Adam Vrabeľ AIS ID: 116327

**Faculty of Informatics and Information Technologies**
**STU in Bratislava**
Ilkovičova 2, 842 16 Bratislava 4

2022/ 2023

# Object-Oriented Programming

TRAIN PLANNER

Excercise: Ing. Mohammad Daud Haiderzai
Time of exercises: 11:00

Created by: Adam Vrabeľ
AIS ID: 116327

# CONTENT

# PROJECT DESCRIPTION

The project is created in Java, with intelliJ IDEA environment using standard libraries.
I'm using JavaFX and SceneBuilder for the GUI.
I use postgreSQL 15 DATABASE for storing and working with data.

The intention of my project was to make a meaningful, clear and user-friendly application for buying and managing train tickets and train journeys. My app is designed for searching train routes and buying tickets for available train routes with focus on the frontend and GUI of application. I work with database, getting informations about logged user, his inventory, available train routes, when user or administrator change something in app, changes are saved in the database.

App allows **admin login** and **user login**.
In **administrator mode**, administrator can view train routes/ delete train routes and add new train routes.
In **user mode**, user can search for route, can choose from the available price options of tickets for train route and can buy ticket for choosen route. User has inventory where can see his bought tickets, can fill up money to his account by APPLE PAY (just simple payment) or can choose card pay (VISA) when user need to fill all correct informations of card and fill up account.

I ran into some problems while developing my project. I learned to work with JavaFX, to create and process database queries and how to work with the database in general. In my project are all comments from the teacher are implemented. I improved some functionality, I did not implement some functions due to the project's intention. Because the details of the project and the intention have partially changed. The application is fully functional, nice and clear, makes sense, works smoothly and fulfills the task.

All admin / user actions in the application are linked to the database (every change in the database will be reflected in the application and vice versa)

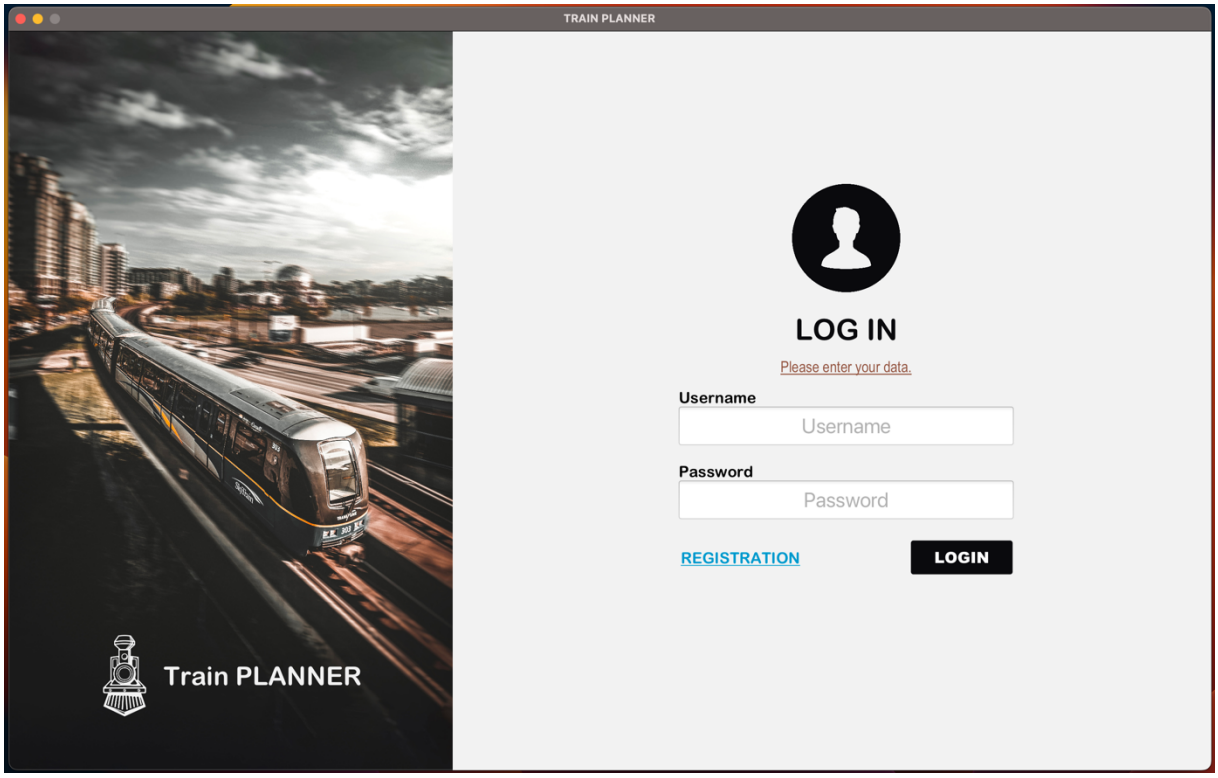
All the functions that my program has are described below.

# FUNCTIONS OF APPLICATION

- LOGIN / REGISTRATION
- LOGIN AS ADMINISTRATOR
    - show all train routes
    - delete route
    - add new train route
- LOGIN AS USER
    - search for train routes
    - choose train ticket (economy/ first class)
    - buy ticket
    - increase credit (by simple apple pay button or credit card)
    - display bought tickets

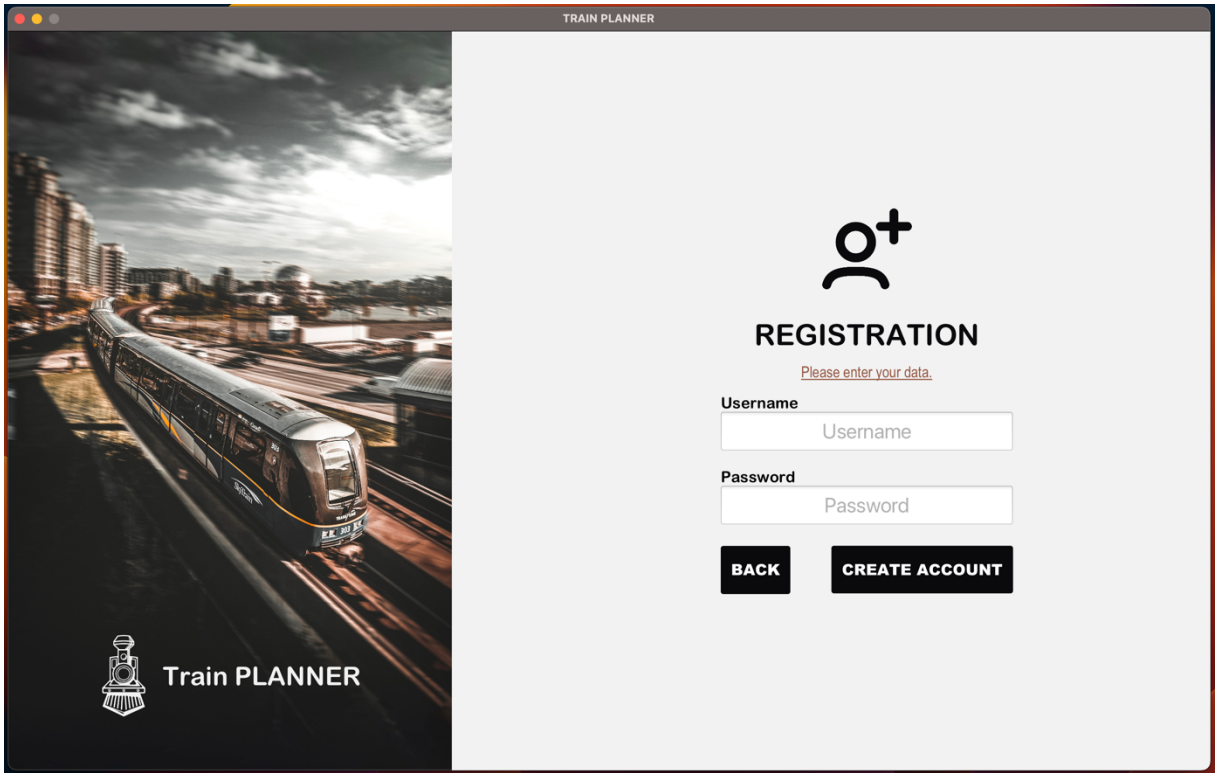
# SCREENS AND DESCRIPTION

LOGIN SCREEN



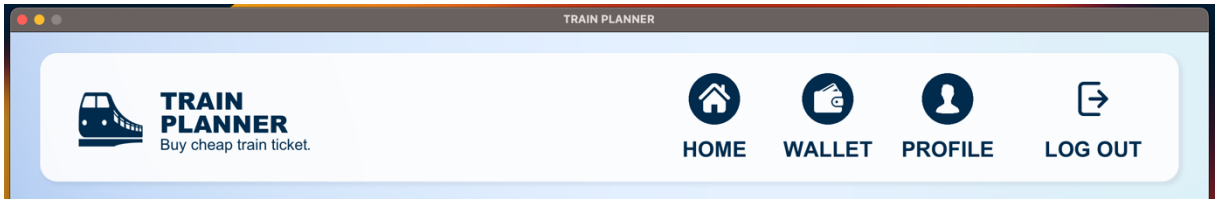

This screen is the first to appear after launching the app. Using this screen, we can log in as a user (if we enter the correct login data). The entered data and data from the database are checked. If the entered data is filled incorrectly or user with entered username and password does not exist in database, an error message will be displayed. If the entered data is correct, the user logs in to the application, all available information from the database about user load to the instance of Passenger class, and the change screen to USER SEARCH SCREEN.
We also have a registration option, when user clicked on REGISTRATION link, screen changed to REGISTRATION SCREEN .

REGISTRATION SCREEN



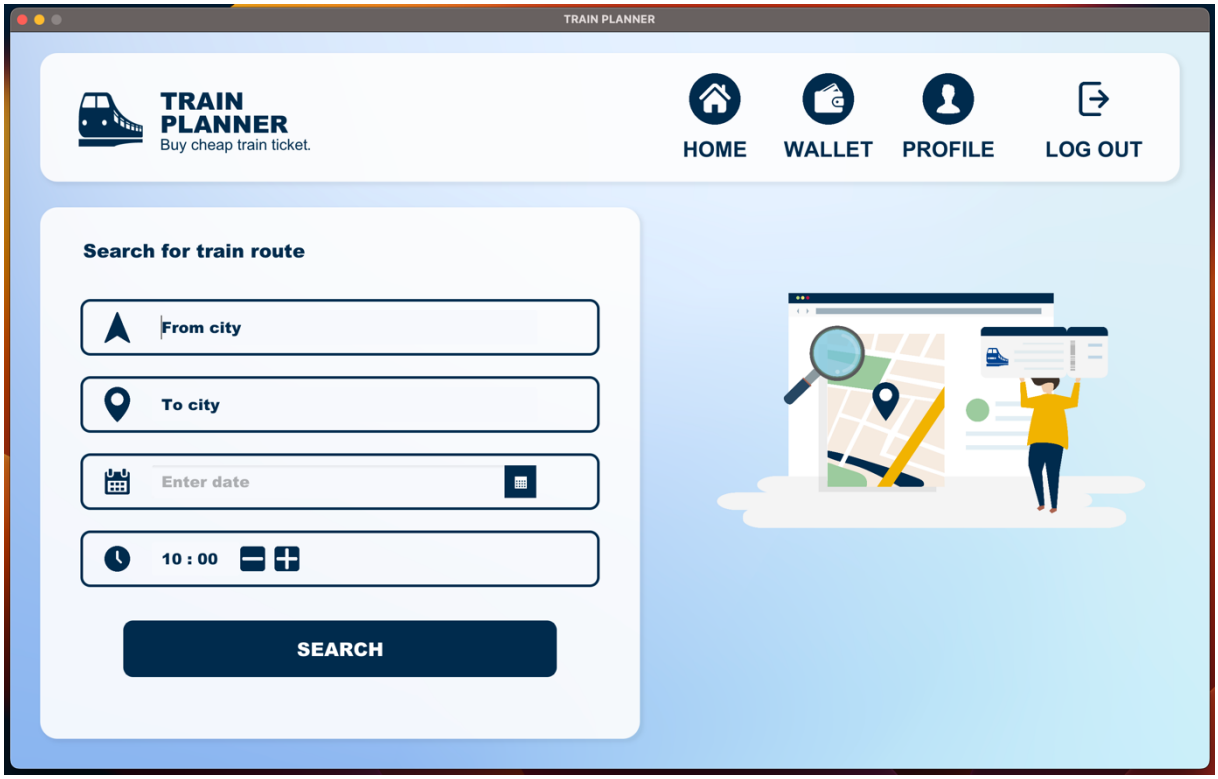

This screen is used for registration of new user. The entered data and data from the database are also checked. If user with entered data does not exist, it will be created, the information write into the database and screen changed to LOGIN SCREEN.

MAIN MENU IN LOGGED USER



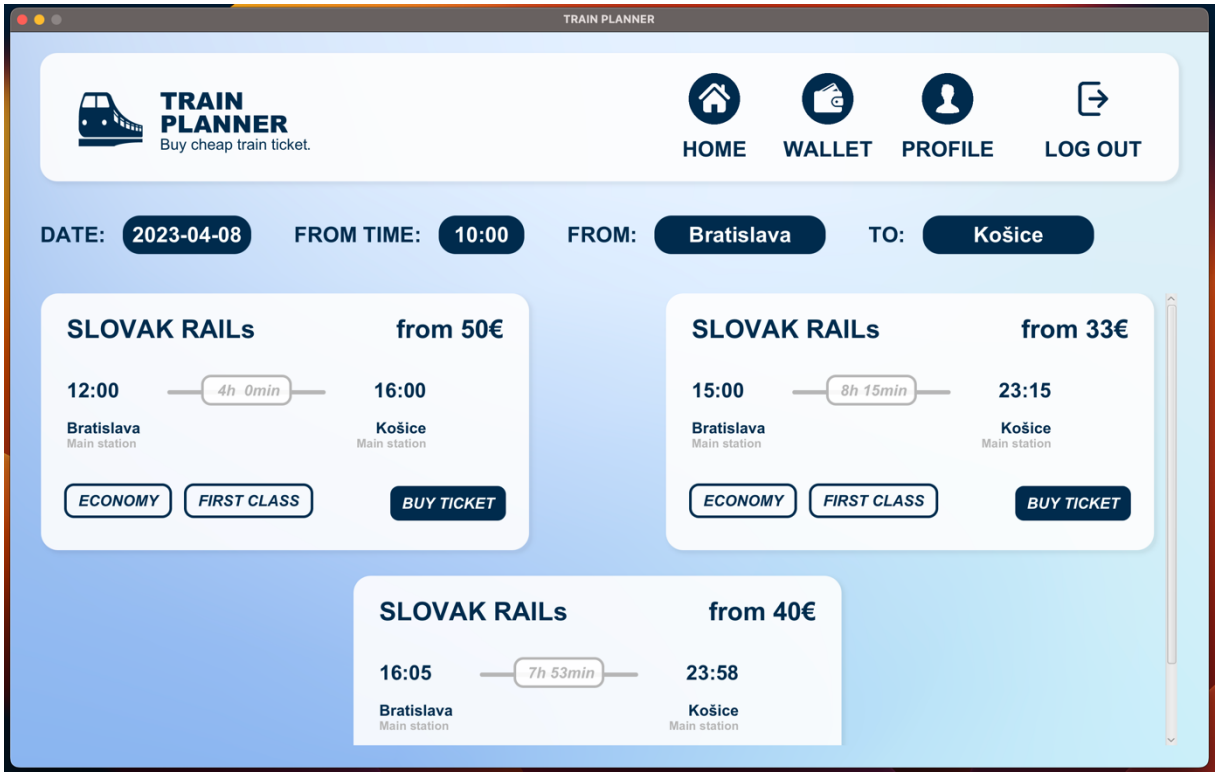

**When user is logged in on each page is same menu. In menu we can choose from HOME(USER SEARCH SCREEN), WALLET(WALLET SCREEN), PROFILE(PROFILE SCREEN).**

USER SEARCH SCREEN




After successful login, we work with the loaded user from the database. On this page can search for train route, when enter search information is correct format and click on SEARCH button, screen change to AFTER SEARCH SCREEN.

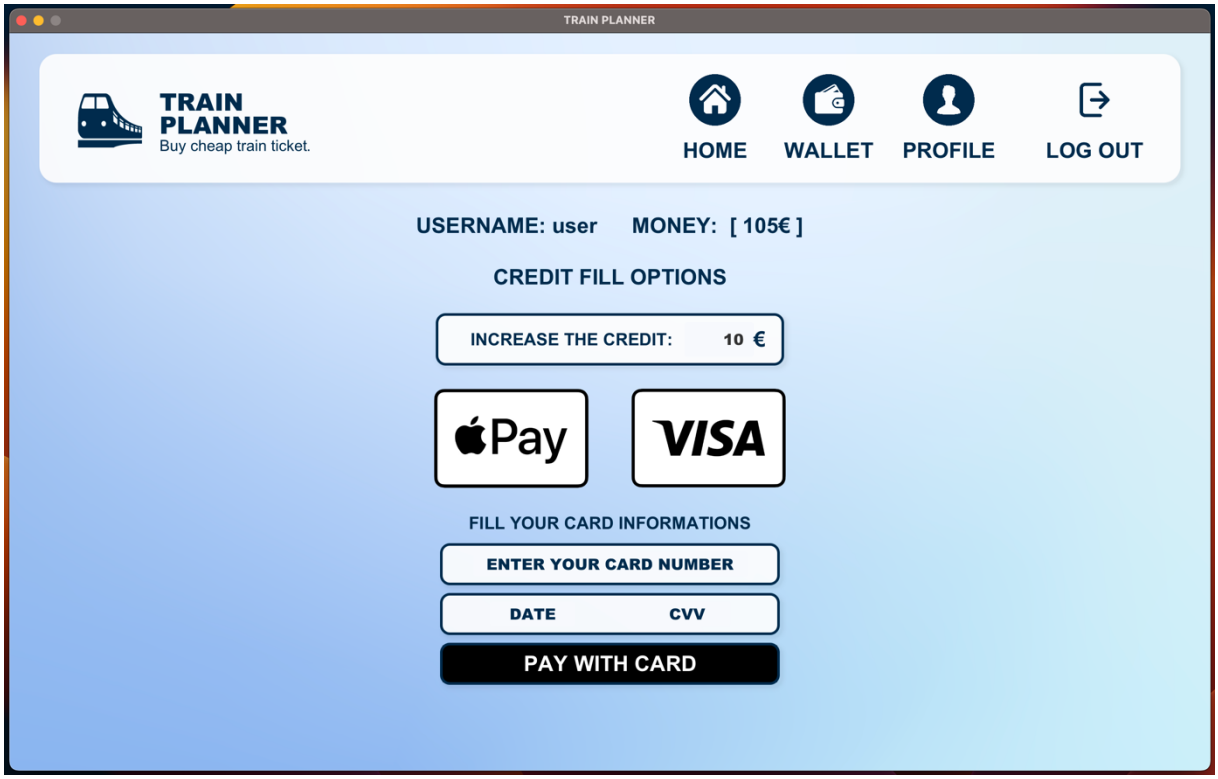
AFTER SEARCH SCREEN




At this screen, all tickets are stored in the database that match the search request are displayed as "cards" with information about the route LIKE, TRAIN COMPANY, LOWEST PRICE FOR TICKET, START AND END TIME OF ROUTE, STARTING STATION AND END STATION, DURATION OF ROUTE.
All ticket informations are loaded from database. The user can choose a ticket, select the type of seat (economy/first class) and buy the ticket. Each option has own settings thanks to POLYMORFIZM and has own range of price for ticket and number of seats. If user has enough money in the wallet, the ticket is added to the user's inventory and is also recorded in the database. AFTER BUYING TICKET, APP REDIRECT ME TO PROFILE SCREEN, WHERE I CAN SEE ALL MY TICKETS ( with some MAIN informations about bought ticket for route)

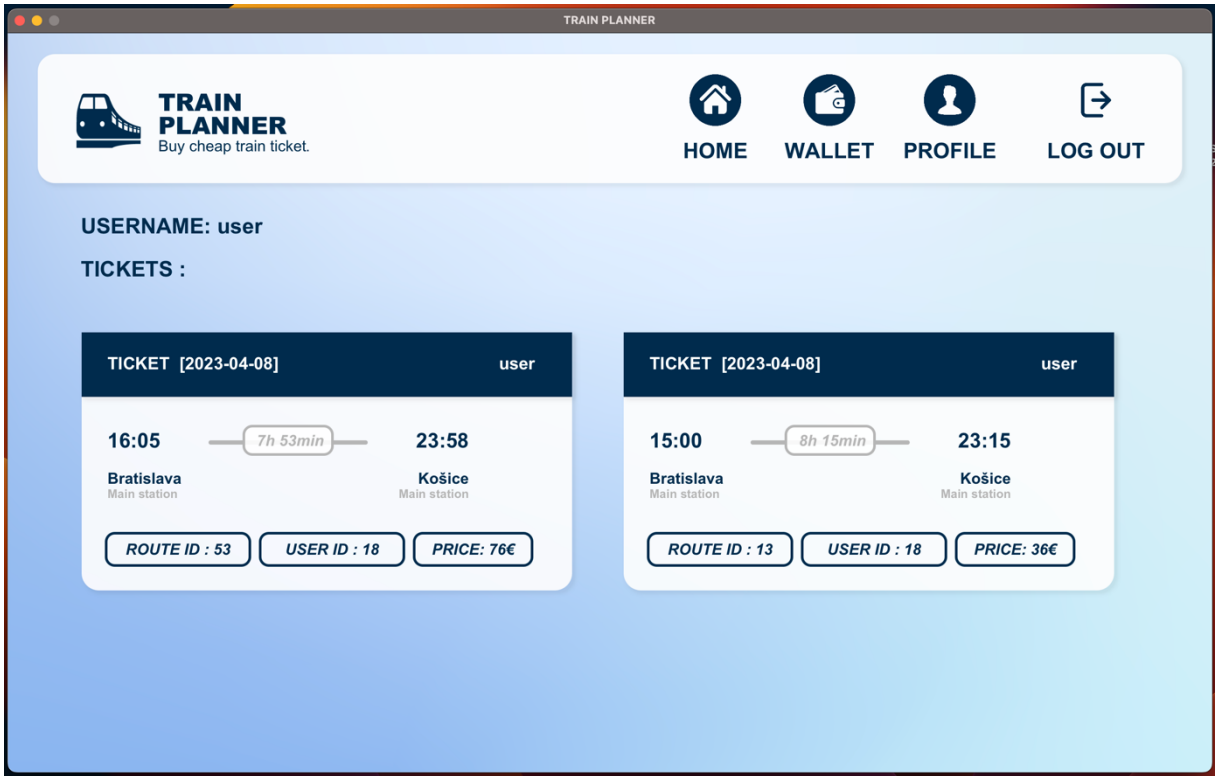
If the user does not have enough money in the wallet, app display a pop up message that it is necessary to fill up the money in the wallet. ALSO, THE PROGRAM DOESN'T ALLOW ME TO BUY MORE TICKETS FOR THE SAME ROUTE THIS PROBLEM IS FIXED BY A POPUP WINDOW.

WALLET SCREEN




This screen will show username and amount of money on user account.
When you click on the apple pay button, the entered amount is added to the wallet
When you click on the VISA button, labels for entering the card number, date and cvv will appear, if the data are valid, the entered amount will be added to the wallet, if they are not valid, an error pop-up will be displayed

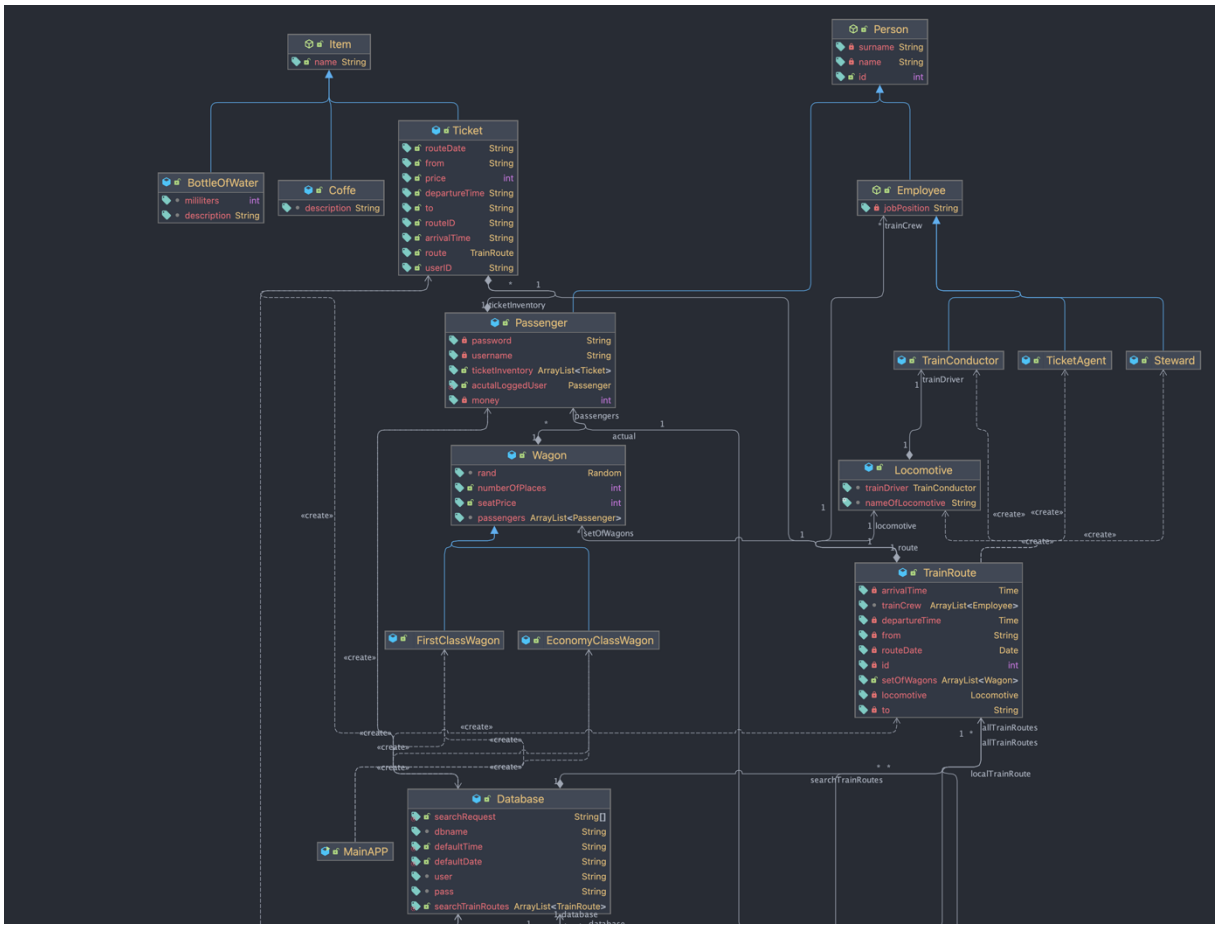
PROFILE SCREEN




On this screen are displayed all purchased tickets for the currently logged in user.
In „tickets“ table in database are stored pairs (userID, routeID), with SQL query I will load and write all the tickets that belong to the user with userID.

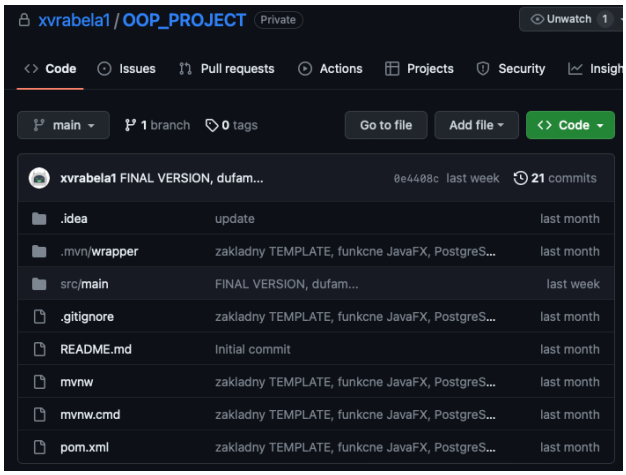
# UML DIAGRAM

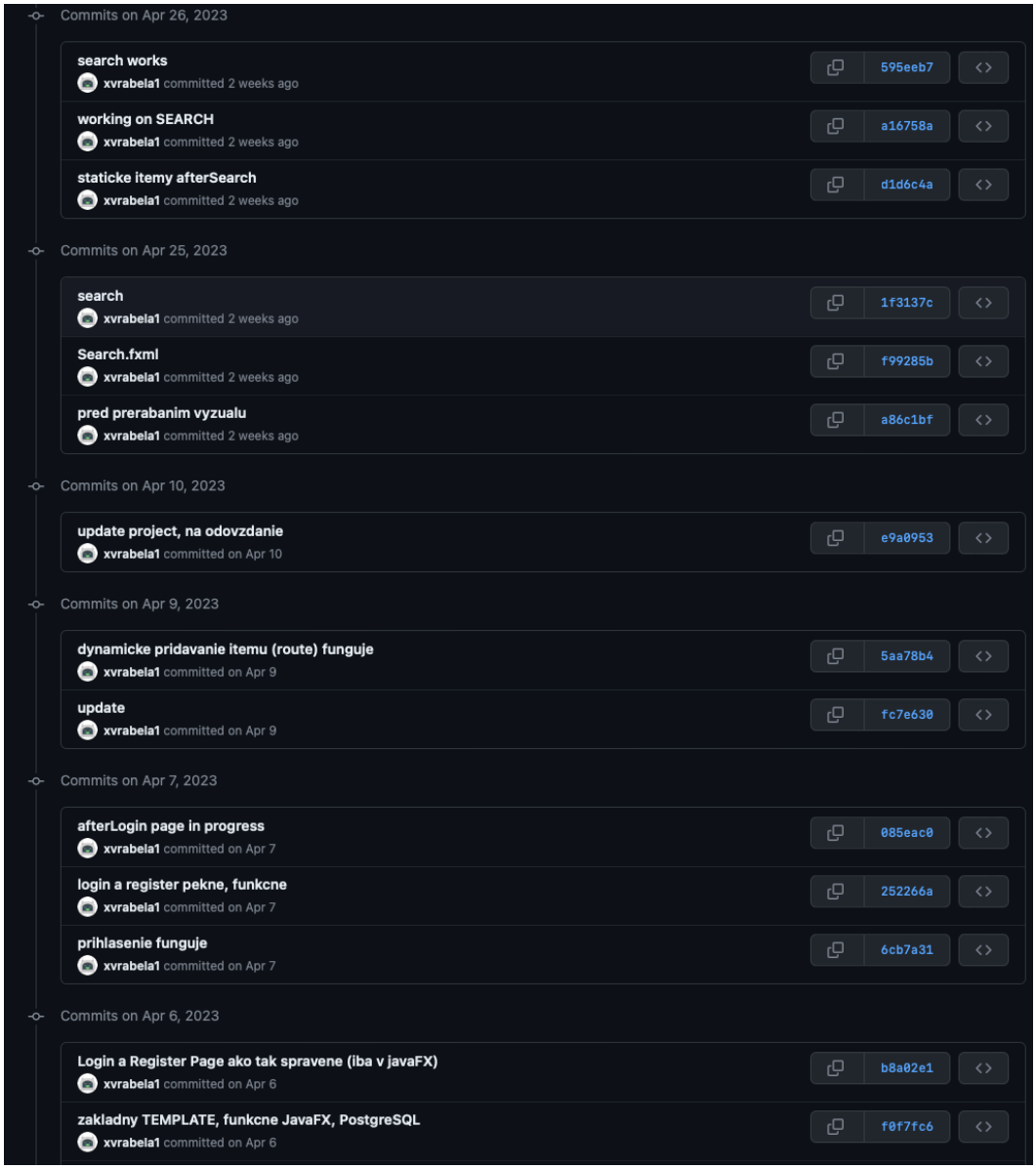
The whole photo of UML is in the .zip archive



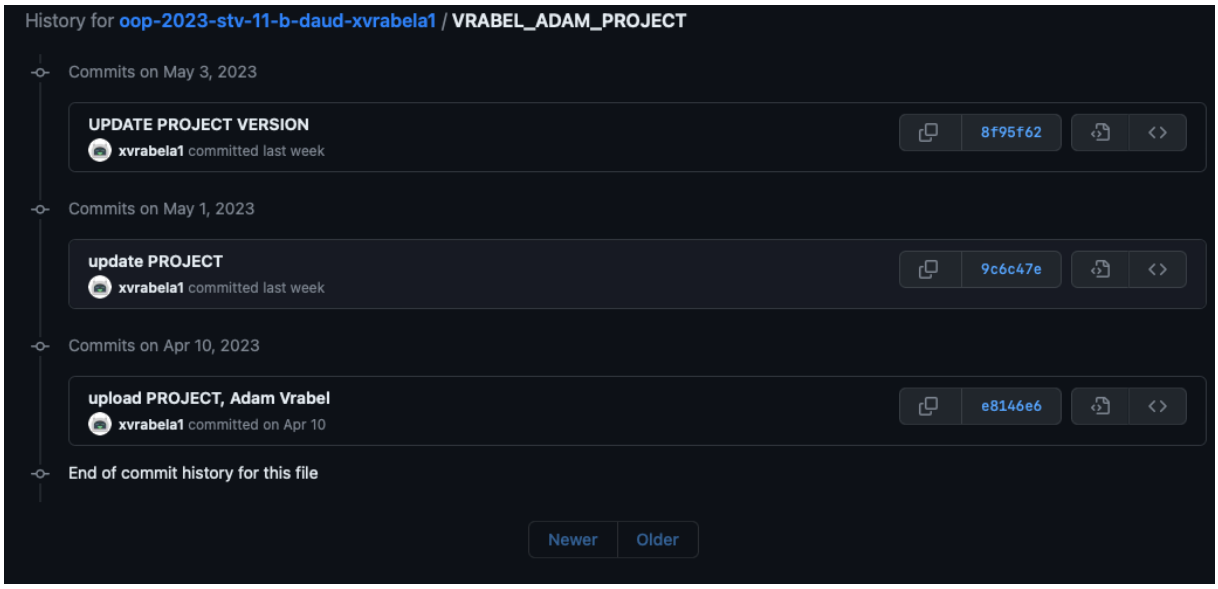
# VERSIONS ON GIT HUB

I primarily used my local git repository, where I made a commit after each completed functional part.



On the main git repository **OOP-FIIT/oop-2023-stv-11-b-daud-xvrabela1** I commit only the main versions of the program (according to uploads in AIS).



# DATABASE INFO

In my project, all functions for working with the database are in the Database class. All changes in the application are always saved/loaded from the database. My database consists of 3 tables: user_acounts, train_routes and tickets. I use a lot of SQL queries to select the manage information in my application.

Table user_accounts contains all created users accounts.

| | account_id [PK] integer | username character varying (50) | password character varying (50) | firstname character varying (50) | lastname character varying (50) | money integer | tickets character varying[] (50) |
|---|---|---|---|---|---|---|---|
| 1 | 15 | adamVrabel | heslo123 | Adam | Vrabel | 5 | [null] |
| 2 | 18 | user | user | testing | user | 45 | [null] |

Table train_routes contains all available routes with all the details.

| | id [PK] integer | from character varying (50) | to character varying (50) | departureTime time without time zone | arrivalTime time without time zone | routeDate date | fclassWagons integer | economyWagons integer |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Bratislava | Košice | 12:00:00 | 16:00:00 | 2023-04-08 | 1 | 2 |
| 2 | 2 | Bratislava | Humenné | 12:05:00 | 18:05:00 | 2023-04-08 | 1 | 2 |
| 3 | 3 | Bratislava | Žilina | 12:10:00 | 14:30:00 | 2023-04-08 | 1 | 2 |
| 4 | 4 | Humenné | Trnava | 08:00:00 | 24:00:00 | 2023-04-08 | 1 | 2 |
| 5 | 5 | Trnava | Košice | 10:00:00 | 13:00:00 | 2023-04-08 | 1 | 2 |
| 6 | 7 | Bratislava | Humenné | 10:00:00 | 14:00:00 | 2023-04-08 | 1 | 2 |
| 7 | 8 | Bratislava | Humenné | 18:00:00 | 22:30:00 | 2023-04-08 | 1 | 2 |
| 8 | 13 | Bratislava | Košice | 15:00:00 | 23:15:00 | 2023-04-08 | 1 | 2 |
| 9 | 14 | Bratislava | Humenné | 05:00:00 | 10:15:00 | 2023-04-08 | 1 | 2 |
| 10 | 15 | Košice | Bratislava | 10:00:00 | 12:00:00 | 2023-04-28 | 1 | 2 |
| 11 | 53 | Bratislava | Košice | 16:05:00 | 23:58:00 | 2023-04-08 | 1 | 2 |

In tickets table are stored purchased tickets with routeId, userId and ticketPrice.

| | ticketId [PK] integer | routeId integer | userId integer | ticketPrice integer |
|---|---|---|---|---|
| 1 | 56 | 8 | 18 | 71 |
| 2 | 57 | 2 | 18 | 84 |
| 3 | 58 | 15 | 18 | 32 |
| 4 | 59 | 3 | 15 | 71 |
| 5 | 60 | 2 | 15 | 37 |

# PROJECT CRITERIA

The program is executable and makes sense, meets the main conditions and comments from the teacher.

## INHERITANCE

For example class Passenger inherits from class Person.

```
2 usages  5 inheritors  xvrabela1
public abstract class Person {
    public int id;
    2 usages
    private String name;
    2 usages
    private String surname;

    // GETTERs / SETTERs
    xvrabela1
    public String getName() { return name; }
    xvrabela1
    public void setName(String name) { this.name = name; }
    5 usages  xvrabela1
    public String getSurname() { return surname; }
    4 usages  xvrabela1
    public void setSurname(String surname) { this.surname = surname; }
```

```
xvrabela1 *
public class Passenger extends Person{

    public static Passenger acutalLoggedUser;

    // HAS name, surname
    4 usages
    private String username;
    4 usages
    private String password;
    3 usages
    private int money;
```

Class Ticket inherits from class Item

```
15 usages  xvrabela1 *
public class Ticket extends Item{
    4 usages
    public String routeID;
    2 usages
    public String userID;
    public TrainRoute route;

    2 usages
    public String routeDate;
    2 usages
    public String departureTime;
    2 usages
    public String arrivalTime;
    public String from;
    public String to;
    2 usages
    public int price;
```

Class EconomyClassWagon inherits from class Wagon

```java
// PARENT CLASS IN POLYMORFIZM
20 usages  2 inheritors  xvrabela1 *
public class Wagon {
    2 usages
    Random rand = new Random();
    //int numberOfWagon;
    2 usages
    public int numberOfPlaces;
    13 usages
    public int seatPrice;
    no usages
    ArrayList<Passenger> passengers;


    2 usages  xvrabela1
    public Wagon() { System.out.println("CONSTRUCTOR of WAGON"); }

    1 usage  2 overrides  xvrabela1
    public void setUpWagon() { System.out.println("POLYMORFIZMUS:  from Wagon"); }
```

```java
4 usages  xvrabela1 *
public class EconomyClassWagon extends Wagon{

    1 usage  xvrabela1 *
    @Override
    public void setUpWagon() {
        //super.setUpWagon();

        System.out.println("POLYMORFIZMUS:  from Economy Class Wagon");
        numberOfPlaces = 30; // miest

        //seatPrice = 5; // euro
        seatPrice = rand.nextInt( origin: 30, bound: 51);     // RANDOM PRICE 30-50€
    }
```

# POLYMORPHISM

I use polymorphism to set up EconomyClassWagon and FirstClassWagon. Each class of Wagon has different settings for number of seats and price for seat.

```java
// PARENT CLASS IN POLYMORFIZM
20 usages  2 inheritors  xvrabela1 *
public class Wagon {
    2 usages
    Random rand = new Random();
    //int numberOfWagon;
    2 usages
    public int numberOfPlaces;
    13 usages
    public int seatPrice;
    no usages
    ArrayList<Passenger> passengers;


    2 usages  xvrabela1
    public Wagon() { System.out.println("CONSTRUCTOR of WAGON"); }

    1 usage  2 overrides  xvrabela1
    public void setUpWagon() { System.out.println("POLYMORFIZMUS:  from Wagon"); }
```

```java
4 usages  xvrabela1 *
public class EconomyClassWagon extends Wagon{

    1 usage  xvrabela1 *
    @Override
    public void setUpWagon() {
        //super.setUpWagon();

        System.out.println("POLYMORFIZMUS:  from Economy Class Wagon");
        numberOfPlaces = 30; // places

        seatPrice = rand.nextInt( origin: 30, bound: 51);    // RANDOM PRICE 30-50€
    }
```

```java
5 usages  xvrabela1 *
public class FirstClassWagon extends Wagon{

    1 usage  xvrabela1 *
    @Override
    public void setUpWagon() {
        //super.setUpWagon();

        System.out.println("POLYMORFIZMUS:  from First Class Wagon");
        numberOfPlaces = 10; // places

        seatPrice = rand.nextInt( origin: 70, bound: 101);    // RANDOM PRICE 70-100€
    }
```

# AGGREGATION

Aggregation represents HAS-A relationship. I using lot of aggregation in my project. For example Passenger HAS-A Ticket (ArrayList of Tickets)

```
xvrabela1 *
public class Passenger extends Person{

    public static Passenger acutalLoggedUser;

    // HAS name, surname
    4 usages
    private String username;
    4 usages
    private String password;
    3 usages
    private int money;

    3 usages
    public ArrayList<Ticket> ticketInventory = new ArrayList<Ticket>();    // INVENTORY OF USER TICKETS, loaded from DB
```

# ENCAPSULATION

In encapsulation, a class's variables are hidden from other classes and can only be accessed by the methods of the class in which they are found.

For example in Person class and in all classes extended from this one.

```
xvrabela1 *
public class Passenger extends Person{

    public static Passenger acutalLoggedUser;

    // HAS name, surname
    4 usages
    private String username;
    4 usages
    private String password;
    3 usages
    private int money;
```

```
public abstract class Person {
    public int id;
    2 usages
    private String name;
    2 usages
    private String surname;

    // GETTERs / SETTERs
    xvrabela1
    public String getName() { return name; }
    xvrabela1
    public void setName(String name) { this.name = name; }
    5 usages  xvrabela1
    public String getSurname() { return surname; }
    4 usages  xvrabela1
    public void setSurname(String surname) { this.surname = surname; }
```

```
4 usages  xvrabela1
public int getUserID() {return id;}
xvrabela1
public Passenger(int id, String username, String password, String name, String surname, int money){...}
xvrabela1
public String getUsername() {return username;}
no usages  xvrabela1
public String getPassword() {return password;}
xvrabela1
public void setUsername(String username) { this.username = username; }
no usages  xvrabela1
public void setPassword(String password) {this.password = password;}
14 usages  xvrabela1
public int getMoney() {return money;}
3 usages  xvrabela1
public void setMoney(int money) { this.money = money; }
```

## OVERLOADING

I also use overloading in my project. For example in Database class i overload disconectDB(), according to the number of parameters, the function is executed

```
10 usages  xvrabela1
private void disconectDB(ResultSet resultSet, Statement statement, Connection connection) throws SQLException {
    try {
        resultSet.close();
        statement.close();
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    //if (connection.isClosed() && statement.isClosed() && resultSet.isClosed()) {
    //    System.out.println("The connection to the database was successfully closed.");
    //}
    //System.out.println("_____");
}
7 usages  xvrabela1
private void disconectDB(Statement statement, Connection connection) throws SQLException {
    try {
        statement.close();
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
//      if (connection.isClosed() && statement.isClosed()) {
//          System.out.println("The connection to the database was successfully closed.");
//      }
//      System.out.println("_____");
}
```

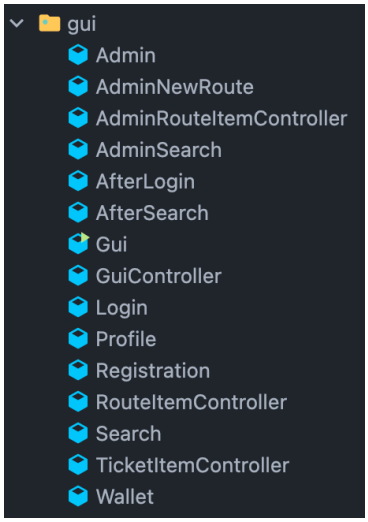
## OVERRIDING

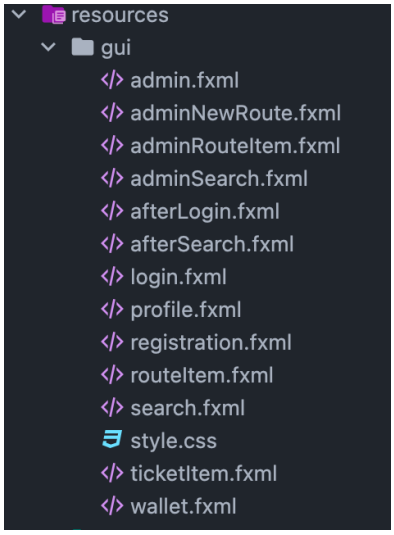
I use overriding in my project. For example i override setUpWagon() in EconomyClassWagon from Wagon class.

```
public class EconomyClassWagon extends Wagon{

    1 usage  xvrabela1 *
    @Override
    public void setUpWagon() {
        //super.setUpWagon();

        System.out.println("POLYMORFIZMUS:  from Economy Class Wagon");
        numberOfPlaces = 30; // places

        seatPrice = rand.nextInt( origin: 30, bound: 51);     // RANDOM PRICE 30-50€
    }
```

# GUI

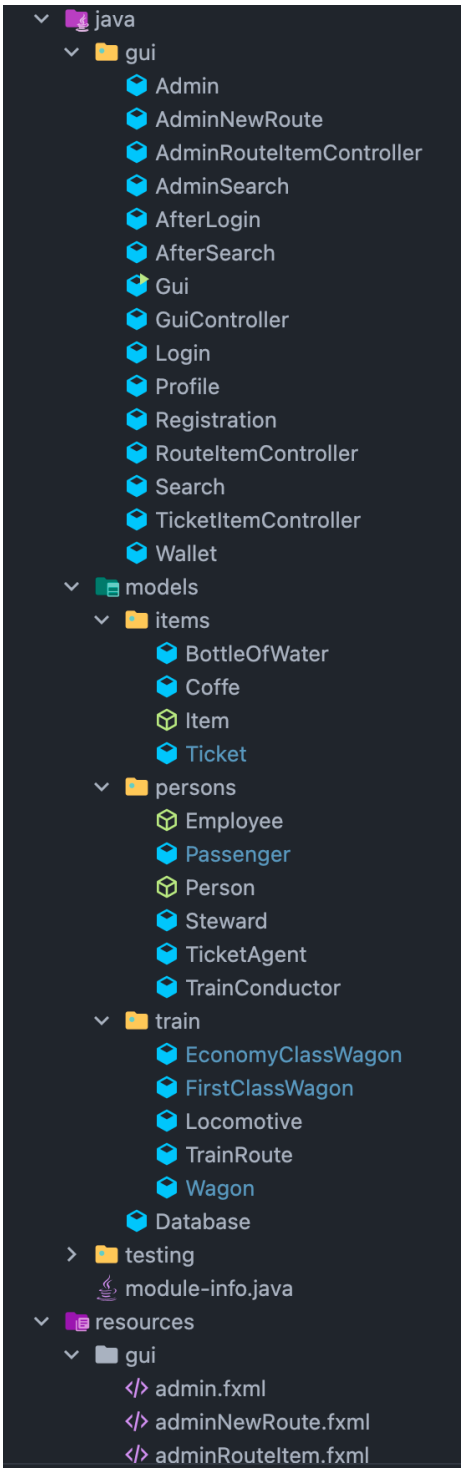
My GUI is nice, clear and user-friendly. Images are in SCREENS AND DESCRIPTION section. I used Scene Builder to create and design .fxml files. For custom design i used .css styles. I also used JavaFX and control app flow with Controllers class for each .fxml file (controller classes is in gui package)

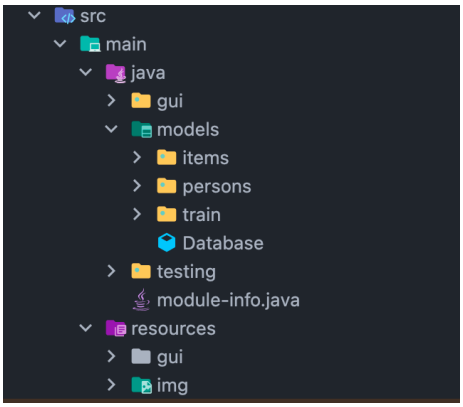


# PACKAGES

Program is divided into packages by MVC pattern = MODEL, VIEW, CONTROLER
MODEL = models package
VIEW = gui package in rescources
CONTROLLER = gui package in java



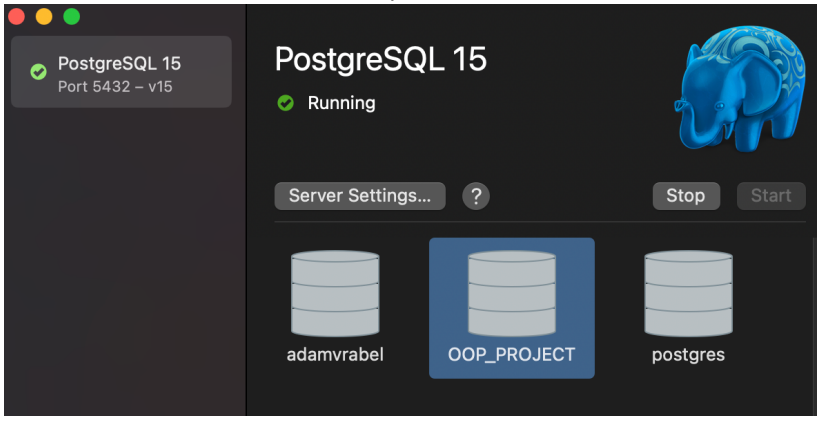
# EXCEPTIONS

I used TRY CATCH exceptions with database conecting, resultSet, close DB connection and everywhere they were needed.

```
16 usages  xvrabela1 *
private Connection connectDB(){
    //System.out.println("_____");
    Connection connection = null;

    try{
        Class.forName( className: "org.postgresql.Driver");             // LOAD CONTROLLER

        String url  = "jdbc:postgresql://localhost:5432/"+dbname;
        connection = DriverManager.getConnection(url, user, pass);    // CREATE CONNECTION

        //if(connection != null){
        //    System.out.println("Connected to database !");
        //}
        //else{
        //    System.out.println("Database connection FAILED !");
        //}

    } catch (ClassNotFoundException e) {
        System.out.println("The driver for the database could not be loaded.");
        //e.printStackTrace();

    } catch (SQLException e) {
        System.out.println("Database connection FAILED !");
        //e.printStackTrace();
    }

    return connection;
}
```

# SERIALIZATION

I don have, because I worked with the database and I use a lot of query commands where I work with data from the database (everything that changes in the application is loaded/saved to the database)



# JAVA DOC AND COMMENTS

All the code is commented and important functions also has javadoc.

```
CLASS FOR WORK WITH DATABASE
USING:
• Postgres.app (PostgreSQL 15)
• pgAdmin 4
• PostgreSQL JDBC Driver

All changes in the application are always saved/loaded from the database.
DB consists of 3 tables: user_acounts, train_routes and tickets.
I use a lot of SQL queries to select the manage information in my application


xvrabela1 *
public class Database {

    5 usages
    public static String defaultDate = "2023-04-08";
    2 usages
    public static String defaultTime = "10:00";
```

```
ECONOMY WAGON
extends from Wagon, has own settings
4 usages  xvrabela1 *
public class EconomyClassWagon extends Wagon{

    1 usage  xvrabela1 *
    @Override
    public void setUpWagon() {
        //super.setUpWagon();

        System.out.println("POLYMORFIZMUS:  from Economy Class Wagon");
        numberOfPlaces = 30; // places

        seatPrice = rand.nextInt( origin: 30, bound: 51);     // RANDOM PRICE 30-50€
    }
```

```
Passenger is user in app
• can search routes
• can buy tickets
• has ticket inventory
• has money in account

xvrabela1 *
public class Passenger extends Person{

    public static Passenger acutalLoggedUser;

    // HAS name, surname
    4 usages
    private String username;
    4 usages
    private String password;
    3 usages
    private int money;

    3 usages
    public ArrayList<Ticket> ticketInventory = new ArrayList<~>();    // INVENTORY OF USER TICKETS, loaded from DB
```