**Protocol**

**Tour Planner**

Swen2 – BIF4 C1

Inhalt

[App Architecture 3](#_Toc104835902)

[MVVM 3](#_Toc104835903)

[Model 3](#_Toc104835904)

[View and View Model 4](#_Toc104835905)

[Layers 5](#_Toc104835906)

[Business 6](#_Toc104835907)

[Data 6](#_Toc104835908)

[Class Diagram (Models) 7](#_Toc104835909)

[Sequence Diagram – Full Text Search 8](#_Toc104835910)

[Use Cases 9](#_Toc104835911)

[Unit Tests 10](#_Toc104835912)

[Business Layer Tests 10](#_Toc104835913)

[DAL Tests 10](#_Toc104835914)

[UX and Technical Decisions 11](#_Toc104835915)

[UX and UI 11](#_Toc104835916)

[Technical Decisions 11](#_Toc104835917)

[Lessons Learned 13](#_Toc104835918)

[Design Patterns 14](#_Toc104835919)

[Unique Feature – The Strikes 15](#_Toc104835920)

[Tracked Time 16](#_Toc104835921)

[GitHub Link 17](#_Toc104835922)

[Table of Figures 18](#_Toc104835923)

# App Architecture

Now working with JavaFX one part of the App Architecture was the visual representation while the other part consisted of working with the data. Due to that the application architecture is parted in : MVVM, Business and Data layer.

## MVVM

The first part is the MVVM Model. Also called Model – View – View Model Architecture. In that architectural style one parts each “Window” in a View, a Model and a View Model.

### Model

At first let’s take a closer look at the Models. The Models are similar to the ones used in Layered Architecture – The Model describes data structures – Objects which are used in the Code.

In our case we have the following two Objects:

1. Tour
2. Tour Log

Each Model describes the attributes of the Objects we have in the application.

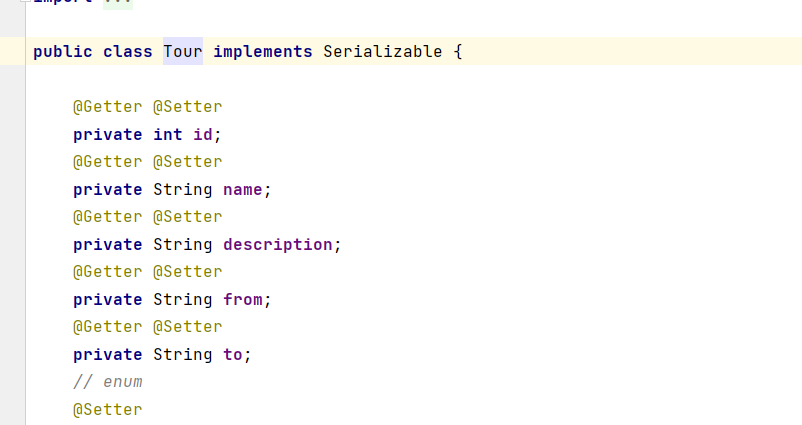


Figure 1: Tour Model Implementation

The Models are used to set Objects to get Database content in that structure or to insert new database content easier.

### View and View Model

The View and the View Model are both based on the amount of .fxml Files. In our case we have the following fxml Files:

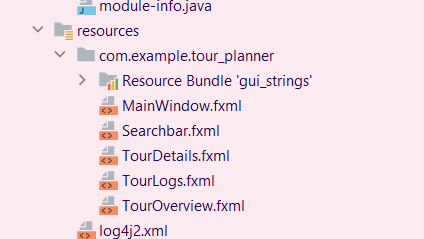


Figure 2: Overview of fxml Files

* MainWindow
* Searchbar
* TourDetails
* TourLogs
* TourOverview

Each fxml File describes the biggest Scenes or rather parts of the applications which each have their own task.

* The **MainWindow.fxml** describes the big picture: the main task bar and inclusion of the other fxml files.
* The **Searchbar**.**fxml** describes the Searchbar on top of the Screen.
* The **TourDetails.fxml** describes the structure of the window showing the Details of each tour as well as managing the Tours itself including editing, reporting and generating reports and summaries.
* The **TourLogs.fxml** shows the structure of the table next to the TourDetails Window.
* The **TourOverview.fxml** consists of the ListView showing each Tour created and includes the fxml files of the TourLogs and the TourDetails.

Each fxml File is connected to a Controller with the same name. The -controller is the namingconvention for our view-classes.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

The View therefore controlls the visual representation of the fxml Files. It initializes the data which is going to be displayed and is also the file in which we implemented the Functions called in the fxml. (Here : OnAction-functions )

Figure 3: Overview of Views

But the View only sets the attributes which we will see – the View Model is there to controll those. If there is any logical change « behind the scenes » , including changing/updating data or other logical tasks, the View Model is called.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungThe View Models have the name ViewModel in them. They are bound to the Controller Classes of the corresponding names.

Figure 4: Overview of View Models

If any changes are supposed to happen, the functions of the ViewModel Classes will be called, changing the data shown and set in the Views. Since they are bound toe ach other there is no need to set a return functions, therefore is each function which only serves for communication purposes between the View and the ViewModel a void function.

Since those ViewModels also have to communicate between each other we have the MainWindowViewModel.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Figure 5: Overview of MainWindowViewModel

The MainWindowViewModel has the ViewModels of each scenes as attributes, since it is there to control the visual representations based on impacts comming from external Views.

Example:

If a user clicks on one of the Tours in the TourList we send the MainWindowViewModel which Tour has been selected. Based on this selection the TourLogs View is going to change and the TourDetails View is changing as well. Therefore we call call the corresponding functions of each of these classes and change the Controller data and therefore the data shown to the user on the screen.

## Layers

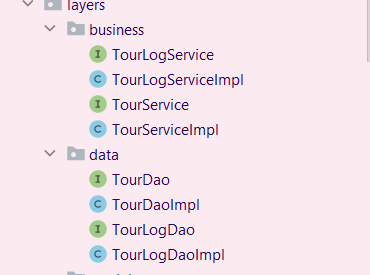
Digging more into the Back-End rather than the Front-End we focused on using Layered Architecture: Separating tasks in Business and a Data Access Layer (DAL).

Figure 6: OverView of the Data and Business Layer Classes

In each Layer we have classes corresponding to our Models mentioned before:

* Tour
* Tour Logs

### Business

The Business Layer is called when any functionalities based on logic have to be used: Calculating Popularity or checking on User Input as well as other logical functionalities. Therefore most of the times the classes from the Businessclass are being called if anything Data related is being done.

Example:

If a User creates a Tour. First the Business class will be called to check on the User Input based on the intantion of creating a Tour.

### Data

The Data Access Layer is the Layer used to access the database. This is the only layer which has a direct access to the database. If another Layer wants to access any data stored in our database then the DAL classes will be called. (In our case we called them DAOs which is short for Data Access Objects)

Let’s continue the example from above to demonstrate the Workflow :

After the Business Layer checked upon the input of the user . we create a Tour Object (from our Model class) and call the function createTour from our DAO class to access the database and save a new Tour.

## Class Diagram (Models)

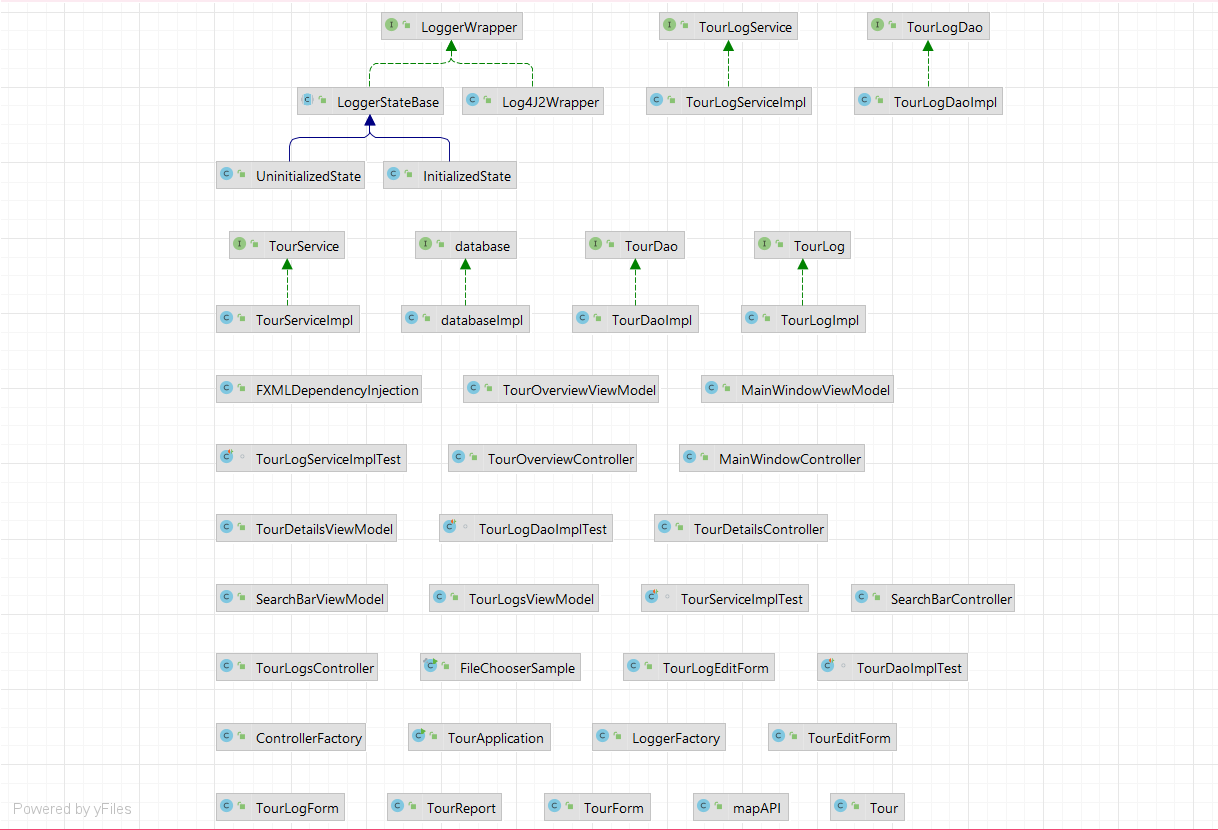


Figure 7: Class Diagram

## Sequence Diagram – Full Text Search

The following Sequence Diagram is a simplified version of the Full Text search possible in the application:

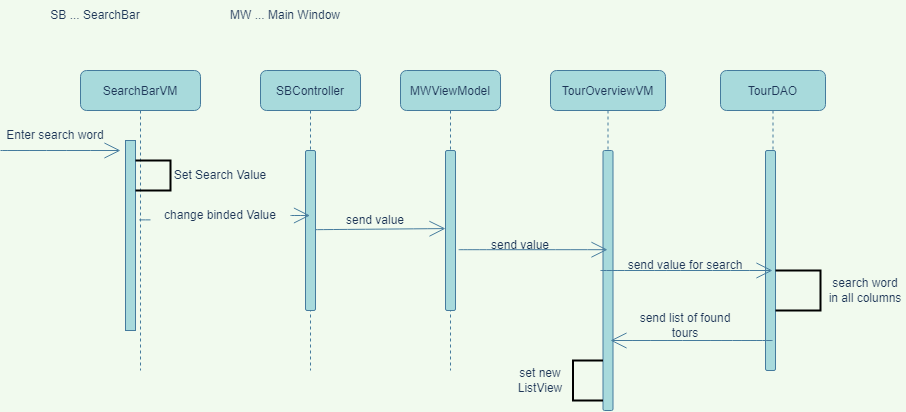


Figure 8: Sequence Diagram for fulltext search

# Use Cases

The Use cases will be described with the following use case Diagram :

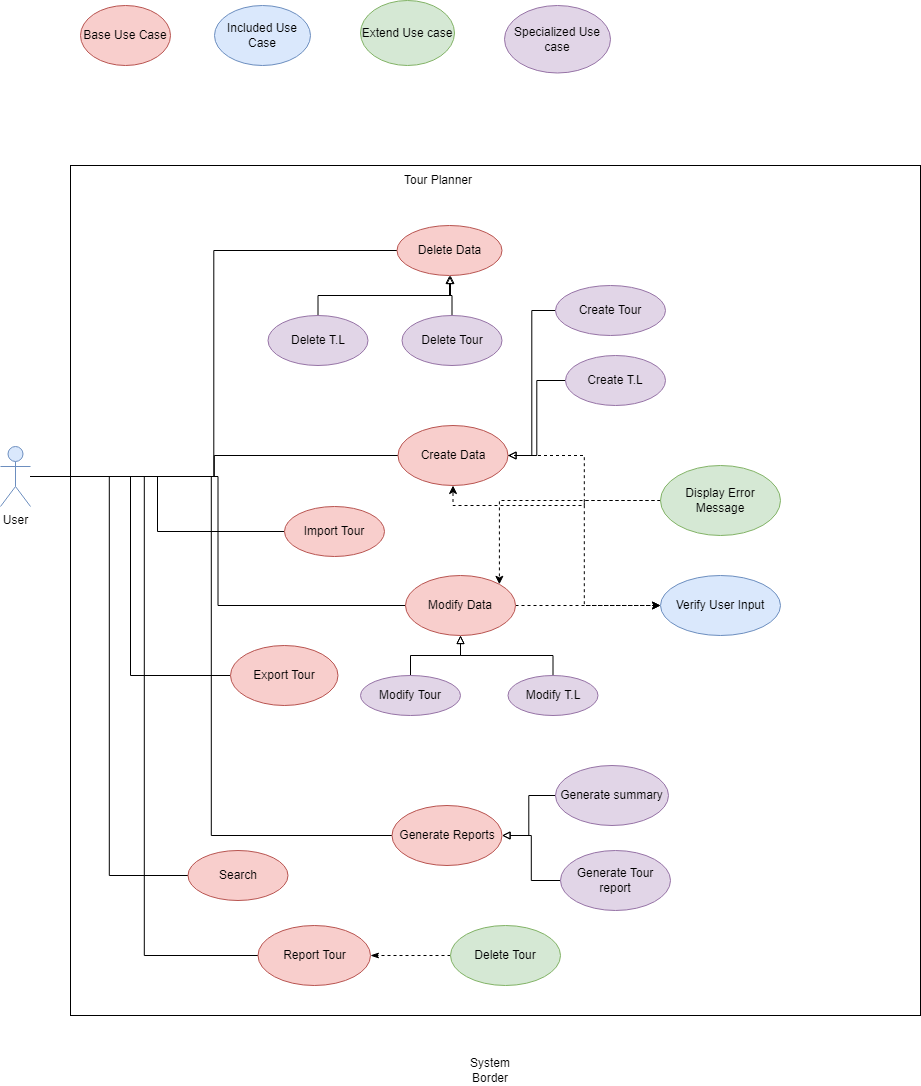


Figure 9: UML Use Case Diagram

# Unit Tests

For our Unit Tests we thought of implementing Unit Tests which are more on the Business Layer and Data Access Layer side of the Application.

The Reasoning for that is that those Functions are the ones deeply impacting the application, the workflow and the User’s View of the application.

## Business Layer Tests

The Tests for the Business Layer all test the calculations and error messages which will be received for certain test cases.

Example :

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Figure 10: Test Example

One of those Tests would bet he calculation of automated Attributes. Here we test:

* Is the sum grabbed correctly from the Data Access Layer
* Is the calculation done properly

## DAL Tests

The Tests for the Data Access Layer test if after accessing the database, the return value of those functions are correct.

In one of our test cases we test if the details of a Tour are being set correctly, so that the correct details of a Tour are being shown to the User.

# UX and Technical Decisions

## UX and UI

While designing the User Interface for this Application we focused on having everything on sight, nothing should be hidden, nothing should not be clear and all options should be visible or at least recognizable.

Therefore we made sure that all Overviews, Lists, Tables and Buttons are visible on sight. If it got too clustered we created Option Buttons which opened a List of Options.

The only windows which are not part of the static Main Window are :

* Forms
* Explorer

Those two forms of windows are pop ups which can be closed and opened with the corresponding button. They have been made as pop ups to separate the task of viewing and editing/creating in a visual sense.

Therefore the user experience while clicking buttons and viewing Tours is really clean and stress-free by not creating too much of a hassle while using the application.

To still bring in a breeze and not make the application seem like it is from 2009, so that users will actually enjoy using i tand fin dit visually pleasing , the team has decided on a cute and peppy theme using pink and rose tones as the main colorpalette of the application.

## Technical Decisions

For the technical decision making steps we first identified the problem we have to solve, which in our case was creating this application and then brain stormed the possible options on how to solve this problem with the programming abilities we have.

The bigger problem was therefore split into many smaller topics – which became issues on our github to be solved for later.

1. We first mainly focused on the MVVM Model and the data binding between View and ViewModel. The Data Binding was one of the most important technical decisions and improvements we have made tot he applications since it not only makes the code cleaner, but also separates the tasks of each layer better. For the Data binding on the Lists and the TableViews we have used ObservableLists. Those Lists are (as the name already says) being observed on changes and whenever a change occures the View is being updated.
2. When moving on the working with external data (importing and exporting data) we have decided to work with json Files and json Objects. Of course other options were possible as well, but working with a Json Object is much more effective in tasks of splitting, grabbing and inserting new data.
3. To generate a Tour Report we have decided to go with a pdf format. For that we have used the itext library which allows the programmer to create pdf files.
4. When separating the program into tasks we firstly focused on separating tour related tasks from tour log related tasks to create an easier overview of the files and the functions.
5. We decided to split extra utils , such as the map api, from the code completely by creating a mapAPI class in the utils directory.

# Lessons Learned

* Our lessons learned is more oriented on applying newly learned methods from the start on. Since the MVVM in combination with the layered architecture was a new concept to us itw as hard at first to 100% distinguish each layer from one another and create classes which would make sense in the programming sense.
* Another thing we have learned is a more frequent use of Unit Tests. We always implement the Unit tests at the end of the project which does not make sense at all. It makes testing the application so much easier. Instead of actually having to create Input or anything else you wanna test alone you can just implement it.
* The last thing we have learned is not to think in way too small pieces. It is good to separate windows and tasks and more from one another in an extend but too much will make the program more complicated than it has to be due to the communication between the classes.
* Use the MVVM possibilities as much as possible. We haven’t worked with data binding in the first project phase which later on was much easier to implement and made the code cleaner.

# Design Patterns

The following design patterns have been used :

* Singleton Pattern

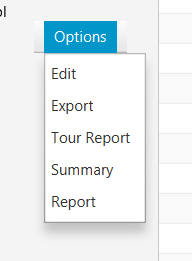
Example:

Here weh ave used the singleton pattern for our logger. The logger should only have on instance which is being shared through the entire application.

This makes more sense since otherwise each class which uses the logger would always create a new logger instance, therefore a new file and therefore a new console which is being to be logged in.

Another example of the usage is our database instance to ensures that only one instance of the class will be created and that instance will have global access within the application.

# Unique Feature – The Strikes



Since in our eyes this application is similar to Google Maps and other community applications, which loads of users use, we have decided to add the possibility of **reporting a Tour.**



We gave those the names: Strikes.

If a user does not like the route, thinks it is ridiculous or there is an easier way to reach your destination, those users can report a tour – give the corresponding tour a strike.

If enough users report a tour, the tour will be deleted. In our case, since it is a small application, a tour will be **deleted after 5 strikes.**

# Tracked Time

The following Tracked Time table will be separated in tracking the time per Git-Issue. When exactly a member has worked on the task can bes een through the git-history.

|  |  |  |  |
| --- | --- | --- | --- |
| Issue Title | Tracked Time ( in Hours/Minutes ) | Issue Opened | Issue Closed |
| Create Tour | 2h | 29.3 | 11.5 |
| Delete Tour | 1h | 29.3 | 10.5 |
| Modify Tour | 2h | 29.3 | 23.5 |
| Report Tours | 3h | 29.3 | 26.5 |
| Search | 6h | 29.3 | 28.5 |
| Show Tour Details | 30min | 4.5 | 10.5 |
| Tour Map | 4h | 4.5 | 24.5 |
| Add Tour Logs | 2h | 11.5 | 23.5 |
| Delete Tour Logs | 1h | 11.5 | 23.5 |
| Input Validation | 3h | 11.5 | 23.5 |
| Edit Tour Logs | 2h | 23.5 | 24.5 |
| Logging | 56min | 25.5 | 26.5 |
| Automated Attributes | 45min | 25.5 | 26.5 |
| Import/Export Tour Data | 2h | 26.5 | 26.5 |
| Generate Reports | 2h | 26.5 | 26.5 |
| Unit Tests | 1h | 26.5 | 28.5 |
| Write a Protocol | 2h | 25.5 | 29.5 |
| Create CSS | 7h | 4.5 | 29.5 |
|  |  |  |  |

# GitHub Link

**Link**: <https://github.com/kiubb02/Tour_Planner2>

# Table of Figures

[Figure 1: Tour Model Implementation 3](#_Toc104748832)

[Figure 2: Overview of fxml Files 4](https://d.docs.live.net/f83b99c0fc23ea74/Dokumente/FH%20TECHNIKUM/4.%20Semester/swen/Protocol.docx#_Toc104748833)

[Figure 3: Overview of Views 4](https://d.docs.live.net/f83b99c0fc23ea74/Dokumente/FH%20TECHNIKUM/4.%20Semester/swen/Protocol.docx#_Toc104748834)

[Figure 4: Overview of View Models 5](https://d.docs.live.net/f83b99c0fc23ea74/Dokumente/FH%20TECHNIKUM/4.%20Semester/swen/Protocol.docx#_Toc104748835)

[Figure 5: Overview of MainWindowViewModel 5](#_Toc104748836)

[Figure 6: OverView of the Data and Business Layer Classes 5](https://d.docs.live.net/f83b99c0fc23ea74/Dokumente/FH%20TECHNIKUM/4.%20Semester/swen/Protocol.docx#_Toc104748837)

[Figure 7: Class Diagram 6](#_Toc104748838)

[Figure 8: UML Use Case Diagram 8](#_Toc104748839)