BSC-AdvancedProgramming-BSCSD-21-14.docx

by Lokukankanamge Chamal Sanjana Peiris

Submission date: 25-Feb-2021 05:00PM (UTC+0000)

Submission ID: 145518201

File name: 80519_Lokukankanamge_Chamal_Sanjana_Peiris_BSC-AdvancedProgramming-BSCSD-21-14_962364_1150214854.docx (8.37M)

Word count: 5594

Character count: 29900

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

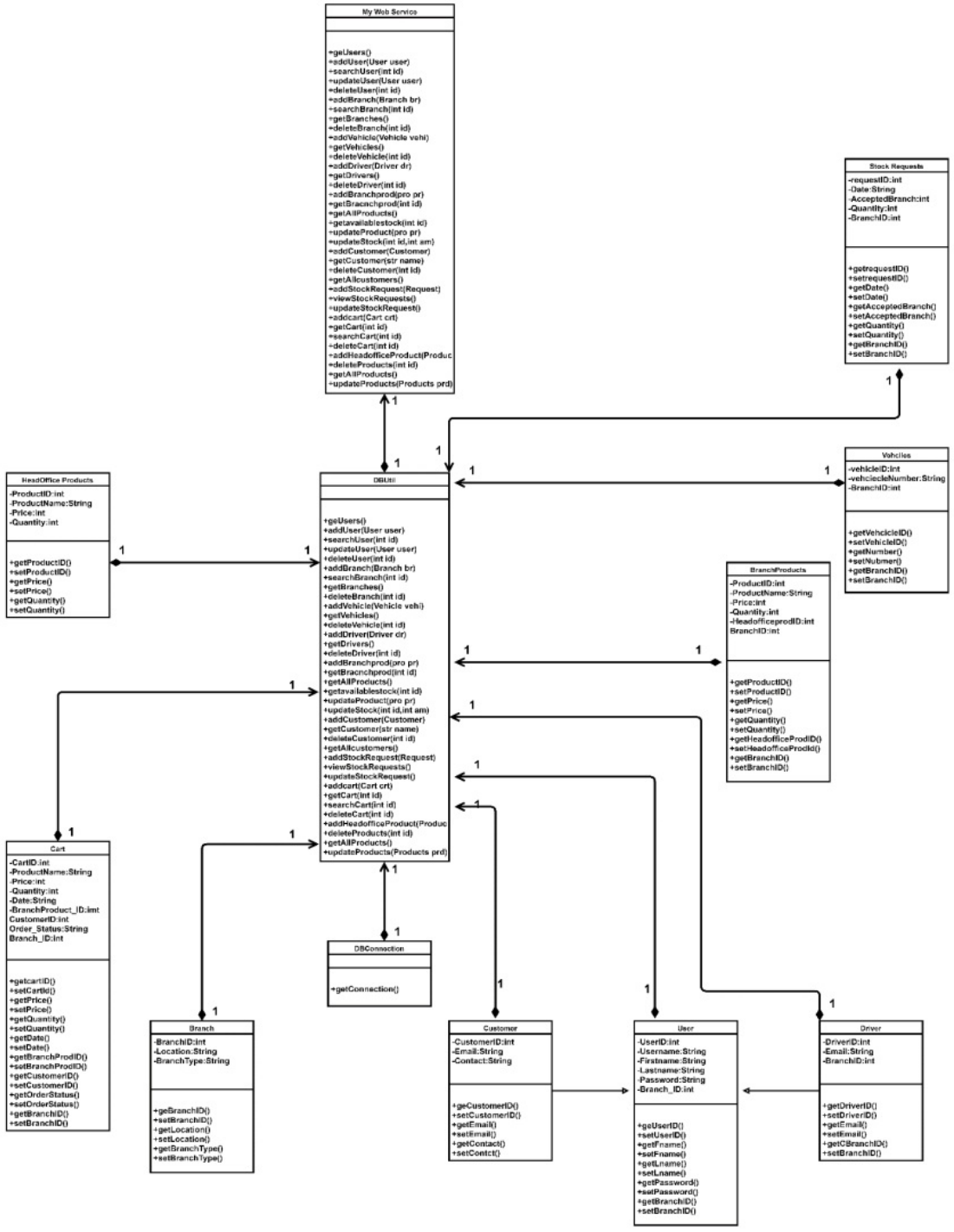
Functionalities of the Branch Users

- **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

3. 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

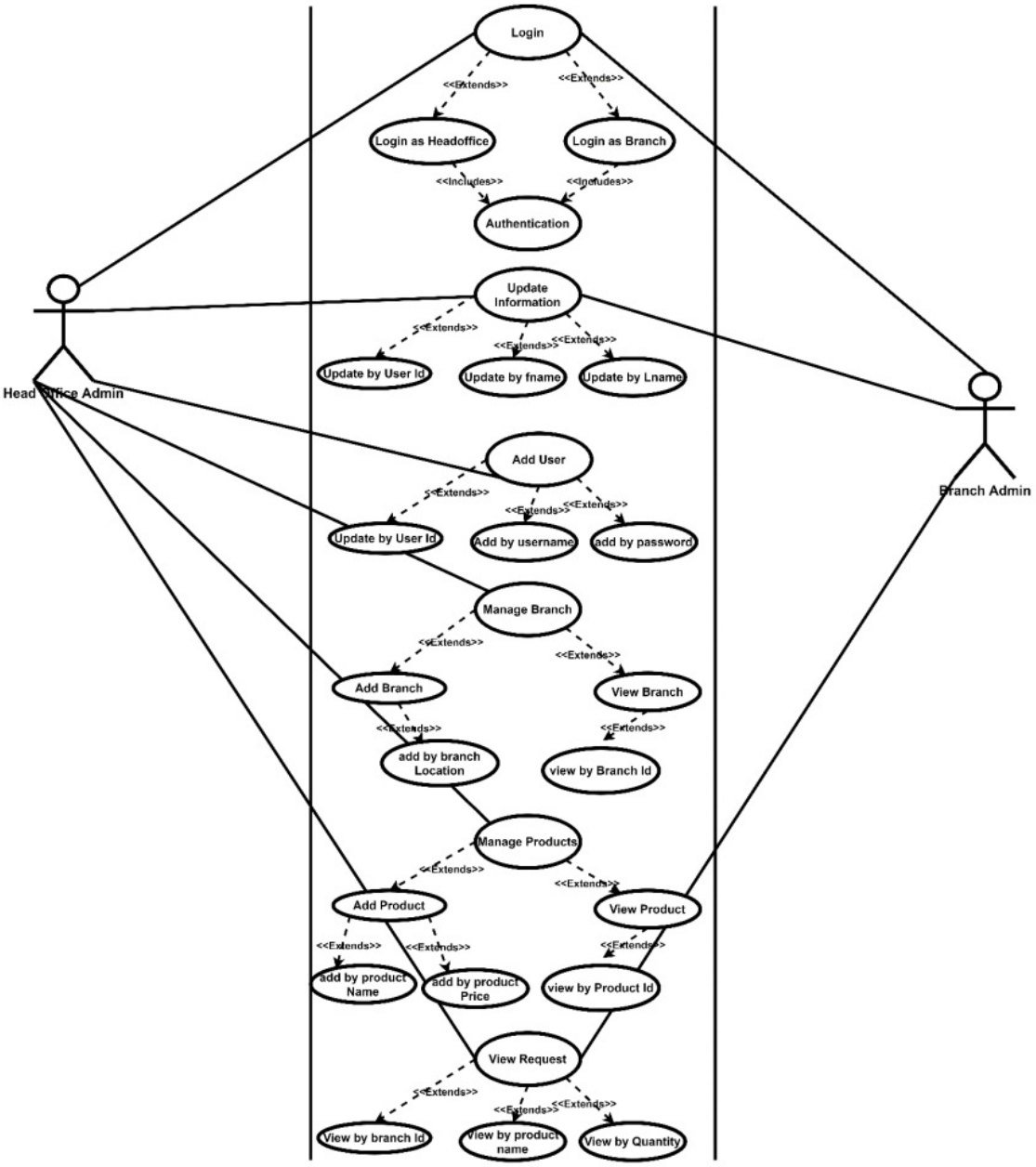
Multiplicity

4. 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.

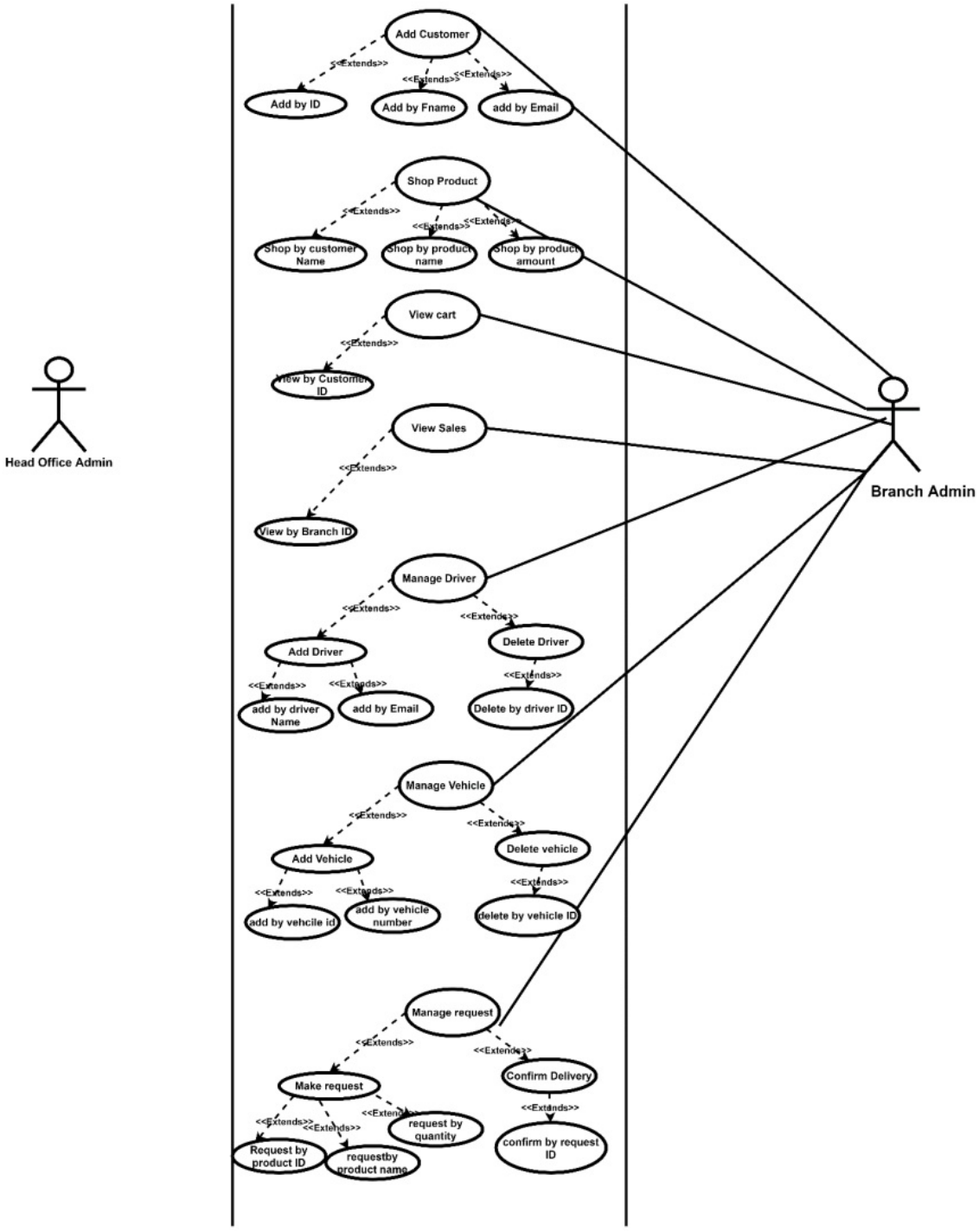
Use-Case Diagram

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.

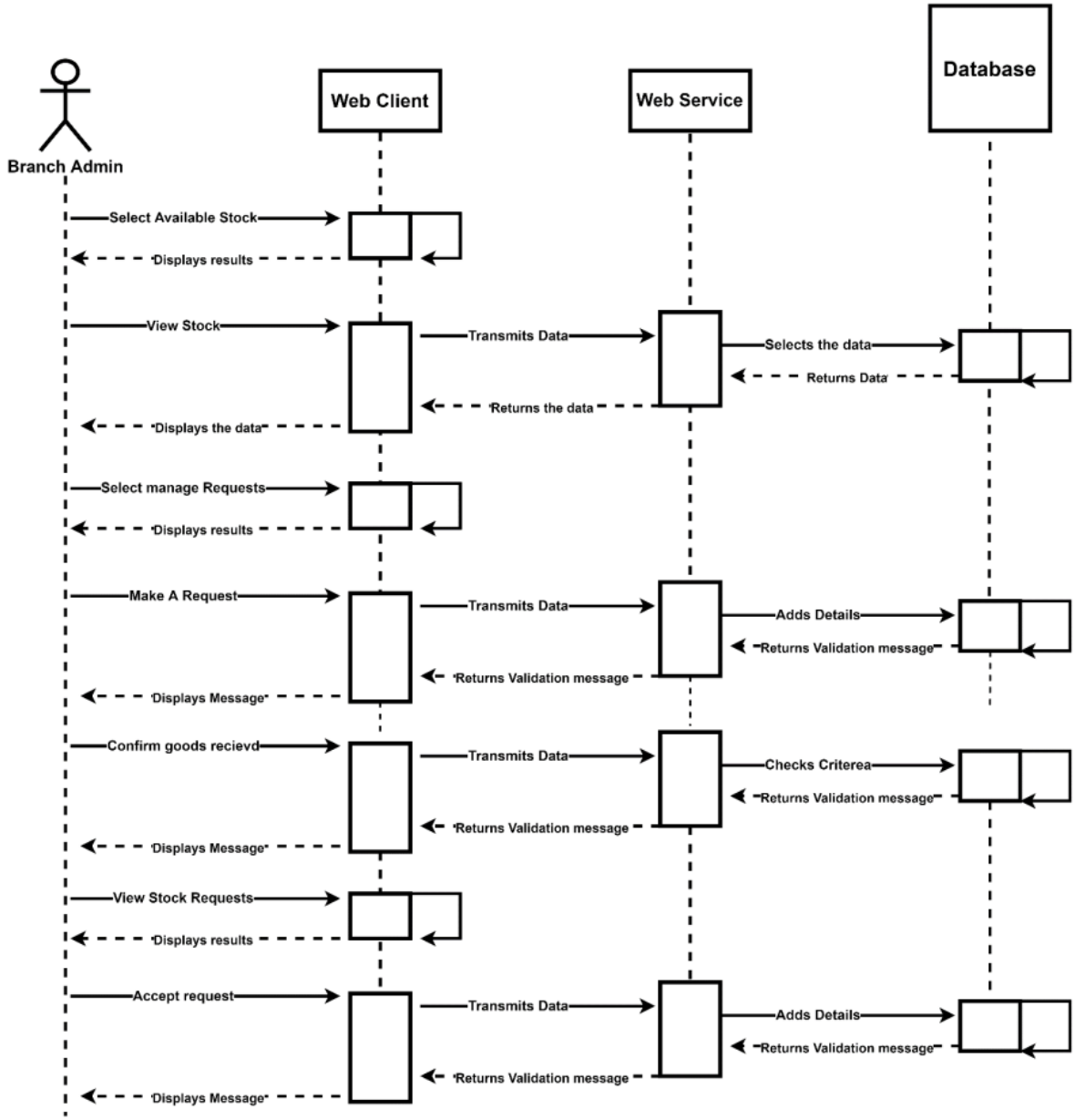
Head-Office Admin Sequence

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

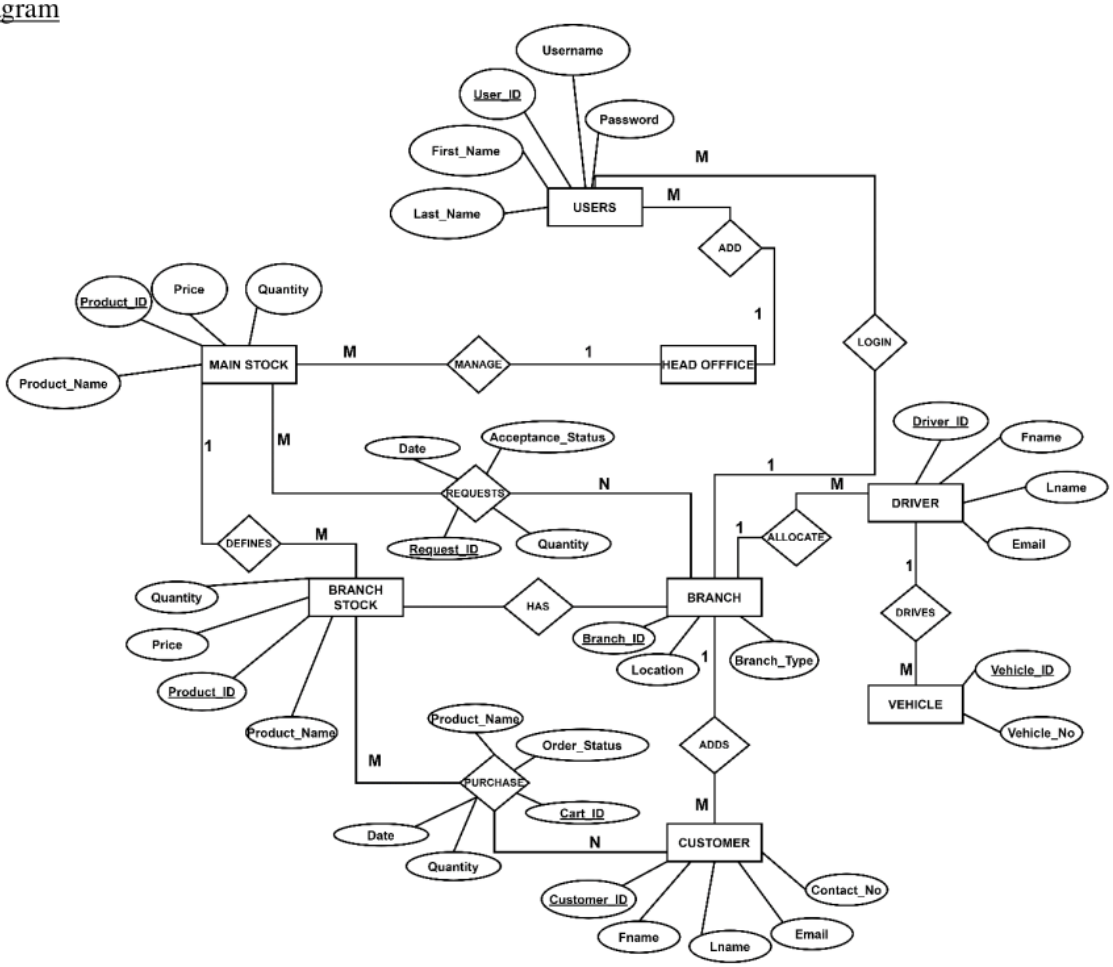


Figure 1 Er Diagram

- 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

```
public class Dbconnection {
   static final String DB_URL = "jdbc:mysql://127.0.0.1:3305/JkCompany";
   static final String USER = "root";
   static final String PASS = "root";
       public static Connection GetconnectConnection()
    {

       Connection con=null;
       try{
           Class.forName("com.mysql.jdbc.Driver");
             con = DriverManager.getConnection(DB_URL, USER, PASS);
           //JOptionPane.showMessageDialog(null, "database connected");
       }
       catch(Exception ex){
           System.out.println("Error connecting"+ex);
       }
       return con;
    }
}
```

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

```
public static boolean updateUser(user user){

    Connection conn = null;  //instantiating an object
    Statement stmt = null;

    try{

        conn=Dbconnection.GetconnectConnection(); //getting the connection

      stmt = conn.createStatement();
      String sql = "update users set Username='"+user.getUsername()+"',Firstna
      stmt.executeUpdate(sql);

      return true;

    }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
    }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
```

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

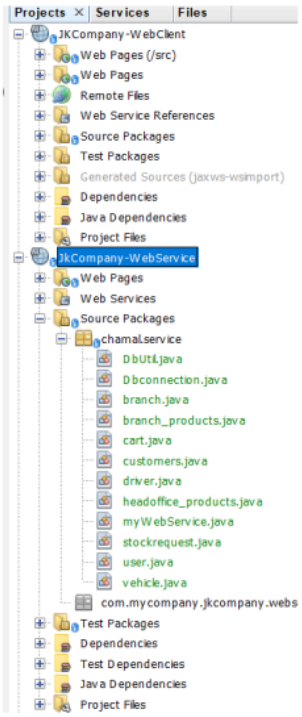


Figure 4 Webservices

- 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

```
//for users

 @WebMethod(operationName = "getusers")
public user getusers(@WebParam(name = "username") String username, String password) {

    user user= DbUtil.getUser(username,password);

    return user;
}

    @WebMethod(operationName = "checkuser")
public user checkuser(@WebParam(name = "checkuser") String username, int branchid) {

    user user=DbUtil.checkuser(username, branchid);
  // insert to db
    return user;
}

   @WebMethod(operationName = "addUser")
public boolean addUser(@WebParam(name = "addUser") user user) {

    boolean success=DbUtil.addUser(user);
  // insert to db
    return success;
}
```

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

```
public class vehicle {
    private int vehicle_ID;
    private String vehicle_no;
    private int Branch_ID;

      public vehicle() {

    }


    public vehicle(int vehicle_ID, String vehicle_no, int Branch_ID) {
        this.vehicle_ID = vehicle_ID;
        this.vehicle_no = vehicle_no;
        this.Branch_ID = Branch_ID;
    }

    public int getVehicle_ID() {
        return vehicle_ID;
    }

    public void setVehicle_ID(int vehicle_ID) {
        this.vehicle_ID = vehicle_ID;
    }

    public String getVehicle_no() {
        return vehicle_no;
    }
```

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.

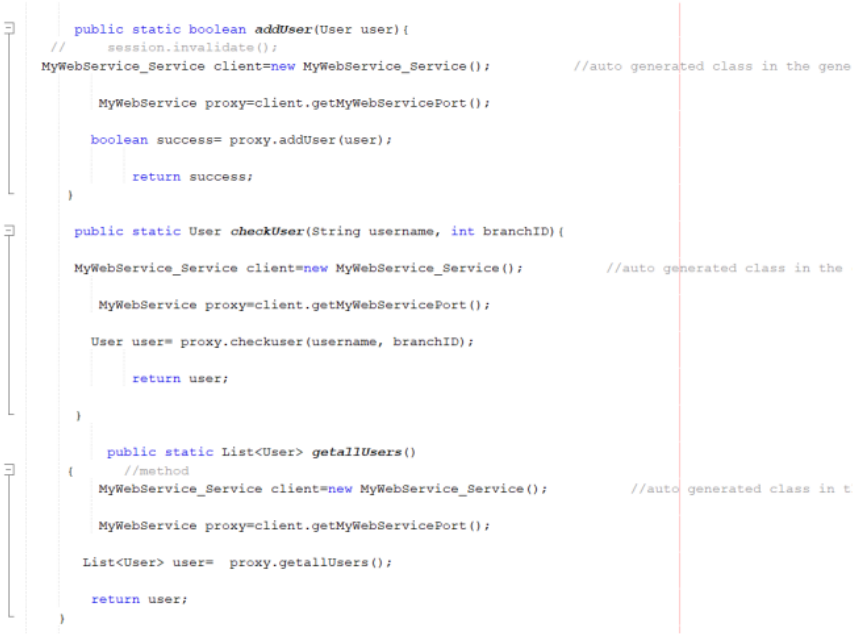
Dbutil class

```
public static user getUser(String username,String password) {
        user user= null;
       Connection conn = null;
       Statement stmt = null;
       try{

        //Class.forName("com.mysql.jdbc.Driver");
        // conn = DriverManager.getConnection(DB_URL, USER, PASS);
          conn=Dbconnection.GetconnectConnection();
         stmt = conn.createStatement();
         String sql = "SELECT * from Users WHERE Username='"+username+"' And Password='"+password+"'";
         ResultSet rs = stmt.executeQuery(sql);
         while(rs.next()){
         user =new user();
           user.setUserID(rs.getInt("UserID"));
           user.setUsername(rs.getString("Username"));
           user.setFname(rs.getString("Firstname"));
           user.setLname(rs.getString("Lastname"));
           user.setPassword(rs.getString("Password"));
           user.setBranch_ID(rs.getInt("Branch_ID"));

         }
         rs.close();
       }catch(SQLException se){
         //Handle errors for JDBC
         se.printStackTrace();
       }catch(Exception e){
         //Handle errors for Class.forName
         e.printStackTrace();
       }finally{
         //finally block used to close resources
         try{
            if(stmt!=null)
               conn.close();
```

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



```
    public static boolean addUser(User user){
//      session.invalidate();
MyWebService_Service client=new MyWebService_Service();          //auto generated class in the gene
        MyWebService proxy=client.getMyWebServicePort();
       boolean success= proxy.addUser(user);
            return success;
    }
    public static User checkUser(String username, int branchID){
    MyWebService_Service client=new MyWebService_Service();          //auto generated class in the
       MyWebService proxy=client.getMyWebServicePort();
      User user= proxy.checkuser(username, branchID);
            return user;
    }
        public static List<User> getallUsers()
    {       //method
       MyWebService_Service client=new MyWebService_Service();          //auto generated class in t
       MyWebService proxy=client.getMyWebServicePort();
     List<User> user=  proxy.getallUsers();
      return user;
    }
```

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

```
public class vehicle {
    private int vehicle_ID;
    private String vehicle_no;
    private int Branch_ID;

      public vehicle() {

    }


    public vehicle(int vehicle_ID, String vehicle_no, int Branch_ID) {
        this.vehicle_ID = vehicle_ID;
        this.vehicle_no = vehicle_no;
        this.Branch_ID = Branch_ID;
    }

    public int getVehicle_ID() {
        return vehicle_ID;
    }

    public void setVehicle_ID(int vehicle_ID) {
        this.vehicle_ID = vehicle_ID;
    }

    public String getVehicle_no() {
        return vehicle_no;
    }

    public void setVehicle_no(String vehicle_no) {
        this.vehicle_no = vehicle_no;
    }

    public int getBranch_ID() {
        return Branch_ID;
```

Figure 9 Encapsulation

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

Inheritance

```
public class driver extends user{
```

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

```
public static User authenticate(String username, String password) {
    User authenticatedUser = null;

    //This should loaded from the DB
     MyWebService_Service client=new MyWebService_Service();          //aut

     MyWebService proxy=client.getMyWebServicePort();

    User user= proxy.getusers(username,password);

    //Authenticated the users password
    if (user!=null && password.equals(user.getPassword())) {
        authenticatedUser= user;
    }

    return authenticatedUser;
}


public static User authenticate(Cookie[] cookies, HttpSession session) {
    // Authenticate the user from cookie session
    User user = null;

    // Iterate all the cookies from the client request
    for (Cookie cookie : cookies) {
        // Checks SESID cookie
        if (cookie.getName().equals("SESID")) {
            // Lookup SESID cookie value from sessions
            Object sessionObj = session.getAttribute(cookie.getValue());

            // Load the user from session object if it exists
            if (sessionObj != null) {
                user = (User)sessionObj;
```

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

```
public static  String  alertByEmail(String emailMessage){
    try{

        final String fromEmail = "jkcompany@gmail.com";
        final String password = "******"; //fromEmail password
        final String toEmail = "chamal.peiris3g@gmail.com";
        System.out.println("Email configuration code start");
        Properties props = new Properties();
        props.put("mail.smtp.host", "smtp.gmail.com"); //SMTP Host set by default this
        props.put("mail.smtp.port", "587"); //TLS Port you can use 465 insted of 587
        props.put("mail.smtp.auth", "true"); //enable authentication
        props.put("mail.smtp.starttls.enable", "true"); //enable STARTTLS
        //create Authenticator object to pass in Session.getInstance argument
        Authenticator auth = new Authenticator()
        {
         protected PasswordAuthentication getPasswordAuthentication()
            {
                return new PasswordAuthentication(fromEmail, password);
            }
        };
                Session session = Session.getInstance(props, auth);

                MimeMessage message = new MimeMessage(session);
                message.setFrom(new InternetAddress(fromEmail));
                message.addRecipient(Message.RecipientType.TO, new
                                                    InternetAddress(toEmail));
                message.setSubject("ALERT");
                message.setText(emailMessage);//here you can write a msg what you want to send.
                System.out.println("text:"+emailMessage);
                Transport.send(message);//here mail sending process start.
                System.out.println("Mail Sent Successfully");
    }
    catch(Exception ex)
    {
                System.out.println("Mail fail");
                System.out.println(ex);
```

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

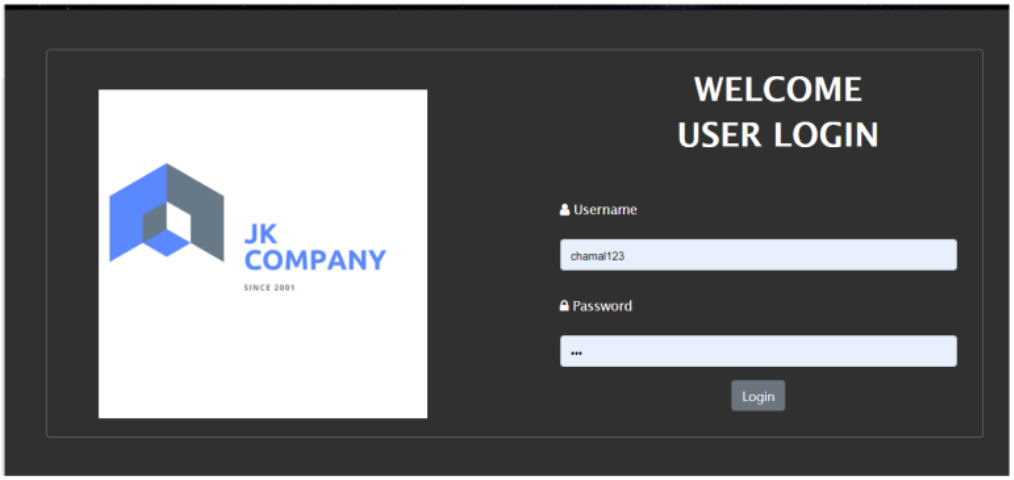


Figure 13Login UI

Code Explanation

Usage of sessions and cookies

```
<%
User user = Webhelper.authenticate(request.getCookies(), session);
if (user == null) {
        String username = request.getParameter("username");
        String password = request.getParameter("password");


        if (username != null && password != null) {

            user = Webhelper.authenticate(username, password);
            if (user != null) {
                String sessionId = UUID.randomUUID().toString().replace("-", "").toUpperCase();
                // Create cookie and attach it to response
                Cookie cookie = new Cookie("SESID", sessionId);
                response.addCookie(cookie);
                // Create session attribute
                session.setAttribute(sessionId, user);
               // out.print("Welcome " + user.getFname()+ " " + user.getBranchID());
                int id=user.getBranchID();
                if(id==1){
                   out.print("Welcome Head office " + user.getFname()+ " " + user.getBranchID());
                   response.sendRedirect("HeadOfficeHomepage.jsp");
                     session.setAttribute("branchID",user);
                // response.sendRedirect("HeadofficeLogin.jsp");
                        }
                        else{
                        response.sendRedirect("BranchHomepage.jsp");
                         out.print("Welcome branch");
                         session.setAttribute("branchID",user);
                            }
                    }
```

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

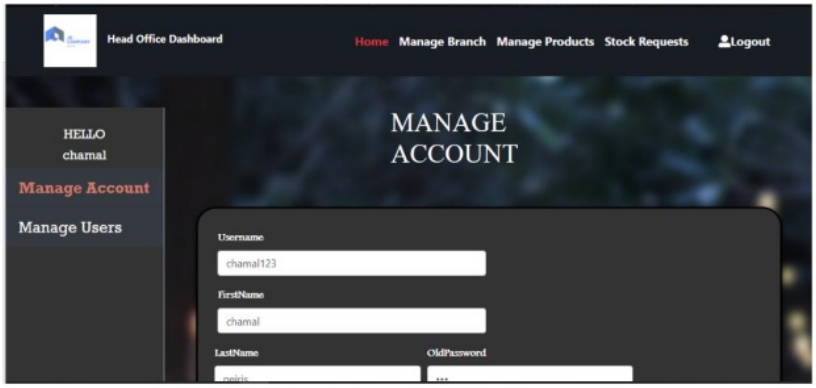


Figure 15 Headoffice Homepage

Branch Home page

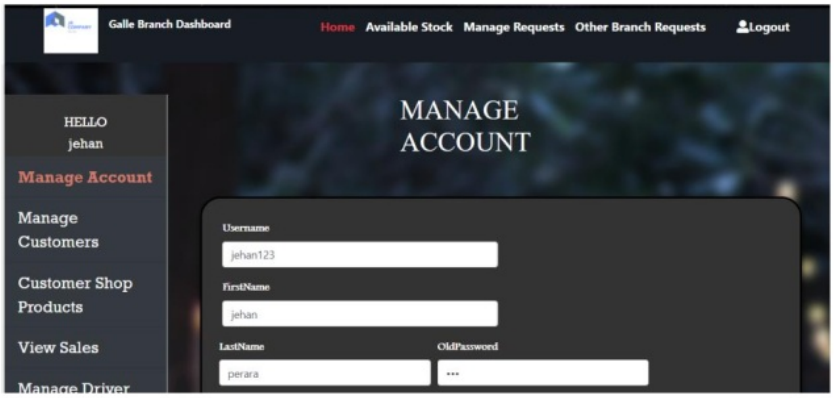
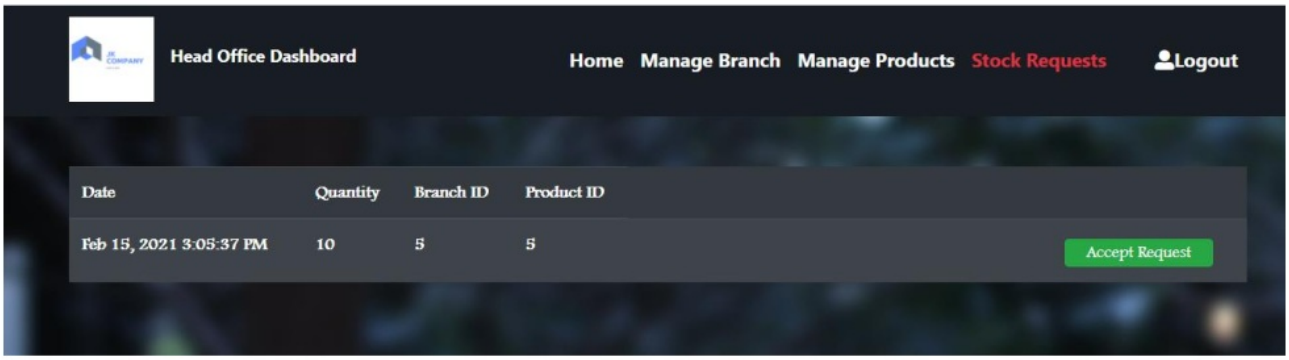


Figure 16 Branch Homepage

Head office Manage Requests

Interface



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

Code

```
<%
    User user= (User)session.getAttribute("branchID");


     Branch branch= Webhelper.getonebranch(user.getBranchID());
     String branchLocation=branch.getLocation();
     int requestID=Integer.valueOf(request.getParameter("RequestID"));
     int quantity=Integer.valueOf(request.getParameter("Quantity"));
     int RecievdbranchID=Integer.valueOf(request.getParameter("BranchID"));
     int RecievedproductID=Integer.valueOf(request.getParameter("HeadofficeproductID"));


      boolean valid= Webhelper.updateStockrequest(requestID, branchLocation);
  // out.println("Success");
  if(valid==true){
   out.println("Success");

   }
   else{
   out.println("Error");

   }
%>
```

- 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

	Request_ID	Date	Accepted_By	Quantity	Branch_ID	Product_ID
	3	Feb 9, 2021 6:26:34 PM	Colombo	10	2	3

Branch Manage requests

Make a Request

Interface

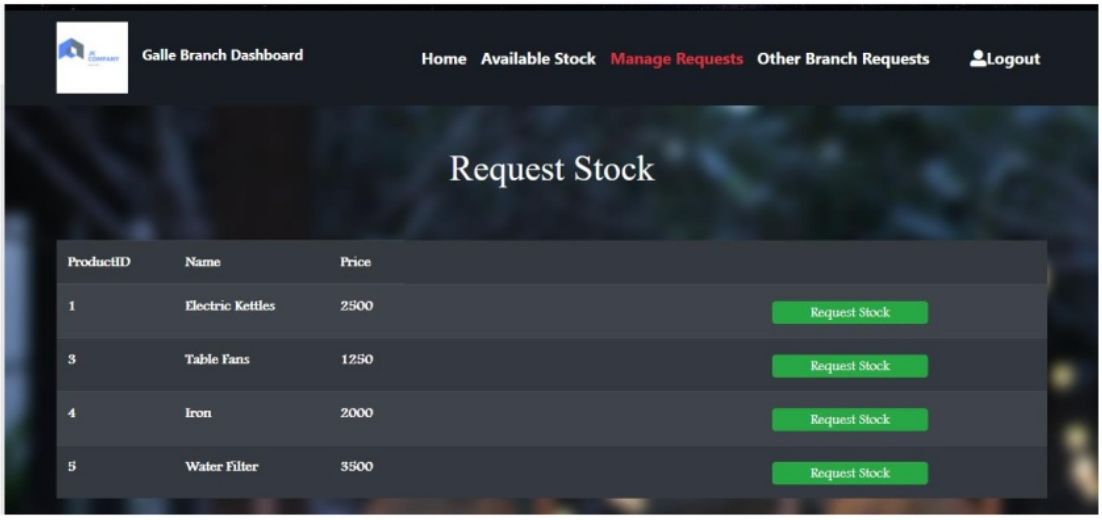


Figure 17 Branch make request

Code

```
    if(request.getParameter("amount")!=null)
    {

    Stockrequest stock=Webhelper.getrequest(branchID, headofficeproduct_ID);
    if(stock==null){
    amount= Integer.valueOf(request.getParameter("amount"));
      out.println(amount);
      String date=new java.util.Date().toLocaleString();



    Stockrequest obj = new Stockrequest();

    obj.setDate(date);
    obj.setAcceptedBranch("Request still not accepted");
    obj.setQuantity(amount);
    obj.setBranchId(branchID);
    obj.setHeadofficeproductId(headofficeproduct_ID);

    boolean success= Webhelper.addRequest(obj);
    if(success==true){
    out.println("success");

    //response.sendRedirect("ManageRequests.jsp");
    }
    else{
    out.println("Error");
    }
   }
     //out.println("Error");
        else{
        out.println("YOU HAVE ALREADY REQUESTED");

            }

}
```

- 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

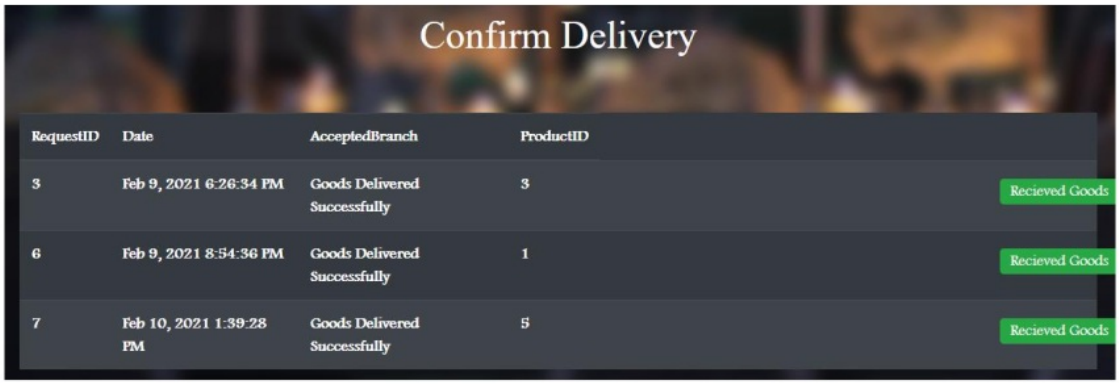


Figure 18 confirm delivery

- 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

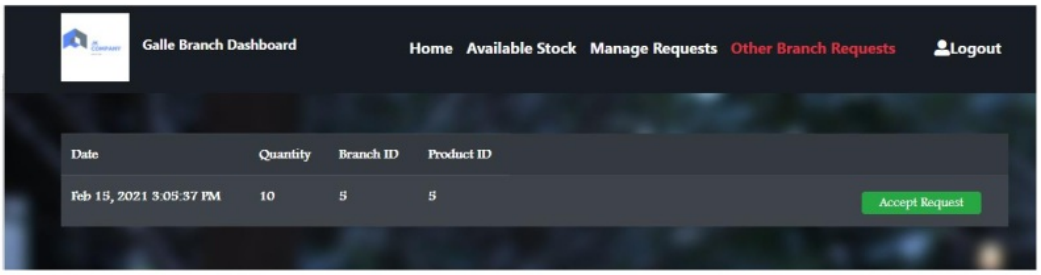


Figure 19 Accept request

Code

```
Branch branch= Webhelper.getonebranch(user.getBranchID());
String branchLocation=branch.getLocation();
int requestID=Integer.valueOf(request.getParameter("RequestID"));
int quantity=Integer.valueOf(request.getParameter("Quantity"));
int RecievdbranchID=Integer.valueOf(request.getParameter("BranchID"));
int RecievedproductID=Integer.valueOf(request.getParameter("HeadofficeproductID"));


BranchProducts products= Webhelper.validatebranchProducts(user.getBranchID(), RecievedproductID);

            if(RecievdbranchID==user.getBranchID()){

           // out.println("");
              out.println("<script> swal('Sorry You Cannot accept your Own Requests!', 'Error!', 'error');</script>");
            }

          else if(products==null){

          // out.println("");
             out.println("<script> swal('Sorry You Dont Have this product in stock!', 'Error!', 'error');</script>");
          }
          else{
          boolean valid= Webhelper.updateStockrequest(requestID, branchLocation);
         // out.println("Success");
         if(valid==true){
          out.println("Success");

          }
          else{
          out.println("Error");

          }
          }
```

Figure 20 accept request

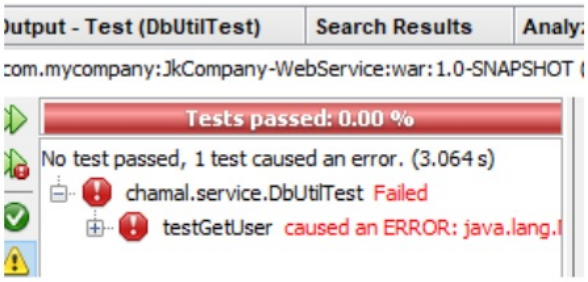
- To accept the request, the branch should have the particular products, so first, it will check whether the requested product is available in that branch if not and error message will be generated.
- If they try to accept their own request, they will be notified as well.
- If not the stock accepted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

TASK E

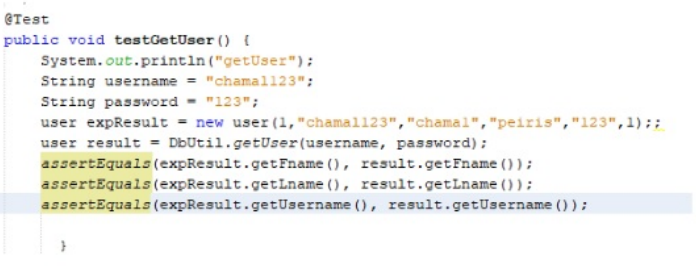
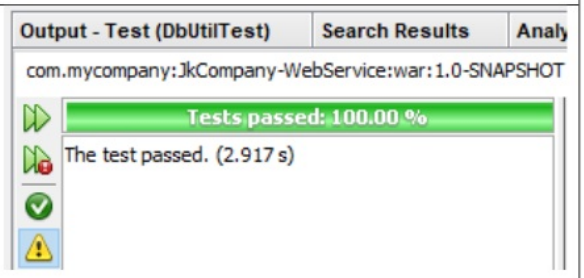
How test driven development is used in the system

- 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

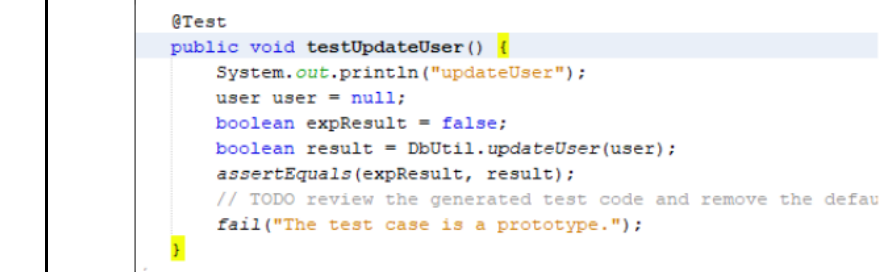
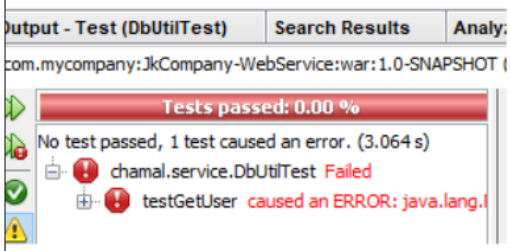
Method Before	Results Before
	

```
@Test
public void testGetUser() {
    System.out.println("getUser");
    String username = "";
    String password = "";
    user expResult = null;
    user result = DbUtil.getUser(username, password);
    assertEquals(expResult.getFname(), result.getLname());

     }
```

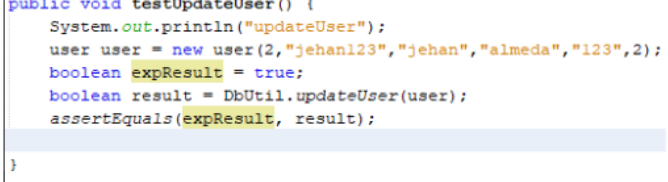
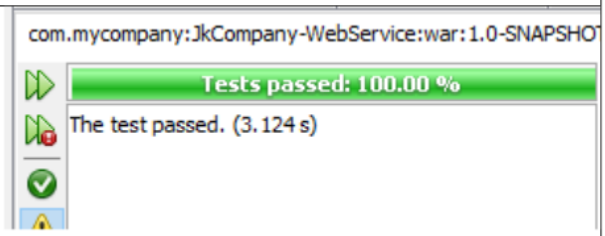

Method After	Results After
	

```
@Test
public void testGetUser() {
    System.out.println("getUser");
    String username = "chamal123";
    String password = "123";
    user expResult = new user(1,"chamal123","chamal","peiris","123",1);;
    user result = DbUtil.getUser(username, password);
    assertEquals(expResult.getFname(), result.getFname());
    assertEquals(expResult.getLname(), result.getLname());
    assertEquals(expResult.getUsername(), result.getUsername());

     }
```

Checking the updateUser() method

Method Before	Results Before
	

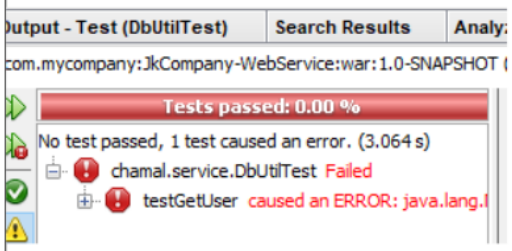
```
@Test
public void testUpdateUser() {
    System.out.println("updateUser");
    user user = null;
    boolean expResult = false;
    boolean result = DbUtil.updateUser(user);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the defau
    fail("The test case is a prototype.");
}
```


Method After	Results After
	

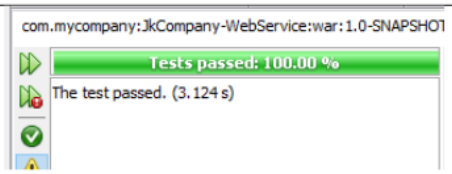
```
@Test
public void testUpdateUser() {
    System.out.println("updateUser");
    user user = new user(2,"jehan123","jehan","almeda","123",2);
    boolean expResult = true;
    boolean result = DbUtil.updateUser(user);
    assertEquals(expResult, result);

}
```

Checking the addUser() method

Method Before	Results Before

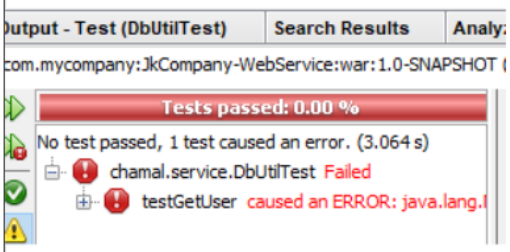
`@Test`
`public void testAddUser() {`
`  System.out.println("addUser");`
`  user user = null;`
`  boolean expResult = false;`
`  boolean result = DbUtil.addUser(user);`
`  assertEquals(expResult, result);`
`  // TODO review the generated test code and remove the de`
`  fail("The test case is a prototype.");`
`}`
	

Method After	Results After

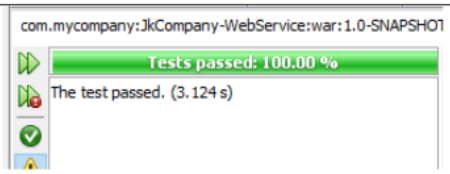
`@Test`
`public void testAddUser() {`
`  System.out.println("addUser");`
`  user user = new user(9,"darren123","darren","sammy","123",5);`
`  boolean expResult = true;`
`  boolean result = DbUtil.addUser(user);`
`  assertEquals(expResult, result);`
`  // TODO review the generated test code and remove the default call`
`  fail("The test case is a prototype.");`
`}`
	

Checking the checkUser() method

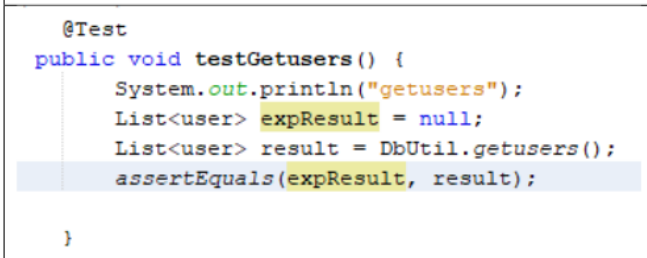
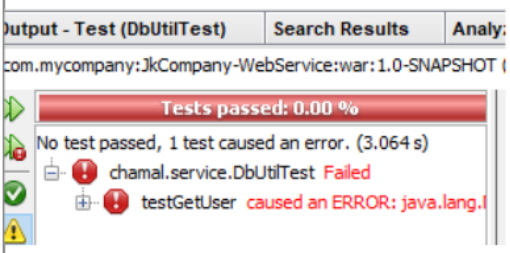
Method Before	Results Before

`@Test`
`public void testCheckuser() {`
`System.out.println("checkuser");`
`String username = "";`
`int branchID = 0;`
`user expResult = null;`
`user result = DbUtil.checkuser(username, branchID);`
`assertEquals(expResult, result);`
`}`
	

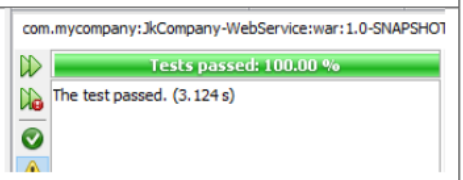
Method After	Results After

`@Test`
`public void testCheckuser() {`
`System.out.println("checkuser");`
`String username = "chamal123";`
`int branchID = 1;`
`user expResult = new user(1,"chamal123","chamal","peiris","123",1);`
`user result = DbUtil.checkuser(username, branchID);`
`assertEquals(expResult, result);`
`}`
	

Checking the getUsers() method

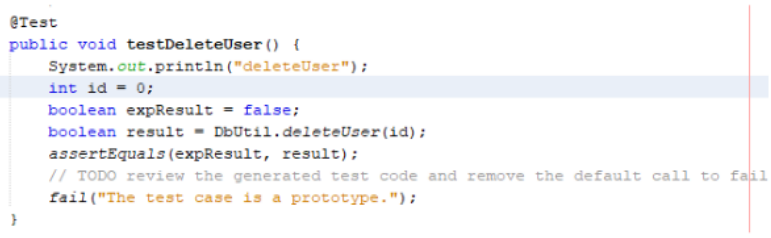
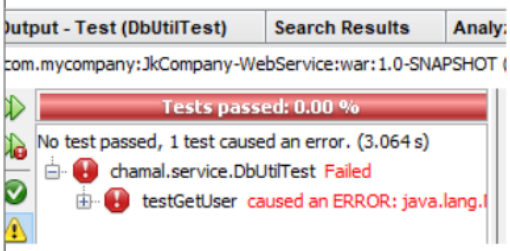
Method Before	Results Before
	

```
@Test
public void testGetusers() {
        System.out.println("getusers");
        List<user> expResult = null;
        List<user> result = DbUtil.getusers();
        assertEquals(expResult, result);

    }
```

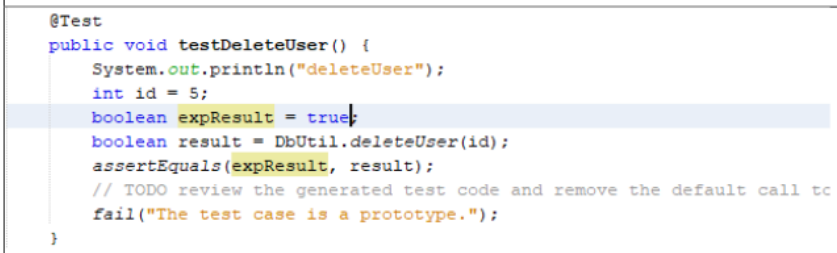
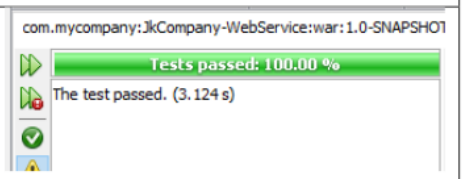

Method After	Results After
	

```
@Test
public void testGetusers() {
        System.out.println("getusers");
        List<user> expResult = new ArrayList<>();
        user obj= new user(1,"chamal123","chamal","peiris","123",1);
        user obj2= new user(2,"jehan123","jehan","perera","123",2);
        expResult.add(obj);
        expResult.add(obj2);
        List<user> result = DbUtil.getusers();
        assertEquals(expResult.get(0).getUsername(), result.get(0).getUsername());

    }
```

Checking the deleteUser() method

Method Before	Results Before
	

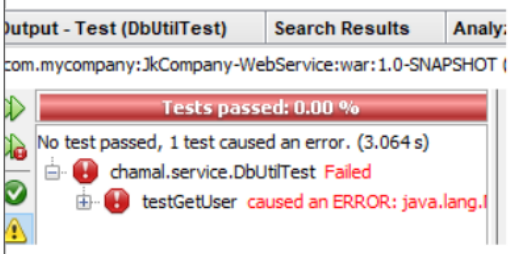
```
@Test
public void testDeleteUser() {
    System.out.println("deleteUser");
    int id = 0;
    boolean expResult = false;
    boolean result = DbUtil.deleteUser(id);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail
    fail("The test case is a prototype.");
}
```


Method After	Results After
	

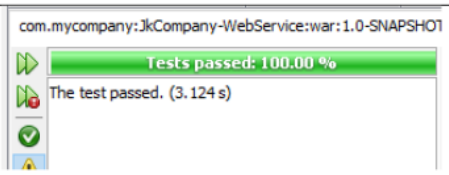
```
@Test
public void testDeleteUser() {
    System.out.println("deleteUser");
    int id = 5;
    boolean expResult = true;
    boolean result = DbUtil.deleteUser(id);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to
    fail("The test case is a prototype.");
}
```

Checking the addBranch() method

Method Before	Results Before

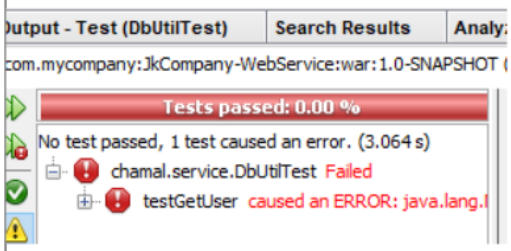
@Test
public void testAddBranch() {
System.out.println("addBranch");
branch branch = null;
boolean expResult = false;
boolean result = DbUtil.addBranch(branch);
assertEquals(expResult, result);
// TODO review the generated test code and remove the default c
fail("The test case is a prototype.");
}
	

Method After	Results After

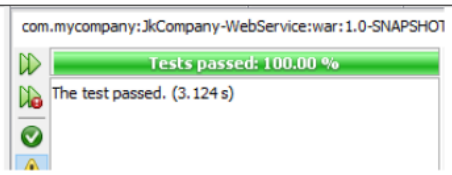
@Test
public void testAddBranch() {
System.out.println("addBranch");
branch branch = new branch(0,"Bambalapitiya","Branch");
boolean expResult = true;
boolean result = DbUtil.addBranch(branch);
assertEquals(expResult, result);
 	
}
	

Checking the getBranch() method

Method Before	Results Before

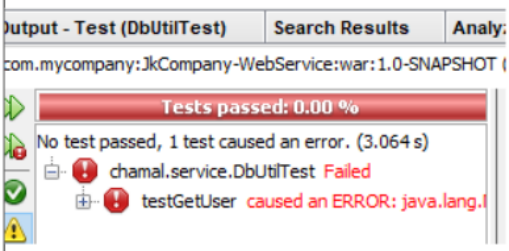
`@Test`
`public void testGetBranch() {`
`System.out.println("getBranch");`
`List<branch> expResult = null;`
`List<branch> result = DbUtil.getBranch();`
`assertEquals(expResult, result);`
`// TODO review the generated test code and remove`
`fail("The test case is a prototype.");`
`}`
	

Method After	Results After

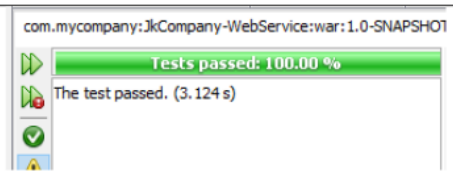
`@Test`
`public void testGetBranch() {`
`System.out.println("getBranch");`
`List<branch> expResult =new ArrayList<>();`
`branch obj= new branch(1,"colombo","Head_office");`
`branch obj2= new branch(2,"Galle","Branch");`
`expResult.add(obj);`
`expResult.add(obj2);`
`List<branch> result = DbUtil.getBranch();`
`assertEquals(expResult.get(0).getBranchId(), result.get(0).getBranchId());`
`}`
	

Checking the getOneBranch() method

Method Before	Results Before

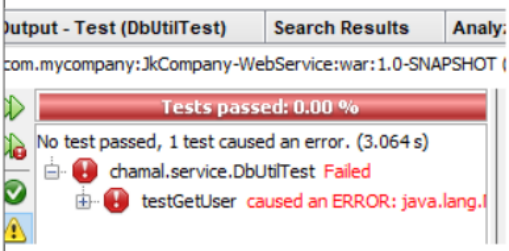
`@Test`
`public void testGetoneBranch() {`
`System.out.println("getoneBranch");`
`int id = 0;`
`branch expResult = null;`
`branch result = DbUtil.getoneBranch(id);`
`assertEquals(expResult, result);`
`}`
	

Method After	Results After

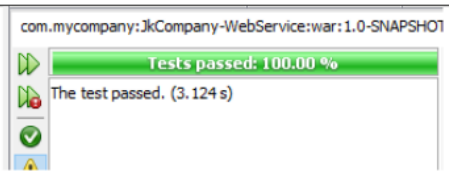
`@Test`
`public void testGetoneBranch() {`
`System.out.println("getoneBranch");`
`int id = 1;`
`branch expResult = new branch(1,"Colombo","Head-office");`
`branch result = DbUtil.getoneBranch(id);`
`assertEquals(expResult, result);`
`}`
	

Checking the deleteBranch() method

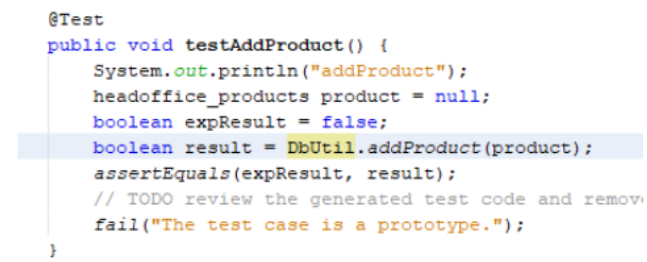
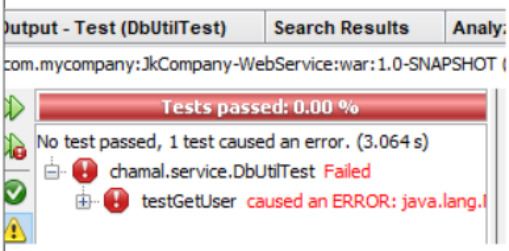
Method Before	Results Before

`@Test`
`public void testDeleteBranch() {`
`  System.out.println("deleteBranch");`
`  int id = 0;`
`  boolean expResult = false;`
`  boolean result = DbUtil.deleteBranch(id);`
`  assertEquals(expResult, result);`
`}`
	

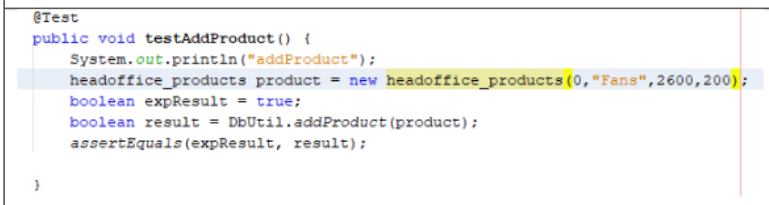
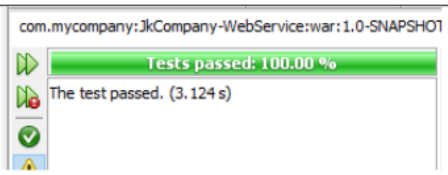
Method After	Results After

`@Test`
`public void testDeleteBranch() {`
`  System.out.println("deleteBranch");`
`  int id = 1;`
`  boolean expResult = true;`
`  boolean result = DbUtil.deleteBranch(id);`
`  assertEquals(expResult, result);`
`}`
	

Checking the addProduct() method

Method Before	Results Before
	

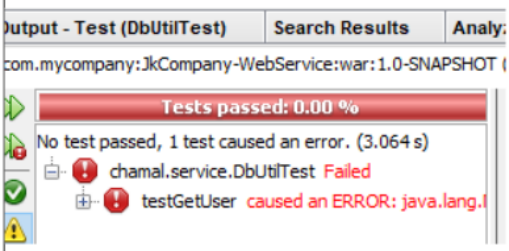
```
@Test
public void testAddProduct() {
    System.out.println("addProduct");
    headoffice_products product = null;
    boolean expResult = false;
    boolean result = DbUtil.addProduct(product);
    assertEquals(expResult, result);
    // TODO review the generated test code and remov
    fail("The test case is a prototype.");
}
```


Method After	Results After
	

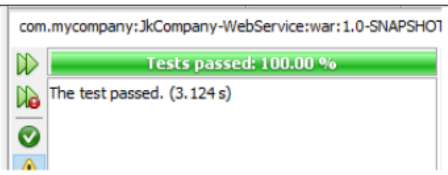
```
@Test
public void testAddProduct() {
    System.out.println("addProduct");
    headoffice_products product = new headoffice_products(0,"Fans",2600,200);
    boolean expResult = true;
    boolean result = DbUtil.addProduct(product);
    assertEquals(expResult, result);

}
```

Checking the addProduct() method

Method Before	Results Before

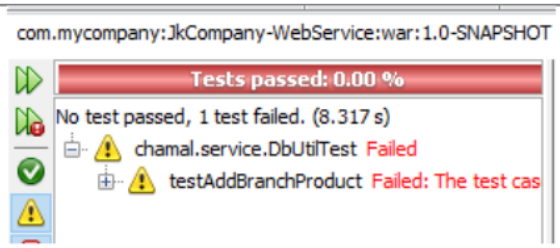
`@Test`
`public void testAddProduct() {`
`  System.out.println("addProduct");`
`  headoffice_products product = null;`
`  boolean expResult = false;`
`  boolean result = DbUtil.addProduct(product);`
`  assertEquals(expResult, result);`
`  // TODO review the generated test code and remov`
`  fail("The test case is a prototype.");`
`}`
	

Method After	Results After

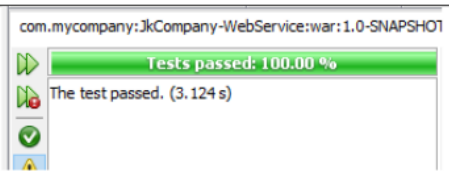
`@Test`
`public void testAddProduct() {`
`  System.out.println("addProduct");`
`  headoffice_products product = new headoffice_products(0,"Fans",2600,200);`
`  boolean expResult = true;`
`  boolean result = DbUtil.addProduct(product);`
`  assertEquals(expResult, result);`
 	
`}`
	

Checking the addBranchProducts() method

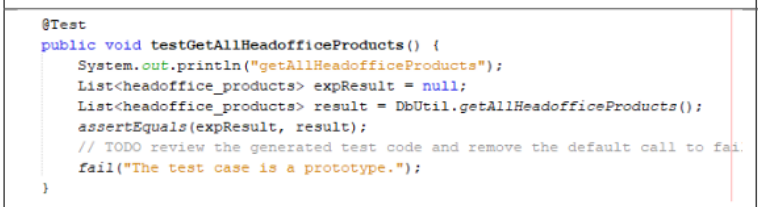
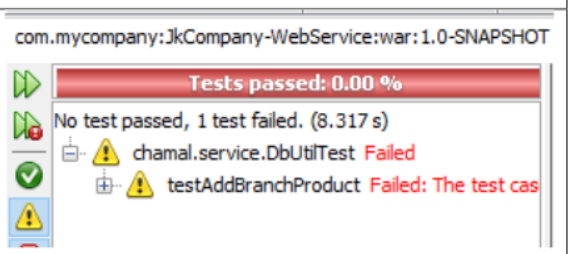
Method Before	Results Before

`@Test`
`public void testAddBranchProduct() {`
`System.out.println("addBranchProduct");`
`branch_products products = null;`
`boolean expResult = false;`
`boolean result = DbUtil.addBranchProduct(products);`
`assertEquals(expResult, result);`
`// TODO review the generated test code and remove the def`
`fail("The test case is a prototype.");`
`}`
	

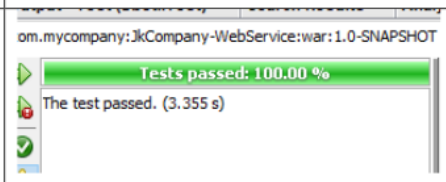
Method After	Results After

`@Test`
`public void testAddBranchProduct() {`
`System.out.println("addBranchProduct");`
`branch_products products = new branch_products(0,"Fans",200,20,3,2);`
`boolean expResult = true;`
`boolean result = DbUtil.addBranchProduct(products);`
`assertEquals(expResult, result);`
`}`
	

Checking the getALLHeadofficeProducts() method

Method Before	Results Before
	

```
@Test
public void testGetAllHeadofficeProducts() {
    System.out.println("getAllHeadofficeProducts");
    List<headoffice_products> expResult = null;
    List<headoffice_products> result = DbUtil.getAllHeadofficeProducts();
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fai
    fail("The test case is a prototype.");
}
```

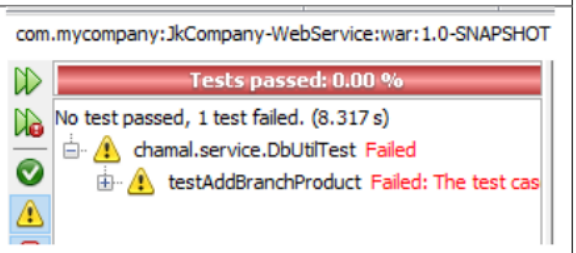

Method After	Results After
	

```
@Test
public void testGetAllHeadofficeProducts() {
    System.out.println("getAllHeadofficeProducts");
    List<headoffice_products> expResult = new ArrayList<>();
    headoffice_products obj= new headoffice_products (1,"Electric Kettles",2500,600);
    headoffice_products obj2= new headoffice_products (3,"Table Fans",1200,470);
    expResult.add(obj);
    expResult.add(obj2);
    List<headoffice_products> result = DbUtil.getAllHeadofficeProducts();
    assertEquals(expResult.get(0).getHeadofficeproduct_name(), result.get(0).getHeadofficeproduct_

}
```

Checking the Getsingleheadofficeproduct() method

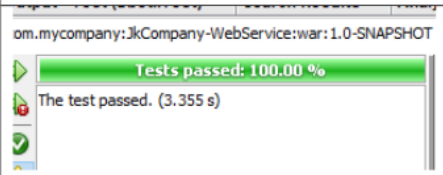
Method Before	Results Before

```
@Test
public void testGetsingleheadofficeproduct() {
    System.out.println("getsingleheadofficeproduct");
    int productID = 0;
    headoffice_products expResult = null;
    headoffice_products result = DbUtil.getsingleheadofficeproduct(productID);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}
```

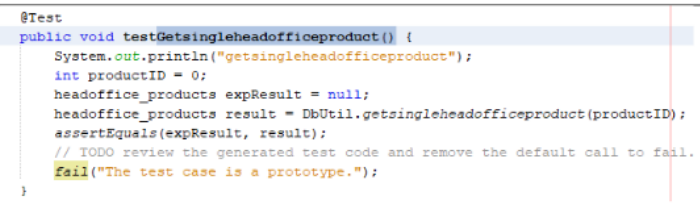
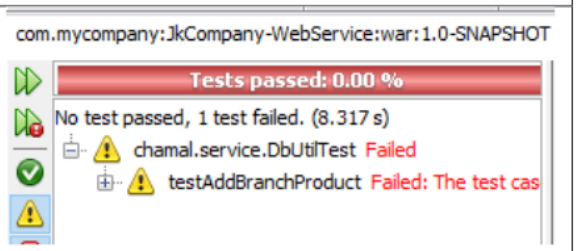


Method After	Results After

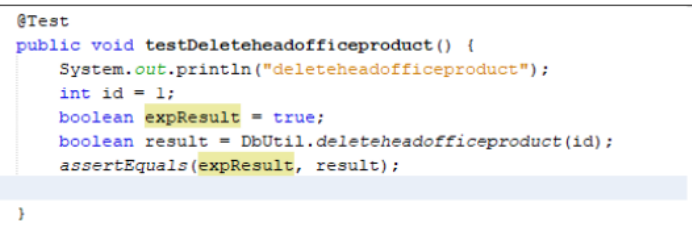
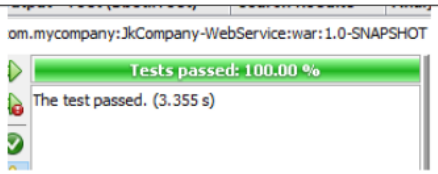
```
@Test
public void testGetsingleheadofficeproduct() {
    System.out.println("getsingleheadofficeproduct");
    int productID = 1;
    headoffice_products expResult = new headoffice_products (1,"Electric Kettles",2500,600);
    headoffice_products result = DbUtil.getsingleheadofficeproduct(productID);
    assertEquals(expResult, result);

}
```



Checking the Deleteheadofficeproduct() method

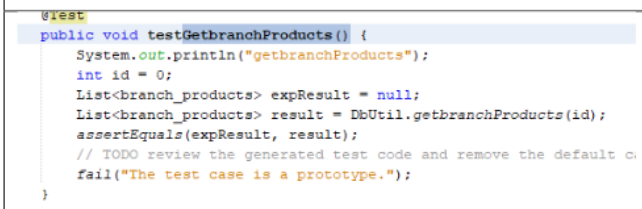
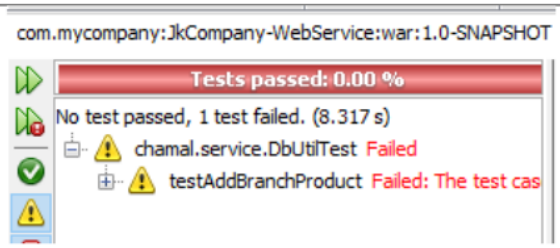
Method Before	Results Before
	

```
@Test
public void testGetsingleheadofficeproduct() {
    System.out.println("getsingleheadofficeproduct");
    int productID = 0;
    headoffice_products expResult = null;
    headoffice_products result = DbUtil.getsingleheadofficeproduct(productID);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}
```

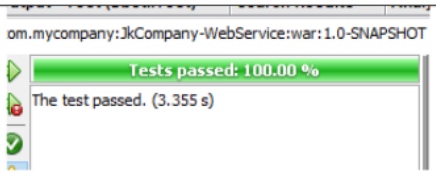

Method After	Results After
	

```
@Test
public void testDeleteheadofficeproduct() {
    System.out.println("deleteheadofficeproduct");
    int id = 1;
    boolean expResult = true;
    boolean result = DbUtil.deleteheadofficeproduct(id);
    assertEquals(expResult, result);

}
```

Checking the GetbranchProducts() method

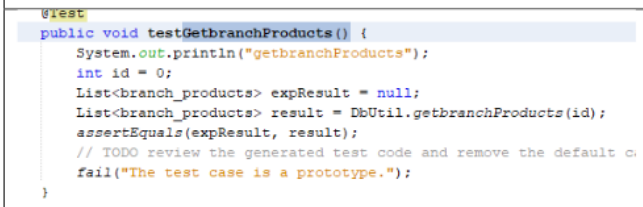
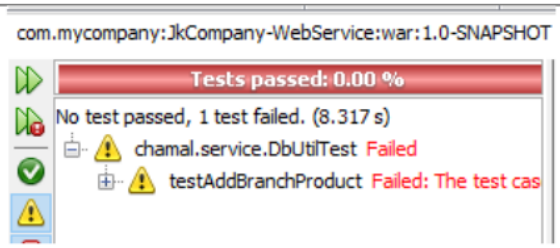
Method Before	Results Before
	

```
@Test
public void testGetbranchProducts() {
    System.out.println("getbranchProducts");
    int id = 0;
    List<branch_products> expResult = null;
    List<branch_products> result = DbUtil.getbranchProducts(id);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default c
    fail("The test case is a prototype.");
}
```

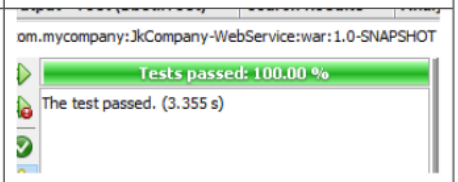

Method After	Results After
	

```
@Test
public void testGetbranchProducts() {
    System.out.println("getbranchProducts");
    int id = 1;
    List<branch_products> expResult = new ArrayList<>();
    branch_products obj = new branch_products(1,"Kettles",2500,20,1,2);
    expResult.add(obj);
    List<branch_products> result = DbUtil.getbranchProducts(id);
    assertEquals(expResult.get(0).getProductName(), result.get(0).getProductName());

}
```

Checking the ValidatebranchProducts() method

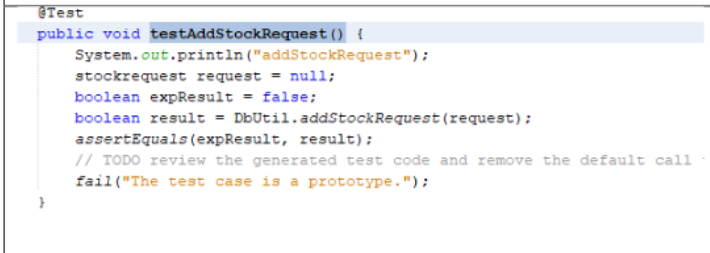
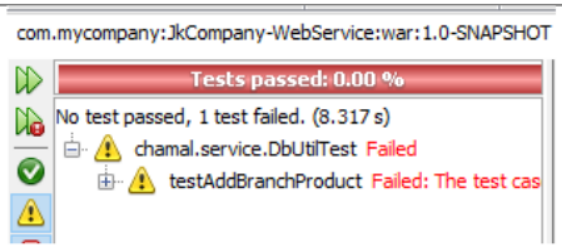
Method Before	Results Before
	

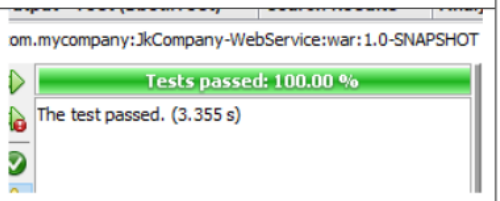
```
@Test
public void testGetbranchProducts() {
    System.out.println("getbranchProducts");
    int id = 0;
    List<branch_products> expResult = null;
    List<branch_products> result = DbUtil.getbranchProducts(id);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default c
    fail("The test case is a prototype.");
}
```


Method After	Results After
	

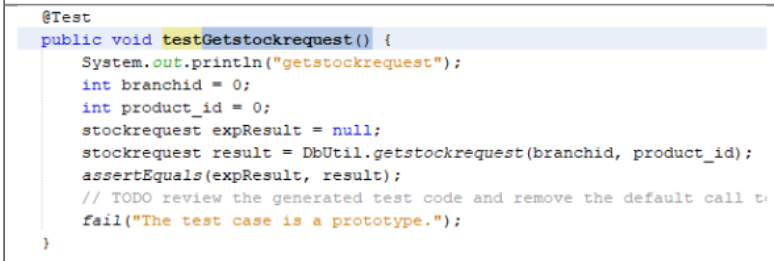
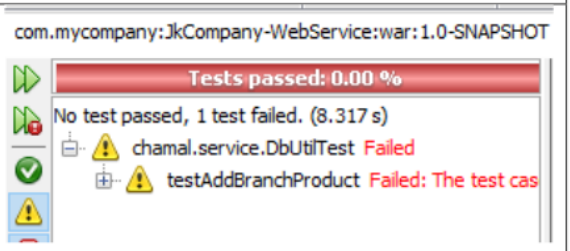
```
@Test
public void testValidatebranchProducts() {
    System.out.println("validatebranchProducts");
    int branchid = 1;
    int product_id = 1;
    branch_products expResult =new  branch_products(1,"Electric Kettles",2500,82,1,2);
    branch_products result = DbUtil.validatebranchProducts(branchid, product_id);
    assertEquals(expResult, result);

}
```

Checking the test AddStockRequest() method

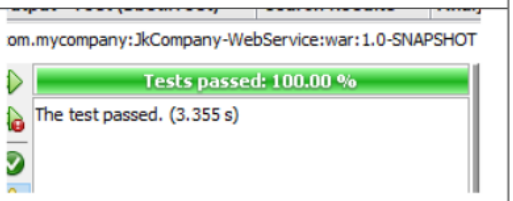
Method Before	Results Before
	

Method After	Results After
	

Checking the test Getstockrequest() method

Method Before	Results Before
	

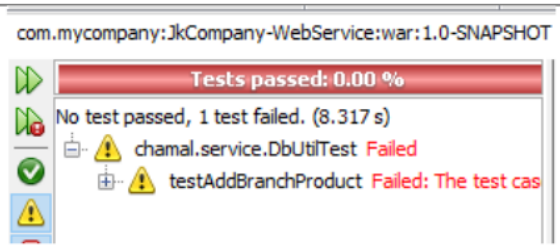
```
@Test
public void testGetstockrequest() {
    System.out.println("getstockrequest");
    int branchid = 0;
    int product_id = 0;
    stockrequest expResult = null;
    stockrequest result = DbUtil.getstockrequest(branchid, product_id);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call t
    fail("The test case is a prototype.");
}
```


Method After	Results After
	

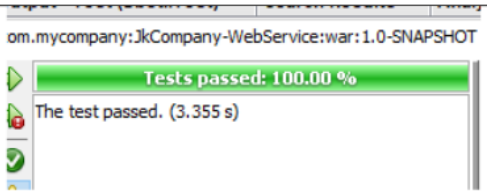
```
@Test
public void testGetstockrequest() {
    System.out.println("getstockrequest");
    int branchid = 2;
    int product_id = 3;
    stockrequest expResult = new stockrequest(0,"Feb 9, 2021 6:26:34 PM","Goods Del
    stockrequest result = DbUtil.getstockrequest(branchid, product_id);
    assertEquals(expResult, result);

}
```

Checking the test GetStockSendingBranchID()method

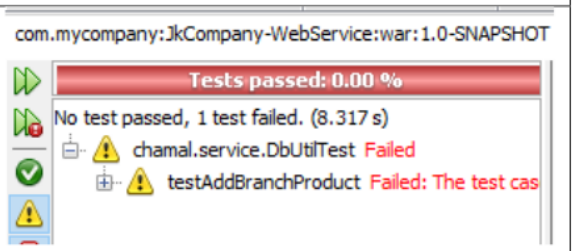
Method Before	Results Before

`@Test`
`public void testGetStockSendingBranchID() {`
`System.out.println("getStockSendingBranchID");`
`String Name = "";`
`int expResult = 0;`
`int result = DbUtil.getStockSendingBranchID(Name);`
`assertEquals(expResult, result);`
`}`
	

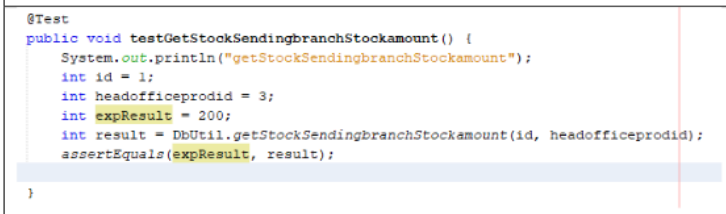
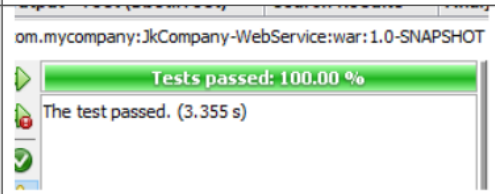
Method After	Results After

`@Test`
`public void testGetStockSendingBranchID() {`
`System.out.println("getStockSendingBranchID");`
`String Name = "Colombo";`
`int expResult = 1;`
`int result = DbUtil.getStockSendingBranchID(Name);`
`assertEquals(expResult, result);`
`}`
	

Checking the test GetStockSendingbranchStockamount() method

Method Before	Results Before
	

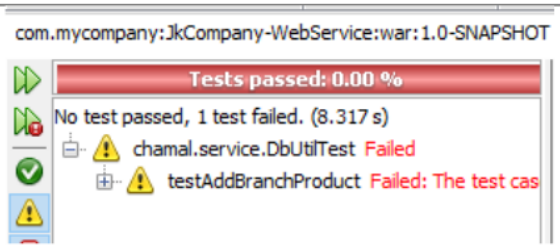
```
@Test
public void testGetStockSendingbranchStockamount() {
    System.out.println("getStockSendingbranchStockamount");
    int id = 0;
    int headofficeprodid = 0;
    int expResult = 0;
    int result = DbUtil.getStockSendingbranchStockamount(id, headofficeprodid)
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail
    fail("The test case is a prototype.");
}
```


Method After	Results After
	

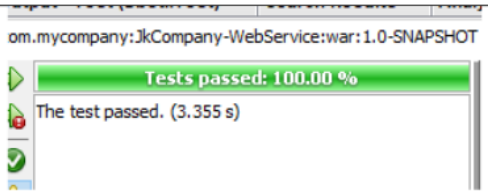
```
@Test
public void testGetStockSendingbranchStockamount() {
    System.out.println("getStockSendingbranchStockamount");
    int id = 1;
    int headofficeprodid = 3;
    int expResult = 200;
    int result = DbUtil.getStockSendingbranchStockamount(id, headofficeprodid);
    assertEquals(expResult, result);

}
```

Checking the test GetStockrecievingbranchAmount() method

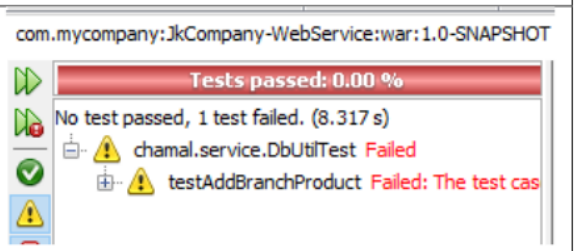
Method Before	Results Before

`@Test`
`public void testGetStockrecievingbranchAmount() {`
`System.out.println("getStockrecievingbranchAmount");`
`int id = 0;`
`int expResult = 0;`
`int result = DbUtil.getStockrecievingbranchAmount(id);`
`assertEquals(expResult, result);`
`}`
	

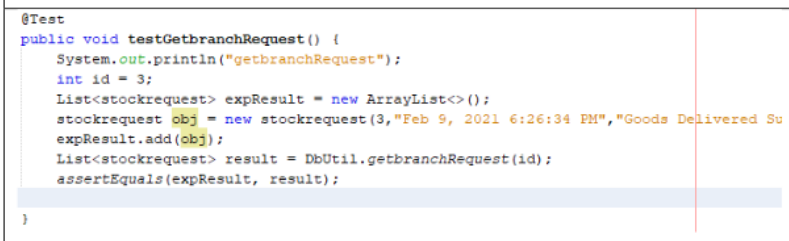
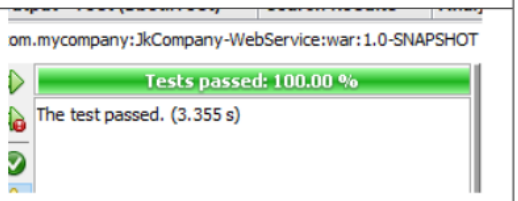
Method After	Results After

`@Test`
`public void testGetStockrecievingbranchAmount() {`
`System.out.println("getStockrecievingbranchAmount");`
`int id = 2;`
`int expResult = 200;`
`int result = DbUtil.getStockrecievingbranchAmount(id);`
`assertEquals(expResult, result);`
`}`
	

Checking the GetbranchRequest() method

Method Before	Results Before
	

```
@Test
public void testGetbranchRequest() {
    System.out.println("getbranchRequest");
    int id = 0;
    List<stockrequest> expResult = null;
    List<stockrequest> result = DbUtil.getbranchRequest(id);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to f
    fail("The test case is a prototype.");
}
```

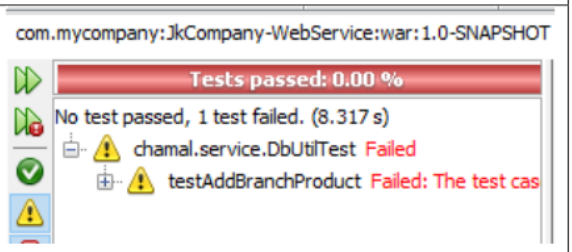

Method After	Results After
	

```
@Test
public void testGetbranchRequest() {
    System.out.println("getbranchRequest");
    int id = 3;
    List<stockrequest> expResult = new ArrayList<>();
    stockrequest obj = new stockrequest(3,"Feb 9, 2021 6:26:34 PM","Goods Delivered Su
    expResult.add(obj);
    List<stockrequest> result = DbUtil.getbranchRequest(id);
    assertEquals(expResult, result);

}
```

Checking the UpdateStcockRequets() method

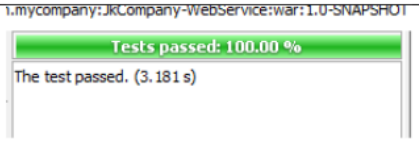
Method Before	Results Before

```
@Test
public void testUpdateStcockRequets() {
    System.out.println("updateStcockRequets");
    int requestID = 0;
    String accept = "";
    boolean expResult = false;
    boolean result = DbUtil.updateStcockRequets(requestID, accept);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call
    fail("The test case is a prototype.");
}
```

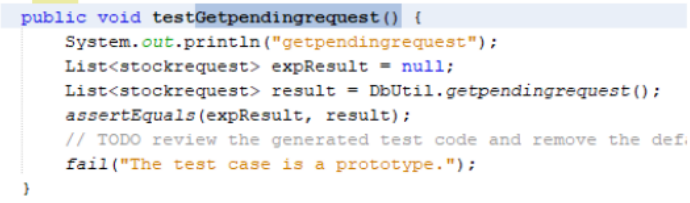
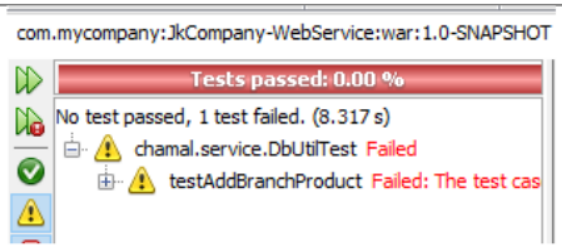


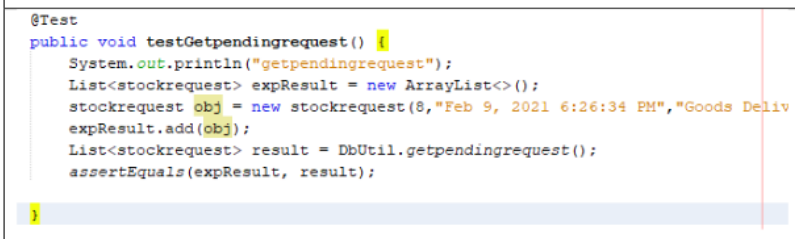
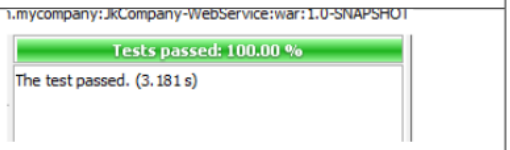
Method After	Results After

```
@Test
public void testUpdateStcockRequets() {
    System.out.println("updateStcockRequets");
    int requestID = 3;
    String accept = "Request still not accepted";
    boolean expResult = true;
    boolean result = DbUtil.updateStcockRequets(requestID, accept);
    assertEquals(expResult, result);

}
```

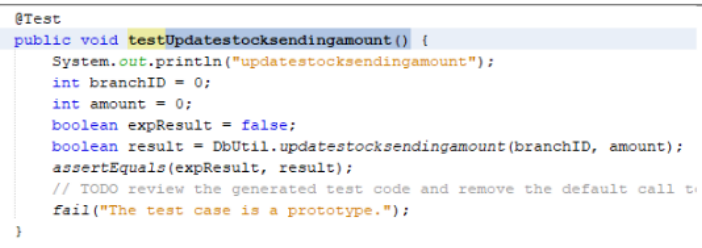
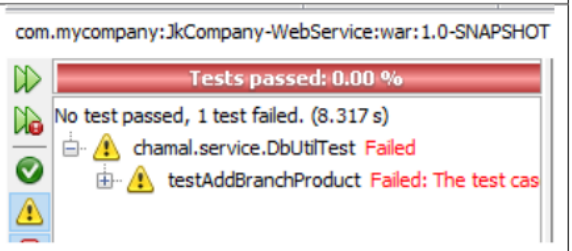


Checking the Getpendingrequest() method

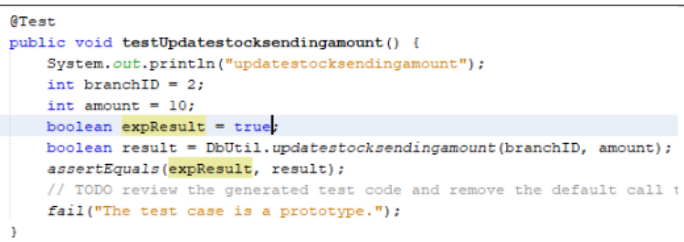
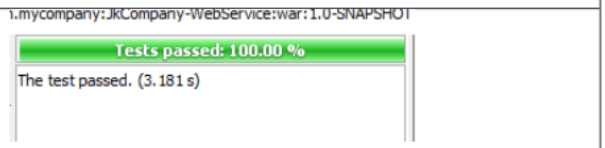
Method Before	Results Before
	

Method After	Results After
	

Checking the Updatestocksendingamount() method

Method Before	Results Before
	

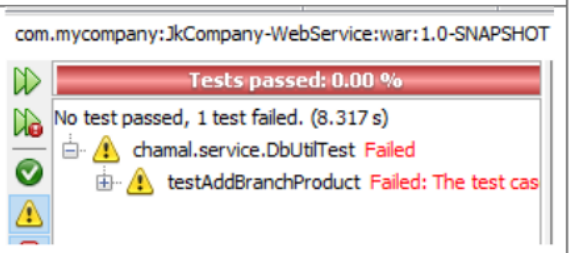
```
@Test
public void testUpdatestocksendingamount() {
    System.out.println("updatestocksendingamount");
    int branchID = 0;
    int amount = 0;
    boolean expResult = false;
    boolean result = DbUtil.updatestocksendingamount(branchID, amount);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call t
    fail("The test case is a prototype.");
}
```


Method After	Results After
	

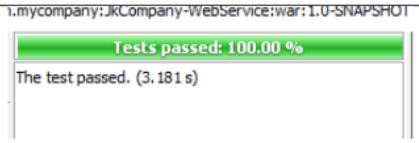
```
@Test
public void testUpdatestocksendingamount() {
    System.out.println("updatestocksendingamount");
    int branchID = 2;
    int amount = 10;
    boolean expResult = true;
    boolean result = DbUtil.updatestocksendingamount(branchID, amount);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call t
    fail("The test case is a prototype.");
}
```

Checking the HeadofficestockAmount() method

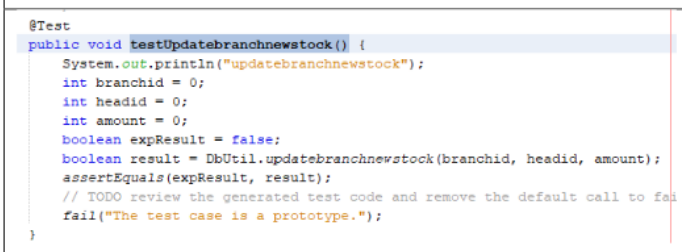
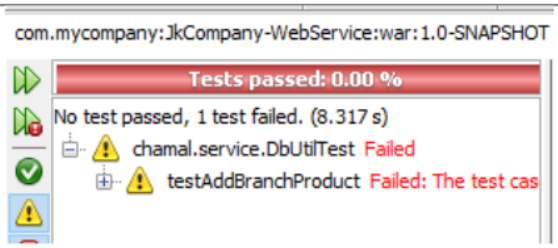
Method Before	Results Before

`@Test`
`public void testHeadofficestockAmount() {`
`  System.out.println("headofficestockAmount");`
`  int productID = 0;`
`  int amount = 0;`
`  boolean expResult = false;`
`  boolean result = DbUtil.headofficestockAmount(productID, amount);`
`  assertEquals(expResult, result);`
`  // TODO review the generated test code and remove the default call to`
`  fail("The test case is a prototype.");`
`}`
	

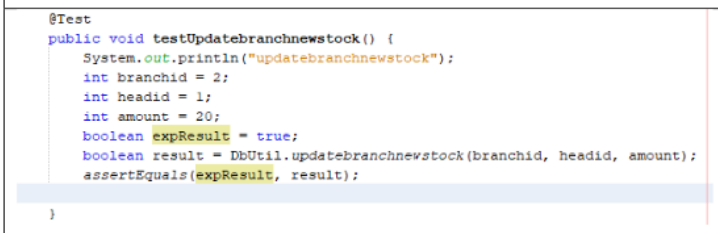
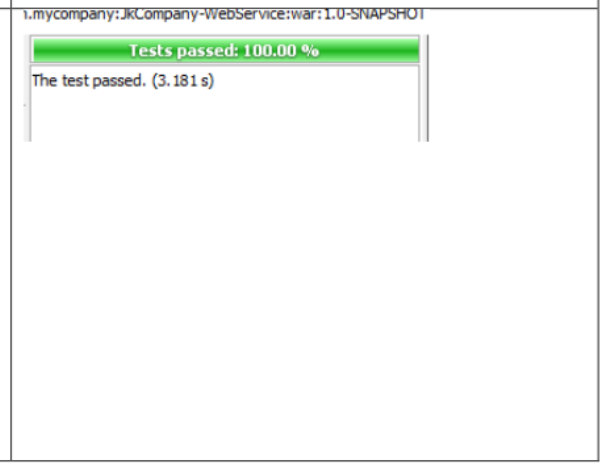
Method After	Results After

`@Test`
`public void testHeadofficestockAmount() {`
`  System.out.println("headofficestockAmount");`
`  int productID = 1;`
`  int amount =320;`
`  boolean expResult = false;`
`  boolean result = DbUtil.headofficestockAmount(productID, amount);`
`  assertEquals(expResult, result);`
`}`
	

Checking the testUpdatebranchnewstock() method

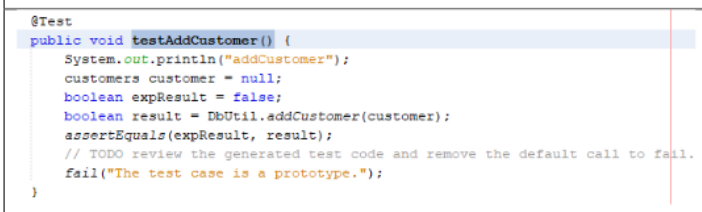
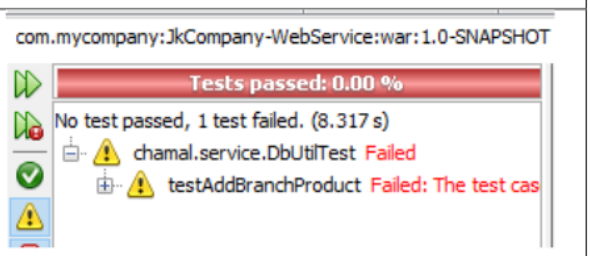
Method Before	Results Before
	

```
@Test
public void testUpdatebranchnewstock() {
    System.out.println("updatebranchnewstock");
    int branchid = 0;
    int headid = 0;
    int amount = 0;
    boolean expResult = false;
    boolean result = DbUtil.updatebranchnewstock(branchid, headid, amount);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fai
    fail("The test case is a prototype.");
}
```

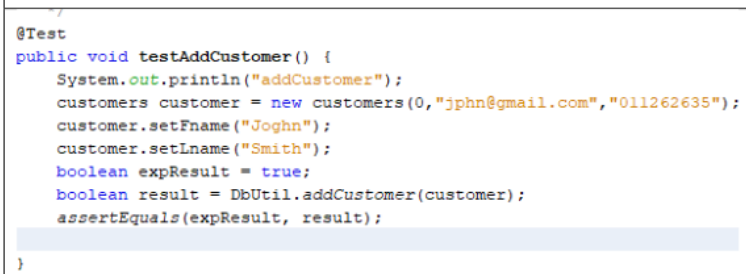

Method After	Results After
	

```
@Test
public void testUpdatebranchnewstock() {
    System.out.println("updatebranchnewstock");
    int branchid = 2;
    int headid = 1;
    int amount = 20;
    boolean expResult = true;
    boolean result = DbUtil.updatebranchnewstock(branchid, headid, amount);
    assertEquals(expResult, result);

}
```

Checking the AddCustomer() method

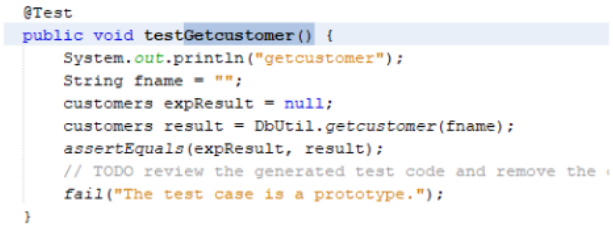
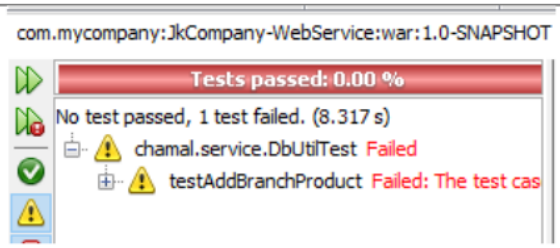
Method Before	Results Before
	

```
@Test
public void testAddCustomer() {
    System.out.println("addCustomer");
    customers customer = null;
    boolean expResult = false;
    boolean result = DbUtil.addCustomer(customer);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}
```

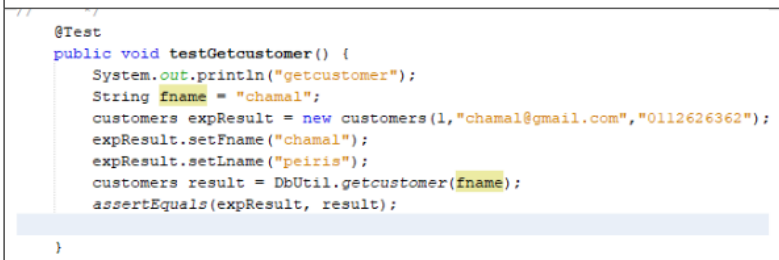
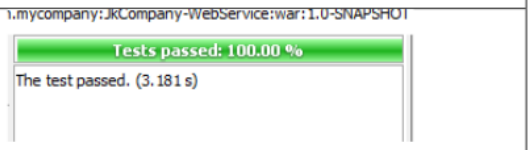

Method After	Results After
	

```
@Test
public void testAddCustomer() {
    System.out.println("addCustomer");
    customers customer = new customers(0,"jphn@gmail.com","011262635");
    customer.setFname("Joghn");
    customer.setLname("Smith");
    boolean expResult = true;
    boolean result = DbUtil.addCustomer(customer);
    assertEquals(expResult, result);

}
```

Checking the Getcustomer() method

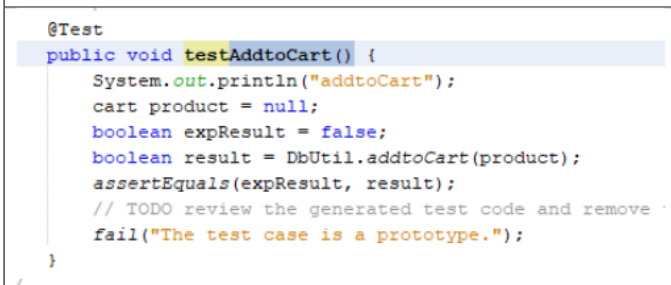
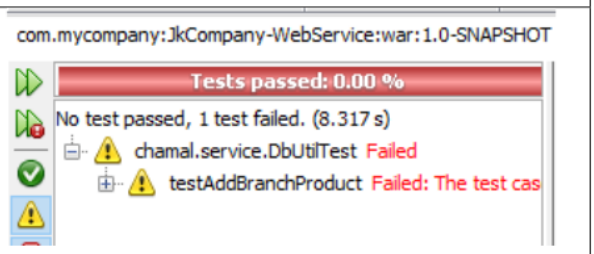
Method Before	Results Before
	

```
@Test
public void testGetcustomer() {
    System.out.println("getcustomer");
    String fname = "";
    customers expResult = null;
    customers result = DbUtil.getcustomer(fname);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the
    fail("The test case is a prototype.");
}
```

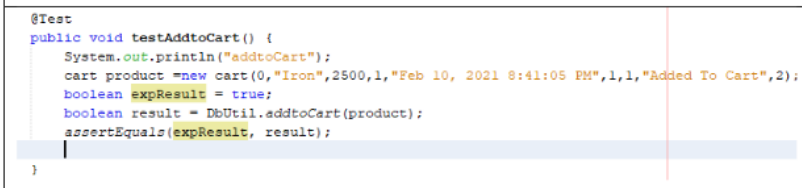
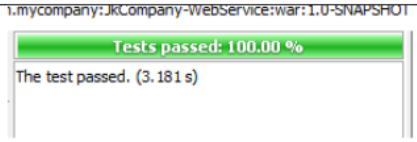

Method After	Results After
	

```
@Test
public void testGetcustomer() {
    System.out.println("getcustomer");
    String fname = "chamal";
    customers expResult = new customers(1,"chamal@gmail.com","0112626362");
    expResult.setFname("chamal");
    expResult.setLname("peiris");
    customers result = DbUtil.getcustomer(fname);
    assertEquals(expResult, result);

}
```

Checking the AddtoCart() method

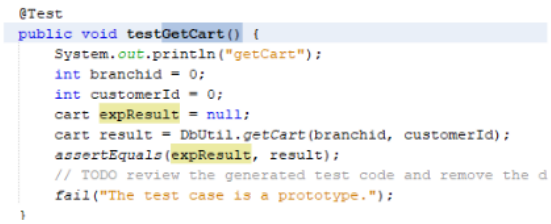
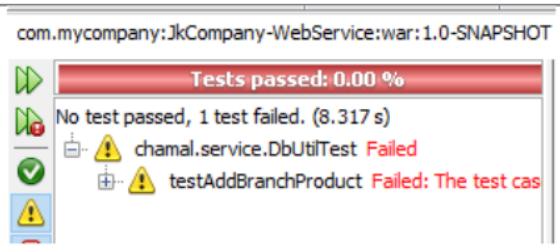
Method Before	Results Before
	

```
@Test
public void testAddtoCart() {
    System.out.println("addtoCart");
    cart product = null;
    boolean expResult = false;
    boolean result = DbUtil.addtoCart(product);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove
    fail("The test case is a prototype.");
}
```

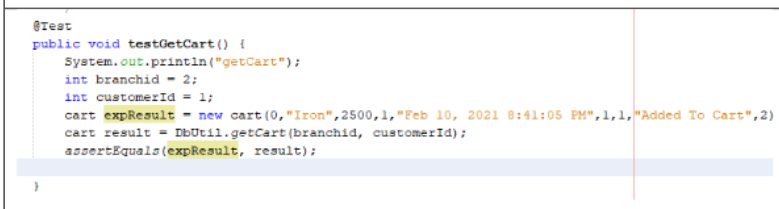
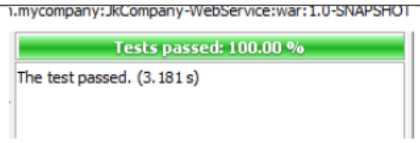

Method After	Results After
	

```
@Test
public void testAddtoCart() {
    System.out.println("addtoCart");
    cart product =new cart(0,"Iron",2500,1,"Feb 10, 2021 8:41:05 PM",1,1,"Added To Cart",2);
    boolean expResult = true;
    boolean result = DbUtil.addtoCart(product);
    assertEquals(expResult, result);

}
```

Checking the GetCart() method

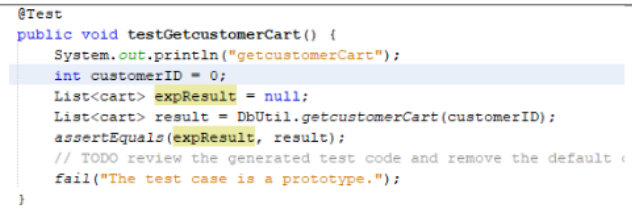
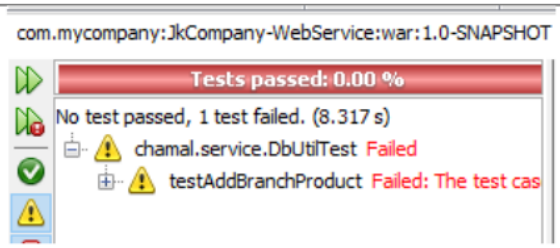
Method Before	Results Before
	

```
@Test
public void testGetCart() {
    System.out.println("getCart");
    int branchid = 0;
    int customerId = 0;
    cart expResult = null;
    cart result = DbUtil.getCart(branchid, customerId);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the d
    fail("The test case is a prototype.");
}
```

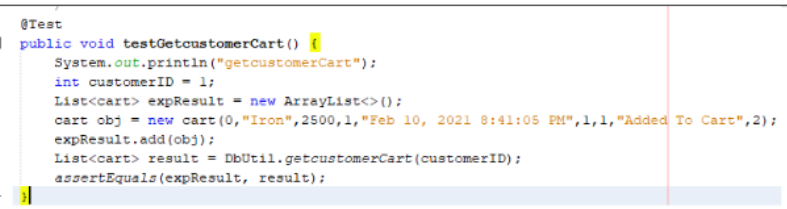
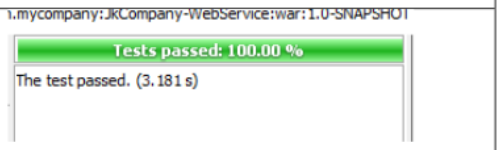

Method After	Results After
	

```
@Test
public void testGetCart() {
    System.out.println("getCart");
    int branchid = 2;
    int customerId = 1;
    cart expResult = new cart(0,"Iron",2500,1,"Feb 10, 2021 8:41:05 PM",1,1,"Added To Cart",2)
    cart result = DbUtil.getCart(branchid, customerId);
    assertEquals(expResult, result);

}
```

Checking the GetcustomerCart()method

Method Before	Results Before
	

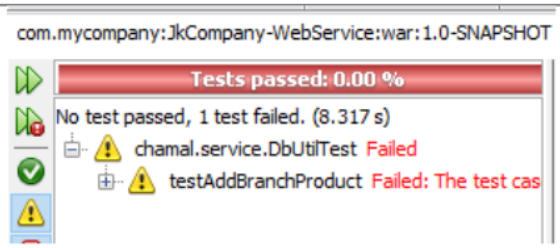
```
@Test
public void testGetcustomerCart() {
    System.out.println("getcustomerCart");
    int customerID = 0;
    List<cart> expResult = null;
    List<cart> result = DbUtil.getcustomerCart(customerID);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default
    fail("The test case is a prototype.");
}
```


Method After	Results After
	

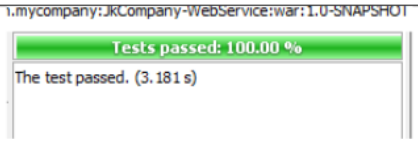
```
@Test
public void testGetcustomerCart() {
    System.out.println("getcustomerCart");
    int customerID = 1;
    List<cart> expResult = new ArrayList<>();
    cart obj = new cart(0,"Iron",2500,1,"Feb 10, 2021 8:41:05 PM",1,1,"Added To Cart",2);
    expResult.add(obj);
    List<cart> result = DbUtil.getcustomerCart(customerID);
    assertEquals(expResult, result);
}
```

Checking the UpdateCart() method

Method Before	Results Before

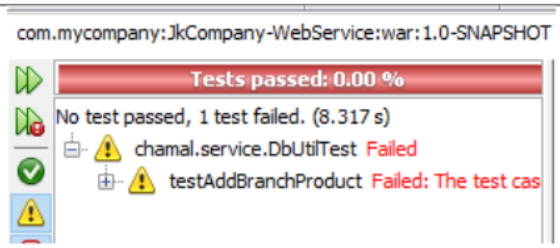
@Test
public void testUpdateCart() {
System.out.println("updateCart");
int custid = 0;
boolean expResult = false;
boolean result = DbUtil.updateCart(custid);
assertEquals(expResult, result);
// TODO review the generated test code and remove th
fail("The test case is a prototype.");
}
	

Method After	Results After

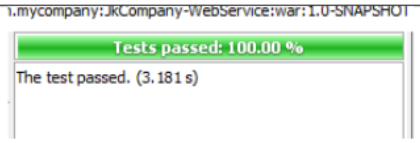
@Test
public void testUpdateCart() {
System.out.println("updateCart");
int custid = 1;
boolean expResult = true;
boolean result = DbUtil.updateCart(custid);
assertEquals(expResult, result);
 	
}
	

Checking the GetBranchsales() method

Method Before	Results Before

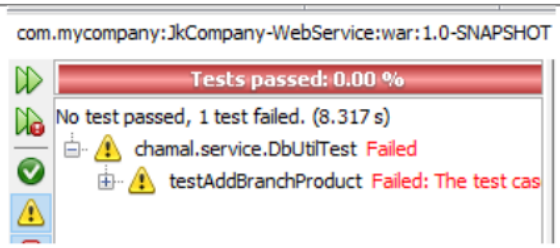
`public void testGetBranchsales() {`
`System.out.println("getBranchsales");`
`int branchID = 0;`
`List<cart> expResult = null;`
`List<cart> result = DbUtil.getBranchsales(branchID);`
`assertEquals(expResult, result);`
`// TODO review the generated test code and remove the defa`
`fail("The test case is a prototype.");`
`}`
	

Method After	Results After

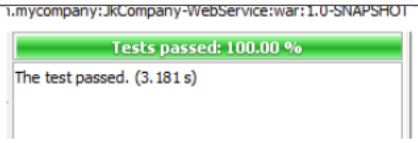
`public void testGetBranchsales() {`
`System.out.println("getBranchsales");`
`int branchID = 2;`
`List<cart> expResult = new ArrayList<>();`
`cart obj= new cart(0,"Iron",2500,1,"Feb 10, 2021 8:41:05 PM",1,1,"Transacti`
`expResult.add(obj);`
`List<cart> result = DbUtil.getBranchsales(branchID);`
`assertEquals(expResult, result);`
`}`
	

Checking the AddDriver() method

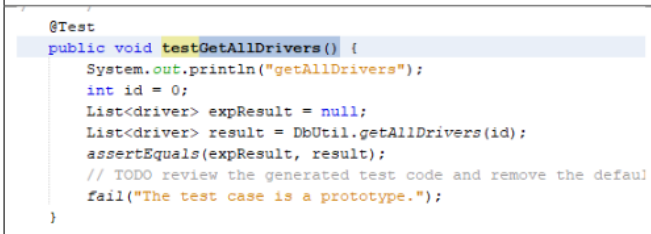
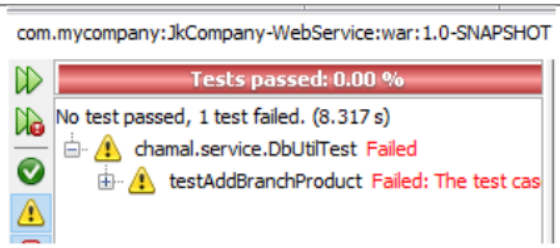
Method Before	Results Before

`@Test`
`public void testAddDriver() {`
`System.out.println("addDriver");`
`driver driver = null;`
`boolean expResult = false;`
`boolean result = DbUtil.addDriver(driver);`
`assertEquals(expResult, result);`
`// TODO review the generated test code and remove the c`
`fail("The test case is a prototype.");`
`}`
	

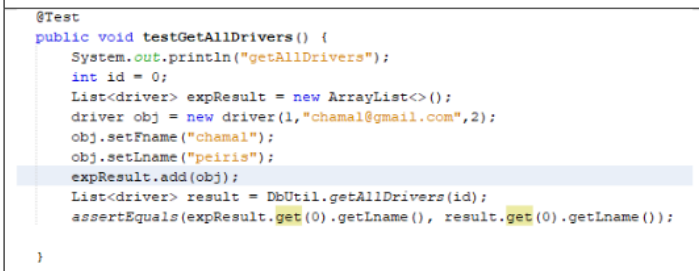
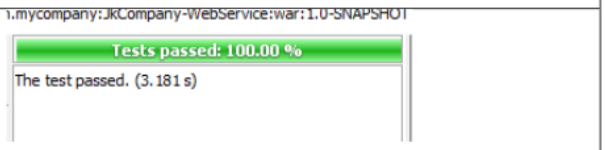
Method After	Results After

`@Test`
`public void testAddDriver() {`
`System.out.println("addDriver");`
`driver driver = new driver(0,"josh@gmail.com",2);`
`driver.setFname("josh");`
`driver.setLname("Perera");`
`boolean expResult = true;`
`boolean result = DbUtil.addDriver(driver);`
`assertEquals(expResult, result);`
`}`
	

Checking the GetAllDrivers() method

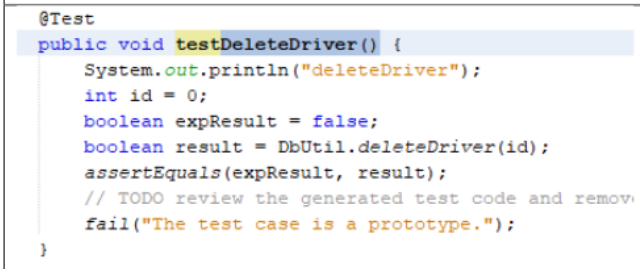
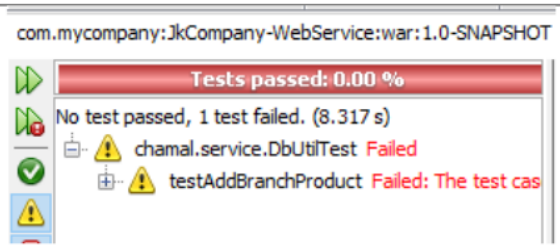
Method Before	Results Before
	

```
@Test
public void testGetAllDrivers() {
    System.out.println("getAllDrivers");
    int id = 0;
    List<driver> expResult = null;
    List<driver> result = DbUtil.getAllDrivers(id);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the defaul
    fail("The test case is a prototype.");
}
```

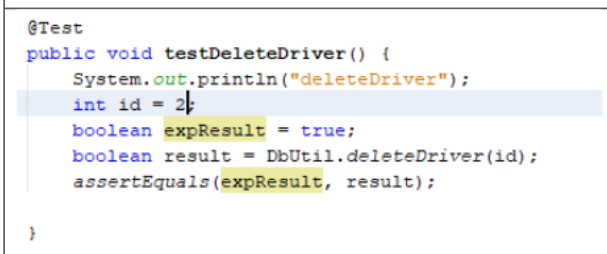
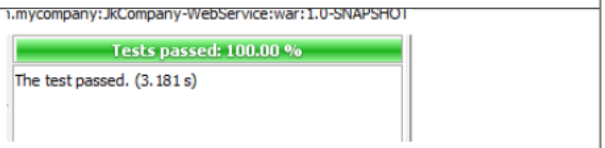

Method After	Results After
	

```
@Test
public void testGetAllDrivers() {
    System.out.println("getAllDrivers");
    int id = 0;
    List<driver> expResult = new ArrayList<>();
    driver obj = new driver(1,"chamal@gmail.com",2);
    obj.setFname("chamal");
    obj.setLname("peiris");
    expResult.add(obj);
    List<driver> result = DbUtil.getAllDrivers(id);
    assertEquals(expResult.get(0).getLname(), result.get(0).getLname());

}
```

Checking the DeleteDriver() method

Method Before	Results Before
	

```
@Test
public void testDeleteDriver() {
    System.out.println("deleteDriver");
    int id = 0;
    boolean expResult = false;
    boolean result = DbUtil.deleteDriver(id);
    assertEquals(expResult, result);
    // TODO review the generated test code and remov
    fail("The test case is a prototype.");
}
```

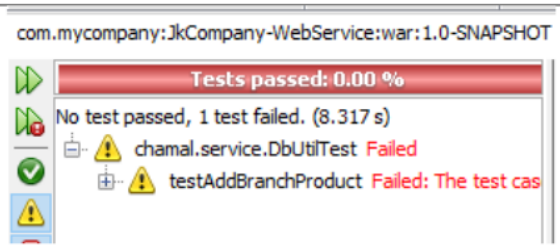

Method After	Results After
	

```
@Test
public void testDeleteDriver() {
    System.out.println("deleteDriver");
    int id = 2;
    boolean expResult = true;
    boolean result = DbUtil.deleteDriver(id);
    assertEquals(expResult, result);

}
```

Checking the Addehicle() method

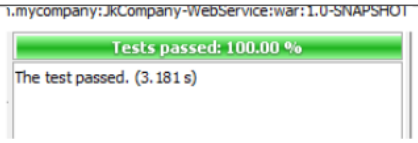
Method Before	Results Before

```
@Test
public void testAddehicle() {
    System.out.println("addehicle");
    vehicle vehicle = null;
    boolean expResult = false;
    boolean result = DbUtil.addehicle(vehicle);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove
    fail("The test case is a prototype.");
}
```



Method After	Results After

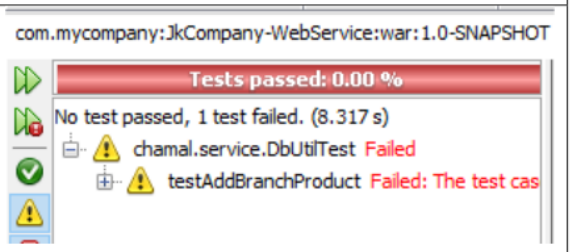
```
@Test
public void testAddehicle() {
    System.out.println("addehicle");
    vehicle vehicle = new vehicle(0,"BHF6391",2);
    boolean expResult = true;
    boolean result = DbUtil.addehicle(vehicle);
    assertEquals(expResult, result);

}
```



Checking the GetAllVehicles() method

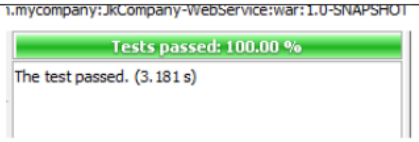
Method Before	Results Before

```
@Test
public void testGetAllVehicles() {
    System.out.println("getAllVehicles");
    int id = 0;
    List<vehicle> expResult = null;
    List<vehicle> result = DbUtil.getAllVehicles(id);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default
    fail("The test case is a prototype.");
}
```



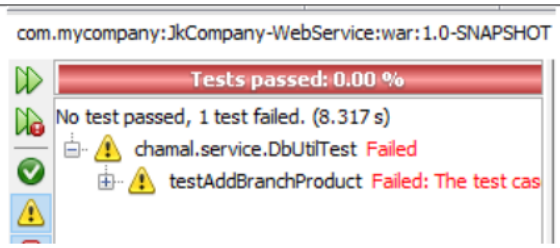
Method After	Results After

```
@Test
public void testGetAllVehicles() {
    System.out.println("getAllVehicles");
    int id = 2;
    List<vehicle> expResult = new ArrayList<>();
    vehicle obj = new vehicle(1,"CAu0304",1);
    expResult.add(obj);
    List<vehicle> result = DbUtil.getAllVehicles(id);
    assertEquals(expResult.get(0).getVehicle_no(), result.get(0).getVehicle_no());

}
```

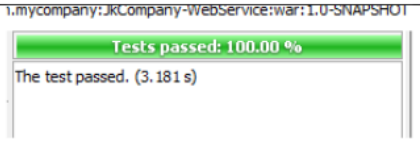


Checking the DeleteVehcile() method

Method Before	Results Before

`@Test`
`public void testDeleteVehcile() {`
`  System.out.println("deleteVehcile");`
`  int id = 0;`
`  boolean expResult = false;`
`  boolean result = DbUtil.deleteVehcile(id);`
`  assertEquals(expResult, result);`
`  // TODO review the generated test code and remove`
`  fail("The test case is a prototype.");`
`}`
	

Method After	Results After

`@Test`
`public void testDeleteVehcile() {`
`  System.out.println("deleteVehcile");`
`  int id = 1;`
`  boolean expResult = true;`
`  boolean result = DbUtil.deleteVehcile(id);`
`  assertEquals(expResult, result);`
 	
`}`
	

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

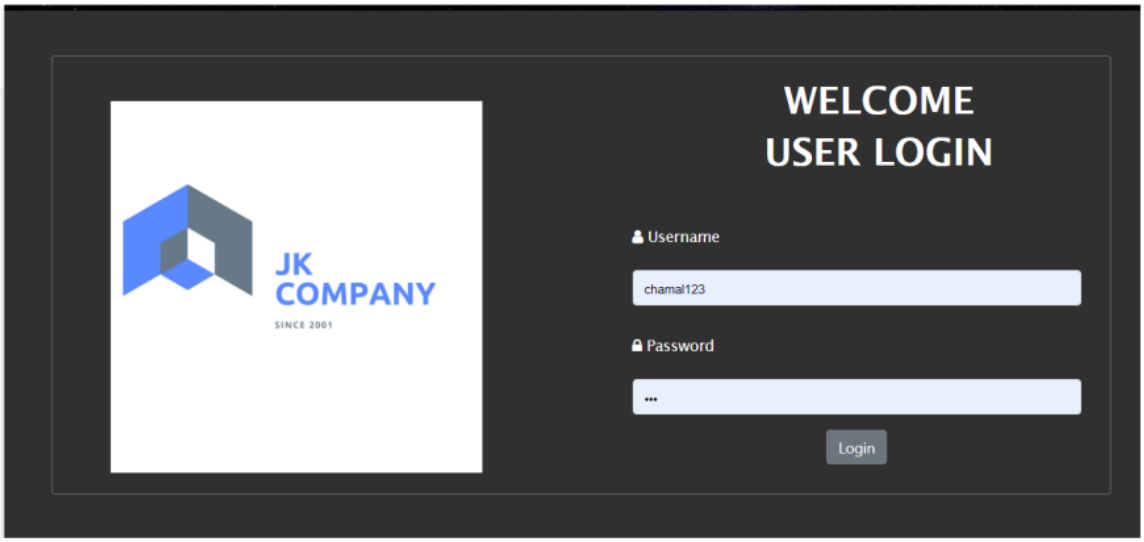


Figure 21-User Login

- 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

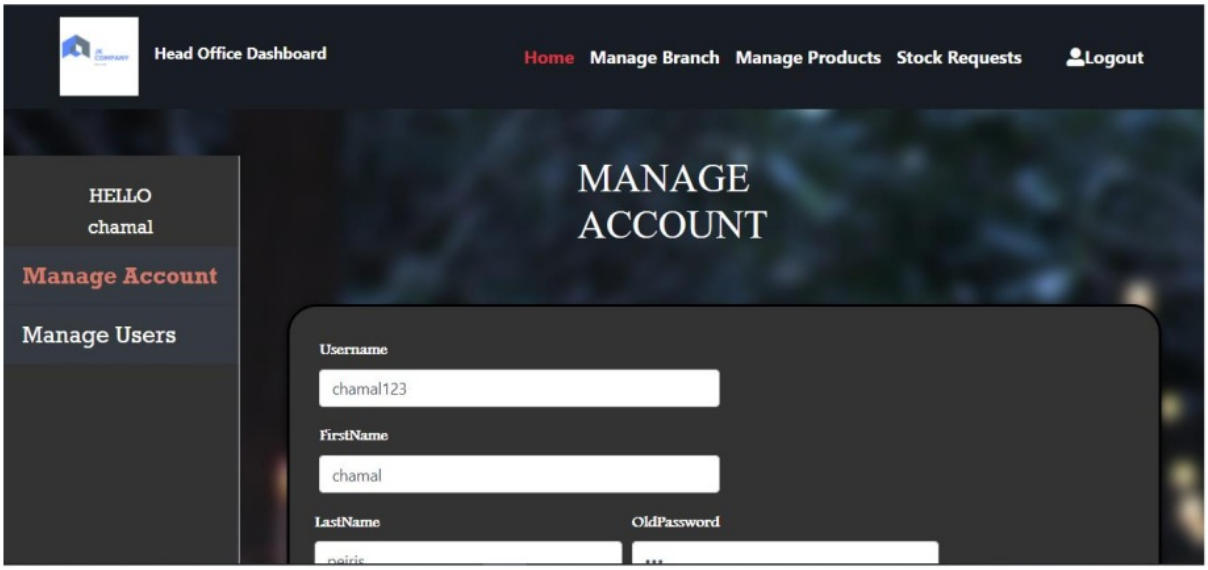


Figure 22-HeadOffice Homepage

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

Manage Account

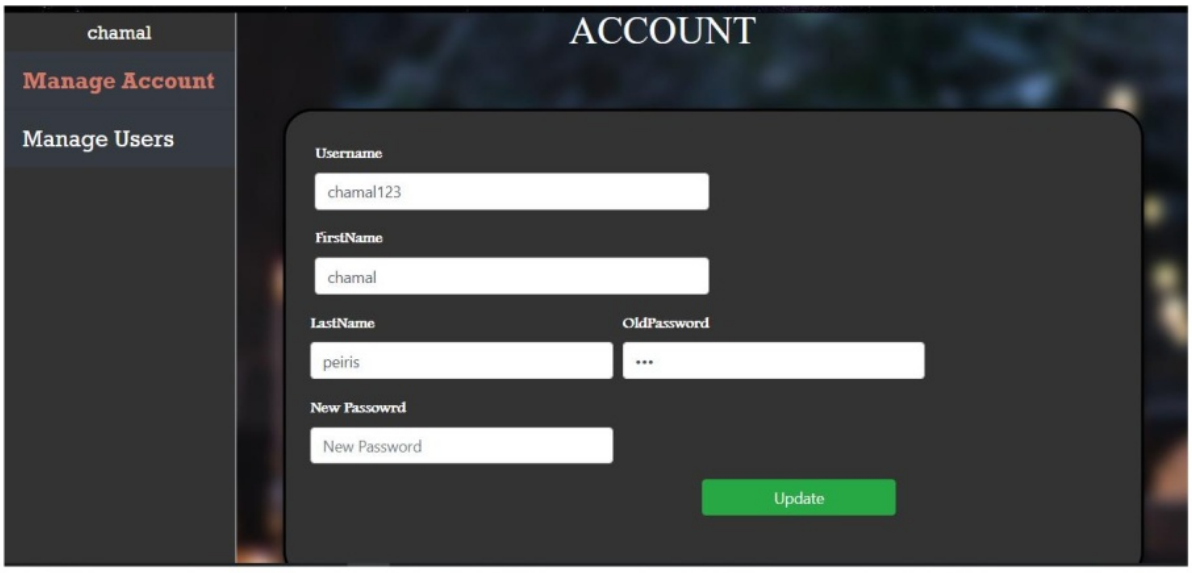


Figure 23 Manage Account

- 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

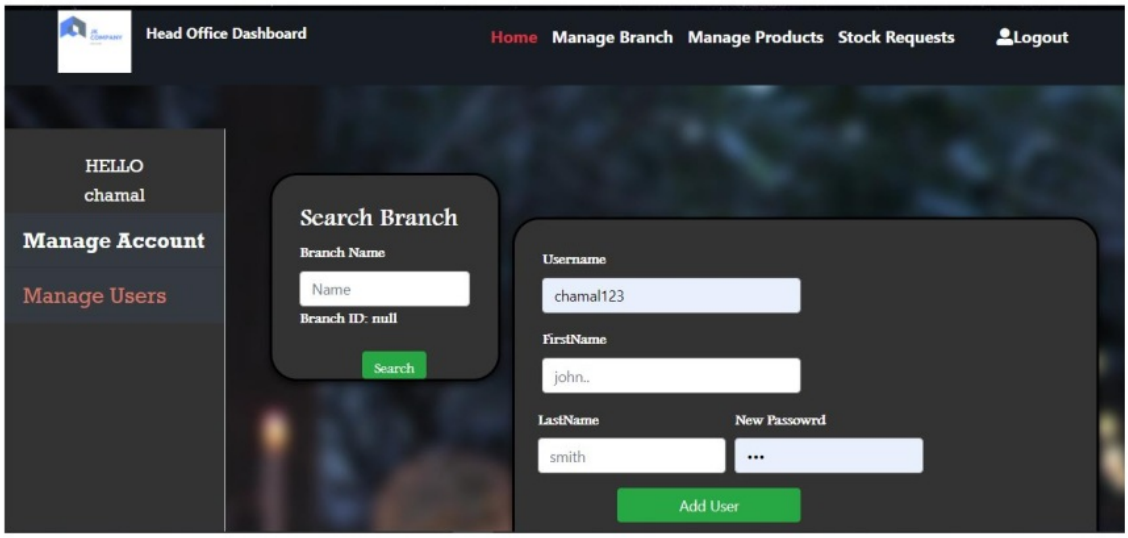


Figure 24 Add User

- 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

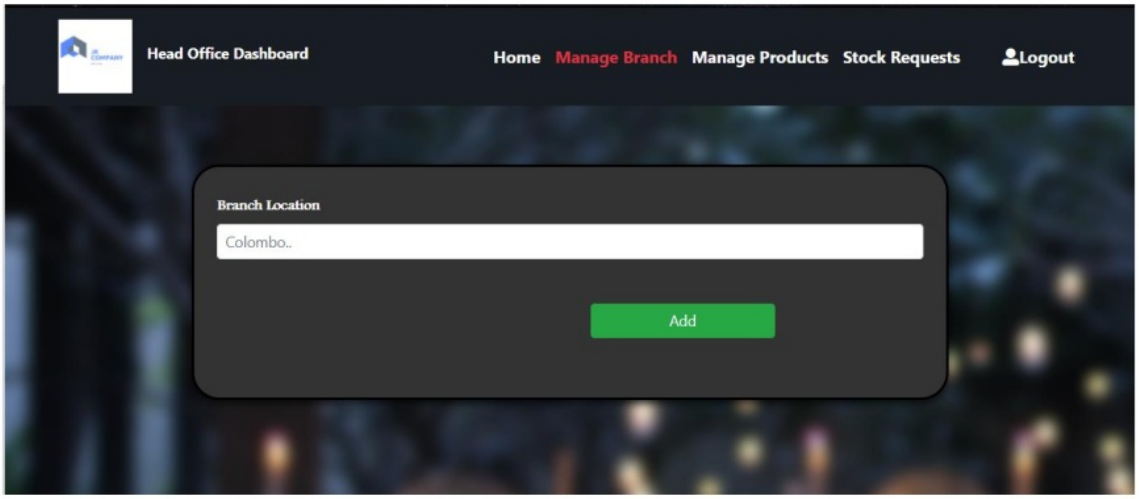


Figure 26 Add Branch

- 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

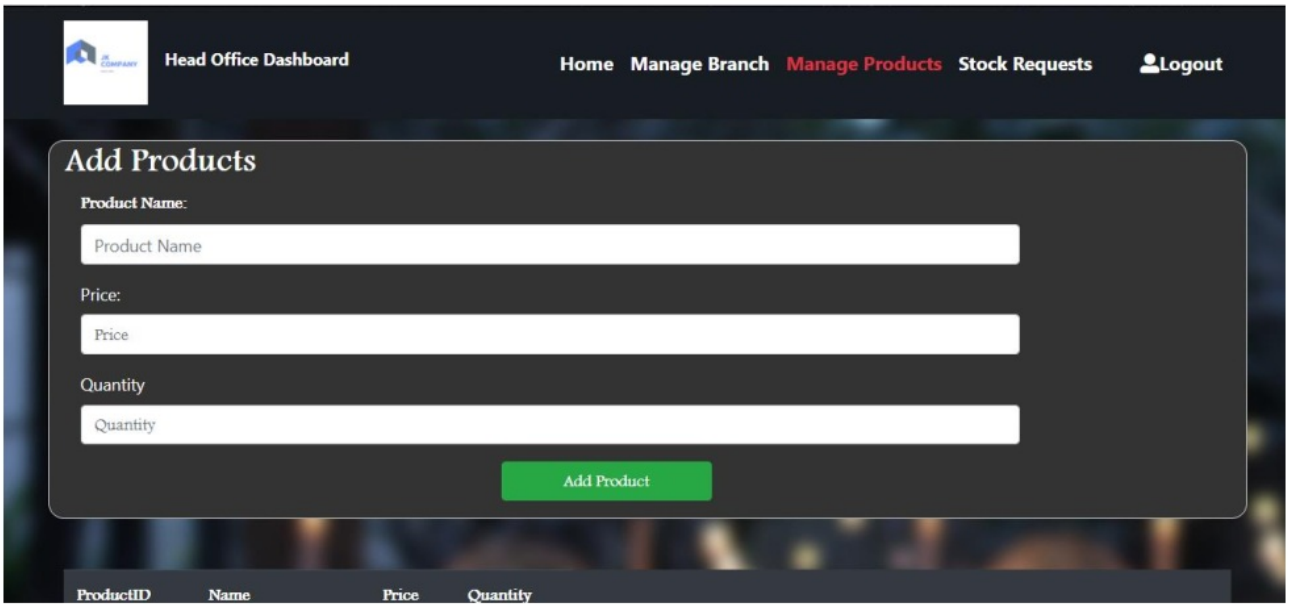


Figure 28 Add Products

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

View All/delete Products

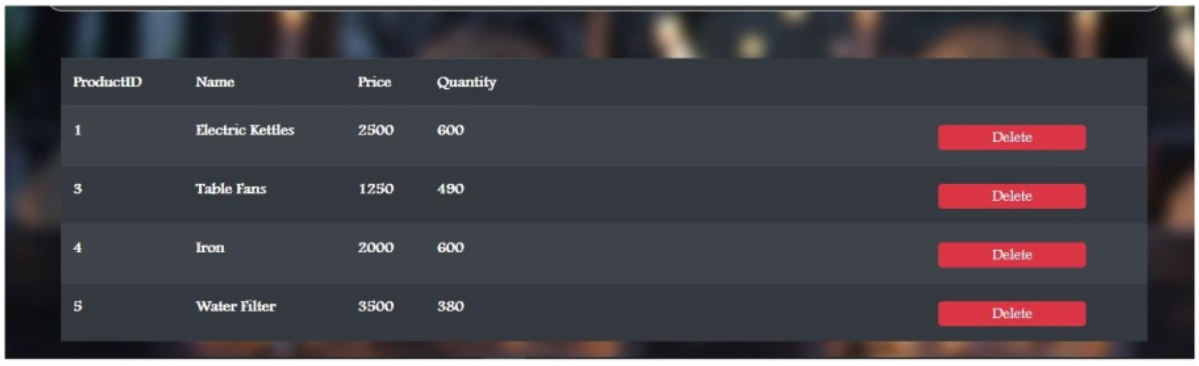


Figure 29 viewall/delete products

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

Stock Requests

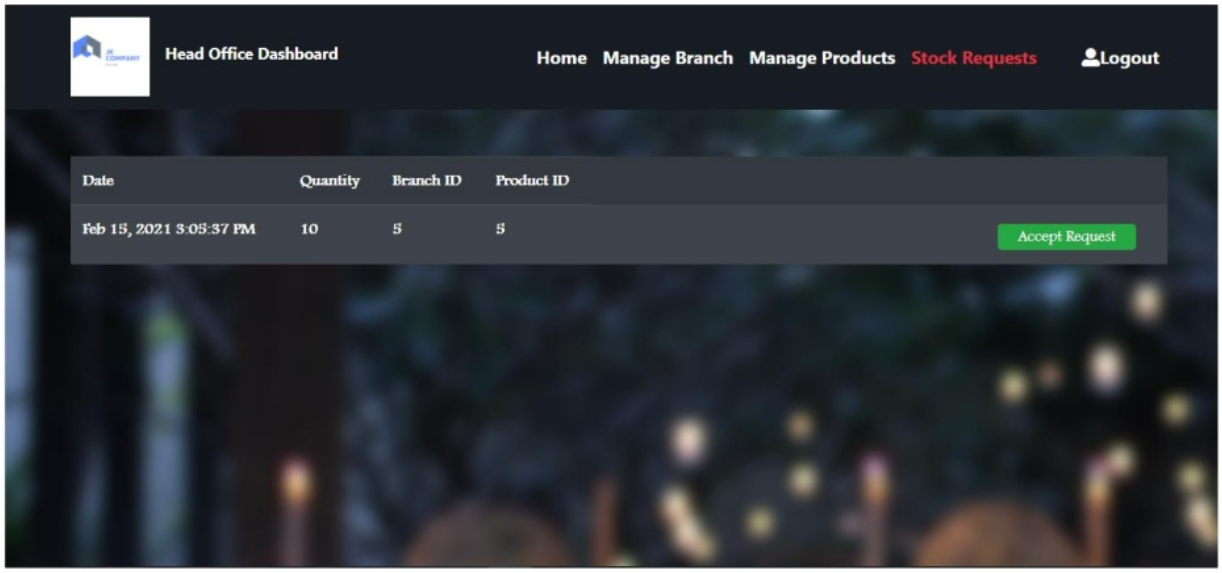


Figure 30 Manage Request

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

Branch Homepage

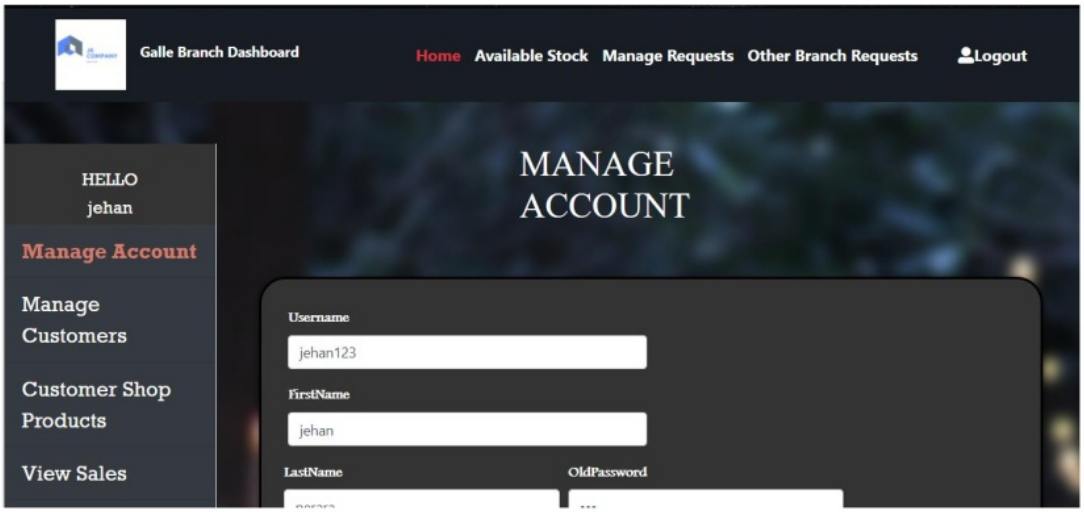


Figure 31 Branch Homepage

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

Manage Account

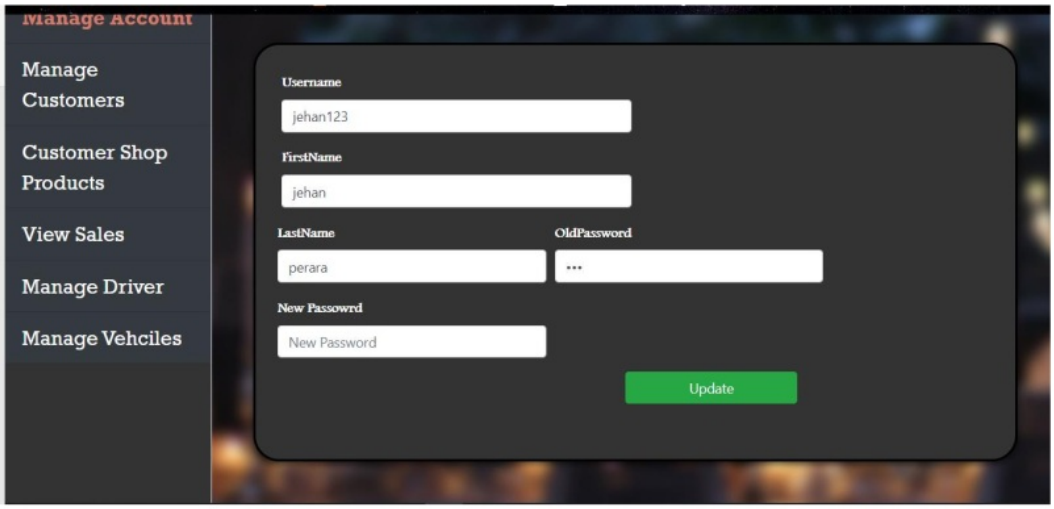


Figure 32 Manage Account

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

Manage Customers

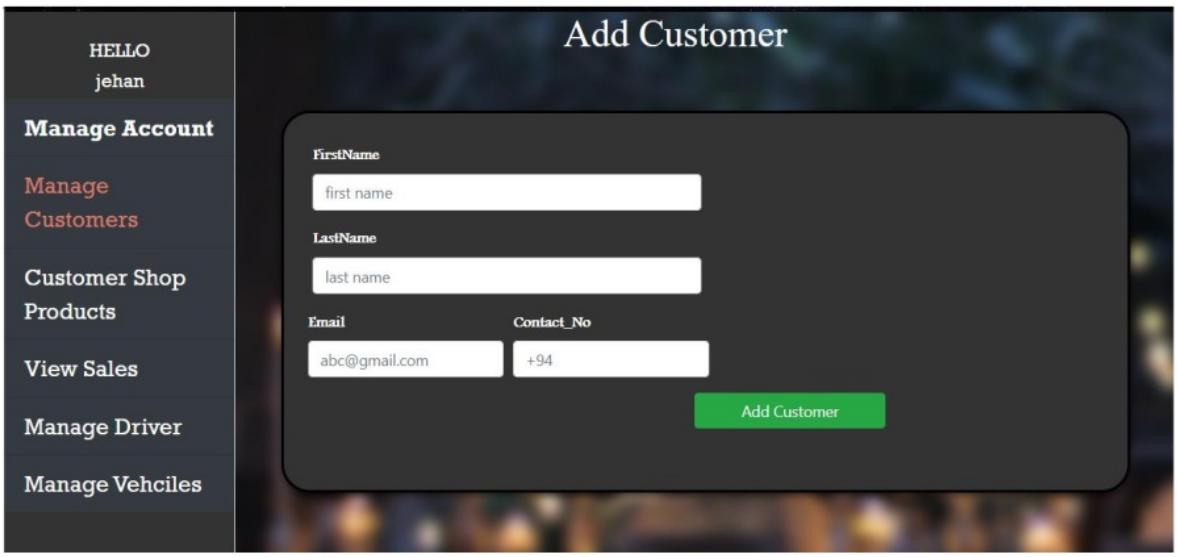


Figure 33 Manage Customers

- 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

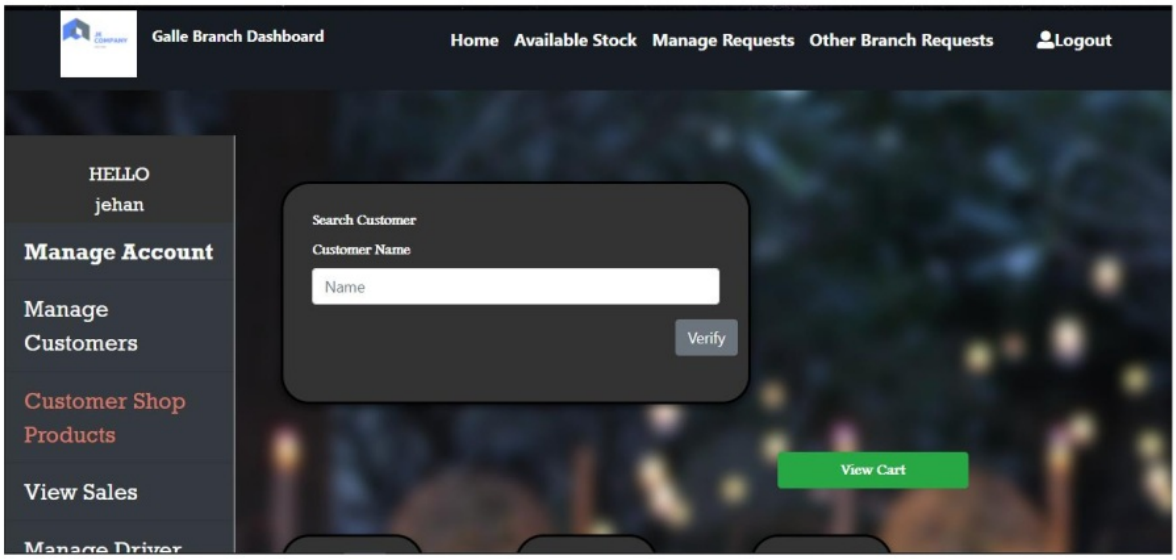


Figure 34 Shop Products

- 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

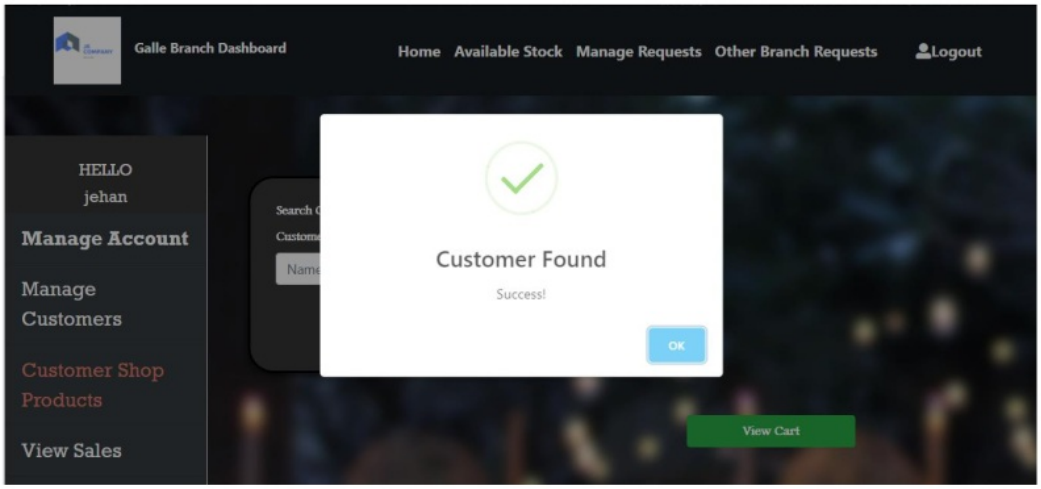


Figure 35 Successful message

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

Searching an Invalid Customer

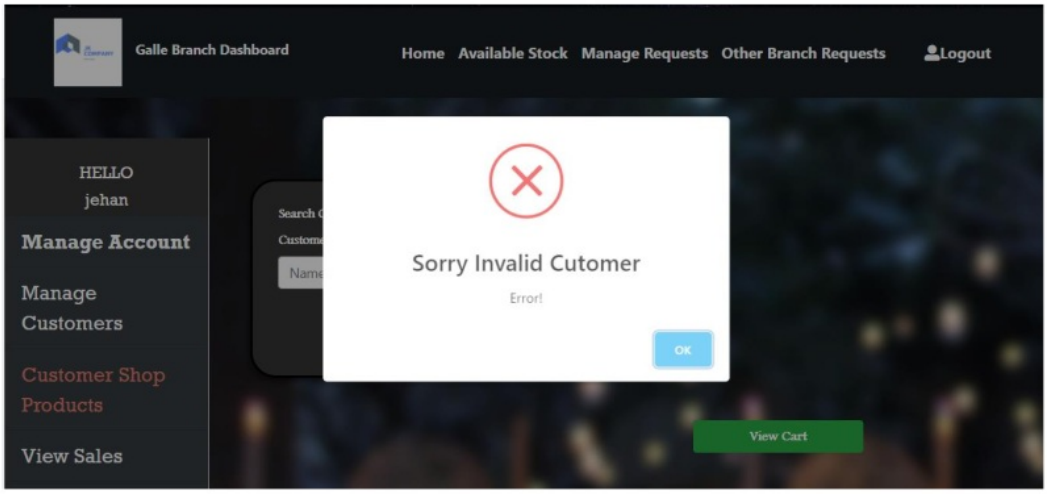


Figure 36 Invalid Message

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

Adding a product to cart

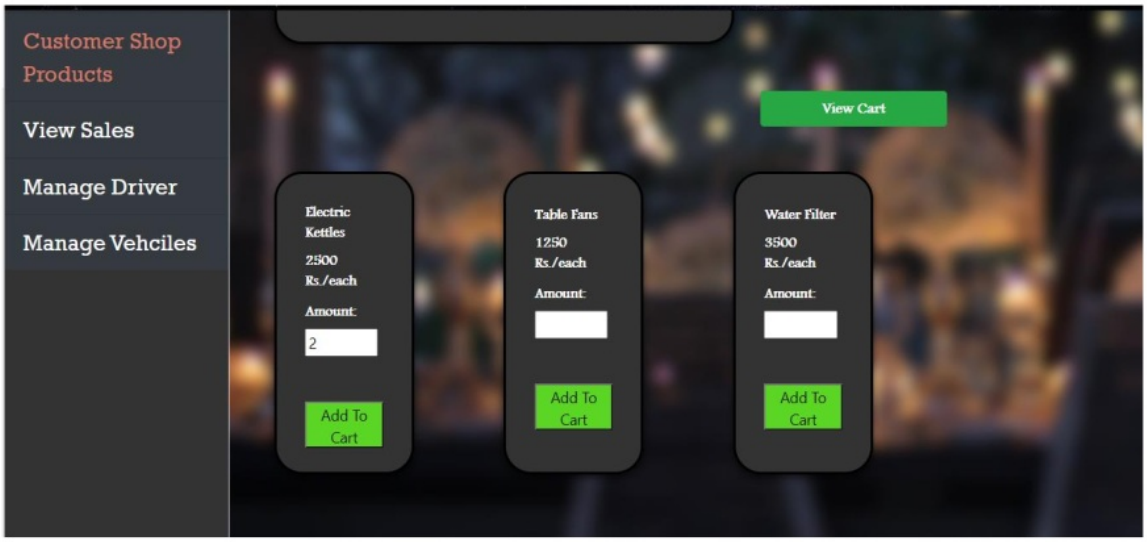


Figure 37 Add to cart

- 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

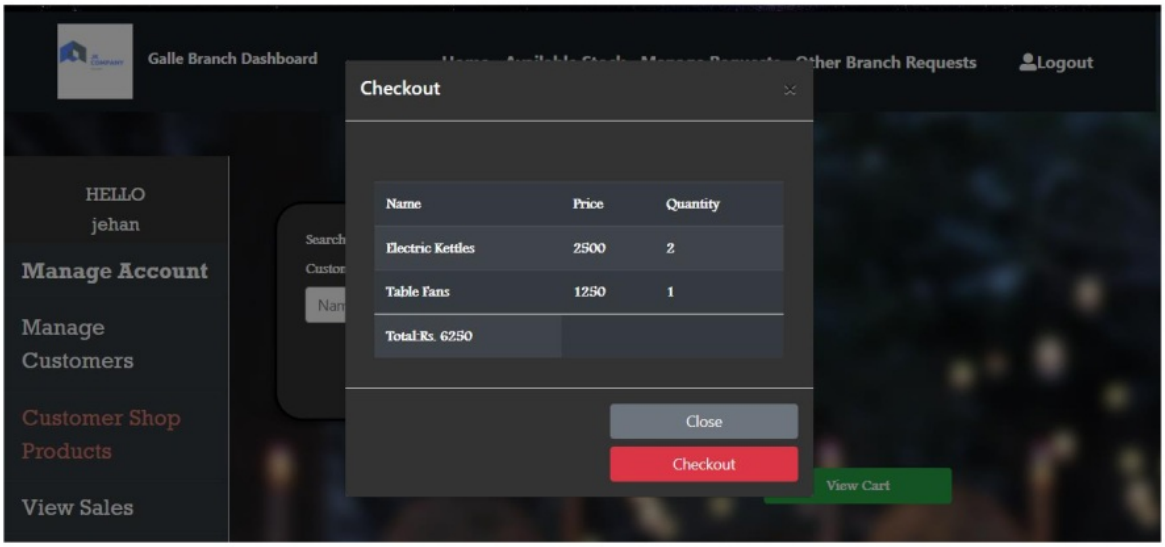


Figure 38 cart

- 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

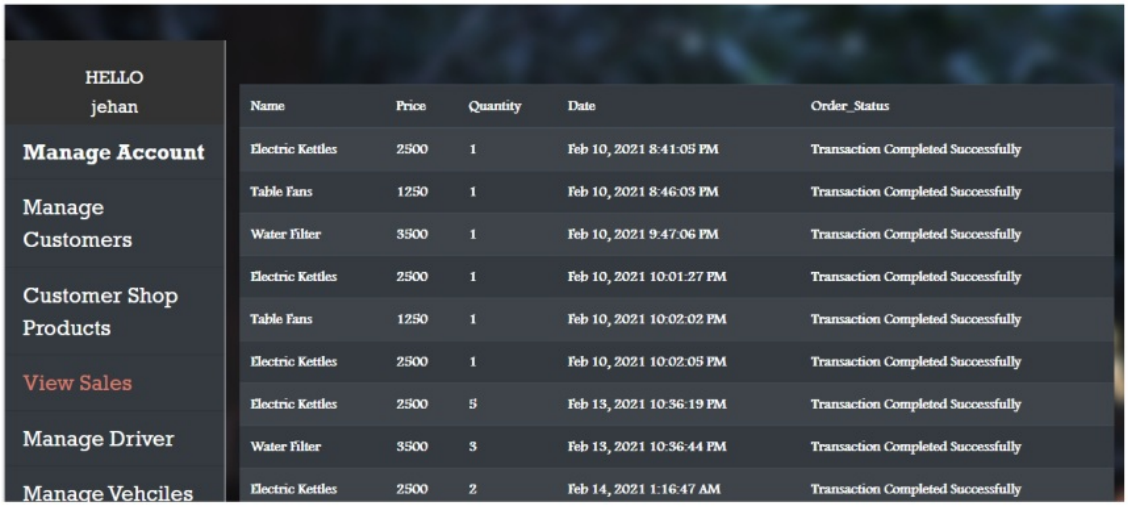


Figure 39 view sales

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

Manage Drivers

Add Driver

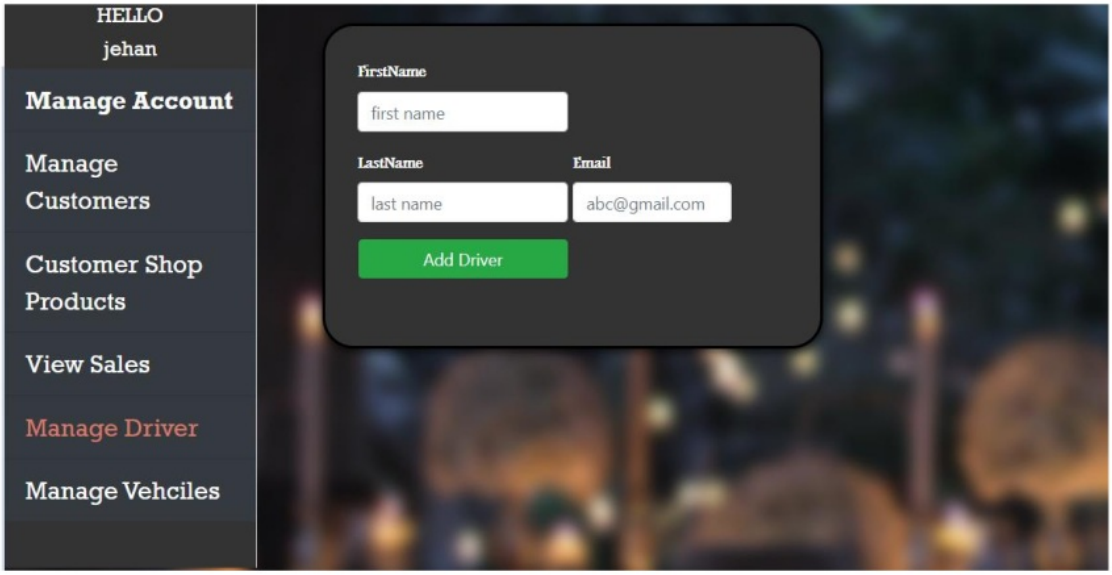


Figure 40 add driver

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

View All/ Delete driver

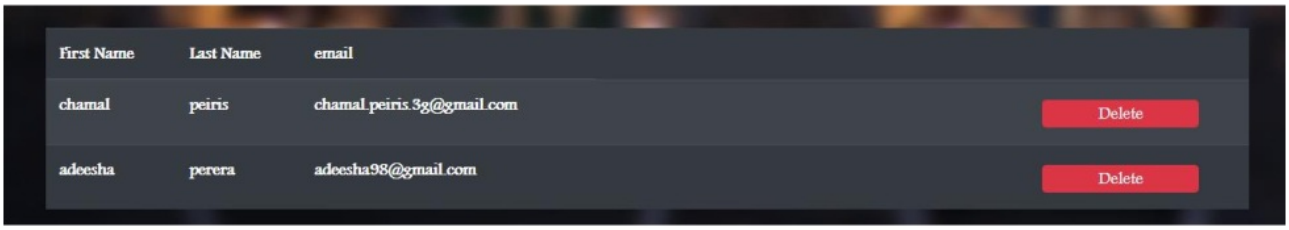


Figure 41 view/delete driver

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

Manage Vehicles

Add Vehicles

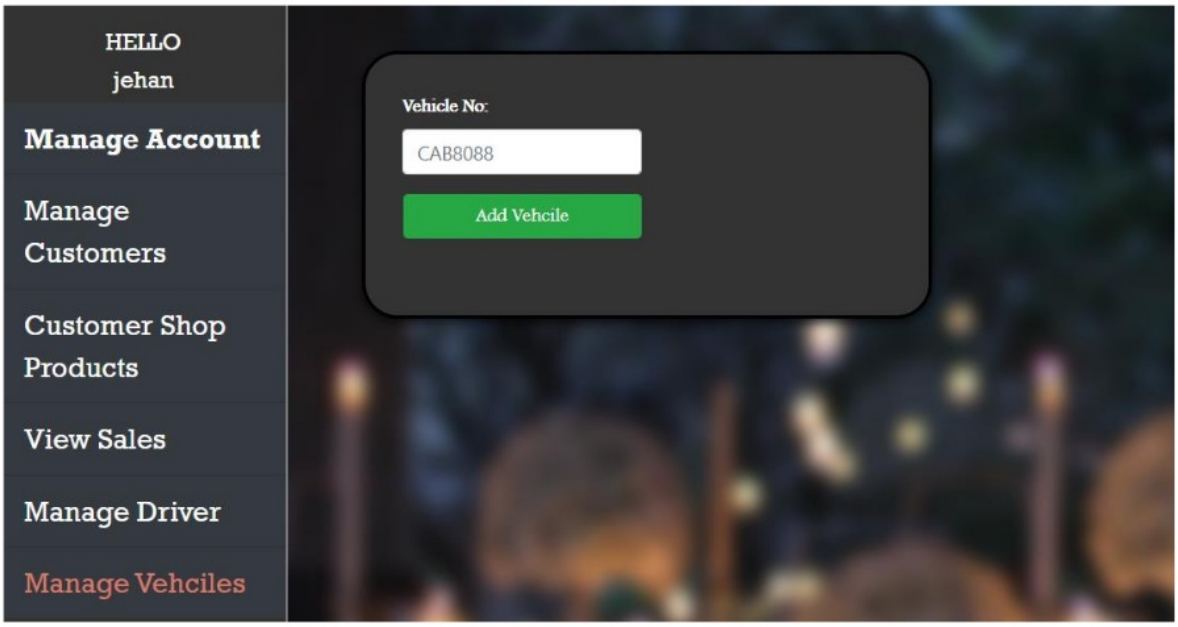


Figure 42 add vehicles

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

View all/ Delete vehicle

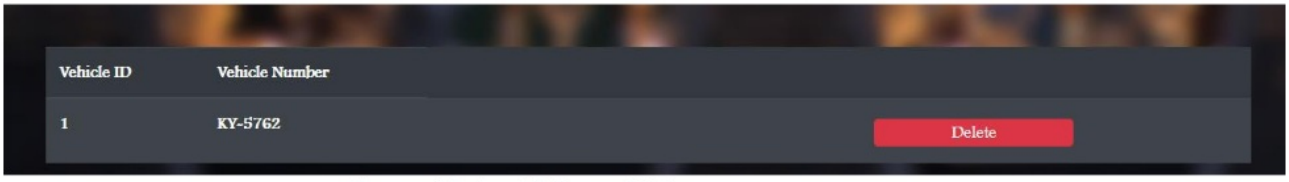


Figure 43 view/delete vehicle

- 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

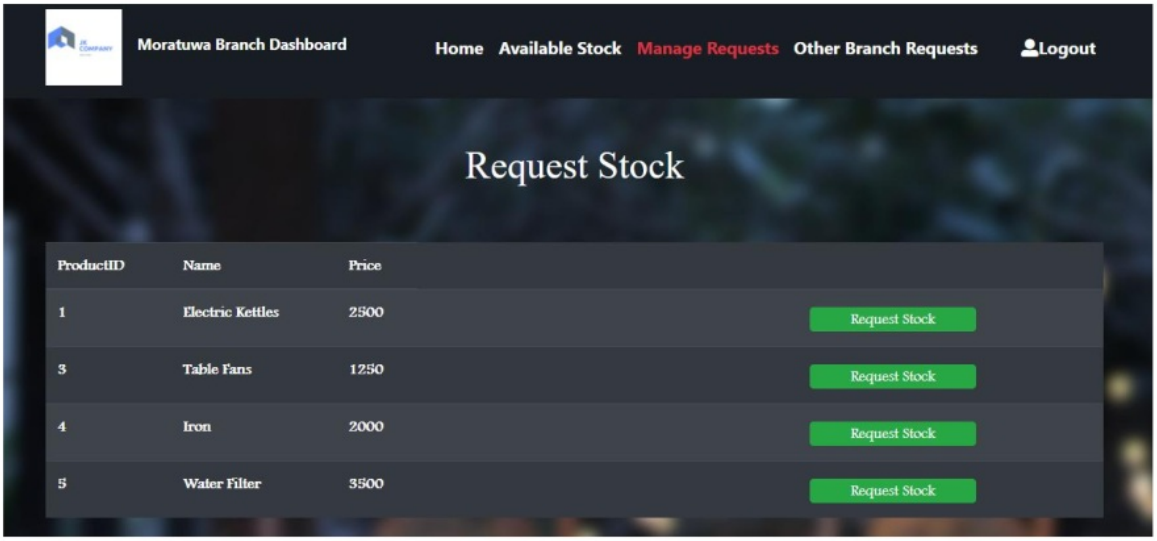


Figure 45 make request

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

Confirm delivery

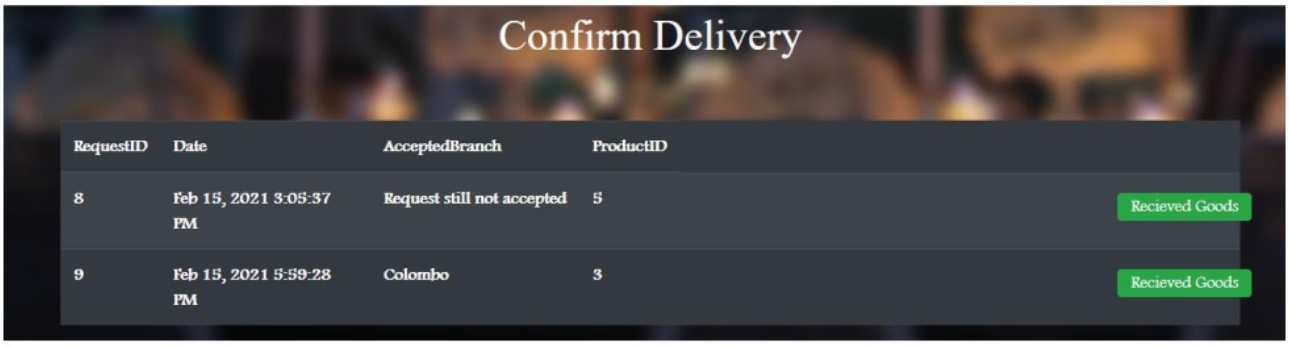


Figure 46 confrim delivery

- 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

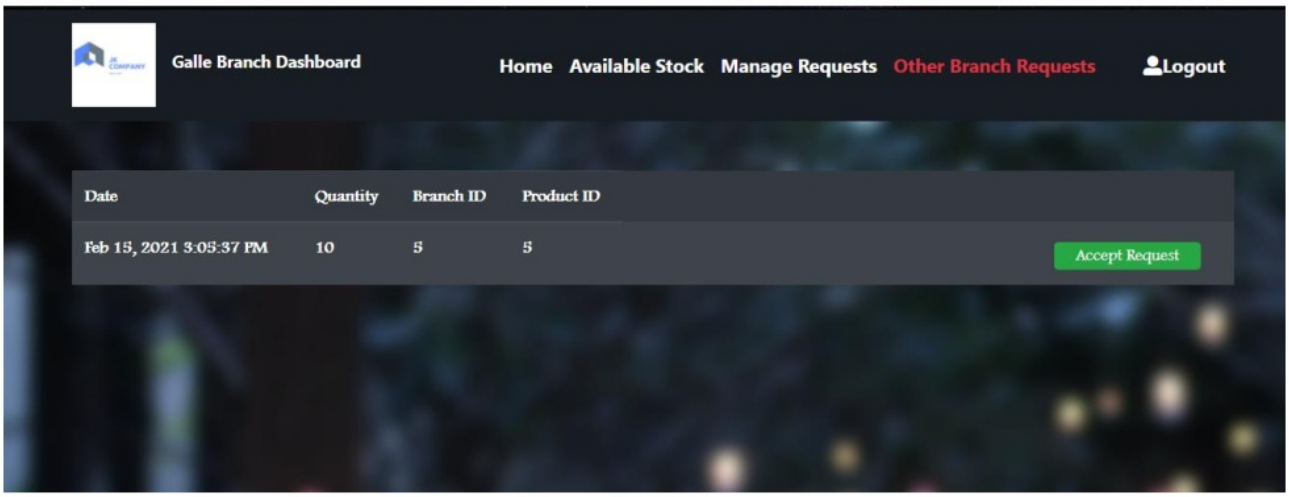


Figure 47 other requests

- 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

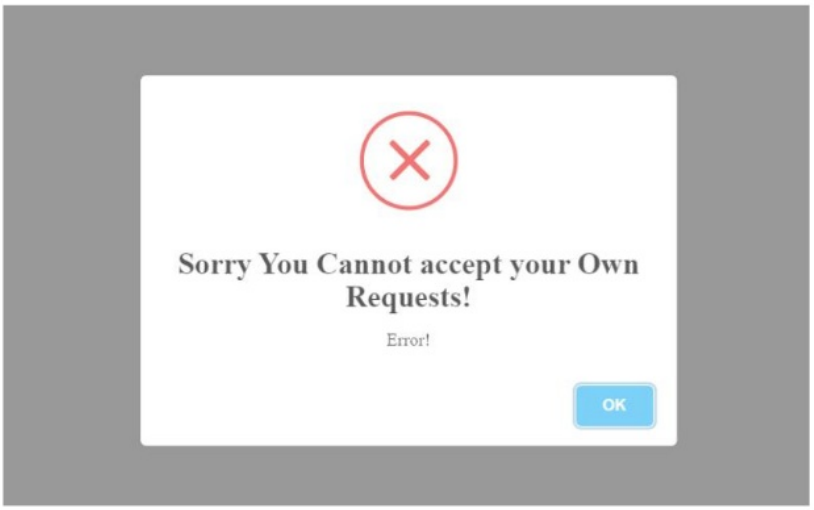


Figure 48 requests

If the requested product not available

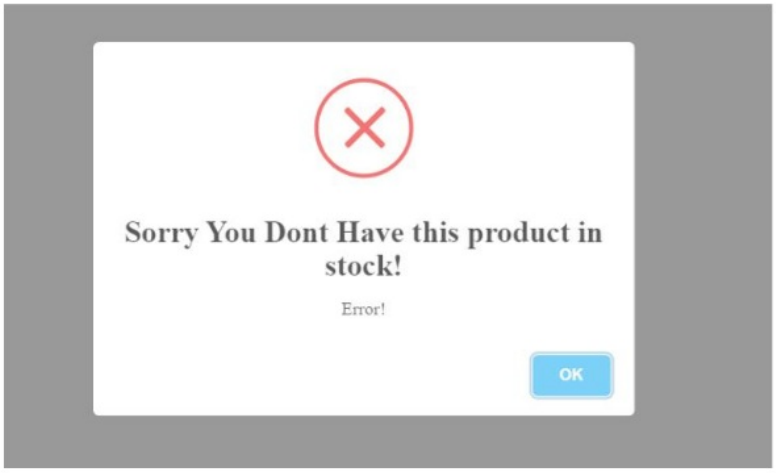


Figure 49 requests

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.

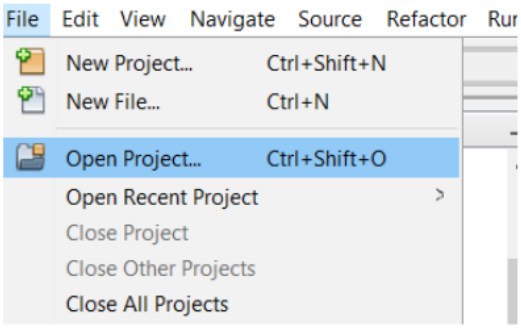


Figure 50 Import files

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

Starting the Project

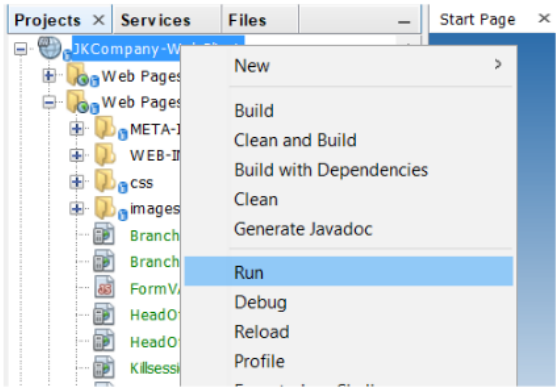


Figure 51 starting the project

- 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.

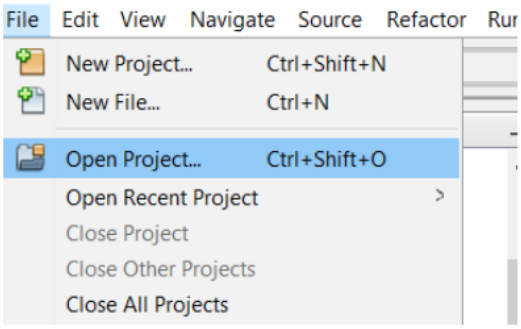


Figure 52 Import files

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

Configuring the Database

Adding the Dependency

```
<dependency>
   <groupId>mysql</groupId>
   <artifactId>mysql-connector-java</artifactId>
   <version>8.0.22</version>
 </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.

```
*/
public class Dbconnection {
   static final String DB_URL = "jdbc:mysql://127.0.0.1:3305/JkCompany";
   static final String USER = "root";
   static final String PASS = "root";
       public static Connection GetconnectConnection()
    {

       Connection con=null;
       try{
           Class.forName("com.mysql.jdbc.Driver");
             con = DriverManager.getConnection(DB_URL, USER, PASS);
           //JOptionPane.showMessageDialog(null, "database connected");
       }
       catch(Exception ex){
           System.out.println("Error connecting"+ex);
       }
       return con;
    }
}
```

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

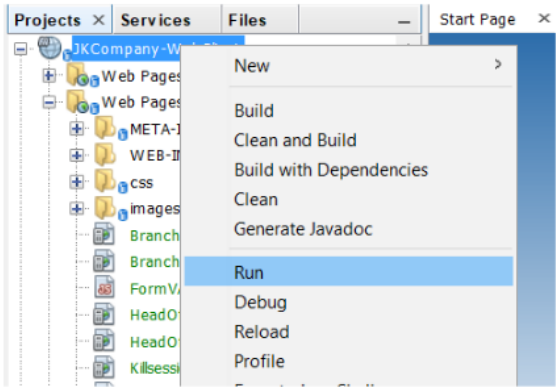


Figure 55 starting the project

- 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.

1

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.

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].

BSC-AdvancedProgramming-BSCSD-21-14.docx

ORIGINALITY REPORT

2%	0%	0%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to University of Wales Institute, Cardiff Student Paper	1%
2	Submitted to NCC Education Student Paper	<1%
3	Submitted to University of Sunderland Student Paper	<1%
4	Submitted to Gusto International College Student Paper	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		