TOUR PLANNER PROJEKT

Software Engineering 2

BIF 4 / GRUPPE C2
CHAHED RAJOUB
If19b166

Table of Contents

1. Selected Solution:

1.1. Analysis:

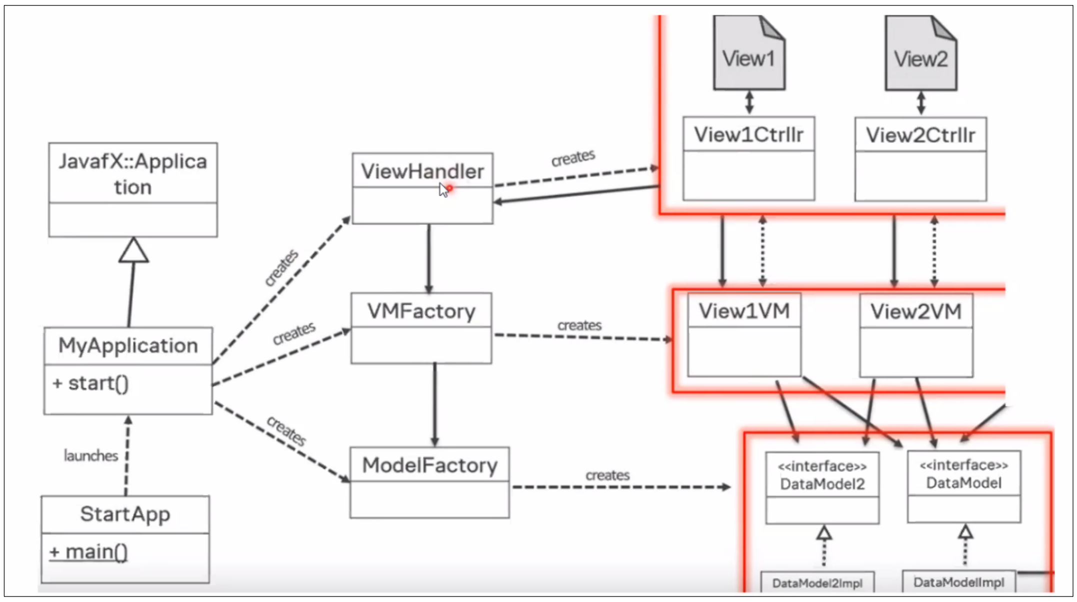
Targeted group: runners (anyone that wants to track his running tours and log them)

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.

1.2. 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.

1.3. 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 with test containers
- 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.
- Postgres Database Configuration to allow Data access using log4j
- Logging the most important steps in Controllers and Data access

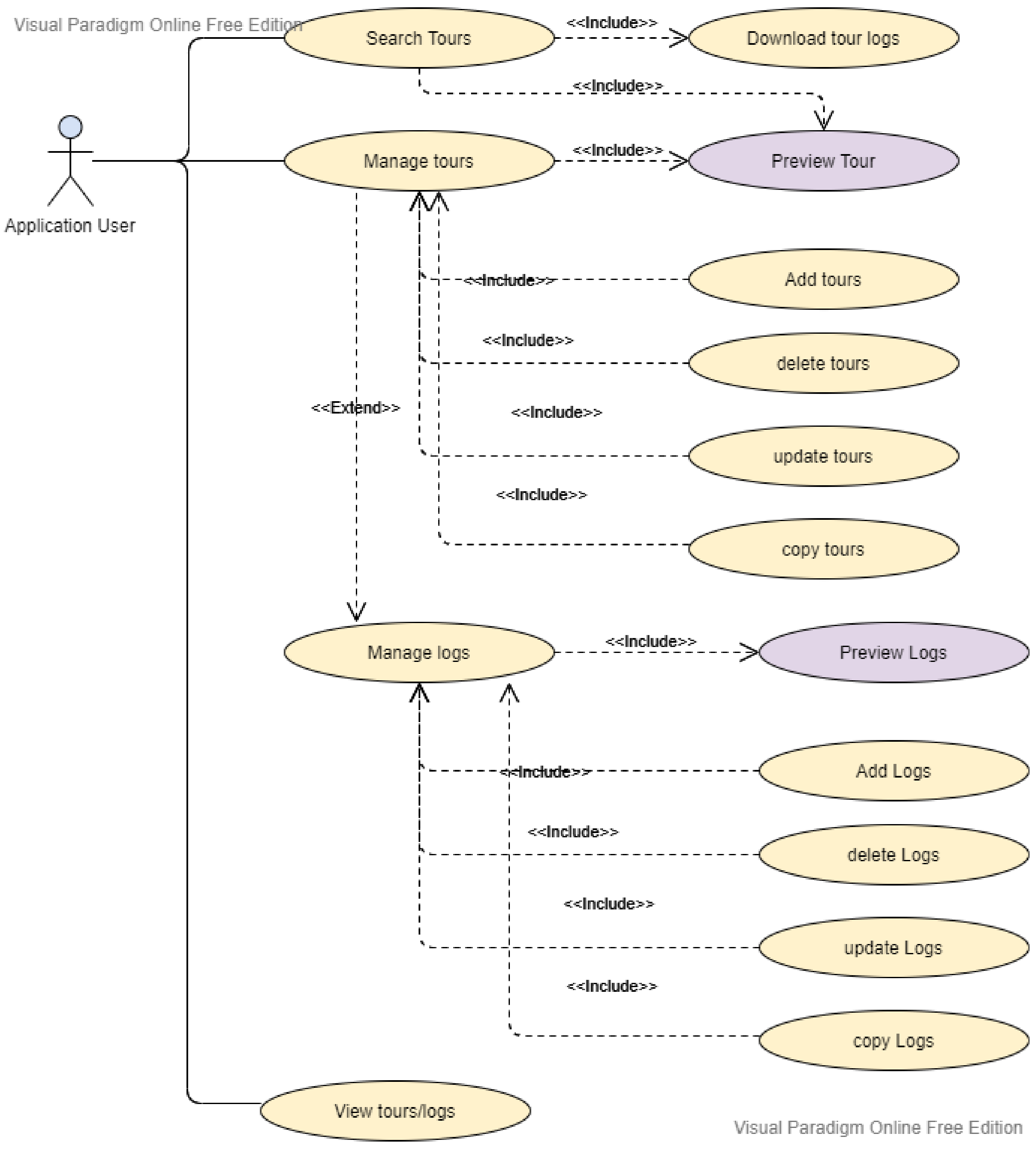
1.4. Result:

The Project include the following Results:

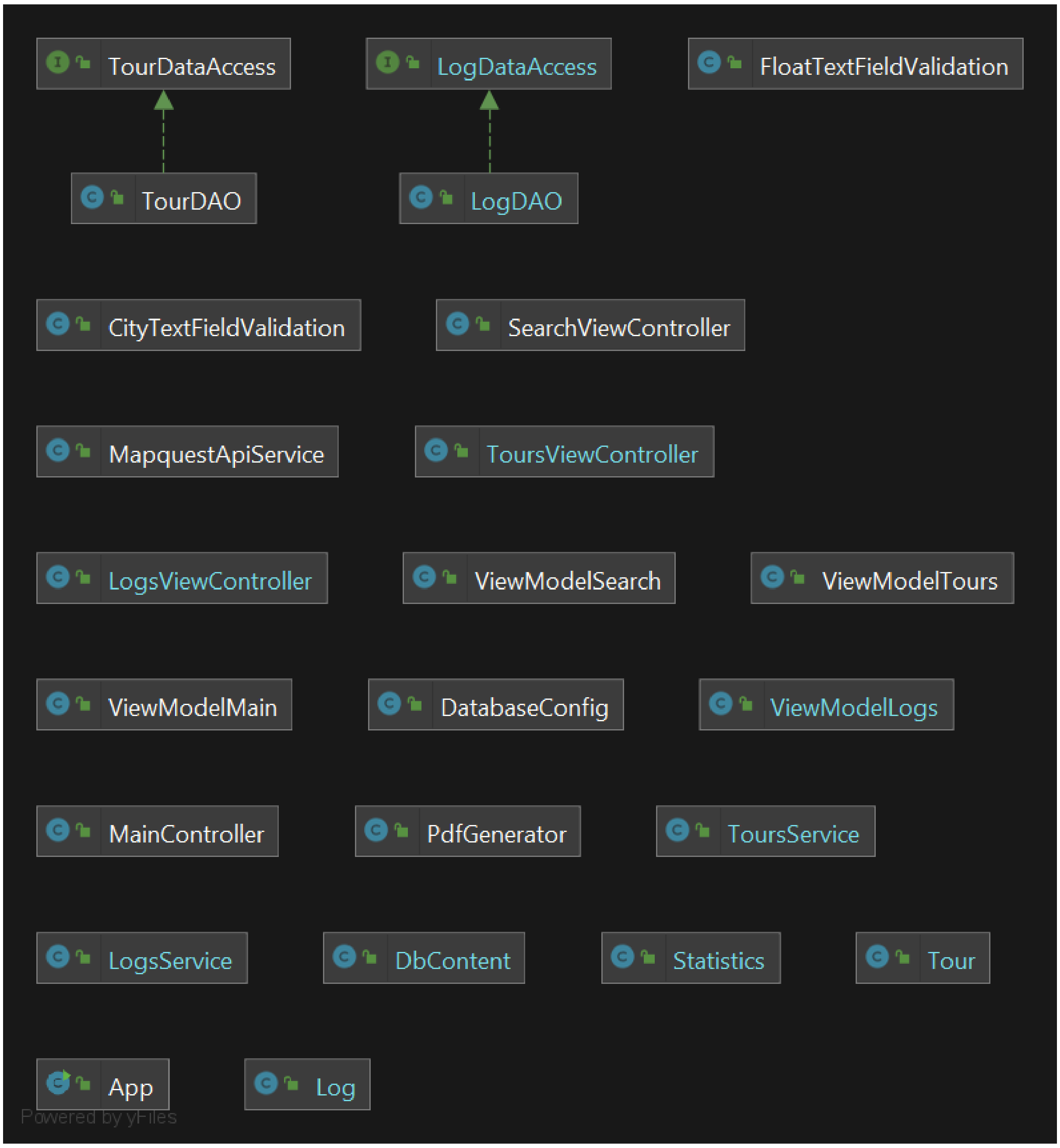
- View Factory interface in Data Access layer
- Data Model interface
- View handler interface (MVVM)
- Packages for views and models
- Fxml controllers as components of the main controller
- Business logic package to include the controllers
- Resources Package where the data is handled and Log Folder
- Postgres Database with the logs and a 1-n relation between tours and logs
- Resources can request the route in the developer map in Utility
- Security is taken in consideration in the methods and the properties with SQL injection
- All Methods create, delete, edit, and Copy for both logs and tours
- Export and import as json data file

2. Structure / Design:

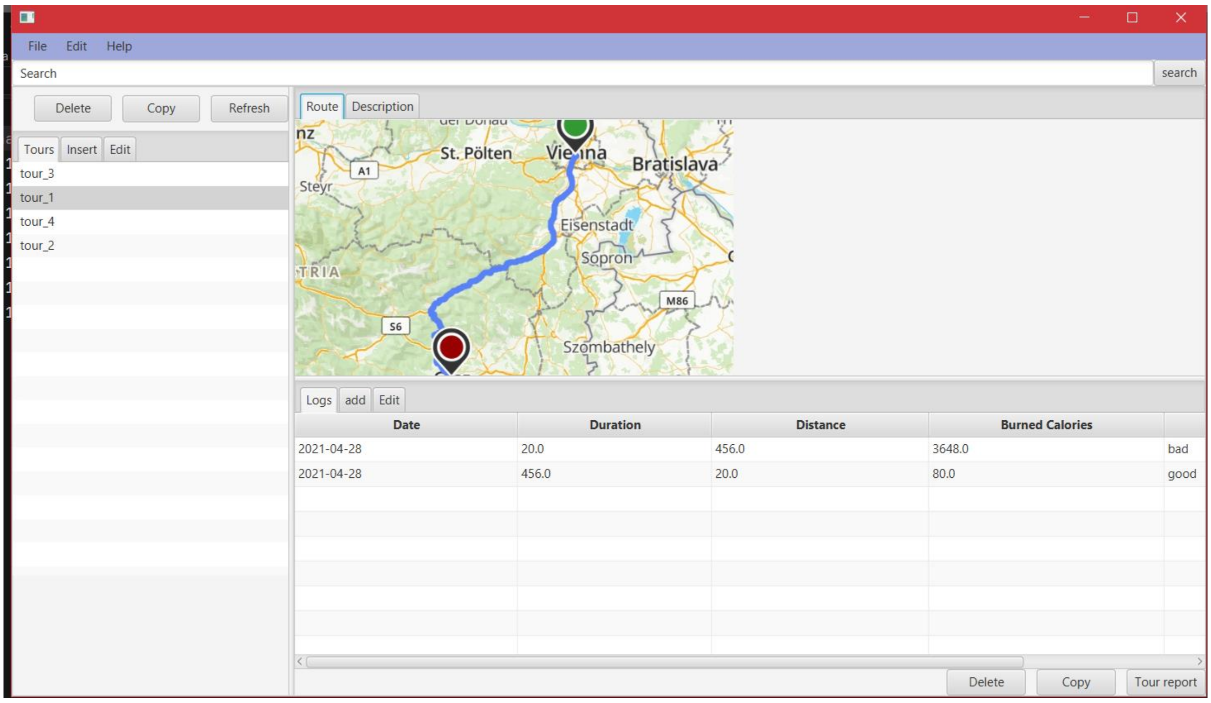
2.1. User Case Diagram:



2.2. Class Diagram:

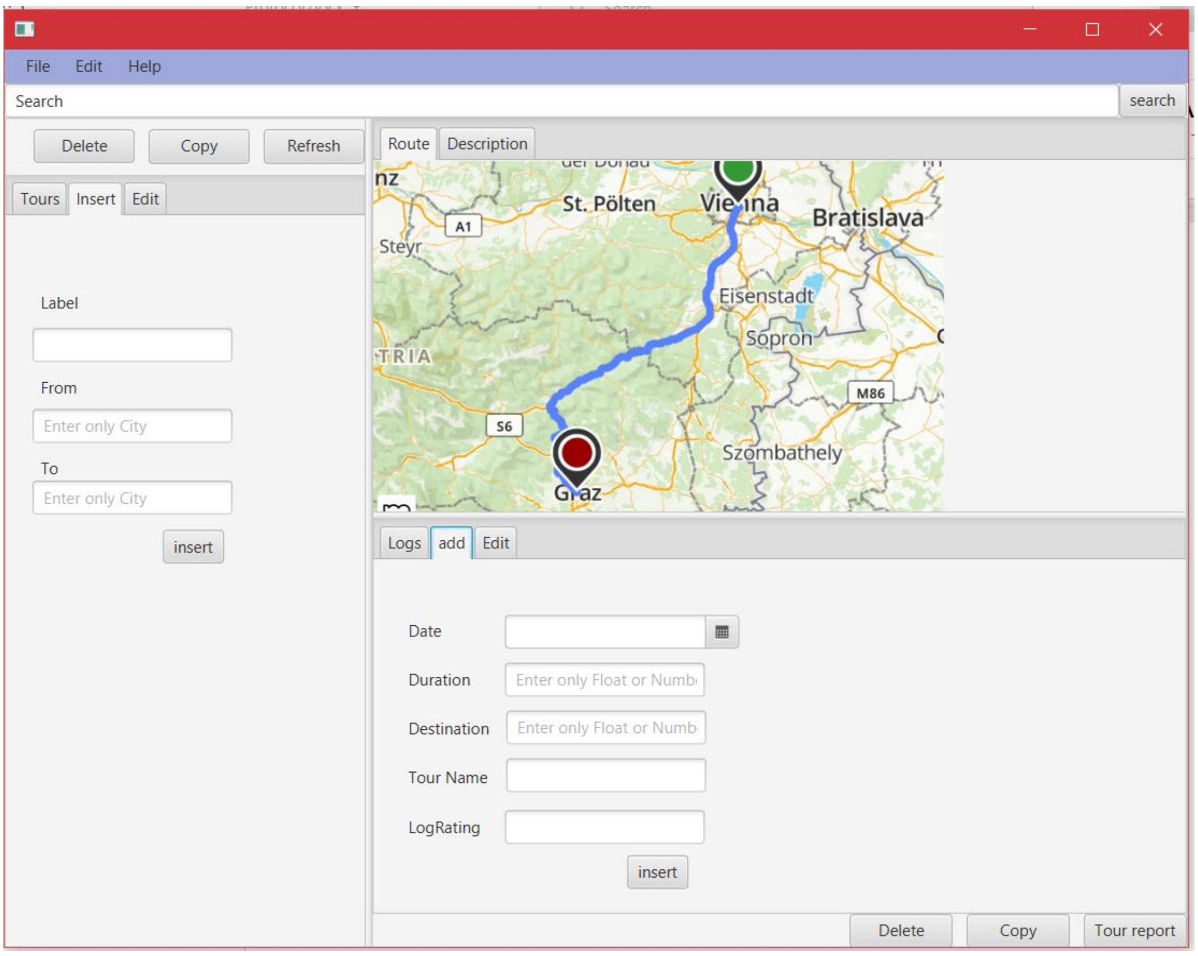


2.3. Visual Snapshots:



tour_3
tour_1
tour_4
tour_2

Date	Duration	Distance	Burned Calories	
2021-04-28	20.0	456.0	3648.0	bad
2021-04-28	456.0	20.0	80.0	good



Label
From
Enter only City
To
Enter only City

Date
Duration
Enter only Float or Numb
Destination
Enter only Float or Numb
Tour Name
LogRating

3. Failures:

3.1. Design Pattern weakness:

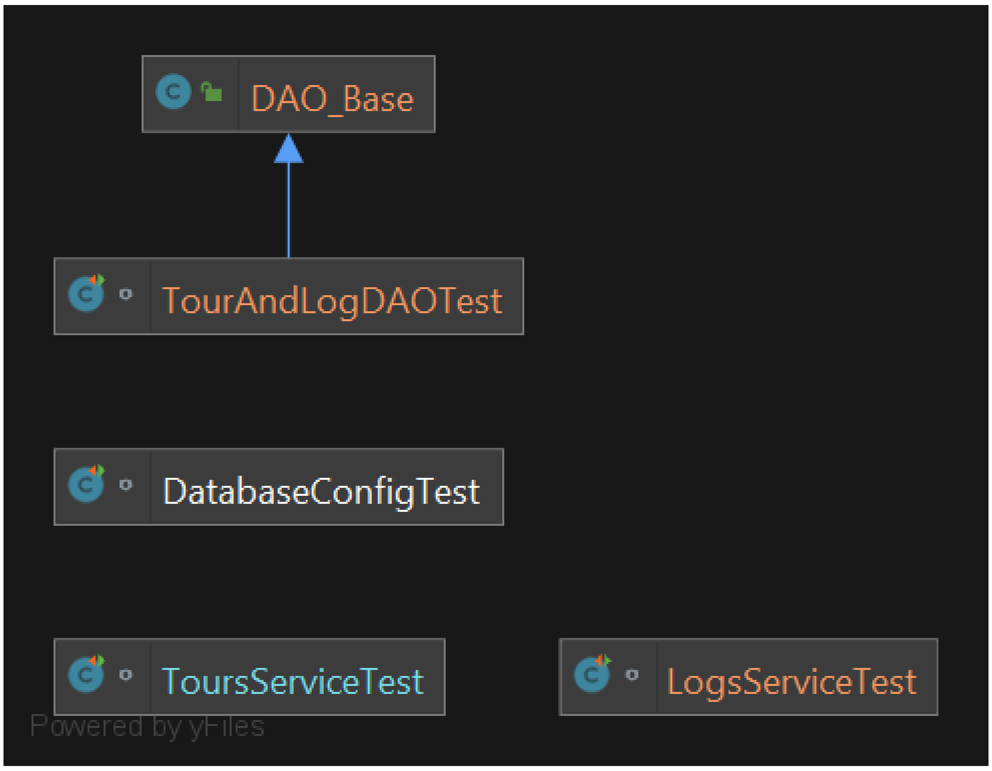
- The rendering of the view is placed in the Presenter, so the interaction between the view and the Presenter is too frequent. If the Presenter renders the view too much, it tends to make it too tightly connected to a particular view.
- Data binding makes bugs hard to debug.
- Large modules
- In MVP, 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.
- Import json was a challenge and is made simple
- Testing the Database is done carefully and took a lot of time

3.2. 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.
- I did not exactly face a huge problem only the flexible sizes to generat

4. Testing:

4.1. Unit Tests:



A Container in Docker with the Script used in my Database is used to mock the Database as a bonus Feature.

Test Container: Test containers is a Java library that supports JUnit tests, providing lightweight, throwaway instances of common databases, Selenium web browsers, or anything else that can run in a Docker container.

Test containers' generic container support offers the most flexibility and makes it easy to use virtually any container images as temporary test dependencies.

https://www.testcontainers.org/

Testing each method in Database and testing their Services by Mocking the Results

5. 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
20.5.2021	6 Hr
1.06.2021	5 Hr
6.06.2021	7 Hr
Sum	69 Hr