Criterion C: Development

Table of Complexities

1. Graphical User Interface

1.1. Netbeans - JFrame Form

1.1.1. Basic Requirements

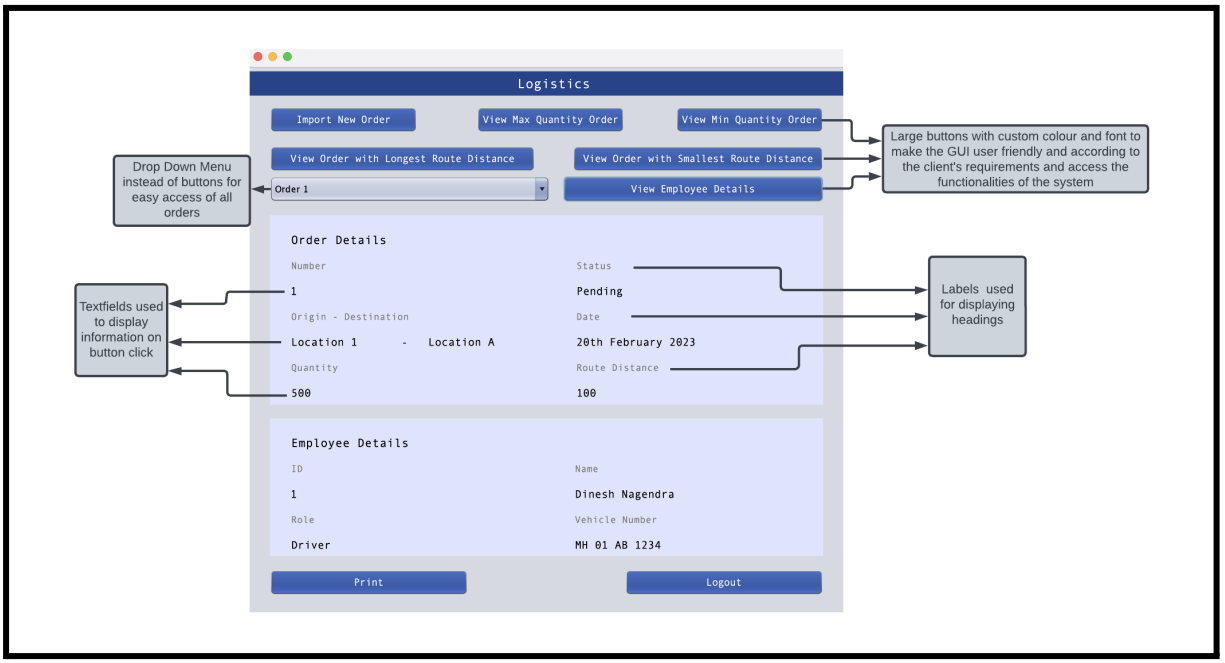


Fig 1: GUI of Logistics Screen (Self-clicked)

This graphical user interface is designed on Netbeans using Java to be user-friendly and visually appealing, making it easier for users to work with the system. They can easily see what actions they can take such as viewing order details, employee details, printing the information etc. It is also consistent with the client's requirements of the blue colour scheme, compact layout, font customization. This makes all the text and elements on the screen extremely legible and more visually attractive. Another advantage is that it allows for multitasking. Firstly, the client is able to multiple data sorting options like view max quantity order, view minimum quantity order etc. all in one screen. He is also able to select specific orders from the drop down menu. Additionally when the view employee details button is clicked, the user is able to view the employee details for that order on the same screen. Overall, the GUI provides a more intuitive method for the users to complete their tasks.

1.1.2. Navigation Buttons

Fig 2: Navigation buttons (Self-clicked)

There are three main navigation buttons in this product. The first button is the login button, which allows the user to switch to the logistics screen when the login details entered are correct. The second button is the logout button that allows the user to switch back to the login screen. The third button is the import new order button that opens a new window with additional buttons and a table for the user to access. Overall, the navigation buttons improve the usability and accessibility of the system.

1.2. Data Validation

1.2.1. Login Screen

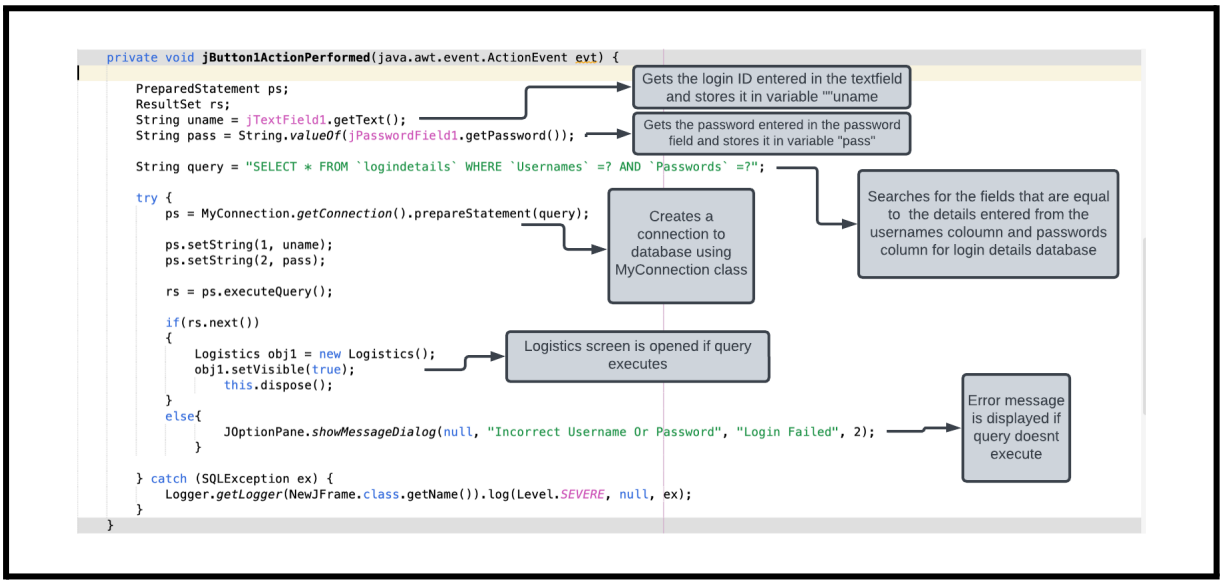


Fig 3: Login Screen Code (Self-clicked)

This code prevents unauthorised access to the system by creating a login screen. This code retrieves the username and password entered by the user, prepares a SQL query to select records from a table where the username and password match the user's input, sets the parameters using the PreparedStatement interface, and executes the query using the executeQuery() method. It then checks if the ResultSet object contains any records by calling the next() method. If the records exist, it means that the user's login credentials are valid and a new instance of the Logistics class is created and displayed. If the login credentials are invalid, an error message is displayed using the JOptionPane.showMessageDialog() method.

1.2.2. Drop Down Menu

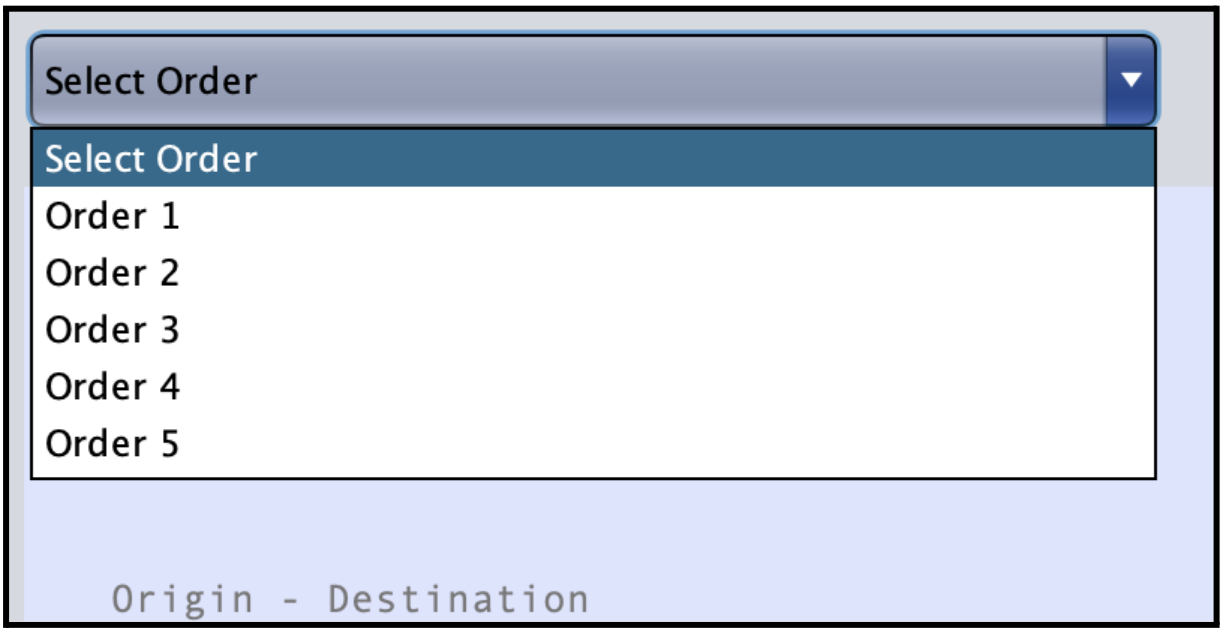


Fig 4: Drop-down menu (Self-clicked)

A drop-down menu is used to reduce errors by limiting the available choices to a predefined list, preventing users from entering invalid data or making typos. This drop down menu displays the orders present in the order details database.

1.2.3. Import File Limit

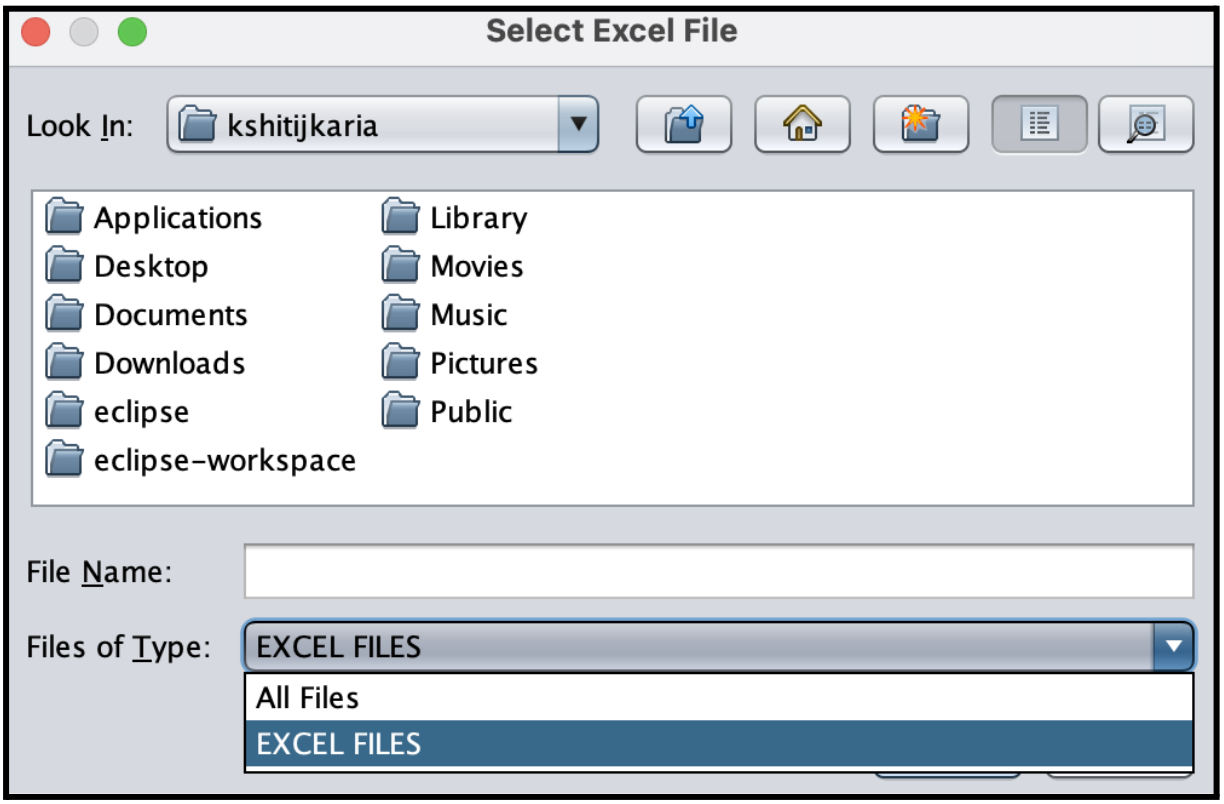
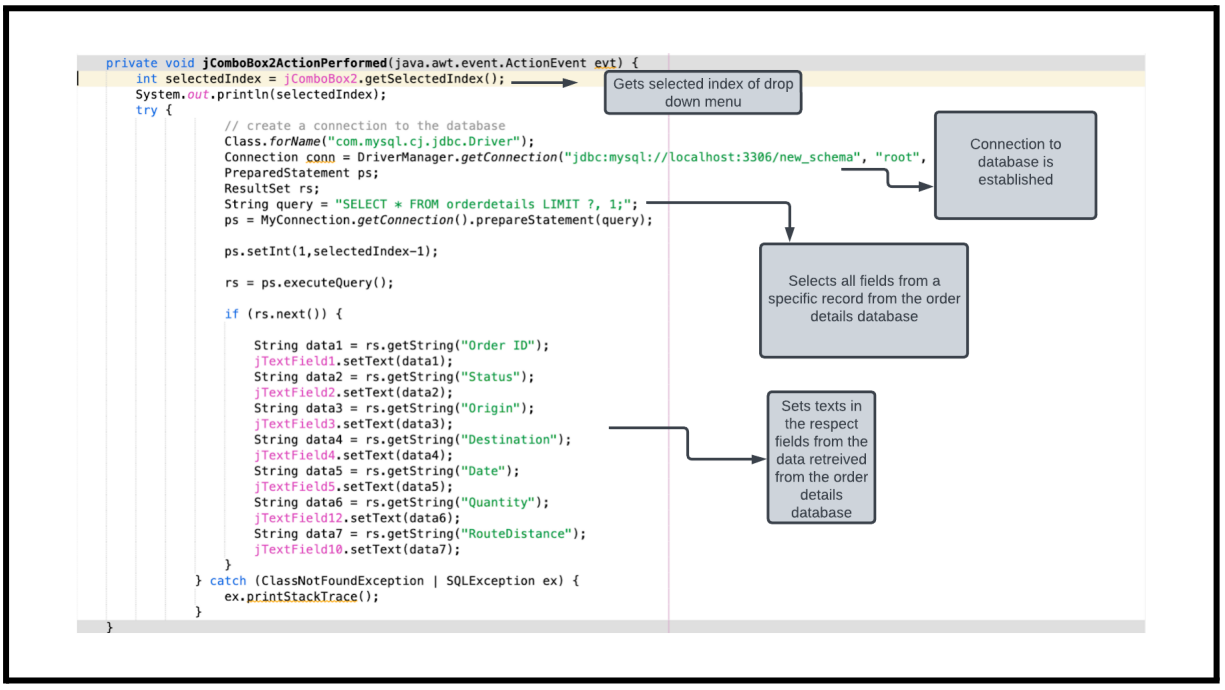


Fig 5: File format limit (Self-clicked)

When the user chooses to import a file, the type of file he can import to only excel files. By only allowing excel files, it reduces the error of importing files of other formats such as pdfs. The program stops working when invalid file formats are imported.

2. Functionality

2.1. Displaying Details




```
private void jComboBox2ActionPerformed(java.awt.event.ActionEvent evt) {
    int selectedIndex = jComboBox2.getSelectedIndex();
    System.out.println(selectedIndex);
    try {
            // create a connection to the database
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/new_schema", "root",
            PreparedStatement ps;
            ResultSet rs;
            String query = "SELECT * FROM orderdetails LIMIT ?, 1;";
            ps = MyConnection.getConnection().prepareStatement(query);

            ps.setInt(1,selectedIndex-1);

            rs = ps.executeQuery();

            if (rs.next()) {

                String data1 = rs.getString("Order ID");
                jTextField1.setText(data1);
                String data2 = rs.getString("Status");
                jTextField2.setText(data2);
                String data3 = rs.getString("Origin");
                jTextField3.setText(data3);
                String data4 = rs.getString("Destination");
                jTextField4.setText(data4);
                String data5 = rs.getString("Date");
                jTextField5.setText(data5);
                String data6 = rs.getString("Quantity");
                jTextField12.setText(data6);
                String data7 = rs.getString("RouteDistance");
                jTextField10.setText(data7);
            }
        } catch (ClassNotFoundException | SQLException ex) {
            ex.printStackTrace();
        }
}
```

Fig 6: Order Details Code(Self-clicked)

This code retrieves data from the MySQL database based on the selected item in a JComboBox component. It first retrieves the index of the selected item and constructs a SQL query that selects a single record from the "orderdetails" table using the LIMIT clause and the selected index as a parameter. It then prepares a SQL statement, sets the parameter, and executes the query, retrieving the results in a ResultSet object. It then retrieves the data and sets the text of the JTextField components to display the retrieved data.

2.2. Sorting Methods

```
    // TODO add your handling code here:
    try {

            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/new_schema", "root", "Football@24");
            PreparedStatement ps;
            ResultSet rs;
            String query = "SELECT * FROM orderdetails ORDER BY RouteDistance ASC LIMIT 0,1;";
            ps = MyConnection.getConnection().prepareStatement(query);
            rs = ps.executeQuery();
            // retrieve the data and display it in the text field
            if (rs.next()) {
                String data1 = rs.getString("Order ID");
                jTextField1.setText(data1);
                String data2 = rs.getString("Status");
                jTextField2.setText(data2);
                String data3 = rs.getString("Origin");
                jTextField3.setText(data3);
                String data4 = rs.getString("Destination");
                jTextField4.setText(data4);
                String data5 = rs.getString("Date");
                jTextField5.setText(data5);
                String data6 = rs.getString("Quantity");
                jTextField12.setText(data6);
                String data7 = rs.getString("RouteDistance");
                jTextField10.setText(data7);

            }

        } catch (ClassNotFoundException | SQLException ex) {
            ex.printStackTrace();
        }
}
```

Fig 7: Minimum Route Distance button Code(Self-clicked)

This system allows the user to directly select orders with maximum quantity, minimum quantity, maximum route distance and minimum route distance. It is similar to the for displaying order details with the difference that instead of selecting all fields from a specific record, it will first sort the data into ascending order for minimum/descending order for maximum and the returns all fields from the first record which will automatically have the maximum/minimum quantity or route distance data. This functionality is extremely useful for the user as he need not manually search for this data by scrolling through each and every order detail record .

2.3. Generating Reports

2.3.1. Logistics Screen

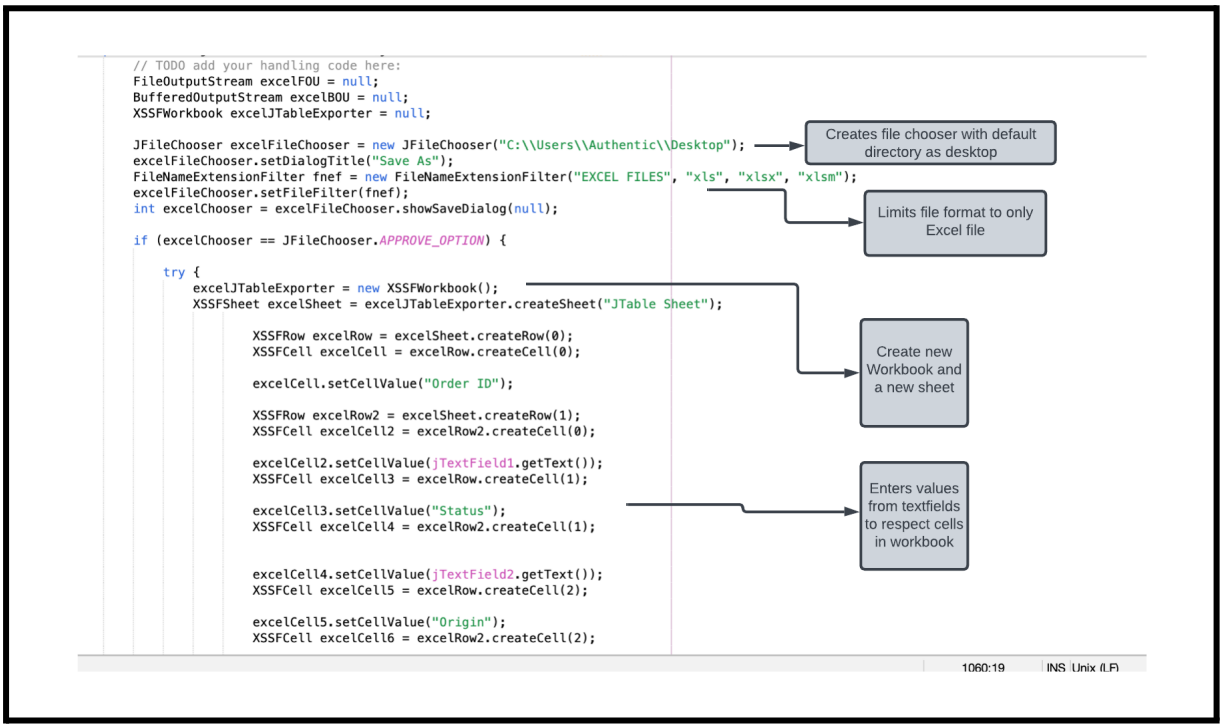


Fig 8: Print button Code(Self-clicked)

This code exports data from a form to an Excel file using Apache POI library. The user is prompted to choose a file location and format to save the Excel file.. The code creates a FileOutputStream and BufferedOutputStream to write to the file. It also creates an instance of XSSFWorkbook to create the workbook and a sheet named "JTable Sheet". The data from the JTextFields is then written to the sheet by creating rows and cells and setting their values. After the data is written to the sheet, the workbook is written to the output stream, and the output streams are closed. This functionality is useful for the client as all data can be recorded in Excel format which is preferred by the client.

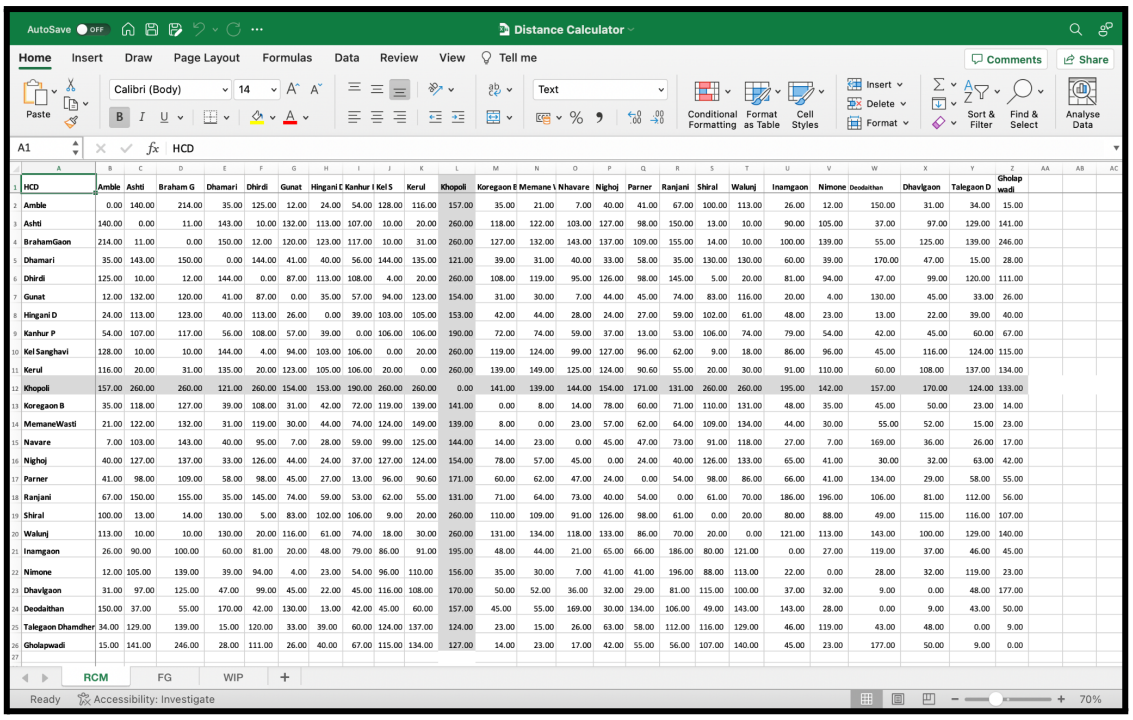
2.3.2. New Order Screen

2.3.2.1. Import Screen

```
private void jButtonImportExcelToJtableActionPerformed(java.awt.event.ActionEvent evt) {
    double distance = 0.0;
    int rows = 0;
    int columns = 0;
    String place1 = "";
    String place2 = "";
    double sum =0.0;
    String defaultCurrentDirectoryPath = "C:\\Users\\Authentic\\Desktop";
    JFileChooser excelFileChooser = new JFileChooser(defaultCurrentDirectoryPath);
    excelFileChooser.setDialogTitle("Select Excel File");
    FileNameExtensionFilter fnef = new FileNameExtensionFilter("EXCEL FILES", "xls", "xlsx", "xlsm");
    excelFileChooser.setFileFilter(fnef);
    int excelChooser = excelFileChooser.showOpenDialog(null);
    if (excelChooser == JFileChooser.APPROVE_OPTION)
    {
        try {
            excelFile = excelFileChooser.getSelectedFile();
            excelFIS = new FileInputStream(excelFile);
            excelBIS = new BufferedInputStream(excelFIS);
            excelImportToJTable = new XSSFWorkbook(excelBIS);
            XSSFSheet excelSheet = excelImportToJTable.getSheetAt(0);

            for (int row = 0 ; row <= excelSheet.getLastRowNum(); row++)
            {
                //System.out.println(excelSheet.getLastRowNum());
                XSSFRow excelRow = excelSheet.getRow(row);
                XSSFCell excelName = excelRow.getCell(0);
                String value1 = excelName.getStringCellValue();

                XSSFRow excelRow2 = excelSheet.getRow(row);
                XSSFCell excelName2 = excelRow2.getCell(1);
                String value2 = excelName2.getStringCellValue();
                    //System.out.println(value2);

                place1 = value1;
                place2 = value2;

                System.out.println(place1);
```

Fig 9: Import Button Code(Self-clicked)

This code reads data from an Excel, and then it processes the data to calculate distance between different places. The import Excel file contains place names in two columns. The code reads each row of the file using the getStringCellValue function and gets the values from the first and second columns. The place names are then used to set values for the rows and columns variables, respectively, based on a set of if-else statements that match the place names to specific integer values from the Distance Calculator Matrix which is another Excel file containing the distance data. This saves a lot of time of manual data entry for the user.



HCD	Amble	Ashti	Braham G	Dhamari	Dhirdi	Gunat	Hingani E	Kanhur I	Kel S	Kerul	Khopoli	Koregaon B	Memane \	Nhavare	Nighoj	Parner	Ranjani	Shiral	Walunj	Inamgaon	Nimone	Deodaithan	Dhavlgaon	Talegaon D	Gholap wadi
Amble	0.00	140.00	214.00	35.00	125.00	12.00	24.00	54.00	128.00	116.00	157.00	35.00	21.00	7.00	40.00	41.00	67.00	100.00	113.00	26.00	12.00	150.00	31.00	34.00	15.00
Ashti	140.00	0.00	11.00	143.00	10.00	132.00	113.00	107.00	10.00	20.00	260.00	118.00	122.00	103.00	127.00	98.00	150.00	13.00	10.00	90.00	105.00	37.00	97.00	129.00	141.00
BrahamGaon	214.00	11.00	0.00	150.00	12.00	120.00	123.00	117.00	10.00	31.00	260.00	127.00	132.00	143.00	137.00	109.00	155.00	14.00	10.00	100.00	139.00	55.00	125.00	139.00	246.00
Dhamari	35.00	143.00	150.00	0.00	144.00	41.00	40.00	56.00	144.00	135.00	121.00	39.00	31.00	40.00	33.00	58.00	35.00	130.00	130.00	60.00	39.00	170.00	47.00	15.00	28.00
Dhirdi	125.00	10.00	12.00	144.00	0.00	87.00	113.00	108.00	4.00	20.00	260.00	108.00	119.00	95.00	126.00	98.00	145.00	5.00	20.00	81.00	94.00	47.00	99.00	120.00	111.00
Gunat	12.00	132.00	120.00	41.00	87.00	0.00	35.00	57.00	94.00	123.00	154.00	31.00	30.00	7.00	44.00	45.00	74.00	83.00	116.00	20.00	4.00	130.00	45.00	33.00	26.00
Hingani D	24.00	113.00	123.00	40.00	113.00	26.00	0.00	39.00	103.00	105.00	153.00	42.00	44.00	28.00	24.00	27.00	59.00	102.00	61.00	48.00	23.00	13.00	22.00	39.00	40.00
Kanhur P	54.00	107.00	117.00	56.00	108.00	57.00	39.00	0.00	106.00	106.00	190.00	72.00	74.00	59.00	37.00	13.00	53.00	106.00	74.00	79.00	54.00	42.00	45.00	60.00	67.00
Kel Sanghavi	128.00	10.00	10.00	144.00	4.00	94.00	103.00	106.00	0.00	20.00	260.00	119.00	124.00	99.00	127.00	96.00	62.00	9.00	18.00	86.00	96.00	45.00	116.00	124.00	115.00
Kerul	116.00	20.00	31.00	135.00	20.00	123.00	105.00	106.00	20.00	0.00	260.00	139.00	149.00	125.00	124.00	90.60	55.00	20.00	30.00	91.00	110.00	60.00	108.00	137.00	134.00
Khopoli	157.00	260.00	260.00	121.00	260.00	154.00	153.00	190.00	260.00	260.00	0.00	141.00	139.00	144.00	154.00	171.00	131.00	260.00	260.00	195.00	142.00	157.00	170.00	124.00	133.00
Koregaon B	35.00	118.00	127.00	39.00	108.00	31.00	42.00	72.00	119.00	139.00	141.00	0.00	8.00	14.00	78.00	60.00	71.00	110.00	131.00	48.00	35.00	45.00	50.00	23.00	14.00
MemaneWasti	21.00	122.00	132.00	31.00	119.00	30.00	44.00	74.00	124.00	149.00	139.00	8.00	0.00	23.00	57.00	62.00	64.00	109.00	134.00	44.00	30.00	55.00	52.00	15.00	23.00
Navare	7.00	103.00	143.00	40.00	95.00	7.00	28.00	59.00	99.00	125.00	144.00	14.00	23.00	0.00	45.00	47.00	73.00	91.00	118.00	27.00	7.00	169.00	36.00	26.00	17.00
Nighoj	40.00	127.00	137.00	33.00	126.00	44.00	24.00	37.00	127.00	124.00	154.00	78.00	57.00	45.00	0.00	24.00	40.00	126.00	133.00	65.00	41.00	30.00	32.00	63.00	42.00
Parner	41.00	98.00	109.00	58.00	98.00	45.00	27.00	13.00	96.00	90.60	171.00	60.00	62.00	47.00	24.00	0.00	54.00	98.00	86.00	66.00	41.00	134.00	29.00	58.00	55.00
Ranjani	67.00	150.00	155.00	35.00	145.00	74.00	59.00	53.00	62.00	55.00	131.00	71.00	64.00	73.00	40.00	54.00	0.00	61.00	70.00	186.00	196.00	106.00	81.00	112.00	56.00
Shiral	100.00	13.00	14.00	130.00	5.00	83.00	102.00	106.00	9.00	20.00	260.00	110.00	109.00	91.00	126.00	98.00	61.00	0.00	20.00	80.00	88.00	49.00	115.00	116.00	107.00
Walunj	113.00	10.00	10.00	130.00	20.00	116.00	61.00	74.00	18.00	30.00	260.00	131.00	134.00	118.00	133.00	86.00	70.00	20.00	0.00	121.00	113.00	143.00	100.00	129.00	140.00
Inamgaon	26.00	90.00	100.00	60.00	81.00	20.00	48.00	79.00	86.00	91.00	195.00	48.00	44.00	21.00	65.00	66.00	186.00	80.00	121.00	0.00	27.00	119.00	37.00	46.00	45.00
Nimone	12.00	105.00	139.00	39.00	94.00	4.00	23.00	54.00	96.00	110.00	156.00	35.00	30.00	7.00	41.00	41.00	196.00	88.00	113.00	22.00	0.00	28.00	32.00	119.00	23.00
Dhavlgaon	31.00	97.00	125.00	47.00	99.00	45.00	22.00	45.00	116.00	108.00	170.00	50.00	52.00	36.00	32.00	29.00	81.00	115.00	100.00	37.00	32.00	9.00	0.00	48.00	177.00
Deodaithan	150.00	37.00	55.00	170.00	42.00	130.00	13.00	42.00	45.00	60.00	157.00	45.00	55.00	169.00	30.00	134.00	106.00	49.00	143.00	143.00	28.00	0.00	9.00	43.00	50.00
Talegaon Dhamdher	34.00	129.00	139.00	15.00	120.00	33.00	39.00	60.00	124.00	137.00	124.00	23.00	15.00	26.00	63.00	58.00	112.00	116.00	129.00	46.00	119.00	43.00	48.00	0.00	9.00
Gholapwadi	15.00	141.00	246.00	28.00	111.00	26.00	40.00	67.00	115.00	134.00	127.00	14.00	23.00	17.00	42.00	55.00	56.00	107.00	140.00	45.00	23.00	177.00	50.00	9.00	0.00

Fig 10: Distance Matrix Excel FIle (Self-clicked)

2.3.2.2. Export Screen

```
        try {
            //Import excel poi libraries to netbeans
            excelJTableExporter = new XSSFWorkbook();
            XSSFSheet excelSheet = excelJTableExporter.createSheet("JTable Sheet");
            //          Loop to get jtable columns and rows
            for (int i = 0; i < model.getRowCount(); i++) {
                XSSFRow excelRow = excelSheet.createRow(i);
                for (int j = 0; j < model.getColumnCount(); j++) {
                    XSSFCell excelCell = excelRow.createCell(j);

                    //Now Get ImageNames From JLabel
                    //get the last column

                    excelCell.setCellValue(model.getValueAt(i, j).toString());

//                      Change the values of the fourth column to image paths

                }
            }   //Append xlsx file extensions to selected files. To create unique file names
            excelFOU = new FileOutputStream(excelFileChooser.getSelectedFile() + ".xlsx");
            excelBOU = new BufferedOutputStream(excelFOU);
            excelJTableExporter.write(excelBOU);
            JOptionPane.showMessageDialog(null, "Exported Successfully!");
        } catch (FileNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            try {
                if (excelBOU != null) {
                    excelBOU.close();
                }
                if (excelFOU != null) {
                    excelFOU.close();
                }
                if (excelJTableExporter != null) {
                    excelJTableExporter.close();
                }
```

Fig 11: Export Button Code(Self-clicked)

This code is similar to the code to print the values in Jtextfields but instead of using the values from the text field, it uses a loop to read the values in each row and column using the getValue function and then writes the data into an Excel file exactly like in the table(Muteti, Maurice).

3. Databases

3.1. MySQL Workbench

Fig 12: Order and Employee Details are displayed using data from MySQL (Self-clicked)

The solution uses the high-performance, secure and reliable MySQL database which is ideal for the client. There are three tables - login details, order details, employee details. The order details table has a foreign key from employee details to make it such that the details employee handling the order is associated with the order details. All data can be accessed using sql statements in the java code(W3schools.com).

Word Count: 1000