

Slovenská technická univerzita

FIIT

## **Správa o realizácii projektu**

FuelFinder

Samuel Račák

## Obsah

Správa o realizácii projektu .....	1
Úvod a zámer projektu.....	2
Riešenie problémov .....	2
Strategy .....	7
Multithreading .....	8
Vlastne výnimky .....	8
Serializácia.....	9
Používanie vhniedzenej triedy .....	9
Explicitne používanie RTTI.....	10
Diagram tried .....	11

# Úvod a zámer projektu

Ako zadanie sme mali vypracovať objektovo orientovaný program so zameraním na cesty.

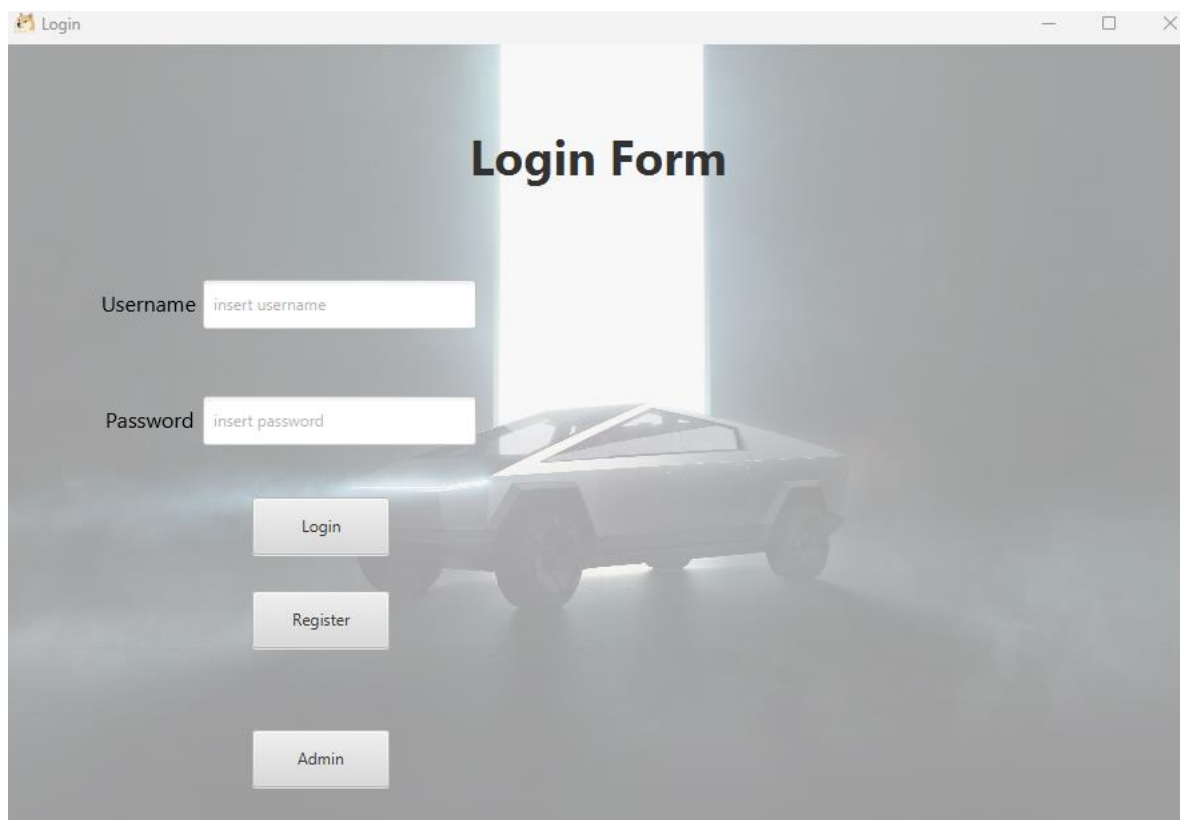
Každý správny vodič si svoju cestu vždy dopodrobna naplánuje. Jeden z problémov, ktorým čelí veľa z nás je aj výber správnej čerpacej stanice. Chceme aby sme natankovali čo najlepšie a najlacnejšie palivo, aby sme si mohli umyť auto, alebo dofúkať pneumatiky. Presne preto by ste si mali stiahnuť FuelFinder.

Táto aplikácia vám umožní rýchlo a efektívne vyhľadávať všetko o pumpách. Pomocou hodnotenia od ostatných používateľov zistíte, ktorá pumpa ponúka nadštandardné služby, kde si môžete umyť auto, kde majú najlepšie jedlo, alebo najčistejšie záchody. Okrem toho si budete môcť napláňovať cestu medzi miestami a aplikácia vás bude navigovať tak, aby ste pri tankovaní za vašu cestu zaplatili čo najmenej. Ďalej si budete môcť priplatiť za prémiové funkcie ako zaplataenie za palivá cez aplikáciu, alebo zbieranie kupónov či iných výhodných zliav.

## Riešenie problémov

Asi najdôležitejšia časť programu, keďže s ňou interaguje každý používateľ. V mojom riešení som vytvoril niekoľko scén, každá z nich rieši iný problém.

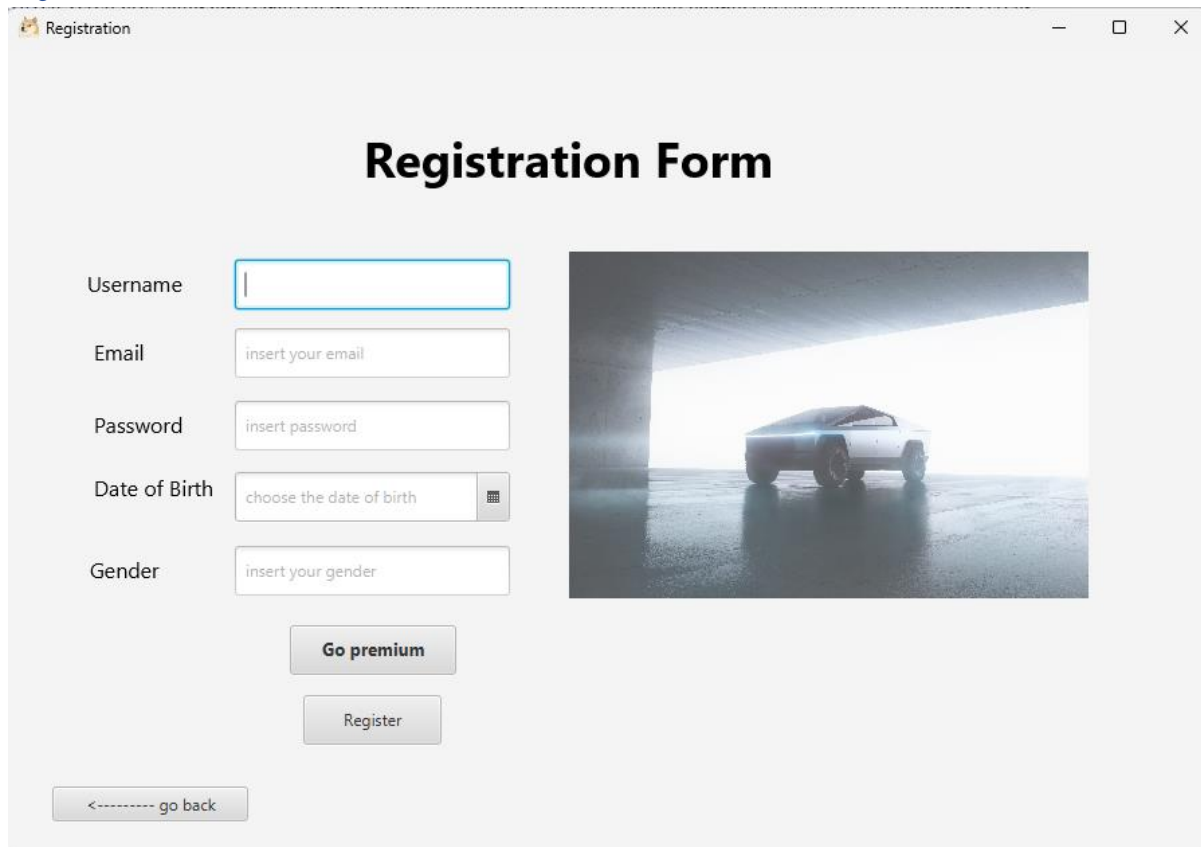
### *Prihlasovanie*



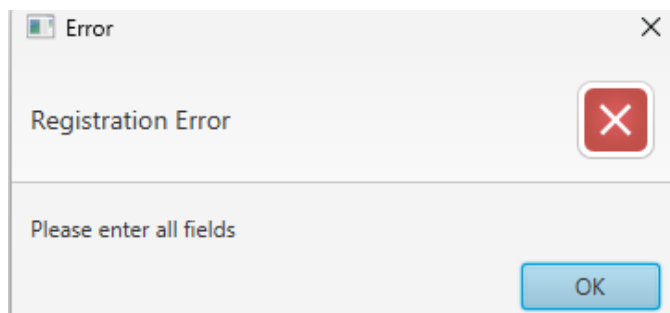
Používateľ tu má na vyber z niekoľkých scenárov používania, pričom každý z nich je ošetrený proti prípadnému zneužitiu. Napríklad sa používateľ nedostane do hlavnej časti programu ak nie je zaregistrovaný. Takisto sa nedostane ani do administrátorskej časti.

Po kliknutí na tlačidlo register je scéna zmenená na registrovanie, kde si ďalej môže vybrať medzi prémiovým a normálnym používateľom.

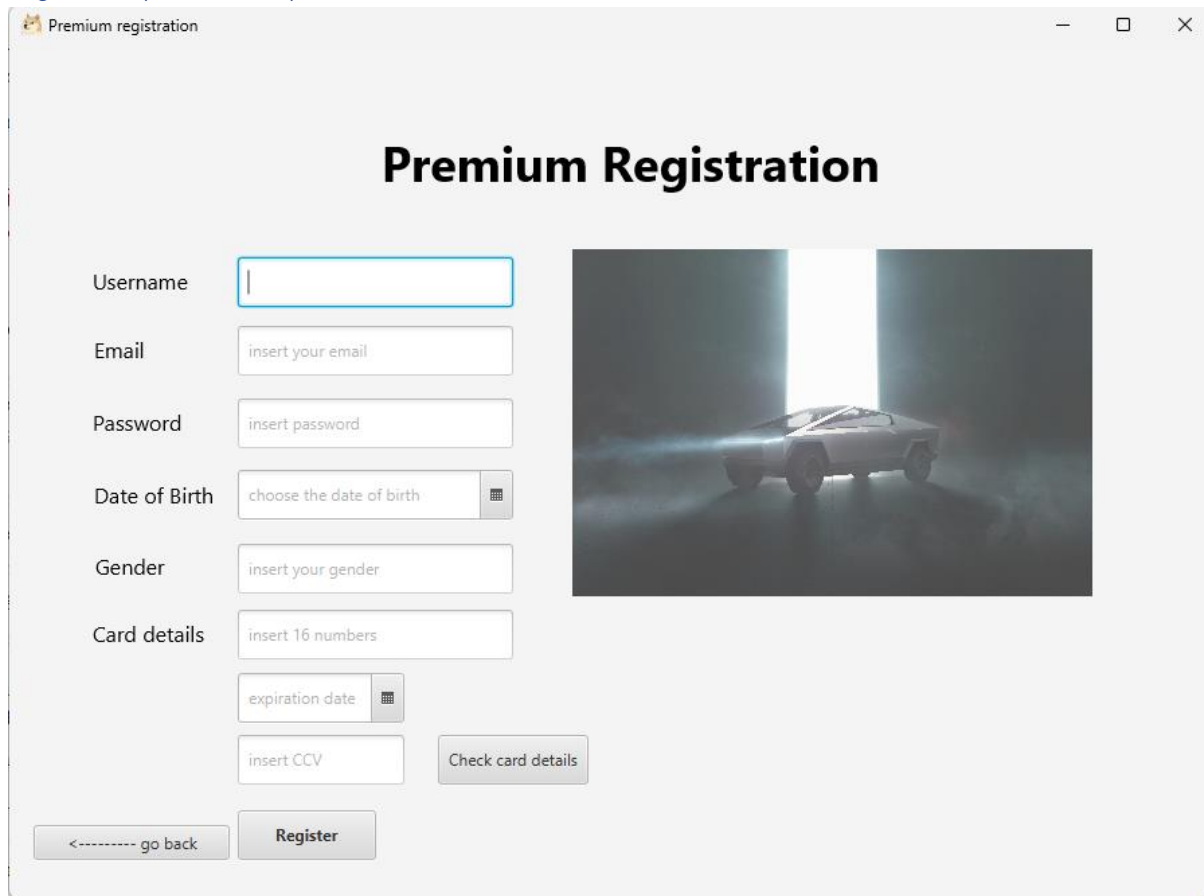
### Registrácia

A screenshot of a web application window titled "Registration". The window has a light gray background and standard window controls (minimize, maximize, close) in the top right corner. The title bar also shows a small cat icon and the text "Registration". The main heading "Registration Form" is centered in a large, bold, black font. Below the heading, there are five input fields arranged vertically on the left side, each with a label to its left: "Username" (with a text input field), "Email" (with a text input field containing the placeholder "insert your email"), "Password" (with a text input field containing the placeholder "insert password"), "Date of Birth" (with a date picker field containing the placeholder "choose the date of birth" and a calendar icon), and "Gender" (with a text input field containing the placeholder "insert your gender"). To the right of these fields is a large rectangular image showing a sleek, futuristic car parked in a dark, industrial-looking space with a bright light source in the background. Below the input fields, there are two buttons: "Go premium" and "Register", both with a light gray background and rounded corners. At the bottom left, there is a button labeled "<----- go back".

Pri tejto scéne musí užívateľ vyplniť všetky údaje a je tu zároveň kontrolované či ešte neexistuje iný používateľ s daným používateľským menom. V prípade, že nastala chyba – používateľ už existuje/nevyplnil všetky kolónky je mu zobrazené upozornenie.

A screenshot of an "Error" dialog box. The window has a light gray title bar with the text "Error" and a close button (X) in the top right corner. The main content area has a light gray background. At the top, the text "Registration Error" is displayed in a bold, black font. To the right of this text is a red square button with a white "X" inside. Below the text, there is a line of text that says "Please enter all fields". At the bottom right of the dialog box, there is a blue button with the text "OK" in white.

Ak sa používateľ rozhodne “zaplatiť si” prémiiu musí kliknúť na tlačidlo Go premium, ktoré ho presmeruje na registráciu prémiového používateľa.



The image shows a 'Premium registration' window with a title bar containing a cat icon and the text 'Premium registration'. The window has standard minimize, maximize, and close buttons. The main heading is 'Premium Registration'. On the left, there are input fields for 'Username', 'Email', 'Password', 'Date of Birth', 'Gender', and 'Card details'. The 'Card details' section includes fields for 'insert 16 numbers', 'expiration date', and 'insert CCV', along with a 'Check card details' button. At the bottom left are 'go back' and 'Register' buttons. On the right is a large image of a car in a dark setting with a bright light source behind it.


Premium registration

## Premium Registration

Username


Email


Password

Date of Birth  

Gender

Card details





Podobne ako pri registrácii základného používateľa musí prémiový používateľ vyplniť všetky údaje. Navyše je tu zadávanie kreditnej karty a jej kontrola, ktorá opäť upozorni používateľa na vyplnenie políček / zadanie správnej karty.

## Hlavná strana program

The screenshot shows the main application window with the following elements:

- Header:** "Main" title bar with standard window controls.
- User Profile:** Username "kapitan" and a circular profile picture.
- Logout Button:** A button labeled "Logout" in the top right corner.
- Location Input:** Fields for "Current location: from" and "Destination: to", each with a "set" button.
- Navigation Mode:** A dropdown menu currently set to "Shortest".
- Navigate Button:** A button labeled "Navigate".
- Map Area:** A large empty rectangular box intended for a map.
- Fuel Station List:** A scrollable list on the right side showing details for several fuel stations:
  - Fuel Station 26:** Fuels: {LPG=8, PETROL=1, HYDROGEN=4, DIESEL=9}, Services: [Car Repair Price: 223\$, Car Parking Price: 114\$], Rating: 1.3477029807724121, #Reviews: 9.
  - Fuel Station 28:** Fuels: {LPG=8, PETROL=1, HYDROGEN=5, DIESEL=1}, Services: [Car Rental Price: 434\$, Car Repair Price: 237\$, Car Parking Price: 114\$], Rating: 3.731178111556353, #Reviews: 87.
  - Fuel Station 4:** Fuels: {LPG=0, PETROL=0, HYDROGEN=8, DIESEL=6}, Services: [Car Repair Price: 5\$, Car Wash Price: 140\$, Car Insurance Price: 140\$], Rating: 3.9982948697660987, #Reviews: 50.
  - Fuel Station 1:** Fuels: {LPG=7, PETROL=4, HYDROGEN=1, DIESEL=3}, Services: [Car Wash Price: 246\$, Car Parking Price: 407\$, Car Rental Price: 114\$], Rating: 1.178611677809931, #Reviews: 33.
  - Fuel Station 27:** Fuels: {LPG=6, PETROL=1, HYDROGEN=2, DIESEL=2}, Services: [Car Repair Price: 308\$, Car Parking Price: 404\$, Car Towing Price: 140\$], Rating: 3.8963308306026505, #Reviews: 77.

Na tejto stránke ma používateľ k dispozícii niekoľko možností. V pravom hornom rohu ma tlačidlo odhlásiť, ktoré používateľa odhlási. Pod ním sa nachádza list miest a púmp, z ktorého si môže ľubovoľne vybrať a tlačidlami set nastaviť či už východzia pozíciu, alebo destináciu. Pod danými tlačidlami sa nachádza možnosť vybraní si počítania cesty. Používateľ ma na vyber 2 typy medzi ktorými si musí vybrať. Prvým je počítanie najkratšej cesty, a druhým tej najlacnejšej.

Potom ako klikne na tlačidlo navigate sa mu v zozname zobrazí výpis bodov, ktorými musí prejsť spolu s ich atribútmi, ako napríklad vzdialenosť, maximálna rýchlosť, atrakcie....

## Ukážka rozdielu medzi počítaním najlacnejšej a najkratšej cesty

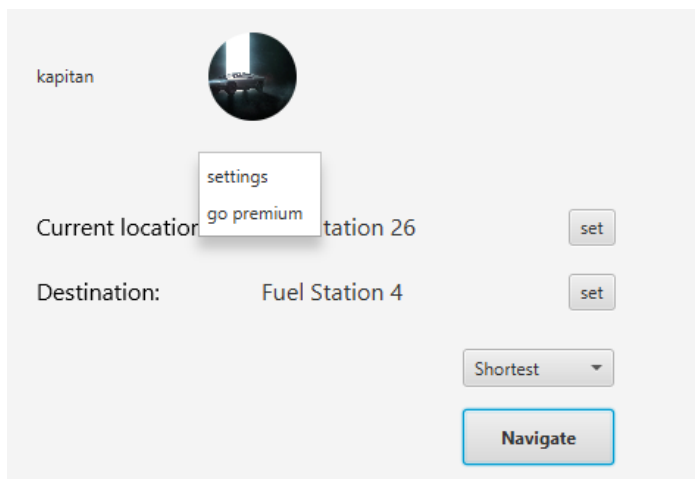
This block shows two side-by-side screenshots of the application interface to compare the 'Cheapest' and 'Shortest' navigation modes.

**Left Screenshot (Cheapest mode):**

- Current location:** Fuel Station 26
- Destination:** Fuel Station 4
- Navigation Mode:** "Cheapest" (selected in the dropdown)
- Navigate Button:** A blue button labeled "Navigate".
- Result Panel:** Displays the route from Fuel Station 26 to Fuel Station 4, including fuel types, services, rating, and maximum speed (127km/h).

**Right Screenshot (Shortest mode):**

- Current location:** Fuel Station 26
- Destination:** Fuel Station 4
- Navigation Mode:** "Shortest" (selected in the dropdown)
- Navigate Button:** A blue button labeled "Navigate".
- Result Panel:** Displays the route from Fuel Station 26 to Fuel Station 4, including fuel types, services, rating, and maximum speed (57km/h).



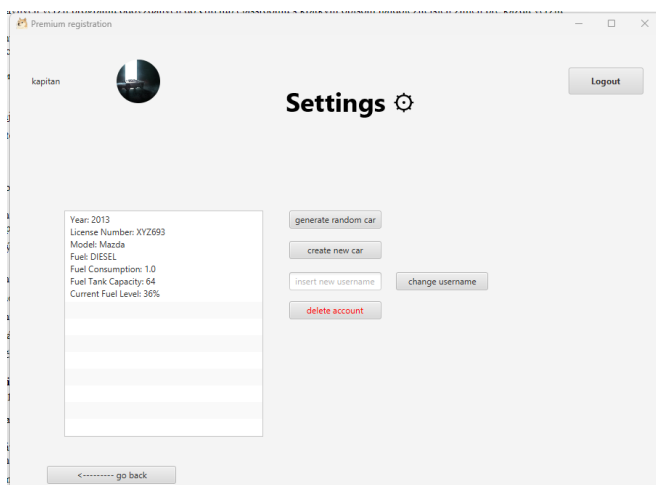
### Go premium

Po kliknutí na profilovú fotku používateľa sa pod ňou zobrazí menu s možnosťami go premium a settings. Ak používateľ klikne na go premium bude mu zobrazený box, kde ma možnosť zadať detaily svojej karty, a po jej overení bude zmenený na prémiového používateľa.

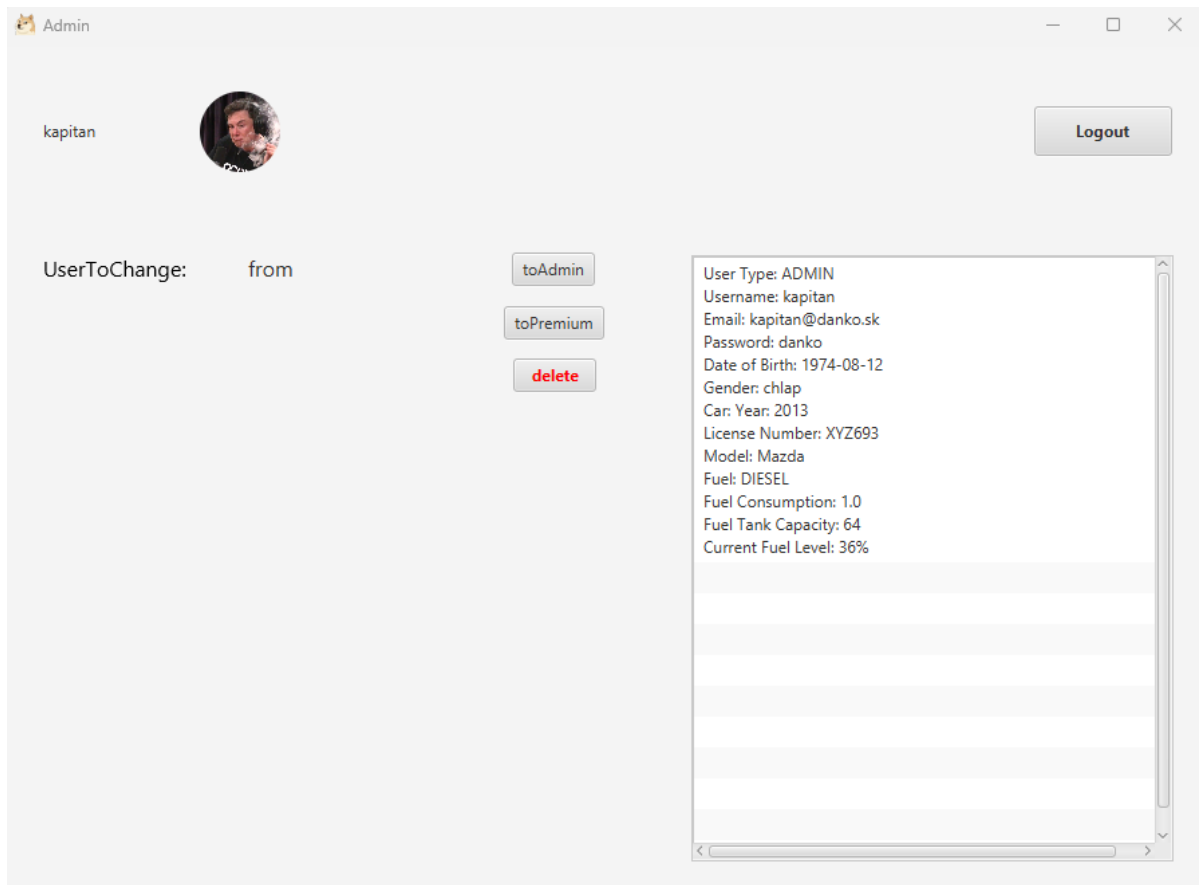
### Settings

V nastaveniach ma používateľ možnosť zmeniť si svoje užívateľské meno, ak zadane už nie je použité, môže si nechať vygenerovať náhodne auto, prípadne si vytvoriť vlastne.

Auto používateľa je následne zobrazené v ľavom liste pod profilovou fotkou.



## Admin



V tejto scéne môže používateľ ak je typu administrátor meniť typy ostatných používateľov, prípadne ich odstraňovať, ak nie sú tiež typu admin. Do tejto scény sa používateľ dostane iba ak je sám typu administrátor.

## Strategy

Asi najzaujímavejším problémom vyplývajúcim zo zadania bolo riešenie grafových algoritmov. Tu som splnil aj druhé kritérium hodnotenie čím bolo použitie návrhového vzoru strategy. Vďaka tomuto vzoru si mohol používateľ voľiť medzi jednotlivými typmi hľadania (najlacnejší a najkratší).

Strategy som implementoval pomocou rozhrania strategy, ktoré vyžadovalo metódu findRoute.

```
/**
 * this interface will be implemented by different classes which will try to
 * find a route between given points
 */
interface Strategy {
    /**
     * tries to find Route between two given points
     *
     * @param start starting point
     * @param finish ending point
     * @param graph graph of routes
     * @param car car used to find route
     * @return a {@code List of} {@link Route} from start to finish
     */
    List<Route> findRoute(PointOfInterest start, PointOfInterest finish, Graph graph, Car car);
}
```



Následne som spravil kontajnerovú classu ktorá si udržiavala aktuálnu stratégiu a na nej potom zavolała metódu find route.

```
/**
 * this class will be used to change the strategy of finding a route between two
 * points of interest (POIs) this is Implementation of Strategy Pattern
 *
 */
public class Container {
    private Strategy strategy;
```

Dve stratégie, ktoré tento interface implementovali boli triedy DijkstraOptimized a DijkstraCost.

```
/**
 * Finds the shortest route between two points of interest (POIs) using strategy
 * which is set in the container
 *
 * @param start      the starting POI
 * @param destination the destination POI
 * @param car        the car that will be used to travel the route
 * @return List<Route>
 */
public List<Route> findRoute(PointOfInterest start, PointOfInterest destination, Car car) {
    return container.findRoute(start, destination, graph, car);
}
```

## Multithreading

Kedže v grafe môže byť uložených veľa miest môže tento výpočet trvať pomerne dlho. Preto som sa rozhodol tento výpočet presunúť do ďalšieho threadu. Vďaka tomu nebude UI zablokovane a používateľ bude môcť napríklad meniť svoje atribúty.

```
Task<List<Route>> navigationTask = new Task<List<Route>>() {
    @Override
    protected List<Route> call() {
        sessionManager.changeStrategy(strategyChoice.getValue());
        return sessionManager.navigate(currentLocation, destinationLocation,
            sessionManager.getCurrentUser().getCar());
    }
};
```

## Vlastne výnimky

Počas vytvárania používateľa môže dôjsť k rozličným chybám, napríklad používateľ zadá neplatnú kartu, alebo sa skúša zaregistrovať už s existujúcim používateľským menom. Pre tento prípad som sa rozhodol vyhadzovať výnimky v mojej singleton classe UserManager, ktorej hlavnou úlohou je práca s používateľskými účtami.

```

package exceptions;

/**
 * The userDoesNotExistException class represents an exception that is thrown
 * when a user does not exist.
 */
public class userDoesNotExistException extends Exception {

    /**
     * deletes the given user from the registered users also deletes admin
     *
     * @param in
     * @return the deleted user
     * @throws userDoesNotExistException
     */
    public User deleteUser(User in) throws userDoesNotExistException {

```

## Serializácia

Na ukladanie a načítavanie používateľov medzi zmenami stavu aplikácie (vypnutá/zapnutá) používa Serializácia.

```

/**
 * saves the registered users to a file
 *
 * @param filename the name of the file
 */
public void saveToFile(String filename) {
    try (FileOutputStream fileOut = new FileOutputStream(filename);
        ObjectOutputStream out = new ObjectOutputStream(fileOut)) {
        out.writeObject(registeredUsers);
    } catch (IOException i) {
        i.printStackTrace();
    }
}

```

## Používanie vnútornej triedy

Trieda comparator je anonymnou vnútornou triedou používanou na zadefinovanie porovnania pre objekty.

```

/**
 * returns a list of all POIs sorted by fuel stations first
 * also has a comparator to sort the list by fuel stations first
 *
 * @return list of all POIs sorted by fuel stations first
 */
public List<PointOfInterest> getPOIs() {
    Comparator<PointOfInterest> poiComparator = new Comparator<PointOfInterest>() {
        @Override
        public int compare(PointOfInterest poi1, PointOfInterest poi2) {
            if (poi1 instanceof FuelStation && !(poi2 instanceof FuelStation)) {
                return -1;
            } else if (!(poi1 instanceof FuelStation) && poi2 instanceof FuelStation) {
                return 1;
            } else {
                return 0;
            }
        }
    };
    List<PointOfInterest> sortedPOIs = poiManager.getPOIs();
    sortedPOIs.sort(poiComparator);
    return sortedPOIs;
}

```

Explicitne používanie RTTI

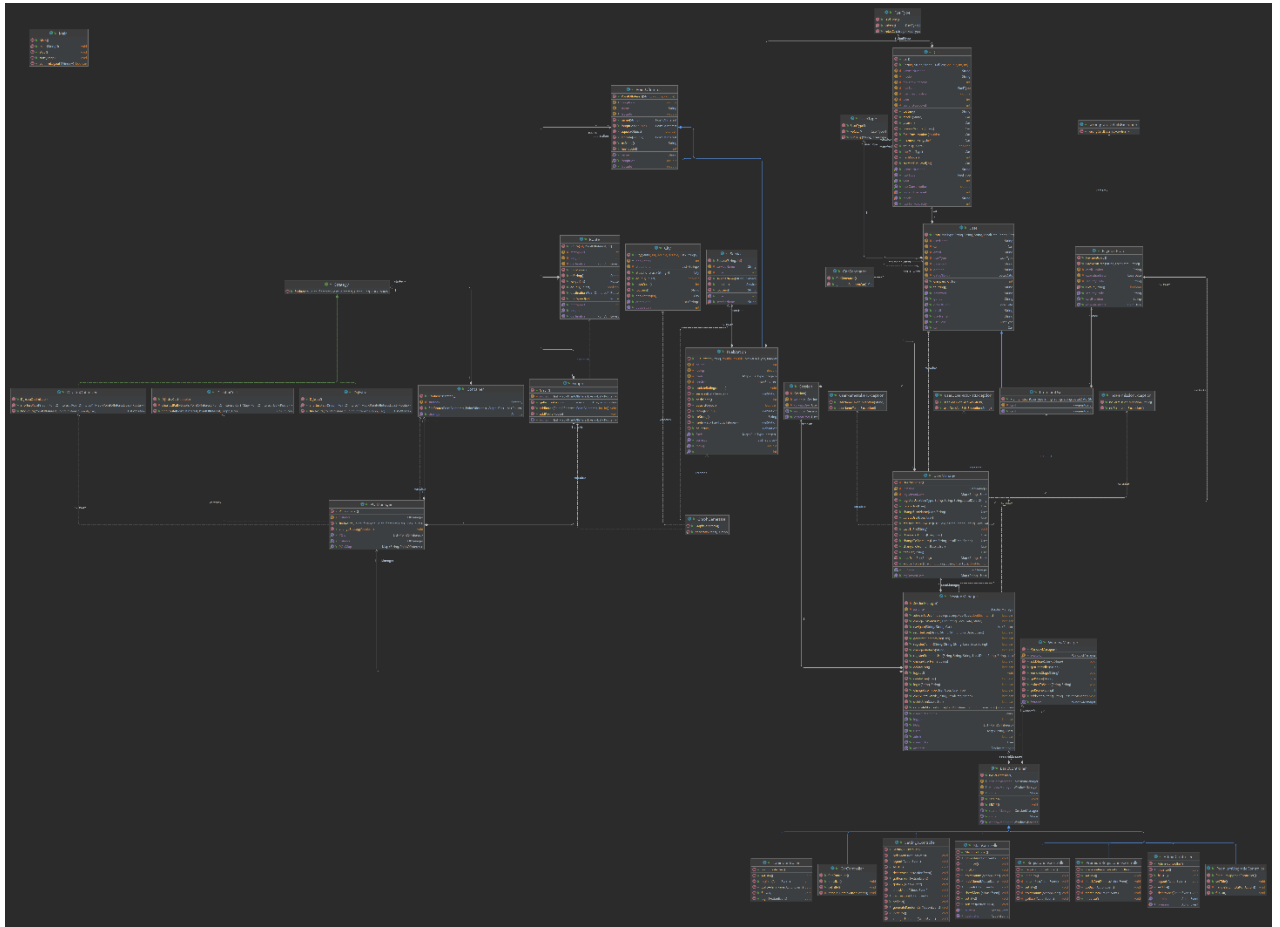
Typ objektu sa zisťuje až počas behu pomocou instance of.

```

/**
 * returns a list of all POIs sorted by fuel stations first
 * also has a comparator to sort the list by fuel stations first
 *
 * @return list of all POIs sorted by fuel stations first
 */
public List<PointOfInterest> getPOIs() {
    Comparator<PointOfInterest> poiComparator = new Comparator<PointOfInterest>() {
        @Override
        public int compare(PointOfInterest poi1, PointOfInterest poi2) {
            if (poi1 instanceof FuelStation && !(poi2 instanceof FuelStation)) {
                return -1;
            } else if (!(poi1 instanceof FuelStation) && poi2 instanceof FuelStation) {
                return 1;
            } else {
                return 0;
            }
        }
    };
    List<PointOfInterest> sortedPOIs = poiManager.getPOIs();
    sortedPOIs.sort(poiComparator);
    return sortedPOIs;
}

```

# Diagram tried



V programe je použité zapuzdrenie, dedenie, prekonávanie metód, komentár pri všetkých funkciách a ďalšie. Navyše spĺňa už vyššie spomenuté ďalšie kritéria.