Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Объектно-ориентированное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
на тему:

“Туристическая фирма”

Выполнил: студент группы 953505
Матвеев Илья Андреевич

Проверил:
Летохо Александр Сергеевич

Минск 2021

Содержание

Введение 2

Постановка задачи 3

Часть 1. Инфологическое проектирование Базы данных
- 1.1 Анализ предметной области 4
- 1.2 Атрибуты. Идентифицирующие ключи 4
- 1.3 UML-диаграмма 5

Часть 2. Логическое проектирование
- 2.1 Отношения 6
- 2.2 UML-диаграмма 6
- 2.3 Описание классов 9
  - 2.3.1 Класс Main 9
  - 2.3.2 Класс EntryController 9
  - 2.3.3 Класс LoginController 10
  - 2.3.4 Класс RegisterController 11
  - 2.3.5 Класс TripController 13
  - 2.3.6 Класс Trip 14
  - 2.3.7 Класс DatabaseCollection 15
  - 2.3.8 Класс Order 15

Часть 3. Создание форм 17

Часть 4. Демо программы 18

Заключение 21

Список использованных источников 22

Приложение 1. Исходный код 23

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 путём сочетания объектно-ориентированных языков программирования и СУБД.
На данный момент существует множество готовых инструментов, которые позволяют разрабатывать графический пользовательский интерфейс и работать с базами данных.

Данный проект реализован на языке Java в среде разработки IntelliJ Idea. Для базы данных использовался СУБД MySQL в составе дистрибутива для разработки веб-сервера XAMPP. Создание графического пользовательского интерфейса осуществлялось в среде разработки Scene Builder с использованием пакета JavaFX.

Информационная система “Туристическая фирма” предназначена для последовательного отображения текущего состояния наличия путевок, добавление путем логирование / регистрации и последующий заказ.

Целью данного курсового проекта является создание удобного и понятного приложения, которое позволит новому пользователю без труда выяснить о наличие тех или иных путевок, их резервирования и добавления.

Постановка задачи

Информационная система “Фирма по продаже запчастей” должна содержать следующий функционал:

- Регистрация новых пользователей
- Валидацию введенных данных
- Проверку на существование пользователя (username - unique)
- Логирование существующих пользователей
- Отображение сообщений при неудачном логировании или регистрации
- Каталог с полным списком путевок, с возможностью выбора конкретной путевки
- Добавление выбранной путевки в корзину
- Выход из аккаунта

Все данные о доступных товарах, сотрудниках и поставщиках должны храниться в базе данных. Таблица аккаунтов должна содержать следующие поля:

- Уникальный id
- Уникальный username
- Имя, Фамилия
- Поле для хранения пароля

Таблица поездок должна содержать следующие поля:

- Уникальный id

- Страна
- Цена

Таблица заказа должна содержать следующие поля:

- Уникальный id
- Полную сумму
- Уникальный ключ на user_id (foreigh key)

Также в базе данных должны быть реализованы следующие хранимые процедуры:

- Добавление нового пользователя
- Получение пользователя по username и паролю
- Получить список путевок

В пояснительной записке необходимо описать логику построения и работы программы.

Часть 1. Инфологическое проектирование Базы данных

В данном курсовом проекте в качестве хранилища данных используется СУБД MySQL в составе пакета XAMPP, поскольку она является локальной, не требует установки в системе дополнительного ПО и обеспечивает хорошую защиту данных от несанкционированного доступа и к тому же имеет поддержку со стороны Java. Также данная СУБД имеет множество простых в использовании визуальных инструментов, что значительно ускоряет построение необходимой архитектуры БД.

1.1 Анализ предметной области

Предметная область информационной системы “Фирма по продаже запчастей” должна содержать информацию о товарах и поставщиках. Для этого были выделены следующие Объекты:

- account
- logs
- order
- trip

1.2 Атрибуты. Идентифицирующие ключи

Проанализировав объекты рассматриваемой предметной области можно выделить следующие атрибуты и идентифицирующие ключи:

- Объект account содержит следующие атрибуты: account_id, firstname, lastname, username, password, role.
  Идентифицирующим ключом является : **account_id**.
- Объект logs содержит следующие атрибуты: log_id, log_nickname
  Идентифицирующим ключом является : **log_id**.
- Объект Product содержит следующие атрибуты: order_id, totalprice, user_id
  Идентифицирующим ключом является : **user_id**.
- Объект order содержит следующие атрибуты: trip_id, country, cost.
  Идентифицирующим ключом является : **trip_id**.

1.3 UML-диаграмма (таблиц базы данных)

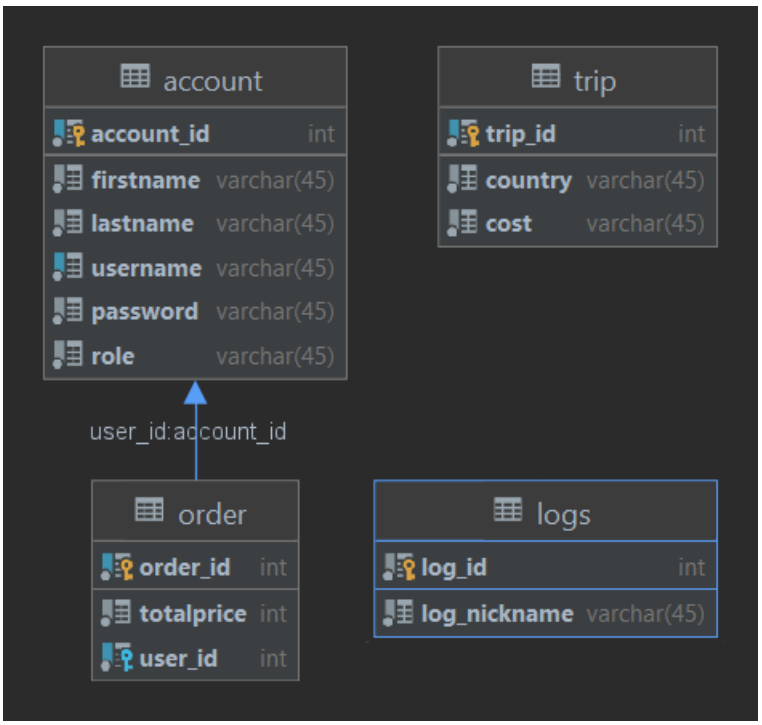

Рисунок 1 – UML диаграмма связей объектов: **account, trip, order, logs**

Часть 2. Логическое проектирование

2.1 Отношения

На основе рассмотренных сущностей и атрибутов можно построить следующие отношения:

Таблица 1. Отношение account

Имя атрибута	Тип	В программе
Имя	Текстовый	firstname
Фамилия	Текстовый	lastname
Никнейм	Текстовый	nickname
Роль	Текстовый	role
Индефикатор	Числовой	user_id

Таблица 2. Отношение order

Имя атрибута	Тип	В программе
Индефикатор	Числовой	user_id

2.2 UML-диаграмма

UML - это унифицированный язык моделирования. UML диаграммы могут применяться для различных задач:

- Проектирование. UML-диаграммы помогут при моделировании архитектуры больших проектов, в которой можно собрать как крупные, так и более мелкие детали и нарисовать каркас (схему) приложения. По нему впоследствии будет строиться код.
- Реверс-инжиниринг — создание UML-модели из существующего кода приложения, обратное построение. Может применяться, например, на проектах поддержки, где есть написанный код, но документация неполная или отсутствует.
- Из моделей можно извлекать текстовую информацию и генерировать относительно удобочитаемые тексты — документировать. Текст и графика будут дополнять друг друга.

Как и любой другой язык, UML имеет собственные правила оформления моделей и синтаксис. С помощью графической нотации UML можно визуализировать систему, объединить все компоненты в единую структуру, уточнять и улучшать модель в процессе работы. На общем уровне графическая нотация UML содержит 4 основных типа элементов:

- фигуры;
- линии;
- значки;

- надписи.

UML-нотация является де-факто отраслевым стандартом в области разработки программного обеспечения, ИТ-инфраструктуры и бизнес-систем.

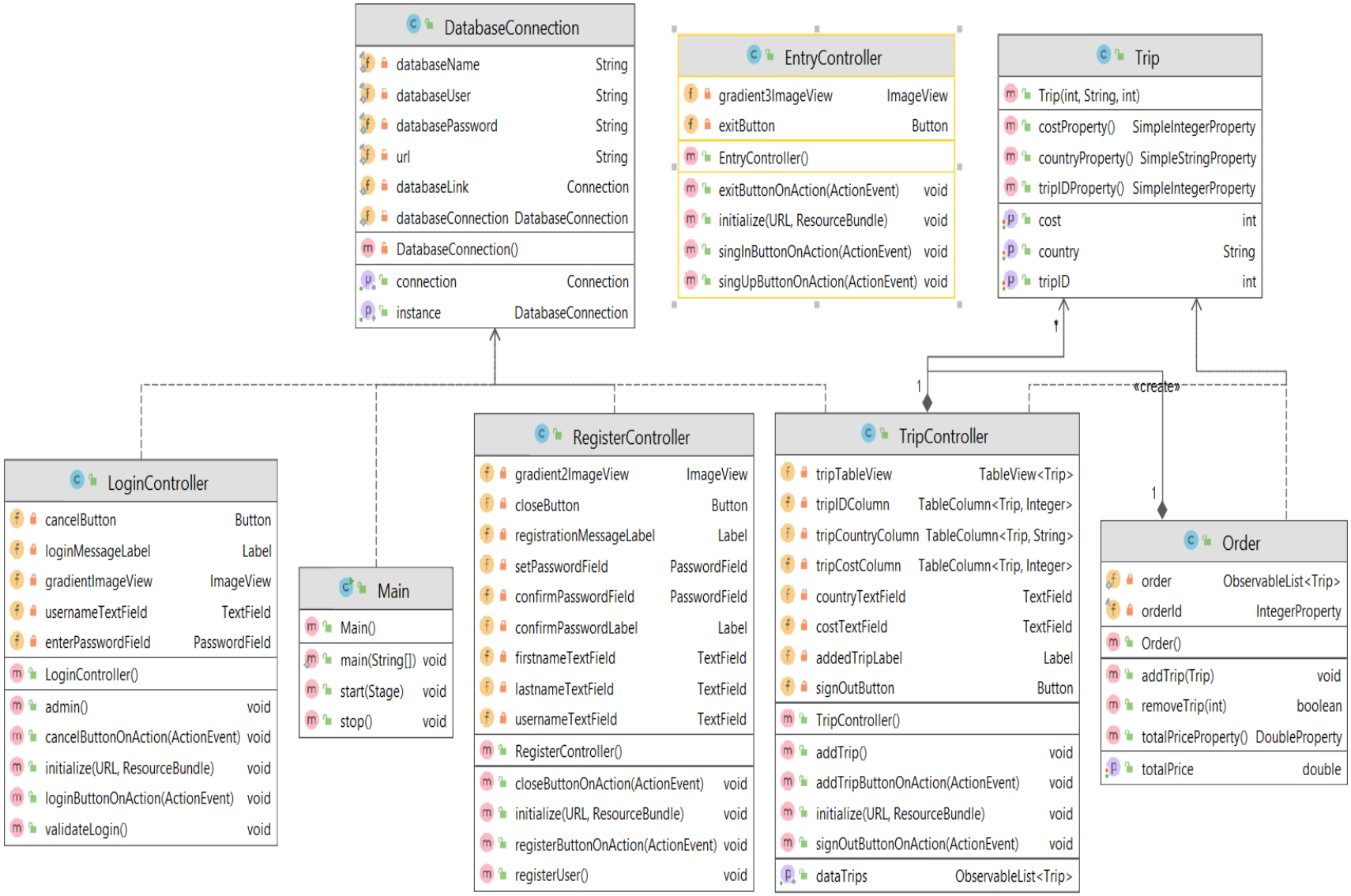
DatabaseConnection
databaseName String
databaseUser String
databasePassword String
url String
databaseLink Connection
databaseConnection DatabaseConnection
DatabaseConnection()
connection Connection
instance DatabaseConnection
EntryController
gradient3ImageView ImageView
exitButton Button
EntryController()
exitButtonOnAction(ActionEvent) void
initialize(URL, ResourceBundle) void
singInButtonOnAction(ActionEvent) void
singUpButtonOnAction(ActionEvent) void
Trip
Trip(int, String, int)
costProperty() SimpleIntegerProperty
countryProperty() SimpleStringProperty
tripIDProperty() SimpleIntegerProperty
cost int
country String
tripID int
LoginController
cancelButton Button
loginMessageLabel Label
gradientImageView ImageView
usernameTextField TextField
enterPasswordField PasswordField
LoginController()
admin() void
cancelButtonOnAction(ActionEvent) void
initialize(URL, ResourceBundle) void
loginButtonOnAction(ActionEvent) void
validateLogin() void
Main
Main()
main(String[]) void
start(Stage) void
stop() void
RegisterController
gradient2ImageView ImageView
closeButton Button
registrationMessageLabel Label
setPasswordField PasswordField
confirmPasswordField PasswordField
confirmPasswordLabel Label
firstnameTextField TextField
lastnameTextField TextField
usernameTextField TextField
RegisterController()
closeButtonOnAction(ActionEvent) void
initialize(URL, ResourceBundle) void
registerButtonOnAction(ActionEvent) void
registerUser() void
TripController
tripTableView TableView<Trip>
tripIDColumn TableColumn<Trip, Integer>
tripCountryColumn TableColumn<Trip, String>
tripCostColumn TableColumn<Trip, Integer>
countryTextField TextField
costTextField TextField
addedTripLabel Label
signOutButton Button
TripController()
addTrip() void
addTripButtonOnAction(ActionEvent) void
initialize(URL, ResourceBundle) void
signOutButtonOnAction(ActionEvent) void
dataTrips ObservableList<Trip>
Order
order ObservableList<Trip>
orderId IntegerProperty
Order()
addTrip(Trip) void
removeTrip(int) boolean
totalPriceProperty() DoubleProperty
totalPrice double
1
1
1
«create»

Рисунок 2 - UML диаграмма программы

2.3 Описание классов

Подробно рассмотрим каждый класс изображенный на рис. 2. Всего программа состоит из следующих классов:

- Main
- EntryController
- RegisterController
- LoginController
- TripController
- Order
- Trip
- DatabaseConnection

2.3.1 Класс Main

Данный класс содержит следующие атрибут:

- stage – является основным окном программы. Используется в рамках двух классов для плавной смены сцен логирования, регистрации и их взаимодействия.

Также в этом классе содержится три метода:

- public static void main(String[] args) - является основным методом программы.
- public void start(Stage primaryStage) - метод, который вызывается при запуске программы. Задача этого метода - создать основное окно и отобразить стартовую сцену для выбора пользователем входа или регистрации.

2.3.2 Класс EntryController

Данный класс содержит следующие атрибуты:

- `gradient3ImageView`

объект на окне заливающий фон в градиент

- `exitButton`

  поле ля полной остановки выполнения программы

В этом классе содержится три метода:

- `public void initialize(URL url, ResourceBundle resourceBundle)`

  метод из имплеметнируемого интерфейса Initializable. Необходим для инициализации находящейся на ImageView картинок.

- `public void singInButtonOnAction(ActionEvent event)`

обработчик событий по нажатию на кнопку Login. Создается новый экземпляр Stage для отображения формы tlogin.fxml

- `public void singUpButtonOnAction(ActionEvent event)`

  эквивалентно логированию, только будет вызвана форма register.fxml.

2.3.3 Класс RegisterController

Данный класс содержит следующие атрибуты:

```
@FXML
private ImageView gradient2ImageView;

@FXML
private Button closeButton;

@FXML
private Label registrationMessageLabel;

@FXML
private PasswordField setPasswordField;

@FXML
private PasswordField confirmPasswordField;

@FXML
private Label confirmPasswordLabel;

@FXML
private TextField firstnameTextField;

@FXML
private TextField lastnameTextField;

@FXML
private TextField usernameTextField;
```

В этом классе содержится также три метода:

- `public void initialize(URL url, ResourceBundle resourceBundle)`

инициализация такая же как в entry

- `public void closeButtonOnAction (ActionEvent event) {`

обработчик закрытия формы, на которой находится эта кнопка, то есть – текущей.

- `public void registerButtonOnAction (ActionEvent event) {`

обработчик и последующий условный вызов функции регистрации пользователя при выполнении требования равенства паролей.

- `public void registerUser() {`

главная функции регистрации, тут уже имеется подклчение через хост к базе данных и последующаяя работа с ее таблицами, описанными выше. Задается строчный запрос к базе данных, состоящий из поле, которые мы заполнили для регистрации. В случае успешной регистрации обновляется значение label на “успешное регистрирование”.

2.3.4 Класс LoginController

Данный класс содержит следующие атрибуты:

```
@FXML
private Button cancelButton;
@FXML
private Label loginMessageLabel;
@FXML
private ImageView gradientImageView;
@FXML
private TextField usernameTextField;
@FXML
private PasswordField enterPasswordField;
```

В этом классе содержатся следующие методы:

- `public void initialize(URL url, ResourceBundle resourceBundle)`
  инициализация такая же для картинки
- `public void loginButtonOnAction(ActionEvent event)`
  обработчкик события нажатия на кнопку логирования. Проверка на пустоту в поле заполнения nickname
- `public void cancelButtonOnAction(ActionEvent event)`

обработчкик события нажатия на кнопку логирования. При ее нажатии происходит возврат на предыдущию форму, то есть на форму логирования / регистрации.

```
public void validateLogin() {
```

функция для валидации введенного логина и пароля
используется запрос к базе данных с полями введенными на форму в поля nickname и password. Идет цикличная сверка с полученными в resultset значениями для обнаружения или (не) в таблице совпадающего логина и пароля. В таком случае возвращает 1.
в случае нахождения пользователя зарегестрированного в базе данных идет проверка его роли. В случае, если его роль – admin, а именно создается еще один запрос к базе данных для вывода столбца role, то вызывается функция admin(), которая в свою очередь ведет к новой форме.

```
public void admin() {
```

вызов для отображения новой формы, в которой будет отображаться в виде TableView данные из таблицы trip базы данных.
будет доступно добавлением новых записей, их сверка, редактирование.

2.3.5 Класс TripController

Данный класс содержит следующие атрибуты:

```
@FXML
private TableView<Trip> tripTableView;

@FXML
private TableColumn<Trip, Integer> tripIDColumn;

@FXML
private TableColumn<Trip, String> tripCountryColumn;

@FXML
private TableColumn<Trip, Integer> tripCostColumn;

@FXML
private TextField countryTextField;

@FXML
private TextField costTextField;

@FXML
private Label addedTripLabel;

@FXML
private Button signOutButton;
```

В этом классе содержатся следующие методы:

```
public void initialize(URL url, ResourceBundle resourceBundle)
```

инициализация, но уже кардинально отличающаяся от предыдущих. здесь мы используем встроенные в атрибуты класса функции сеттеров для того чтобы через ObservalList создать список Trip-ов с помощью функции, возвращающей этот самый список из базы данных (с помощью запроса). Описание метода получения, а именно **getDataTrips()** будет ниже.

```
    public void addTripButtonOnAction (ActionEvent event) {
```

обработчик событий для кнопки добавления новой записи в базу данных, ну и по совместительству в TableView.

```
public void signOutButtonOnAction (ActionEvent event) {
```

обработчик для выхода с текущей формы на предыдущую, а именно на tlogin.fxml.

```
public void addTrip(){
```

функция по добавлению новой записи в базу данных и получается в TableView с ее последующим обновлением на текущей форме.

создается запрос к базе данных для вставки новой записи в таблицу trip и задаются значения для записи, берущиеся из полей на форме, в которые мы заносим свои данные для новой записи.

```
public static ObservableList<Trip> getDataTrips() {
```

функция необходимая для возвращения листа trip-ов для заполнения им TableView. Создается запрос к базе данных для вывода всех значений из таблицы trip.

2.3.6 Класс Order

Данный класс содержит следующие атрибуты:

```
private final IntegerProperty orderId;
private final DoubleProperty totalPrice;
```

Конструктор по умолчанию сгенерированный

```
public Order() {
    orderId = new SimpleIntegerProperty();
    totalPrice = new SimpleDoubleProperty();
}
```

- ну и также сгенерированные геттеры и сеттеры, а также удаление по индексу из листа

```
public void setTotalPrice(double totalPrice) { this.totalPrice.set(totalPrice); }

public double getTotalPrice() { return totalPrice.get(); }

public DoubleProperty totalPriceProperty() { return totalPrice; }

public void addTrip(Trip trip) { order.add(trip); }

public boolean removeTrip(int tripNum){
    order.remove(tripNum);
    return true;
}
```

2.3.7 Класс DatabaseConnection

Данный класс содержит следующие атрибуты:

```
private static final String databaseName = "demo_db";
private static final String databaseUser = "root";
private static final String databasePassword = "167457";
private static final String url = "jdbc:mysql://localhost/" + databaseName;
private static Connection databaseLink;
private static DatabaseConnection databaseConnection = null;
```

Конструктор для определения статического поля databaseLink, сформированного из других определенных выше строк и встроенного метода getConnetion(attributes…)

```
private DatabaseConnection() {

    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        databaseLink = DriverManager.getConnection(url, databaseUser, databasePassword);
    } catch (Exception e) {
        e.printStackTrace();
        e.getCause();
    }
}
```

Также здесь был реализован паттерн Одиночка (Singleton). Порождающий паттерн, который гарантирует, что для определенного класса будет создан только один объект, а также предоставит к этому объекту точку доступа.

```
public static DatabaseConnection getInstance()
{
    if (databaseConnection == null)
        databaseConnection = new DatabaseConnection();
    return databaseConnection;
}

public static Connection getConnection() { return databaseLink; }
```

Часть 3. Создание форм

Чтобы приложение выглядело хорошо и привлекало пользователей своим видом, необходимо реализовать красивый пользовательский интерфейс с интуитивно понятным расположением всех элементов.

С этой целью дополнительная работа проводилась в среде разработки SceneBuilder. Программа позволяет настроить практически для всех элементов стили, расположение, ключевые связи между формой и кодом.

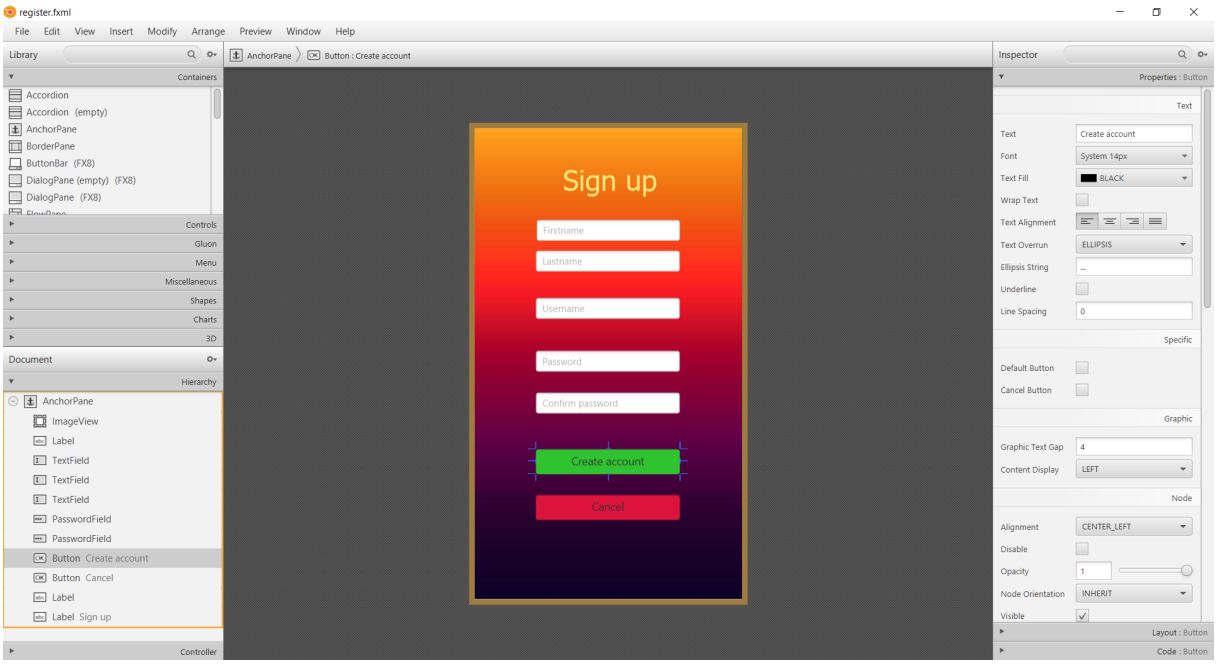

Рисунок 3 – Рабочее окно программы SceneBuilder

Сторонняя библиотека iconli включает в себя иконки font-awesome позволяет редактору формы ее украшать и тем самым проявлять интерес у пользователя. Вариабельность конфигурации элемента может влиять на такие параметры на форме как размер, поворот, заливка и тп.
Также благодаря поддержке CSS со стороны JavaFX элементам можно задавать характеристики не только в обычном положении, но и при наведении курсора, нажатии на кнопку и др.

Часть 4. Демо программы

Для изменения внешнего вида программы использовался язык описания CSS.

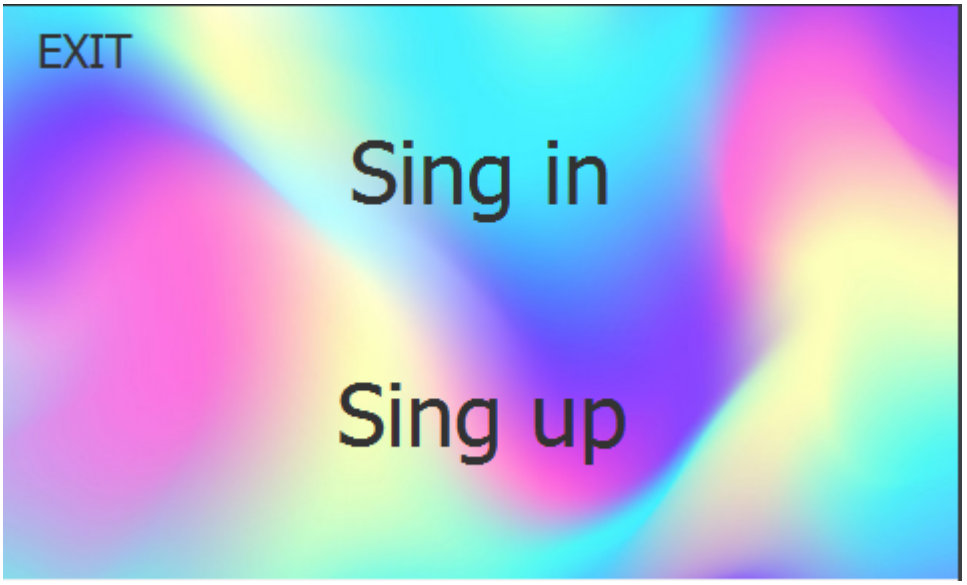

Рисунок 4 - Окно логирования/регистрации

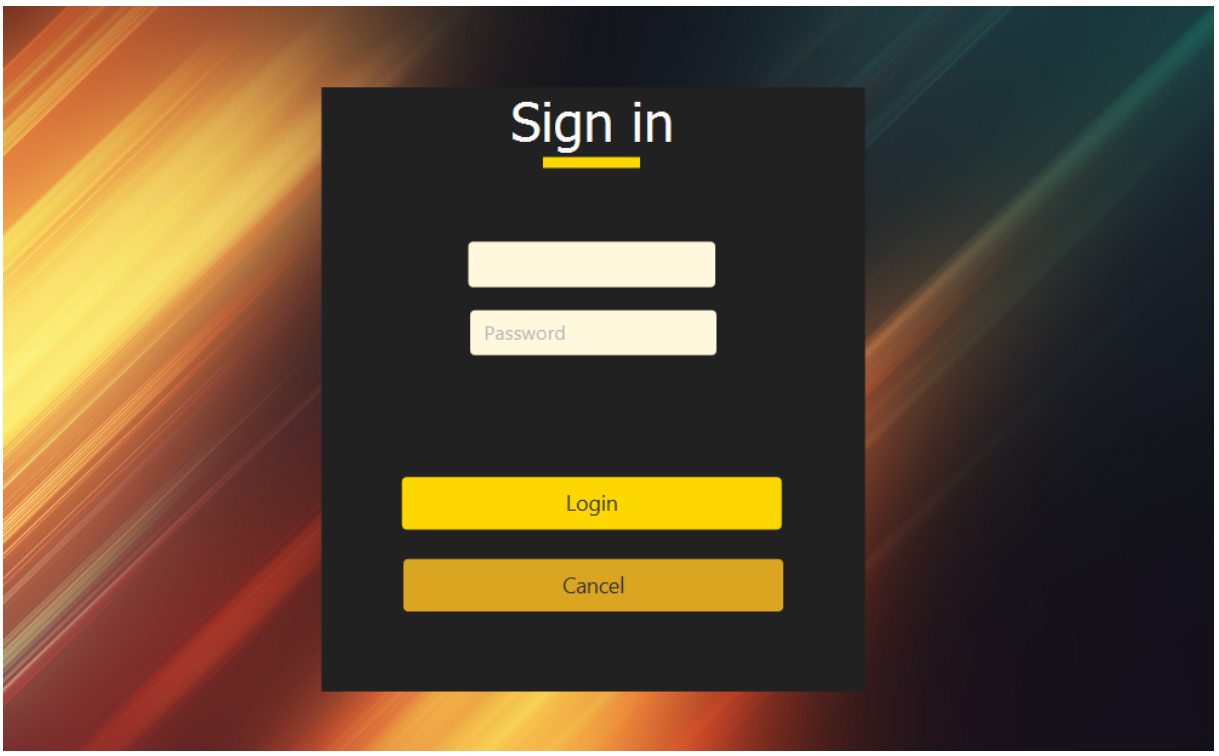

Рисунок 5 – Окно логирования

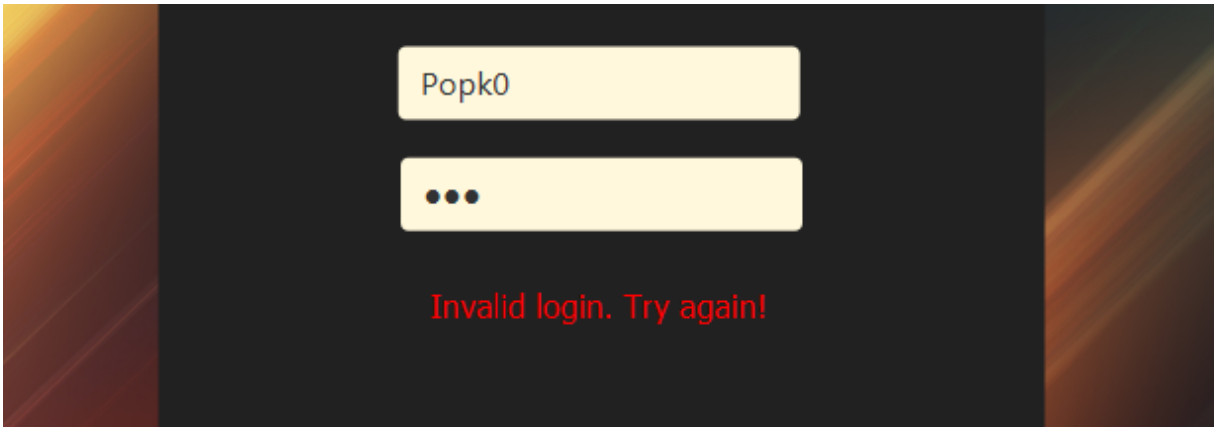

Рисунок 6 – Ошибка при входе в аккаунт

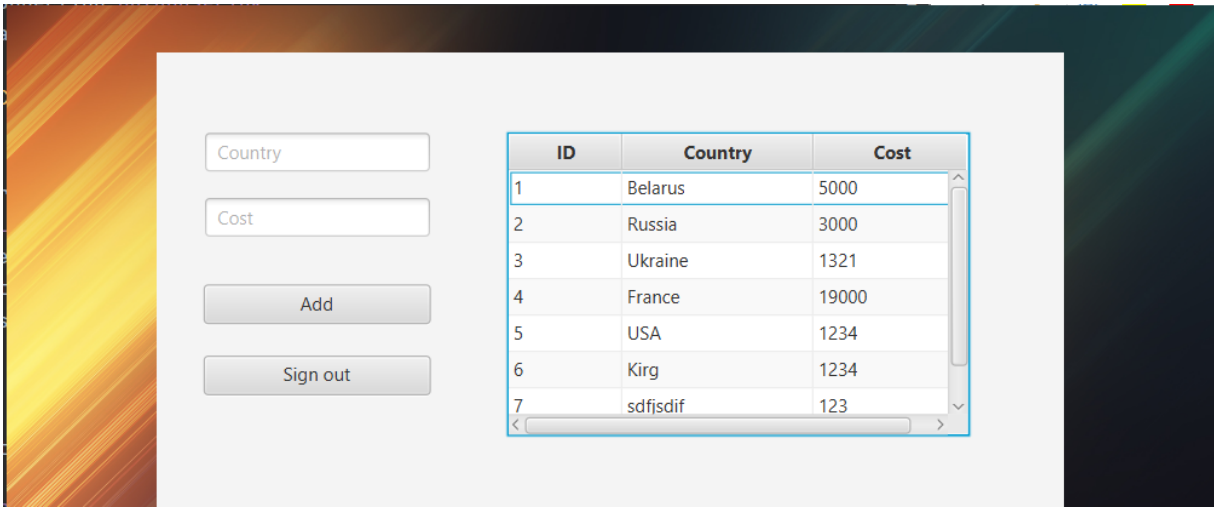

Рисунок 7 – Окно при входе в роли admin

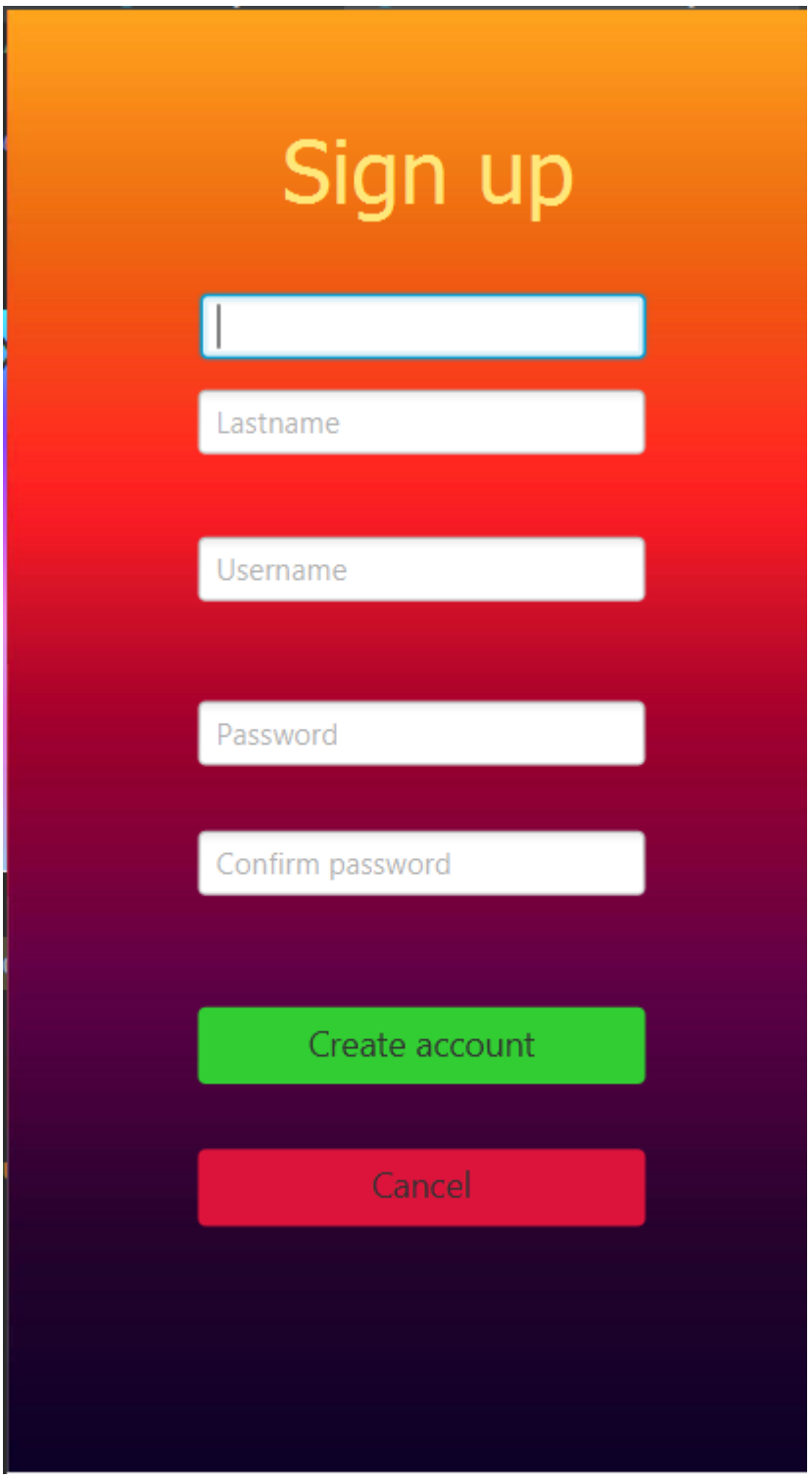

Рисунок 8 – Окно регистрации

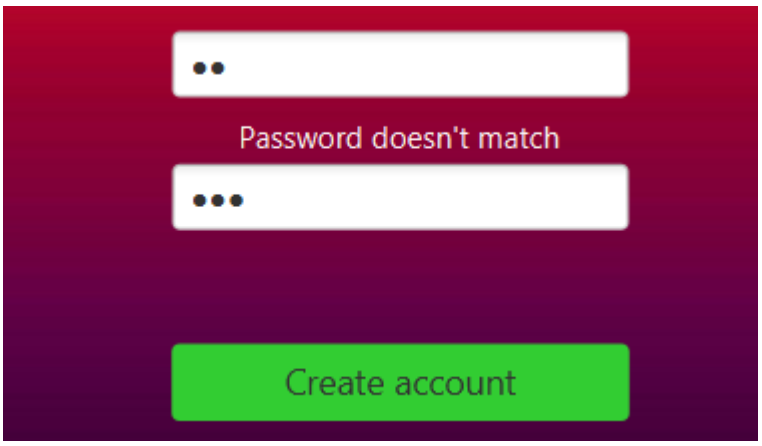

Рисунок 9 – Несовпадение паролей при регистрации

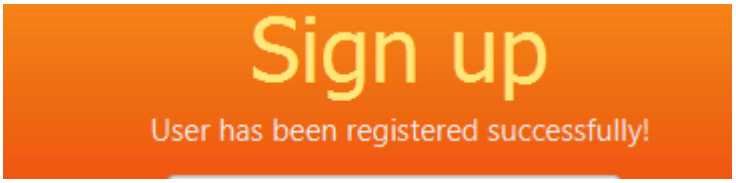

Рисунок 10 – Успешная регистрация пользователя

Заключение

В результате работы над курсовым проектом было разработано программное средство для отслеживания и, в случае возникновения необходимости, редактирования, добавления данных таблиц.

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

Реализована информационная система с удобным и информативным интерфейсом для обработки информации.

Толковая архитектура позволяет новому, не столь успешно освоившим компоненты программы пользователю, без труда осваивать и проводить работы с базой данных.

При желании улучшения функционала программного средства с легкость можно добавить новые расширения, результативные функции, редактировать или сделать нововведения в таблицы баз данных, для более точного представления пользователю. Что касается интерфейса, а именно цветовой гаммы, были использованы различные речки градиентов, что создавала приятное впечатление, отчего рекомендуется отталкиваться от схожих цветовых и форменных дополнений программного средства.

Список использованных источников

1) Java. Методы программирования – И. Н. Блинов, В. С. Романчик
2) openjfx.io – JavaFX
3) stackoverflow.com – Stack Overflow - Where Developers Learn, Share, & Build Careers
4) cyberforum.ru – Форум программистов и сисадминов Киберфорум
5) coderoad.ru – Вопросы – CodeRoad
6) codeproject.com – CodeProject - For those who code

Приложение 1. Исходный код

```
package sample;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.stage.StageStyle;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        DatabaseConnection connectNow = DatabaseConnection.getInstance();
        Parent root = FXMLLoader.load(getClass().getResource("entry.fxml"));
        primaryStage.initStyle(StageStyle.UNDECORATED);
        primaryStage.setScene(new Scene(root, 381, 229));
        primaryStage.show();
    }

    @Override
    public void stop() throws Exception {

    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

```
package sample;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;
import javafx.stage.StageStyle;

import java.io.File;
import java.net.URL;
import java.util.ResourceBundle;

public class EntryController implements Initializable {

    @FXML
    private ImageView gradient3ImageView;

    @FXML
    private Button exitButton;

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        File gradientFile = new File("images/gradient3.jpg");
        Image gradientImage = new Image(gradientFile.toURI().toString());
        gradient3ImageView.setImage(gradientImage);
    }

    public void singInButtonOnAction(ActionEvent event)
    {
```

```
        try{
            Parent root = FXMLLoader.load(getClass().getResource("tlogin.fxml"));
            Stage loginStage = new Stage();
            loginStage.initStyle(StageStyle.UNDECORATED);
            loginStage.setScene(new Scene(root, 799, 494));
            loginStage.show();

        }catch (Exception e) {
            e.printStackTrace();
            e.getCause();
        }
    }

    public void singUpButtonOnAction(ActionEvent event)
    {
        try{
            Parent root =
FXMLLoader.load(getClass().getResource("register.fxml"));
            Stage registerStage = new Stage();
            registerStage.initStyle(StageStyle.UNDECORATED);
            registerStage.setScene(new Scene(root, 326, 597));
            registerStage.show();

        }catch (Exception e) {
            e.printStackTrace();
            e.getCause();
        }
    }

    public void exitButtonOnAction(ActionEvent event)
    {
        try{
            Stage stage = (Stage) exitButton.getScene().getWindow();
            stage.close();
```

```
        }catch (Exception e) {
            e.printStackTrace();
            e.getCause();
        }
    }
}
```

```
package sample;

import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;
import javafx.event.ActionEvent;
import javafx.stage.StageStyle;

import java.io.File;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.ResultSetMetaData;
import java.util.ResourceBundle;

import java.net.URL;

public class LoginController implements Initializable {
```

```
    @FXML
    private Button cancelButton;
    @FXML
    private Label loginMessageLabel;
    @FXML
    private ImageView gradientImageView;
    @FXML
    private TextField usernameTextField;
    @FXML
    private PasswordField enterPasswordField;

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle)
    {
        File gradientFile = new File("images/gradient.jpg");
        Image gradientImage = new Image(gradientFile.toURI().toString());
        gradientImageView.setImage(gradientImage);
    }


    public void loginButtonOnAction(ActionEvent event)
    {
        if (usernameTextField.getText().isBlank() == false &&
enterPasswordField.getText().isBlank() == false) {
            validateLogin();
        }
        else {
            loginMessageLabel.setText("Please enter username and(or) password");
        }
    }

    public void cancelButtonOnAction(ActionEvent event)
    {
        Stage stage = (Stage) cancelButton.getScene().getWindow();
        stage.close();
```

```
    }

    public void validateLogin() {
        Connection connectDB = DatabaseConnection.getConnection();

        String verifyLogin = "SELECT count(1) FROM account WHERE
username = '" + usernameTextField.getText() + "' and password = '" +
enterPasswordField.getText() + "'";

        try {
            Statement statement = connectDB.createStatement();
            ResultSet rs = statement.executeQuery(verifyLogin);

            while(rs.next()) {
                if(rs.getInt(1) == 1) {
                    String getRoleByAccountID = "SELECT role FROM account
WHERE username = '" + usernameTextField.getText() + "'";

                    Statement st = connectDB.createStatement();
                    ResultSet rs_ = st.executeQuery(getRoleByAccountID);

                    rs_.next();
                    if (rs_.getString("role").equals("admin")) {
                        admin();

                    } else {
                        //user();
                    }
                }else {
                    loginMessageLabel.setText("Invalid login. Try again!");
                }
            }

        } catch (Exception e) {
            e.printStackTrace();
```

```
            e.getCause();
        }
    }

    public void admin() {
        try{
            Parent root = FXMLLoader.load(getClass().getResource("trip.fxml"));
            Stage tripStage = new Stage();
            tripStage.initStyle(StageStyle.UNDECORATED);
            tripStage.setScene(new Scene(root, 600, 400));
            tripStage.show();

        }catch (Exception e) {
            e.printStackTrace();
            e.getCause();
        }
    }
}
```

==========

```
package sample;

import javafx.application.Platform;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.control.PasswordField;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
```

```
import javafx.stage.Stage;
import javafx.event.ActionEvent;
import javafx.stage.StageStyle;

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


public class RegisterController implements Initializable {

    @FXML
    private ImageView gradient2ImageView;

    @FXML
    private Button closeButton;

    @FXML
    private Label registrationMessageLabel;

    @FXML
    private PasswordField setPasswordField;

    @FXML
    private PasswordField confirmPasswordField;

    @FXML
    private Label confirmPasswordLabel;

    @FXML
    private TextField firstnameTextField;

    @FXML
```

```
private TextField lastnameTextField;

@FXML
private TextField usernameTextField;


public void initialize(URL url, ResourceBundle resourceBundle)
{
   File gradient2File = new File("images/gradient2.jpg");
   Image gradient2Image = new Image(gradient2File.toURI().toString());
   gradient2ImageView.setImage(gradient2Image);
}

public void closeButtonOnAction (ActionEvent event) {
   Stage stage = (Stage) closeButton.getScene().getWindow();
   stage.close();
   //Platform.exit();
}

public void registerButtonOnAction (ActionEvent event) {
   if (setPasswordField.getText().equals(confirmPasswordField.getText())) {
      registerUser();
      confirmPasswordLabel.setText("");
   } else {
      confirmPasswordLabel.setText("Password doesn't match");
   }
}

public void registerUser() {
   Connection connectDB = DatabaseConnection.getConnection();

   String firstname = firstnameTextField.getText();
   String lastname = lastnameTextField.getText();
   String username = usernameTextField.getText();
   String password = setPasswordField.getText();
```

```
        String insertFields = "INSERT INTO account (lastname, firstname,
username, password, role) VALUES ('";
        String insertValues = lastname + "','" + firstname + "','" + username + "','" +
password + "', 'user')";
        String insertToRegister = insertFields + insertValues;

        try {
            Statement statement = connectDB.createStatement();
            statement.executeUpdate(insertToRegister);

            registrationMessageLabel.setText("User has been registered
successfully!");

        } catch(Exception e) {
            e.printStackTrace();
            e.getCause();
        }
}}
```

==========

==========

```
package sample;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.image.Image;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
```

```
import java.io.File;
import java.net.URL;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ResourceBundle;

public class TripController implements Initializable {

    @FXML
    private TableView<Trip> tripTableView;

    @FXML
    private TableColumn<Trip, Integer> tripIDColumn;

    @FXML
    private TableColumn<Trip, String> tripCountryColumn;

    @FXML
    private TableColumn<Trip, Integer> tripCostColumn;

    @FXML
    private TextField countryTextField;

    @FXML
    private TextField costTextField;

    @FXML
    private Label addedTripLabel;

    @FXML
    private Button signOutButton;
```

```
    @Override
    public void initialize(URL url, ResourceBundle resourceBundle)
    {
        tripIDColumn.setCellValueFactory(new
PropertyValueFactory<Trip,Integer>("tripID"));
        tripCountryColumn.setCellValueFactory(new
PropertyValueFactory<Trip,String>("country"));
        tripCostColumn.setCellValueFactory(new
PropertyValueFactory<Trip,Integer>("cost"));

        ObservableList<Trip> listM = getDataTrips();
        tripTableView.setItems(listM);
    }

    public void addTripButtonOnAction (ActionEvent event) {
        if (countryTextField.getText().isBlank() == false &&
costTextField.getText().isBlank() == false) {
            addTrip();
        }
        else {
            addedTripLabel.setText("Please enter country and(or) cost");
        }
    }

    public void signOutButtonOnAction (ActionEvent event) {
        Stage stage = (Stage) signOutButton.getScene().getWindow();
        stage.close();
    }

    public void addTrip(){

        Connection connectDB = DatabaseConnection.getConnection();

        String country = countryTextField.getText();
        String cost = costTextField.getText();
```

```
        String insertFields = "INSERT INTO trip (country, cost) VALUES ('";
        String insertValues = country + "','" + cost + "')";
        String insertToRegister = insertFields + insertValues;

        try {
            Statement statement = connectDB.createStatement();
            statement.executeUpdate(insertToRegister);

            addedTripLabel.setText("Trip has been added successfully!");

            ObservableList<Trip> listM = getDataTrips();
            tripTableView.setItems(listM);

        } catch(Exception e) {
            e.printStackTrace();
            e.getCause();
        }
    }

    public static ObservableList<Trip> getDataTrips() {
        Connection connectDB = DatabaseConnection.getConnection();

        ObservableList<Trip> list = FXCollections.observableArrayList();

        String getAllTrips = ("SELECT * FROM trip");

        try {
            Statement statement = connectDB.createStatement();
            ResultSet rs = statement.executeQuery(getAllTrips);

            while (rs.next()) {
                list.add(new Trip(Integer.parseInt(rs.getString("trip_id")),
rs.getString("country"), rs.getInt("cost")));
            }
```

```
        } catch (Exception e) {
            e.printStackTrace();
            e.getCause();
        }

        return list;
    }
}
```

==========

==========

```
package sample;

import java.sql.Connection;
import java.sql.DriverManager;

public class DatabaseConnection {

    private static final String databaseName = "demo_db";
    private static final String databaseUser = "root";
    private static final String databasePassword = "167457";
    private static final String url = "jdbc:mysql://localhost/" + databaseName;
    private static Connection databaseLink;
    private static DatabaseConnection databaseConnection = null;

    private DatabaseConnection() {

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            databaseLink = DriverManager.getConnection(url, databaseUser,
databasePassword);
        } catch (Exception e) {
```

```
            e.printStackTrace();
            e.getCause();
        }
    }

    public static DatabaseConnection getInstance()
    {
        if (databaseConnection == null)
            databaseConnection = new DatabaseConnection();
        return databaseConnection;
    }

    public static Connection getConnection() {
        return databaseLink;
    }
}
```

==========

==========

```
package sample;

import com.mysql.cj.conf.IntegerProperty;
import com.mysql.cj.conf.StringProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;

public class Trip {

    private SimpleIntegerProperty tripID;
    private SimpleStringProperty country;
    private SimpleIntegerProperty cost;

    public Trip(int tripID, String country, int cost) {
        this.tripID = new SimpleIntegerProperty(tripID);
        this.country = new SimpleStringProperty(country);
```

```
        this.cost = new SimpleIntegerProperty(cost);
    }
    public int getTripID() {
        return tripID.get();
    }
    public SimpleIntegerProperty tripIDProperty() {
        return tripID;
    }
    public String getCountry() {
        return country.get();
    }
    public SimpleStringProperty countryProperty() {
        return country;
    }
    public int getCost() {
        return cost.get();
    }
    public SimpleIntegerProperty costProperty() {
        return cost;
    }
    public void setTripID(int tripID) {
        this.tripID.set(tripID);
    }
    public void setCountry(String country) {
        this.country.set(country);
    }
    public void setCost(int cost) {
        this.cost.set(cost);
    }
}
```