Criterion C: Product Development

Techniques used to develop the product

- Creation of database – explanation and justification for structure, normalization to third normal form and event scheduler.
- Connection to the online database for easy access.
- Authentication and security – Password protected database, access levels and login form to ensure protection of sensitive data.
- Complex queries used for entering, updating, and deleting data and performing calculations to create a user-friendly product.
- Methods and algorithms used in java.
- Generating and displaying reports which can be printed or saved.
- User-friendly GUI to allow employees to work more efficiently.

Database Creation – Explanation and justification for structure

XAMPP[1], a free, open-source web solution, was used to create and host the MYSQL database. The database is in 3NF which reduces redundancy and repetition of the data. It is also important as updated data is cascaded to other tables, and deletion of data is restricted to maintain data integrity. Here are the relationships:

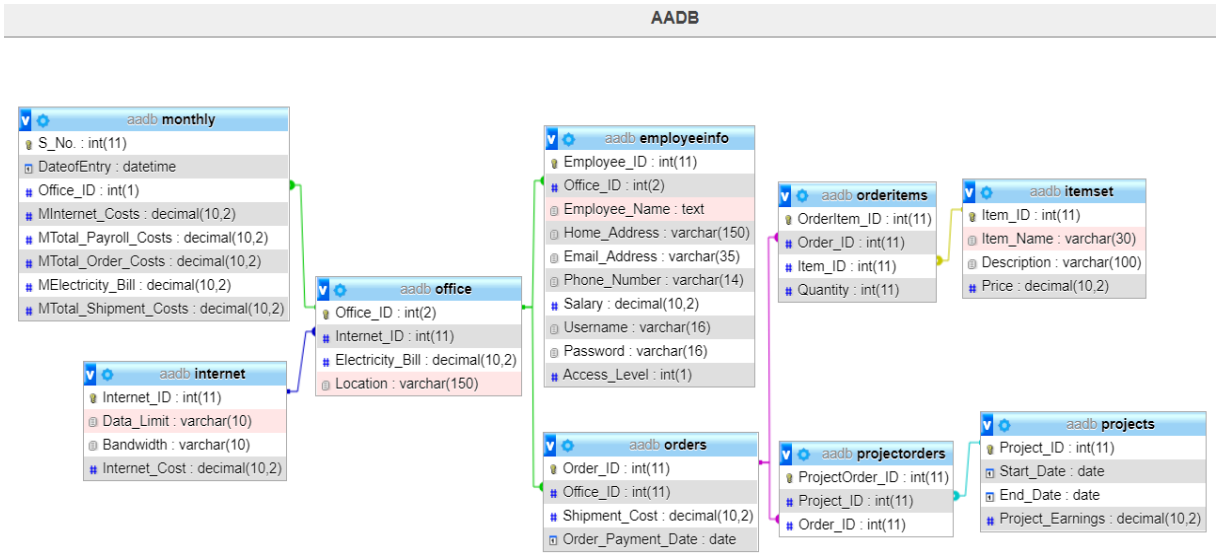


Figure 1: Tables and their relationships in

The orderitems and projectorders table are important to the structure as they allow the employees to assign multiple items to an order and multiple orders to a project. It is not possible to create a many-to-many relationship in the database and thus I had to create these two tables to connect projects and itemset tables to the orders table. The monthly table is necessary for the creation of the monthly expenses report.

[1] *XAMPP Tutorial - javatpoint*. (n.d.). Www.Javatpoint.Com. https://www.javatpoint.com/xampp

Every month, all the office expenses for that month are entered into the monthly table using an event scheduler:

It is important as it can store data which may be lost, for example: an employee's salary can change and thus the previous month's salary would be lost.

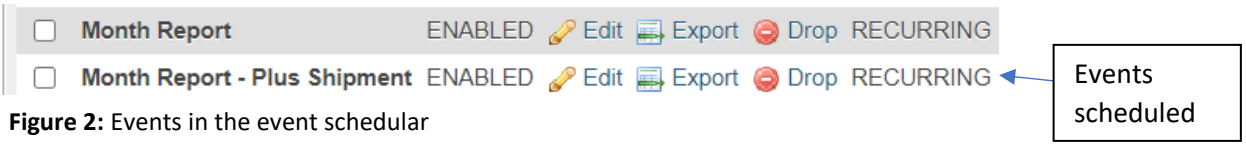


Figure 2: Events in the event schedular

Connection to the online database

One feature which was requested by the client was to make the database accessible through the internet. To set up the online connection, a new user had to be created which had all privileges granted. Therefore, I created aauser with a password for security:

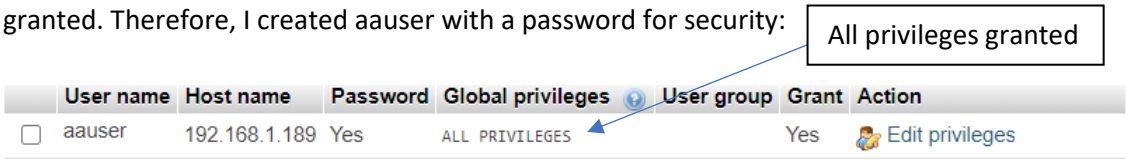


Figure 3: XAMPP hosting form

The MySQL Database was hosted on the port 3306, which acts as identifier for the database, as seen below on the XAMPP control panel:

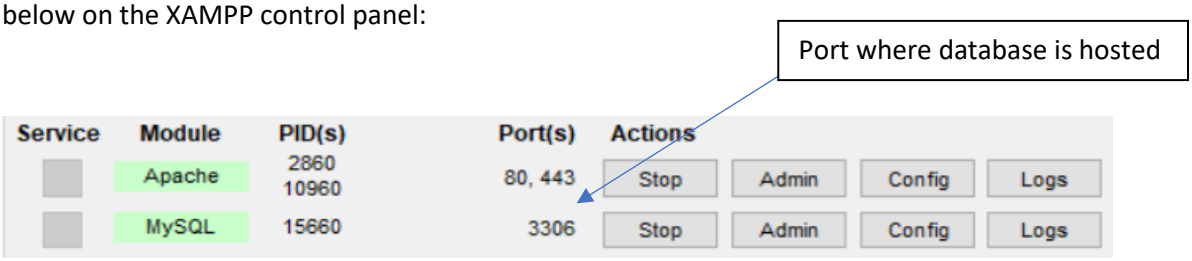
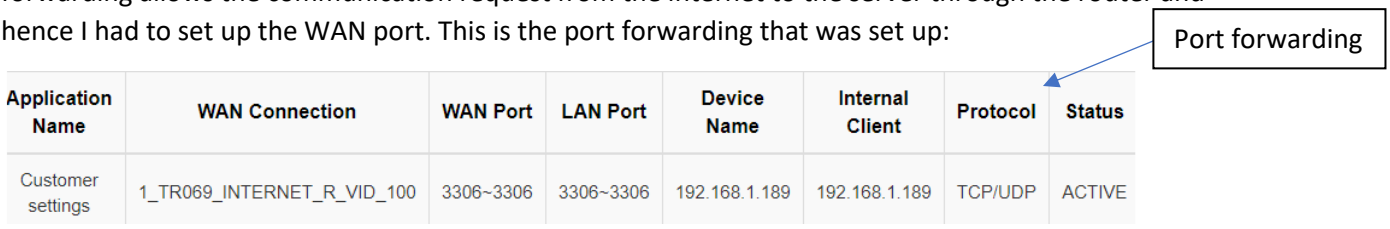


Figure 4: XAMPP control panel

The port forwarding[2] had to be set up to allow access from outside the local area network. The port forwarding allows the communication request from the internet to the server through the router and hence I had to set up the WAN port. This is the port forwarding that was set up:



Application Name	WAN Connection	WAN Port	LAN Port	Device Name	Internal Client	Protocol	Status
Customer settings	1_TR069_INTERNET_R_VID_100	3306~3306	3306~3306	192.168.1.189	192.168.1.189	TCP/UDP	ACTIVE

Figure 5: Router port forwarding table

Variables were created to simplify the java code for establishing the connection with the database:

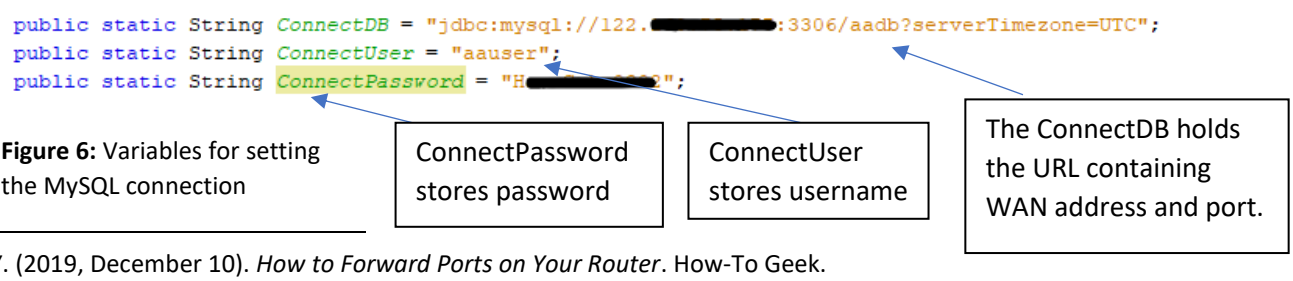


Figure 6: Variables for setting the MySQL connection

[2]Y. (2019, December 10). *How to Forward Ports on Your Router*. How-To Geek. https://www.howtogeek.com/66214/how-to-forward-ports-on-your-router/

A try-catch statement was setup to finally establish the connection using the MySQL java connector:

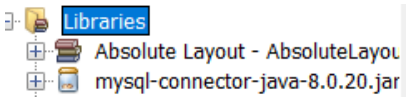


Figure 7: NetBeans project libraries

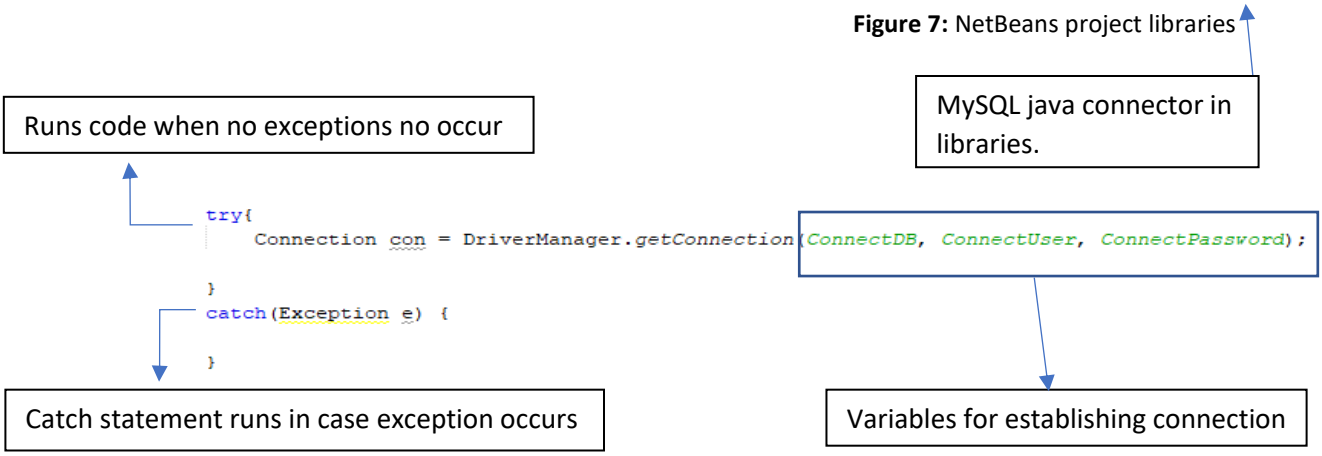


Figure 8: NetBeans code for MySQL connection

This is the general outline of the java code where I can use predefined objects to execute statements.

Authentication and security

Security is another essential requirement as the company contains sensitive information which needs to be safeguarded. A login page was setup which requires authentication before they are granted access to the GUI. The user is required to enter their username and password in the login page:

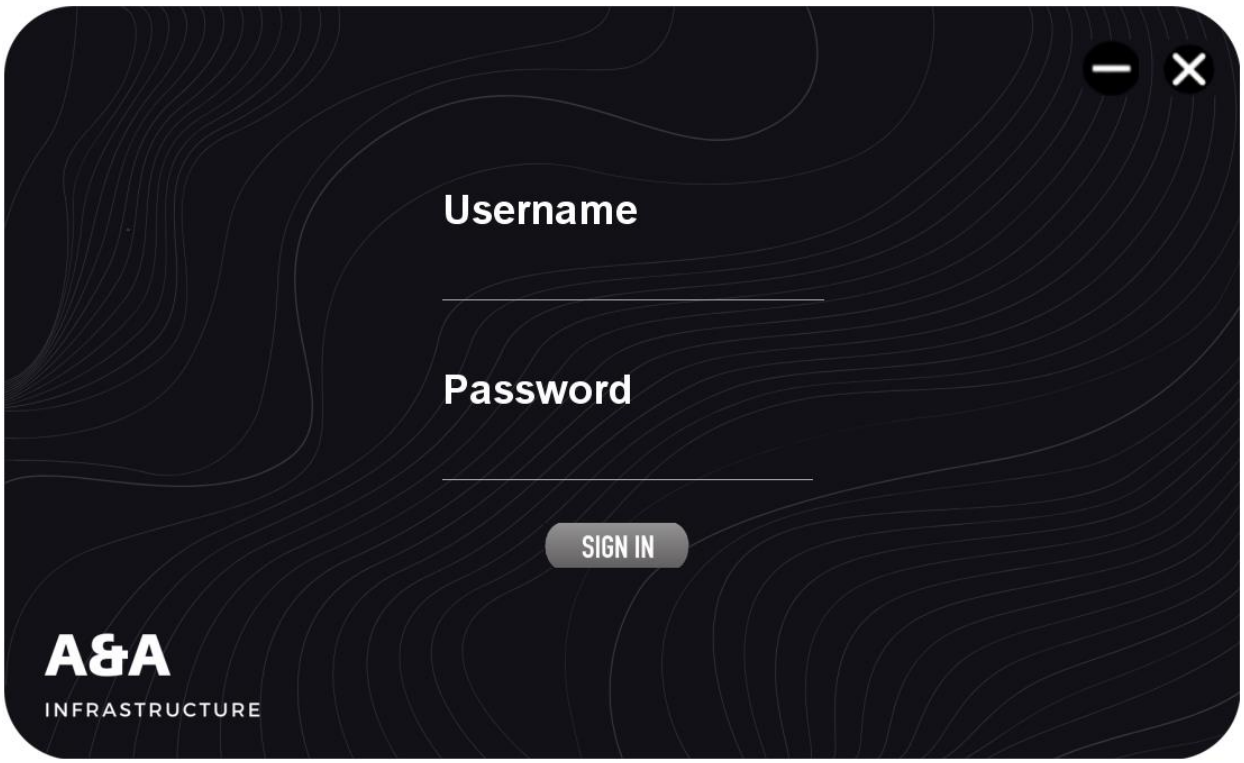


Figure 9: Login interface

In case the password or username is left blank or entered incorrectly, the appropriate error message will be shown:

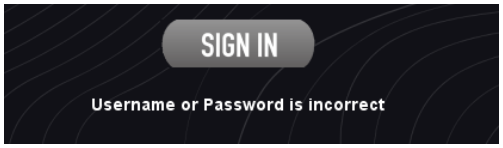


Figure 10: Incorrect data entered in the login page

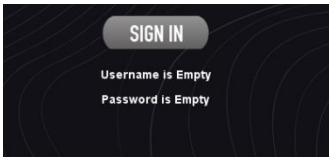


Figure 11: Missing data in the login page

The username and password for each employee is stored in the employeeinfo table of the database:

Employee_Name	Username	Password	Access_Level
Sumer Kaistha	SumerK2002	[redacted]	1
Dinesh Aggarwal	Dinesh1968	[redacted]	2
Anuj Kaistha	K. Anuj	[redacted]	1
Ayush Sharma	AyushS	[redacted]	2

Figure 12: Employeeinfo Table of the database

The access levels are further used to restrict access to certain features. Access level '1' is given to only the administrators who are given full access to all features which involve viewing a list of the employees and the office expenses. Access level '2' is given to the employees which allows them to only use the necessary features:

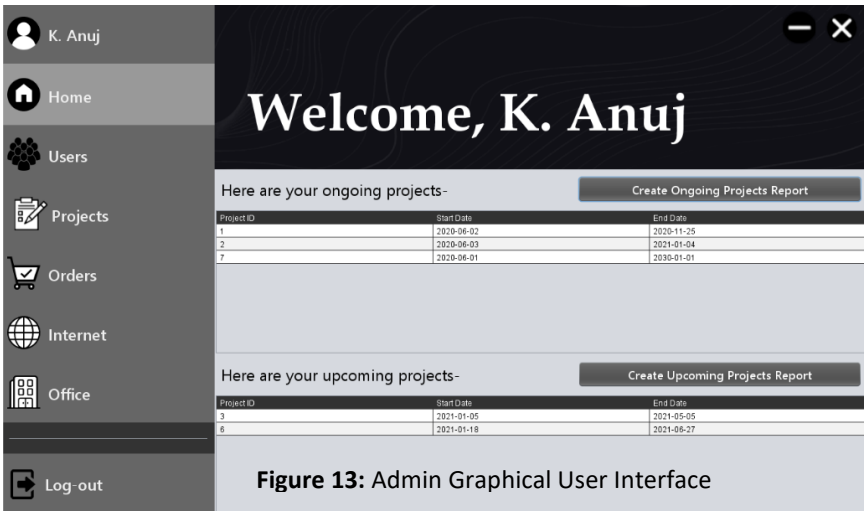
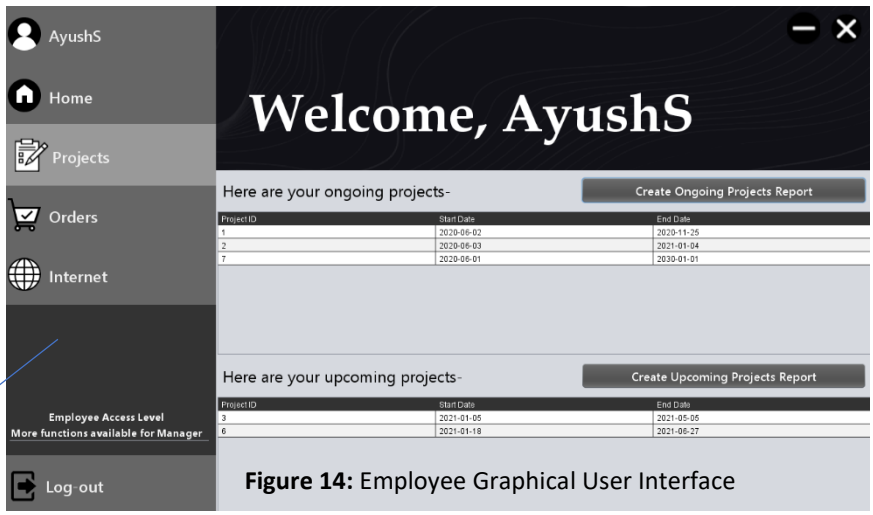


Figure 13: Admin Graphical User Interface



Users and Office tabs removed

Figure 14: Employee Graphical User Interface

The SQL statement retrieves the username and password from the database and compares it with the entered username and password. If it matches, the user is sent to the GUI according to their access level:

```
public static String AccessLevelFlag;
```

Stores the access level of the user

Figure 15: Variable for storing access level

```
if(InputUsername.getText().trim().isEmpty() && InputPassword.getText().trim().isEmpty()){
    IncorrectUsername.setText("Username is Empty");
    IncorrectPassword.setText("Password is Empty");
}
else if(InputUsername.getText().trim().isEmpty()){
    IncorrectUsername.setText("Username is Empty");
}
else if (InputPassword.getText().trim().isEmpty()){
    IncorrectPassword.setText("Password is Empty");
}
else{
    try{
        Connection con = DriverManager.getConnection(ConnectDB, ConnectUser, ConnectPassword);
        String sql = "Select Employee_ID, Username, Access_Level from employeeinfo where username=? and password=?";
        PreparedStatement pst = con.prepareStatement(sql);
        pst.setString(1, InputUsername.getText());
        pst.setString(2, InputPassword.getText());
        ResultSet rs = pst.executeQuery();
        if(rs.next()){
            EID = rs.getString(1);
            UID = rs.getString(2);
            AccessLevelFlag = rs.getString(3);
            if("1".equals(AccessLevelFlag)){
            Main_Menu menu = new Main_Menu();
            menu.setVisible(true);
            dispose();
            }else{
            Main_Menu_Employee Emenu = new Main_Menu_Employee();
            Emenu.setVisible(true);
            dispose();
            }
        }
        else{
            IncorrectUsername.setText("Username or Password is incorrect");
            InputPassword.setText("");
        }
        con.close();
    }
    catch(Exception e) {
        JOptionPane.showMessageDialog(null,e);
    }
```

Displays appropriate error when input is empty

Question marks replaced in order with details entered

Checks entered details

Appropriate main menu interface opened according to access level

Error message displayed if details are incorrect

Figure 16: Code for user authentication and login

Complex Queries

A complex query[3] is used to enter, update, delete and calculate data.

1) Entering Data:
   Upon clicking the add button, the data entry form is made visible:

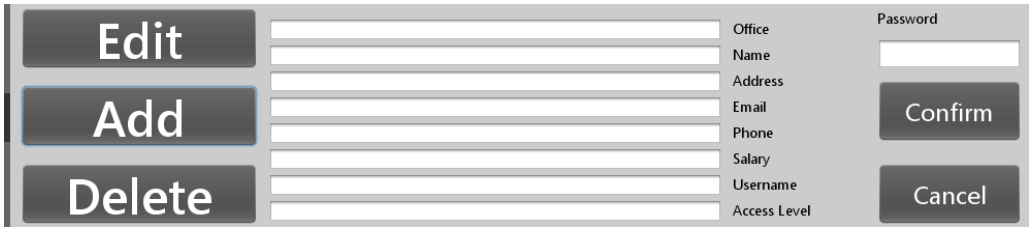


Figure 17: Data entry form

[3] *MySQL Queries - javatpoint. (n.d.). Javatpoint. https://www.javatpoint.com/mysql-queries*

Pressing confirm enters the data into the database:

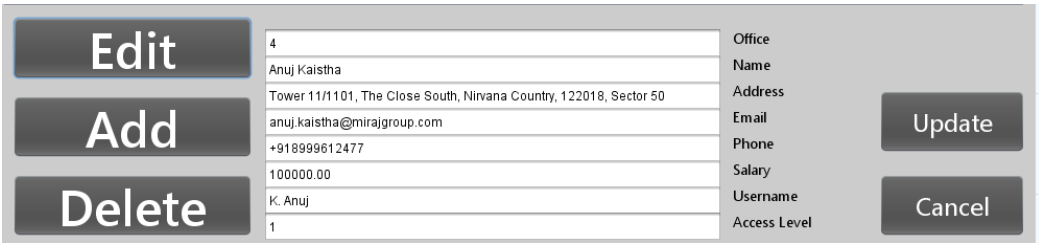
```
try{
    Connection con = DriverManager.getConnection(ConnectDB, ConnectUser, ConnectPassword);
    String sql = "insert into employeeinfo (Office_ID, Employee_Name,"
            + " Home_Address, Email_Address, Phone_Number, Salary, "
            + ", Access_Level, Password) values (?,?,?,?,?,?,?,?,?)";
    PreparedStatement pst = con.prepareStatement(sql);
    pst.setString(1, OfficeTF.getText());
    pst.setString(2, NameTF.getText());
    pst.setString(3, AddressTF.getText());
    pst.setString(4, EmailTF.getText());
    pst.setString(5, PhoneTF.getText());
    pst.setString(6, SalaryTF.getText());
    pst.setString(7, UsernameTF.getText());
    pst.setString(8, AccessLevelTF.getText());
    pst.setString(9, PasswordPF.getText());
    int rs = pst.executeUpdate();
    if(rs == 1) {
        JOptionPane.showMessageDialog(null, "Updated Database Successfully.");
    } else {
        JOptionPane.showMessageDialog(null, "Updating Database Failed.");
    }
}
catch(Exception e){
    JOptionPane.showMessageDialog(null, e);
}
```

Query to insert new record

Appropriate message displayed

Figure 18: Add Button code

2) Updating Data:
Clicking the edit button opens up the data entry form:



The row from the table presented in the GUI which contains the data to be modified must be selected:

ID	Office	Name	Address	Email	Phone	Salary (Rs)	Username	Access Level
1	1	Sumer Kaistha	[redacted]					1
2	2	Dinesh Aggarwal	[redacted]					2
3	3	Ayush Sharma	[redacted]					2
15	4	Anuj Kaistha	[redacted]					1

Figure 19: Employeeinfo table in User page

Selected row

The update button will confirm the modifications made to the data of the database:

```
try{
    int i = UsersTable.getSelectedRow();
    TableModel model = UsersTable.getModel();
    Connection con = DriverManager.getConnection(ConnectDB, ConnectUser, ConnectPassword
    String sql = "update employeeinfo set Office_ID=?, Employee_Name=?,"
            + " Home_Address=?, Email_Address=?, Phone_Number=?, Salary=?,"
            + " Username=?, Access_Level=? where Employee_ID=?";
    PreparedStatement pst = con.prepareStatement(sql);
    pst.setString(1, OfficeTF.getText());
    pst.setString(2, NameTF.getText());
    pst.setString(3, AddressTF.getText());
    pst.setString(4, EmailTF.getText());
    pst.setString(5, PhoneTF.getText());
    pst.setString(6, SalaryTF.getText());
    pst.setString(7, UsernameTF.getText());
    pst.setString(8, AccessLevelTF.getText());
    pst.setString(9, model.getValueAt(i, 0).toString());
    int rs = pst.executeUpdate();
    if(rs == 1) {
        JOptionPane.showMessageDialog(null, "Database Updated.");
    } else {
        JOptionPane.showMessageDialog(null, "Database Update Failed.");
    }
}
catch(Exception e){
    JOptionPane.showMessageDialog(null, e);
}
```

Query to update database

Figure 20: Code for Update Button

3) Deleting Data:

Selecting a row and pressing the delete button will remove that data from the database as long as it does not break the referential integrity of the data:

ID	Office	Name	Address	Email	Phone	Salary (Rs)	Username	Access Level
1	1	Sumer Kaistha	[redacted]					1
2	2	Dinesh Aggarwal	[redacted]					2
3	3	Ayush Sharma	[redacted]					2
15	4	Anuj Kaistha	[redacted]					1

```
String ID_to_be_deleted = "";
try {
    int i = UsersTable.getSelectedRow();
    TableModel model = UsersTable.getModel();
    ID_to_be_deleted = model.getValueAt(i, 0).toString();

    if(ID_to_be_deleted.equals(EID)) {
        JOptionPane.showMessageDialog(null, "You cannot delete your own account.");
    } else {
        Connection con = DriverManager.getConnection(ConnectDB, ConnectUser, ConnectPassword);
        String sql = "delete from employeeinfo where Employee_ID=?";
        PreparedStatement pst = con.prepareStatement(sql);
        pst.setString(1, ID_to_be_deleted);
        int rs = pst.executeUpdate();
        if(rs == 1) {
            JOptionPane.showMessageDialog(null, "Row Deleted.");
        } else {
            JOptionPane.showMessageDialog(null, "Row Deletion Failed.");
        }
    }
}
catch(Exception e){
    JOptionPane.showMessageDialog(null, e.getMessage());
}
```

Query to delete record

Figure 21: Code for delete button

4) Calculating expenses:

The use of complex queries allows the calculations for the total office expenses. All expenses are calculated through this query:

```
Statement st = con.createStatement();
String s = "SELECT office.Office_ID, Internet_Cost, SUM(Salary) as Total_Payroll_Costs,"
        + " SUM((Quantity*Price)) as Total_Order_Costs, Electricity_Bill, location FROM office,"
        + " employeeinfo, internet, itemset, orders, orderitems WHERE orders.Office_ID = office.Office_ID AND"
        + " orderitems.Order_ID = orders.Order_ID AND orderitems.Item_ID = itemset.Item_ID AND"
        + " employeeinfo.Office_ID = office.Office_ID AND office.Internet_ID = internet.Internet_ID AND"
        + " Order_Payment_Date <= CURRENT_DATE AND MONTH(CURRENT_DATE) = MONTH(Order_Payment_Date) AND"
        + " YEAR(CURRENT_DATE) = YEAR(Order_Payment_Date) GROUP BY office.Office_ID";
ResultSet rs = st.executeQuery(s);
```

Query to calculate all expenses

Figure 22: Code for calculation of total office expenses

Shipment is calculated separately through this query which is then added onto the total order costs to display the final total orders costs to the GUI:

```
String z = "SELECT SUM(Shipment_Cost) as TSC FROM orders, office WHERE office.Office_ID = orders.Office_ID AND"
        + " Order_Payment_Date <= CURRENT_DATE AND MONTH(CURRENT_DATE) = MONTH(Order_Payment_Date) AND YEAR(CURRENT_D
        + " = YEAR(Order_Payment_Date) GROUP BY office.Office_ID";
ResultSet rs = st.executeQuery(z);
```

Query to calculate shipment which is added to total order

Figure 23: Code for calculation of shipment costs

5) Event scheduler:

Every month, the event schedular needs to execute a query to enter data into 'monthly' table. The schedular executes the query every minute however, the WHERE statement in the query ensures it only occurs at the last minute of the end of the month. All office expenses are calculated and inserted into 'monthly' table:

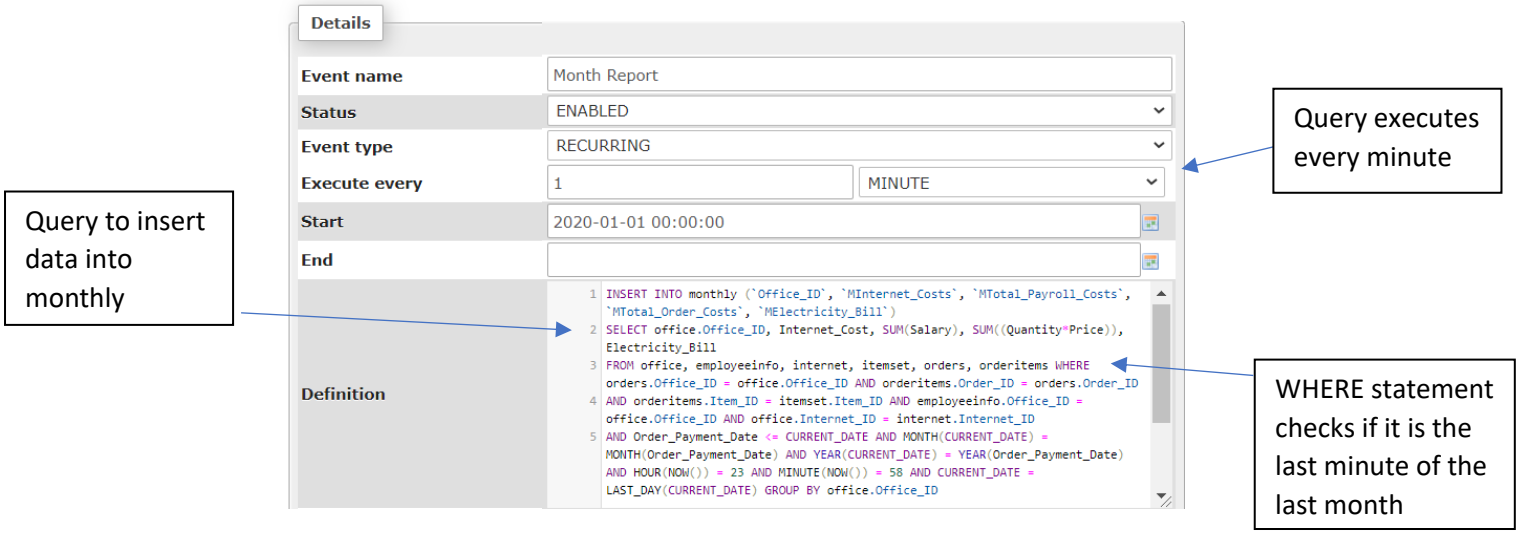


Figure 24: Code to insert office expenses into monthly table

The total shipment cost is calculated separately and entered as it is important to show each individual cost in the monthly and yearly expenses report:

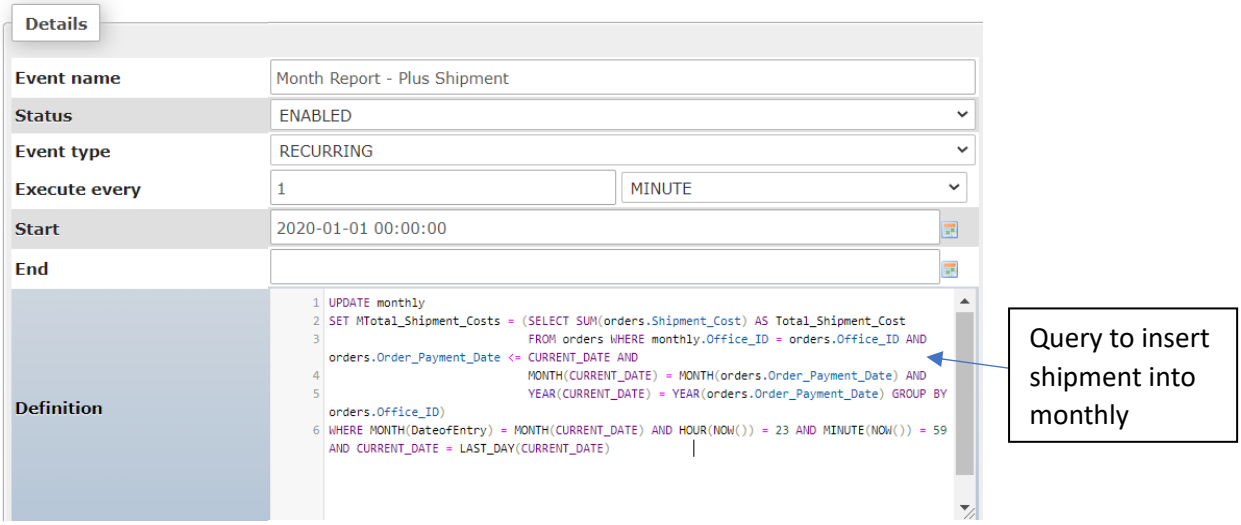


Figure 25: Code to insert total shipment cost into monthly table

Methods and Algorithms

This technique displays the data from the database[4]. The data is input into a Jtables which then show the necessary information:

Jtable to display data

OrderItemID	Order ID	Item ID	Quantity	Total Price (Rs. Not Including Shipment)
1	1	1010	250	1250.00
2	2	1110	150	1200.00
4	2	1010	500	2500.00
7	3	2010	150	6000.00
8	4	1110	100	800.00
9	10	2010	250	10000.00
10	11	1010	1000	5000.00
11	15	1010	12	60.00

Figure 26: Jtable which shows data collected from the database

[4] *JTable Display Data From MySQL Database*. (n.d.). Rose India. https://www.roseindia.net/java/example/java/swing/jtable-display-database-data.shtml

'The class, 'UserPage', is where the table and data entry form are displayed. The variables are defined in a new class, 'User':

```
class User {
    private int Employee_ID, Office_ID, Access_Level;
    private String Employee_Name, Username,Home_Address, Email_Address, Phone_Number;
    private BigDecimal Salary;
    public User(int Employee_ID, int Office_ID, int Access_Level,
            String Employee_Name, String Username, String Home_Address,
            String Email_Address, String Phone_Number, BigDecimal Salary){
        this.Employee_ID = Employee_ID;
        this.Employee_Name = Employee_Name;
        this.Username = Username;
        this.Email_Address = Email_Address;
        this.Phone_Number = Phone_Number;
        this.Office_ID = Office_ID;
        this.Access_Level = Access_Level;
        this.Salary = Salary;
        this.Home_Address = Home_Address;
    }
    public int getEmployee_ID() {
        return Employee_ID;
    }
    public String getEmployee_Name() {
        return Employee_Name;
    }
    public String getUsername() {
        return Username;
    }
    public String getEmail_Address() {
        return Email_Address;
    }
    public String getPhone_Number() {
        return Phone_Number;
    }
    public BigDecimal getSalary() {
        return Salary;
    }
    public int getAccess_Level() {
        return Access_Level;
    }
    public String getHome_Address() {
        return Home_Address;
    }
    public int getOffice_ID() {
        return Office_ID;
    }
```

Variables for table are declared

User constructor is defined with

Variables outside the constructor are initialized with variables inside

Method used to return the value of the variables outside

Figure 27: Data types, functions, and fields code for User class

The methods inside are used to return the variables when they are executed. An ArrayList is declared and initialized to store the data which has to be displayed in the table. The variables are now assigned values from the database using a query which are then stored in the array:

```
public ArrayList<User> getUsersList(){
        ArrayList<User> UL = new ArrayList<>();
        try{
            Connection con = DriverManager.getConnection(ConnectDB, ConnectUser, ConnectPassword);
            Statement st = con.createStatement();
            String s = "SELECT * FROM employeeinfo";
            ResultSet rs = st.executeQuery(s);
            User user;
            while(rs.next()){
                user = new User(rs.getInt("Employee_ID"), rs.getInt("Office_ID"), rs.getInt("Access_Level"),
                        rs.getString("Employee_Name"), rs.getString("Username"), rs.getString("Home_Address"),
                        rs.getString("Email_Address"), rs.getString("Phone_Number"), rs.getBigDecimal("Salary"));
                UL.add(user);
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
        return UL;
    }
```

User constructor is called and query assigns values to variables from the database

Variables added to ArrayList in getUsersList method

Figure 28: Code for extracting data from the database

Another method, 'ShowTable', is now defined to loop through the ArrayList and enter the data for each object into the table until there are no more values in the array

```
public void ShowTable(){
    ArrayList<User> list = getUsersList();
    DefaultTableModel model = (DefaultTableModel)UsersTable.getModel();
    model.setRowCount(0);
    Object[] row = new Object[9];
    for(int i = 0; i<list.size(); i++){
        row[0] = list.get(i).getEmployee_ID();
        row[1] = list.get(i).getOffice_ID();
        row[2] = list.get(i).getEmployee_Name();
        row[3] = list.get(i).getHome_Address();
        row[4] = list.get(i).getEmail_Address();
        row[5] = list.get(i).getPhone_Number();
        row[6] = list.get(i).getSalary();
        row[7] = list.get(i).getUsername();
        row[8] = list.get(i).getAccess_Level();
        model.addRow(row);
    }
```

Inside ShowTable method, ArrayList 'list' is set by getUsersList method.

Number of columns defined

Loops till all rows added

Inputs data into the object 'row' column by

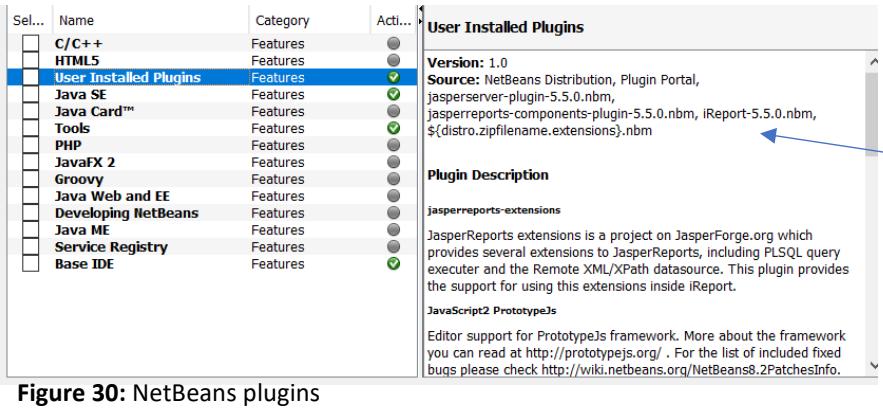
Object 'row' is added to create a row for the table

Figure 29: Code for inserting data into the Jtable

Reports

The client requires detailed reports that he can easily keep track of the scheduled projects, upcoming orders and most importantly, the office expenses for every month, which posed the greatest problems to him.

JasperReports[5] and iReport[6] were the NetBeans plugins used. These plugins are for generating reports in java which can be then printed or downloaded. The plugins were downloaded and installed:



iReport and JasperReports plugins installed and activated

Figure 30: NetBeans plugins

[5] *JasperReports® Library*. (n.d.). Jaspersoft Community. https://community.jaspersoft.com/project/jasperreports-library

[6] *iReport - NetBeans Plugin detail*. (n.d.). NetBeans. http://plugins.netbeans.org/plugin/4425/ireport

JasperReports is used to set a query to calculate the necessary data and then compile and display the report. The code shown first is used to automatically set the file path. During compilation of code, some characters were added to the path which are removed. Reports folder contains all reports. This code is important for the generation of reports on different computers.

```
try{
    con = DriverManager.getConnection(ConnectDB, ConnectUser, ConnectPassword);
    String a = ""+getClass().getResource("UpcomingProjectsReport.jrxml")+"";
    String b = a.substring(10, a.length() - 44);
    String c = b + "Reports/reportorders.jrxml";
    JasperDesign jdesign = JRXmlLoader.load(c);
```

Report file path added

JasperReports accesses file path

Unnecessary characters are removed

File path is automatically set

Figure 31: File path identifier code

The reports are generated when the 'report button' is pressed. The query for the upcoming orders report takes the order payment date as a filter and creates a report.

```
    JasperDesign jdesign = JRXmlLoader.load(c);
    String query =  "SELECT\n" +
    "     (quantity*price) AS Order_Cost,\n" +
    "     itemset.`Item_ID` AS itemset_Item_ID,\n" +
    "     itemset.`Item_Name` AS itemset_Item_Name,\n" +
    "     orders.`Office_ID` AS orders_Office_ID,\n" +
    "     orders.`Order_ID` AS orders_Order_ID,\n" +
    "     orders.`Shipment_Cost` AS orders_Shipment_Cost,\n" +
    "     orders.`Order_Payment_Date` AS orders_Order_Payment_Date\n" +
    "FROM\n" +
    "     `orders` orders INNER JOIN `orderitems` orderitems ON orders.`Order_ID` = orderitems.`Order_ID`\n" +
    "     INNER JOIN `itemset` itemset ON orderitems.`Item_ID` = itemset.`Item_ID`\n" +
    "WHERE\n" +
    "     CURRENT_DATE < Order_Payment_Date";
    JRDesignQuery updateQuery = new JRDesignQuery();
    updateQuery.setText(query);

    jdesign.setQuery(updateQuery);

    JasperReport jreport = JasperCompileManager.compileReport(jdesign);
    JasperPrint jprint = JasperFillManager.fillReport(jreport, null, con);

    JasperViewer jv = new JasperViewer(jprint, false);
    jv.setIconImage(null);
    jv.setTitle("A&A Upcoming Orders");
    jv.setVisible(true);

} catch (SQLException ex) {
    Logger.getLogger(OrderItemPage.class.getName()).log(Level.SEVERE, null, ex);
} catch (JRException ex) {
    Logger.getLogger(OrderItemPage.class.getName()).log(Level.SEVERE, null, ex);
}
```

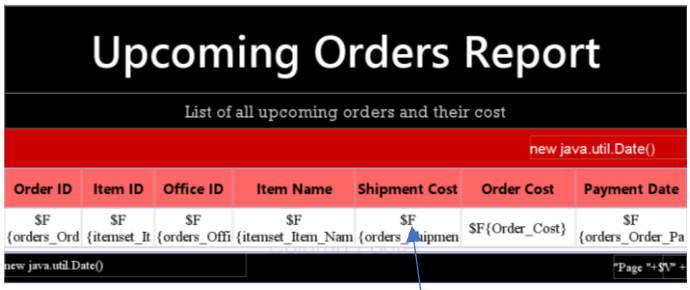
Query used to build the report

Report is compiled and filled

Title and icon of report is set, and report is displayed

Figure 32: Upcoming Orders Report code

The design of the report makes use of the functions to display the final product:



Design view of report

Function to display values from query

Figure 33: JasperReports Design View

Final report generated

Figure 34: Preview of the report

User-friendly, intuitive GUI and Extensible Code

The client requested for an easy and intuitive interface. These features help create a user-friendly user interface:

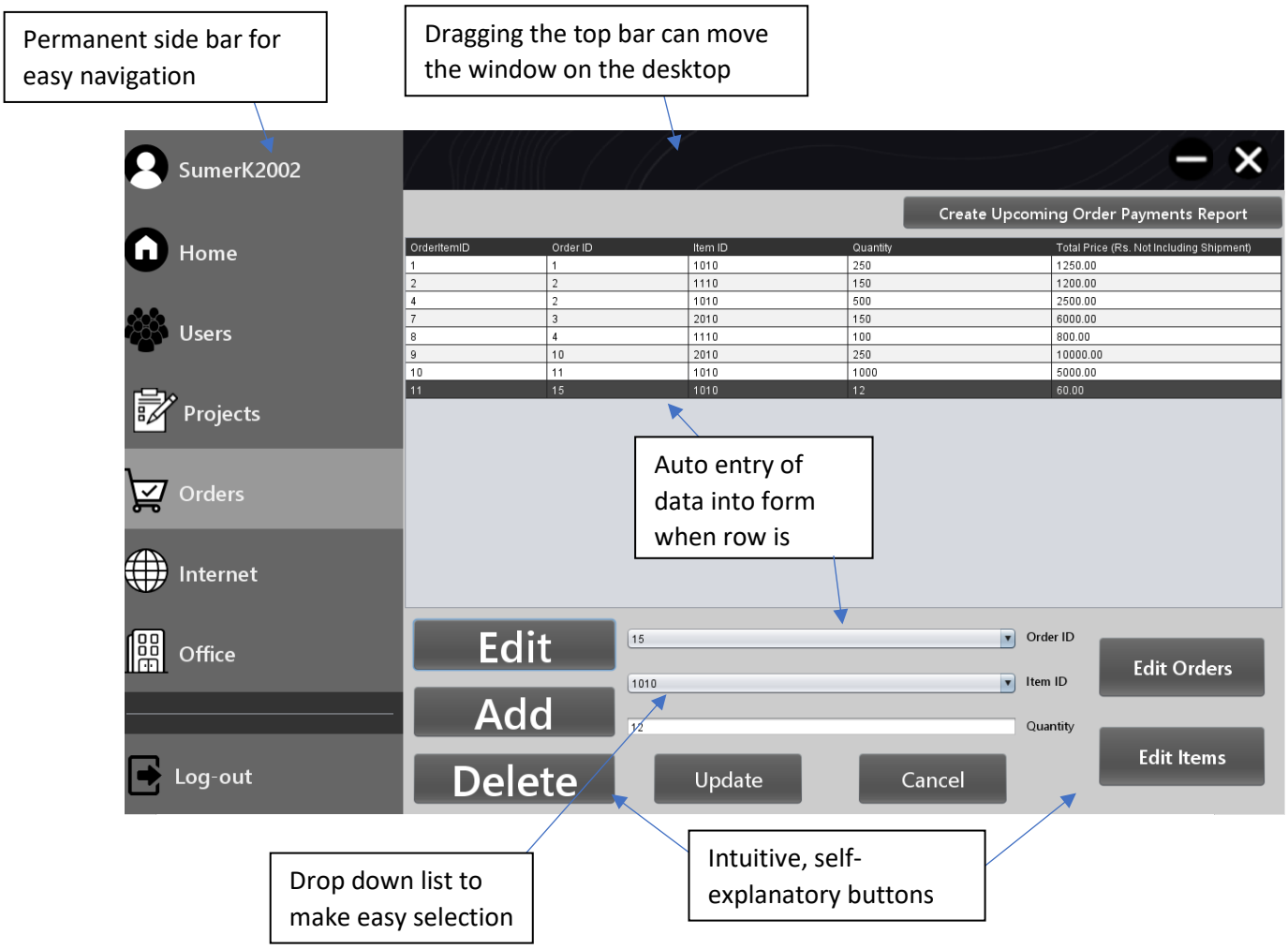


Figure 35: Orders page with the data entry form open

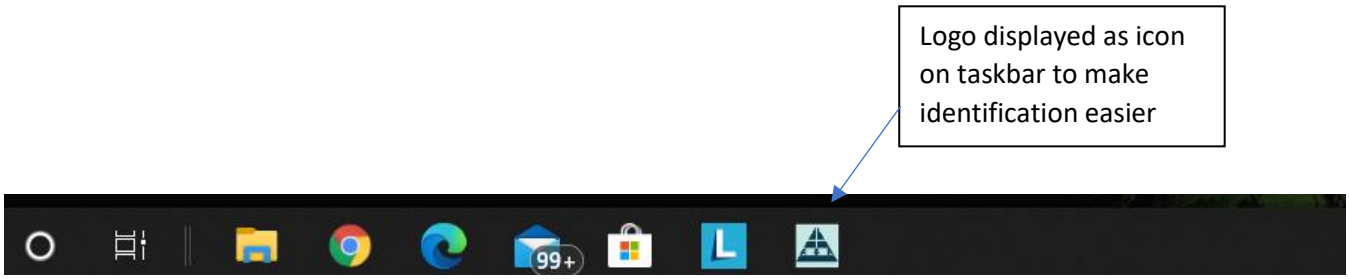


Figure 36: Task bar of computer

Lastly, the product was made extensible by adding comments when a cursor hovers over the code, making it easy to follow and modify.

Figure 37: Pop up comment when hovering over code

Word Count: 1211