

Software Engineering 2

Burak Konca

BIF4 4A1 – SWE2

Inhalt

[Technische Schritte 3](#_Toc73467587)

[Design 3](#_Toc73467588)

[Layer 3](#_Toc73467589)

[Layer content / functionality 4](#_Toc73467590)

[Architectural Decisions 4](#_Toc73467591)

[UX Decisions 4](#_Toc73467592)

[Library Decisions 4](#_Toc73467593)

[Fehlschläge 4](#_Toc73467594)

[Ausgewählten Lösungen/ Lösungsansätze 4](#_Toc73467595)

[Unit Tests 4](#_Toc73467596)

[Unique Feature 4](#_Toc73467597)

[Tracked time 4](#_Toc73467598)

[Git history 4](#_Toc73467599)

[Abbildungsverzeichnis 5](#_Toc73467600)

Protokoll

# Technische Schritte

## Design

Wie in der untenstehenden Abbildung zu sehen ist, verfügt die TourPlanner Projektmappe über sieben Projekte.

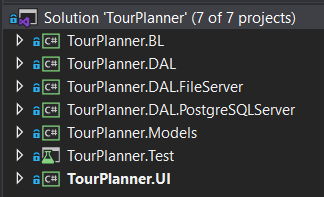


Abbildung : Projektmappe

Unter Anderem ist der TourPlanner gegliedert in einen Business Layer (BL), Data Access Layer (DAL), der Grafischen Benutzeroberfläche (GUI), den einzelnen Modellen und einem Testprojekt zum Schreiben von Unit Tests etc.

### Layer

#### Business Layer

Der Business Layer ist der Teil des Programms, der die realen Geschäftsregeln codiert, die bestimmen, wie Daten erstellt, gespeichert und geändert werden können. Es schreibt vor, wie Geschäftsobjekte miteinander interagieren, und erzwingt die Routen und Methoden, mit denen auf Geschäftsobjekte zugegriffen und diese aktualisiert werden.

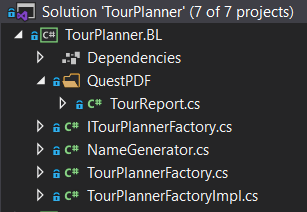


Abbildung 2: BL Projekt

In diesem Projekt habe ich beschlossen das **Factory Pattern** anzuwenden. Die Factory-Pattern-Methode definiert dabei ein Interface für die Erstellung von Objekten und delegiert die Objekterstellung an die Sub-Klassen. Ziel ist, dass der Client nichts von der Objekt-Instanziierung mitbekommt und über eine gemeinsame Schnittstelle auf die Objekte zugreift.

Daher beinhaltet der Business Layer bei mir folgende drei Klassen, die dem Factory Pattern angehören:

* **ITourPlannerFactory**: Schnittstelle zur Definition der Struktur der Funktionalitäten/Methoden in Bezug auf die Touren und Tourlogs. Diese Klasse definiert die Strukturen für folgende
* **TourPlannerFactory**: Stellt durch den Einsatz des **Singleton** Patterns nur eine einzige Instanz der Factory zur Verfügung auf, die dann zugegriffen werden kann.
* **TourPlannerFactoryImpl**: Enthält die Implementierung der im Interface ITourPlannerFactory festgelegten Methodenstrukturen.

Des Weiteren gehören noch zwei weitere Klassen dem Business an und zwar eine Klasse **NameGenerator**, welche für Testzwecke verwendet wurde und um den Aufwand zum Erstellen von Touren und TourLogs zu minimieren. Und einer Klasse **TourReport**, welche für die Erstellung eines PDF Tour Reports dient, die Infos zu einer bestimmten Tour, einem Bild der Route und der dazugehörigen TourLogs beinhaltet.

#### Data Access Layer

Ein Data Access Layer (DAL) ist eine Schicht eines Computerprogramms, die einen vereinfachten Zugriff auf Daten bereitstellt, die in irgendeiner Art von Dauerspeicher gespeichert sind, wie beispielsweise einer Datenbank oder einem lokalen Filesystem.

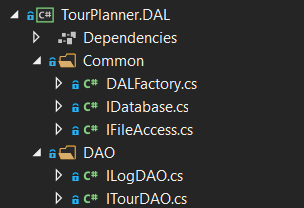


Abbildung : DAL Projekt

In meiner Anwendung beschloss ich zwei Zugriffsmöglichkeiten für Daten bereitzustellen. Und zwar über eine Datenbank und einem lokalen Filesystem.

Das Ganze läuft abstrahiert ab, indem die beiden Zugriffsmöglichkeiten ihre Gemeinsamkeiten von entsprechenden Interfaces **IDatabase** für den Datenbankzugriff und **IFileAccess** für den Filesystemzugriff erben und wie sie zum Einsatz kommen wird in der Klasse **DALFactory** klar festgelegt. Die DALFactory entscheidet anhand einer Konfigurationsdatei, welche Zugriffsart erfolgen soll.

Des Weiteren muss die Struktur der Methoden der Data Access Objects (DAO) festgelegt werden. Dies erfolgt in den entsprechenden Klassen **ILogDAO** und **ITourDAO**.

##### FileServer

Eine der beiden Unterteilungen des Data Access Layers ist der FileServer. Das FileServer Projekt ist für Handlen des Zugriffs auf das lokale FileSystem zuständig.

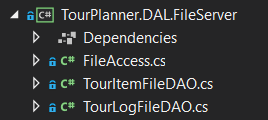


Abbildung : FileServer Projekt

Das Projekt besteht aus drei Klassen. Einer Klasse **FileAccess**, welche für das Erzeugen von Textdateien für Touren und TourLogs zuständig ist, als auch das Erstellen des Routen Bildes durch **integration der MapQuest API.** Des Weiteren gibt es die Klasse **TourItemFileDAO**, welche für das Interagieren von Touren mit dem FileSystem zuständig ist, um zum Beispiel Touren von einer Datei einzulesen, etc.

Natürlich müssen dieselben Funktionalitäten auch für die TourLogs bereitgestellt werden. Dies wird dann in der **TourLogFileDAO** Klasse bereitgestellt.

##### PostgreSQLServer

Die Zweite Unterteilung des Data Access Layers ist der PostgreSQLServer. Das PostgreSQLServer. Das PostgreSQLServer Projekt ist für Handlen des Zugriffs auf die Datenbank zuständig.

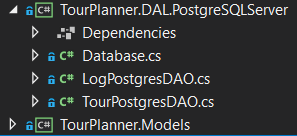
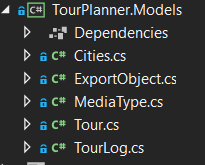


Abbildung : PostgreSQLServer Projekt

Das Projekt besteht aus drei Klassen. Einer Klasse **Database**, welche für das Aufbauen der Datenbankverbindung und erstellen und durchführen von SQL Statements zuständig ist. Die Klassen **LogPostgresDAO** und **TourPostgresDAO** sind für die Datenbankabfragen bezüglich Touren und TourLogs zuständig.

#### Models

In dem Projekt Models werden die einzelnen Modelle der Objekte abgespeichert. Hier habe ich zunächst eine Klasse **Cities** wurde für Komfortzwecke entwickelt. Sie beinhaltet ein Enum von Orten das in der weiteren Abfolge der Anwendung zur schnellen Erstellung von random Touren verwendet wurde.



#### Test

#### GUI

### Layer content / functionality

### Architectural Decisions

### UX Decisions

### Library Decisions

## Fehlschläge

## Ausgewählten Lösungen/ Lösungsansätze

# Unit Tests

# Unique Feature

# Tracked time

# Git history

//no need to copy into documentation

# Abbildungsverzeichnis