Protokoll – „1 Nicer Tour Planner“

Die Applikation ist in 3 Layer und ein Testprojekt aufgeteilt. Das Testprojekt ist ausschließlich für Unit Tests mittels NUnit verantwortlich. Im Layer **Models** befinden sich die Klassen Tour, TourLog und MapInfo, welche jeweils nur dafür zuständig sind die Objekte und deren Attribute zur Verfügung zu stellen. Während Tour und TourLog mehrfach Anwendung finden, wird MapInfo exklusiv für den Download der Map herangezogen.

Durch die Klassen im **DataAccessLayer** werden zunächst die Daten aus der Datenbank entnommen oder in die Datenbank gespeichert. Darüber hinaus ist das Tour-Data-Object hier zu finden, welches die als Kommunikationsmittel zwischen BusinessLayer und DataAccessLayer fungiert.

Um das Bild über die MapQuestAPI zu beziehen, existiert die Klasse HTTPConnection, die sich um das Handling der Request kümmert. Der ImageHandler kümmert sich darum, dass der korrekte Pfad zum Bild erstellt wird und das Bild heruntergeladen und somit in den Ordner Images gespeichert wird.

Zuletzt ist die Klasse JSONImport in diesem Layer, die den Import einer JSON-File über ein Windows Dialog-Fenster ermöglicht.

Im **BusinessLayer** befindet sich in verschiedene Ordner aufgeteilt die Handler der jeweiligen Funktionen. Einzig im Ordner Helper befindet sich die Helfer-Klasse Validator, die nur zu Validierungszwecken existiert. Alle anderen Klassen decken die einzelnen Funktionen ab. Die Logik der Funktionen ist vorwiegend in diesen Klassen zu finden.

Der **PresentationLayer** stellt die verschiedenen Views in Form von Fenstern zur Verfügung.

Das MainWindow wird beim Starten der Applikation angezeigt und deckt folgende **Funktionen** ab:

* Add Tour \*
* Delete Tour
* Modify Tour \*
* Copy Tour
* Import Tour
* Export Tour
* Print Tour \*
* Get Files \*
* Upload Files
* Add Logs \*
* Get Logs \*
* Search

\* es öffnet sich ein neues Fenster

Da manche Funktionen nur mit einer ausgewählten Tour verfügbar sind, werden jene Funktionen ausgeblendet.

Jedes Fenster hat ein eigenes ViewModel, das entweder dafür sorgt, dass ein neues Fenster geöffnet wird (z.B. Add Tour im MainWindow öffnet das NewTourWindow) oder aber, dass gewisse Handler des BusinessLayers aufgerufen werden (z.B. Import Tour im MainWindow ruft den ImportHandler auf).

Weiter befindet sich folgende Ordner in diesem Layer:

* Exports (exportierte Tours als JSON-Datei)
* Images (Map-Bilder der MapQuest-API)
* Logging (logging.txt mit allen relevanten Logs, vor allem Exception-Logs)
* PDF (Detailed und Summarize Reports)
* Uploads (Unique Feature, wird in Folge besprochen)

Die **Config-File** speichert neben dem Connection String für die Datenbank auch relevante Ordnerpfade, URIs für die MapQuest-API und Einstellungen für Log4Net.

Aufgrund der guten Verfügbarkeit und einfachen Anwendung wurde auf Log4Net zurückgegriffen, um das **Logging** zu implementieren.

Für den **Report** kam iText7 zur Anwendung, da die Dokumentation und Tutorials hiermit sehr detailliert sind und sich für jedes Problem eine Lösung finden ließ.

Ein File-Upload für die einzelnen Touren, der .pdf, .png und .txt erlaubt, ist als **Unique** **Feature** implementiert. Pro Tour lassen sich Dateien der oben genannten Typen hochladen und werden im Ordner Uploads gespeichert. Minimale Modifikation lassen diese Funktion auch für andere Dateitypen ausweiten. Außerdem werden mit Get Files die Dateinamen angezeigt, nach Auswahl des Elements in der Liste und einem Klick auf Open Files, werden PDFs geöffnet, .txt-Dateien lassen ihren Inhalt im selben Fenster anzeigen, .png-Dateien erscheinen ebenfalls im Fenster.

Das **Design Pattern**, das für dieses Projekt verwendet wurde, ist das Strategy Pattern. Dieses Pattern wurde für das Unique Feature herangezogen. Ein gemeinsames Interface, das auf die drei verschiedenen Handler vererbt, lässt je nachdem was für eine Datei ausgewählt wird, mit einem anderen Handler arbeiten, da jede Datei-Art anders abgehandelt wird.

Da nur wenige Methoden einen Rückgabewert haben und die **Unit-Tests** absolut nicht gewillt waren mit einer Config-File zu arbeiten, liegen die meisten Unit-Tests in den Details. Auch da es für die Implementierung der Details wichtig war, dass sie absolut reibungslos laufen, wurden Methoden, die validieren, relevante String zurückgeben (dass zwischen dem Pfad und der Dateiendung das +“\\“ + , wird auf ewig ein genervtes Zucken bei mir auslösen) oder Ähnliches. Um das Config-File-Problem zu umgehen, wurde der String stellenweise für den Test hineinkopiert und anschließend wieder entfernt.

**Time Tracking:**

Initialisierung, erstes Set-Up, Layeraufbau 3h

Datenbank Set-Up 1h

Config-File (+ Research) 2h

MapQuestApi 5h

Log4Net (+ Research) 4h

Report Research 1h

Model: Tour, TourLog 2h

Create, Update, Delete Tour 4h

Create, Update, Delete Logs 4h

Search (Tour/TourLog) 1h

Export Tour 2h

Import Tour (+ Logs) 3h

Validation 1h

Show Tour Details 1h

Show Log Details 1h

Single Report 4h (Formatierung…)

Summarize Report 1h

Unique Feature (+ Design Pattern) 5h

Testing (mehrfach neues Projekt aufgesetzt) 4h

Refactoring 12h

GESAMT 61h