Protocol

Tour Planner

Technical Aspects & Design Implementation

For this project we have tried following the MVVM (Model View ViewModel) – Pattern, while also strictly abiding to the SOLID – Principles. Therefore, the project is structured in several packages, each following its own component or feature towards the program.

The biggest three overarching packages, following the MVVM – pattern, are:

- The model package: This is where we declared and initialized the base attributes of our classes or objects. In this case, for example, attributes of Tours (tourID, title, distance, time etc.) while also using Enums for certain standardized values such as the type of Transporter or Level difficulty of the Tour. Essentially, it is our data and apps domain model.
- The ViewModel package: The ViewModel implements properties and commands to which the view can data bind to. This defines the functionality provided by the UI to the user and also notifies the View of any state changes through notification events. In the example of our project, the View Model is responsible for coordinating the Views or UIs interaction with any of our Model classes of the Tour.
- The View package: The View is responsible for defining the structure, layout and appearance of what the user sees on the screen. Each View is defined by FXML. The view does not contain business logic. In the case of our project, we made an overarching package called View, where we again branched into 2 packages, the ViewModel and the Controller, the Controller representing the implementation of the View Concept of the MVVM – pattern.

Furthermore, within this project, we have packages purely for the database (binding and operations), where within the database package we also have packages responsible for dealing with saving files from within the local computer through paths. For database operations we used the concept of Data Access Objects (DAO).

We also created packages for utility tools that we used. Such as: Hash generators, text colors, serializers (to/from JSON), date and data formatting. In addition to these common tools, we also created a ThreadFactory or ThreadMaker class. With the usage of threads, we wanted to increase the speed of the program so more processes can be run concurrently to result in a more efficient program.

Significantly, we also used a config file to easily edit and define fixed configurations. The configurations include database credentials, file paths, fixed window size settings of the program, versioning and the ID credential required for the MapQuest API usage.

Furthermore, at first, we used a local SQL database such as Postgres within a Docker container. However later during development we played around with an online database in the hopes of achieving the objective of everyone being able to use the program without requiring a local database setup as it can be quite bothersome for certain conditions. With this we wanted to achieve coherence. We found several free online host options for SQL databases but eventually settled with ElephantSQL, which offers PostgreSQL as a Service. The free option offers us 20MB and 5 concurrent connections which is more than sufficient in our case. Lastly, for logging our project, we used Log4J.

In terms of unique features, we created a visual display of tour log statistics in terms of bar charts. This can be found within a tour, once it has been selected and further information of the tour is shown. Furthermore, we find the solution of an online SQL database to be unique and above the requirements of this project. As we offer both a traditional local SQL database with Postgres, as well as an online database, we count the addition of the online option to be a unique feature. Another unique feature is the possibility of choosing where to save the file from the program by asking the user to choose the path. What we are most proud of though is the inclusion of a .jar file of the application. Meaning it is an executable which can be opened without any setup by anyone on any computer and it will work as designed.

For collaborative and concurrent working we used GitHub. We set up a GitHub repository where we both would push and pull our progress to each other.

External Libraries and/or APIs included in the project:

- GSON
- Jackson
- Log4J
- MapQuest API
- Junit
- API Guardian
- Mockito
- Lombok
- SLF4J
- OpenTest
- Ikonli
- Core Maven dependencies

GitHub Link: https://github.com/immxmmi/4-Semester-TourPlanner

Docker Link: https://hub.docker.com/r/immxmmi/4-semester-tour-planer

Online Database: postgres://zqiwlukj:WVjZqzj-Jn-OzEPJ6ngmOYhhcHen4VyC@rogue.db.elephantsql.com/zqiwlukj

Failures & Selected Solutions

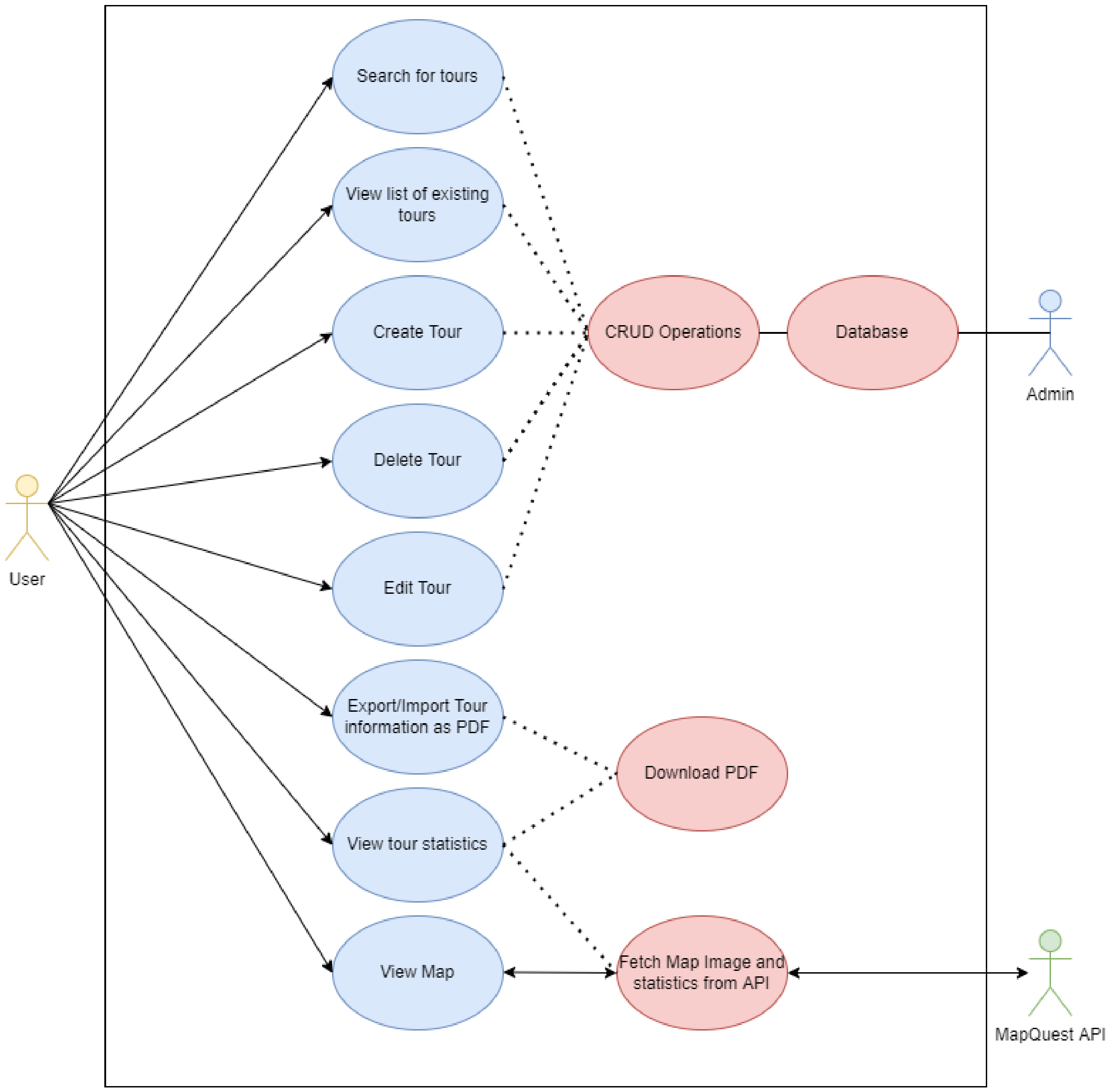
During the development of the project, we did not end up with failures. Everything we set out to achieve, we were able to successfully implement. However, naturally, we did have some complications with certain areas of the project, of which we will share instead. We had some complications with the Logger SLF4J, which is why we just ended up using the simpler Log4J instead. Furthermore, it also took a bit getting the hang of exporting Java objects onto a PDF document with the correct layout we wanted.

However, the biggest nuisance was JavaFX. JavaFX required a lot of dependencies to function properly. Consequently, at first it was difficult getting accustomed to how JavaFX works exactly with object binding and event triggers. Getting used to the workflow of JavaFX took a little time.

The biggest culprit of JavaFX was the SceneController, it was difficult coordinating the construction and achieving a certain structure which works dynamically.

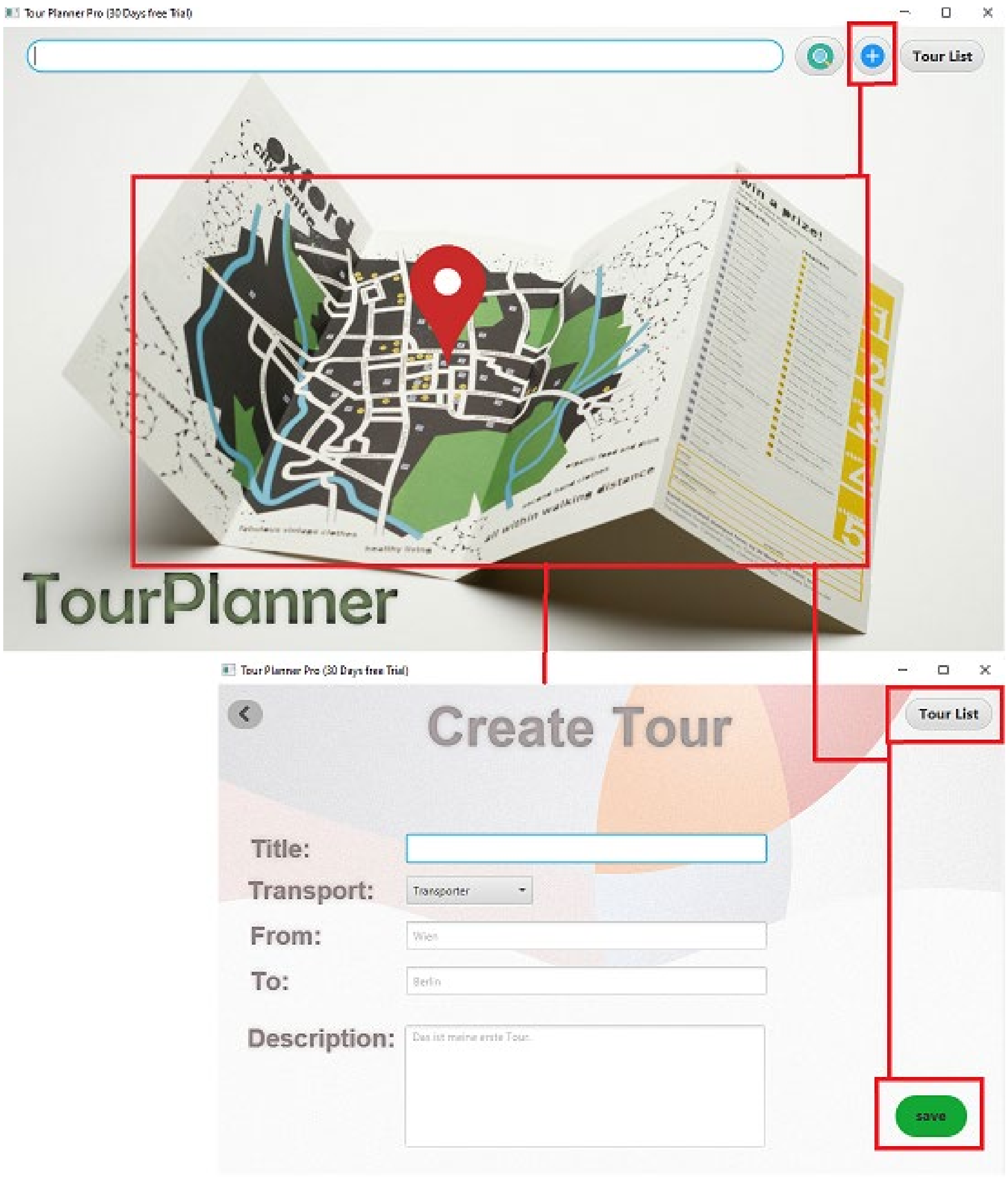
However, with the experience from last semester regarding the MonsterCardTrading Game, it was not such an intense challenge as most concepts generally did not feel foreign anymore. Understanding and using frameworks became more natural and faster.

Application features using an UML use case diagram

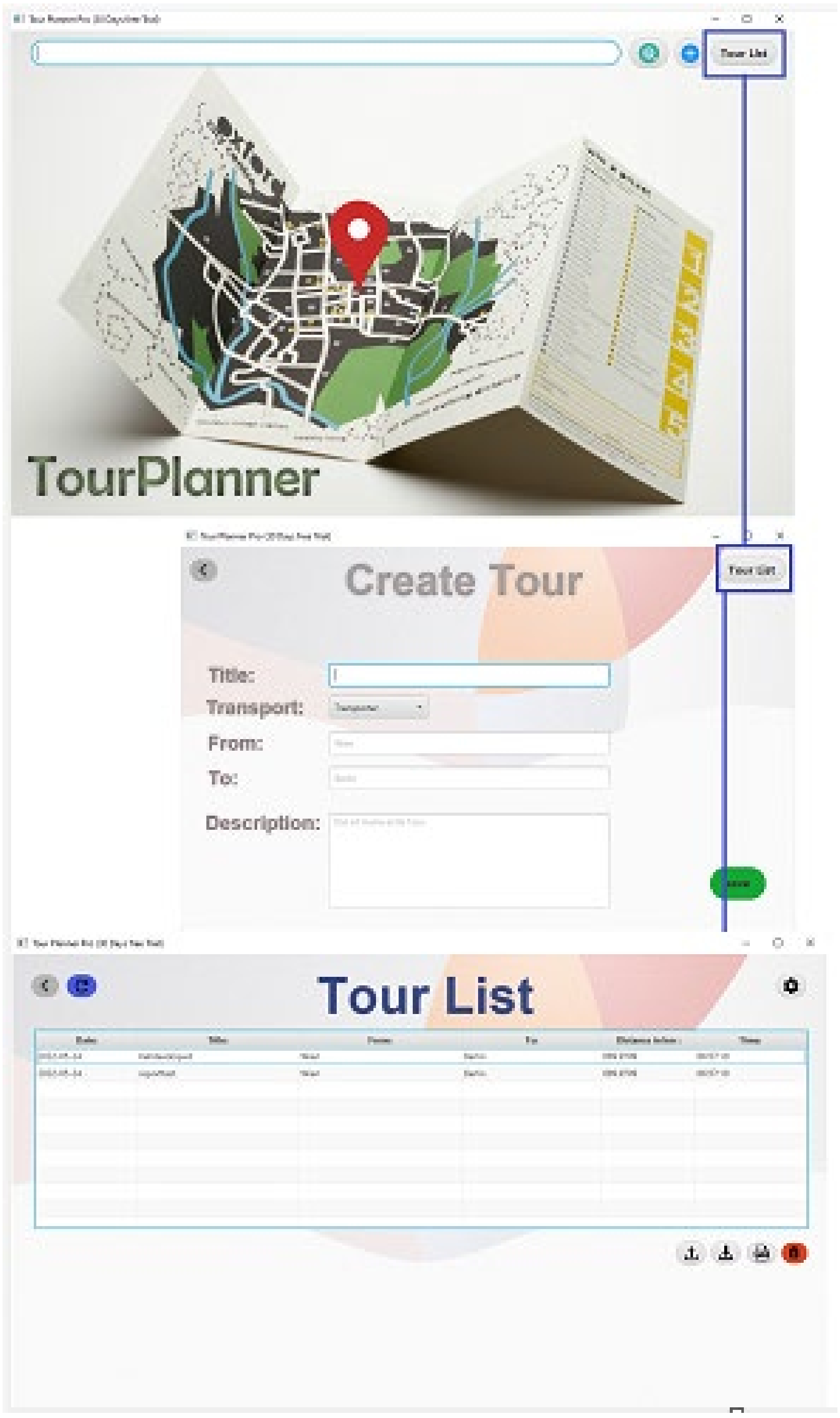


We want the program to run automatically without manual commands from the admin. Therefore, all features will be executed automatically without the admin requiring checking anything manually as error handling is built into the program. Therefore, any misuse from the user will be notified from the system. This is an effort to make the program independent. Merely the database will be administrated by the creators of the project. In addition, the MapQuest API also acts as a third-party to offer crucial tour statistics and information including the map drawn with distance and starting/end – points.

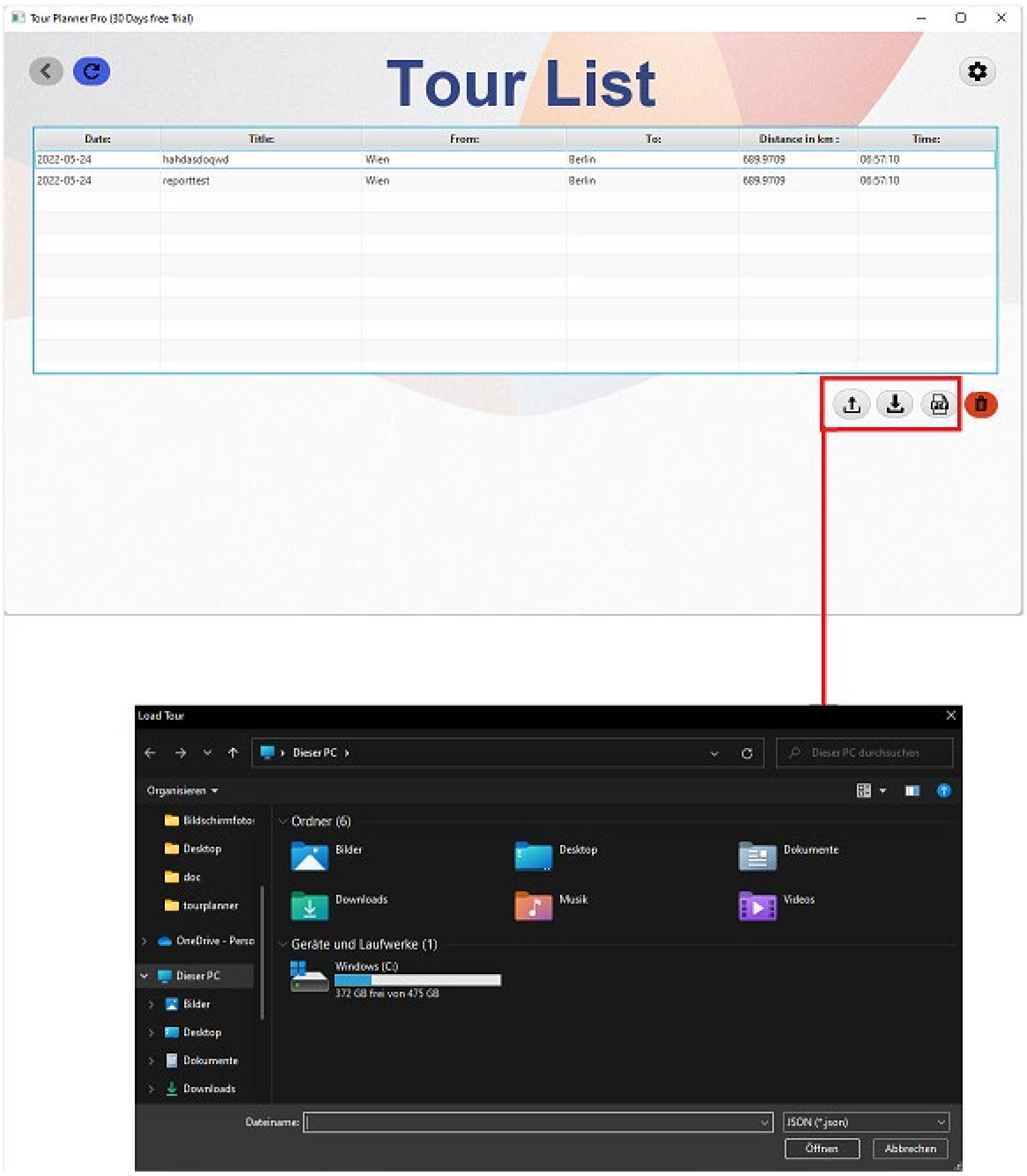
UI Flow using wireframes



The first image is the homepage, when clicking on the image in the middle or the blue plus icon on the right top, it will lead to the "Create Tour" screen. As the name suggests, here you can create tours and save them with the "Create" button. Clicking the "save" button or simply clicking the "Tour List" icon at the right top leads back to the homepage where further operations can be performed as shown on the images on the next page.

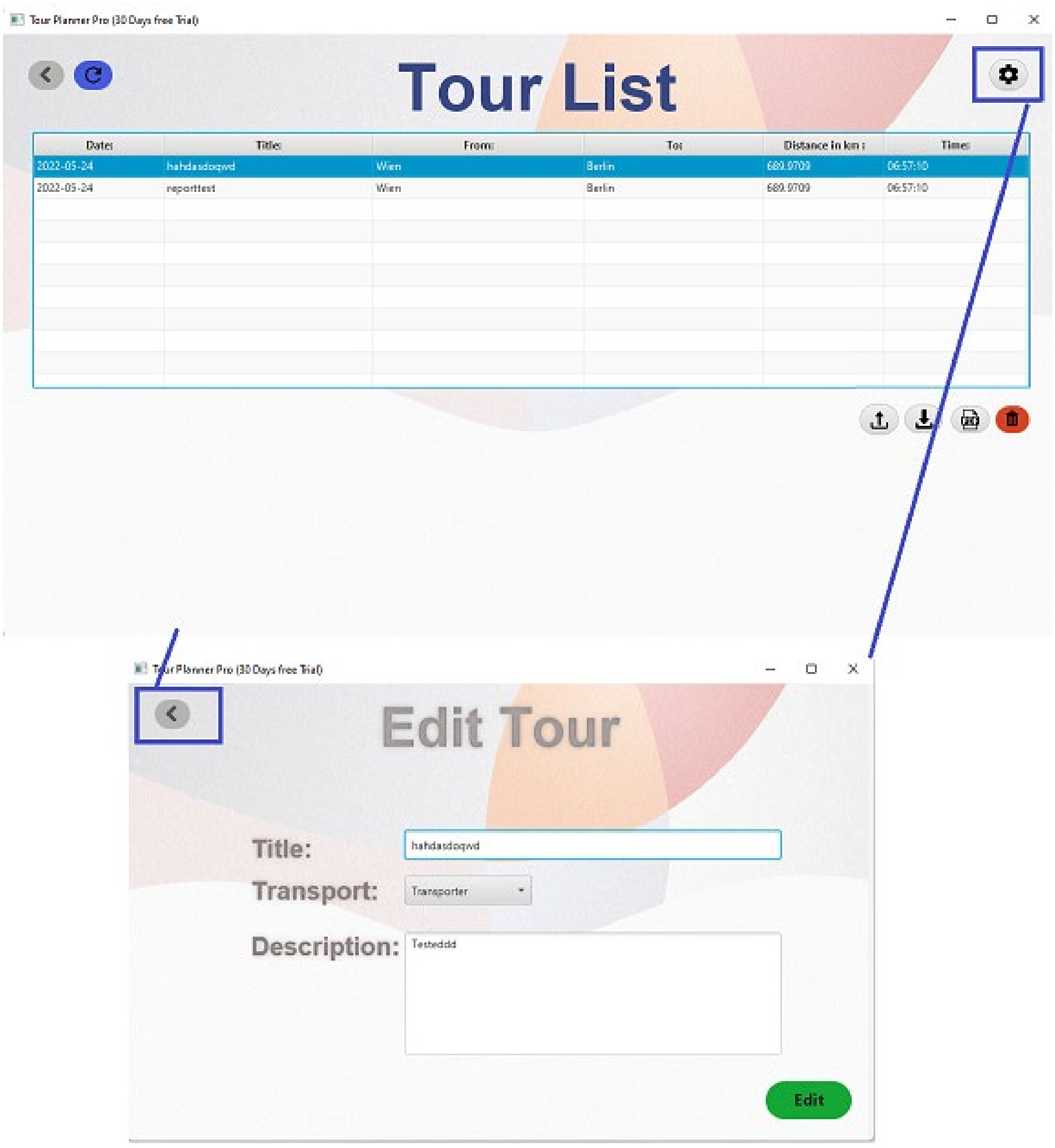


By clicking the “Tour List” button on both the homepage and the “Create Tour” page, or by simply searching for a tour in the homepage search bar, you will be led to the 3rd image. This, as the name suggests, is a list of all the tours that exist or have been created. From here further operations can be performed as shown in the images on the next page.



Date:	Title:	From:	To:	Distance in km :	Time:
2022-05-24	hahdasdoqwd	Wien	Berlin	689.9709	06:57:10
2022-05-24	reporttest	Wien	Berlin	689.9709	06:57:10

While on the “Tour List” page, when selecting a tour from the list, and then clicking on the pdf icon on the right bottom, the selected tour can be exported and saved as a pdf on the local computer of the user. If a user has a tour himself that he wants to upload, he can do so by clicking the upload icon on the right bottom of the screen. Both operations will lead to the file manager where the user can choose where to save the file, or in the case of importing, where to choose the file from.


Tour Planner Pro (30 Days free Trial)
Tour List

Date:	Title:	From:	To:	Distance in km :	Time:
2022-05-24	hahdasdoqwd	Wien	Berlin	689.9709	06:57:10
2022-05-24	reporttest	Wien	Berlin	689.9709	06:57:10

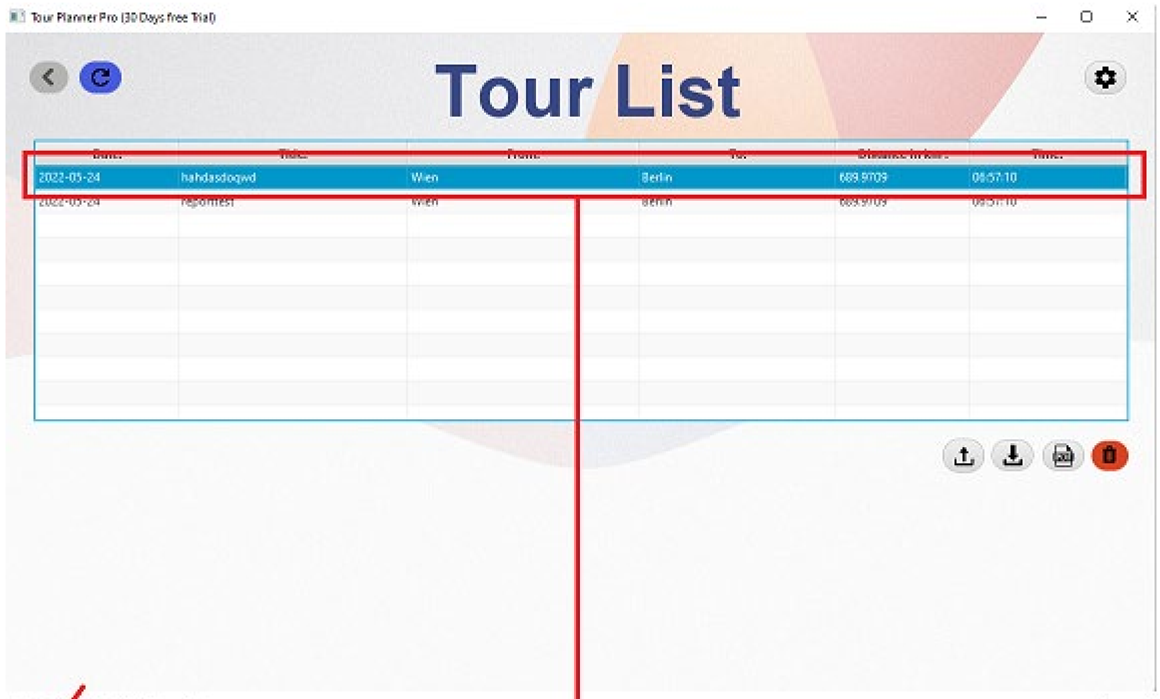
Tour Planner Pro (30 Days free Trial)
Edit Tour

Title: hahdasdoqwd

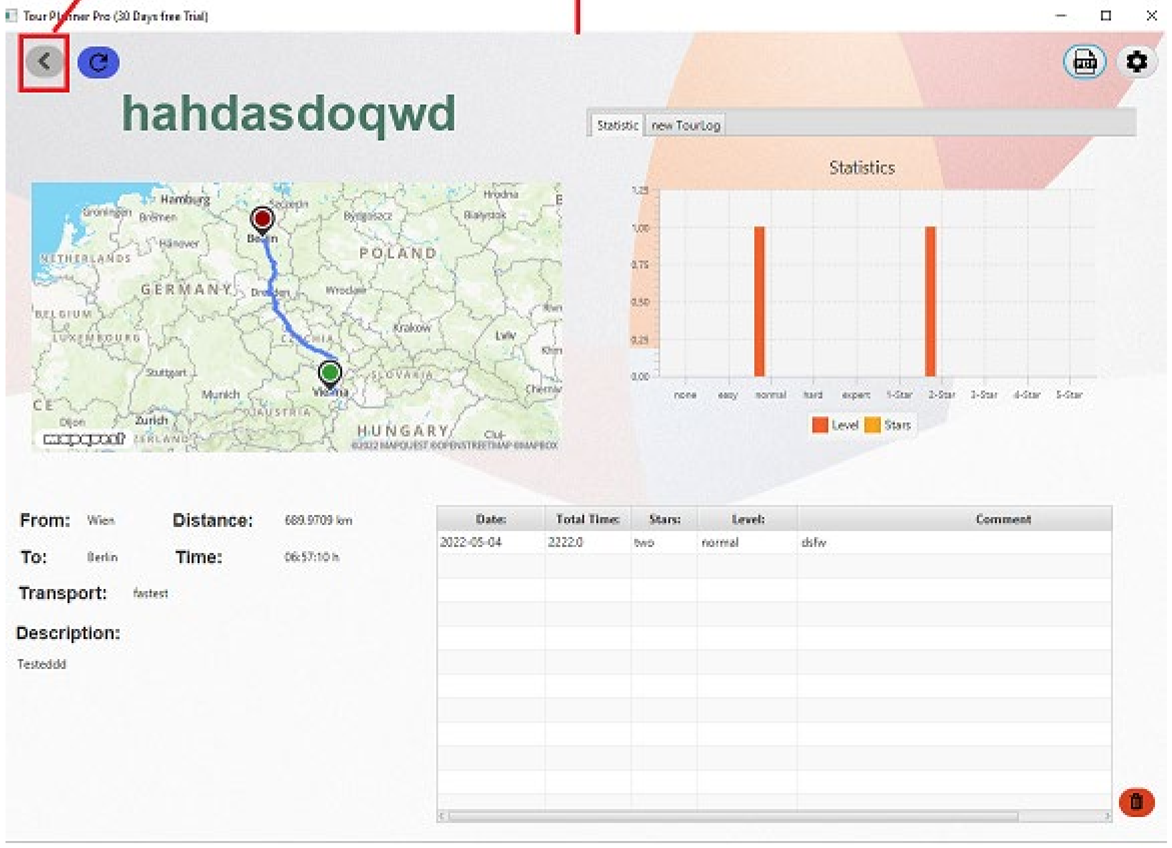
Transport: Transporter

Description: Testeddd

Still on the “Tour List” page, while having a tour selected and clicking on the wheel icon on the right top, the program will switch to another page, the “Edit Tour” page. This is where the user can change basic tour information such as title, transport, and description. Once saved by clicking on “edit” or simply by going back through the “Back arrow” button on the left top, the user will be led back to the tour list page where the edited tour will already be updated.



Date:	Title:	From:	To:	Distance in km:	Time:
2022-05-24	hahdasdoqwd	Wien	Berlin	689.9709	06:57:10
2022-05-24	reporttest	Wien	Berlin	689.9709	06:57:10

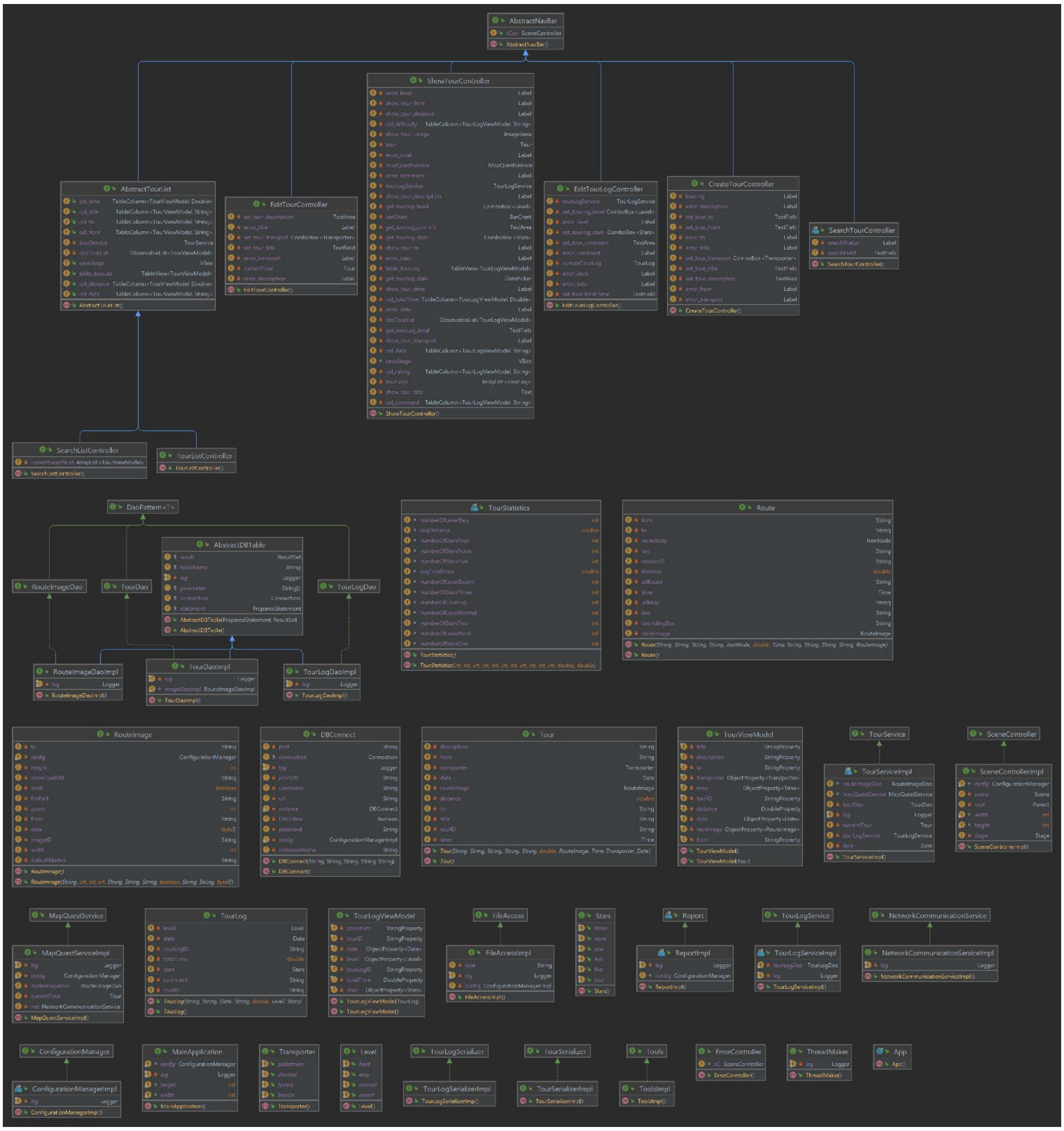


From: Wien
To: Berlin
Transport: fastest
Distance: 689.9709 km
Time: 06:57:10 h
Description:
Testeddd

Date:	Total Time:	Stars:	Level:	Comment
2022-05-04	2222.0	two	normal	dsfw

Lastly, while on the “Tour List” page, when double-clicking a tour, the actual tour page will open. Here the user can see exact details and statistics of the chosen tour, including an image with starting and destination point. The statistics include: the starting and destination city, distance, time it takes for travel while taking the form of transportation into account, the transportation and lastly the description. For a more advanced view, the statistics can also be viewed with a bar chart. Finally, all changes to the tour will be logged inside the log box on the right bottom. Clicking the green arrow on the right top will also lead back to the “Tour List” page.

Application architecture using UML diagrams



For a better image and view, the UML diagram is saved with the project as a png and uml file. We chose to opt with taking the entire project architecture as an UML diagram instead of single packages for model or similar. We believe that most of us already understand the basic model UML diagram and therefore did not find it necessary to break the UML down into smaller simpler ones. On this note, we let IntelliJ generate an UML diagram on the entire project. We believe this reflects the scope of our project better than any other alternative.

Unit tests chosen and why the picked code is critical

For the unit tests we wanted to be thorough and accurate. Therefore, we ended up with 68 unit-tests which cover areas where we believed makes most sense.

Starting off, we wanted to make sure that all the functions related to the MapQuest API work flawlessly as this is extremely important in making the entire project work. So the unit tests for the MapQuest API test the return objects from the MapQuest API and check if it's the right objects by making checks and requests.

Similarly, we tested the tour and tour log functions. We created unit tests for the most important functions and checked if they return the correct values by using dummy values.

We also did unit tests for the config file, by checking if the correct values from the config file are even used and/or passed.

Lastly, we tested the SQL operations inside our DAO classes for the MapQuest image, the tours and the tour logs. This was done in a similar manner by creating dummy objects and passing it, then performing operations such as deleting or inserting it to see if it was null or not. Further detailed tests have been done depending on the exact function. For the update function we updated the dummy object with dummy values and checked if it returned the correct value.

All the unit tests are sufficient and provide enough security and confidence in our project for a bugless and/or errorfree experience. Everything should work as intended.

Track time spent within the project

The entire project took us an estimate of 50 hours in total. This time is the sum of the collective hours we both spent on the project across the duration of the semester. Much time went into research and getting accustomed to JavaFX. We also count the time spent on research and learning during the project, this means time spent for the project outside of actual implementation. It is important to note that we did not actually actively track our time as it easy getting lost and losing track of time when programming. In conclusion, this is a rough estimate of hours spent on the project.