FH University of Applied Sciences TECHNIKUM WIEN

TOUR_PLANNER

SWEN2

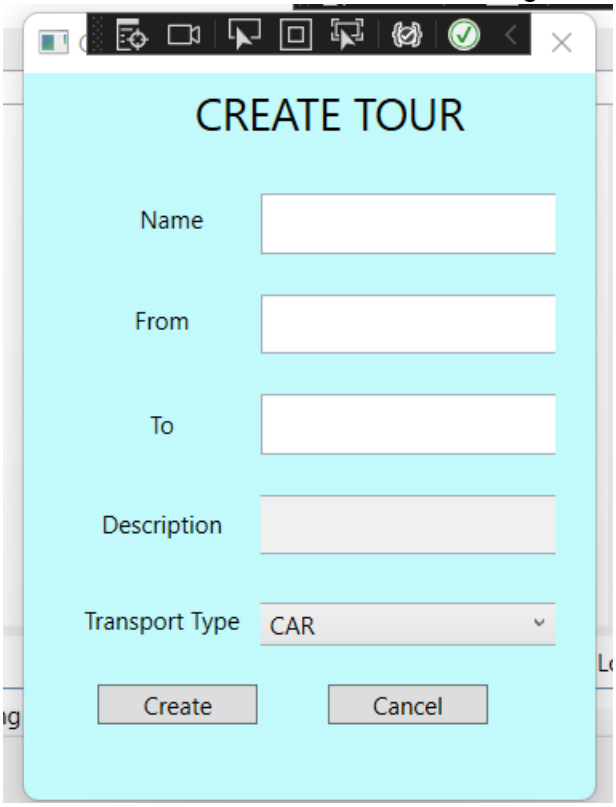
Christian Lauer (if21b064)

Introduction

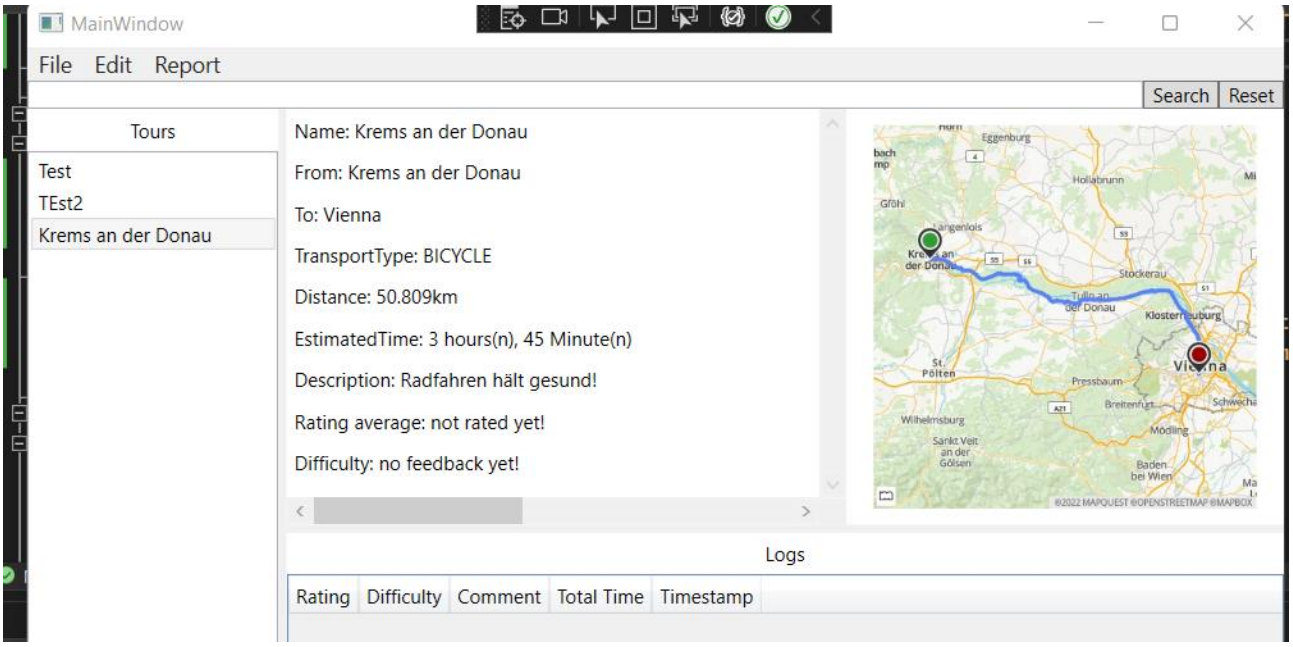
1. Due to a hardware failure this program was a last-minute implementation, but it has got all features which are stated in the excel sheet.
2. All must-have criteria are met.
3. Databinding was used for all inputs and fields where it makes sense.
4. The MainWindow is resizeable and has got some Gridsplitters.
5. The reusable Component is the MapView which is a grid for the map visualization
6. All CRUD operations have been implemented.
7. Tours have computed attributes (average rating and a overview about the difficulty)
8. User-input is validated via regular expressions (for creation and updating of tours/tourlogs) and there should be no crashes (at least as far as I tested)
9. Tourlogs are listed in a DataGrid when the corresponding tour is selected.
10. A full-text search was implemented. (without input validation because it filters the properties locally and there are no DB accesses)
11. Report-Generation was also implemented with iText.
12. Import and Export of Tours was also implemented (Format: JSON)
13. There is a strict separation between layers. (Only the search is handled in the ViewModel because it filters local data)
14. Own Exceptions have been implemented.
15. MapQuest Directions API was used (UNIQUE FEATURE -> Calculations are different if you select CAR/Bicycle and so on)
16. All data is stored in a postgre DB (see DB Schema)
17. There is a Configuration file for the most import values (DB String, MapQuestKey,…)
18. Serilog was used as a logging tool

Use-Cases

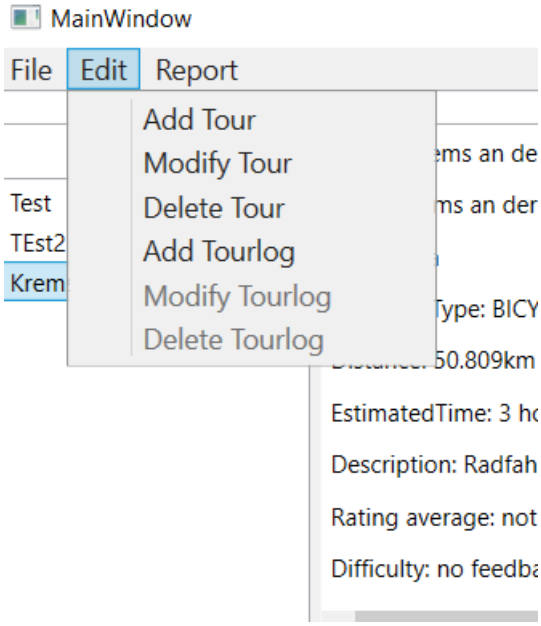
1. Users can create a Tour with the following Form:



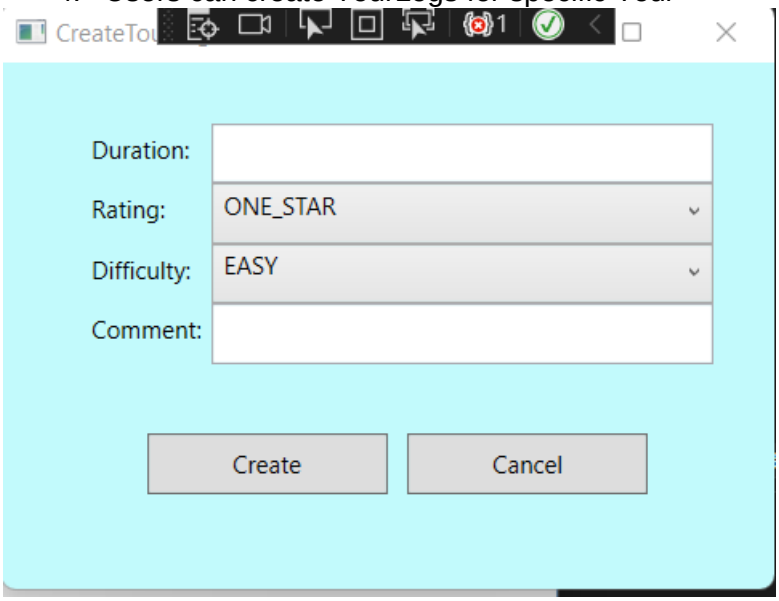
2. Users can select the tour on the left side



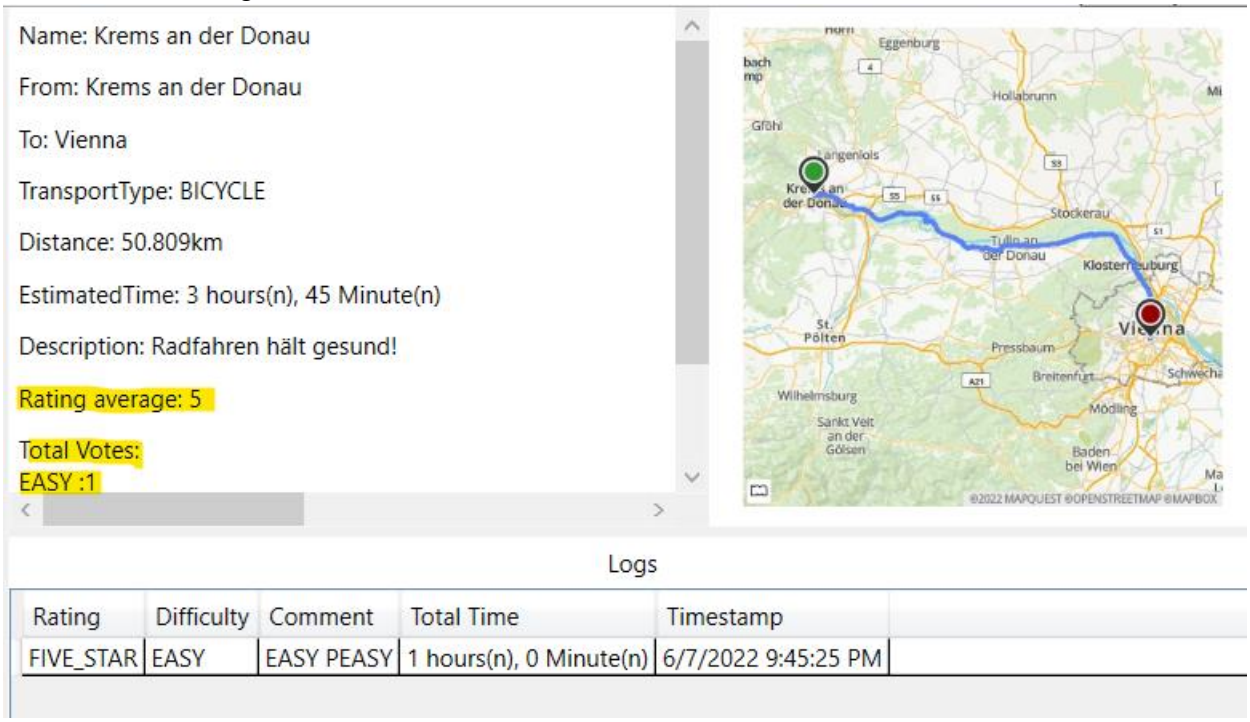
3. Users can delete / modify a selected Tour (only possible if a tour is selected)



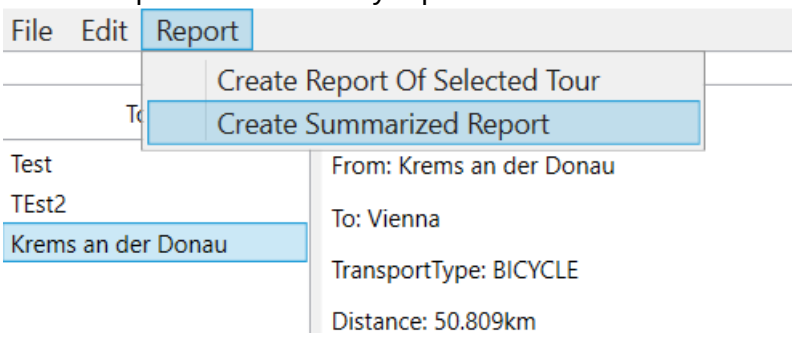
4. Users can create TourLogs for specific Tour



5. After a tour log was added there are new calculated values for the tour and the Logs are shown in the Datagrid



6. Reports and summery reports can be created:



7. Screenshot of a report:

Test

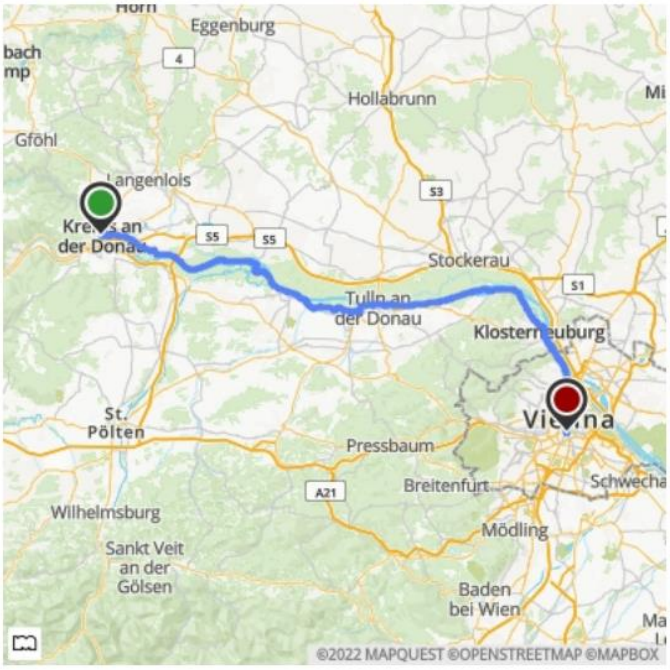
Description

Nach WIEN! MIT DEM FAHRRAD

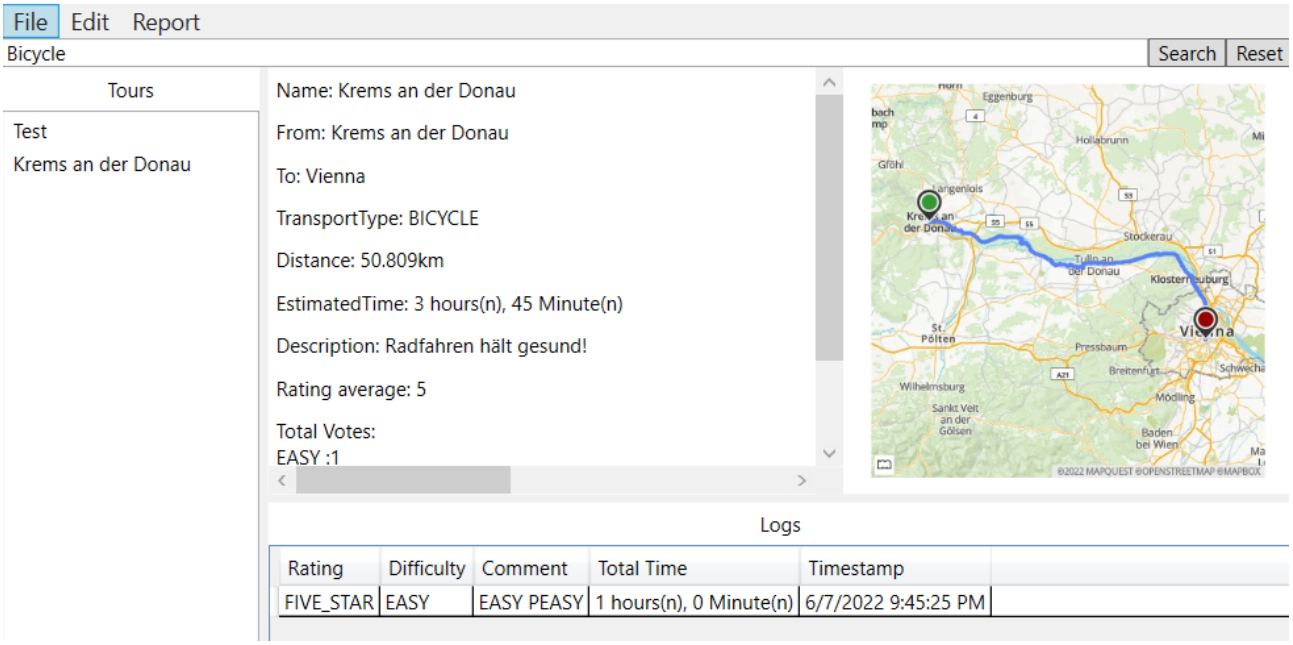
BICYCLE

From: Krems an der Donau

To: Vienna



8. There is a full-text search e.g. if a user searches for bicycle:



Most important classes / packages

1. Business Logic
   Here you can find all methods/classes which interact with the DataAccessLayer e.g.: The TourLogic class handles the creation/deletion/update of Tours, the IOLogic handles Import/Export of tours, the ReportCreator creates the pdf reports and so on

2. DataAccessLayer
   Here you can find the Repository, the UnitOfWork, the config mapper and the IOHelper (Which could be used to save images locally but in my case now everything is stored in the DB)

3. GUI
   Here you can Find all ViewModels and Windows. The ViewModels are calling the TourLogic/TourLogsLogic for the DAL access and logical handling of the operations

4. MAPQUEST
   Includes a mapping class for the mapquest responses and a handler for the requests.

Implemented Design-Pattern

I used the UnitOfWork-Pattern for accessing the repositories via transactions which are committed when the unit is disposed. In the unit repositories are only instantiated once (lightweight singleton-pattern because the repositories themselves have public constructors -> could be changed to fully singleton)

The command pattern was used to bind functionality to different usercontrols and to bind them to requirements (e.g. that a tour must be selected to click a certain button)

UX/Libraries

I'm not really a frontend developer therefore the UI is very simple the only choices I made:

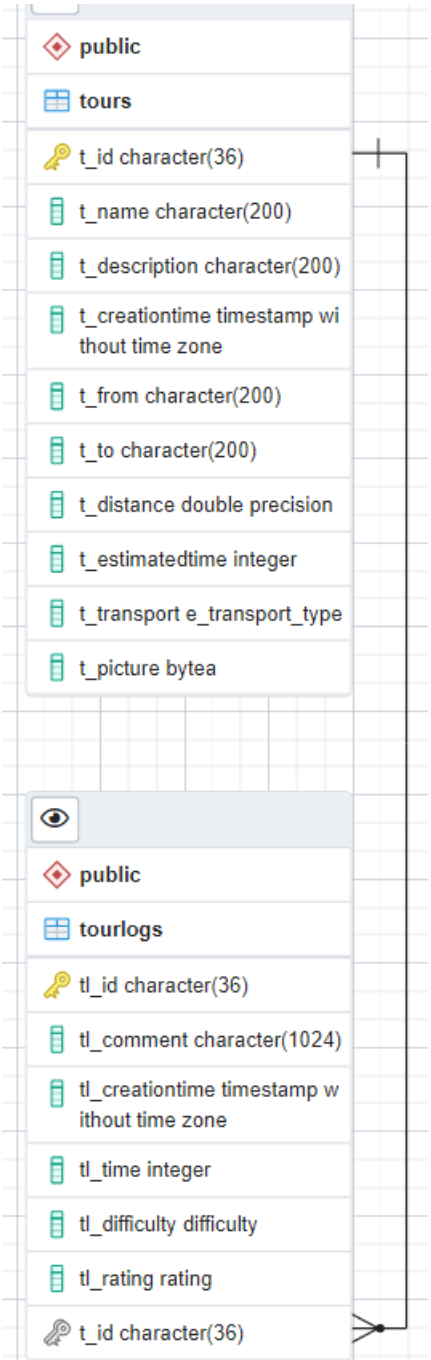
- Everything is managed via the top menu -> less buttons
- I used scrollviews and grid splitters hence the UI can be adapted for the user (choice between bigger map or bigger data grid)
- Libraries:
  - Serilog for logging because the global logger is easy to use
  - iText for the pdf creation because I saw other people use it and it seemed quite easy as well
  - NUnit for testing
- I worked mostly with grid rows and columns

Implementation

Models

For the models I used a minimal approach due to the extreme lack of time (described in the lessons learned)

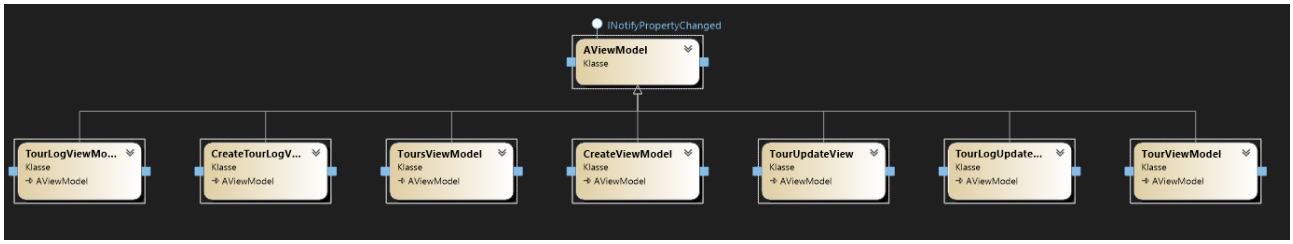
The DB Schema:



As you can see everything is stored in the DB (also the images!)

Class Diagrams:

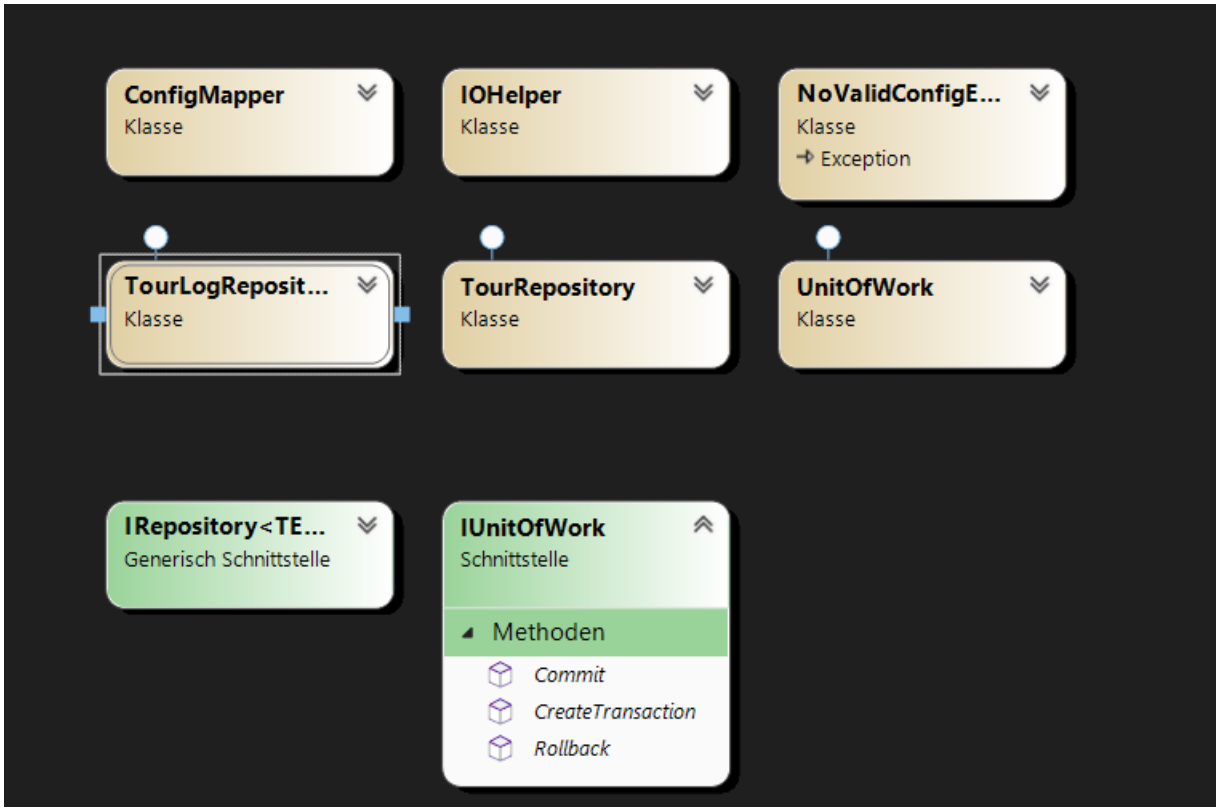
ViewModels



Windows

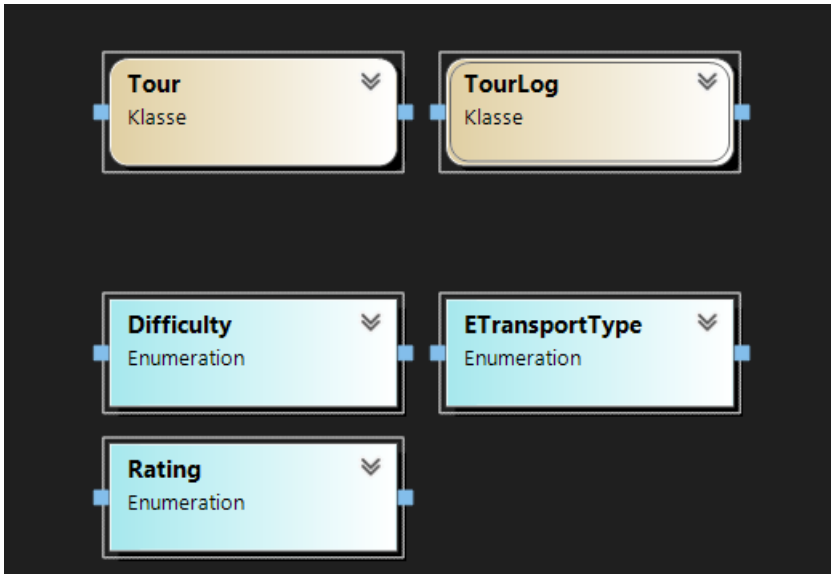
Businesslayer

Data Access Layer



MapQuest

Model



Architecture

The Program is structured into the following parts:

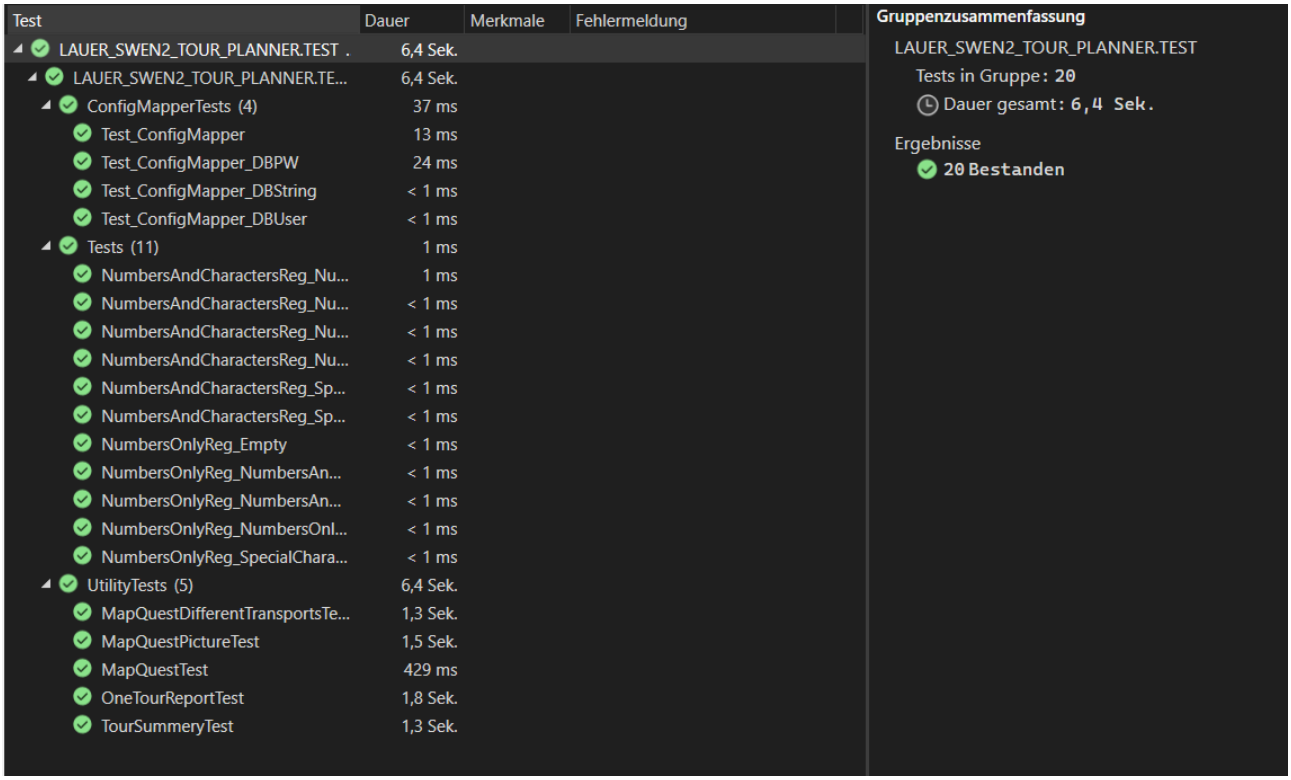
1. Model which contains the Tour and the TourLog classes
2. REST_API which is just a minimalistic approach due to lack of implementation time
3. TEST which contains the unit tests
4. MAPQUEST. This package includes the MapQuest-Response mapping-class and the handler for the tour/picture-requests
5. GUI which includes the ViewModels, the Windows and the reuseable Component

Unit Tests

For the Unit tests I chose to test the regular expressions which are used to validate the user inputs because I had some troubles here (because I'm not very used to regular expressions).

Moreover the config mapping has been tested with a single serialize/deserialize check.

The last tests I've implemented are about the Utilities of the system (Mapquest requests, pdf generation,…) because they were new to me and therefore error-prone.



Lessons Learned

1. Even if I worked alone, I saw how important version control systems like GitHub are. Due to a hardware failure I had to start over four days before deadline because I used a local version control (SVN) and had no remote repository.
2. Refreshed my WPF skills (especially the MVVM Pattern and how it is done correctly) and could earn some valuable experience with PDF creation systems.
3. I learned that I have to work on my time management and should stop to do things “right on time”, next time a hardware failure could “knock“ me out.
4. Refreshing of my knowledge about regular expressions

Tracked Time

04.06.2022 10hours

05.06.2022 10 hours

06.06.2022 12 hours

In total 32 hours

Git-Link: R3KAR0/TOUR_PLANNER (github.com)