Państwowa Uczelnia
im. Stefana Batorego

Bobrowniki, 28.05.2022 r.

Podstawy Baz Danych

Projekt

Prosta aplikacja bazodanowa

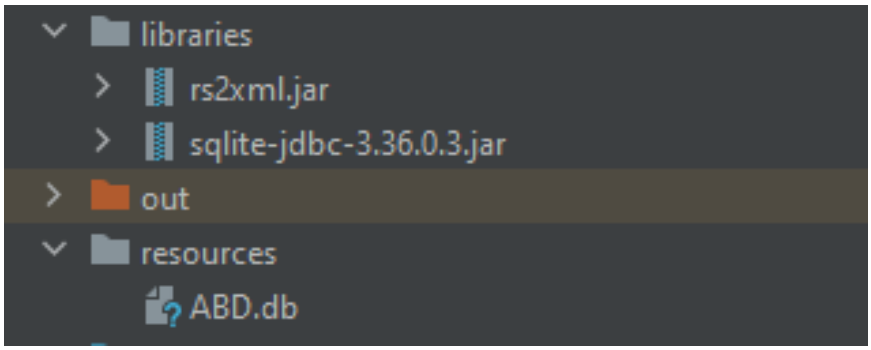
Wykonawca sprawozdania:
Jakub Zdanowski ST INF II

Prowadzący:
mgr inż. Łukasz Rybak

1. Cel ćwiczenia

Stworzenie prostej aplikacji bazodanowej z interfejsem graficznym w języku Java z wykorzystaniem gotowej bazy danych stworzonej podczas laboratoriów.

2. Lista zasobów zewnętrznych użytych w projekcie



Rysunek 1 - wykaz zasobów

Biblioteki:

- Sqlite-jdbc verson 3.36.0.3 – Biblioteka użyta do połączenia się z bazą danych sqlite
- Rs2xml – biblioteka użyta aby dopasować format ResultSet do modelu tabeli interfejsu graficznego

Baza danych:

- ABD.db – Baza danych stworzona podczas laboratoriów

3. Lista zaimportowanych pakietów

java.sql:

- java.sql.Connection
- java.sql.DriverManager
- java.sql.ResultSet
- java.sql.SQLException
- java.sql.Statement

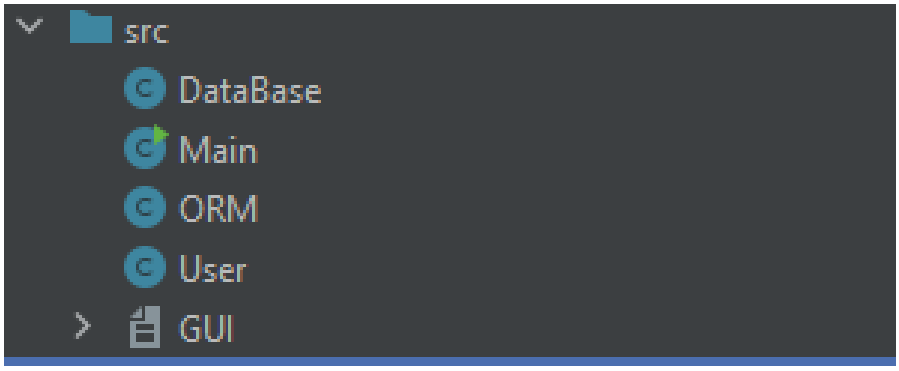
net.proteaint.sql:

- net.proteanit.sql.DbUtils

java.util:

- java.util.LinkedList

4. Lista klas



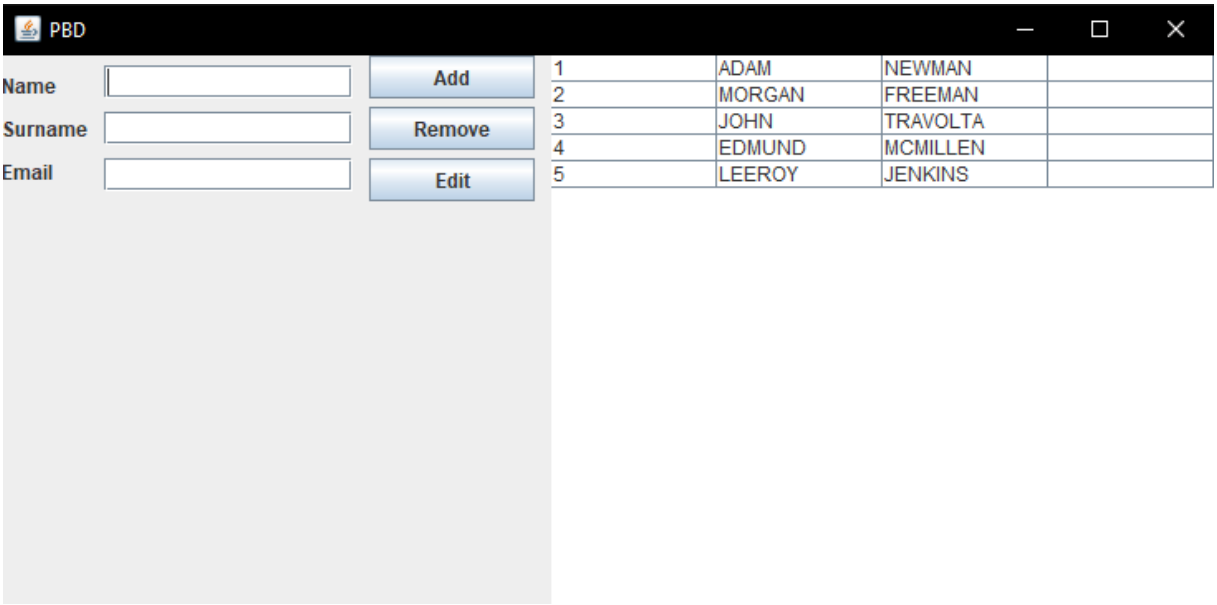
Rysunek 2 - wykaz klas

- DataBase.java – obsługa bazy danych
- GUI.java – renderowanie interfejsu graficznego, posiada funkcję main
- ORM.java – zamiana odpowiedzi bazy danych na obiekty klasy User
- User.java – stanowi szablon instancji obiektu, umożliwia ustawianie oraz pobieranie wartości pól obiektu

5. Interfejs graficzny

GUI zostało wykonane z wykorzystaniem narzędzia do tworzenia interfejsów graficznych w środowisku IntelliJ IDEA Community Edition 2022.1.

Kod tworzący okno jak i poszczególne jego pola został w całości wygenerowany przez wyżej wspomniane narzędzie. Aplikacja zawiera formularz służący do wprowadzania danych oraz tabele, w której wyświetlane są wyniki zapytań do bazy danych.



Rysunek 3 - Interfejs graficzny aplikacji

5.2 Pola formularza

Formularz składa się z pól tekstowych wraz z ich etykietami:

- NAME
- SURNAME
- EMAIL

Oraz przycisków:

- Add
- Remove
- Edit

6. Połączenie z bazą danych

```
1 usage   zdanowskijacob
public void connect(){
    try {
        c = DriverManager.getConnection(url);
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}
zdanowskijacob
public void disconnect(){
    try {
        c.close();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}
```

Rysunek 4- metody łączące i rozłączające z bazą danych

- C jest polem klasy DataBase typu Connection
- Url jest polem typu String przechowującym podaną jako argument konstruktora klasy DataBase ścieżke do bazy danych

7. Wyświetlanie bazy danych

Dane wyświetlono za pomocą funkcji showDataBase(), która ustawia model tabeli na zgodny z odpowiedzią bazy danych, która przekazywania jest jako return z funkcji selectAll()

```
4 usages   zdanowskijacob
public void showDataBase(DataBase db) { table.setModel(DbUtils.resultSetToTableModel(db.selectAll())); }
```

Rysunek 5- implementacja metody showDataBase()

```
2 usages   zdanowskijacob
public ResultSet selectAll(){
    Statement s;
    ResultSet res = null;
    try {
        s = c.createStatement();
        res = s.executeQuery( sql: "SELECT * FROM USERS");
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    return res;
}
```

Rysunek 6 - implementacja metody selectAll()

8. Obsługa przycisków

Przyciski obsługiwane się poprzez Action Listener na każdym z nich.

8.1 Przycisk Add

Przycisk Add tworzy nowy obiekt klasy User przyjmując wartości odpowiednich pól formularza dla odpowiadających im pól klasy z wyjątkiem pola ID, które generowane jest automatycznie za pomocą metody getIndex(), która sprawdza rozmiar listy i inkrementuje go o 1 co gwarantuje unikalne id dla nowego obiektu po czym wywołuje metode addToDb() , która dodaje do bazy danych rekord stworzony na podstawie już powstałego obiektu następnie wyświetle aktualną bazę danych i czyści formularz.

```
zdanowskijacob
addBtn.addActionListener(new ActionListener() {
    zdanowskijacob
    @Override
    public void actionPerformed(ActionEvent e) {
        User u = new User(orm.getNextIndex(), nameText.getText().toUpperCase(), surnameText.getText().toUpperCase(),
                emailText.getText().toUpperCase());
        if(u.getSurname().length() > 0 && u.getName().length() > 0)
        db.addToDb(u);
        else
            System.out.println("Podaj imie i nazwisko");
        showDataBase(db);
        clearForm();
    }
});
```

Rysunek 7- implementacja ActionListener przycisku Add

8.2 Przycisk Remove

Przycisk remove pobiera z JTable aktywny rząd oraz wartość z pierwszej kolumny tego rzędu. Po czym wywołuje metode usuwającą rekord o wskazanym id i wyświetla aktualną baze danych.

Rysunek 8- implementacja ActionListener przycisku remove

8.3 Przycisk Edit

Przycisk remove pobiera z JTable aktywny rząd oraz wartość z pierwszej kolumny tego rzędu oraz tworzy obiekt na podstawie wartości pól z formularza. Po czym wywołuje metode modyfikującą rekord o wskazanym id, wyświetla aktualną baze danych i czyści formularz.

Rysunek 9- implementacja ActionListener przycisku edit

8.4 Czyszczenie formularza

Metoda zeruje zawartość tekstową pól formularza za pomocą metody setText()

```
2 usages   zdanowskijacob
public void clearForm(){
    nameText.setText("");
    surnameText.setText("");
    emailText.setText("");
}
```

Rysunek 10 - implementacja metody clearForm()

9. Operacje na bazie danych

Program oferuje 3 podstawowe operacje na bazie danych dodaj, usuń, modyfikuj.

9.1 Funkcja dodaj

```
1 usage   zdanowskijacob
public boolean addToDb(User u) {
    Statement s;
    String query =  buildAddQuery(u);

    try {
        s = c.createStatement();
        s.executeUpdate(query);

        return true;
    } catch (SQLException e) {
        e.printStackTrace();
    }

        return false;
}
```

Rysunek 11-implementacja metody dodającej rekord do bazy danych

```
1 usage  zdanowskijacob
public String buildAddQuery(User u) {

    String query = "INSERT INTO USERS(NAME, SURNAME, EMAIL) VALUES ('" + u.getName() + "', '" + u.getSurname()
            + "', '" +
            u.getEmail() + "');";
    if (u.getName().length() > 0 && u.getSurname().length() > 0)
        return query;

    return null;
}
```

Rysunek 12- implementacja metody tworzącej zapytanie SQL dla funkcji addToDb()

9.2 Funkcja usuń

```
1 usage  zdanowskijacob
public boolean deleteFromDb(int index){
    Statement s;
    String query = "DELETE FROM USERS WHERE ID=" + index;

    try {
        s = c.createStatement();
        s.executeUpdate(query);

        return true;
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}
```

Rysunek 13- implementacja metody usuwającej rekord z bazy danych

9.3 Funkcja modyfikuj

```
1 usage   zdanowskijacob
public boolean updateDb(User u, int index){
    Statement s;
    String query = buildUpdateQuery(u, index);

    try {
        System.out.println(query);
        s = c.createStatement();
        s.executeUpdate(query);
        return true;
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }

}
```

Rysunek 14- implementacja metody modyfikującej rekord bazy danych

```
1 usage   zdanowskijacob
public String buildUpdateQuery(User u, int index){
    String query = "UPDATE USERS SET ";
    if (u.getName().length() > 0) {
        query += "NAME='" + u.getName() + "' ";
    }
    if (u.getName().length() > 0 && u.getSurname().length() > 0) {
        query += ", SURNAME='" + u.getSurname() + "' ";
    }else if(u.getSurname().length() > 0 ) {
        query += "SURNAME='" + u.getSurname() + "' ";
    }
    if ((u.getName().length() > 0 || u.getSurname().length() > 0) && u.getEmail().length() > 0) {
        query += ", EMAIL='" + u.getEmail() + "' ";
    } else if(u.getEmail().length() > 0) {
        query += "EMAIL='" + u.getEmail() + "' ";
    }

    query += "WHERE ID=" + index +";";

    return query;
}
```

Rysunek 15- implementacja metody generującej zapytanie SQL dla funkcji updateDb()

10. Pozostałe klasy

```java
public class ORM {
    6 usages
    private ResultSet res;
    6 usages
    private LinkedList<User> li;

    1 usage  zdanowskijacob
    public ORM(ResultSet res) { this.res = res; }

    1 usage  zdanowskijacob *
    public void create() {
        try {
            li = new LinkedList<User>();
            while (res.next()) {
                li.add(
                        new User(res.getInt( columnIndex: 1),
                                res.getString( columnIndex: 2),
                                res.getString( columnIndex: 3),
                                res.getString( columnIndex: 4)));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    zdanowskijacob *
    public void appendList(User u) {
        li.addLast(u); }

    zdanowskijacob *
    public LinkedList<User> getList() {
        return li; }

    1 usage  zdanowskijacob *
    public int getNextIndex() {
        if (li.size() == 0) {
            return 1;
        } else
            return li.size() + 1;
    }
}
```

Rysunek 16- implementacja klasy ORM

```java
public class User {
    2 usages
    private int id;
    2 usages
    private String name;
    2 usages
    private String surname;
    2 usages
    private String email;

    3 usages  zdanowskijacob *
    User(int id, String name, String surname, String email) {
        this.setId(id);
        this.setName(name);
        this.setSurname(surname);
        this.setEmail(email);
    }
    1 usage  zdanowskijacob
    public void setId(int id) { this.id = id; }
    1 usage  zdanowskijacob
    public void setName(String name) { this.name = name; }
    1 usage  zdanowskijacob
    public void setSurname(String surname) { this.surname = surname; }
    1 usage  zdanowskijacob
    public void setEmail(String email) { this.email = email; }
     zdanowskijacob
    public int getId() { return id; }
    7 usages  zdanowskijacob
    public String getName() { return name; }
    8 usages  zdanowskijacob
    public String getSurname() { return surname; }
    5 usages  zdanowskijacob
    public String getEmail() { return email; }


}
```

Rysunek 17- implementacja klasy User