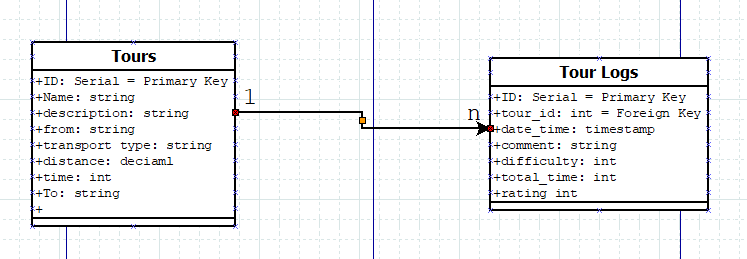
TourPlanner

**Database:**

The database consists of two Tables; one for the Tours and one for the Logs. They are linked by the ID of the Tour.



The Table “Tours” has a Primary Key “ID” that is automatically assigned by the database. The other columns are the required attributes for the assignments that are displayed on the User Interface.

The Table “Tour Logs” also has a Primary Key “ID”, but it also saves the ID of the Tour it belongs to in “tour\_id”.

The communication is handled by NPGSql from ADO.NET. The Connection String to the database is saved in the App.config file, which can be accessed using the Configuration Manager. We have also created a separate class that is responsible for all communication between our application and the database

**Layers:**

We used Layered Architecture to structure our application more efficiently and the structure is as follows:

* Data Layer
  + Models
  + Database
* Business Layer
  + JSON
  + Logging
  + Mapquest
  + PDF
* Presentation Layer
  + Views
  + ViewModels
  + Images
  + DioalgService
  + Commands

**Approach**

Firstly, we divided our solution into three layers:

1. Presentation Layer
2. Business Layer
3. Data Layer

We decided to create the UI first to easier orient ourselves. We then created the Models of the data structures we needed for the project, like the Tours, Tour logs, Tour information etc. We then created our main ViewModel for the MVVM and tested it using dummy data. Afterwards, we created the database and worked on fetching the data from the MapQuest API. After implementing the API part of the assignment, we focused on the logging and PDF creation parts of the assignment. Lastly, we created Unit Tests using the NUnit module and wrote this protocol.

1. **Presentation Layer**

In the Presentation Layer we grouped things that are closely related to our UI, i.e. the Views themselves, the ViewModels, the Commands that implement the ICommand interface, but also the Tour Images that are acquired from the MapQuest API are also saved in this layer.

The most notable part of this layer would be our primary ViewModel, the “TourViewModel”. This ViewModel is closely linked to our main UI and is a catalyst for all of the actions the user wishes to make.

1. **Business Layer**

The business layer handles most of the work. It contains the classes responsible for the communication with the API, managing of Tour Images, PDF creation, logging, export/import of data etc.

1. **Data Layer**

This layer is responsible for the communication between our application and the database. It also contains the models/data structures we use for managing the data received from the MapQuest API.

**Design Pattern**

For the database, to ensure that only one connection is open at a time, we used the Singleton pattern.

**Unit Tests**

Our unit tests focus on ensuring the functionality of our functions.

**Unique features**

As unique features we have implemented a refresh button, we can now easy refresh the tours if some data has gotten added to the database outside of the program.

**Input Validation**

We essentially allow the user to input whatever they want. By using prepared statements, we made sure that SQL-Injections are impossible and the user is not restricted in any way.

**Libraries**

We used the following libraries:

* NPGSql for handling the connection to the database and communication
* IText7 for creation of PDFs
* Log4Net for logging
* Drawing.Common for displaying and saving Tour Images

**Tracked Time & GIthub**

We spent approximately 30h on the project.

Github: <https://github.com/Garucx/TourPlanner>