Министерство образования Республики Беларусь
«Учреждения образования информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Объектно-ориентированное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

«Автоматизированная система «Аукцион продаж произведений искусства»»

Студент: гр.053505
Шебеко Ю.А.

Руководитель: ассистент кафедры
Информатики Летохо А. С.

Минск 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современные информационные системы ориентированы на пользователя, не обладающего высокой квалификацией в области вычислительной техники. Поэтому клиентские приложения информационной системы должны обладать простым, удобным, легко осваиваемым интерфейсом. Этого можно добиться путем сочетания объектно-ориентированных языков и СУБД.

Таким образом перед клиентом предстанет приложение легкое в использовании, а перед администратором возможность вести учет клиентов и услуг в больших объемах.

СУБД (система управления базами данных) представляет собой комплекс ПО, с помощью которого можно создавать базы данных и проводить над ними различные операции: обновлять, удалять, выбирать, редактировать.

Данный проект реализован на языке программирования Java в среде разработки IntelliJ IDEA.

Язык программирования Java предоставляет большое количество удобных инструментов, которые можно использовать в своей программе, для разработки удобного интерфейса пользователя, а также для взаимодействия с базами данных.

Для базы данных использовался СУБД PostgreSQL, а создание графического пользовательского интерфейса осуществлялось в среде разработки Scene Builder с использованием пакета JavaFX.

Целью данного курсового проекта является создание удобного и информативного приложения, которое позволит новому пользователю без труда выяснить наличие тех или иных произведений искусств, узнать их стоимость, количество в наличии и совершить покупку.

1. ПОСТАНОВКА ЗАДАЧИ

Согласно техническому заданию, программа должна содержать:

1. Графический интерфейс
2. Классы для создания сущностей
3. Класс для объединения сущностей
4. View (работы с формами)
5. Класс для объединения работы программы с базами данными

Пользователю предоставляется информация о произведениях искусства:

1. Название произведения искусства
2. Автор произведения искусства
3. Количество произведений искусства
4. Стоимость произведения искусства
5. Категорию к которой относится произведение искусства
6. Дата проведения аукциона

А также программа должна обеспечивать функцию поиска информации:

1. По названию произведения искусств
2. По автору произведения искусств
3. По стоимости произведения искусств
4. По категории произведения искусств

Администратору предоставляется возможность:

1. Добавление произведений искусств
2. Изменение информации о произведении искусств
3. Удаление произведений искусств
4. Добавление даты аукциона

2 АРХИТЕКТУРА ПРИЛОЖЕНИЯ

Для выполнения данной курсовой работы использовалась MVC архитектура (рисунок 2.1).

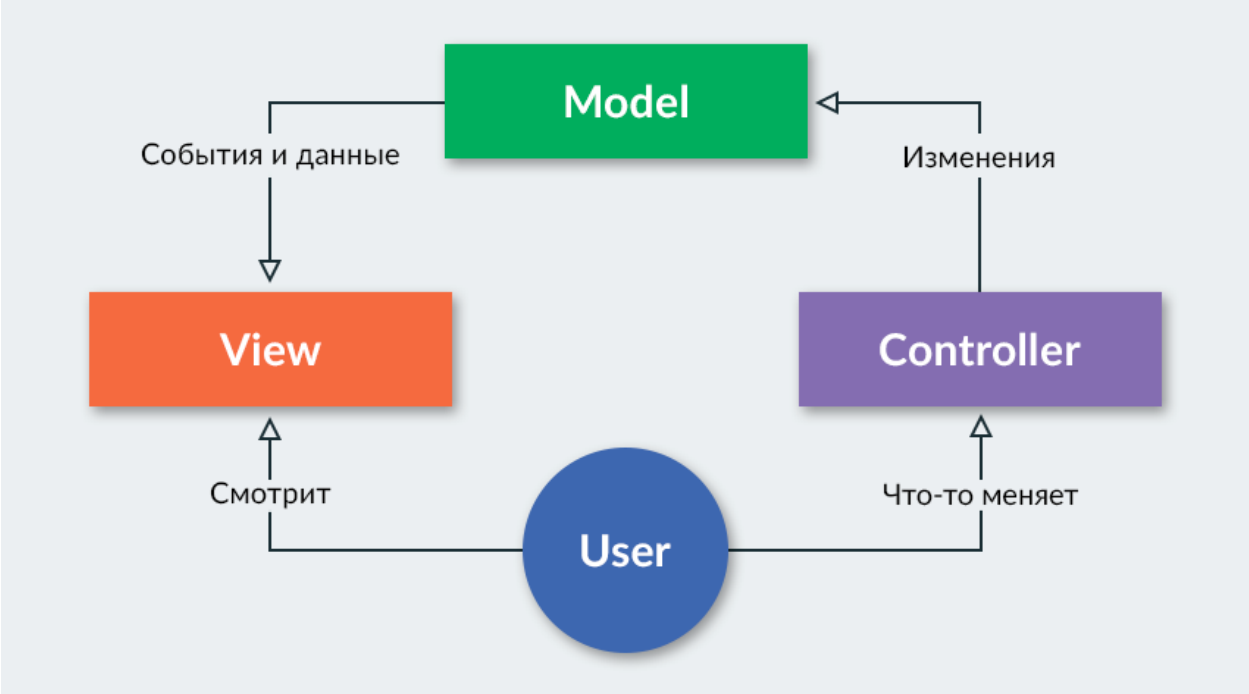


Рисунок 2.1 - Схема MVC

Основная идея MVC – разделить внешний вид приложения от логики. Т.к. мы говорим о веб-приложениях, то под внешним видом у нас понимается HTML-разметка документа и его CSS оформление. А под логикой мы понимаем различные скрипты, функции, классы и т.д.

Шаблон MVC – это аббревиатура, которая состоит из первых букв тех составляющих, которые в нее входят: Модель и Контроллер (скрипты и логика) и Вид (HTML и CSS (шаблонизатор)).

В логике не должно быть внешнего вида, а во внешнем виде не должно быть логики. Модель MVC – это всего лишь набор рекомендаций, которым нужно придерживаться при написании кода.

В модели MVC самым главным элементом, с которого все начинается и на котором все, как правило, заканчивается - это контроллер.

Задача контроллера - принять запрос пользователя и решить, что делать с этим запросом далее, если требуется, то перенаправить запрос в модель, который обработает информацию и возвращает ее контроллеру.

После того, как информация обработана, контроллер решает, что с ней делать дальше, если пользователю достаточно предоставить просто набор каких-то данных, не в виде html-странице, а например, в формате json, контроллер этот набор данных ему выдает.

Если необходимо сформировать html-страницу, контроллер передает эти данные в вид и внутри вида шаблонизатор формирует каркас страницы, выдает

ее назад контроллеру и контроллер уже выдает этот каркас пользователю в виде html-страницы.

Из 3 частей MVC модели, контроллер является обязательной частью. Остальные части являются опциональными. Если пользователю достаточно только отдать какой-то набор данных, то можно обойтись без вида. Если не нужно обрабатывать данные, то можно обойтись без модели.

2.1 Model (Модель)

Модель получает данные от контроллера, выполняет необходимые операции и передаёт их в вид.

При этом «*Модель*» не имеет никакой связи или информации о том, что происходит с данными, когда они передаются компонентам «*Вид*» или «*Контроллер*». Единственная задача «*Модели*» - обработка данных в постоянном хранилище, поиск и подготовка данных, передаваемых другим составляющим **MVC**.

К **model** относятся такие классы как:

1. Art
2. User
3. Package dataBase

2.2 View (Представление)

Представление получает данные от модели и выводит их для пользователя.

Представление также перехватывает действие пользователя, которое затем передается «Контроллеру». Характерным примером этого является кнопка, генерируемая «Представлением». Когда пользователь нажимает ее, запускается действие в «Контроллере».

К **view** относятся такие классы как:

1. Admin-page.fxml
2. Enter-view.fxml
3. Registration-view.fxml
4. User-page.fxml

2.3 Controller (Контроллер)

Контроллер обрабатывает действия пользователя, проверяет полученные данные и передаёт их модели.

К **controller** относятся такие классы как:

1. Controller

2. RegistrationController
3. AdminController

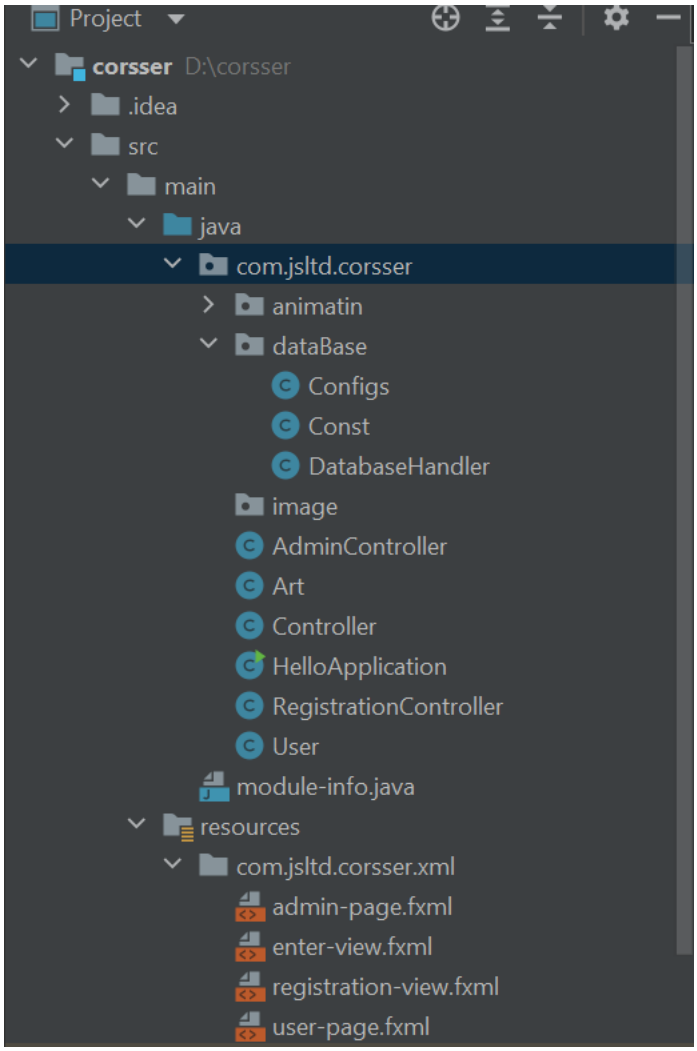


Рисунок 2.1. - Структура проекта

3 ЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

3.1 Отношение

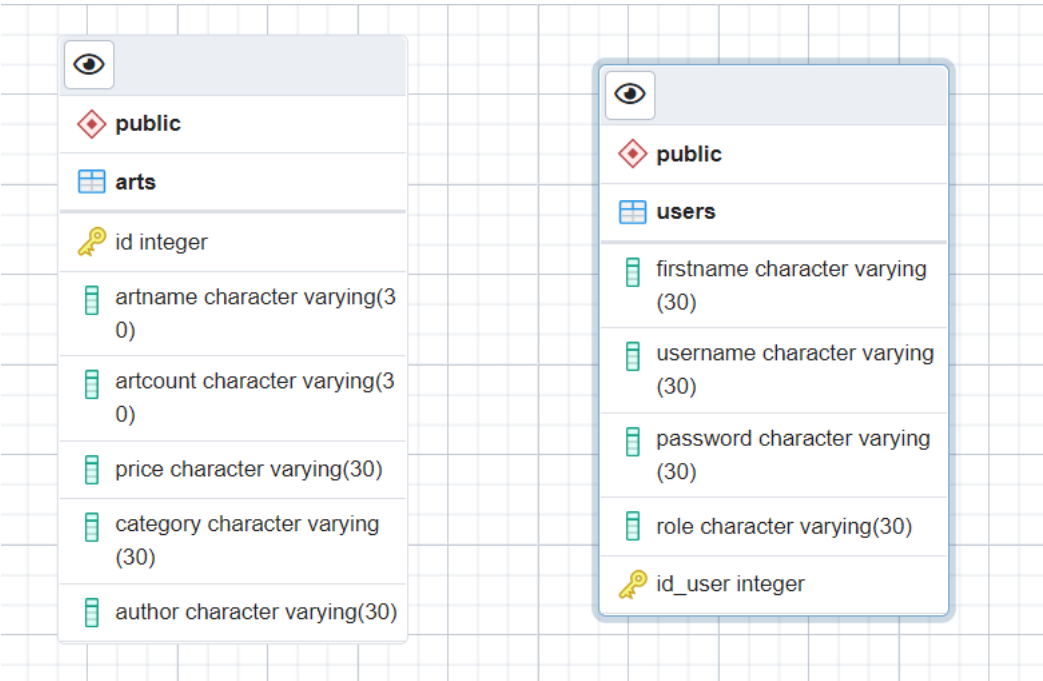


Рисунок 3.1 – Схема данных в PostgreSQL

3.2 База данных

База данных — это место для хранения данных (рисунок 3.2).

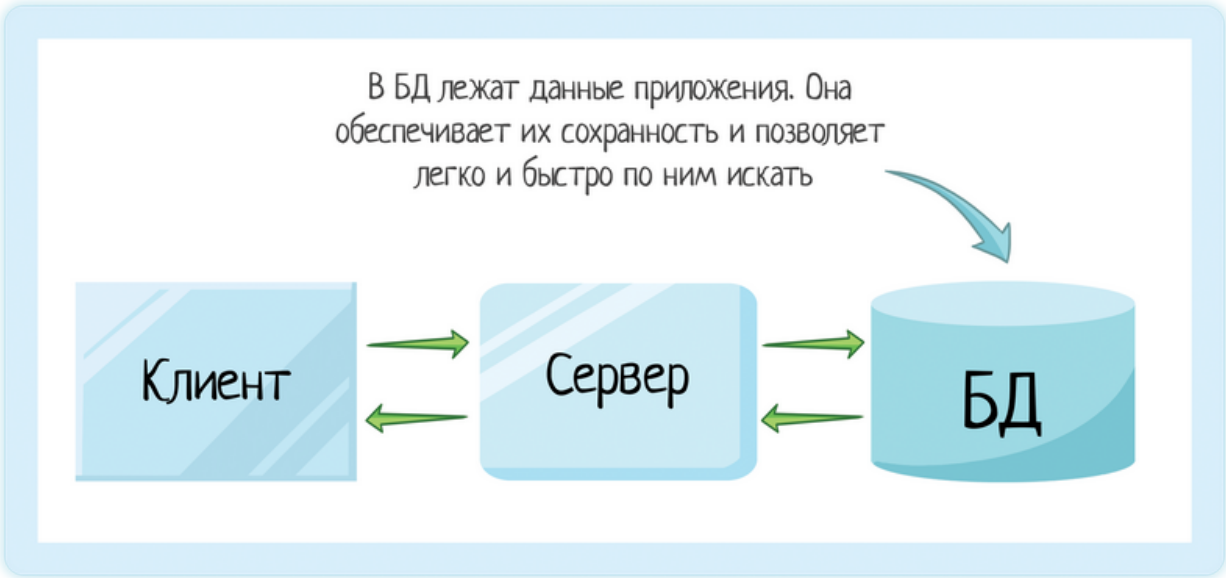


Рисунок 3.2. – База данных

PostgreSQL — это объектно-реляционная система управления базами данных (ORDBMS), наиболее развитая из открытых СУБД в мире. Имеет открытый исходный код и является альтернативой коммерческим базам данных.

Объектно-реляционная модель. Традиционно популярные СУБД — реляционные. Это значит, что данные, которые в них хранятся,

представляются в виде записей, связанных друг с другом отношениями, — relations. Получаются связанные списки, которые могут иметь между собой те или иные отношения, — так и образуется таблица.

Поддержка множества типов данных. Еще одна особенность PostgreSQL — поддержка большого количества типов записи информации. Это не только стандартные целочисленные значения, числа с плавающей точкой, строки и булевы значения («да/нет»), но и денежный, геометрический, перечисляемый, бинарный и другие типы. PostgreSQL «из коробки» поддерживает битовые строки и сетевые адреса, массивы данных, в том числе многомерные, композитные типы и другие сложные структуры. В ней есть поддержка XML, JSON и NoSQL-баз.

3.3 Диаграмма классов

Unified Modeling Language (UML) — унифицированный язык моделирования. Расшифруем: *modeling* подразумевает создание модели, описывающей объект. *Unified* (универсальный, единый) — подходит для широкого класса проектируемых программных систем, различных областей приложений, типов организаций, уровней компетентности, размеров проектов. UML описывает объект в едином заданном синтаксисе, поэтому где бы вы не нарисовали диаграмму, ее правила будут понятны для всех, кто знаком с этим графическим языком — даже в другой стране.

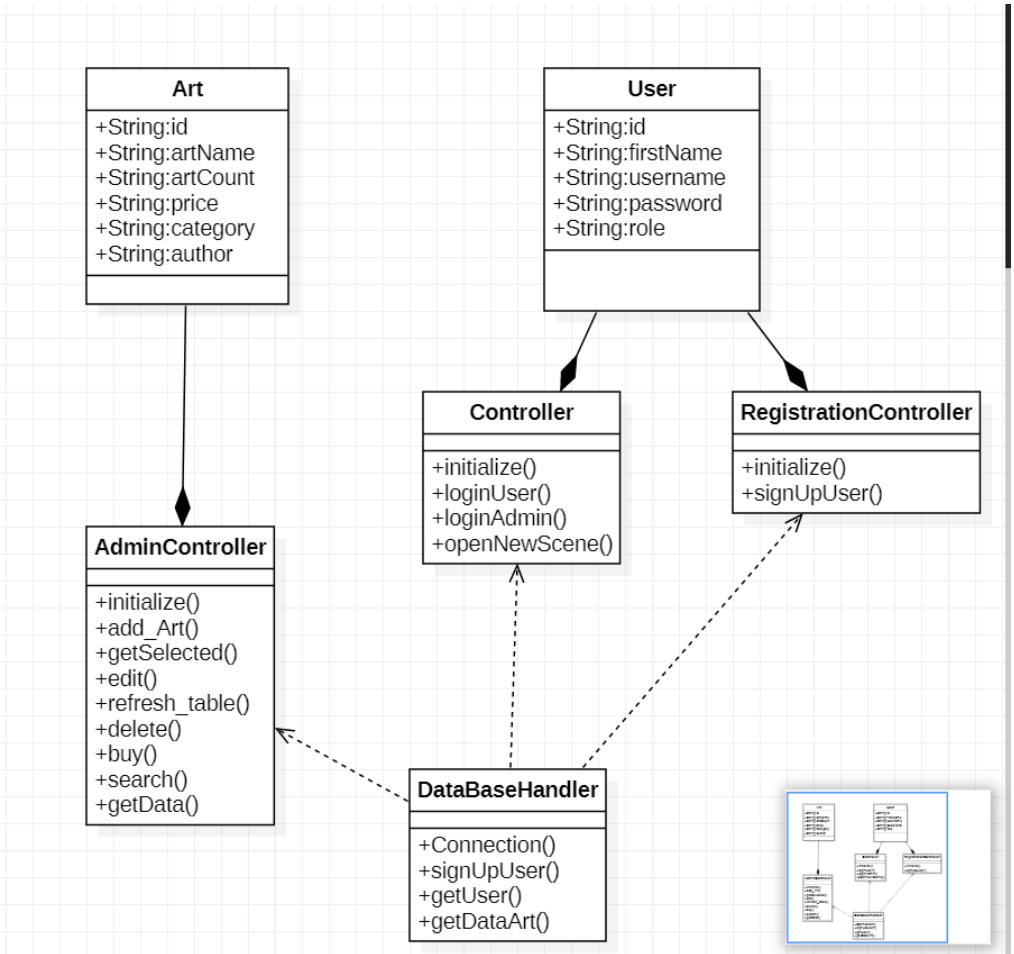


Рисунок 3.3. – Диаграмма классов

4 РЕАЛИЗАЦИЯ МОДУЛЯ

4.1 User

В этом классе мы объявляем новые переменные и строки для построения новой таблицы базы данных.

```
    private String firstname;
    private String username;
    private String password;
    private String role;
```

Поля с именами **firstname, username, password** и **role** предназначены для названия имени пользователя, логина, пароля и роли.

Следом мы создаём конструктор класса User, в качестве параметров которого используем переменные и строки, названия которых схожи с нашими созданными в классе, для создания экземпляра объекта. Иными словами, инициализируем вновь созданный объект перед его использования.

```
    public User(String firstname, String username, String password, String role)
{
    this.firstname = firstname;
    this.username = username;
    this.password = password;
    this.role = role;
}
```

С помощью методов **Set и Get** мы устанавливаем новые значения для полей и получаем их обратно для дальнейшего пользования.

```
    public String getFirstname() {
    return firstname;
}

public void setFirstname(String firstname) {
    this.firstname = firstname;
}

public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
```

```
    this.password = password;
}

public String getRole() {
    return role;
}

public void setRole(String role) {this.role = role;}
```

4.2 Art

В этом классе мы объявляем новые переменные и строки для построения новой таблицы базы данных.

```
    private String id,artName,artCount,price,category,author;
```

Следом мы создаём конструктор класса Art.

```
    public Art(String id,String artName, String artCount, String price, String category, String author) {
    this.id = id;
    this.artName = artName;
    this.artCount = artCount;
    this.price = price;
    this.category = category;
    this.author = author;
}
```

С помощью методов **Set и Get** устанавливаем новые значения для полей и получаем их обратно для дальнейшего пользования.

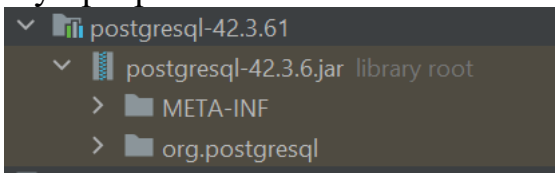
```
    public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}

public String getArtName() {
    return artName;
}

public void setArtName(String artName) {
    this.artName = artName;
}

public String getArtCount() {
```

```
    return artCount;
}

public void setArtCount(String artCount) {
    this.artCount = artCount;
}

public String getPrice() {
    return price;
}

public void setPrice(String price) {this.price = price;}

public String getCategory() {
    return category;
}

public void setCategory(String category) {this.category = category;}

public String getAuthor() {
    return author;
}

public void setAuthor(String author) {this.author = author;}
```

4.3 DatabaseHandler

В этом классе мы выполняем соединение с нашей базой данных для того, чтобы получить данные, хранящиеся там и использовать в нашу программу.

В первую очередь для работы с PostgreSQL в Java необходимо установить официальный драйвер. Сам файл имеет расширение jar и его же нужно положить в папку программы.



С помощью статического метода **Connection ConnectionDB()** мы осуществляем взаимодействие с MySQL через данный драйвер:

```
    public static Connection ConnectDB() {
    try{
        Class.forName("org.postgresql.Driver");
        Connection connection =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/corsses","postgre
s","ntktw2002");
```

```
        // JOptionPane.showMessageDialog(null, "Connection Established");
        return connection;

    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, e);
        return null;
    }
}
```

Для загрузки драйвера здесь применяется строка
Class.*forName*(**"com.postgresql.jdbc.Driver"**);

Метод Class.forName() в качестве параметра принимает строку, которая представляет полный путь к классу драйвера с учетом всех пакетов. В случае PostgreSQL это путь "com.postgresql.jdbc.Driver".

Для соединения именно с нашим сервером базы данных мы используем следующую строку:

```
Connection connection = DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/corsses","postgres","ntktw2002");
```

И с помощью сообщения, вызванное с помощью команды JOptionPane.showMessageDialog(), убеждаемся в том, что наша программа успешно соединена с базой данных, в противном случае – нет.

Статический метод **getDataArt()** позволяет создать нам новый список произведений искусств и заполнить его данными полученными из базы данных.

```
public static ObservableList<Art> getDataArt()  {
    Connection conn = ConnectDB();
    ObservableList<Art> list = FXCollections.observableArrayList();
    try{
        PreparedStatement ps = conn.prepareStatement("SELECT * FROM arts");
        ResultSet rs = ps.executeQuery();

        while(rs.next()){
            list.add(new Art(rs.getString("id"), rs.getString("artName"), rs.getString("artCount"), rs.getString("price"), rs.getString("category"), rs.getString("author")));
        }
    }catch (Exception e){
    }
    return list;
}
```

Статический метод **getUser()** позволяет создать нам новый список клиентов и заполнить его данными полученными из базы данных.

```
    public ResultSet getUser(User user) {
  ResultSet resultSet = null;

  String select = "SELECT * FROM " + Const.USER_TABLE + " WHERE " +
      Const.USER_LOGIN + "=? AND " + Const.USER_PASSWORD + "=? ";
  try {
    PreparedStatement prSt = getDbConnection().prepareStatement(select);
    prSt.setString(1, user.getUsername());
    prSt.setString(2, user.getPassword());

    resultSet = prSt.executeQuery();
  } catch (SQLException e) {
    e.printStackTrace();
  } catch (ClassNotFoundException e) {
    e.printStackTrace();
  }
  return resultSet;
}
```

5 РЕАЛИЗАЦИЯ КОНТРОЛЛЕРА

5.1 Controller

В данном контроллере есть два метода: loginUser для входа пользовтеля и loginAdmin для входа адмінімтратора.

```
    private void loginAdmin(String loginText,String loginPassword) {
    DatabaseHandler databaseHandler = new DatabaseHandler();
    User user = new User();
    user.setUsername(loginText);
    user.setPassword("1234");
    ResultSet result = databaseHandler.getUser(user);

    int counter = 0;

    while(true)
    {
        try {
            if (!result.next()) break;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        counter++;
    }

    if(counter>=1)
    {
            System.out.println("Admin");
            openNewScene("xml/admin-page.fxml");
    }
    else
    {
        Shake userLogin = new Shake(login_f);
        Shake userPass = new Shake(password_f);
        userLogin.playAnimation();
        userPass.playAnimation();
    }
}

private void loginUser(String loginText, String loginPassword) {
    DatabaseHandler databaseHandler = new DatabaseHandler();
    User user = new User();
    user.setUsername(loginText);
    if (loginPassword=="1234") {
        return;
```

```
    } else
    {
        user.setPassword(loginPassword);
    }
    ResultSet result = databaseHandler.getUser(user);

    int counter = 0;

    while(true)
    {
        try {
            if (!result.next()) break;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        counter++;
    }

    if(counter>=1)
    {
        System.out.println("User");
        openNewScene("xml/user-page.fxml");
    }
    else
    {
        Shake userLogin = new Shake(login_f);
        Shake userPass = new Shake(password_f);
        userLogin.playAnimation();
        userPass.playAnimation();
    }
}
```

5.2 AdminController

В методе **Add_art** обрабатывается данные, которые вводит администратор и добавляет новые полученные данные в таблицу базы данных.

```
    public void Add_art(){
    conn = DatabaseHandler.ConnectDB();
    String sql = "insert into arts (artName, artCount, price, category, author) values
(?,?,?,?,?)";
    try{
        pst = conn.prepareStatement(sql);
        pst.setString(1, txt_name.getText());
```

```
            pst.setString(2, txt_count.getText());
            pst.setString(3, txt_price.getText());
            pst.setString(4, txt_category.getText());
            pst.setString(5, txt_author.getText());
            pst.execute();

            JOptionPane.showMessageDialog(null, "Art add successful!");
            RefreshTable();
        }catch (Exception e){
            JOptionPane.showMessageDialog(null, e);
        }
    }
```

В методе **Edit** администратор может изменять данные строки.

```
    public void Edit(){

        try{
            conn = DatabaseHandler.ConnectDB();
            String value1 = txt_id.getText();
            String value2 = txt_name.getText();
            String value3 = txt_count.getText();
            String value4 = txt_price.getText();
            String value5 = txt_category.getText();
            String value6 = txt_author.getText();

            String sql = "update arts set id='"+value1+"', artName='"+value2+"',
artCount='"+
                    value3+"', price='"+value4+"', category='"+value5+"',
author='"+value6+"' where id='"+value1+"' ";
            pst = conn.prepareStatement(sql);
            pst.execute();
            JOptionPane.showMessageDialog(null, "The table has changed!");
            RefreshTable();
        }catch (Exception e){
            JOptionPane.showMessageDialog(null, e);
        }
    }
```

Метод **Delete** позволяет удалять данные (целую строку) по номеру элементов.

```
    public void Delete(){
        conn = DatabaseHandler.ConnectDB();
        String sql = "delete from arts where artName = ?";
        try {
            pst = conn.prepareStatement(sql);
```

```
            pst.setString(1, txt_name.getText());
            pst.execute();
            JOptionPane.showMessageDialog(null, "Delete");
            RefreshTable();
        }catch (Exception e){
            JOptionPane.showMessageDialog(null, e);
        }
    }
```

5.3RegistrationController

Метод для добавления пользователя

```
    private void signUpNewUser() {
DatabaseHandler dbHandler = new DatabaseHandler();

String firstname = signUpName.getText();
String username = signUpUsername.getText();
String password = signUpPassword.getText();
String role = "user";

User user = new User(firstname,username,password,role);

dbHandler.signUpUser(user);
}
```

Интерфейс программы:

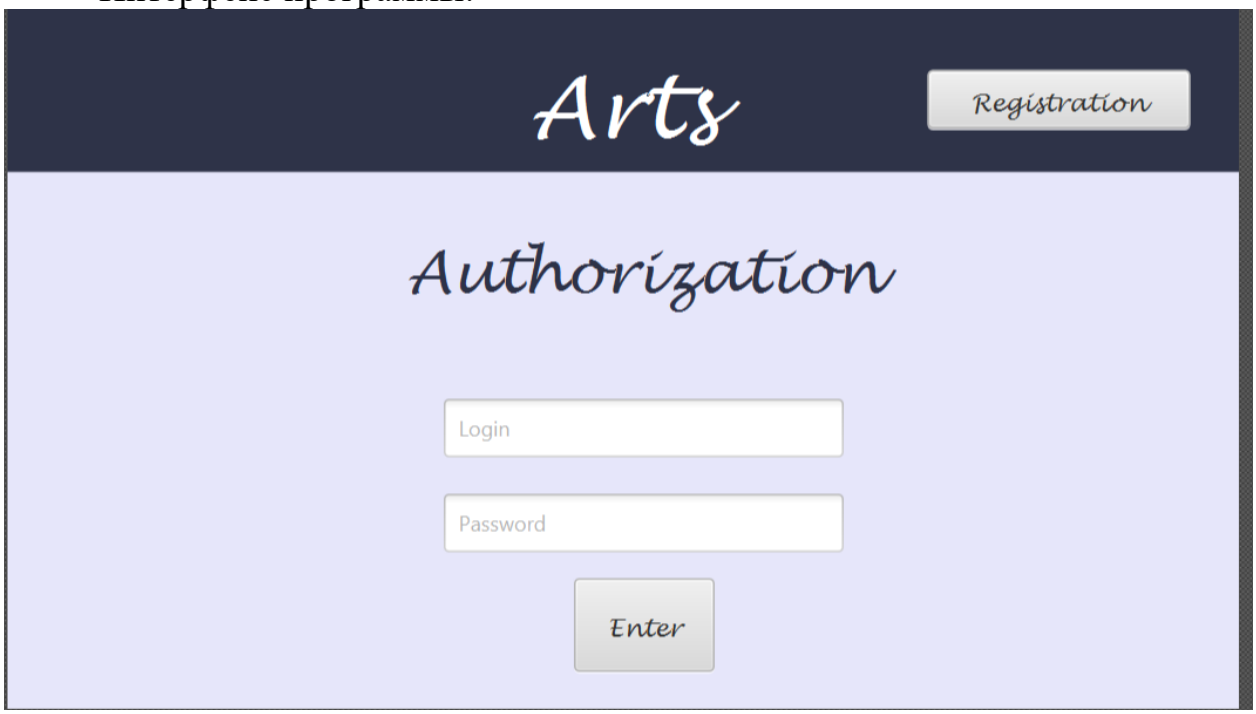


Рисунок 5.1. - Вход

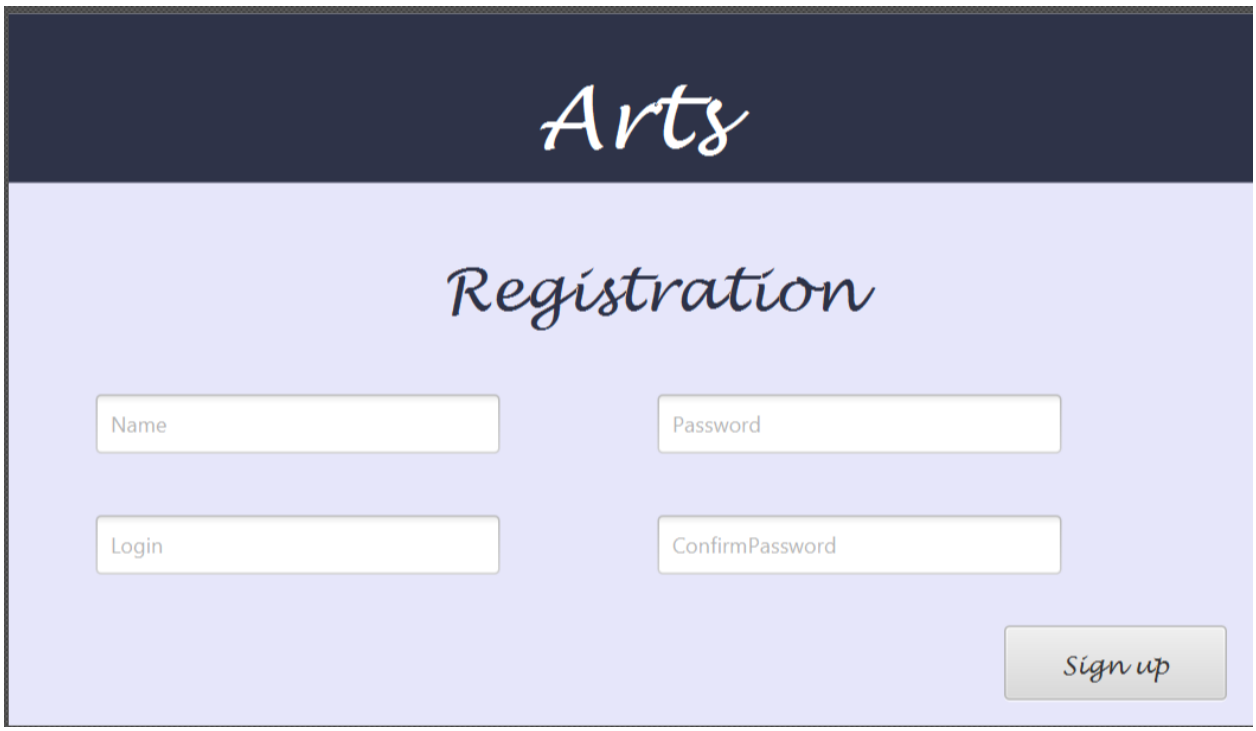


Рисунок 5.2. - Регистрация

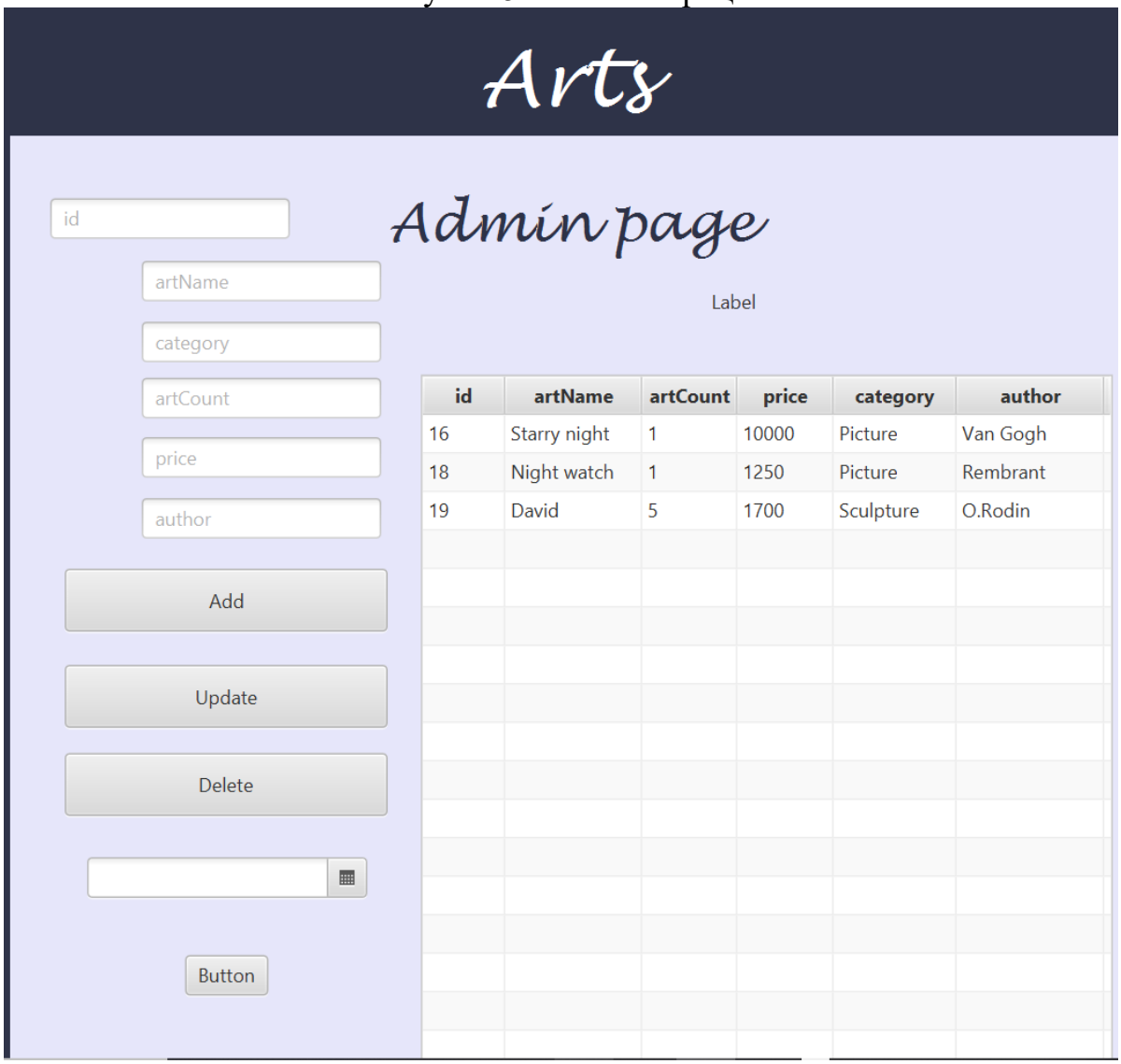


Рисунок 5.3. – Страница администратора

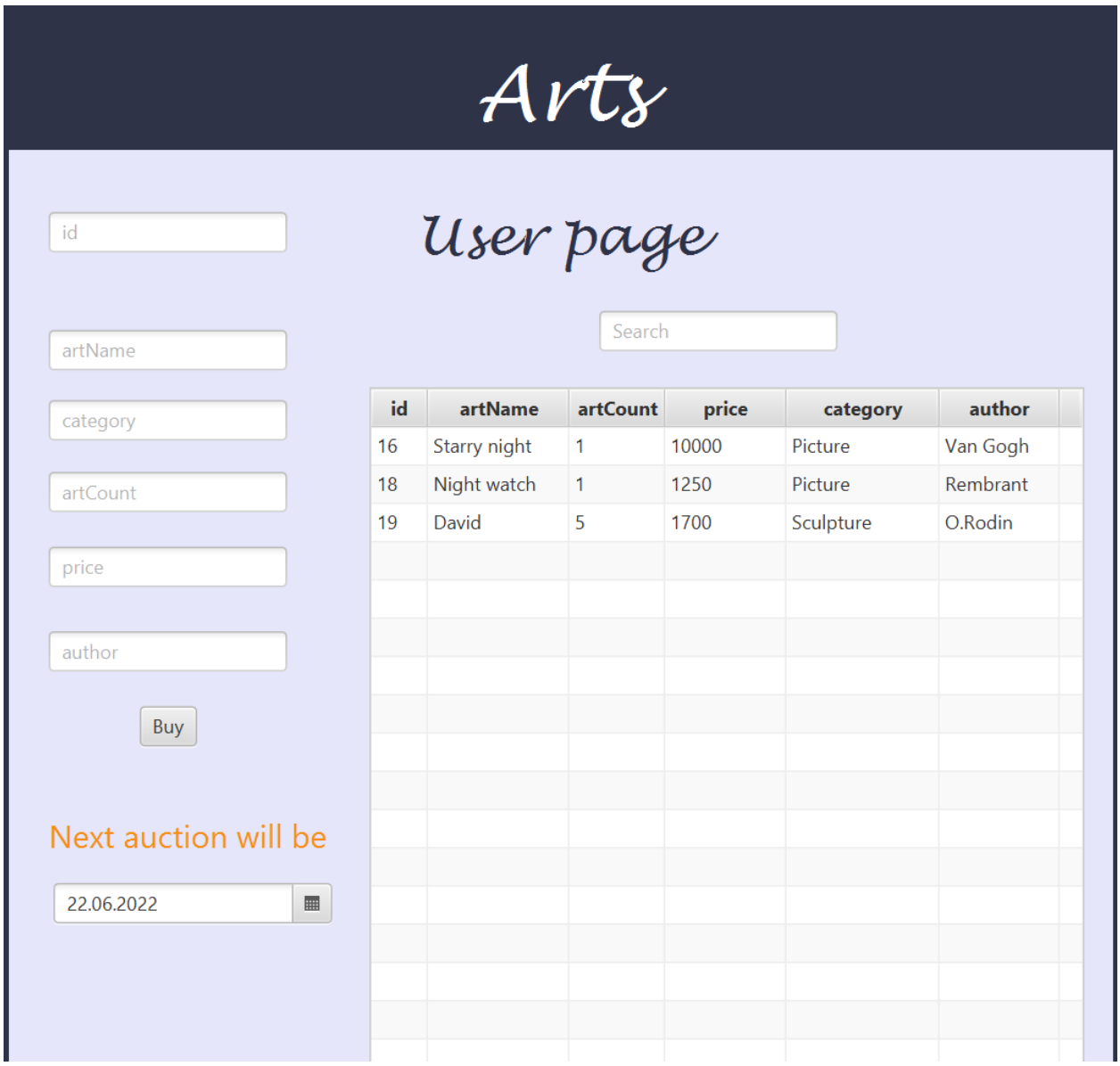


Рисунок 5.3. – Страница пользователя

ЗАКЛЮЧЕНИЕ

Реализована программа с удобным и интуитивно-понятным интерфейсом. Благодаря тому, что все данные хранятся в базе, нет возможности их потери, а простой интерфес способствует тому, что с программой может работать любой неподготовленный пользователь.

В ходе работы было изучено много информации о платформе JavaFX. Были успешно освоены основы языка CSS и основные элементы управления Scene Builder.

При желании улучшения функционала программного средства с легкостью можно добавить новые расширения, функции, редактировать или сделать нововведения в таблицах баз данных.

ИСТОЧНИКИ

1. https://itglobal.com/ru-ru/company/glossary/subd-sistema-upravleniya-bazami-dannyh/
2. https://tproger.ru/experts/oop-in-simple-words/
3. https://webkyrs.info/page/shablon-proektirovaniia-programmirovaniia-mvc
4. https://skillbox.ru/media/code/chto_takoe_mvc_bazovye_kontseptsii_i_primer_prilozheniya/
5. https://eax.me/plantuml/
6. https://habr.com/ru/post/555760/
7. https://blog.skillfactory.ru/glossary/postgresql/
8. https://www.youtube.com/watch?v=3Ht-JMQh2JI
9. https://www.youtube.com/watch?v=8K0RyKwynT8&list=PLxaMIx7eqffLc9mkqFoBFANcZmJVBtzvp&index=61

ПРИЛОЖЕНИЕ

User

```
package com.jsltd.corsser;

public class User {
    private String firstname;
    private String username;
    private String password;
    private String role;

    public User(String firstname, String username, String password, String role) {
        this.firstname = firstname;
        this.username = username;
        this.password = password;
        this.role = role;
    }

    public User() {

    }

    public String getFirstname() {
        return firstname;
    }

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
```

```
    }
}
```

Art

```
package com.jsltd.corsser;

public class Art {
    private String id,artName,artCount,price,category,author;

    public Art()
    {

    }

    public Art(String id,String artName, String artCount, String price, String category, String
author) {
        this.id = id;
        this.artName = artName;
        this.artCount = artCount;
        this.price = price;
        this.category = category;
        this.author = author;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getArtName() {
        return artName;
    }

    public void setArtName(String artName) {
        this.artName = artName;
    }

    public String getArtCount() {
        return artCount;
    }

    public void setArtCount(String artCount) {
        this.artCount = artCount;
    }

    public String getPrice() {
        return price;
    }
```

```
    public void setPrice(String price) {
        this.price = price;
    }

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }
}
```

Controller

```
package com.jsltd.corsser;

import java.io.IOException;
import java.net.URL;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ResourceBundle;

import com.jsltd.corsser.animatin.Shake;
import com.jsltd.corsser.dataBase.DatabaseHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

public class Controller {

    @FXML
    private ResourceBundle resources;

    @FXML
    private URL location;

    @FXML
    private Button authSignButton;
```

```
@FXML
private Button loginSignUpButton;

@FXML
private TextField login_f;

@FXML
private PasswordField password_f;

@FXML
void initialize() {
    authSignButton.setOnAction(event ->{
        String loginText = login_f.getText().trim();
        String loginPassword = password_f.getText().trim();

        if(!loginText.equals("") && !loginPassword.equals(""))
        {
            loginUser(loginText,loginPassword);
            loginAdmin(loginText,loginPassword);
        }
        else
            System.out.println("Error! Login and password is empty");
    });

    loginSignUpButton.setOnAction(event ->{
        openNewScene("xml/enter-view.fxml");
    });
}

private void loginAdmin(String loginText,String loginPassword) {
    DatabaseHandler databaseHandler = new DatabaseHandler();
    User user = new User();
    user.setUsername(loginText);
    user.setPassword("1234");
    ResultSet result = databaseHandler.getUser(user);

    int counter = 0;

    while(true)
    {
        try {
            if (!result.next()) break;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        counter++;
    }

    if(counter>=1)
    {
            System.out.println("Admin");
            openNewScene("xml/admin-page.fxml");
```

```
    }
    else
    {
        Shake userLogin = new Shake(login_f);
        Shake userPass = new Shake(password_f);
        userLogin.playAnimation();
        userPass.playAnimation();
    }
}

private void loginUser(String loginText, String loginPassword) {
    DatabaseHandler databaseHandler = new DatabaseHandler();
    User user = new User();
    user.setUsername(loginText);
    if (loginPassword=="1234") {
        return;
    } else
    {
        user.setPassword(loginPassword);
    }
    ResultSet result = databaseHandler.getUser(user);

    int counter = 0;

    while(true)
    {
        try {
            if (!result.next()) break;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        counter++;
    }

    if(counter>=1)
    {
        System.out.println("User");
        openNewScene("xml/user-page.fxml");
    }
    else
    {
        Shake userLogin = new Shake(login_f);
        Shake userPass = new Shake(password_f);
        userLogin.playAnimation();
        userPass.playAnimation();
    }
}

public void openNewScene(String window)
{
    loginSignUpButton.getScene().getWindow().hide();
```

```
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(getClass().getResource(window));

        try {
            loader.load();
        } catch (IOException e) {
            e.printStackTrace();
        }

        Parent root = loader.getRoot();
        Stage stage = new Stage();
        stage.setScene(new Scene(root));
        stage.showAndWait();
    }
}
```

RegistrationController

```
package com.jsltd.corsser;

import java.net.URL;
import java.sql.Connection;
import java.sql.Statement;
import java.util.ResourceBundle;

import com.jsltd.corsser.dataBase.DatabaseHandler;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;

public class RegistrationController {

    @FXML
    private ResourceBundle resources;

    @FXML
    private URL location;

    @FXML
    private TextField signUpUsername;

    @FXML
    private TextField signUpName;

    @FXML
    private PasswordField signUpPassword;

    @FXML
    private Button signUpButton;

    @FXML
    void initialize() {
```

```
        signUpButton.setOnAction(event -> {

            signUpNewUser();

        });
    }

    private void signUpNewUser() {
        DatabaseHandler dbHandler = new DatabaseHandler();

        String firstname = signUpName.getText();
        String username = signUpUsername.getText();
        String password = signUpPassword.getText();
        String role = "user";

        User user = new User(firstname,username,password,role);

        dbHandler.signUpUser(user);
    }
}
```

AdminController

```
package com.jsltd.corsser;

import java.net.URL;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.time.LocalDate;
import java.util.ResourceBundle;

import com.jsltd.corsser.dataBase.DatabaseHandler;
import javafx.collections.ObservableList;
import javafx.collections.transformation.FilteredList;
import javafx.collections.transformation.SortedList;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;

import javax.swing.*;

public class AdminController {

    @FXML
    private ResourceBundle resources;

    @FXML
    private URL location;

    @FXML
    private DatePicker data;
```

```
@FXML
private TableColumn<Art, String> col_artCount;

@FXML
private TableColumn<Art, String> col_artName;

@FXML
private TableColumn<Art, String> col_id;

@FXML
private TableColumn<Art, String> col_price;

@FXML
private TableColumn<Art, String> col_category;

@FXML
private TableColumn<Art, String> col_author;

@FXML
private TableView<Art> table_arts;

@FXML
private TextField txt_id;

@FXML
private TextField txt_count;

@FXML
private TextField txt_name;

@FXML
private TextField txt_price;

@FXML
private TextField txt_category;

@FXML
private TextField txt_author;

@FXML
private TextField txt_search;

@FXML
private Label lb;

ObservableList<Art> dataList;

ObservableList<Art> listM;

int index = -1;

Connection conn = null;
```

```
ResultSet rs = null;
PreparedStatement pst = null;

@FXML
public void initialize() {
    RefreshTable();
    Search();
    data.setValue(LocalDate.of(2022,06,22));
}

public void Add_art(){
    conn = DatabaseHandler.ConnectDB();
    String sql = "insert into arts (artName, artCount, price, category, author) values (?,?,?,?,?)";
    try{
        pst = conn.prepareStatement(sql);
        pst.setString(1, txt_name.getText());
        pst.setString(2, txt_count.getText());
        pst.setString(3, txt_price.getText());
        pst.setString(4, txt_category.getText());
        pst.setString(5, txt_author.getText());
        pst.execute();

        JOptionPane.showMessageDialog(null, "Art add successful!");
        RefreshTable();
    }catch (Exception e){
        JOptionPane.showMessageDialog(null, e);
    }
}


//Метод по выборке
public void getSelected(javafx.scene.input.MouseEvent mouseEvent) {
    index = table_arts.getSelectionModel().getSelectedIndex();

    if(index <= -1){

        return;
    }

    txt_id.setText(col_id.getCellData(index).toString());
    txt_name.setText(col_artName.getCellData(index).toString());
    txt_count.setText(col_artCount.getCellData(index).toString());
    txt_price.setText(col_price.getCellData(index).toString());
    txt_category.setText(col_category.getCellData(index).toString());
    txt_author.setText(col_author.getCellData(index).toString());
}

public void Edit(){

    try{
        conn = DatabaseHandler.ConnectDB();
        String value1 = txt_id.getText();
```

```
            String value2 = txt_name.getText();
            String value3 = txt_count.getText();
            String value4 = txt_price.getText();
            String value5 = txt_category.getText();
            String value6 = txt_author.getText();

            String sql = "update arts set id='"+value1+"', artName='"+value2+"', artCount='"+
                    value3+"', price='"+value4+"', category='"+value5+"', author='"+value6+"' where
id='"+value1+"' ";
            pst = conn.prepareStatement(sql);
            pst.execute();
            JOptionPane.showMessageDialog(null, "The table has changed!");
            RefreshTable();
        }catch (Exception e){
            JOptionPane.showMessageDialog(null, e);
        }
    }

    public void RefreshTable(){
        col_id.setCellValueFactory(new PropertyValueFactory<Art, String>("id"));
        col_artName.setCellValueFactory(new PropertyValueFactory<Art, String>("artName"));
        col_artCount.setCellValueFactory(new PropertyValueFactory<Art, String>("artCount"));
        col_price.setCellValueFactory(new PropertyValueFactory<Art, String>("price"));
        col_category.setCellValueFactory(new PropertyValueFactory<Art, String>("category"));
        col_author.setCellValueFactory(new PropertyValueFactory<Art, String>("author"));


        listM = DatabaseHandler.getDataArt();
        table_arts.setItems(listM);
    }

    public void Delete(){
        conn = DatabaseHandler.ConnectDB();
        String sql = "delete from arts where artName = ?";
        try {
            pst = conn.prepareStatement(sql);
            pst.setString(1, txt_name.getText());
            pst.execute();
            JOptionPane.showMessageDialog(null, "Delete");
            RefreshTable();
        }catch (Exception e){
            JOptionPane.showMessageDialog(null, e);
        }
    }

    public void Buy(){
        conn = DatabaseHandler.ConnectDB();
        String sql = "delete from arts where artName = ?";
        try {
            pst = conn.prepareStatement(sql);
            pst.setString(1, txt_name.getText());
            pst.execute();
```

```
            JOptionPane.showMessageDialog(null, "Purchase completed");
            RefreshTable();
        }catch (Exception e){
            JOptionPane.showMessageDialog(null, e);
        }
    }

    @FXML
    void Search(){
        col_id.setCellValueFactory(new PropertyValueFactory<Art, String>("id"));
        col_artName.setCellValueFactory(new PropertyValueFactory<Art, String>("artName"));
        col_artCount.setCellValueFactory(new PropertyValueFactory<Art, String>("artCount"));
        col_price.setCellValueFactory(new PropertyValueFactory<Art, String>("price"));
        col_category.setCellValueFactory(new PropertyValueFactory<Art, String>("category"));
        col_author.setCellValueFactory(new PropertyValueFactory<Art, String>("author"));



        dataList = DatabaseHandler.getDataArt();
        table_arts.setItems(dataList);
        FilteredList<Art> filteredData = new FilteredList<>(dataList, b -> true);
        txt_search.textProperty().addListener((observable, oldValue, newValue) -> {
            filteredData.setPredicate(artts -> {
                if(newValue == null || newValue.isEmpty()){
                    return true;
                }
                String lowerCaseFilter = newValue.toLowerCase();

                if(artts.getArtName().toLowerCase().indexOf(lowerCaseFilter) != -1){
                    return true;//фильтр по названию
                }else if(artts.getPrice().toLowerCase().indexOf(lowerCaseFilter) != -1){
                    return true;//фильтр по цене
                } else if(artts.getCategory().toLowerCase().indexOf(lowerCaseFilter) != -1){
                    return true;
                }else if(artts.getAuthor().toLowerCase().indexOf(lowerCaseFilter) != -1){
                    return true;
                }

                else
                    return false;
            });
        });
        SortedList<Art> sortedData = new SortedList<>(filteredData);
        sortedData.comparatorProperty().bind(table_arts.comparatorProperty());
        table_arts.setItems(sortedData);
    }

    @FXML
    void getData(javafx.event.ActionEvent event){
        System.out.println(data.getValue().toString());
    }
}
```

DatabaseHandler

```
package com.jsltd.corsser.dataBase;

import com.jsltd.corsser.Art;
import com.jsltd.corsser.User;
import com.jsltd.corsser.dataBase.Configs;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;

import javax.swing.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.ResultSet;

public class DatabaseHandler extends Configs {
    Connection dbConnection;

    public Connection getDbConnection()
            throws ClassNotFoundException,SQLException{
        String connectionString = "jdbc:postgresql://127.0.0.1:5432/corsses";

        Class.forName("org.postgresql.Driver");

        dbConnection = DriverManager.getConnection(connectionString, dbUser,dbPass);

        return dbConnection;
    }

    public static Connection ConnectDB() {
        try{
            Class.forName("org.postgresql.Driver");
            Connection connection =
DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/corsses","postgres","ntktw2002
");
            // JOptionPane.showMessageDialog(null, "Connection Established");
            return connection;

        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, e);
            return null;
        }
    }

    public void signUpUser(User user)
    {
        String insert = "INSERT INTO
"+Const.USER_TABLE+"("+Const.USER_NAME+","+Const.USER_LOGIN+","+
                Const.USER_PASSWORD+","+Const.USER_ROLE+")"+"VALUES(?,?,?,?)";
```

```
        try {
            PreparedStatement prSt = getDbConnection().prepareStatement(insert);
            prSt.setString(1,user.getFirstname());
            prSt.setString(2,user.getUsername());
            prSt.setString(3,user.getPassword());
            prSt.setString(4,user.getRole());

            prSt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }


    public ResultSet getUser(User user) {
        ResultSet resultSet = null;

        String select = "SELECT * FROM " + Const.USER_TABLE + " WHERE " +
                Const.USER_LOGIN + "=? AND " + Const.USER_PASSWORD + "=? ";
        try {
            PreparedStatement prSt = getDbConnection().prepareStatement(select);
            prSt.setString(1, user.getUsername());
            prSt.setString(2, user.getPassword());

            resultSet = prSt.executeQuery();
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        return resultSet;
    }

    public static ObservableList<Art> getDataArt()  {
        Connection conn = ConnectDB();
        ObservableList<Art> list = FXCollections.observableArrayList();
        try{
            PreparedStatement ps = conn.prepareStatement("SELECT * FROM arts");
            ResultSet rs = ps.executeQuery();

            while(rs.next()){
                list.add(new Art(rs.getString("id"), rs.getString("artName"), rs.getString("artCount"),
rs.getString("price"), rs.getString("category"),  rs.getString("author")));
            }
        }catch (Exception e){
        }
        return list;
    }
}
```