FH University of Applied Sciences TECHNIKUM WIEN

PROTOCOL

Degree: Bachelor Computer Science

Subject: Software Engineering 2

Meeting-Planner

Creators: Josua Hämmerle
Personenkennzeichen/Id: if21b258

Professor: Raoul Holzer MSc

Vienna, 24.10.2023

Design and architecture

The Metting-Planner project is based on the GUI framework WPF and written in C#. The project is organized into 3 main layers. A UI-layer which contains the views and view-models, a business-layer implementing the main logic of the project and a data-access-layer communicating to the database.
It implements the MVVM-pattern, using models to hold the data, views to define the UI, and view-models to apply logic and keep the models and views synchronized. To communicate from the views to the view-models the Command-pattern is used, which allows to bind to a command defined in the view-model and execute it.
Keeping the different view-models synchronized is done via the Observer-pattern. This pattern uses events to broadcast changes to each object which is subscribed to the event, so it can adapt to these changes.
As a persistent storage solution, Microsofts Entity Framework is used, which allows to implement repositories. These handle CRUD operations on the database. The string for the database access is stored in a config file for easy customization.
Furthermore, an IoC container is used to handle the creation of objects when needed.

Technical steps and decisions

As this Meeting-Planner project was very similar to the Tour-Planner project I did before in cooperation with my colleague Felix Riemenschneider, I decided to take the Tour-Planner as a baseline and refactor existing code instead of building the project from the ground up.

At first, I had a look at the new requirements and removed any classes which would not be used anymore. These were mainly classes

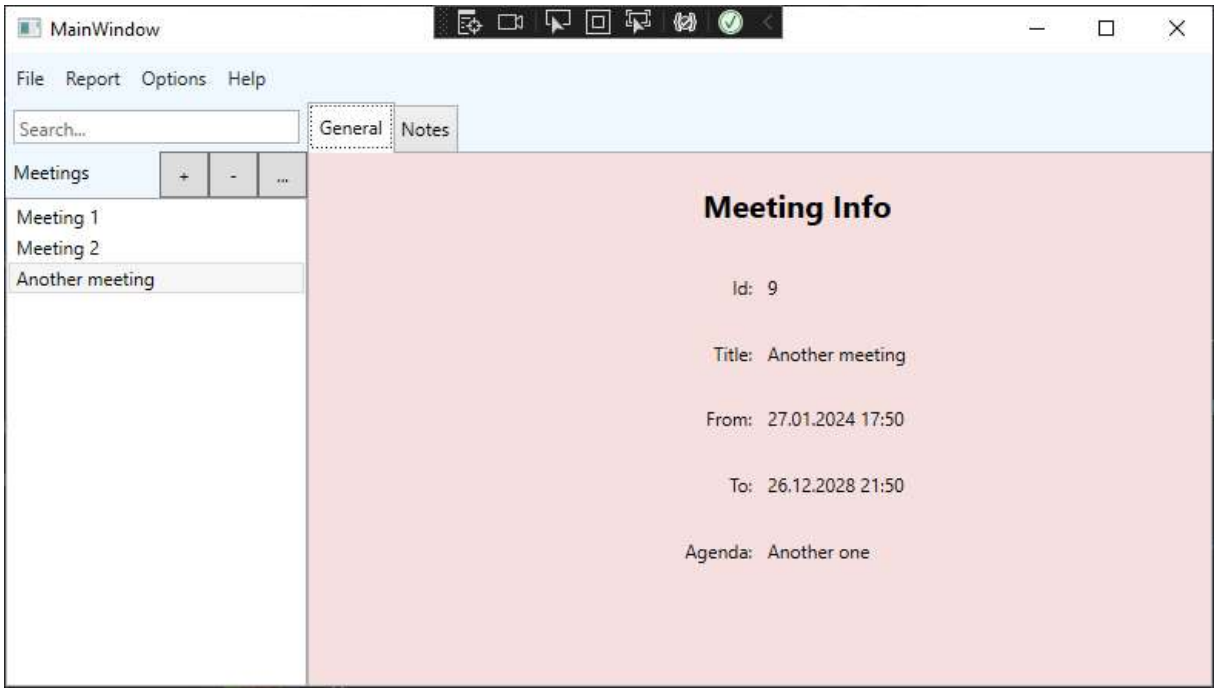
interacting with the RouteService as there was no requirement for an external API call in the new Meeting-Planner project.
After all unnecessary classes were removed, I did a quick search and replace, to replace any occurrences of Tour with Meeting and some other renaming to have a clean baseline.
As a next step I decided to start from the backend and work my way to the frontend. My reasoning for this was that if I changed classes in the backend, as the frontend would build up on them, I could just follow the error messages and fix most of the problems this way.
Adapting the models to conform to the new requirements was my priority as these did not depend on anything else but most classes depend on them. There wasn't much refactoring to do in the backend as the search and replace did most of the work already, and functionally the DAL, MeetingService and LoggingService stayed the same. So, after some small changes to validation and report generation the backend was done.
Next, I decided to work on the view models where I mostly had to adapt the fields to the changes in the models. Afterwards I adjusted the views to comply with the changes in the view models and made some small UI changes where things started to look ugly.
When all this was done, I fixed some small bugs, and the main part of the project was finished. As a next step I worked on the unit tests.

Unit Tests

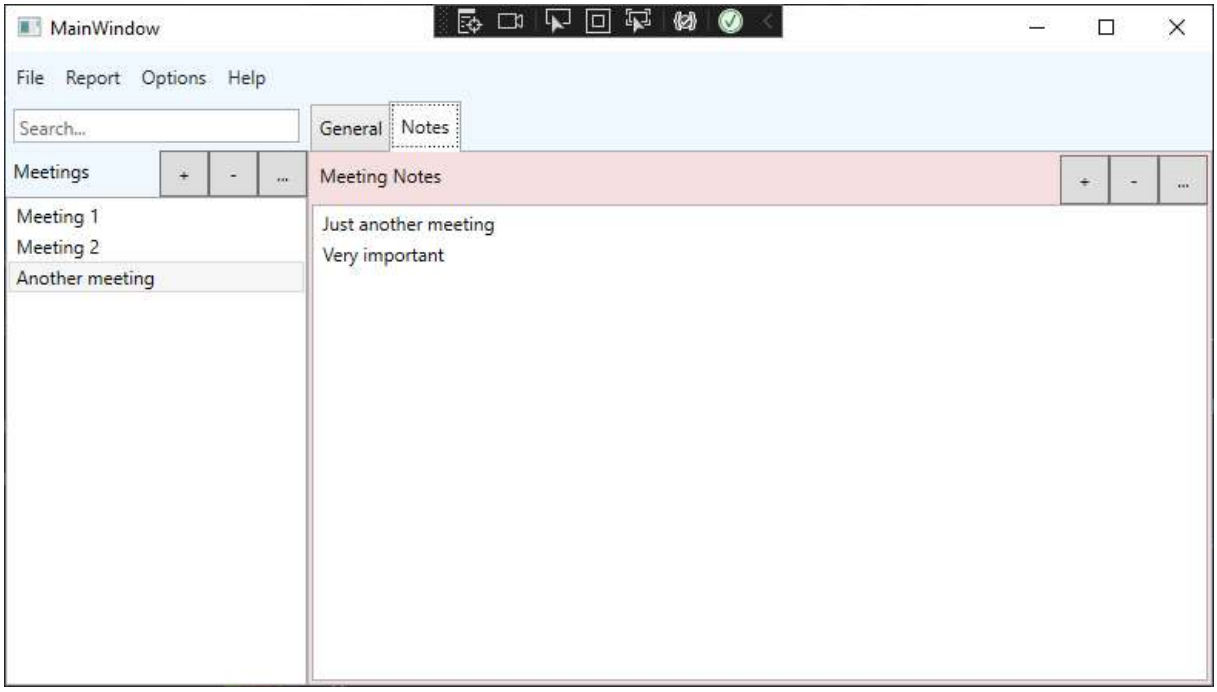
For the Unit-tests I decided to focus on the MeetingService because the handling of data received from the frontend, is a key part of the project. I also thought it was insightful to work with a in-memory database and learn how to create unit tests for that. Furthermore I wrote some tests to verify that the input validation is working correctly.

UI-flow

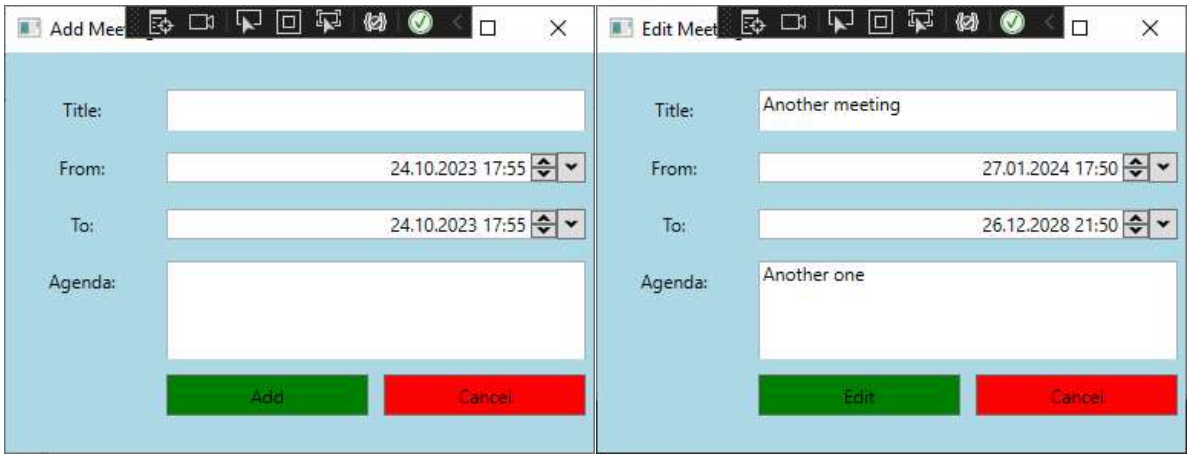
Main window (general info):



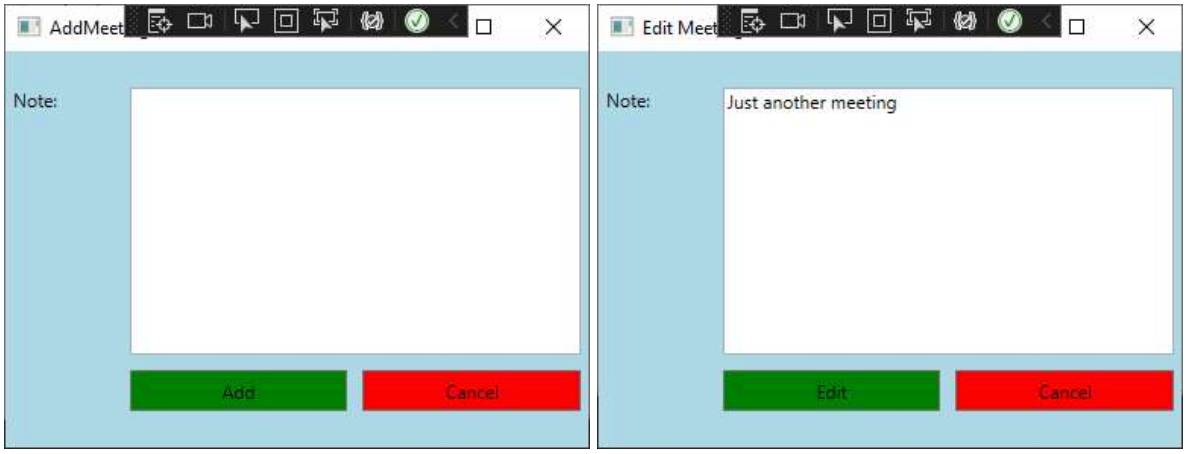
Main window (notes):



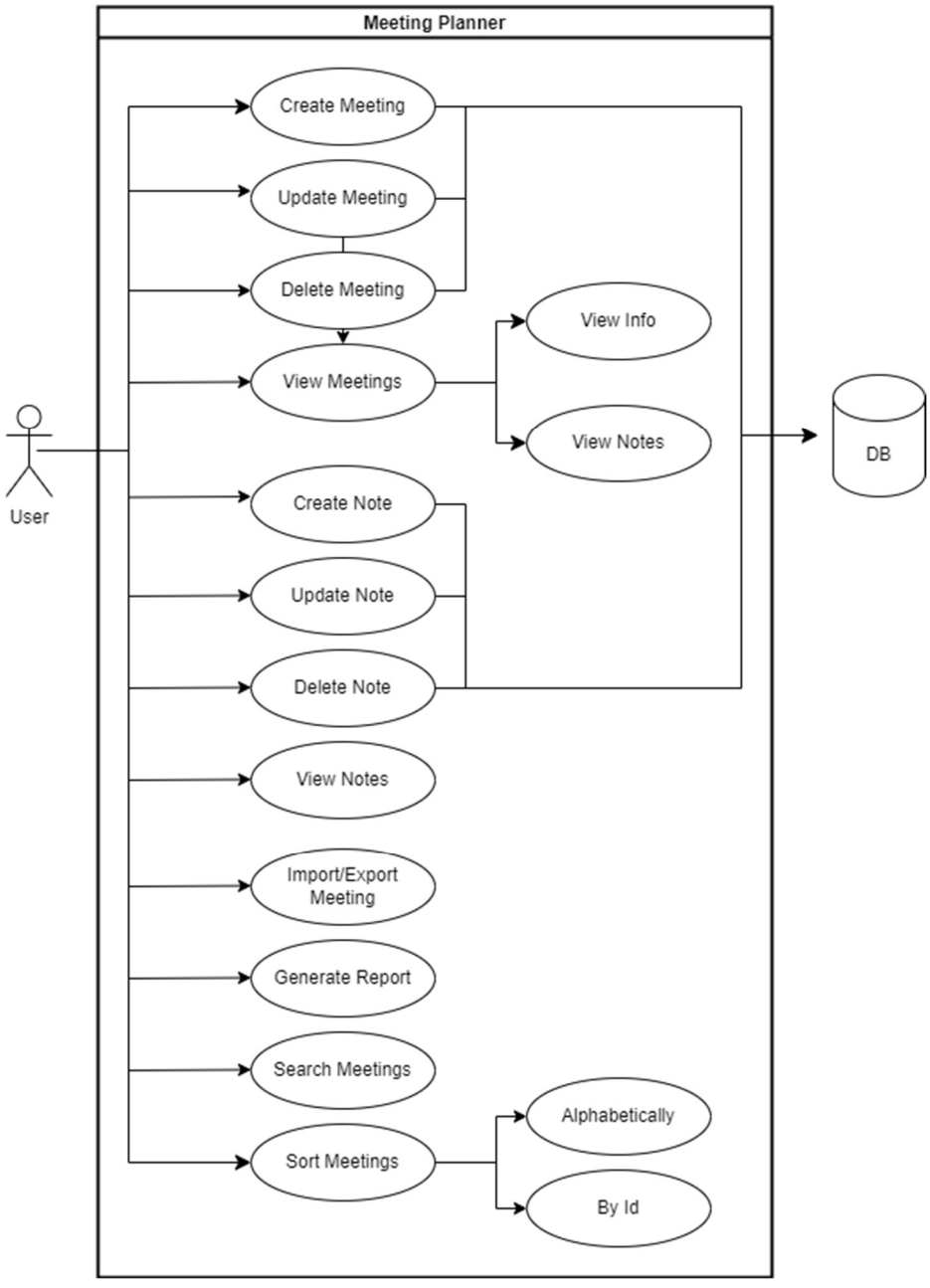
Add/edit meeting:



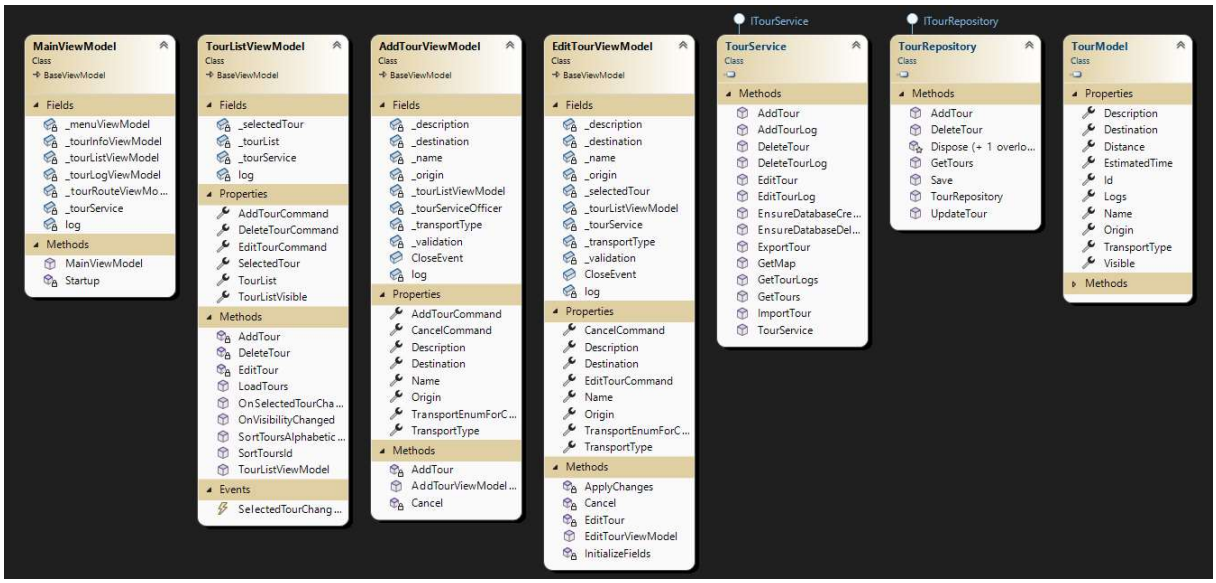
Add/edit note:



Use-case diagram

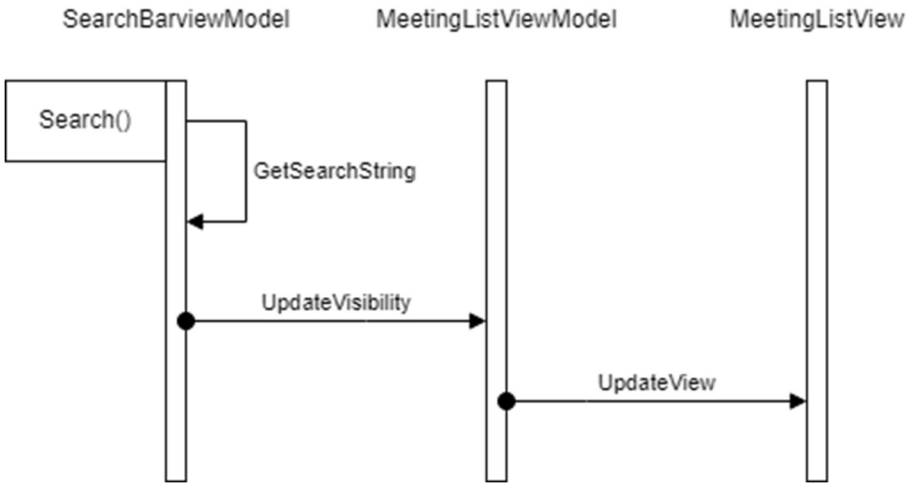


Class-diagram



Full-text search

Sequence-diagram:



When the user types anything into the search-bar at the top of the UI, the SearchBarViewModel will create a regex for the input. Then it will get a search-string consisting of all the relevant data in the meeting-list including the associated notes and compare it to the regex. Any meeting that does

not match the regex will be set to invisible on the MeetingListViewModel which, after a property update, gets reflected in the MeetingListView.

Tracked Time

Date	Josua
14.10.2023	2h
15.10.2023	3h
21.10.2023	2h
24.10.2023	4h
Total	11h

Link to Git:

https://github.com/if21b258/MeetingPlanner