# Acknowledgement

This research was supported by the BSCSD of computing and software engineering. Well, I Might not have completed the assignment successfully without the support of a few people and also some forums. Especially I would like to convey my sincere thanks to our module lecturer and as well as the assignment assessor Mr Tharik Kanaka for supporting not only me, but our whole batch impartially with the assignment tasks, Providing the necessary guidance, Assistance and also sharing his immense knowledge with us. Concluding my Description, I would like to thank our institute management and its workers for providing us the facilities we need, and also, I would thank my fellow mates and my family for their support and the guidance that helped me to complete the assignment successfully.

Thank you!

Yours Sincerely

Chamal Peiris

Table of Contents

[Acknowledgement 1](#_Toc65229615)

[TASK A 8](#_Toc65229616)

[SRS (System Requirement Specification for JK Company) 8](#_Toc65229617)

[Introduction 8](#_Toc65229618)

[Functional Requirement of the Web Application 8](#_Toc65229619)

[Functionalities of the Head Office Users 8](#_Toc65229620)

[Functionalities of the Branch Users 9](#_Toc65229621)

[Non Functional requirements of the application 9](#_Toc65229622)

[TASK B Class Diagram 10](#_Toc65229623)

[Explanation 11](#_Toc65229624)

[Classes 11](#_Toc65229625)

[Database Connection 11](#_Toc65229626)

[Methods 11](#_Toc65229627)

[Attributes 11](#_Toc65229628)

[Inheritance 11](#_Toc65229629)

[Relationships(Composition) 11](#_Toc65229630)

[Multiplicity 12](#_Toc65229631)

[Use-Case Diagram 13](#_Toc65229632)

[Use-Case Part 1 13](#_Toc65229633)

[13](#_Toc65229634)

[Use-Case Part(2) 14](#_Toc65229635)

[Explanation 15](#_Toc65229636)

[Actors 15](#_Toc65229637)

[Use-cases 15](#_Toc65229638)

[Sequence Diagram 16](#_Toc65229639)

[Lifelines 16](#_Toc65229640)

[Activation Bars 16](#_Toc65229641)

[Actors 16](#_Toc65229642)

[Messages 16](#_Toc65229643)

[Head-Office Admin Sequence 17](#_Toc65229644)

[17](#_Toc65229645)

[Branch Admin Sequence 18](#_Toc65229646)

[Branch Admin Sequence -Part 1 18](#_Toc65229647)

[Branch Admin Sequence -Part 2 19](#_Toc65229648)

[ER Diagram 20](#_Toc65229649)

[TASK C 22](#_Toc65229650)

[What are Design Patterns? 22](#_Toc65229651)

[Identifying different design pattern and their advantages 22](#_Toc65229652)

[Singleton Design Pattern 22](#_Toc65229653)

[Factory Design Pattern 22](#_Toc65229654)

[According to (JavaTPoint, 2018), Factory pattern simply defines an interface creating and object probably a method and the other classes, the subclasses could make a choice of what class to instantiate or of what class should they create and object. 22](#_Toc65229655)

[ It could reduce the runtime and memory allocation since, only the resultant interface or the abstract class interferes with the object created, it will not run the rest of the other classes during runtime if they are not instantiated. 22](#_Toc65229656)

[ When u have factory, you do not need to write the same code in several locations. Rather, simply by calling the instance of the corresponding interface class it would perform its task. This could improvise the efficiency 22](#_Toc65229657)

[Abstract Factory Design Pattern 23](#_Toc65229658)

[ It will facilitate exchanging objects. 23](#_Toc65229659)

[Applying the Suitable Design Pattern 24](#_Toc65229660)

[Evaluate the impact of Design patterns 26](#_Toc65229661)

[ Applying design patterns into a development allows or makes it reusable in several projects but not just stick into one. 26](#_Toc65229662)

[TASK D 27](#_Toc65229663)

[System Explanation 27](#_Toc65229664)

[Distributed (3 tier) Application 27](#_Toc65229665)

[Webservice Methods 28](#_Toc65229666)

[Classes in the Web Service 29](#_Toc65229667)

[Dbutil class 30](#_Toc65229668)

[Webhelper Class 31](#_Toc65229669)

[Object Oriented Principles Used 32](#_Toc65229670)

[Encapsulation 32](#_Toc65229671)

[Inheritance 32](#_Toc65229672)

[Polymorphism 33](#_Toc65229673)

[Sending Emails 34](#_Toc65229674)

[Code 34](#_Toc65229675)

[User Login 36](#_Toc65229676)

[Interface 36](#_Toc65229677)

[Head Office Homepage 38](#_Toc65229678)

[Branch Home page 38](#_Toc65229679)

[Head office Manage Requests 39](#_Toc65229680)

[Interface 39](#_Toc65229681)

[Code 39](#_Toc65229682)

[Database when a request is accepted 40](#_Toc65229683)

[Branch Manage requests 41](#_Toc65229684)

[Make a Request 41](#_Toc65229685)

[Confirm Delivery 43](#_Toc65229686)

[Branch Accept request 44](#_Toc65229687)

[TASK E 46](#_Toc65229688)

[How test driven development is used in the system 46](#_Toc65229689)

[Checking the getUser() method 47](#_Toc65229690)

[Checking the updateUser() method 48](#_Toc65229691)

[Checking the addUser() method 49](#_Toc65229692)

[Checking the checkUser() method 50](#_Toc65229693)

[Checking the getUsers() method 51](#_Toc65229694)

[Checking the deleteUser() method 52](#_Toc65229695)

[Checking the addBranch() method 53](#_Toc65229696)

[Checking the getBranch() method 54](#_Toc65229697)

[Checking the getOneBranch() method 55](#_Toc65229698)

[Checking the deleteBranch() method 56](#_Toc65229699)

[Checking the addProduct() method 57](#_Toc65229700)

[Checking the addProduct() method 58](#_Toc65229701)

[Checking the addBranchProducts() method 59](#_Toc65229702)

[Checking the getALLHeadofficeProducts() method 60](#_Toc65229703)

[Checking the Getsingleheadofficeproduct() method 61](#_Toc65229704)

[Checking the Deleteheadofficeproduct() method 62](#_Toc65229705)

[Checking the GetbranchProducts() method 63](#_Toc65229706)

[Checking the ValidatebranchProducts() method 64](#_Toc65229707)

[Checking the test AddStockRequest() method 65](#_Toc65229708)

[Checking the test Getstockrequest() method 66](#_Toc65229709)

[Checking the test GetStockSendingBranchID()method 67](#_Toc65229710)

[Checking the test GetStockSendingbranchStockamount() method 68](#_Toc65229711)

[Checking the test GetStockrecievingbranchAmount() method 69](#_Toc65229712)

[Checking the GetbranchRequest() method 70](#_Toc65229713)

[Checking the UpdateStcockRequets() method 71](#_Toc65229714)

[Checking the Getpendingrequest() method 72](#_Toc65229715)

[Checking the Updatestocksendingamount() method 73](#_Toc65229716)

[Checking the HeadofficestockAmount() method 74](#_Toc65229717)

[Checking the testUpdatebranchnewstock() method 75](#_Toc65229718)

[Checking the AddCustomer() method 76](#_Toc65229719)

[Checking the Getcustomer() method 77](#_Toc65229720)

[Checking the AddtoCart() method 78](#_Toc65229721)

[Checking the GetCart() method 79](#_Toc65229722)

[Checking the GetcustomerCart()method 80](#_Toc65229723)

[Checking the UpdateCart() method 81](#_Toc65229724)

[Checking the GetBranchsales() method 82](#_Toc65229725)

[Checking the AddDriver() method 83](#_Toc65229726)

[Checking the GetAllDrivers() method 84](#_Toc65229727)

[Checking the DeleteDriver() method 85](#_Toc65229728)

[Checking the Addehicle() method 86](#_Toc65229729)

[Checking the GetAllVehicles() method 87](#_Toc65229730)

[Checking the DeleteVehcile() method 88](#_Toc65229731)

[TASK F 89](#_Toc65229732)

[User Documentation 89](#_Toc65229733)

[User Login 89](#_Toc65229734)

[Head-Office Homepage 90](#_Toc65229735)

[Manage Account 90](#_Toc65229736)

[Manage Users 91](#_Toc65229737)

[Add User 91](#_Toc65229738)

[View/Delete All Users 92](#_Toc65229739)

[Add Branch 92](#_Toc65229740)

[View All/ Delete Branch 93](#_Toc65229741)

[Manage Products 94](#_Toc65229742)

[Add Products 94](#_Toc65229743)

[View All/delete Products 94](#_Toc65229744)

[Stock Requests 95](#_Toc65229745)

[Branch Homepage 96](#_Toc65229746)

[Manage Account 96](#_Toc65229747)

[Manage Customers 97](#_Toc65229748)

[Customer Shop Products 98](#_Toc65229749)

[Searching an Available Customer 99](#_Toc65229750)

[Searching an Invalid Customer 99](#_Toc65229751)

[Adding a product to cart 100](#_Toc65229752)

[View cart 101](#_Toc65229753)

[View sales 102](#_Toc65229754)

[Manage Drivers 103](#_Toc65229755)

[Add Driver 103](#_Toc65229756)

[View All/ Delete driver 103](#_Toc65229757)

[Manage Vehicles 104](#_Toc65229758)

[Add Vehicles 104](#_Toc65229759)

[View all/ Delete vehicle 104](#_Toc65229760)

[Available stock 105](#_Toc65229761)

[Manage Requests 106](#_Toc65229762)

[Make a Request 106](#_Toc65229763)

[Confirm delivery 106](#_Toc65229764)

[Other branch requests 107](#_Toc65229765)

[IF same branch tries to accept their own request 108](#_Toc65229766)

[If the requested product not available 108](#_Toc65229767)

[Technical Documentation 109](#_Toc65229768)

[Importing Files 109](#_Toc65229769)

[Configuring the Server 109](#_Toc65229770)

[JDK Configuration 109](#_Toc65229771)

[Using The Web Client 109](#_Toc65229772)

[Starting the Project 110](#_Toc65229773)

[Login URL 110](#_Toc65229774)

[Using The Web Service 110](#_Toc65229775)

[Configuring the Database 111](#_Toc65229776)

[Starting the Project 113](#_Toc65229777)

[TASK G 114](#_Toc65229778)

[Identifying the ethical Issues 114](#_Toc65229779)

[Analyzing the ethical considerations 114](#_Toc65229780)

[Are the software’s used cracked versions or free versions? 114](#_Toc65229781)

[Does the system provide safety for the stakeholders information(privacy)? 114](#_Toc65229782)

[How supportive was the community around during the period of development? 115](#_Toc65229783)

[Addictive design of user interfaces (UX engineering) 115](#_Toc65229784)

[Tools and functions used to achieve software carpentry 115](#_Toc65229785)

[Tools and functions used to achieve software Codemanship 116](#_Toc65229786)

[References 117](#_Toc65229787)

# TASK A

# SRS (System Requirement Specification for JK Company)

## Introduction

Jk Company currently has no specific mean of communication or checking details like customers and products available in the branch. Therefore the proposed solution will cover up all the aspects that the company requires from the developer for the maximum customer satisfaction. And the solution will consist of a combination of two applications which is a client application and a web service.

### Functional Requirement of the Web Application

In the proposed solution there will be two types of user who access the system

* Head Office Users
* Branch Users

### Functionalities of the Head Office Users

* **Login**:-Login via the unique username and password
* **Manage Account**:-Update the user details including passwords.
* **Manage Other Users**: Should be able to create or delete users and assign them into the relevant branch
* **Manage Branch**:-Should be able to add or delete branches
* **Manage Products**:-should be able to update and delete Products
* **Stock Requests**:-Should be able to view and accept other branches stock requests.
* **Email confirmation**:-should be able to send and email the requested branch once the request is accepted

### Functionalities of the Branch Users

* **Login**:-Login via the unique username and password
* **Manage Account**:-Update the user details including passwords.
* **Manage Customer**:-should be able to add and delete a customer
* **Shop Products**:-should be able to shop for products after selecting the relevant customer.
* **Email confirmation**:-should be able to send an email to the relevant customer once checked out
* **View Cart**:-Should be abler to view the cart of products once added.
* **View sales**:-should be able to view sales reports.
* **Manage Driver**:-should be able to add or delete a driver
* **Manage Vehicle**:-should be able to add or delete a vehicle.
* **View available stock**: should display the amount of products available in the particular branch
* **Manage request**: should be able to send a stock request and confirm that the requested goods are received or not
* **Other branch requests**:-should be able to view and accept other branches stock requests if the requested products are available.

### Non Functional requirements of the application

* **Performance:** Web application should be delayed and interactive. Which mean the site shouldn’t take much time when loading.
* **Usability:** Web application should not be complex to the user and should be easy to handle.

And also, it is better if the system has user friendly messages also.

* **Reliability :** If there is an error it should be clear and should be strategy for correction.
* **Availability –:**Web application should be available 24/7 to the user.

# TASK B Class Diagram

## Explanation

### Classes

1. Basically, The Class Diagram has 12 classes. The DB Connection class, Vehicles, Users etc.. includes various methods and variables in order to perform tasks. These classes were created by the developer. All its attributes are kept as private so that only the class can access them . the (-) sign represent the privacy.
2. The Dbutil class and the Webhelper classes are quite different where the Dbutil class resides in the webservice and perform various actions related to the database whereas the webhelper class performs the same task residing in the client side.

### Database Connection

1. The DB connection class Plays the Major role where the Dbutil gets the MySQL database connection via the dB Connection class.

### Methods

5. All the Methods of the class are kept as Public so other classes too can access them.

### Attributes

6. The Attributes of each class are kept as private so that only the class can access them

### Inheritance

7. There is an Inheritance between the Users class and the Customer class and between the users class and the drivers class where both the child classes inherits the Fname and Lname getter and setters from their parent User class.

### Relationships(Composition)

8. The Entire Class Diagram Only consists of composition relationships. Where all the classes are correlated to each Relationship. For instance, the Dbutil class cannot exist without the dbconnection class

### Multiplicity

1. There is only a (1-1) multiplicity seen throughout the diagram. For instance, there will be only one database connection class which supports one dbtUtil class and similarly there will be only one User class that supports the Dbutil class.

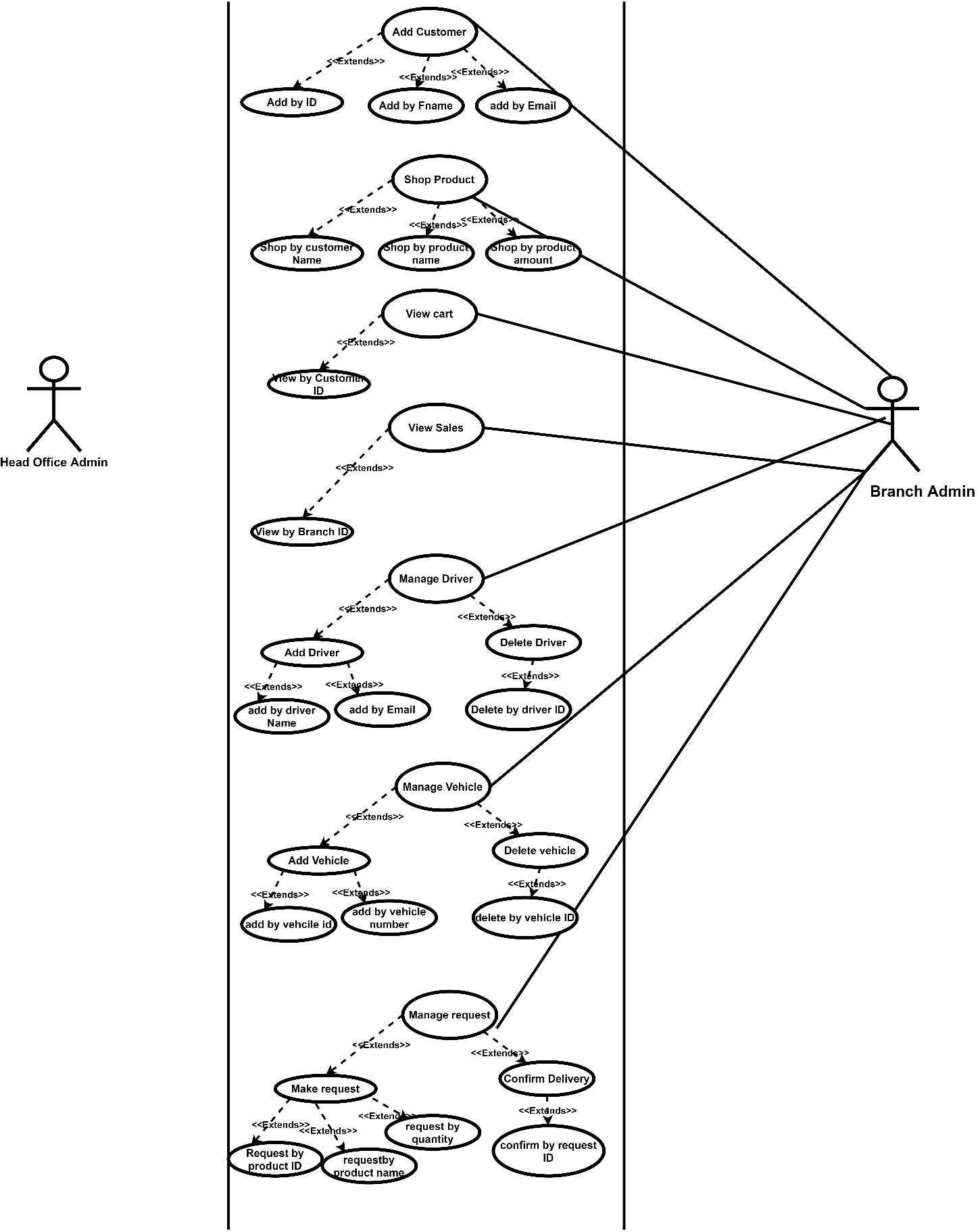
## Use-Case Diagram

**NOTE: SINCE THE DIAGRAM IS TOO LARGE TO PUT IN ONE PAGE, I HAVE SPLIT THE DIAGRAM INTO TWO PARTS**

### Use-Case Part 1

## 

### Use-Case Part(2)



## Explanation

### Actors

1. There are basically 2 actors in the system. The Head Office user and he Branch User

### Use-cases

2. Initially Both the actors Login to the system using the login Use-case. The Login as Headoffice admin or Branch admin comes as a part of the login use case. This relationship is denoted by the dashed connected line with the arrow.

3. The <<Includes>> Relationship between the connections of the login use cases takes it to another use case Authorization. This defines that one-use case is needed by the other to perform its tasks.

4. First the Users use the Update information use case which has the <<Extends>>(an alternative option under a certain use case) Relationship with the update by userid, update by fname and update by lname use cases. In order to register the user.

5.Similarly, all the other use cases will function just as the update use case where via the <<Extends>> keyword which defines the alternatives.

## Sequence Diagram

The sequence diagrams represent the Head office admins and the Branch Admins parts of the system.

### Lifelines

Denotes the life of an object during a sequence placed vertically across the top of the diagram

### Activation Bars

Simply, the activation bars will represent the active period of the lifeline usually it starts when it receives a message and stops after sending the response

### Actors

These are the people who are external to the system. Here the Head office admins and the Branch Admins are the actors

### Messages

Conveys information from one object to another. When an object sends a message it is shown as an arrow between their lifelines. Here the fully lined represent the messages sent by the user and the dotted lines represent the messages returned from the database.

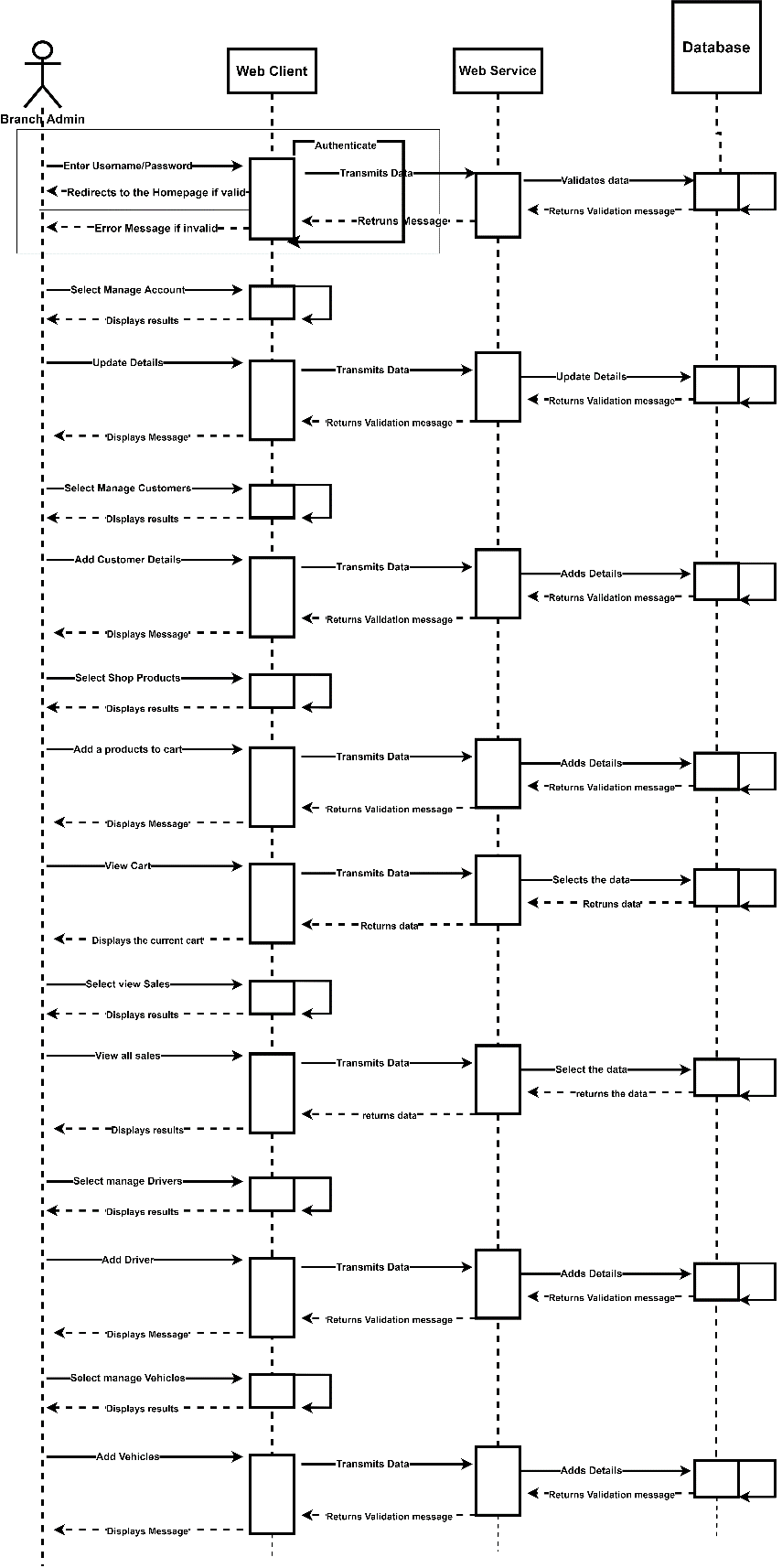
### Head-Office Admin Sequence

## 

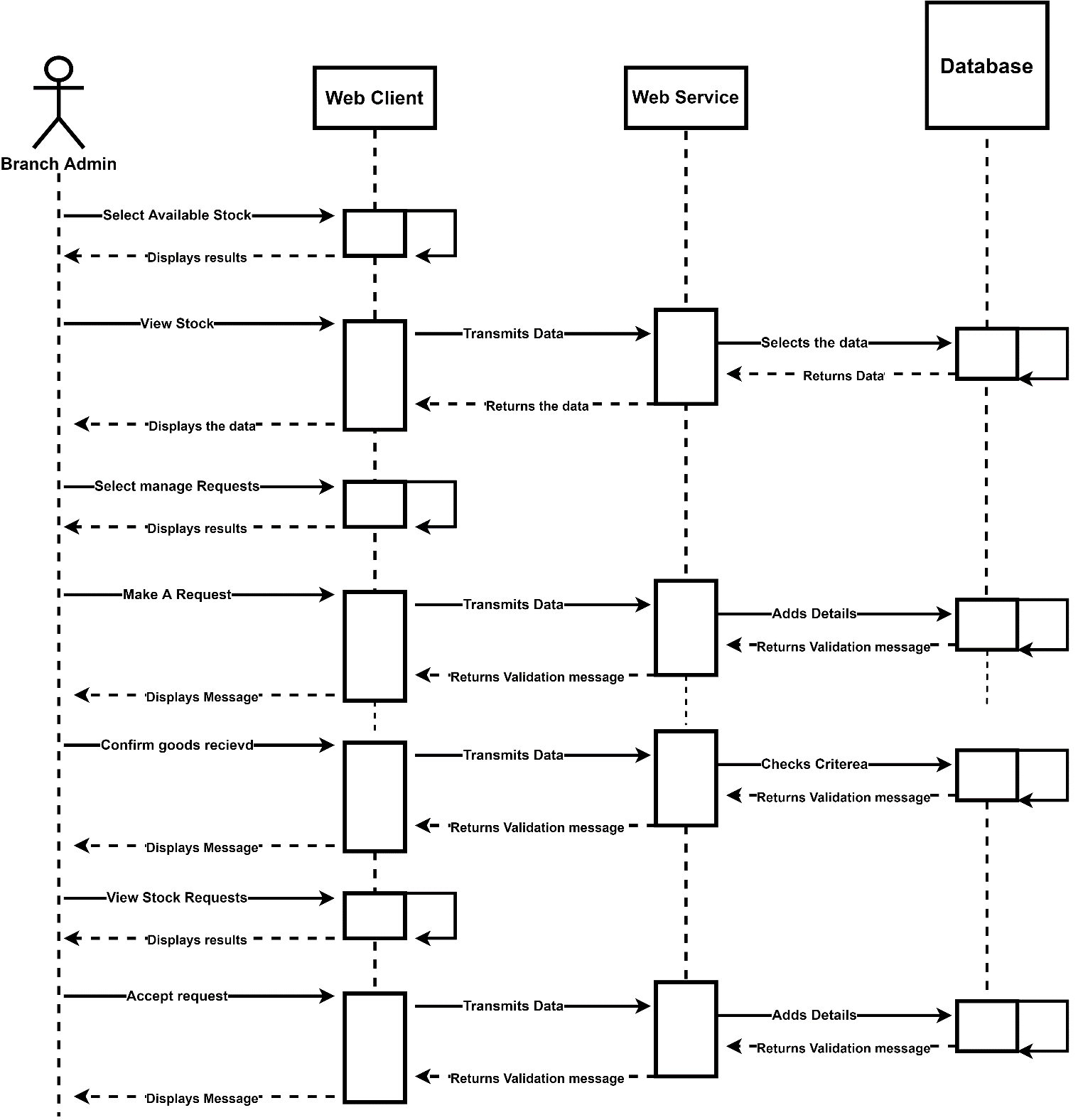
## Branch Admin Sequence

**NOTE: SINCE THE DIAGRAM IS TOO LARGE TO PUT IN ONE PAGE, I HAVE SPLIT THE DIAGRAM INTO TWO PARTS**

### Branch Admin Sequence -Part 1



### Branch Admin Sequence -Part 2



## ER Diagram

Figure 1 Er Diagram

* This is the basic ER diagram, therefore the database includes the following entities and its attributes
* Additionally, the only difference in the database and ER is that there is no separate head office table rather, Branch 1 is considered as the headoffice.

# TASK C

## What are Design Patterns?

According to ( (Rahman, 2019) , Design patterns are simply the solutions for the problems that arises for us as developers in the designing level of a software or any application.

## Identifying different design pattern and their advantages

### Singleton Design Pattern

The definition of singleton is quite simple where there is one class that allows only one object to be created and gives access to all other classes via that single object or the instance (Poyiyas, 2018).

#### Advantages of Singleton

#### It supports instance control where it prevents objects from creating copies of their own rather, using a same object they access the class.

#### It is very usefully when making use of multi thread pool

#### As the class controls the instantiation, the class has the chance of changing the instantiation at any time.

### Factory Design Pattern

### According to (JavaTPoint, 2018), Factory pattern simply defines an interface creating and object probably a method and the other classes, the subclasses could make a choice of what class to instantiate or of what class should they create and object.

#### Advantages of Factory

### It could reduce the runtime and memory allocation since, only the resultant interface or the abstract class interferes with the object created, it will not run the rest of the other classes during runtime if they are not instantiated.

### When u have factory, you do not need to write the same code in several locations. Rather, simply by calling the instance of the corresponding interface class it would perform its task. This could improvise the efficiency

### Abstract Factory Design Pattern

In abstract factory you can define objects of interfaces or abstract classes but, you will not define its corresponding

#### Advantages of Abstract Factory

# It will facilitate exchanging objects.

* It will promote consistency among families, for instance it is recommended to use objects related to one family at a time during runtime and abstract factory supports this requirement.

## Applying the Suitable Design Pattern

Note: I have only used the singleton design pattern

#### Returning a singleton Object

Figure 2: singleton object

* This getConnection method is embedded in the Dbconnection class where it returns a single connection object at each time the method is called. Additionally this method compromises the basic coding that is needed to establish a connection between the database and the application

#### Receiving the singleton Object



Figure 3 Receiving the singleton object

* These methods are related to perform various CRUD operations via the database. And it can be noticed that via a single connection object created , this method receives the database connection from the Dbconnection class. This is an example of singleton

## Evaluate the impact of Design patterns

## Applying design patterns into a development allows or makes it reusable in several projects but not just stick into one.

* Design pattern does not provide solutions but, it allows experts to use their knowledge on design patterns to apply the most suitable design pattern for a given software problem. Thus, this action will make the experts knowledge of the design patterns widely available and makes freshers, new developers to understand of how the system development has been done and how the experts have chosen the most appropriate pattern for the given problem. And it will efficiently improve the programming skills of the freshers and will help them to write some quality ,standardize and efficient set of codes.
* Usage of design pattern allows developers to make more sophisticated approaches towards the development rather than using ordinary data structures like binary tress ,array lists etc..
* As we all know structured programming is no more used among industrial developments rather object oriented programming is the one that has risen. And all these design pattern works upon the OOP principles which bring out a great advantage.

# TASK D

# System Explanation

## Distributed (3 tier) Application

Figure 4 Webservices

* With the above screenshot, it is clearly seen that this is a distributed system that consists of 2 applications that interconnects one another to perform necessary tasks.
* The webservices includes the classes related to the scenario and it includes classes that has methods that are used to perform actions related to the database.
* The client application simply consists of User interfaces and classes that were automatically developed via the WSDL file that was imported from the webservice.

## Webservice Methods



Figure 5 webservice method

* These are few of the methods available in the Webservice.
* Each method will be responsible for a particular action.
* These methods are the one’s that connects the client side and the web service for instance the web client could access the webservice method via the method’s operation name and perform any action that the method allows residing from the client side.

## Classes in the Web Service

Figure 6 webservice classes

* This is one of the classes that are available in the webservice, Each class consists of an empty constructor and as well as an overloaded constructor (these will be useful for testing the application).
* Each class will also include getters and setters for each variable available.

### Dbutil class



Figure 7 dbutil

* Dbutil class is quite different compared to other classes.
* It consists of various methods that are used to perform CRUD operations related to the database.
* Some methods accept parameters, some returns object, some returns Booleans etc..
* Upon the requirement, the webservice methods will access the Dbutil classes methods to fulfil the necessary requirement

## Webhelper Class

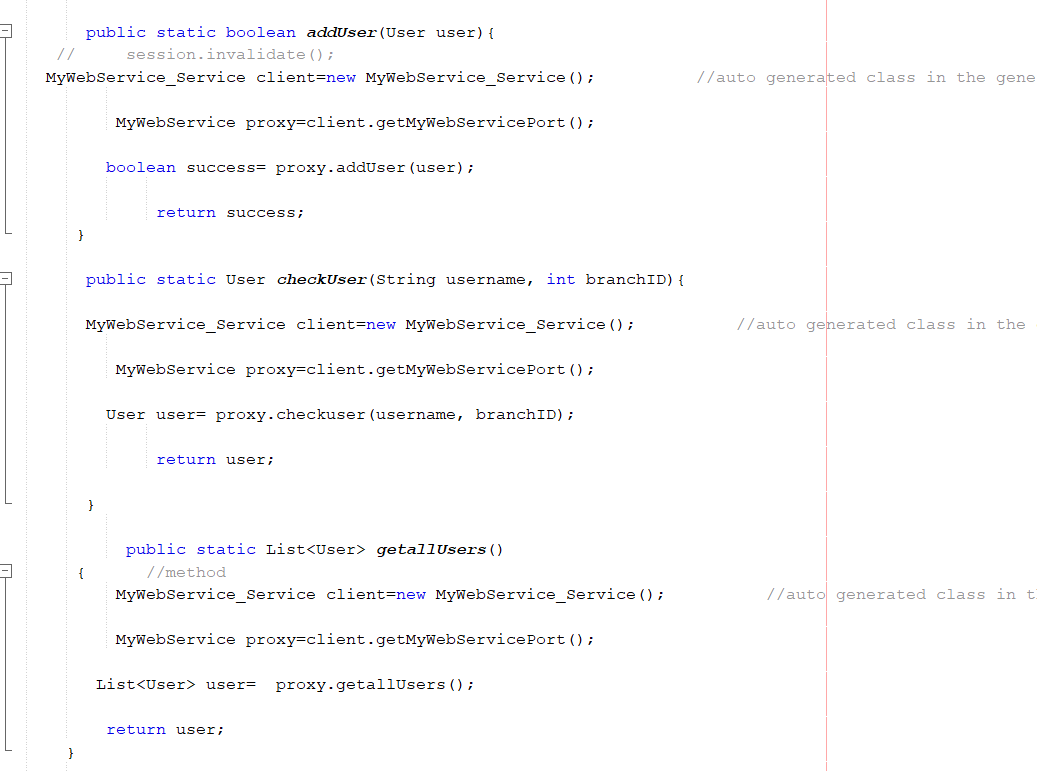


Figure 8 webhelper

* The Webhelper class resides on the client side of the application.
* These methods will be used by the client side to communicate with the webservice via the proxy.
* It is noticed that each method, via the proxy access another method and these methods are the ones that were created in the web service.

## Object Oriented Principles Used

### Encapsulation

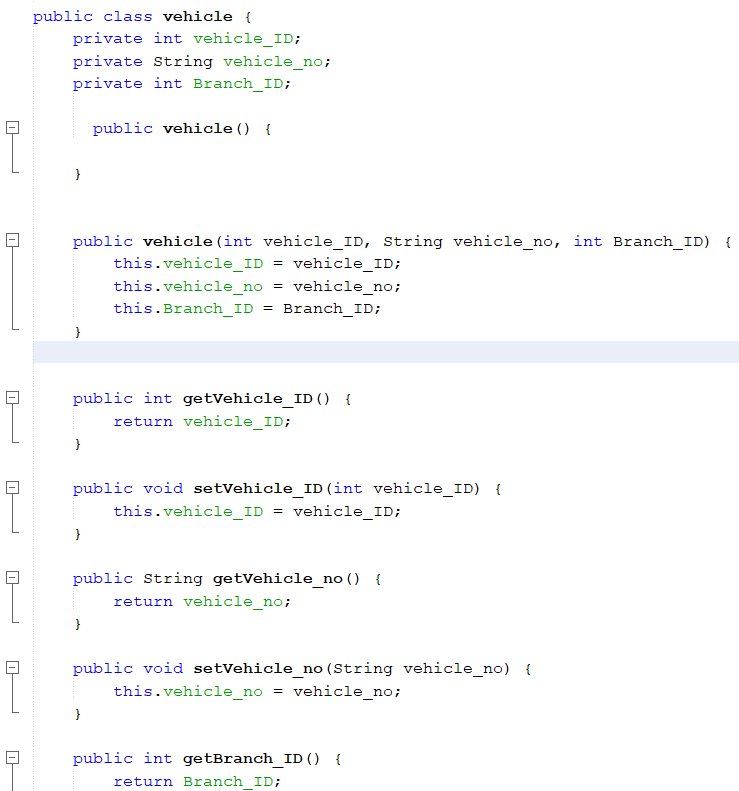


Figure 9 Encapsulation

* Each class has private variables and public getters and setters therefore it proves that encapsulation is used

### Inheritance



Figure 10 Inheritance

* The extends keyword shows that inheritance concept is used where the driver class inherits methods and variables from the users class.

### Polymorphism

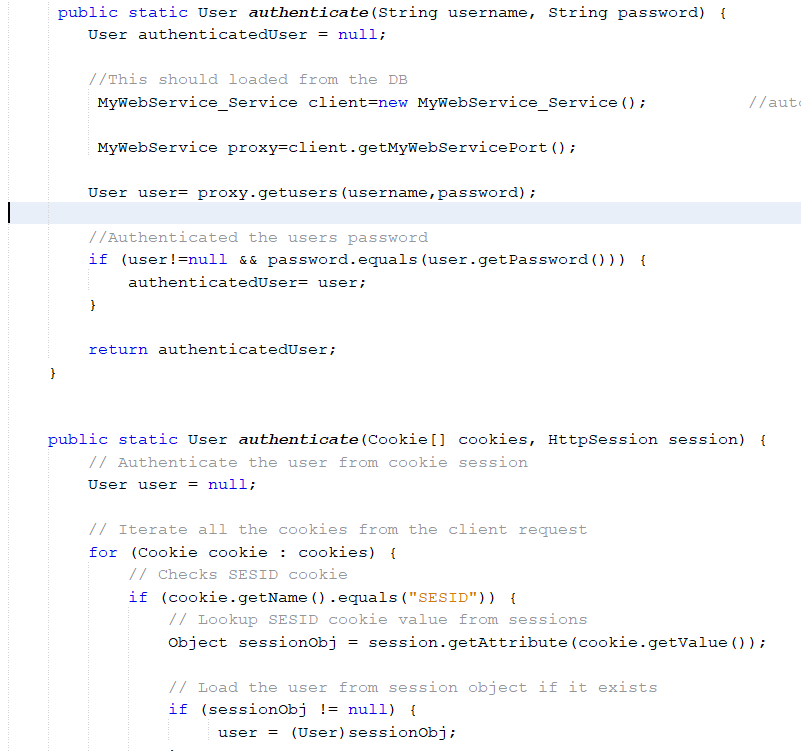


Figure 11 Polymorphism

* Polymorphism method overloading concept is used where the same authenticate method accepts different parameter.

## Sending Emails

* In order to send email via the system, the java.mail.api library is used.
* The mailing function will be used to notify a branch when the stock request is accepted and as well as to notify the customers once they shop for products

### Code

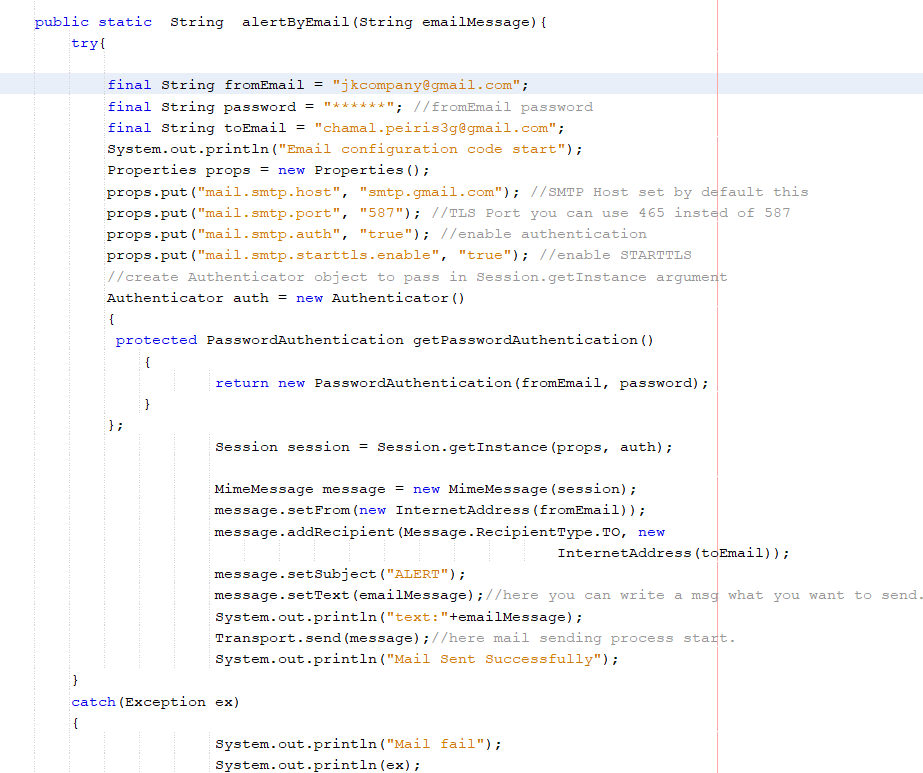


Figure 12 sending email

* This method will be simply used to send email. First the desired email message is accepted a parameters.
* For the demonstration purpose, I have used my personal email as the recipient address.
* Then, using properties data type some basic smtp server configurations are done
* Finally using the authenticator object the email is sent to the recipient.
* In the system, emails are sent to customers when they checkout

## User Login

### Interface

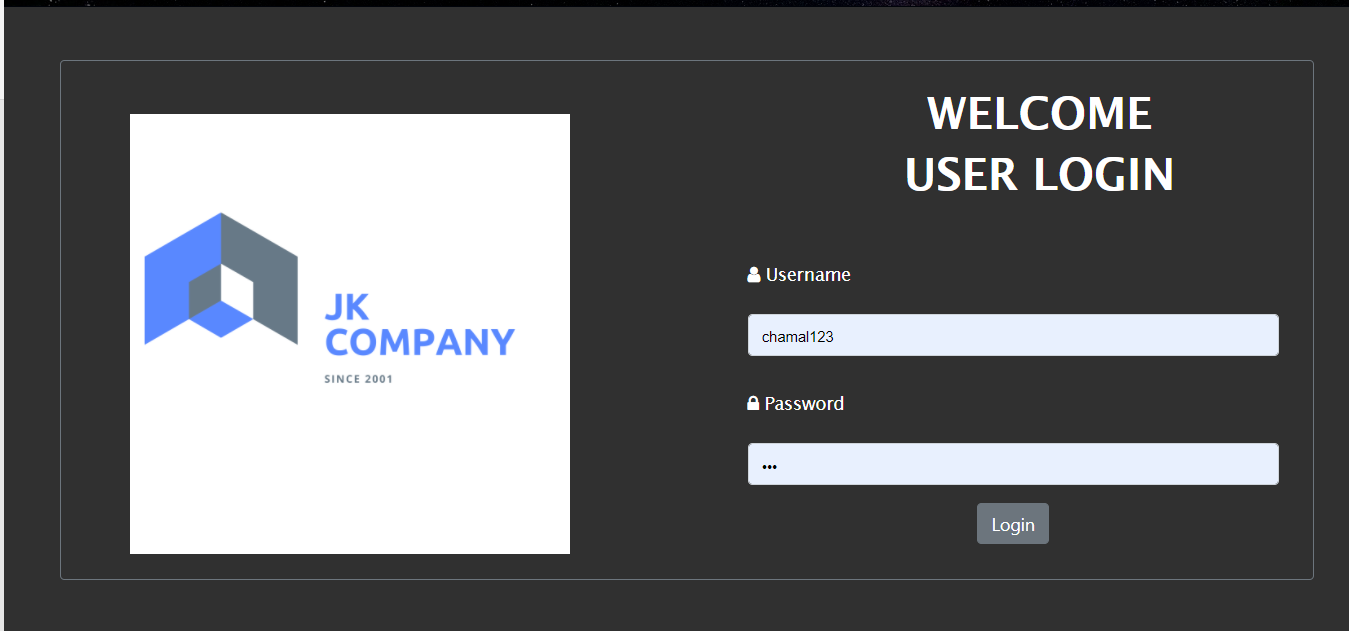


Figure 13Login UI

#### Code Explanation

#### Usage of sessions and cookies



Figure 14 usage of session

* Once the user clicks on the login button on the login form, it will redirect the user to the following page.
* Here, first the input username and password parameters will be sent to the getuser() method which is place in the Webhelper class and the method would return and object of the user.
* And if the object is not null (data exists) a cookie for the particular user will be created and at the same time the following user object will be stored into a session so that the session object could be used anywhere in the system.
* And over the branch id, a simple validation is done where of the user objects branch id=1, it will redirect the user to the head office page this is because the branch id 1 I considered to be the head office. If not, users will be directed to the corresponding branch page.

## Head Office Homepage



Figure 15 Headoffice Homepage

## Branch Home page

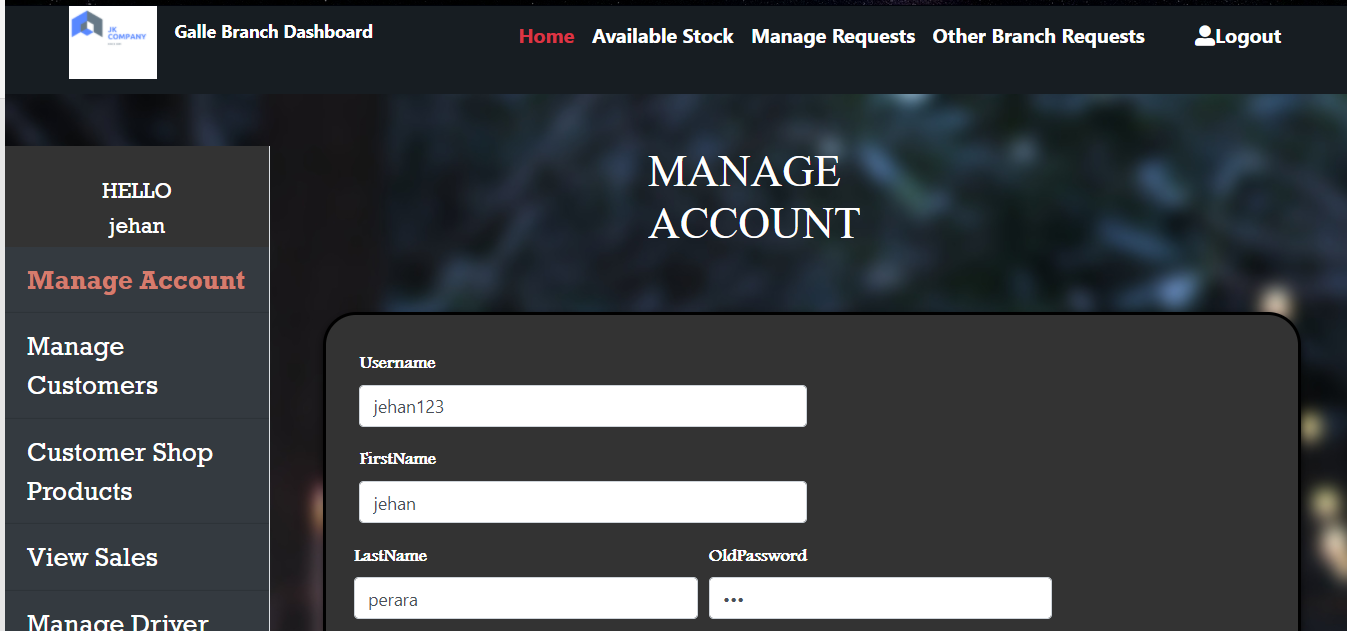
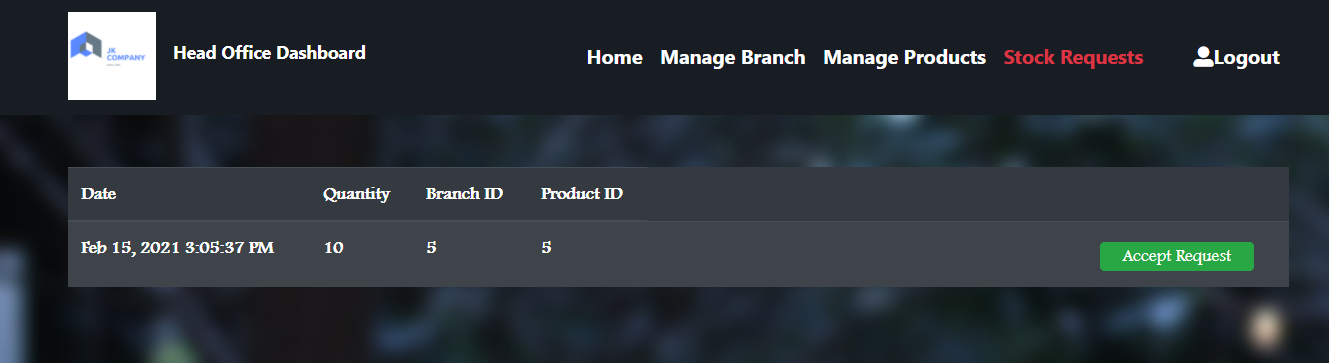


Figure 16 Branch Homepage

### Head office Manage Requests

### Interface



* Via this interface , the head office can view an accept stock requests from other branches

### Code

* Once, the admin accepts a particular stock request it will redirect to this page where the necessary parameters are retrieved and stored.
* Finally the received parameters are passed to the updateStock() method to perform its task.it is noticed that the branch location is passed as parameters as well. This is to display the location of the branch which has accepted the request

### Database when a request is accepted



## Branch Manage requests

### Make a Request

#### Interface



Figure 17 Branch make request

#### Code

* Once the user request the desired stock with its amount, it will redirect to this page.
* After storing the necessary data into variables. It will check whether a stock request with the relevant branch and product id exists if it does, it will notify the user if not, the new stock request will be added.

### Confirm Delivery

#### Interface

Figure 18 confirm delivery

* Once the user confirms the requested stocks are received. It will redirect the user to the confirmstocksrecievd.jsp page where necessary validations are done and finally the stock amount from the sending branch will be deducted and the stock receiving branches stock amount will be increments.
* This is a very complex code logic and the code is quite large to be put on the document. Therefore I request the accessor to check the confirmstocksrecievd.jsp to check my code logic of implementation.

## Branch Accept request

#### Interface

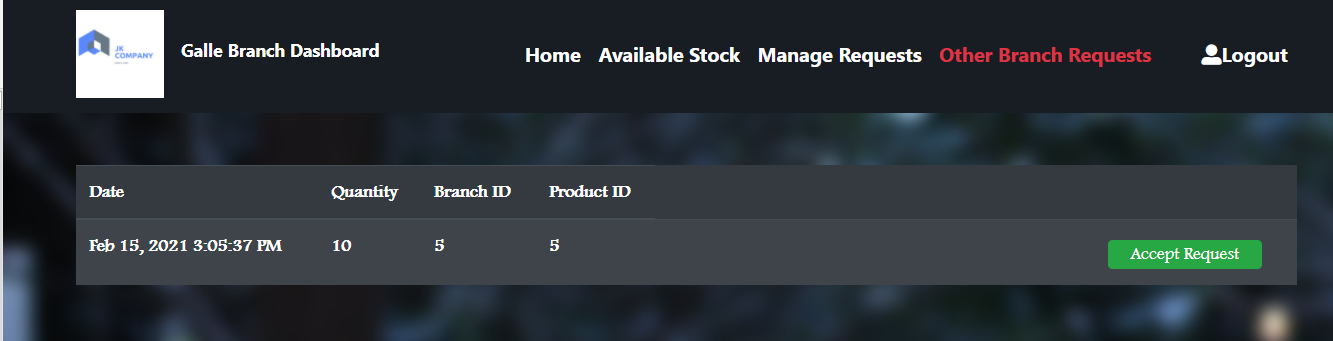


Figure 19 Accept request

#### Code

Figure 20 accept request

* To accept the request, the branch should have the particular products, so first, it will check whether the requested product is available in that branch if not and error message will be generated.
* If they try to accept their own request, they will be notified as well.
* If not the stock accepted details will be stored in the stock request table in the database

# TASK E

## How test driven development is used in the system

* Test driven development is simply used to develop test cases to validate all the codes in the system. The codes in the sense means the various methods that are responsible for various system functionalities.
* The system developed will be tested via the Unit testing approach.
* The Dbutil class is the one that will be tested. The main reason for this is, it is the class that consists with the methods that directly performs the CRUD operations with the database.
* Therefore using unit testing each methods will be tested individually.
* For instance imagine a method that returns an object. Via the unit testing, a sample set of test data will be tested with the actual data that is been retrieved from the database and if the dataset matches each other a success message appears. If not an error message will be displayed showing from where in the code has gone wrong. Here, the developer can correct the error and rerun the test and check whether it passes.
* Similarly all the other methods that returns Booleans, strings will be tested as well.
* To be more specific ,43 methods are tested individually

## Checking the getUser() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the updateUser() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the addUser() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the checkUser() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the getUsers() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the deleteUser() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the addBranch() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the getBranch() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the getOneBranch() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the deleteBranch() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the addProduct() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the addProduct() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the addBranchProducts() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the getALLHeadofficeProducts() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the Getsingleheadofficeproduct() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the Deleteheadofficeproduct() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the GetbranchProducts() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the ValidatebranchProducts() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the test AddStockRequest() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the test Getstockrequest() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the test GetStockSendingBranchID()method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the test GetStockSendingbranchStockamount() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the test GetStockrecievingbranchAmount() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the GetbranchRequest() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the UpdateStcockRequets() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the Getpendingrequest() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the Updatestocksendingamount() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the HeadofficestockAmount() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the testUpdatebranchnewstock() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the AddCustomer() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the Getcustomer() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the AddtoCart() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the GetCart() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the GetcustomerCart()method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the UpdateCart() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the GetBranchsales() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the AddDriver() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the GetAllDrivers() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the DeleteDriver() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the Addehicle() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the GetAllVehicles() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

## Checking the DeleteVehcile() method

|  |  |
| --- | --- |
| **Method Before** | **Results Before** |
|  |  |

|  |  |
| --- | --- |
| **Method After** | **Results After** |
|  |  |

# TASK F

## User Documentation

This technical documentation will guide the users upon all the aspect and the functionalities available in the system. And it is recommended to read this documentation before using the system.

## User Login

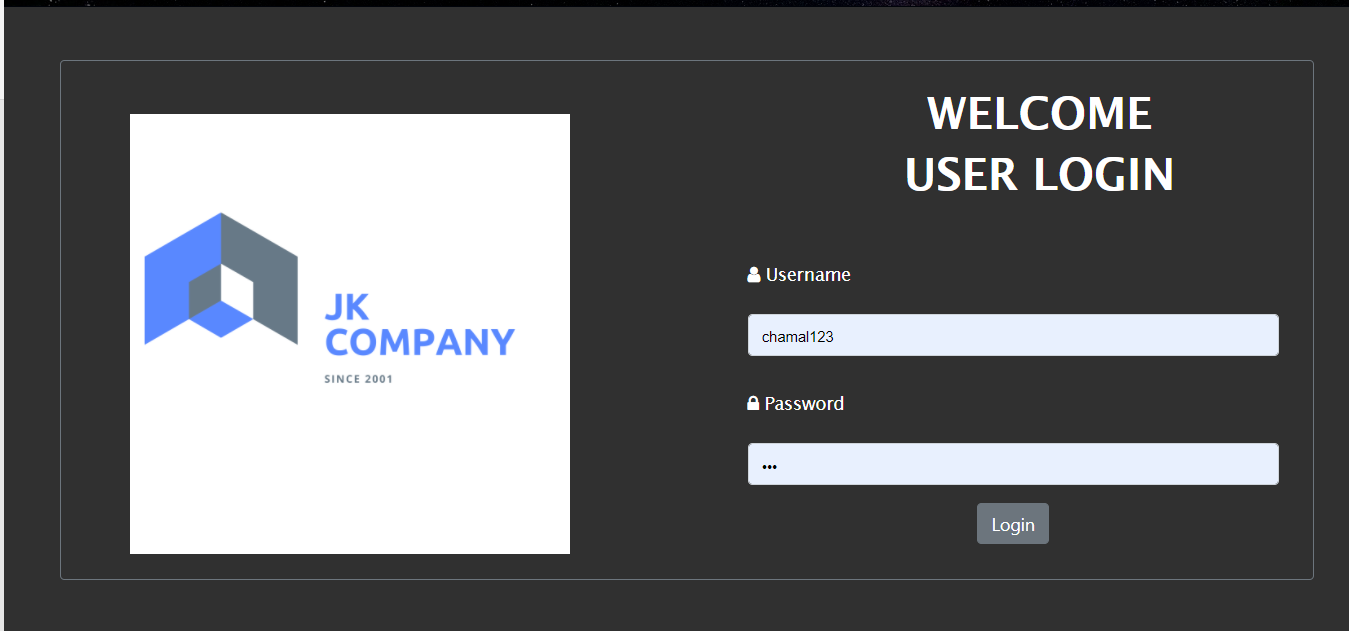


Figure 21-User Login

* Before Accessing the relevant interface, users are required to input their login credentials.
* Based on the credentials users will be granted access to either the Head-office Homepage or the Branch Home-page.

## Head-Office Homepage

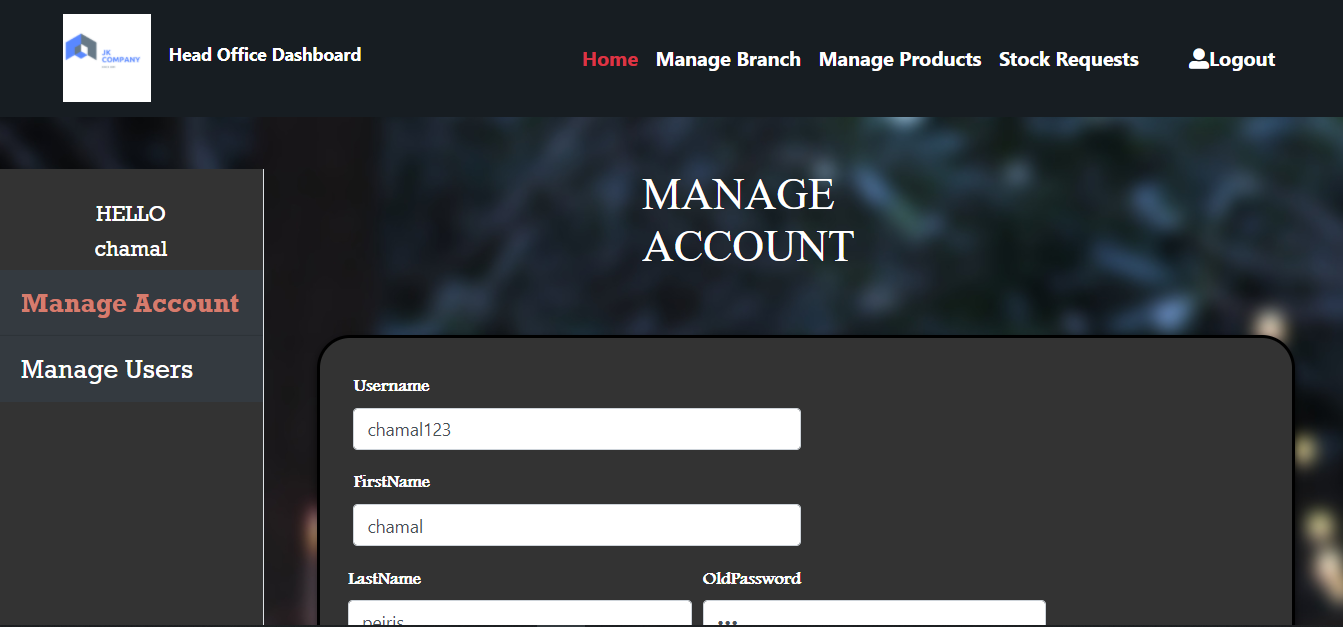


Figure 22-HeadOffice Homepage

* Once logged in as a head office user, they will be initially directed to this homepage.

## Manage Account

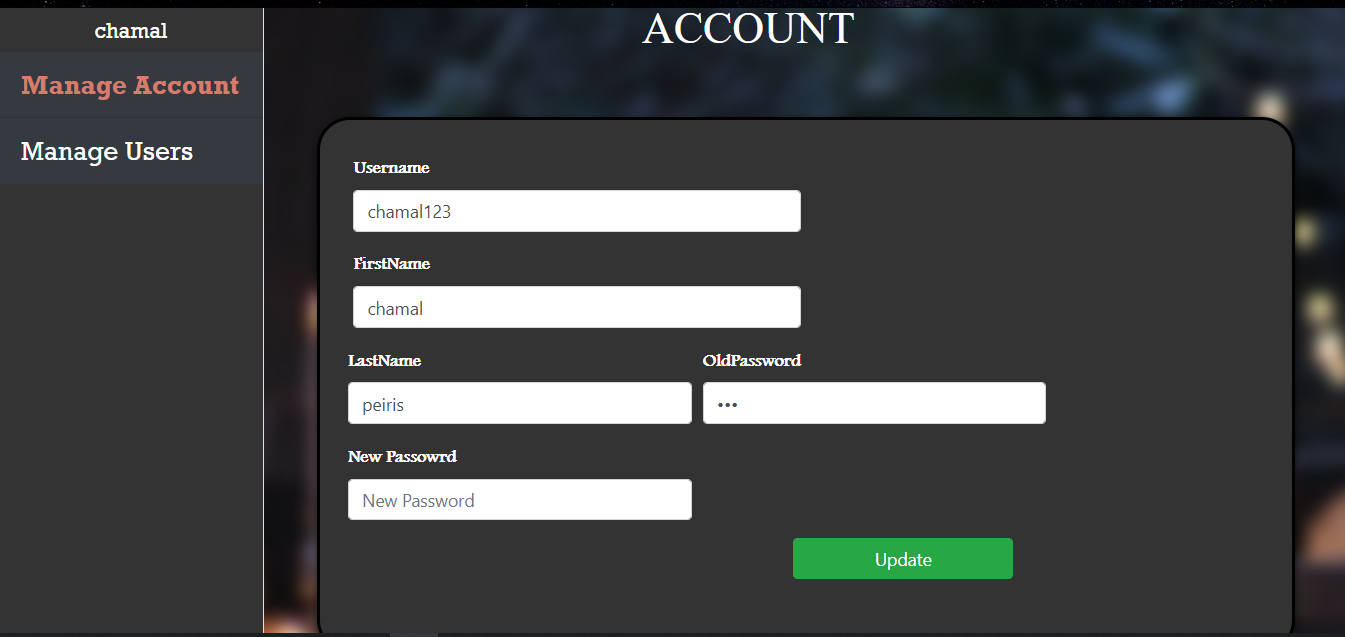


Figure 23 Manage Account

* Each User is given the chance of managing their credentials. Simply insert the new fields and click on the update Button.

## Manage Users

### Add User

Figure 24 Add User

* Via this interface, the User can Create users and allocate them into the required branch.
* First, Select a branch by Searching its location.
* Then, Add the necessary input fields and click on the Add User button. User will be added successfully

### View/Delete All Users

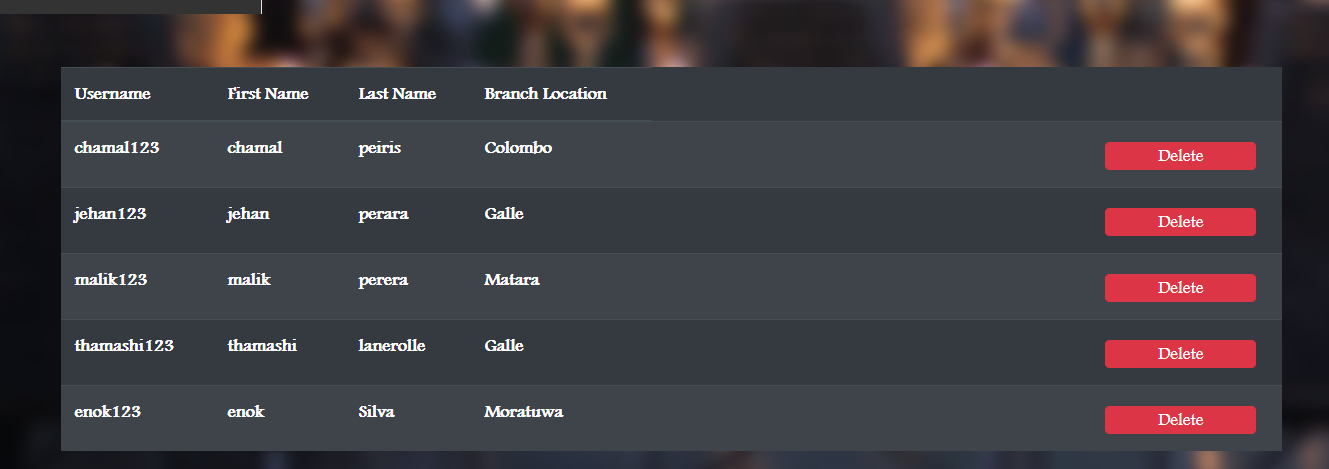


Figure 25-View Users

* Via this interface, the Admin can Either view all users allocated to each branch or delete them.

## Add Branch

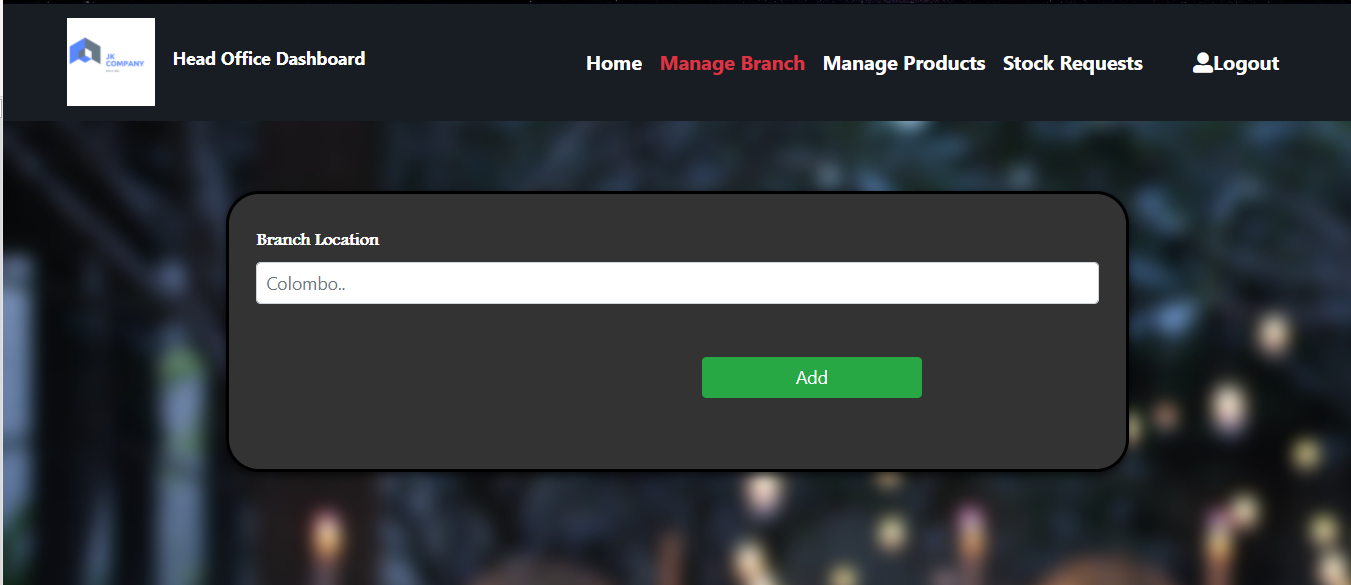


Figure 26 Add Branch

* This interface allows the admin to add a branch by just inserting the new branch location and clicking on the add button.

## View All/ Delete Branch

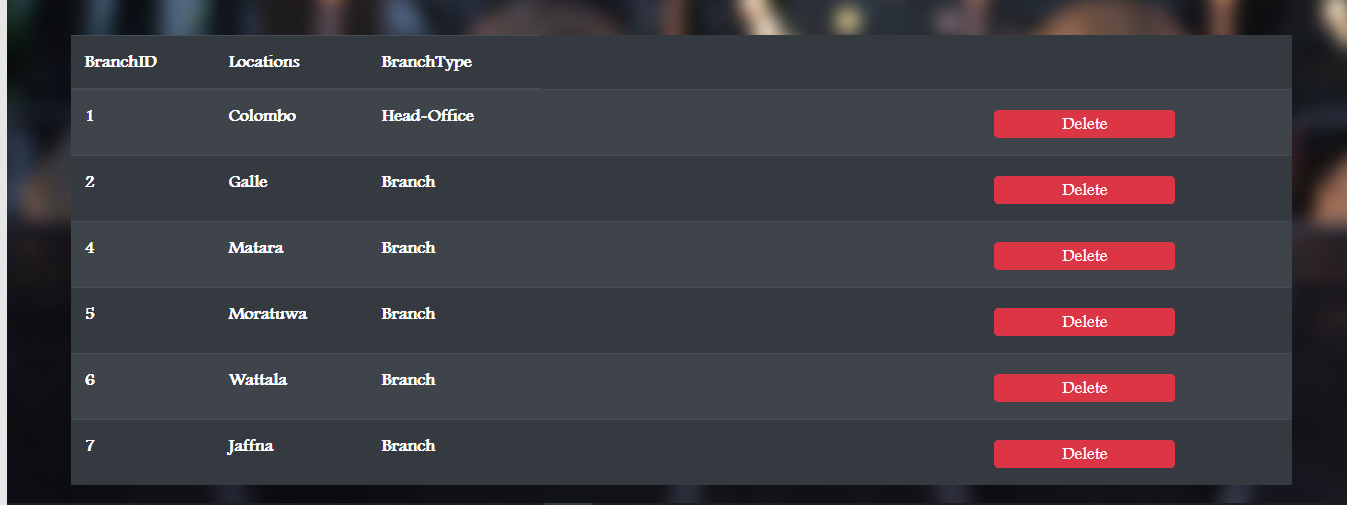


Figure 27 View/Delete Branch

* Via this interface, the Admin can Either view all Branches with locations or delete any of them.

## Manage Products

### Add Products

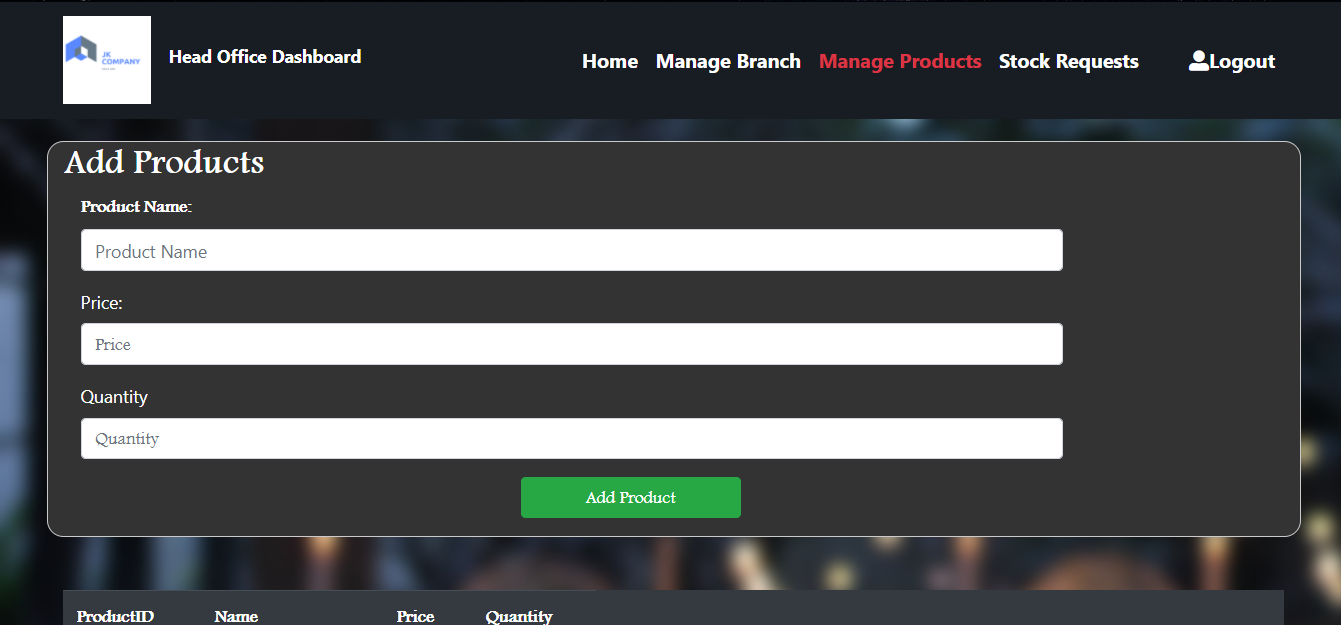


Figure 28 Add Products

* Simply, Enter the relevant input fields and click on the add products button, Thew new products will be successfully added.

### View All/delete Products

Figure 29 viewall/delete products

* Via this interface, the admin can either view all the products available or delete any of them.

## Stock Requests

Figure 30 Manage Request

* Via this interface the head office admin can view the stock requests from other branches and accept it if they desire.

## Branch Homepage

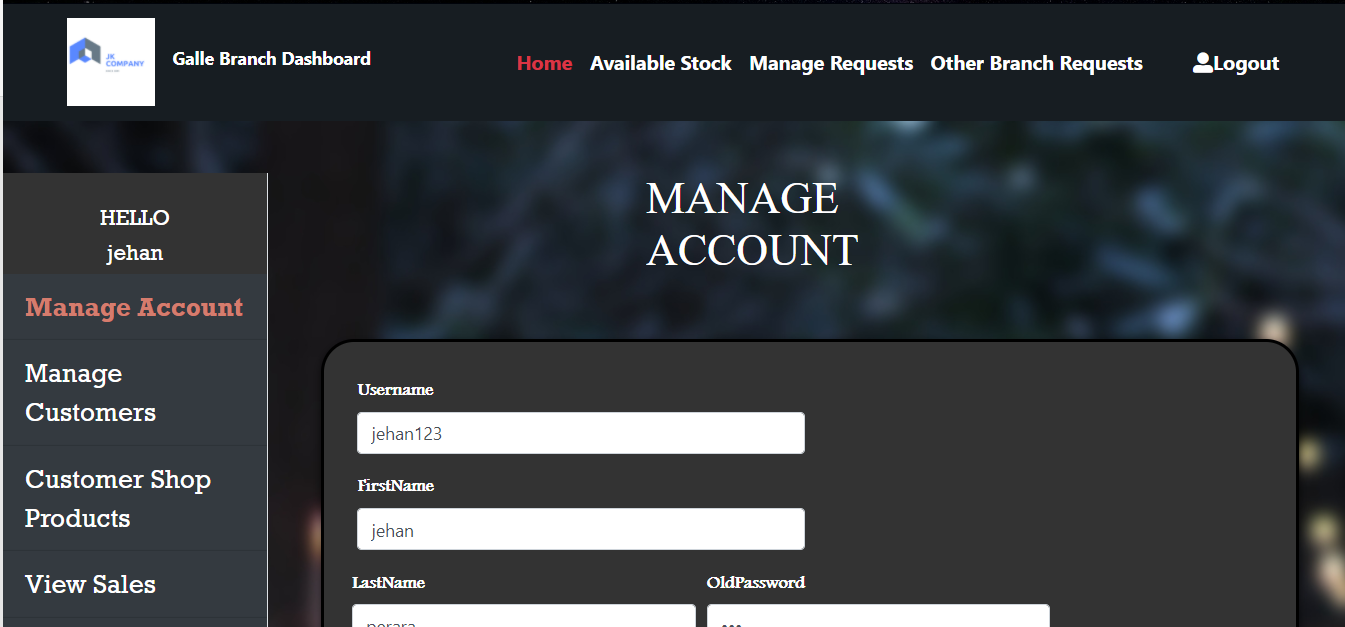


Figure 31 Branch Homepage

* Once logged in as a Branch user , they will be redirected to the relevant branch homepage

## Manage Account

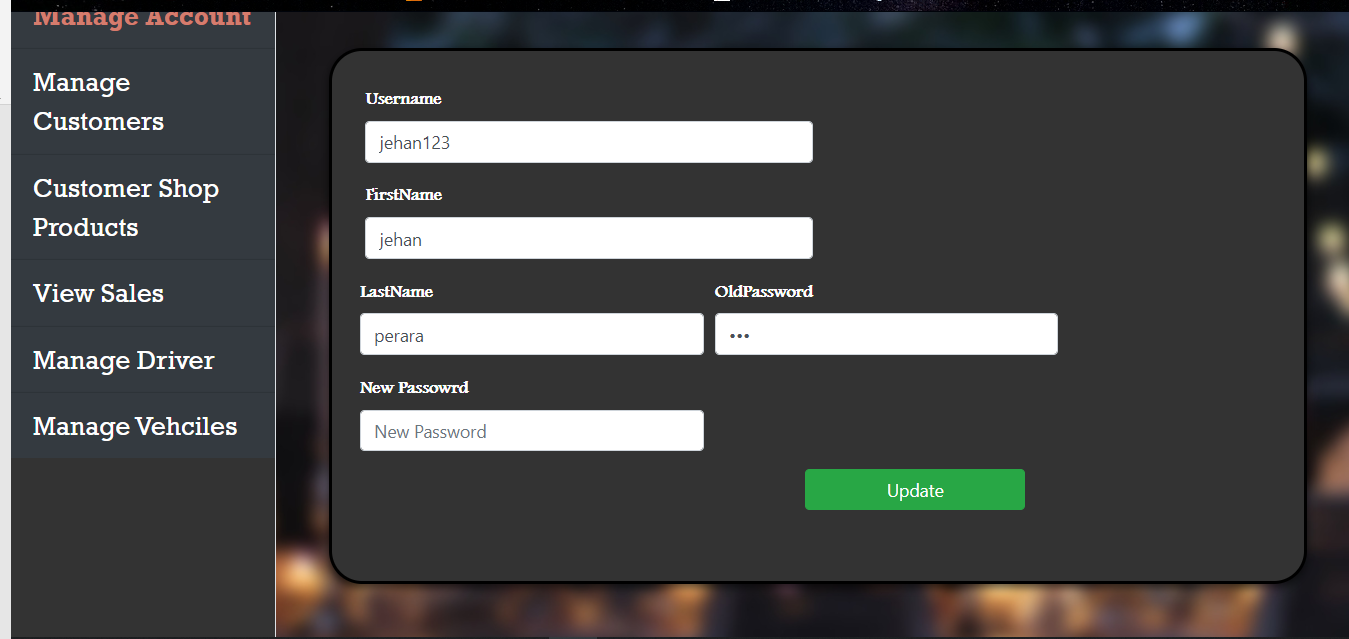


Figure 32 Manage Account

* This Interface allows the branch users to manage their account details. Simply, input the new fields and click on the update button.

## Manage Customers

Figure 33 Manage Customers

* Branch users, can add a new customer using this interface. Simply add the relevant customer information and click on the add button.

## Customer Shop Products

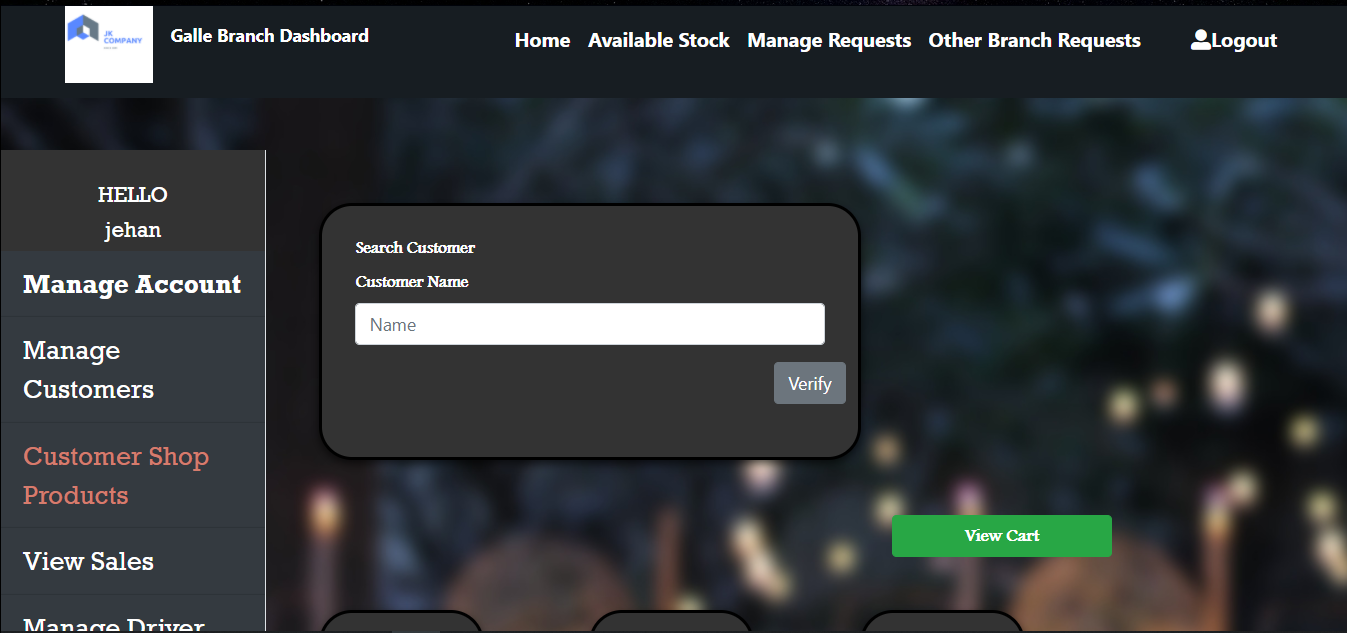


Figure 34 Shop Products

* This page allows the User to shop products.
* First, You have to select the customer that the goods are be shopped for. To do that simply, search the customers name.

### Searching an Available Customer

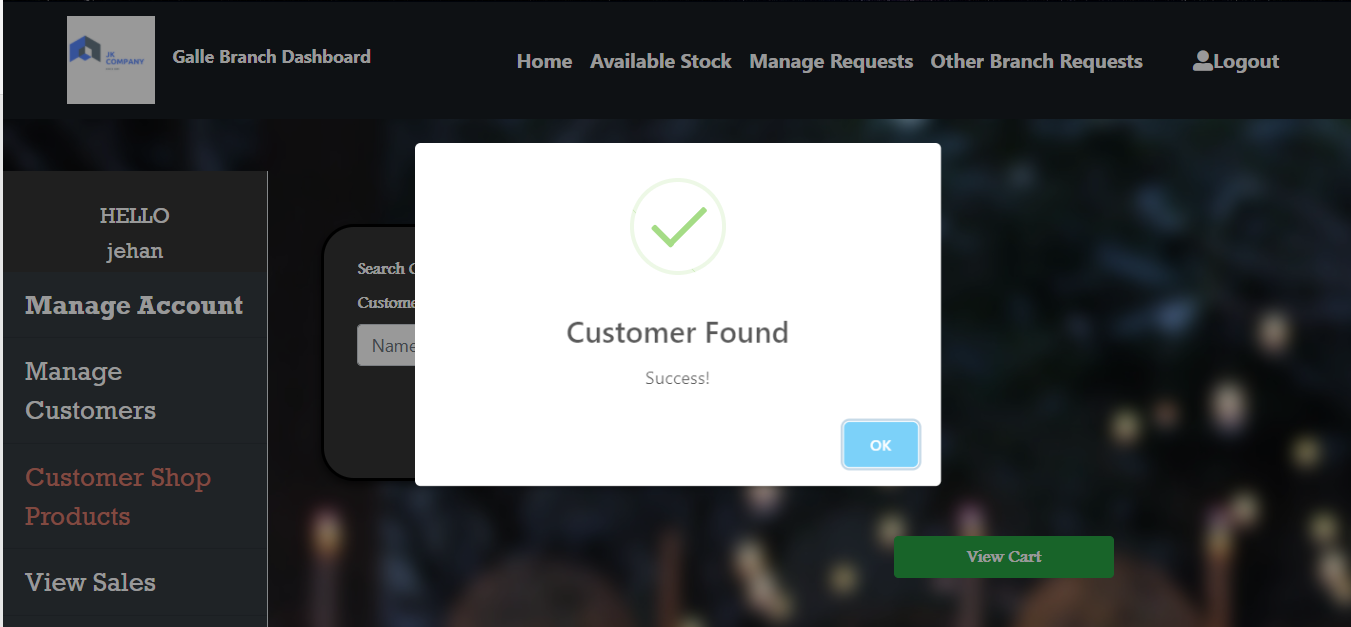


Figure 35 Successful message

* If the user search for an available customer, a success message will be displayed

### Searching an Invalid Customer

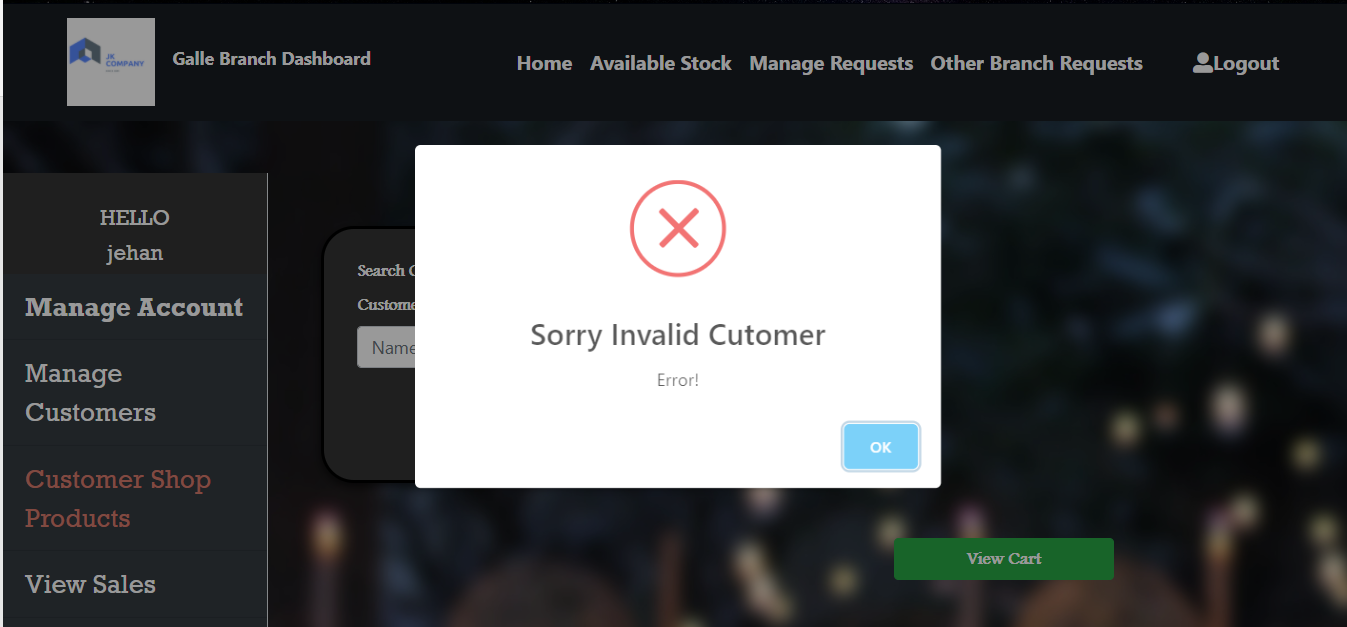


Figure 36 Invalid Message

* If the user search for an unavailable customer, an error message will be displayed

### Adding a product to cart

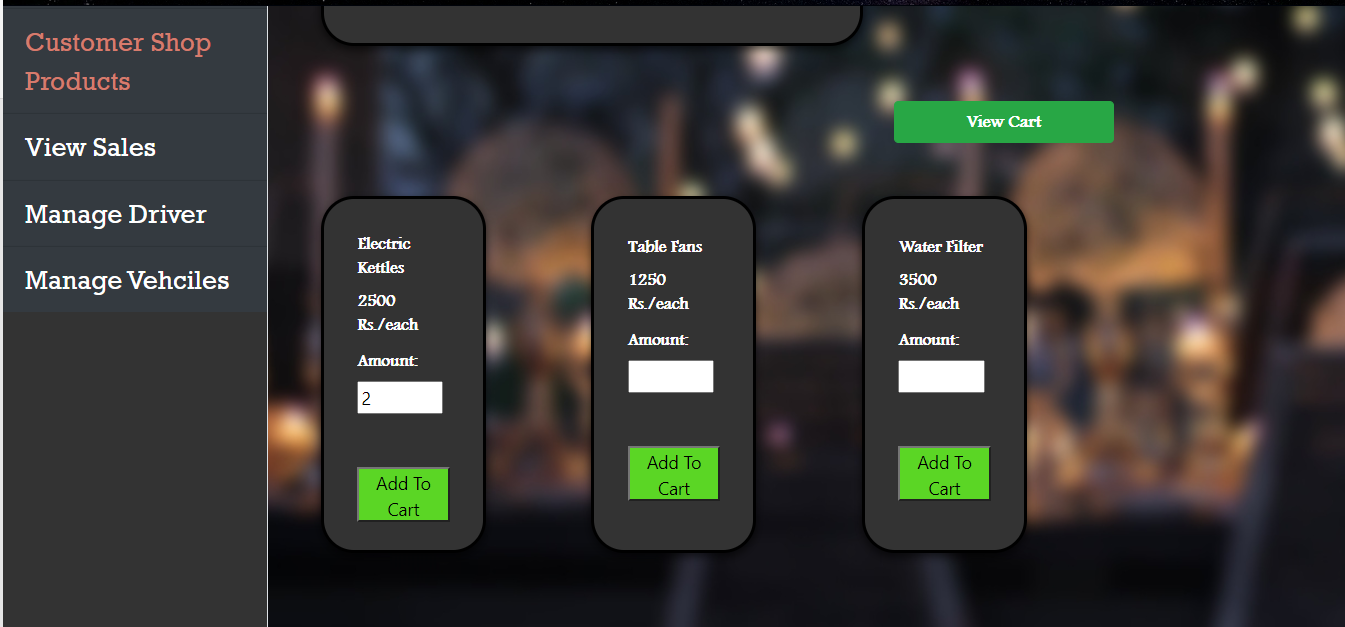


Figure 37 Add to cart

* To add a product to cart, select a desired product and input the required quantity and finally, click on the add to cart button

### View cart

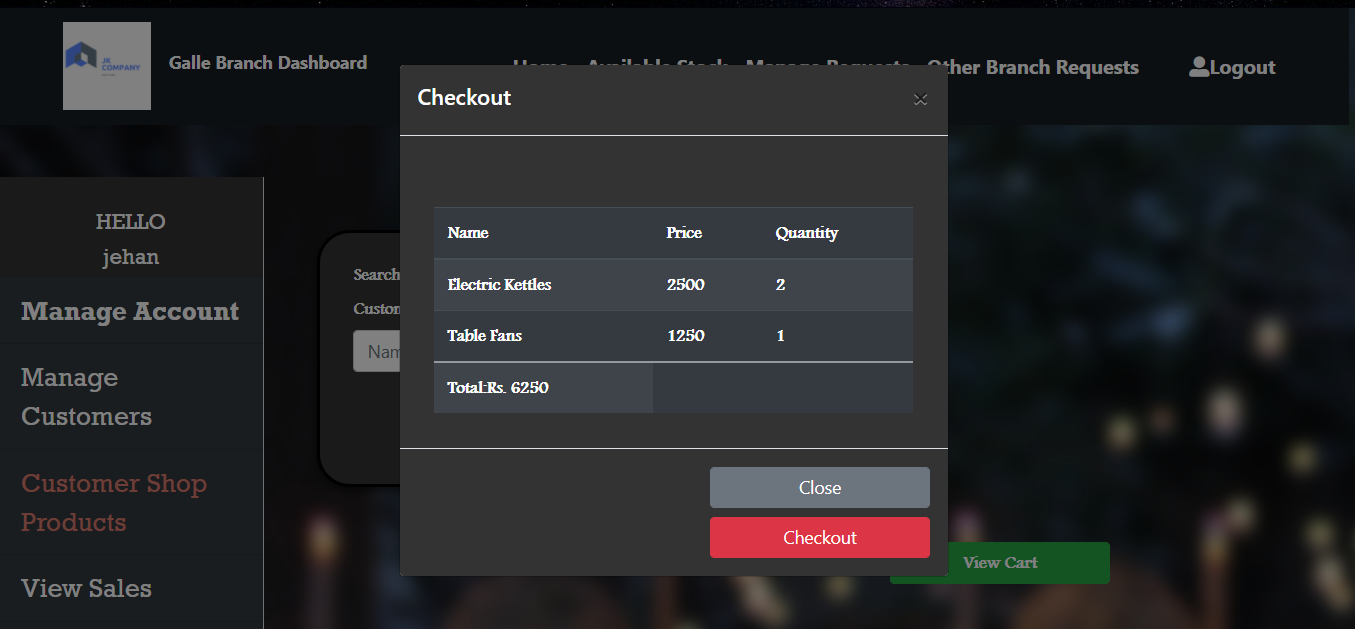


Figure 38 cart

* To view the cart, click on the view cart button, then the cart
* will appear.
* It displays the names of products added, its price, its quantity and finally it will display the total price of the cart.
* If the customer desired to checkout, simply click on the checkout button and the car will be checked out.

## View sales

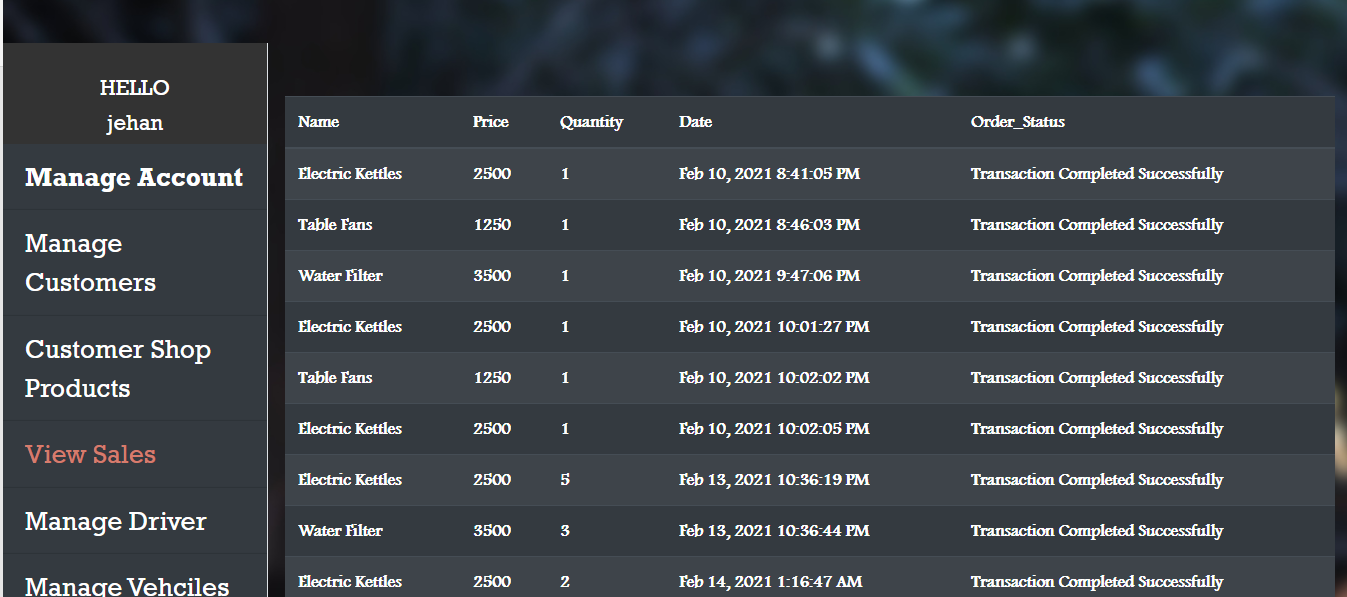


Figure 39 view sales

* This interface allows the branch admin to view the over-all completed sales within the given period of time

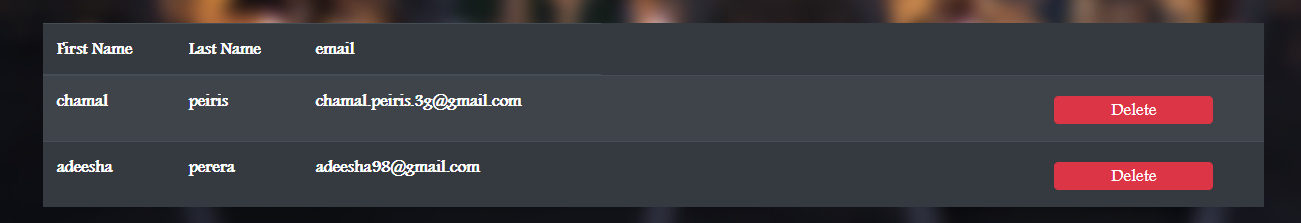
## Manage Drivers

### Add Driver

Figure 40 add driver

* To add a driver, simply insert the field and click on the add button.

### View All/ Delete driver

 Figure 41 view/delete driver

* With this interface, the admins can either view the driver assigned to their branch or delete any of them.

## Manage Vehicles

### Add Vehicles

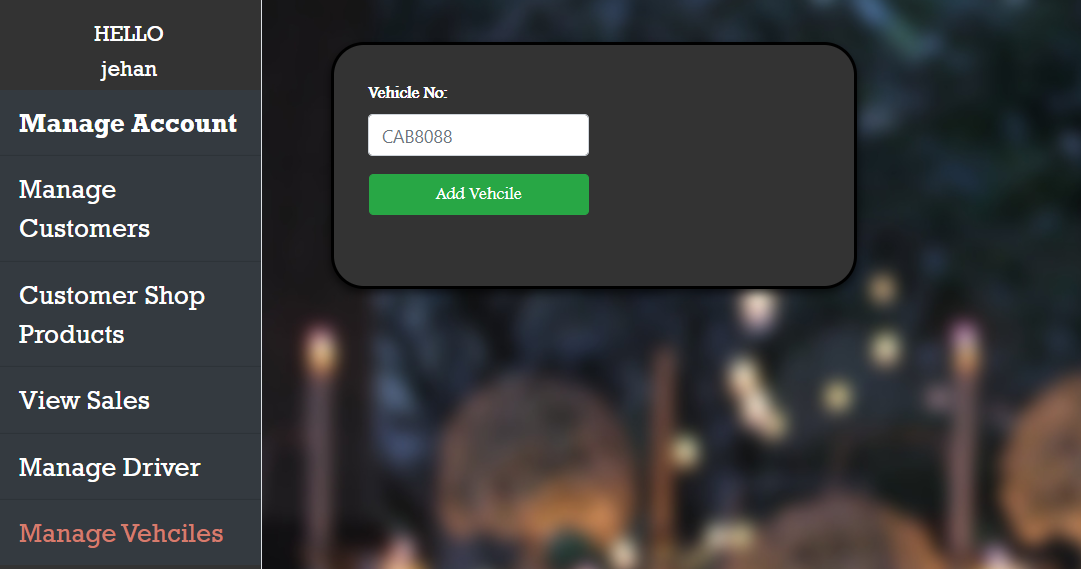


Figure 42 add vehicles

* Simply add the vehicle number and click on the add button, the new vehicle will be added.

### View all/ Delete vehicle

Figure 43 view/delete vehicle

* This interface allows the admin to either view all vehicles available or delete any of them

## Available stock

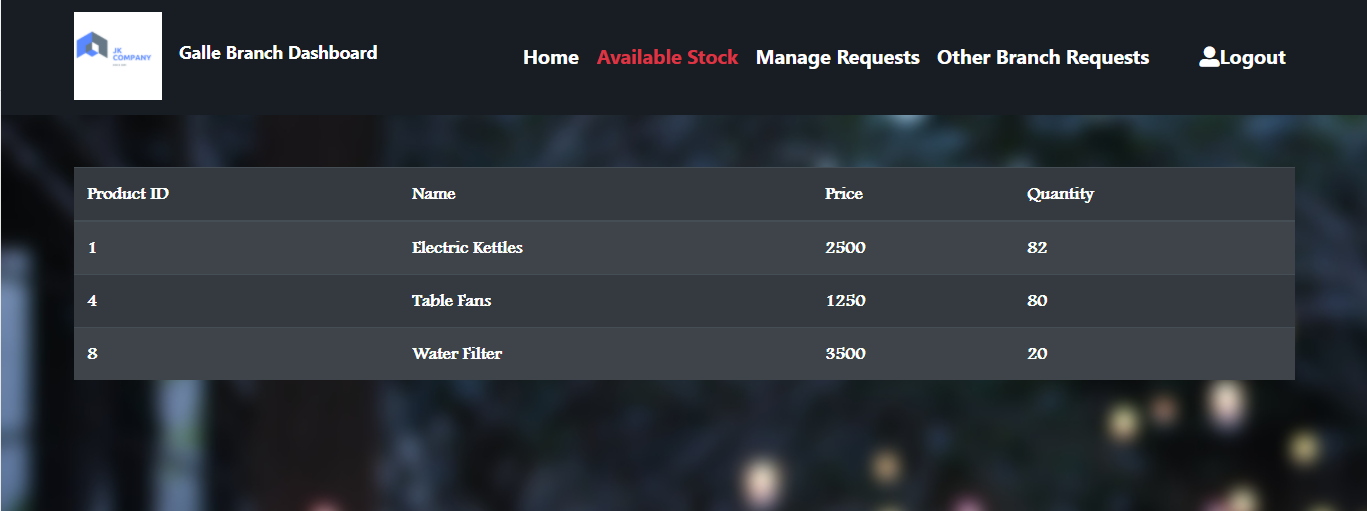


Figure 44 available stock

* This interface displays the available products in the particular branch

## Manage Requests

### Make a Request

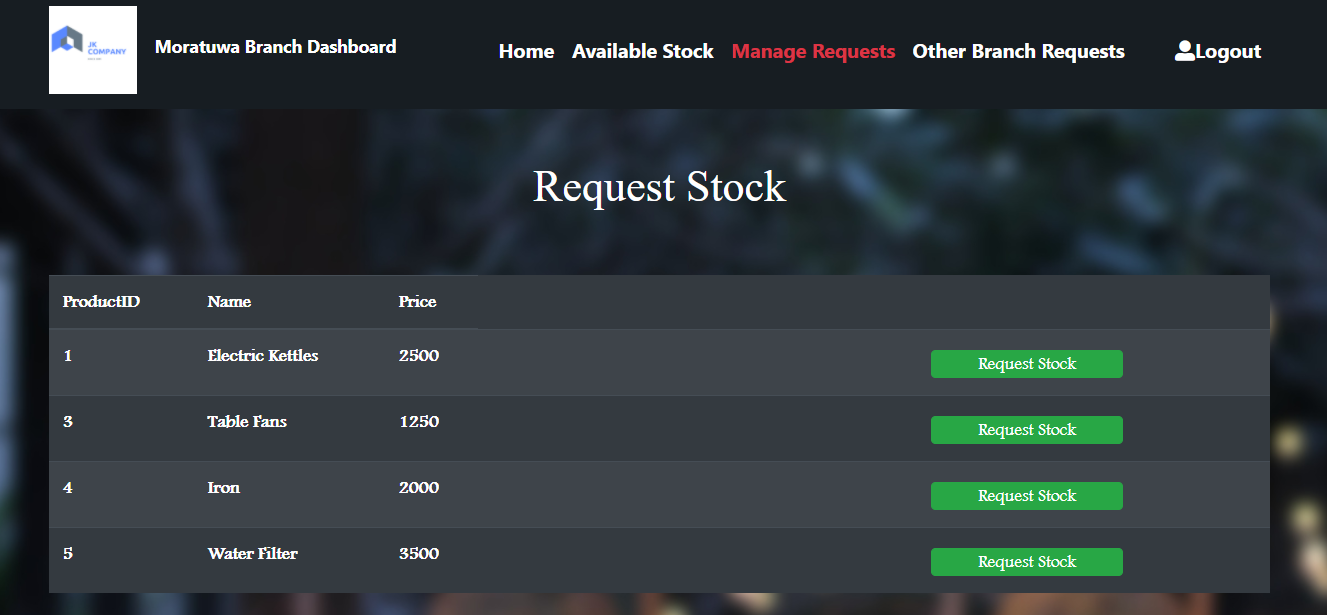


Figure 45 make request

* In order to male a stock request, simply click on the request stock button besides the product name.

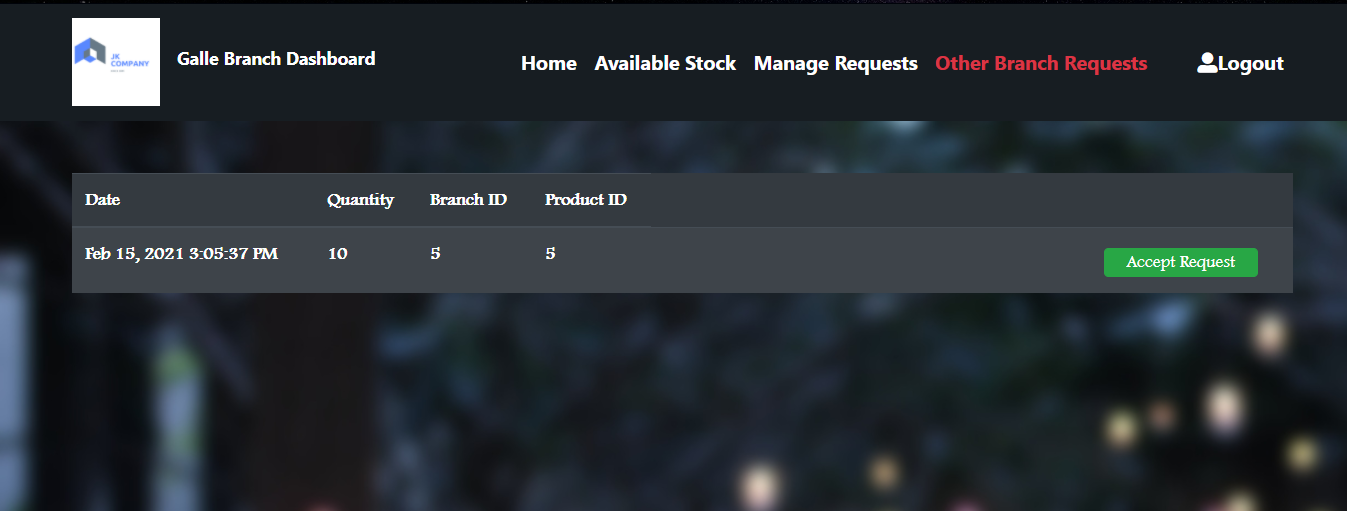
### Confirm delivery



Figure 46 confrim delivery

* This interface allows the users to view the products that they have requested from other branches.
* If the stock is accepted by any branch it will display the accepted branch name if not it will display that its not yet accepted.
* IF the stock is accepted by any branch and if the good are received, the branch admin can confirm that the goods are received by using the received goods button.
* Once, clicked the stock amount from the sending branch will be deducted and be added to the receiving branch stock amount( This is where the stock management feature works).

## Other branch requests

 Figure 47 other requests

* Each branch admin is capable of viewing other branches’ stock requests.
* But, to accept the request they should have the required product in their stock as well, if not, an error message will be displayed.
* And also the stock requesting branch cannot accept their own stock requests

### IF same branch tries to accept their own request

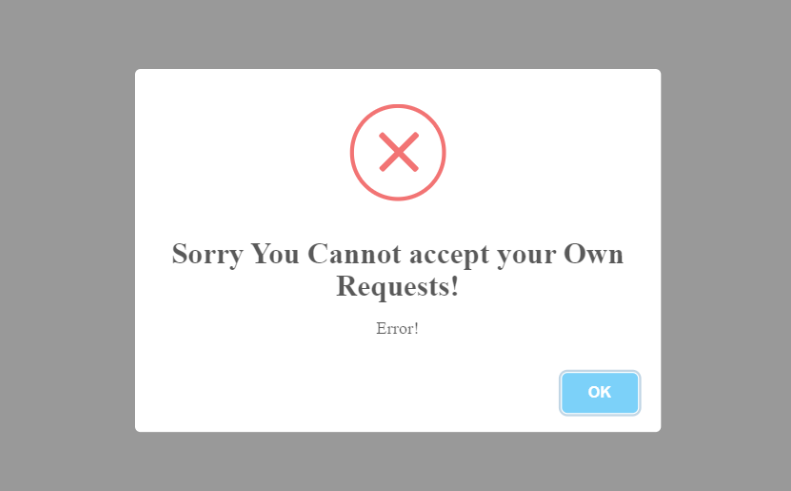


Figure 48 requests

### If the requested product not available

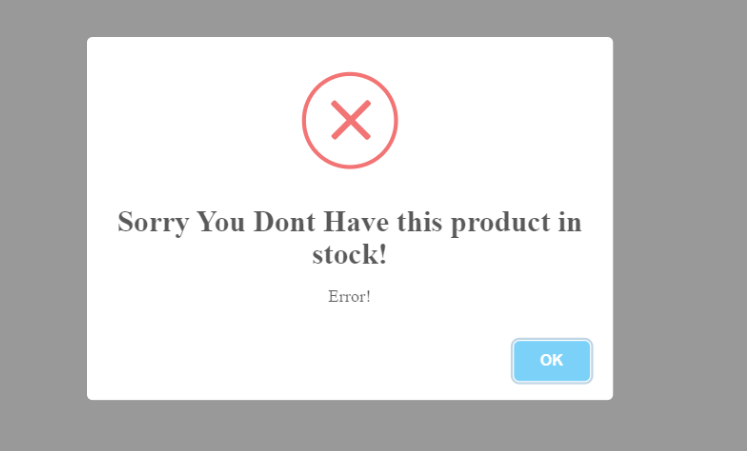


Figure 49 requests

## Technical Documentation

### Importing Files

* Download the zip file that was provided.
* Extract the zip file into your local device.
* The extracted file will contain the Web Client project, the Web service project and the MySQL Database.

### Configuring the Server

* The server used is apache tomcat server version 8.5.6
* So install the corresponding Tomcat server version in the local machine

### JDK Configuration

* The entire development is done based on the JDK 1.8 version. Therefore install the JDK 1.8 in your local machines as well.

### Using The Web Client

Note: the demonstration is done using NetBeans IDE. The same procedure would apply for any other IDE as well.

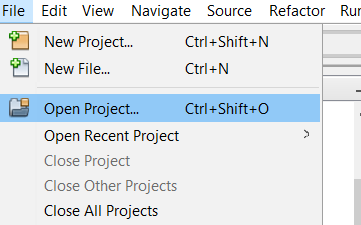


Figure 50 Import files

* You can import the web client project into you ide simply by browsing the downloaded project location.

### Starting the Project

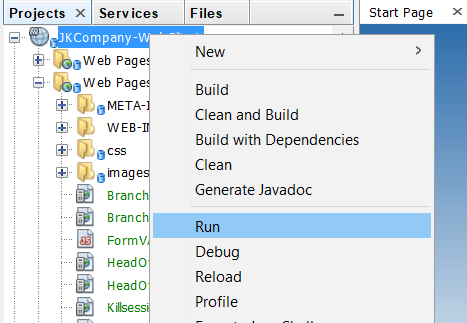


Figure 51 starting the project

* According to the above screen shot, run the program
* If you have all the required specifications in your system as mentioned earlier, the system itself will automatically setup the server environment and start the project.

### Login URL

http://localhost:8080/JKCompany-WebClient/UserLogin.jsp

* The login page will be available via the above URL

### Using The Web Service

Note: the demonstration is done using NetBeans IDE. The same procedure would apply for any other IDE as well.

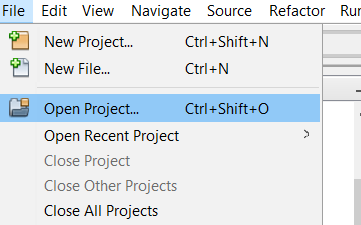


Figure 52 Import files

* You can import the web client project into you ide simply by browsing the downloaded project location.

### Configuring the Database

#### Adding the Dependency

Figure 53 adding dependency

* To work with sql, you will have to use the MySQL connector driver. To install this driver. Simply add the above dependency into your web service project pom.xml file and run the project. All the necessary files will be downloaded automatically

#### Importing the DB

* Import the database available in the extracted rar file into the local MySQL workbench.

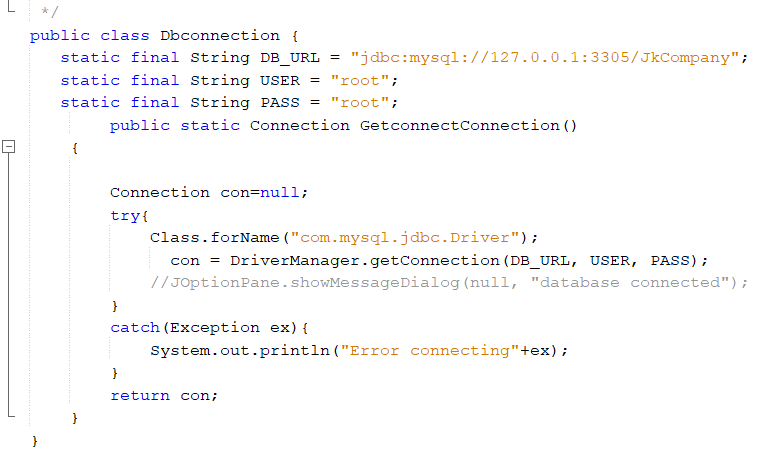


Figure 54Db connection

* This dB connection class consists of all the utilities required to connect to the imported MySQL database.
* Navigate yourself to the following dB connection class through the source packages available in the system,
* Once in, all what you got to do is to change the DB\_URL,USER,PASS variables according to your local devices MySQL Workbench Configuration

### Starting the Project

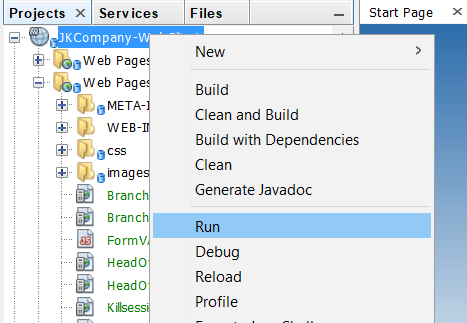


Figure 55 starting the project

* According to the above screen shot, run the program
* If you have all the required specifications in your system as mentioned earlier, the system itself will automatically setup the server environment and start the project.

# TASK G

## Identifying the ethical Issues

* Are the software’s used cracked versions or free versions?
* Does the system provide safety for the stakeholders’ information(privacy)?
* How supportive was the community around during the period of development?
* Addictive design of user interfaces (UX engineering)

## Analyzing the ethical considerations

### Are the software’s used cracked versions or free versions?

* When it comes to software violations, usage of cracked versions of genuine software can be highlighted. This could be considered as a great issue when it comes to the ethical considerations as it is not right to violate genuine software license and agreements and start using the cracked, free version of the original software.
* But, in the development I have clearly not violated any cracked version issues because the entire development was done based on NetBeans IDE which is a free and open source application and to draw diagrams, I have used the Draw.io online tool which is free and open source as well. So clearly, I have not violated such ethical rights during the development.

### Does the system provide safety for the stakeholders information(privacy)?

* The passwords of the users are visible in the database. So anyone who has the access to the database could see the password. This can be considered as a great ethical issue violation. So, the system should be further developed in order to support the users privacy by the usage of means like password encryption methodologies such as MD5 hashing functions.

### How supportive was the community around during the period of development?

* Here, community support does not mean of asking someone for their code and copying and pasting it into mine.
* During the period of development I had to face various issues for instance, to figure out the email function it took me days. But I still had no answer, so I had to post my question in stack overflow. To be honest I didn’t expect a reply. But, to see within 1 hour there were so much of people around the world who supported me with my issue and even provided the way how to solve the problem.
* This fact proves that the community around was very supportive during the development period.

### Addictive design of user interfaces (UX engineering)

* Even though attractive user interfaces brings out advantages, it has its drawbacks as well.
* For instance strongly addictive interfaces could grab the attraction of consumers soon and they would expect the same attraction from every other website or application they use.
* In my development I believe I have not violated this ethical issue because the user interfaces are at satisfactory level.

## Tools and functions used to achieve software carpentry

* The Back end Development is done entirely using **java** as the programming language.
* The webservice is a SOAP based API.
* The webpages are all in the JSP format.
* Usage of Html/CSS for user interfaces and Javascript for client side validations.
* Bootstrap 4 is used to give the system a blend of attractive user interfaces.
* jQuery is used to give animations and support the data tables.
* Sweet alert javascript library was used to give alert messages in a more descriptive and an attractive way.

## Tools and functions used to achieve software Codemanship

These were the things that were considered when achieving Codemanship

1. Readability:- The code should be easy to understand. And the coding in the system is understandable to anyone who has a basic knowledge in programming and OOP concepts.
2. Complexity:-If the code is complex it hard to understand. But, in the system. Except for the stock management part, the rest of the coding is very simple and the flow of arguments are very straight forward so anyone who understand OOP can get to know how the development is done.
3. Duplication:- refers to the copying and pasting code. Well, The methods for the CRUD operations related to the database which are available in the Dbutil class were obtained from online sources and the texts and functions were altered according to my requirements. Other than that the entire system development was done by me.
4. Dependencies:-In the system, dependencies are placed in the pom.xml file and if some dependencies were altered, the system would not function. For instance, if the java.mail dependency was removed, the mailing functions would not work. Similarly, if the MySQL connector dependency was removed, the operations related to the database will not be applicable.

These were the disciplines that were considered when achieving Codemanship

1. Test-Driven Development:-The system was tested via automated testing by the usage of unit test approach, each method was individually tested for any bugs and the issues were fixed then and there.
2. Object-Oriented design:- The system uses best practices related to OOP concepts. Design patterns like singleton was used, Concepts like Inheritance, Encapsulation, Polymorphism were used through-out the development as well.

# Conclusion

The developed system meets the requirements of the Assignment. But, this system should be further developed if it is to be placed in the industry. For instance, security features of the system should be developed. And additionally, since there is a web service, the system could be made more distributed by developing a Mobile application and a desktop application by using the same webservice. This, will improve the user experience in a greater extend.

# References

JavaTPoint, 2018. *JavaTPoint.* [Online]   
Available at: https://www.javatpoint.com/factory-method-design-pattern  
[Accessed 18 02 2021].

Poyiyas, A., 2018. *Andreas Poyiyas.* [Online]   
Available at: https://medium.com/@andreaspoyias/design-patterns-a-quick-guide-to-singleton-pattern-60732ed43956  
[Accessed 18 02 2021].

Rahman, S., 2019. *FreeCodeCamp.* [Online]   
Available at: https://www.freecodecamp.org/news/the-basic-design-patterns-all-developers-need-to-know/  
[Accessed 18 02 2021].