Software Requirements Specification

for

<Inventory Manager>

Version 1.0 approved

Prepared by <Rojitha Horawala>

<Team Name: Ultimate-Destroyers>

<09/22/2020>

Table of Contents

Conventions and Abbreviations:

- The Two Layer:
    - Used when referencing the Manager and Employee Layer

Prototype Link:

- https://xd.adobe.com/view/ce1de73b-91e4-42ff-a726-0c1adb65f717-9b4e/

1. Introduction

1.1 Purpose

- Our application software is created Android Lollipop 5.0 via Android Studio. An older level foundation was used so that older versions can utilize the applications. This will benefit even smaller establishments that have yet to update their devices. Our program is a management application that will be conventionally used by inventory workers to send out the item count for their workplace.

1.2 Document Conventions

- We did not use any remarkable abbreviations or conventions in this document.

1.3 Intended Audience and Reading Suggestions

- Document is intended for the individuals in a company's branch that may make decisions for what tech to implement in their workplace. Furthermore any company's developer who wishes to licence this application and wishes to make adjustments for their application such as images, implementation of company logos, or even additional pages for their work place can also refer to this document to reach a greater understanding of the software.

- Within this document you will find the specific details and specifications that have been implemented in our final product, along with, various other implementations, design patterns, and ideas that have been adjusted or abandoned during the development cycle.

1.4 Product Scope

- The main platform that is used to create this project is Android Studio due to its flexibility towards development. The application built on this software will simulate inventory management between the Manager of a given company and its employees. The app serves a form of communication in the workplace and for the Managers perspective, to gather information of which items would need certain management.
- An example of this simulation would be in a restaurant in which the management of the inventory of a certain item gets used more than others over a course of time. Then that information gathered to determine if more of the known item should be invested and to determine which product to cut out if it does not meet a certain quota.

1.5 References

- https://youtu.be/USenYOBJw9Y (Video used to learn and create the fragments.)
- https://www.youtube.com/playlist?list=PLrnPJCHvNZuDQ-jWPw13-wY2J57Z6ep6G (Additional Fragment Research)

- https://www.youtube.com/playlist?list=PLrnPJCHvNZuBtTYUuc5Pyo4V7xZ2HNtf4 (Log list)
- https://www.youtube.com/watch?v=sJ-Z9G0SDhc (Filtering and Searching)
- https://www.youtube.com/watch?v=BWjGQIlkgT4 (Networking and Sockets)

2. Overall Description

2.1 Product Perspective

- This product is not a member of a follow up or a product family, as it is a stand alone application. In terms of other requirements; the application is dependent on an android device of android version 5.0 or greater. As stated in 1.1, we chose this version for the accessibility of facilities that choose not to update their software.

Two Major Components of Program:

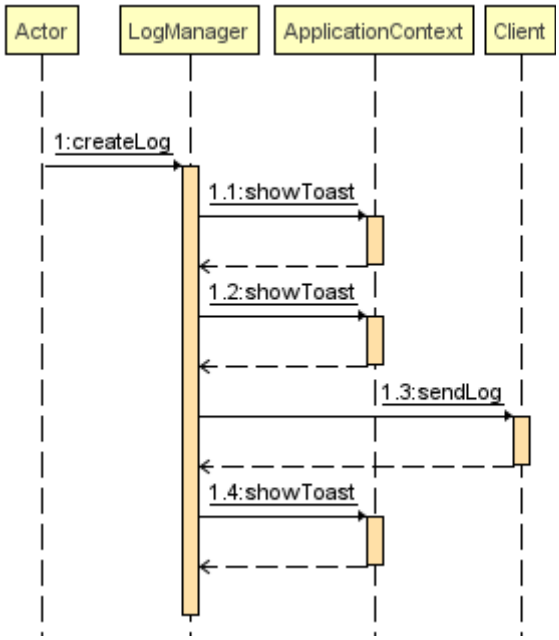
- **Log Interaction Across Employee and Manager layers:** An item log will be given a Type, name, and minimum amount. This is an object created with the properties of a fillable log that an employee gets to complete and send back. Essentially, the contents of
- **Server implementation:** For the practicality of a manager and employee interaction we assume that these two will not always be together to transfer information manually. Therefore, we rely on this server client to communicate with so that different interfaces can receive this information, and require a manual save from the server client to save and send it though.

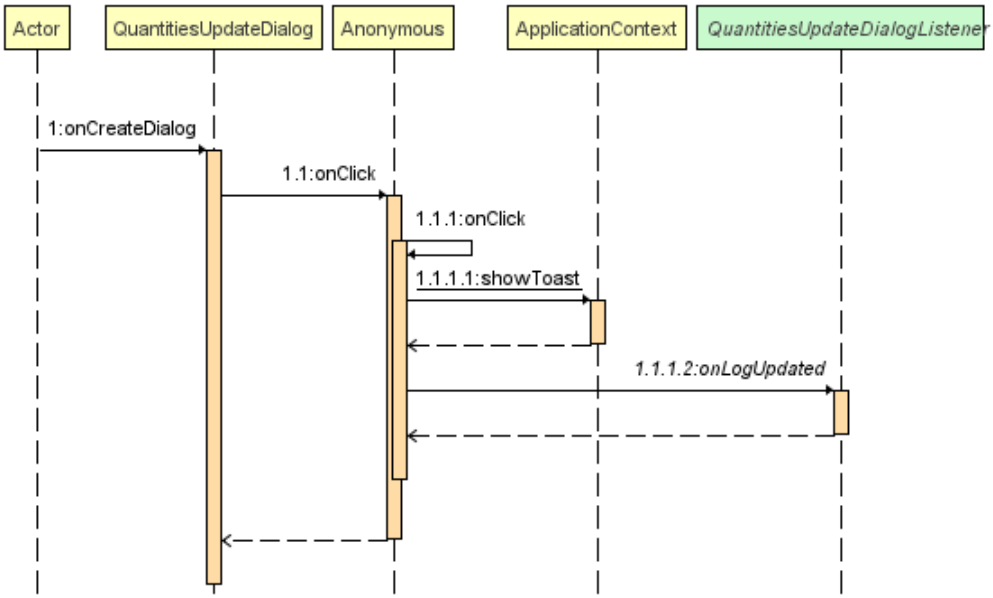
2.2 Product Functions

Our product provides the user to fulfill multiple duties depending on their role as an employee or manager in their respective company or organization. In general, the user can create, view, and manage logs and manipulate the data within those logs. For the Manager side, he or she can create a new type of log consisting of a minimum amount of the item requested. The type is an indication of what category the requested item comes from.

- Employees are to report inventory quantity and indicate from the report which item is either consumed frequently or not in frequent use.
- Manager Creates and Set New Logs of different types to request and check on item space
    - Types being the category of which an item comes from (ex; Type:Kitchenware, Item: Spoons)
- UML (Full Diagram next page)

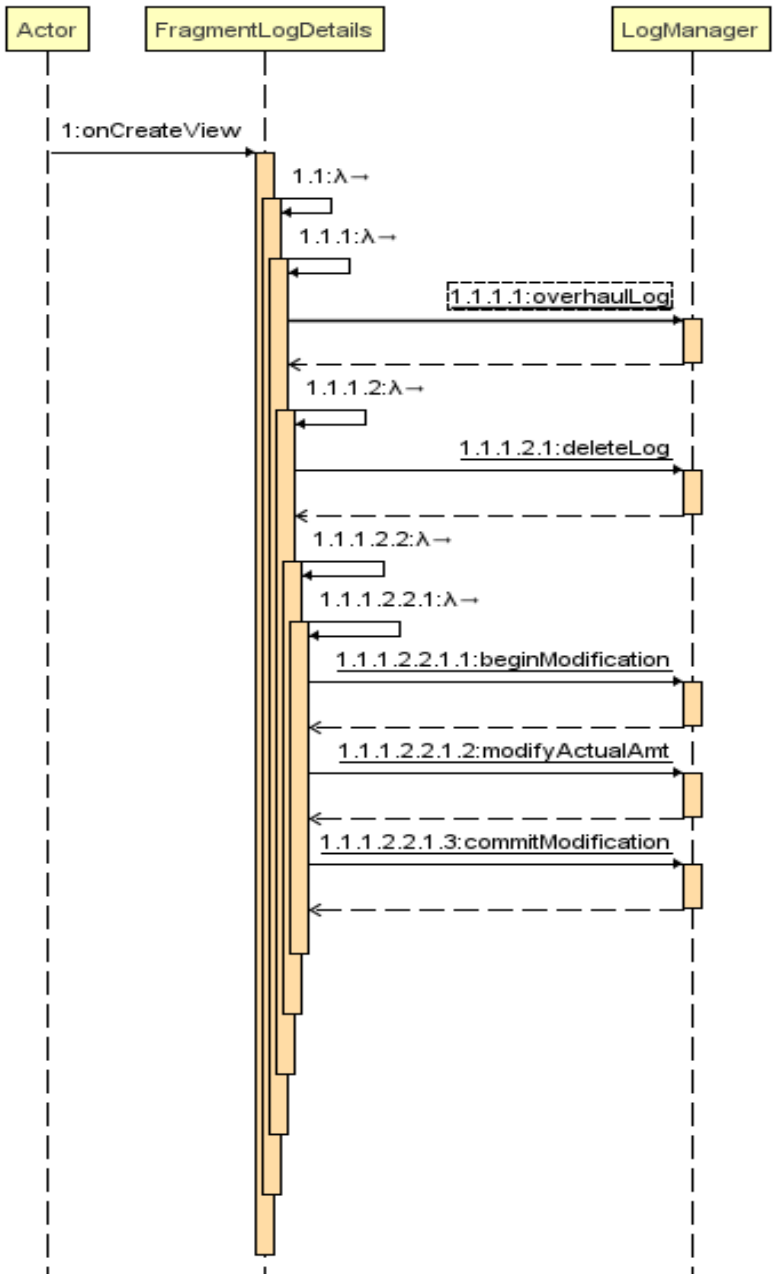
MANAGER LOG DIALOG:





EMPLOYEE LOG DIALOG:

SERVER LOG INTERACTION:



2.3 User Classes and Characteristics

We anticipate that anyone that works with multiple products will use this application. Users ranging from hardware stores to restaurants to the hospitals could possibly rely on an application like ours for manageability of all materials required in these fields. The most important user classes for this product would probably be those that aid in the improvement of life, like a hospital, rather than a restaurant.

Classes:

- Log Management
- Login
- Can edit log
- Quantities Dialog
- Employee
- Employee login
- Item Log
- Manager
- Manager Fragment to Set log
- Manager Login
- Application Context
- Log Details
- Quantities Editor
- Application Details
- Main Activity
- Client

Actions the user can perform in the application.

Manager

- Controls Employee's accounts
- Adds/Deletes logs
- Approves/denies log received from employee?
- Can edit logs (in real time).

Employee

- Received logs assigned from manager.
- Complete's logs and approves from for resend.
- Can search through logs.
- Can delete logs.

2.4 Operating Environment

The software will operate on Android Lollipop 5.0. The hardware projected to be for this application to be used in any Android supported device, phone or tablet would be ideal in an active inventory check. Original compilation was conducted on Android Supported Emulations such as the Google Pixel emulator that runs on Android Lollipop 5.0, then Switched to a Emulation that uses Android 10.0 to run the app built on Android 5.0. This will also use the java net server to complete data it's data transfer operation.

2.5 Design and Implementation Constraints

Use of Android Studio may limit developers in terms of creativity and problem solving due to the minor unfamiliarity with this programming software. In terms of constraints from the nature of our projects we needed to ensure that the Employee's tasks were presented in an intuitive manner. Because we are unable to perform actual beta testing on people outside of our development team, we lack the ability to get user feedback and tweak to the application.
WE USED:

- Fragmentation.
- Allocation.
- Android Studio and Android emulator
- Server and Database utilization for data transfer(Java NET).
- Reading and writing to files(Java IO).
- LinkedList and Queue

2.6 User Documentation

Content within readMe file:

Login: Page dedicated for the employee's and managers to access their designated account.

Employee:

- *Home:* Home page for the Employee.
- *Edit:* Actual job of employee: log recording.
- *Search:* Employees can search though completed logs.

Manager:

- *Home:* Where the manager is first exposed to the side bar which will allow them to navigate through the menus they need.
- *Login*: Login page for both Employee and Manager.
- *Search:* Searches for a log by name or date.
- *Set Logs:* The log creation only accessible to the manager. This function creates a new log where the manager creates the log name and the type of log it belongs to i.e. a spoon or a fork would belong in 'type: utensil'. Additionally the manager would also write in a minimum amount for said item.
- *Manage Logs*: Actual completion of logs possible by both Employee and Manager.
- *Full List:* Page will provide a list with the item names and the quantity of that item. This page will update the quantity when a new log is sent in from an employee detailing a change in the number of said items.
- Server class This class is integral for the transfer of data and utilizes Java net and Java dedicated servers for the transfer.

2.7 Assumptions and Dependencies

We assume that Android Studio will provide the features needed to bring our vision to life. If this assumption is incorrect, the final product may not function as originally intended.

The Internet is needed to send log data between Manager and Employee.

It is assumed that our application will be secure -- manager and employee login info will be safe. If this assumption is incorrect, data stored in the application may be at risk.

There is an assumption in regards to the creation of a log from manager to the employee and as well as sending a completed log from the employee to the manager.

3. External Interface Requirements

3.1 User Interfaces

Interface -

Employee:
- Can see Log details.
- Can edit only current stock of logs.
- Search logs.

Manager:
- Two windows (all logs and create logs)
- Create logs:
    - Can establish details of a log (Type, name, current stock quantity and minimum quantity). With a button to actually create the logs.
- All logs
    - Once the logs will be displayed in the “All Logs” window for both the employee and the manager.
    - In “All Logs” you can search logs, delete logs, edit the entire log and view log details.

Much of what

3.2 Hardware Interfaces

The supported device types for this software are any Android devices that can support Lollipop 5.0. If possible, we plan on providing support for iOS devices as well.

3.3 Software Interfaces

Data items coming into the system are what the user inputs, whether that is the employee or the manager. What comes out of the system are the logs that are created from the user's inputs. These logs are shared across the software so that the manager and employee can access them.
Our data sharing mechanism was implemented with the use of java.net's Socket feature. With the socket feature, we can establish endpoints for the data to be sent. Using this method can allow us to communicate with different machines/devices, in the likely event that employees and managers are using different devices, this will prove useful. Java also being our group's most known language along with our unfamiliarity with networking, we found this the easiest way to implement this feature. Database all provided by Java to complete our cycle.

3.4 Communications Interfaces

Data will be transferred on a server. The majority of communication between the two parties will be in the form of our logs, which will be transferred with the server. Otherwise we mean Android's warning and safety features that we can use to prevent the user from inputting something that will harm the process or system.

4. System Features

4.1 System Feature 1

Log Manager Class

4.1.1 Description and Priority

- About mid priority, this will provide significant help with replaying information between the two parties, but is there manual and less efficient ways to do this.

4.1.2 Stimulus/Response Sequences

- **Manager side and Manager interaction**: Process on how one cycle with the majority of the important functions are used: The employee would first create a log with all the necessary types and limit numbers and will assign this log to the employee to complete. When the employee completes this log, the employee will receive it. If this item is below the minimum amount then this item will be flagged so the manager can acknowledge and order more otherwise this will be received as normal.

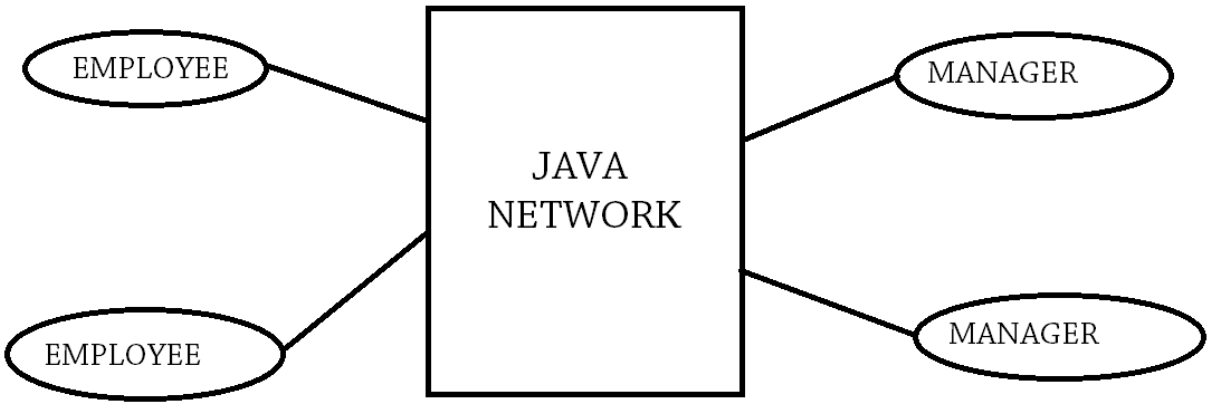
- **Employee side and Employee interaction**: This side is what actually records information and records if something actually below the needed amount. Necessary for half of a program's cycle to complete.
- **Client interaction**: This client makes it important for information to be transferred with manual saving with inputted information.

4.1.3 Functional Requirements

- An instance of discrepancy in this application can be identified if an employee inputs a value that is greater than the previously imputed value in a previous log for the same item. This would cause a problem when considering that things need to be re-ordered to suffice the workplace. To combat this, we consider an approach that can look at the value that was inputted in the previous value and set error returns to reject an in inputted value that is greater than the previous value.
- As some items may be an individual count, others may come in bunches or boxes. Because of this, we need to to input a set number of counters to make sure that the employee can manage

Client/Server System and Database usage:

- The client is how the user is able to interject the information between the two layers. Despite the interaction, everything within the server class happens on the back end with the user only receiving minimal functions to send the information over. This is the case with both layers.



4.2 System Feature 2 (and so on)

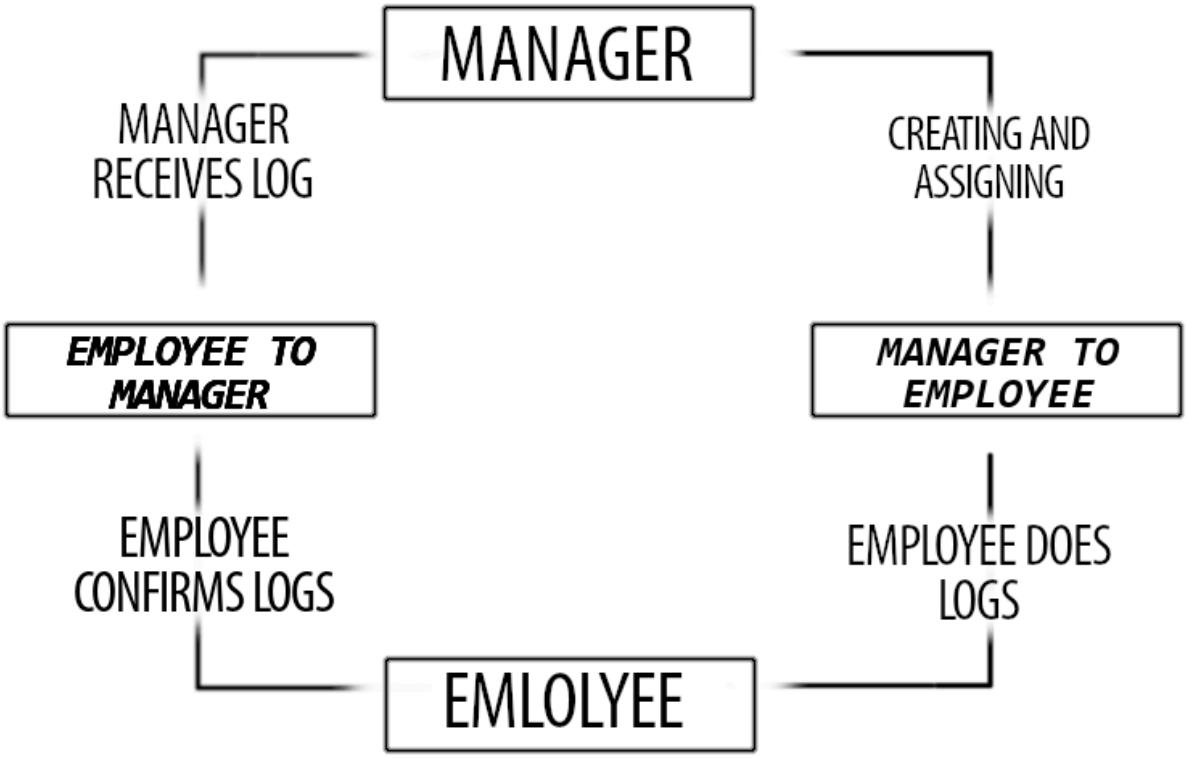
Java Net: Middle Priority

- While this feature is a very important feature for relaying information between layers and completing the cycle of our program, it is not a feature critical to our programs ability to function.

5. Other Requirements

5.1 Performance Requirements

Put program functionality in a form that can be understood



5.2 Safety Requirements

One of the concerns with the data logging is originating with human error, in which a user can over or underestimate the quantity of an item. Also to avoid the same multiple logs from overlapping, a log would disappear from any employee's side the moment any employee completes it. Values inputted in logs are set so only numeric values can be inputted.

5.3 Security Requirements

Managers and employees are required to have login credentials in order to use the application. These login credentials may not be secure and thus create potential risk for hackers to target the data stored in this application.

5.4 Software Quality Attributes

We want our application to be easy to use so that anyone can use it. Usability and flexibility are some important characteristics that we want our application to meet. As for our version build, we wanted to ensure that most devices are able to run the application without any issues.

- **AVAILABILITY:** Employee completes only the logs that manager has completed and sent to the employee.
- **CORRECTNESS:** Notifications given to make sure logs are complete correctly.
- **MAINTAINABILITY:** Manager would maintain the cycle of the logs and employee's, thus maintainability is dependent on the manager's decisions.
- **USABILITY:** Logs should properly inform the manager of the current and needed products.

5.5 Business Rules

Employees are limited to adding items to a log, editing those items, and view logs. Managers can perform all functions that an employee can do, as well as create new logs and even delete items. (**add to this later if necessary**)

6. Other Requirements

- Nothing other than an android device and two parties to send data and perform the programs intended function.

7. Tasks

Unit Testing

Welcome:

- Welcome Screen -> Login (Employee) : Works. Presents home screen.
- Login (Employee) -> Employee home: Works. Presents home screen.

- Welcome Screen -> Login(Manager): Works. Presents home screen.
- Login(Manager) -> Manager Home: Works Presents home screen.

Slider:

- Employee: Works -> displays All Logs page.
- Manager: Works -> Displays All Logs and Set Logs page.

Employee Slider:

- All logs -> Displays List of current logs
    - Log -> Works: Has the ability to view the log.
    - Log edits-> Works.has the ability to adjust only the current stock amount as intended.
    - Log search-> Works: logs searchable by Name and Type.

Manager Slider:

- All log -> Displays List of created logs and can.
    - Log edits->
    - Log search-> Works: logs searchable by Name and Type.
- Set logs-> Works: Send to logs creation page.
    - Create log-> Displays log properties for input and creation. Created log viewable in both employee and
    - Log deletion-> Works: Deletes across both layers.
    - Log Detail -> Type and Name work with allowing characters and numbers.
    - Log Details for quantity-> Works: Only allows numerical input as intended.

Real Time Display.

- When logs have been edit the logs will display the change the moment they are changed. This feature works and can be viewed on either the manager or employee's "All Logs" page.

Server:

- Transferring Data-> Works: Data between layers is updated and communicates with the java server when !save is submitted on the server side.

Solid Principle

Fortunately, we did not have to refactor much of our code, as most of what was created followed the principle. An example of how we followed the Single Responsibility Principle is followed in how we set the log editor in one class, the search is in another, etc. Our fragments are constructed in a manner where it is extended from the parent fragment class to receive the properties for implementation of other features. Thus the classes are not modified in order to implement completely new fragments and this follows the Open/Closed Principle. For Liskov's Substitution Principle, we set our logs to require input for its typing, name, amount. As such, we extend the fragmentation class to our logs in order to get it's behavior to our logs. To implement the Interface Segregation Principle we make our interfaces use each layer would have similar properties to fragments making extensions. Finally, to adhere to the Dependency Inversion Principle, we use the Serializable interface on the ItemLog class for load/saving and send/receiving logs.

Fasade

Our application is primarily a back end application with the user not having access to much of the server and data storage features. The cycle of our application begins with the manager creating a new log. When the user does this, we create an object with the properties for the logs and send them to our employee with the server client saving in the console. A significant complexity that is hidden from our users is the data transfer. The user is not given any dialog or notification indicating that the data is being written and sent via Java sockets internet. Because of this, our program can be friendly for a user and it allows us to keep our front end and UI simple.

Backlog/ Design Specification.

<u>Settings Page</u>

- Font Size adjusting.

- Button for access on other pages.
- Dark mode.
- Notifications/Push notification.

Login Page

- Password encryption for account.
- Designated IDs.
- Customization.
- Deadlines for completion.

Employee Page

- Log Filtering by date/recent/quantity.
- Log confirmation before sending.
- Customization: Employee picture, employee quota, etc.
- Direct sms contact with manager's or additional note messages that can be sent between users.

Manager Page

- Log Filtering.
- Log confirmation before sending.
- Employee Management.
- Log Approval/Deny.
- Creating employee's.
- Assigning different tasks for individual employees.
- Restrictions for making logs have a required stock of unreasonable numbers.
- Direct sms contact with manager's or additional note messages that can be sent between users.

Server:

- Automatic data saving without console use.
- More warning to layers incase server faces issues.

Design Specification.

This Program's architecture is created to transfer data in between two layers. It is practical in a field with more finite items and quantities since the logs are designed with the behavior to keep track of inventory supply. Data transfer occurs with the information being saved and sent with the !save command on the console. We connect the two layers as endpoints to send and receive the data. This cycle would repeat until all data is transferred or inventory stock is recorded to the user's contempt.

Appendix A: To Be Determined List

<Collect a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.>

- *Adding Settings Page*