

BIF4B2

TourPlaner SWEII

DOKUMENTATION
TAREK ABOUZAID

Inhaltsverzeichnis

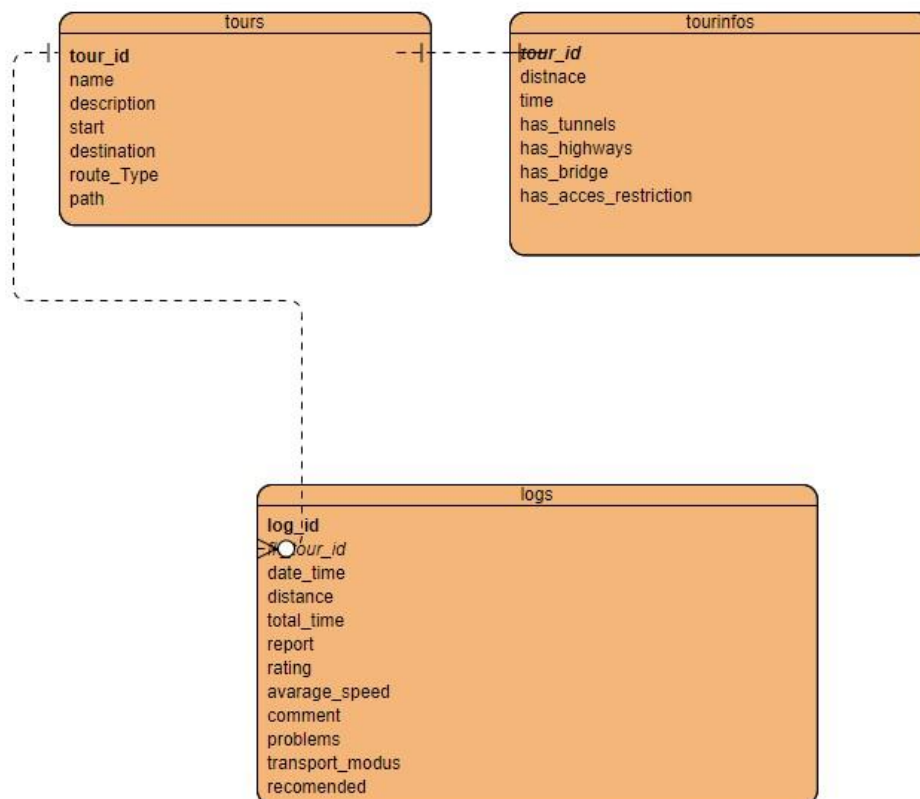
Datenbank	2
Datenbankbeschreibung	2
UML Diagramm.....	2
Beschreibung der Tabellen	2
SQL-Statements	3
Create tours	3
Create tourinfos	3
.....	3
Create logs.....	4
Die App Architecture	5
Models.....	5
View.....	6
ViewModel	7
BusinessLayer	8
DataAccessLayer	9
Kommunikationsweg.....	9
Warum diese Architektur?	9
Verwendete Libraries	10
Warum diese Libraries?	11
Design Pattern	11
Unit Tests.....	12
Unique Features	15
Das Aussuchen eines bestimmten Wertes beim Editieren der Tour	15
Das Aussuchen eines bestimmten Wertes beim Editieren der Logs.....	17
Beim Importieren kann sich der User aussuchen, ob er seine alten Daten behalten möchte oder sie löschen will.	18
Zeitübersicht.....	20

Datenbank

Datenbankbeschreibung

Für die Datenbank habe ich mich für PostgreSQL entschieden. Für die Erstellung und Bearbeitung der Tabellen wurde das Webinterface PgAdmin4 verwendet, um eine Übersicht zu behalten.

UML Diagramm



Beschreibung der Tabellen

Die Tabelle „tours“ speichert die einzelnen Touren, alle Columns bis auf „tour_id“ und „path“ sind Informationen, die der User eingibt. „tour_id“ ist eine UUID, die beim Erstellen einer Tour vom Code erstellt wird, außerdem ist die Spalte der Primary-Key dieser Tabelle.

Die Tabelle „tourinfos“ enthält die Informationen, die aus dem Request an die MapQuestApi, zurückgeschickt wird. Hier hat der User keinen Einfluss auf die gespeicherten Infos, es sind lediglich die Informationen der Tour die zurückgelegt wurde. Der Primary-Key in dieser Tabelle ist die „tour_id“ und dient auch als Fremd-Key, der auf die „tour“ Tabelle verweist. Wird die Tour von der „tour“ Tabelle gelöscht, wird auch der dazugehöriger Datensatz aus der Tabelle „tourinfos“ gelöscht.

In der Tabelle „logs“ sind die logs gespeichert, die der User für eine Tour einträgt. „log_id“ ist der Primary-Key und „fk_tour_id“ ist der Fremd-Key der auf die jeweilige Tour verweist. Hier gilt auch, wird die Tour gelöscht, werden die dazugehörigen logs gelöscht. Dabei ist zu beachten, dass eine Tour mehrere Logs beinhalten kann.

SQL-Statements

Create tours

```
1 CREATE TABLE IF NOT EXISTS public.tours
2 (
3     tour_id character varying COLLATE pg_catalog."default" NOT NULL,
4     name character varying COLLATE pg_catalog."default" NOT NULL,
5     description character varying COLLATE pg_catalog."default",
6     start character varying COLLATE pg_catalog."default" NOT NULL,
7     destination character varying COLLATE pg_catalog."default" NOT NULL,
8     route_type character varying COLLATE pg_catalog."default" NOT NULL,
9     path character varying COLLATE pg_catalog."default" NOT NULL,
10    CONSTRAINT tours_pkey PRIMARY KEY (tour_id)
11 )
12
13 TABLESPACE pg_default;
14
15 ALTER TABLE public.tours
16     OWNER to postgres;
```

Create tourinfos

```
1 CREATE TABLE IF NOT EXISTS public.tourinfos
2 (
3     tour_id character varying COLLATE pg_catalog."default" NOT NULL,
4     distance character varying COLLATE pg_catalog."default",
5     "time" character varying COLLATE pg_catalog."default",
6     has_tunnels boolean,
7     has_highways boolean,
8     has_bridge boolean,
9     has_acces_restriction boolean,
10    CONSTRAINT tourinfos_pkey PRIMARY KEY (tour_id),
11    CONSTRAINT fk_tour_id FOREIGN KEY (tour_id)
12        REFERENCES public.tours (tour_id) MATCH SIMPLE
13        ON UPDATE CASCADE
14        ON DELETE CASCADE
15        NOT VALID
16 )
17
18 TABLESPACE pg_default;
19
20 ALTER TABLE public.tourinfos
21     OWNER to postgres;
```

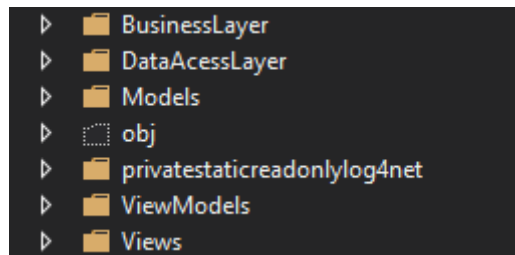
Create logs

```
1 CREATE TABLE IF NOT EXISTS public.logs
2 (
3     log_id character varying COLLATE pg_catalog."default" NOT NULL,
4     fk_tour_id character varying COLLATE pg_catalog."default" NOT NULL,
5     date_time character varying COLLATE pg_catalog."default",
6     distance character varying COLLATE pg_catalog."default",
7     total_time character varying COLLATE pg_catalog."default",
8     report character varying COLLATE pg_catalog."default",
9     rating character varying COLLATE pg_catalog."default",
10    avarage_speed character varying COLLATE pg_catalog."default",
11    comment character varying COLLATE pg_catalog."default",
12    problems character varying COLLATE pg_catalog."default",
13    transport_modus character varying COLLATE pg_catalog."default",
14    recomendend character varying COLLATE pg_catalog."default",
15    CONSTRAINT logs_pkey PRIMARY KEY (log_id),
16    CONSTRAINT fk_tour_id FOREIGN KEY (fk_tour_id)
17        REFERENCES public.tours (tour_id) MATCH SIMPLE
18        ON UPDATE CASCADE
19        ON DELETE CASCADE
20        NOT VALID
21 )
22
23 TABLESPACE pg_default;
24
25 ALTER TABLE public.logs
26     OWNER to postgres;
27 -- Index: fki_fk_tour_id
28
29 -- DROP INDEX public.fki_fk_tour_id;
30
31 CREATE INDEX fki_fk_tour_id
32     ON public.logs USING btree
33     (fk_tour_id COLLATE pg_catalog."default" ASC NULLS LAST)
34     TABLESPACE pg_default;
```

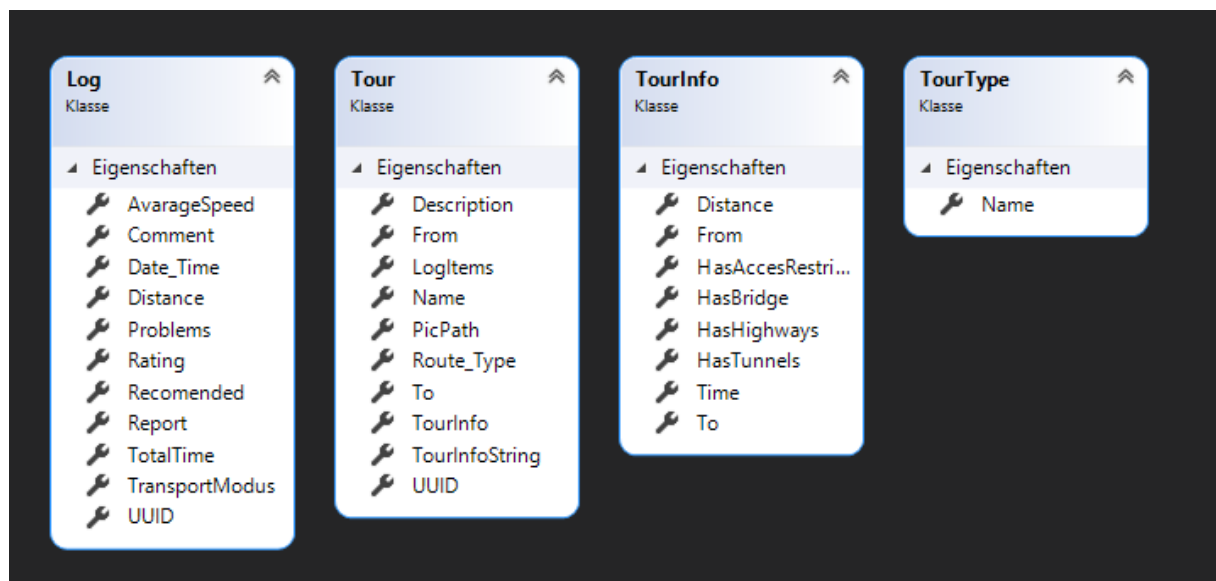
Die App Architecture

Für den Aufbau der App wurde das Mvvm Pattern implementiert.

Mvvm steht für Model, View, ViewModel.

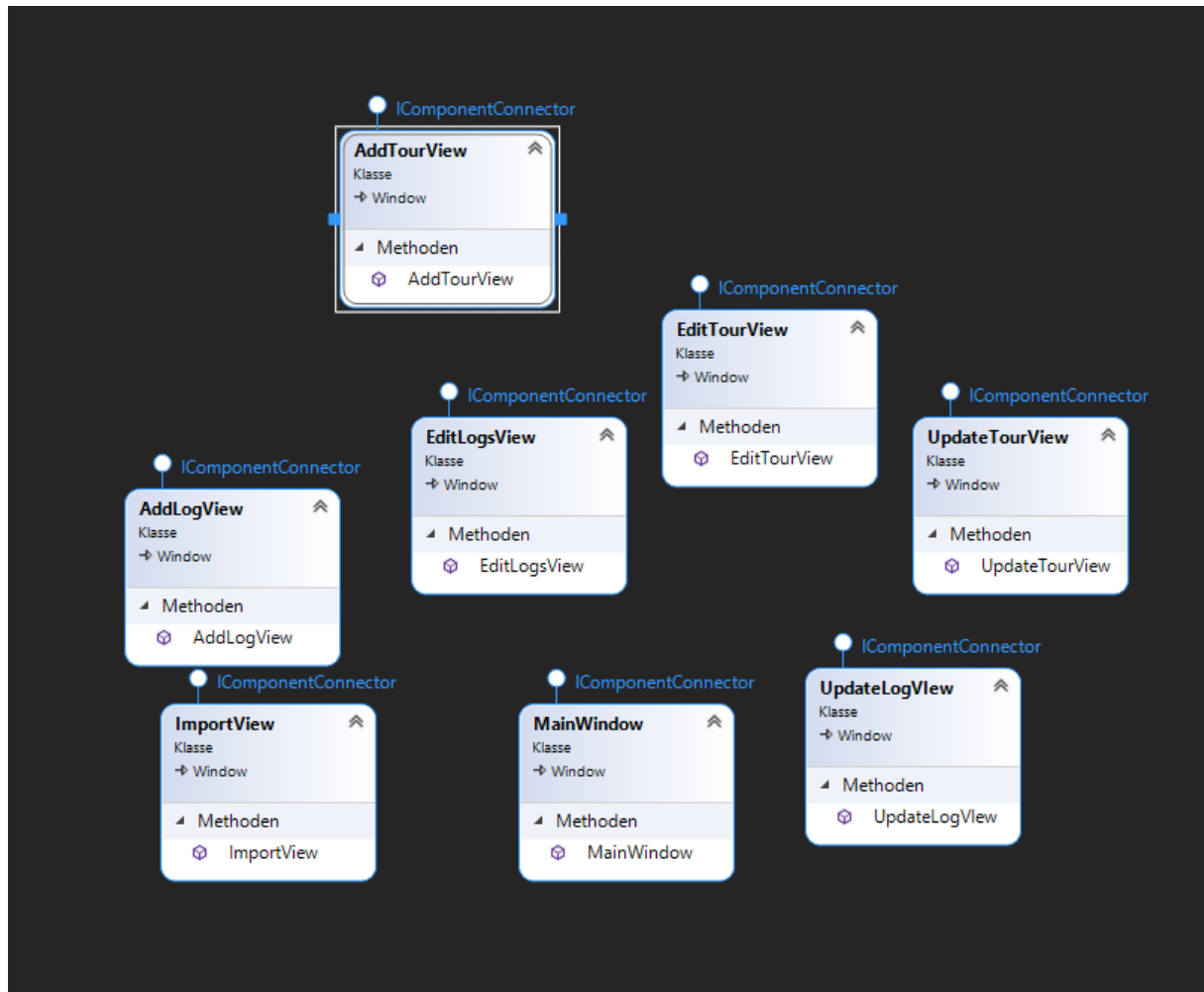


Models



Das Modell repräsentiert die tatsächlichen Daten und/oder Informationen, mit denen wir es zu tun haben. Das Wichtigste beim Modell ist, dass es die Informationen enthält, aber keine Verhaltensweisen oder Dienste, die die Informationen manipulieren. Es ist nicht dafür verantwortlich, Text so zu formatieren, dass er auf dem Bildschirm schön aussieht, oder eine Liste von Elementen von einem entfernten Server zu holen (in dieser Liste wäre jedes Element höchstwahrscheinlich ein eigenes Modell). Die Geschäftslogik wird normalerweise vom Modell getrennt gehalten und in anderen Klassen gekapselt, die auf das Modell wirken.

View



Die View ist das, womit die meisten von uns vertraut sind und das Einzige, mit dem der Endbenutzer wirklich interagiert. Sie ist die Präsentation der Daten. Die Ansicht nimmt sich gewisse Freiheiten, um diese Daten besser darstellbar zu machen. Die View ist zusammengefasst das UI.

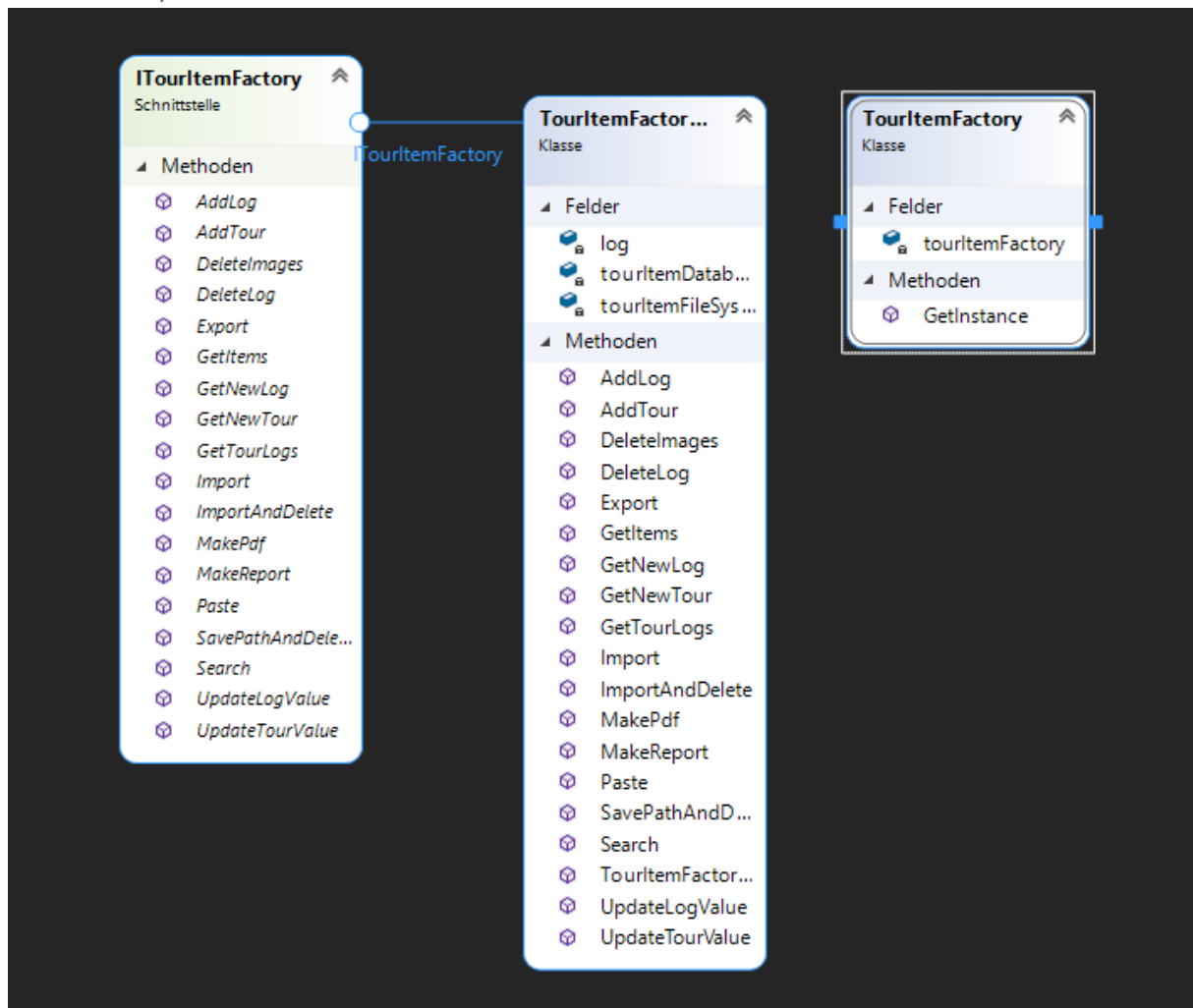
ViewModel



Das Viewmodel ist ein Schlüsselement des Dreiklangs, weil es die Präsentationstrennung einführt, oder das Konzept vom Modell getrennt zu halten. Anstatt dem Modell die Sicht des Benutzers auf ein Datum mitzuteilen, so dass es das Datum in das Anzeigeformat konvertiert, hält das Modell einfach die Daten, die Ansicht hält einfach das formatierte Datum und der Controller fungiert als Verbindungsglied zwischen den beiden. Der Controller kann Eingaben von der View entgegennehmen und sie auf dem Modell platzieren, oder er kann mit einem Dienst interagieren, um das Modell abzurufen, dann Eigenschaften übersetzen und es auf der View platzieren.

Das Viewmodel stellt auch Methoden, Befehle und andere Punkte zur Verfügung, die helfen, den Zustand der Ansicht zu erhalten, das Modell als Ergebnis von Aktionen in der Ansicht zu manipulieren und Ereignisse in der Ansicht selbst auszulösen.

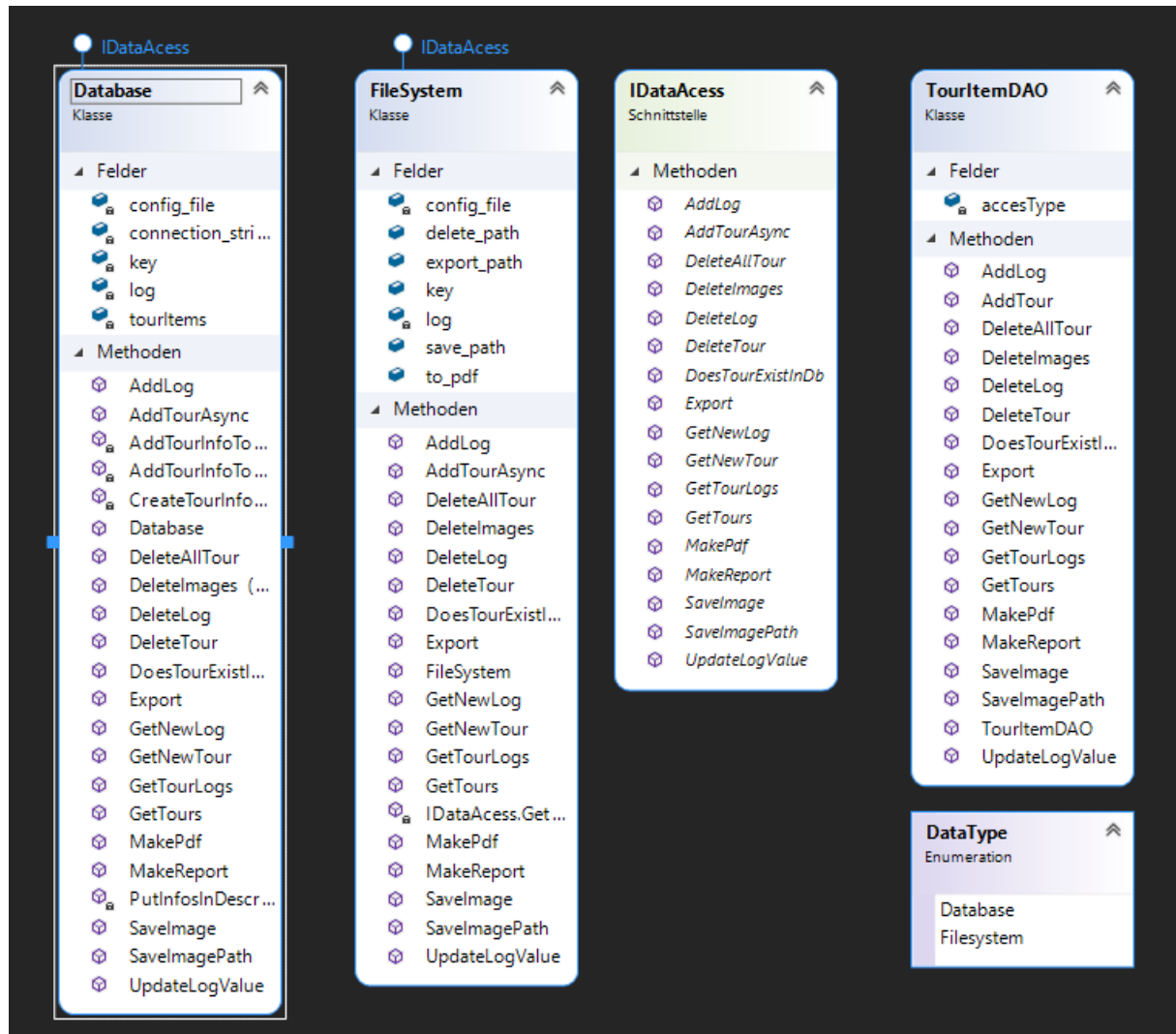
BusinessLayer



Im BusinessLayer findet die Logik statt. Da werden die Funktionalitäten beschrieben, die für die App notwendig sind. Die business logic enthält Objekte, die die business functions ausführen. Für die Implementierung dieser Objekte wurde das Command pattern in Betracht gezogen. Mit dem Command pattern wird jeder Anwendungsfall im Anforderungsdokument als separater Befehl oder Satz von Befehlen implementiert, die in der Businesslogikschicht ausgeführt werden. Jedes Befehlsobjekt implementiert eine Befehlsschnittstelle. Die Befehlsschnittstelle verfügt über eine einfache Methode `execute` (ValueObject)

Jedes Befehlsobjekt hat eine Geschäftslogik in seiner Ausführungsmethode. Das Value-Objekt-Argument für die `Execute`-Methode enthält die Anfragedaten, die für die Ausführung des Anwendungsfalles erforderlich sind. Die typische Ausführungsmethode führt Funktionen wie den Zugriff auf die DataAccessLayer, die Ausführung der Geschäftslogik und die Rückgabe eines Wertobjekts, das die Ergebnisse des Anwendungsfalles enthält aus.

DataAccessLayer



Der Data AccessLayer ist die Schicht wo die Daten für die weitere bearbeitung bereitgestellt werden, indem diese Schicht auf die DB oder das FileSystem zugreift. Außerdem werden hier neue Daten, die gespeichert werden sollen, an die DB oder das FileSystem geleitet.










Kommunikationsweg

View -> ViewModel -> BusinessLayer -> DataAccessLayer -> DB + FileSystem

Warum diese Architektur?

Mit dieser Architektur hat jeder Layer, seine eigene Aufgabe/Funktion. Der Code ist somit sauber getrennt und klar strukturiert und aufgebaut. Sollte man sich in der Zukunft Updates oder Änderung vornehmen, weiß man leichter, auf welchem Teil des Codes/auf welchem Layer man die Änderung vornimmt. Durch die Gliederung und Trennung, kann man den Code außerdem leichter testen und man könnte die einzelnen Layers für andere Projekte wieder verwenden (Reusability).

Verwendete Libraries

	NUnit durch Charlie Poole, Rob Prouse	v3.12.0
	NUnit is a unit-testing framework for all .NET languages with a strong TDD focus.	v3.13.2
	NUnit3TestAdapter durch Charlie Poole, Terje Sandstrom	v3.16.1
	NUnit 3 adapter for running tests in Visual Studio. Works with NUnit 3.x, use the NUnit 2 adapter for 2.x tests.	v3.17.0
	Newtonsoft.Json durch James Newton-King	v13.0.1
	Json.NET is a popular high-performance JSON framework for .NET	
	Npgsql durch Shay Rojansky, Yoh Deadfall, Brar Piening, Nikita Kazmin, Austin	v5.0.4
	Npgsql is the open source .NET data provider for PostgreSQL.	v5.0.5
	Aspose.PDF durch Aspose	v21.5.0
	Aspose.PDF for .NET is an amazing component which provides the feature to create and manipulate PDF documents.	
	ceTe.DynamicPDF.CoreSuite.NET durch ceTe Software	v11.11.0
	.NET API for PDF development. PDF creation, merging, form filling, stamping, securing/encryption and much more. This SDK is easy to use a...	v11.12.0
	log4net durch The Apache Software Foundation	v2.0.12
	log4net is a tool to help the programmer output log statements to a variety of output targets. In case of problems with an application, it is helpful to e...	
	MaterialDesignColors durch James Willock	v2.0.0
	ResourceDictionary instances containing standard Google Material Design swatches, for inclusion in a XAML application.	v2.0.1
	MaterialDesignThemes durch James Willock	v4.0.0
	ResourceDictionary instances containing Material Design templates and styles for WPF controls in .NET.	v4.1.0

Warum diese Libraries?

NUnit -> Letztes Semester kennengelernt, schon Erfahrungen damit gesammelt und kenne keine andere Library.

Newtonsoft.Json -> gut erklärte Dokumentation im Internet zu finden und marktführend.

Npgsql -> Da meine Datenbank eine PostgreSQL Datenbank ist.

ceTe -> Die erste Library die funktioniert hat, bei der Erstellung der PDF dateien. Hatte davor andere probiert, doch leider gab es keine oder veraltete Dokumentationen im Internet und haben nicht funktioniert.

Aspose -> Nachdem ich für die Erstellung des Summarys eine Tabelle gebraucht habe, und dies mit ceTe nicht ging, habe ich auf Aspose zurückgreifen müssen. Dennoch wollte ich die Funktionen, die ich bereit mit ceTe geschrieben habe, nicht löschen, deshalb habe ich beide verwendet.

Log4net -> gut beschriebene Dokumentation im Internet zu finden und wurde uns im Unterricht empfohlen.

MaterialDesign -> für das UI habe ich die zwei Libraries verwendet, um ein einheitliches Designmuster zu haben. Meine UI wurde von mir selbst erstellt (keine Layouts oder dergleichen verwendet). Diese Library diente hauptsächlich für die Farben und die Effekte der Buttons.

Design Pattern

Das von mir verwendete Design Pattern ist Factory.

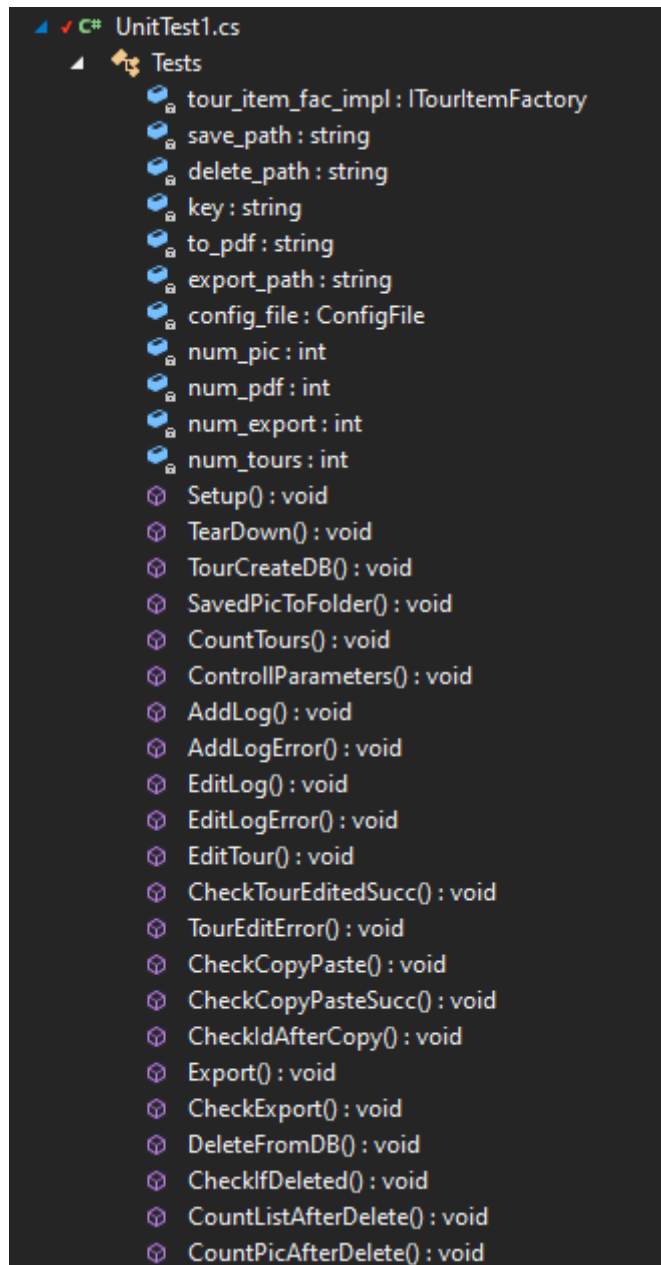
```
namespace TourPlaner.BusinessLayer
{
    9 Verweise
    public class TourItemFactory
    {
        private static ITourItemFactory tourItemFactory;

        9 Verweise
        public static ITourItemFactory GetInstance()
        {
            if(tourItemFactory == null)
            {
                tourItemFactory = new TourItemFactoryImpl();
            }
            return tourItemFactory;
        }
    }
}
```

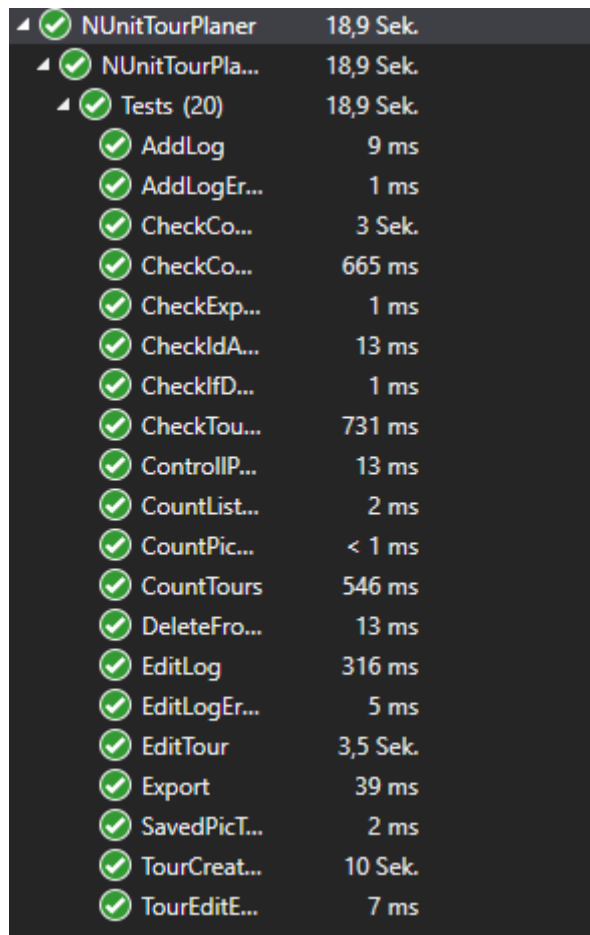
Die Klasse „TourItemFactory“ besitzt eine statische Methode und ein Objekt „TourItemFactoryImpl“.

In der „TourItemFactoryImpl“ befindet sich die BusinessLogik und der Zugriff auf den DataAccessLayer. Die statische Methode der Klasse erstellt dieses Objekt, falls nicht vorhanden, und return ihn. Bei der Erstellung dieses Objekt, von anderen Klassen oder Layer wird nicht der Konstruktor der Klasse aufgerufen, sondern die statische Methode der „TourItemFactory“ Klasse. Somit verwenden aller Layer und Klassen, dasselbe Objekt.

Unit Tests



```
UnitTest1.cs
└─ Tests
    ├── tour_item_fac_impl : ITourItemFactory
    ├── save_path : string
    ├── delete_path : string
    ├── key : string
    ├── to_pdf : string
    ├── export_path : string
    ├── config_file : ConfigFile
    ├── num_pic : int
    ├── num_pdf : int
    ├── num_export : int
    ├── num_tours : int
    ├── Setup() : void
    ├── TearDown() : void
    ├── TourCreateDB() : void
    ├── SavedPicToFolder() : void
    ├── CountTours() : void
    ├── ControllParameters() : void
    ├── AddLog() : void
    ├── AddLogError() : void
    ├── EditLog() : void
    ├── EditLogError() : void
    ├── EditTour() : void
    ├── CheckTourEditedSucc() : void
    ├── TourEditError() : void
    ├── CheckCopyPaste() : void
    ├── CheckCopyPasteSucc() : void
    ├── CheckIdAfterCopy() : void
    ├── Export() : void
    ├── CheckExport() : void
    ├── DeleteFromDB() : void
    ├── CheckIfDeleted() : void
    ├── CountListAfterDelete() : void
    ├── CountPicAfterDelete() : void
```

A screenshot of the NUnit test runner interface showing a list of tests and their execution times. The tests are organized into a tree structure. The root is 'NUnitTourPlaner' (18,9 Sek.), which contains 'NUnitTourPla...' (18,9 Sek.), which in turn contains 'Tests (20)' (18,9 Sek.). Under 'Tests (20)', there are 20 individual test methods, each with a green checkmark icon and its execution time.

✓ NUnitTourPlaner	18,9 Sek.
✓ NUnitTourPla...	18,9 Sek.
✓ Tests (20)	18,9 Sek.
✓ AddLog	9 ms
✓ AddLogEr...	1 ms
✓ CheckCo...	3 Sek.
✓ CheckCo...	665 ms
✓ CheckExp...	1 ms
✓ CheckIdA...	13 ms
✓ CheckIfD...	1 ms
✓ CheckTou...	731 ms
✓ ControllP...	13 ms
✓ CountList...	2 ms
✓ CountPic...	< 1 ms
✓ CountTours	546 ms
✓ DeleteFro...	13 ms
✓ EditLog	316 ms
✓ EditLogEr...	5 ms
✓ EditTour	3,5 Sek.
✓ Export	39 ms
✓ SavedPicT...	2 ms
✓ TourCreat...	10 Sek.
✓ TourEditE...	7 ms

Für das Testen der App, wird im SetUp eine configfile eingelesen und die Pfade der Folder für das Filesystem und der Key für die DB gespeichert. Außerdem wird am Anfang die Anzahl der Tours in der Db und die Files (zB Fotos oder Export) in den Folder abgezählt und in Variablen gespeichert, um diese Variablen dann für das Testen verwenden zu können.

Ich habe mich dazu entschieden, die Funktionalitäten der App zu testen. Alles (oder das meiste) was die App können soll, wird in den Unittests getestet. Die erstellten Fotos, Logs oder Touren werden im Nachhinein gelöscht. Leider ging es zeitlich nicht aus, eine eigene Datenbank für das Testen zu erstellen, weil ich dafür mehrere Klassen verändern müsste. Es werden nicht nur Touren oder Logs erstellt, sondern es wird auch getestet, ob sie im Fehlerfall nicht erstellt werden.

Die getesteten Funktionen:

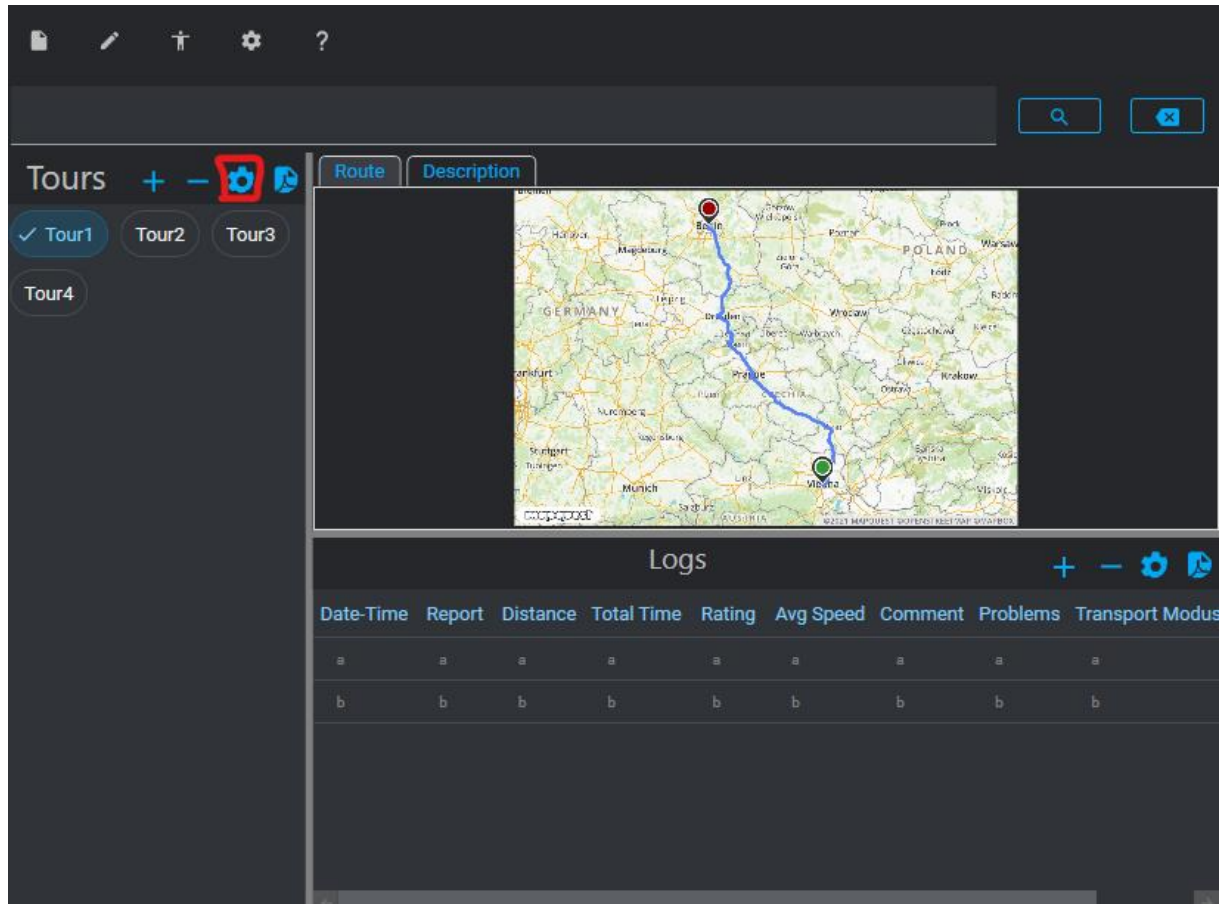
- Tour Adden
- Existiert das Foto im File system?
- Wurde die Tour in der Db angelegt?
- Wurden die Parameter der Tour richtig gespeichert?
- Log Adden
- Log nicht adden (Fehlerfall)
- Log editieren
- Log nicht editieren (Fehlerfall)
- Tour editieren
- Checken ob die Tour richtig editiert wurde in der DB
- Checken ob die Tour nicht editiert wurde (Fehlerfall)
- Kopieren und Einfügen

- Checken ob die Tour in der DB kopiert wurde und ob das Foto für die Kopierte Tour im Filesystem existiert
- Checken ob die ID in der DB für die Kopierte Tour die richtige ist
- Export, ob die json datei im Filesystem angelegt wurde
- Tour, Fotos und Logs löschen
- Checken ob sie tatsächlich nichtmehr existieren
- Checken ob sie aus der Liste im Programm gelöscht wurde
- Schauen ob die Fotos im Filesystem gelöscht wurden






Unique Features



Ich habe 3 Unique Features implementiert, da sie für mich, für die Verwendung der App, sehr hilfreich erschienen.

Das Aussuchen eines bestimmten Wertes beim Editieren der Tour



Nach dem Auswählen einer Tour und das Klicken auf das Editierbutton, erscheint dem User ein Fenster, wo er sich aussuchen kann welchen Wert er ändern möchte.



Tour Name:	Tour1	
Starting Place:	Vienna	
Destination:	Berlin	
Route Type:	Fastest	
Description:	asd	



Der User kann sich nun Aussuchen welchen Wert er ändern möchte. Nach dem Klicken auf dem Editbutton des jeweiligen Feldes, erscheint dem User ein anderes Fenster wo er den neuen Wert eingeben kann.






New Value



Tour99999



Sollte der User, seine neue Eingabe bestätigen (mit dem hackerl unten links). Dann wird diese Änderung übernommen.




Der User wird dann auf das vorige Fenster weitergeleitet und kann weitere Änderungen vornehmen.

Tour Name:	Tour99999	
Starting Place:	Vienna	
Destination:	Berlin	
Route Type:	Fastest	
Description:	asd	



Das Aussuchen eines bestimmten Wertes beim Editieren der Logs



Wie bei dem Editieren der Touren, gilt auch der selbe Vorgang beim Editieren der Logs.

Date - Time:	c	
Distance:	c	
Total Time:	c	
Report:	c	
Rating:	c	
Avarage Speed:	c	
Comment:	c	
Problems:	c	
Transport:	c	
Recomended:	c	



New Value

aaa

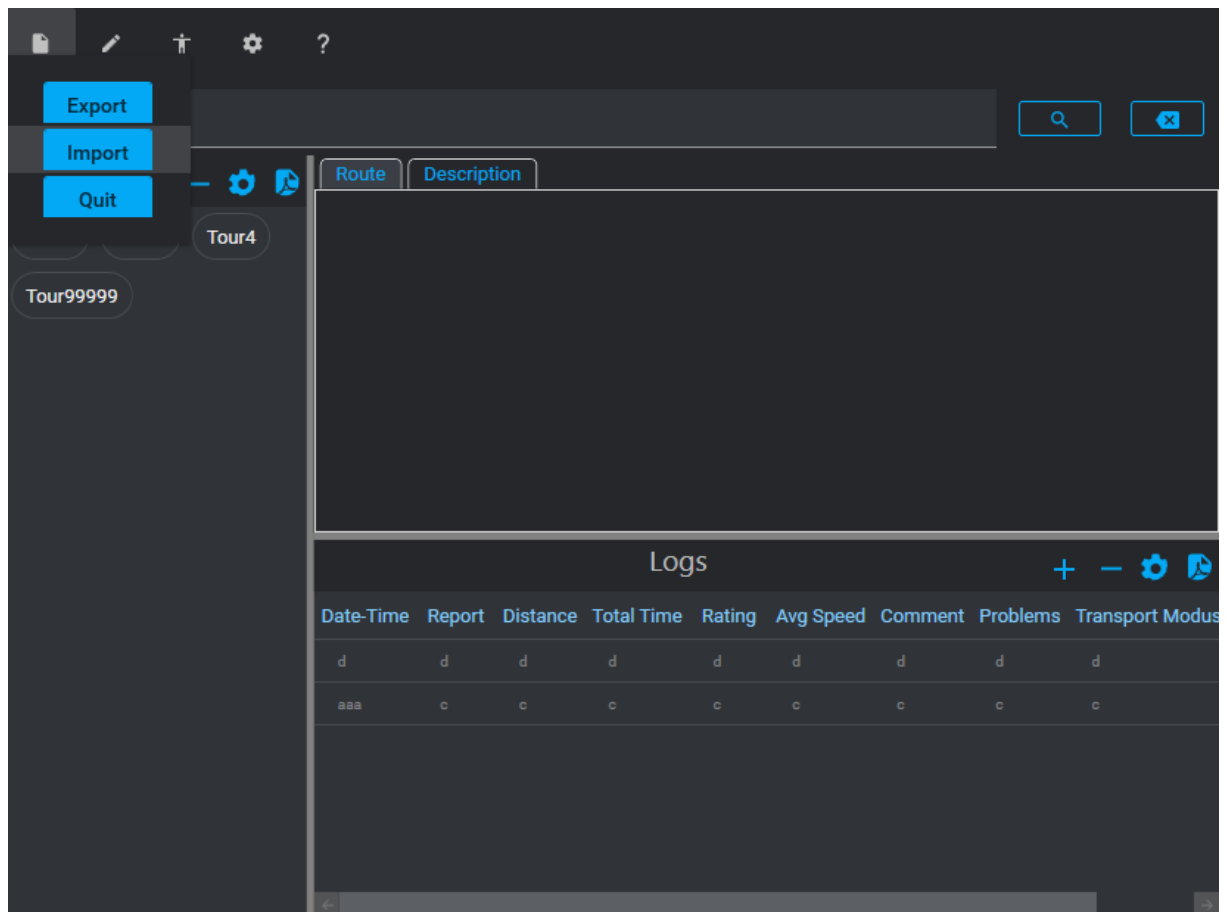


Date - Time:	aaa	
Distance:	c	
Total Time:	c	
Report:	c	
Rating:	c	
Avarage Speed:	c	
Comment:	c	
Problems:	c	
Transport:	c	
Recomended:	c	



Beim Importieren kann sich der User aussuchen, ob er seine alten Daten behalten möchte oder sie löschen will.

Beim Importieren eines bereits existierenden Tourliste, kann der User sich aussuchen, ob er diese Liste 1:1 übernehmen möchte oder seine selbst erstellten Touren neben den importierten behalten will.



Do you want to keep your old Data?



Zeitübersicht

Aufgabe	Stunden
Einlesen und lernen des MvVM Patern	30
Tutorials anschauen	10
Implementierung der MvVM Grundstruktur	10
Design	30
Testen und Debuggen/Beheben von Fehlern	30
Umsetzung der Logik und den Aufbau der App	70
Komplikationen, Denkfehler, Alles andere rundherum	10
Insgesamt	+190