

LAPORAN PRAKTIKUM PBO

PEKAN KE – 2

INTEGRASI DATABASE



Oleh:

Daffa Aira Adrin

NIM 2411531006

MATA KULIAH PRAKTIKUM PEMROGRAMAN BERORIENTASI

OBJEK

DOSEN PENGAMPU: NURFIAH, S.ST, M.KOM.

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

PADANG

A. Pendahuluan

a) Latar Belakang

Dalam pengembangan aplikasi berbasis Java, pengelolaan data yang tersimpan di dalam database merupakan aspek penting yang harus diperhatikan. Salah satu pendekatan yang umum digunakan adalah dengan memisahkan logika akses data ke dalam lapisan tersendiri, yaitu **Data Access Object (DAO)**. DAO berfungsi sebagai jembatan antara aplikasi dan database, sehingga proses penyimpanan, pengambilan, penghapusan, dan pembaruan data dapat dilakukan secara terstruktur dan terpisah dari logika bisnis.

b) Tujuan

Tujuan dari pelaksanaan praktikum ini sebagai berikut:

- 1) Memahami konsep dasar **Data Access Object (DAO)** dalam pengembangan aplikasi Java berbasis database.
- 2) Mengimplementasikan koneksi database menggunakan **JDBC** untuk menghubungkan aplikasi dengan database MySQL.
- 3) Menerapkan prinsip **Object-Oriented Programming (OOP)** melalui pemisahan logika data dan tampilan menggunakan class User, interface UserDao, dan class UserRepo

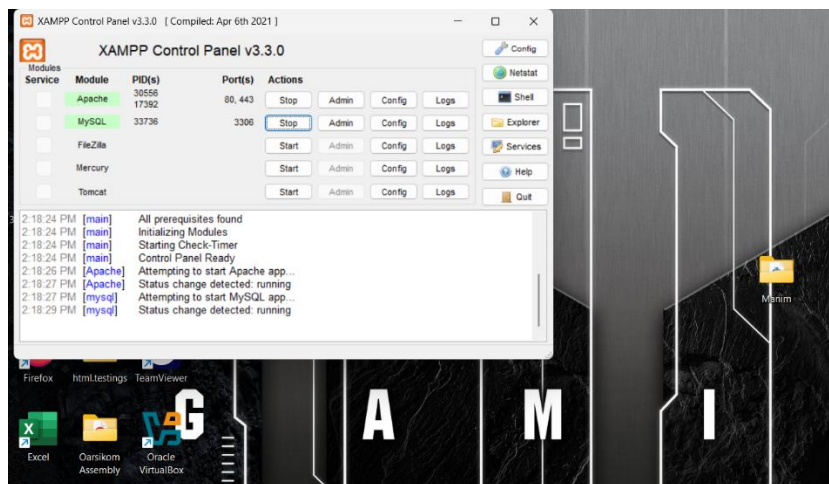
c) Landasan Teori

- 1) a Object-Oriented Programming (OOP) adalah paradigma pemrograman yang berfokus pada objek sebagai representasi dari entitas dunia nyata. Dalam OOP, objek memiliki atribut (data) dan method (fungsi) yang saling terkait.
- 2) Data Access Object (DAO) adalah pola desain yang digunakan untuk memisahkan logika akses data dari logika bisnis aplikasi. DAO menyediakan antarmuka untuk melakukan operasi terhadap database, seperti menyimpan, mengambil, memperbarui, dan menghapus data.
- 3) JDBC (Java Database Connectivity) adalah API dalam bahasa pemrograman Java yang digunakan untuk menghubungkan aplikasi dengan database. JDBC menyediakan berbagai class dan interface untuk melakukan koneksi, eksekusi query, dan pengelolaan hasil query dari database relasional seperti MySQL.
- 4) AbstractTableModel adalah class abstrak dalam pustaka javax.swing.table yang digunakan untuk membuat model data khusus untuk komponen JTable. Dengan meng-extend AbstractTableModel,

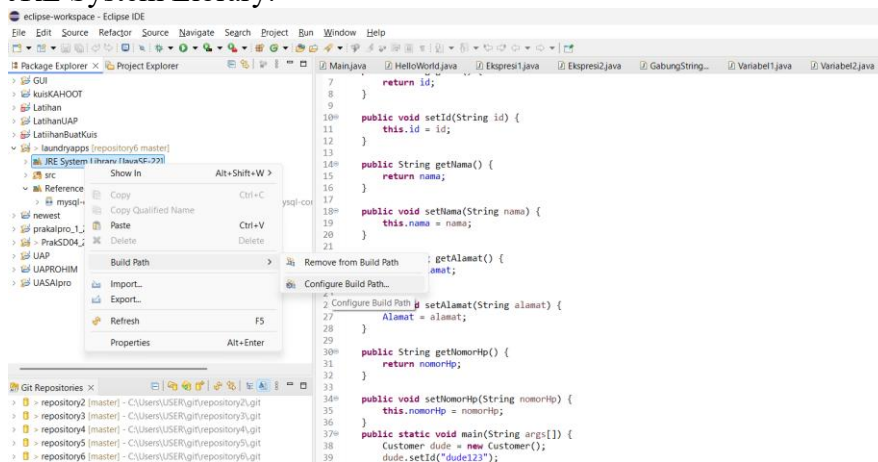
B. Langkah-Langkah Pengerjaan

a. Membuat Database

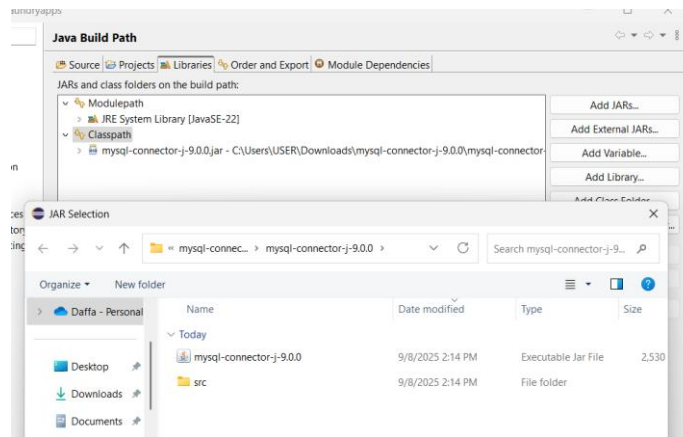
1. Buka aplikasi XAMPP, dan aktifkan APACHE, dan MySQL



2. Unduh mysql-connector, dan atur pada eclipse IDE dengan *build path* pada JRE System Library.



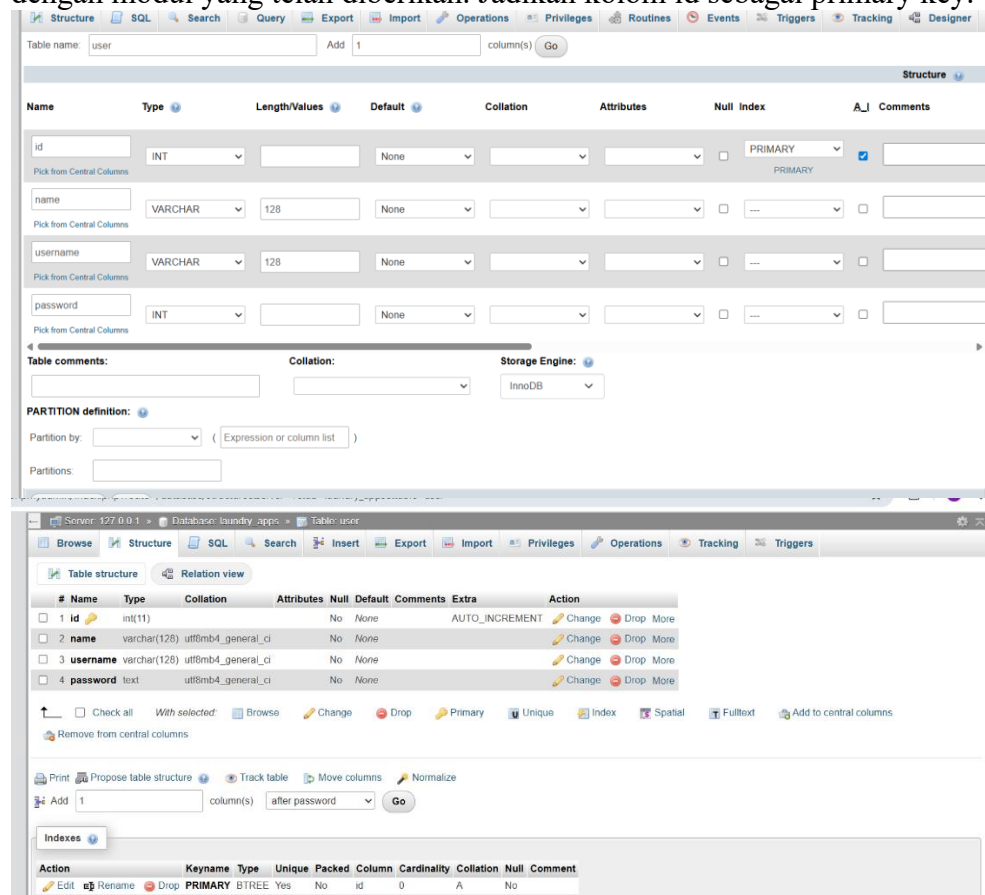
3. Kemudian, pada window Java Build Path, klik Add External JARs, dan pilih mysql-connector yang telah diunduh.



4. Masuk ke database dengan mengetik <http://127.0.0.1/phpmyadmin> pada web browser.



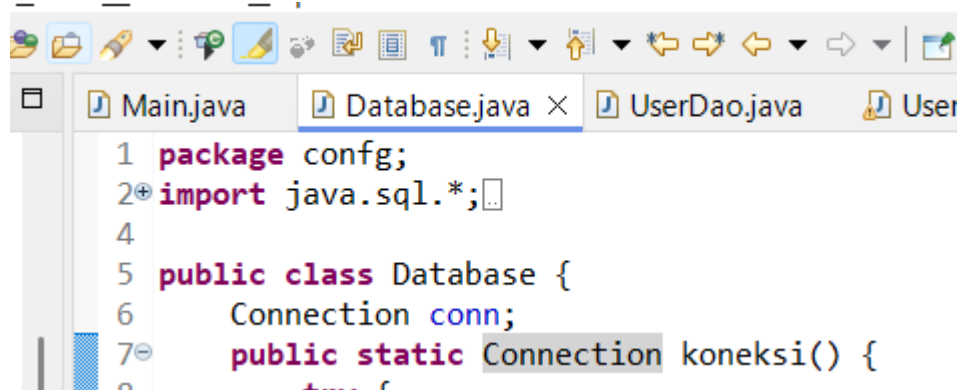
5. Buat database baru dengan nama laundryapps, dan buat tabel baru dengan nama user. Isikan nama masing-masing kolom, dan input yang lainnya sesuai dengan modul yang telah diberikan. Jadikan kolom id sebagai primary key.



- b. Mengakses Database menggunakan Java

1. Buat sebuah **package** baru dengan nama config pada project laundryapps.

2. Di dalam package `cfg`, buat sebuah **class** baru dengan nama `Database`.
3. Tambahkan **import library** `java.sql.*` untuk kebutuhan koneksi ke database dan `javax.swing.JOptionPane` untuk menampilkan pesan dialog jika terjadi error.
4. Deklarasikan atribut `conn` dengan tipe data `Connection` di dalam class.

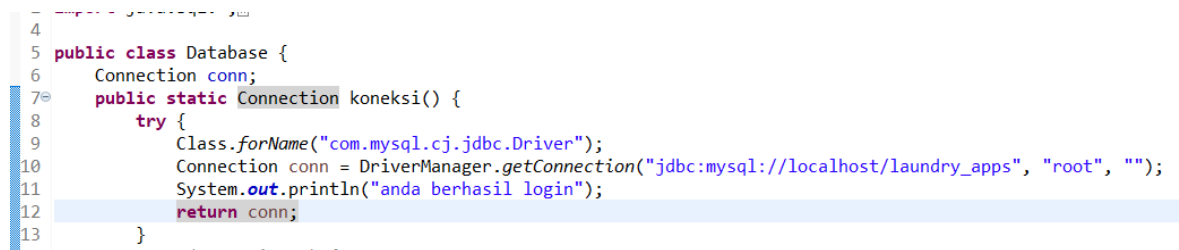


```

1 package cfg;
2 import java.sql.*;
3
4
5 public class Database {
6     Connection conn;
7     public static Connection koneksi() {

```

5. Buat sebuah method static dengan nama `koneksi()` yang bertipe `Connection`.
6. Di dalam method `koneksi()`, tuliskan blok try-catch untuk menangani proses koneksi ke database.
7. Di dalam blok try, panggil `Class.forName("com.mysql.cj.jdbc.Driver")` untuk memuat driver JDBC MySQL.
8. Buat koneksi ke database dengan menggunakan `DriverManager.getConnection("jdbc:mysql://localhost/laundry_apps", "root", "")`. Jika koneksi berhasil, tampilkan pesan "anda berhasil login" ke console menggunakan `System.out.println()`.
9. Kembalikan objek `Connection` yang telah berhasil dibuat.



```

4
5 public class Database {
6     Connection conn;
7     public static Connection koneksi() {
8         try {
9             Class.forName("com.mysql.cj.jdbc.Driver");
10            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/laundry_apps", "root", "");
11            System.out.println("anda berhasil login");
12            return conn;
13        }

```

10. Di dalam blok catch, tangkap exception dan tampilkan pesan error menggunakan `JOptionPane.showMessageDialog(null,e)`.
Kembalikan nilai `null` jika koneksi gagal.
11. Buat method `main()` untuk menguji koneksi ke database.
12. Di dalam method `main()`, panggil method `Database.koneksi()`.

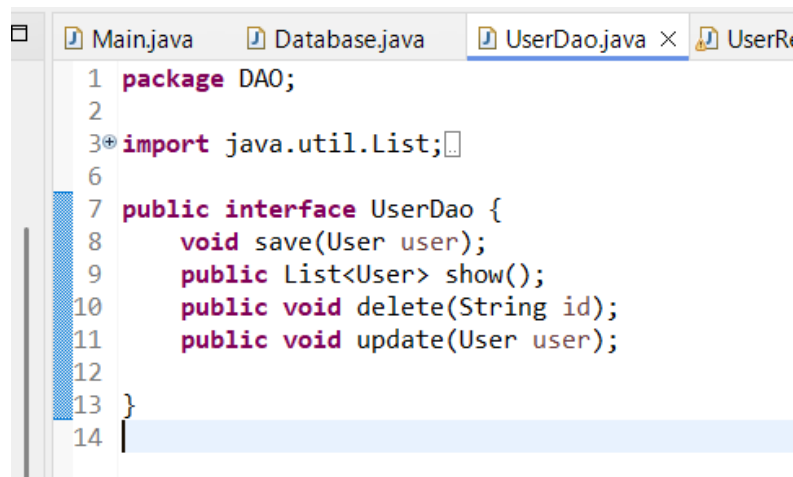
```

4
5 public class Database {
6     Connection conn;
7     public static Connection koneksi() {
8         try {
9             Class.forName("com.mysql.cj.jdbc.Driver");
10            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/laundry_apps", "root", "");
11            System.out.println("anda berhasil login");
12            return conn;
13        }
14        catch (Exception e) {
15            JOptionPane.showMessageDialog(null, e);
16            return null;
17        }
18    }
19    public static void main(String args[]) {
20        Database.koneksi();
21    }
22 }
23 }

```

c. DAO

1. Buat sebuah package baru dengan nama DAO.
2. Di dalam package DAO, buat sebuah interface baru dengan nama UserDao.
3. Tambahkan import untuk java.util.List karena interface ini akan menggunakan tipe data List untuk menampung kumpulan objek User.
4. Tambahkan juga import untuk class User dari package model.
5. Di dalam interface UserDao, deklarasikan method-method berikut:
6. Method save(User user) untuk menyimpan data user ke dalam database.
7. Method show() yang mengembalikan List<User> untuk menampilkan seluruh data user.
8. Method delete(String id) untuk menghapus data user berdasarkan ID.
9. Method update(User user) untuk memperbarui data user.



```

1 package DAO;
2
3 import java.util.List;
4
5
6
7 public interface UserDao {
8     void save(User user);
9     public List<User> show();
10    public void delete(String id);
11    public void update(User user);
12
13 }
14

```

d. Query

1. Buat sebuah class baru dengan nama UserRepo di dalam package DAO.
2. Pastikan class UserRepo mengimplementasikan interface UserDao.
3. Tambahkan import untuk library berikut:
 - java.sql.* untuk koneksi dan manipulasi database.
 - java.util.* untuk penggunaan List dan ArrayList.

- `java.util.logging.Logger` untuk logging error.
- `config.Database` untuk memanggil method koneksi database.
- `model.User` untuk menggunakan class `User`.
- Deklarasikan atribut `connection` bertipe `Connection`.

4. Buat beberapa query SQL sebagai final String:

- insert untuk menyimpan data user.
- select untuk menampilkan semua data user.
- delete untuk menghapus data user berdasarkan ID.
- update untuk memperbarui data user berdasarkan ID.

```

Main.java × Database.java UserDao.java UserRepo.java × LoginFrame.java MainFrame.java UserFrame
1 package DAO;
2
3
4
5* import java.lang.System.Logger.Level;
18
19 public class UserRepo implements UserDao {
20     private Connection connection;
21     final String insert = "INSERT INTO user (name, username, password) VALUES (?, ?, ?);";
22     final String select = "SELECT * FROM user;";
23     final String delete = "DELETE FROM user WHERE id=?;";
24     final String update = "UPDATE user SET name=?, username=?, password=? WHERE id=?;";
25

```

5. Buat constructor `UserRepo()` yang akan menginisialisasi koneksi database dengan memanggil `Database.koneksi()`.

```

<
3
4
5* import java.lang.System.Logger.Level;
18
19 public class UserRepo implements UserDao {
20     private Connection connection;
21     final String insert = "INSERT INTO user (name, username, password) VALUES (?, ?, ?);";
22     final String select = "SELECT * FROM user;";
23     final String delete = "DELETE FROM user WHERE id=?;";
24     final String update = "UPDATE user SET name=?, username=?, password=? WHERE id=?;";
25
26 public UserRepo() {
27     connection = Database.koneksi();
28 }

```

6. Implementasikan method `save(User user)`:

- Buat objek `PreparedStatement` untuk query insert.
- Set nilai parameter berdasarkan atribut `User`.
- Jalankan query dengan `executeUpdate()`.

```

    }
    public void save(User user) {
        PreparedStatement st = null;
        try {
            st = connection.prepareStatement(insert);
            st.setString(1, user.getNama());
            st.setString(2, user.getUsername());
            st.setString(3, user.getPassword());
            st.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                st.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}

```

7. Implementasikan method show():

- Buat objek Statement dan jalankan query select.
- Iterasi hasil ResultSet dan buat objek User untuk setiap baris data.
- Tambahkan objek User ke dalam List<User>.

```

    }
    @Override
    public List<User> show() {
        List<User> ls = null;
        try {
            ls = new ArrayList<User>();
            Statement st = connection.createStatement();
            ResultSet rs = st.executeQuery(select);
            while(rs.next()) {
                User user = new User();
                user.setId(rs.getString("id"));
                user.setNama(rs.getString("name"));
                user.setUsername(rs.getString("username"));
                user.setPassword(rs.getString("password"));
                ls.add(user);
            }
        } catch (SQLException e) {
            Logger.getLogger(UserDao.class.getName()).log(java.util.logging.Level.SEVERE, null, e);
        }
        return ls;
    }
}

```

8. Implementasikan method delete(String id):

- Buat objek PreparedStatement untuk query delete.
- Set parameter ID dan jalankan query.

```

        return ls;
    }
    @Override
    public void delete(String id) {
        PreparedStatement st = null;
        try {
            st = connection.prepareStatement(delete);
            st.setString(1, id);
            st.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```


9. Implementasikan method `update(User user)`:
 - Buat objek `PreparedStatement` untuk query update.
 - Set semua parameter berdasarkan atribut `User`.
 - Jalankan query dengan `executeUpdate()`.
10. Tambahkan blok `try-catch` di setiap method untuk menangani `SQLException`.
11. Tambahkan blok `finally` untuk menutup `PreparedStatement` setelah digunakan.

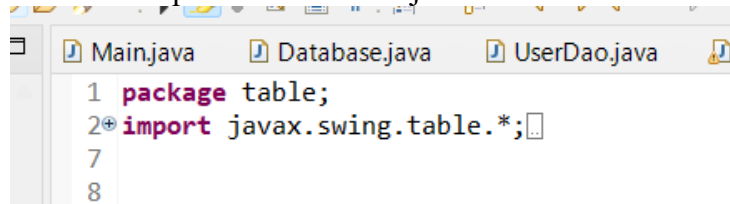
```

    }
    @Override
    public void update(User user) {
        PreparedStatement st = null;
        try {
            st = connection.prepareStatement(update);
            st.setString(1, user.getName());
            st.setString(2, user.getUsername());
            st.setString(3, user.getPassword());
            st.setString(4, user.getId());
            st.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                st.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}

```

e. Akses untuk table

1. Buat sebuah package baru dengan nama table.
2. Buat sebuah class baru dengan nama `TableUser` di dalam package tersebut.
3. Tambahkan import library `javax.swing.table.AbstractTableModel` untuk membuat model tabel yang dapat digunakan pada komponen `JTable`.
4. Tambahkan juga import untuk `model.User` dan `java.util.List` karena class ini akan menampilkan data dari objek `User` dalam bentuk tabel.

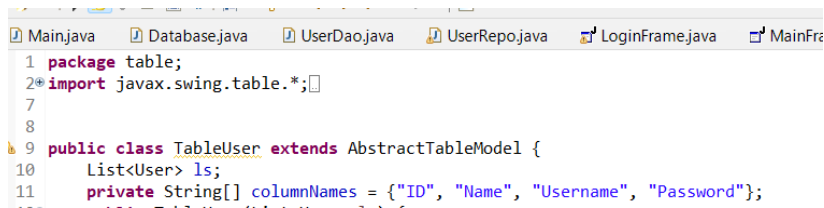


```

1 package table;
2 import javax.swing.table.*;
7
8

```

5. Jadikan class `TableUser` sebagai turunan dari `AbstractTableModel`.
6. Deklarasikan atribut `ls` bertipe `List<User>` untuk menampung data user yang akan ditampilkan.
7. Deklarasikan array `columnNames` bertipe `String[]` yang berisi nama-nama kolom: "ID", "Name", "Username", dan "Password".



```

1 package table;
2 import javax.swing.table.*;
3
4
5
6
7
8
9 public class TableUser extends AbstractTableModel {
10     List<User> ls;
11     private String[] columnNames = {"ID", "Name", "Username", "Password"};

```

8. Buat constructor TableUser(List<User> ls) untuk menginisialisasi data user yang akan ditampilkan di tabel.
9. Override method getRowCount() untuk mengembalikan jumlah baris berdasarkan ukuran list ls.
10. Override method getColumnCount() untuk mengembalikan jumlah kolom, yaitu 4.
11. Override method getColumnName(int column) untuk mengembalikan nama kolom berdasarkan indeks.

```

private String[] columnNames = {"ID", "Name", "Username", "Password"};
public TableUser(List<User> ls) {
    this.ls = ls;
}
@Override
public int getRowCount() {
    return ls.size();
}

@Override
public int getColumnCount() {
    return 4;
}

@Override
public String getColumnName(int column) {
    return columnNames[column];
}

```

12. Override method getValueAt(int rowIndex, int columnIndex) untuk mengembalikan nilai dari objek User berdasarkan baris dan kolom yang dipilih:
 - Kolom 0: ID
 - Kolom 1: Nama
 - Kolom 2: Username
 - Kolom 3: Password


```

Main.java × Database.java UserDao.java UserRepo.java LoginFrame.java
1 package ui;
2
3 import java.awt.EventQueue;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24 public class UserFrame extends JFrame {
25
26     private static final long serialVersionUID = 1L;
27     private JPanel contentPane;
28     private JTextField txtName;
29     private JTextField txtUsername;
30     private JTextField txtPassword;
31     private JTable tableUsers;
32     UserRepo usr = new UserRepo();
33     List<User> ls;
34     public String id;
35
36

```

3. Tambahkan actionlistener pada tombol save untuk melakukan operasi terhadap input yang dimasukkan oleh pengguna ketika menekan tombol save.

```

JButton btnSave = new JButton("Save");
btnSave.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        User user = new User();
        user.setNama(txtName.getText());
        user.setUsername(txtUsername.getText());
        user.setPassword(txtPassword.getText());
        usr.save(user);
        reset();
    }
});

```

C. KESIMPULAN

Pada praktikum ini, dilakukan implementasi koneksi database menggunakan **JDBC (Java Database Connectivity)** yang terintegrasi dengan MySQL. Koneksi ini diatur melalui class Database, yang bertanggung jawab untuk membuka akses ke database laundry_apps. Selanjutnya, dibuat interface UserDao yang mendefinisikan operasi dasar terhadap data pengguna, seperti save, show, delete, dan update.

Implementasi dari interface tersebut dilakukan dalam class UserRepo, yang berisi logika SQL untuk berinteraksi langsung dengan tabel user di database. Selain itu, untuk menampilkan data pengguna dalam bentuk tabel pada GUI, digunakan class TableUser yang merupakan turunan dari AbstractTableModel.

