SMART CANTEEN

1. INTRODUCTION

1.1 PROBLEM DEFINITION

It might have happened so many times that we cannot really come out of our classroom and buy food from the canteen or wait for your turn to order in very long queues, well not only in this case but also for the lack of time issues, this system helps a lot since it is online and automated. Select the food item and the required quantity and just place the order. It saves time, energy and is easy to operate. Each individual student gets their own Login Id (USN) and password which helps with the privacy and maintain detailed record.

1.2 OBJECTIVES

The main objectives of the project are:

- ✓ To provide user friendly interface.
- ✓ To help the students to order food from college canteen without any problems.
- ✓ To help reduce time spent in long queue and wait for your coupons until the payment is done.

1.3 METHODOLOGY TO BE FOLLOWED

The project comprises of core Java which makes the project platform independent and hence increasing its usability. This project uses JavaFX to provide a user friendly Interface to interact with the system. The project uses a database system to store all the data and information. MySQL Database is used for all the database operations and storing the data. The project greets an already existing user with a login page and also provides an option to sign up for new users. The admin module allows the user to alter the inventory and the database. The

admin can add, remove or edit food details for the event as per the need whereas the other users can place order, view bill and order details and also change their password.

1.4 EXPECTED OUTCOMES

The following outcomes are expected from this project:

- ✓ Using this project, the digitalization of college canteen is possible.
- ✓ The Smart Canteen management system will help the admin to manage his/her canteen business in an efficient and time-saving way.
- ✓ The students can manage their resources and orders in an organized manner without having to wait in long queues to order and do payment.
- ✓ All the details will be stored in the database system.
- ✓ It provides a simple and user friendly interface which helps the canteen owner as well as the students to use online mode rather than offline mode.

1.5 HARDWARE REQUIREMENTS AND SOFTWARE REQUIREMENTS

Hardware Requirements:

- Processor : 1 gigahertz (GHz) or faster processor or SoC
- Hard Disk : 8 GB for 32-bit OS or 12 GB for 64-bit OS
- RAM : 1 gigabyte (GB) for 32-bit or 2 GB for 64-bit

Software Requirements:

- Operating System : Windows 10
- Software : Any Java IDE like NetBeans, IntelliJ or Eclipse Database system and MySQL

2. FUNDAMENTALS OF JAVA PROGRAMMING

2.1 INTRODUCTION

Java is a programming language related to C++, which itself is a direct descendant of C programming language. Java therefor inherits a lot from these two languages. It can be said that from C, Java derives its syntax while many of Java's object-oriented features were influenced by C++.

Also, the creation of Java in itself was deeply rooted in the process of refinement and adaptation occurring in programming languages for the past several decades, driven by the need to solve a fundamental problem that the preceding languages could not solve.

2.2 CLASS

Any concept to be implemented in a Java program must be encapsulated within a class. A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type which means we can create multiple objects from a class. A class is just a logical entity and does not occupy any space/memory. It also lets us define the complete syntax for handling encapsulation, abstraction, polymorphism, and inheritance.

A class is declared by use of the class keyword. An example of a class is shown below

```
public class ClassName{
String name;
 int age;
void display(){
//method body;
}
 }
```

The data / variables, defined within a class are called instance variables. The code is contained within methods and collectively, the methods and variables defined within a class are called members of the class.

In general, class declarations can include these components, in order:

1. **Modifiers**: A class can be public or has default access.
2. **class keyword:** class keyword is used to create a class.
3. **Class name:** The name should begin with an initial letter (capitalized by convention).
4. **Superclass(if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
5. **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
6. **Body:** The class body surrounded by braces, { }.

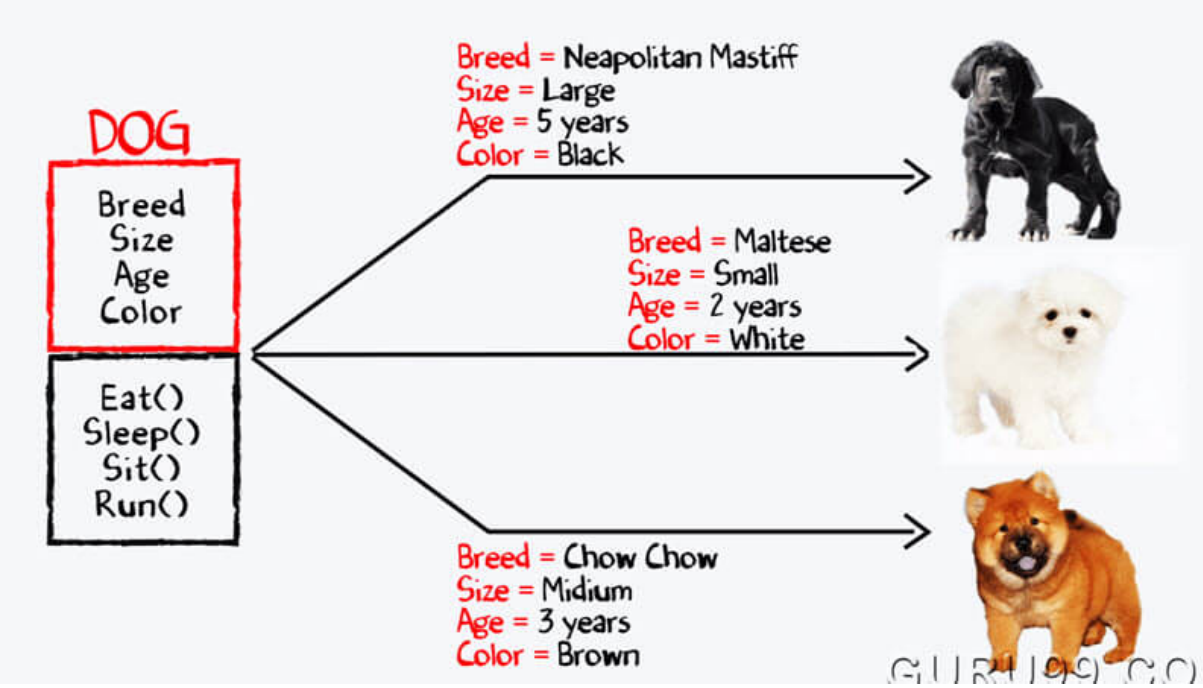


Fig. 2.2.1

2.3 OBJECT

When we create a class in Java, we are actually creating a new data type. This type can then be used to declare objects of that type. Obtaining objects of a class is however a two-step process.

First, we must declare a variable of the class type. This variable does not define an object, instead, it is simply a variable that can refer to an object.

Second, we must acquire an actual, physical copy of the object and assign it to that variable. This can be done using a new operator.

The new operator dynamically allocates (at run time) memory for an object and returns a reference to it. This reference is then stored in the variable. Thus, in Java, all class objects must be dynamically allocated. An object consists of:

Fig. 2.3.1

When an object of a class is created, the class is said to be instantiated and all the instances share the attributes and the behaviour of the class. However, the values of those attributes, i.e. the state are unique for each object. A single class may thus have any number of instances.

2.4 INHERITANCE

Inheritance in Java is a mechanism which allows one object to obtain / acquire all the properties and behaviours of another object. It is one of the cornerstones of object-oriented programming as it allows the creation of hierarchical classifications.

Using inheritance, we can create a general class that defines the traits common to a set of related items. This general class can then be inherited by other, more specific classes, each adding those things that are unique to them.

A class that is inherited is called a superclass/parent class and the class that does the inheriting is called a subclass/child class. A subclass is thus a specialized version of a superclass. It inherits all of the instance variables and methods defined by the superclass and adds its own, unique elements.

For example,

```
class Subclass-name extends Superclass-name
{
    //methods and fields
}
```

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical. Java does not support the inheriting of multiple classes.

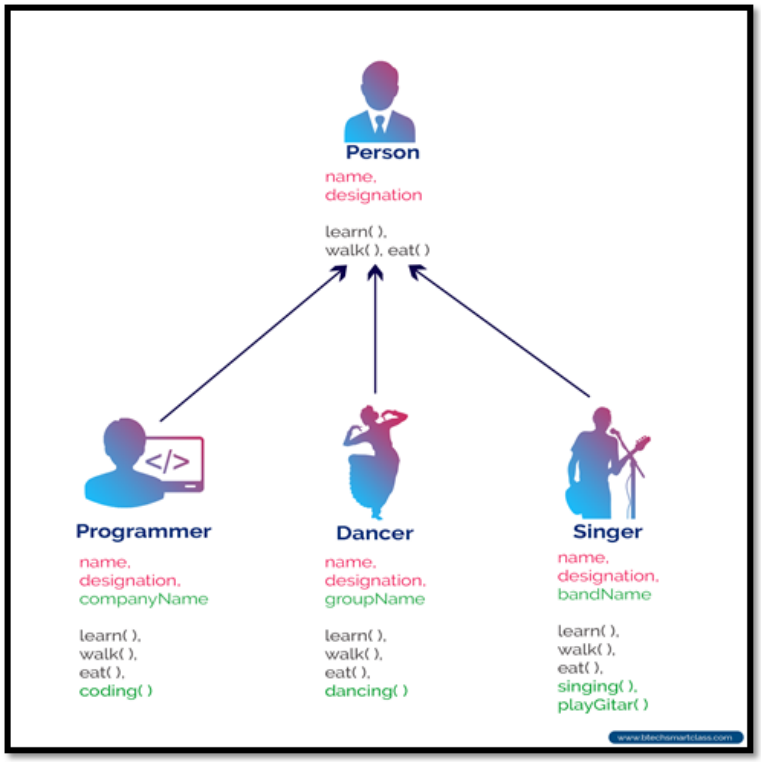


Fig 2.4.1

2.5 POLYMORPHISM

Polymorphism comes from Greek words, poly meaning many and morphs meaning forms, hence polymorphism, meaning "many forms". It is a feature that allows one interface to be used for a general class of actions. Polymorphism allows a child class to share the information and behaviour of its parent class while also incorporating its own functionality.

METHOD OVERLOADING METHOD OVERRIDING

COMPILE TIME POLYMORPHISM	**RUN TIME POLYMORPHISM**
STATIC BINDING	**DYNAMIC BINDING**
IMPLEMENT IN A SINGLE CLASS	**IMPLEMENT IN TWO CLASS**
NO NEED TO USE INHERITANCE	**WITH INHERITANCE CONCEPT WE CAN ACHIEVE METHOD OVERRIDING**
METHOD NAME SHOULD BE SAME	**METHOD NAME SHOULD BE SAME**

DIFFERENCE IN PARAMETER	SAME PARAMETER
RETURN TYPES CAN BE DIFFERENT	RETURN TYPE SHOULD BE SAME
STATIC METHODS CAN BE OVERLOAD	STATIC METHOD CANNOT BE OVER RIDE
EARLY BINDING	LATE BINDING

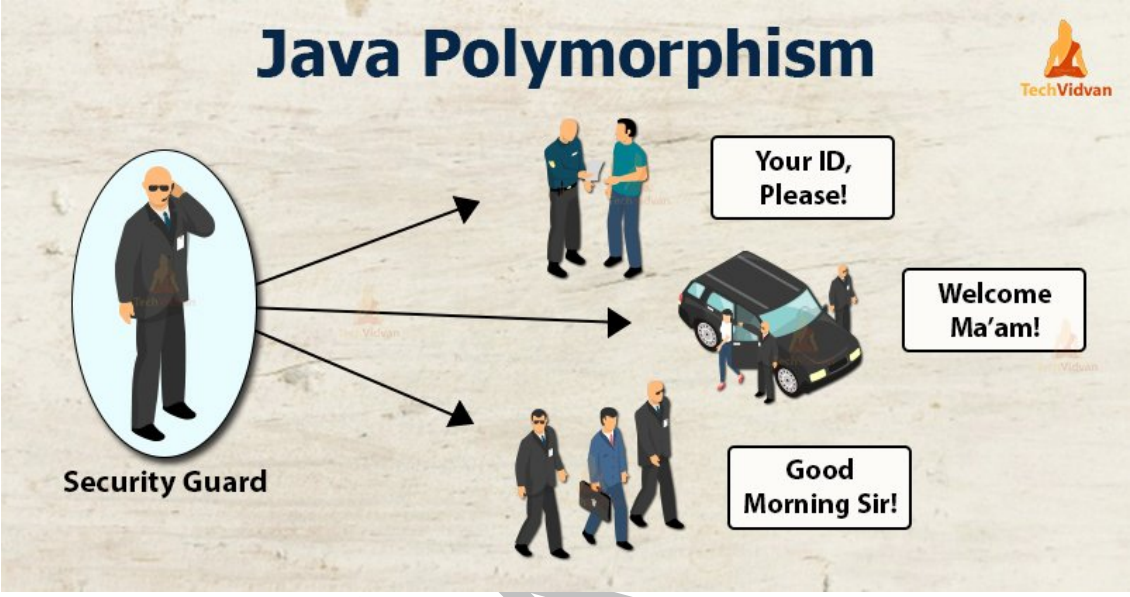


Fig. 2.5.1

2.6 JAVA PACKAGES

A package in Java is used to group similar or related classes, interfaces, and even sub-packages. Packages can be and are used to avoid name conflicts and to write better maintainable code. They are divided into two categories:

- Built-in Packages

  The Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

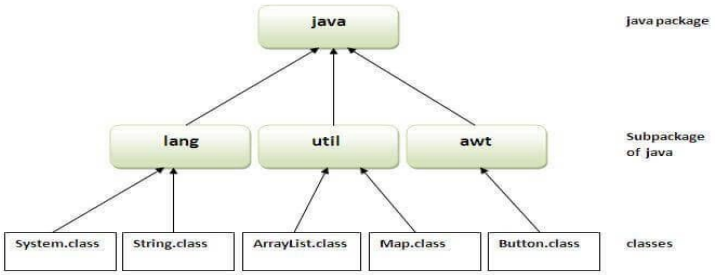


Fig. 2.7.1

- User-defined Packages

  Java also allows the programmers to make custom packages for ease in development and file management.

2.7 EXCEPTION HANDLING

Exception Handling in Java is a mechanism to handle the runtime errors so that normal flow of the application can be maintained. In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

There are five keywords used for handling exceptions in JAVA –

Try:

The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.

Catch:

The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.

Finally:

The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.

Throw:

The "throw" keyword is used to throw an exception.

Throws:

The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

3. DESIGN

3.1 DESIGN GOALS

This project includes:

- ✓ Keeps clean record of the students.
- ✓ Ensures efficient management of time
- ✓ Easy access and manipulation data.
- ✓ Storing of data for longer period of time.

3.2 ALGORITHM

LOGIN MODULE

- Login page is displayed.
- User ID and password..
- If correct & found to be admin then moves to admin window
- Else if correct & found to be a student then moves to student window.
- Else displays incorrect credentials warning.

HOME PAGE FOR ADMIN

- Place order
- View Bill & Order Details
- Change Password
- Manage Category
- Add New Product
- View, Edit & Delete Product
- Verify Users
- Logout
- Exit

ADD NEW PRODUCT

- If Admin clicks ADD, the entered details gets added in the database.

VIEW, EDIT & DELETE PRODUCT

- If Admin clicks UPDATE, the changed details gets updated in the database.
- If Admin clicks DELETE, the selecte4d customer details gets deleted from the database

VIEW BILL & ORDER DETAILS

- If Admin clicks PRINT, the customers details can be printed

CLEAR

- If Admin clicks RESET, the entered details gets reset in the text field

EXIT

- If Admin clicks EXIT, the admin can exit from the application

MANAGE CATEGORY

- If Admin clicks Manage Category, it displays window to manage and add new categories.

VERIFY USERS

- Only admin has the privilege to verify a user and grant him access to login.

3.3 FLOWCHART

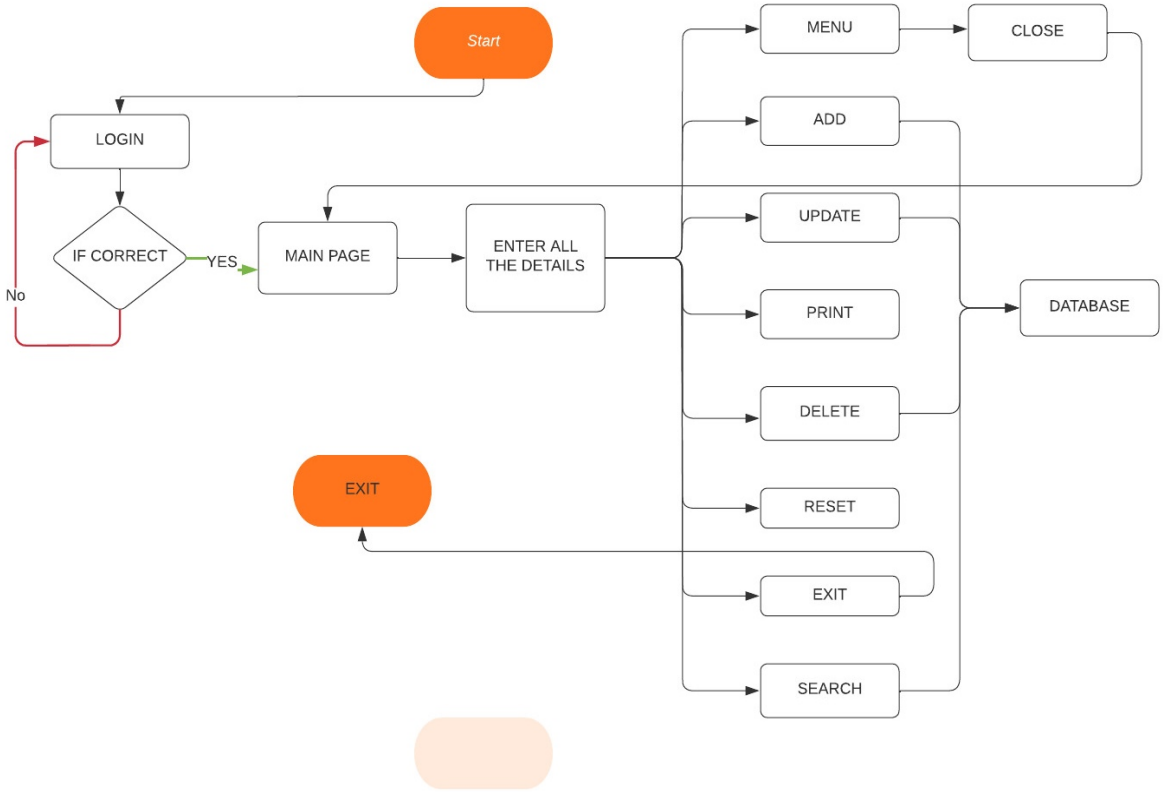


Fig. 3.3.1

4. IMPLEMENTATION

MODULE 4.1: Login Screen

```
public class Login extends javax.swing.JFrame {

    /** Creates new form Login */
    public Login() {
        initComponents();
        btnLogin.setEnabled(false);
    }

    public void clear(){
        txtUsn.setText("");
        txtPassword.setText("");
        btnLogin.setEnabled(false);
    }

    public void validateFields(){
        String usn = txtUsn.getText();
        String password = txtPassword.getText();
        if(!usn.equals("") && !password.equals("")){
            btnLogin.setEnabled(true);
        }
        else{
            btnLogin.setEnabled(false);
        }
    }
```

```
private void btnLoginActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String usn = txtUsn.getText();
    String password = txtPassword.getText();
    User user = null;
    user = UserDao.login(usn, password);
    if(user==null)
        JOptionPane.showMessageDialog(null, "<html><b style=\"color:red\">Incorrect USN or Password</b></html>", "Message", JOptionPane.ERROR_MESSAGE);
    else{
        if(user.getStatus().equals("false")){
            ImageIcon icon = new ImageIcon("src/popupicon/wait.png");
            JOptionPane.showMessageDialog(null, "<html><b>Wait for Admin Approval</b></html>","Message", JOptionPane.INFORMATION_MESSAGE, icon);
            clear();
        }
        if(user.getStatus().equals("true")){
            setVisible(false);
            new Home(usn).setVisible(true);
        }
    }

}

private void btnExitActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int a = JOptionPane.showConfirmDialog(null, "Dp you really want to Close the Application", "Select", JOptionPane.YES_NO_OPTION);
    if(a==0)
        System.exit(0);
}
```

```
private void txtUsnKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    validateFields();
}

private void txtPasswordKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    validateFields();
}

private void btnClearActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    clear();
}

private void btnSignupActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    setVisible(false);
    new Signup().setVisible(true);
}
```

Fig 4.1

MODULE 4.2: Main function

```
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        Look and feel setting code (optional)

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Login().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton btnClear;
    private javax.swing.JButton btnExit;
    private javax.swing.JButton btnLogin;
    private javax.swing.JButton btnSignup;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JPasswordField txtPassword;
    private javax.swing.JTextField txtUsn;
    // End of variables declaration

}
```

Fig 4.2

MODULE 4.3: SIGNUP

```
public class Signup extends javax.swing.JFrame {

    public String mobileNumberPattern = "^[0-9]*$";
    /**
     * Creates new form Signup`
     */
    public Signup() {
        initComponents();
        btnSave.setEnabled(false);

    }
    public void clear(){
        txtName.setText("");
        txtUsn.setText("");
        txtMobileNumber.setText("");
        txtDepartment.setText("");
        txtSemester.setText("");
        txtPassword.setText("");
        btnSave.setEnabled(false);
    }

    public void validateFields(){
        String name = txtName.getText();
        String usn = txtUsn.getText();
        String mobileNumber = txtMobileNumber.getText();
        String department = txtDepartment.getText();
        String semester = txtSemester.getText();
    String password = txtPassword.getText();
    if(!name.equals("") && !usn.equals("") && mobileNumber.matches(mobileNumberPattern) && mobileNumber.length()==10 &&
            !password.equals("") && !department.equals("") && !semester.equals("")){
        btnSave.setEnabled(true);
    }
    else{
        btnSave.setEnabled(false);
    }
}
```

```
private void btnExitActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int a =JOptionPane.showConfirmDialog(null, "Do you really want to Close the Application", "Select", JOptionPane.YES_NO_OPTION);
    if(a==0)
        System.exit(0);
}

private void btnSaveActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    User user = new User();
    user.setName(txtName.getText());
    user.setUsn(txtUsn.getText());
    user.setMobileNumber(txtMobileNumber.getText());
    user.setDepartment(txtDepartment.getText());
    user.setSemester(txtSemester.getText());
    user.setPassword(txtPassword.getText());
    UserDao.save(user);
    clear();

}

private void btnClearActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    clear();
}
```

```
private void txtNameKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    validateFields();
}

private void txtUsnKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    validateFields();
}

private void txtMobileNumberKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    validateFields();
}

private void txtDepartmentKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    validateFields();
}

private void txtSemesterKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    validateFields();
}
private void txtPasswordKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    validateFields();
}

private void btnLoginActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    setVisible(false);
    new Login().setVisible(true);
}
```

Fig 4.3

MODULE 4.4: Manage Category

```
private void txtPasswordKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    validateFields();
}

private void btnLoginActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    setVisible(false);
    new Login().setVisible(true);
}
```

```
public class ManageCategory extends javax.swing.JFrame {

    /**
     * Creates new form ManageCategory
     */
    public ManageCategory() {
        initComponents();
        btnSave.setEnabled(false);
    }

    public void validateField() {
        String category = txtName.getText();
        if (!category.equals("")) {
            btnSave.setEnabled(true);
        } else {
            btnSave.setEnabled(false);
        }
    }
```

```
private void txtNameKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    validateField();
}

private void btnSaveActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Category category = new Category();
    category.setName(txtName.getText());
    CategoryDao.save(category);
    setVisible(false);
    new ManageCategory().setVisible(true);
}

private void formComponentShown(java.awt.event.ComponentEvent evt) {
    // TODO add your handling code here:
    DefaultTableModel dtm = (DefaultTableModel) jTable1.getModel();
    ArrayList<Category> list = CategoryDao.getAllRecords();
    Iterator<Category> itr = list.iterator();
    while (itr.hasNext()) {
        Category categoryObj = itr.next();
        dtm.addRow(new Object[]{categoryObj.getId(), categoryObj.getName()});
    }
}
```

```
private void jTable1MouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int index = jTable1.getSelectedRow();
    TableModel model = jTable1.getModel();
    String id = model.getValueAt(index, 0).toString();
    String name = model.getValueAt(index, 1).toString();
    int a = JOptionPane.showConfirmDialog(null, "Do you want to Delete " + name + "Category", "Select", JOptionPane.YES_NO_OPTION);
    if (a == 0) {
        CategoryDao.delete(id);
        setVisible(false);
        new ManageCategory().setVisible(true);
    }
}

private void btnClearActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    setVisible(false);
    new ManageCategory().setVisible(true);
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    setVisible(false);
}
```

Fig 4.4

MODULE 4.5: Add New Product

```
public class AddNewProduct extends javax.swing.JFrame {

    /**
     * Creates new form AddNewProduct
     */
    public AddNewProduct() {
        initComponents();
        btnSave.setEnabled(false);
    }

    public void validateFields() {
        String name = txtName.getText();
        String price = txtPrice.getText();
        if (!name.equals("") && !price.equals("")) {
            btnSave.setEnabled(true);
        } else {
            btnSave.setEnabled(false);
        }
    }
private void txtNameKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    validateFields();
}

private void txtPriceKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    validateFields();
}

private void formComponentShown(java.awt.event.ComponentEvent evt) {
    // TODO add your handling code here:
    ArrayList<Category> list = CategoryDao.getAllRecords();
    Iterator<Category> itr = list.iterator();
    while (itr.hasNext()) {
        Category categoryObj = itr.next();
        txtCategory.addItem(categoryObj.getName());
    }
}
private void btnSaveActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Product product = new Product();
    product.setName(txtName.getText());
    product.setCategory((String)txtCategory.getSelectedItem());
    product.setPrice(txtPrice.getText());
    ProductDao.save(product);
    setVisible(false);
    new AddNewProduct().setVisible(true);

}

private void btnClearActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    setVisible(false);
    new AddNewProduct().setVisible(true);
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
        setVisible(false);
}
```

Fig 4.5

MODULE 4.6: View, Edit and Delete Product

```
public class ViewEditDeleteProduct extends javax.swing.JFrame {

    /**
     * Creates new form ViewEditDeleteProduct
     */
    public ViewEditDeleteProduct() {
        initComponents();
        btnUpdate.setEnabled(false);
        btnDelete.setEnabled(false);
    }

    public void validateField() {
        String name = txtName.getName();
        String price = txtPrice.getText();
        String category = (String) jComboBox1.getSelectedItem();
        if (!name.equals("") && !price.equals("") && category != null) {
            btnUpdate.setEnabled(true);
        } else {
            btnUpdate.setEnabled(false);
        }
    }
```

```
private void txtNameKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    validateField();
}

private void txtPriceKeyReleased(java.awt.event.KeyEvent evt) {
    // TODO add your handling code here:
    validateField();
}

private void btnUpdateActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Product product = new Product();
    int id = Integer.parseInt(lblId.getText());
    product.setId(id);
    product.setName(txtName.getText());
    product.setCategory((String)jComboBox1.getSelectedItem());
    product.setPrice(txtPrice.getText());
    ProductDao.update(product);
    setVisible(false);
    new ViewEditDeleteProduct().setVisible(true);

}
```

```
private void formComponentShown(java.awt.event.ComponentEvent evt) {
    // TODO add your handling code here:
    DefaultTableModel dtm = (DefaultTableModel) jTable1.getModel();
    ArrayList<Product> list = ProductDao.getAllRecords();
    Iterator<Product> itr = list.iterator();
    while (itr.hasNext()) {
        Product productObj = itr.next();
        dtm.addRow(new Object[]{productObj.getId(), productObj.getName(), productObj.getCategory(), productObj.getPrice()});
    }
}

private void jTable1MouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int index = jTable1.getSelectedRow();
    TableModel model = jTable1.getModel();
    String id = model.getValueAt(index, 0).toString();
    lblId.setText(id);
    String name = model.getValueAt(index, 1).toString();
    txtName.setText(name);
    String category = model.getValueAt(index, 2).toString();
    String price = model.getValueAt(index, 3).toString();
    txtPrice.setText(price);
    btnUpdate.setEnabled(true);
    btnDelete.setEnabled(true);
    jComboBox1.removeAllItems();
    ArrayList<Category> list = CategoryDao.getAllRecords();
    Iterator<Category> itr = list.iterator();
```

```
        while (itr.hasNext()) {
            Category categoryObj = itr.next();
            if (!categoryObj.getName().equals(category)) {
                jComboBox1.addItem(categoryObj.getName());
            }
        }
    }

    private void btnDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        String id = lblId.getText();
        int a = JOptionPane.showConfirmDialog(null, "Do you what to Delete this product", "Select", JOptionPane.YES_NO_OPTION);
        if (a == 0) {
            ProductDao.delete(id);
            setVisible(false);
            new ViewEditDeleteProduct().setVisible(true);
        }
    }

    private void btnClearActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        setVisible(false);
        new ViewEditDeleteProduct().setVisible(true);
    }
```

Fig 4.6

MODULE 4.7: Verify Users

```
public class VerifyUsers extends javax.swing.JFrame {

    /**
     * Creates new form VerifyUsers
     */
    public VerifyUsers() {
        initComponents();
    }

    public void getAllRecords(String email){
        DefaultTableModel dtm = (DefaultTableModel) jTable1.getModel();
        dtm.setRowCount(0);
        ArrayList<User> list = UserDao.getAllRecords(email);
        Iterator<User> itr = list.iterator();
        while(itr.hasNext()){
            User userObj = itr.next();    Object anObject
            if(!userObj.getEmail().equals())
        }
```

```
private void formComponentShown(java.awt.event.ComponentEvent evt) {
    // TODO add your handling code here:
    getAllRecords("");
}
```

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO ad boolean b handling code here:
    setVisible(false);
}
```

```
private void jTable1MouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int index = jTable1.getSelectedRow();
    TableModel model = jTable1.getModel();
    String email = model.getValueAt(index, 2).toString();
    String status = model.getValueAt(index, 6).toString();
    if(status.equals("true"))
        status = "false";
    else
        status = "true";
    int a = JOptionPane.showConfirmDialog(null,"Do you want to change status of "+email+""
    if(a==0)
    {
        UserDao.changeStatus(email, status);
        setVisible(false);          boolean b
        new VerifyUsers().setVisible(true);
    }
}
```

Fig 4.7

MODULE 4.8: Place Order

```
public void validateField(){
    String customerName = txtCusName.getText();
    String customerMobileNumber = txtCusMobileNo.getText();
    String customerEmail = txtCusEmail.getText();
    if(!customerEmail.equals("") && customerMobileNumber.matches(mobileNumberPattern) && customerMobileNumber.
        btnGenerateBillPrint.setEnabled(true);
    }
    else{                                    boolean b
        btnGenerateBillPrint.setEnabled(false);
    }
}
```

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    setVisible(false);
    new Home(userEmail).setVisible(true);
}
```

```
private void formComponentShown(java.awt.event.ComponentEvent evt) {
    // TODO add your handling code here:
    billId = Integer.parseInt(BillDao.getId());
    jLabel3.setText(BillDao.getId());
    ArrayList<Category> list = CategoryDao.getAllRecords();
    Iterator<Category> itr = list.iterator();
    while(itr.hasNext()){
        Category categoryObj = itr.next();
        jComboBox1.addItem(categoryObj.getName());
    }
    String category = (String) jComboBox1.getSelectedItem();
    productNameByCategory(category);
}
```

```
private void jTable1MouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int index = jTable1.getSelectedRow();
    TableModel model = jTable1.getModel();
    String productName = model.getValueAt(index, 0).toString();
    Product product = ProductDao.getProductByname(productName);
    txtProName.setText(product.getName());
    txtProPrice.setText(product.getPrice());
    jSpinner1.setValue(1);
    txtProTotal.setText(product.getPrice());
    productPrice = Integer.parseInt(product.getPrice());
    productTotal = Integer.parseInt(product.getPrice());
    btnAddToCart.setEnabled(true);
}
```

Fig 4.8

```
private void jSpinner1StateChanged(javax.swing.event.ChangeEvent evt) {
    // TODO add your handling code her
    int quantity = (Integer) jSpinner1
    if(quantity <=1){
        jSpinner1.setValue(1);
        quantity = 1;
    }
    productTotal = productPrice * quan
    txtProTotal.setText(String.valueOf(productTotal));
}
```

Object obj
char[] data
char[] data, int offset, int count
boolean b
char c
int i
long l
float f
double d

```
public class VerifyUsers extends javax.swing.JFrame {

    /**
     * Creates new form VerifyUsers
     */
    public VerifyUsers() {
        initComponents();
    }

    public void getAllRecords(String email){
        DefaultTableModel dtm = (DefaultTableModel) jTable1.getModel();
        dtm.setRowCount(0);
        ArrayList<User> list = UserDao.getAllRecords(email);
        Iterator<User> itr = list.iterator();
        while(itr.hasNext()){
            User userObj = itr.next();
            if(!userObj.getEmail().equals())
        }
```

Object anObject

```
private void formComponentShown(java.awt.event.ComponentEvent evt) {
    // TODO add your handling code here:
    getAllRecords("");
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    setVisible(false);
}
private void jTable1MouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int index = jTable1.getSelectedRow();
    TableModel model = jTable1.getModel();
    String email = model.getValueAt(index, 2).toString();
    String status = model.getValueAt(index, 6).toString();
    if(status.equals("true"))
        status = "false";
    else
        status = "true";
    int a = JOptionPane.showConfirmDialog(null,"Do you want to change status of "+email+""
    if(a==0)
    {
        UserDao.changeStatus(email, status);
        setVisible(false);
        new VerifyUsers().setVisible(true);
    }
}
```

MODULE 4.9: View Bill & Order Details

```
public class ViewBillsOrderPlacedDetails extends javax.swing.JFrame {

    /**
     * Creates new form ViewBillsOrderPlacedDetails
     */
    public ViewBillsOrderPlacedDetails() {
        initComponents();
        SimpleDateFormat dFormat = new SimpleDateFormat("dd-MM-yyyy");
        Date date = new Date();
        String todayDate = dFormat.format(date);
        jTextField1.setText(todayDate);
    }

    public void tableDetails(){
        String date = jTextField1.getText();
        String incDec =(String) jComboBox1.getSelectedItem();
        DefaultTableModel dtm = (DefaultTableModel) jTable1.getModel();
        dtm.setRowCount(0);
```

```
    if (incDec.equals("INC")) {
        ArrayList<Bill> list = BillDao.getAllRecordsByInc(date);
        Iterator<Bill> itr = list.iterator();
        while (itr.hasNext()) {
            Bill billObj = itr.next();
            dtm.addRow(new Object[]{billObj.getId(), billObj.getName(), billObj.getMobileNumber(), billObj.getEmail(),
        }
    } else {
        ArrayList<Bill> list = BillDao.getAllRecordsByDesc(date);
        Iterator<Bill> itr = list.iterator();
        while (itr.hasNext()) {
            Bill billObj = itr.next();
            dtm.addRow(new Object[]{billObj.getId(), billObj.getName(), billObj.getMobileNumber(), billObj.getEmail(),
        }
    }
}
```

```
private void formComponentShown(java.awt.event.ComponentEvent evt) {
    // TODO add your handling code here:
    getAllRecords("");
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO ad boolean b handling code here:
    setVisible(false);
}
```

Fig 4.9

MODULE 4.10: Change Password

```
public class ChangePassword extends javax.swing.JFrame {

    public String userEmail;

    /**
     * Creates new form ChangePassword
     */
    public ChangePassword() {
        initComponents();
    }

    public ChangePassword(String email) {
        initComponents();
        userEmail = email;  boolean b
        btnUpdate.setEnabled(false);
    }
```

```
public ChangePassword(String email) {
    initComponents();
    userEmail = email;
    btnUpdate.setEnabled(false);
}

public void validateField(){
    String oldPassword = txtOldPassword.getText();
    String newPassword = txtNewPassword.getText();
    String confirmPassword = txtConfirmPassword.getText();
    if(!oldPassword.equals("") && !newPassword.equals("") && !confirmPassword.equals("") && newPassword.equals
        btnUpdate.setEnabled(true);
    }
    else
        btnUpdate.setEnabled(false);
}
```

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    setVisible(false);
    new ChangePassword(userEmail).setVisible(true);
}

private void btnUpdateActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String oldPassword = txtOldPassword.getText();
    String newPassword = txtNewPassword.getText();
    UserDao.changePassword(userEmail, oldPassword, newPassword);
    setVisible(false);                                    boolean b
    new ChangePassword(userEmail).setVisible(true);
}

private void formComponentShown(java.awt.event.ComponentEvent evt) {
    // TODO add your handling code here:
    getAllRecords("");
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO ad boolean b handling code here:
    setVisible(false);
}
```

Fig 4.10

MODULE 4.11: Packages Imported

```
package smart.canteen;
import java.util.*;
import model.Category;
import model.Product;
import dao.CategoryDao;
import dao.ProductDao;
import javax.swing.JOptionPane;
import javax.swing.JOptionPane;
import javax.swing.ImageIcon;
import model.User;
import dao.UserDao;
import model.Category;
import dao.CategoryDao;
import java.util.ArrayList;
import java.util.Iterator;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
```

RESULTS

An overview of the application is given in this section.

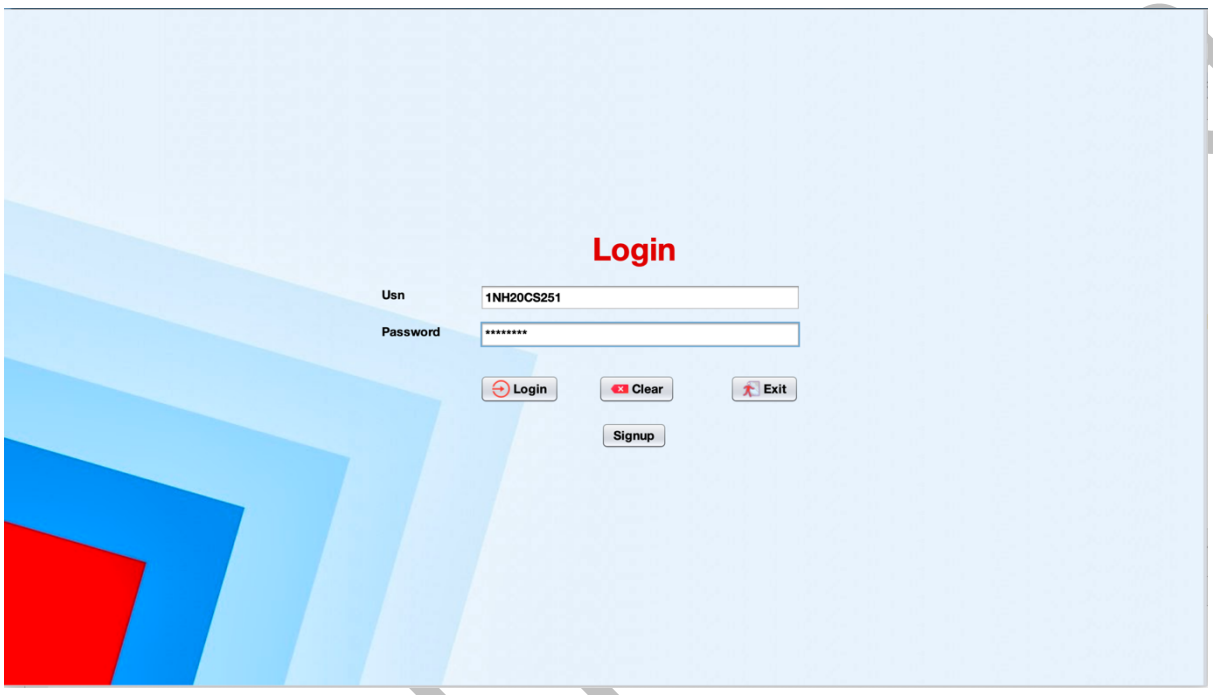


Fig. 5.1 Login Page

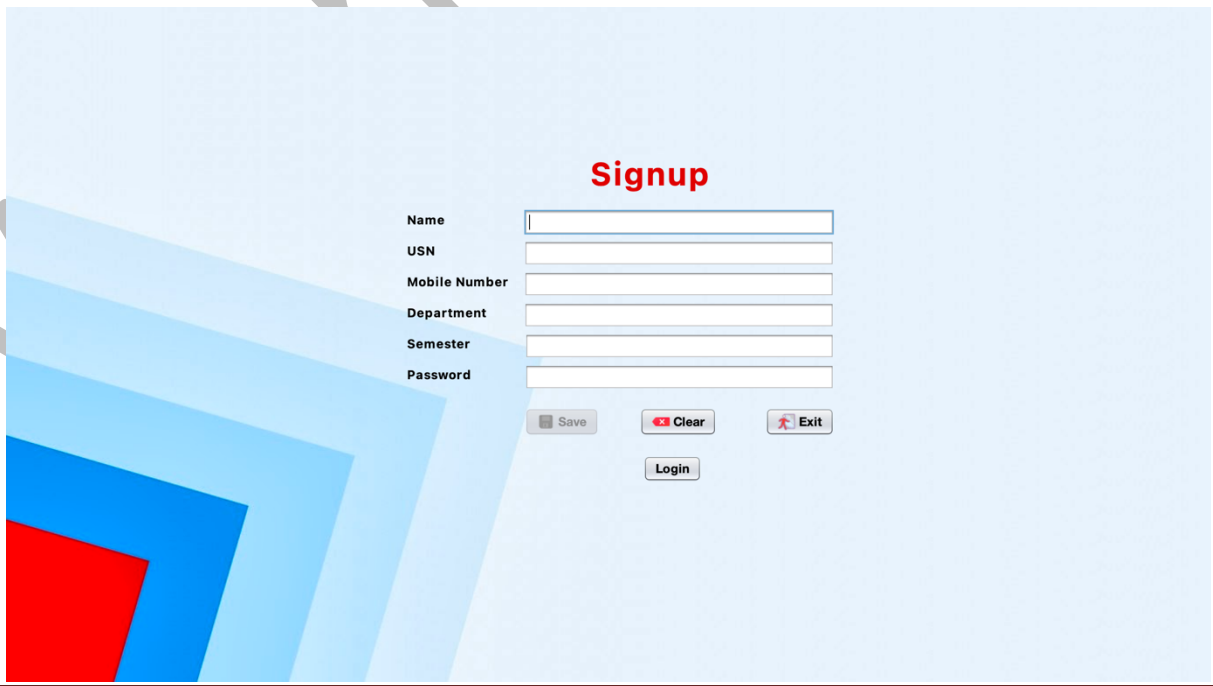


Fig. 5.2 Sign up Page

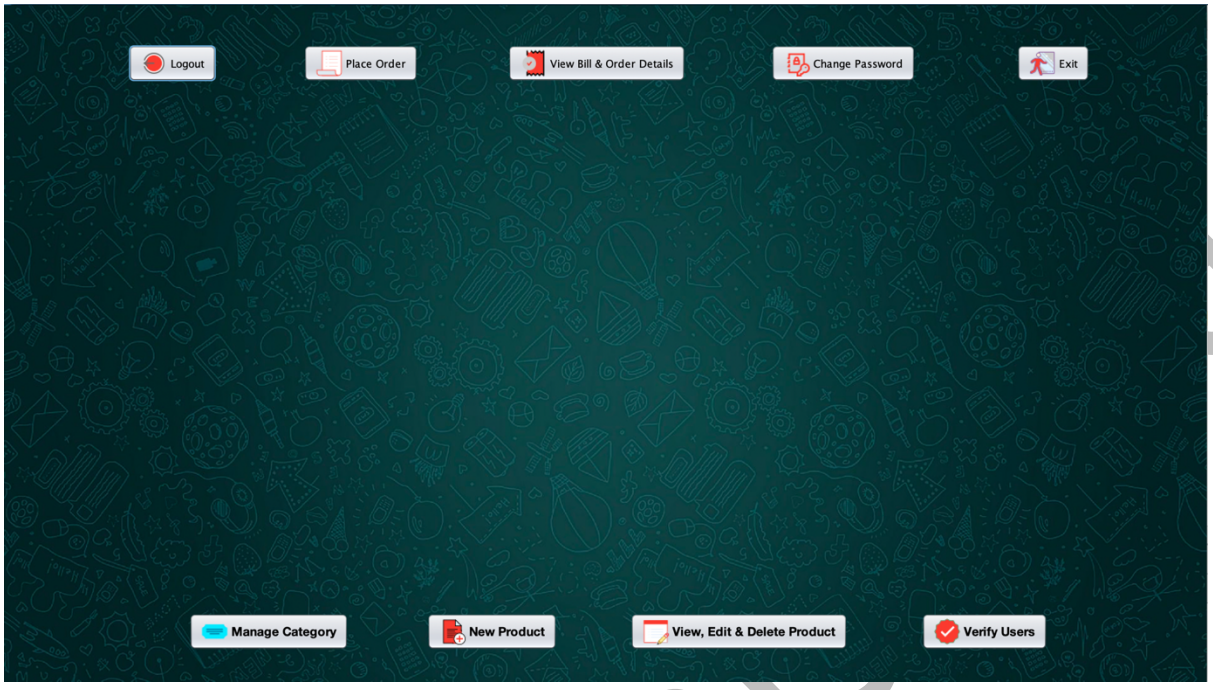


Fig. 5.3 Admin Interface

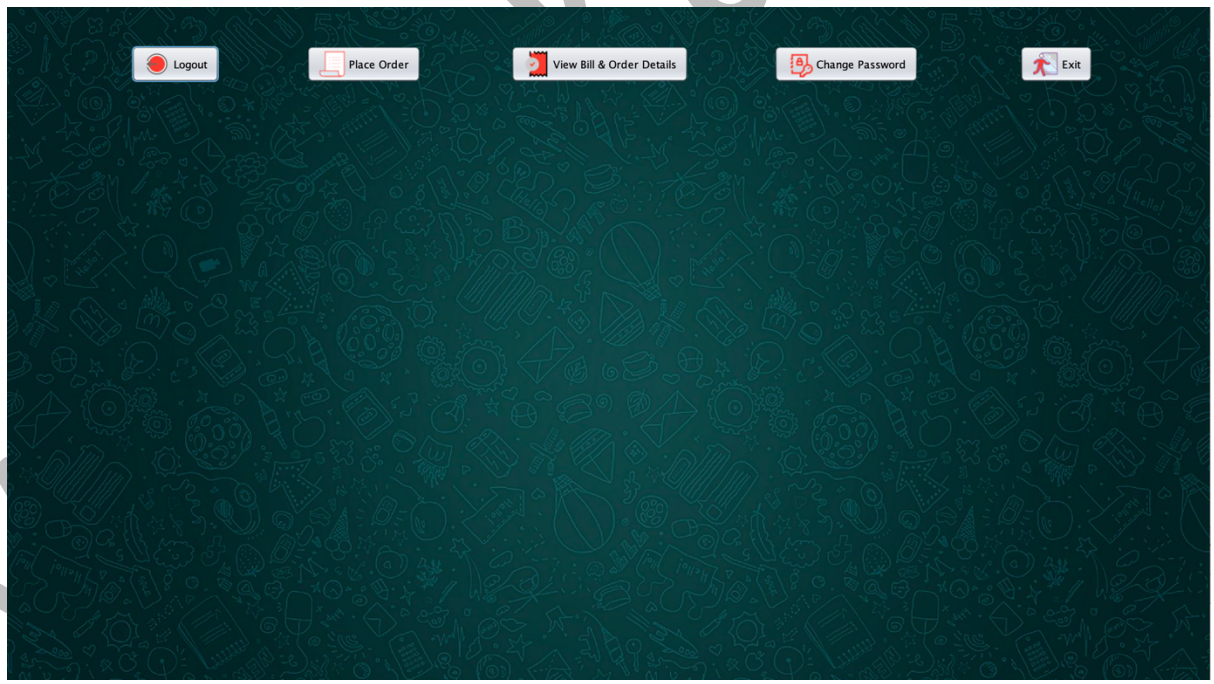


Fig. 5.4 Non-admin i.e Student Interface

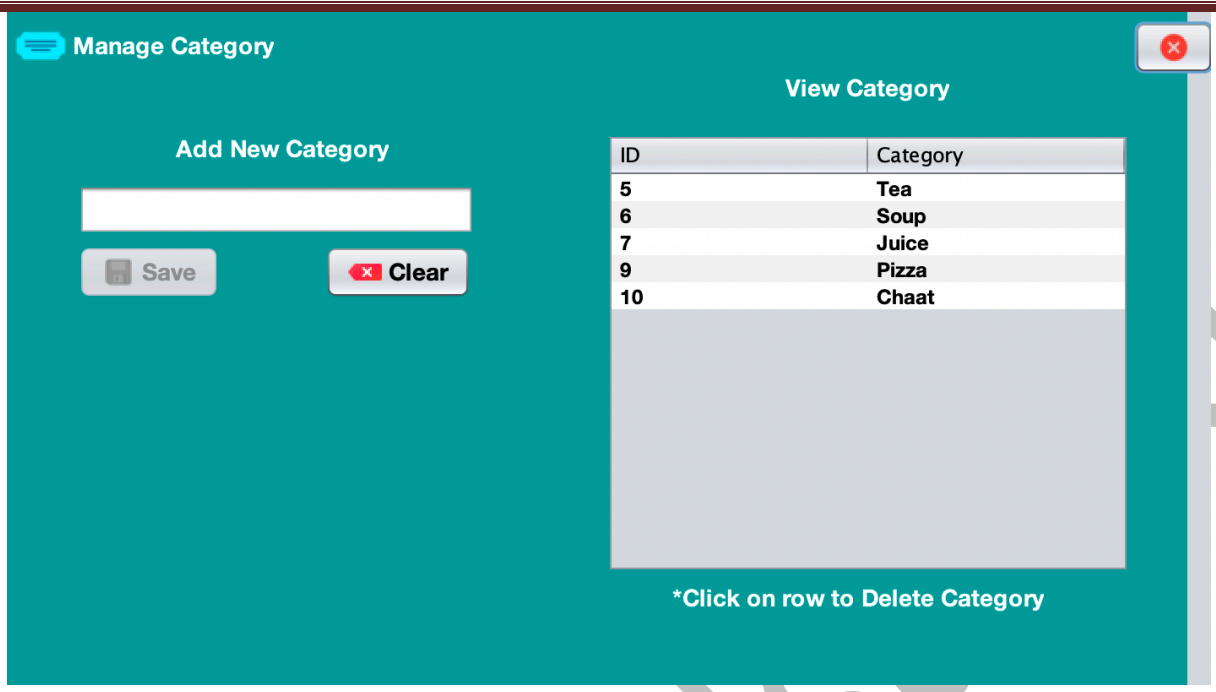


Fig. 5.5 Manage Category

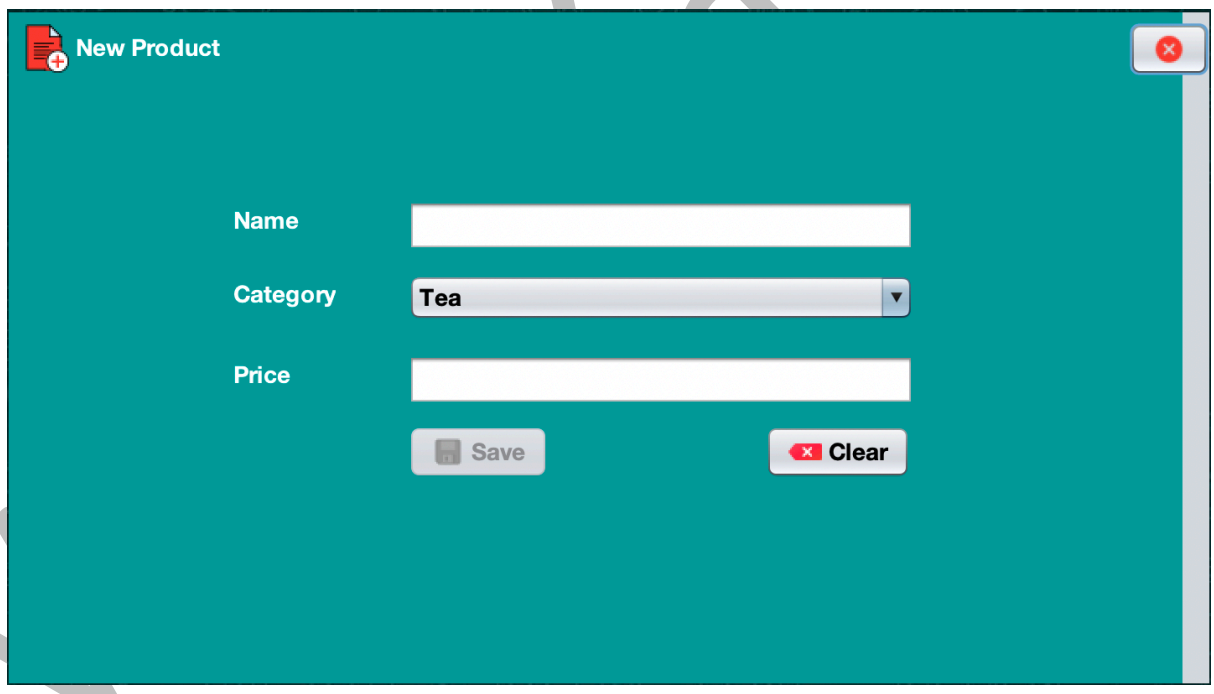


Fig. 5.6 Add New Product

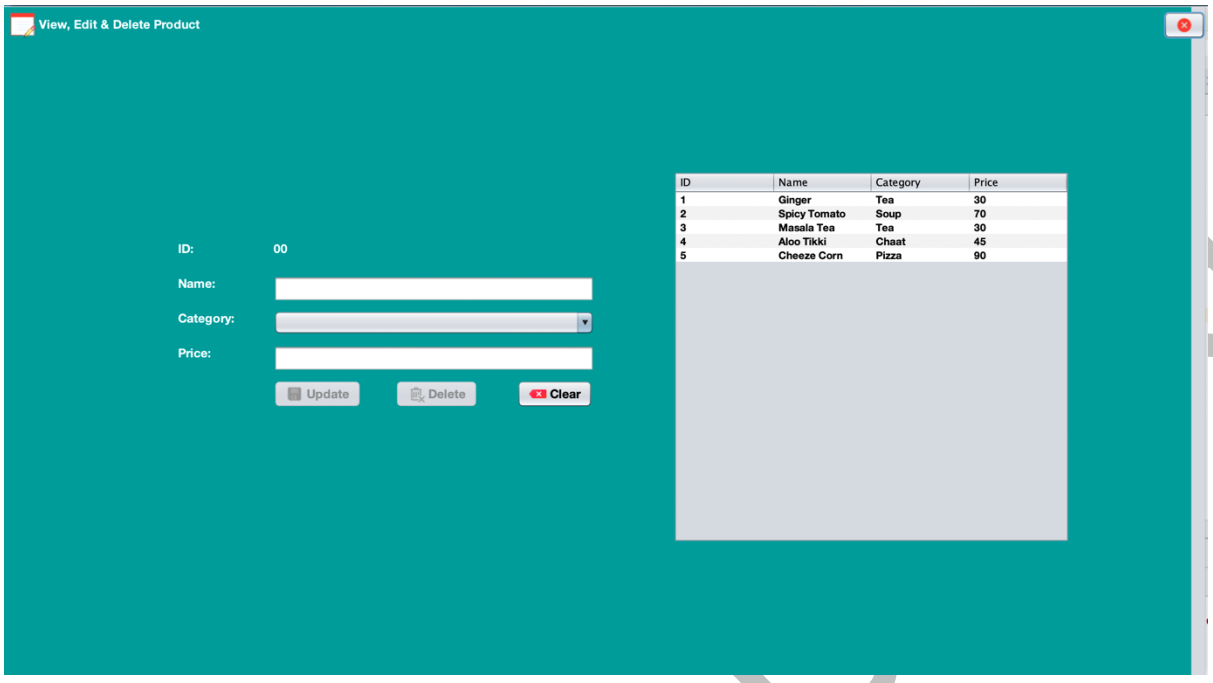


Fig. 5.7 View, Edit & Delete Product

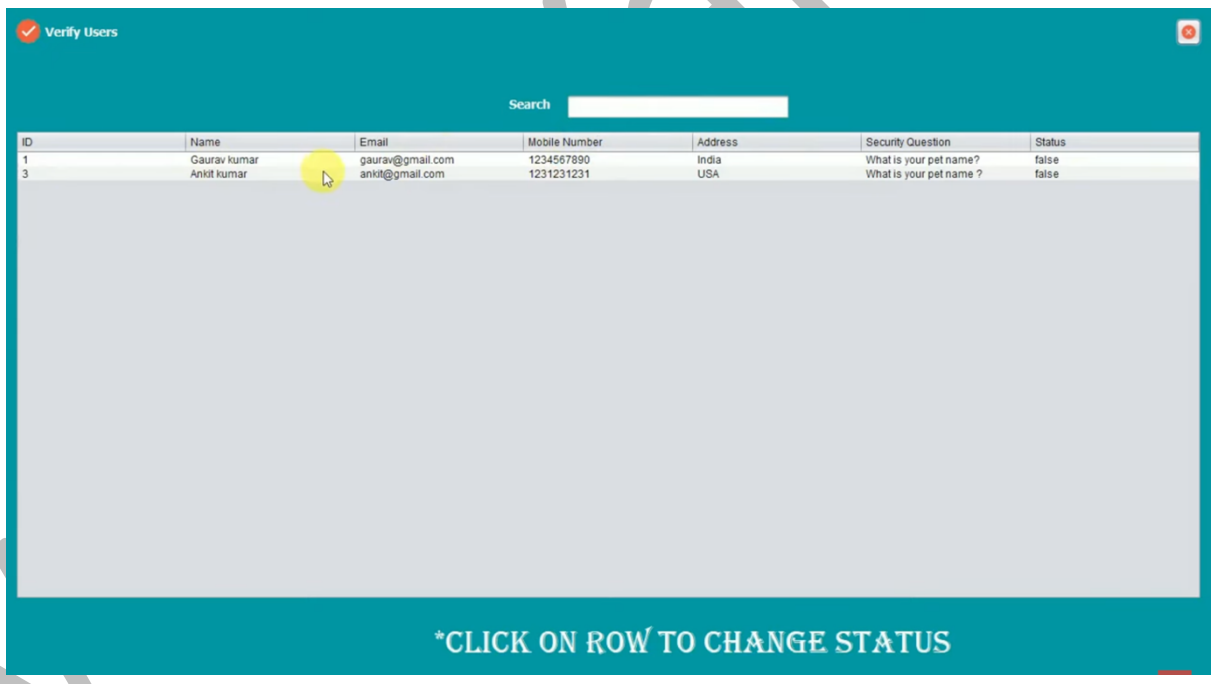


Fig. 5.8 Verify a new Sign Up

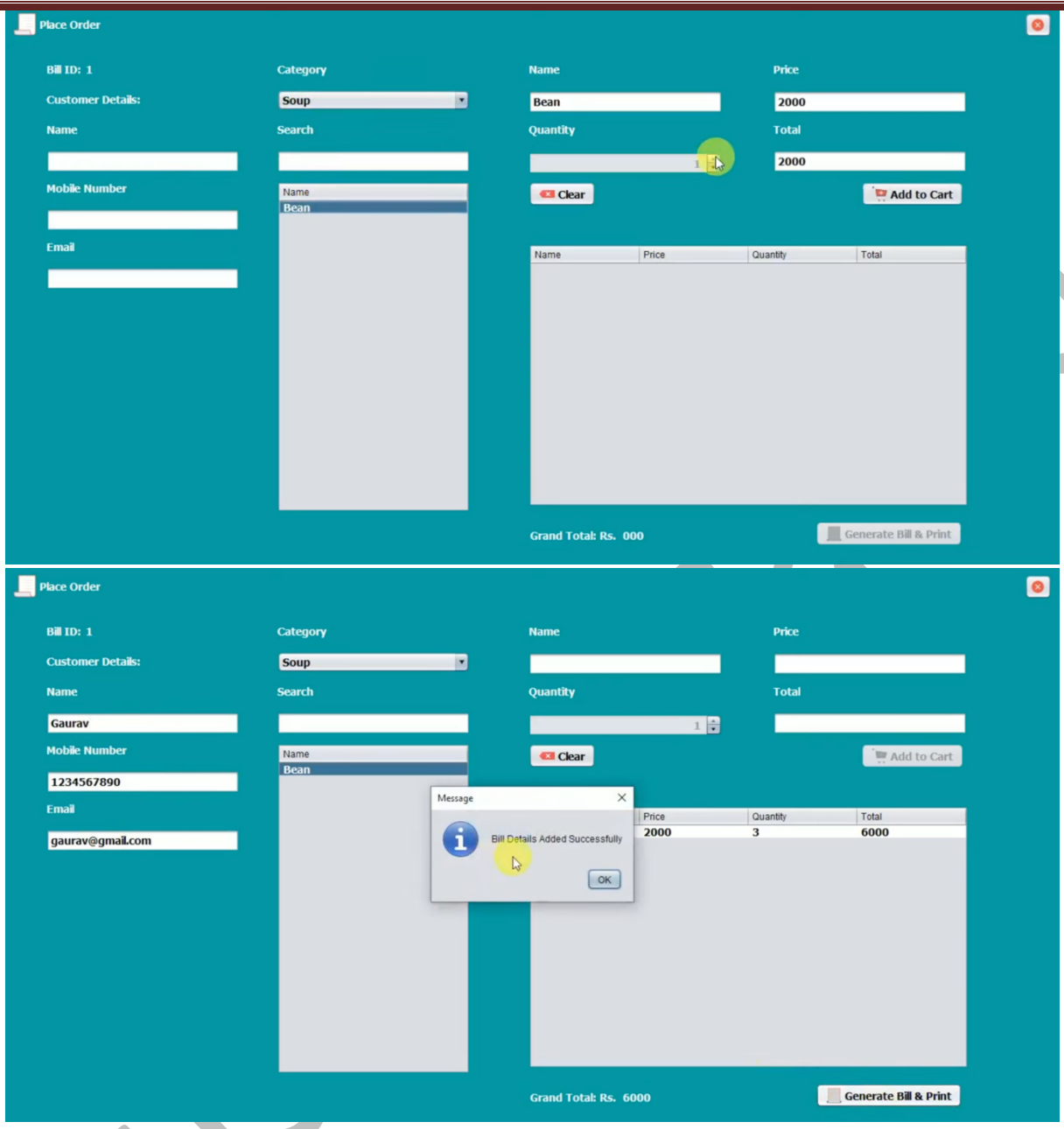
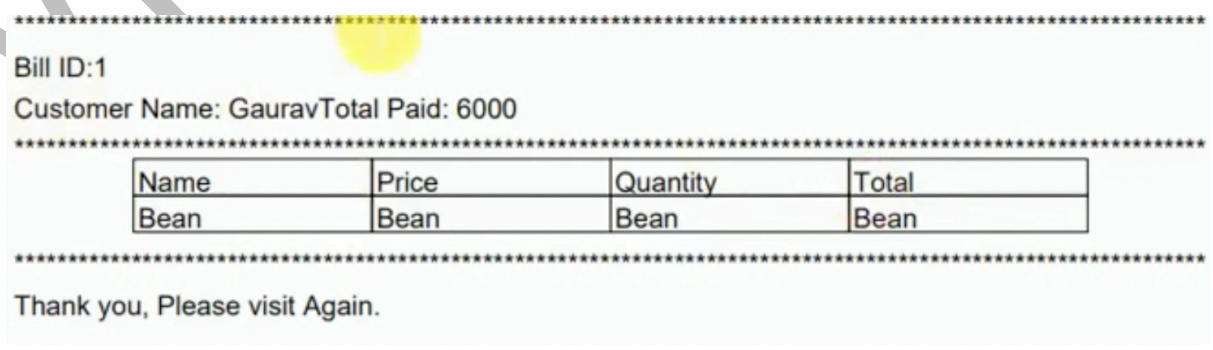


Fig. 5.9 Place Order



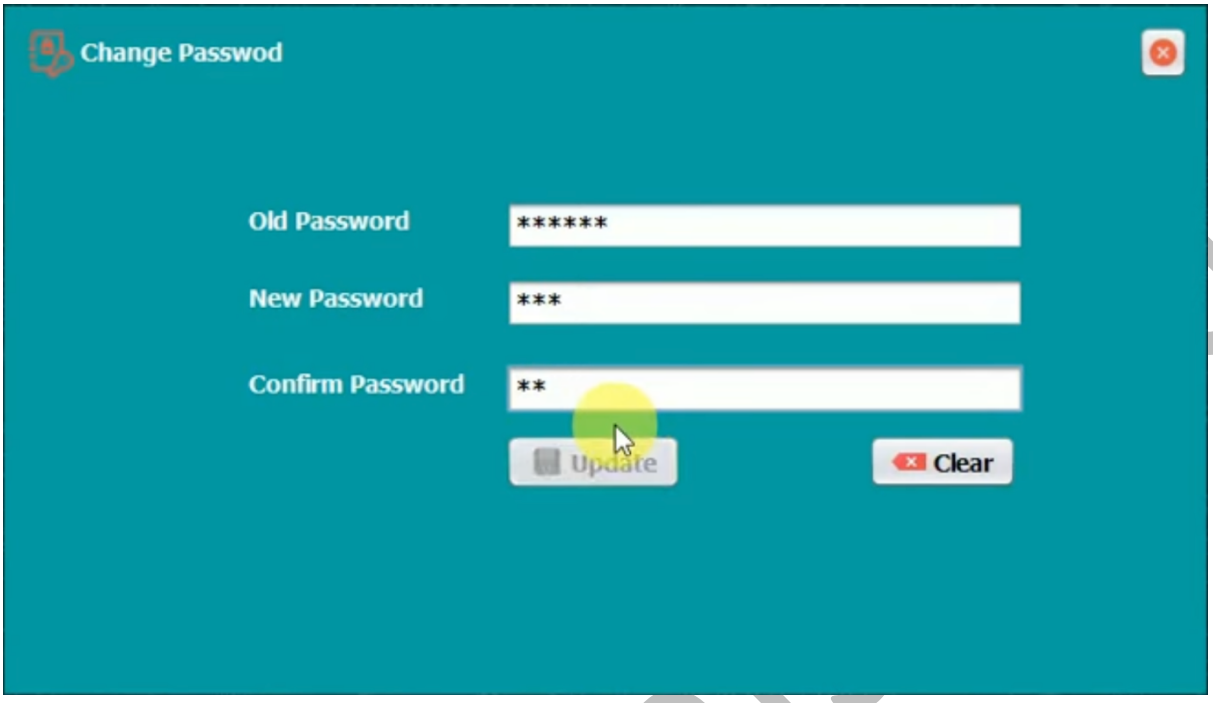
Bill ID:1

Customer Name: GauravTotal Paid: 6000

Name	Price	Quantity	Total
Bean	Bean	Bean	Bean

Thank you, Please visit Again.

Fig. 5.10 Generate Bill



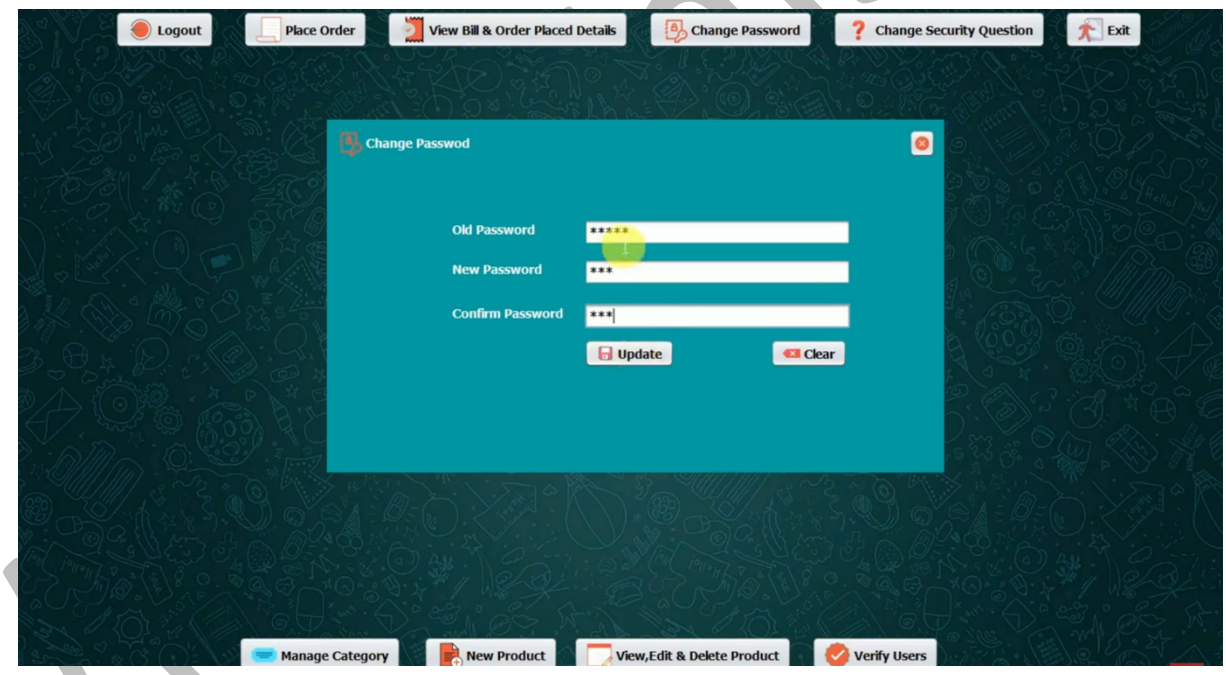


Fig. 5.11 Change Password

CONCLUSION

Thus, my mini project titled “Smart Canteen” has been successfully implemented using the concepts of OBJECT ORIENTED PROGRAMMING and the approach MySQL and JAVAFX. The project achieves its aim of helping the canteen owners as well as the students to manage their time and efficiently use this project for their own benefit.

This project not only allowed me to apply the concepts of OOPS but also helped my gain knowledge about some advanced java concepts which will surely prove to be beneficial in future.

REFRENCES

- Geeks For Geeks – (www.geeksforgeeks.org)
- BTech Days – (https://www.youtube.com/c/BTechDays)
- JAVA The Complete Reference – by Herbert Schildt
- JavaPoint – (www.javatpoint.com)