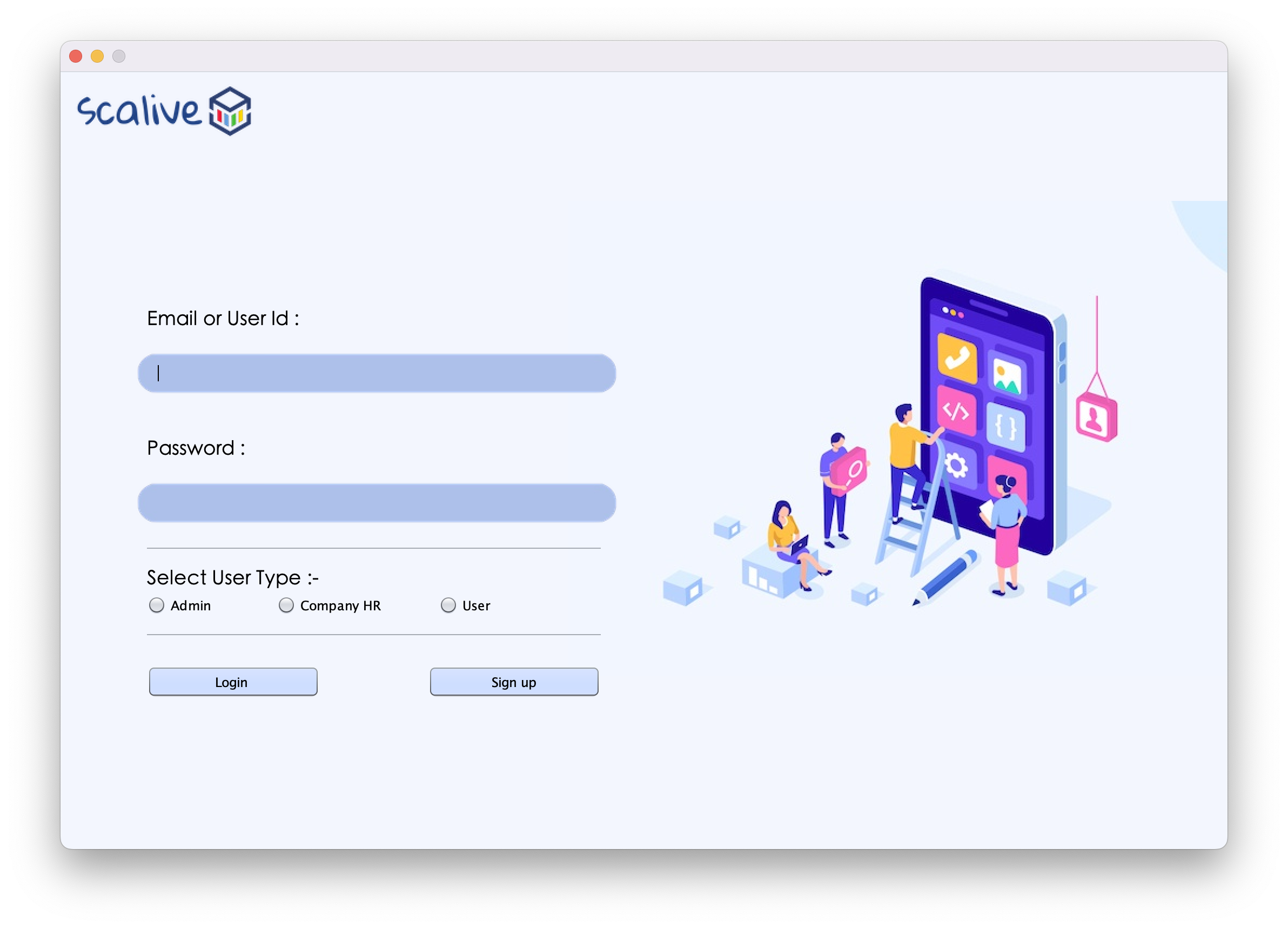
Java SE Project

Training And Placement Application

***Login Documentation***

1. **Login Frame -**

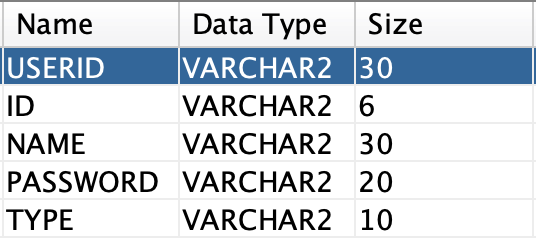
Here’s the login frame , first of all , create a login frame by name **“LoginFrame.java”**

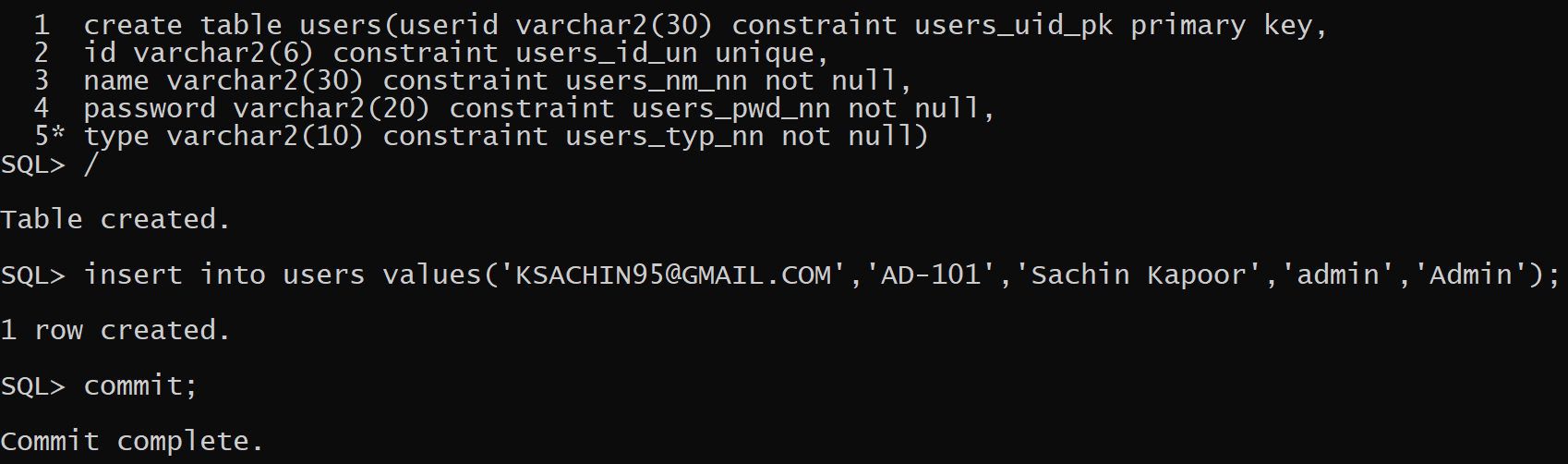
******

**2. Create table for login Credentials -**

For Login functionality we have to create a Table in database by name of “**USERS**”.  
Column name and data type are show in below snapshot :

And we have to put an **initial record** for admin login.







**3. Opening DataBase Connection -**

1. create a package inside at your project by the name of **“trandpl.dbutil”**   
   
 2. create a class inside package by the name of **“DBConnection.java”**

So the driver loading and opening of connection has to be only once so we should write this code inside the static block of **DBConnection** class.

There should be 2 methods in this class called **getConnection( )** and **closeConnection( )** which return the **Connection** for dataBase and close the **Connection** respectively.

package trandpl.dbutil;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DBConnection

{

private final static String DB\_URL="jdbc:oracle:thin:@//SachinKapoor:1521/orcl";

private final static String DB\_USER\_NAME="tnp";

private final static String DB\_PASS="project";

private static Connection conn=null;

static

{

try

{

Class.forName("oracle.jdbc.OracleDriver");

conn=DriverManager.getConnection(DB\_URL ,DB\_USER\_NAME,DB\_PASS);

}

catch(SQLException ex)

{

ex.printStackTrace();

System.out.println("Error in DBConnection :- ");

}

catch(ClassNotFoundException ex)

{

ex.printStackTrace();

System.out.println("Error in DBConnection :- ");

}

}

public static Connection getConnection()throws SQLException

{

return conn;

}

public static void closeConnection()

{

try

{

if(conn!=null)

conn.close();

}

catch(SQLException ex)

{

ex.printStackTrace();

System.out.println("Error while Closing Conn ");

}

}

}

**4. POJO ( Plain Old Java Object ) for login process -**

1. create a package inside at your project by the name of **“trandpl.pojo”**   
   
 2. create a class inside package by the name of **“UserPojo.java”**

POJO classes are those classes that are used to carry a user data from one package to another by creating their objects and for accessing the object data we can create methods that are called **setter** and **getter** for *“set a data in object”* and *“get or retrieve a data from the object”* respectively .

So the setter and getter are depends on the no of variables that we have created in the class.

In our scenario we have 5 variable by the name of :-

private String userId;

private String id;

private String name;

private String password;

private String type;

So we have to create 5 setters and 5 getters in our pojo class -

public String getUserId( )

public void setUserId( String userId )

public String getId( )

public void setId( String Id )

public String getName( )

public void setUserId( String name)

public String getPassword( )

public void setPassword( String Password )

public String getType( )

public void setType( String userId )

**5. DAO ( Data Access Object ) for login process -**

1. create a package inside at your project by the name of **“trandpl.dao”**   
   
 2. create a class inside package by the name of **“UserDao.java”**

So we have to create a method called **“validateUser( UserPojo user )”** *( And pass the pojo object as arguments we have created in step 4 )* in which we will check login credentials validation of the user and after that we will proceed to next frame according to the user type i.e. **“Admin”** , **“HR”** and **“Participant”**.

public class UseDAO {

public static boolean validateUser(UserPojo user)throws SQLException{

Connection conn=DBConnection.getConnection();

PreparedStatement ps=conn.prepareStatement("Select \* from users where userid=? and password=? and type=?");

ps.setString(1, user.getUserid());

ps.setString(2, user.getPassword());

ps.setString(3, user.getType());

ResultSet rs=ps.executeQuery();

if(rs.next()){

CurrentUser.setUserId(rs.getString(1));

CurrentUser.setId(rs.getString(2));

CurrentUser.setName(rs.getString(3));

CurrentUser.setType(rs.getString(5));

return true;

}

return false;

}

}

**6. Login Button Code & Functionality –**

1. When the user will click the **Login** button then it will verify the login details and accordingly open the next screen

2. Following are it's important points:

**a. It will first validate whether all the data has been properly filled all or not.**

**b. If not , then it will generate an error message and return.**

**c. Then it will verify whether "ADMIN" or "HR" or "User" option has been selected or not.**

**d. If not , then it will generate an error message and return.**

**e. Otherwise , it will create an UserPojo object , fill all the values in it and pass it to the method validateUser( ) of the UserDAO.**

**f. If the method validateUser( ) returned false then it will display an error message.**

**g. Otherwise , it will also store username and usertype as static fields in another class called UserProfile. This class will be used throughout the app to display the username on every frame.**

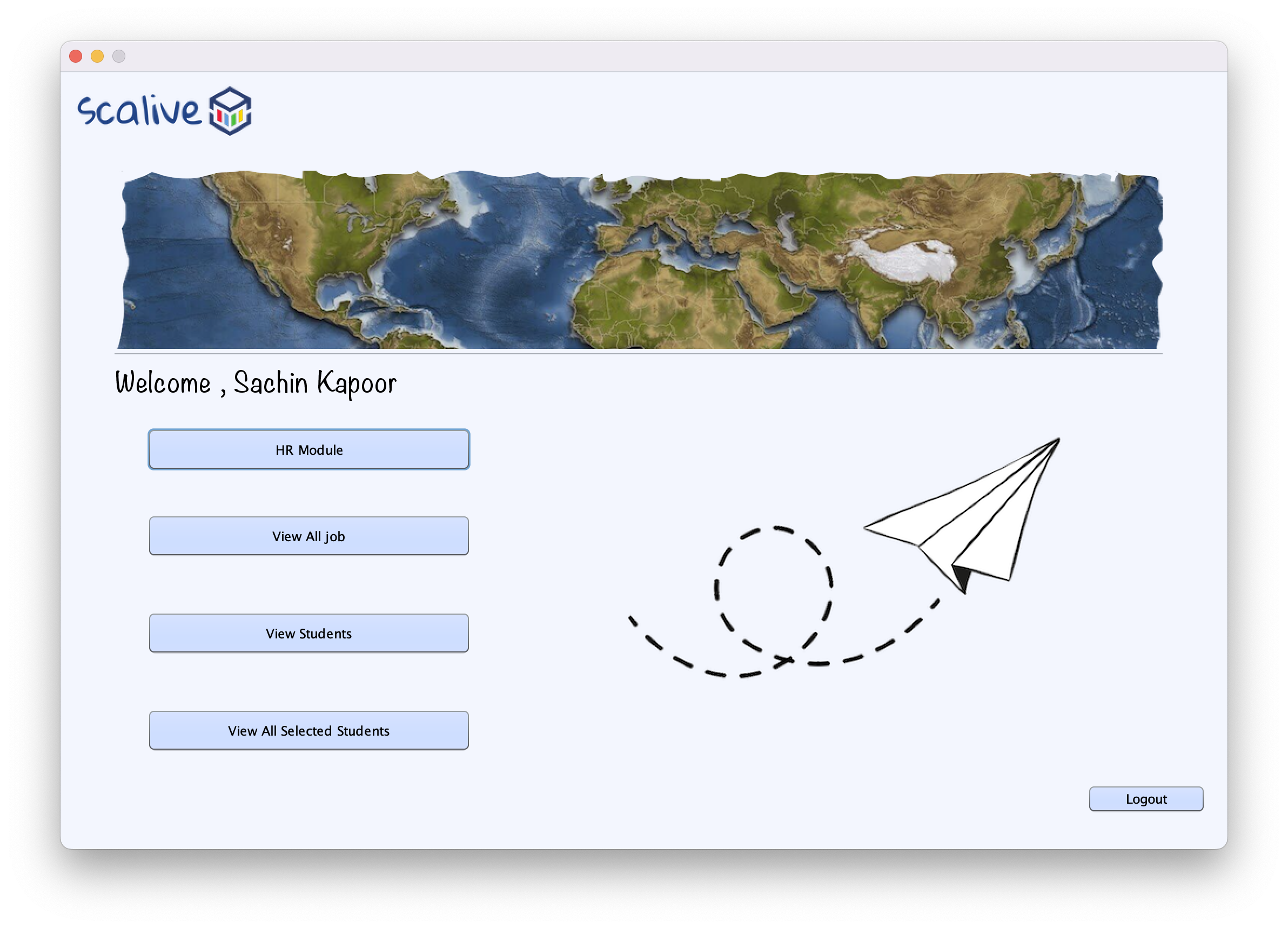
**h. Then if the user is Admin it will open the AdminOptionsFrame , if the user is HR it will open the HROptionsFrame and if the user is Student it will open the ParticipantOptionsFrame**

**i. It will also handle any SQLException that will be thrown by the method validateUser( )**

//Try writing code for LoginFrame

**7. Next step -**

After Admin validation we have to shift on next frame i.e. **“AdminOptionFrame.java”**

****

****