# 1.Ausbaustufe

# Verwendete Framework-Tools:

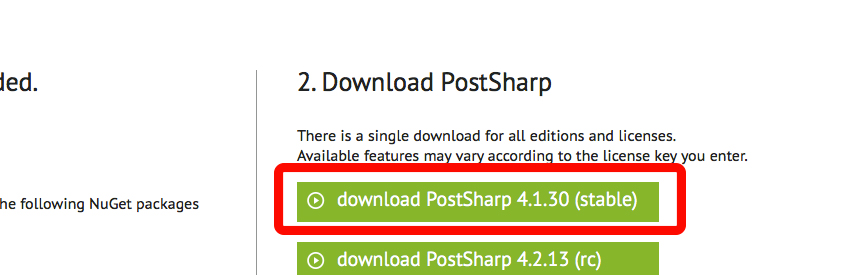
PostSharp (Aspekt orientierte Programmierung):

Dieses Tool wird verwendet, um eigene Aspekte bzw. Attribute schreiben zu können. Konkret wurde es für das Exception Handling in der DAL Schnittstelle verwendet. Das Attribut „DaoExceptionhandlerAttribute“ fängt eine auftretende Exception (z.B. bei einer Datenbank Abfrage) ab und wrappt diese in das generische DAOResponse Objekt. In weiterer Folge wird dieses Konzept noch näher erläutert. Nachfolgend wird beschrieben wie PostSharp in Visual Studio eingebunden werden kann. Für die Ausführung wird eine Free-Lizenz verwendet.

**Installation:**

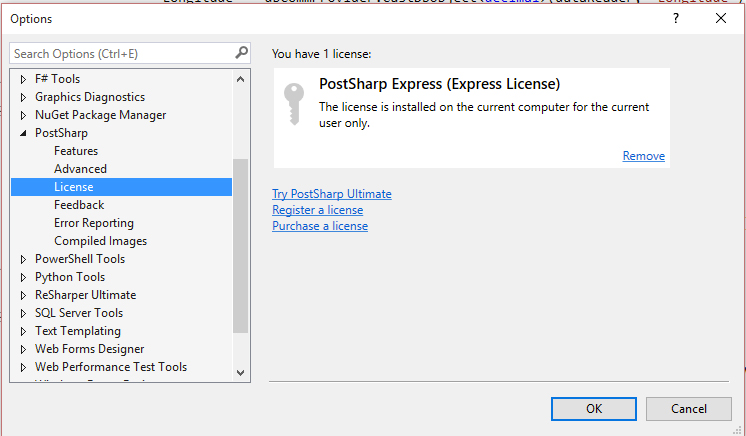
Unter folgendem Link kann PostSharp runtergeladen werden.

https://www.postsharp.net/download



**Registrierung / Account anlegen:**

Um PostSharp im PostCompile Ablauf verwenden zu können, wird wie unten dargestellt, eine Freie Lizenzregistrierung benötigt.



# Datenbankmodell

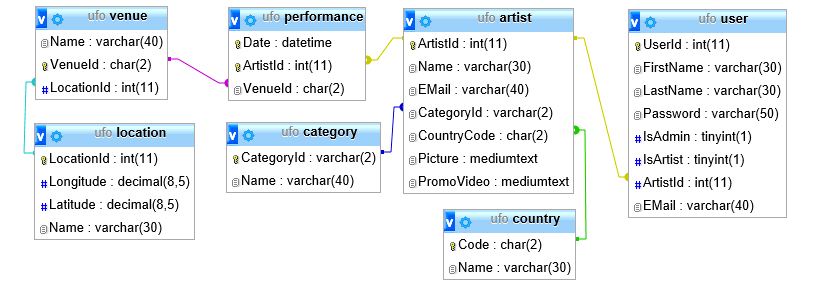
Nach Analyse der Anforderungen, wurde das nachfolgende Domainmodell entworfen.

Zusätzlich zu den vorgegebenen Entitäten (user, artist, ...), werden noch zwei weitere benötigt: Location und Country.

Da es möglich ist, dass an einer Location (z.B. Landhaus, Hauptplatz...) mehrere Spielstätten (Venues) existieren, wird durch auslagern der Information in eine eigene Entität Datenredundanz vermieden.

Für die konkrete Implementierung, wurde in Erwin das Schema entworfen per ForwardEngineering an eine MySQL Datenbank exportiert. Diese wird über phpMyAdmin verwaltet und die SQL-Scripte wurden nachträglich via Exportfunktionalität von MySQL generiert.

Die Primärschlüssel (IDs) der Entitäten User, Location und Artist werden in der Datenbank per Auto-Incrementation automatisch nummeriert.



**Performance:**

Um Aufführungen eindeutig abbilden zu können, wurde eine Primärschlüssel-Beziehung zwischen Artist und Venue in der Performance Entität erstellt. Diese bilden eine eindeutige Abbildung zu welcher Zeit und an welchem Ort ein Artist aufführt. Durch die Überprüfung der Zeit vor einfügen eines Datensatzes wird sichergestellt, dass ein Artist nicht eine Stunde vor bzw. nach seinem Auftritt wieder eingetragen werden kann.

**User:**

Die Stammdatenverwaltung der User beinhaltet eine Referenz auf ein Artist Objekt, welches impliziert, dass der User nicht nur administrative Funktionalitäten beinhalten kann, sondern auch selbst als Artist funktionieren kann.

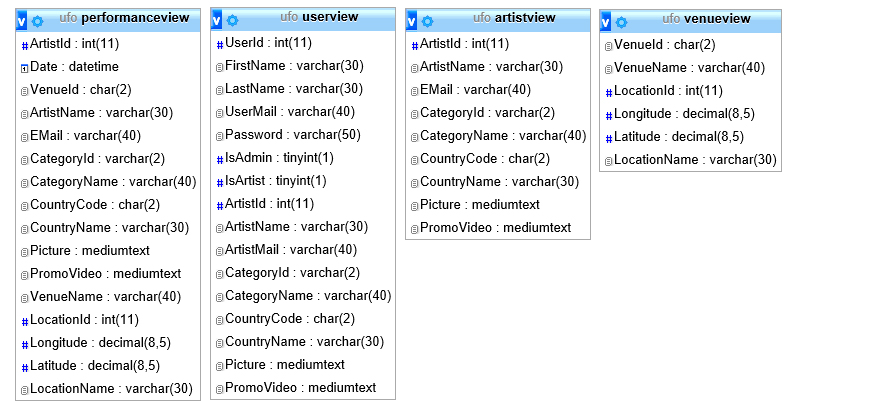
Wenn ein User auch ein Artist ist, wird das Flag IsArtist = 1 gesetzt.

Ein User kann auch als Admin deklariert werden mit IsAdmin = 1. Das heißt, er kann beide Eigenschaften beinhalten.

Diese Abbildung bildet die Basis für eine mögliche Erweiterung der Anwendung, wo sich User, die einem Artisten zugeordnet sind, einloggen können und Stammdaten wie Foto oder Promovideo selbst künftig ändern könnten.

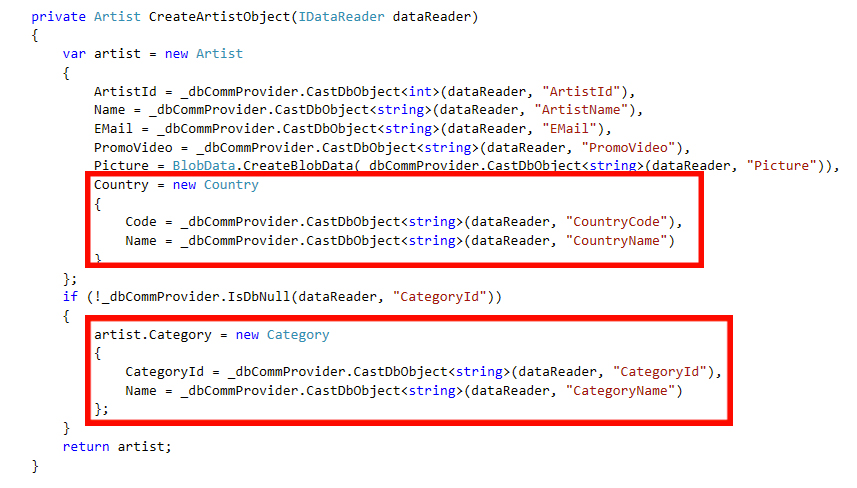
**Artist:**

Für Picture und PromoVideo werden Links in Form eines Strings in der Entität abgebildet. Die Pictures werden in einem BlobData Objekt abgeblidet, welches Meta-Informationen wie Name oder Pfad beinhaltet. Dieses Objekt ist serialisierbar und beinhaltet den Binärdatenstrom, welcher bis zum Client ins Frontend durchgetragen werden kann.



Des Weiteren werden vier Sichten erstellt: Performance-View, User-View, Artist-View und Venue-View. Dadurch können alle zusammenhängenden Daten mit einer Abfrage geladen werden und das verhindert Join-Statements im C# Code.

Der wesentliche Vorteil besteht darin, dass gegebenenfalls alle benötigten Objekte mit einer Abfrage instanziiert werden können, wo ansonsten (wie im folgenden Beispiel) drei Anfragen an die Datenbank benötigt würden.



Zusätzlich um Nullwerte aus der Datenbank unterscheiden zu können und zum richtigen Datentyp mappen zu können, werden alle DataReader Zugriffe über die CastDbObject<T> Methode delegiert. Diese sorgt bei Nullwerten für den richtigen Standardwert.

**Die Daten in der Datenbank stammen von:**

User: <http://convertcsv.com/generate-test-data.htm>

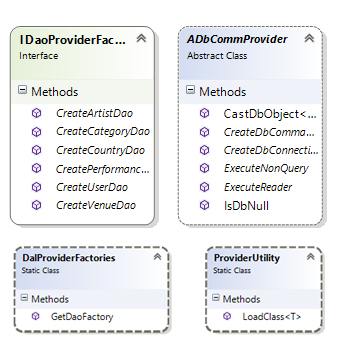
Country: <http://blog.plsoucy.com/wp-content/uploads/2012/04/countries-20140629.csv>

Artist, Venue, Category, Location: <http://www.pflasterspektakel.at/2015/de/1443.asp> (wurden manuell mit Excel angepasst)

Die jeweiligen SQL Scripten, wurden im Verzeichnis UFO.Database/sql bereitgestellt.

# UFO.Server

**DalProviderFactories**: Erstellt zur Laufzeit eine DAO Factory, welche Methoden zur Erstellung von DAO-Instanzen beinhaltet. Die Klasse besteht aus einer statischen Methode, welche eine lose Koppelung zwischen Assemblies erstellt. Hierfür werden anstelle von internen Implementierungen die benötigten Daten aus der app.config Datei (XML) referenziert. Alle konkreten DAO Implementierungen beinhalten, welche von IDaoProviderFactory signiert werden. Wie intern die Instanziierung der DAOs gehandhabt wird obliegt der verwendeten Implementierung.



# UFO.Server.Dal.Dummy

Für die erste Implementierung wurde ein Dummy Assembly erstellt, welches von der IDaoProviderFactory ableitet und demonstrativ den Wechsel der Assemblies veranschaulichen soll. Es wurde jedoch nur ein Bruchteil der DAO Funktionalität implementiert und wird nur noch für Testzwecke verwendet.

# UFO.Server.Dal.MySQL

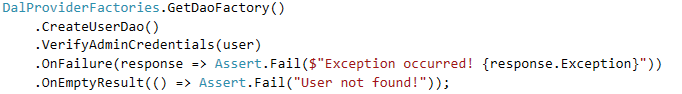
Es werden die einzelnen DAOs instanziiert, diese stellen die benötigten Methoden zur Kommunikation (Connection, SELECT, INSERT, UPDATE, DELETE...) mit der Datenbank zu Verfügung.

Im UFO.Server.Dal.Common befindet sich eine abstrakte Klasse ADbCommProvider, welche die Basisklasse für eine gemeinsame Datenbankkommunikation darstellt. In diesem werden nur abstrakte Klassen wie DbConnection, DbCommand usw. (Klassen des .NET Frameworks) verwendet und des weiteren bietet diese abstrakte Methode, welche von den konkreten Technologien wie Beispielsweise die MySQL-Adaptoren implementiert werden können.

Das Basiskonzept der DAOs beruht auf ein gemeinsamen Responseobjekt, welches als Wrapper für das eigentliche Rückgabeobjekt dient. Dieses bietet zusätzliche Metainformationen und Funktionalitäten.

Es wurde nach dem Fluetinterface Modell nachempfunden. Das heißt, das Methoden wie OnSuccessful und OnFailure auch wiederum DAOResponse Objekte (also sich selbst) retournieren und diese per Chaining aufgerufen werden können.

Bsp.:

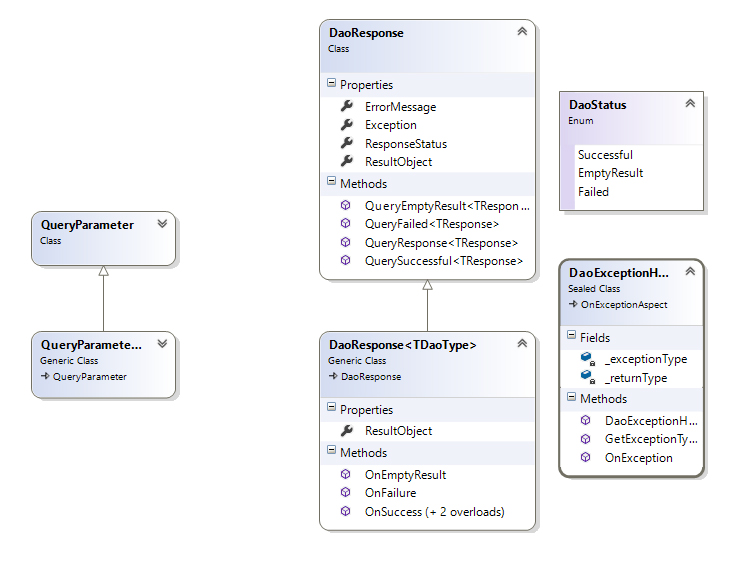


Möglich ist diese Aufruffolge nur, weil keine der DAO Methoden eine Exception im Fehlerfall wirft.

Die Fehler werden durch das oben erwähnte Attribut (DaoExceptionHandlerAttribute) abgefangen und in ein DAOResponse Objekt gepackt und es wird sichergestellt, dass der Methoden-Kontrollfluss normal retourniert.

Die DAOResponses bestehen neben dem jeweiligen Datenobjekt, aus einen StatusCode (Successful, Failed, EmptyResult) gegebenenfalls auch eine Exception und einer Exception Message.

Der Vorteil dabei ist, dass alle Informationen (inkl. mögliche Fehler) in einem Objekt zusammengepackt werden und einfacher darauf reagiert werden kann. Die Exception werden an einer zentralen Stelle behandelt Anstelle von laufenden try-catch Blöcken.



# UFO.Server.Dal.Common

Beinhaltet die **Interfaces der einzelnen DAOs**.

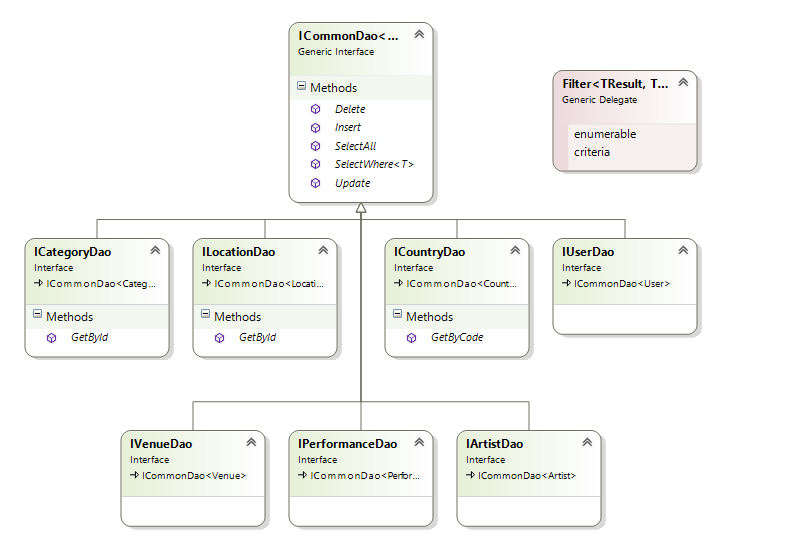
Die Implementierung der DAOs wird nicht direkt vom ICommonDao Interface instanziiert, sondern es gibt für jedes DAO noch ein eigenes Interface, welches von ICommonDao erbt.

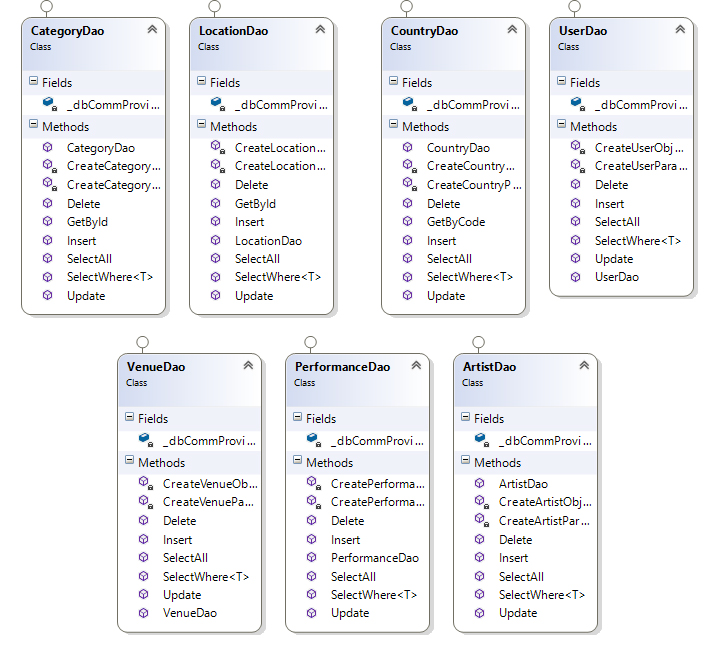
Es kann vorkomme, dass gewisse DAOs unterschiedliche Funktionalitäten benötigen können, welche nicht in einem gemeinsamen Basisinterface aggregiert werden können.

Ein konkretes Beispiel hierfür wäre die GetById Methoden, welche unterschiedliche Parameterdatentypen entgegennehmen.

Z.B.: Ist der Parameter für die CategoryId vom Typ „string“ wohingegen die LocationId vom Typ „Integer“ ist. Desweitern stehen dem Benutzer des Frameworks nur die Interfaces zur Verfügung und dieser kann nicht auf die konkreten Klassen zugreifen.

Die Methode SelectWhere<T> wird derzeit intern in der MySql Implementierung über die SelectAll Methode delegiert und verwendet den Lambda Ausdruck „Nur für in Memory Filterung“. Da jedoch eine Expression als Schnittstellen Definition zur Verfügung steht, kann diese in Zukunft zu einer nativen SQL Abfrage abgeändert werden, um somit eine Performance Steigerung zu erzielen. Für Testzwecke wurde auf Basis von SelectWhere auch die Extensionmethods erstellt.



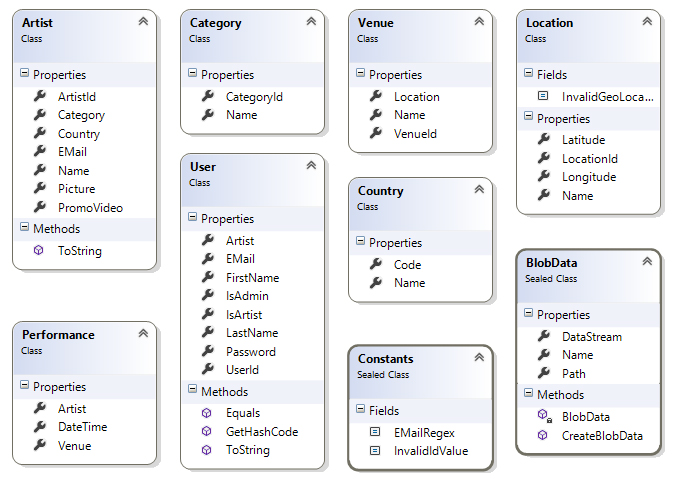


In manchen Fällen ist es Vorteilshaft, wenn zu den DAOs Erweiterungen (Extensions) implementiert werden.

So können Interfaces erweitert werden, ohne diese selbst zu verändern und Extensions sind einfach austauschbar bzw. lassen sich einfach wieder entfernen. Hier werden sie für Methoden verwendet, welche für die Testfälle nützlich sind, aber eventuell bei der nächsten Ausbaustufe des Projekts obsolet werden.

# Domain

Beinhaltet die Objekte zur Abbildung der Entitäten aus der Datenbank sowie die Klasse BlobData, welche in späterer Folge für die Abbildung von Mediendateien benötigt wird.



Außerdem wird die Klasse **Crypto** zur Verfügung gestellt. Welche für die Verschlüsselung von Passwörtern zuständig ist.

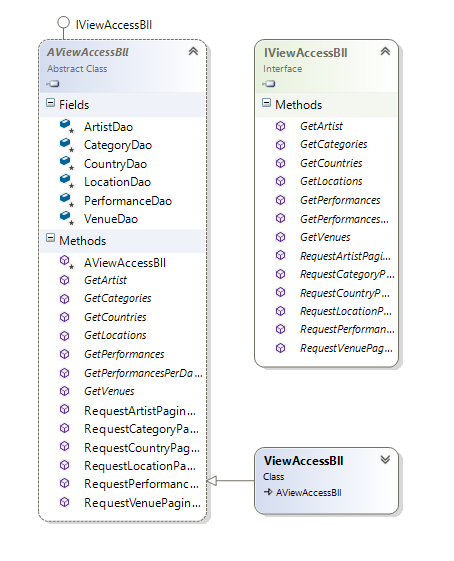
Diese Klasse beinhaltet zwei Methoden. Die eine Methode transformiert einen Klartext Stringwert in einen mit MD5 verschlüsselten Hashwert, welcher auch in die Datenbank gespeichert wird.

Die zweite Methode vergleicht den neu berechneten Hashcode mit dem aus der Datenbank.

Für diese Funktionalität werden Methoden aus dem .NET-Framework (System.Security.Cryptography) verwendet.

# 2.Ausbaustufe

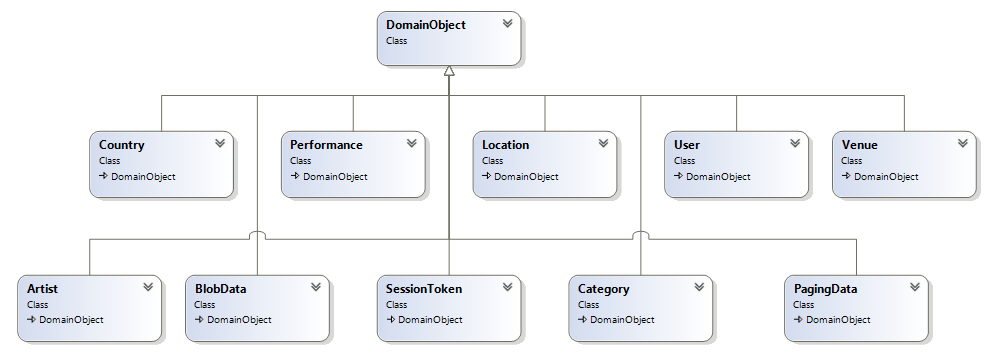
# Business Logic Layer



Die Domänenklassen wurden dahingehend erweitert, ein Sessionhandling zu unterstützen. Hierfür wird die Klasse SessionToken als Transportobjekt verwendet. Hierfür muss der Client an den Server einen Request mit den Authentifizierungsdaten senden und erhält bei einem gültigen Request ein Sessionobjekt.

In Voraussicht auf die dritte Ausbaustufe und zu übungszwecken wurde bereits hierfür ein Webservice implementiert. Um Daten über das Webservice austauschen zu können, wurde das PagingData Objekt angelegt, damit die maximale Payload limitiert wird. Des Weiteren besitzen Webservices auch eine maximale Message Größe.

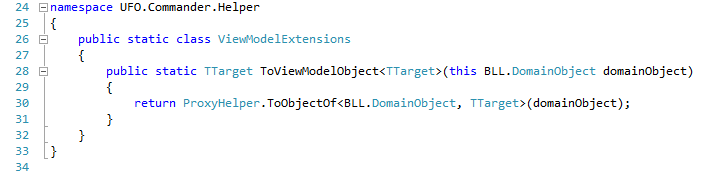
Die im folgenden Diagramm ersichtlich, leiten alle Domänenklassen von DomainObject ab. Der Grund hierfür wird im nachfolgenden Kapitel näher erläutert.

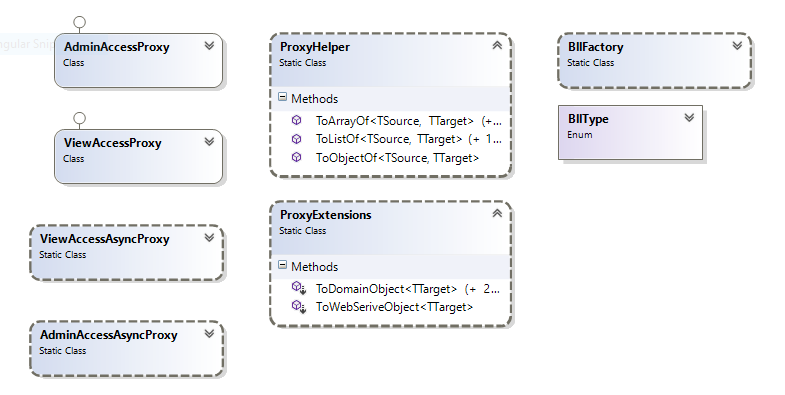
Webservice

Für die Webservice Realisierung wurde WCF verwendet, da sich Client und Server in derselben Technologiesprache C# implementiert sind. Der Webservice verwendet das SOAP basierte Transportprotokoll für Kommunikation. Die Clientklassen werden vom WSDL (serverseitigem Service) generiert.

Um eine Abstraktionsschicht zwischen generierten Klassen und den im WPF Client verwendeten Funktionalen Klassen herzustellen, wurde ein Proxy assembly erstellt. Dieses bietet auch eine Erweiterung (Extension Methods) um Domänenklassen zu Webserviceklassen (und vice versa) mappen zu können, da der Client nur mit den Domänobjekten operieren soll, um Technologie unabhängig zu fungieren.

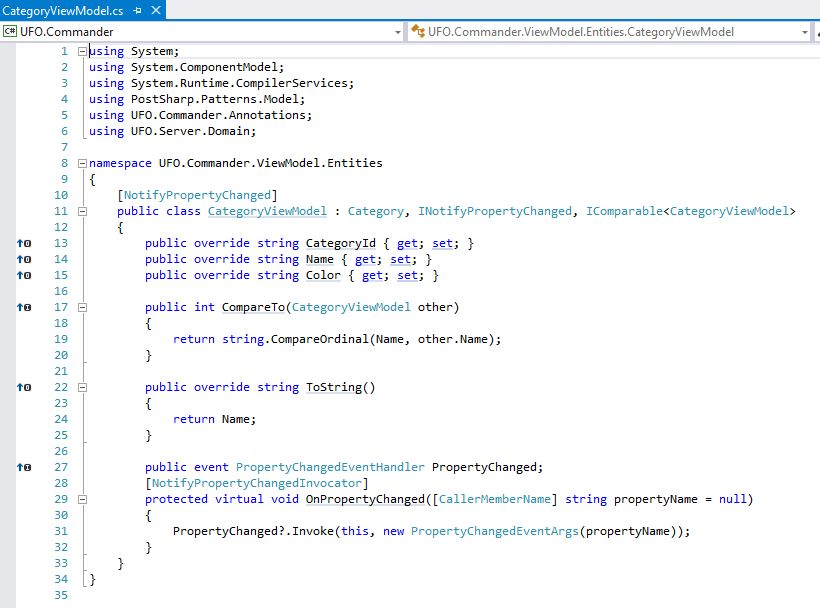
Im folgenden Beispiel wird eine Extensionmethod demonstrativ dargestellt. Diese nützen den Mechanismus der Reflexion, um von einer Klasse zu einer anderen Klasse mit gleichen Properties (Primitivtyp und Name sind equivalent) Daten zu mappen.



Um zwischen dem Webservice und eventuell anderen Implementierungen wechseln zu können, wird das Factorypattern angewendet (BllFactory), welche die Instanziierung der darunterliegenden Schicht vornimmt. 

WPF

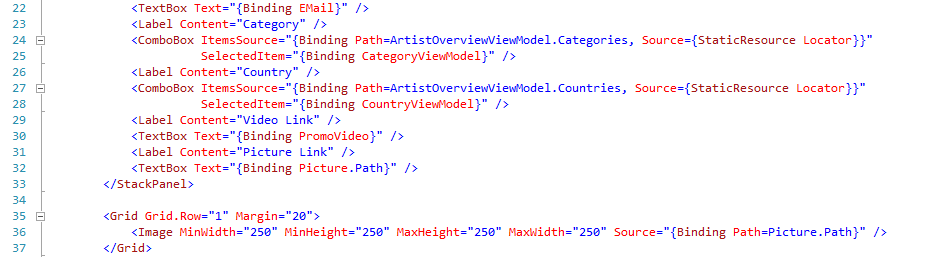
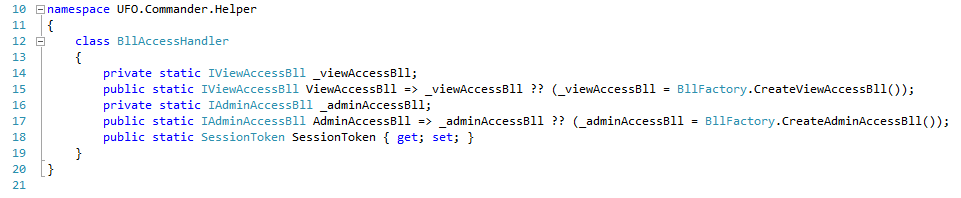
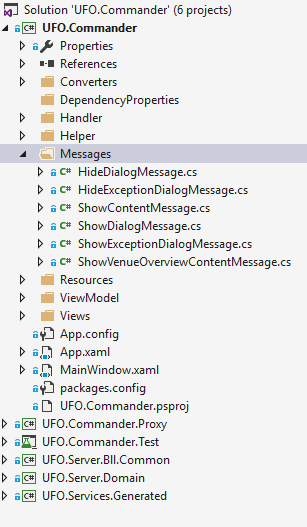
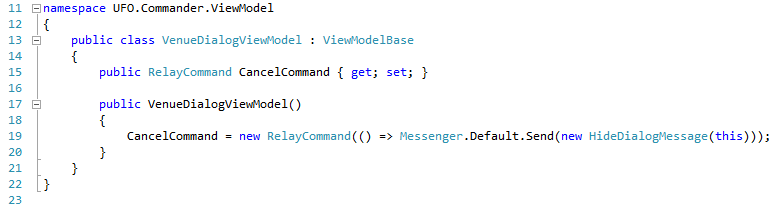
Für die Realisierung des MVVM Patterns, wurden zusätzliche Softwarekomponenten (Frameworks) wie MahApps Metro, MVVM Light und Caliburn.Micro verwendet. Dabei fungiert MahApps Metro als UI-Framework für ein ansehnliches Look & Feel des User-Interfaces. Caliburn.Micro hingegen erweitert die XAML Funktionalität und bietet beispielsweise ein komfortables Möglichkeit für das Binding mit der TreeView Komponente, um das selektierte Element zu erhalten. MVVM Light wird verwendet um ViewModels via DataTemplates auf UI Komponenten projizieren zu können und per ServiceLocator ansprechen zu können. Das heißt konkret, dass Messages gesendet werden können, welche das als nächstes anzuzeigende ViewModel beinhaltet. Aus dieser kann die dazugehörige View gemappt wird.



Das Exceptionhandling wird wie in der ersten Ausbaustufe durch Aspekt orientierte Attribute gesteuert und auf eine Userfreundliche View via Messaging projeziert.Wie in der folgenden Illustrierung erben alle ViewModel-Klassen von ViewModelBase, welche von MVVM Light bereitgestellt wird.



Die Domänen-ViewModels werden jedoch direkt von den eigentlichen Domänenklassen abgeleitet. Und via Attribute für das Event OnPropertyChanged aufbereitet.



Das UserInterface verfügt über drei Mainpages (Artists, Venues und Performances). Bevor mit diesen Pages interagiert werden kann, muss sich der User über den Login-Dialog authentifizieren. Alle Neu hinzuzufügenden Einträge werden auch per Dialog gesteuert. Existierende Einträge können direkt, wie unten zu sehen ist, editiert werden.

