Tourplaner – Protokoll

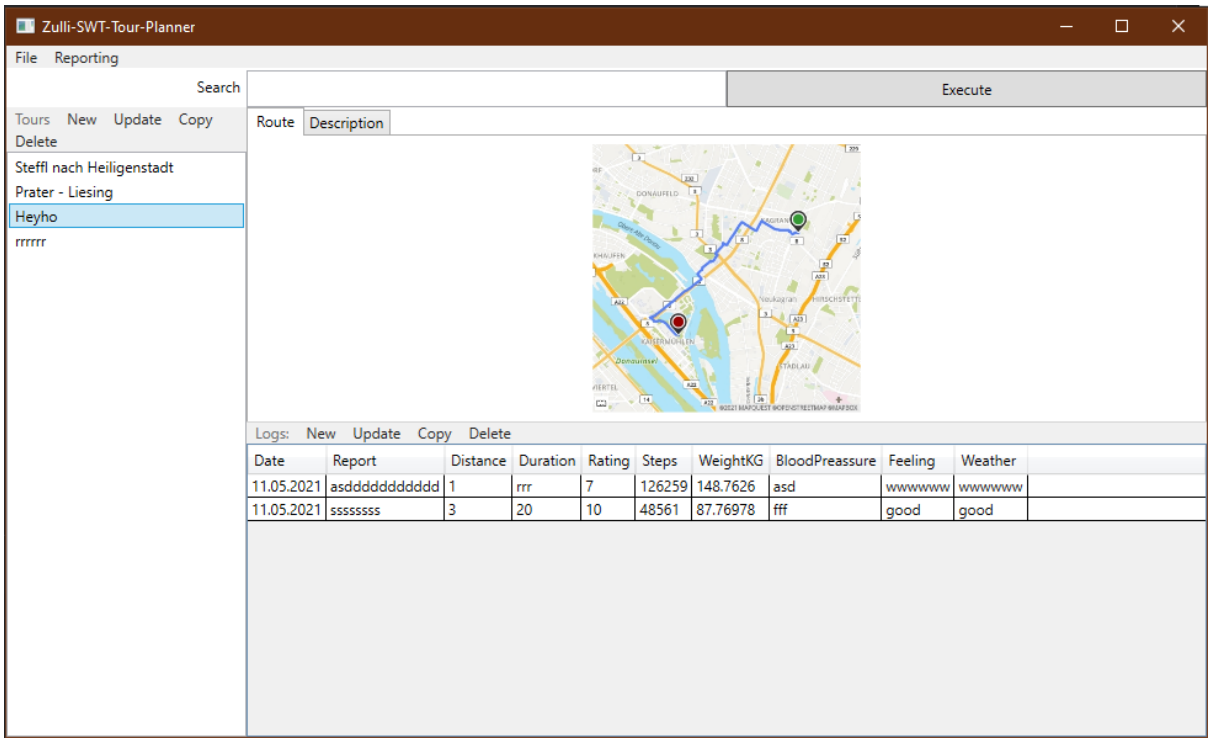
Maximiliano Zulli (if19b183) /// BIF4 - SS2021

Inhalt

1 Design 2
1.1 UI 2
1.2 MVVM Pattern 3
1.3 Layers 3
1.4 Database 3
1.5 Design Patterns 4
1.6 Report Library 4
1.7 Configuration 4
2 Lessons Learned / Failures 4
3 Unit Test Design 5
4 Time Spent 5
5 Link To Git 5
6 UML 6

1 Design

1.1 UI



Das UI ist wie man unschwer erkennen wird stark an das vorgeschlagene Beispiel in der Angabe angelehnt.

Oben gibt es ein Kontext-Menü mit den Optionen „File“ und „Reporting“.
- File dient zur Auswahl zwischen „Import“ und „Export“ von Tours mittels eines JSON Files.
- Report dient zur Auswahl zwischen der PDF-Erstellung von einer Tour, allen Touren, oder einer statistischen Zusammenfassung von allen gespeicherten Touren und Logs mit z.B. der Gesamtlänge aller Touren, oder der gesamten gegangenen Kilometer.

Unter dem Kontext Menü befindet sich eine Suchleiste, hier kann im Suchfeld eine beliebige Zahl oder ein beliebiges Wort eingegeben werden. Nachdem „Execute“ gedrückt wurde, werden nun nurmehr all jene Touren in der Liste links angezeigt, welche entweder in den Tourinformationen diese Daten beinhalten oder deren Logs sie beinhalten.

Auf der linken Seite des Bildschirmes befindet sich ein eine Anzeige aller Touren, aus welchen immer jeweils eine ausgewählt werden kann.

Wenn eine Tour gewählt wurde, wird nun im Hauptbereich der App eine Route angezeigt, sowie alle gespeicherten TourLogs zu dieser Tour. Es ist auch möglich alle Informationen zu einer Tour unter dem Tab „Description“ anzuzeigen.

Wenn eine Tour ausgewählt ist, kann ein Tourlog hinzugefügt, geändert, kopiert oder gelöscht werden. Dafür kann man das Kontext-Menü im Log-Bereich nutzen. Ändern, Kopieren und Löschen ist nur dann möglich wenn ein Log aus der Liste ausgewählt ist.

Mit der Auswahl einer Tour ist es außerdem möglich die darüberliegenden Buttons zu nutzen. Ändern, Kopieren und Löschen einer Tour funktioniert nur dann.

Um eine Neue Tour hinzuzufügen kann im Tour-Menü auf „Add“ geklickt werden.

Wenn eine Tour oder ein Log erstellt oder aktualisiert werden soll, öffnet sich jeweils ein neues Fenster, in welches die Daten eingetragen werden können.

Die gesamte UI wurde nur mit Hilfe von XAML und WPF erstellt.

1.2 MVVM Pattern

Das Model View ViewModel Pattern wird in meinem Fall dadurch eingehalten, dass es eine MainViewModel-Klasse gibt, welche Properties beinhaltet die in jedem UI Fenster binded werden können. Außerdem werden Commands verwendet welche von der UI auch aufgerufen werden können, um Funktionalität im Programm zu gewährleisten.

1.3 Layers

Beim Layer Pattern geht es darum, dass eine Schicht nur mit der jeweils darüber oder darunterliegenden Schicht kommuniziert, so soll eine geringe Kopplung der einzelnen Klassen gewährleistet werden.

In diesem Programm, kommuniziert die UI nur mit dem User und dem Main View Model. Das Main View Model hingegen nur mit dem Business Layer, welcher darunter liegt und der UI die darüber liegt. Das MVM beinhaltet alle Eigenschaften, welche im XAML gebinded werden können.

Der Business Layer beinhaltet alle Funktionen und Prozesse, welche für den „business“ relevant sind. In diesem Fall sind das, die Anbindung des DataAccessLayers, das Sortieren und Filtern von Daten und das Erstellen von Reports (PDFs).

Im DataAccessLayer werden die Unterschiedlichen Arten des Datenaufrufs implementiert. Dieses Layer kommuniziert nur mit dem Business Layer und z.B. dem File System und der Datenbank. Hier werden alle Aktionen die Daten speichern oder abrufen implementiert

Das Ziel ist es alle Layer so zu implementieren, dass sie Austauschbar sind, so dass jedes Layer in unterschiedlichen Umständen wiederverwendbar gemacht wird.

1.4 Database

Die Datenbank ist eine PostgreSql Datenbank und beinhaltet alle Informationen der Tour und des Logs.

Tours:

id INT | name TEXT | startlocation TEXT | endlocation TEXT | description TEXT | distance DOUBLE | mapimagepath TEXT

Logs:

Id INT | tourid INT | date TEXT | report TEXT | distance INT | duration TEXT | rating INT | steps INT | weightkg DOUBLE | bloodpreassure TEXT | feeling TEXT | weather TEXT

1.5 Design Patterns

In diesem Projekt wurde nur eine Designt Pattern verwendet, da es für die anderen keinen triftigen Verwendungsgrund (bis auf die Aufgabenstellung) gegeben hätte: der Singleton

Singleton:

Der Singleton ist eine Klasse, welche im gesamten Programmverlauf nur einmal instanziiert wird. Der Konstruktor ist in dieser Klasse privat, weshalb von außen nur eine spezielle statische Funktion aufgerufen werden kann, welche immer eine Instanz der Klasse zurückgibt.

Falls bereits eine Instanz der Klasse existiert, wird keine neue erstellt sondern diese zurückgegeben, falls es diese noch nicht gibt, wird eine neue erstellt, und die neue zurückgegeben.

Folgende Klassen sind in diesem Programm Singletons:
- DataLayerAccessManager
- RestDataClass (MapQuestAPI Klasse)
- PostgresSqlConnector

Bei diesen drei Klassen war es sinnvoller, nur eine Instanz zu haben, da diese Instanz mit externen Systemen kommunizieren muss.

1.6 Report Library

Für die Report Generation Library habe ich die PDF Library von Syncfusion gewählt.

Sie war einfach einzubinden und hat alle Vorrausetzungen für das Projekt erfüllt.

Man kann damit die Bilder der Tour einbinden und Logs in einer Tabelle darstellen.

1.7 Configuration

Für die Konfiguration des Programms wurde als Konfigurations-Library System.Configuration von .Net verwendet. Sie stellt die ConfigurationManager-Klasse zur Verfügung mit welcher man sehr schnell und einfach, Informationen aus einem „App.config“ File auslesen kann um diese im Code zu verarbeiten.

In dieser Datei befindet sich die Konfiguration für den Logger von log4net, der ConnectionString für PosgresSql Datenbank und neben dem MapQuestAPIKey auch alle Ordnerpfade welche für das Programm relevant sind, und statisch angegeben werden müssen.

2 Lessons Learned / Failures

Während ich die Unit Tests implementiert habe (am Ende des Projekts), viel mir auf, dass eine Factory Pattern für den DataAccessLayer doch keine schlechte Idee gewesen wäre. Mocking einer Datenbank ist nämlich nur dann möglich, wenn diese dynamisch ausgetauscht werden kann (was bei der Factory möglich ist aber in dem statisch gekoppelten Code den ich geschrieben haben nicht).

Sich in ein gänzlich unbekanntes Framework, mit Konzepten welche man noch nie gehört hat, einzuarbeiten ist sehr Zeitaufwändig, aber wenn man diese einmal grundsätzlich verstanden hat, nicht unmöglich.

Das MVVM Pattern mit dem Layer Konzept zu kombinieren ist gar nicht so schwer, hat mich allerdings sehr lange gekostet da sie recht kompliziert sind.

Funktionen des .Net-WPF Frameworks in den NUnit Projekten wiederzuverwenden kann problematisch sein (ConfigurationManager und App.Config -> testhost.dll.config).

3 Unit Test Design

Die Unit Tests sind bei mir in zwei Teile geteilt.

- JSON und Configuration Tests
- JSON :
    - Hier geht es primär darum, herauszufinden ob die JSON files welche von MapQuest oder anderen Quellen gespeichert werden auszulesen und als Tour zu speichern.
- Configuration:
    - Hier werden die Konfigurationen in App.config validiert.
- Funktionale Tests
    - Hier werden alle Teile des Programms getestet – Erstellung von Instanzen, Tours aus der Datenbank lesen etc.
    -

4 Time Spent

~79 Stunden

Wieviel Zeit für das Projekt aufgewendet wurde kann detailliert in der Datei „Zeitaufzeichnung.pdf“ entnommen werden.

5 Link To Git

https://github.com/SchuhFiedel/TourPlaner

6 UML

Der Code für das PlantUML-Diagramm befindet sich in der Datei: UML.txt