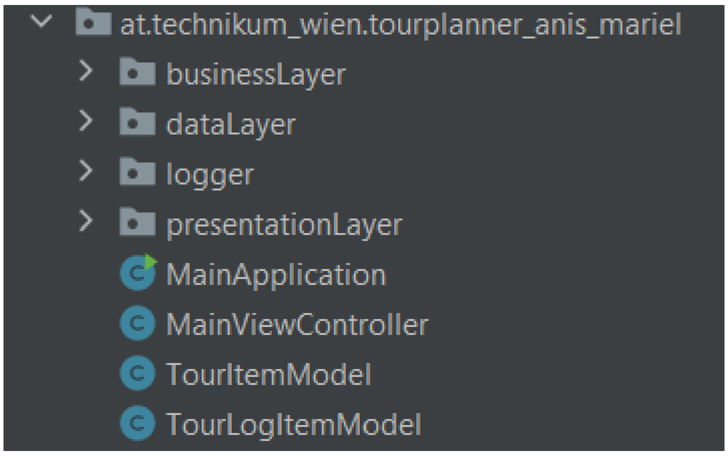
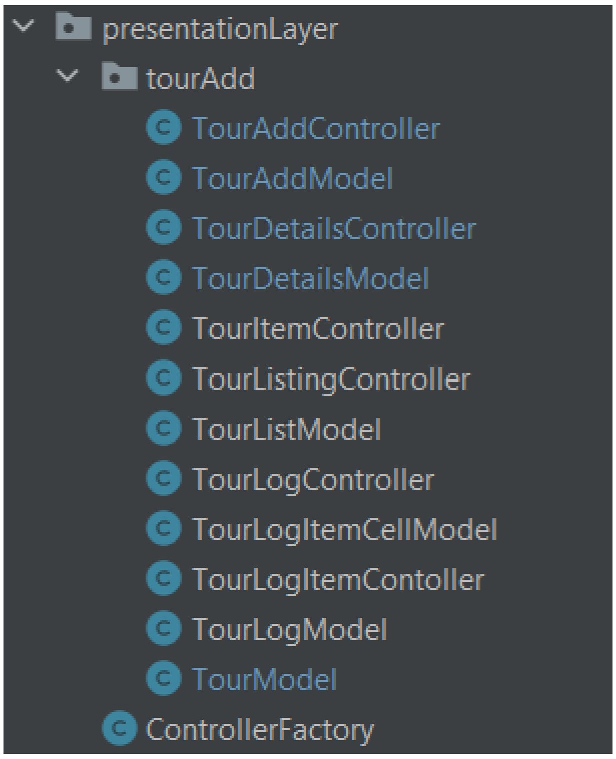
TourPlanner Documentation

App Architecture:

This App's architecture is a layered architecture divided into three layers. The presentation, data and business layers as shown in the picture below.

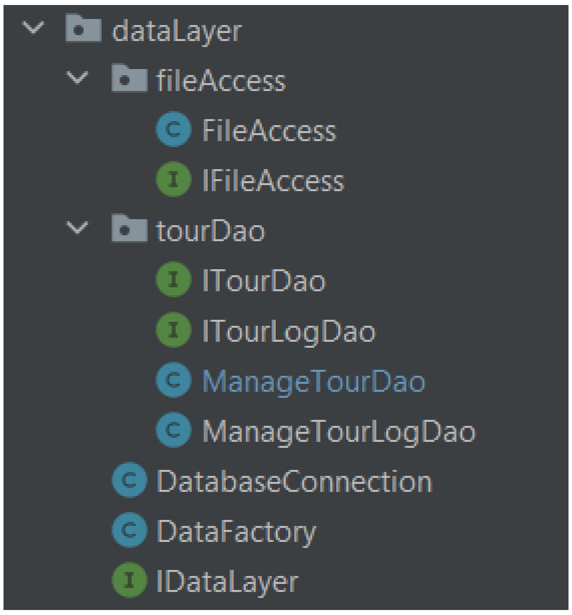

```
at.technikum_wien.tourplanner_anis_mariel
    businessLayer
    dataLayer
    logger
    presentationLayer
    MainApplication
    MainViewController
    TourItemModel
    TourLogItemModel
```

The presentationLayer is where the models and the controllers of our UI were developed. Everything that the user interacts in the frontend with can be found inside this folder.


```
presentationLayer
    tourAdd
        TourAddController
        TourAddModel
        TourDetailsController
        TourDetailsModel
        TourItemController
        TourListingController
        TourListModel
        TourLogController
        TourLogItemCellModel
        TourLogItemContoller
        TourLogModel
        TourModel
    ControllerFactory
```

The ControllerFactory class is a factory design pattern which is use to give us an object for every controller in the presentation layers and is going to be called at the Main class for initializing all of them.

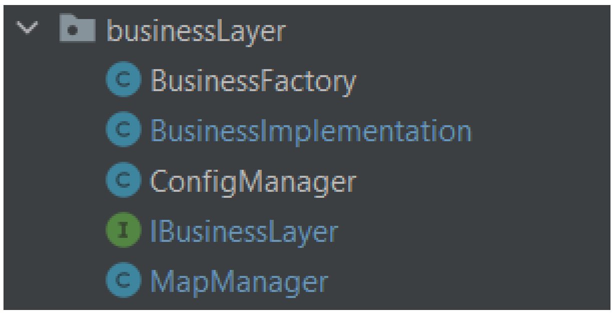
Next is the dataLayer which is use for database operations like connection and the insert, update, delete methods.



```
dataLayer
    fileAccess
        FileAccess
        IFileAccess
    tourDao
        ITourDao
        ITourLogDao
        ManageTourDao
        ManageTourLogDao
    DatabaseConnection
    DataFactory
    IDataLayer
```

```
public ManageTourDao() throws FileNotFoundException {
    dataLayer = DataFactory.getDatabase();
}
```

IDataLayer is the interface which this package uses to communicate with the business layer. We have a DataFactory Class which gives us a single instance of the database and the ManageTourDao and ManageTourLogDao classes which implement their respective interfaces. I decided to divide the tour detail and logs into two different classes and implement them separately for a more structured architecture. The fileAcccess package is used for the saving of the image in a file and loading it into the program again, import and export, summarization of the tours. On the businessLayer we implement the dataLayer classes and prepare them for the presentationLayer.



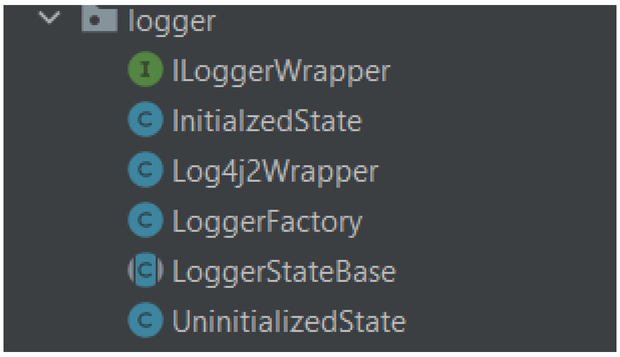
```
businessLayer
    BusinessFactory
    BusinessImplementation
    ConfigManager
    IBusinessLayer
    MapManager
```

```
private IBusinessLayer businessLayer = BusinessFactory.getBusiness();
```

```
ITourDao tourDAO = DataFactory.ManageTourDao();
```

Again, we have a Factory which gives us a single instance of the BusinessImplementation class where we implement all the methods that we want to pass to the presentationLayer.

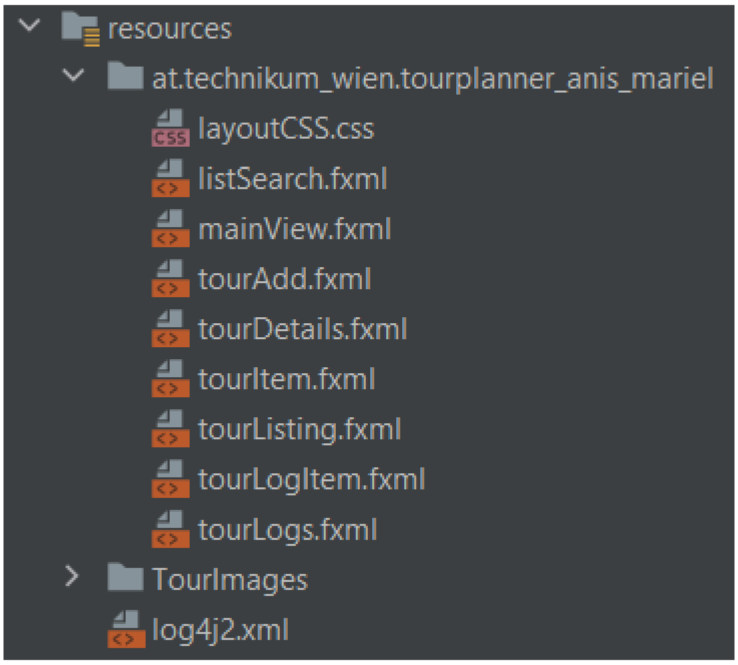
The logger package is used for logging during the development with the log4j library. We implement it everywhere by calling its factory which creates a logger implementing the ILoggerWrapper interface.



```
logger
    ILoggerWrapper
    InitialzedState
    Log4j2Wrapper
    LoggerFactory
    LoggerStateBase
    UninitializedState
```

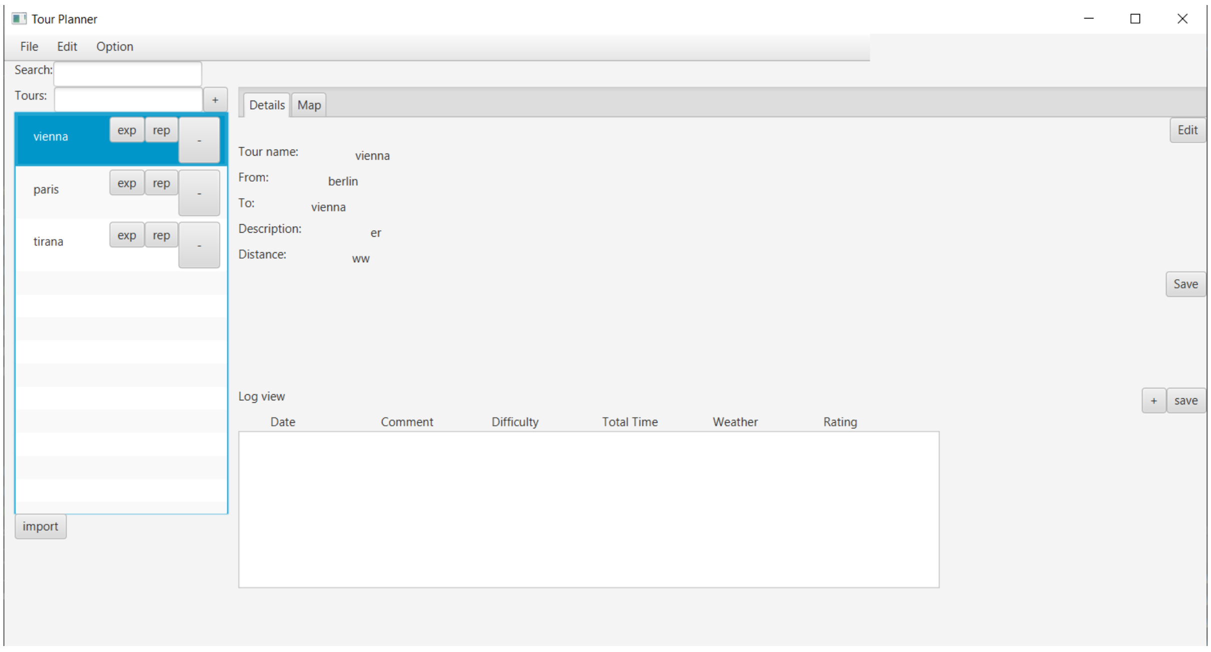
```java
private final ILoggerWrapper logger = LoggerFactory.getLogger();
```

The resources folder contains all the .fxml that were used in the project and also a tourImages folder where I store the images and later load them again in the program. There we also store the log4j2.xml file which we need for logging and where we defined the loggin specifics.



```
resources
    at.technikum_wien.tourplanner_anis_mariel
        layoutCSS.css
        listSearch.fxml
        mainView.fxml
        tourAdd.fxml
        tourDetails.fxml
        tourItem.fxml
        tourListing.fxml
        tourLogItem.fxml
        tourLogs.fxml
    TourImages
    log4j2.xml
```

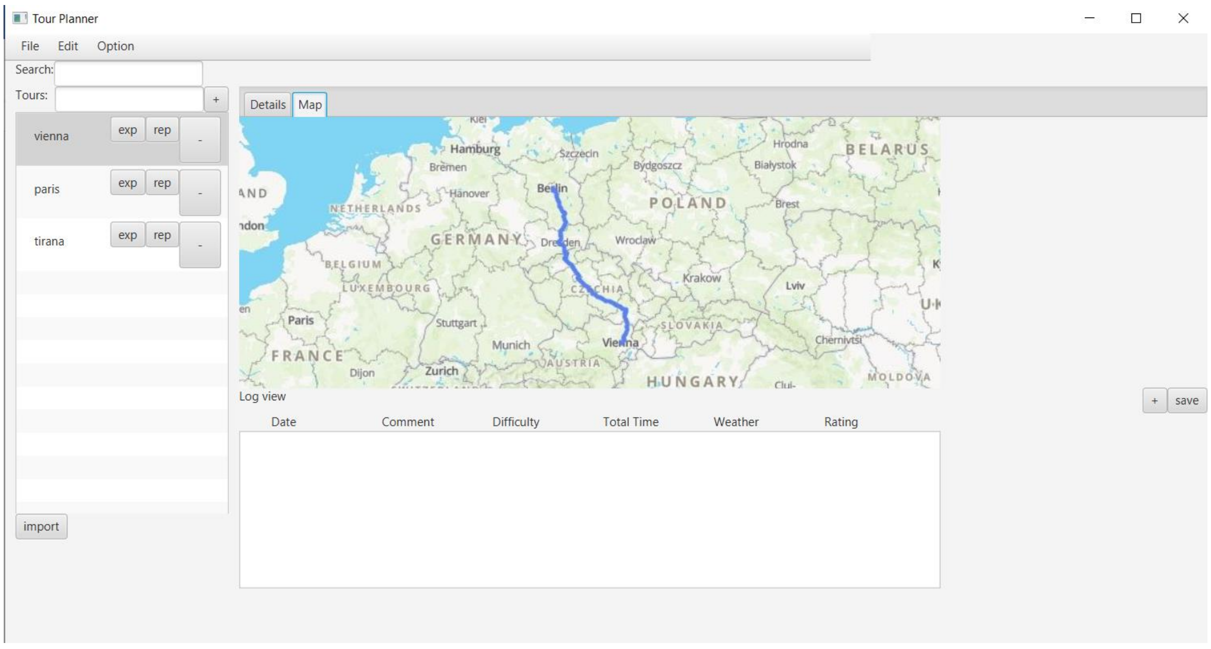
UI/UX design



Tour name: vienna
From: berlin
To: vienna
Description: er
Distance: ww

Log view

Date	Comment	Difficulty	Total Time	Weather	Rating



Log view

Date	Comment	Difficulty	Total Time	Weather	Rating

I used and observable List to add the Tours on the left side and I gave each element three buttons which are exp (as .txt), rep(as .pdf) and – for deleting the element from the list and also from the database. The import button at the end of the list can add an element into the list but only its name + (imported).

Design Patterns

Observer:

Used for the tour elements on the list and also for the tour logs, but unfortunately the tour logs were not implemented.

Singleton:

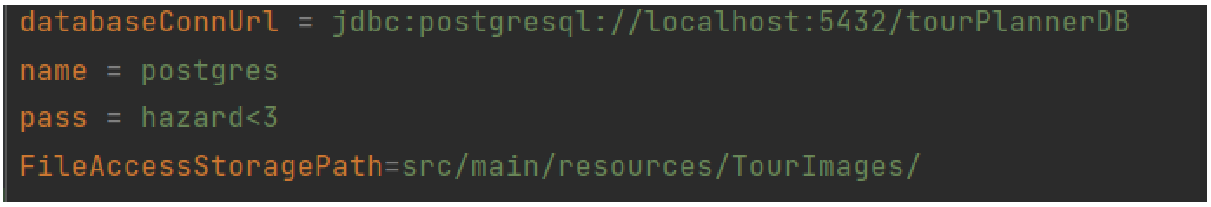
Used for giving a single instance of their respective classes from each layer.

Factory:

Used to give an instance of the controllers and also in the dataLayer and businessLayer to give an instance of the classes we needed by implementing also their interface.

Configuration File

The Configuration file was used to store the database strings like connectionString, name and pass and also to store the fileAccessPath for image storing.



```
databaseConnUrl = jdbc:postgresql://localhost:5432/tourPlannerDB
name = postgres
pass = hazard<3
FileAccessStoragePath=src/main/resources/TourImages/
```

Lessons learned

I learned a lot this project including the layered architecture as well as implementing different design patterns. What I enjoyed the most was the in-class coding sessions, which helped me a lot to understand how certain things work and how to implement them on my own. Also, I learned during development how much easier it is to divide your work and have each class and method not do more than needed, otherwise it gets trickier and trickier to develop them further. I enjoyed a lot working with the layered architecture and it showed me how important it is to set up everything right from the beginning and pick up from there. The log4j library is also a great point that needs mentioning. Before I never used to put the logger into a single package but I used the LogManager everytime I needed to log something. The implementation of the logger package made it a lot simpler and, in the future, if something changes in the LogManager I only change the package and not every instance of the logger. I had difficulties at first since I was not used with the Layered architecture but once I got the hand of it, it became simpler and more enjoyable.

Github Link:

Everything was developed, committed and pushed on the **anis_dev** branch not on master.

https://github.com/ShkembiAnis/TourPlanner/tree/anis_dev