**Protocol – Tour\_Planner**

**App architecture:**

The App is split in 3 layers: User Interface (UI), Business Layer (BL) and Data Access Layer (DAL).  
The User Interface includes the MVVM pattern: Model – View – ViewModel. When starting the application, the App.xaml.cs creates the ViewModels and the MainViewModel. The ViewModels are the bound to the Views (Databinding) and get/show the data.  
In the BL the logic is implemented. The Controllers are called from the UI, handling requests like adding and deleting tours/logs, create the map for the tours through the MapQuest calls and then they send the data to the DAL. The two DataHandlers in the DAL handle the incoming requests from the BL, get the SQL Statements from the SqlCommand classes and return the data from the postgre sql database.

**Use Cases:**

Use Case 1: Add Tour  
The User adds a new tour via the User Interface. In the UI the user enters the data of the tour and this data is sent to the Tour Controller in the Business Layer. The Tour Controller creates the map with the MapQuest included and sends this data to the TourDataHandler in the Data Access Layer. There the TourDataHandler saves the data in the postgresql database with the provided SQLCommands.

Use Case 2: Delete Tour  
The User selects a tour from the Shown Tours in the UI. The data is sent to the Tour Controller which calls the TourDataHandler to delete the selected Tour with the provided SQLCommands.

Use Case 3: Add Log  
The User selects a tour from the Shown Tours and adds a new Log to it. The selected tour and the data input from the user is collected in the UI and sent to the LogController. The LogController in the Business Layer calls the LogDataHandler in the Data Access Layer which saves the new log in the database with the provided SQLCommands.

Use Case 4: Edit Tour  
The user selects a tour from the Shown Tours in the UI and enters the changes. The data is sent to the TourController which creates a new map with the new data. Then the TourController calls the TourDataHandler which saves the changed tour in the database with the provided SQLCommands.

Use Case 5: Create Report  
The User selects a tour from the Shown tours in the UI and creates a new Report. The UI sends the data to the ReportController in the Business Layer which creates the PDF. Then the TourDataHandler in the Data Access Layer is called to save the report in the database.

Use Case 6: Edit Log  
The User selects a tour from the Shown Tours to show all logs to that tour, clicks on edit tour and enters the new log data. The data input from the user is sent to the LogController in the Business Layer, which calls the LogDataHandler in the Data Access Layer. The LogDataHandler saves the changed log in the database with the provided SQLCommands.

Use Case 7: Delete Log  
The User selects a tour from the Shown Tours to show all logs to that tour, selects the log which should be deleted and clicks on delete. This calls the LogController from the Business Layer including the log data. Then the LogController calls the LogDataHandler in the Data Access Layer and deletes the log from the database with the provided SQLCommands.

Use Case 8: Import/Export Report

**UX, library decisions, lessons learned:**

Library decisions

We decided to use the following libraries because they are well documented and easy to use. In addition, it was required to integrate some of them in the project.

* Log4net for logging
* iText for generating PDF reports
* Newtonsoft for serializing and deserializing JSON Objects.

Lessons learned

* Working in a team, it was the first time for both of us to work on such a project with a colleague.
* Working under stress since we had only two weeks to bring our project to work. We thought that we will submit it in September, so it was quite challenging to submit it in such a short time span.
* More understanding of the idea of MVVM and getting to know WPF. Also data binding was a big challenge for us.
* Working with git. This project helped us a lot to know how to work with git more efficiently.
* Time planning.

**Design pattern:**

We decided to use Singleton for the database to ensure that the class has only one instance and that instance can be accessed globally.

**Unit Testing Decisions:**

**Unique Feature:**

We decided to include a Help Page with a FAQ for the Tour Planner as unique feature. It contains questions and answers about the functionality of the application.

**Tracked time:**

Sum: ~150h

**Git Link:**

https://github.com/BeccaDreams/SWEN2\_Tour\_Planner