

Tour Planner projekt

Software Engineering 2



BIF 4 / Gruppe C2

Chahed Rajoub

If19b166

Table of Contents

[**1.** **Selected Solution:** 2](#_Toc66028884)

[1.1. Analysis: 2](#_Toc66028885)

[1.2. GUI Design Pattern: 2](#_Toc66028886)

[1.3. Configurations and preparations: 2](#_Toc66028887)

[1.4. Result: 2](#_Toc66028888)

[**2.** **Structure / Design:** 3](#_Toc66028889)

[2.1. User Case Diagram: 3](#_Toc66028890)

[2.2. Class Diagram: 3](#_Toc66028891)

[2.3. Visual Snapshots: 3](#_Toc66028892)

[**3.** **Failures:** 4](#_Toc66028893)

[3.1. Design Pattern weakness: 4](#_Toc66028894)

[3.2. IDE and implementations: 4](#_Toc66028895)

[**4.** **Testing:** 5](#_Toc66028896)

[4.1. Unit Tests: 5](#_Toc66028897)

[4.2. Integration Tests: 5](#_Toc66028898)

[**5.** **Spent time:** 6](#_Toc66028899)

# **Selected Solution:**

## Analysis:

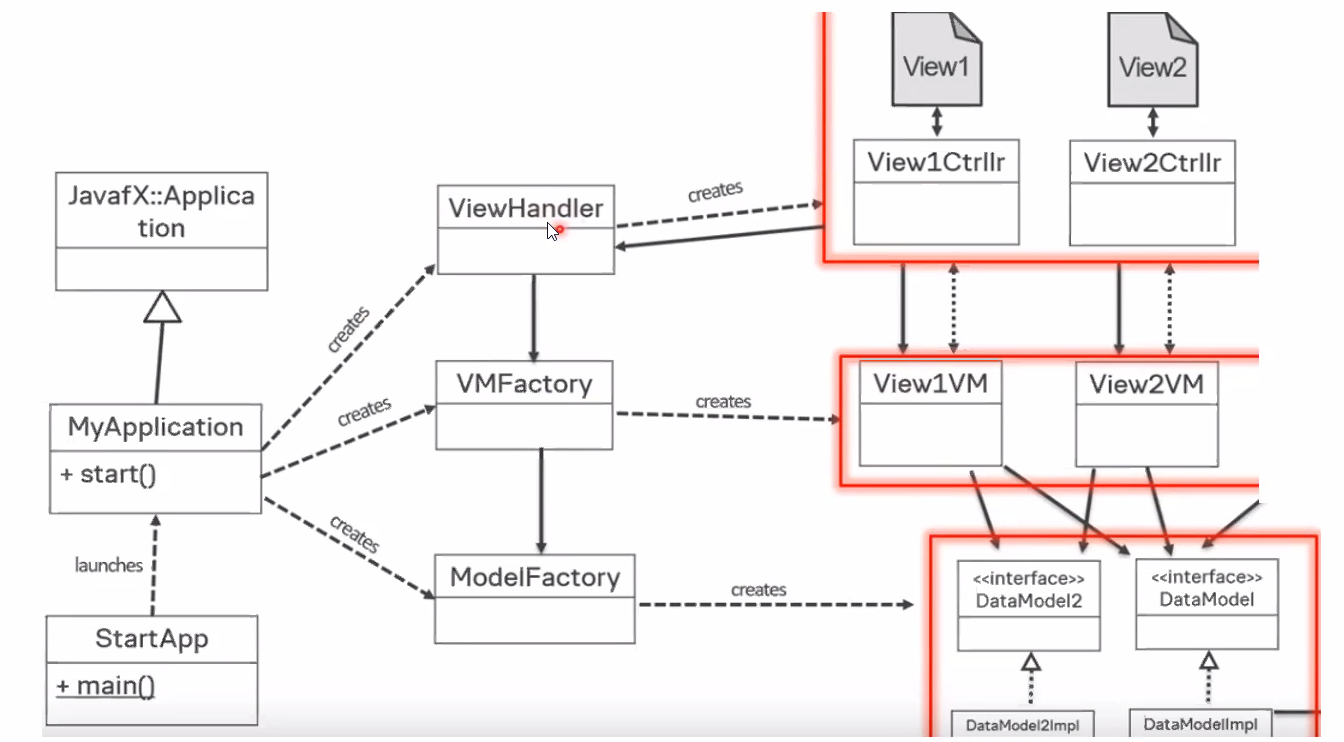
**Targeted group:** runners (anyone that wants to track his running tours and log them)

**Usability Goal:** This application aims to track the user tours and manage them in advance. Logs should contain statistic data of accomplished tours. As a personal feature, tracking calorie would be a great outcome with a graphic statistical analysis.

**Usability Description:** from a usability point of view, the app should be responsive, clear, and flexible. The User should know all the time what’s happening (Hints) and find the saved data easily. Every tour consists of name, tour description, route information (an image with the tour map) and tour distance the image should be retrieved by a REST request using (<https://developer.mapquest.com/documentation/directions-api/>). (CRUD) operations to manage the tours and the logs. Full text search and printed report should be available.

## GUI Design Pattern:

GUI-framework JavaFX as my UI Framework where MVVM model is implemented but as a preferred style in JavaFX MVP will be used and instead a presenter a view-Model will be implemented.



The Application communicates with the view handler/ factory and the Model-Factory. Create requests should create views and Data Models through binding functions and the Objects should be reachable and easy to render/update all the time to insure performance and flexibility.

## Configurations and preparations:

* JavaFX project with Maven in IntelliJ IDE
* Java version is 15
* JavaFX version is 15
* Libraries are imported like JavaFX controllers, JavaFX scenes builder, Jackson, PostgreSQL, Lombok, Slf4j.
* Nunit /Junit tests
* Documentation notation in the IDE should be generated
* the image should be retrieved by a REST request using (<https://developer.mapquest.com/documentation/directions-api/>)
* Integration tests would be a good extra step.

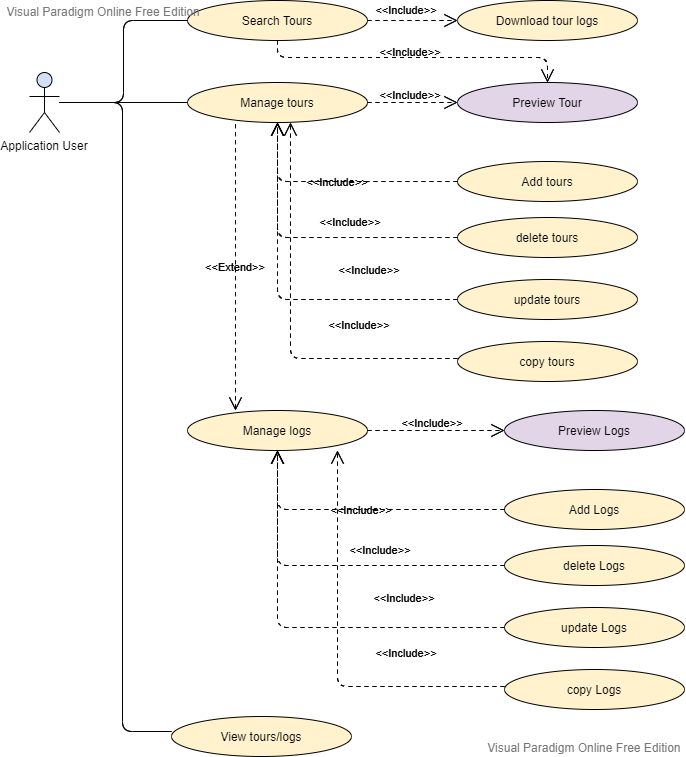
## Result:

The Project should include the following Results:

* View Factory interface
* Data Model interface
* View handler interface
* Packages for views and models
* Fxml controllers as components of the main controller
* Business logic package should include the controllers
* Resources Package where the data is handled
* Postgress Database with the logs and a 1-n relation between tours and logs
* Resources should be able to request the route in the developer map
* Security should be taken in consideration in the methods and the properties

# **Structure / Design:**

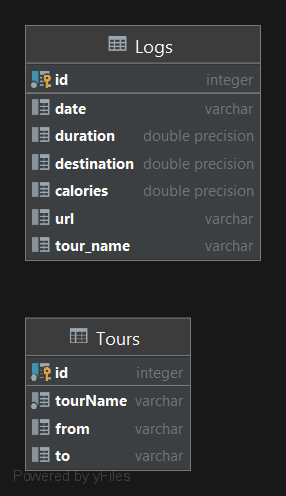
## User Case Diagram:



## Class Diagram:



## Visual Snapshots:



# **Failures:**

## Design Pattern weakness:

* The rendering of the view is placed in the Presenter, so the interaction between the view and the Presenter is too frequent. If the Presenter renders the view too much, it tends to make it too tightly connected to a particular view.
* Data binding makes bugs hard to debug.
* Large modules
* In MVP, your Presenter takes care almost everything because of dumb View, so it will become big and complicated gradually. Meanwhile, in MVVP, View-Model have support from View, especially Data Binding, you can reduce a part of logic codes.

## IDE and implementations:

* Working with maven without module info and the need to clean frequently
* Scene Builder in IntelliJ is not as good as the Gloun Builder and therefore, the need to switch between them.

# **Testing:**

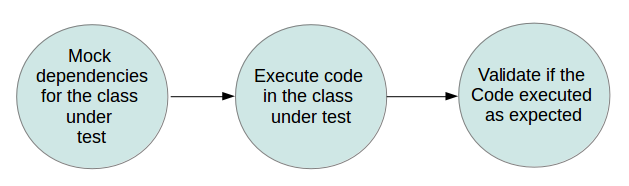
## Unit Tests:

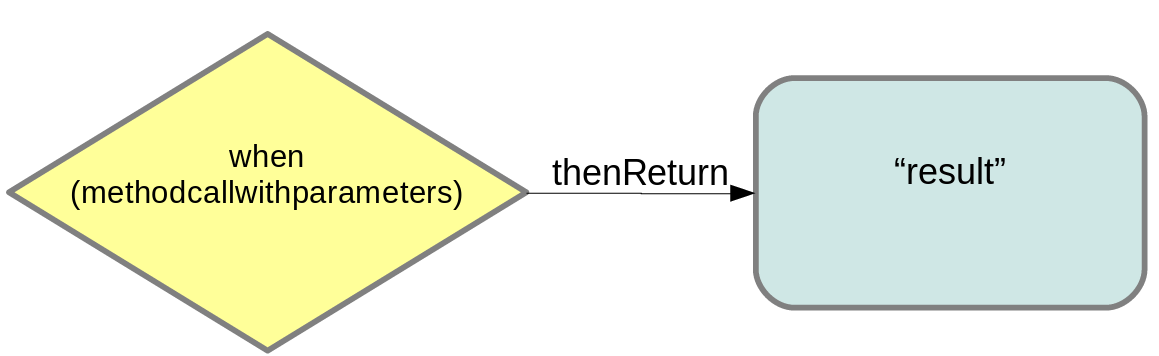
### [Using Mockito for mocking objects](https://www.vogella.com/tutorials/Mockito/article.html#using-mockito-for-mocking-objects)

Mockito is a popular mock framework which can be used in conjunction with JUnit. Mockito allows you to create and configure mock objects. Using Mockito greatly simplifies the development of tests for classes with external dependencies.

If you use Mockito in tests you typically:

* Mock away external dependencies and insert the mocks into the code under test
* Execute the code under test
* Validate that the code executed correctly





Unit tests are added too to test the methods in the classes:

* Checking the existing of an instance
* Checking the strings generated
* Checking the values that are generated
* Checking null points

# **Spent time:**

|  |  |
| --- | --- |
| Date | Hours |
| 8.3.2021 | 10 Hr |
| 10.3.2021 | 7 Hr |
| 21.3.2021 | 6 Hr |
| 2.4.2021 | 3 Hr |
| 3.4.2021 | 3 Hr |
| 4.4.2021 | 5 Hr |
| 10.4.2021 | 5 Hr |
| 26.4.2021 | 7 Hr |
| 27.4.2021 | 5 Hr |
| 9.5.2021 | 10 Hr |