Oleksandr Oksanich - Azure Sophia's Advisor

Imagining that the current situation in Eastern Europe is a tad more peaceful, Azure Sofia's Advisor is an application that is meant to acquaint a traveler with every fascinating sight spread over the area of 840 km^2 that belongs to Kyiv—the capital city of Ukraine and the cradle of Eastern European culture. The purpose is to give insight into the city's mix of historical and modern architecture, its landmarks and monuments which have absorbed more than a thousand of years of historic events; its nature, entertainment options, cultural events, and a bunch of sights to explore for tourists of all ages for affordable prices. The app's another objective is to let its users organize and schedule a plan for the trip based on a place or a tip they are interested in.

Icon Attribution

- User icons created by kmg design - Flaticon
- Home icon by Icons8
- What I Do icon by Icons8
- Customer icon by Icons8
- Settings icon by Icons8
- Error icon by Icons8
- Approval icon by Icons8
- Info icon by Icons8
- Icon by manshagraphics on freeicons.io

Table of Contents

Fulfillment of criteria

Main criteria

- Polymorphism
  - Place.java - line #34
  - Tip.java - line #32
  - PlaceDatabase.java - lines #29, #33
  - TipDatabase.java - lines #29, #33
- Inheritance
  - Place.java - line #8
  - Tip.java - line #6
  - SelfUser.java - line #12
  - InsufficientPlanDetailException.java - line #7
- Encapsulation
  - PlaceDatabase.java - line #24
  - TipDatabase.java - line #24
  - UserCredentials.java - line #13
  - Place.java - line #9
  - Plan.java - line #9
  - PlanBuilder.java - line #9
  - SelfUser.java - line #13
  - Tip.java - line #7

    - User.java - line #11
    - MainApplication.java - line #14
- Aggregation
    - Place.java - line #75
    - Plan.java - line #77
    - Version.java - line #50

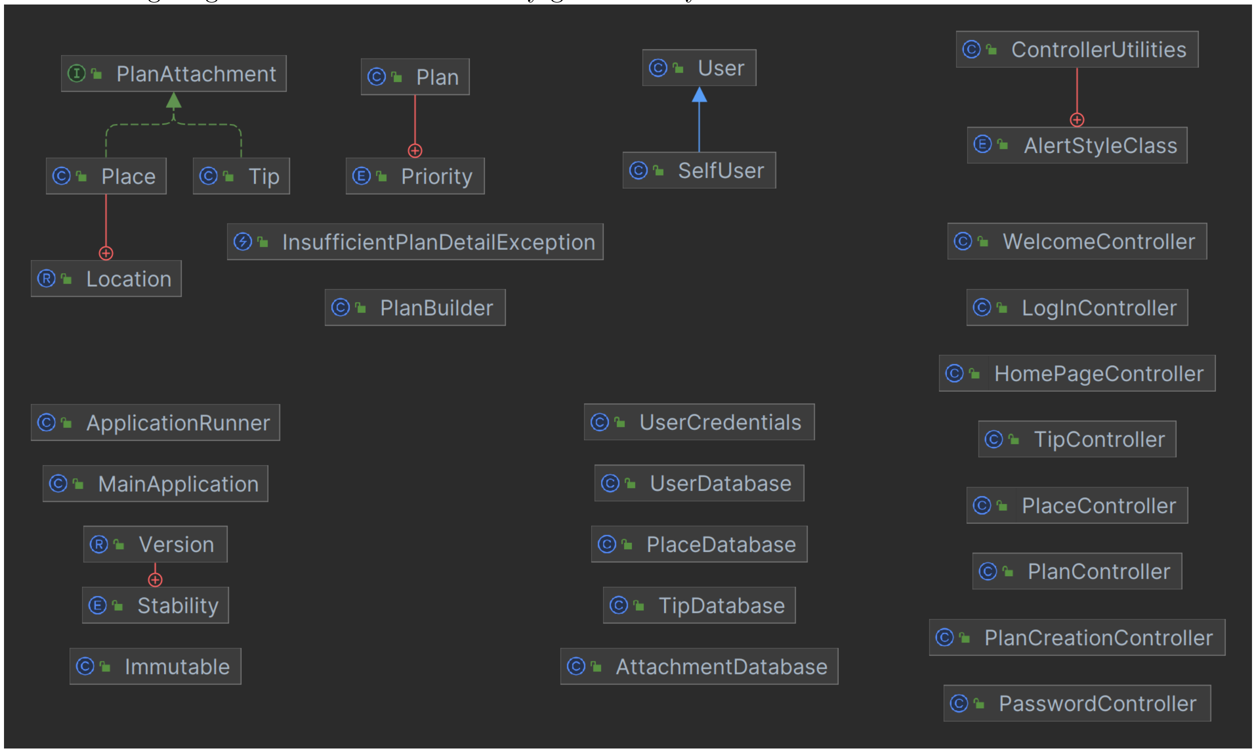
Secondary criteria

- Design patterns
    - Singleton: MainApplication.java
    - Builder: PlanBuilder.java
- Own exception classes
    - InsufficientPlanDetailException.java (PlanCreationController.java - line #192, ControllerUtilities.java - line #1490)
- Nested classes
    - Place.java - line #75
    - Plan.java - line #77
    - Version.java - line #50
- Lambda expressions
    - SelfUser.java - line #90
    - PlaceDatabase.java - line #49
- RTTI
    - ControllerUtilities.java - lines #375, #1310, #1331
- Default method implementation
    - PlanAttachment.java - line #19
- GUI providing + event handling:
    - ControllerUtilities.java - lines #62, #74, #162, #523, #1437, etc.
    - HomePageController.java - line #163
    - PlanCreationController.java - lines #135, #159

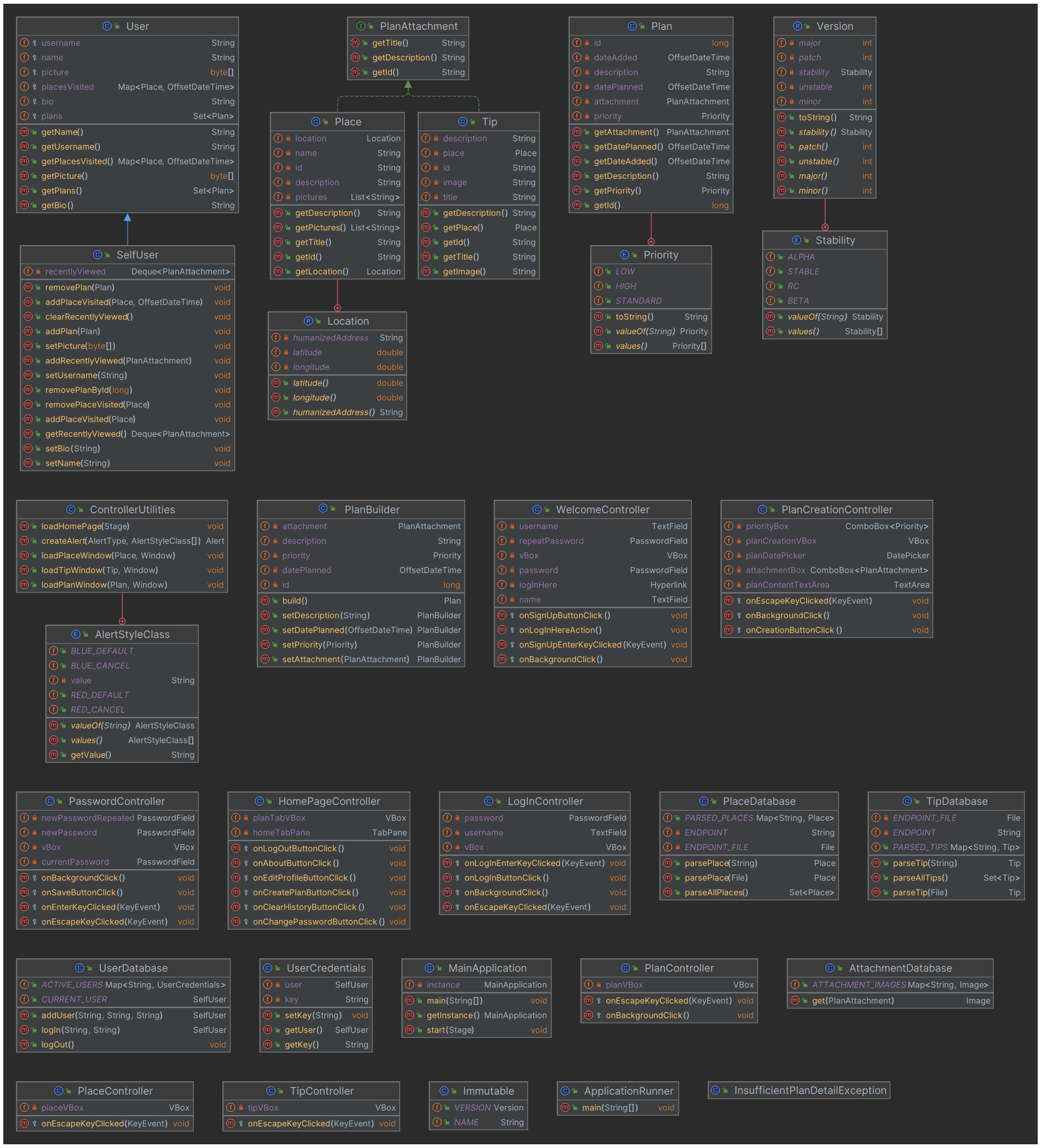
UML Diagrams

Class Diagram Illustration

The following diagram has been automatically generated by IntelliJ IDEA:



Full class diagram:



Class Description

- Entity classes:
  - Plan - contains all the plan data used in the program (e.g., the plan's priority, deadline, description, etc.)
    - Priority - an enum of degrees of a plan's priority
  - *PlanAttachment* - an interface object that can be used as an attachment inside a plan
    - Place - contains data about a sight in the city
      - Location - contains information about location of a place (e.g., the place's humanized address and geographical coordinates)

        * Tip - contains data about advice for traveling in the city
    - User - stores all the data about an application user
        * SelfUser - allows to modify the currently logged-in user's data
- Builder classes:
    - PlanBuilder - used to create an instance of the Plan object
- Exception classes:
    - InsufficientPlanDetailException - used in case a plan cannot be created due to an insufficient amount of data provided
- Database classes:
    - AttachmentDatabase - used mainly for caching JavaFX images of plan attachment objects (i.e., Tips and Places)
    - PlaceDatabase - used for parsing place data into a Place object
    - TipDatabase - used for parsing tip data into a Tip object
    - UserDatabase - used for operating user profiles
    - UserCredentials - used for simulation of storing a user's security data in a database
- GUI classes:
    - ControllerUtilities - contains a number of essential GUI methods (e.g., home page dynamic loading, alert styling, place/tip/plan window loading, etc,)
        * AlertStyleClass - an enum of pre-defined alert button CSS styles
    - Individual window controllers:
        * HomePageController
        * LogInController
        * PasswordController
        * PlaceController
        * PlanCreationController
        * TipController
        * WelcomeController
- Running classes:
    - Version - used for better program version organization
        * Stability - specifies the version's suffix (e.g., ALPHA, BETA, RC, etc.)
    - Immutable - contains a number of static fields used all over the program (e.g., the app's title, the version, etc.)
    - MainApplication - extends JavaFX Application, creates a starting window
    - ApplicationRunner - the program's main class (which is also provided as the main class while generating a JAR file)

Noteworthy Program Versions

- 0.1.4
- 0.1.5
- 0.1.6
- 0.2
- 0.2.1
- 0.2.2
- 0.2.3
- **1.0**

Version 0.1.4

- Plan list implementation

Version 0.1.5

- Image loading optimization

Version 0.1.6

- Tip GUI implementation
- Profile editing functionality
- Dataset expansion

Version 0.2

- The "Settings" tab:
  - Password changing
  - View history clearing
- Alert styling
- Tab icons
- Validation improvements:
  - Regex check for username fields
  - Username length range has been updated
  - New password is now to be repeated in another field
- Now places and tips are also saved into the view history when opened via attachment cards (e.g., from a tip window or a plan window)

Version 0.2.1

- The "About" window has been implemented
- The application icon is now present at the title bar of alerts

Version 0.2.2

- Password change hotfix
- Fix for the "Recently Visited" section after history clearing
- Successful password change alert has been added

Version 0.2.3

- Place parsing fix
- Logout confirmation

Version 1.0

- Re-logging in hotfix
- Additional password validation (the password cannot be the same as the username any longer)
- Plan window closing confirmation
- Plan deletion confirmation
- Successful history clearing alert
- Massive code cleanup

Technical Details

Software Setup

- Amazon Corretto 17.0.7
- JavaFX 17.0.7
- IntelliJ IDEA 2023.1.1
- SceneBuilder 19.0.0
- Gradle 7.6

Running via an IDE

- In case the project is run via IntelliJ IDEA, all the dependencies should be gotten automatically from Gradle and one should be able to run the program without any further modifications (such as additional VM arguments)
- In Eclipse IDE it is recommended to use the following VM argument if Gradle does not fetch JavaFX dependencies: `--module-path "PATH_TO_JAVAFX_SDK/lib" --add-modules=javafx.controls,javafx.fxml`

Fat JAR Generation

In order to generate a fat JAR, one should run the task named “fatJar” provided in “build.gradle”

Executable JAR Running

In order to run the executable JAR file, it is enough to open it directly, yet it is also important to keep the “data” directory in the same directory as it contains JSON data essential for the program to function.

Application Nuances

- In order to create an account, you must follow these guidelines:
  - No registration field must be empty
  - The username may only contain English alphabet letters, digits, underscores, and full stops
  - The username’s length must be in the range from 4 through 32 characters
  - The password must contain at least 8 characters
  - The password cannot repeat the username
- When creating a plan it is only obligatory to provide either a description or an attachment (a tip or a place); the rest (date and priority) is optional
- A plan card in the user’s plan list contains an image in case it has an attachment; no custom image can be added to a plan without an attachment manually
- An attachment can only be added to a plan directly from the place/tip view or via the view history (“Recently Viewed”) from the plan creation window

Simulation and demonstration of use

- A video demonstration of the project