Report 2

Part 1.

1. Interaction Diagrams

Figure 1 - Use Case 1

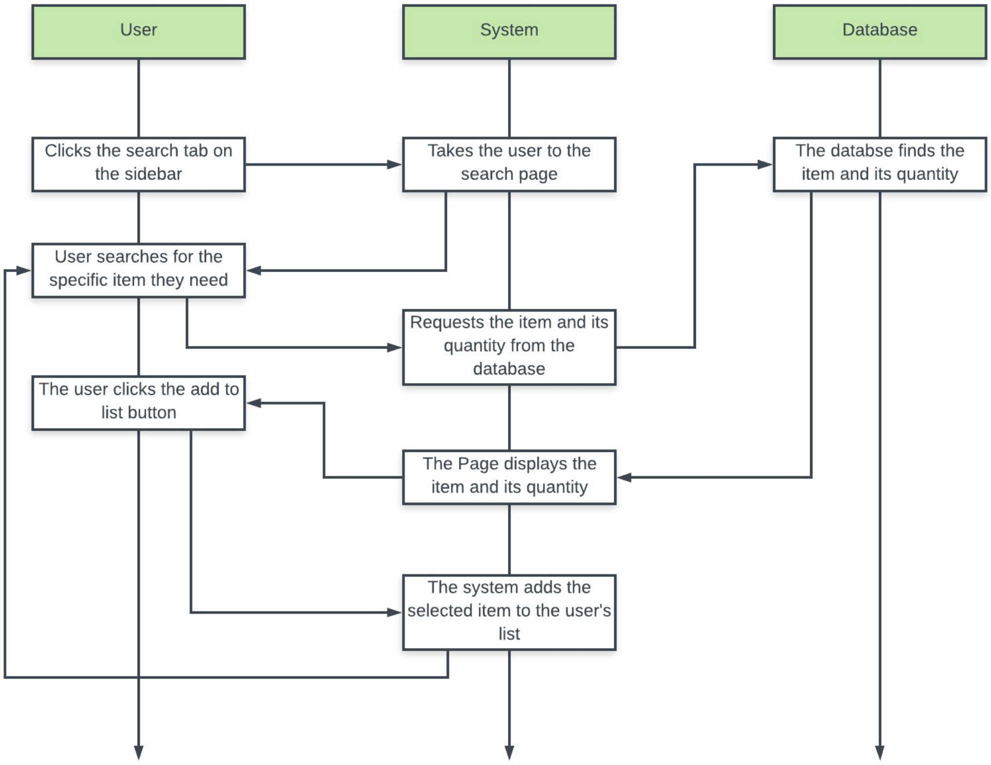

User
System
Database
Clicks the search tab on the sidebar
Takes the user to the search page
The databse finds the item and its quantity
User searches for the specific item they need
Requests the item and its quantity from the database
The user clicks the add to list button
The Page displays the item and its quantity
The system adds the selected item to the user's list

Figure 1 shows the model of use case 1 which is the user adding an item to their list. First the user will click on the sidebar menu button and will click on the "search" tab. This will take them to the search page which then the user will search for the item they want. The system will request the item from the database and the database will send the item information and quantity back to the user. The user will then add the item to their personal list by clicking the "add item" button. The user can repeat this process for as many items as they need.

Figure 2 - Use Case 2

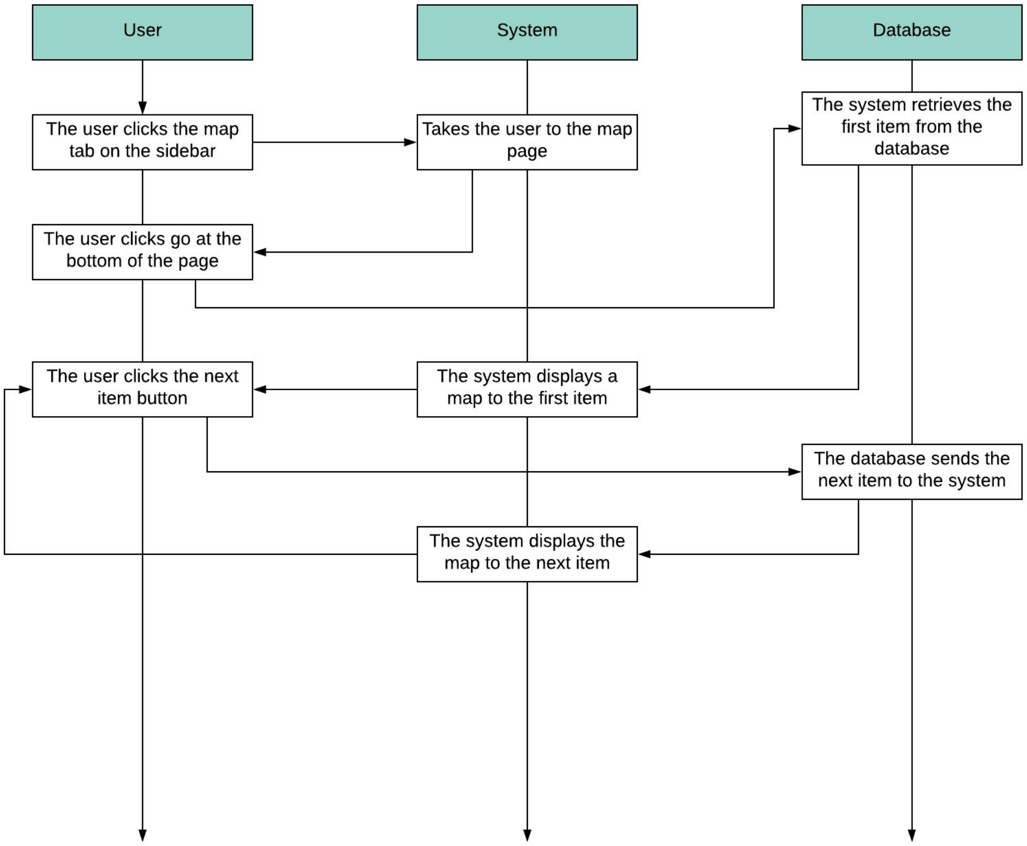

User
System
Database
The user clicks the map tab on the sidebar
Takes the user to the map page
The system retrieves the first item from the database
The user clicks go at the bottom of the page
The user clicks the next item button
The system displays a map to the first item
The database sends the next item to the system
The system displays the map to the next item

Figure 2 shows the model for use case 2 which is the user using the map to find their items. First the user will click the sidebar menu button and click the “map” tab. This will take them to the map page which then the user will use the on screen map to find the easiest path to their item. The user will click the “go” button at the bottom of the page which will retrieve the first item in the list from the database and display the map for the item in the map display box. Once the user has found their item they can click the “next” button and the system will display the map for the next item in the database’s user item list. The user can repeat the next item step until they have found all of the items in their list.

Figure 3 - Use Case 3

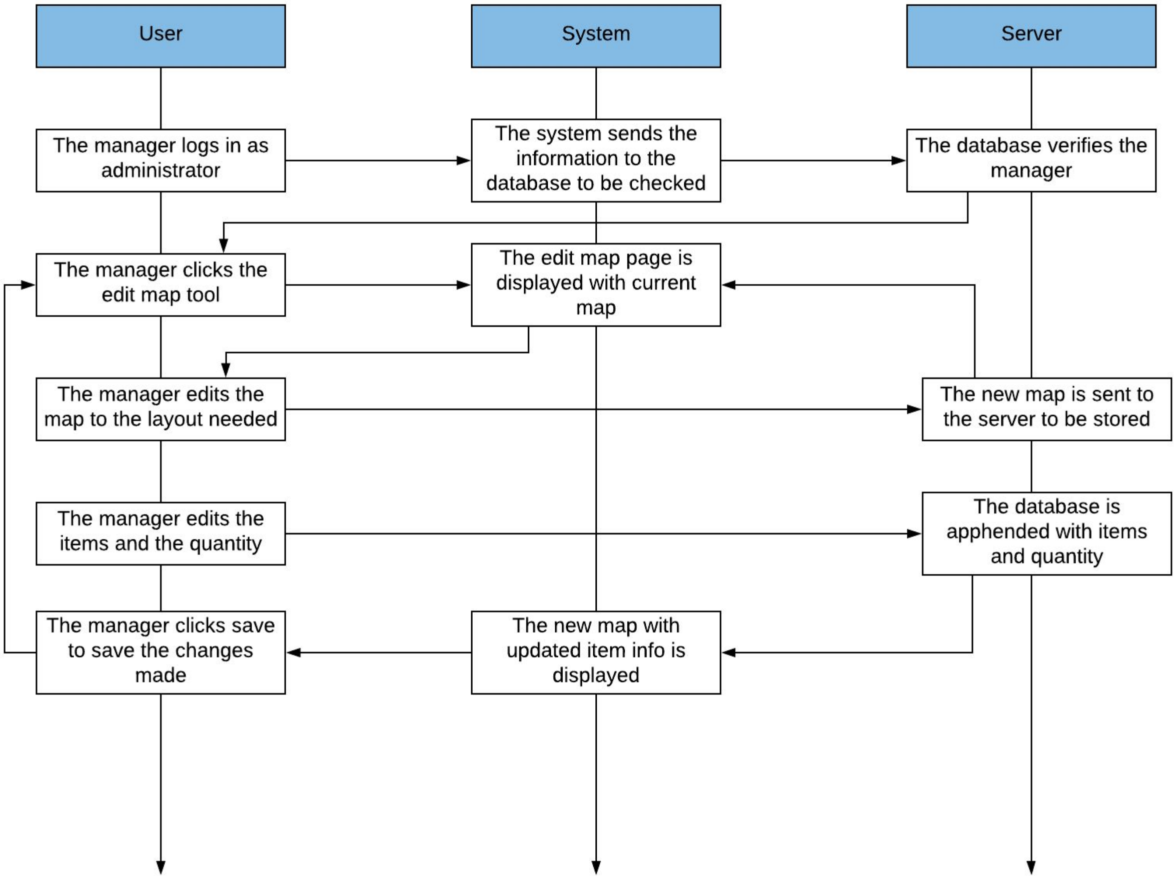

User
System
Server
The manager logs in as administrator
The system sends the information to the database to be checked
The database verifies the manager
The manager clicks the edit map tool
The edit map page is displayed with current map
The manager edits the map to the layout needed
The new map is sent to the server to be stored
The manager edits the items and the quantity
The database is apphended with items and quantity
The manager clicks save to save the changes made
The new map with updated item info is displayed

Figure 3 shows the model for use case 3 which is the manager editing and saving the layout for the base map. First the manager will log in as an administrator, which will be verified by the database. Then the manager will click the “edit map” tool to edit the base map that the user sees. The manager can then edit the map to the layout desired with the given tools. When the layout is to the manager’s liking, the manager will click the “save” button which will send the new map to the server to replace the old one. The manager can then edit the items, their quantity, and where they are place in the store. The manager can click the save button one final time and the final map with all of the updated items will be ready to be used by the user.

Figure 4 - Use Case 4

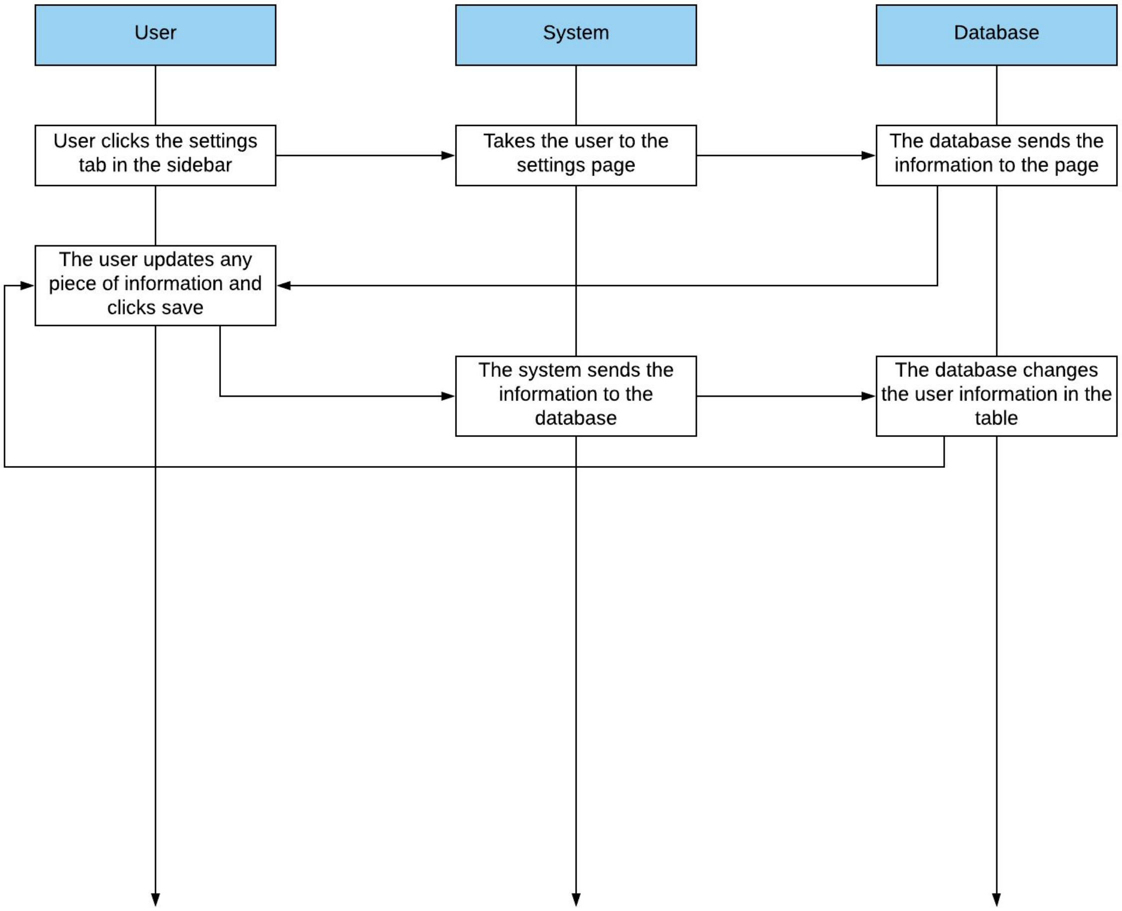
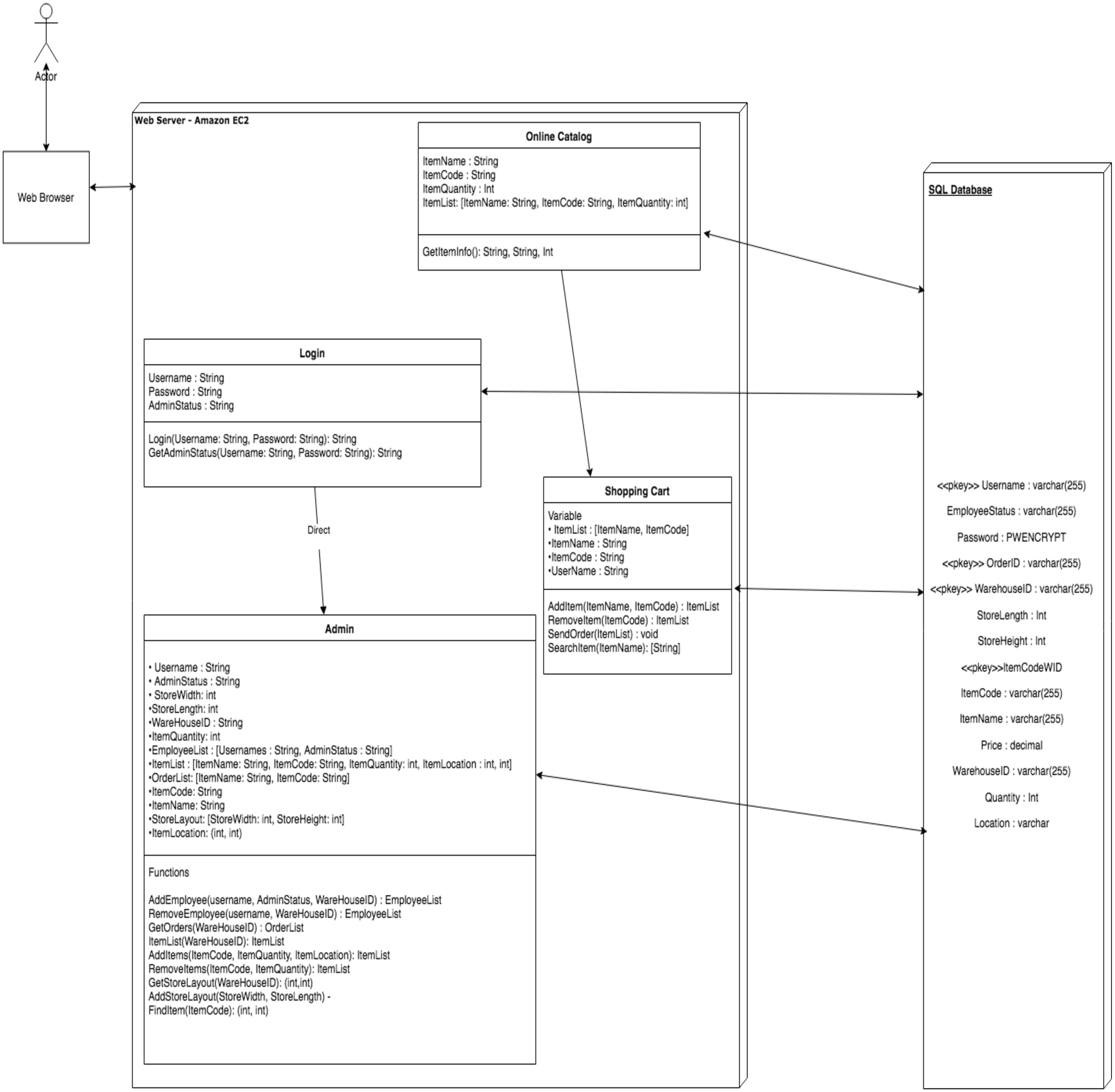

User
System
Database
User clicks the settings tab in the sidebar
Takes the user to the settings page
The database sends the information to the page
The user updates any piece of information and clicks save
The system sends the information to the database
The database changes the user information in the table

Figure 4 shows the model for use case 4 which is the user editing their personal information. The user will click on the “settings” tab in the sidebar which will take them to the corresponding page. The system retrieves the information from the database and displays it on the page. The user can update any information they need to, and will then save that information, which will be sent to the database and saved. The user can repeat this process when need be.

Part 2.

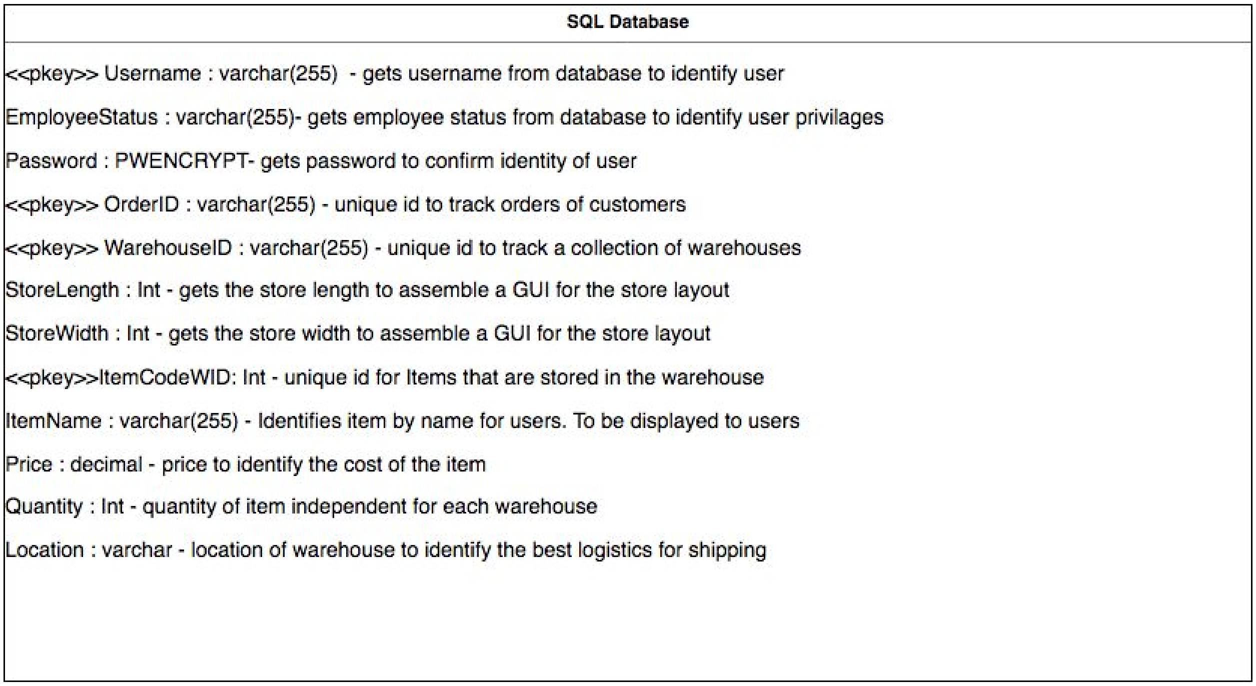
2. Class Diagram and Interface Specification

A. Class Diagram


Actor
Web Browser
Web Server - Amazon EC2
Online Catalog
ItemName : String
ItemCode : String
ItemQuantity : Int
ItemList: [ItemName: String, ItemCode: String, ItemQuantity: int]
GetItemInfo(): String, String, Int
Login
Username : String
Password : String
AdminStatus : String
Login(Username: String, Password: String): String
GetAdminStatus(Username: String, Password: String): String
Direct
Shopping Cart
Variable
• ItemList : [ItemName, ItemCode]
•ItemName : String
•ItemCode : String
•UserName : String
AddItem(ItemName, ItemCode) : ItemList
RemoveItem(ItemCode) : ItemList
SendOrder(ItemList) : void
SearchItem(ItemName): [String]
Admin
• Username : String
• AdminStatus : String
• StoreWidth: int
•StoreLength: int
•WareHouseID : String
•ItemQuantity: int
•EmployeeList : [Usernames : String, AdminStatus : String]
•ItemList : [ItemName: String, ItemCode: String, ItemQuantity: int, ItemLocation : int, int]
•OrderList: [ItemName: String, ItemCode: String]
•ItemCode: String
•ItemName: String
•StoreLayout: [StoreWidth: int, StoreHeight: int]
•ItemLocation: (int, int)
Functions
AddEmployee(username, AdminStatus, WareHouseID) : EmployeeList
RemoveEmployee(username, WareHouseID) : EmployeeList
GetOrders(WareHouseID) : OrderList
ItemList(WareHouseID): ItemList
AddItems(ItemCode, ItemQuantity, ItemLocation): ItemList
RemoveItems(ItemCode, ItemQuantity): ItemList
GetStoreLayout(WareHouseID): (int,int)
AddStoreLayout(StoreWidth, StoreLength) -
FindItem(ItemCode): (int, int)
SQL Database
<<pkey>> Username : varchar(255)
EmployeeStatus : varchar(255)
Password : PWENCRYPT
<<pkey>> OrderID : varchar(255)
<<pkey>> WarehouseID : varchar(255)
StoreLength : Int
StoreHeight : Int
<<pkey>>ItemCodeWID
ItemCode : varchar(255)
ItemName : varchar(255)
Price : decimal
WarehouseID : varchar(255)
Quantity : Int
Location : varchar

B. Data Types and Operation Signatures

This SQL Database will contain 4 different tables. This shows all the distinct variables for all the tables as these will be shared amongst the tables. This Database contains information about the Warehouse, it's workers and it's stock.



SQL Database

<<pkey>> Username : varchar(255) - gets username from database to identify user

EmployeeStatus : varchar(255)- gets employee status from database to identify user privilages

Password : PWENCRYPT- gets password to confirm identity of user

<<pkey>> OrderID : varchar(255) - unique id to track orders of customers

<<pkey>> WarehouseID : varchar(255) - unique id to track a collection of warehouses

StoreLength : Int - gets the store length to assemble a GUI for the store layout

StoreWidth : Int - gets the store width to assemble a GUI for the store layout

<<pkey>>ItemCodeWID: Int - unique id for Items that are stored in the warehouse

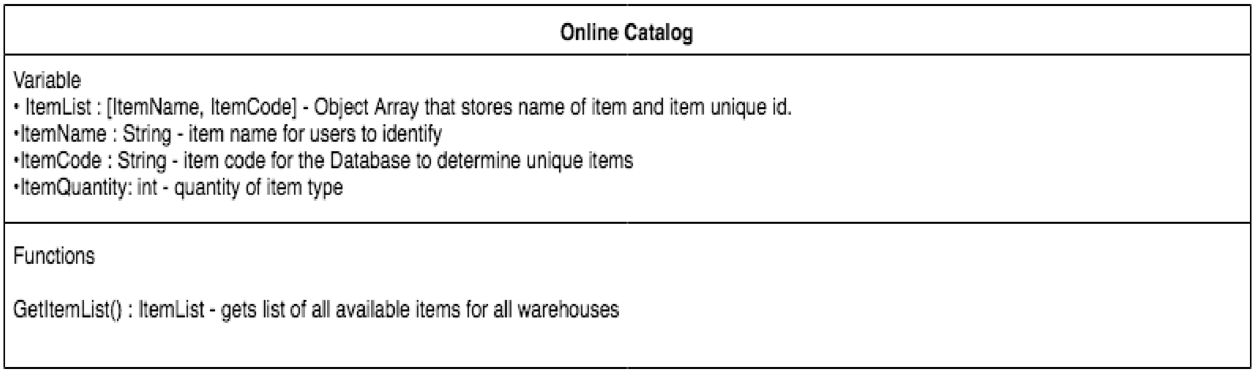
ItemName : varchar(255) - Identifies item by name for users. To be displayed to users

Price : decimal - price to identify the cost of the item

Quantity : Int - quantity of item independent for each warehouse

Location : varchar - location of warehouse to identify the best logistics for shipping

The Online Catalog provides the customers will all available items they can order from the warehouses.



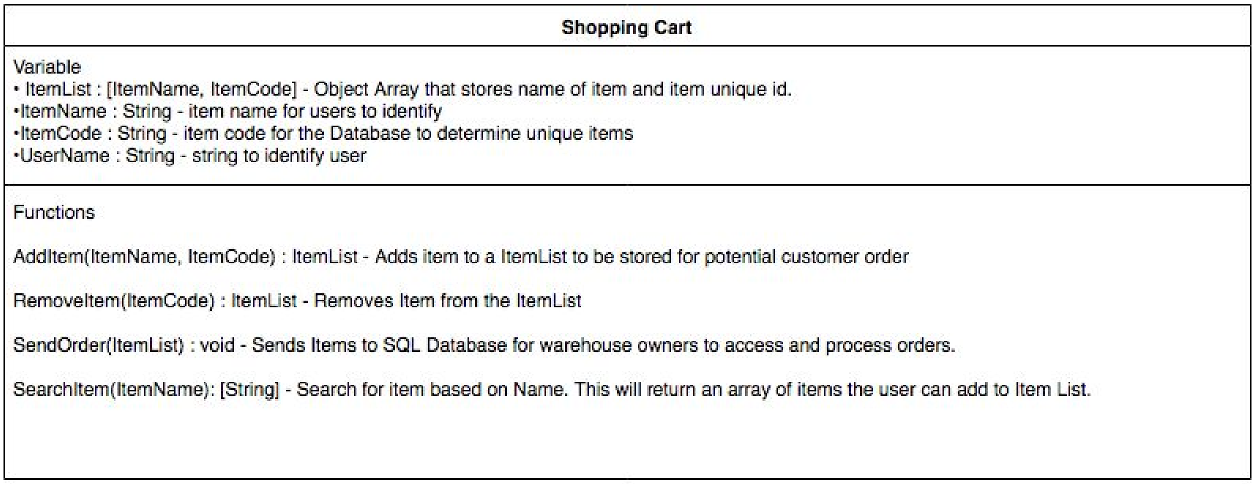
Online Catalog

Variable
- ItemList : [ItemName, ItemCode] - Object Array that stores name of item and item unique id.
- ItemName : String - item name for users to identify
- ItemCode : String - item code for the Database to determine unique items
- ItemQuantity: int - quantity of item type

Functions

GetItemList() : ItemList - gets list of all available items for all warehouses

The shopping cart interface allows customers to pick out items they would like to order and send out the information to any warehouse that can process the order.



Shopping Cart

Variable
• ItemList : [ItemName, ItemCode] - Object Array that stores name of item and item unique id.
•ItemName : String - item name for users to identify
•ItemCode : String - item code for the Database to determine unique items
•UserName : String - string to identify user

Functions

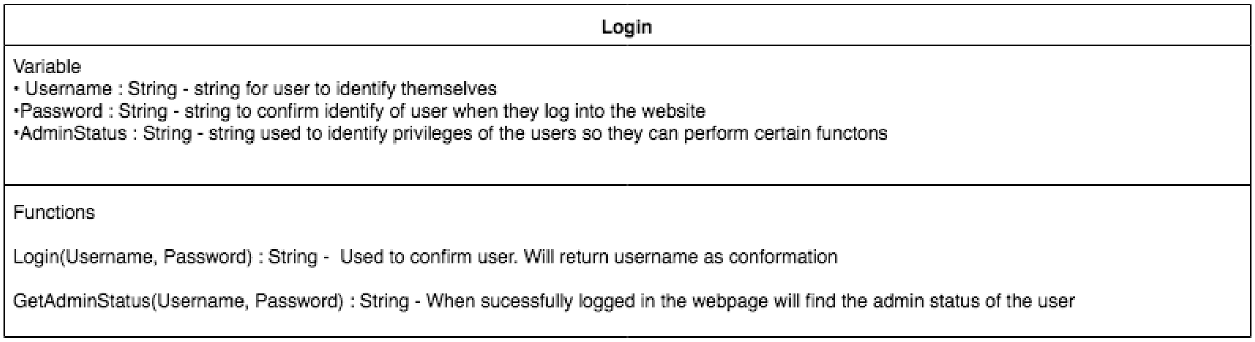
AddItem(ItemName, ItemCode) : ItemList - Adds item to a ItemList to be stored for potential customer order

RemoveItem(ItemCode) : ItemList - Removes Item from the ItemList

SendOrder(ItemList) : void - Sends Items to SQL Database for warehouse owners to access and process orders.

SearchItem(ItemName): [String] - Search for item based on Name. This will return an array of items the user can add to Item List.

The Login Interface is to confirm Employees and Customers. This is meant to grant employees privileges once they are verified.



Login

Variable
• Username : String - string for user to identify themselves
•Password : String - string to confirm identify of user when they log into the website
•AdminStatus : String - string used to identify privileges of the users so they can perform certain functons

Functions

Login(Username, Password) : String - Used to confirm user. Will return username as conformation

GetAdminStatus(Username, Password) : String - When sucessfully logged in the webpage will find the admin status of the user

The Admin Usage Interface is to give employees access to edit the warehouses items, add/remove employees, and process orders.

Admin Usage

Variable

- Username : String - unique id to identify user
- AdminStatus : String - string to identify privileges of user
- StoreWidth: int - int to represent store width to create store layout for GUI
- StoreLength: int - int to represent store length to create store layout for GUI
- WareHouseID : String - String to identify warehoue unique ID
- ItemQuantity: int - int to represent item items count
- EmployeeList : [Usernames, AdminStatus] - List employees for Admin to Customize
- ItemList : [ItemName, ItemCode, ItemQuantity, ItemLocation] - List items in the warehouse
- OrderList: [ItemName, ItemCode] - List Orders of customers
- ItemCode: String - unique id to identify item
- ItemName: String - string to identify item for users
- StoreLayout: [StoreWidth: int, StoreHeight: int] - 2D array to virtualize storelayout
- ItemLocation: [int, int] - represent item location in store layot

Functions

AddEmployee(username, AdminStatus, WareHouseID) : EmployeeList - Adds employee to SQL database, returns new refreshed Employee List.

RemoveEmployee(username, WareHouseID) : EmployeeList - Removes employee from SQL database, returns new Employee List

GetOrders(WareHouseID) : OrderList - Gets List of Orders unique to warehouse from SQL database

ItemList(WareHouseID): ItemList - Gets list of Items unique to warehouse that are stored in SQL database

AddItems(ItemCode, ItemQuantity, ItemLocation): ItemList - Adds Item to warehouse. Stored in SQL database. Returns ItemList

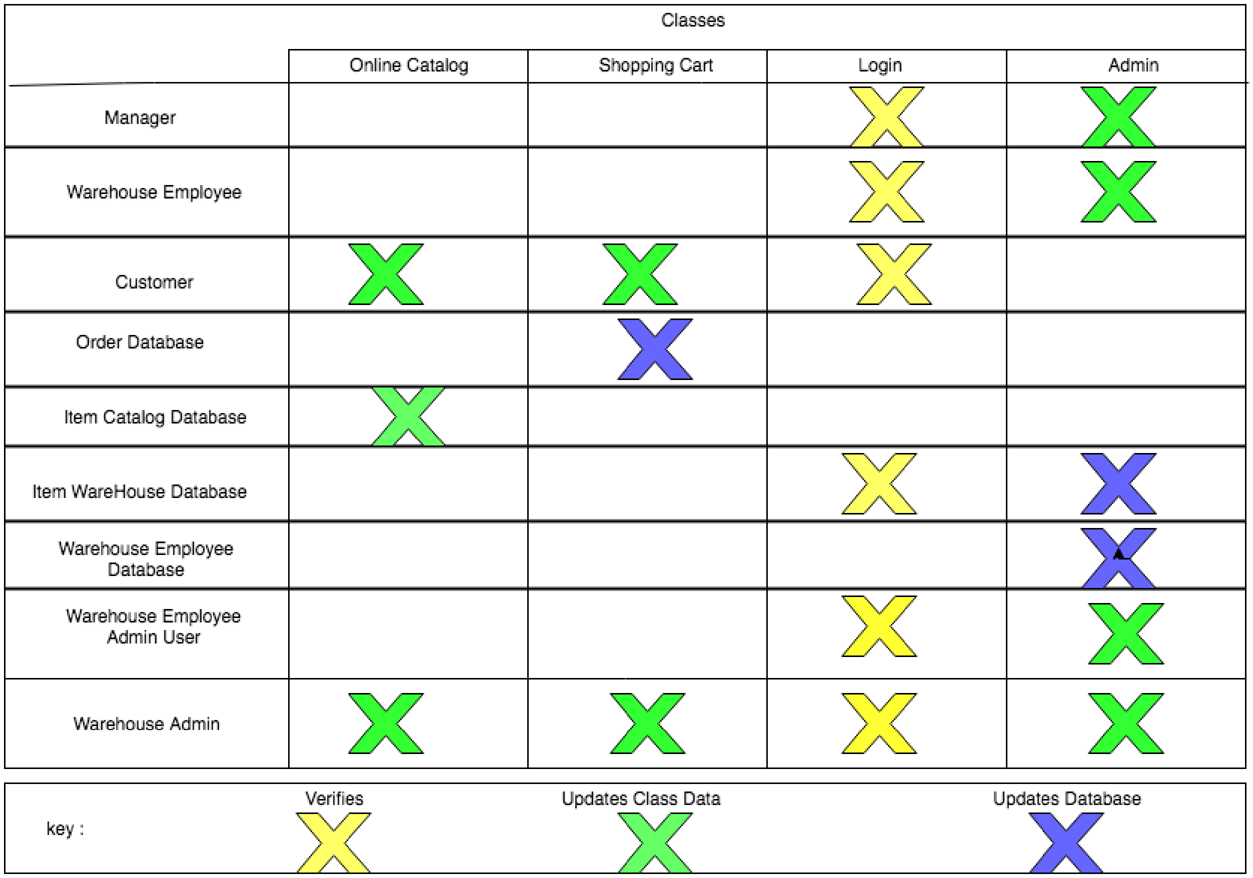
RemoveItems(ItemCode, ItemQuantity): ItemList - Removes Item to warehouse. Stored in SQL database. Returns new ItemList

GetStoreLayout(WareHouseID): StoreLayout - Creates GUI based on storelayout. Returns StoreLayout

AddStoreLayout(StoreWidth, StoreLength) - Edits StoreLayout. Calls GetStoreLayout

FindItem(ItemCode): ItemLocation - finds location of item in GUI Map of store layout

C. Traceability Matrix



	Classes			
	Online Catalog	Shopping Cart	Login	Admin
Manager			🟨	🟩
Warehouse Employee			🟨	🟩
Customer	🟩	🟩	🟨	
Order Database		🟦		
Item Catalog Database	🟩			
Item WareHouse Database			🟨	🟦
Warehouse Employee Database				🟦
Warehouse Employee Admin User			🟨	🟩
Warehouse Admin	🟩	🟩	🟨	🟩

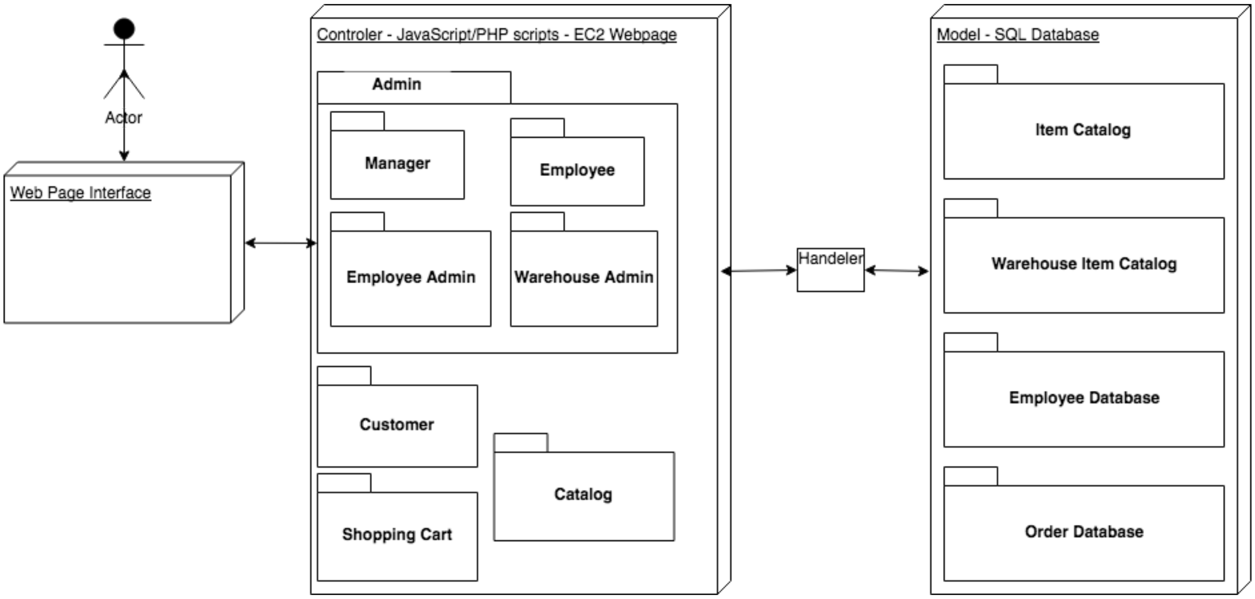
key :	Verifies	Updates Class Data	Updates Database
	🟨	🟩	🟦

The Traceability matrix is meant to represent what aspect of external sources will be updated or notified by each class structure. The Manager, Warehouse Employee, Warehouse Admin, and Customer are human resources and users of our application. In the traceability matrix they are meant to show their current or supposed access to each class. Manager and Employee can use the online catalog but the Admin class provides them with all the tool necessary so they do not need it. Customers are only able to login, view the online catalog and purchase goods. Their orders are sent out to the orders database table. The rest of the database tables that will be updated by each class. There will be a total of 4 different database tables : Item Catalog - for all items in every warehouse, Item Warehouse Database - for all items in the particular warehouse, Warehouse Employee Database - stores employee information and Admin access, Order database - stores orders that need to be distributed to individual warehouses for item processing. Item Catalog updates the Online Catalog class. Warehouse Employee Database check user input from the Login class and is updated by the Admin Class. Item Warehouse Database gets updated by the Admin class. Order Database gets updated by the Shopping Cart class.

3. System Architecture

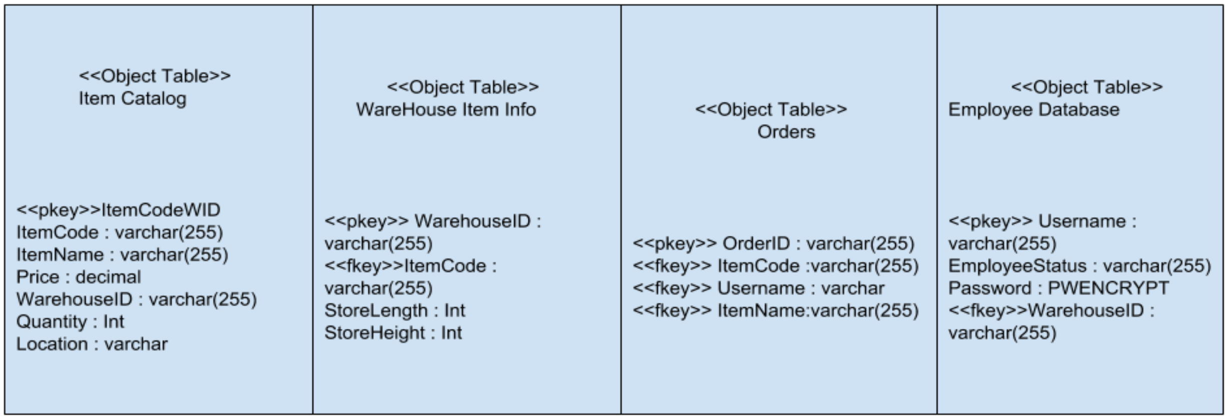
a.) Our architectural style is Event-driven or implicit invocation. The main system (server) relies on inputs from the user (client) in order to determine the action that should be taken. This is true for almost all actions except for pre programmed actions that require no user intervention such as displaying the login screen initially and if the user is already logged in, redirecting to the main page.

b.)



c.) Since our application is web based, that implies that there will be two main actors in a connection. The server and the client. The server contains the database of items created with SQL and the server side php scripts to fulfill client requests. The client will be any device utilizing the services of the server. The client will utilize a clean user interface in order to create requests and receive data from the server. This data will mostly consist of item information that is relevant while shopping such as: item name, price, quantity, and physical location. Well will also implement the EC2 amazon web servers for our controller.

d.) Our application requires persistent data storage to hold the information on each item located in the store. It does this by utilizing a relational database (SQL) located on a remote server.



<<Object Table>> Item Catalog	<<Object Table>> WareHouse Item Info	<<Object Table>> Orders	<<Object Table>> Employee Database
<<pkey>>ItemCodeWID ItemCode : varchar(255) ItemName : varchar(255) Price : decimal WarehouseID : varchar(255) Quantity : Int Location : varchar	<<pkey>> WarehouseID : varchar(255) <<fkey>>ItemCode : varchar(255) StoreLength : Int StoreHeight : Int	<<pkey>> OrderID : varchar(255) <<fkey>> ItemCode :varchar(255) <<fkey>> Username : varchar <<fkey>> ItemName:varchar(255)	<<pkey>> Username : varchar(255) EmployeeStatus : varchar(255) Password : PWENCRYPT <<fkey>>WarehouseID : varchar(255)

e.) The network protocol utilized by our server system is the TCP/IP suite of protocols. More specifically, to find the SQL server the UDP protocol is used. Communication between SQL server and client uses TCP protocol. The client utilizes both HTTP protocol, to connect to the web server, and TCP/IP to transfer data back and forth between the SQL server and the end user.

f.) Our system is event-driven as specified in part a.) under the System Architecture header. Users may generate web application actions in an arbitrary order minus some required initial steps such as logging in. Our system is not time dependent and has no concern for real time. Our system also does not utilize threads or concurrent processing as it is not required for our project.

g.) Our system relies on quite a numerous amount of hardware since it requires a client and server, which is usually two different machines. The server requirements are as follows:

PHP 5.5+ require at least Windows 2008/Vista, or 2008r2, 2012, 2012r2, 2016 or 7, 8, 8.1, 10. Either 32-Bit or 64-bit (aka X86 or X64. PHP does not run on Windows RT/WOA/ARM). As of PHP 7.2.0 Windows 2008 and Vista are no longer supported. PHP requires the Visual C runtime(CRT). Many applications require that so it may already be installed. PHP 5.5 and 5.6 require VC CRT 11 (Visual Studio 2012)

Since SQL will also be present on the server, the system requirements for that are:

Memory Requirements: a minimum of 1 GB, but optimally your machine should have 4 or more GB of memory as database size increases to ensure optimal performance.

Processor Speed: a minimum of 64 bit processor and clock speed of 1.4 GHz is required, but the preferred is a 64 bit processor that is 2.0 GHz or faster.

Processor Types: The list of acceptable processor types are: AMD Opteron, AMD Athlon 64, Intel Xeon with Intel EM64T support, Intel Pentium IV with EM64T support

Disk Space Requirements:

Database Engine and data files, Replication, Full-Text Search, and Data Quality Services	1480 MB
Database Engine (as above) with R Services (In-Database)	2744 MB
Database Engine (as above) with PolyBase Query Service for External Data	4194 MB
Analysis Services and data files	698 MB
Reporting Services	967 MB
Microsoft R Server (Standalone)	280 MB

Reporting Services - SharePoint	1203 MB
Reporting Services Add-in for SharePoint Products	325 MB
Data Quality Client	121 MB
Client Tools Connectivity	328 MB
Integration Services	306 MB
Client Components (other than SQL Server Books Online components and Integration Services tools)	445 MB
Master Data Services	280 MB
SQL Server Books Online Components to view and manage help content*	27 MB
All Features	8030 MB

The system requirements for the client are significantly less. The only client requirement is a functional operating system and web browser. Chrome seems to be the biggest memory hog of all web browsers currently so we will use that as a baseline for our client requirements. Those requirements are, Google Chrome will run on computers equipped with a Pentium 4 processor or higher, which encompasses most machines manufactured since 2001. The computer must have approximately 100MB of free hard drive space and 128MB of RAM. The oldest version of Windows supported by Chrome is Windows XP with Service Pack 2 installed. Chrome also runs on computers with Windows Vista or Windows 7 installed.

Part 3.

4. Algorithms and Data Structures

A.) Algorithms

Most of our application does not rely on algorithms as is more dependant on our database structure. We plan on introducing two quality algorithms to our application. The first algorithm will be implementing store layout to match and fit the webpage. When we return the store width and height form out database we plan on creating a GUI that can change with each different store layout. We plan on use a basic mathematical formula where each 5 ft squared unit of the layout will become a squared grid on the GUI. We will set a max width and length for the GUI and have the squared grids resize accordingly.

E.g. Store Layout: Width = 250ft Length = 400 → 50x80 grid is then created and each grids size will be adjusted accordingly to the allocated size for the GUI

GUI length = Length / 5
GUI Width = Length /5

After that we plan on introducing a system to implement an algorithm to find any item in that warehouse in the fastest time. We will do this by getting the items location on the grid and the location of the employee. The location of the employee can be put anywhere on the GUI map. From there we will use Dijkstra's algorithm to find the best available path for the employee to take. The GUI will then display the route the employee should take.

B.) Data Structures

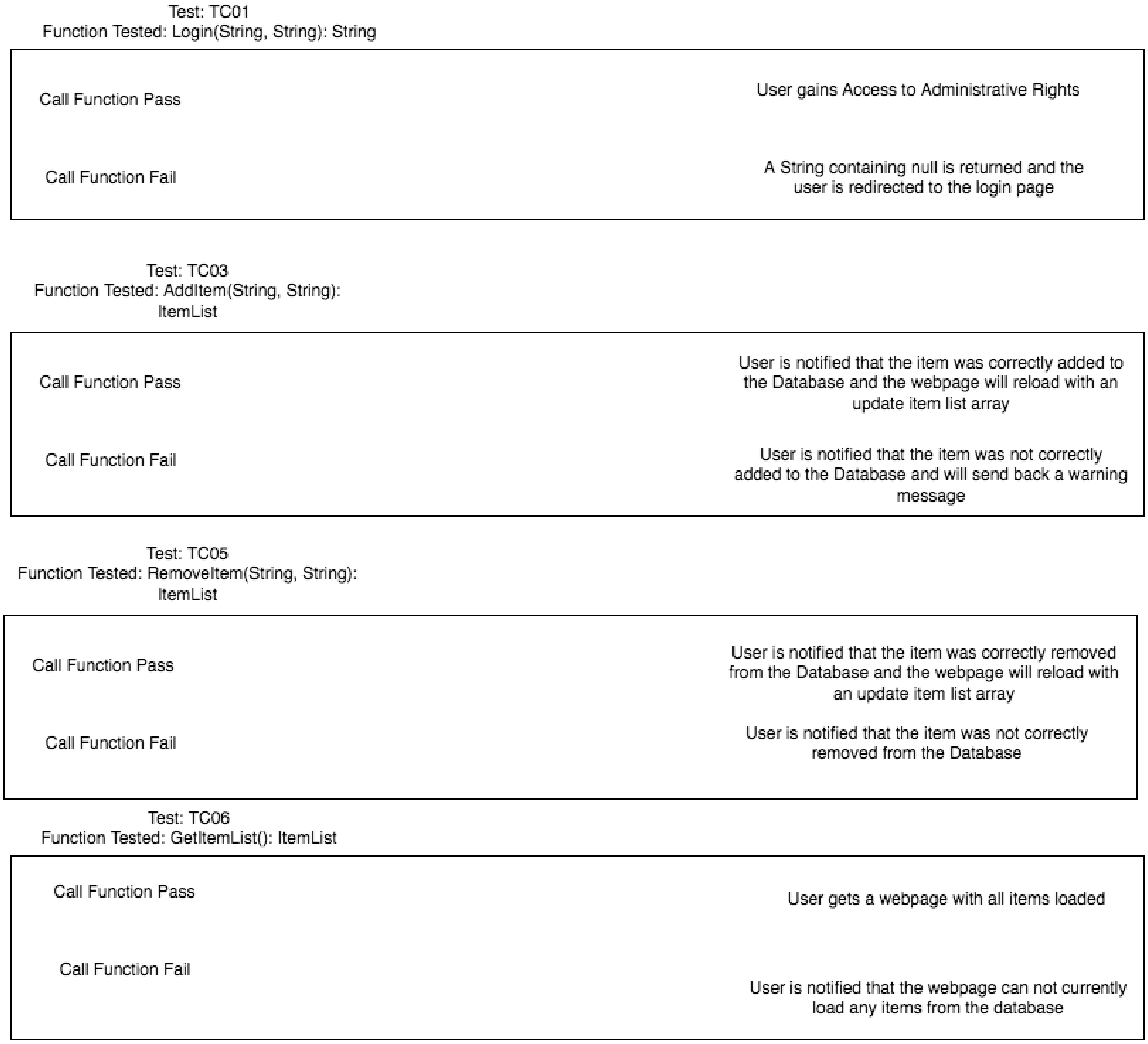
Since we will be mainly using PHP and Javascript for our controller's data structures we will not have any complex data structures there. Mostly we will create Objects such as an

Item List that contain two strings and an integer. These Objects include ItemList[String, String, int], ItemLocation[int, int], and StoreLayout[int, int].

5. UI Design and Implementation

The biggest changes in design are the list and search pages and the sidebar menu. The list page has become the initial splash page once a user logs in. It will still display the list as shown and have a go button at the bottom which will take the user to the map page. The search page will have a large search bar under the page title which will allow the user to search for any item they need. The list of searched items will then show up below the persistent search bar. The sidebar menu can retract to be hidden and then activated with a press of the menu button, it will not be a persistent menu on the side of the page. Each page will be simple to use and will only need a few clicks to get the user where they need to go.

6. Design of Test



Test: TC01
Function Tested: Login(String, String): String

Call Function Pass	User gains Access to Administrative Rights
Call Function Fail	A String containing null is returned and the user is redirected to the login page

Test: TC03
Function Tested: AddItem(String, String): ItemList

Call Function Pass	User is notified that the item was correctly added to the Database and the webpage will reload with an update item list array
Call Function Fail	User is notified that the item was not correctly added to the Database and will send back a warning message

Test: TC05
Function Tested: RemoveItem(String, String): ItemList

Call Function Pass	User is notified that the item was correctly removed from the Database and the webpage will reload with an update item list array
Call Function Fail	User is notified that the item was not correctly removed from the Database

Test: TC06
Function Tested: GetItemList(): ItemList

Call Function Pass	User gets a webpage with all items loaded
Call Function Fail	User is notified that the webpage can not currently load any items from the database

C.)Testing Strategy

We plan on testing each of these functions separately. We will conduct each individual test and examine every possible outcome. We will make sure the user will not have much text input to prevent an even wider margin of error.

7. Project Management and Plan of Work

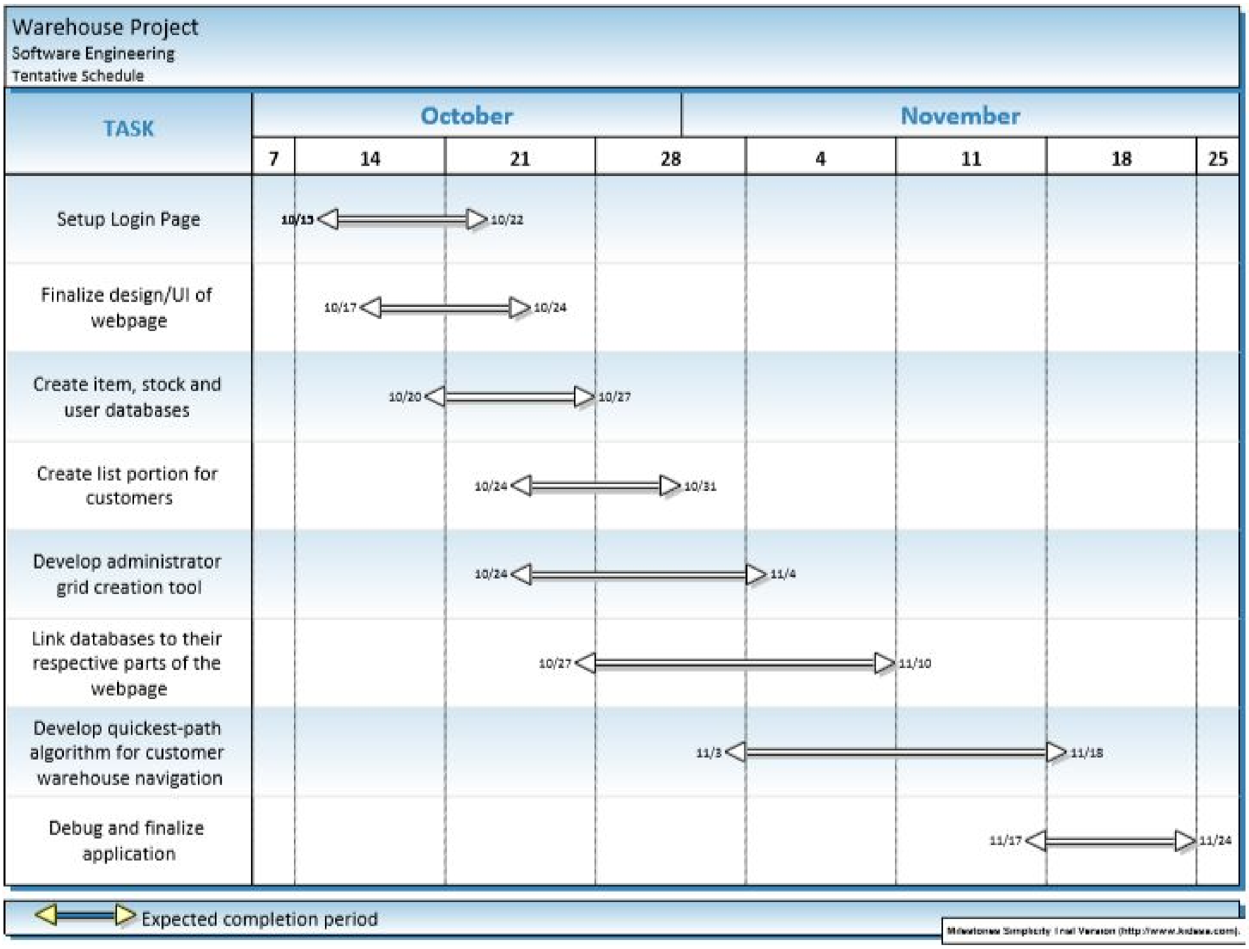
A.) Merging and Collecting Contributions

Everyone made sure to contribute to the project in the most synchononimous way. We made sure that all of our work was available to everyone else in the team by providing our share on Google docs and Github. Github holds our code for our project and everyone has their own branch and we put our documentation on Google Docs.

B.) Project Coordination

Every week we will meet up and coordinate our next plan of action. We currently have gotten the UI finished for the main webpage, we finished our Login functions and our working on our items list functions. All of our database tables have been established but we know we might have to drop the tables sometime and re-establish them if our project will need more information.

C.)



D.) BreakDown of Responsibilities:

Parker Jones : Hardware implementation, Javascript implementation of algorithms, PHP implementation

Skyler Todd: UI implementation includes: interactive controls and documentation leader

Liam Flaherty: PHP implementation, web server access, JQuery to PHP communication

Christian Reed: Database creation and formulation, database access, Project Management

Mitchell Hoffman: Javascript and UI implementation.

Michael Ehnes: Javascript Implementation of store layout and item ordering, Diagram leader

8. References

- Decker, Fred. “Google Chrome Software Requirements.” Small Business - Chron.com, Chron.com, 26 Oct. 2016, smallbusiness.chron.com/google-chrome-software-requirements-48820.html.
- “Install Requirements - Manual.” Php, php.net/manual/en/install.windows.requirements.php.
- MashaMSFT. “Hardware and Software Requirements for Installing SQL Server 2016.” 2016 | Microsoft Docs, docs.microsoft.com/en-us/sql/sql-server/install/hardware-and-software-requirements-for-installing-sql-server?view=sql-server-2017#pmosr.
- http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Melissa.pdf