Equipment Tracker

SOFTWARE DESIGN DOCUMENT

Thomas Smiley

4/28/2024 | REVISION: 1.0

Table of Contents

1 Introduction

1.1 Problem Statement

Managing equipment and resources in a company is a very difficult task. Often the equipment is bought by the company for permanent use or rented for temporary use. There are also subcomponent modules which are bought and built into a product and sold. Tracking where all this equipment is or who is using it in one location does not exist to the extent that is needed. Often equipment is lost or unknowingly needed by multiple people at the same time. This leads to company downtime and annoying periods of unproductivity due to equipment mismanagement.

1.2 Business Requirements

Functionality

- Add Equipment to equipment database
- Remove equipment from equipment database
- Check in/check out equipment from equipment database
- Reserve equipment for future use from equipment database
- Track where the equipment physically is in warehouse
- Track who checked out equipment and for how long
- Track where temp items (submodule components used in products to be sold) are and what project they are assigned to
- Track last calibration and upcoming calibration dates
- Integrate with QR codes on the equipment for easy check in and check out
- Track the number of each equipment in stock

Target users

- Managers – Reserve equipment for their projects
- Shipping and Receiving – Update status of equipment when received, stored in warehouse, and shipped
- Procurement – Add equipment to the database as its being shipped in
- Technicians/Engineers – Check in and check out equipment for use in assigned projects

Business Goals

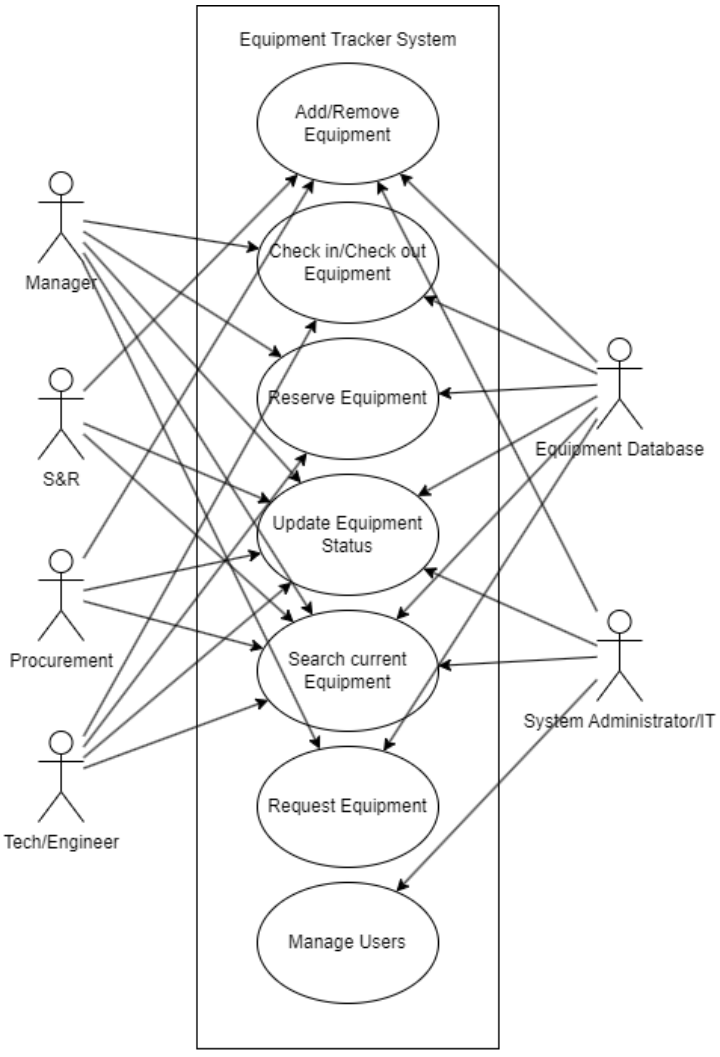
- The system should be an application only accessible with a login from company network
- The system should be updateable in real time
- The system should be able to recover from any failures
- The system should have a simple and easy to use user interface
- The system should facilitate a lower employee and project downtime caused by the lack of visibility of equipment location

1.3 Non-Functional Requirements

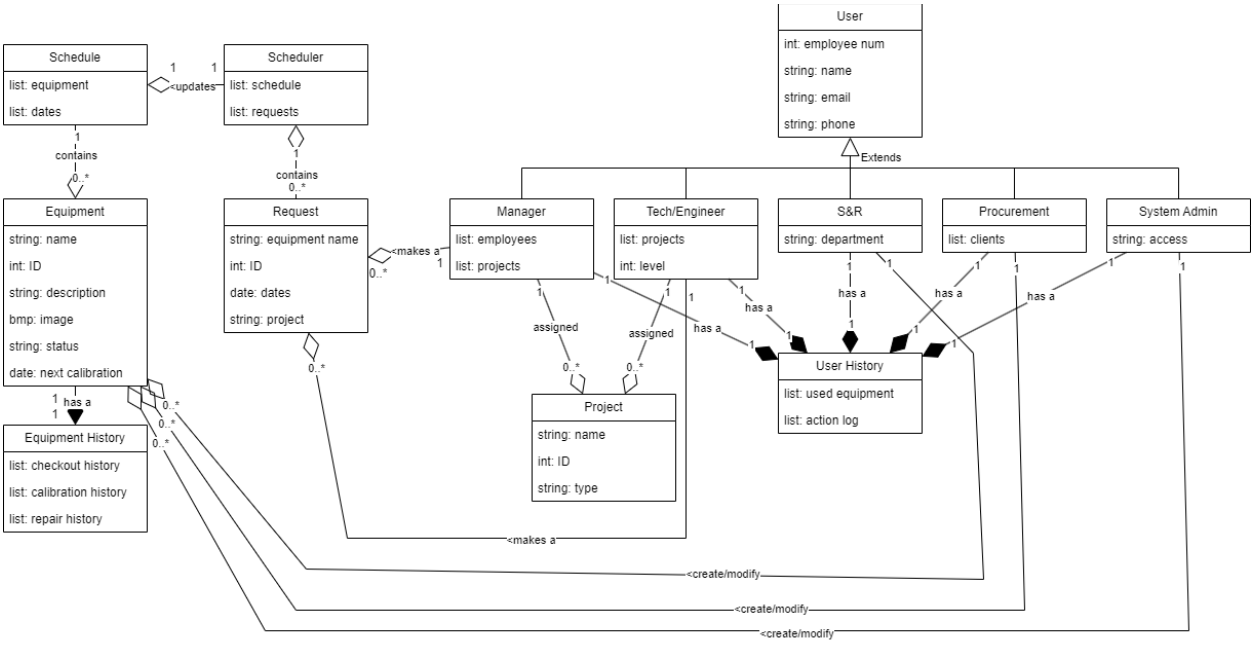
- The system should be able to support up to 100,000 individual pieces of equipment
- The system should be able to respond to any user input within 100 ms
- The system should be able to update the database from any action within 2 seconds
- Ther system should be able to support user authentication with single sign on via Kerberos on the company network
- The system should support data encryption in flight and at rest
- The codebase should be highly modular to support changing business requirements

2 System Design

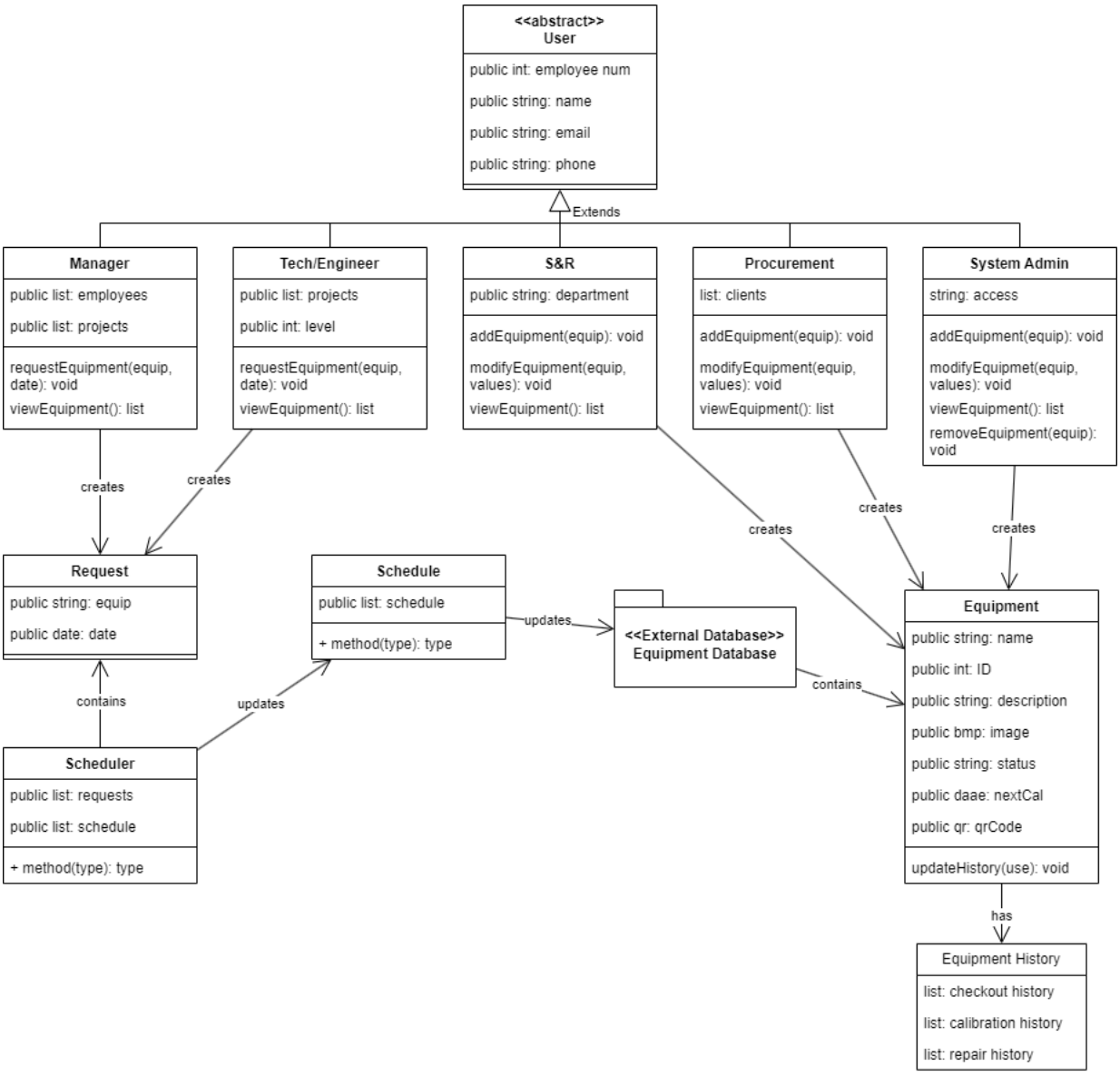
2.1 UML Use Case Diagram



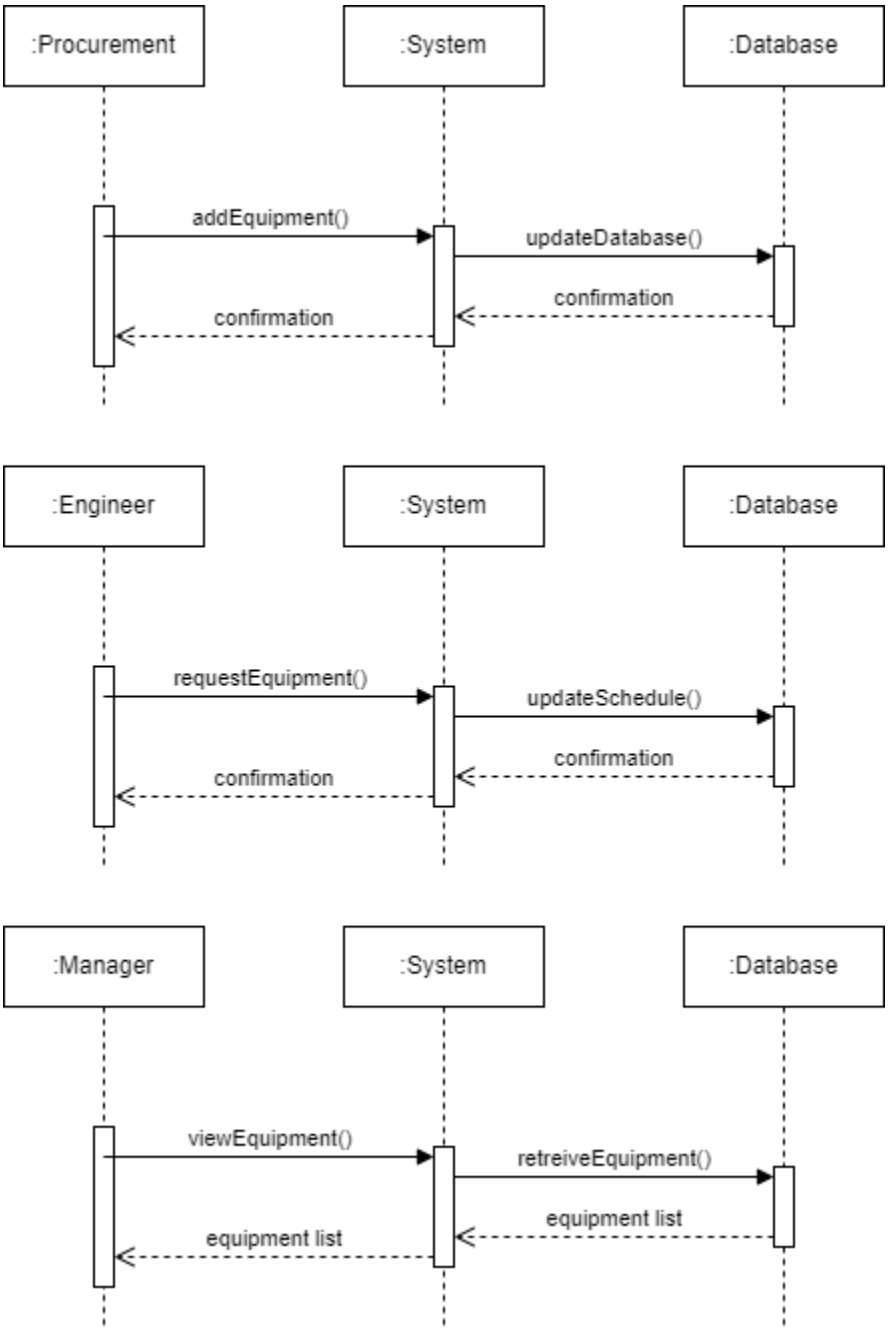
2.2 UML Domain Model



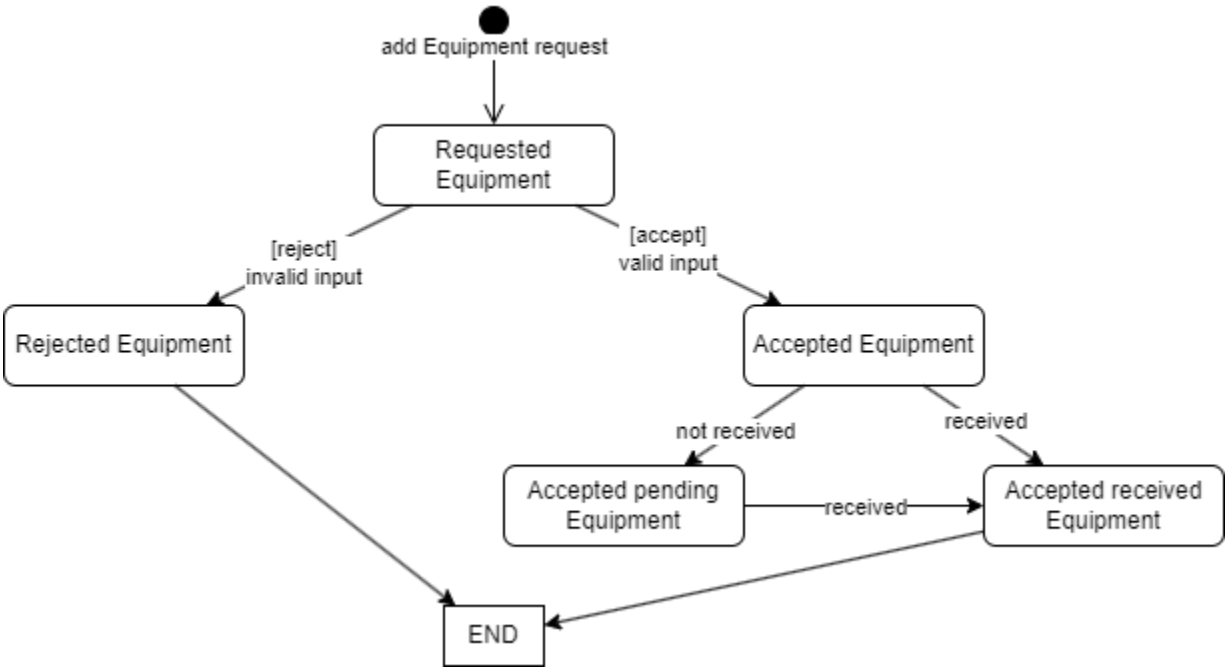
2.3 UML Class Diagram



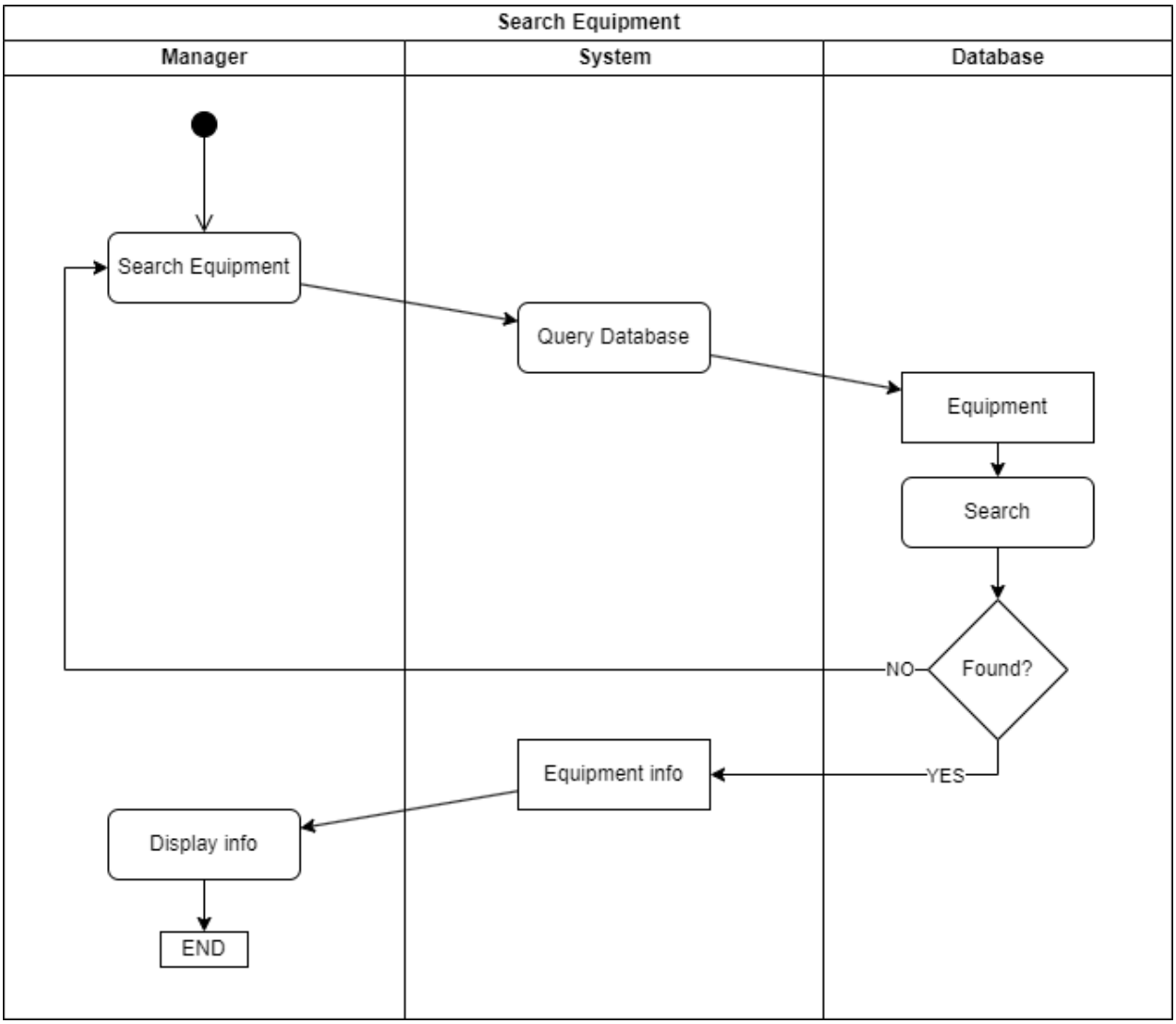
2.4 UML Sequence Diagram



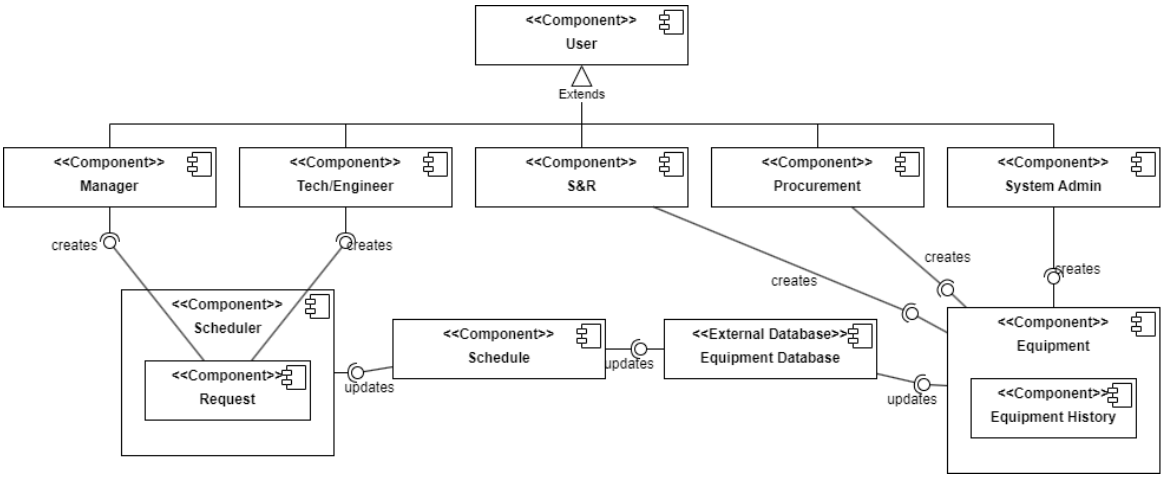
2.5 UML State Diagram



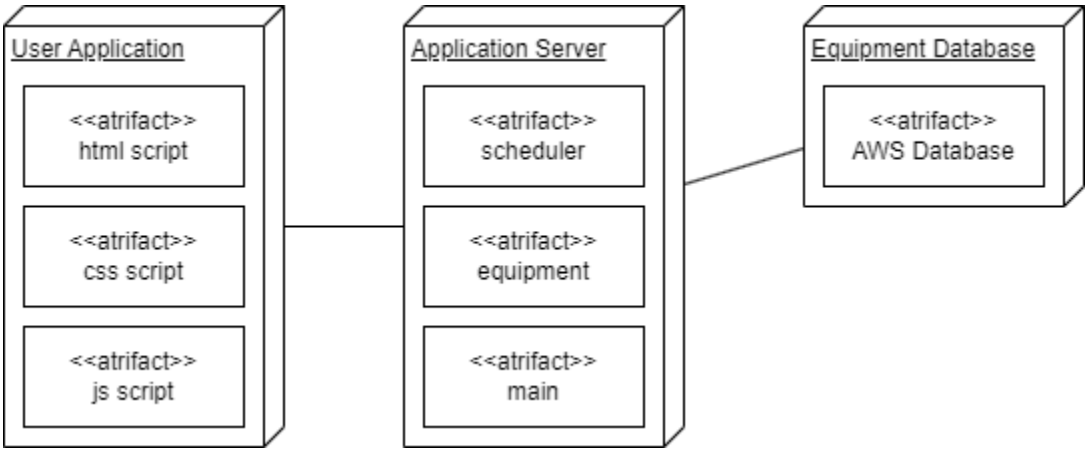
2.6 UML Activity Diagram



2.7 UML Component Diagram



2.8 Cloud Deployment Diagram



2.9 Skeleton Classes and Tables Definition

Manager class:

Attributes: employees, projects

Methods: requestEquipment(), viewEquipment()

Engineer class:

Attributes: projects, level

Methods: requestEquipment(), viewEquipment()

S&R class:

Attributes: department

Methods: addEquipment(), modifyEquipment(), viewEquipment()

Procurement class:

Attributes: clients

Methods: addEquipment(), modifyEquipment(), viewEquipment()

System administrator class:

Attributes: access

Methods: addEquipment(), modifyEquipment(), viewEquipment(), removeEquipment()

Equipment class:

Attributes: name, ID, description, image, status, nextCal, qrCode

Methods: updateHistory()

Database table:

Equipment, ID, status, schedule, project, location, person checked

2.10 Design Patterns

Many great design patterns are used in this project, and many can be used during development to ensure the software solution is easily maintainable and scalable. One example in this design is Single Responsibility Principle (SRP) shown in the Scheduler class. This class takes in data and adds it to the schedule where applicable. Another design pattern used is polymorphism. This is seen in classes such as the manager and engineer classes both calling viewEquipment() for example. These can be treated differently as the manager class would have access to view more equipment on projects that an engineer might not be on. This design also follows the Don't Repeat Yourself (DRY) principle in the instance of having a single database that all classes access and modify from.

3 Conclusion

This software solution solves the problem of the mismanagement of equipment and resources within a company by streamlining and consolidating the item's locations and updates. As a result, this tool could also be used for auditing purposes and organization of physical products. Data received from this tool over time can also show trends where a company can then take action to automate or correct any slow actions.