Session: 2023 – 2027

Submitted by:

M.Tayyab Amir 2023-CS-101

Supervised by:

Prof. Dr. Muhammad Awais Hassan

Course:

CSC-103 Object Oriented Programming

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

Introduction:

A Tech is a place which gives information about the latest technology devices/things to the customer and customers can buy those things according to their will or need. He can get different information related to these devices/things i.e. Price of the device, its company and which device he can get in his budget. This business application will ask the customer about his maximum range and the device (He wants to buy) and the program will show him the device according to his range and its price with the discount (if applicable).The shop also includes second hand devices for the customers as many people like to buy second hand devices. Admin of the shop can change the prices of devices, add devices and delete devices.

OOP-CONCEPTS:

The major concepts of OOP that have been used in this project include Association which further comprises Aggregation and Composition. The other major concept that has been used is Inheritance.

- **Association:**

Aggregation and Composition have been used a number of times in the project, especially aggregation.

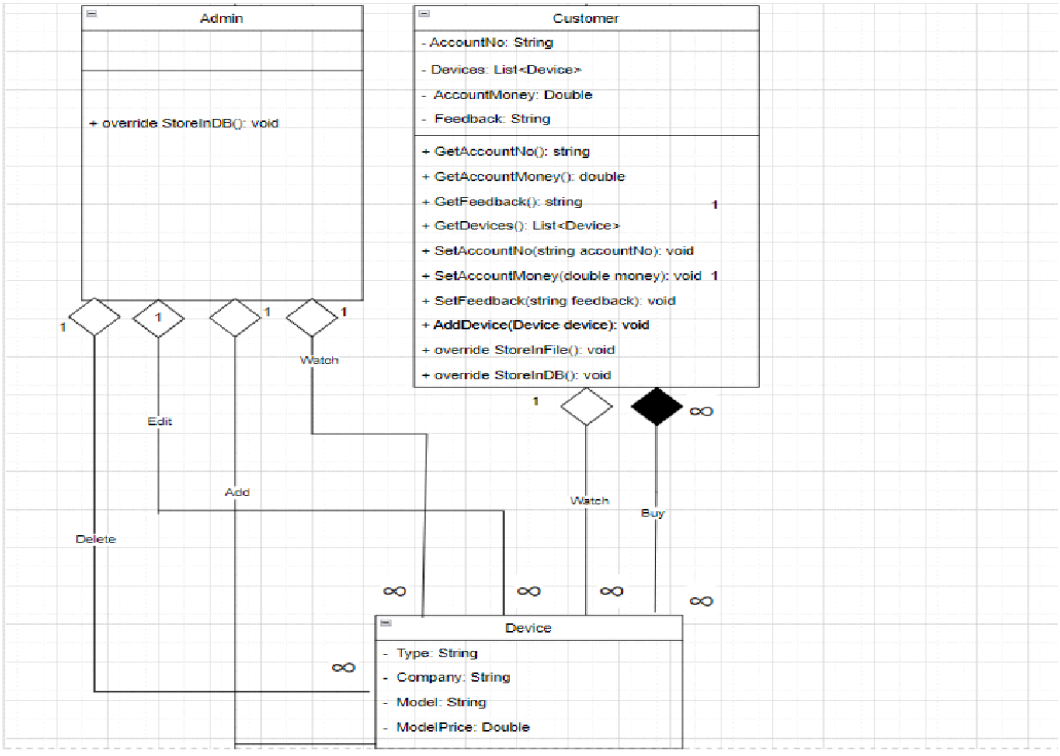


Figure 1

- **Encapsulation:**

Encapsulation is a fundamental principle of OOP that involves bundling data and related methods within a class, hiding the internal implementation details from other parts of the system. In the tech shop, encapsulation can be employed to encapsulate the properties and methods of classes such as Admin, Customer and Device protecting their internal state and providing controlled access to their data.

- **Inheritance:**

Inheritance is used to reduce the duplication of data. By using inheritance, attributes can only be declared in a single parent class and then all of these can be used in child classes with the help of inheritance. This approach reduces the replication of data. The attributes in the parent class can be made either protected or private.

'

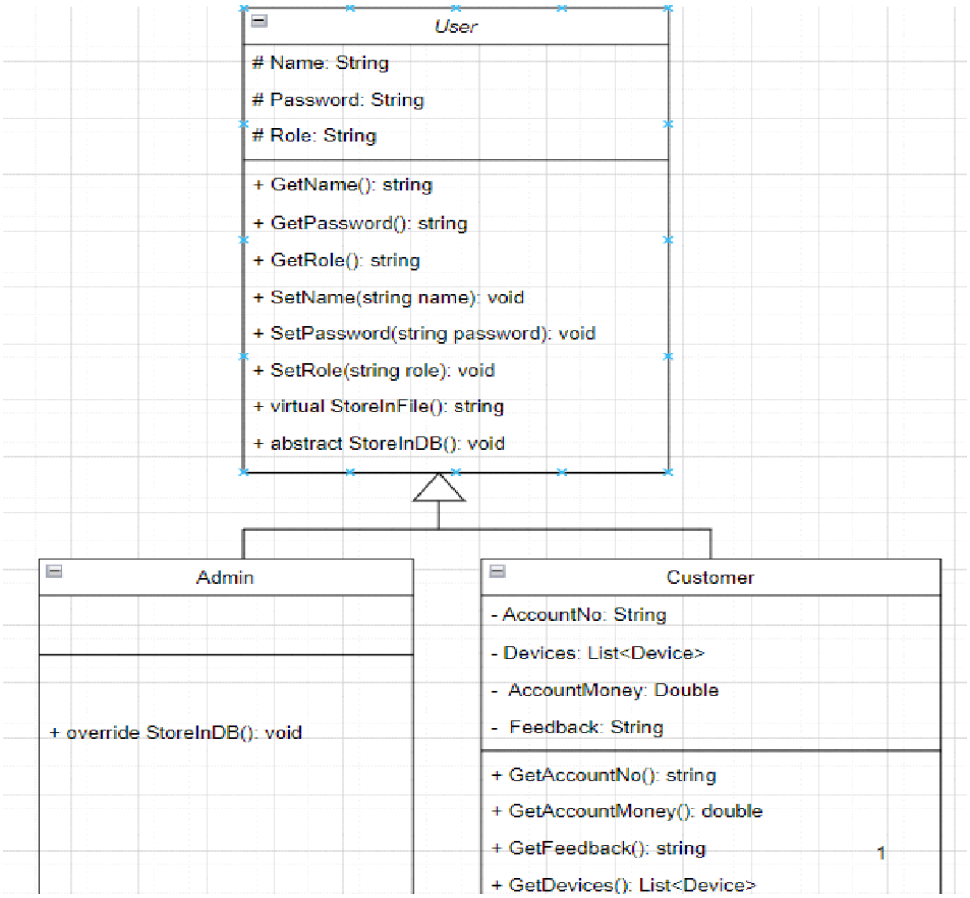


Figure 2

In the above diagram User is the parent class and Admin and Customer are the child classes. The attributes of parent class can be accessed by child classes. This is a conventional way of reducing replication.

- **Polymorphism:**

Polymorphism enables objects of different classes to be treated interchangeably, allowing for flexibility and extensibility in the system. In the blood donation management system, polymorphism can be applied when handling different types of donors and recipients. Methods or functions that operate on donors or recipients can be designed to accept objects of the general class type named Person, enabling the system to handle various specific instances transparently. It is also helpful when storing it into a list.

- **DESIGN PATTERN IMPLEMENTATION:**

The whole project is totally based upon the three types of layers i.e. BL, DL and UI. The BL folder contains all the BL classes ,the DL folder contains all the DL classes and the UI folder contains all the UI classes All the attributes, constructors and logics are placed in the BL layers. The operations that are to be performed on the data are placed in the DL layers. The UI layers contain all those things that should be displayed on the screen. Moreover all the lists are placed in the respective DL layers. All these are connected with each other through different relations. The DL layers first get access to the BL layers to get the attributes. The UI layers need DL for performing different operations on data. In this way all layers are connected with each other in some way.

- **CLASSES DETAILS:**

The major classes in the project are User and Device and their purpose and responsibilities are given.

- **User:**

This class contains all the functions that are to be performed on the users of this application like on customers and admins .This is also a parent class which further consists of Customer and Admin classes. All the responsibilities related to any of the users are performed through this class or its child classes.

- **Admin:**

Admin is the child class of user, and it is responsible for admin's operation.

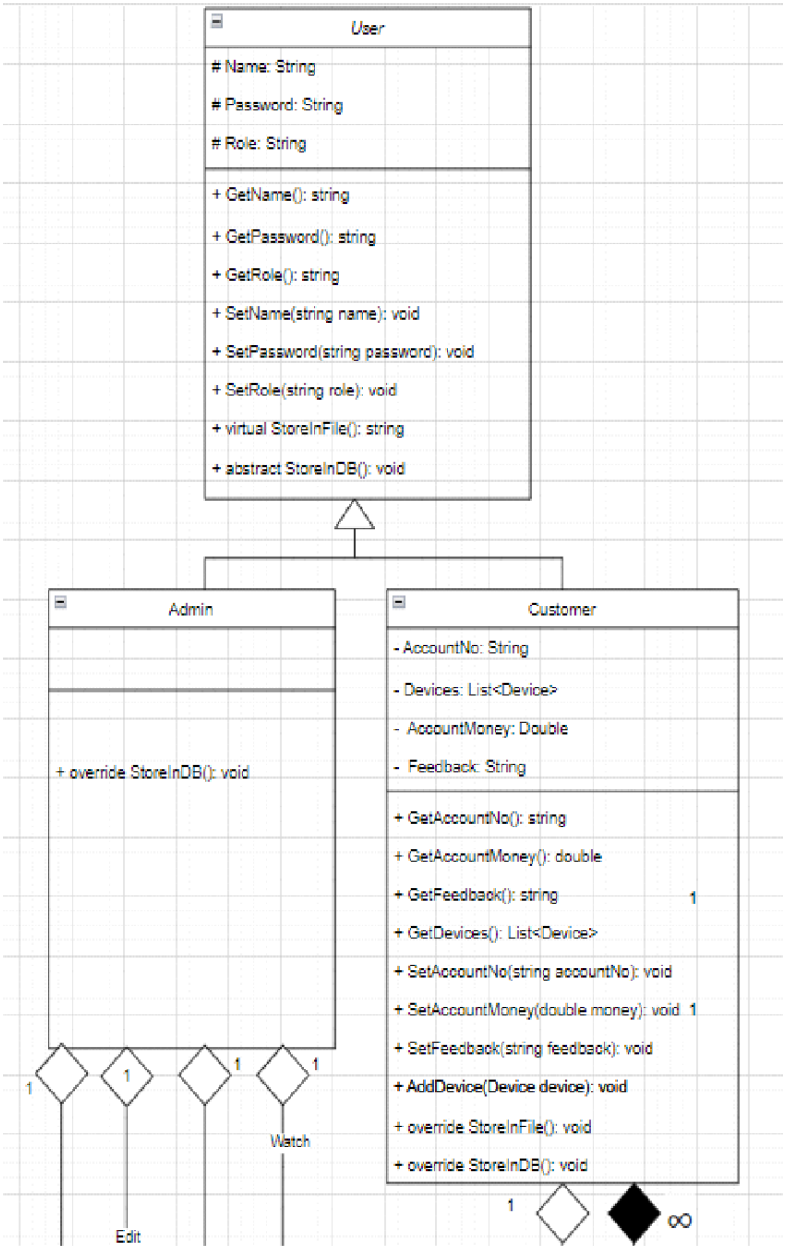
- **Customer:**

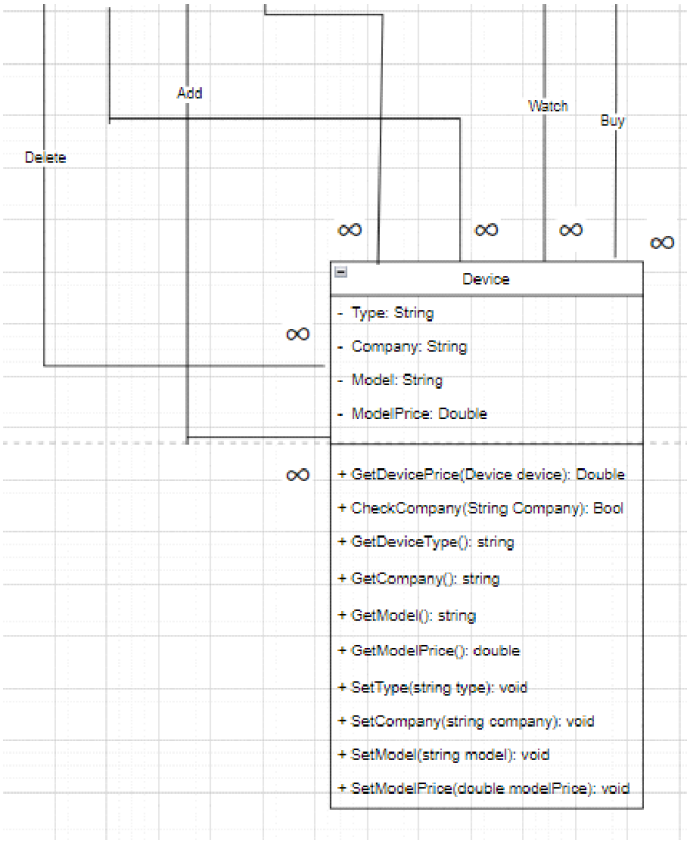
Admin is the child class of user, and it is responsible for customer's operation.

- **Device:**

  This class contains all attributes and their getter setter related to the device. All the functionalities related to any device are done using these classes. This class also interacts with other classes for several purposes like Admin class is capable of adding a new device in the shop,however their existence does not depend on one another.

- **CRC Model:**





- **Wireframes:**

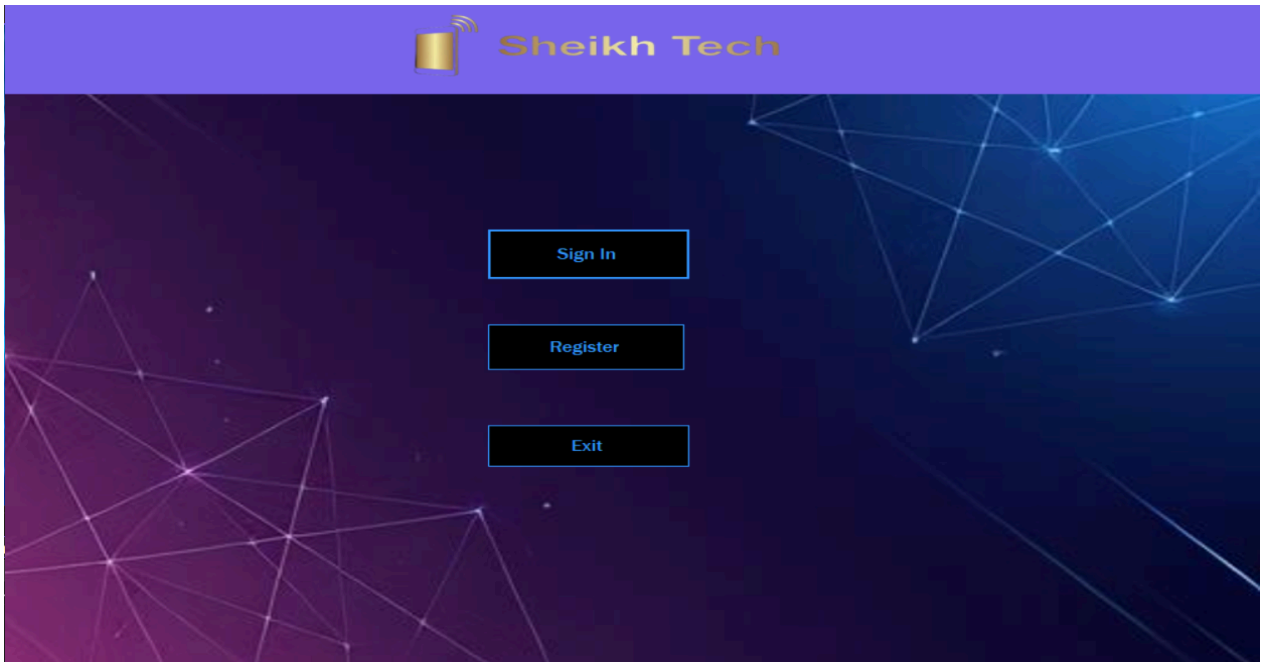


Figure 1: Home

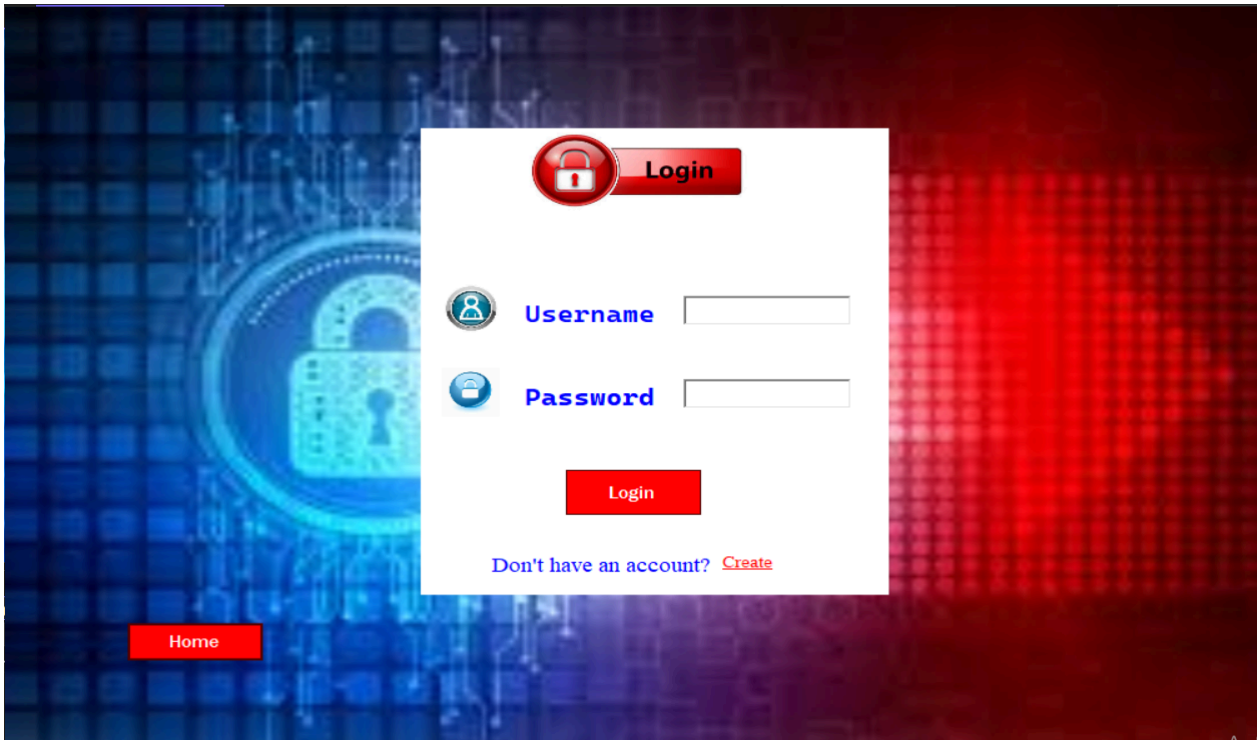


Figure 2: Login

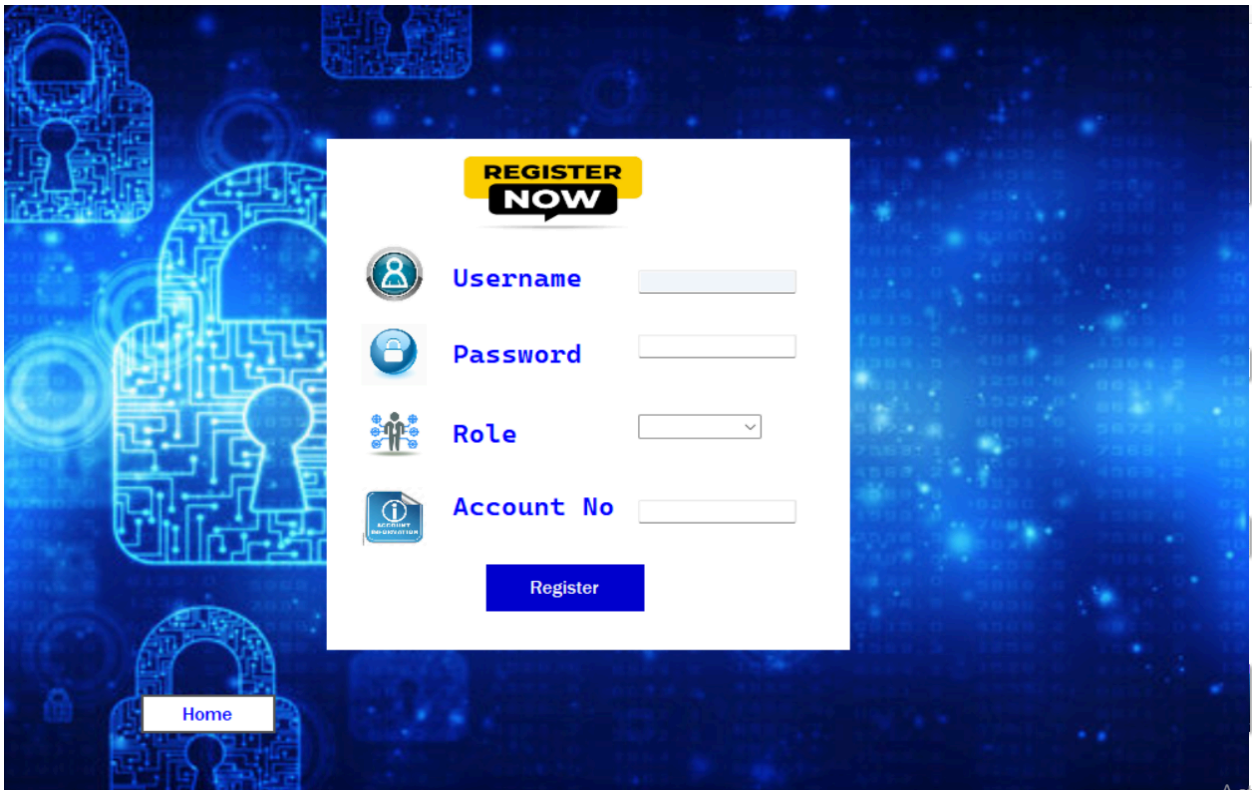


Figure 3: Sign Up

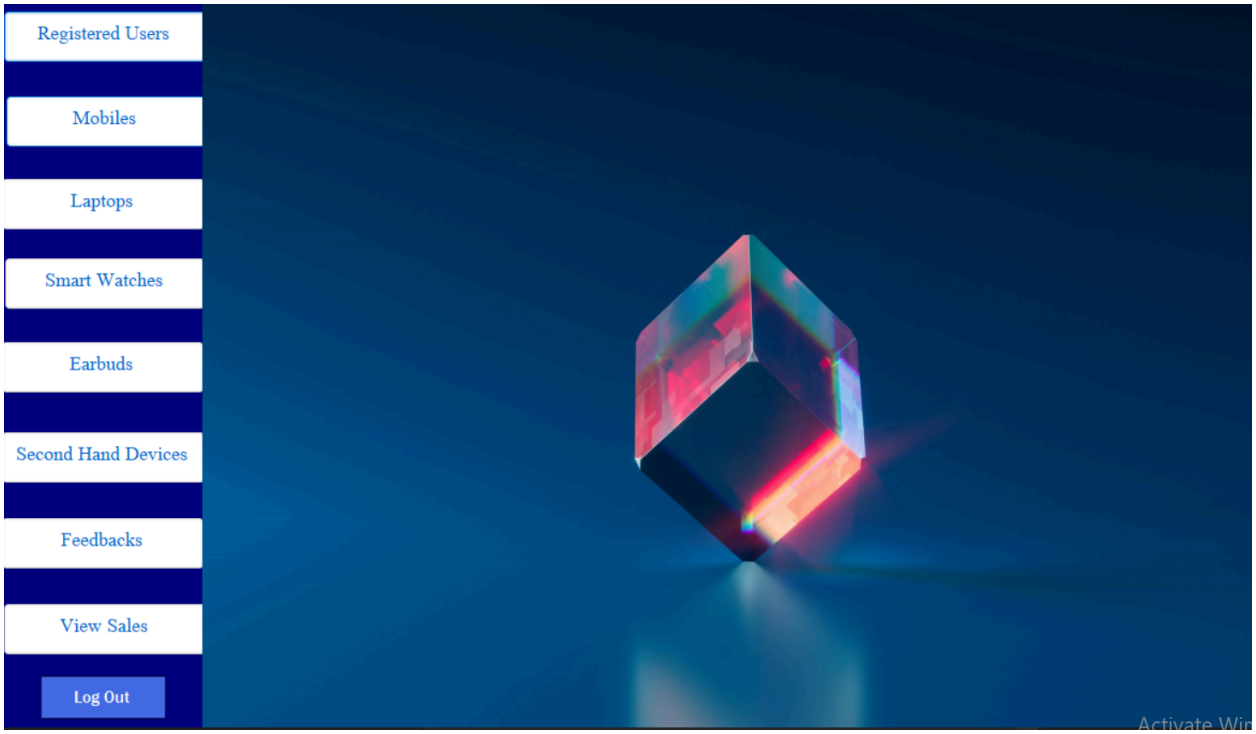


Figure 4: Admin Menu

Figure 5: Customer Menu

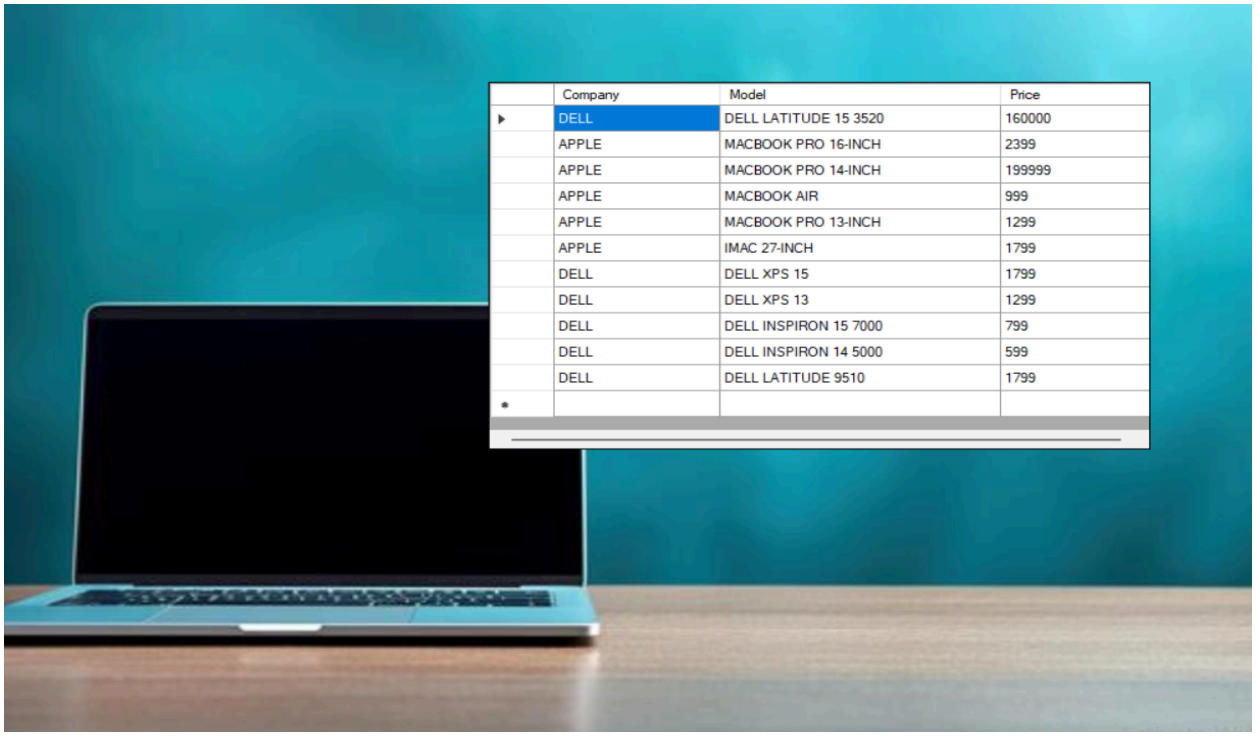


Figure 6: View Devices

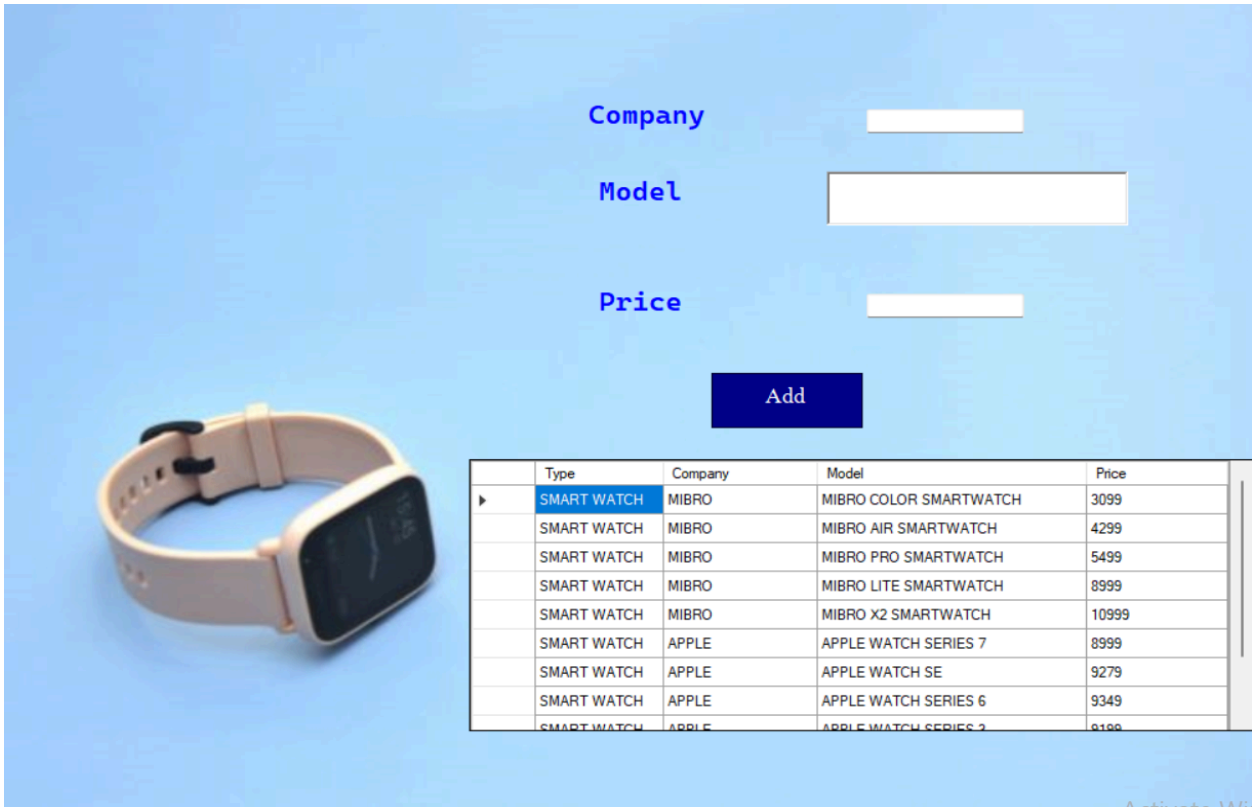


Figure 7: Add Device

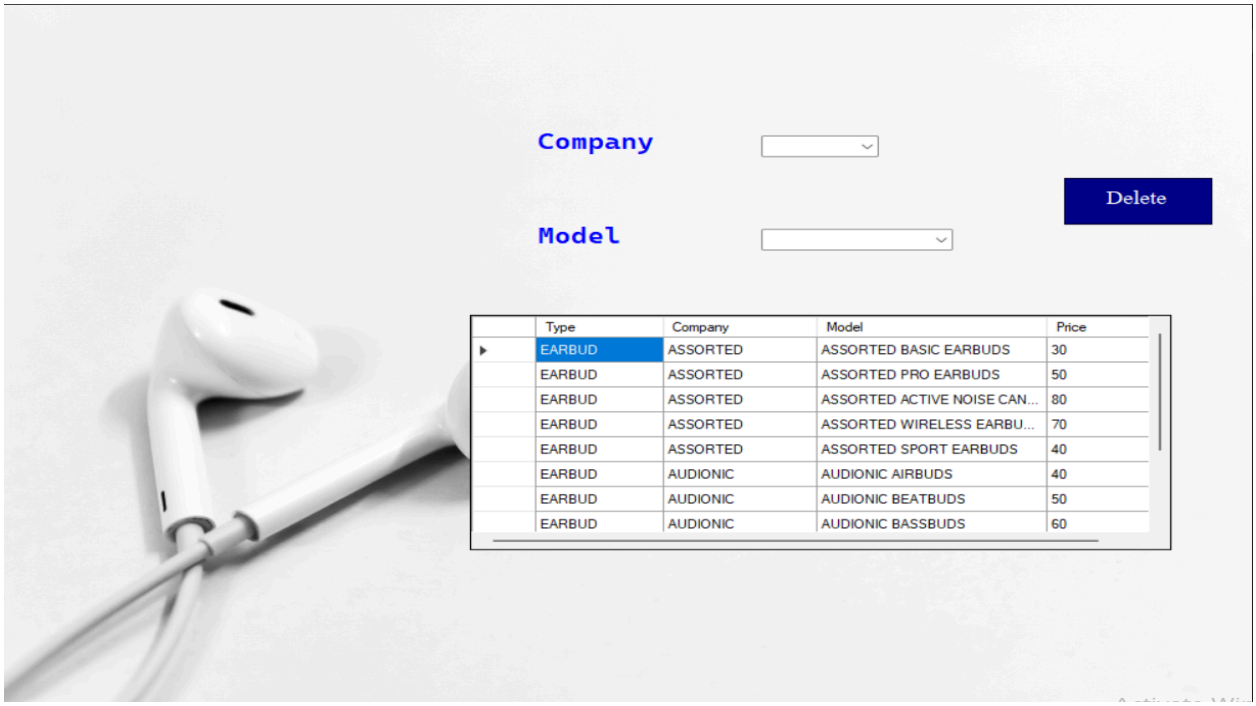


Figure 8: Delete Device

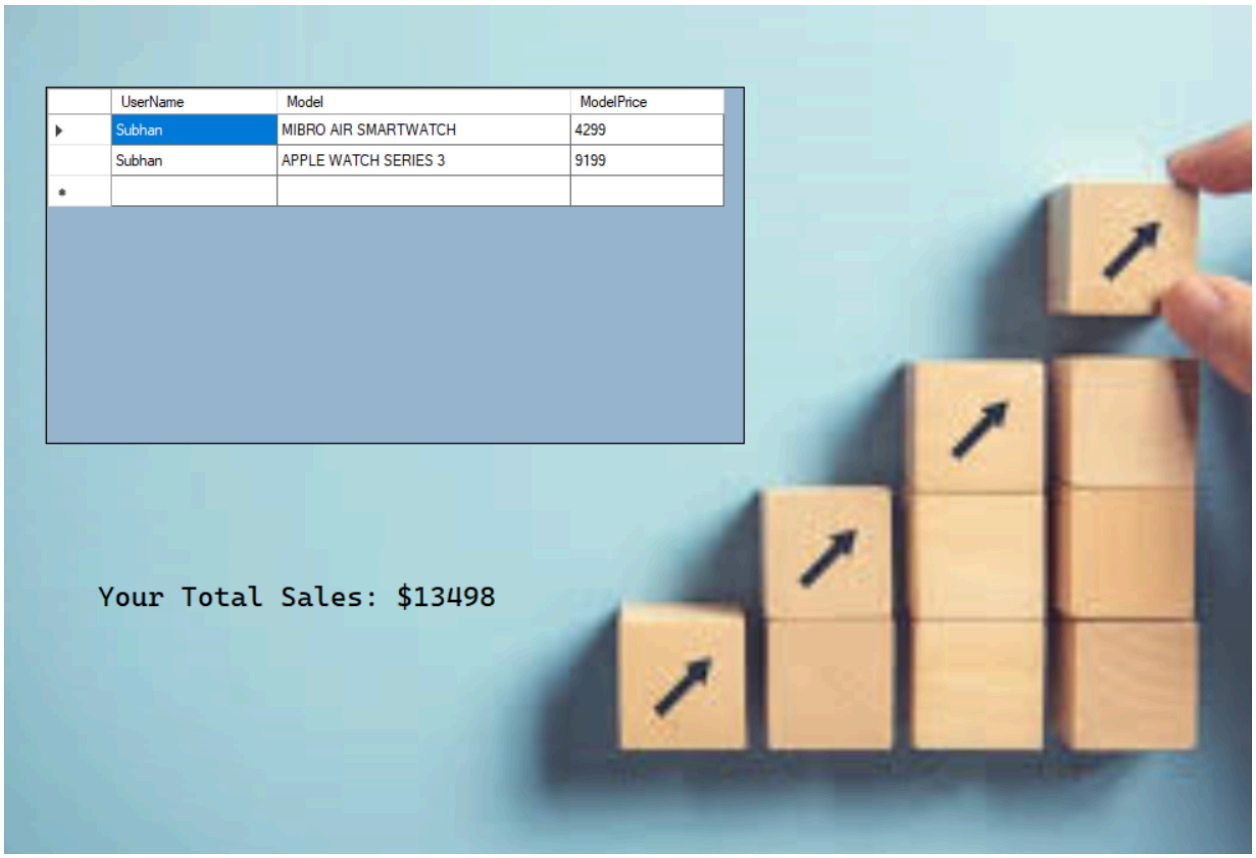


Figure 9: Sales

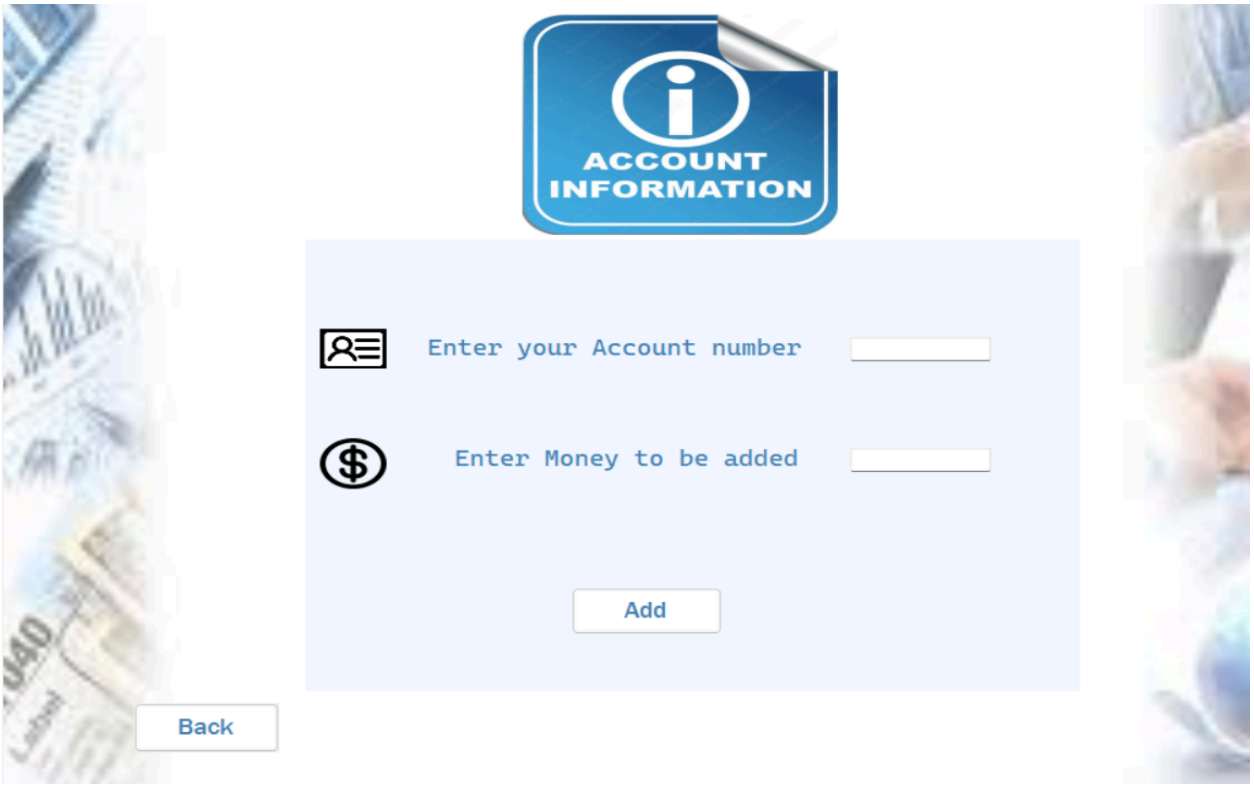


Figure 10: Account Information

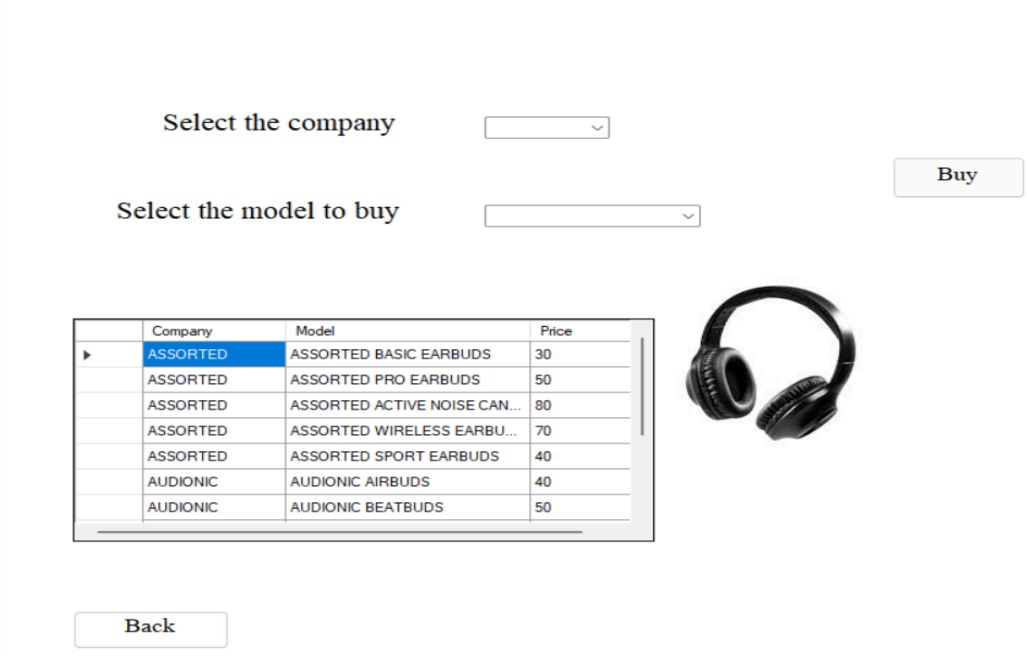


Figure 11: Buy Device

Figure 12: Bill

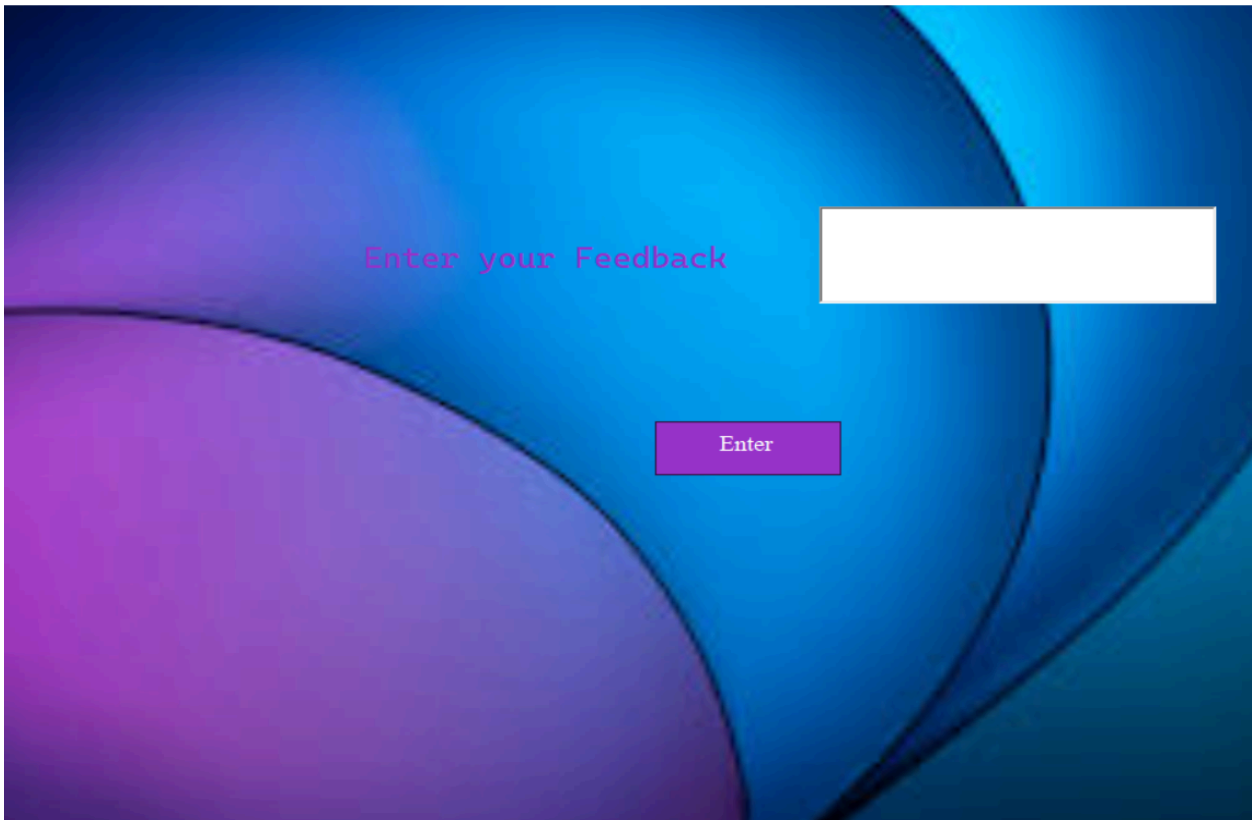


Figure 13: Feedback

•COMPLETE CODE:

•Business Logic (BL) Classes:

User:

```
public abstract class User
{
   protected string Name;
   protected string Password;
   protected string Role;
   public User(string name,string password, string role)
   {
      Name = name;
      Password = password;
      Role = role;
   }
   public string GetName() {  return Name; }
   public string GetPassword() { return Password; }
   public string GetRole() { return Role.ToUpper(); }
   public void SetName(string name) {  Name = name; }
   public void SetPassword(string password) {  Password = password; }
   public void SetRole(string role) {  Role = role; }
   public virtual string StoreInFile()
   {
      return $"{Name},{Password},{Role}";
   }
   public abstract void StoreInDB();
}
```

Admin:

```
public class Admin : User
{
   public Admin(string name, string password, string role) : base(name, password, role)
   {
   }
   public override void StoreInDB()
   {
```

```
            string ConnectionString = ConString.GetConnectionString();
            using (SqlConnection conn = new SqlConnection(ConnectionString))
            {
                conn.Open();
                SqlCommand cmd = new SqlCommand("INSERT INTO [Users] ([Username],
[Userpassword], [Userrole], [Accountno], [Accountmoney],[Feedback])
values(@Username,@Password, @Role, null, null,null)", conn);
                cmd.Parameters.AddWithValue("@Username", Name);
                cmd.Parameters.AddWithValue("@Password", Password);
                cmd.Parameters.AddWithValue("@Role", Role);
                cmd.ExecuteNonQuery();
            }
    }
}
```

Customer:

```
public class Customer: User
{
    private string AccountNo;
    private double AccountMoney;
    private List<Device> Devices;
    private string Feedback;
    public Customer(string name, string password, string role, string accountNo) :
base(name,password,role)
    {
        AccountNo = accountNo;
        AccountMoney = 0;
        Devices = new List<Device>();
    }
    public string GetAccountNo() {  return AccountNo; }
    public double GetAccountMoney() {  return AccountMoney; }
    public void SetAccountNo(string accountNo) {  AccountNo = accountNo; }
    public bool SetAccountMoney(double accountMoney)
    {
        if (accountMoney > 400000) return false;
        AccountMoney = accountMoney;
        return true;
    }
    public string GetFeedback() { return Feedback; }
    public void SetFeedback(string feedback) {  Feedback = feedback; }
```

```
    public  void AddDevice(Device device) { Devices.Add(device); }
    public List<Device> GetDevices() { return Devices; }
    public void ClearDevices() {  Devices.Clear(); }
    public override string StoreInFile()
    {
        int i = 0;
        string fileHandling =
$"{Name},{Password},{Role},{AccountNo},{AccountMoney},{Feedback},";
        foreach (Device device in Devices)
        {
            if (device != null)
            {
                fileHandling += device.GetModel();
                if (i < Devices.Count - 1)
                    fileHandling += ";";
                i++;
            }
        }
        return fileHandling;
    }
    public override void StoreInDB()
    {
        string ConnectionString = ConString.GetConnectionString();
        using (SqlConnection conn = new SqlConnection(ConnectionString))
        {
            conn.Open();
            SqlCommand cmd = new SqlCommand("INSERT INTO [Users] ([Username],
[Userpassword], [Userrole], [Accountno], [Accountmoney],[Feedback])
values(@Username,@Password, @Role, @AccountNo, 0,@Feedback)", conn);
            cmd.Parameters.AddWithValue("@Username", Name);
            cmd.Parameters.AddWithValue("@Password", Password);
            cmd.Parameters.AddWithValue("@Role", Role);
            cmd.Parameters.AddWithValue("@AccountNo", AccountNo);
            cmd.Parameters.AddWithValue("@Feedback", " ");
            cmd.ExecuteNonQuery();
        }
    }
}
```

Device:

```
public class Device{
```

```
    private string Type;
    private string Company;
    private string Model;
    private double ModelPrice;
    public Device(string type, string company, string model, double modelPrice)
    {
        Type = type;
        Company = company;
        Model = model;
        ModelPrice = modelPrice;
    }
    public string GetDeviceType() {  return Type; }
    public string GetCompany() {  return Company; }
    public string GetModel() { return Model; }
    public double GetModelPrice() {  return ModelPrice; }
    public void SetType(string type) { Type = type; }
    public void SetCompany(string company) {  Company = company; }
    public void SetModel(string model) {  Model = model; }
    public void SetPrice(double price) { ModelPrice = price; }
}
```

Data Access Layer (DL) Classes:

DeviceDB:

```
            static DeviceDB instance = null;
    public static DeviceDB GetInstance()
    {
        if(instance == null)
            instance = new DeviceDB();
        return instance;
    }
    public List<Device> GetDevices()
    { return LoadDeviceData(); }

    public Device GetDeviceByName(string name)
    {
        string ConnectionString = ConString.GetConnectionString();
        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
            string query = "SELECT * FROM Device WHERE model = @Model";
```

```
            using (SqlCommand command = new SqlCommand(query, connection))
            {
                command.Parameters.AddWithValue("@Model", name);

                connection.Open();
                SqlDataReader reader = command.ExecuteReader();

                if (reader.Read())
                {
                    string deviceType = Convert.ToString(reader["devicetype"]);
                    string company = Convert.ToString(reader["company"]);
                    string model = Convert.ToString(reader["model"]);
                    double modelPrice = Convert.ToDouble(reader["modelPrice"]);

                    return new Device(deviceType, company, model, modelPrice);
                }
                else
                {
                    return null;
                }
            }
        }
    }
    public void AddDevice(Device device)
    {
        string ConnectionString = ConString.GetConnectionString();
        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
            string query = "INSERT INTO Device (devicetype, company, model,
modelPrice) VALUES (@DeviceType, @Company, @Model, @ModelPrice)";

            using (SqlCommand command = new SqlCommand(query, connection))
            {
                connection.Open();
                command.Parameters.AddWithValue("@DeviceType",
device.GetDeviceType());
                command.Parameters.AddWithValue("@Company", device.GetCompany());
                command.Parameters.AddWithValue("@Model", device.GetModel());
                command.Parameters.AddWithValue("@ModelPrice",
device.GetModelPrice());
                command.ExecuteNonQuery();
```

```
        }
    }
}
public List<Device> LoadDeviceData()
{
    List<Device> devices = new List<Device>();

    string ConnectionString = ConString.GetConnectionString();
    using (SqlConnection connection = new SqlConnection(ConnectionString))
    {
        string query = "SELECT devicetype, company, model, modelPrice FROM
Device";

        using (SqlCommand command = new SqlCommand(query, connection))
        {
            connection.Open();
            SqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                string deviceType = Convert.ToString(reader["devicetype"]);
                string company = Convert.ToString(reader["company"]);
                string model = Convert.ToString(reader["model"]);
                double modelPrice = Convert.ToDouble(reader["modelPrice"]);

                Device device = new Device(deviceType, company, model, modelPrice);
                devices.Add(device);
            }

            reader.Close();
        }
    }
    return devices;
}
public bool EditPrice(string deviceType,string company, string model, double
modelPrice)
{
    string ConnectionString = ConString.GetConnectionString();
    using (SqlConnection connection = new SqlConnection(ConnectionString))
    {
```

```
            string query = "UPDATE Device SET modelPrice = @ModelPrice WHERE
company = @Company AND model = @Model";

            using (SqlCommand command = new SqlCommand(query, connection))
            {
                command.Parameters.AddWithValue("@ModelPrice", modelPrice);
                command.Parameters.AddWithValue("@Company", company);
                command.Parameters.AddWithValue("@Model", model);

                connection.Open();
                int rowsAffected = command.ExecuteNonQuery();
                return rowsAffected > 0;
            }
        }
    }
    public bool DeleteModel(string company, string model)
    {
        string ConnectionString = ConString.GetConnectionString();
        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
            string query = "DELETE FROM Device WHERE company = @Company AND
model = @Model";

            using (SqlCommand command = new SqlCommand(query, connection))
            {
                command.Parameters.AddWithValue("@Company", company);
                command.Parameters.AddWithValue("@Model", model);

                connection.Open();
                int rowsAffected = command.ExecuteNonQuery();

                return rowsAffected > 0;
            }
        }
    }

    public bool ModelExisted(string buyModel)
    {
        string ConnectionString = ConString.GetConnectionString();
        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
```

```
        string query = "SELECT COUNT(*) FROM Device WHERE model =
@Model";

        using (SqlCommand command = new SqlCommand(query, connection))
        {
            command.Parameters.AddWithValue("@Model", buyModel);

            connection.Open();
            int count = Convert.ToInt32(command.ExecuteScalar());

            return count > 0;
        }
    }
}

public double GetDevicePrice(string buyModel)
{
    string ConnectionString = ConString.GetConnectionString();
    using (SqlConnection connection = new SqlConnection(ConnectionString))
    {
        string query = "SELECT modelPrice FROM Device WHERE model =
@Model";

        using (SqlCommand command = new SqlCommand(query, connection))
        {
            connection.Open();
            command.Parameters.AddWithValue("@Model", buyModel);

            object result = command.ExecuteScalar();

            if (result != null )
            {
                return Convert.ToDouble(result);
            }
            else
            {
                return 0.0;
            }
        }
    }
}
```

```
    public void DeviceSold(string name, string model, double modelPrice)
    {
        string ConnectionString = ConString.GetConnectionString();
        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
            string query = "INSERT INTO SoldDevices (Username, Model, ModelPrice)
VALUES (@Name, @Model, @ModelPrice)";

            using (SqlCommand command = new SqlCommand(query, connection))
            {
                connection.Open();
                command.Parameters.AddWithValue("@Name", name);
                command.Parameters.AddWithValue("@Model", model);
                command.Parameters.AddWithValue("@ModelPrice", modelPrice);
                command.ExecuteNonQuery();
            }
        }
    }

    public List<string> GetSoldDevices()
    {
        List<string> devices = new List<string>();
        string ConnectionString = ConString.GetConnectionString();
        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
            string query = "SELECT Username, Model, ModelPrice FROM SoldDevices";

            using (SqlCommand command = new SqlCommand(query, connection))
            {
                connection.Open();
                SqlDataReader reader = command.ExecuteReader();

                while (reader.Read())
                {
                    string Username = Convert.ToString(reader["Username"]);
                    string model = Convert.ToString(reader["Model"]);
                    string modelPrice = Convert.ToString(reader["ModelPrice"]);

                    devices.Add($"{Username},{model},{modelPrice}");
                }
```

```
            reader.Close();
        }
    }
    return devices;
  }
}
```

DeviceFH:

```
public class DeviceFH:IDeviceDL
{
  public static List<Device> Devices = new List<Device>();
  static DeviceFH instance = null;
  public static DeviceFH GetInstance()
  {
    if (instance == null)
      instance = new DeviceFH();
    return instance;
  }
  public List<Device> GetDevices()
  { return Devices; }

  public Device GetDeviceByName(string name)
  {
    foreach(Device device in Devices)
      if(device.GetModel().Equals(name))
        return device;
    return null;
  }
  public void AddDevice(Device device)
  {
    Devices.Add(device);
    SaveDeviceData();
  }
  public List<Device> LoadDeviceData()
  {
    Devices.Clear();
    string record;
    StreamReader file = new StreamReader("Device.txt");
    while ((record = file.ReadLine()) != null)
    {
```

```
            string[] devicess = record.Split(',');
            Devices.Add(new Device(devicess[0], devicess[1], devicess[2],
double.Parse(devicess[3])));
        }
        file.Close();
        return Devices;
    }
    public void SaveDeviceData()
    {
        using (StreamWriter writer = new StreamWriter("Device.txt",false))
        {
            foreach (Device device in Devices)
            {

writer.WriteLine($"{device.GetDeviceType()},{device.GetCompany()},{device.GetMo
del()},{device.GetModelPrice()}");
            }
        }
    }
    public bool EditPrice(string deviceType,string company,string model, double
modelPrice)
    {
        foreach (Device device in Devices)
        {
            if ((company == null || device.GetCompany() == company ) &&
device.GetModel() == model)
            {
                device.SetPrice(modelPrice);
                SaveDeviceData();
                return true;
            }
        }
        return false;
    }
    public bool DeleteModel(string company, string model)
    {
        for (int j = 0; j < Devices.Count; j++)
        {
            if ((Devices[j].GetCompany() == company || company == null) &&
Devices[j].GetModel() == model)
            {
```

```
            Devices.RemoveAt(j);
            SaveDeviceData();
            return true;
        }
    }
    return false;
}
public bool ModelExisted(string buyModel)
{
    foreach (Device device in Devices)
        if (buyModel.Equals(device.GetModel()))
            return true;
    return false;
}
public double GetDevicePrice(string buyModel)
{
    for (int i = 0; i < Devices.Count; i++)
    {
        if (Devices[i].GetModel() == buyModel)
        {
            return Devices[i].GetModelPrice();
        }
    }
    return 0.0;
}

public void DeviceSold(string name, string model, double modelPrice)
{
    using (StreamWriter writer = new StreamWriter("SoldDevices.txt", true))
    {
        writer.WriteLine($"{ name},{model},{modelPrice}");
    }
}

public List<string> GetSoldDevices()
{
    List<string> devices = new List<string>();
    string record;
    StreamReader file = new StreamReader("SoldDevices.txt");
    while ((record = file.ReadLine()) != null)
    {
```

```
            devices.Add(record);
        }
        file.Close();
        return devices;
    }
}
```

UserDB:

```
    static UserDB instance = null;
    public static UserDB GetInstance()
    {
        if (instance == null)
            instance = new UserDB();
        return instance;
    }
    public List<User> LoadUsers()
    {
        string ConnectionString = ConString.GetConnectionString();
        List<User> users = new List<User>();

        string query = "SELECT * FROM Users";

        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
            using (SqlCommand command = new SqlCommand(query, connection))
            {
                connection.Open();

                using (SqlDataReader reader = command.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        string role = Convert.ToString(reader["Userrole"]);

                        if (role == "ADMIN")
                        {
                            Admin admin = new Admin(
                                Convert.ToString(reader["Username"]),
                                Convert.ToString(reader["Userpassword"]),
                                role
                            );
```

```
                    users.Add(admin);
                }
                else if (role == "CUSTOMER")
                {
                    Customer customer = new Customer(
                        Convert.ToString(reader["Username"]),
                        Convert.ToString(reader["Userpassword"]),
                        role,
                        Convert.ToString(reader["Accountno"])
                    );

customer.SetAccountMoney(Convert.ToDouble(reader["Accountmoney"]));
                    customer.SetFeedback(Convert.ToString(reader["Feedback"]));
                    users.Add(customer);
                }
            }
        }
    }
        return users;
}
public List<User> GetUsers() { return LoadUsers(); }
public void StoreAllUsers()
{
    return;
}
public void SaveUserData(User user)
{
        user.StoreInDB();
}
public void SaveUserDevice(Customer customer,Device device)
{
    string connectionString = ConString.GetConnectionString();

    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open();

        string sql = "INSERT INTO UserDevice (Username, Model) VALUES
(@Username, @Model)";
        using (SqlCommand cmd = new SqlCommand(sql, conn))
```

```
        {
            cmd.Parameters.AddWithValue("@Username", customer.GetName());
            cmd.Parameters.AddWithValue("@Model", device.GetModel());
            cmd.ExecuteNonQuery();
        }
    }
}
public bool RemoveUser(User user)
{
    string connectionString = ConString.GetConnectionString();

    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open();
        string sql = "DELETE FROM Users WHERE Username = @Username";
        using (SqlCommand cmd = new SqlCommand(sql, conn))
        {
            cmd.Parameters.AddWithValue("@Username", user.GetName());
            int rowsAffected = cmd.ExecuteNonQuery();
            if (rowsAffected > 0)
                return true;
            else
                return false;
        }
    }
}
public User SignIn(string name,string password)
{
    List<User> UsersList = LoadUsers();

    foreach(User user in UsersList)
    {
        if (user.GetName()==name && user.GetPassword()==password)
        {
            return user;
        }
    }
    return null;
}
public int CheckUserName(string word)
{
```

```
    // Check if the username already exists
    if (ExistedUsername(word))
    {
        return 0;
    }
    // Check the length of username to be 6 or greater than 6
    if (word.Length > 5)
    {
        int letterCount = 0;
        // Count the number of letters in the username
        for (int i = 0; i < word.Length; i++)
        {
            char c = word[i];

            if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z'))
            {
                letterCount++;
            }
            else if (!(c >= '0' && c <= '9'))
            {
                return 1; // Username contains non-alphanumeric characters
            }
        }

        // Determine the validity of the username based on letter count
        if (letterCount >= 3)
        {
            return 2; // Only true case for username
        }
        else
        {
            return 3; // Username does not contain 3 characters
        }
    }
    else
    {
        return 4; // Username is less than 6 characters
    }
}
public bool ExistedUsername(string name)
{
```

```
        string ConnectionString = ConString.GetConnectionString();
        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
            connection.Open();
            using (SqlCommand command = new SqlCommand("SELECT COUNT(*)
FROM Users WHERE Username = @Name", connection))
            {
                command.Parameters.AddWithValue("@Name", name);
                int count = (int)command.ExecuteScalar();
                return count > 0;
            }
        }
    }
    public List<Device> GetUserDevices(Customer customer)
    {
        List<Device> userDevices = new List<Device>();

        string connectionString = ConString.GetConnectionString();
        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            conn.Open();
            string sql = "SELECT model FROM UserDevice WHERE Username =
@Username";
            using (SqlCommand cmd = new SqlCommand(sql, conn))
            {
                cmd.Parameters.AddWithValue("@Username", customer.GetName());
                using (SqlDataReader reader = cmd.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        IDeviceDL deviceDL = new DeviceDB();

userDevices.Add(deviceDL.GetDeviceByName(Convert.ToString(reader["model"])));
                    }
                }
            }
        }
        return userDevices;
    }
    public bool AccountExisted(string account)
    {
```

```
        string ConnectionString = ConString.GetConnectionString();
        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
            connection.Open();
            using (SqlCommand command = new SqlCommand("SELECT COUNT(*)
FROM Users WHERE Accountno = @account", connection))
            {
                command.Parameters.AddWithValue("@account", account);
                int count = (int)command.ExecuteScalar();
                return count > 0;
            }
        }
    }
    public bool CheckAccount(string aNo, string userAccount)
    {
        for (int i = 0; i < aNo.Length; i++)
        {
            if (aNo[i] != userAccount[i])
            {
                return false;
            }
        }
        return true;
    }
    public void AddFeedback(Customer customer, string feedback)
    {
        string connectionString = ConString.GetConnectionString();

        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            conn.Open();
            string sql = "UPDATE Users SET Feedback = @Feedback WHERE Username =
@Username";
            using (SqlCommand cmd = new SqlCommand(sql, conn))
            {
                cmd.Parameters.AddWithValue("@Feedback", feedback);
                cmd.Parameters.AddWithValue("@Username", customer.GetName());
                cmd.ExecuteNonQuery();
            }
        }
    }
```

```
    public bool UpdateMoney(Customer customer, double amount)
    {
        if(customer.GetAccountMoney()+amount > 400000)
            return false;
        string ConnectionString = ConString.GetConnectionString();
        using(SqlConnection conn = new SqlConnection(ConnectionString))
        {
            conn.Open();
            string query = "Update Users Set AccountMoney = @AccountMoney where
Username = @Username";
            using(SqlCommand cmd = new SqlCommand(query,conn))
            {
                cmd.Parameters.AddWithValue("@Username", customer.GetName());
                cmd.Parameters.AddWithValue("@AccountMoney", amount);
                cmd.ExecuteNonQuery();
            }
        }
        return true;
    }
    public double AccountMoney(Customer cust)
    {
        double accountMoney = 0.0;
        string connectionString = ConString.GetConnectionString();
        string query = "SELECT AccountMoney FROM Users WHERE Username =
@Username";
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            using (SqlCommand command = new SqlCommand(query, connection))
            {
                connection.Open();
                command.Parameters.AddWithValue("@Username", cust.GetName());
                object result = command.ExecuteScalar();
                if (result != null)
                {
                    double.TryParse(result.ToString(), out accountMoney);
                }
            }
        }
        return accountMoney;
    }
    public List<string> GetCurrentCustomerNames()
```

```
    {

        string ConnectionString = ConString.GetConnectionString();
        List<string> names = new List<string>();

        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
            string query = "SELECT Username FROM Users WHERE Userrole =
'CUSTOMER'";

            using (SqlCommand command = new SqlCommand(query, connection))
            {
                connection.Open();
                SqlDataReader reader = command.ExecuteReader();

                while (reader.Read())
                {
                    string name = reader.GetString(reader.GetOrdinal("Username"));
                    names.Add(name);
                }
            }
        }
        return names;
    }
    public List<string> GetFeedbacks()
    {
        string ConnectionString = ConString.GetConnectionString();
        List<string> feedbacks = new List<string>();

        using (SqlConnection connection = new SqlConnection(ConnectionString))
        {
            string query = "SELECT feedback FROM Users WHERE feedback IS NOT
NULL";

            using (SqlCommand command = new SqlCommand(query, connection))
            {
                connection.Open();
                SqlDataReader reader = command.ExecuteReader();

                while (reader.Read())
                {
```

```
                string feedback = reader.GetString(reader.GetOrdinal("feedback"));
                feedbacks.Add(feedback);
            }
        }
    }

    return feedbacks;
}

public double GetSales()
{
    double sales = 0.0;
    string connectionString = ConString.GetConnectionString();
    string query = "SELECT SUM(ModelPrice) FROM SoldDevices";
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        using (SqlCommand command = new SqlCommand(query, connection))
        {
            connection.Open();
            object result = command.ExecuteScalar();
            if (result != null)
            {
                sales = (double)result;
            }
        }
    }
    return sales;
}
}
```

UserFH:

```
static UserFH instance = null;
public static UserFH GetInstance()
{
    if (instance == null)
        instance = new UserFH();
    return instance;
}
public static List<User> UsersList = new List<User>();
public List<User> LoadUsers()
{
```

```
    UsersList.Clear();
    string record;
    StreamReader file = new StreamReader("Users.txt");
    while ((record = file.ReadLine()) != null)
    {
        string[] userData = record.Split(',');
        string name = userData[0];
        string password = userData[1];
        string role = userData[2];

        if (role.ToUpper() == "ADMIN")
        {
            UsersList.Add(new Admin(name, password, role));
        }
        else if (role.ToUpper() == "CUSTOMER")
        {
            string accountNo = userData[3];
            double accountMoney = double.Parse(userData[4]);
            string feedback = userData[5];
            List<Device> devices = new List<Device>();

            Customer customer = new Customer(name, password, role, accountNo);
            if (userData.Length > 6)
            {
                string[] devicesData = userData[6].Split(';');
                foreach (string deviceData in devicesData)
                {
                    IDeviceDL deviceFH = new DeviceFH();
                    customer.AddDevice(deviceFH.GetDeviceByName(deviceData));
                }
            }
            customer.SetAccountMoney(accountMoney);
            customer.SetFeedback(feedback);
            UsersList.Add(customer);
        }
    }
    file.Close();
    return UsersList;
}
public List<User> GetUsers() {  return UsersList; }
public void StoreAllUsers()
```

```
{
    using (StreamWriter file = new StreamWriter("Users.txt", false))
    {
        foreach (User userr in UsersList)
        {
            file.WriteLine(userr.StoreInFile());
        }
    }
}
public List<Device> GetUserDevices(Customer customer)
{
    return customer.GetDevices();
}
public void SaveUserData(User user)
{
    UsersList.Add(user);
}
public void SaveUserDevice(Customer customer, Device device)
{
    if (customer.GetDevices()[0] == null) {customer.ClearDevices(); }
    customer.AddDevice(device);
}
public bool RemoveUser(User user)
{ return UsersList.Remove(user); }
public int CheckUserName(string word)
{
    if (ExistedUsername(word))
    {
        return 0;
    }

    if (word.Length > 5)
    {
        int letterCount = 0;

        for (int i = 0; i < word.Length; i++)
        {
            char c = word[i];

            if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z'))
            {
```

```
                letterCount++;
            }
            else if (!(c >= '0' && c <= '9'))
            {
                return 1;
            }
        }

        if (letterCount >= 3)
        {
            return 2;
        }
        else
        {
            return 3;
        }
    }
    else
    {
        return 4;
    }
}
public bool ExistedUsername(string name)
{
    for (int i = UsersList.Count - 1; i >= 0; i--)
    {
        if (name == UsersList[i].GetName())
        {
            return true;
        }
    }
    return false;
}
public User SignIn(string name, string password)
{
    foreach (User user in UsersList)
    {
        if (user.GetName() == name && user.GetPassword() == password)
        {
            return user;
        }
```

```
    }
    return null;
}
public bool CheckAccount(string aNo, string userAccount)
{
    for (int i = 0; i < aNo.Length; i++)
    {
        if (aNo[i] != userAccount[i])
        {
            return false;
        }
    }
    return true;
}
public bool AccountExisted(string account)
{
    foreach(User user in UsersList)
    {
        if (user is Customer)
        {
            Customer customer = (Customer)user;
            if (account == customer.GetAccountNo())
            {
                return true;
            }
        }

    }
    return false;
}
public void AddFeedback(Customer customer, string feedback)
{
    customer.SetFeedback(feedback);
}
public bool UpdateMoney(Customer customer, double amount)
{
    return customer.SetAccountMoney(amount);
}
public double AccountMoney(Customer cust)
{
    return cust.GetAccountMoney();
```

```
    }
    public List<string> GetCurrentCustomerNames()
    {
        List<string> names = new List<string>();
        foreach (User user in UsersList)
            if(user.GetRole()=="CUSTOMER")
                names.Add(user.GetName());
        return names;
    }
    public List<string> GetFeedbacks()
    {
        List<string> result = new List<string>();
        foreach (User user in UsersList)
        {
            if (user is Customer)
            {
                Customer customer = (Customer)user;
                result.Add(customer.GetFeedback());
            }
        }
        return result;
    }

    public double GetSales()
    {
        string record;
        double sales = 0;
        StreamReader file = new StreamReader("SoldDevices.txt");
        while ((record = file.ReadLine()) != null)
        {
            string[] data = record.Split(',');
            sales += Convert.ToDouble(data[2]);
        }
        file.Close();
        return sales;
    }
}
```

- **User Interface (UI) Forms:**

Home:

```
public Sign()
{
   InitializeComponent();
}

private void login_Click(object sender, EventArgs e)
{
   Hide();
   Form form = new SignIn();
   form.ShowDialog();
}

private void register_Click(object sender, EventArgs e)
{
   Hide();
   Form form = new SignUp();
   form.ShowDialog();
}

private void Exit_Click(object sender, EventArgs e)
{
   ObjectHandler.GetUserDL().StoreAllUsers();
   Close();
}
```

SignIn:

```
public static Customer currentUser;
public static Customer GetCurrentUser() {  return currentUser; }
public SignIn()
{
   InitializeComponent();
}
private void home_Click(object sender, EventArgs e)
{
   Hide();
   Form form = new Sign();
   form.ShowDialog();
```

```
}
private void button1_Click(object sender, EventArgs e)
{
   IUserDL Iuser = ObjectHandler.GetUserDL();

   string Name = richTextBox1.Text;
   string Password = richTextBox2.Text;
   List<string> users = Iuser.GetCurrentCustomerNames();
   User user = Iuser.SignIn(Name, Password);
   if (user != null && user.GetRole() == "ADMIN")
   {
      MessageBox.Show("You have successfully logged in as Admin", "SignIn",
MessageBoxButtons.OK, MessageBoxIcon.Information);
      Hide();
      Form adminUI = new AdminForm();
      adminUI.ShowDialog();
   }
   else if (user != null && user.GetRole() == "CUSTOMER")
   {
      currentUser = (Customer)user;
      MessageBox.Show("You have successfully logged in as Customer", "SignIn",
MessageBoxButtons.OK, MessageBoxIcon.Information);
      Hide();
      CustomerUI customerUI = new CustomerUI();
      customerUI.ShowDialog();
   }
   else
   {
      MessageBox.Show("You are not registered yet.", "SignIn",
MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
}

private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
   Hide();
   Form form = new SignUp();
   form.ShowDialog();
}
```

SignUp:

```
public SignUp()
{
   InitializeComponent();
}

private void home_Click(object sender, EventArgs e)
{
   Hide();
   Form form = new Sign();
   form.ShowDialog();
}

private void button1_Click(object sender, EventArgs e)
{
   IUserDL Iuser = ObjectHandler.GetUserDL();
   string name = textBox1.Text;
   int checkResult = Iuser.CheckUserName(name);

   switch (checkResult)
   {
      case 0:
         MessageBox.Show("UserName Already Taken", "SignUp",
MessageBoxButtons.OK, MessageBoxIcon.Error);
         return;
      case 1:
         MessageBox.Show("Invalid character in username. Use only letters and
numbers.", "SignUp", MessageBoxButtons.OK, MessageBoxIcon.Error);
         return;
      case 3:
         MessageBox.Show("Invalid username. Username must contain at least 3
letters.", "SignUp", MessageBoxButtons.OK, MessageBoxIcon.Error);
         return;
      case 4:
         MessageBox.Show("Invalid username. Username must be at least 6 characters
long.", "SignUp", MessageBoxButtons.OK, MessageBoxIcon.Error);
         return;
      default:
         break;
   }
   string password = textBox2.Text;
   if (password.Length < 8)
```

```
    {
        MessageBox.Show("Password should be at least 8 characters long", "SignUp",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    if (comboBox1.Text.ToUpper() == "ADMIN")
    {
        Iuser.SaveUserData(new Admin(name, password, "ADMIN"));
    } else
    {
        string account = textBox3.Text;
        if(account.Length != 13)
        {

            MessageBox.Show("Account Number must be equal to 13 DIGITS", "SignUp",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
        else if(ObjectHandler.GetUserDL().AccountExisted(account))
        {
            MessageBox.Show("Account Number cannot be repeated", "SignUp",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
        Iuser.SaveUserData(new Customer(name, password, "CUSTOMER",
textBox3.Text));
    }
    MessageBox.Show("You are registered successfully", "SignUp",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    ObjectHandler.GetUserDL().StoreAllUsers();
}

private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (comboBox1.Text == "Admin")
    {
        textBox3.Enabled= false;
        textBox3.Text = string.Empty;
    }
    else
        textBox3.Enabled = true;}
```

WatchUsers:

```
public WatchUsers()
{
   InitializeComponent();
   List<User> users = ObjectHandler.GetUserDL().GetUsers();
   ShowRegisteredUsers(users);
   dataGridView1.Columns["Username"].Width = 170;
   dataGridView1.Columns["Role"].Width = 120;
}
private void ShowRegisteredUsers(List<User> users)
{
   DataTable dataTable = new DataTable();
   dataTable.Columns.Add("UserName",typeof(string));
   dataTable.Columns.Add("Role",typeof(string));

   foreach(User user in users)
   {
      dataTable.Rows.Add(user.GetName(),user.GetRole());
   }
   dataGridView1.DataSource = dataTable;
}

private void button1_Click(object sender, EventArgs e)
{
   Hide();
   Form form = new AdminForm();
   form.ShowDialog();
}
```

Watch:

```
public Watch()
{
   InitializeComponent();
   ShowDevice();
   if (ObjectHandler.GetDeviceType() == "MOBILE")
      BackgroundImage =
Image.FromFile("D:\\Tayyab\\BusinessAppProject\\BusinessApp(Forms)\\BusinessApp(
Forms)\\Mobile2.jpeg");
   else if (ObjectHandler.GetDeviceType() == "LAPTOP")
```

```
        BackgroundImage =
Image.FromFile("D:\\Tayyab\\BusinessAppProject\\BusinessApp(Forms)\\BusinessApp(
Forms)\\laptop.jpg");
    else if (ObjectHandler.GetDeviceType() == "SMART WATCH")
        BackgroundImage =
Image.FromFile("D:\\Tayyab\\BusinessAppProject\\BusinessApp(Forms)\\BusinessApp(
Forms)\\SW4.jpg");
    else if (ObjectHandler.GetDeviceType() == "EARBUD")
        BackgroundImage =
Image.FromFile("D:\\Tayyab\\BusinessAppProject\\BusinessApp(Forms)\\BusinessApp(
Forms)\\ear.jpg");
    else
        BackgroundImage =
Image.FromFile("D:\\Tayyab\\BusinessAppProject\\BusinessApp(Forms)\\BusinessApp(
Forms)\\SH.jpeg");

    BackgroundImageLayout = ImageLayout.Stretch;

    if (ObjectHandler.GetDeviceType() != "SECOND HAND DEVICE")
        dataGridView1.Columns["Company"].Width = 130;
    dataGridView1.Columns["Model"].Width = 220;
    dataGridView1.Columns["Price"].Width = 120;
}
private void button1_Click(object sender, EventArgs e)
{
    Hide();
    Form form = new CRUDMenu();
    form.ShowDialog();
}
private void ShowDevice()
{
    List<Device> devices = ObjectHandler.GetDeviceDL().GetDevices();
    DataTable dataTable = new DataTable();
    if (ObjectHandler.GetDeviceType() != "SECOND HAND DEVICE")
        dataTable.Columns.Add("Company", typeof(string));
    dataTable.Columns.Add("Model", typeof(string));
    dataTable.Columns.Add("Price", typeof(int));

    foreach (Device device in devices)
        if (device.GetDeviceType() == ObjectHandler.GetDeviceType())
        {
            if (ObjectHandler.GetDeviceType() != "SECOND HAND DEVICE")
```

```
            dataTable.Rows.Add(device.GetCompany(), device.GetModel(),
device.GetModelPrice());
          else
            dataTable.Rows.Add(device.GetModel(), device.GetModelPrice());
      }
   dataGridView1.DataSource = dataTable;
}
```

Add:

```
 public Add()
 {
    InitializeComponent();
    ShowDevice();
 }

 private void button8_Click(object sender, EventArgs e)
 {
    if (ObjectHandler.GetDeviceType() != "SECOND HAND DEVICE")
       ObjectHandler.GetDeviceDL().AddDevice(new
Device(ObjectHandler.GetDeviceType(), textBox1.Text.ToUpper(),
richTextBox1.Text.ToUpper(), double.Parse(textBox2.Text)));
    else
       ObjectHandler.GetDeviceDL().AddDevice(new
Device(ObjectHandler.GetDeviceType(), null, richTextBox1.Text.ToUpper(),
double.Parse(textBox2.Text)));
    MessageBox.Show("Device Model Added Successfully", "Device Crud",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    ShowDevice();
 }
 private void ShowDevice()
 {
    List<Device> devices = ObjectHandler.GetDeviceDL().GetDevices();
    DataTable dataTable = new DataTable();
    dataTable.Columns.Add("Type", typeof(string));
    if (ObjectHandler.GetDeviceType() != "SECOND HAND DEVICE")
       dataTable.Columns.Add("Company", typeof(string));
    dataTable.Columns.Add("Model", typeof(string));
    dataTable.Columns.Add("Price", typeof(int));

    foreach (Device device in devices)
       if (device.GetDeviceType() == ObjectHandler.GetDeviceType())
```

```
        {
            if (ObjectHandler.GetDeviceType() != "SECOND HAND DEVICE")
                dataTable.Rows.Add(device.GetDeviceType(), device.GetCompany(),
device.GetModel(), device.GetModelPrice());
            else
                dataTable.Rows.Add(device.GetDeviceType(), device.GetModel(),
device.GetModelPrice());
        }
    dataGridView1.DataSource = dataTable;
 }
 private void button1_Click(object sender, EventArgs e)
 {
    Hide();
    Form adminUI = new CRUDMenu();
    adminUI.Show();
 }
```

Update:

```
public Edit()
{
   InitializeComponent();
   ShowDevice();
}
private void ShowDevice()
{
   List<Device> devices = ObjectHandler.GetDeviceDL().GetDevices();
   DataTable dataTable = new DataTable();
   dataTable.Columns.Add("Type", typeof(string));
   if (ObjectHandler.GetDeviceType() != "SECOND HAND DEVICE")
      dataTable.Columns.Add("Company", typeof(string));
   dataTable.Columns.Add("Model", typeof(string));
   dataTable.Columns.Add("Price", typeof(int));

   foreach (Device device in devices)
      if (device.GetDeviceType() == ObjectHandler.GetDeviceType())
      {
         if (ObjectHandler.GetDeviceType() != "SECOND HAND DEVICE")
            dataTable.Rows.Add(device.GetDeviceType(), device.GetCompany(),
device.GetModel(), device.GetModelPrice());
         else
```

```
            dataTable.Rows.Add(device.GetDeviceType(), device.GetModel(),
device.GetModelPrice());
        }
    dataGridView1.DataSource = dataTable;
}
private void button2_Click(object sender, EventArgs e)
{
    if (ObjectHandler.GetDeviceType() != "SECOND HAND DEVICE")
        ObjectHandler.GetDeviceDL().EditPrice(ObjectHandler.GetDeviceType(),
textBox1.Text, richTextBox1.Text, double.Parse(textBox2.Text));
    else
        ObjectHandler.GetDeviceDL().EditPrice(ObjectHandler.GetDeviceType(), null,
richTextBox1.Text, double.Parse(textBox2.Text));

    MessageBox.Show("Device Price Updated Successfully", "Device Crud",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    ShowDevice();
}

private void dataGridView1_CellContentClick(object sender,
DataGridViewCellEventArgs e)
{
    if (e.RowIndex >= 0)
    {
        DataGridViewRow row = dataGridView1.Rows[e.RowIndex];
        if (ObjectHandler.GetDeviceType() != "SECOND HAND DEVICE")
            textBox1.Text = row.Cells["Company"].Value.ToString();
        richTextBox1.Text = row.Cells["Model"].Value.ToString();
        textBox2.Text = row.Cells["Price"].Value.ToString();
    }
}

private void button1_Click(object sender, EventArgs e)
{
    Hide();
    CRUDMenu cRUD = new CRUDMenu();
    cRUD.ShowDialog();
}
```

Sales:

```
public Sales()
```

```
{
   InitializeComponent();
   List<string> soldDevices = ObjectHandler.GetDeviceDL().GetSoldDevices();
   ShowDevices(soldDevices);
   sale.Text = "Your Total Sales: $" + ObjectHandler.GetUserDL().GetSales();
   dataGridView1.Columns["Username"].Width = 130;
   dataGridView1.Columns["Model"].Width = 230;
   dataGridView1.Columns["ModelPrice"].Width = 120;
}
private void ShowDevices(List<string> soldDevices)
{
   DataTable dataTable = new DataTable();
   dataTable.Columns.Add("UserName", typeof(string));
   dataTable.Columns.Add("Model", typeof(string));
   dataTable.Columns.Add("ModelPrice", typeof(double));

   foreach (string record in soldDevices)
   {
      string[] data = record.Split(',');
      dataTable.Rows.Add(data[0], data[1], data[2]);
   }
   dataGridView1.DataSource = dataTable;
}
```

AccountForm:

```
public AddMoney()
{
   InitializeComponent();
}

private void button1_Click(object sender, EventArgs e) // button_1 = Back Button
{
   Hide();
   Form form = new CustomerUI();
   form.ShowDialog();
}

private void button2_Click(object sender, EventArgs e) // button_2 = Add Money
Button
{
   IUserDL Iuser = ObjectHandler.GetUserDL();
```

```
   string accountNo = textBox1.Text;
   Customer user = SignIn.GetCurrentUser();
   if (Iuser.CheckAccount(accountNo, user.GetAccountNo()) && accountNo.Length ==
13)
   {
      string accountMoney = textBox2.Text;
      double money;
      if (CheckValidation(accountMoney))
      {
         money = double.Parse(accountMoney);
      }
      else
      {
         MessageBox.Show("Wrong Input", "Account Money", MessageBoxButtons.OK,
MessageBoxIcon.Error);
         return;
      }
      if (Iuser.UpdateMoney(user, user.GetAccountMoney() + money))
      {
         MessageBox.Show($"Your amount of {money} has been updated", "Account
Money", MessageBoxButtons.OK, MessageBoxIcon.Information);
         money = user.GetAccountMoney();
      }
      else
         MessageBox.Show("Amount cannot be upgraded to greater than 400k",
"Account Money", MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
   else
   MessageBox.Show("Wrong Account No", "Account No", MessageBoxButtons.OK,
MessageBoxIcon.Error);
}
```

BuyMobile:

```
 public static double PaidAmount=0;
 public BuyMobile()
 {
    InitializeComponent();
    List<Device> devices = ObjectHandler.GetDeviceDL().GetDevices();
    ShowMobiles(devices);
    dataGridView1.Columns["Company"].Width = 105;
```

```
        dataGridView1.Columns["Model"].Width = 180;
        dataGridView1.Columns["Price"].Width = 90;
        var uniqueCompanies = devices
            .Where(d => d.GetDeviceType().Equals("MOBILE",
StringComparison.OrdinalIgnoreCase))
            .Select(d => d.GetCompany().ToUpper())
            .Distinct();
        foreach (string companyName in uniqueCompanies)
            company.Items.Add(companyName);
    }
    private void ShowMobiles(List<Device> devices)
    {
        DataTable dataTable = new DataTable();
        dataTable.Columns.Add("Company", typeof(string));
        dataTable.Columns.Add("Model", typeof(string));
        dataTable.Columns.Add("Price", typeof(int));

        foreach (Device device in devices)
            if (device.GetDeviceType() == "MOBILE")
                dataTable.Rows.Add(device.GetCompany(), device.GetModel(),
device.GetModelPrice());
        dataGridView1.DataSource = dataTable;
    }
    private void button1_Click(object sender, EventArgs e)
    {
        string buyModel = model.Text.ToUpper();
        IDeviceDL Idevice = ObjectHandler.GetDeviceDL();
        IUserDL Iuser= ObjectHandler.GetUserDL();
        Customer user = SignIn.GetCurrentUser();

        if (Idevice.ModelExisted(buyModel))
        {
            double devicePrice = Idevice.GetDevicePrice(buyModel);

            if (devicePrice > 0 && devicePrice <= Iuser.AccountMoney(user))
            {
                Device device = Idevice.GetDeviceByName(buyModel);
                MessageBox.Show("The selected Mobile (" + buyModel + ") is within your
budget and is bought successfully.","Buying Mobile",MessageBoxButtons.OK,
MessageBoxIcon.Information);
                Console.WriteLine("Price: " + devicePrice);
```

```
            PaidAmount += devicePrice;
            user.SetAccountMoney(Iuser.AccountMoney(user) - devicePrice);
            Iuser.SaveUserDevice(user, device);
            ObjectHandler.GetDeviceDL().DeviceSold(user.GetName(),
device.GetModel(), device.GetModelPrice());
        }
        else if (devicePrice > Iuser.AccountMoney(user))
        {
            MessageBox.Show("The selected Mobile (" + buyModel + ")  is out of your
budget.", "Buying Mobile", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        }
    }
    else
    {
        MessageBox.Show("Wrong Model Name", "Buying Mobile",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
 }

 private void companym_SelectedIndexChanged(object sender, EventArgs e)
 {
    model.Items.Clear();
    string selectedItem = company.SelectedItem.ToString().ToUpper();
    List<Device> devices = ObjectHandler.GetDeviceDL().GetDevices();
    foreach(Device device in devices)
    {
        if(device.GetCompany() == selectedItem && device.GetDeviceType() ==
"MOBILE")
        {
            model.Items.Add(device.GetModel());
        }
    }
 }
```

GiveFeedback:

```
public GiveFeedback()
{
   InitializeComponent();
}
private void button1_Click(object sender, EventArgs e){
```

```
   ObjectHandler.GetUserDL().AddFeedback(SignIn.GetCurrentUser(),
richTextBox1.Text);
   MessageBox.Show("Feedback entered successfully", "Feedback",
MessageBoxButtons.OK, MessageBoxIcon.Information);
}
private void button3_Click(object sender, EventArgs e)
{
   Hide();
   Form form = new CustomerUI();
   form.ShowDialog();
}
```

Bill:

```
public Bill()
{
   InitializeComponent();
   UpdateAmountLabels(BuyMobile.PaidAmount + BuyLaptop.PaidAmount +
BuySW.PaidAmount +
BuyEarbuds.PaidAmount+Buy_Second_Hand_Device.PaidAmount,
ObjectHandler.GetUserDL().AccountMoney(SignIn.currentUser));
}
private void UpdateAmountLabels(double paid, double remaining)
{
   initialAmount.Text = "Your Initial Amount: $" + (paid+remaining).ToString();
   paidAmount.Text = "Your Paid Amount: $" + paid.ToString();
   remainingAmount.Text = "Your Remaining Amount: $" + remaining.ToString();
}
```

• CONCLUSION:

In conclusion I will summarize my project idea and its achievements, challenges faced during developments and the new things I learnt from this project.

• Challenges:

However, the project also encountered some challenges along the way. One significant challenge was communication between different classes of the system. Managing a large data of admin and customers along with their account no and bought devices. while maintaining

data maintainability and validation also posted its own set of challenges. Additionally, the most challenging moment for me is ORM during file-handling.

• Lesson Learned:

Throughout the project, several valuable lessons were learned. It became evident that effective planning and requirement gathering are crucial for the successful development of such a complex system. Modularity and separation of concerns played a vital role in achieving maintainability and flexibility. The implementation of design patterns, such as the BL, DL, and UI patterns, helped in achieving a structured and organized codebase. Furthermore, continuous testing and quality assurance were imperative to identify and resolve any issues early on.