Protokoll – „1 Nicer Tour Planner“

Link zu GitHub Repository

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. MapInfo löst das Problem, das mehrere Werte zurück- und weitergegeben werden müssen, die dadurch gemeinsam in einem Objekt gesammelt werden.

Durch die Klassen im **DataAccessLayer** werden zunächst die Daten aus der PostgreSQL-Datenbank entnommen oder in die Datenbank gespeichert. Darüber hinaus ist das Tour-Data-Object hier zu finden, welches 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 (dieser wird anschließend in der Datenbank gespeichert), das Bild heruntergeladen und somit in den Ordner Images gespeichert wird.

Zuletzt ist die Klasse JSONImport in diesem Layer, die den Import einem 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

* ein neues Fenster öffnet sich

Da manche Funktionen nur mit einer ausgewählten Tour verfügbar sind, werden jene Funktionen ausgeblendet, solange keine Tour angeklickt wurde.

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

Weiter befindet sich folgende Ordner in diesem Layer:

- Exports (Zielordner für exportierte Tours als JSON-Datei)
- Images (Zielordner für Map-Bilder der MapQuest-API)
- Logging (enthält die Datei logging.txt mit allen relevanten Logs, vor allem Exception-Logs)
- PDF (Zielpfad für detailed und summarize Reports)
- Uploads (Zielpfad für das Unique Feature, wird in Folge besprochen)

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

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

Für den **Report** kam iText7 zur Anwendung, da die Dokumentation und Tutorials hiervon 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 dem Button Get Files die Dateinamen in einem neuen Fenster angezeigt. Nach Auswahl des Elements in der Liste und einem Klick auf Open Files, werden PDFs mit dem eingestellten Standard-Reader 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.

Darüber hinaus ist der Zugriff auf die Datenbank als Singleton angelegt. Der Zugriff auf die Klasse erfolgt somit nur über die Methode GetInstance() und stellt sicher, dass nur eine einzige Instanz der Klasse DB gleichzeitig verwendet wird.

Da nur wenige Methoden einen Rückgabewert haben und der Ablauf der Applikation nicht davon abhängen, liegen die meisten **Unit-Tests** in den Details. Auch da es für die Implementierung der Details wichtig war, dass sie absolut reibungslos laufen und keine kleinen Fehler untergehen, wurden Methoden, die validieren, relevante String zurückgeben (dass zwischen dem Pfad und der Dateiendung das +“\\“ + nicht fehlen darf, wird auf ewig ein genervtes Zucken bei mir auslösen) oder Ähnliches. Damit das Config-File keine Probleme macht, ist im Testprojekt das File testhost.dll.config, das eine Kopie des Config-Files ist.

Time Tracking:

Initialisierung, erstes Set-Up, Layeraufbau	3h
Datenbank Set-Up	2h

Config-File (+ Research)	3h
MapQuestApi	7h
Log4Net (+ Research)	4h
Report Research	1h
Model: Tour, TourLog	2h
Create, Update, Delete Tour	5h
Create, Update, Delete Logs	3h
Search (Tour/TourLog)	1h
Export Tour	2h
Import Tour (+ Logs)	6h
Validation	1h
Show Tour Details	2h
Show Log Details	2h
Single Report	4h (Formatierung…)
Summarize Report	1h
Unique Feature (+ Design Pattern)	6h
Testing (mehrfach neues Projekt aufgesetzt)	4h
Refactoring	15h
Protokoll	1h
GESAMT	75h