**Design Document**

A picture containing text, warehouse, orange, stack

Description automatically generated

**Tiger Dam International Flood Control Inventory System**

**Group Members**

Dylano Van Der Meer, Joshua Law, Trung Hieu Tran, Daniel Wong, Rajdeep Sangha

**Client**

Trevor Wright

**Instructor**

Ali Moussa

Date

09-29-22

[**Preface** 5](#_Toc121497306)

[**Purpose of Document** 5](#_Toc121497307)

[**Documentation Standards** 5](#_Toc121497308)

[**User Requirements** 5](#_Toc121497309)

[**Business Overview and Objectives** 5](#_Toc121497310)

[**Project Overview** 6](#_Toc121497311)

[**Statement of the Problem** 6](#_Toc121497312)

[**Project Scope** 6](#_Toc121497313)

[**System Environment** 7](#_Toc121497314)

[**Current System** 7](#_Toc121497315)

[**System Requirements** 8](#_Toc121497316)

[**Fact-Finding Methodology** 8](#_Toc121497317)

[**Use Case Diagram** 9](#_Toc121497318)

[**Use Case Descriptions and Scenarios** 9](#_Toc121497319)

[Use Case 1: Fulfill order 10](#_Toc121497320)

[Use Case 2: Create Order 10](#_Toc121497321)

[Use Case 3: Create User 11](#_Toc121497322)

[Use Case 4: Create Item Registry 11](#_Toc121497323)

[Use Case 5: Show Data 12](#_Toc121497324)

[Use Case 6: Update Data 12](#_Toc121497325)

[Use Case 7: Delete Data 13](#_Toc121497326)

[Use Case 8: Log out 15](#_Toc121497327)

[Use Case 9: Analyze statistics 15](#_Toc121497328)

[**Non-Functional Requirements** 15](#_Toc121497329)

[**System Interface Requirements** 16](#_Toc121497330)

[**Maintainability and Administration Requirements** 16](#_Toc121497331)

[**Usability Requirements** 16](#_Toc121497332)

[System Design 22](#_Toc121497333)

[Layered Architecture 22](#_Toc121497334)

[Hardware Architecture 22](#_Toc121497335)

[Software Platform 23](#_Toc121497336)

[Interaction Model 23](#_Toc121497337)

[Persistence Model 51](#_Toc121497338)

[Class Diagrams 52](#_Toc121497339)

[Interaction Sequence Diagrams 53](#_Toc121497340)

[**Project Management** 64](#_Toc121497341)

[**Schedule** 64](#_Toc121497342)

[**Team Configuration** 64](#_Toc121497343)

[**Members:** 64](#_Toc121497344)

[**Roles** 65](#_Toc121497345)

[**Reporting relationships** 65](#_Toc121497346)

[**Contact information(Emails):** 65](#_Toc121497347)

[**Project Standards and Procedures** 65](#_Toc121497348)

[**Glossary** 66](#_Toc121497349)

[**Index** 67](#_Toc121497350)

# **Preface**

## **Purpose of Document**

The purpose of this document is to analyze and log the requirements of making an for the community flood assistance company called “Tiger Dam.” This document aims to document the team's approach to development. The information will include the functional/non-functional, admin, maintenance requirements along with a milestone schedule for the project. The document will also provide a use case diagram accompanied by a brief essential description of each scenario.

## **Documentation Standards**

* **Fonts:** We will be using “**Calibri light**” to make the documentation readable, simple, and professional, font size .11
* **Colors:** We will be using a monochromatic color scheme for this document to keep the text clean. Using proper bolding and italic etiquette to emphasize the importance of topics.

# **User Requirements**

## **Business Overview and Objectives**

Tiger Dam™ is completely reusable, and is the ONLY system patented to connect together to create an impervious barrier for miles, in any shape (circles, 90 degrees turns etc.) and is the ONLY system that is stackable, from 18 inches to 32 ft. in height.

Tiger Dams™ are 1/2 the price of sandbags on their first use and protect our environment with no environmental damage. Tiger Dams™ are a water filled bladder technology and when deployed properly, this system may be able to divert up to 100% of floodwaters.

Our System can be assembled within minutes using floodwaters or any water source. Tiger Dams™ fill in as little as 90 seconds with minimal manpower and no heavy equipment. Each 24” Tiger Dam weighs 85 lbs dry and 12,000 lbs when filled with water and replaces around 1000 sandbags. Plus, they roll up for easy storage after each deployment to be used repeatedly.

# **Project Overview**

## **Statement of the Problem**

Tiger Dams International Flood Control is a company that provides flood control utilities. The current inventory management system is inefficient and out of date. The employees must use pen and paper to manually log inventory. These employees need as automated method that saves time, enabling them to fulfill and ship orders more efficiently. To solve this problem, we have chosen to create an application as an effective way to help manage his inventory. This well create a more centralized area for tasks to be completed and for information to be stored that will make it accessible for people that will need access to it without having to worry about it being leaked.

## **Project Scope**

The Tiger Dams Inventory management system is a dynamic inventory system that will allow the user to update and keep track of inventory. This system will consist of a simple and efficient interface while allows for better insight into inventory and orders. The system will store data by taking in CSV file as order inputs which will be processed so it can then export orders and inventories. The system will allow multiple users access to the data. Users can check order history and associated inventory as well as allowing users to update, change, or override orders, they also have access to fulfill orders fully or partially. Finally, users can export the data from the database in the system back into a CSV file for further use. Clients can then get a CSV file with their order information. Order will be able to be changed after they are processed this means that the system must be dynamically mutated by the user.

* + - Build an inventory system for users to update and keep track of inventory
  + System generates a simple and efficient interface
  + System generates insights on both inventories and orders
  + System processes CSV file as the order input
  + System stores and processes input orders
  + System allows access for multiple users
  + System allows export of orders and inventories
  + User has access to order history and associated inventory
  + User has access to update, change, or override orders.
  + User has access to fulfill orders fully or partially
  + User should be able to export data from the database into a csv file for further use
  + Client will often get CSV’s files with order information; the system should be able to read the file and upload the data to the database
  + Orders sometimes need to be changed even after the order is processed, the system must be dynamically mutated by the user

## **System Environment**

* The system will be a web-based system, the system will live offsite on the Company’s server or a personal server.
* The user will be able to access the system via any device that has access to the web application
  + Mobile devices
  + Computers
  + Laptops

## **Current System**

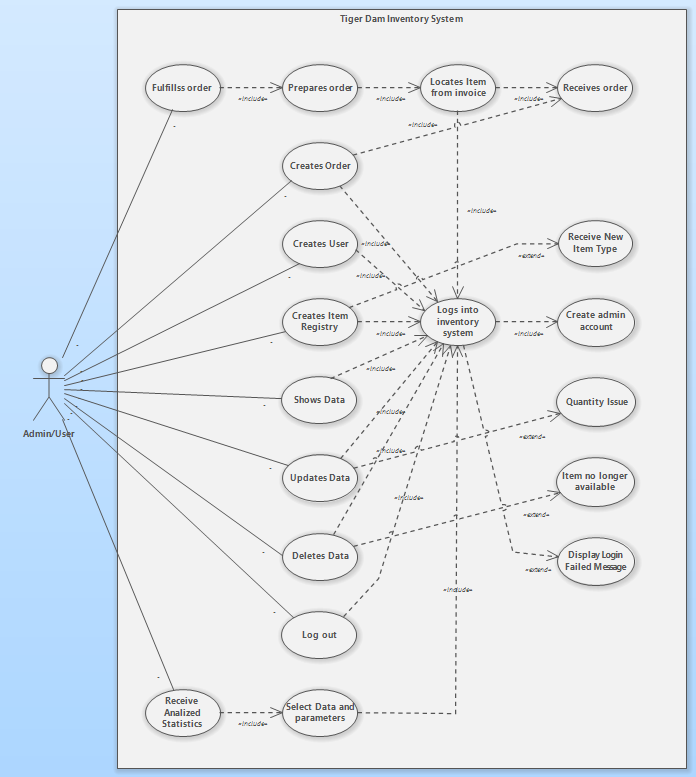
The current system being utilized by the client is an impartial system that relies on excel generated files that are used as orders. As for inventory, the client uses a method of written data that is used to check quantities. The Client then proceeds to fulfill the order based on the inventory available, if inventory is available and the order is fulfillable, the client generates a CSV style invoice and stored both the invoice and the order in his local domain, if the order is partially fulfillable, the client will contact the customer and generate needed changes to the order.

# **System Requirements**

## **Fact-Finding Methodology**

* We interviewed our client via a call.
* Our first approach was enabling our client take the stand and walk us through a typical day at the warehouse, as he described his day, we would take notes and think of problems and solutions he faces
* One of the questions we asked was, “would you want the system to be accessible from more than a local computer.” and “Would you benefit from being able to edit all aspects of the data/orders.”

## **Use Case Diagram**



## **Use Case Descriptions and Scenarios**

* Actor role definition: Manager role is granted to all users of the application. Their role is to manage inventory which consists of orders, items and other user accounts to ensure they are created, updated, maintained and displayed properly.

### Use Case 1: Fulfill order

**Precondition:** Manager has an account; Manager is successfully logged in; Dashboard is displayed; Manager received an order  
**Postcondition:** Order is shipped  
**Limitation:** Manager can abort any operations and return to dashboard

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| Manager selects either:  1 Item Icon from Dashboard (Alternate Flow 1.1) |  |
| -or-  2 Items Page (Alternate Flow 2.2) | 3 Item Description is displayed |
| 3 Manager locates item storage location and prepare order |  |
| 4 Manager fulfills order | 5 Item quantity deducted from database |

|  |  |
| --- | --- |
| Alternative flow 2.1 | 2.1.1 Display Items page with search text fields (name, sku, etc.) |
| 2.1.2 Manager enters item name, sku, etc. | 2.1.3 Display filtered list of items |
| 2.1.4 Manager select item icon from list | 2.1.5 Item Description is displayed |

### Use Case 2: Create Order

**Precondition:** Manager has an account; Manager is successfully logged in; Dashboard is displayed; Manager received an order, order file included  
 **Postcondition:** New order is added  
 **Limitation:** Manager can abort any operations and return to dashboard; Orders file can only be in .csv format

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1 Manager selects Orders page | 2 Display Orders page with all orders and button [Create order] on top |
| 3 Manager selects [Create order] button | 4 Display Create order menu with file attachment field |
| 5 Manager insert orders file (.csv format) | 6 System processes order |
|  | 7 Order added and displayed; Alert box fades in with message “Order created” |

Use Case 3: Create User

**Precondition:** Manager has an account; Manager is successfully logged in; Dashboard is displayed  
 **Postcondition:** New user is added  
 **Limitation:** Manager can abort any operations and return to dashboard; Password must have at least 8 characters including at least one capital letter and a number

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1 Manager selects Users page | 2 Display Users page with all users and button [Create user] on top |
| 3 Manager selects [Create user] button | 4 Display Create user menu with text fields for email and password |
| 5 Manager inputs email and password | 6 System processes user |
|  | 7 User added and displayed; Alert box fades in with message “User created” |

Use Case 4: Create Item Registry

**Precondition:** Manager has an account; Manager is successfully logged in; Dashboard is displayed; (Optional) New type of item received  
 **Postcondition:** New item registry is added  
 **Limitation:** Manager can abort any operations and return to dashboard; Item must contain at least name and sku.

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1 Manager selects Items page | 2 Display Items page with all items and button [Create item registry] on top |
| 3 Manager selects [Create item registry] button | 4 Display Create item registry menu with text fields: name, sku, quantity, price CAD, price USD, storage location |
| 5 Manager inputs into text fields (at least name and sku) | 6 System processes item |
|  | 7 Item registry added and displayed; Alert box fades in with message “Item created” |

Use Case 5: Show Data

**Precondition:** Manager has an account; Manager is successfully logged in; Dashboard is displayed  
 **Postcondition:** Selected data is shown  
 **Limitation:** Manager can abort any operations and return to dashboard

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| Manager selects either:  1 Orders page (Alternative flow 1.1) | 2 Display Orders page with all orders and search field for order id |
| 3 Manager enters order id in the field to search | 4 System processes search |
| -or-  5 Items page (Alternative flow 5.1) | 7 Display filtered list of data |
| -or-  6 Users page (Alternative flow 6.1) |

|  |  |
| --- | --- |
| Alternative flow 5.1 | 5.1.1 Display Items page with all items and search field for item name/sku |
| 5.1.2 Manager enters item name/sku in the field to search | 5.1.3 System processes search |

|  |  |
| --- | --- |
| Alternative flow 6.1 | 6.1.1 Display Items page with all users and search field for email/username |
| 6.1.2 Manager enters email/username in the field to search | 6.1.3 System processes search |

Use Case 6: Update Data

**Precondition:** Manager has an account; Manager is successfully logged in; Dashboard is displayed; (Optional) Occurs data issue/discrepancies inside database  
 **Postcondition:** Selected data is updated  
 **Limitation:** Manager can abort any operations and return to dashboard; Manager can select [Cancel] during data change to return to previously selected data records.

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| Manager selects either:  1 Orders page (Alternative flow 1.1) | 2 Display Orders page with all orders and search field for order id |
| 3 Manager enters order id in the field to search | 4 System processes search |
| -or-  5 Items page (Alternative flow 5.1) | 7 Display filtered list of data |
| -or-  6 Users page (Alternative flow 6.1) |
| 8 Manager selects the [Edit] button on the record that needs to be updated | 9 Displays Order, Item, or user update menu |
| 10 Manager changes data fields | 11 Data changed reflected on the screen |
| 11 Manager selects [Save] button | 12 Changes is processed |
|  | 13 Changes to data is saved and displayed; Alert box fades in with message “Order/Item/User updated” |

|  |  |
| --- | --- |
| Alternative flow 5.1 | 5.1.1 Display Items page with all items and search field for item name/sku |
| 5.1.2 Manager enters item name/sku in the field to search | 5.1.3 System processes search |

|  |  |
| --- | --- |
| Alternative flow 6.1 | 6.1.1 Display Items page with all users and search field for email/username |
| 6.1.2 Manager enters email/username in the field to search | 6.1.3 System processes search |

Use Case 7: Delete Data

**Precondition:** Manager has an account; Manager is successfully logged in; Dashboard is displayed; (Optional) Item no longer available/ User no longer needed/ Order redundancy  
 **Postcondition:** Selected data is deleted  
 **Limitation:** Manager can abort any operations and return to dashboard; Manager can select [Cancel] during data deletion warning to return to previously selected data records.

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| Manager selects either:  1 Orders page (Alternative flow 1.1) | 2 Display Orders page with all orders and search field for order id |
| 3 Manager enters order id in the field to search | 4 System processes search |
| -or-  5 Items page (Alternative flow 5.1) | 7 Display filtered list of data |
| -or-  6 Users page (Alternative flow 6.1) |
| 8 Manager selects the [Delete] button on the record of the data that needs to be deleted | 9 Display confirmation window with warning “Are you sure you want to delete [data]” |
| 9 Manager selects [Confirm] | 10 Data is deleted; Alert box fades in with message “Order/Item/User deleted” |

|  |  |
| --- | --- |
| Alternative flow 5.1 | 5.1.1 Display Items page with all items and search field for item name/sku |
| 5.1.2 Manager enters item name/sku in the field to search | 5.1.3 System processes search |

|  |  |
| --- | --- |
| Alternative flow 6.1 | 6.1.1 Display Items page with all users and search field for email/username |
| 6.1.2 Manager enters email/username in the field to search | 6.1.3 System processes search |

### Use Case 8: Log out

**Precondition:** Manager has an account; Manager is successfully logged in; Dashboard is displayed;   
Postcondition: Account logged out  
 **Limitation:** Manager can abort any operations and return to dashboard; Manager can select [Cancel] during log out message window to return to session.

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
| 1 Manager selects [Log out] button | 2 Display confirmation window with message “Are you sure you want to log out?” |
| 3 Manager selects [Confirm] button | 4 Invalidates session; Display log in screen |

Use Case 9: Analyze statistics  
 **Precondition:** Manager has an account; Manager is successfully logged in; Dashboard is displayed;   
Postcondition: Analyzed data displayed  
 **Limitation:** Manager can abort any operations and return to dashboard

|  |  |
| --- | --- |
| **Actor Action** | **System Response** |
|  | 1 Display [Low stock] and [Frequently Used Items] Panels as default |
| 2 Manager selects Analyzed Data tab | 3 Display Analyzed Data tab with all data panels; Displays fields for filter |
| 4 Manager selects wanted filters | 5 Display corresponding data tables |

## **Non-Functional Requirements**

* **Security:** The system requires users to create accounts to access sensitive data. There are two accounts, “users,” and “Admin.” Users have permission to perform basic functions of the inventory system like create invoice, find inventory, Check pricing, and print inventory list. Admin accounts have permission to create other admin accounts, modify permissions in user accounts, add items to inventory, and any user permissions are also permitted.
* **Speed:** This will determine how fast the application responds. Will use stress testing to determine efficiency of application. Application should be functional on different devices

## **System Interface Requirements**

The inventory management systems will only interface with programs that manage CSV files like excel, and the CSV files themselves. The system interface should allow the users to update, change or override orders. The interface should be able to let the user complete their orders partially or fully. There should also be something that will allow the users to export the data inside the inventory system.

## **Maintainability and Administration Requirements**

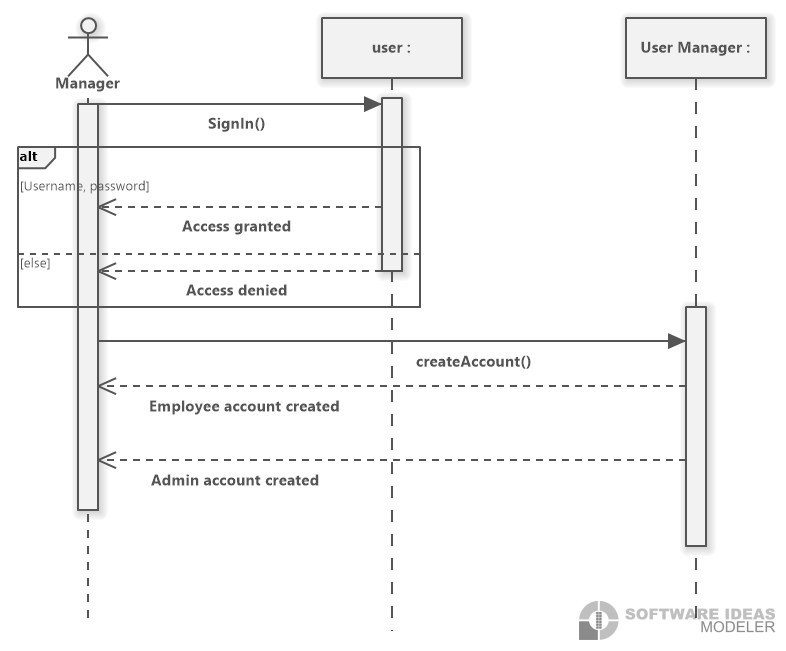
**Maintainability:** We plan to update our system to incorporate different inventory items after our initial implementation. We will also perform regular system maintenance like backing up the database and bug fixes.

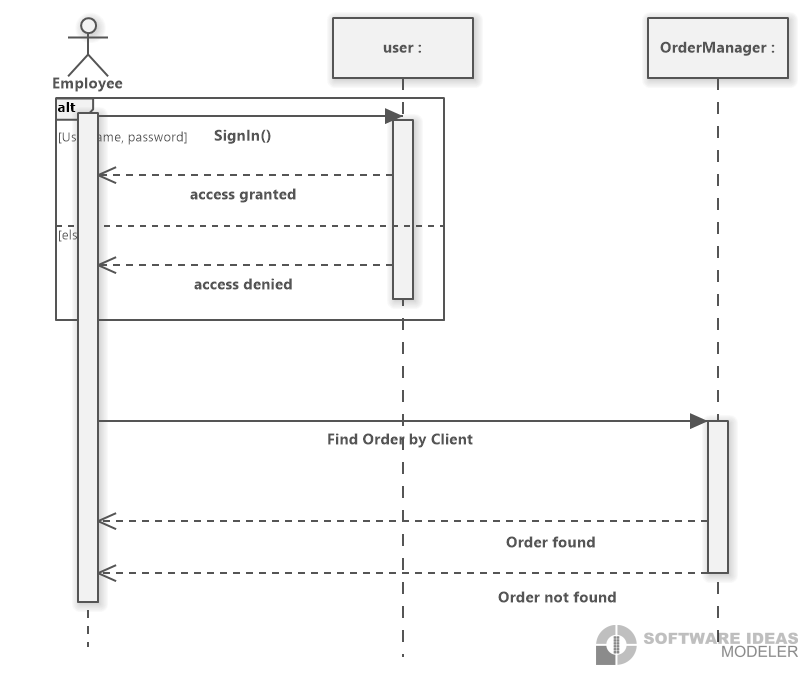
**Administration:** There will be two accounts for users on the system. The user account and admin account. The user account allows the employee to create invoices from existing orders and perform a total count of inventory. The admin account will have permissions to customize parts of the application such as adding or removing existing items to the inventory database and creating new admin accounts.

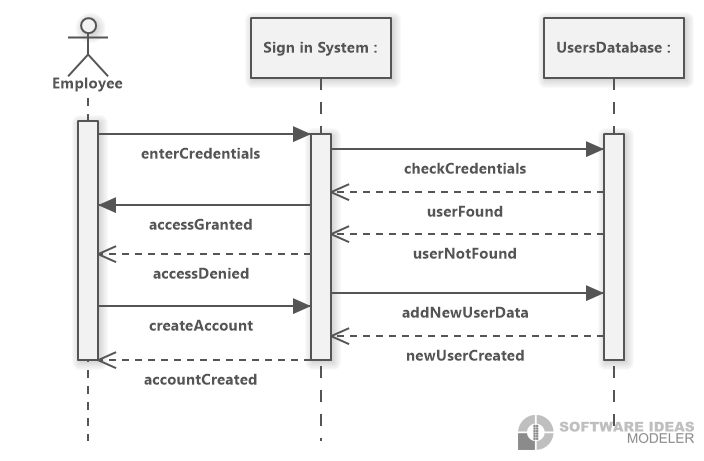
## **Usability Requirements**

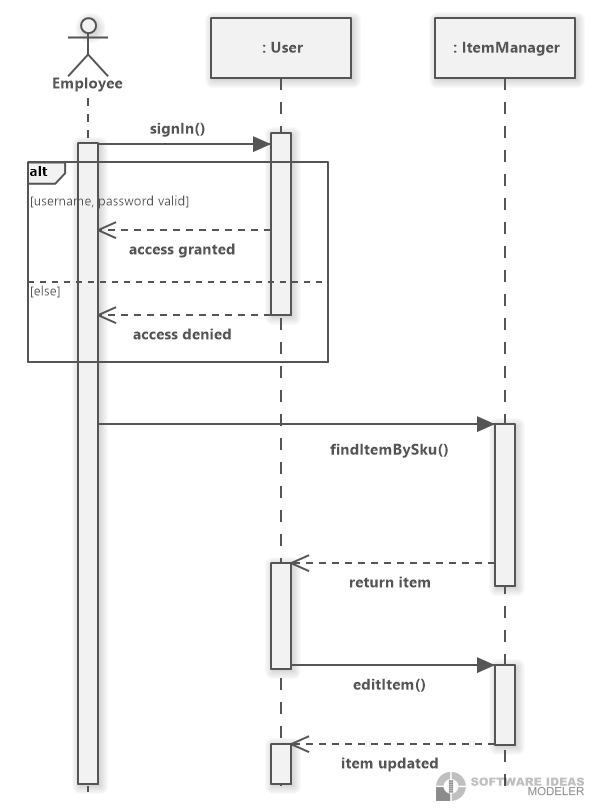
The users of the application will have little to no experience with managing a database. The GUI will be intuitive and uncluttered reducing the confusion for inexperienced users. Avoiding fancy designs and using simple monochromatic color schemes with larger buttons and font signifying importance of functions. We also plan to provide an instruction manual to thoroughly cover the basics of the inventory management system.

Interaction Sequence Diagrams





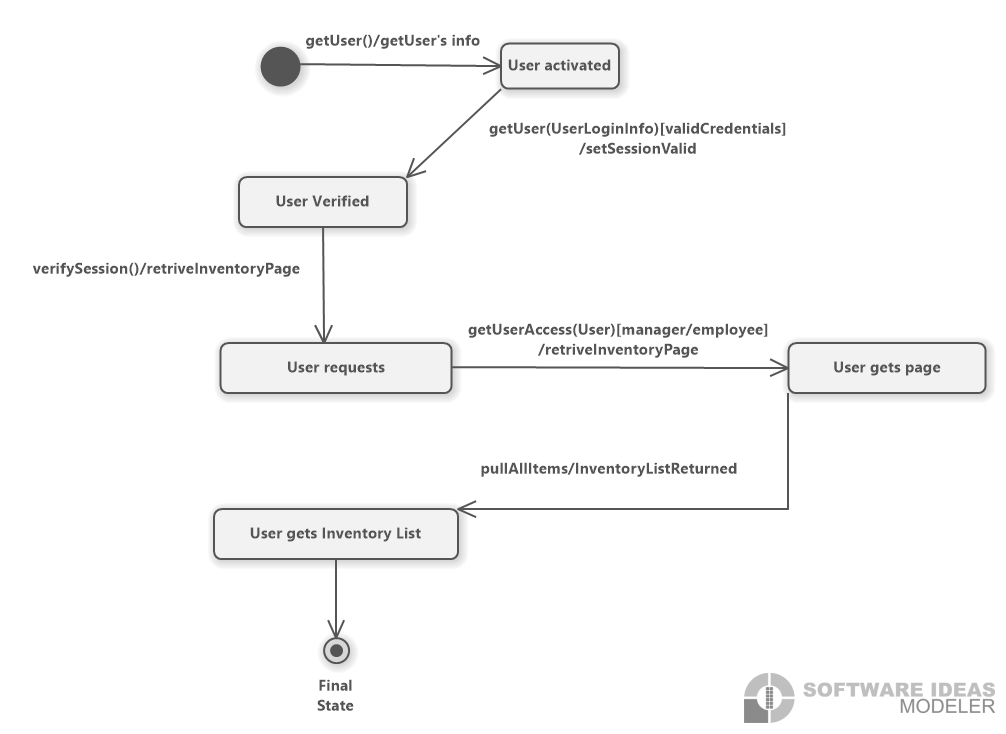




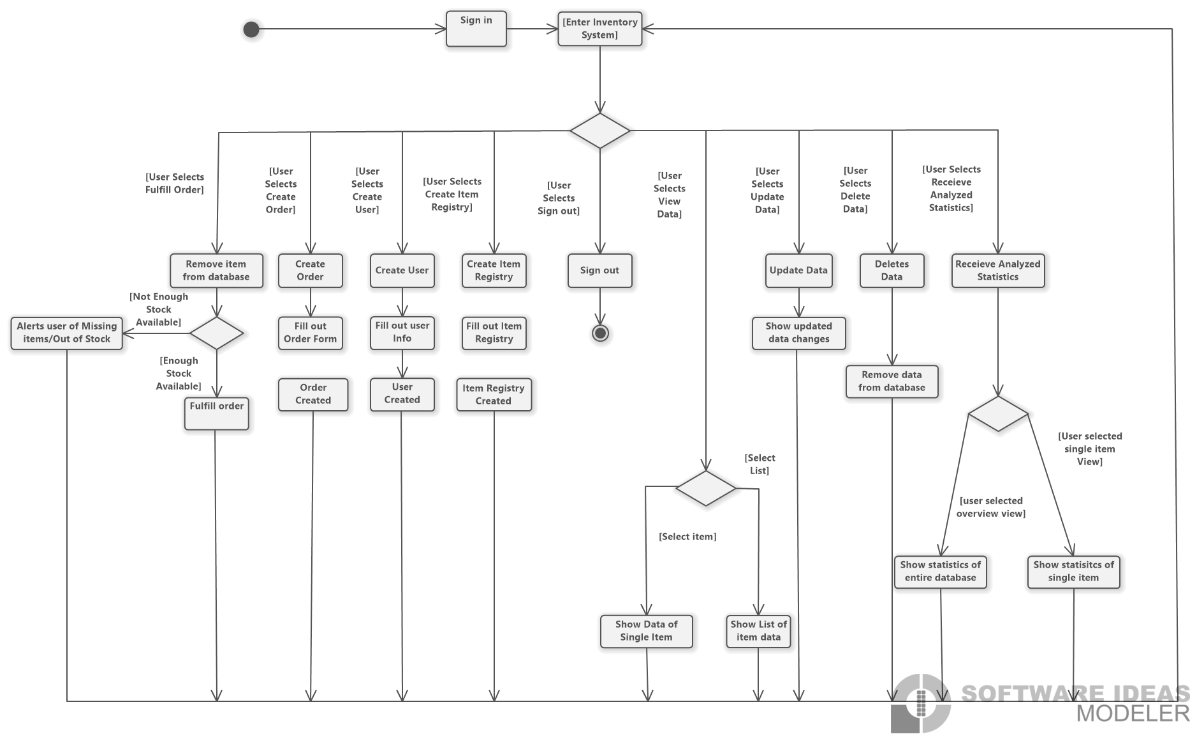
Diagram

Description automatically generated with medium confidence

State Machine Diagrams



Activity Diagrams



# System Design

## Layered Architecture

Diagram

Description automatically generated

## Hardware Architecture

* *The TigerDams inventory management system is a web application that will need to run smoothly on multiple users’ devices. Therefore, it is mandatory to have a server enough to handle multiple requests concurrently. (An exception will be during initial launch where traffic will be minimal.) The baseline requirements for the server hardware architecture will be an i5 Multicore CPU, a motherboard with dual 10Gbe and m.2 with NVMe and IPMI support, at least 500gb of storage, and 32gb or RAM. These specifications will allow the server to run at an optimal speed while allowing for the server capabilities to be scaled horizontally and vertically.*

Diagram

Description automatically generated

* *The main components of the web application will be the database, application, application server, and client Machine.*
  + *The database will communicate with the application through NodeJs Express. NodeJs contains an ExpressJs library that enables communication between database and the backend by simply installing the appropriate driver.*
  + *Within the application, the frontend and the backend must pass data between each other. We will be using ReactJS for the frontend and NodeJs for the backend. The main method of communication will be through JSON. The backend will use response and request parameters to pass data, while React will use the built in “Fetch” functionality to request and send data.*
  + *The Application itself will be hosted on the LAN server. That the client will provide. The communication between the application and the Web browser on the clients machine will be done through HTTPS.*

**Hardware Platform**

*The team is going to use laptops with i5 or i7 processors with installed RAM of 16 GB or 32GB with versions of windows 10 or 11 installed on the working machines for the development and maintenance of the application.*

*For the production system, AWS is the server that the application will eventually be deployed. The current prospective specifications for the production system is quad-core CPU with 16GB or 32GB RAM, whichever sits in the budget.*

## Software Platform

* *For the development and maintenance, we are going to use VSCode for the IDE because it will perform both front-end and back-end development which helps in the consistency. Another software service we are going to use is Git and GitHub for version control. We will be working with APIs (Application Programming Interface) and in order to understand and testing them Postmaster will help us.*
* *For the production system, we will be using Netsparker. As we are storing users’ data in our application database, there will be continuous threat for data leaks therefore Netsparker will help us*
* *by routinely scanning for security loopholes in the application*

## Interaction Model

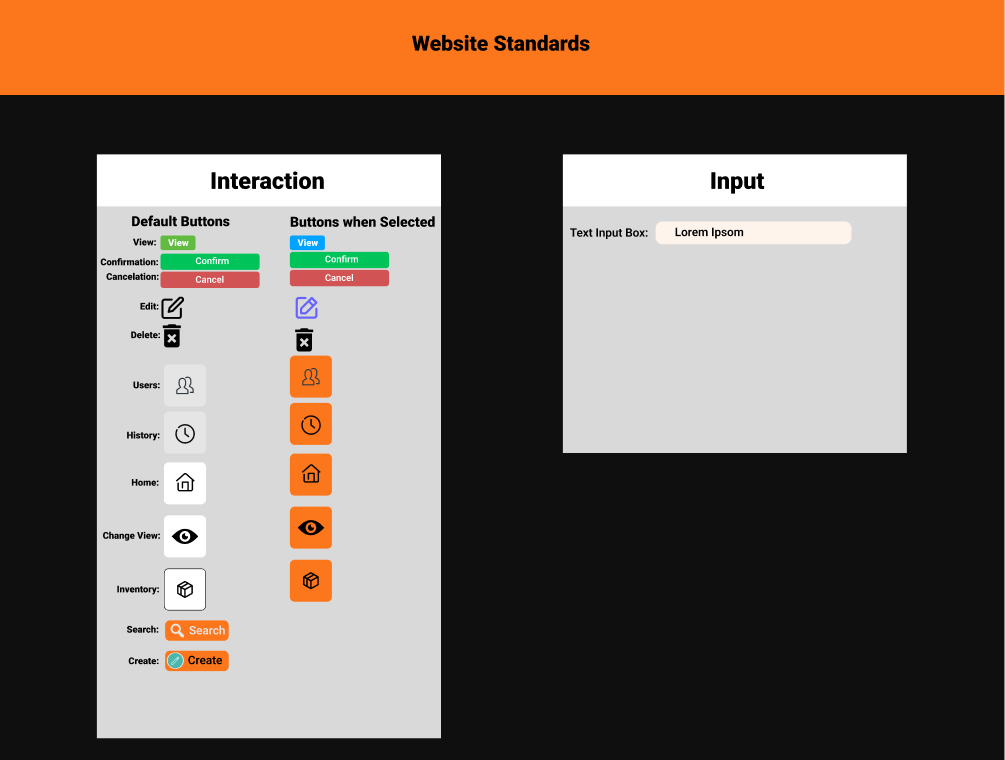
<https://www.figma.com/file/jL6FeaCYMV4Bu3WJUEqxvh/Final-Prototype?t=NUWh6G8prWfL0Bde-1>

How to go through the Prototype. The prototype starts at Frame 1

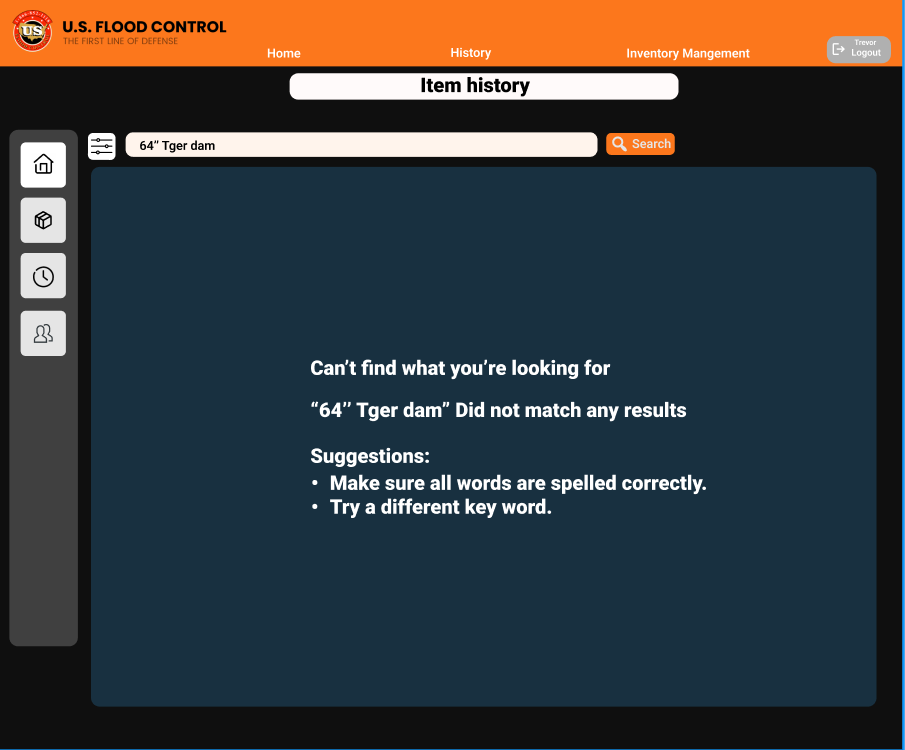
The prototype has all the standard things working like button to get to different screens and to get back and forth from different aspects of the prototype.

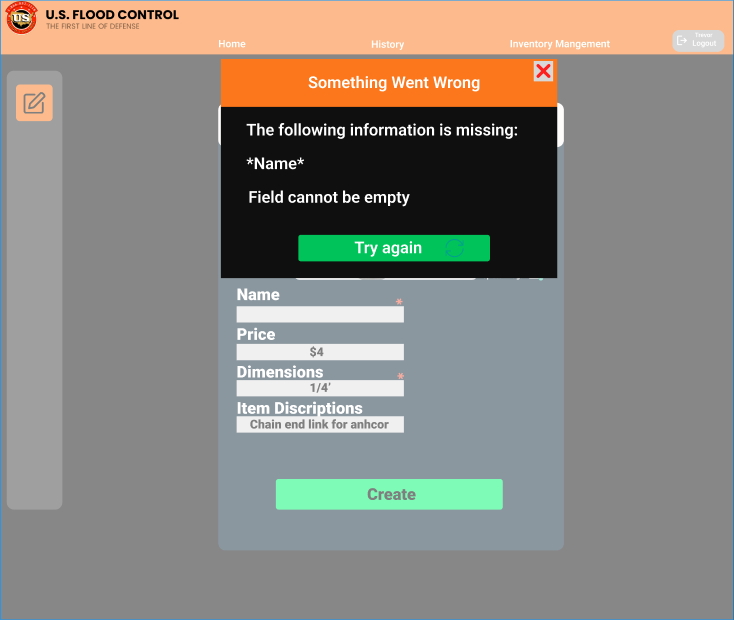
When click things like “Search” or “Create” the prototype will take you through the error screens first, for example when you click “Create” on the inventory page, it will take you to the item creation page where the item info is already filled out but one field will be left empty, once you hit create on the item it will show you the error screen and a button to continue, one you hit continue it will bring you back to the page with the error corrected this time and than you can proceed like normal. Most of the things you can do that we would want an error on will follow this process.

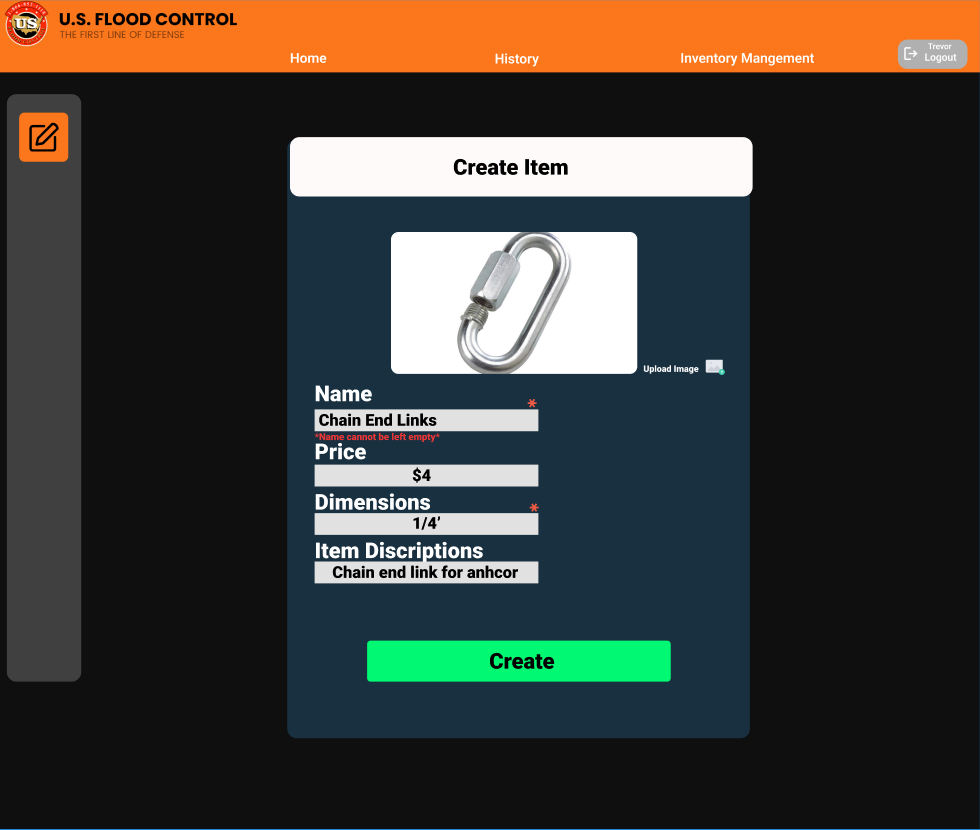
**Style for manipulation**

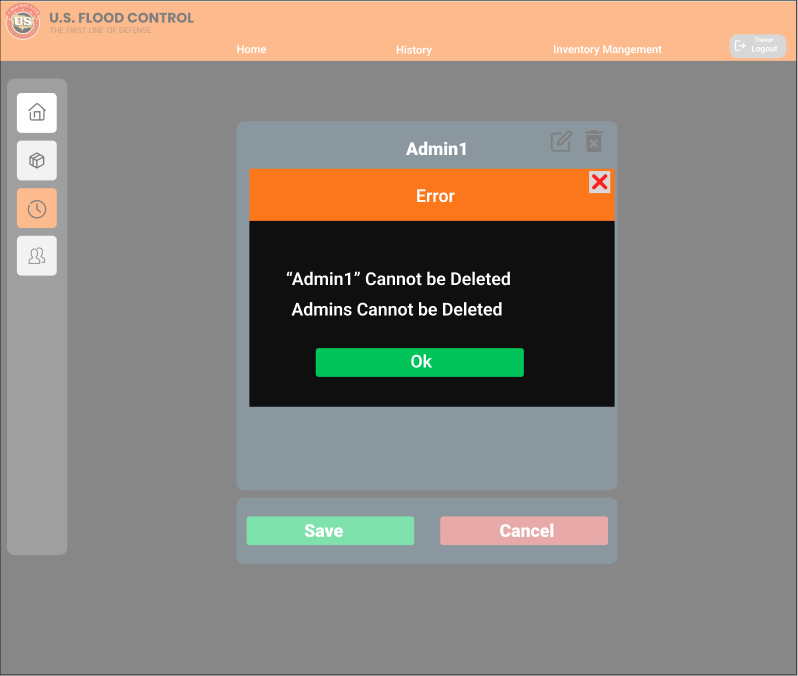


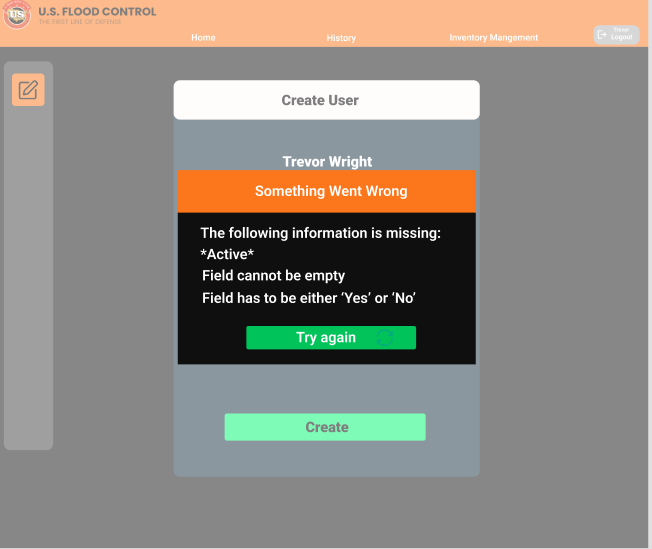
* ***desired user support***
* ***Help messages/Error messages when an user makes an error***

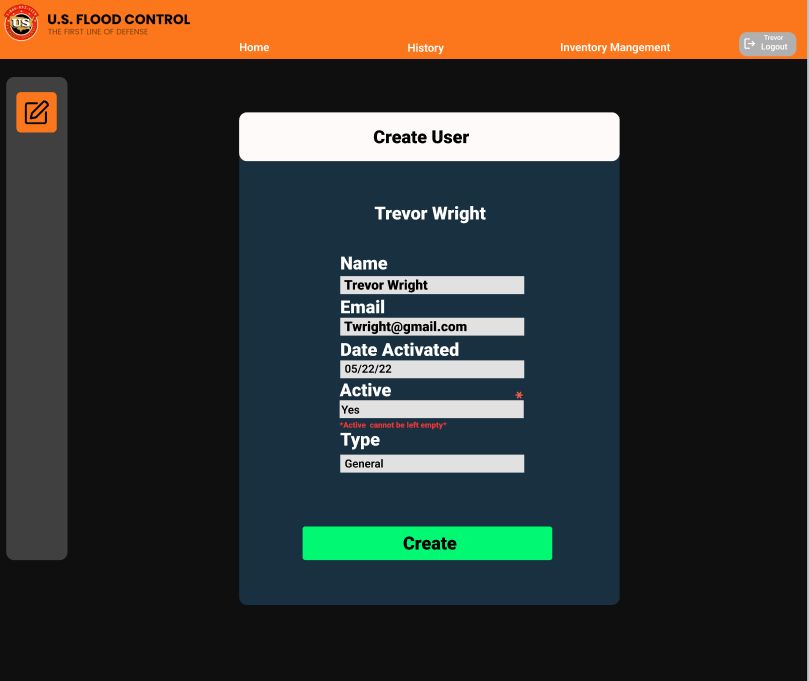




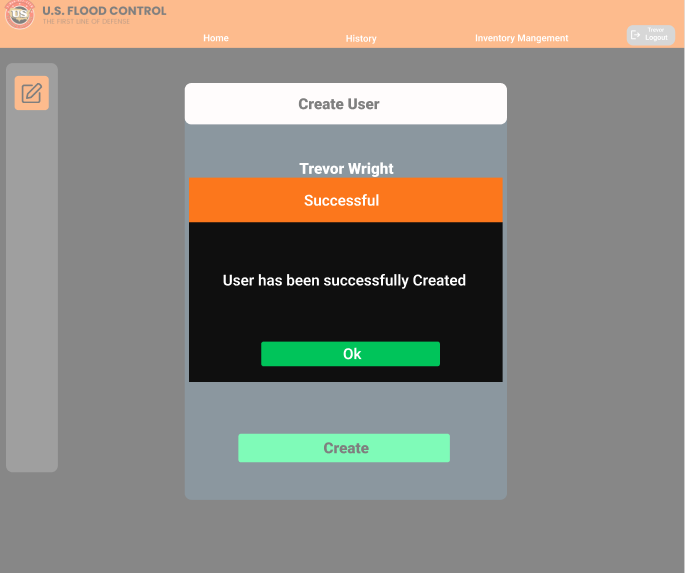


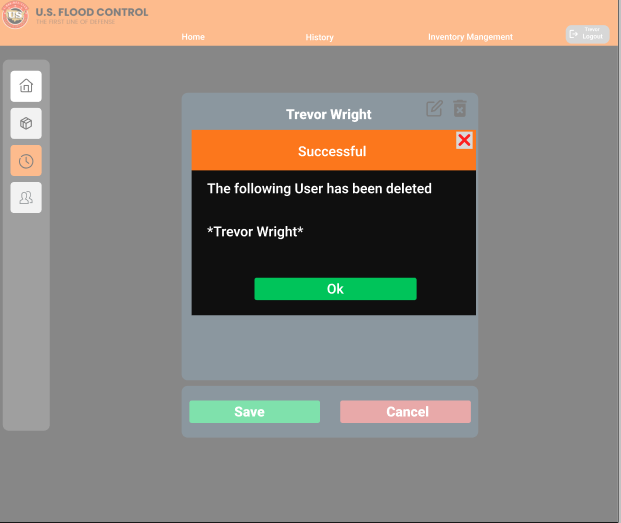


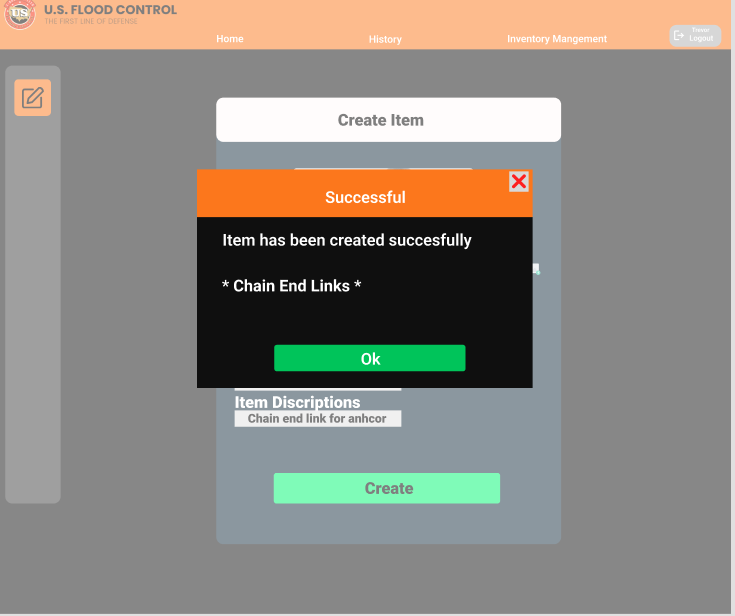




* ***system feedback style***
* ***Successful action screen***

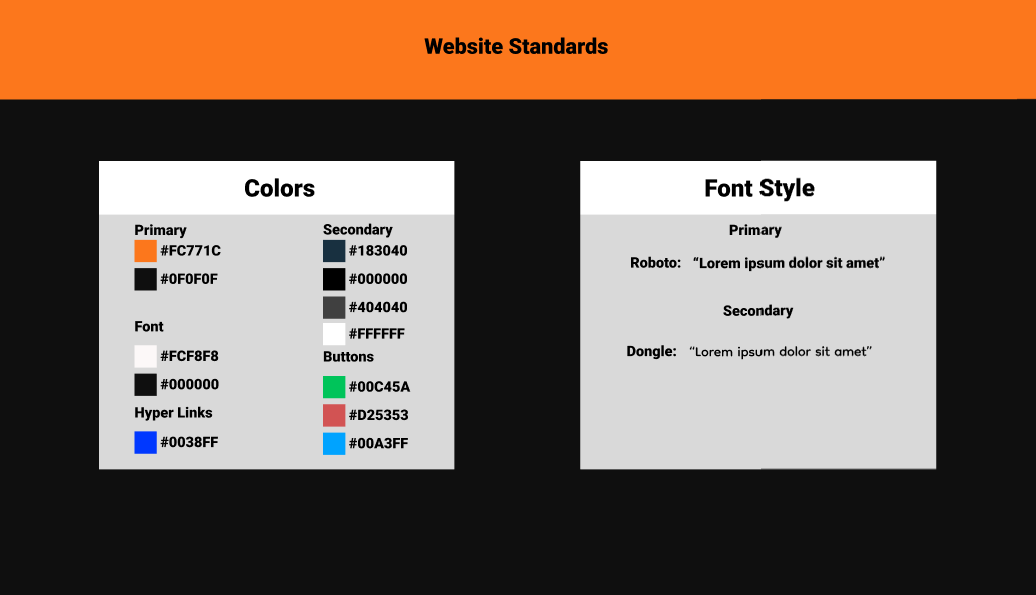






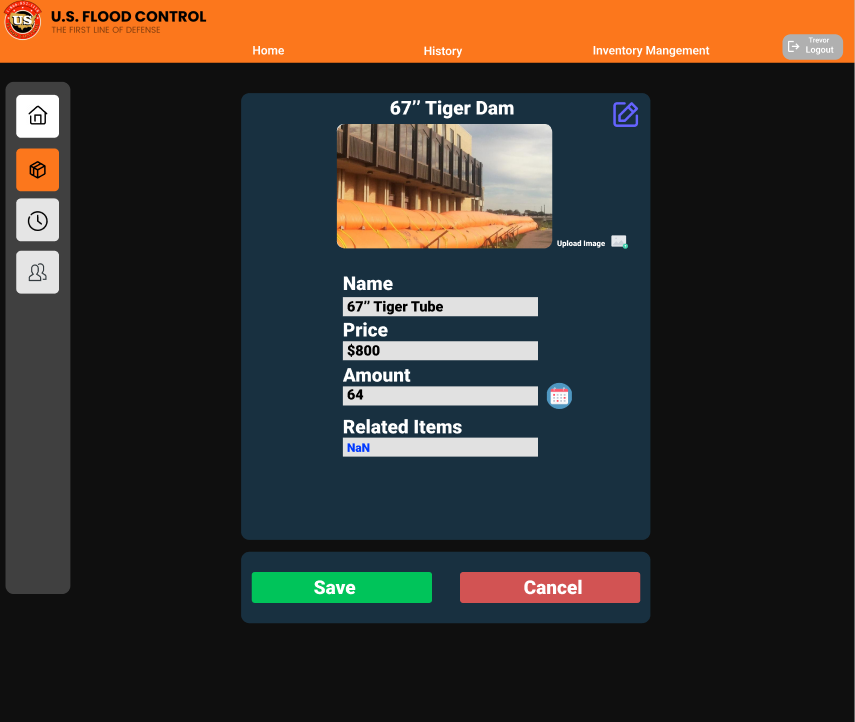


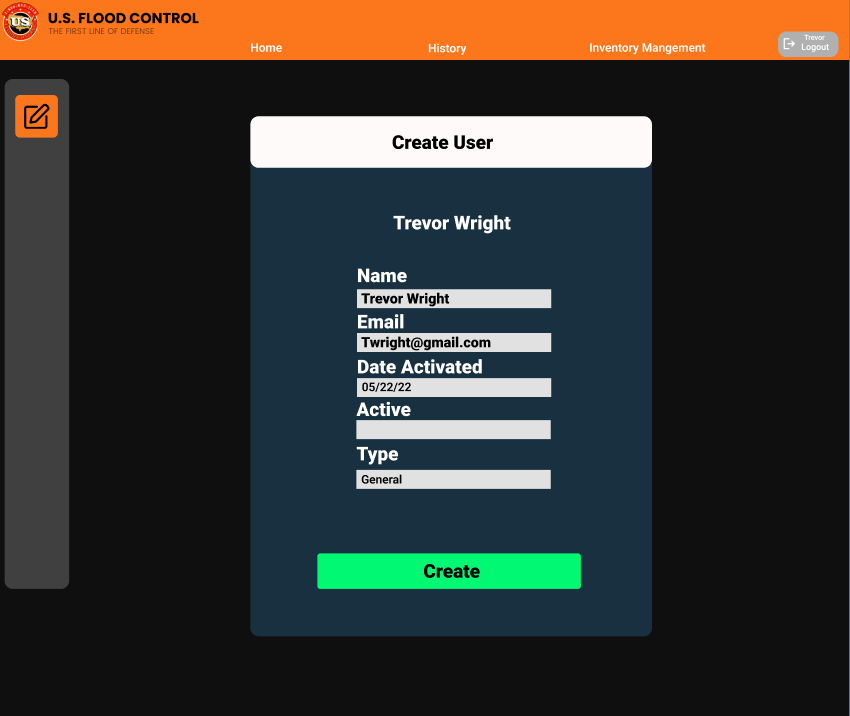
* ***Standards***

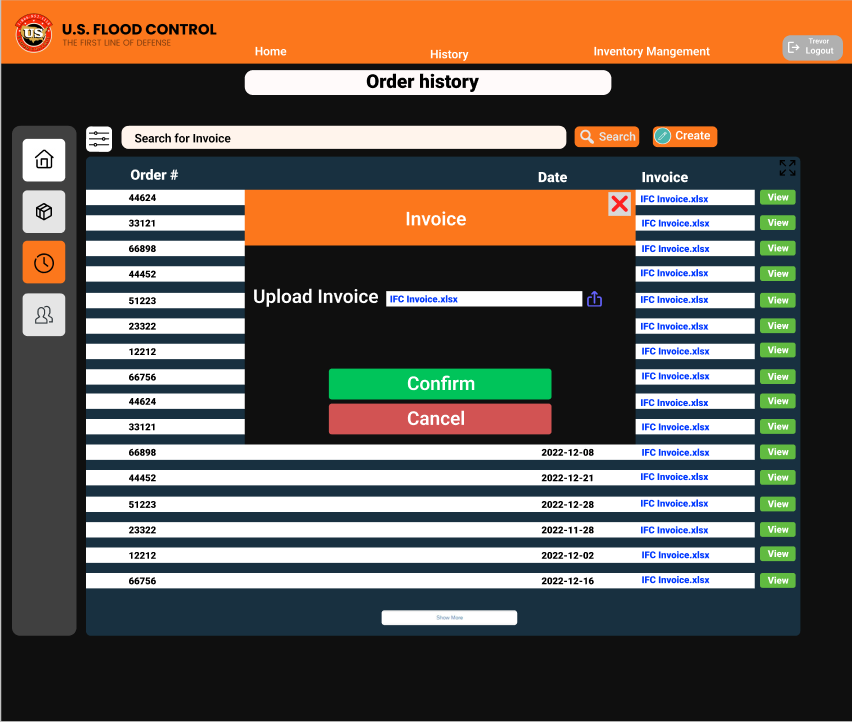


* ***Types of Interactions***

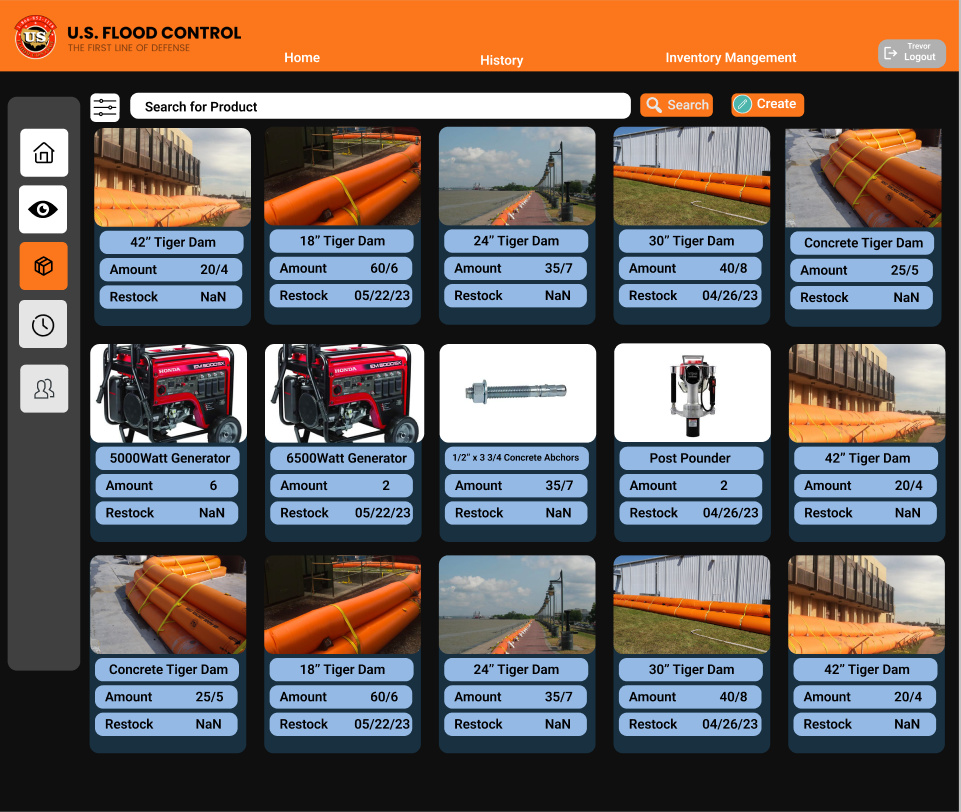
***Data Entry Screens***

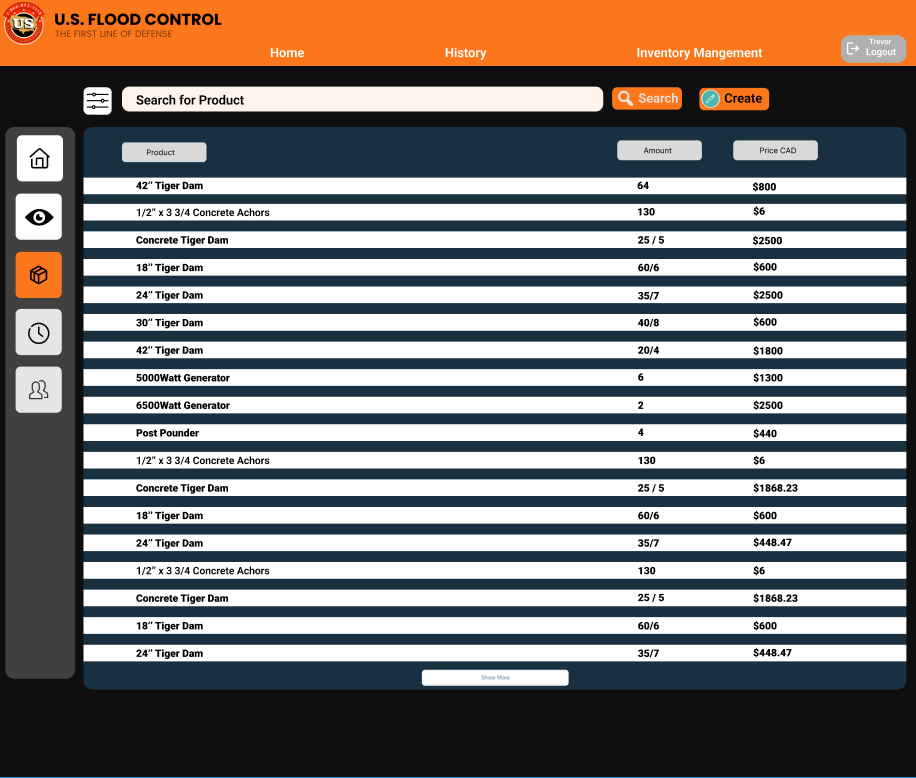




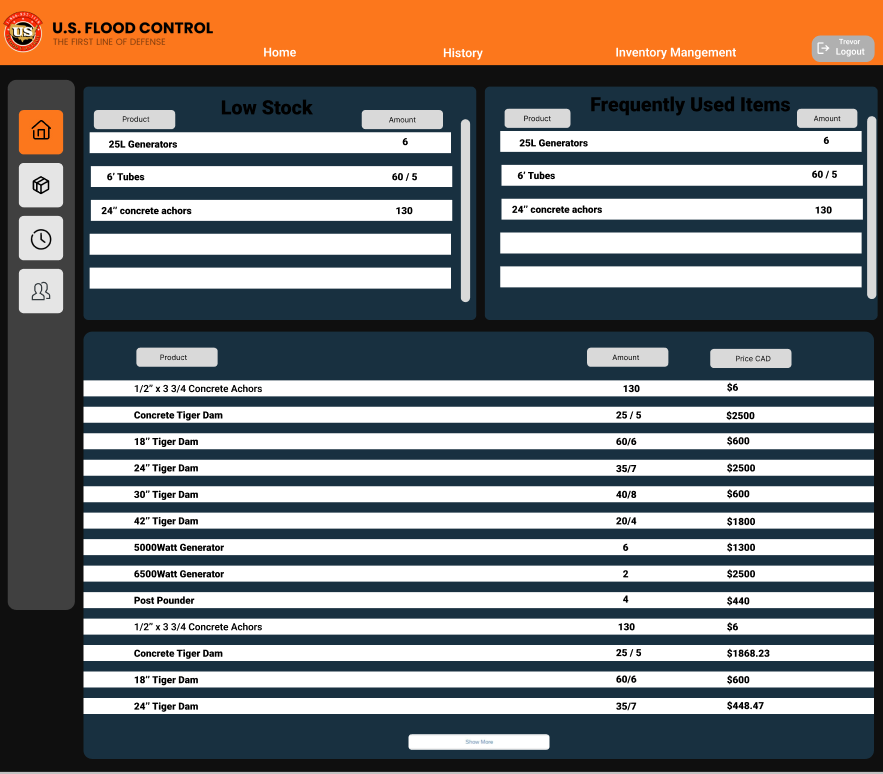


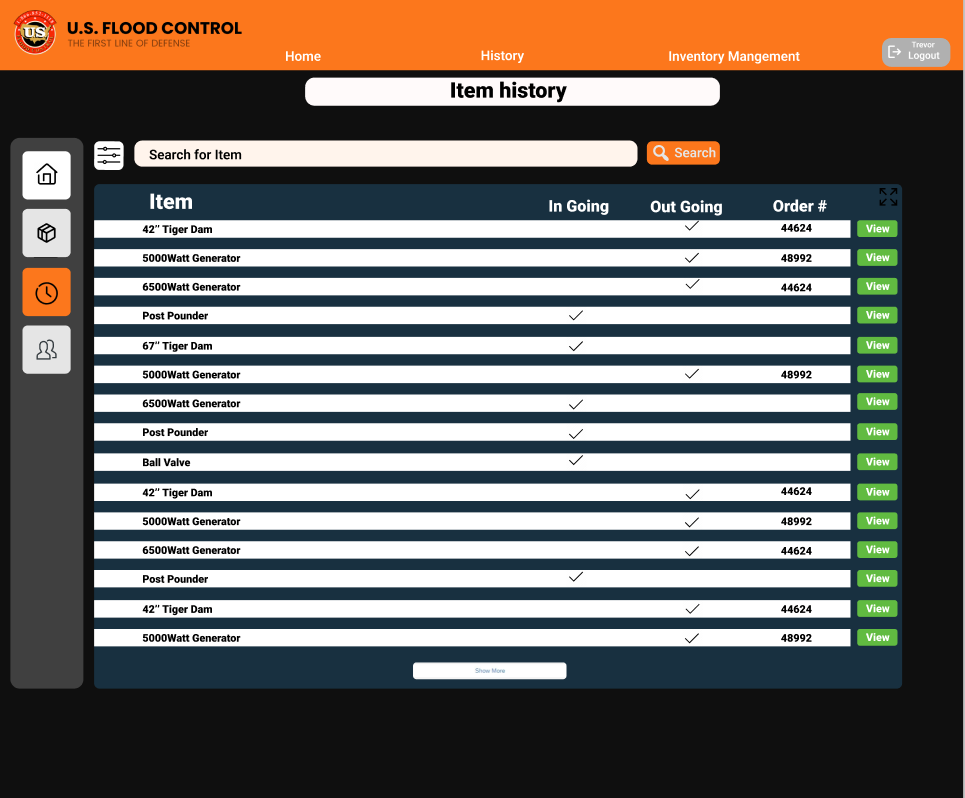
* **Data View Screens**

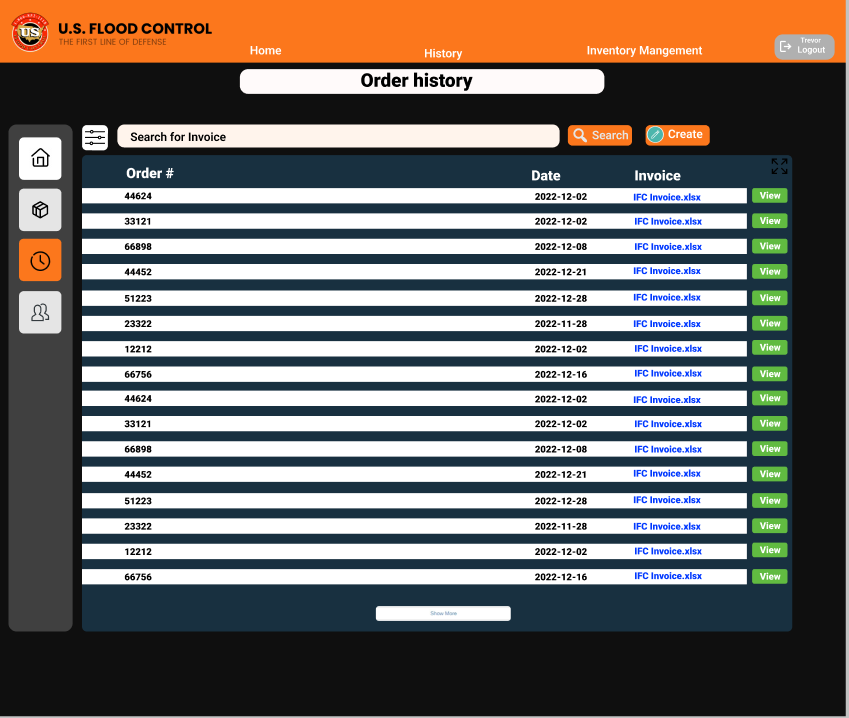


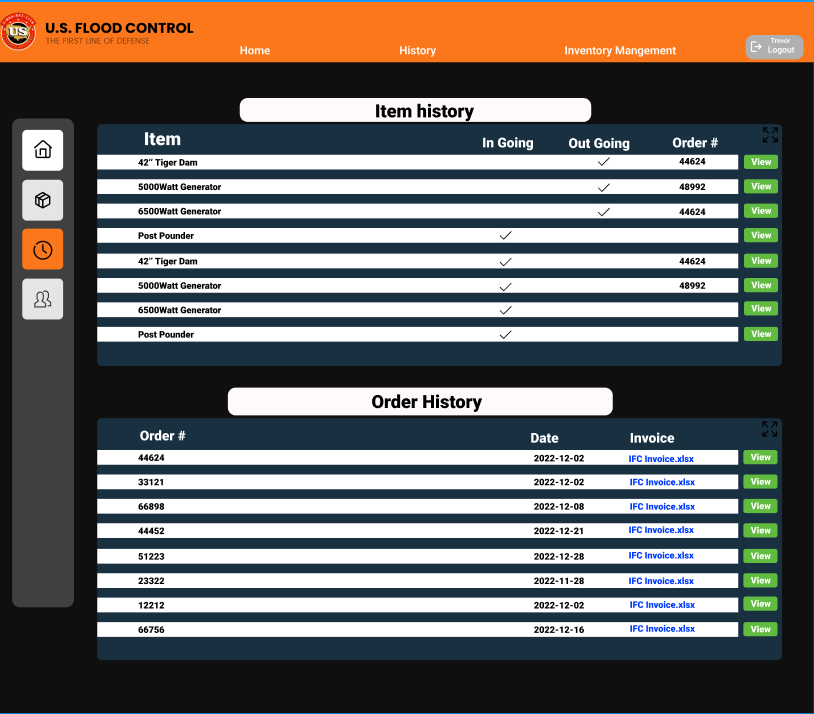


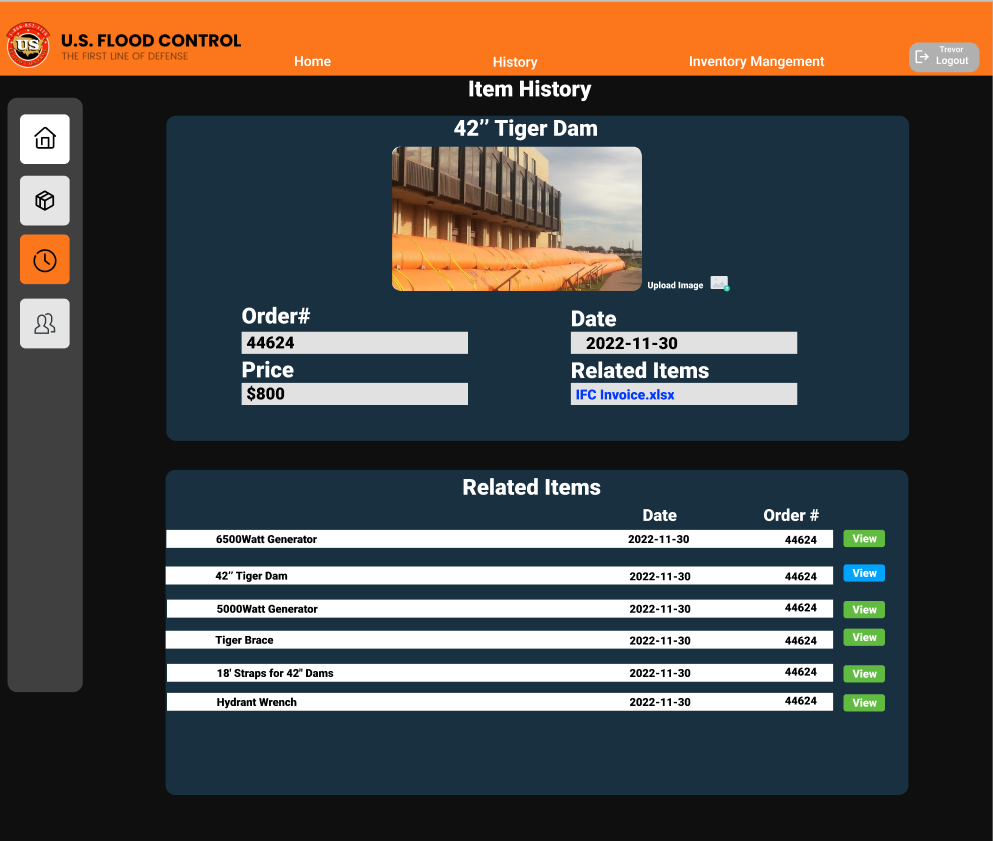


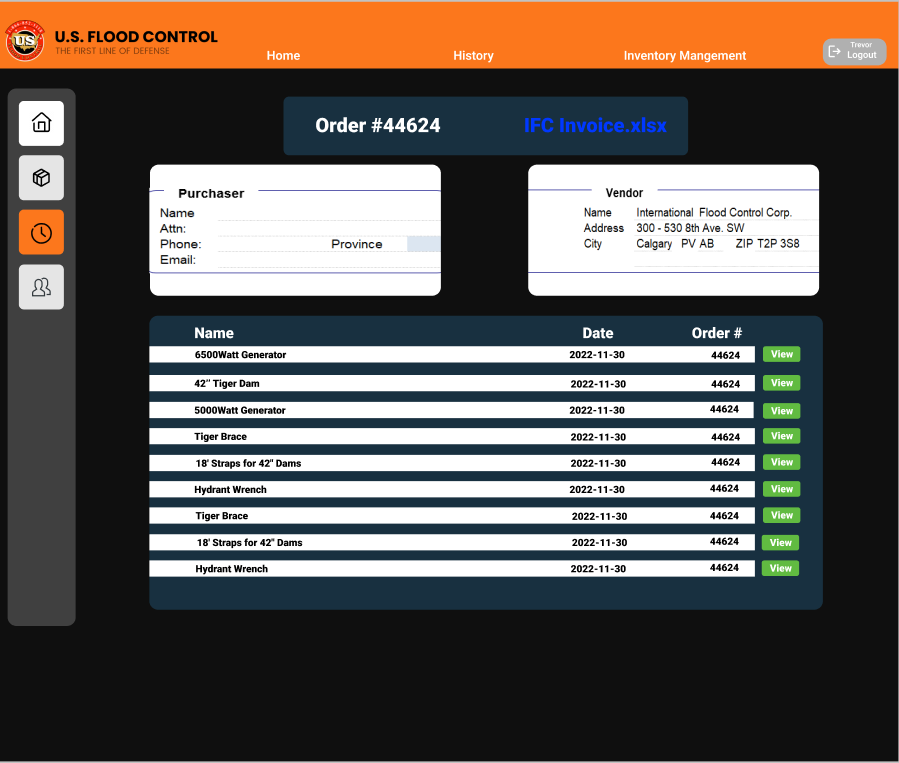




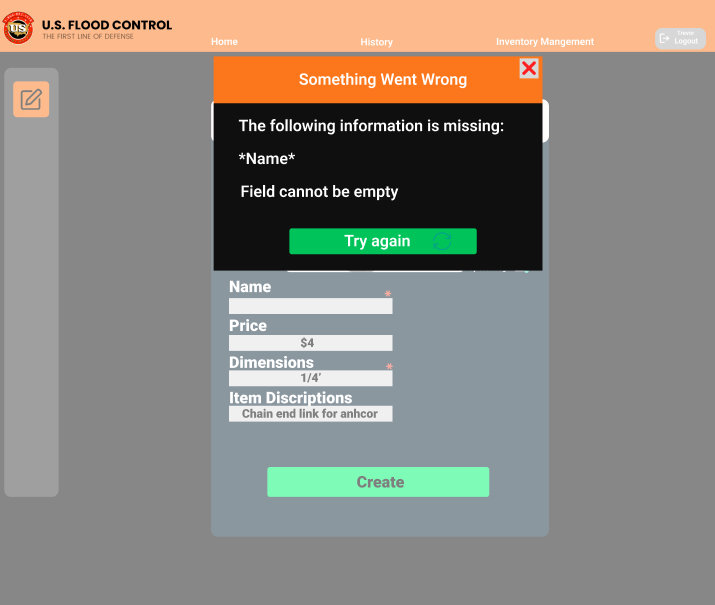


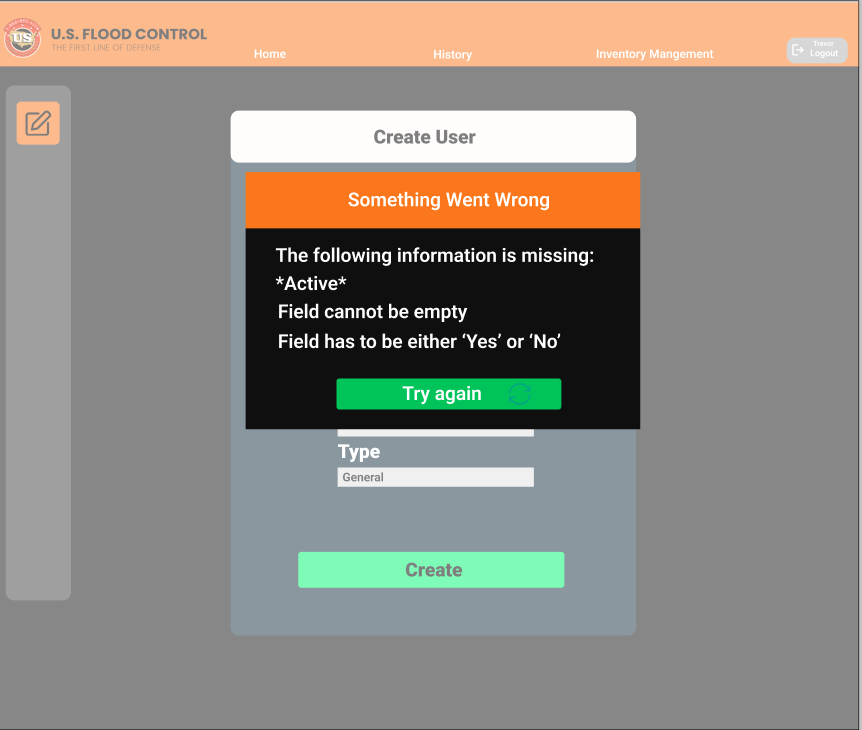


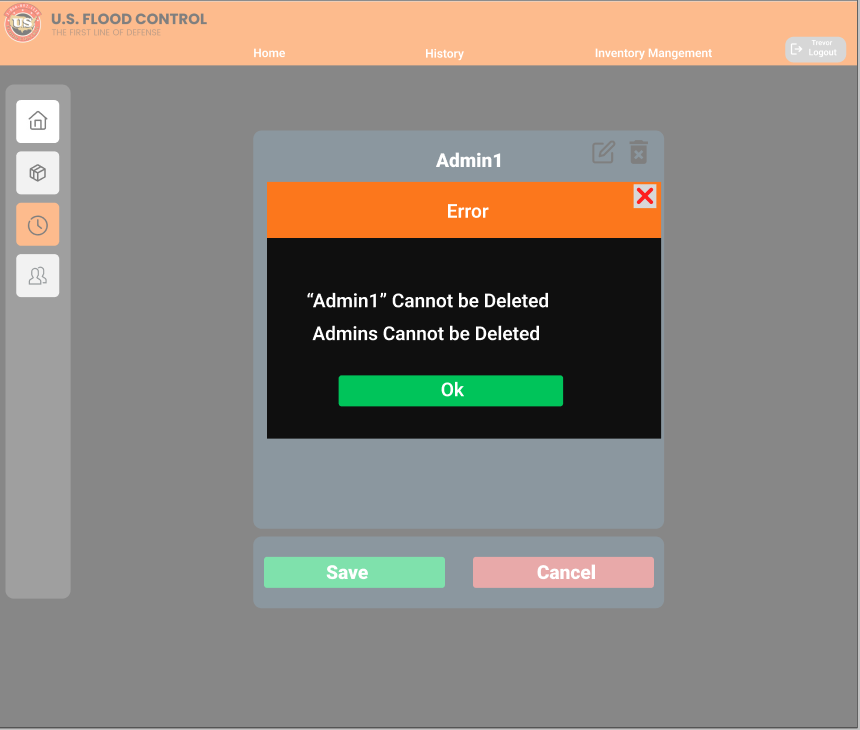




* **Error Screens**







***List of Screens***

***Login Screen***

* *Banner: U.S. Flood Control*

***Home Screen***

* *Banner: “Home”,”History”,”Inventory Management”*
* *Subscreen: “Low Stock”*
* *Subscreen: “Frequently used Items”*

***Inventory Image view Screen***

* *Banner: “Home”,”History”,”Inventory Management”*

***Inventory List view Screen***

* *Banner: “Home”,”History”,”Inventory Management”*

***Item view Screen***

* *Banner: “Home”,”History”,”Inventory Management”*
* *Item Description Title:”Lorem Ipsum”*

***Item Creation Screen***

* *Banner: “Home”,”History”,”Inventory Management”*
* *Screen Title:”Create Item ”*

***Item Edit Screen***

* *Banner: “Home”,”History”,”Inventory Management”*
* *Screen Title:”Lorem Ipsum”*

***History Screen***

* *Banner: “Home”,”History”,”Inventory Management”*
* *Subscreen:”Item History”*
* *Subscreen:“Order History”*

***Order History Screen***

* *Banner: “Home”,”History”,”Inventory Management”*
* *Screen Title:”Order History”*

***Item History Screen***

* *Banner: “Home”,”History”,”Inventory Management”*
* *Screen Title:”Item History”*

***Invoice view Screen***

* *Banner: “Home”,”History”,”Inventory Management”*
* *Screen Title:”Order”+ order#*

***Order Item view Screen***

* *Banner: “Home”,”History”,”Inventory Management”*
* *Screen Title:”Item History”*

***Users Screen***

* *Banner: “Home”,”History”,”Inventory Management”*
* *Screen Title:”Users ”*

***User view Screen***

* *Banner: “Home”,”History”,”Inventory Management”*
* *Screen Title: “lorem ipsum”*

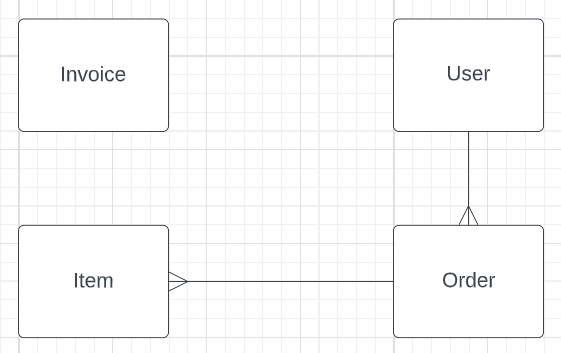
***Create User Screen***

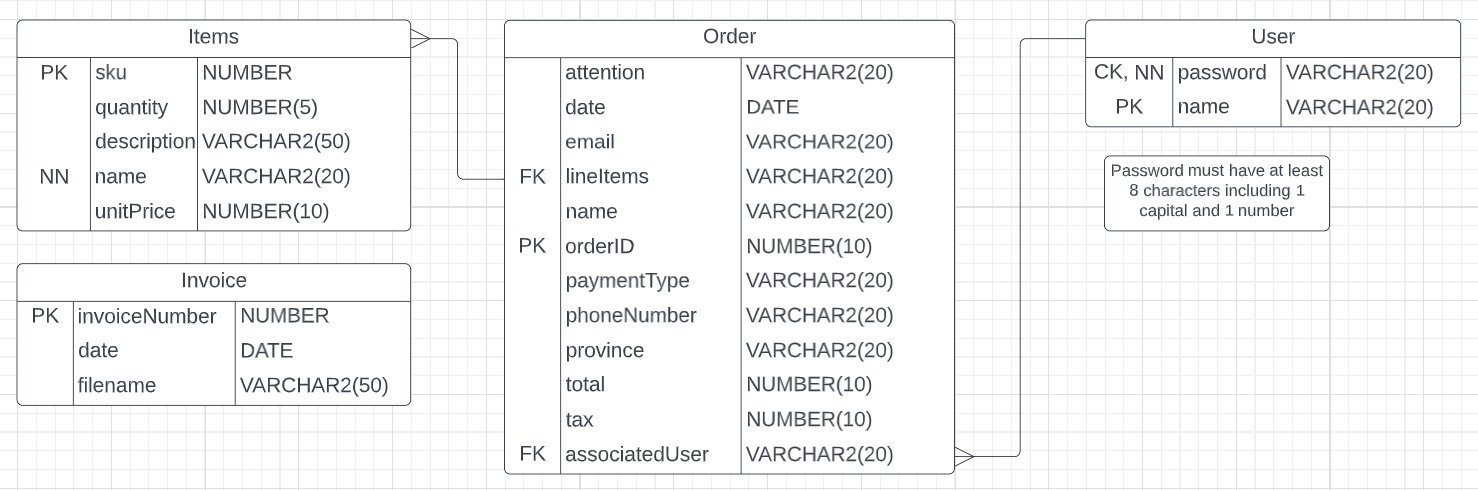
* *Banner: “Home”,”History”,”Inventory Management”*
* *Screen Title:”Create User”*

## Persistence Model

* *We are using MongoDB, JPA, stored procedures.*

***Conceptual*** *Schema ERD:*



***Internal*** *Schema ERD:*

* *The expected calculated data size would be (Item(~120bytes) \* 200) + (order(~200bytes) \* 1000) + (user(~40bytes) \*20) = 224800bytes or ~224 kilobytes(KB)*

## Class Diagrams

**Package “CA.SAIT.MODELS”:**

This will hold java class object representing a user, order, and item. These classes are utilized by all the other packages.

Fig(1, Models package):

Diagram

Description automatically generated with medium confidence

**Package “CA.SAIT.SERVLET”:**

This will hold HCI classes that allow for user interactions and processes to be read and used inside the System, it will utilize CA.SAIT.MODELS and communicate bilaterally with CA.SAIT.SERVICES. Servlet classes do not directly communicate with the database, only with the Java internal of the system, and the User Interface HTML.

Fig(2, Servlets package):

Graphical user interface

Description automatically generated with medium confidence

**Package “CA.SAIT.SERVICES”:**

This package is responsible for connecting the servlets to out persistent classes, this package follows a composite pattern and is called on for more than one specific use case. It will communicate bilaterally with CA.SAIT.SERVLETS to push data to the HCI and will also bilaterally communicate with the CA.SAIT.DATAACCESS to both read and add database information. Services classes do not directly communicate with the HCI and associated HTML classes.

Fig(3, Services package):

Text

Description automatically generated

**Package “CA.SAIT.DATAACCESS”:**

This will hold Persistence classes that allow for database interactions, it will communicate bilaterally with CA.SAIT.SERVICES to push data from the database to the system, allowing the system to get the information and data that is needed. Package does not communicate with any HCI, or System Processes.

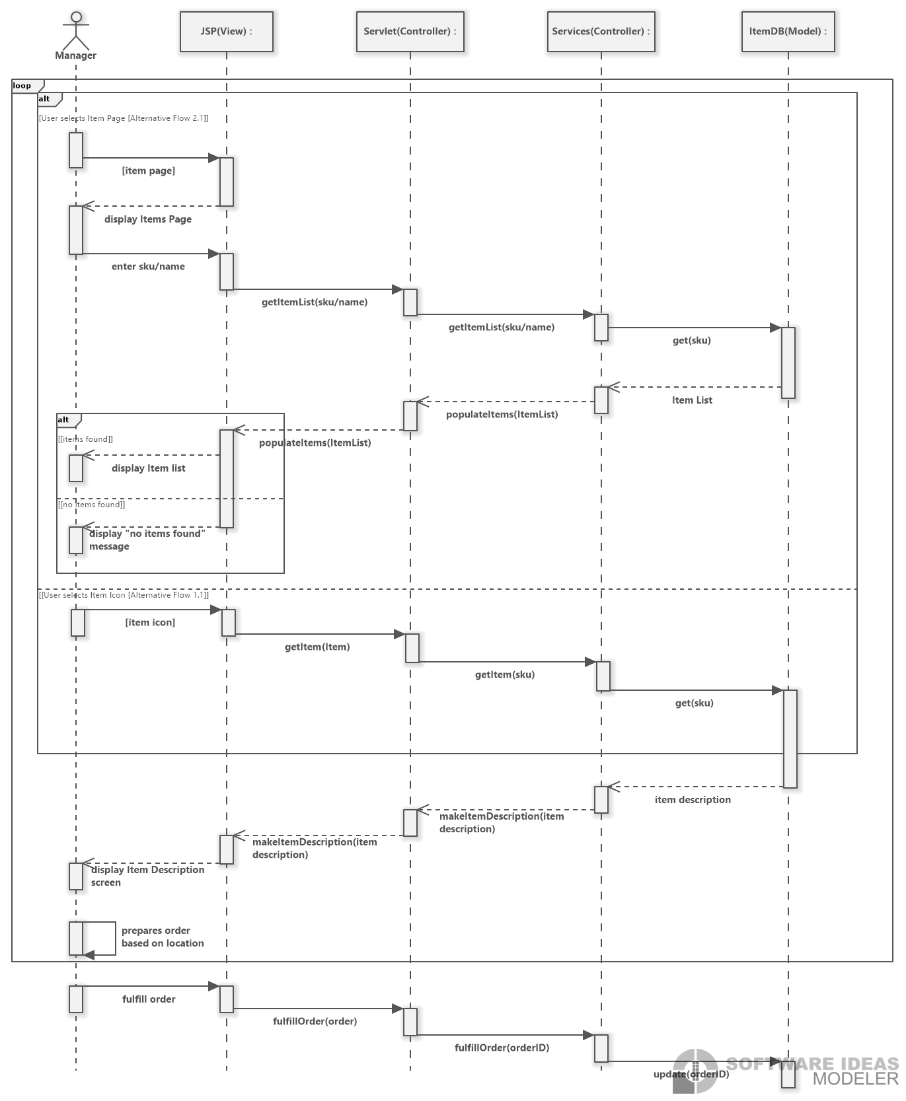
Fig(4, DATAACCESS package):

**Graphical user interface

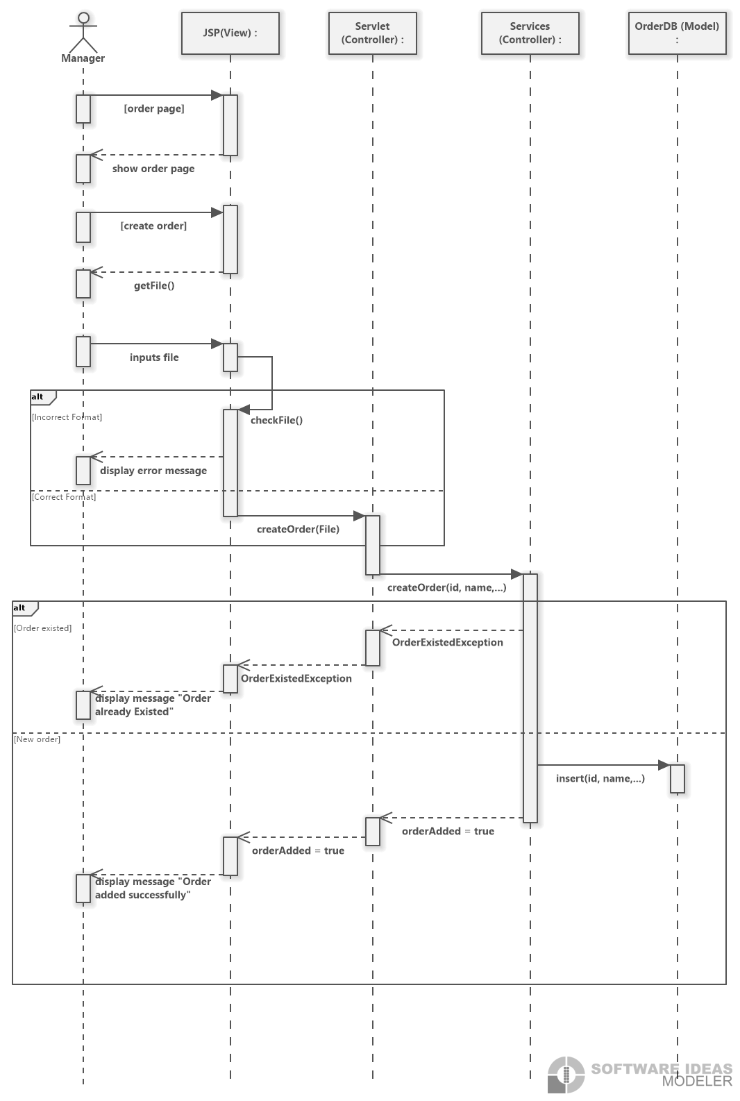
Description automatically generated with low confidence**

## Interaction Sequence Diagrams

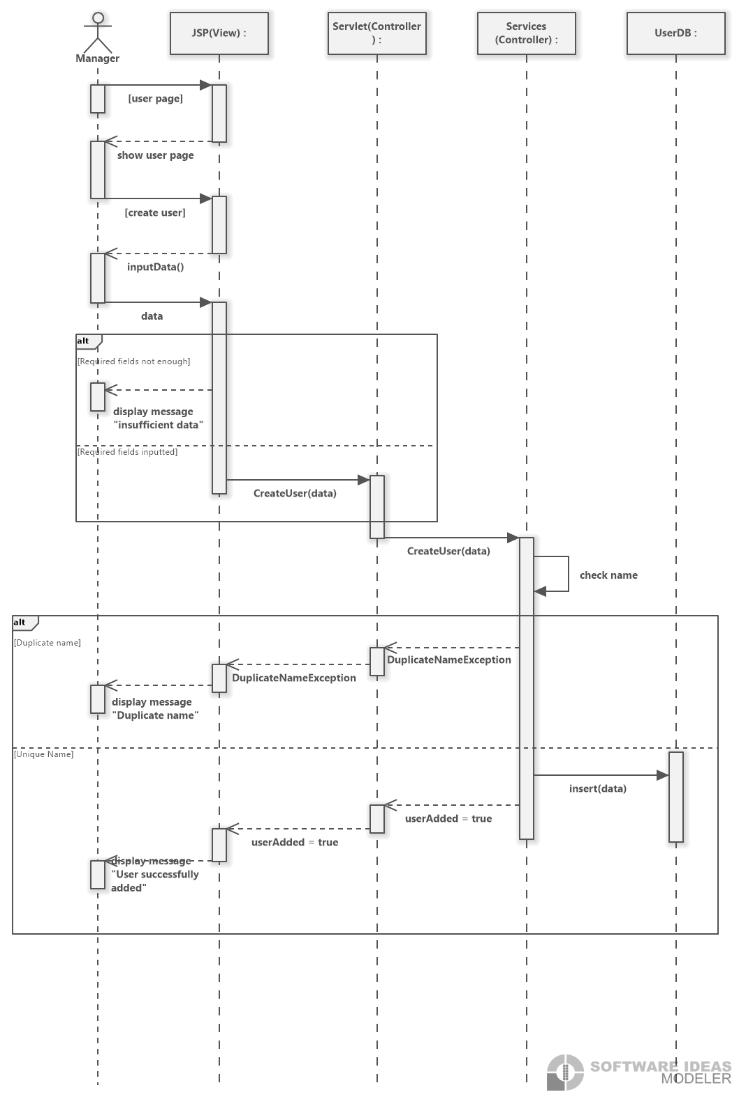
Use Case 1



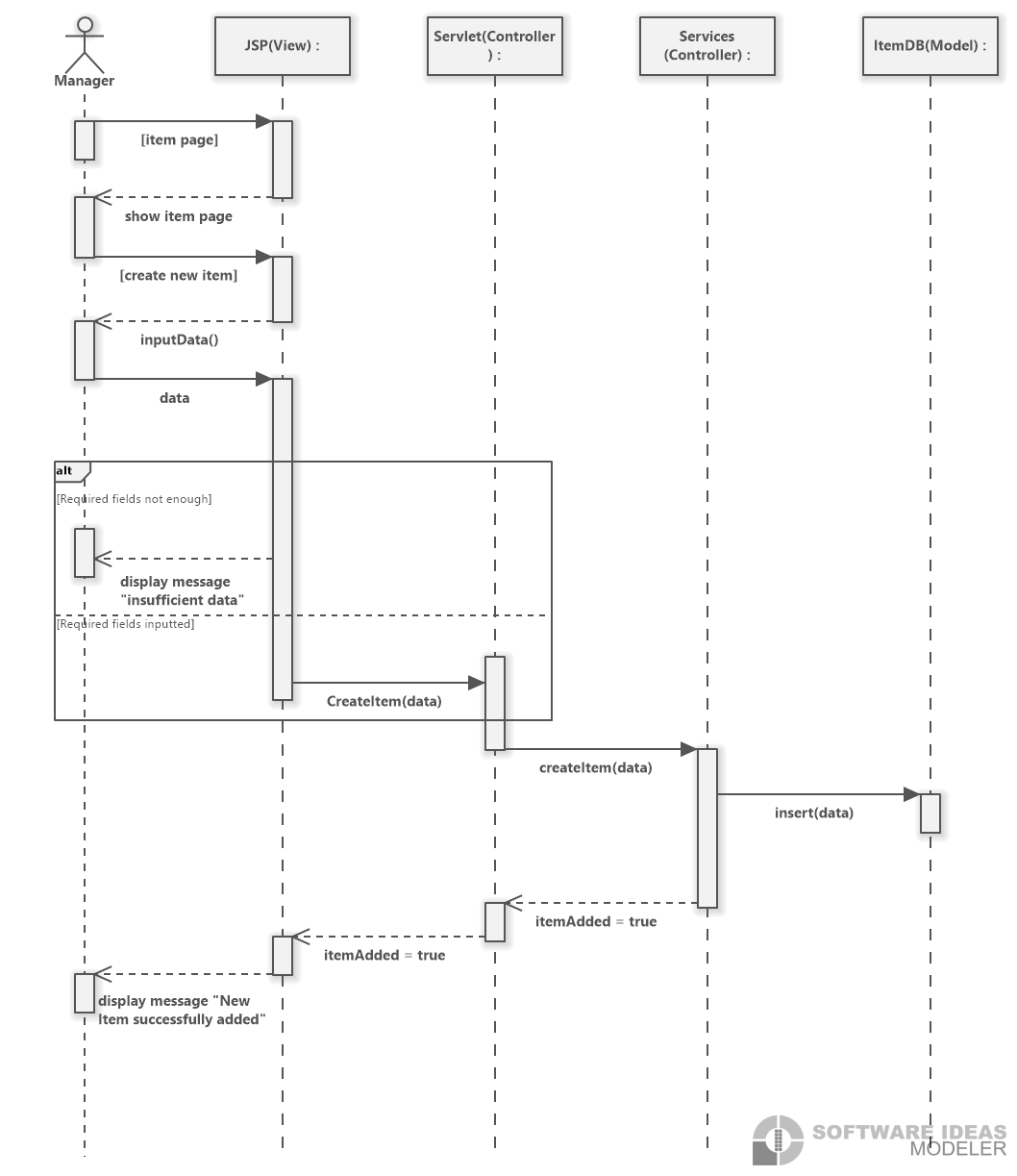
Use Case 2



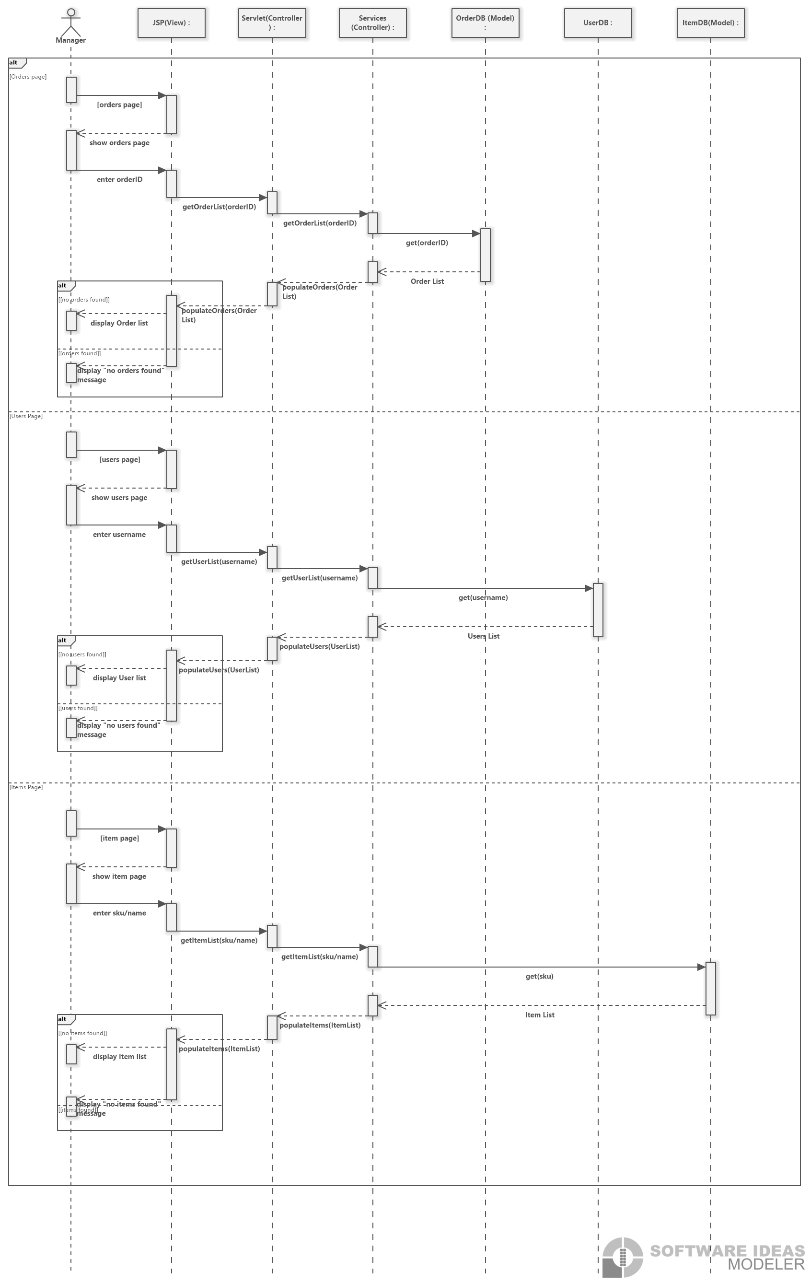
Use Case 3



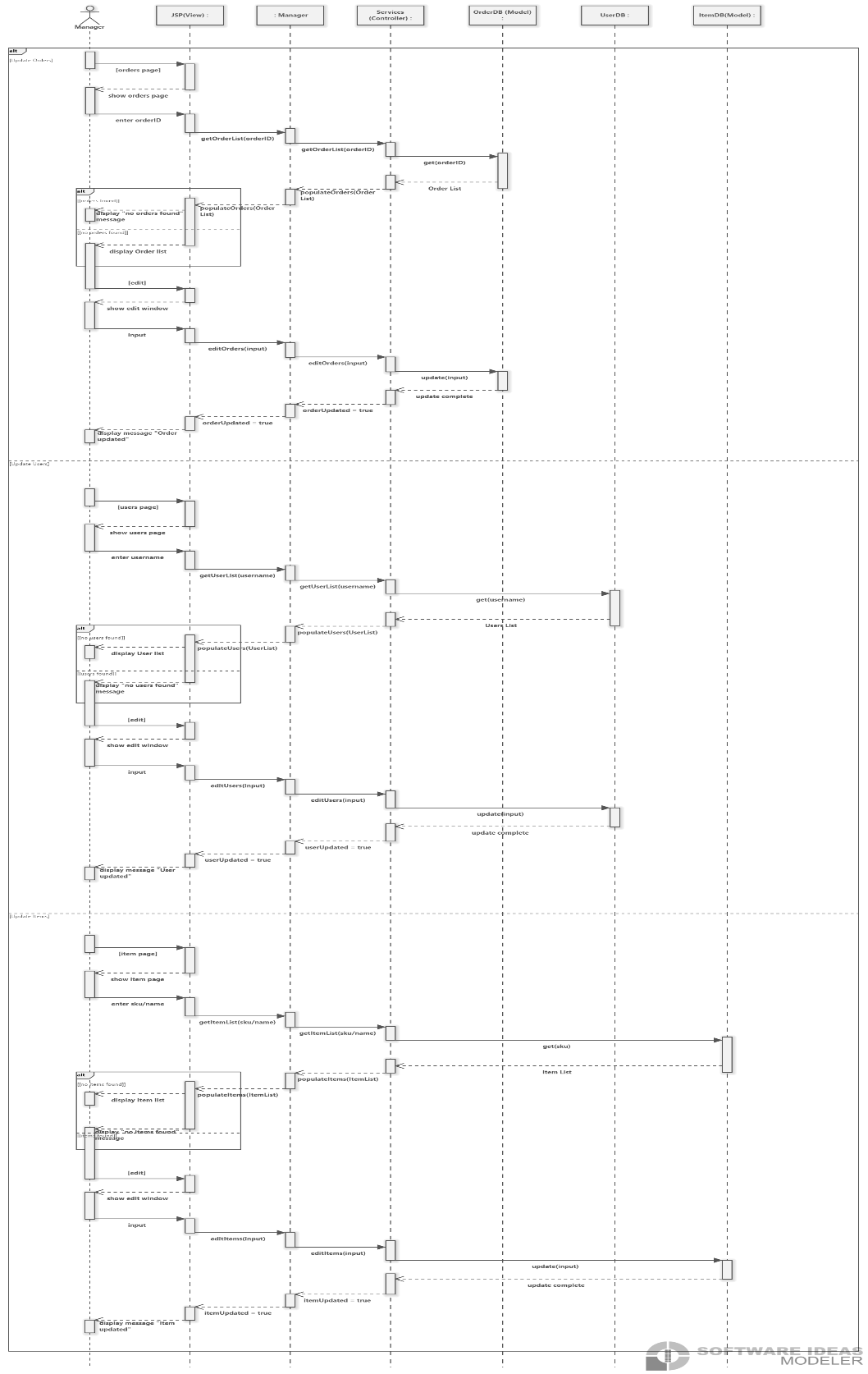
Use Case 4



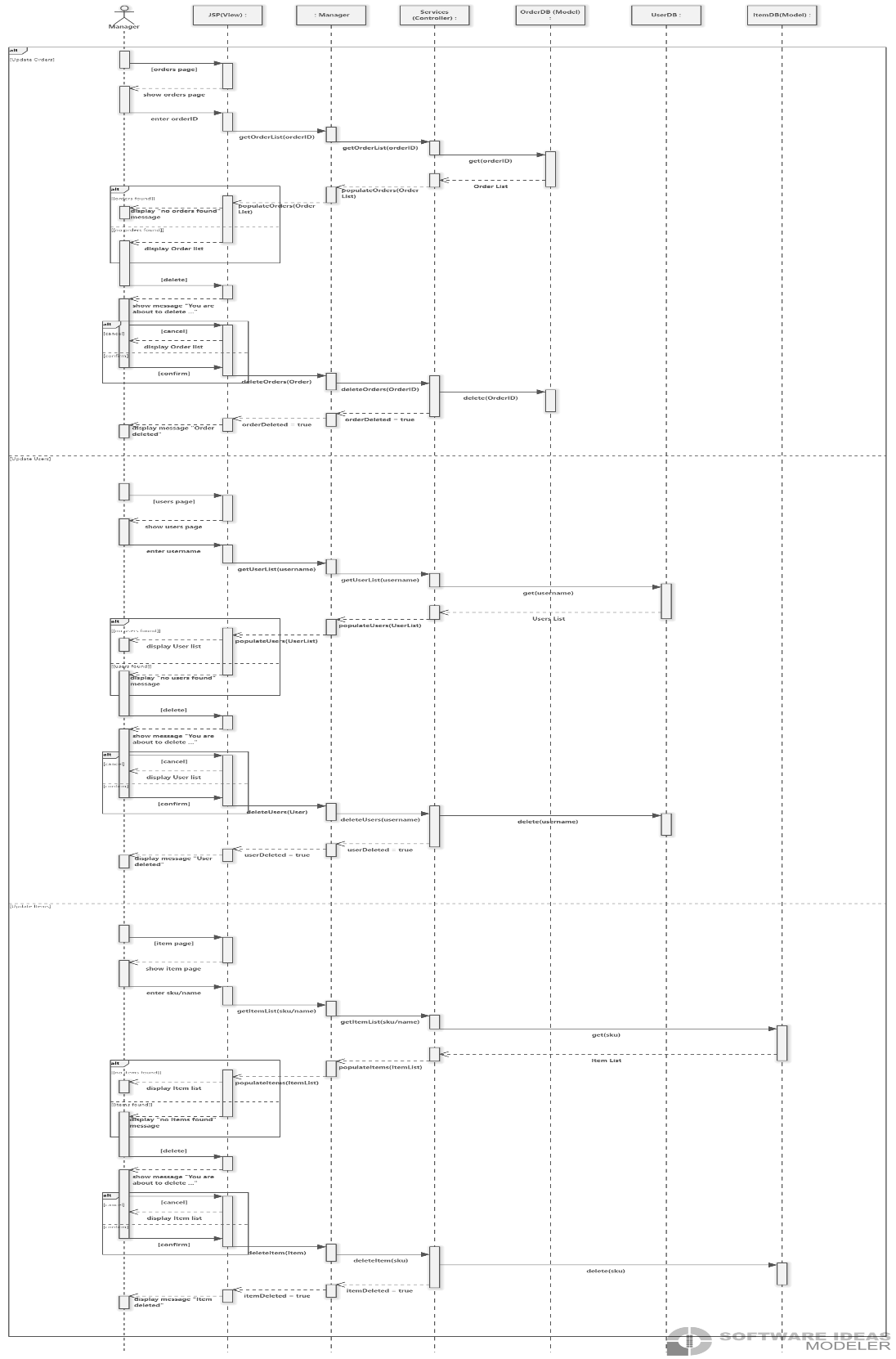
Use Case 5



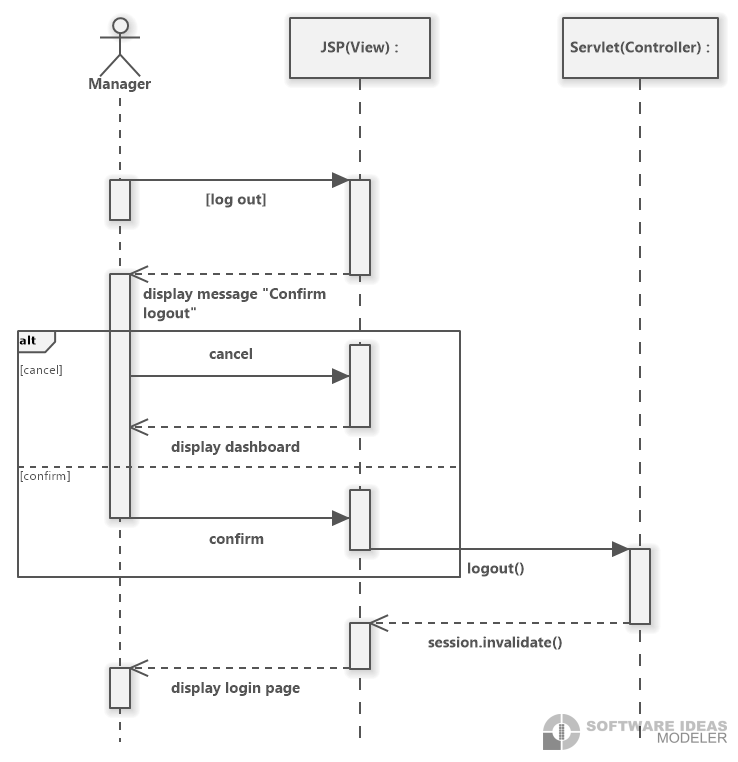
Use Case 6



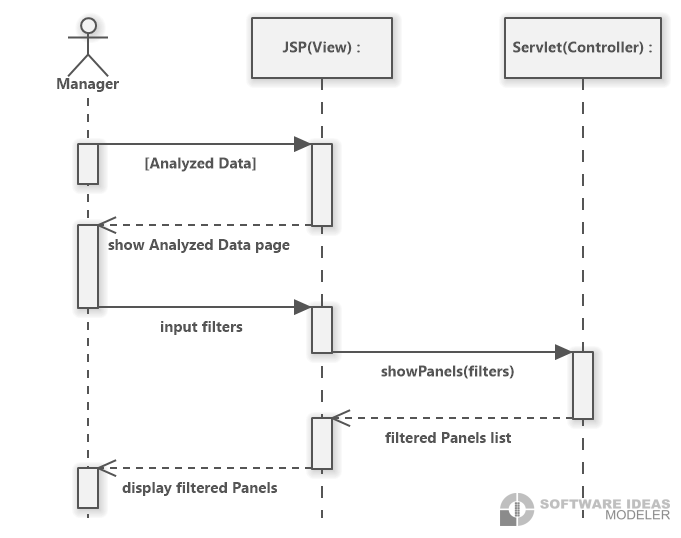
Use Case 7



Use Case 8



Use Case 9



# **Project Management**

## **Schedule**

|  |  |  |
| --- | --- | --- |
| **#** | **Milestone** | **phase** |
| 1 | Inquire about initial requirements | Assessment |
| 2 | Define Scope and goal of project | Definition |
| 3 | Work on prototype of frontend | Execution |
| 4 | Work on prototype of backend | Execution |
| 5 | Create working prototype of complete app (minor functionality) | Execution |
| 6 | Present complete prototype to client for approval | Assessment |
| 7 | Completion of database | Execution |
| 7 | Completion of backend | Execution |
| 8 | Completion of frontend | Execution |
| 10 | Connection of components successful | Execution |
| 11 | Debugging of complete product successfully |  |
| 12 | Documentation of final product finalized | Execution |
| 13 | Project completion | Warranty and Closure |

## **Team Configuration**

### **Members:**

* Dylano Van Der Meer
* Joshua Law
* Trung Hieu Tran
* Daniel Wong
* Rajdeep Sangha

### **Roles**

* *Front-end developers*: Rajdeep, Daniel, Joshua
* *Back-end developers*: Dylano, Trung, Rajdeep
* *Scribes*: Daniel, Trung
* *Client liaison:* Rajdeep
* *Team leader:* Joshua (Dictatorship)

### **Reporting relationships**

* All members of the team will organize a meeting after completion of components or if an issue arises. We will also all report to the instructor Ali Moussa.

### **Contact information(Emails):**

* [Daniel.y.wong@edu.sait.ca](mailto:Daniel.y.wong@edu.sait.ca)
* [Rajdeepsangha166@gmail.com](mailto:Rajdeepsangha166@gmail.com)
* [*TrungHieu.Tran@edu.sait.ca*](mailto:TrungHieu.Tran@edu.sait.ca)
* [*Joshuasmlaw@gmail.com*](mailto:Joshuasmlaw@gmail.com) *or* [*Joshua.Law@edu.sait.ca*](mailto:Joshua.Law@edu.sait.ca)
* [*Dylanovandermeer@gmail.com*](mailto:Dylanovandermeer@gmail.com) *or* [*Dylano.vandermeer@edu.sait.ca*](mailto:Dylano.vandermeer@edu.sait.ca)

### **Project Standards and Procedures**

* **Communication:** Main method of communication will be Microsoft Teams. Each team member has provided their schedules to find the date and time for a consistent meeting.
* **Execution:** The team will do our best to complete most of the work during class time and save the meeting time for debriefs and extra planning.

# 

# **Glossary**

|  |  |  |
| --- | --- | --- |
|  | **Term** | **Definition** |
| 1 | Inventory System | A process used to track stock, supplies and sales through an application |
| 2 | System | A set of things working together to give an output |
| 3 | User Interface | The way a user and a computer/application interact |
| 4 | Comma Separated values (CSV) | Is a formatted excel sheet file |
| 5 | Discord | Discord is a communication app where users can voice, video and text chat with other users |

# 

# **Index**

Maintainabiity and Administration Requirements

Maintainability 16

**Maintainability and Administration Requirements** 16

Administration 16

**Non-Functional Requirements** 16

Security 16

**Preface** 5

Documentation Standards 5

Purpose of Document 5

Project Management 17

Schedule 17

Team Configuration 18

**Project Overview** 6

Current System 7

Project Scope 6

Statement of the Problem 6

System Environment 7

Scenario #1\ receiving orders

Figure 2 9

Scenario #1: receiving orders

Figure 1 8

Figure 3 9

Figure 4 10

Figure 5 10

Figure 6 11

Scenario #2

\ Acquiring new inventory items

Figure 7 11

Figure 8 12

Figure 9 12

**Scenario #2: Acquiring new inventory items** 11

Scenario #3

\ Accounting for inventory

Figure 10 13

Figure 11 13

Figure 12 14

Figure 13 15

***Scenario #3: Accounting for inventory*** 13

**System Interface Requirements** 16

**System Requirements** 8

Fact-Finding Methodology 8

Use Case Diagram 8

Team Configuration

Contact information(Emails): 18

Members: 18

Project Standards and Procedures 18

Reporting relationships 18

Roles 18

**Usability Requirements** 17

Use Case Diagram

Scenario #1: receiving orders 8

**User Requirements** 5

Business Overview and Objectives 5

**Appendix A: Data Dictionary**

|  |  |  |
| --- | --- | --- |
| Class | Attribute | Operation |
|  |  |  |
| ***User*** – A superclass that defines the attributes and operations to be inherited by sub class | **username: String *–*** A unique attribute that every user must have, used to sign-in to an account.  **password: String *–*** An attribute that every user must have, used to sign-in to an account.  **verifySession: Boolean** – An attribute utilized to confirm if a sign in is required, is *FALSE* when not signed-in and session is created, and returns *TRUE* if user session is active and user is signed in | **getUsername (): String** – An operation that returns the username of a selected user in the form of a *String*  **setUsername (username : String)**– An operation that updates and sets the username attribute to that of the *String username* parameter  **getPassword (): String** – An operation that returns the password of a selected user in the form of a *String*  **setPassword (password : String)** – An operation that updates and sets the password attribute to that of the *String password* parameter  **signIn ()** – An operation that checks if the user *verifySession* is *FALSE.* The operation than proceeds to force the user to sign back into their user account for system access  **getUser () : String, String** – returns the user’s login information for validation |
| ***Manager*** – A subclass of ***User***, it provides addition attributes and operations specific to that of a manager | **name: String** – An attribute that specifies the full(first and last) name of the manager in the form of a *String*  **emailAddress: String** – An attribute that specifies the email address of the manager, this address should be a corporate/company email address  **isActiveEmployee: Boolean** – An attribute that specifies whether the employee is active in the present moment, should be utilized if employee is suspended, on leave, or currently unavailable.  **phoneNumber: String** – an attribute specifying the manager’s phone number, in the form of *String*, this should be a corporate phone number, where *NULL* is reserved for those without such a number  **hasAdmin: Boolean** - An attribute to determine if the manager has administrative system privileges | **getName (): String** – An operation that returns the name of a selected user in the form of a *String*  **setName (name : String)**– An operation that updates and sets the name attribute to that of the *String name* parameter  **getEmailAddress (): String** – An operation that returns the email address of a selected user in the form of a *String*  **setEmailAddress (emauilAddress : String)**– An operation that updates and sets the email address attribute to that of the *String emailAddress* parameter  **getPhoneNumber (): String** – An operation that returns the phone number of a selected user in the form of a *String*  **setPhoneNumber (phoneEmail : String)**– An operation that updates and sets the phone numbers attribute to that of the *String phoneNumber* parameter |
| ***Employee*** – A subclass of ***User***, it provides additional attributes and operations specific to that of an employee | **name: String** – An attribute that specifies the full(first and last) name of the manager in the form of a *String*  **emailAddress: String** – An attribute that specifies the email address of the manager, this address should be a corporate/company email address  **isActiveEmployee: Boolean** – An attribute that specifies whether the employee is active in the present moment, should be utilized if employee is suspended, on leave, or currently unavailable.  **phoneNumber: String** – an attribute specifying the manager’s phone number, in the form of *String*, this should be a corporate phone number, where *NULL* is reserved for those without such a number  **supervisingManager: String** - An attribute to determine what manager is the supervisor for the employee | **getName (): String** – An operation that returns the name of a selected user in the form of a *String*  **setName (name : String)**– An operation that updates and sets the name attribute to that of the *String name* parameter  **getEmailAddress (): String** – An operation that returns the email address of a selected user in the form of a *String*  **setEmailAddress (emauilAddress : String)**– An operation that updates and sets the email address attribute to that of the *String emailAddress* parameter  **getPhoneNumber (): String** – An operation that returns the phone number of a selected user in the form of a *String*  **setPhoneNumber (phoneEmail : String)**– An operation that updates and sets the phone numbers attribute to that of the *String phoneNumber* parameter |
| ***AccountManager*** – A manager class that allows the user to manage, generate, delete, and edit a single/group of account(s) | **sessionStatus: Boolean** – An attribute that determines that the user session has not expires, corrupted, or manually ended. Boolean *TRUE* indicates session is live, whereas *FALSE* will require a new login to reinitialize the session. | **createAccount(String username, String password)** – An operation that creates a new account with the username and password equating to the parameter *String* inputs.  **deleteAccount()**  **editAccount()** |
| ***InvoiceManager*** – A manager class that allows the user to manage, generate, delete, and edit a single/group of invoices | **sessionStatus: Boolean** – An attribute that determines that the user session has not expires, corrupted, or manually ended. Boolean *TRUE* indicates session is live, whereas *FALSE* will require a new login to reinitialize the session. | **createNewInvoice(Order invoiceOrder)** – An operation that creates a new invoice based on the information provided by the parameter *invoiceOrder* object.  **deleteExistingInvoice()**  **findInvoiceByDate()**  **findInvoiceByName()**  **findInvoiceByNumber()**  **findInvoiceByTotal()**  **getAllInvoices()**  **updateExistingInvoice()** |
| ***ItemManager*** – A manager class that allows the user to manage, generate, delete, and edit a single/group of invoice(s) | **sessionStatus: Boolean** – An attribute that determines that the user session has not expires, corrupted, or manually ended. Boolean *TRUE* indicates session is live, whereas *FALSE* will require a new login to reinitialize the session. | **findItemBySKU(Int itemSKU): Item** – An operation that retrieves a single item that contains the same SKU number as that in the *int itemSKU* parameter.  **findItemByName(String itemName): Item** – An operation that retrieves items that contains the same name as that in the *String itemName* parameter.  **findItemByType(String itemType): Item** – An operation that retrieves items that contains the same type as that in the *String itemType* parameter.  **pullAllItems(): List itemList** – An operation that retrieves all items in the item database in the form of an traversable list called *List itemList*.  **editItem()** |
| ***OrderManager*** – A manager class that allows the user to manage, and delete a single order | **sessionStatus: Boolean** – An attribute that determines that the user session has not expires, corrupted, or manually ended. Boolean *TRUE* indicates session is live, whereas *FALSE* will require a new login to reinitialize the session. | **findOrderByClient(String orderClient)** – An operation that retrieves an order that contains the same client name as that in the *String orderClient* parameter.  **findOrderByDate(String orderDate)** – An operation that retrieves an order that contains the same order date as that in the *String orderDate* parameter.  **findOrderByNumber(int orderNumber)** – An operation that retrieves an order that contains the same order date as that in the *String orderDate* parameter.  **getAllOrders(): List orderList** – An operation that retrieves all order that  exist within the order database in the form of a traversable list called *List orderList*.  **addNewOrder()** |
| ***Account*** | **Username**  **password** |  |
| ***Invoice*** | **Date**  **Description**  **invoiceNumber**  **name**  **total** |  |
| ***Item*** | **Description**  **Name**  **Quantity**  **Sku**  **type** |  |
| ***Order*** | **orderNumber**  **orderName**  **clientName**  **description**  **orderItems** |  |