OOP Project – Trip Planner

Andrej Stuchlý

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

2023

Introduction

This project is a trip planner app designed to assist users in searching for landmarks and events in various cities and countries. Users can add selected places to their trip and associate them with their profiles. The app also calculates the total price of all the selected places to visit.

One noteworthy feature of this project is that users can perform all tasks directly within the app, eliminating the need to access the database separately. Users can log in, sign up, add or remove places from their profiles, and even insert new landmarks and events into the database if logged in as an admin.

The primary objective of this project was to gain more practice in implementing Object-Oriented Programming (OOP) principles and coding in Java. Additionally, I aimed to acquire experience working with databases, as it was a new area for me. Creating the graphical user interface (GUI) was also a valuable learning experience, even though my primary focus lies in backend development.

Logic of my app

First and foremost, the user is required to either log in or sign up. Upon entering the username and password, the application sends a request to the database. The login functionality verifies if the entered username and password match an existing record, while the sign-up process checks if username is available, and the two passwords provided are identical before adding a new user to the database.

Once successfully logged in, the user is directed to the main screen of the project. This screen displays tables containing events and landmarks, along with a search bar. Upon entering a destination and clicking a button, a request is sent to the database to filter places of interest specific to that destination. The filtered results are then displayed in their respective tables.

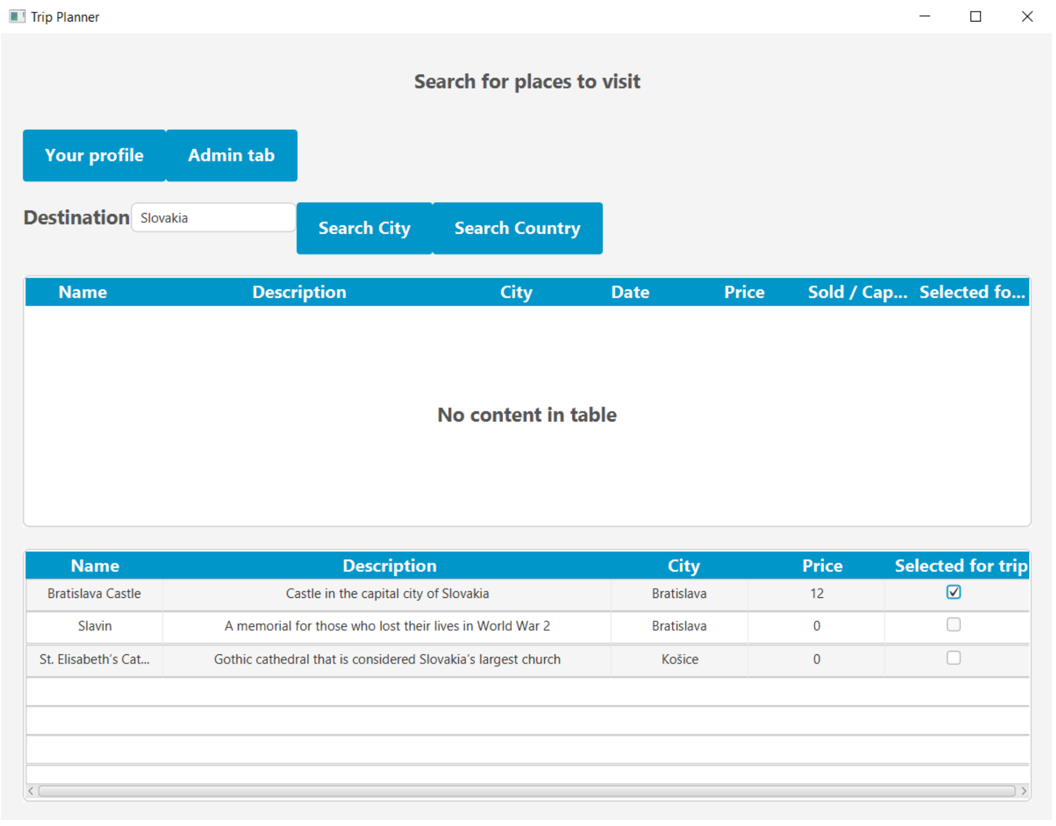
To add a place to their trip, the user can simply select the checkbox associated with the desired place. This action adds the place to an array list of places to visit, increases the total trip price, and sends an update request to the database, associating the place's ID with the user's profile. Removing a place from the trip follows a similar process: unchecking the checkbox removes the place from the array list, decreases the total trip price, and sends an update request to the database.

As an admin user, I have the privilege of inserting new landmarks and events into the database. The application provides radio buttons to choose between the two options. Upon selection, I can enter the necessary data and send an insert request to the appropriate table in the database.

Thanks to this logic, the application retains all changes even after being shut down and reopened, ensuring data persistence.

App GUI

This screenshot showcases the main tab of the application. The GUI was developed using JavaFX, and I opted for a colour scheme that features fresh and bright tones, including white, light grey, and light blue. User feedback indicates that navigation is intuitive, with a clear presentation and the absence of unnecessary elements.



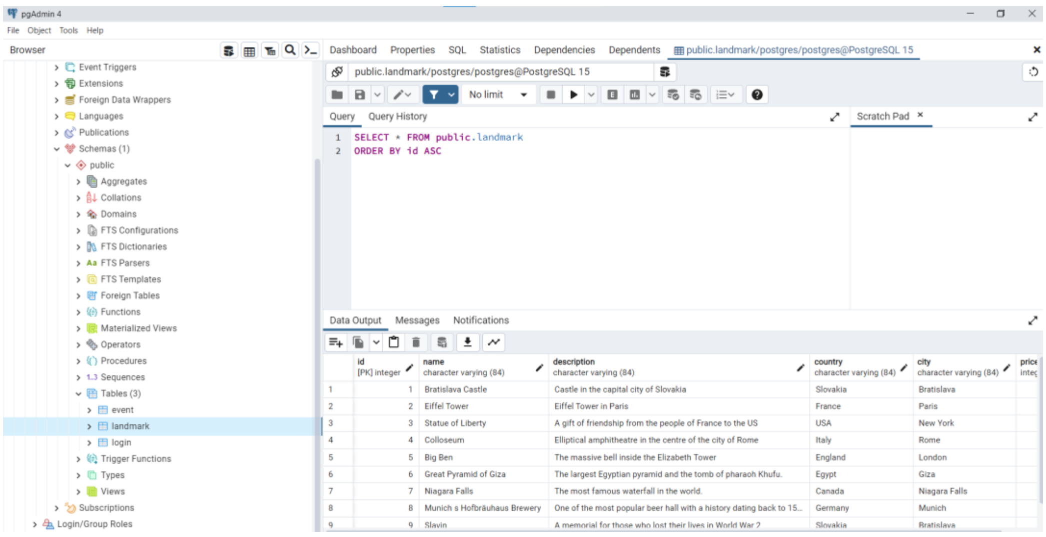
Name	Description	City	Date	Price	Sold / Cap...	Selected fo...

No content in table

Name	Description	City	Price	Selected for trip
Bratislava Castle	Castle in the capital city of Slovakia	Bratislava	12	☑
Slavin	A memorial for those who lost their lives in World War 2	Bratislava	0	☐
St. Elisabeth's Cat...	Gothic cathedral that is considered Slovakia's largest church	Košice	0	☐

Database

I have made the decision to store my data in a PostgreSQL database. I specifically chose to work with this particular database because it is suitable for this type of project and also provides an opportunity for me to learn how to work with it effectively.



```
SELECT * FROM public.landmark
ORDER BY id ASC
```

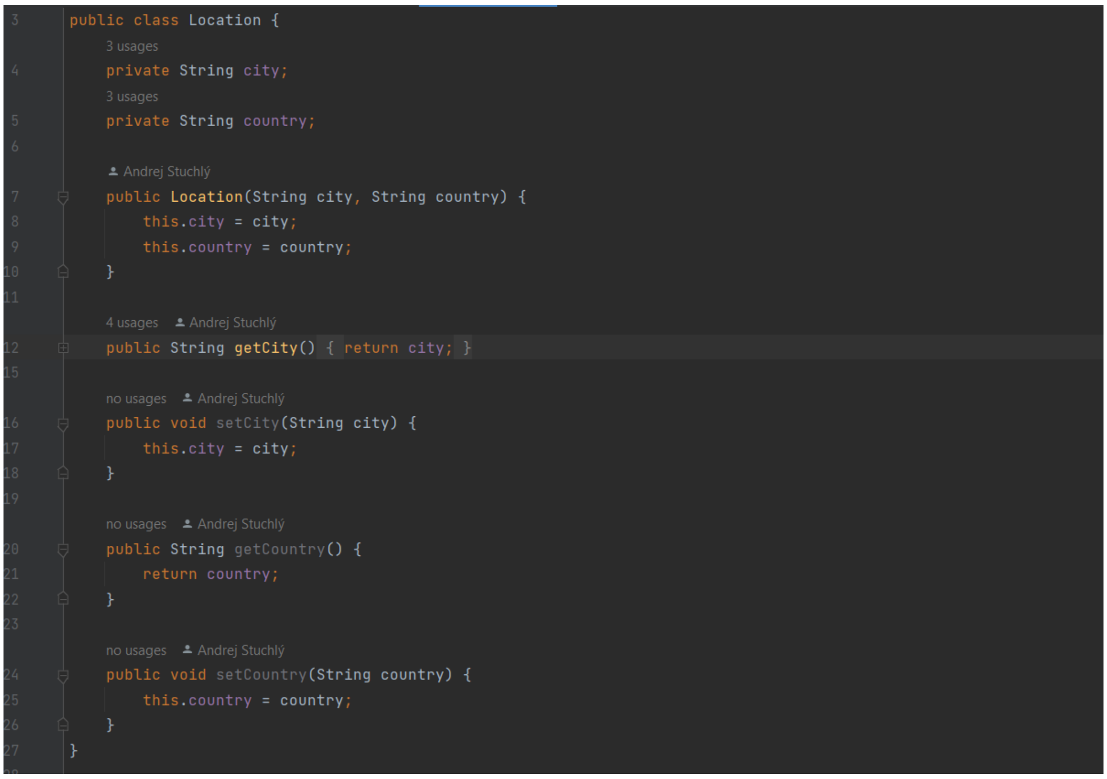
	id [PK] integer	name character varying (84)	description character varying (84)	country character varying (84)	city character varying (84)	price integ
1	1	Bratislava Castle	Castle in the capital city of Slovakia	Slovakia	Bratislava	
2	2	Eiffel Tower	Eiffel Tower in Paris	France	Paris	
3	3	Statue of Liberty	A gift of friendship from the people of France to the US	USA	New York	
4	4	Colloseum	Elliptical amphitheatre in the centre of the city of Rome	Italy	Rome	
5	5	Big Ben	The massive bell inside the Elizabeth Tower	England	London	
6	6	Great Pyramid of Giza	The largest Egyptian pyramid and the tomb of pharaoh Khufu.	Egypt	Giza	
7	7	Niagara Falls	The most famous waterfall in the world.	Canada	Niagara Falls	
8	8	Munich s Hofbräuhaus Brewery	One of the most popular beer hall with a history dating back to 15...	Germany	Munich	
9	9	Slavin	A memorial for those who lost their lives in World War 2	Slovakia	Bratislava	

OOP principles

There are several OOP principles implemented in my project:

Encapsulation

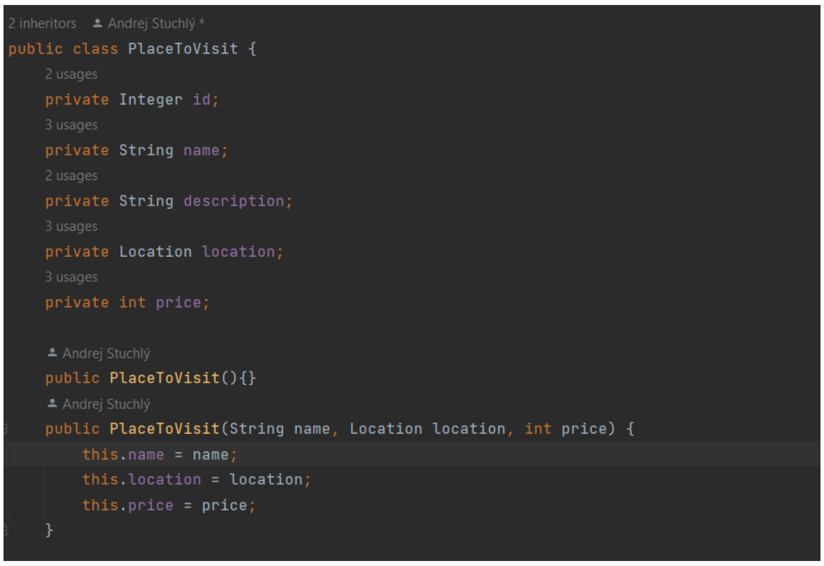
Encapsulation is demonstrated in my project through the encapsulation of attributes and methods within classes. For example, the Location class exhibits encapsulation.



```
public class Location {
    3 usages
    private String city;
    3 usages
    private String country;

    Andrej Stuchlý
    public Location(String city, String country) {
        this.city = city;
        this.country = country;
    }

    4 usages   Andrej Stuchlý
    public String getCity() { return city; }

    no usages   Andrej Stuchlý
    public void setCity(String city) {
        this.city = city;
    }

    no usages   Andrej Stuchlý
    public String getCountry() {
        return country;
    }

    no usages   Andrej Stuchlý
    public void setCountry(String country) {
        this.country = country;
    }
}
```

Aggregation

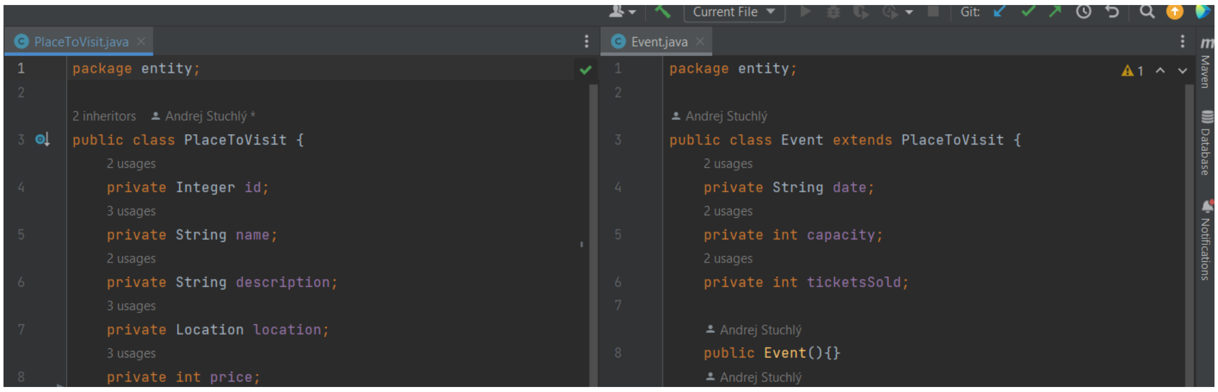
Aggregation is utilized in the project through the presence of the location attribute in the PlaceToVisit class, where the Location class is referenced.



```
2 inheritors   Andrej Stuchlý *
public class PlaceToVisit {
    2 usages
    private Integer id;
    3 usages
    private String name;
    2 usages
    private String description;
    3 usages
    private Location location;
    3 usages
    private int price;

    Andrej Stuchlý
    public PlaceToVisit(){}
    Andrej Stuchlý
    public PlaceToVisit(String name, Location location, int price) {
        this.name = name;
        this.location = location;
        this.price = price;
    }
```

Inheritance

Inheritance is applied in the project, with the PlaceToVisit class serving as the main class for a place, and two classes, Event and Landmark, extending it. The subclasses inherit parameters and methods from the parent class.



PlaceToVisit.java

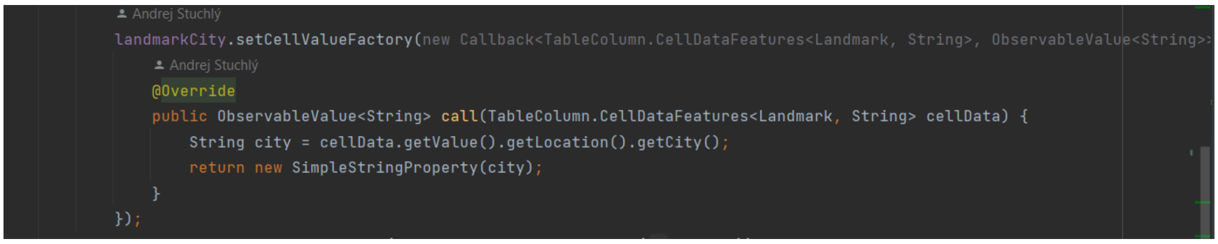
```java
package entity;

2 inheritors  Andrej Stuchlý *
public class PlaceToVisit {
    2 usages
    private Integer id;
    3 usages
    private String name;
    2 usages
    private String description;
    3 usages
    private Location location;
    3 usages
    private int price;
```

Event.java

```java
package entity;

Andrej Stuchlý
public class Event extends PlaceToVisit {
    2 usages
    private String date;
    2 usages
    private int capacity;
    2 usages
    private int ticketsSold;

    Andrej Stuchlý
    public Event(){}
    Andrej Stuchlý
```

Overriding

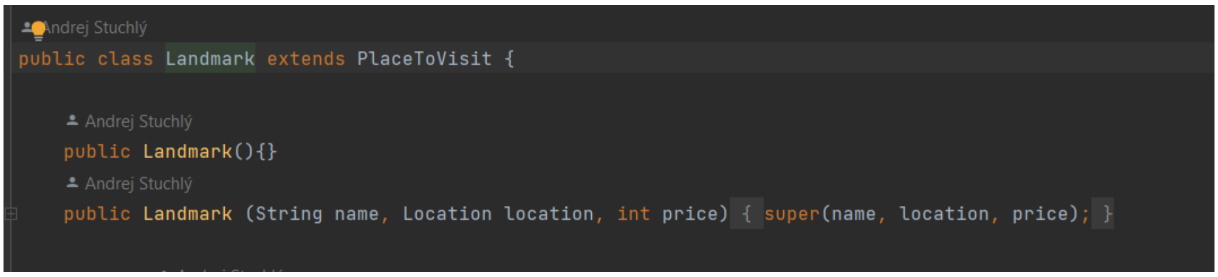
Method overriding is employed in the project to modify the functionality of subclass methods. This approach is implemented in my GUI controllers to make them more suitable for the specific requirements of the project.



```java
Andrej Stuchlý
landmarkCity.setCellValueFactory(new Callback<TableColumn.CellDataFeatures<Landmark, String>, ObservableValue<String>>
    Andrej Stuchlý
    @Override
    public ObservableValue<String> call(TableColumn.CellDataFeatures<Landmark, String> cellData) {
        String city = cellData.getValue().getLocation().getCity();
        return new SimpleStringProperty(city);
    }
});
```

Overloading

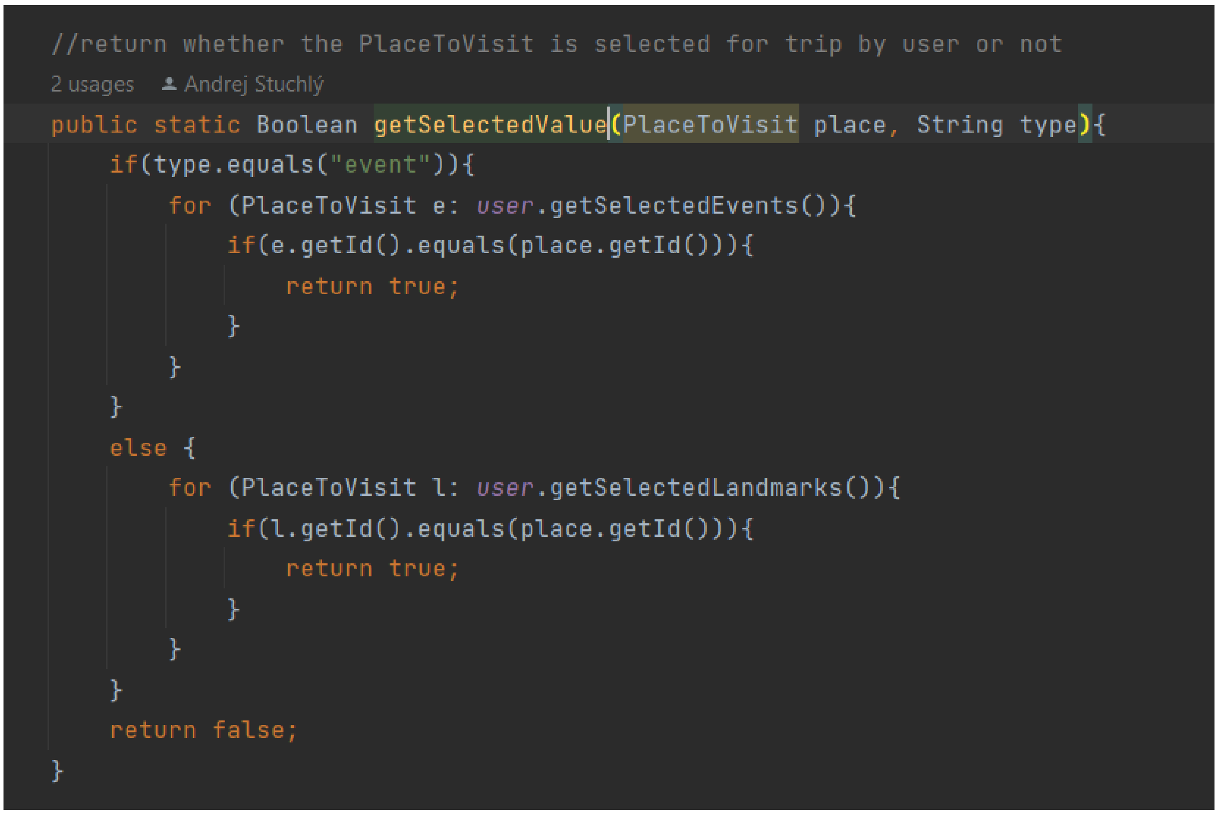
Method overloading is utilized in the project, particularly in the constructors. Different versions of the same method with varying parameters are defined to perform different tasks. The Landmark class showcases the usage of method overloading.



```java
Andrej Stuchlý
public class Landmark extends PlaceToVisit {

    Andrej Stuchlý
    public Landmark(){}
    Andrej Stuchlý
    public Landmark (String name, Location location, int price) { super(name, location, price); }
```

Polymorphism

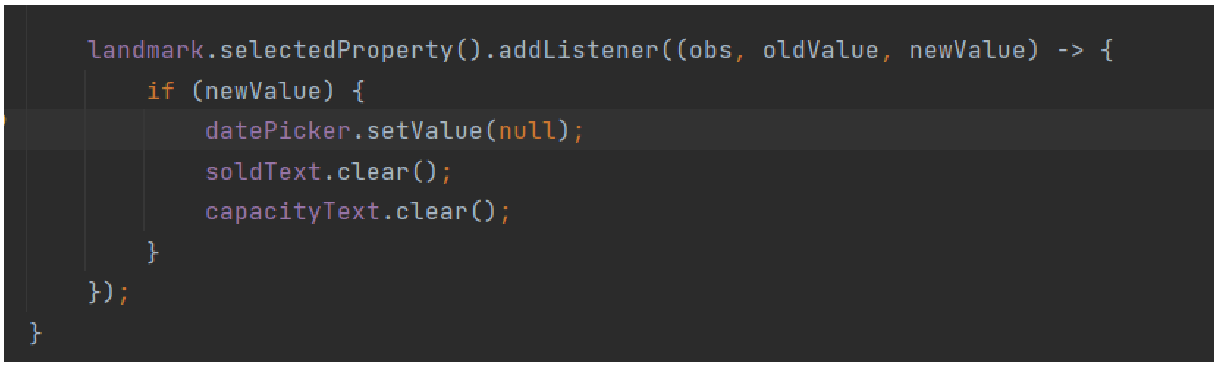
Polymorphism is incorporated into the project through several methods that accept input values of the PlaceToVisit class. The processing of the input values differs based on their subtype, whether it is a landmark or an event.



```java
//return whether the PlaceToVisit is selected for trip by user or not
2 usages  Andrej Stuchlý
public static Boolean getSelectedValue(PlaceToVisit place, String type){
    if(type.equals("event")){
        for (PlaceToVisit e: user.getSelectedEvents()){
            if(e.getId().equals(place.getId())){
                return true;
            }
        }
    }
    else {
        for (PlaceToVisit l: user.getSelectedLandmarks()){
            if(l.getId().equals(place.getId())){
                return true;
            }
        }
    }
    return false;
}
```

Lambda expression

Lambda expressions are present in my FXML controller classes. For instance, the provided example features an admin tab controller with a lambda expression that controls the selection of either a landmark or an event.



```java
    landmark.selectedProperty().addListener((obs, oldValue, newValue) -> {
        if (newValue) {
            datePicker.setValue(null);
            soldText.clear();
            capacityText.clear();
        }
    });
}
```

Packages

To ensure a clear code structure, I have decided to organize my classes into packages following the MVC (Model-View-Controller) architecture. In this approach, I have created three main packages: entity, gui, and logic.

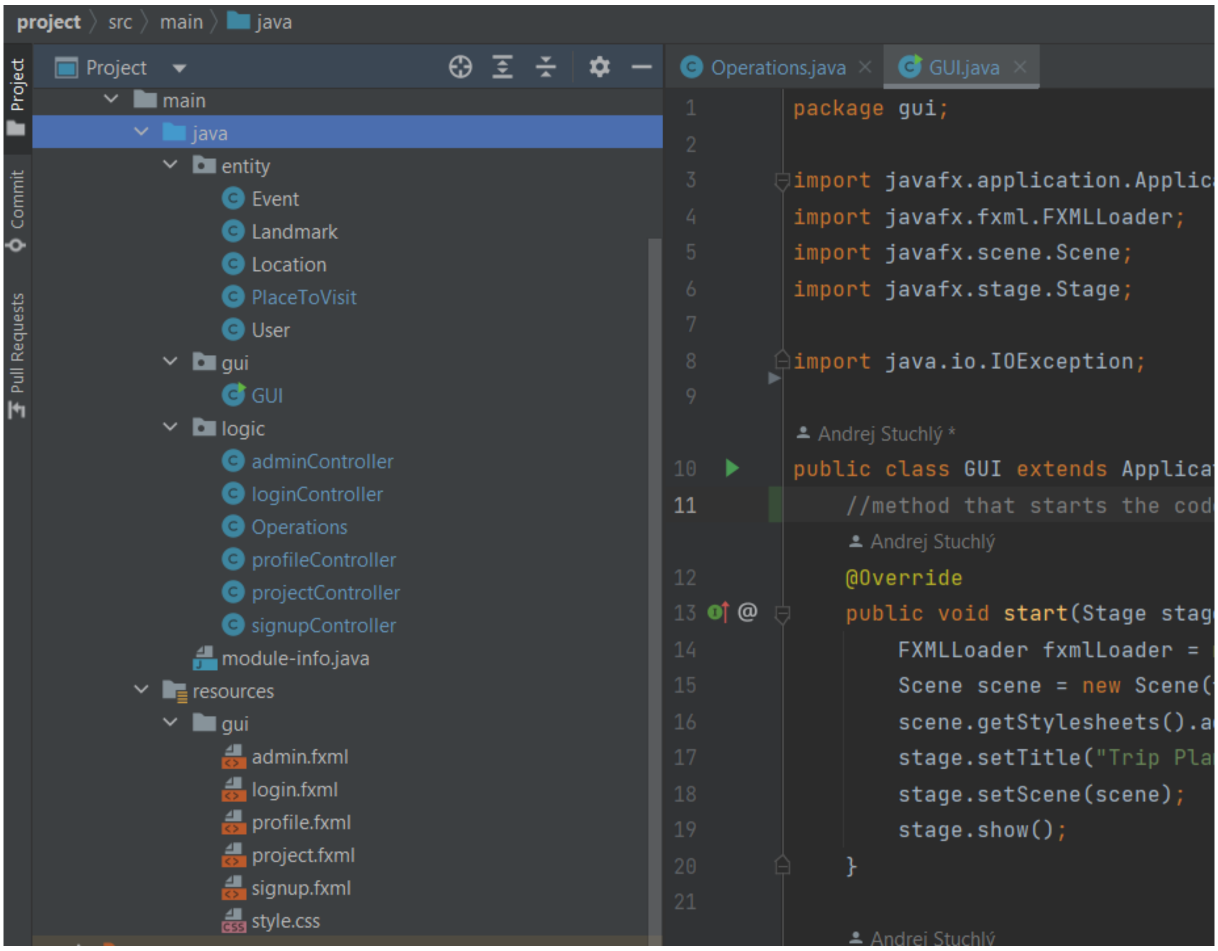
The entity package represents the model and provides data classes that encapsulate the essential data and behaviour of the application.

The gui package serves as the view part, responsible for what the user can see and interact with. It contains classes related to the graphical user interface.

The logic package acts as the controller, managing the operations and orchestrating the communication between the model and the view. This package contains classes that handle the logic and behaviour of the application.

In addition to these packages, I store the FXML and CSS files in the resources directory, separate from the codebase. This separation helps to keep the codebase clean and ensures a more organized structure.

By following this MVC architecture and organizing my classes into appropriate packages, I can maintain a clear separation of concerns and improve the readability and maintainability of my code.

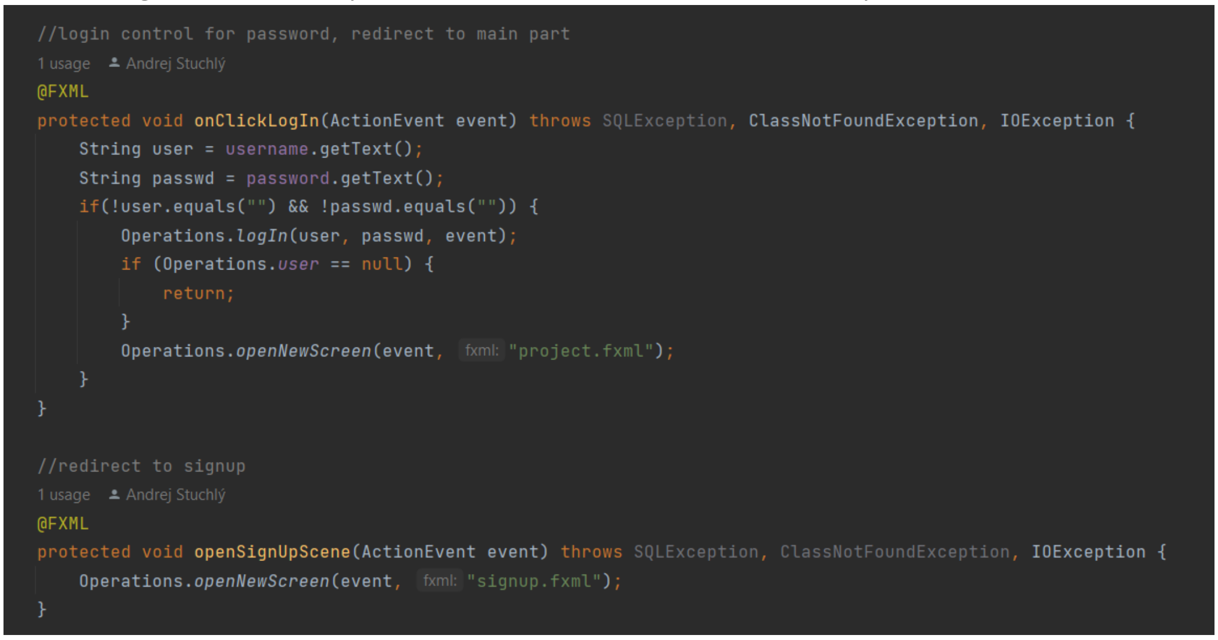


```
main
  java
    entity
      Event
      Landmark
      Location
      PlaceToVisit
      User
    gui
      GUI
    logic
      adminController
      loginController
      Operations
      profileController
      projectController
      signupController
    module-info.java
  resources
    gui
      admin.fxml
      login.fxml
      profile.fxml
      project.fxml
      signup.fxml
      style.css
```

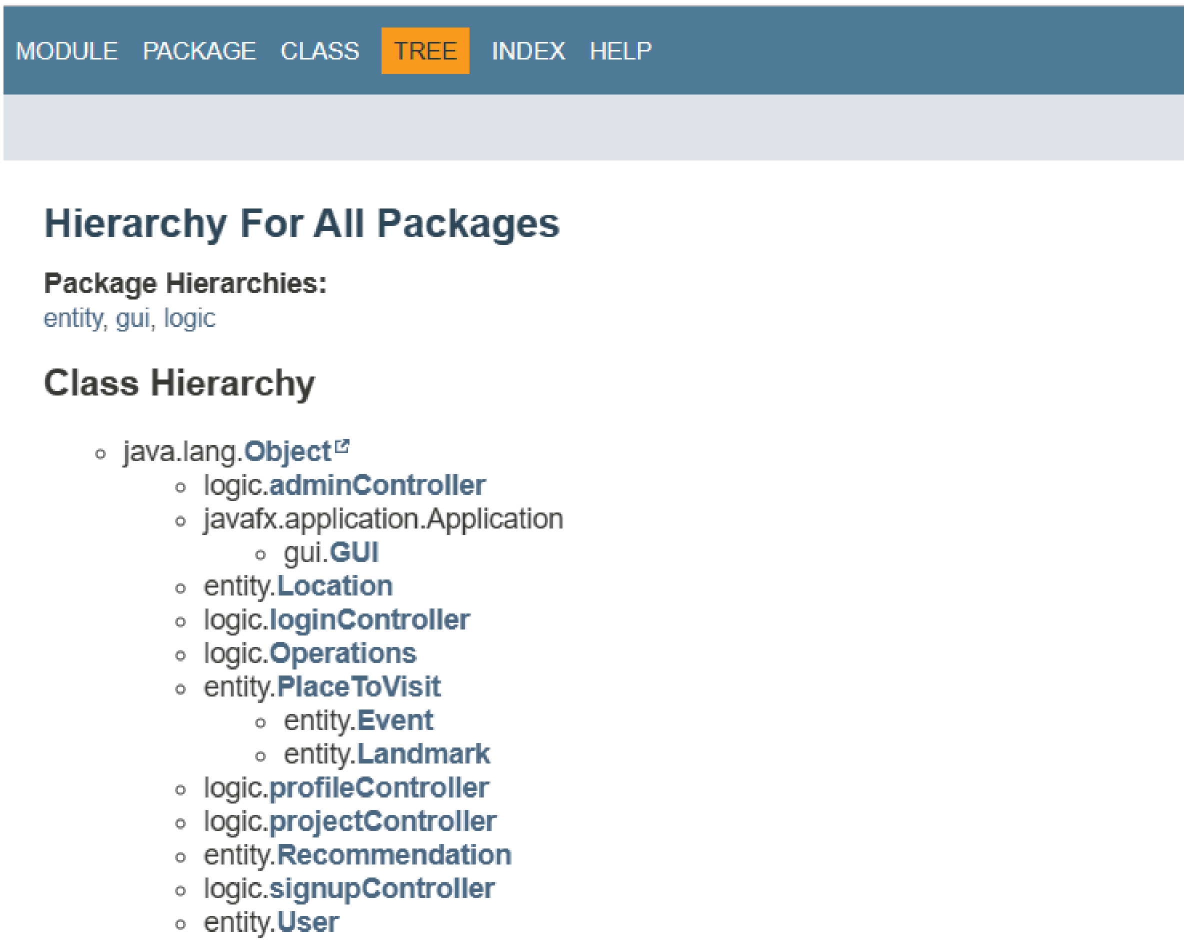
```
package gui;

import javafx.application.Applic
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

Andrej Stuchlý *
public class GUI extends Applica
    //method that starts the cod
    Andrej Stuchlý
    @Override
    public void start(Stage stag
        FXMLLoader fxmlLoader =
        Scene scene = new Scene(
        scene.getStylesheets().a
        stage.setTitle("Trip Pla
        stage.setScene(scene);
        stage.show();
    }

    Andrej Stuchlý
```

Comments

I have added comments to my classes and methods to improve code readability. Additionally, I have generated a Javadoc for the project, providing a comprehensive reference guide for developers to understand the code more easily.



```
//login control for password, redirect to main part
1 usage  Andrej Stuchlý
@FXML
protected void onClickLogIn(ActionEvent event) throws SQLException, ClassNotFoundException, IOException {
    String user = username.getText();
    String passwd = password.getText();
    if(!user.equals("") && !passwd.equals("")) {
        Operations.logIn(user, passwd, event);
        if (Operations.user == null) {
            return;
        }
        Operations.openNewScreen(event,  fxml: "project.fxml");
    }
}

//redirect to signup
1 usage  Andrej Stuchlý
@FXML
protected void openSignUpScene(ActionEvent event) throws SQLException, ClassNotFoundException, IOException {
    Operations.openNewScreen(event,  fxml: "signup.fxml");
}
```



Hierarchy For All Packages

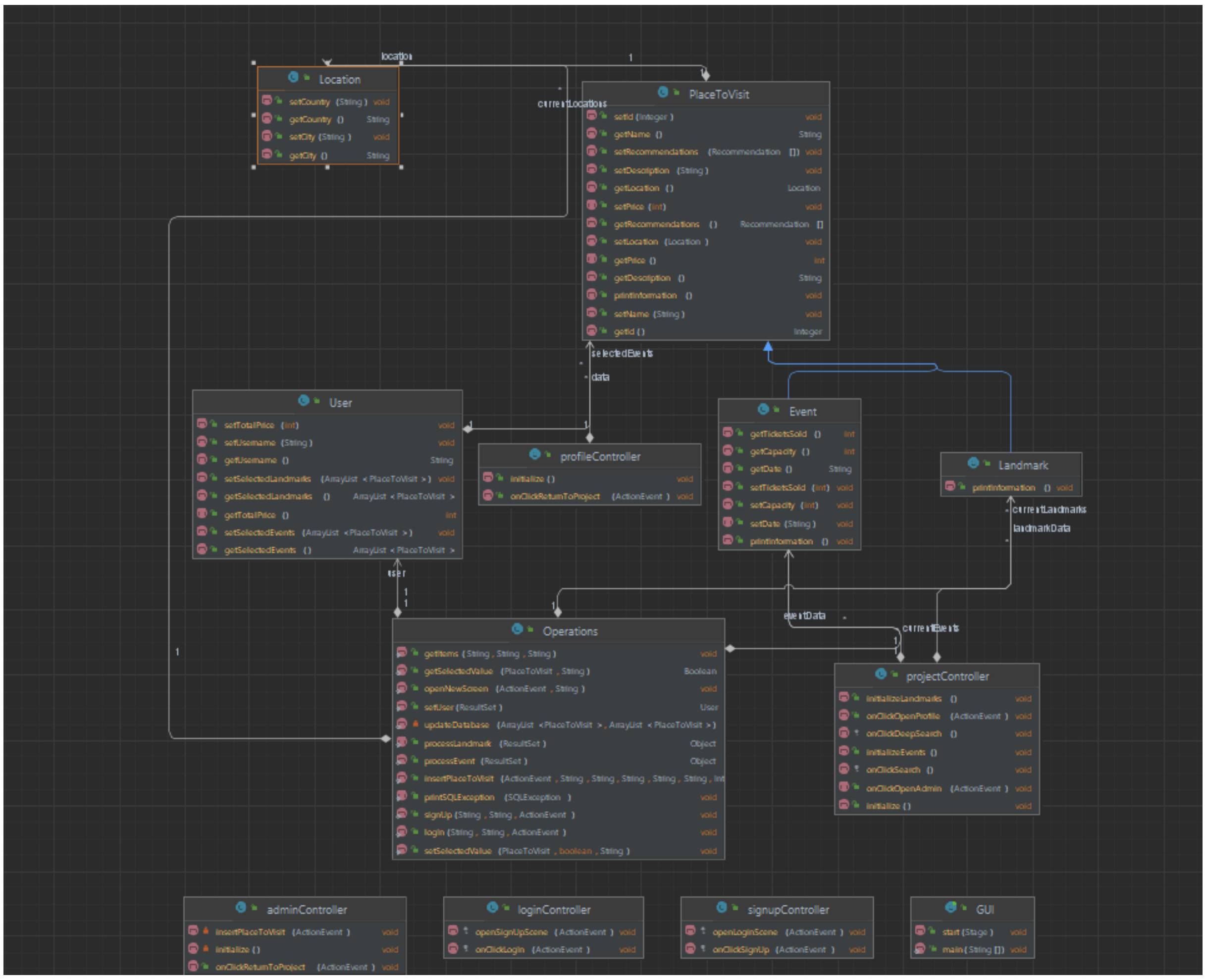
Package Hierarchies:
entity, gui, logic

Class Hierarchy

- java.lang.Object
  - logic.adminController
  - javafx.application.Application
    - gui.GUI
  - entity.Location
  - logic.loginController
  - logic.Operations
  - entity.PlaceToVisit
    - entity.Event
    - entity.Landmark
  - logic.profileController
  - logic.projectController
  - entity.Recommendation
  - logic.signupController
  - entity.User

UML diagram

Below is the UML diagram illustrating the classes and methods utilized in my project:



location
1
currentLocations
Location
setCountry (String) void
getCountry () String
setCity (String) void
getCity () String
PlaceToVisit
setId (Integer) void
getName () String
setRecommendations (Recommendation []) void
setDescription (String) void
getLocation () Location
setPrice (int) void
getRecommendations () Recommendation []
setLocation (Location) void
getPrice () int
getDescription () String
printInformation () void
setName (String) void
getId () Integer
selectedEvents
data
User
setTotalPrice (int) void
setUsername (String) void
getUsername () String
setSelectedLandmarks (ArrayList < PlaceToVisit >) void
getSelectedLandmarks () ArrayList < PlaceToVisit >
getTotalPrice () int
setSelectedEvents (ArrayList < PlaceToVisit >) void
getSelectedEvents () ArrayList < PlaceToVisit >
profileController
initialize () void
onClickReturnToProject (ActionEvent) void
Event
getTicketsSold () int
getCapacity () int
getDate () String
setTicketsSold (int) void
setCapacity (int) void
setDate (String) void
printInformation () void
Landmark
printInformation () void
currentLandmarks
landmarkData
user
eventData
currentEvents
Operations
getItems (String, String, String) void
getSelectedValue (PlaceToVisit, String) Boolean
openNewScreen (ActionEvent, String) void
setUser (ResultSet) User
updateDatabase (ArrayList <PlaceToVisit >, ArrayList < PlaceToVisit >)
processLandmark (ResultSet) Object
processEvent (ResultSet) Object
insertPlaceToVisit (ActionEvent, String, String, String, String, String, Int
printSQLException (SQLException) void
signUp (String, String, ActionEvent) void
login (String, String, ActionEvent) void
setSelectedValue (PlaceToVisit, boolean, String) void
projectController
initializeLandmarks () void
onClickOpenProfile (ActionEvent) void
onClickDeepSearch () void
initializeEvents () void
onClickSearch () void
onClickOpenAdmin (ActionEvent) void
initialize () void
adminController
insertPlaceToVisit (ActionEvent) void
initialize () void
onClickReturnToProject (ActionEvent) void
loginController
openSignUpScene (ActionEvent) void
onClickLogin (ActionEvent) void
signupController
openLoginScene (ActionEvent) void
onClickSignUp (ActionEvent) void
GUI
start (Stage) void
main (String []) void

Conclusion

Overall, the project was a great success – I managed to develop an app that has practical utility and can be utilized in real-life scenarios. This project provided me with valuable insights into working with databases, Java classes, and further deepened my understanding of essential Object-Oriented Programming (OOP) principles, enabling me to enhance my skills as a developer.