**Requirements 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

# **Table of Contents**

[Table of Contents 3](#_Toc116323967)

[**Preface** 5](#_Toc116323968)

[**Purpose of Document** 5](#_Toc116323969)

[**Documentation Standards** 5](#_Toc116323970)

[**User Requirements** 5](#_Toc116323971)

[**Business Overview and Objectives** 5](#_Toc116323972)

[**Project Overview** 6](#_Toc116323973)

[**Statement of the Problem** 6](#_Toc116323974)

[**Project Scope** 6](#_Toc116323975)

[**System Environment** 7](#_Toc116323976)

[**Current System** 7](#_Toc116323977)

[**System Requirements** 8](#_Toc116323978)

[**Fact-Finding Methodology** 8](#_Toc116323979)

[**Use Case Diagram** 8](#_Toc116323980)

[**Scenario #1: receiving orders** 8](#_Toc116323981)

[Figure 1 8](#_Toc116323982)

[Figure 2 9](#_Toc116323983)

[Figure 3 9](#_Toc116323984)

[Figure 4 10](#_Toc116323985)

[Figure 5 10](#_Toc116323986)

[Figure 6 11](#_Toc116323987)

[**Scenario #2: Acquiring new inventory items** 11](#_Toc116323988)

[Figure 7 11](#_Toc116323989)

[Figure 8 12](#_Toc116323990)

[Figure 9 12](#_Toc116323991)

[***Scenario #3: Accounting for inventory*** 13](#_Toc116323992)

[Figure 10 13](#_Toc116323993)

[Figure 11 13](#_Toc116323994)

[Figure 12 14](#_Toc116323995)

[Figure 13 15](#_Toc116323996)

[**Non-Functional Requirements** 16](#_Toc116323997)

[**System Interface Requirements** 16](#_Toc116323998)

[**Maintainability and Administration Requirements** 16](#_Toc116323999)

[**Usability Requirements** 16](#_Toc116324000)

[Project Management 17](#_Toc116324001)

[**Schedule** 17](#_Toc116324002)

[**Team Configuration** 17](#_Toc116324003)

[**Members:** 17](#_Toc116324004)

[**Roles** 18](#_Toc116324005)

[**Reporting relationships** 18](#_Toc116324006)

[**Contact information(Emails):** 18](#_Toc116324007)

[**Project Standards and Procedures** 18](#_Toc116324008)

[Glossary 19](#_Toc116324009)

[Index 20](#_Toc116324010)

# **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 International Flood Control is an emergency flood supply company who specializes in sending flood prevention supplies.
* Tiger Dam’s mission is to help communities negate massive floods damage; they strive to be as efficient as possible to assist communities
* Our mission with this company is to build them an inventory management system to better manage their warehouse inventory to process and manage inventory in an instance

# **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. We aim to create an inventory management application that uses a database to store the inventory system that allows data manipulation by the employees.

## **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**

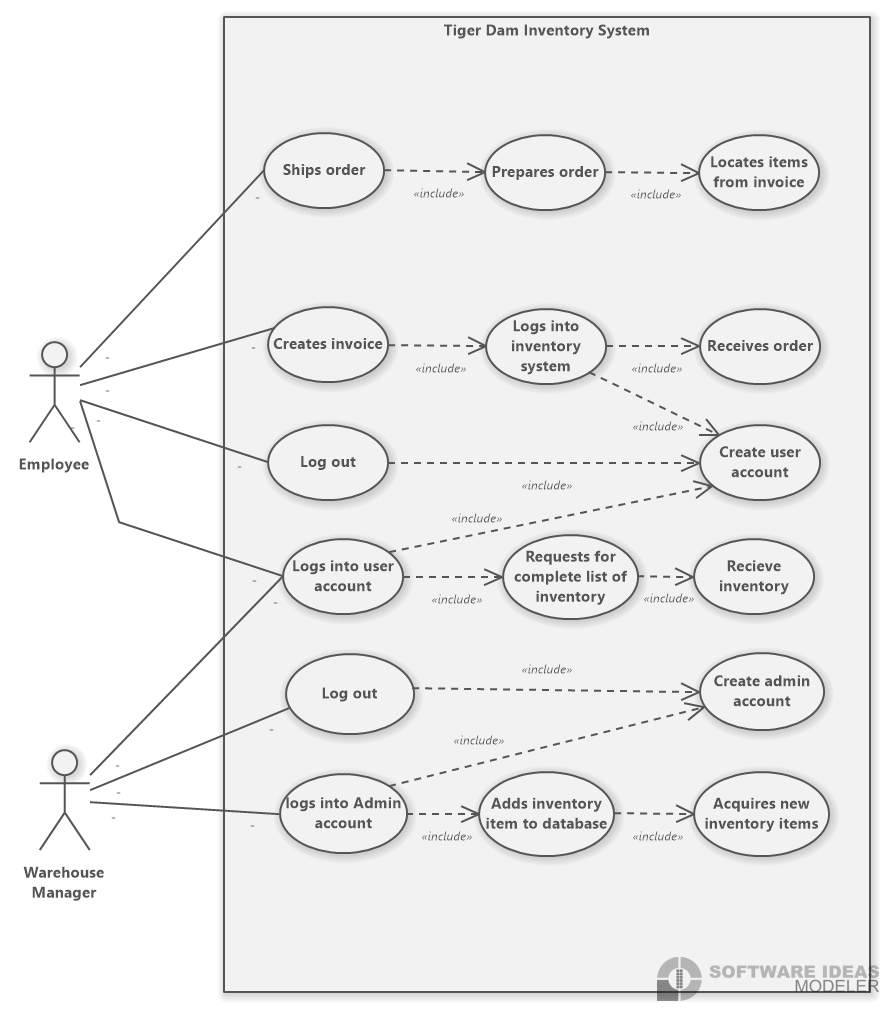
The current system being utilized by the client is an impartial system that relies on excel generated files that are used as orders. As for inventory, the client uses a method of written data that is used to check quantities. The Client then proceeds to fulfill the order based on the inventory available, 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**

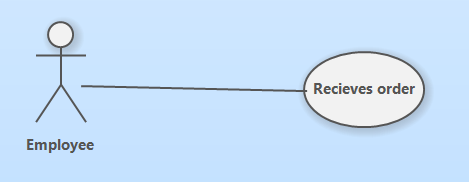
* We interviewed our client via a call.
* Our first approach was enabling our client take the stand and walk us through a typical day at the warehouse, as he described his day, we would take notes and think of problems and solutions he faces
* One of the questions we asked was, “would you want the system 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**



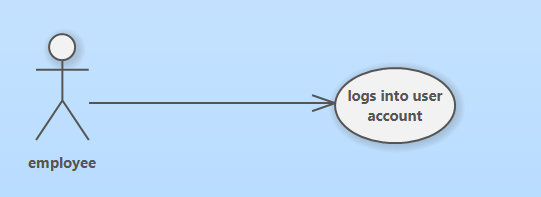
The employee then requests a system inventory check. This prompts the system to create a CSV file that contains the complete list of the inventory accompanied by the unique ID’s, number of items, and the location.

### **Scenario #1: receiving orders**



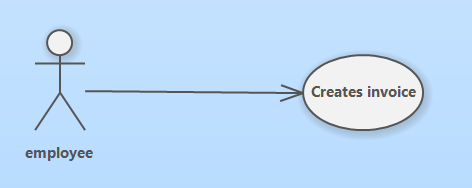
#### Figure 1

An employee receives an order for supplies. The data received contains item numbers, amount, pricing, and the date/time the order is required. All this information is contained in a CSV file.



#### Figure 2

Employee logs into inventory management system via username and password. Inventory System contains the company’s current inventory organized by unique IDs for each item, pricing, category of item, and location within the warehouse.



#### Figure 3

Employee inputs the CSV file from the order into the inventory management system. The system will then parse the information and create an invoice containing the supplies and pricing information for the order.

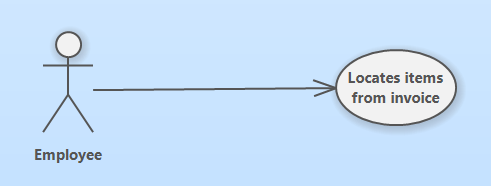
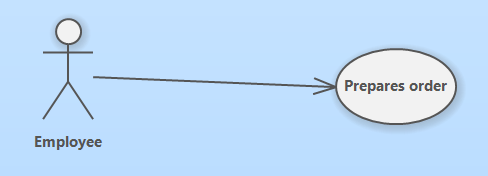


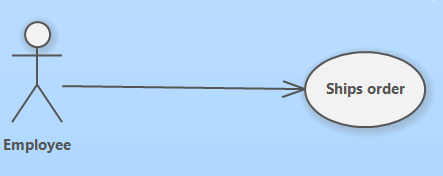
Figure 4

Once the invoice is completed the employee uses the inventory system to locate the items from the invoice. Items can be searched via search-bar. The search results will show the location of the item, and the number of items.



#### Figure 5

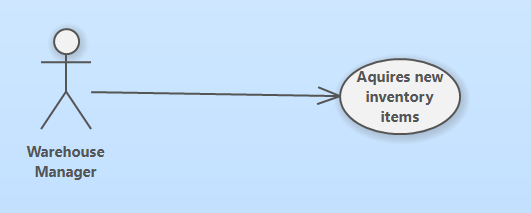
Employees then gather items listen on invoice, packages, then load items onto delivery vehicle.



#### Figure 6

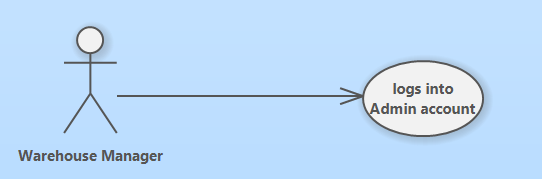
Once the order has been prepared the employee can now ship the order to destination.

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



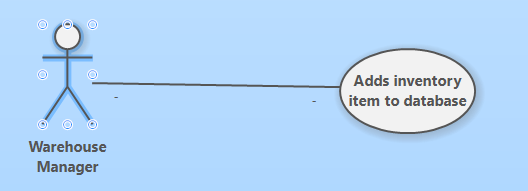
#### Figure 7

The warehouse manager receives new inventory items that are not currently contained in the systems database. These items do not have any unique ID’s, the only information is the name of the product.



#### Figure 8

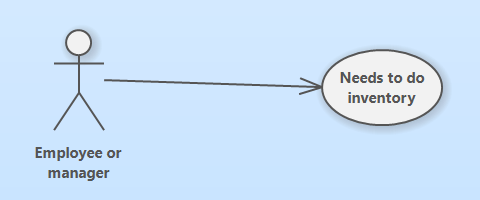
Employee logs into an admin account on the system with a specific username and password. This admin account has permission to add and remove items from the inventory system.



#### Figure 9

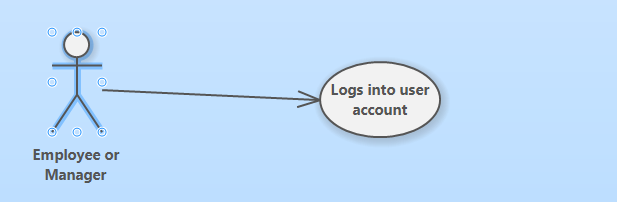
The manager then adds item information to inventory database and saves the changes using the item name, unique identifier, price, and location. If any mistakes occur the admin account has permission to edit items in database.

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



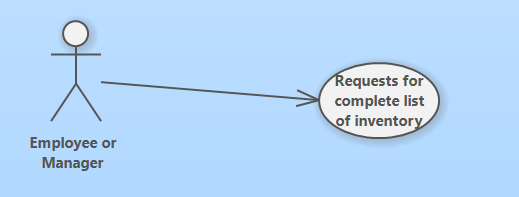
#### Figure 10

Employees need to perform an inventory check on the entire warehouse. The employee needs a complete breakdown of all the inventory items accompanied by the amount of each unique item.



#### Figure 11

The employee then logs into a user account with username and password. An admin account is not necessary because no edits to the system inventory are being made.



#### Figure 12

The employee then requests a system inventory check. This prompts the system to create a CSV file that contains the complete list of the inventory accompanied by the unique ID’s, number of items, and the location.

## **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**

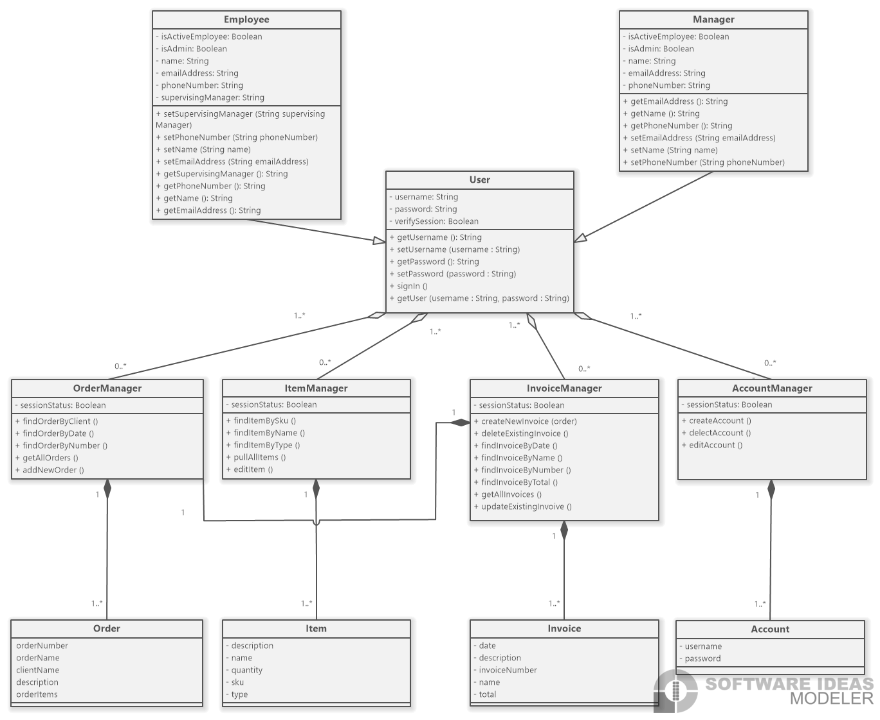
**Maintainability:** We plan to update our system to incorporate different inventory items after our initial implementation. We will also perform regular system maintenance like backing up the database and bug fixes.

**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.

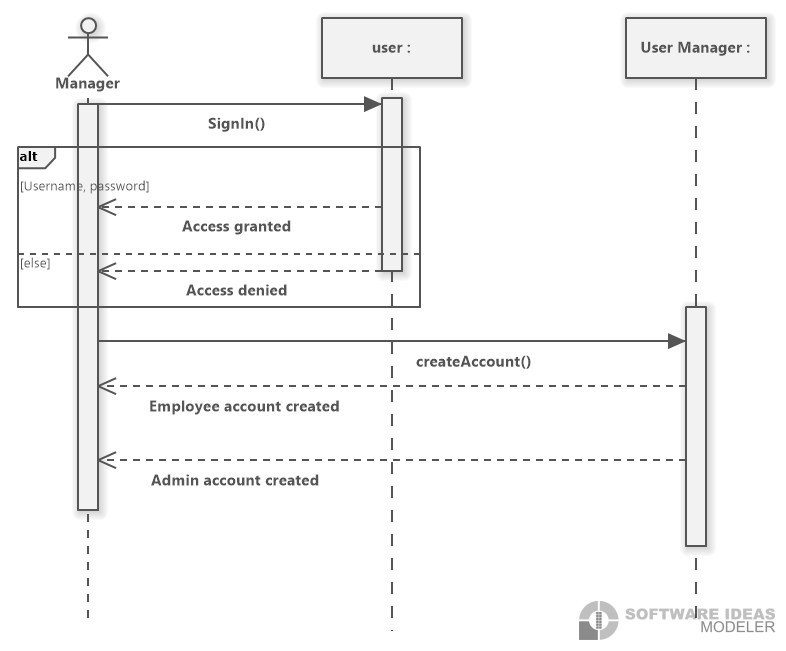
## **Usability Requirements**

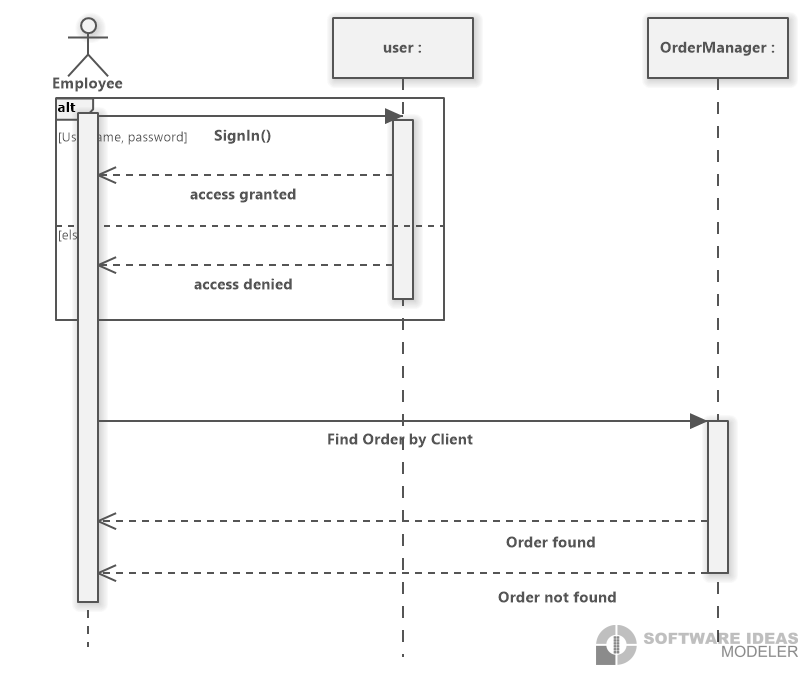
The users of the application will have little to no experience with managing a database. The GUI will be intuitive and uncluttered reducing the confusion for inexperienced users. Avoiding fancy designs and using simple monochromatic color schemes with larger buttons and font signifying importance of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

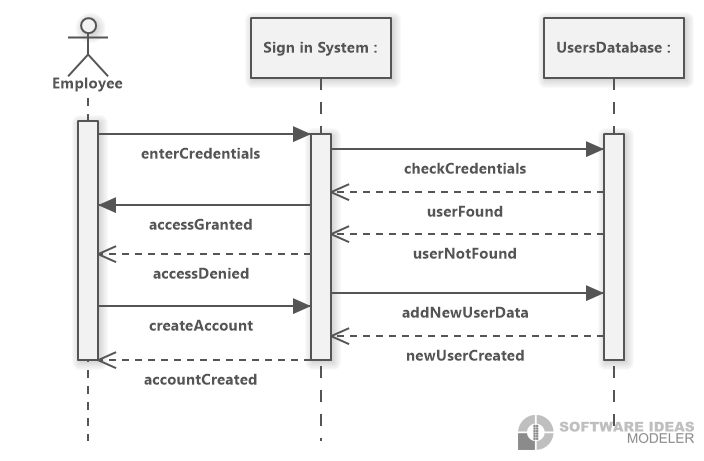
Problem Domain Class Diagram

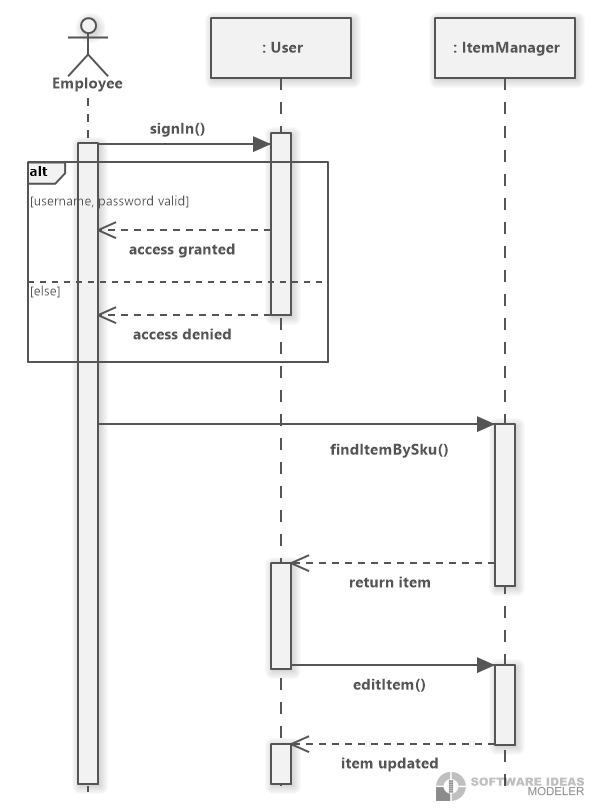


Interaction Sequence Diagrams





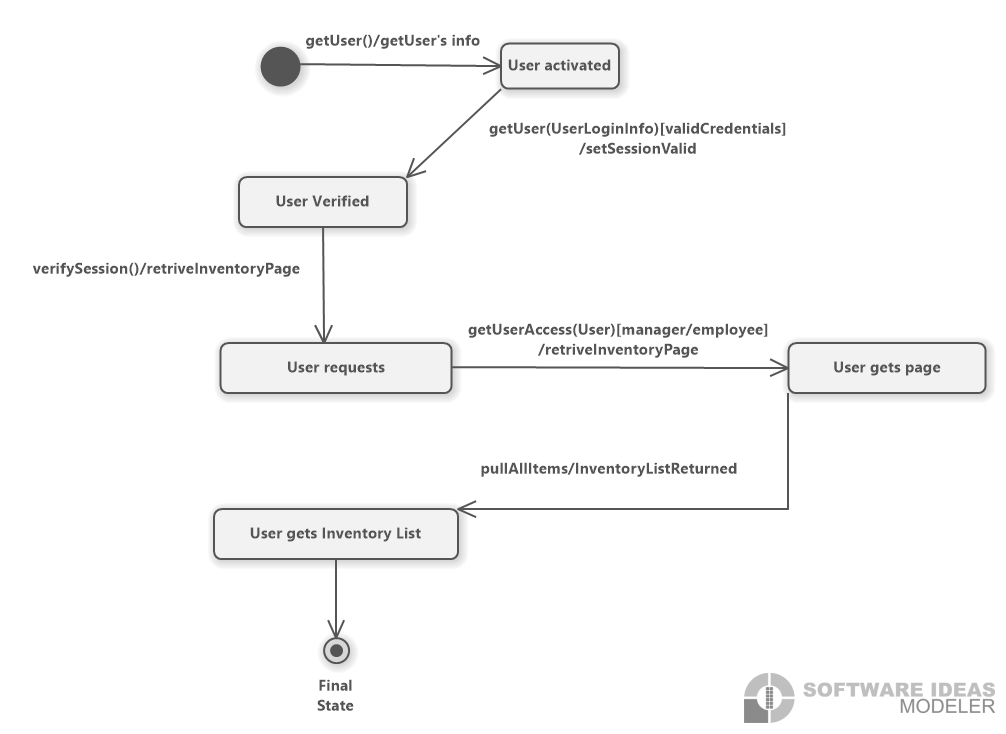




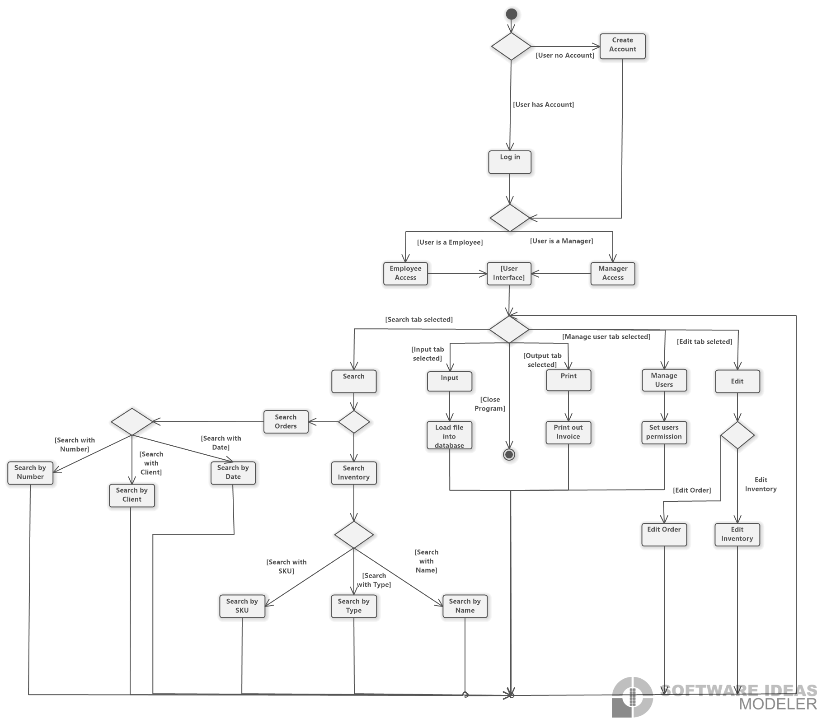
Diagram

Description automatically generated with medium confidence

State Machine Diagrams



Activity Diagrams



# **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, Dylano
* *Back-end developers*: Joshua, Trung, Rajdeep
* *Scribes*: Daniel, Trung
* *Client liaison:* Joshua
* *Team leader:* Dylano

### **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** |  |