.

16. **Customizable dashboards:** The system should allow users to create custom dashboards that display real-time information about their inventory, their status and purchasing activities.

17. **Data import/export:** The system should allow users to import and export data from other systems, such as spreadsheets or databases, to help streamline data entry and reduce manual data entry.

18. **Automated alerts:** The system should provide automated alerts for low stock levels, and other events such as status updates, helping users stay informed and take timely action.

19. **Stock adjustment:** The system should allow users to make adjustments to stock levels, such as to correct discrepancies or to account for damaged items.

20. **Warehouse management:** The system should provide a way to manage and track warehouse activities, such as receiving, storing. This may include features such as barcode scanning, pick and pack functionality.

22. **Customizable workflows:** The system should allow users to customize workflows, such as purchase order approval processes, to meet the unique needs.

23. **Analytics and business intelligence:** The system should provide robust analytics and business intelligence tools to help users gain insights into their inventory and purchasing activities. This may include data visualization, trend analysis, and predictive analytics.

24. **Purchase order approval:** The system should provide a way for users to approve purchase orders, helping ensure that only authorized purchases are made.

25. **Product management:** The system should provide a way to manage and track product information, such as product descriptions, status, pricing, and images. This helps users keep their product information up-to-date and accurate.

26. The requirements for an "on-click login" feature using a Microsoft account could include the following:

- **User registration:** The ability for users to create a Microsoft account if they do not already have one.
- **Login button:** A button or link that allows users to initiate the login process with a single click.
- **Microsoft account authentication:** The ability to authenticate a user's Microsoft account credentials and retrieve their profile information.
- **User authorization:** The ability to grant the user's authorization for the application to access their Microsoft account information.
- **Token management:** The ability to securely store and manage the access token received from Microsoft.
- **Single sign-on (SSO):** The ability to recognize a user who has already logged in with their Microsoft account and automatically log them in without requiring additional authentication.
- **User profile information:** The ability to access and display the user's Microsoft account information, such as their name, email, and profile picture.
- **Logout:** The ability for users to log out of their Microsoft account and end the session.
- **Error handling:** The ability to handle and display error messages in the event of authentication or authorization failure.
- **Security:** The ability to securely store and transmit sensitive user information, such as passwords and access tokens, in accordance with industry standards.

3.2 Non-Functional Requirements

3.2.1 Product Requirements

3.2.1.1 User Interface Requirements

In general:

- The user interface must be simple, user-friendly, and easy to navigate.
- Each role in the system has its own user interface.
- The user interface should be securely accessible only upon successful login with valid credentials, after which the system directs the user to their authorized dashboard based on their role
- Each user page should have a horizontal navigation menu occupying approximately one-third of the screen and allowing users to easily browse through different sections. The remaining two-thirds of the screen will be a dynamic dashboard that displays content based on user characteristics.
- The application should be fully responsive and compatible with various screen sizes, including desktop, mobile, and tablet devices.

User – based Requirements:

Finance (Admin)

- The navigation bar will contain: Homepage, Check Stock, Edit Location, Manage Users.
- Homepage will display stock requests from IT and Service Departments in a card layout with buttons: Approve, Pending, and Decline.

- An automated notification will be sent based on the selected action.

- Check Stock will show cards representing university buildings and clubs. Upon selecting a building, users can choose to view Class, Office, or Storage Room and see item status via detailed cards or a table format.

- Edit Location page will allow Finance to add, edit, or delete buildings, classes, and offices using an intuitive card-based UI. Clicking a "+" card will open a form for adding new locations.

- The Manage Users page will display all users with filters by role. Finance can add, edit roles, or delete users. User info stored includes: name, email, encrypted password, and assigned role.

IT

- The navigation bar will contain: Homepage, Check Stock, Stock Request.

- Homepage shows incoming requests from University Staff or Club Coordinators (e.g., QA or stock requests) as cards with buttons: Accept, Forward, Decline.

- Check Stock displays only IT-related equipment for classes, offices, and storages. IT can view, add, edit, or delete items.

- The Stock Request form requires fields like item name, quantity, and reason for request, which will be forwarded to Finance.

University Staff

- The navigation bar will contain: Homepage, QA Request, Stock Request.

- Homepage will display either:
- Their assigned office with listed items and editable status fields.
- Or a building/classroom navigation to find and update item statuses themselves.
- If an item is marked “Not Working,” a request will be auto generated for the relevant department.
- QA Request and Stock Request will share one page with two image-based buttons:
- Clicking QA opens a form requiring location and reason.
- Clicking Stock Request opens the request form with a department field.

Club Coordinators

- The navigation bar will contain: Homepage, Stock Request.
- The homepage displays items assigned to their club, with options to update item status.
- Stock Request will use the same form format, allowing coordinators to select the target department (IT or Service).

Service Department

- The navigation bar will contain: Homepage, Check Stock, Stock Request

- The homepage displays requests similar to IT's, including QA or maintenance tasks from Staff or Coordinators

- Check Stock allows the Service Department to view and manage only service-related items, with full CRUD operations

- The Stock Request page will follow the same format as IT's, including a department-specific request form

3.2.1.2 Usability

1. Learnability: New users (e.g., university staff or club coordinators) should be able to learn how to use the system within 15 minutes of first interaction, without the need for formal training, as a short onboarding tooltip or guide will be available to assist them during their first use.

2. Navigation & Interface Consistency: All users will see a similar navigation bar with options based on their role, making it easy to navigate the system. The overall design and layout of pages such as forms, cards, and tables will stay consistent for everyone, with only the content changing depending on the user. This consistency helps users complete their main tasks, like checking stock or making a request, in just a few clicks from the homepage.

3. **Feedback & Error Handling:** The system provides instant visual feedback whenever a user takes an action, such as submitting a request or updating an item's status. If a user input is invalid, like leaving a required field empty, the system highlights the problem and shows a clear message explaining what needs to be fixed. After successfully submitting or updating a form, a confirmation message appears to let the user know the action was completed.

4. **Accessibility:** All features in the system should be fully usable using keyboard navigation, allowing users to move through elements with keys like Tab and Enter. To ensure accessibility, all button and label text must be clear and easy to read, with proper contrast between text and background.

5. **Task Efficiency:** Users should be able to filter and sort data (like stock items or users) quickly with dropdowns or search bars.

6. **Mobile Responsiveness (Optional/Stretch Goal):** If the system will be used on tablets or mobile, all pages should be responsive and maintain usability on smaller screens.

7. **Satisfaction:** During testing, at least 80% of users should rate the system as "easy to use" or better on usability surveys.

3.2.1.3 Efficiency

3.2.1.3.1 Performance Requirements

The application will be a web-based platform hosted on a web server.

Its performance will depend on:

- The speed and stability of the internet connection
- The efficiency of data retrieval from the database
- The number of users actively accessing the application simultaneously

Performance also depends on:

1. **Response time:** The system should respond to user interactions (such as submitting a request or viewing stock) within **2 seconds** under normal network conditions.

2. **Throughput:** The system should be able to manage large volumes of transactions and data while maintaining fast and efficient processing speeds. This will help ensure high throughput and smooth performance, even when multiple users are interacting with the system at the same time.

3. **Performance optimization:** The system should be designed with performance in mind, using efficient algorithms and well-structured data handling to reduce latency and ensure quick response times. This will help provide a smooth and responsive experience for all users.

4. **Scalability:** The application should be designed to scale horizontally, allowing additional server resources to be added when usage increases, especially during peak periods like semester starts or inventory updates.

5. **Load testing:** The system should undergo load testing to assess its performance under varying levels of user demand and to identify potential performance bottlenecks.

6. **Data Retrieval:** Queries fetching stock or user data should return results within 1–3 seconds, even when accessing large datasets (e.g., 5000+ items).

7. **Monitoring and analysis:** The system should include tools for monitoring and analysis, like real-time tracking and logging, to help detect performance issues and improve response times

8. **Caching:** The system should use caching methods, like keeping frequently accessed data in memory, to speed up response times and lower latency.

9. **Distributed systems:** If the system runs on multiple servers, it should apply distributed system techniques like load balancing and data replication to boost performance and maintain fast, responsive operation.

3.2.1.3.2 Space Requirements

1. **Efficient Resource Utilization:** The system should be optimized to use a reasonable amount of RAM and storage, ensuring smooth operation on standard university hardware and web servers.

2. **Optimized Data Storage and Management:** The database must efficiently store inventory data, including item records, location details, user accounts, and request logs, while minimizing redundancy.

Our system will primarily store structured inventory records—item details, location data, user accounts, and request logs—which are very lightweight. Assuming we support 1,000 active users generating about **5 MB** of text-based data per month, that amounts to **5 MB × 12 × 10 = 600 MB** of logs over 10 years. In addition, if we include up to **5,000 item photos** at **0.2 MB** each, that adds another **1 GB** of storage. Even allowing for a comfortable growth buffer, all of this fits well within a **1 TB** disk, ensuring ample room for backups and future expansion.

3.2.1.4 Dependability Requirements

3.2.1.4.1 Availability

1. **Uptime:** The system should have a high uptime, with minimal downtime, to ensure that users have continuous access to the system and its data.

2. **Load balancing:** The system should provide load balancing, such as through a load balancer, to distribute traffic across multiple servers and ensure that the system remains available even under heavy load.

3. **Disaster recovery:** The system should provide a disaster recovery mechanism, such as a secondary backup server, to ensure that the system can recover from failures and continue functioning even in the event of a disaster.

4. **Backups:** The system should provide regular backups, such as daily or weekly backups, to ensure that data can be recovered in the event of failures or outages.

5. **Error Handling and Notifications:** If an issue occurs or an action results in an error, the system will display clear and relevant messages to inform the user about the problem.

3.2.1.4.2 Latency

- Database size, which can influence how quickly data is accessed and processed
- Network bandwidth and internet connection speed, impacting data transfer rates
- Efficiency of database queries and data retrieval processes

- The number of concurrent users interacting with the application at a given time

Latency also depends on:

1. **Latency targets:** The system must achieve defined latency goals, delivering responses within a specified number of milliseconds to maintain a fast and responsive user experience.

2. **Real-time updates:** The system should deliver live updates, ensuring that users can access the most current information instantly, without noticeable delays.

3. **Network optimization:** The system should be designed to use network bandwidth efficiently and minimize network latency, ensuring a fast and responsive experience for users.

3.2.1.4.3 Monitoring

- When an error occurs, a message will appear to notify the user about the issue and its possible causes
- Users must enter the correct and corresponding email and password to access the application. If the login credentials are invalid, an error message will notify the users
- The application will be designed with reliability in mind, ensuring that all user-entered information is properly validated
- A user hierarchy will be established as previously outlined by the HR department

3.2.1.4.4 Maintenance

If the application crashes, the system will automatically restart and redirect the user to the homepage, preserving any actions previously performed. In such cases, it is safe for users to refresh the webpage. However, if the issue persists and the application crashes again, a maintenance break will be triggered, and users will be notified accordingly.

- The system will undergo scheduled maintenance once a week, typically during midnight on weekends, for approximately 30 to 60 minutes when necessary

- The system will be maintained by the development team, particularly when updates are needed or issues arise. New features will be introduced based on user needs and feedback

- Ongoing support for the system will always be available. Developers can be contacted through messages sent by the platform administrator

3.2.1.4.5 Integrity

- The integrity of the university inventory system is maintained through detailed system logs and routine daily reviews of these logs to identify and understand any issues that may arise during system operation.

3.2.1.5 Security

1. **Authentication and Authorization:** Only registered users (e.g., staff, coordinators, IT, admin) can log in. Role-based access control (RBAC) ensures users only access features relevant to their roles.

2. **Data Protection:** Sensitive data (e.g., login credentials, user info) must be encrypted in transit (using HTTPS) and at rest (e.g., hashed passwords in the database).

3. **Audit Logging:** All critical actions (e.g., adding items, approving requests, editing inventory) should be logged with timestamps and user information. Logs should be protected from tampering and regularly reviewed by administrators.

4. **Input Validation:** The system must validate all user inputs to prevent SQL injection, cross-site scripting (XSS), and other common attacks.

5. **Access Control:** All critical actions (item updates, request approvals, login attempts) should be logged for tracking and accountability.

6. **Backup and Recovery:** Regular backups of inventory data should be maintained to recover from system failure or data loss. Backup data should also be secured and accessible only to authorized personnel.

7. **Security Testing:** The system should allow for quick patching and updates to fix vulnerabilities as they are discovered. Only authorized developers or admins should be able to perform system maintenance or updates.

3.2.2 Organizational Requirements

3.2.2.1 Environmental Requirements

1. **Operating Environment:** The system should be compatible with a range of operating systems, including Windows, macOS, Linux, and Unix.

2. **Server and Hosting:** The application shall be deployed on the university's internal web server or a secure university-managed cloud environment that supports high availability and data integrity.

3. **Hardware Compatibility:** The system shall be optimized to run efficiently on typical university hardware configurations, requiring a minimum of 4 GB RAM, 2.0 GHz processor, and at least 1 GB of available storage for system components.

4. **Network Requirements:** The system shall function reliably within the university's internal network (LAN/Wi-Fi), maintaining stable performance over typical bandwidth speeds provided on campus.

5. **Database Environment:** The system shall integrate with a university-approved relational database system (e.g., MySQL) and support existing database access policies and backup protocols.

6. **Power and Stability:** The system shall remain stable and recoverable in the event of temporary power outages, supporting auto-restart and session recovery features where feasible.

7. **Physical Environment:** The system shall perform in secure office and lab conditions typical of university departments, with no special temperature, humidity requirements and appropriate measures in place to protect the hardware, software, and data from theft, tampering, or damage.

3.2.2.2 Operational Requirements

Our software is a web-based inventory management application designed to support university operations by tracking and managing assets across various departments. It enables efficient handling of items, requests, and updates in real time, ensuring transparency and accountability.

- The system must be deployed on the university's secure web server and accessible 24/7 via standard web browsers
- Daily automated backups of all inventory and user data will be performed at midnight
- System maintenance will occur once a week during low-usage hours (e.g., Sunday 12:00 AM to 1:00 AM)
- If the system encounters a crash, it will automatically restart and redirect users to the homepage. All unsaved data will be preserved where possible
- The university IT department will be responsible for basic technical maintenance, while the development team will handle updates and feature requests

3.2.2.3 Development Requirement

1. Technology Stack

- The frontend must be developed using HTML, CSS, and JavaScript to ensure a responsive and user-friendly interface
- The backend logic and APIs will be implemented using .NET framework (preferably ASP.NET Core for scalability and modern support)
- MySQL will serve as the relational database for storing all inventory records, user accounts, requests, and logs

2. Code Organization

- The application code should follow the WEB API architectural pattern for maintainability and separation of concerns
- The backend should follow a clean architecture or layered architecture pattern, separating concerns across different layers such as Controllers, Services, Repositories, and Models
- Controllers will handle HTTP requests and route them to the appropriate service methods

3. Security Practices

- Passwords must be hashed using secure algorithms (e.g., bcrypt or SHA-256) before being stored in the database
- Role-based access control (RBAC) must be implemented to restrict access to features based on user roles (Finance, IT, Staff, Coordinators, etc.)

4. Version Control

- All source code must be maintained in a Git repository (e.g., GitHub, GitLab)
- Development will follow a structured branching model (e.g., main, dev, feature branches)

5. **Deployment**

- The final system will be deployed to a university-hosted web server or cloud server (if applicable)
- Environment variables or config files should be used to manage sensitive data such as database credentials

3.2.3 External Requirements

3.2.3.1 Regulatory Requirement

The University Inventory System will adhere to relevant data protection and privacy policies, in alignment with institutional regulations and applicable national laws concerning the protection of personal information. In particular, the system will comply with the principles outlined in the Albanian Law No. 9887, dated 10.03.2008, "On the Protection of Personal Data", as well as supporting sub-legal acts.

Regulatory requirements also depend on:

1. Legal Compliance

- The system must adhere to national or international laws (e.g., GDPR)
- When handling user data, the system must comply with data protection regulations

2. Industry Standards

- The system must be compliant with specific standards to education, and finance

3. Data Protection and Privacy

- The system must protect sensitive or personal data and give users control over it

4. Copyright and Licensing

- Use of third-party libraries, fonts, images, or frameworks must respect licensing agreements

3.2.3.2 Ethical Requirement

The university inventory system will be designed and implemented with strong ethical considerations to ensure responsible and fair use by all stakeholders. The following principles will guide its development and usage:

1. **Data Privacy and Confidentiality:** The system will safeguard all user data, including login credentials, inventory records, and request logs. Access to sensitive data will be limited to authorized roles such as IT personnel or administrators, ensuring that staff and club coordinators can only view information relevant to their responsibilities.

2. **Transparency:** Users will be informed about how their data is stored, accessed, and used within the system. System actions such as item updates, approvals, and requests will be logged, allowing for transparency without violating user privacy.

3. **Accountability:** All user activities will be traceable through system logs, ensuring that any misuse or errors can be investigated and resolved. The development team and administrators will be responsible for maintaining a secure and reliable platform.

4. **Non-Discrimination and Fair Access:** The system will provide equal access to features for all designated user roles, without favoritism or bias. User permissions will be based strictly on institutional roles and predefined responsibilities.

5. Preventing Misuse: To maintain ethical standards, the system will include security and validation mechanisms to prevent unauthorized access, data tampering, or fraudulent requests. Any suspicious activity will be flagged and reviewed.

6. Professional Integrity: Developers working on the system will adhere to professional and academic standards, including respecting intellectual property rights, using open-source libraries responsibly, and maintaining clean, well-documented code.

Moreover, Sure, user data from Finance, IT, University Staff, and Club Coordinators will be used solely for verification, communication, and internal system operations. Consent is required to ensure proper tracking and accountability within the inventory system. The personal information that workers will share is f*ull name* and *email.*

3.2.3.3 Legislative Requirement

The system will fully comply with the laws of Albania and follow all relevant standards and regulations established by Albanian government authorities and law enforcement agencies.

Additionally, these regulations require users to consider the safety risks to users and to carry out a risk assessment to protect users from exposure to reasonably foreseeable risks. Those risks include work-related violence. A risk assessment is an examination to:

- Comply with data protection and privacy laws.

- Establish the significance of the risk.
- Identify and implement prevention and control measures.

3.2.3.3.1 Accounting Requirement

The personal information of all users in the university inventory system, including staff, IT personnel, finance, and club coordinators, will be protected in accordance with Albanian legislation. The system will comply with Law Number 9887 dated 10 March 2008, as amended by Law Number 48 of 2012, titled On the Protection of Personal Data. Access to sensitive data will be restricted only to authorized users as defined by institutional roles and responsibilities.

3.2.3.3.2 Security Requirement

This web application will also comply with the most recent updates of the General Data Protection Regulation (GDPR), which became enforceable on May 25, 2018, across European Union (EU) and European Economic Area (EEA) member states.

3.3 Domain Requirements

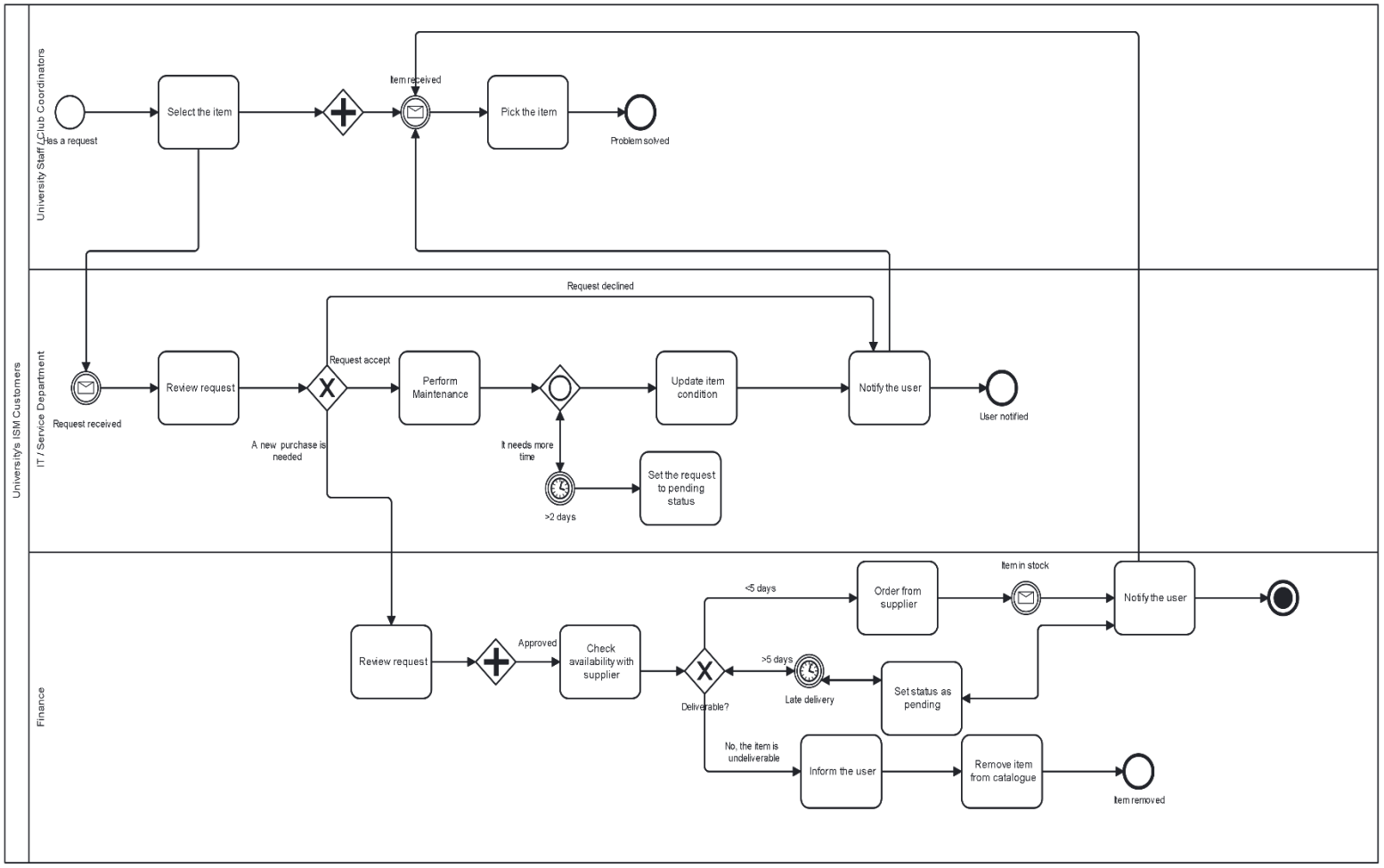
The university inventory system must follow internal policies and procedures, with role-based access for departments like IT, Finance, and Club Coordinators. Inventory items will be categorized and tagged for easy tracking, and all changes must follow approval workflows. The system should support regular audits, maintain a full history log, and generate reports to help departments manage assets. It will

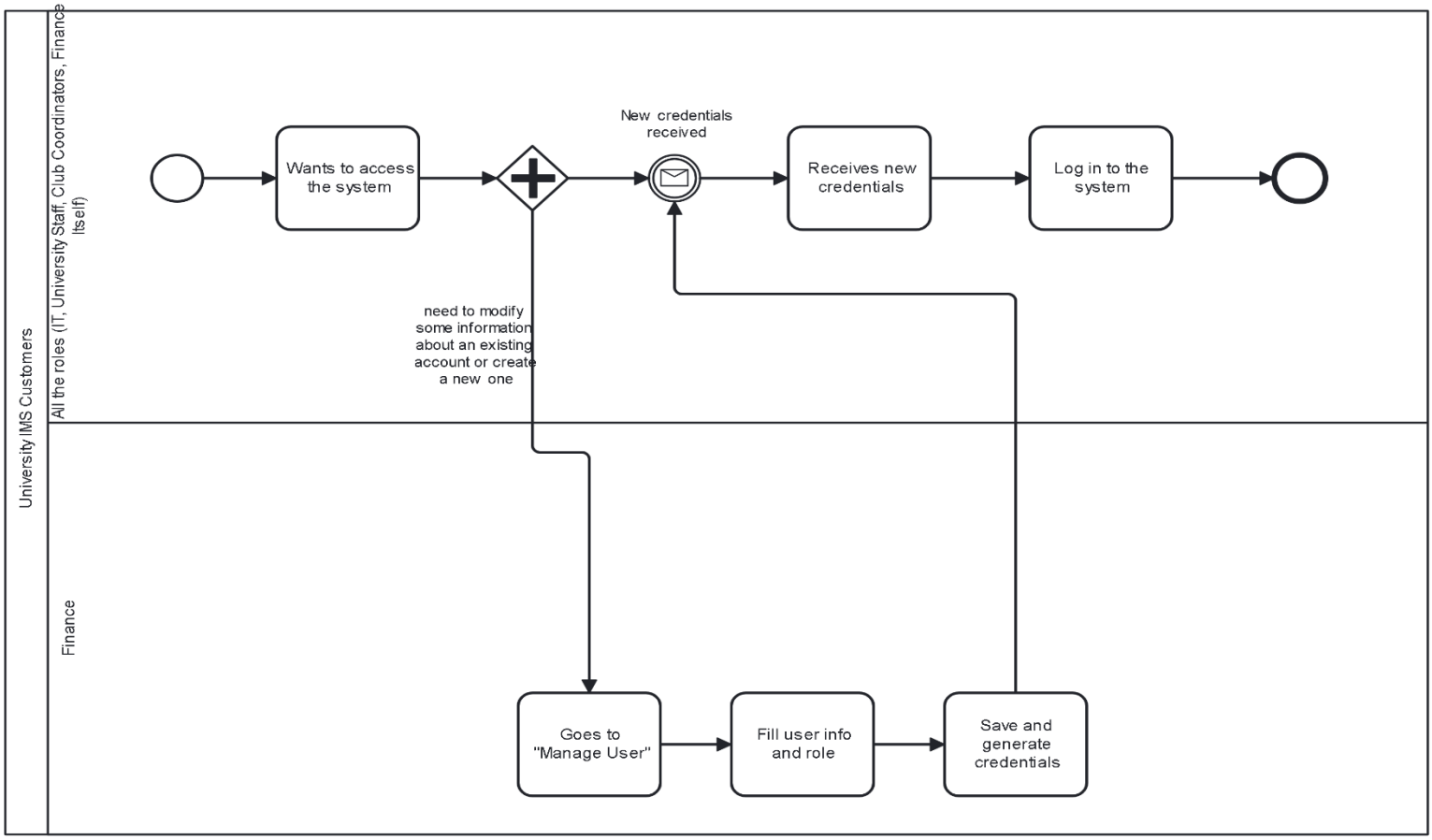
also comply with local standards, including Albanian language support and regional formats.

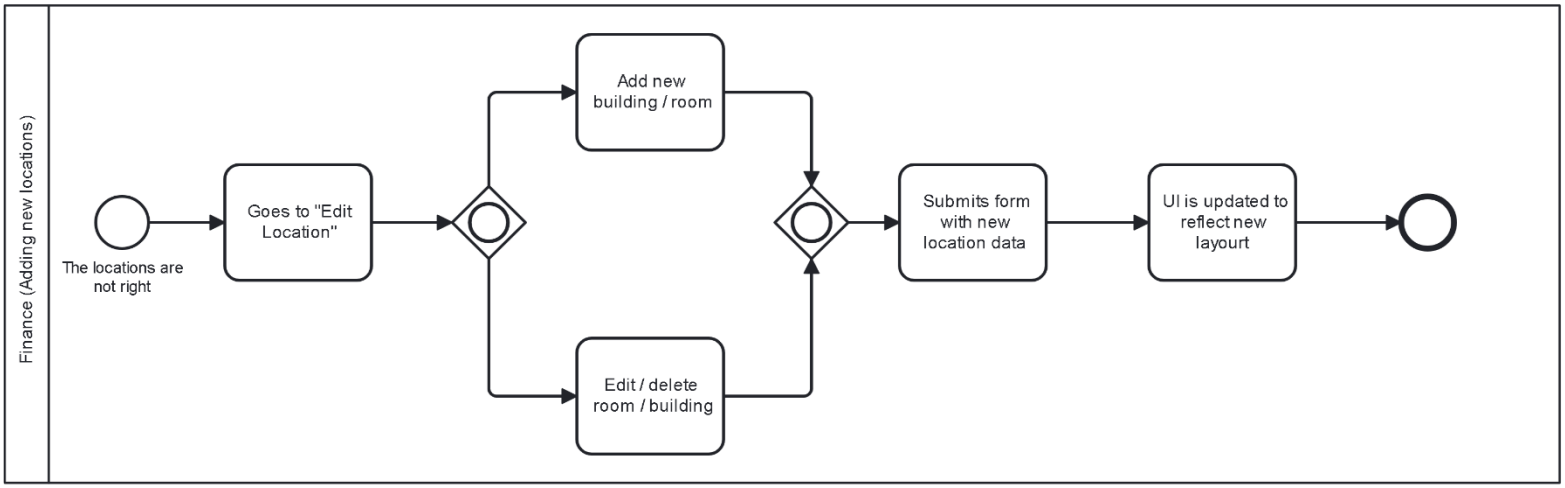
4. Domain Requirements

4.1 BPMN

We organized the BPMN diagrams by separating them into individual, well-defined workflows to avoid confusion and make each process easier to follow and understand.


University Staff / Club Coordinators
Has a request
Select the item
Item received
Pick the item
Problem solved
University's ISM Customers
IT / Service Department
Request declined
Request received
Review request
Request accept
A new purchase is needed
Perform Maintenance
It needs more time
>2 days
Set the request to pending status
Update item condition
Notify the user
User notified
Finance
Review request
Approved
Check availability with supplier
Deliverable?
<5 days
Order from supplier
Item in stock
Notify the user
>5 days
Late delivery
Set status as pending
No, the item is undeliverable
Inform the user
Remove item from catalogue
Item removed


University IMS Customers
All the roles (IT, University Staff, Club Coordinators, Finance Itself)
Wants to access the system
New credentials received
Receives new credentials
Log in to the system
need to modify some information about an existing account or create a new one
Finance
Goes to "Manage User"
Fill user info and role
Save and generate credentials


Finance (Adding new locations)
The locations are not right
Goes to "Edit Location"
Add new building / room
Edit / delete room / building
Submits form with new location data
UI is updated to reflect new layourt

4.2 Use Case Descriptions

Use case name:	Check Stock	
Scenario:	It , Service or Finance user browses current inventory by university building or club, drills down to specific rooms (classroom, office, storage), and inspects item details and condition.	
Triggering event:	Clicks on “Check Stock” from the main menu.	
Brief description:	The user views a high-level dashboard: cards representing each of the three main university buildings plus one card for “Clubs.” Upon selecting a building card, Finance can choose to view inventory by Room Type (Class, Office, or Storage). Within a room type, location names are displayed; clicking a location shows a listing of items—with their quantity and condition (New, Good, Barely Working, Not Working). Optionally, each item row can be displayed as a card with a photo.	
Actors:	Primary Actor: Finance Primary Actor: It Primary Actor: Service	
Related use cases:		
Stakeholders:	User(monitors overall inventory) IT & Service Departments (coordinate maintenance tasks when items are flagged as “Barely Working” or “Not Working”) University Staff (relies on accurate inventory records) Club Coordinators (their items appear under “Clubs”)	
Preconditions:	User is authenticated with role assigned as It or Service Department. Inventory records for all buildings, rooms, and clubs are already in the database.	
Postconditions:	User has viewed the up-to-date listing of all items, their quantities, and conditions for the selected building/room/club. No data is modified (read-only scenario).	
Flow of activities:	**Actor**	**System**
	• User clicks “Check Stock” in the horizontal navigation bar. • User clicks on, say, the “Building A” card. • User clicks the “Classrooms” sub-card.	• System retrieves the three building records plus the “Clubs” category and displays four cards (e.g., “Building A,” “Building B,” “Building C,” “Clubs”). • System shows three sub-cards or buttons: “Classrooms,” “Offices,” “Storage Rooms.”

	• User clicks on "Class 201." • User scrolls or searches within that table to inspect individual item details.	• System fetches and displays a list of classroom names in Building A (e.g., "Class 201," "Class 202," etc.), each with a brief description (e.g., "1 PC, 1 monitor, 20 tables"). • System displays a tabular listing (or optional card-style listing) of each item in "Class 201," showing columns: Item Name, Quantity, Condition, Photo Thumbnail (if enabled).
Exception conditions:	No Inventory Data: If no classrooms exist under that building—for example, if "Classrooms" is empty—system displays "No locations found. Please add new location under 'Edit Location.'" Database Error: If the inventory fetch fails, the system shows an error message ("Unable to retrieve stock data. Try again later.")	

Figure 1: Fully developed use case description for "*Checking Stock*"

Use case name:	Edit Location
Scenario:	Finance browses current inventory by university building or club, drills down to specific rooms (classroom, office, storage), and inspects item details and condition.
Triggering event:	Finance clicks "Edit Location" from the main menu
Brief description:	The Finance user sees the same UI layout as "Check Stock" (cards for buildings and clubs), but instead of showing item details, each location card includes an "Edit" icon at its top right. Finance can click an existing location's edit icon to modify its name/description or delete it, or click a "+" (Add) card to open a form to create a new building, classroom, office, or storage room. After submitting, updates are saved to the database and reflected in "Check Stock."
Actors:	Primary Actor: Finance (System Administrator)
Related use cases:	
Stakeholders:	Finance Department (manages location hierarchy) IT & Service Departments (need correct location mappings for their items) University Staff and Club Coordinators (items are tied to these locations)
Preconditions:	Finance is authenticated.

<table>
<tr><td></td><td colspan="2">Existing location hierarchy (buildings → room types → specific rooms) is present in the system.</td></tr>
<tr><td>Postconditions:</td><td colspan="2">If a new location is added, a new record is created in the database under the appropriate parent (e.g., new “Building D,” or new “Office 105” inside Building A).

If an existing location is edited, its name/description fields are updated.

If an existing location is deleted, the record (and any child sub-records) are removed or flagged for archival (depending on implementation).

The “Check Stock” view now reflects the new/modified hierarchy.</td></tr>
<tr><td rowspan="2">Flow of activities:</td><td>Actor</td><td>System</td></tr>
<tr><td>• Finance clicks “Edit Location” in the navigation bar.

• Finance clicks the “+ Add Location” card.

• Finance fills out required fields (e.g., “Building D,” description “New Engineering Building”) and clicks “Save.”

• Finance clicks the “Edit” icon on “Class 201.”

• Finance modifies fields (e.g., rename to “Class 201A”) and clicks “Save.”

• Finance clicks the “Delete” icon on “Office 305.”</td><td>• System displays cards for all buildings and “Clubs,” each with an “Edit” icon, plus an extra “+ Add Location” card.
• System opens a modal or separate page containing a form with fields:

Location Type (Building / Classroom / Office / Storage Room)

Parent Location (if applicable)

Location Name

Description

Photo (optional)

Save and Cancel buttons
• System validates input (e.g., non-empty name).
• If validation passes, system creates a new location record with:

Type = Building

Name = “Building D”

Description = [provided]

And persists it to the database.
• System returns to “Edit Location” view and shows the updated set of cards (including the new building card).</td></tr>
</table>

		• System loads the "Edit Location" form populated with the current data for "Class 201" (e.g., location type = Classroom, parent = Building A, name = "Class 201," description). • System validates the new data. If valid, updates the "Class 201" record in the database to reflect the new name. • System returns to "Edit Location" view, reflecting the updated name. • System prompts: "Are you sure you want to delete Office 305 and all associated inventory records?" If Finance clicks "Confirm," go to next step. If "Cancel," abort deletion. • System deletes (or flags as inactive) the "Office 305" record and cascades or flags any children (if they exist), then refreshes the view.
Exception conditions:	Validation Failure on Add/Edit: If "Location Name" is blank or duplicates an existing location at the same level, the system displays "Invalid name—please ensure it's nonempty and unique." Database Save Error: If saving to the database fails (e.g., connectivity issue), the system shows "Unable to save changes. Try again later." Changes are not applied. Delete with Dependent Inventory: If Finance tries to delete a location that still has items assigned, the system warns "Cannot delete location—items still assigned. Please move or delete items first."	

Figure 2: Fully developed use case description for "*Editing Location*"

Use case name:	Manage Users	
Scenario:	Finance (as System Admin) views all user accounts, filters by role, adds new users, edits or deletes existing users, and assigns roles.	
Triggering event:	Finance clicks “Manage Users” in the main menu.	
Brief description:	The Finance user accesses a paginated list of all system users (IT, Service, University Staff, Club Coordinators, and other Finance users). Finance can filter by role, search by name or email, click “Edit” to change a user’s role (or reset password), click “Delete” to remove a user, or click “Add New User” to open a form that captures Name, Email, Password, and Role. Upon submission, the system encrypts the password and creates (or updates) the user record.	
Actors:	Primary Actor: Finance (System Administrator)	
Related use cases:		
Stakeholders:	All Departments (user accounts must be accurate so that requests and permissions flow correctly) IT Department (may rely on Finance to create their accounts) University Staff, Club Coordinators (their access depends on correct roles)	
Preconditions:	Finance is authenticated with “Manage Users” permissions. The underlying User table exists, with fields: Name, Email (unique), Encrypted Password, Role.	
Postconditions:	If adding a user: A new user record is created, password is stored as an encrypted hash, and an activation email is sent. If editing a user: That user’s role (or password) is updated, and the change is logged. If deleting a user: That user’s record is removed or flagged as inactive; any open requests or assignments are re-assigned or flagged.	
Flow of activities:	**Actor**	**System**
	• Finance clicks “Manage Users” from the navigation bar. • Finance optionally selects a role filter (e.g., “IT”). • Finance clicks “Add New User.” • Finance fills in the user’s details, selects Role = “Service,” and clicks “Create.”	• System fetches and displays a table of all users, showing columns: Name, Email, Role, Status (Active/Inactive), and action icons (Edit/Delete). • System applies filter, re-queries the Users table for Role = “IT,” and updates the table display. • System opens a modal or page with a form: Name Email

	• Finance clicks the “Edit” icon next to user “jane.doe@example.edu.” • Finance changes “Role” from “Club Coordinator” to “University Staff” and clicks “Save.” • Finance clicks the “Delete” icon next to user “john.smith@it.univ.edu.”	Password Confirm Password Role (dropdown) Buttons: “Create” and “Cancel” • System validates all fields (e.g., unique email, password match). • If validation passes: Encrypts the password Creates new user record in the database Sends activation/notification email to new user Refreshes the user list to include the newly added user • If validation fails (e.g., email already exists): System displays an appropriate error • System loads that user’s existing data into an editable form (Name, Email read-only or editable, Role dropdown, optional “Reset Password”). • System updates the user’s Role in the database, logs the change, and refreshes the user list. • System prompts: “Are you sure you want to delete john.smith@it.univ.edu? This action cannot be undone.” If Finance confirms, proceed to next step If cancels, abort deletion • System either removes or flags that user as Inactive in the database; any open assignments or requests are reassigned or flagged. Then refresh the list.

Exception conditions:	Duplicate Email on Create/Edit: If Finance enters an email already in use, the system rejects and displays "Email already exists. Choose a different email." Weak Password: If password does not meet complexity rules, the system shows "Password must be at least 8 characters, containing uppercase, lowercase, and a digit." Deletion with Active Dependencies: If the user being deleted has pending requests or open approvals, the system warns "User has pending requests—cannot delete until reassigned."

Figure 3: Fully developed use case description for "*Managing Users*"

Use case name:	Process Requests (IT/Service)
Scenario:	An IT or Service Department user reviews requests (either stock requests or quality-assurance requests) made by University Staff or Club Coordinators, and chooses to Accept, Forward (to Finance), or Decline each.
Triggering event:	A new request (stock or QA) arrives from University Staff or Club Coordinators.
Brief description:	The IT (or Service) user logs in, sees a list of pending requests (displayed on their Homepage as request cards). For each request, the user can "Accept" (meaning the department will fulfill the request), "Forward" (escalate to Finance for budget approval), or "Decline." Once an action is chosen, the system updates the request status accordingly and notifies the original requester.
Actors:	Primary Actor: IT Department User or Service Department User (depends on which department the request applies to)
Related use cases:	
Stakeholders:	IT Department (responds to IT-related QA/stock requests) Service Department (responds to Service-related QA/stock requests) University Staff and Club Coordinators (requesters) Finance Department (if forwarded)
Preconditions:	The IT/Service user is authenticated. There is at least one pending request (status = "Pending") targeted at IT or Service.
Postconditions:	The request status is updated to "Accepted," "Forwarded," or "Declined." If "Accepted":

<table>
<tr><td></td><td colspan="2">The system decrements relevant in-department inventory (or flags item for repair in case of QA).

A notification is sent to the requester: “Your request has been accepted by IT (or Service).”

If “Forwarded”:

The request is re-assigned to Finance with status = “Pending Finance Approval.”

A notification is sent to Finance for review.

If “Declined”:

The request is closed; requester notified of decline.</td></tr>
<tr><td>Flow of activities:</td><td>Actor</td><td>System</td></tr>
<tr><td></td><td>• IT/Service user clicks “Homepage” in the navigation.

• IT/Service user clicks on a request card submitted by “Prof. X.”

• IT/Service user clicks “Accept.”

• OR IT/Service user clicks “Forward.”

• OR IT/Service user clicks “Decline.”</td><td>• System retrieves all requests where (Department = “IT” or “Service”) AND status = “Pending,” displaying each as a card with summary details.
• System displays full request details (what is being requested, quantity, justification, location, date) and shows three buttons: Accept, Forward, Decline.
• If “Accept” is clicked:

System updates request.status = “Accepted by IT” (or “Accepted by Service”)

If stock request: system locates relevant inventory items and reduces available quantity (or flags for reorder if insufficient)

If QA request: system flags the item for repair and notifies the maintenance queue

System sends notification to the requester: “Your request has been accepted by [Department].”
• If “Forward” is clicked:</td></tr>
</table>

		System updates request.status = "Forwarded to Finance" Creates a new pending record in Finance's queue Sends notification to Finance user(s) Notifies original requester: "Your request has been forwarded to Finance for approval." • If "Decline" is clicked: System updates request.status = "Declined by [Department]" Logs reason if entered Sends notification to the requester
Exception conditions:	Insufficient Inventory (upon Accept): If department inventory cannot fulfill the requested quantity, the system: Displays an alert: "Insufficient stock to fulfill request. Consider forwarding to Finance or declining." Does not let the user confirm "Accept" until they choose to reorder via Finance. Concurrent Processing Conflict: If another IT/Service user already processed the request, the system warns "Request already handled—status is [Accepted/Declined/Forwarded]." Notification Error: If the email server is down, the system logs the failure and queues the notification for retry, showing a message "Unable to send notification now—will retry."	

Figure 4: Fully developed use case description for "*Processing Requests*"

Use case name:	Process Stock Request (For Finance)	
Scenario:	Finance reviews stock requests submitted by IT and Service Departments and decides whether to approve, pend, or decline each request.	
Triggering event:	A new stock request is submitted by an IT or Service-Department user.	
Brief description:	The Finance user (acting as system administrator) logs into the system, views incoming stock requests, and for each request chooses one of three actions—Approve, Pending, or Decline. Once an action is chosen, the system updates the request's status, adjusts any related inventory or budget records if approved, and sends an automated notification (email or in-system message) back to the requester.	
Actors:	Primary Actor: Finance (System Administrator)	
Related use cases:		
Stakeholders:	Finance Department (needs to track and authorize purchases) IT Department (requests new stock) Service Department (requests new stock) University Accounting (needs accurate budget and inventory records)	
Preconditions:	The Finance user is authenticated and has "System Admin" privileges. There is at least one pending stock request submitted by either IT or Service. Inventory catalog and budget information are up to date.	
Postconditions:	The selected request's status is updated to Approved, Pending, or Declined. If Approved: The requester's department's "pending procurement" record is updated. The inventory re-ordering workflow is triggered. If Pending: The request remains open and flagged for follow-up (e.g., awaiting funds or supplier availability). If Declined: The request is closed without procurement. An automatic notification is sent to the original requester indicating the new status (Approved, Pending, or Declined).	
Flow of activities:	**Actor**	**System**

	• Finance navigates to the "Homepage" page where pending requests are listed. • Finance clicks on a specific request card. • Finance reviews the details and clicks "Approve," "Pending," or "Decline." • (If "Approve") Finance confirms procurement details (e.g., budget code).	• System retrieves all stock-request records with status = "Pending" and displays them as cards (one per request), showing requester name, department, item details, and justification. • System opens a detailed view of the request, showing full item list, quantities, reason, date submitted, and requester contact. • System updates the request record in the database with the chosen status. • System: Marks request as "Approved." Automatically deducts estimated cost from Finance's available budget. Flags items for purchase order generation. | | 5. (If "Pending") Finance enters a comment or note (e.g., "Awaiting supplier quote"). | 5.1 System saves the "Pending" status and the comment, leaving the request in pending state for later action. | | 6. (If "Decline") Finance may optionally enter a reason for decline. | 6.1 System sets status = "Declined" and records the reason. | | 7. Finance clicks "Send Notification." | 7.1 System sends an automated email (and/or in-system notification) to the original requester with their new request status and any comments. |
Exception conditions:	Invalid Request ID: If the request ID is missing or malformed, the system displays an error: "Request not found." Finance must choose another request to process. Insufficient Budget (upon Approve): If the Finance user's budget is insufficient to cover the total, the system:	

	Rejects the "Approve" action, displays an error message ("Insufficient funds—please adjust amount or choose Pending") and suggests marking the request as "Pending." Notification Failure: If the email server is unreachable, the system queues the notification; Finance is shown a warning ("Notification pending—will retry automatically"). Concurrent Update Conflict: If another admin already processed the request moments ago, the system informs Finance that the request's status has changed; Finance must refresh the list.

Figure 5: Fully developed use case description for "*Processing stock request*"

Use case name:	Submit QA Request
Scenario:	University Staff or Club Coordinator identifies malfunctioning or substandard equipment in their assigned locations and submit a QA request to the IT or Service Department.
Triggering event:	Clicks the "QA Request" icon (from their homepage or the combined page).
Brief description:	A University Staff or Club Coordinator user chooses "QA Request," selects the location where the QA is needed (building, room), specifies the item(s) or equipment requiring QA, enters a justification, and clicks "Submit." The system creates a new QA request record addressed to the appropriate department (either IT or Service, based on the item category) with status = "Pending." The request appears in the IT/Service queue, and the requester is notified.
Actors:	Primary Actor: University Staff and Club Coordinator
Related use cases:	
Stakeholders:	University Staff /Club Coordinator(needs malfunctioning equipment fixed) IT/Service Departments (will inspect/repair the item) Finance Department (if a purchase or budget is needed post-QA)
Preconditions:	University Staff/Club Coordinator is authenticated. The staff has assigned locations (e.g., an office or classroom) where they can identify items to report. The system's inventory is up to date.
Postconditions:	A new QA request record is created with fields: Requester ID, Location (e.g., "Office 212"), Item ID (or description), Justification, Target Department ("IT" or "Service"), Status = "Pending," Timestamp. The request appears in the IT/Service queue. An automatic notification is sent to the IT/Service department user(s).

Flow of activities:	Actor	System
	• Actor clicks “QA Request” on their homepage.	• System displays a “QA Request” form with fields: Location (dropdown populated with that staff’s assigned locations, e.g., Office 212) Item (dropdown or text; if a dropdown, lists all items in that location) Justification/Description (text area) Target Department (automatically set based on item category but editable) Submit/Cancel buttons. • Staff selects “Office 212,” chooses “Projector Model X,” enters “No power” as justification, and clicks “Submit.” | 2.1 System validates all required fields. - If validation passes: Creates a QA Request record with status = “Pending,” linking to the item, location, requester, and timestamp. Notifies the designated department user(s) (IT or Service) via email/in-system. Shows confirmation: “QA Request submitted successfully. You will be notified when the department responds.” - If validation fails, system shows errors (e.g., “Please select an item”). • Staff logs out or returns to Homepage. | 3.1 System retains QA request for action by IT/Service. |
Exception conditions:	Validation Failure: If Staff tries to submit without selecting an item or location, system displays “All fields are required.”	

	Invalid Item: If Staff types a free-text item not found in inventory, system prompts "Item not recognized—please choose from list or contact Service." Notification Failure: If the department notification fails, system queues it and shows "Request saved; notification will be sent when the system is available."

Figure 6: Fully developed use case description for "*Submitting QA Request*"

Use case name:	Submit Stock Request	
Scenario:	Users who need new items (IT, Service, University Staff, or Club Coordinator) fill out and submit a stock request form.	
Triggering event:	A user clicks "Stock Request" from their homepage or from the combined QA/Stock page.	
Brief description:	The actor (role varies) opens a form to specify the item(s) they require, quantity, justification, and which department should receive the request (IT or Service). After completing the form and clicking "Submit," the system creates a new request record, assigns it the status "Pending," and routes it to the appropriate department's queue. An email/in-system notification is sent to that department.	
Actors:	Primary Actor: IT Department User or Service Department User (depends on which department the request applies to)	
Related use cases:		
Stakeholders:	IT Department (processes or forwards the request) Service Department (same) Finance Department (if forwarded for budget approval) Requesting actor's department (needs items)	
Preconditions:	The user is authenticated (any of the four roles). At least one valid "Department" option is available (IT or Service).	
Postconditions:	A new stock request record is created in the database with fields: Requester ID, Department (target), Item Description, Quantity, Justification, Date Submitted, and Status = "Pending." The request appears on the target department's Homepage under pending requests. A notification (email or in-system) is sent to the target department user(s).	
Flow of activities:	**Actor**	**System**
	• Actor clicks "Stock Request" (or, for University Staff, selects the stock icon on the combined page).	• System displays a blank "Stock Request" form with fields: Item Name / Description (text)

		Quantity (numeric) Justification (text area) Target Department (dropdown: “IT” or “Service”) Submit and Cancel buttons. | • Actor enters “Wireless Keyboard,” Quantity = 10, Justification = “Lab upgrades,” selects Department = “IT,” and clicks “Submit.” | System validates that all mandatory fields are filled and quantity is > 0. If validation passes: System creates a new Request record in the database with status = “Pending,” storing all fields plus timestamp and ActorID. System sends a notification to the IT department queue and to any designated “IT Approver” role. System displays a confirmation message: “Request submitted successfully. You will be notified when the status changes.” - If validation fails: displays appropriate error (e.g., “Quantity must be at least 1”). | • Actor logs out or navigates away. | System retains the request for later processing by IT/Service. |
Exception conditions:	Validation Error: If “Quantity” is missing or ≤ 0, system shows “Quantity must be a positive number.” If “Item Name” is blank, “Please describe the item.” System Error on Save: If database insert fails, system displays “Unable to submit request—please try again.”	

	Notification Failure: If email/in-system notification to the target department is down, system queues the notification and shows "Request saved, but notification will retry shortly."

Figure 7: Fully developed use case description for "*Submitting Stock Request*"

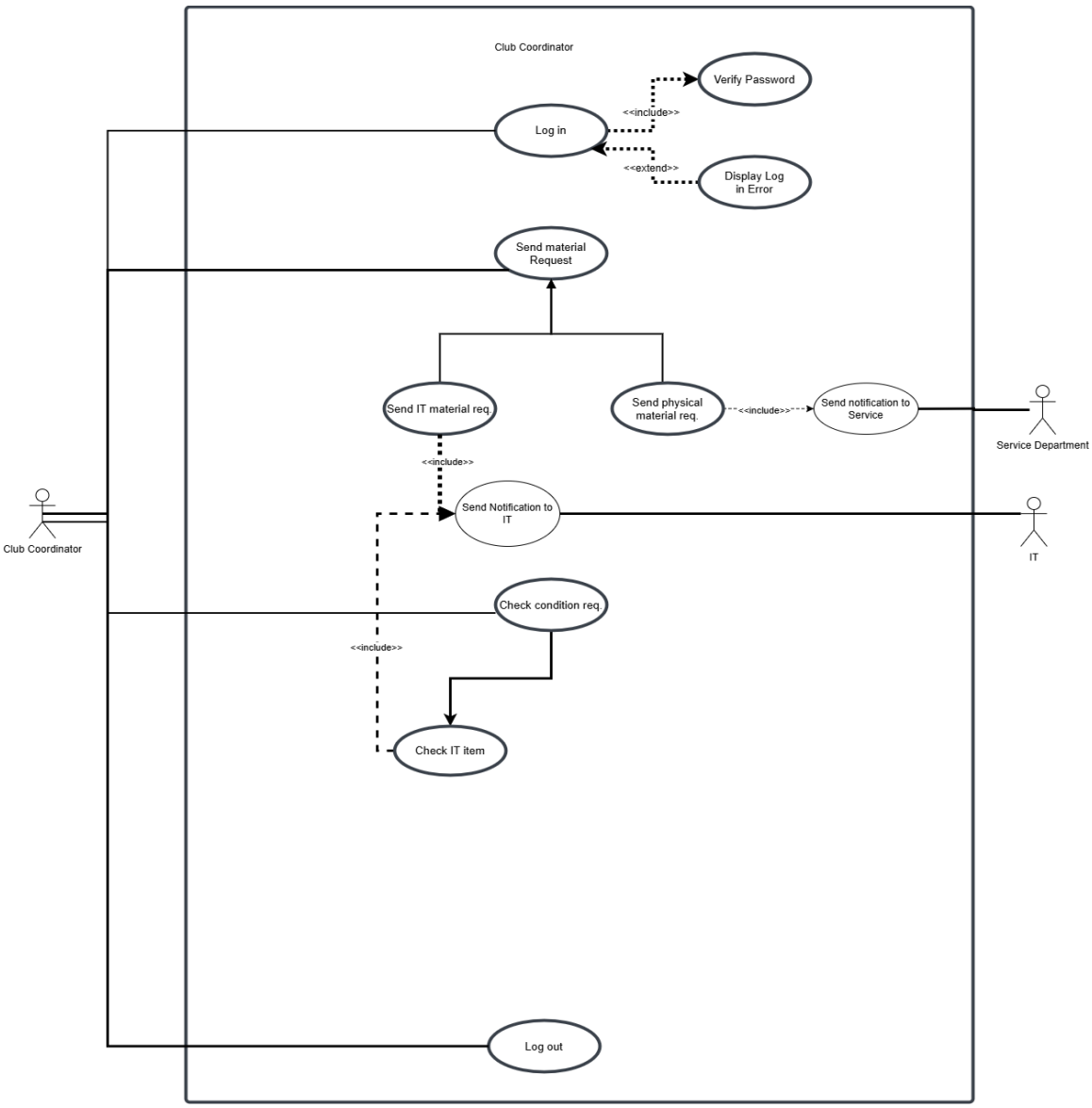
Use case name:	View and Update Item State
Scenario:	A University Staff or Club Coordinator browses their assigned location(s), sees a list of items, and updates the condition/status of any item (e.g., marking it "Barely Working" or "Not Working"). If an item's updated state equals "Not Working," the system automatically generates a new QA request. Triggering Event: The user (University Staff or Club Coordinator) logs in and clicks their "Homepage," then selects their assigned building/location.
Triggering event:	The user (University Staff or Club Coordinator) logs in and clicks their "Homepage," then selects their assigned building/location/Item.
Brief description:	On their personalized homepage, University Staff either sees a direct listing of items for their assigned office, or they first select their building, then room. Club Coordinators see their club's inventory. In either case, a table or card view shows each item's name, quantity, and current state. For each item, the user can click an "Edit" or "Update State" icon, select a new state, and click "Save." If the new state is "Not Working," the system auto-creates a QA Request to the appropriate department. Otherwise, the system simply updates the record.
Actors:	Primary Actor: University Staff and Club Coordinator
Related use cases:	QA Request
Stakeholders:	University Staff (ensures equipment in their office/class is accurately flagged) Club Coordinators (manage club assets) IT/Service Departments (handle QA requests when items are broken) Finance Department (needs accurate condition data for budgeting/replacement)
Preconditions:	User (University Staff or Club Coordinator) is authenticated. The system knows the user's assigned location(s) (e.g., Staff A is assigned to Office 212; Club B is assigned "Club Room 5"). Inventory records for those locations exist.
Postconditions:	The selected item's "Condition" field is updated in the database. If condition = "Not Working," a QA Request is automatically generated (in back end) assigned to the correct department (IT or Service).

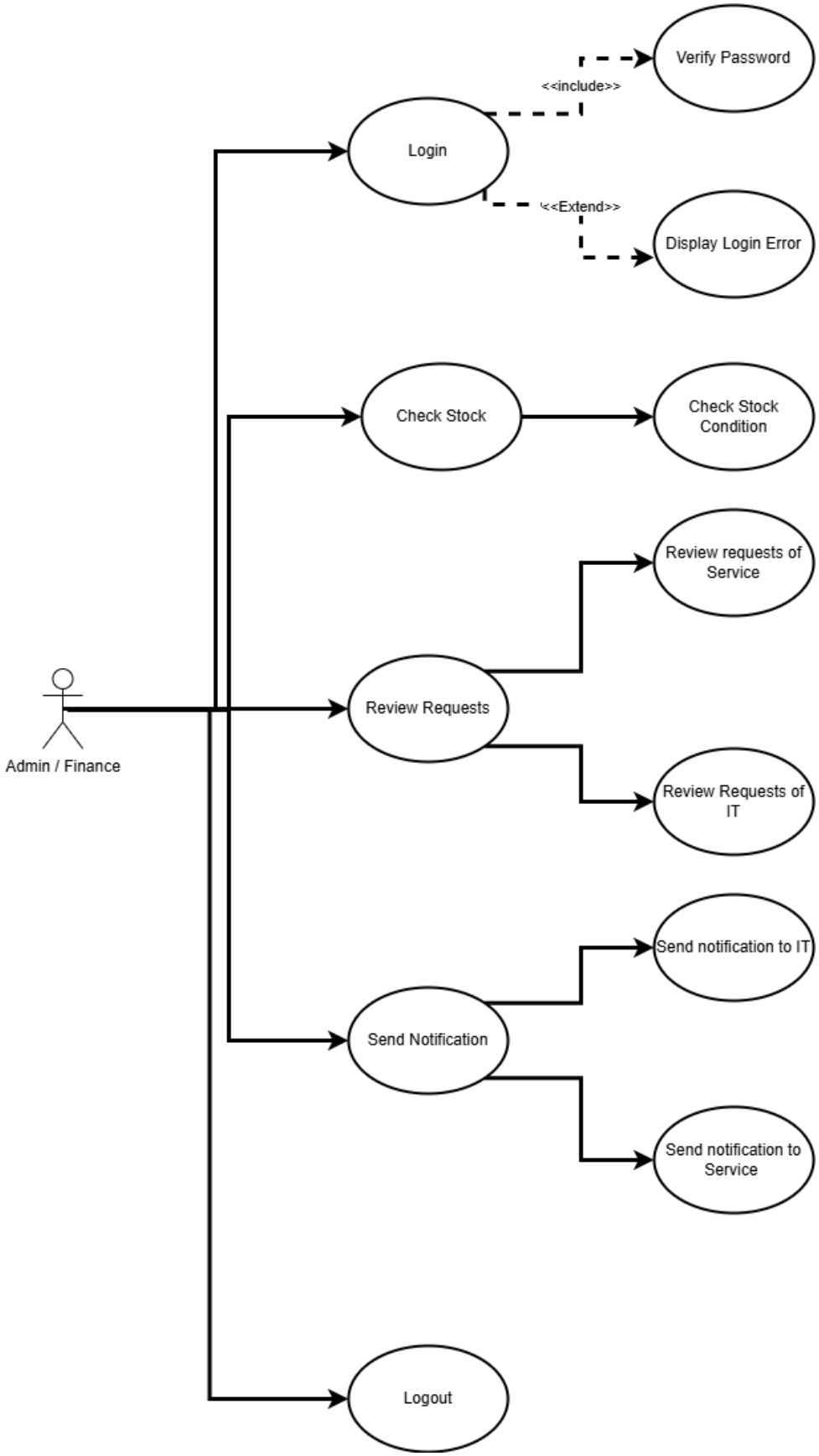
	A notification ("You flagged [Item] as 'Not Working.' A QA request has been created.") appears to confirm.	
Flow of activities:	**Actor**	**System**
	• University Staff/Club Coordinator clicks "Homepage." • Actor selects "Office 212" (if needed). • Actor clicks the "Update State" button for "Desk PC #3." • Actor changes state to "Not Working" and clicks "Save."	• System determines the actor's role and assigned location(s). For Staff: if multiple, system prompts to select (e.g., Building A → Office 212) For Club Coordinator: goes directly to "Club Room 5" • System retrieves all inventory items where Location = "Office 212," and displays them in a table with columns: Item Name Quantity Current State "Update State" action button • System opens an inline editor or modal showing current state (e.g., "Good Condition") and a dropdown to change to: {New, Good, Barely Working, Not Working} • System updates the "Condition" field for "Desk PC #3" in the inventory table. • System recognizes new state = "Not Working," so: Automatically creates a QA Request record with: Requester = current user Item ID = "Desk PC #3" Location = "Office 212" Department = "IT" (since it's a computer)

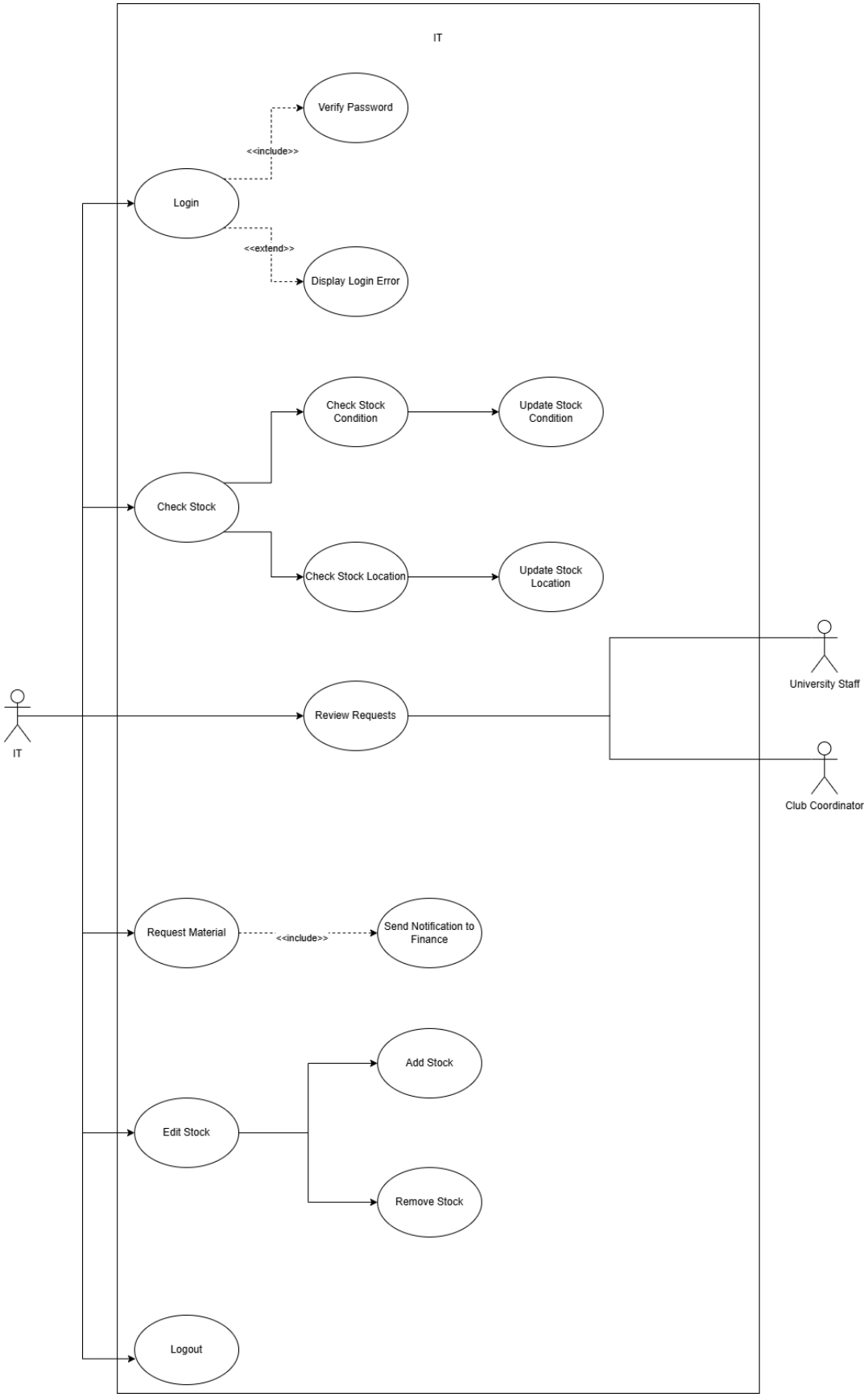
		Justification = “Marked not working by staff” Status = “Pending” Sends notification to IT queue • System updates the inventory view to reflect “Not Working.” • System displays confirmation: “Item state updated to ‘Not Working.’ QA request #12345 has been generated.”
Exception conditions:	Concurrent Update: If another user just updated the same item’s state, the system warns “This item’s condition was just changed by someone else—please refresh.” Invalid State Transition: If user tries to change from “Not Working” directly back to “New,” system allows it but logs a comment (“Possible data inconsistency—please confirm inventory physically”). Automatic QA Failure: If the automatic QA Request creation fails (e.g., database queue is down), system still updates the item state but shows “Item state saved, but unable to generate QA request. Please contact IT manually.”	

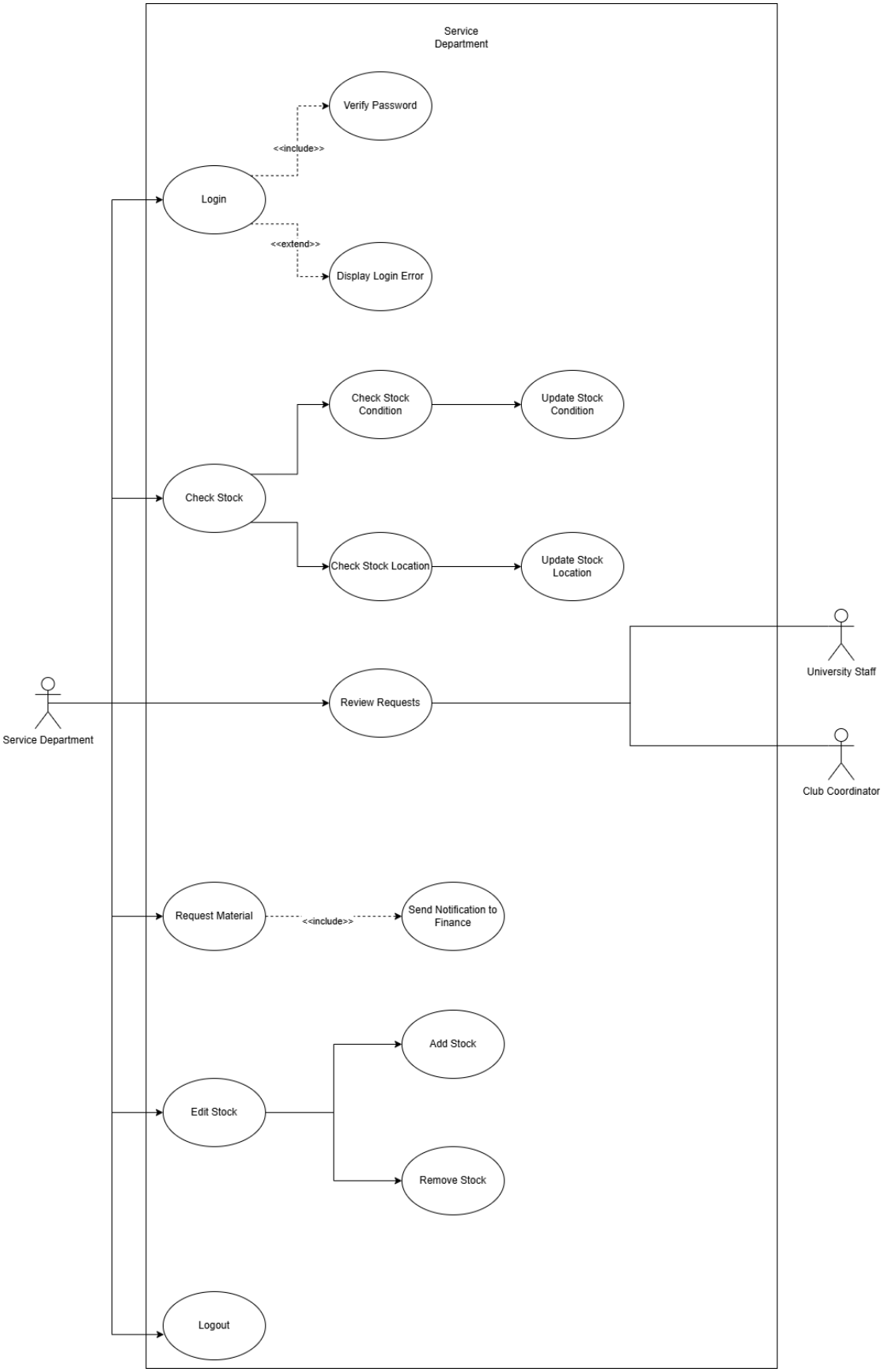
Figure 8: Fully developed use case description for “*Viewing and updating Item State*”

4.3 Use Case Diagrams

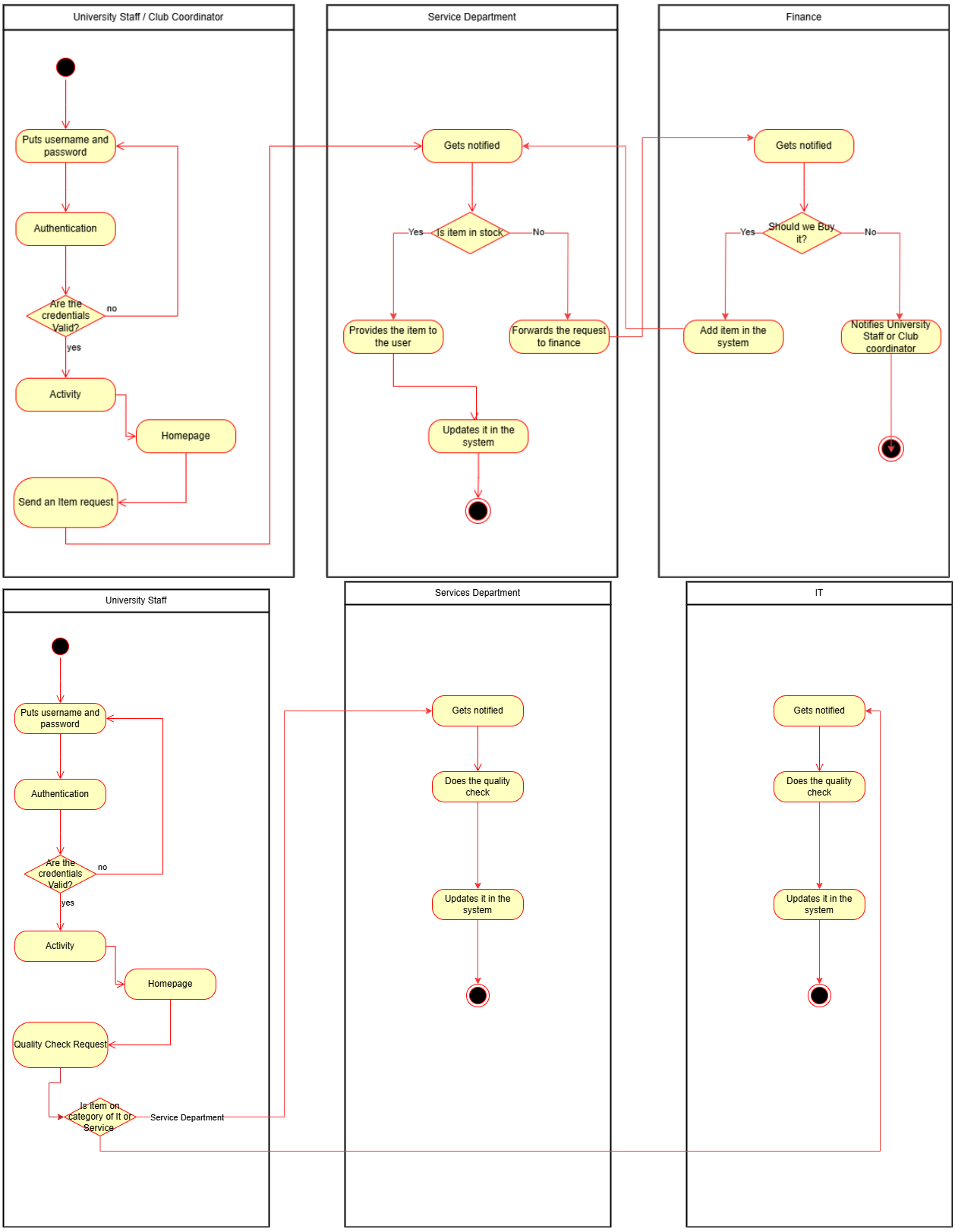


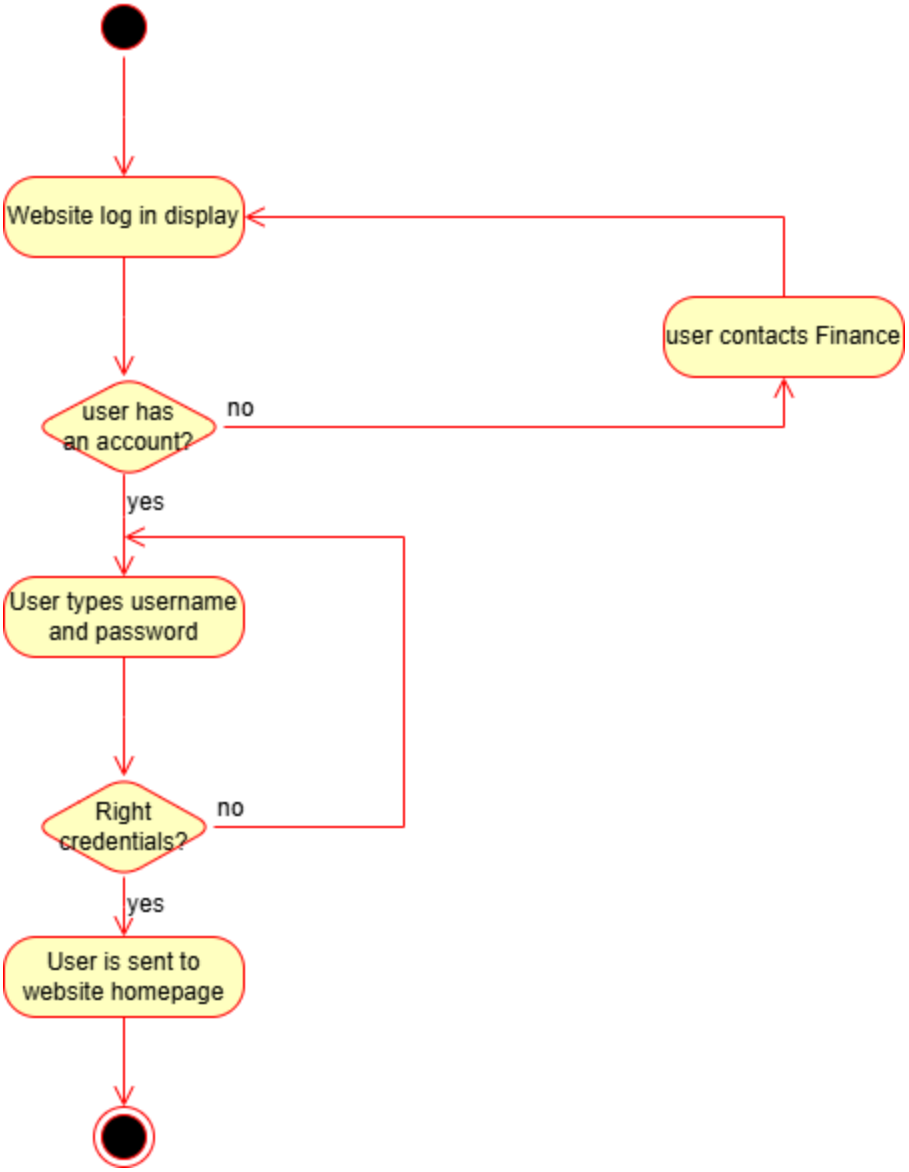

Verify Password
<<include>>
Login
<<Extend>>
Display Login Error
Check Stock
Check Stock Condition
Review requests of Service
Review Requests
Review Requests of IT
Send notification to IT
Send Notification
Send notification to Service
Logout
Admin / Finance

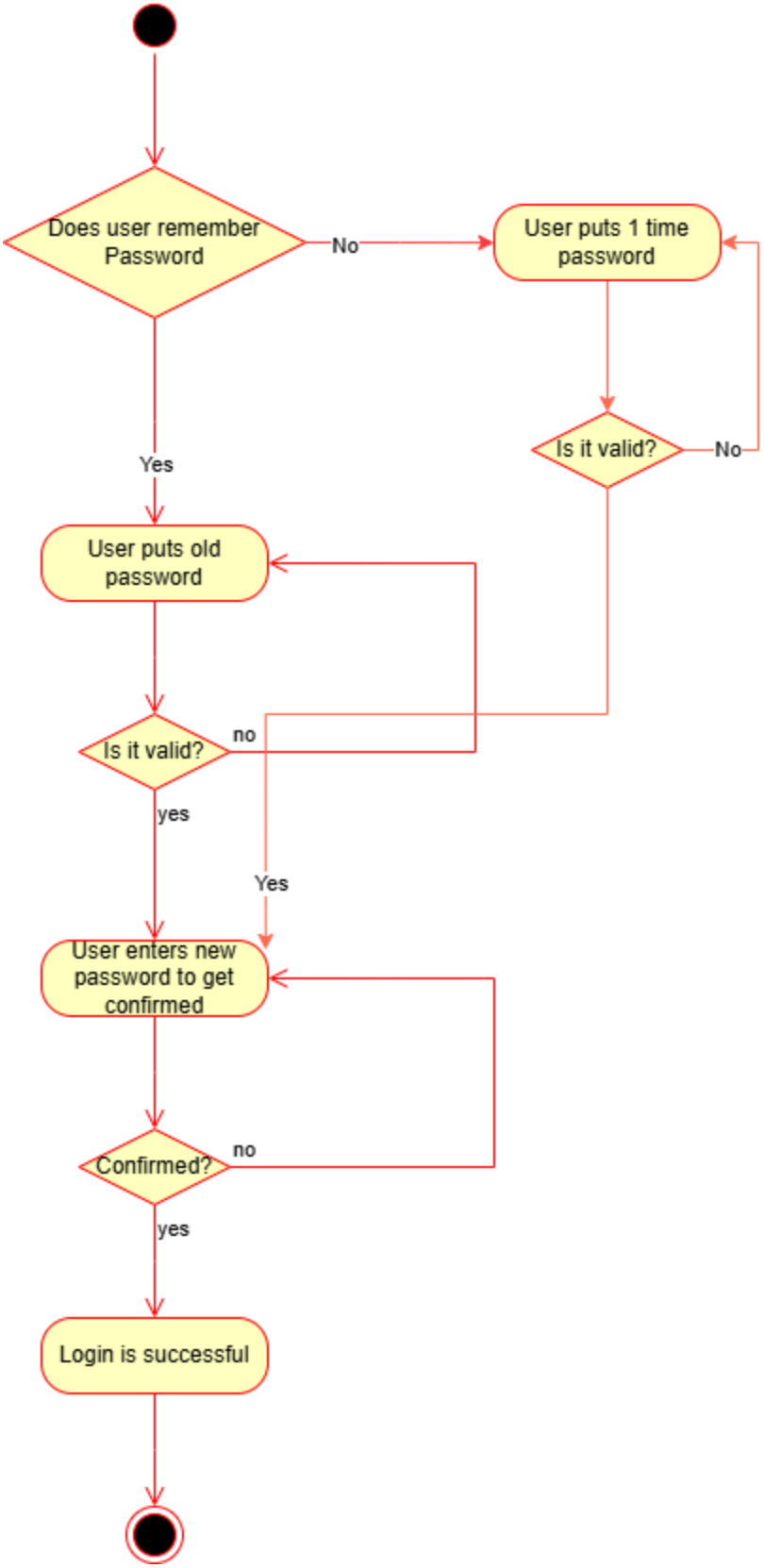

IT
Verify Password
<<include>>
Login
<<extend>>
Display Login Error
Check Stock Condition
Update Stock Condition
Check Stock
Check Stock Location
Update Stock Location
University Staff
IT
Review Requests
Club Coordinator
Request Material
<<include>>
Send Notification to Finance
Add Stock
Edit Stock
Remove Stock
Logout

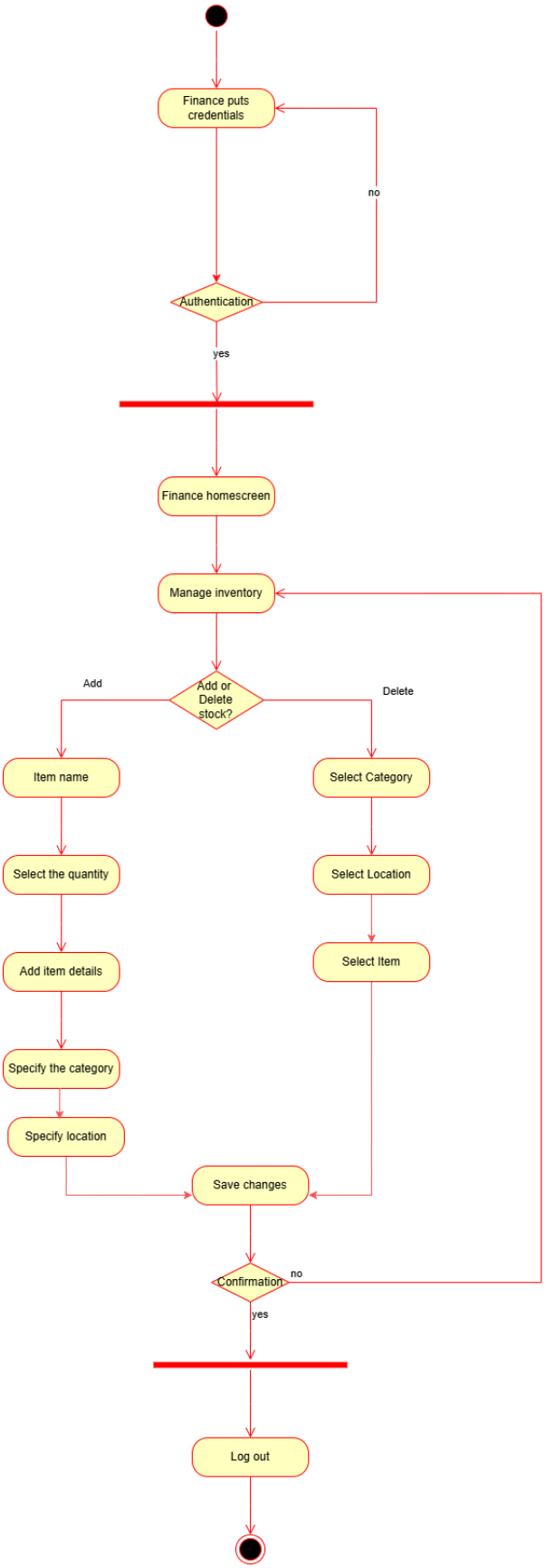

Service Department
Verify Password
<<include>>
Login
<<extend>>
Display Login Error
Check Stock Condition
Update Stock Condition
Check Stock
Check Stock Location
Update Stock Location
University Staff
Service Department
Review Requests
Club Coordinator
Request Material
<<include>>
Send Notification to Finance
Add Stock
Edit Stock
Remove Stock
Logout

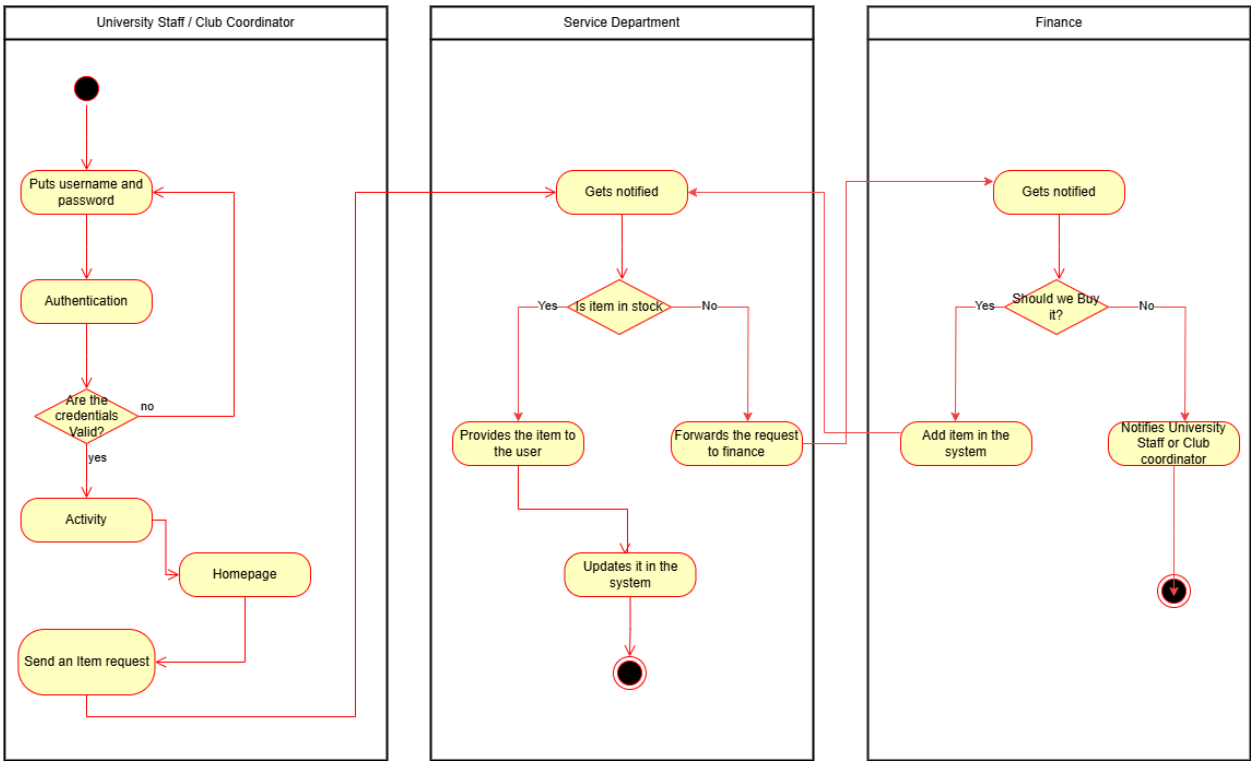
4.4 Activity Diagrams:


University Staff / Club Coordinator
Puts username and password
Authentication
Are the credentials Valid?
no
yes
Activity
Homepage
Send an Item request
Service Department
Gets notified
Is item in stock
Yes
No
Provides the item to the user
Forwards the request to finance
Updates it in the system
Finance
Gets notified
Should we Buy it?
Yes
No
Add item in the system
Notifies University Staff or Club coordinator
University Staff
Puts username and password
Authentication
Are the credentials Valid?
no
yes
Activity
Homepage
Quality Check Request
Is item on category of It or Service
Service Department
Services Department
Gets notified
Does the quality check
Updates it in the system
IT
Gets notified
Does the quality check
Updates it in the system

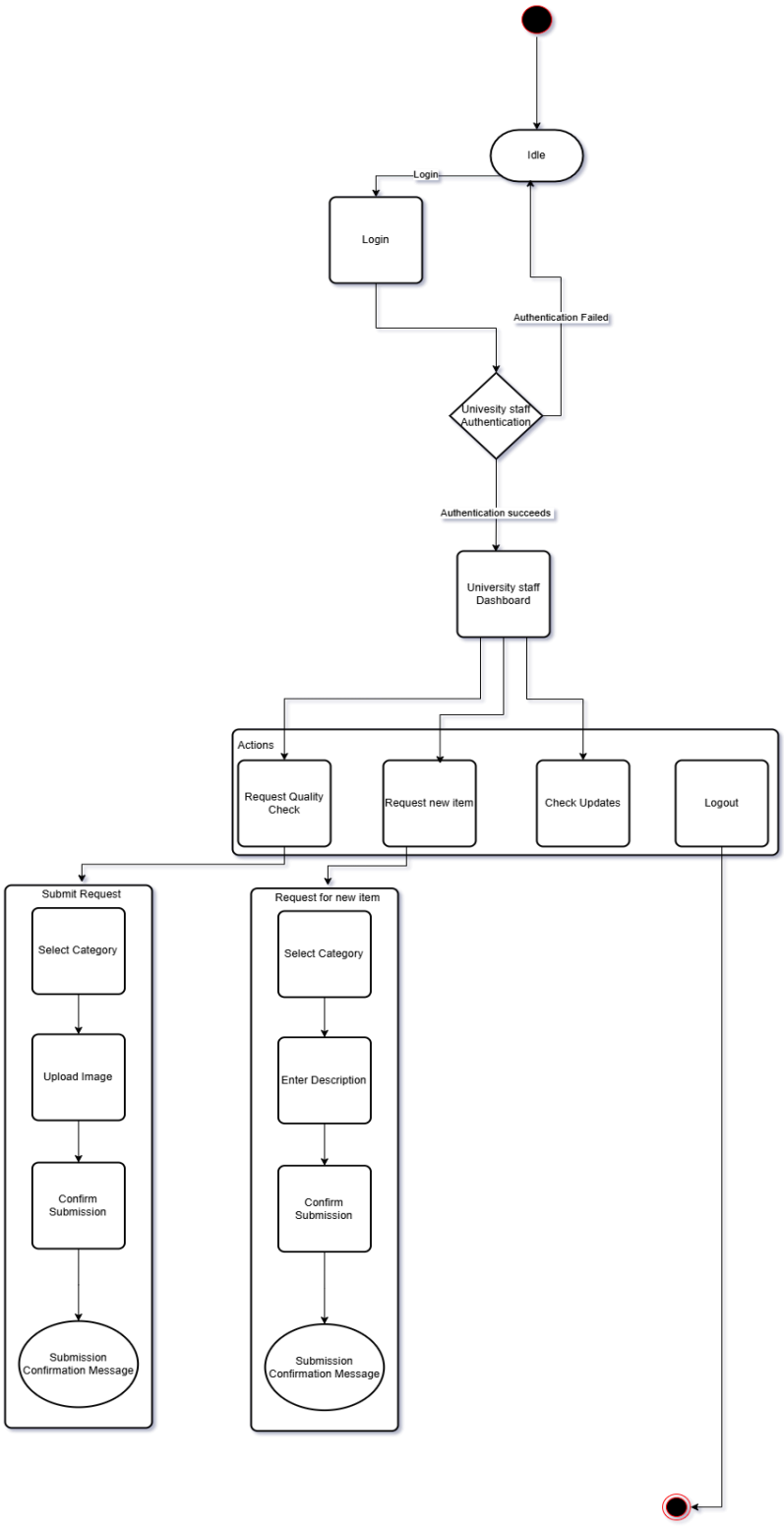

Website log in display
user contacts Finance
user has an account?
no
yes
User types username and password
Right credentials?
no
yes
User is sent to website homepage

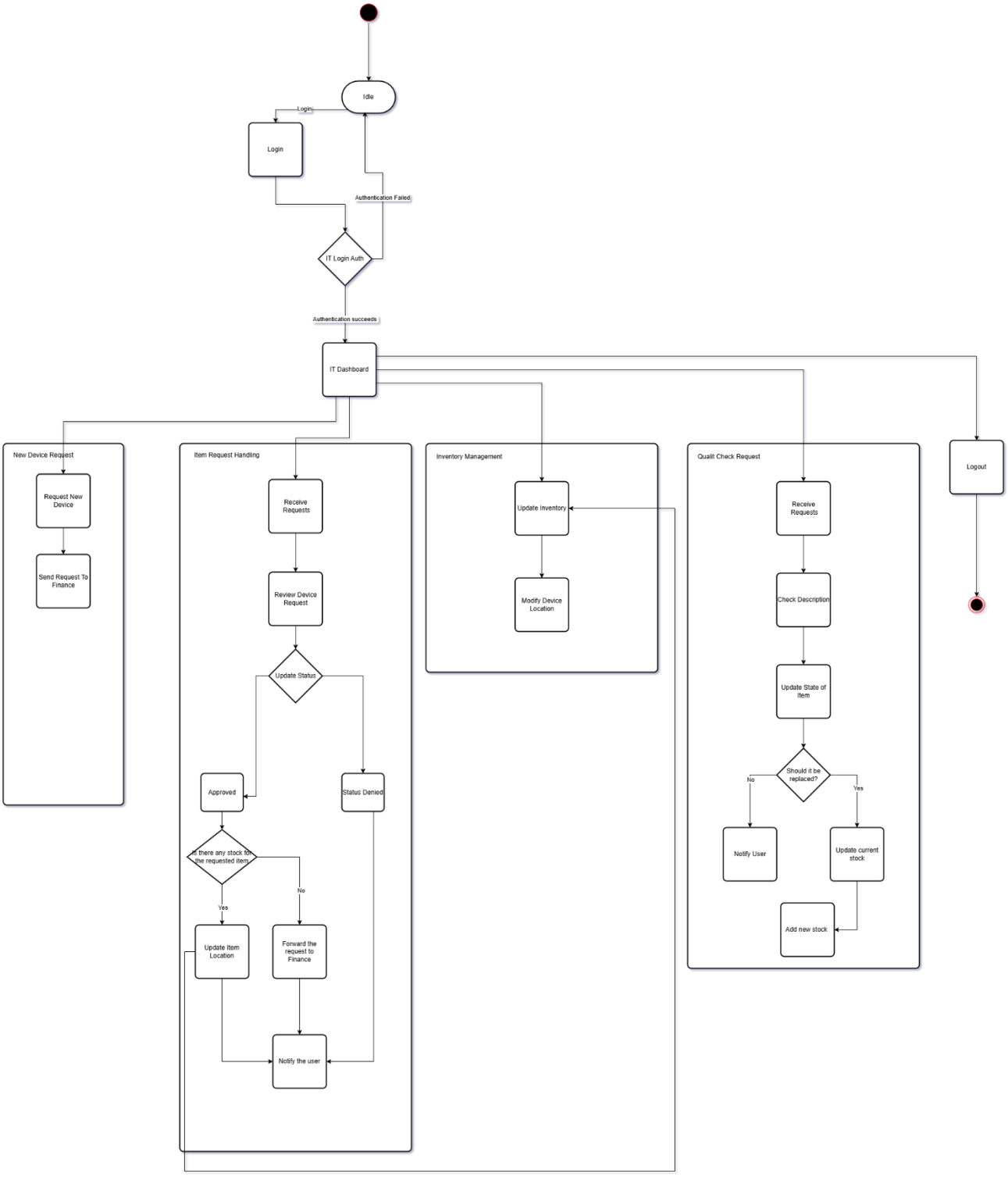

Does user remember Password
No
User puts 1 time password
Is it valid?
No
Yes
User puts old password
Is it valid?
no
yes
Yes
User enters new password to get confirmed
Confirmed?
no
yes
Login is successful

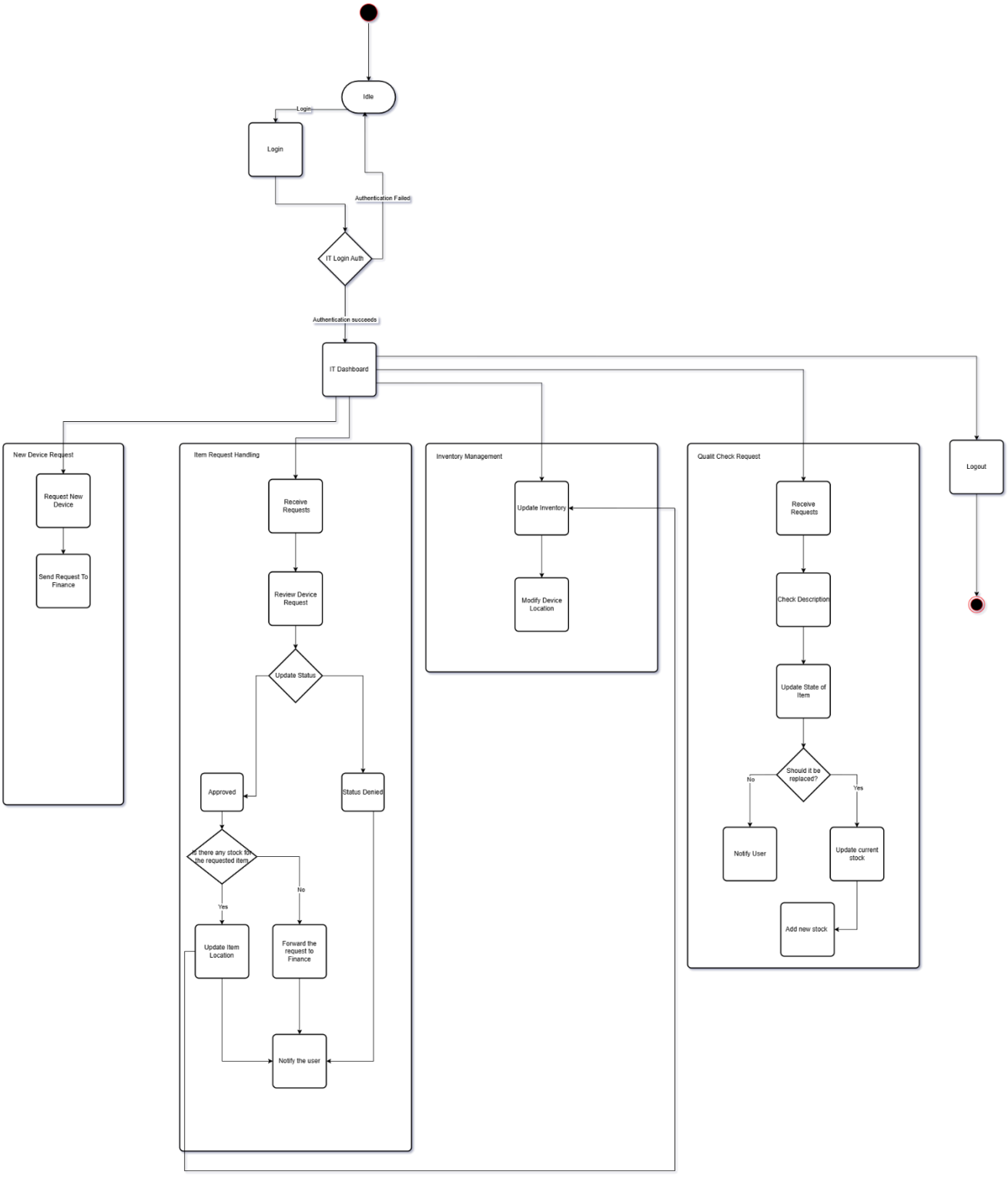

Finance puts credentials
no
Authentication
yes
Finance homescreen
Manage inventory
Add
Add or Delete stock?
Delete
Item name
Select the quantity
Add item details
Specify the category
Specify location
Select Category
Select Location
Select Item
Save changes
Confirmation
no
yes
Log out

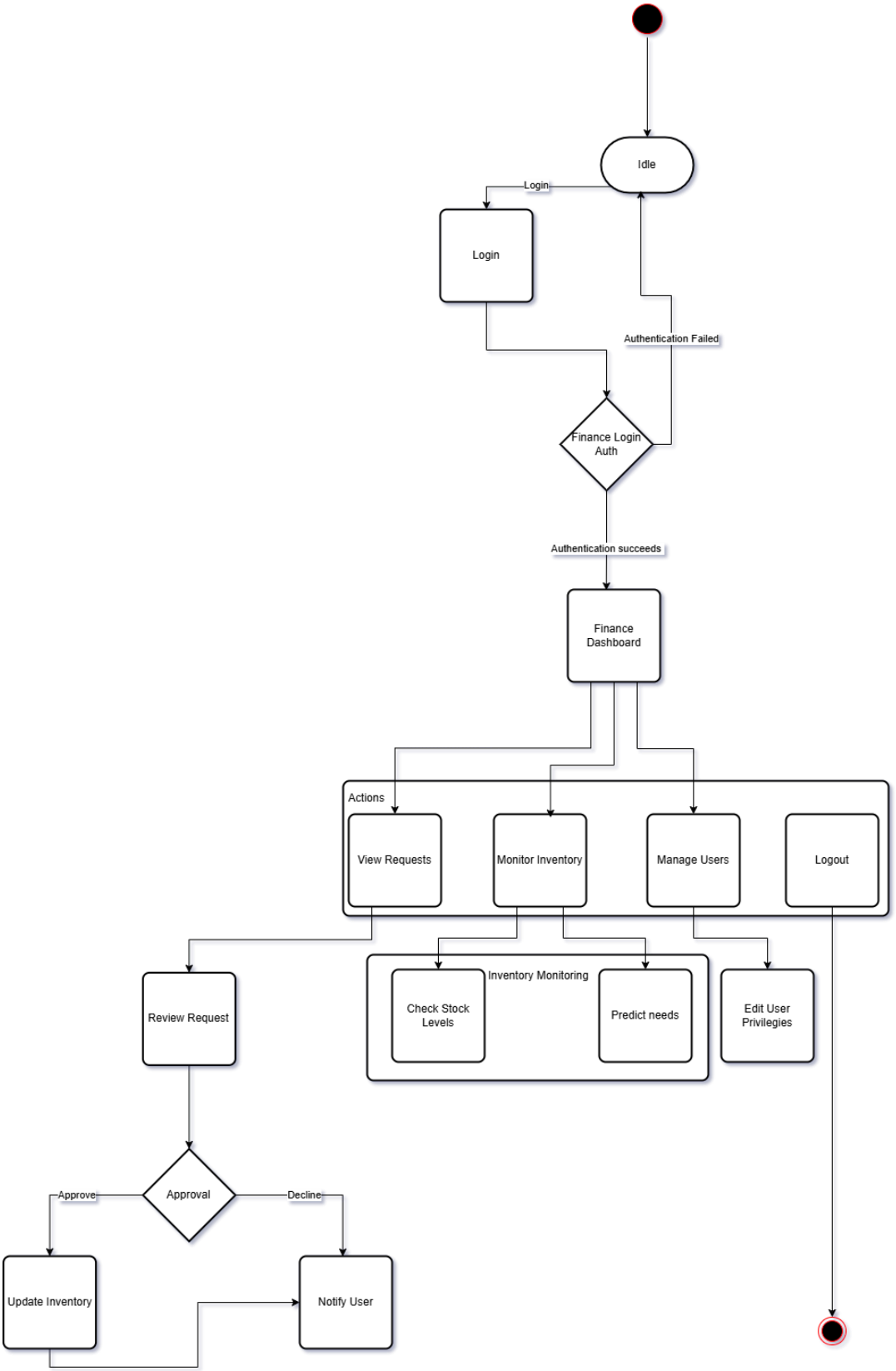


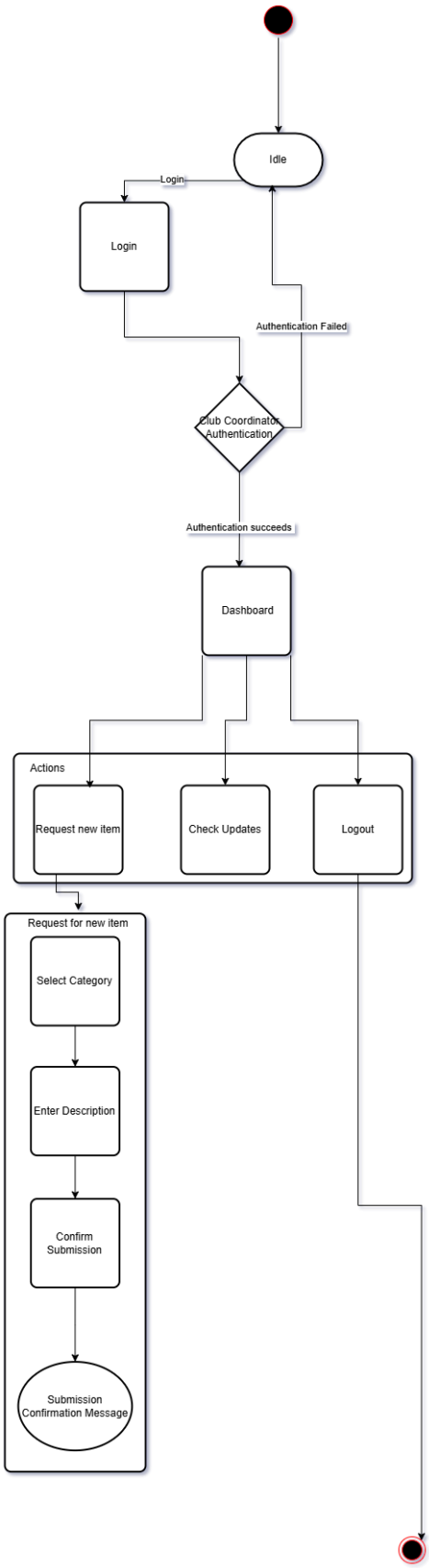
4.5 State Diagrams


Idle
Login
Login
Authentication Failed
Univesity staff Authentication
Authentication succeeds
University staff Dashboard
Actions
Request Quality Check
Request new item
Check Updates
Logout
Submit Request
Select Category
Upload Image
Confirm Submission
Submission Confirmation Message
Request for new item
Select Category
Enter Description
Confirm Submission
Submission Confirmation Message

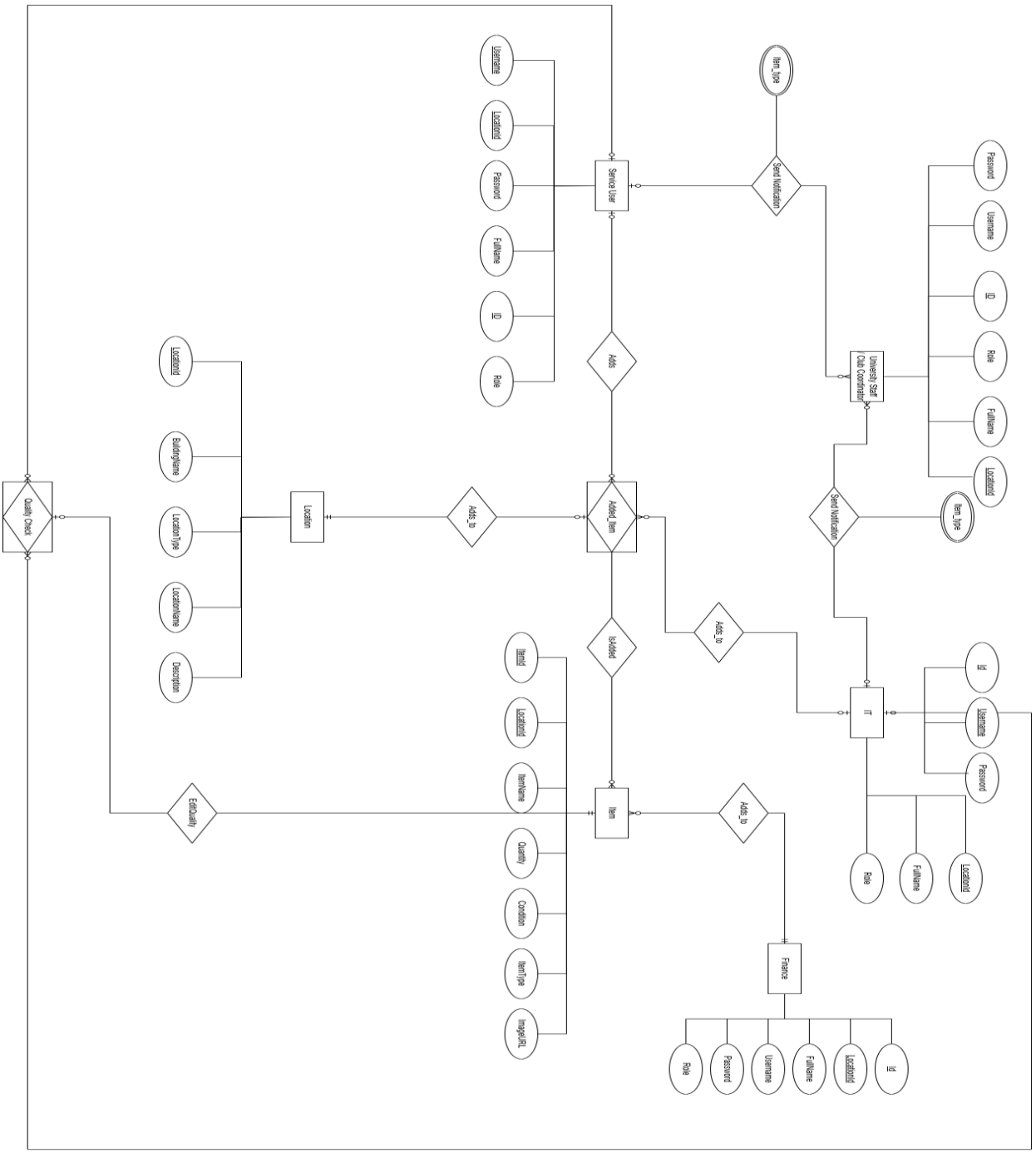

Idle
Login
Login
Authentication Failed
IT Login Auth
Authentication succeeds
IT Dashboard
New Device Request
Request New Device
Send Request To Finance
Item Request Handling
Receive Requests
Review Device Request
Update Status
Approved
Status Denied
Is there any stock for the requested item
Yes
No
Update Item Location
Forward the request to Finance
Notify the user
Inventory Management
Update Inventory
Modify Device Location
Qualit Check Request
Receive Requests
Check Description
Update State of Item
Should it be replaced?
No
Yes
Notify User
Update current stock
Add new stock
Logout


Idle
Login
Login
Authentication Failed
IT Login Auth
Authentication succeeds
IT Dashboard
New Device Request
Request New Device
Send Request To Finance
Item Request Handling
Receive Requests
Review Device Request
Update Status
Approved
Status Denied
Is there any stock for the requested item
Yes
No
Update Item Location
Forward the request to Finance
Notify the user
Inventory Management
Update Inventory
Modify Device Location
Qualit Check Request
Receive Requests
Check Description
Update State of Item
Should it be replaced?
No
Yes
Notify User
Update current stock
Add new stock
Logout


Idle
Login
Login
Authentication Failed
Finance Login Auth
Authentication succeeds
Finance Dashboard
Actions
View Requests
Monitor Inventory
Manage Users
Logout
Review Request
Inventory Monitoring
Check Stock Levels
Predict needs
Edit User Privilegies
Approval
Approve
Decline
Update Inventory
Notify User

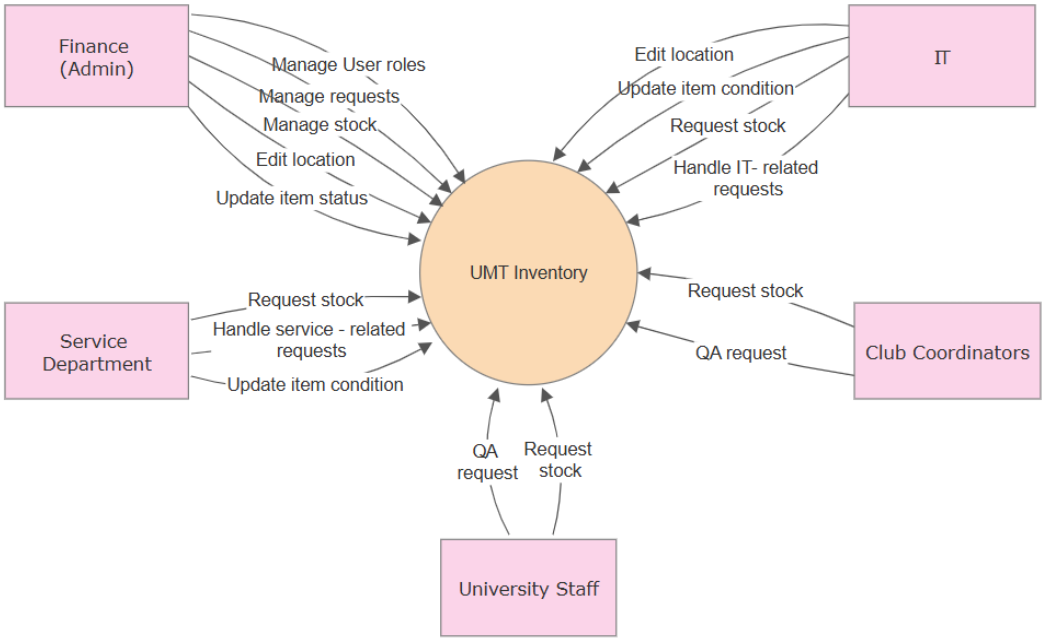

Idle
Login
Login
Authentication Failed
Club Coordinator Authentication
Authentication succeeds
Dashboard
Actions
Request new item
Check Updates
Logout
Request for new item
Select Category
Enter Description
Confirm Submission
Submission Confirmation Message

4.6 Entity Relationship Diagrams

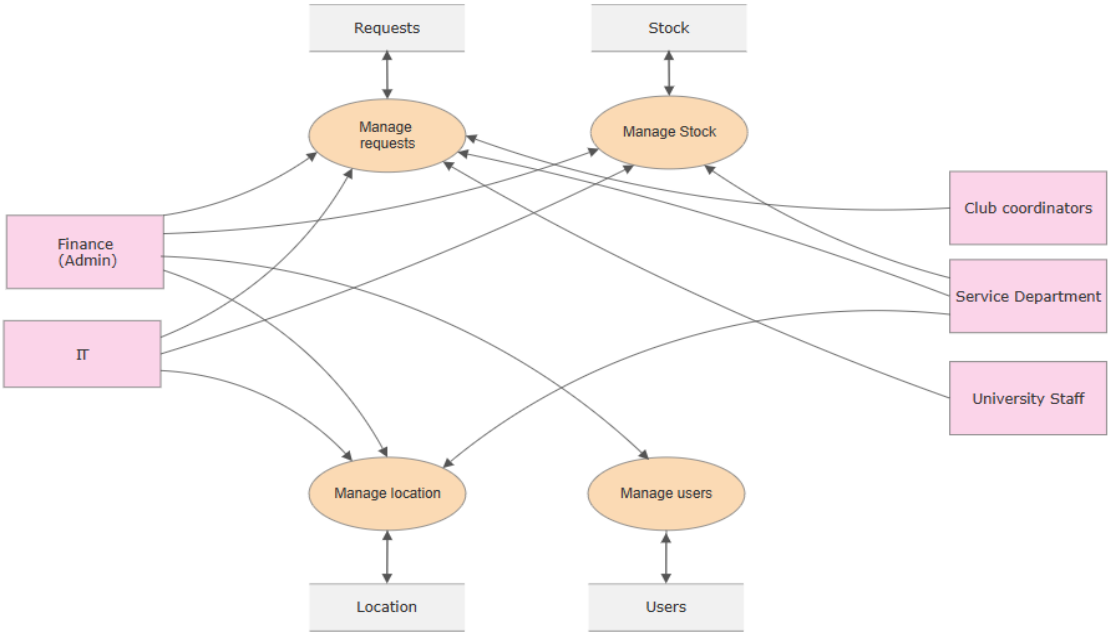


4.7 Data Flow Diagrams

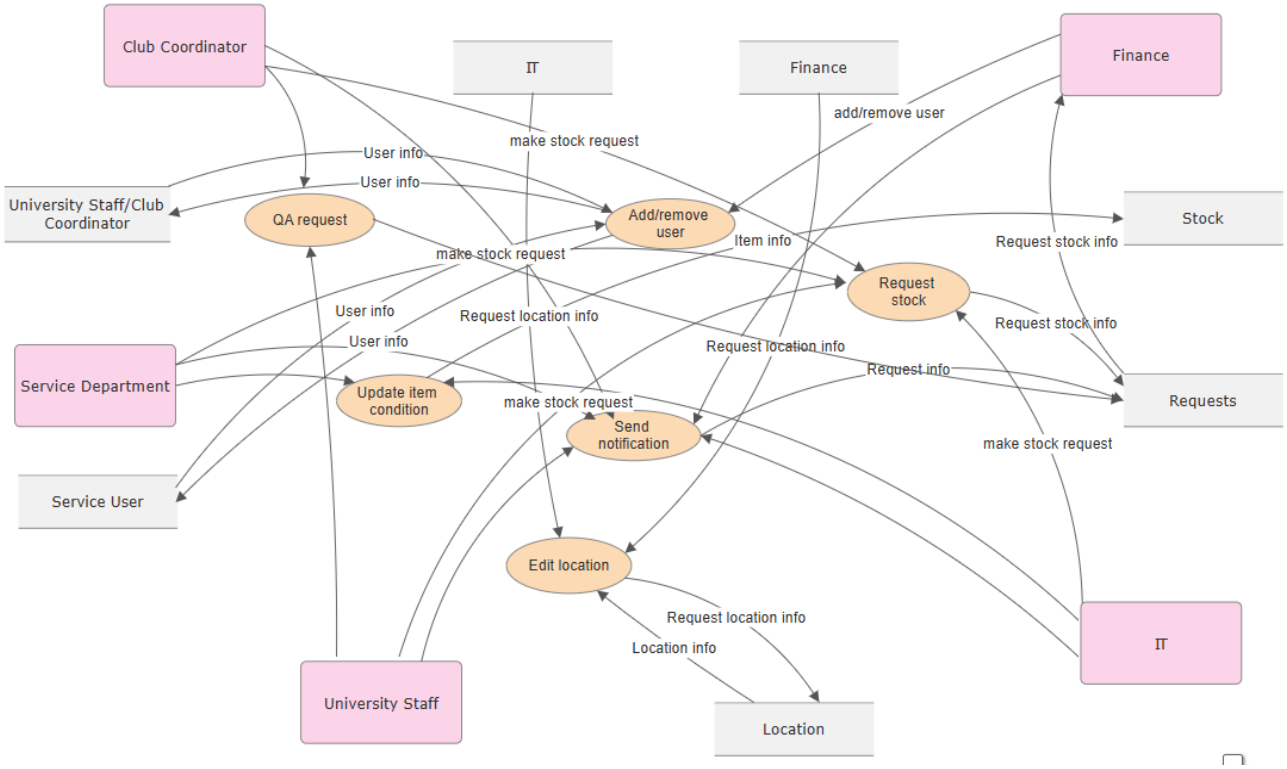
Level 0



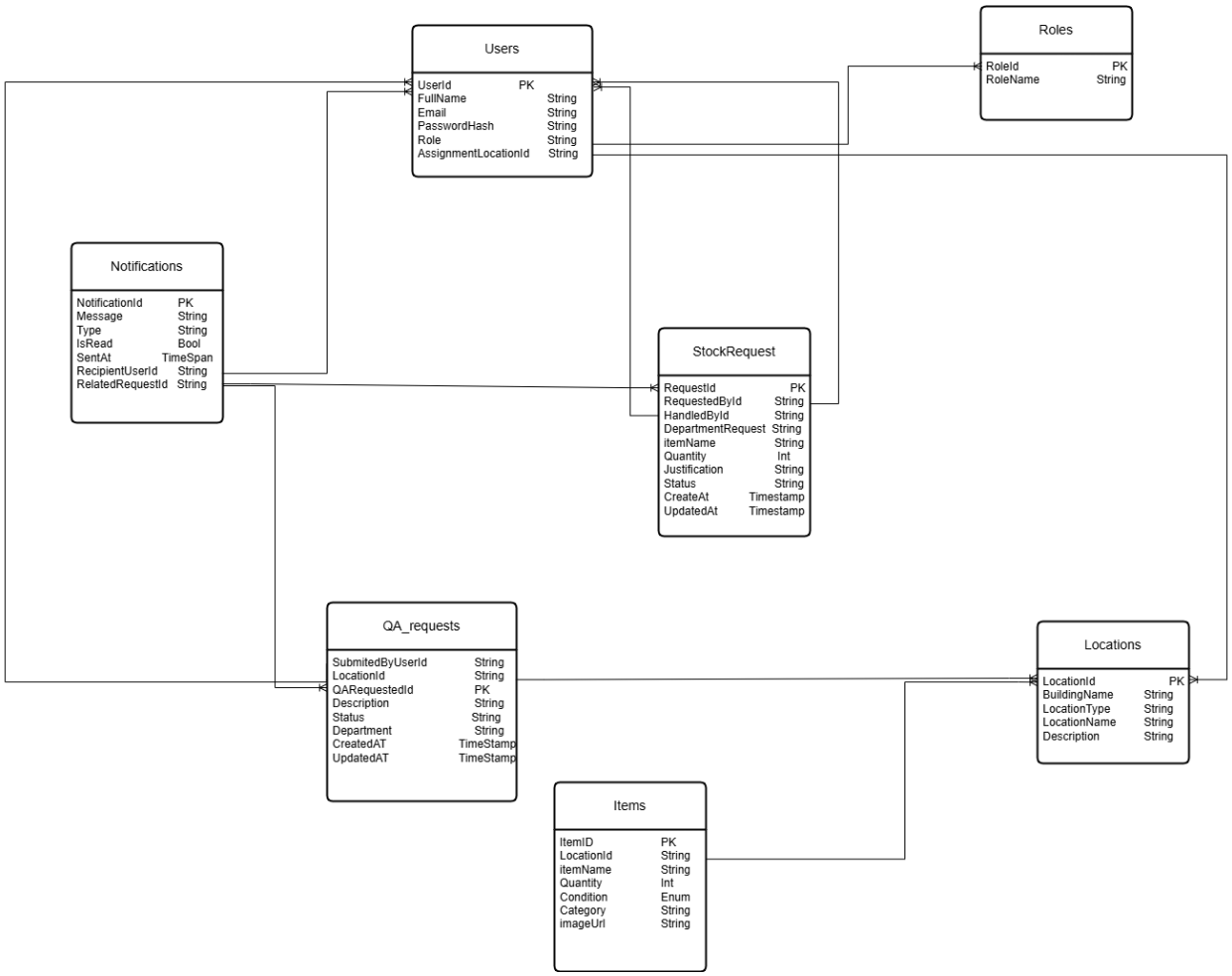
Level 1



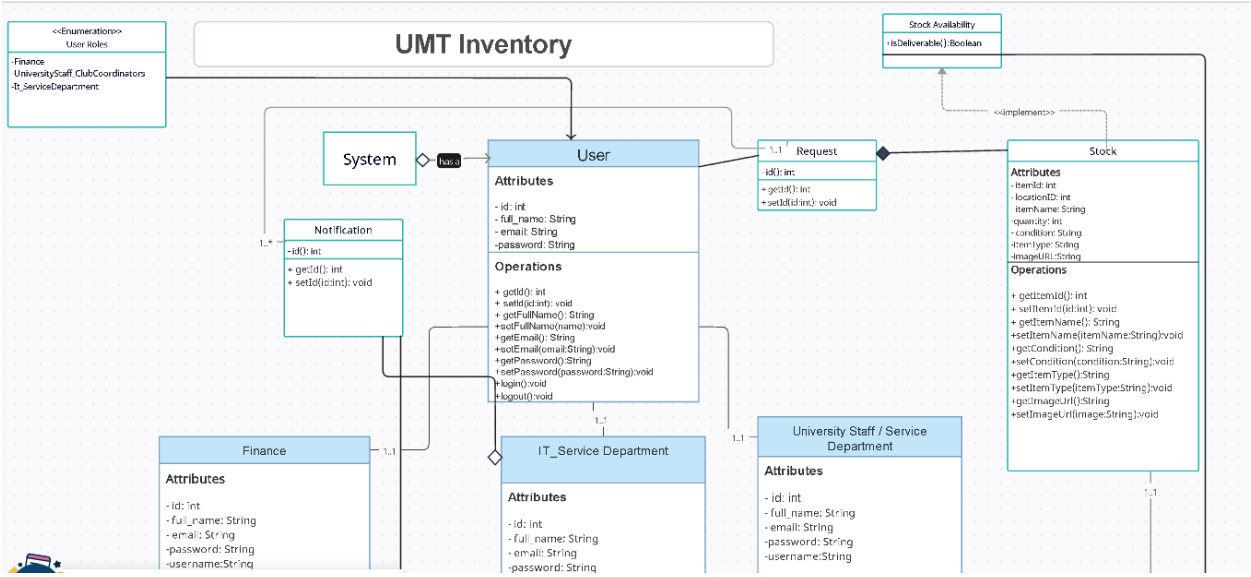
Level 2

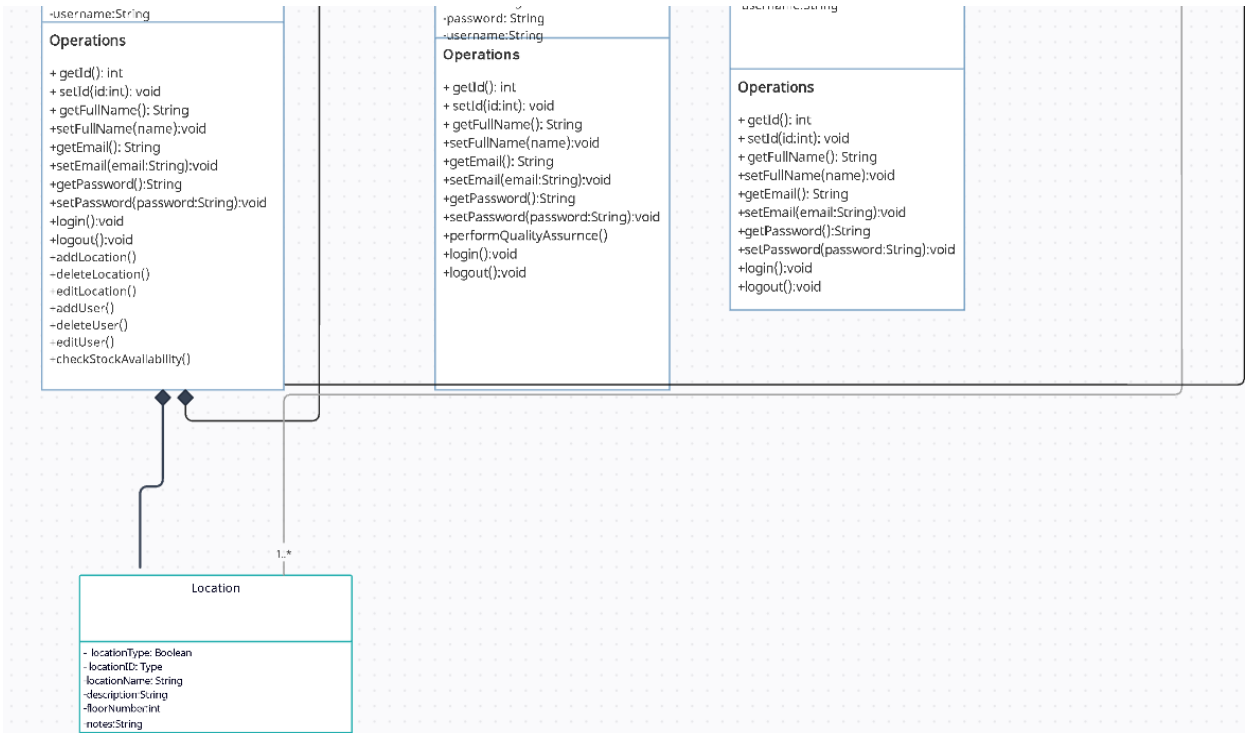

Club Coordinator
IT
Finance
Finance
add/remove user
make stock request
User info
User info
University Staff/Club Coordinator
QA request
Add/remove user
Item info
Stock
Request stock info
make stock request
Request stock
User info
Request location info
Request stock info
User info
Request location info
Request info
Service Department
Update item condition
make stock request
Requests
Send notification
make stock request
Service User
Edit location
Request location info
Location info
IT
University Staff
Location

4.8 RS:

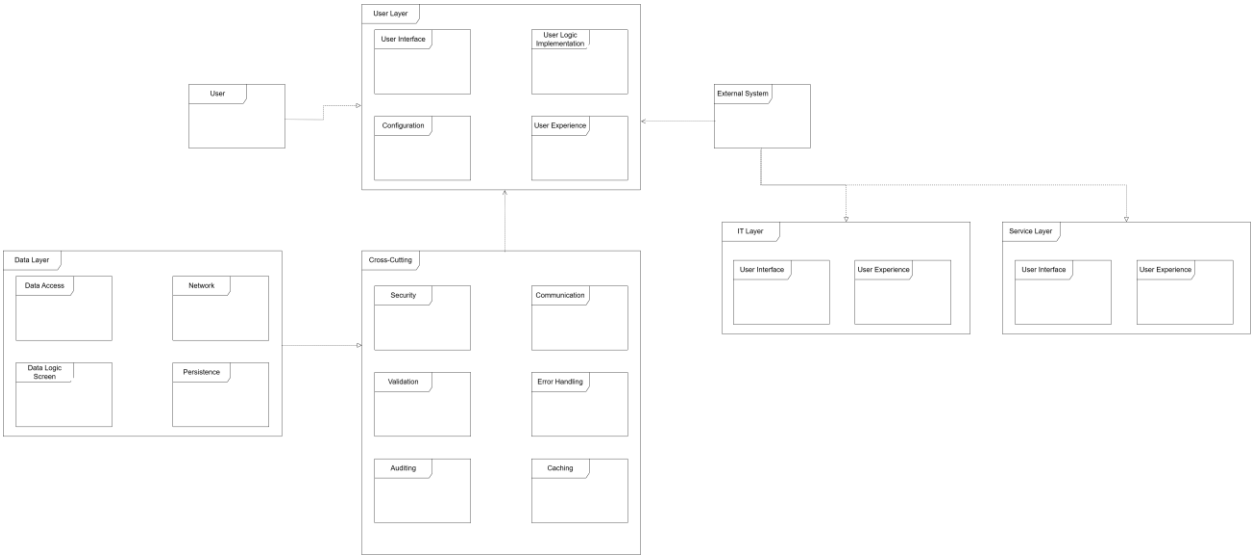


4.9 Class Diagrams:

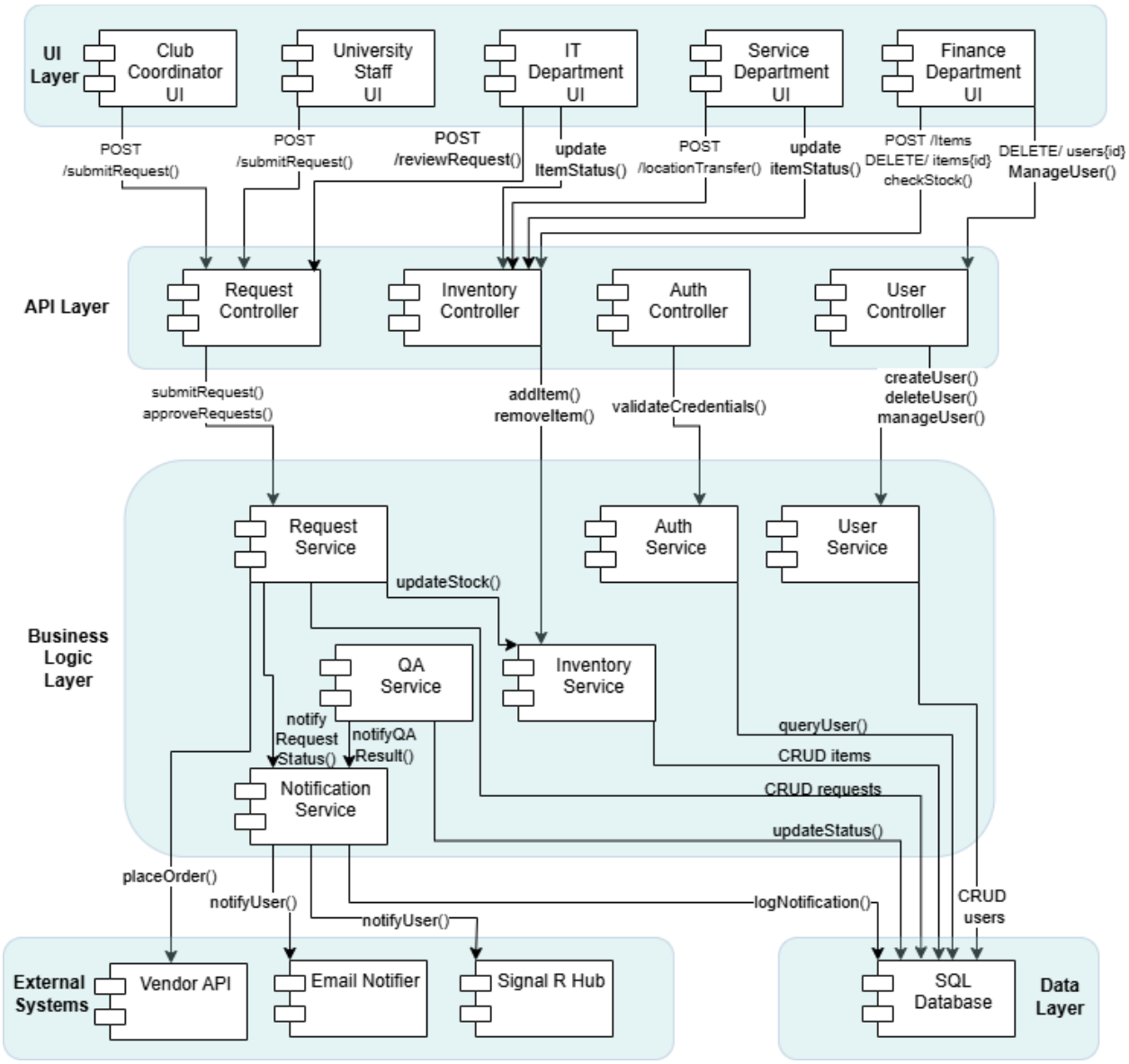

UMT Inventory
<<Enumeration>>
User Roles
-Finance
-UniversityStaff_ClubCoordinators
-It_ServiceDepartment
System
User
Attributes
- id: int
- full_name: String
- email: String
-password: String
Operations
+ getId(): int
+ setId(id:int): void
+ getFullName(): String
+setFullName(name):void
+getEmail(): String
+setEmail(email:String):void
+getPassword():String
+setPassword(password:String):void
+login():void
+logout():void
Request
id(): int
+ getId(): int
+ setId(id:int): void
Stock Availability
+isDeliverable():Boolean
<<implement>>
Stock
Attributes
-itemId: int
- locationID: int
-itemName: String
-quantity: int
- condition: Sting
-itemType: String
-imageURL:String
Operations
+ getItemId(): int
+ setItemId(id:int): void
+ getItemName(): String
+setItemName(itemName:String):void
+getCondition(): String
+setCondition(condition:String):void
+getItemType():String
+setItemType(itemType:String):void
+getImageUrl():String
+setImageUrl(image:String):void
Notification
-id(): int
+ getId(): int
+ setId(id:int): void
Finance
Attributes
- id: int
- full_name: String
- email: String
-password: String
-username:String
IT_Service Department
Attributes
- id: int
- full_name: String
- email: String
-password: String
University Staff / Service Department
Attributes
- id: int
- full_name: String
- email: String
-password: String
-username:String


-username:String
Operations
+ getId(): int
+ setId(id:int): void
+ getFullName(): String
+setFullName(name):void
+getEmail(): String
+setEmail(email:String):void
+getPassword():String
+setPassword(password:String):void
+login():void
+logout():void
-addLocation()
-deleteLocation()
+editLocation()
-addUser()
-deleteUser()
+editUser()
-checkStockAvailability()
-password: String
-username:String
Operations
+ getId(): int
+ setId(id:int): void
+ getFullName(): String
+setFullName(name):void
+getEmail(): String
+setEmail(email:String):void
+getPassword():String
+setPassword(password:String):void
+performQualityAssurnce()
+login():void
+logout():void
Operations
+ getId(): int
+ setId(id:int): void
+ getFullName(): String
+setFullName(name):void
+getEmail(): String
+setEmail(email:String):void
+getPassword():String
+setPassword(password:String):void
+login():void
+logout():void
1..*
Location
- locationType: Boolean
- locationID: Type
-locationName: String
-description:Suring
-floorNumber:int
-notes:String

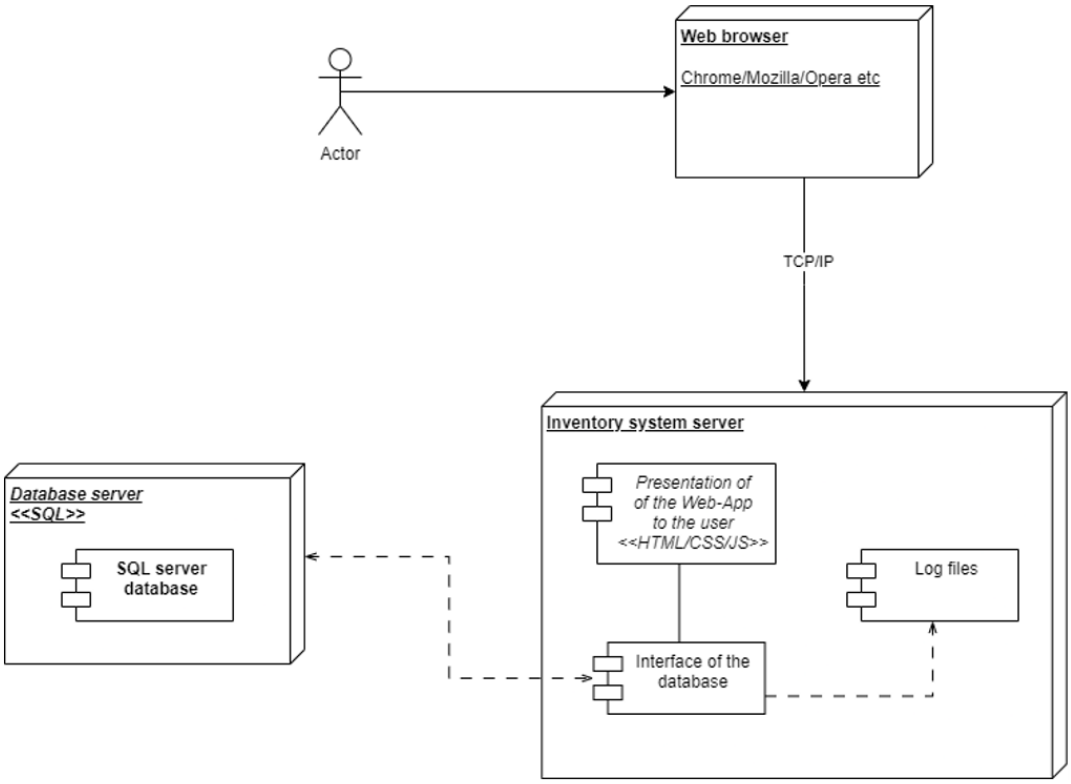
4.10 Package Diagrams:



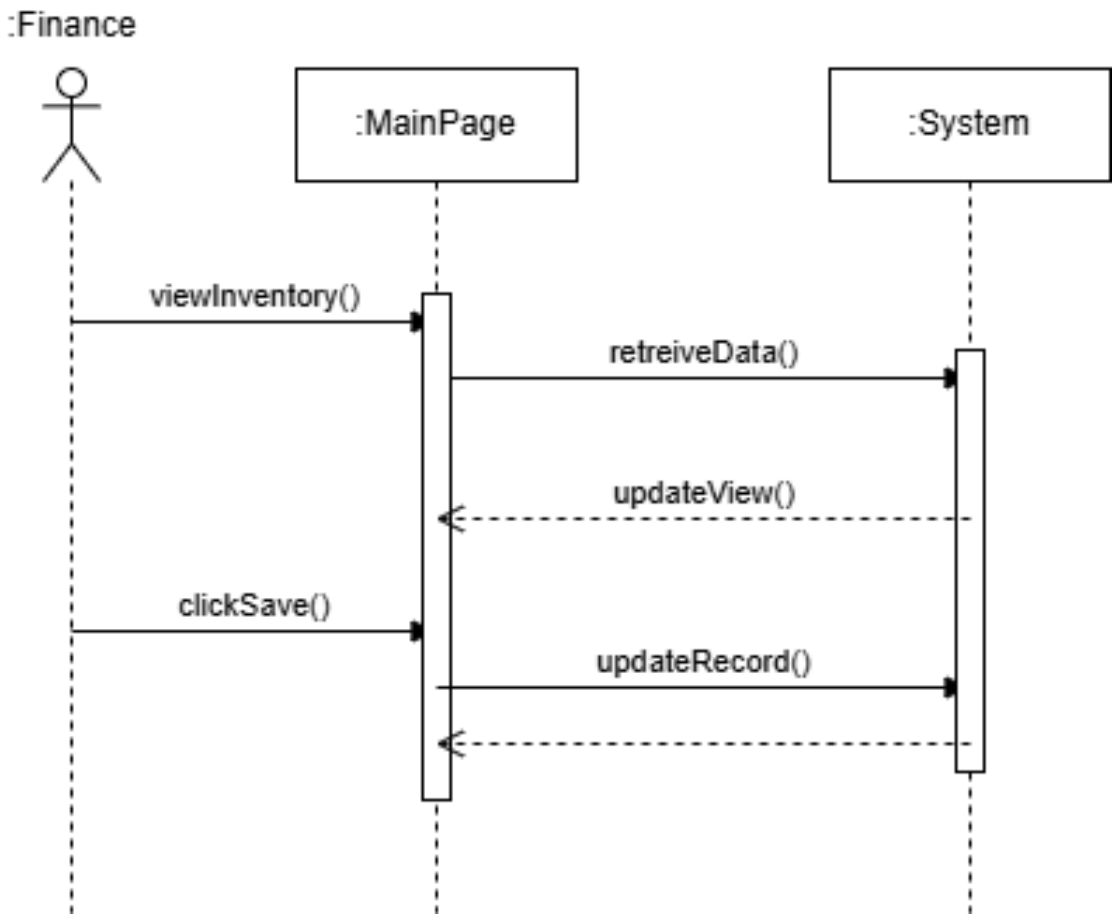
4.11 Component Diagrams:

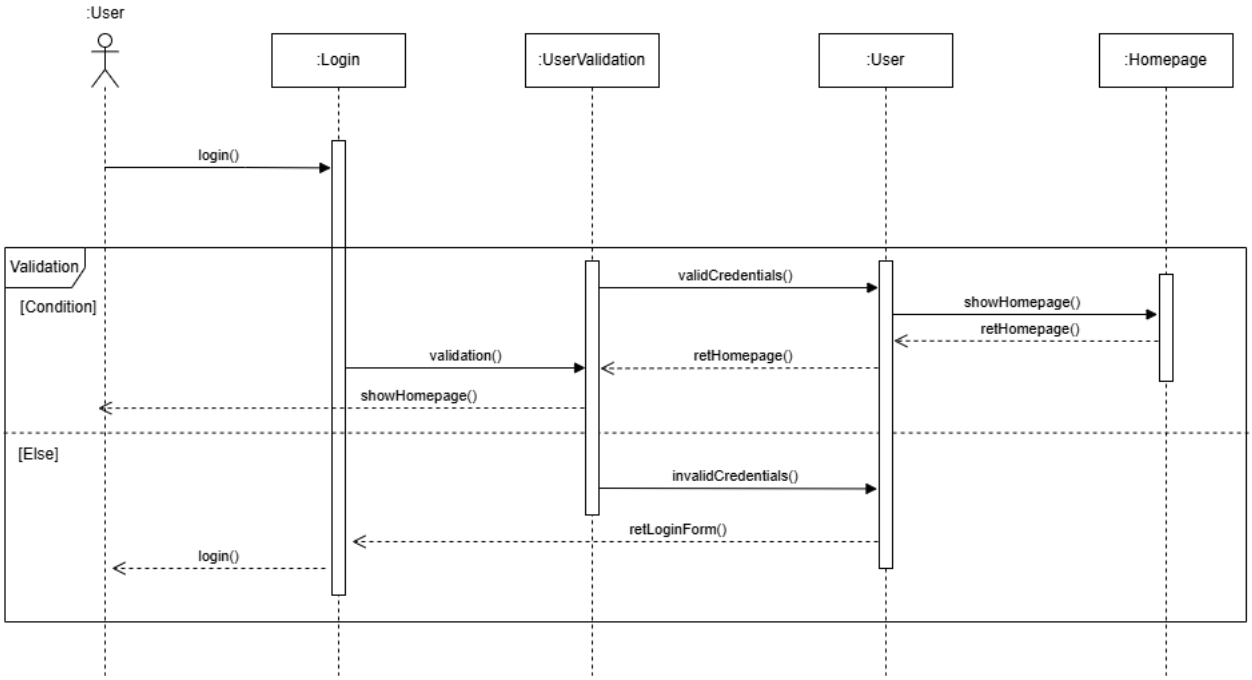

UI Layer
Club Coordinator UI
University Staff UI
IT Department UI
Service Department UI
Finance Department UI
POST /submitRequest()
POST /submitRequest()
POST /reviewRequest()
update ItemStatus()
POST /locationTransfer()
update itemStatus()
POST /Items
DELETE/ items{id}
checkStock()
DELETE/ users{id}
ManageUser()
API Layer
Request Controller
Inventory Controller
Auth Controller
User Controller
submitRequest()
approveRequests()
addItem()
removeItem()
validateCredentials()
createUser()
deleteUser()
manageUser()
Business Logic Layer
Request Service
Auth Service
User Service
updateStock()
QA Service
Inventory Service
notify Request Status()
notifyQA Result()
queryUser()
CRUD items
Notification Service
CRUD requests
updateStatus()
placeOrder()
notifyUser()
notifyUser()
logNotification()
CRUD users
External Systems
Vendor API
Email Notifier
Signal R Hub
SQL Database
Data Layer

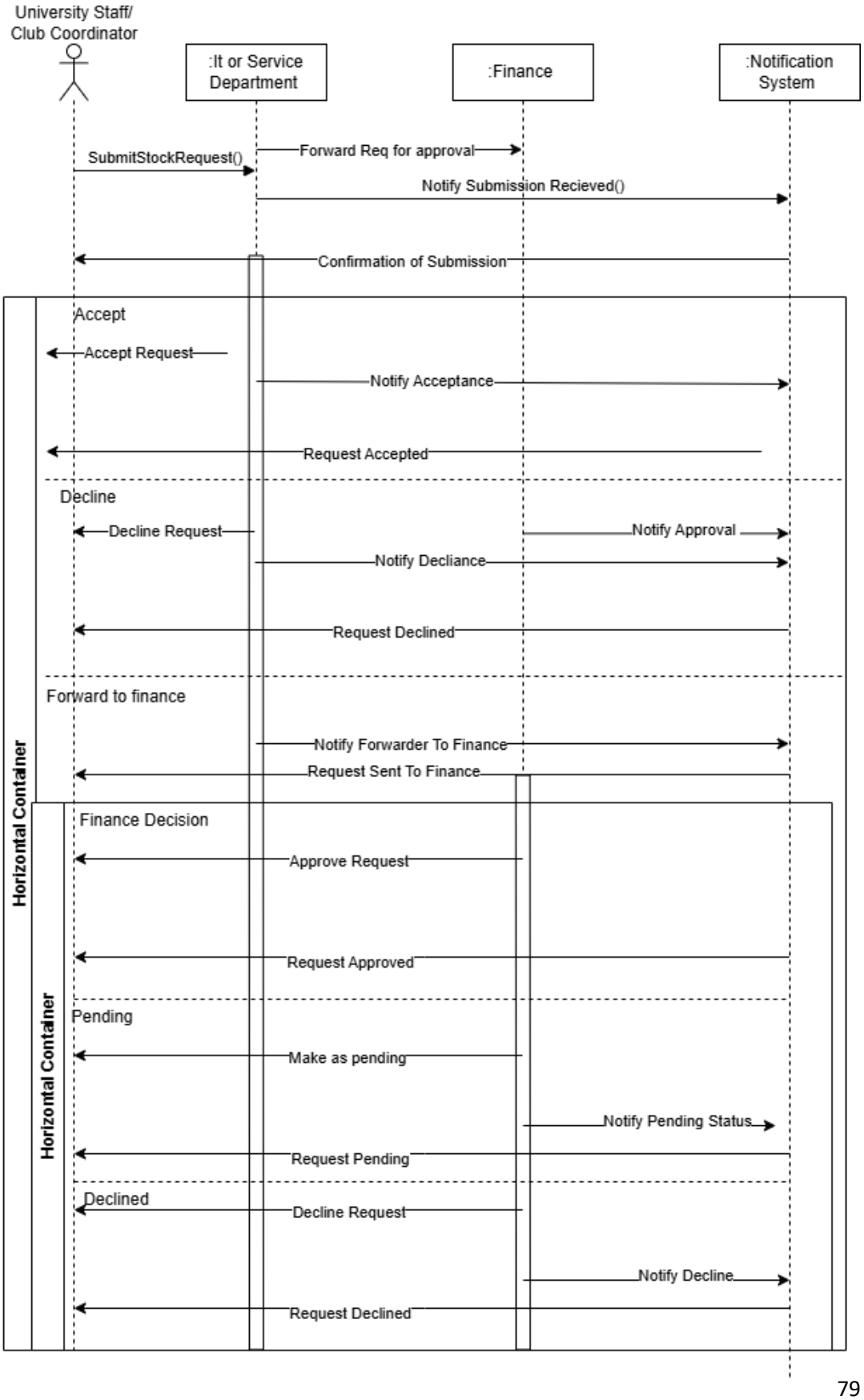
4.12 Deployment Diagrams:

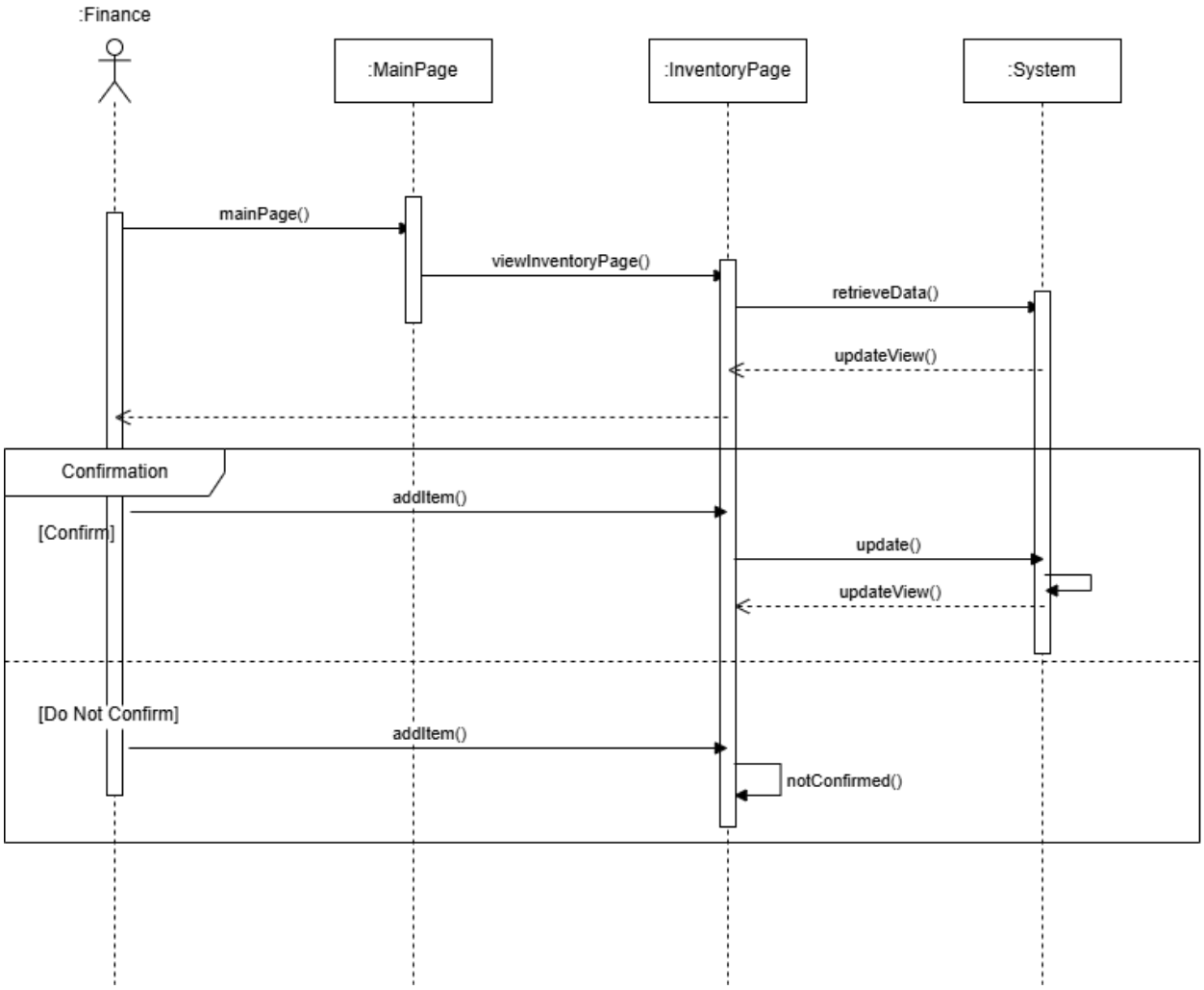


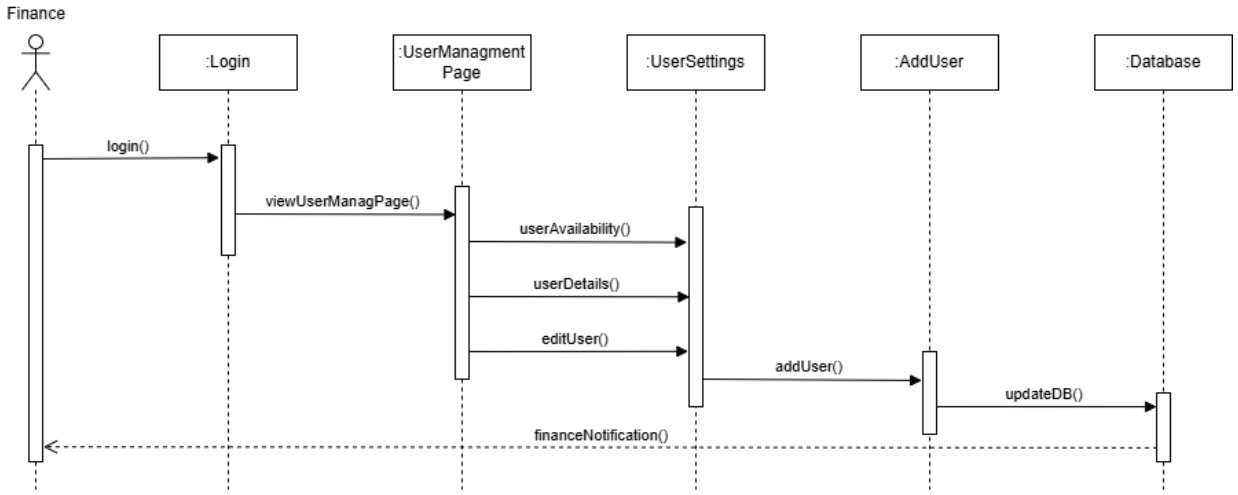
4.13 Sequence Diagrams:

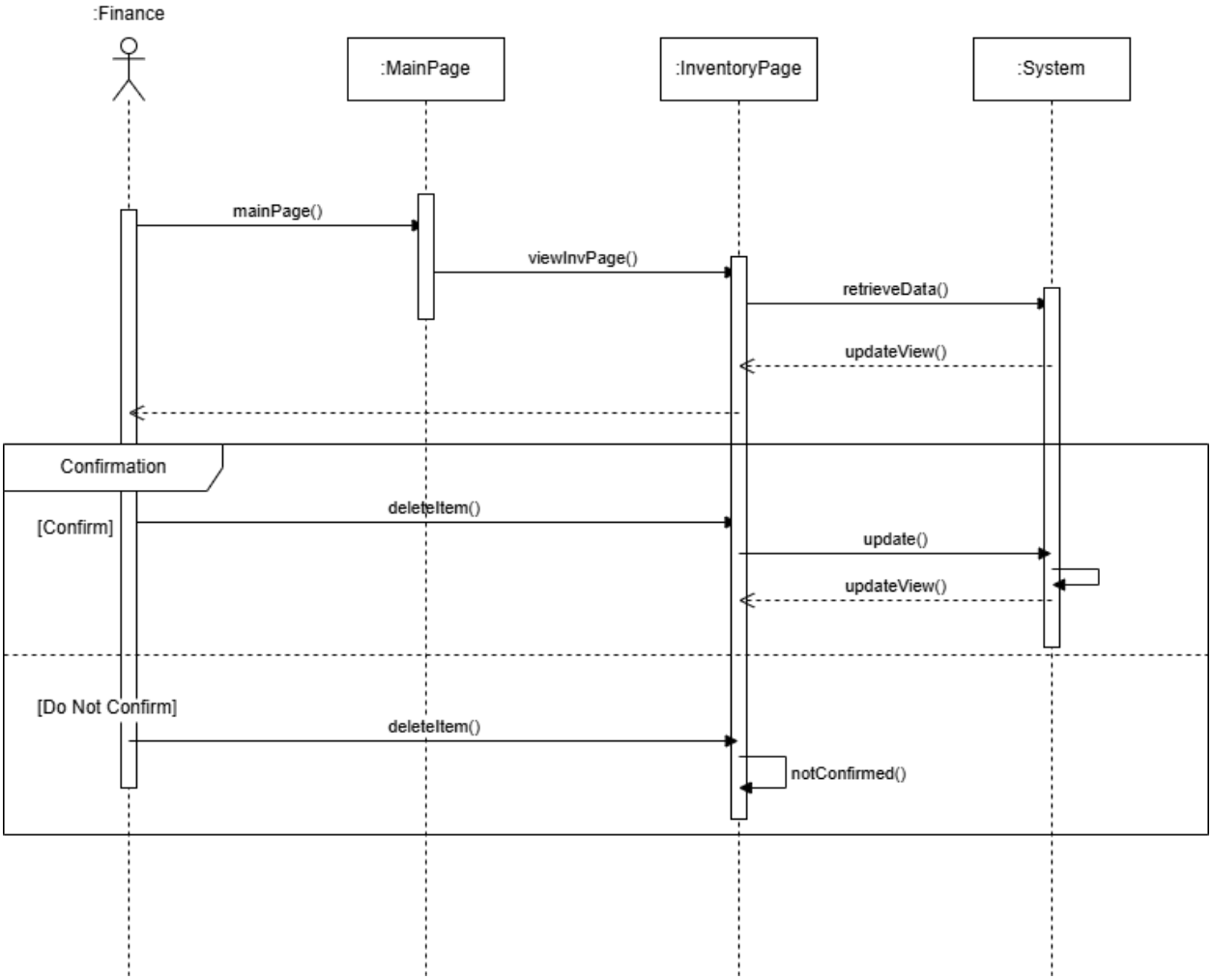

:Finance
:MainPage
:System
viewInventory()
retreiveData()
updateView()
clickSave()
updateRecord()

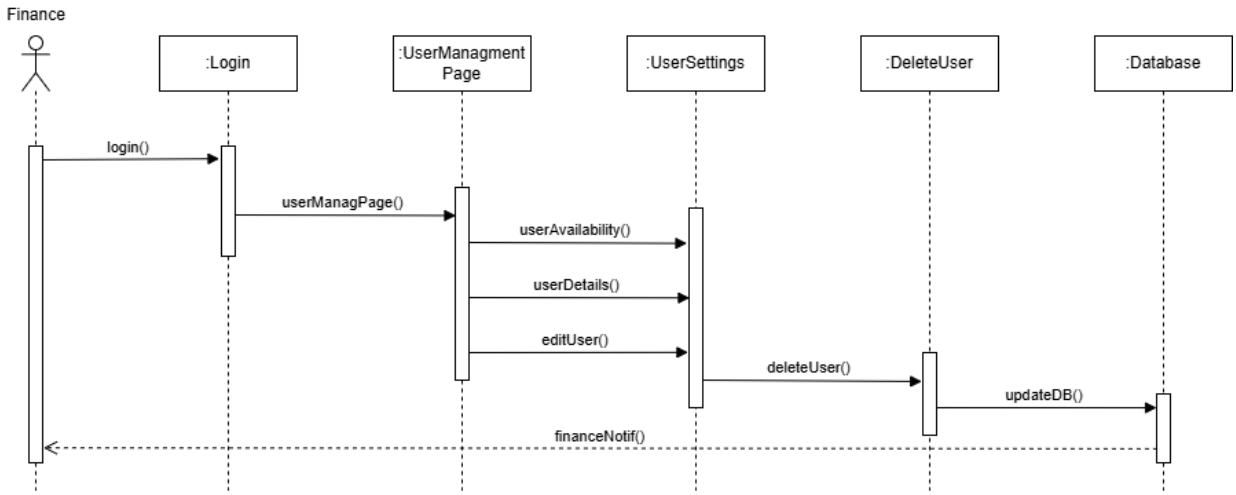

:User
:Login
:UserValidation
:User
:Homepage
login()
Validation
[Condition]
validCredentials()
showHomepage()
retHomepage()
validation()
retHomepage()
showHomepage()
[Else]
invalidCredentials()
retLoginForm()
login()

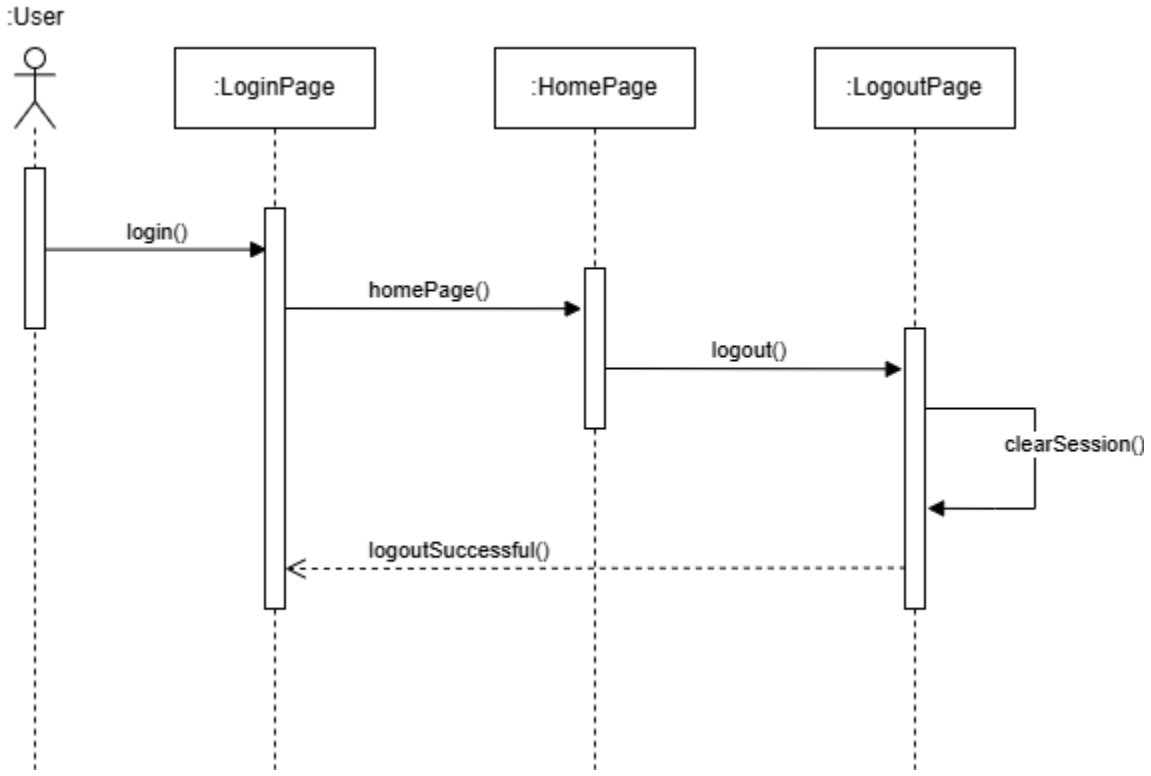

University Staff/
Club Coordinator
:It or Service Department
:Finance
:Notification System
SubmitStockRequest()
Forward Req for approval
Notify Submission Recieved()
Confirmation of Submission
Accept
Accept Request
Notify Acceptance
Request Accepted
Decline
Decline Request
Notify Approval
Notify Decliance
Request Declined
Forward to finance
Notify Forwarder To Finance
Request Sent To Finance
Horizontal Container
Finance Decision
Approve Request
Request Approved
Horizontal Container
Pending
Make as pending
Notify Pending Status
Request Pending
Declined
Decline Request
Notify Decline
Request Declined

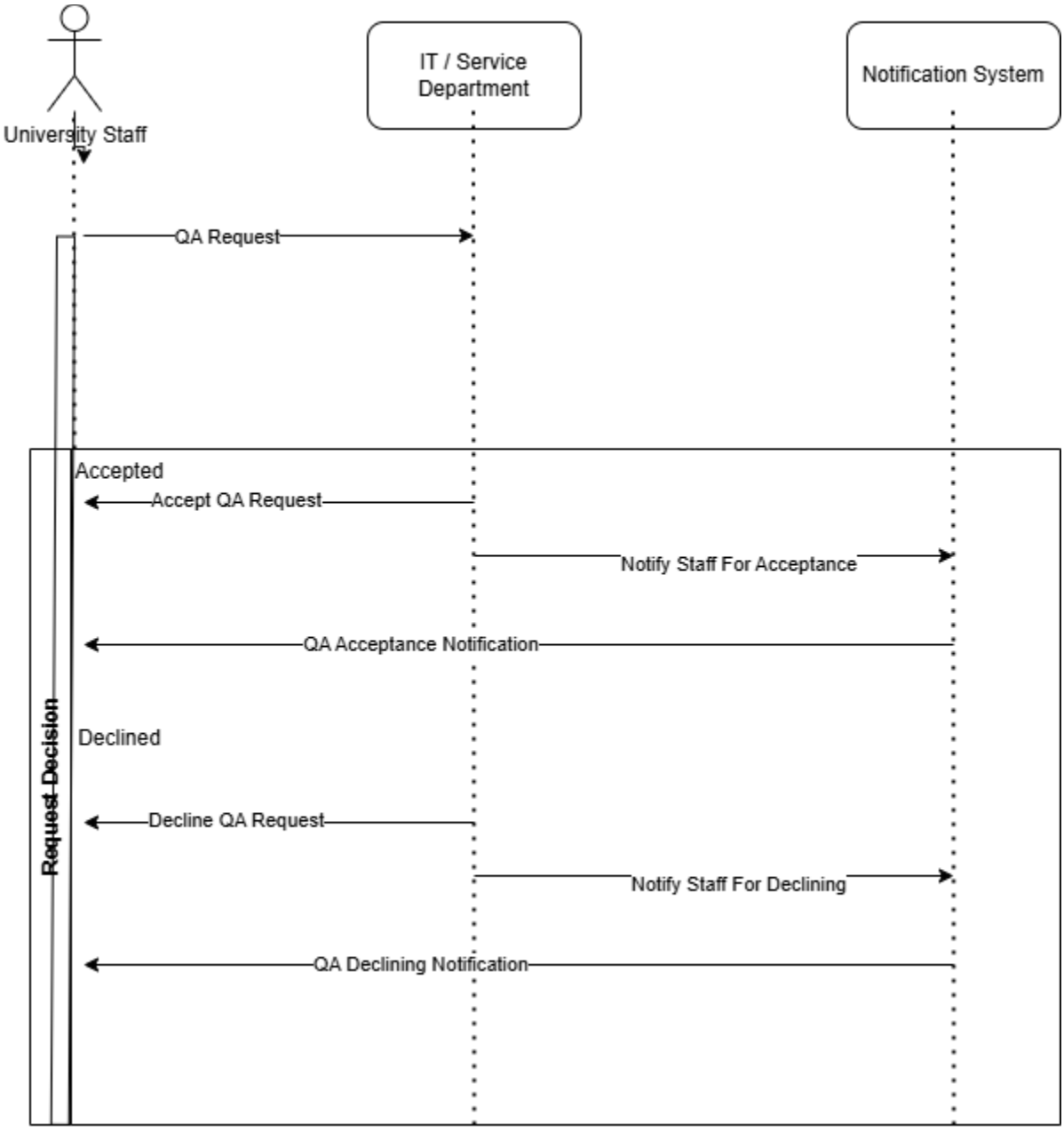

:Finance
:MainPage
:InventoryPage
:System
mainPage()
viewInventoryPage()
retrieveData()
updateView()
Confirmation
addItem()
[Confirm]
update()
updateView()
[Do Not Confirm]
addItem()
notConfirmed()

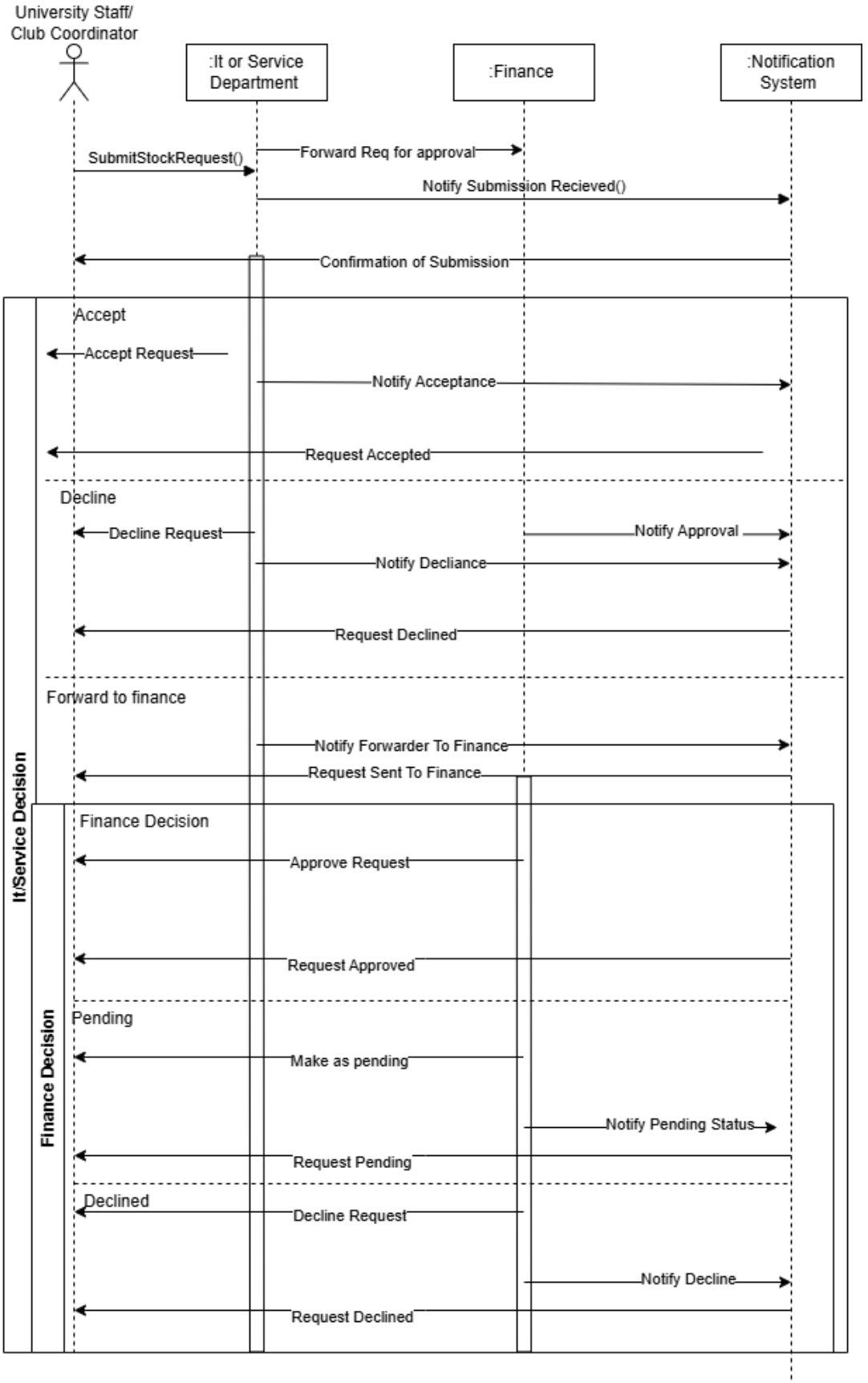

Finance
:Login
:UserManagment Page
:UserSettings
:AddUser
:Database
login()
viewUserManagPage()
userAvailability()
userDetails()
editUser()
addUser()
updateDB()
financeNotification()


:Finance
:MainPage
:InventoryPage
:System
mainPage()
viewInvPage()
retrieveData()
updateView()
Confirmation
[Confirm]
deleteItem()
update()
updateView()
[Do Not Confirm]
deleteItem()
notConfirmed()


Finance
:Login
:UserManagment Page
:UserSettings
:DeleteUser
:Database
login()
userManagPage()
userAvailability()
userDetails()
editUser()
deleteUser()
updateDB()
financeNotif()


:User
:LoginPage
:HomePage
:LogoutPage
login()
homePage()
logout()
clearSession()
logoutSuccessful()


University Staff
IT / Service Department
Notification System
QA Request
Request Decision
Accepted
Accept QA Request
Notify Staff For Acceptance
QA Acceptance Notification
Declined
Decline QA Request
Notify Staff For Declining
QA Declining Notification


University Staff/
Club Coordinator
:It or Service Department
:Finance
:Notification System
SubmitStockRequest()
Forward Req for approval
Notify Submission Recieved()
Confirmation of Submission
Accept
Accept Request
Notify Acceptance
Request Accepted
Decline
Decline Request
Notify Approval
Notify Decliance
Request Declined
Forward to finance
Notify Forwarder To Finance
Request Sent To Finance
It/Service Decision
Finance Decision
Approve Request
Request Approved
Finance Decision
Pending
Make as pending
Notify Pending Status
Request Pending
Declined
Decline Request
Notify Decline
Request Declined