Title of the project: Sport Shoes House Inventory Management System

# INTRODUCTION

# The Sport Shoes House Inventory Management System is a web-based application that use database to save product details such as type of shoes, price, quantity of stock available, size of shoes, brand of shoes and many more. This can be used to track the inventory of the store, or to manage the product details. However, the system merely records product quantity, restocking data and product details. The goal of our system is to reduce the strain of tracking rather than to do it manually by using paper. Furthermore, our system will include with the ability to generate reports of product details. In addition, having an online web-based application of Sport Shoes House Inventory Management System provide multiple advantages such as, correcting stock quantities online and it is a time-saving tool, because it can lessen the hassle of doing inventory recounts that give and assure the company’s accurate record.

# AIM AND OBJECTIVES

1. **AIM**
2. To minimize the task of administrator.
3. To ensure the stocks availability in sufficient quantity.
4. To provide an efficient system for administrator to manage their store.
5. **OBJECTIVES**
6. To implement an online inventory system for Sport Shoes House Company that easier to tracking down the inventory records.
7. To ensure a consistent availability of products.
8. To create a computerized inventory system that use database to store product details rather than using paper.

# SCOPE OF PROJECT

# TECHNOLOGY USED

* + 1. **Java Language**

The language that we used for this project is java language through NetBeans IDE. The factors why we use java language are java is an object-oriented programming language, a great collection of open source libraries and have powerful development tools like NetBeans IDE. NetBeans was used because it is primarily intended for development in Java.

* + 1. **MySQL Database**

We used MySQL database because it is a free and an open source database management system. It manages database and connects them to the software.

* + 1. **Xampp and PhpMyAdmin**

Xampp was designed to help the designer or developer to open their work using laptop to connect to the internet without the connection. We have to start the MySQL engine to connect to the database through PhpMyAdmin.

1. **Project Implementation**

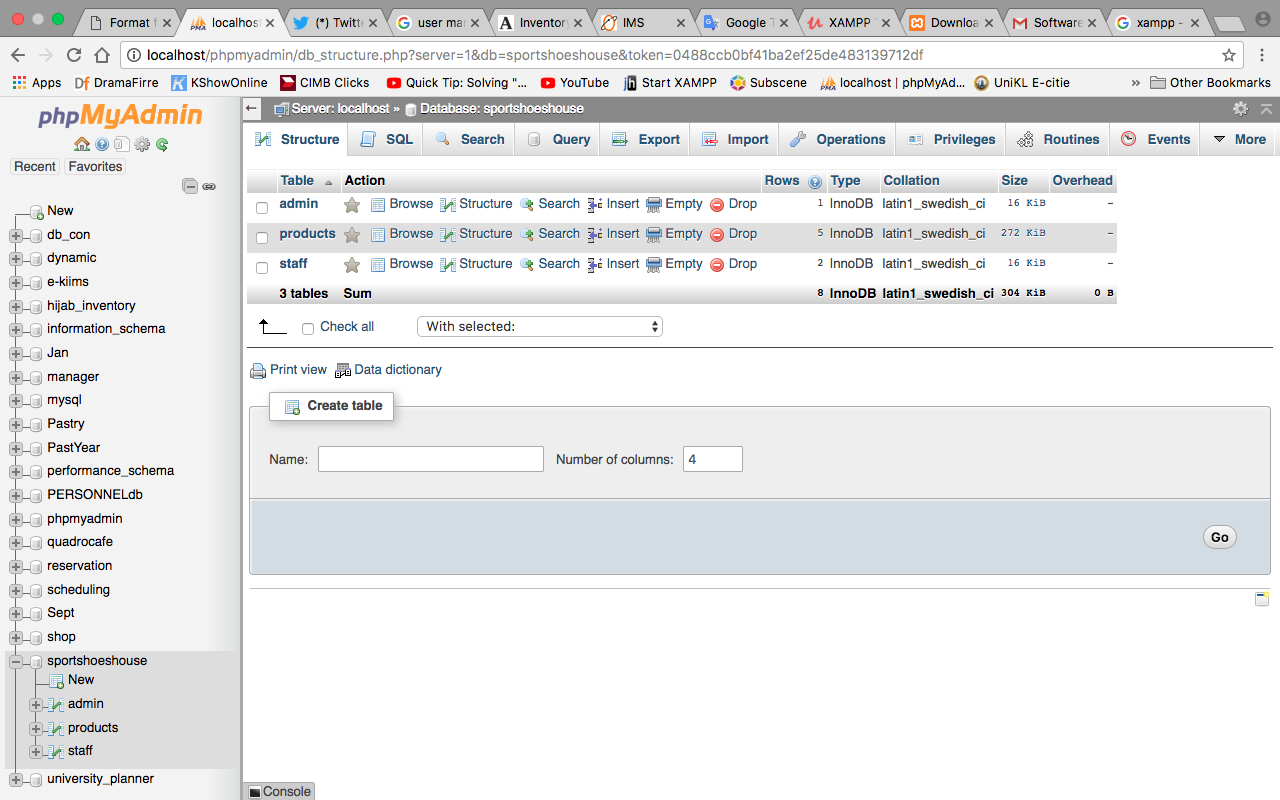
Sport Shoes House Inventory Management System is a system that will records stock of goods. This system is built to checking the inventory. Its main function is adding new items into the database or update the items quantity from the inventory database. Stock verification can be done in case of damaged or loss. The functionalities supported by the module have been divided into the following categories: Login(Admin & Staff), Product Catalog, Register Product, Product Order.

**XAMPP**

It’s a free and open source cross-platform web server. It has four main components which is Apache, MySQL, PHP and Perl. In our system, we used “MySQL”. However, XAMPP application is different for every operating system. User need to download [XAMPP](https://www.apachefriends.org/download.html) and run the XAMPP and click button “Start” for Module **Apache & MySQL** only before access to [MySQL database](http://localhost/phpmyadmin/).

**PhpMyAdmin**

It is an open source database that being used in the system. User need to go to google and search [localhost/phpmyadmin/](http://localhost/phpmyadmin/) to view the inventory database. But before that, user need to import the “[mysql connector java](https://dev.mysql.com/downloads/connector/j/)” in folder “Libraries” of the system. In our system, we used version “5.1.44”. The mysql connector is compatible for MAC and Windows. Database need to be import in the [phpMyAdmin](http://localhost/phpmyadmin). And it will show as figure below.



**Figure 2.2: phpMyAdmin**

**MainPage**

The MainPage will lead user to choose either user’s is Admin or Staff.

**Admin Portal**

AdminPortal form contains “username” and “password” which it needs user to insert their username and password that has been registered. This function as only authenticate user can access the system.

Login AdminPortal Code:

//Connection Database

Connection conn = null;

PreparedStatement pst = null;

ResultSet rs = null;

private void jButtonLoginActionPerformed(java.awt.event.ActionEvent evt)

{

conn = MysqlConnect.ConnectDB();

String sql = "Select \* from admin where admin\_username=? and admin\_password=?";

try

{

pst = conn.prepareStatement(sql);

pst.setString(1,jTextFieldUsername.getText());

pst.setString(2,jPasswordFieldPwd.getText());

rs = pst.executeQuery();

if(rs.next())

{

JOptionPane.showMessageDialog(null,"WELCOME TO ADMIN PAGE");

AdminHomepage r = new AdminHomepage();

r.setVisible(true);

this.dispose();

}

else

{

JOptionPane.showMessageDialog(null,"INVALID USERNAME OR PASSWORD", "ACCESS DENIED", JOptionPane.ERROR\_MESSAGE);

}

}

catch(Exception e)

{

JOptionPane.showMessageDialog(null, e);

}

}

**Admin Homepage**

AdminHomepage will be shown only after admin user, login into the system. This page consists of “Admin Profile” to change current user password, “Employee Details” to view employee records, “Product Details” to register new product, update or delete. “Report” to view record of goods quantity available for every item.

**Register Product**

The “RegisterProduct” will allow admin users to register new product into the inventory database. ‘Add’ button function as adding new item. ‘Edit’ button function as update selected item in the inventory database such as edit item quantity. ‘Delete’ button function as remove all selected item data from the database. ‘Clear’ button function as clear all the data in “Text Field”. ‘Record’ button function as view inventory records. ‘Search’ button function as finding product based on character/’s or number/’s typed.

JTable Code :

//populate the JTable

public void Show\_Products\_In\_JTable()

{

ArrayList<Products> list = getProductList();

DefaultTableModel model = (DefaultTableModel)jTableProductsDetail.getModel();

// clear jtable content

model.setRowCount(0);

Object[] row = new Object[6];

for(int i = 0; i < list.size(); i++)

{

row[0] = list.get(i).getProductCode();

row[1] = list.get(i).getProductName();

row[2] = list.get(i).getProductBrand();

row[3] = list.get(i).getProductPrice();

row[4] = list.get(i).getProductQuantity();

model.addRow(row);

}

}

Add Product Code:

private void jButtonAddProductActionPerformed(java.awt.event.ActionEvent evt)

{

if (checkInputs() && Images != null)

{

Connection con = getConnection();

try

{

PreparedStatement ps = con.prepareStatement("Insert INTO products (ProductCode, ProductName, ProductBrand, ProductPrice, ProductQuantity, ProductImage)" + "values(?,?,?,?,?,?)");

ps.setString(1, jTextFieldProductCode.getText());

ps.setString(2, jTextFieldProductName.getText());

String value = jComboBoxProductBrand.getSelectedItem().toString();

ps.setString(3, value);

ps.setString(4, jTextFieldProductPrice.getText());

ps.setString(5, jTextFieldProductQuantity.getText());

InputStream img = new FileInputStream(new File(Images));

ps.setBlob(6, img);

ps.executeUpdate();

Show\_Products\_In\_JTable();

JOptionPane.showMessageDialog(null, "Data inserted");

}

catch (Exception ex)

{

JOptionPane.showMessageDialog(null, ex.getMessage());

}

}

else

{

JOptionPane.showMessageDialog(null, "One or more field are empty");

}

}

Edit Product Code:

private void jButtonEditProductActionPerformed(java.awt.event.ActionEvent evt)

{

if (checkInputs() && jTextFieldProductCode.getText() != null)

{

String UpdateQuery = null;

PreparedStatement ps = null;

Connection con = getConnection();

//update without image

if (Images == null)

{

try

{

UpdateQuery = "UPDATE products SET ProductName = ?, ProductBrand = ?, ProductPrice = ?, ProductQuantity =? WHERE ProductCode =?";

ps = con.prepareStatement(UpdateQuery);

ps.setString(1, jTextFieldProductName.getText());

String value = jComboBoxProductBrand.getSelectedItem().toString();

ps.setString(2, value);

ps.setString(3, jTextFieldProductPrice.getText());

ps.setString(4, jTextFieldProductQuantity.getText());

ps.setInt(5, Integer.parseInt(jTextFieldProductCode.getText()));

ps.executeUpdate();

Show\_Products\_In\_JTable();

JOptionPane.showMessageDialog(null, "Data updated!");

}

catch (SQLException ex)

{

Logger.getLogger(RegisterProduct.class.getName()).log(Level.SEVERE, null, ex);

}

}

//update with image

else

{

try

{

InputStream img = new FileInputStream(new File(Images));

UpdateQuery = "UPDATE products SET ProductName = ?, ProductBrand = ?, ProductPrice = ?, ProductQuantity =?, ProductImage = ? WHERE ProductCode =?";

ps = con.prepareStatement(UpdateQuery);

ps.setString(1, jTextFieldProductName.getText());

String value = jComboBoxProductBrand.getSelectedItem().toString();

ps.setString(2, value);

ps.setString(3, jTextFieldProductPrice.getText());

ps.setString(4, jTextFieldProductQuantity.getText());

ps.setBlob(5,img);

ps.setInt(6, Integer.parseInt(jTextFieldProductCode.getText()));

ps.executeUpdate();

Show\_Products\_In\_JTable();

JOptionPane.showMessageDialog(null, "Data updated!");

}

catch (Exception ex)

{

JOptionPane.showMessageDialog(null, ex.getMessage());

}

}

}

else

{

JOptionPane.showMessageDialog(null, "One or more fields are empty or wrong");

}

}

Delete Product Code:

private void jButtonDeleteProductActionPerformed(java.awt.event.ActionEvent evt)

{

if (!jTextFieldProductCode.getText().equals(""))

{

try

{

Connection con = getConnection();

PreparedStatement ps = con.prepareStatement("DELETE FROM products WHERE ProductCode =?");

String ObjButtons[] = {"Yes","No"};

int PromptResult = JOptionPane.showOptionDialog(null,"Are you sure you want to delete this product?","Product Details",JOptionPane.DEFAULT\_OPTION,JOptionPane.WARNING\_MESSAGE,null,ObjButtons,ObjButtons[1]);

if(PromptResult==JOptionPane.YES\_OPTION)

{

int id = Integer.parseInt(jTextFieldProductCode.getText());

ps.setInt(1, id);

ps.executeUpdate();

Show\_Products\_In\_JTable();

JOptionPane.showMessageDialog(null, "Product Deleted ");

}

}

catch (SQLException ex)

{

Logger.getLogger(RegisterProduct.class.getName()).log(Level.SEVERE, null, ex);

JOptionPane.showMessageDialog(null, "Product Not Deleted!");

}

}

else

{

JOptionPane.showMessageDialog(null, "Product not deleted : Hijab code to delete");

}

}

Show Product Records Code:

private void jButtonShowProductsListActionPerformed(java.awt.event.ActionEvent evt)

{

Show\_Products\_In\_JTable();

}

Search Product Records Code:

public void findProduct()

{

ArrayList<Products> products = ListProduct(jTextFieldProductSearch.getText());

DefaultTableModel model = new DefaultTableModel();

model.setColumnIdentifiers(new Object[]{"ProductCode", "ProductName", "ProductBrand", "ProductPrice", "ProductQuantity"});

Object[] row = new Object[6];

for(int i = 0; i < products.size(); i++)

{

row[0] = products.get(i).getProductCode();

row[1] = products.get(i).getProductName();

row[2] = products.get(i).getProductBrand();

row[3] = products.get(i).getProductPrice();

row[4] = products.get(i).getProductQuantity();

model.addRow(row);

}

jTableProductsDetail.setModel(model);

}

**Product Catalog**

“ProductCatalog” will shows all the items available in the store in a table form. User can perform searching for product by product code, product name, product brand, product price and product quantity in the text field provided besides the ‘Search’ button. And user need to click the ‘Search’ button to view the data needed.

Search Product Records Code:

//connection to database

public Connection getConnection()

{

Connection con = null;

try

{

con = DriverManager.getConnection("jdbc:mysql://localhost/sportshoeshouse", "root", "");

return con;

}

catch (SQLException ex)

{

Logger.getLogger(RegisterProduct.class.getName()).log(Level.SEVERE, null, ex);

return null;

}

}

//function to return products arraylist with particular data

public ArrayList<Products> ListProduct (String ValToSearch)

{

ArrayList<Products> productList = new ArrayList<Products>();

Statement st;

ResultSet rs;

try

{

Connection con = getConnection();

st = con.createStatement();

String searchQuery = "SELECT \* FROM products WHERE CONCAT (ProductCode,ProductName,ProductBrand,ProductPrice,ProductQuantity, ProductImage) LIKE '%"+ValToSearch+"%'";

rs = st.executeQuery(searchQuery);

Products products;

while(rs.next())

{

products = new Products(rs.getInt("ProductCode"), rs.getString("ProductName"),rs.getString("ProductBrand"),Float.parseFloat(rs.getString("ProductPrice")), rs.getInt("ProductQuantity"), rs.getBytes("ProductImage"));

productList.add(products);

}

}

catch (Exception ex)

{

System.out.println(ex.getMessage());

}

return productList;

}

//search for product

public void findProduct()

{

ArrayList<Products> products = ListProduct(jTextFieldProductSearch.getText());

DefaultTableModel model = new DefaultTableModel();

model.setColumnIdentifiers(new Object[]{"ProductCode", "ProductName", "ProductBrand", "ProductPrice", "ProductQuantity", "ProductImage"});

Object[] row = new Object[6];

for(int i = 0; i < products.size(); i++)

{

row[0] = products.get(i).getProductCode();

row[1] = products.get(i).getProductName();

row[2] = products.get(i).getProductBrand();

row[3] = products.get(i).getProductPrice();

row[4] = products.get(i).getProductQuantity();

row[5] = products.get(i).getProductImage();

model.addRow(row);

}

jTableProdCat.setModel(model);

}

private void jButtonSearchActionPerformed(java.awt.event.ActionEvent evt)

{

findProduct();

}

**Product Report**

This “ProductReport” will allows user to view all the inventory records. and ‘Print’ button will print the records in a paper.

**Report Code**

private void jButtonPrintProdDetailsActionPerformed(java.awt.event.ActionEvent evt)

{

MessageFormat header = new MessageFormat("Products Report");

MessageFormat footer = new MessageFormat("Page {0,number,integer}");

try

{

jTableProductReports.print(JTable.PrintMode.NORMAL, header, footer);

}

catch (java.awt.print.PrinterException e)

{

System.err.format("Cannot print %s%n", e.getMessage());

}

}

**Register Staff**

This “RegisterStaff” will allow admin users to register new employee into the system. ‘Add’ button function as adding new employee. ‘Edit’ button function as update selected employee in the database such as edit employee designation. ‘Delete’ button function as remove all selected employee data from the database. ‘Clear’ button function as clear all the data in “Text Field”. ‘Record’ button function as view employees’ records. ‘Search’ button function as finding employee based on character/’s or number/’s typed. The coding used in this form is same with register product, only the database called is different.

**Staff Details**

“StaffDetails” will shows all the employee that works in the store. Admin user can perform printing all staff records.

**Ordering**

The “Ordering” will show product available in the Store and staff user can buy item/s for customer. To buy the item/s, user need to filled-up customer name and address. Then click button buy, select size, insert quantity, and select payment method. After that, user need to click ‘Calculate(RM)’ button to view the price of the item includes with GST. And the item that user buy will be shown in the table at the right side. To buy more items, user can click ‘Add More Item’ button and select the item and repeat as explained in above text. After done with buying, user need to click ‘Confirm Order’ button to get the receipt of item/s. ‘Reset’ button function as clear all text field.

**Calculate Total Code:**

private void jButtonCalTotalActionPerformed(java.awt.event.ActionEvent evt)

{

if(jTextFieldItem.getText().equalsIgnoreCase(""))

{

jTextFieldItem.requestFocus();

jTextFieldCustName.setText(null);

jTextAreaAddress.setText(null);

jTextFieldPrice.setText(null);

jTextFieldSubTotal.setText(null);

jTextFieldGst.setText(null);

jTextFieldOverallTot.setText(null);

jTextAreaSummary.setText(null);

subtotal=0;

tax=0;

total=0;

price=0;

JOptionPane.showMessageDialog(null,"Error, Please Enter Item First");//edit

}

else

{

calculateprice();

while(z<x)

{

subtotal = subtotal + price2[z];

z++;

}

tax = subtotal \* 0.06;

total = subtotal + tax;

jTextFieldSubTotal.setText(String.valueOf(df.format(subtotal)));

jTextFieldGst.setText(String.valueOf(df.format(tax)));

jTextFieldOverallTot.setText(String.valueOf(df.format(total)));

}

**Change Password**

User can change their password to tighten their security of user’s account. To perform the password changes, it requires username, old password, new password and confirm password. Any of the requires field that is wrongly entered, it will show error and user password can’t be change.

2.1 Explanation for every function and modules

2.1.1 Discussion: Design Pattern

1. Singleton Design Pattern (Login, RegisterProduct)
2. Behavioral Design Pattern (StaffDetails, import java.util.logging.Logger;)
3. Momento Design Pattern (import java.io.Serializable; Products\_1)
4. Bridge Design Pattern ()

2.1.2 Discussion: Design Principle

1. Correctness (Error Message)
2. Robustness (Multiform)
3. Reusability (Existing Code)
4. Flexibility (Add Product/Add Employee)