The Traveler

Martin Horský

22. apríla 2023

Obsah

1 Úvod

1.1 Cieľ dokumentácie

Táto dokumentácia má za cieľ poskytnúť podrobný opis návrhu, implementácie a používania projektu. Slúži ako zdroj informácií pre mňa ako vývojára projektu, ako aj pre užívateľov, ktorí chcú pochopiť, ako aplikácia funguje a ako ju používať. Dokumentácia tiež zohľadňuje špecifické kritériá zadania, ako sú použitie návrhových vzorov, vlastné výnimky, grafické používateľské rozhranie a ďalšie pokročilé programátorské koncepty.

1.2 Popis projektu

Projekt je softvérová aplikácia navrhnutá na zobrazovanie možností cestovania, ktoré sú poskytnuté inými používateľmi. Aplikácia umožňuje užívateľom vyhľadávať, porovnávať a hodnotiť rôzne možnosti, ako sú reštaurácie, zážitky a obchody. Aplikácia poskytuje prostredie, kde užívatelia môžu zdieľať svoje názory a vytvárať komunitu záujemcov o objavovanie nových zážitkov a miest na návštevu.

1.3 Cieľová skupina

Cieľovou skupinou tejto aplikácie sú ľudia, ktorí majú záujem o objavovanie nových miest, ako sú reštaurácie, zážitky a obchody, a chcú zdieľať svoje názory a skúsenosti s ostatnými užívateľmi. Aplikácia je navrhnutá tak, aby bola jednoduchá a intuitívna pre použitie a aby zodpovedala potrebám širokej škály užívateľov. Zaujme hlavne:

- **Turistickej komunity:** Aplikácia je užitočná pre turistov, ktorí navštevujú nové mestá alebo oblasti a chcú získať odporúčania na miesta, ktoré stoja za návštevu.
- **Miestnych obyvateľov:** Aplikácia môže byť tiež užitočná pre miestnych obyvateľov, ktorí chcú objaviť nové miesta vo svojom okolí alebo hľadajú inšpiráciu pre zábavu a voľný čas.
- **Vlastníkov nehnuteľností:** Vlastníci nehnuteľností môžu pridávať svoje miesta do aplikácie a získavať tak väčšiu viditeľnosť a zvýšiť záujem potenciálnych zákazníkov.

2 Analýza a návrh

V tejto časti sa budeme zaoberať analýzou a návrhom aplikácie vrátane procesu identifikácie kľúčových funkcií, celkovej architektúry a vzťahov medzi triedami.

2.1 Identifikácia kľúčových funkcií

Pri návrhu aplikácie sme zvážili nasledujúce hlavné funkcie:

- Zobrazenie reštaurácií, zážitkov a obchodov podľa používateľových preferencií a polohy.
- Hodnotenie a recenzie reštaurácií, zážitkov a obchodov od používateľov.
- Správa a sprístupnenie zážitkov, reštaurácií a obchodov pre majiteľov nehnuteľností.
- Vyhľadávanie a filtrovanie záznamov podľa rôznych kritérií.

2.2 Architektúra

Aplikácia je navrhnutá s využitím objektovo orientovaného programovania (OOP) a nasleduje zásady zapuzdrenia, dedenia, polymorfizmu a agregácie. Architektúra aplikácie je rozdelená do nasledujúcich vrstiev:

- Vrstva prezentácie (GUI): Táto vrstva je zodpovedná za interakciu s užívateľom, teda za zobrazenie dát a prijímanie vstupov od užívateľa. Obsahuje grafické komponenty a logiku potrebnú na spracovanie udalostí generovaných užívateľom, ako sú kliknutia na tlačidlá, výber možností atď. Vrstva prezentácie je navrhnutá tak, aby bola oddelená od aplikačnej logiky, čo zabezpečuje väčšiu modularitu a znovupoužiteľnosť kódu.
- Vrstva aplikačnej logiky: Táto vrstva obsahuje triedy a metódy, ktoré implementujú hlavnú funkcionalitu aplikácie. Zabezpečuje spracovanie dát, manipuláciu s objektami a komunikáciu s databázovou vrstvou. Aplikačná logika je navrhnutá tak, aby bola nezávislá od konkrétneho grafického rozhrania a mohla byť použitá s rôznymi typmi užívateľských rozhraní alebo služieb..

- Vrstva údajov: Vrstva údajov je zodpovedná za správu a ukladanie údajov aplikácie. Zabezpečuje komunikáciu s databázou a obsahuje triedy a metódy pre prácu s údajmi, ako sú čítanie, zápis, aktualizácia a mazanie záznamov. Táto vrstva je navrhnutá tak, aby bola schopná pracovať s rôznymi typmi databáz a úložísk údajov, čo umožňuje ľahkú zmenu databázového systému v prípade potreby.

3 Splnenie bonusových kritérií zadania

3.1 Návrhové vzory

3.1.1 Template

Template design pattern je návrhový vzor, ktorý definuje kostru algoritmu v abstraktnej triede a umožňuje potomkom prekryť konkrétne kroky algoritmu bez zmeny jeho štruktúry. Tento vzor sa skladá z abstraktnej triedy, konkrétnych tried potomkov a metódy vzoru, ktorá volá abstraktné metódy implementované v triedach potomkov. V tomto programe je jeho zástupcom abstraktná trieda User a jej podtriedy.

3.1.2 Factory

Factory design pattern je návrhový vzor, ktorý umožňuje vytvárať objekty bez potreby znalosti ich konkrétnej implementácie. Tento vzor sa skladá z továrne (v tomto programe triedy UserMap a Venues), ktorá vytvára objekty, a produktov (objekty typov User, Venue a ich podtried), ktoré sú vytvárané továrňou. Továreň obsahuje metódu, ktorá vytvára produkty na základe vstupných parametrov. Týmto spôsobom sa oddeluje vytváranie objektov od ich použitia a umožňuje sa ľahšia údržba kódu a znovupoužiteľnosť. Factory design pattern sa často používa v situáciách, kedy vytváranie objektov vyžaduje zložité procesy, alebo ak chceme vytvárať objekty dynamicky na základe požiadaviek používateľa.

3.1.3 Facade

Facade design pattern je návrhový vzor, ktorý poskytuje jednoduché rozhranie pre zložité systémy, čím zjednodušuje ich používanie a zlepšuje ich prehľadnosť. Tento vzor sa skladá z fasády, ktorá predstavuje jednoduché rozhranie pre zložitý systém, a z rôznych tried, ktoré tvoria zložitý systém a sú skryté za fasádou. Fasáda poskytuje metódy, ktoré zjednodušujú používanie systému tým, že skrývajú jeho zložitosť a detaily. V tomto programe sa knižnica Gson dá považovať za využitie designového vzoru typu Facade. Gson poskytuje jednoduché rozhranie pre spracovanie JSON dát a skrýva zložitosť spracovania a parsovania týchto dát. Týmto spôsobom sa zlepšuje prehľadnosť a zjednodušuje používanie knižnice. Gson využíva množstvo rôznych tried a procesov na spracovanie JSON dát, ale vďaka fasáde sa používanie knižnice stáva jednoduchým a zrozumiteľným.

3.1.4 Decorator

V tomto programe je aplikovateľný aj návrhový vzor Decorator.Je to návrhový vzor, ktorý umožňuje dynamicky pridávať funkcionalitu objektom. Tento vzor sa skladá z dekorátorov, ktoré implementujú rovnaké rozhranie ako objekty, ktoré dekorujú. Každý dekorátor má referenciu na dekorovaný objekt a môže k nemu pridávať funkcionalitu. V kontexte tohto programu, by mohol Admin pridelovať používateľom typu Casual meniť povolenia (atribút permissions), čím by mu umožňoval vykonávať iné funkcie.

3.2 Serializácia

Serializácia je proces prevodu objektu alebo dátového typu na reťazec, ktorý môže byť neskôr prenesený alebo uložený. Tento reťazec môže byť potom obnovený späť na objekt alebo dátový typ pomocou procesu deserializácie. V tomto programe je riešená pomocou metód load a upload v triedach implementujúce vlastné rozhranie ObjectMap. V prípade triedy Venues, pri ukladaní dát, táto metóda rozdelí Venues do 3 súborov, každý pre 1 podtriedu triedy venues a uloží ich pomocou gson knižnice do json súborov:

```
public synchronized void upload(String filePath){
        synchronized (venueThreadLock){
            upload(filePath + "/Shop.json", venueMap.get("Shop"));
            upload(filePath + "/Restaurant.json", venueMap.get("Restaurant"));
            upload(filePath + "/Experience.json", venueMap.get("Experience"));
        }
    }

    private void upload(String filePath, HashMap<Long, Venue> venueMap) {
        Gson gson = new Gson();
        String jsonString = gson.toJson(venueMap);
```

```
        try (FileWriter fileWriter = new FileWriter(filePath)) {
            fileWriter.write(jsonString);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Funkcia load načíta dáta z rôznych súborov a uloží ich do spoločnej hashovacej tabuľky:

```
public synchronized void load(String filePath){
        synchronized (venueThreadLock){
            venueMap = new HashMap<>();
            venueMap.put("Shop", new HashMap<>());
            venueMap.put("Restaurant", new HashMap<>());
            venueMap.put("Experience", new HashMap<>());
            try {venueMap.get("Shop").putAll(load(filePath + "/Shop.json", Shop.class));
            } catch (NullPointerException ignored){}
            try {venueMap.get("Restaurant").putAll(load(filePath + "/Restaurant.json", Restaurant.class));
            } catch (NullPointerException ignored){}
            try {venueMap.get("Experience").putAll(load(filePath + "/Experience.json", Experience.class));
            } catch (NullPointerException ignored){}
        }
    }

    private HashMap<Long, ? extends Venue> load(String filePath, Class<? extends Venue> venueType) {
        Gson gson = new Gson();
        Type venueMapType = TypeToken.getParameterized(HashMap.class, Long.class, venueType).getType();
        try (FileReader fileReader = new FileReader(filePath)) {
            HashMap<Long, ? extends Venue> temp = gson.fromJson(fileReader, venueMapType);
            return temp;
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }
```

Pri triede userMap tieto funkcie pracujú rovnako.

3.3 Vlastné výnimky

Funkcie load a upload taktiež využívajú vlastné výnimky. Zachytávajú prípady, kedy súbor neexistuje a vrátia null, alebo v prípade funkcie load, tiež zachytáva ak jej príde null a ignoruje to, namiesto vyhodenia chyby. (ukážka kódu vyššie) Vlastná výnimka je využitá aj v triede AppScreen, kde vypíše užívateľovi upozornenie v prípade nesprávne zadaného vstupu.

```
try {
                long selectedVenueId = Long.parseLong(venueId.getText());
                inspectVenueWindow(selectedVenueType, selectedVenueId);
            } catch (NullPointerException e ) {
                venueTextArea.setText("Pay closer attention to available venue IDs!");
            } catch (NumberFormatException e) {
                venueTextArea.setText("Browse venues again and pick one of the IDs!");
            }
```

3.4 Multi-threading

Táto vlastnosť je opäť využitá pri funkciách load a upload.

```
    private void uploadUsers(){
        new Thread(() -> {
            users.upload("src/main/resources/users");
            System.out.println("Users uploaded");
```

```
        }).start();
    }

    private void loadUsers(){
        new Thread(() -> {
                users.load("src/main/resources/users"); // make sure to have valid starting json file
                System.out.println("Users loaded");
            }).start();
    }

    private void uploadVenues(){
        new Thread(() -> {
            venues.upload("src/main/resources/venues");
            System.out.println("Venues uploaded");
        }).start();
    }

    private void loadVenues(){
        new Thread(() -> {
            venues.load("src/main/resources/venues");
            System.out.println("Venues loaded");
        }).start();
    }
```

Spúšťajú sa na vedľajších vláknach pri spustení a vypnutí prihlasovacej a hlavnej obrazovky. Využíva kľúčové slovo synchronized, vďaka ktorému sa nemôže načítavať do toho istého súboru naraz. Taktiež využíva synchronizáciu na pomocou zámku, ktorá synchronizuje funkcie load a upload (ukážka kódu vyššie).

3.5 Implicitná implementácia metód

Opäť sa vzťahuje na metódy load a upload. Tieto metódy sú implicitne zadefinované v rozhraní ObjectMap. V prípade, že triedy implementujúce toto rozhranie danú triedu vynechali, táto metóda len vypíše, že je potrebné načítať a uložiť dáta manuálne, nakoľko to daná trieda nepodporuje.

```
public interface ObjectMap {
    void upload(String filePath);
    void load(String filePath);

    default void upload(Object o) {
        System.out.println("Upload this object manually: " + o);
    }

    default void load(Object o) {
        System.out.println("Load this object manually: " + o);
    }
}
```

3.6 Override metód

Väčšina tried má preťaženú toString metódu zdedenú z triedy Object. Okrem toho je preťažená metóda napr. v triede Professional.

```
    @Override
    public void rateVenue(Venue venue, float rating){
        venuesRated++;
        venue.incrementRatingCount(); venue.incrementRatingCount();
        var previousRating = venue.getRating();
        venue.setRating(((previousRating == null ? 0 : previousRating) * (venue.getRatingCount() - 2) + 2*rat
    }
```

3.7 Genericksť

Genericksť je využitá v triede Shop vo vnútornej triede ShopItem reprezentujúca vec dostupnú v obchode.

```
private class ShopItem<C> {
        private String name;
        private C content;

        public ShopItem(String name, C content) {
            this.name = name;
            this.content = content;
        }

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        public C getContent() {
            return content;
        }

        public void setContent(C content) {
            this.content = content;
        }
    }
```

3.8 Oddelené grafické prostredie

Grafické prostredie je implementované oddelene v balíku gui. Funkcionalita grafického prostredia je oddelená od funkcionality tried. Implementácia je podrobnejšie opísaná nižšie v sekcii Implementácia.

3.9 Explicitné použitie RTTI

RTTI (Run-Time Type Information) je mechanizmus, ktorý umožňuje získať informácie o triede objektu za behu programu. V triede AppScreen som použil RTTI na identifikáciu tpyu momentálneho užívateľa a na základe tejto informácie sa zobrazujú rôzne button.

```
if (user instanceof Traveler) {
            Button pathManagerButton = new Button("Path Manager");
            pathManagerButton.setOnAction(event -> {
                inspectPathsWindow();
            });
            buttonLayout.getChildren().add(pathManagerButton);
        }

        if (user instanceof Admin) {
            Button adminButton = new Button("Administration");
            adminButton.setOnAction(event -> {
                Stage adminStage = new Stage();
                VBox adminLayout = createAdminWindow();
                Scene adminScene = new Scene(adminLayout, 500, 600);
                adminStage.setTitle("Administration");
                adminStage.setScene(adminScene);
                adminStage.show();
            });
            buttonLayout.getChildren().add(adminButton);
        }
```

3.10 Vhniezdené triedy

V triede Traveler, je definovaná vnútorná trieda Path. Trieda Path slúži na uchovávanie ciest medzi objektmi typu Venue, ktoré sú záujmovými bodmi pre cestovateľa. Užívateľ môže tieto cesty vytvárať, a neskôr si môže pozerať sebou vytvorené cesty.

```
private class Path{
     private String name;
     private ArrayList<Venue> venues = new ArrayList<>();

     public void addToPath(Venue property){
         venues.add(property);
     }

     public void setName(String name) {
         this.name = name;
     }

     public String getName(){
         return name;
     }

     public void removeFromPath(Venue property){ venues.remove(property); }

     @Override
     public String toString() {
         String output = "";
         for (var venue: venues){
             output += "->" + venue.getName();
         }
         return output;
     }
 }
```

3.11 Využitie lambda výrazov

Metóda verifyPasswordRegister triedy UserMap využíva lambda výraz na overenie, či heslo obsahuje aspoň 3 zo 4 kritérií (obsahuje malé písmeno, veľké písmeno, číslo, špeciálny znak).

```
private boolean verifyPasswordRegister(String password){
     return getPasswordComplexity.apply(password) >= 3;
 }

 private static Function<String, Integer> getPasswordComplexity = (password) -> {
     int output = 0;
     if (password.length() < 4 || password.length() > 30) return -1;
     boolean containsLowerCase = false;
     boolean containsUpperCase = false;
     boolean containsNumber = false;
     boolean containsSpecialChar = false;
     for (char i: password.toCharArray()){
         if (i >= 'a' && i <= 'z') containsLowerCase = true;
         else if (i >= 'A' && i <= 'Z') containsUpperCase = true;
         else if (i >= '0' && i <= '9') containsNumber = true;
         else containsSpecialChar = true;
     }
     output += (containsLowerCase?1:0) + (containsUpperCase?1:0) +
         (containsNumber?1:0) + (containsSpecialChar?1:0);
     return output;
 };
```

4 Implementácia

4.1 Knižnice použité v projekte

Pri projekte boli okrem built-in knižníc využívané knižnice javaFX a Gson. JavaFX je moderná knižnica pre tvorbu grafických používateľských rozhraní. Poskytuje sadu komponentov a nástrojov pre efektívne a príjemné dizajnovanie a programovanie GUI.

Gson je populárna knižnica pre prácu s formátom JSON, ktorú vyvinul google. Umožňuje relatívne ľahko prevádzať objekty Java na reťazce JSON a naopak. Najzákladnejšie funkcie Gson zahŕňajú serializácia a deserializácia objektov do a z json súborov.

4.2 štruktúra

Projekt je rozdelený do niekoľkých balíkov a tried, aby bol kód zrozumiteľný, organizovaný a ľahko rozšíriteľný. Hlavné balíky a triedy sú nasledovné:

4.2.1 GUI

Tento balík je hlavný balík a obsahuje triedu Main, ktorá spúšťa celú aplikáciu prostredníctvom JavaFX. Táto trieda inicializuje hlavné okno a načítava základné grafické užívateľské rozhranie. Ďalej obsahuje triedy súvisiace s GUI a ovládačmi pre rôzne obrazovky aplikácie. Tieto triedy zahŕňajú LoginScreen a AppScreen.

4.2.2 Users

Tento balík obsahuje triedy zodpovedajúce rôznym typom užívateľov, ako sú Traveler (a jej podtriedy Casual a Professional), PropertyOwner a Admin. Tieto triedy rozširujú základnú triedu User a poskytujú špecifické metódy a atribúty pre každý typ užívateľa.

4.2.3 Venue data

Tento balík obsahuje triedy reprezentujúce rôzne miesta a zážitky, ako sú Shop, Restaurant a Experience. Všetky tieto triedy rozširujú základnú triedu Venue, ktorá poskytuje spoločné atribúty a metódy pre všetky miesta.

4.2.4 Data storage

Balík Data Storage obsahuje metódy uskladňujúce iné typy metód, ktoré taktiež ukladajú a načítavajú súbory a údaje.

4.3 Popis tried a ich funkcionalita

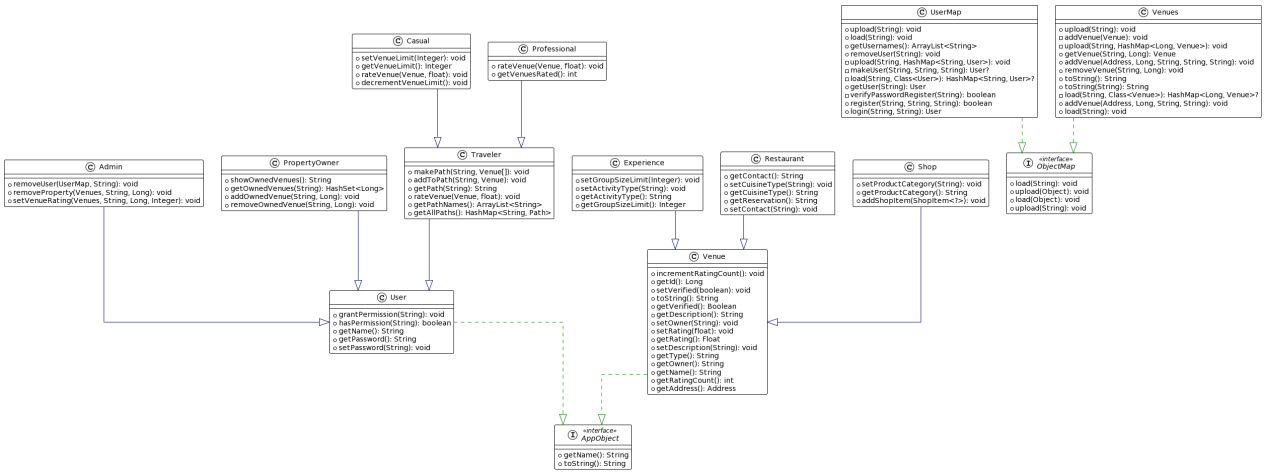

Diagram tried

Funkčné triedy Hlavné funkcionality aplikácie spočívajú v triedach pre používateľov a triedach pre podniky. Základná abstraktná trieda pre užíateľov aj pre podinky rozširuje rozhranie AppObject, ktoré určuje, že tieto triedy musia mať prepísanú toString metódu a musia mať metódu ktorá vracia ich meno. public interface AppObject String toString(); String getName(); **Užívateľské triedy** Obsahujú užívateľské funkcionality, aké údaje môžu obsahovať a ich vnútorné triedy. Základom je abstraktná trieda User. Definuje, že každý používateľ musí mať meno, heslo, set povolení s pravdivostnou hodnotou a ku všetkému príslušné set-ery a get-ery poskytujúce enkapsuláciu.

```
public abstract class User implements AppObject {
public User(String name, String password){
    ...
```

```
}

private String name, password;
private HashMap<String, Boolean> permissions;

...

}
```

Podtrieda Admin túto triedu rozširuje o metódy na vymazanie užívateľa a podniku ako aj na prepísanie hodnotenia podniku.

```
public void removeUser(UserMap users, String name){
    users.removeUser(name);
}

public void removeProperty(Venues venues, String type, Long id){
    venues.removeVenue(type, id);
}

public void setVenueRating(Venues venues, String type, Long id, Integer rating){
    venues.getVenue(type, id).setRating(rating);
}
```

Podtrieda PropertyOwner rozširuje triedu User o to, že pri majiteľovi podnikov sa uchovávajú jeho podniky a obsahuje k tomu príslušné metódy.

```
private HashMap<String, HashSet<Long>> ownedVenues = new HashMap<>();

public void addOwnedVenue(String type, Long id){
    ...
}

public HashSet<Long> getOwnedVenues(String type){
    ...
}

public String showOwnedVenues() {
    ...
}

public void removeOwnedVenue(String type, Long id){
    ...
}
```

Podtrieda Traveler obsahuje vlastnú podtriedu cesty, ktorá slúži na uchovávanie postupností podnikov, aké chce používateľ navštíviť.

Podtriedy tejto podtriedy Casual a Professional majú len mierne zmeny funkcionality. Casual používateľ má limit koľko podnikov môže pridať. Keď ho minie, môže pridať ďalší podnik až po ohodnotení nejakého podniku. Professional používateľ má metódu hodnotenia prepísanú tak, aby jeho hodnotenie malo vyššiu váhu.

Triedy podnikov Základom týchto tried je abstraktná trieda Venue. Obsahuje atribúty pre adresu, id, tyo, meno, popis, majiteľa, či sú verifikované, momentálne hodnotenie a počet hodnotení. Počet hodnotení sa využíva pri výpočte ako sa hodnotenie mení po pridaní nového hodnotenia. Obsahuje set-ery a get-ery (okrem set-erov pre triedy, ktoré sa už nemajú meniť a na počet hodnotení, ktoré sa môže len inkrementovať).

Podtriedy Restaurant, Shop a Experience obsahujú ďalšie špeciálne atribúty, ktoré poskytujú dodatočné informácie o podniku. Okrem toho Shop obsahuje aj vnútornú generickú triedu ShopItem, ktorá však zatiaľ nie je implementovaná v aplikácii, nakoľko nie je užitočná bez užívateľskej angažovanosti, ale pri raste aplikácie by bola táto trieda vysoko užitočná.

Triedy na pracovanie s dátami Najjednoduchšou s týchto tried je IdManager, ktorý slúži len na uchovávanie momentálneho počtu podnikov do textového súboru. Následujúce ID bude vždy počet podnikov (zahrňujúci aj podniky, ktoré už boli vymazané) + 1.

Ďalšie triedy sú implementácie rozhrania ObjectMap, ktoré požaduje načítavanie a ukladanie údajov.

Je tam trieda UserMap, ktorá slúži na vytváranie, ukladanie a načítavanie užívateľov. Užívateľov ukladá pomocou metód Load a Upload opísaných vyššie. Kľúčovým atribútom tejto triedy je hashovacia tabuľka obsahujúca všetkých užívateľov.

```
private HashMap<String, User> userMap;
```

Pri registrácii užívateľa kontroluje, či už užívateľ s rovnakým menom existuje a tiež kontroluje zložitosť hesla (podrobnejšie vyššie pri ukážke lambda metód):

```
public boolean register(String name, String password, String type){
    if (!verifyPasswordRegister(password) || userMap.containsKey(name)) return false;
    userMap.put(name, makeUser(name, password, type));
    return true;
}
```

Pri prihlasovaní používateľa kontroluje, či užívateľ existuje a ak áno, či sa jeho heslo zhoduje so zadaným heslom.

```
public User login(String name, String password){
    if (!userMap.containsKey(name) || !userMap.get(name).getPassword().equals(password)){
        return null;
    }
    return userMap.get(name);
}
```

Posledná trieda na prácu s údajmi je trieda Venues. Slúži na vytváranie podnikov, ich uchovávanie a ich ukladanie a načítavanie so súborov. Funkcia na pridanie podniku je 3x preťažená. Podnik sa dá pridať pomocou zadania údajov potrebných pre vytvorenie podniku, tým istým, s tým, že je tam aj popis, alebo vložením rovno vytvoreného podniku.

```
public void addVenue(Universal.Address address, Long id, String name, String type){
    ...
}

public void addVenue(Universal.Address address, Long id, String name, String type, String description){
    ...
}

private void addVenue(Venue venue){
    ...
}
```

Grafické prostredie

4.3.1 Main

Spúšťa login screen

```
public static void main(String[] args) {
    Application.launch(LoginScreen.class, args);
}
```

4.3.2 LoginScreen

Trieda LoginScreen je zodpovedná za zobrazenie prihlasovacej obrazovky a spracovanie používateľských údajov. Na začiatku funkcia zavolá userMap, a načíta existujúcich užívateľov.

```
users = new UserMap();
new Thread(() -> {
    users.load("src/main/resources/users"); // make sure to have valid starting json file
    System.out.println("Users loaded");
}).start();
```

Ďalej vytvára hlavné okno.

```
VBox mainLayout = new VBox(10);
mainLayout.setAlignment(Pos.CENTER);
HBox buttonsLayout = new HBox(30);
buttonsLayout.setAlignment(Pos.CENTER);
systemMessage = new Label();
systemMessage.setText("Welcome to Traveler!");
```

Nachádzajú sa tam dve primárne tlačídlá. Login a Register zviditeľňujú respektívne prihlasovaciu a registrovaciu obrazovku a vypínajú viditeľnosť tej druhej.

```
Button loginButton = new Button("Log In");
Button registerButton = new Button("Register");

...

loginButton.setOnAction(event -> {
        loginLayout.setVisible(true);
        registrationLayout.setVisible(false);
    });

registerButton.setOnAction(event -> {
    loginLayout.setVisible(false);
    registrationLayout.setVisible(true);
});
```

Fukncia na vytváranie prihlasovacieho vytvorí dve textové polia na meno a heslo a vytvorí tlačidlo na odoslanie. Pri stlačení tlačidla enter sa zmení focus na následujúce okno. V prípade nesprávnych údajov, program vypíše problém do systemMesage (label vytvorený predtým).

```
TextField usernameField = new TextField();
usernameField.setPromptText("Username");

PasswordField passwordField = new PasswordField();
passwordField.setPromptText("Password");

Button submitLoginButton = new Button("Submit");

usernameField.setOnAction(event ->
    passwordField.requestFocus());
passwordField.setOnAction(event ->
    submitLoginButton.fire());

submitLoginButton.setOnAction(event -> {
    var newUser = users.login(usernameField.getText(), passwordField.getText());
    usernameField.clear();
    passwordField.clear();
    if (newUser != null) {
        systemMessage.setText("Login successful");
        startApp(newUser);
        primaryStage.close();
    } else {
        systemMessage.setText("Login failed");
    }
});
```

Registračné okno okrem prihlasovacieho mena a hesla taktiež vytvorí radio buttony, pomocou ktorej si užívateľ vyberie aký typ účtu si registruje. Automaticky je to nastavené na typ Casual.

```
RadioButton radioButton1 = new RadioButton("Casual");
    RadioButton radioButton2 = new RadioButton("Professional");
    RadioButton radioButton3 = new RadioButton("Property Owner");
    RadioButton radioButton4 = new RadioButton("Admin");

    radioButton1.setSelected(true);

    ToggleGroup toggleGroup = new ToggleGroup();
    radioButton1.setToggleGroup(toggleGroup);
    radioButton2.setToggleGroup(toggleGroup);
```

```
radioButton3.setToggleGroup(toggleGroup);
radioButton4.setToggleGroup(toggleGroup);

toggleGroup.getSelectedToggle();
```

Registrácia, rovnako ako prihlasovanie, funguje pomocou triedy userMap. Ak userMap neuzná prihlasovacie údaje (buď už existuje prihlasovacie meno, alebo nie je dostatočne komplikované heslo)

```
submitRegisterButton.setOnAction(event -> {
        if (!users.register(usernameField.getText(),
                passwordField.getText(),
                ((RadioButton)toggleGroup.getSelectedToggle()).getText()))
        {systemMessage.setText("Registration unsuccessful");}
        else {systemMessage.setText("Registration successful");}
        usernameField.clear();
        passwordField.clear();
    });
```

Po úspešnom prihlásení sa zapne aplikačné okno (na tej istej niti, keďže JavaFX nepodporuje multithreading):

```
private void startApp(users.User user) {
    Platform.runLater(() -> {
        AppScreen appScreen = new AppScreen(user, users);
        appScreen.start(new Stage());
    });
}
```

4.3.3 AppScreen

Trieda AppScreen je zodpovedná za zobrazenie hlavnej obrazovky aplikácie a jej rôzne funkcie. Táto trieda rozširuje triedu Application z JavaFX.

Aplikácia vezme ako parameter z prihlasovacieho okna momentálneho používateľa ako aj mapu (triedy UserMap) všetkých používateľov.

```
public AppScreen(User user, UserMap users) {
    this.user = user; this.users = users;
}
```

V hlavnej metóde start sa inicializuje hlavné rozvrhnutie obrazovky. Všeobecne sú tam tlačidlá na vytvorenie podniku, zobrazenie podnikov (k čomu patrí dropdown menu, kde je možné vybrať si typ zobrazených podnikov), detailné zobrazenie konkrétneho podniku (k čomu takisto patrí dropdown a aj textové pole, kde sa zadáva id podniku) a na odhlásenie. Taktiež tam patrí okno, na zobrazovanie textu systémom.

```
Button logoutButton = new Button("Log Out");
    logoutButton.setOnAction(event -> {
        ...
    });
    Button createVenueButton = new Button("Create Venue");
    createVenueButton.setOnAction(event -> {
        ...
    });

    Button browseButton = new Button("Browse Venues");
    ComboBox<String> venueBrowseDropdown = new ComboBox<>(FXCollections.observableArrayList("All", "Resta
    venueBrowseDropdown.getSelectionModel().selectFirst();

    TextArea venueTextArea = new TextArea();
    venueTextArea.setEditable(false);
    ScrollPane scrollPane = new ScrollPane(venueTextArea);
    scrollPane.setFitToWidth(true);
    scrollPane.setFitToHeight(true);
```

Ďalej sú zobrazené rôzne funkcionality aplikácie podľa typu príhlaseného používateľa. Adminovi sa ukáže administrátorské tlačidlo a cestovateľom sa ukáže tlačidlo na nastavovanie ciest.

```
if (user instanceof Traveler) {
        Button pathManagerButton = new Button("Path Manager");
        pathManagerButton.setOnAction(event -> {
            inspectPathsWindow();
        });
        buttonLayout.getChildren().add(pathManagerButton);
    }

    if (user instanceof Admin) {
        Button adminButton = new Button("Administration");
        adminButton.setOnAction(event -> {
            Stage adminStage = new Stage();
            VBox adminLayout = createAdminWindow();
            Scene adminScene = new Scene(adminLayout, 500, 600);
            adminStage.setTitle("Administration");
            adminStage.setScene(adminScene);
            adminStage.show();
        });
        buttonLayout.getChildren().add(adminButton);
    }
```

Na vytvorenie jednotlivých okien, umožňujúcich určité funkcionality programu, sú v tejto triede samostatné metódy.

Na vytvorenie okna na pridanie nového miesta slúži metóda createVenueForm. Tá obsahuje label s názvom:

```
Label systemMessage = new Label();
    systemMessage.setText("Create Venue");
```

Textové polia pre názov podniku a jeho adresu (mesto, ulicu, číslo ulice):

```
TextField nameField = new TextField();
    nameField.setPromptText("Name");

    TextField cityField = new TextField();
    cityField.setPromptText("City");

    TextField streetField = new TextField();
    streetField.setPromptText("Street");

    TextField streetNumberField = new TextField();
    streetNumberField.setPromptText("Street Number");
```

tlačidlá na výber typu podniku:

```
RadioButton radioButton1 = new RadioButton("Experience");
RadioButton radioButton2 = new RadioButton("Restaurant");
RadioButton radioButton3 = new RadioButton("Shop");
...
```

a miesto na popis:

```
TextArea descriptionField = new TextArea();
descriptionField.setPromptText("Description");
```

Pre používateľov, ktorí sa zaregistrovali ako majiteľ podniku sa tiež zobrazí checkbox, kde môžu zaškrtnúť, či tento podnik vlastnia. Ak áno, zobraziasa im na vrchu obrazovky taktiež ďalšie polia, kam môžu doplniť dodatočné informácie. To aké polia sa zobrazí na základe typu podniku.

```
CheckBox ownershipCheckBox = new CheckBox("I own this property");
venueLayout.getChildren().add(ownershipCheckBox);
ownershipCheckBox.setVisible(false);
```

```
VBox ExperienceInfoLayout = new VBox(10);
...

VBox RestaurantInfoLayout = new VBox(10);
...

VBox ShopInfoLayout = new VBox(10);
...

if (user instanceof PropertyOwner){
        ownershipCheckBox.setVisible(true);

        ownershipCheckBox.setOnAction(e -> {
            ...
        });
    }
```

Na zobrazenie podrobných informácií o jednom podniku slúži metóda inspectVenueWindow. V tomto okne sa nachádza
- názov podniku:

```
Label venueTitle = new Label(venue.getName());
venueTitle.setFont(Font.font("Arial", FontWeight.BOLD, 14));
```

- popis podniku:

```
Label descriptionLabel = new Label(venue.getDescription());
```

- ďalšie údaje o podniku na základe typu podniku:

```
if (venue instanceof Restaurant) {
        Restaurant restaurant = (Restaurant) venue;
        Label contactLabel = new Label(restaurant.getContact() != null ? "Contact: " + restaurant.getCont
        Label cuisineTypeLabel = new Label(restaurant.getCuisineType() != null ? "Cuisine type: " + resta

        venueLayout.getChildren().addAll(contactLabel, cuisineTypeLabel);
    } else if (venue instanceof Experience) {
        Experience experience = (Experience) venue;
        Label activityTypeLabel = new Label(experience.getActivityType() != null ? "Activity type: " + ex
        Label groupSizeLimitLabel = new Label(experience.getGroupSizeLimit() != null ? "Group size limit:

        venueLayout.getChildren().addAll(activityTypeLabel, groupSizeLimitLabel);
    } else if (venue instanceof Shop) {
        Shop shop = (Shop) venue;
        Label productCategoryLabel = new Label(shop.getProductCategory() != null ? "Product category: " +

        venueLayout.getChildren().add(productCategoryLabel);
    }
```

- tlačidlo na ohodnotenie podniku a tlačidlo na zatvorenie okna

```
Button closeButton = new Button("Close");
closeButton.setOnAction(event -> venueStage.close());

Button submitReviewButton = new Button("Submit Review");
submitReviewButton.setOnAction(event -> {
    submitReviewButton.setVisible(false);
    VBox reviewForm = createReviewForm(venue, venueStage);
    venueLayout.getChildren().add(reviewForm);
});
```

Pri hodnotení sa otvorí nové okno pomocou metódy createReviewForm. Nachádza sa tam názov podniku, 5 hviezdičiek z ktorých si užívateľ vyberie koľkými podnik ohodnotí a tlačidlo na odoslanie hodnotenia. V prípade, že užívateľ nie je typu cestovateľ, neumožní mu to hodnotenie odoslať.

```
private VBox createReviewForm(Venue venue, Stage venueStage) {
    ...

    Label systemMessage = new Label();
    systemMessage.setText("Review " + venue.getName());

    Rating rating = new Rating();

    Button submitButton = new Button("Submit");
    submitButton.setOnAction(event -> {
        if (user instanceof Traveler) {
            int ratingValue = (int) Math.round(rating.getRating());
            ((Traveler) user).rateVenue(venue, ratingValue);
            systemMessage.setText("Review submitted!");
            venueStage.close();
        }
        else {
            systemMessage.setText("Only travelers can submit reviews");
        }

    });

    ...

}
```

Ďalej sa nachádza okno sprístupnené len určitým uživateľom (pomocou RTTI). Cestovatelia môžu spravovať svoje cesty pomocou inspectPathsWindow. Toto okno slúži na zobrazenie manažéra ciest. Odtiaľ môže užívateľ pridať novú cestu alebo pozrieť svoje vytvorené cesty:

```
Button makePathButton = new Button("Make Path");
makePathButton.setOnAction(event -> {
    ...
});

Button viewPathsButton = new Button("View Paths");
viewPathsButton.setOnAction(event -> {
    ...
});
```

Na vytvorenie cesty sa spustí nové okno pomocou metódy makePathForm. Užívateľ zadá názov cesty, typ a ID podniku (ktoré môhol vidieť v hlavnom okne po stlačení Browse Venues). V prípade nesprávnych údajov sa užívateľovi vypíše varovanie. Pomocou tlačidla môže podnik vložiť.

```
...

TextField nameField = new TextField("name");
...

ComboBox<String> venueTypeDropdown = new ComboBox<>(FXCollections.observableArrayList("Restaurant", "Expe
...

TextField venueIdField = new TextField();
venueIdField.setPromptText("ID");
...

Button addButton = new Button("Add Venue");
```

Tieto podniky sa ukladajú do stacku. Keď je používateľ s cestou spokojný, môže ju odovzdať pomocou ďalšieho tlačidla:

```
...
```

```
Stack<Venue> selectedVenues = new Stack<>();
...

Button submitPathButton = new Button("Submit Path");
submitPathButton.setOnAction(event -> {
        ...
    });
```

Pomocou metódy viewPathsForm môže užívateľ svojími cestami prechádzať. Po vybratí cesty mu ju program vypíše.

```
private VBox viewPathsForm() {
    ...

    ComboBox<String> pathsDropdown = new ComboBox<>();
    pathsDropdown.setItems(FXCollections.observableArrayList(((Traveler)user).getPathNames()));

    pathsDropdown.valueProperty().addListener((observable, oldValue, newValue) -> {
        systemMessage.setText(((Traveler) user).getPath(pathsDropdown.getValue()));
    });
    ...

}
```

Posledné okno, ktoré sa nachádza v aplikácii je určené pre administrátorov (užívateľov typu Admin). Zapína sa pomocou metódy createAdminWindow.

Administrátor môže podľa používateľského mena používateľa vymazať. V prípade, že sa jedná o majiteľa podniku, zmažú sa aj všetky jeho podniky, ak nejaké má.

```
Label removeUserLabel = new Label("Remove user");
    TextField removeUserNameField = new TextField();
    removeUserNameField.setPromptText("Enter username");
    Button removeUserButton = new Button("Remove user");
    removeUserButton.setOnAction(event -> {
        var removedUser = users.getUser(removeUserNameField.getText());
        if (removedUser instanceof PropertyOwner){
            for (String type: new String[]{"Restaurant", "Experience", "Shop"}){
                for (Long id: ((PropertyOwner) removedUser).getOwnedVenues(type)){
                    venues.removeVenue(type, id);
                }
            }
        }
        ((Admin) user).removeUser(users, removeUserNameField.getText());
        systemMessage.setText("User removed.");

    });
```

Taktiež môže mazať podniky. Pri zmazaní podniku ho zmaže aj majiteľovi zo zoznamu vlastnených podnikov (v prípade, že má zadaného majiteľa).

```
removePropertyButton.setOnAction(event -> {
        var removedPropertyOwner = users.getUser(venues.getVenue(removePropertyTypeField.getText(), Long.
        if (removedPropertyOwner instanceof PropertyOwner){
            ((PropertyOwner) removedPropertyOwner).removeOwnedVenue(removePropertyTypeField.getText(), Lo
        }
        ((Admin) user).removeProperty(venues, removePropertyTypeField.getText(), Long.parseLong(removePro
        systemMessage.setText("Property removed.");
    });
```

5 Používateľský manuál

5.1 Registrácia

Používateľ si naprv musí založiť účet. Musíte zadať užívateľské meno, dlhé aspoň 4 znaky, ktoré nemá žiadny iný používateľ. Taktiež musíte pri hesle dať pozor, aby malo aspoň 4 znaky a aby aspoň 3 z týchto kritérii boli splnené: 1.obsahuje veľký znak; 2.obsahuje malý znak; 3.obsahuje číslo; 4.obsahuje špeciálny znak. V prípade, že bola registrácia neúspešná, skontrolujte, či vaše údaje spĺňajú podmienky. Ak áno, musíte zmeniť meno, lebo užívateľ s týmto prihlasovacím menom už existuje.

5.2 Používanie aplikácie

Na vytovrenie podniku stlačte príslušné tlačidlo a vyplňte informácie. Ak ste si pre registrácii zvolili typ používateľa majiteľ podniku, môžte zaškrtnúť, že ste majiteľ tohto podniku a to vytvorí nové okná, ktoré môžte vyplniť.

Ak ste si pri registrácii vybrali jeden z typov cestovateľa (Casual, Professional) pri tvorbe cesty postupujte následovne. Ak pridávate cestu, postupne zadávajte ID podniku a pridajte ho tlačidlom add(s tým, že ste vybrali správny typ podniku). Keď ste vybrali všetky podniky, ktoré chcete aby cesta obsahovala, stlačte tlačidlo submit. Na následné zobrazenie ciest zavrite momentálne okno a v okne Path manager kliknite na view Paths. Tam si cestu nájdete podľa mena, aké ste jej predtým zadali.

Ak ste užívateľ typu Casual a nedarí sa vám vytvoriť podnik, zrejme vám vypršal limit. Na to aby ste mohli vytvoriť ďalší podnik ohodnoťte ostatné podniky.