SWK5 UEG2 SEv WS15/16 Ausbaustufe 1

Marius-Constantin Dinu (S1310307054)
Florian Wurm (S1310307112)

22. November 2015

Inhaltsverzeichnis

Ausbaustufe 1

(Abgabe: 22. 11. 2015, 24:00 Uhr)

(A1.1) Sämtliche Komponenten von *UFO* sind mit den in den Lehrveranstaltungen SWK5 und WEA5 behandelten Technologien zu realisieren. In dieser ersten Ausbaustufe benötigen Sie nur das .NET-Framework.

Datenbank/Datenmodell: Alle auf der *UFO*-Plattform erfassten Daten sind in einer relationalen Datenbank zu speichern. Entwerfen Sie ein strukturiertes (normalisiertes) Datenmodell für *UFO*. Erstellen Sie eine Testdatenbank, welche für die Problemstellung repräsentative und ausreichend umfangreiche Daten enthält. Die Qualität und der Umfang der Testdaten fließen in die Bewertung mit ein. Das gilt für alle Ausbaustufen. Selbstverständlich können Sie die Testdaten auch automatisch generieren.

Das Mengengerüst für Ihre Testdaten soll sich an der realen Situation orientieren. An einem Pflasterspektakel sind (pro Jahr) ca. 60 Künstler beteiligt. An jeder der 40 Spielstätten finden pro Tag bis zu 9 Aufführungen statt. Jedes Jahr sind 3 Spieltage vorgesehen. Berücksichtigen Sie in Ihrem System zumindest die in einem Jahr anfallenden Daten. Über Details können Sie sich auf der Homepage der Veranstaltung informieren.

(A1.2) *Datenzugriffsschicht:* Überlegen Sie sich für den Zugriff auf die für dieses Projekt relevanten Entitäten (*Künstler*, *Spielstätte, Aufführung* etc.) die erforderlichen Zugriffsfunktionen und fassen Sie diese zu Interfaces zusammen. Die Geschäftslogik soll ausschließlich von diesen Interfaces abhängig sein.

Überlegen Sie sich ein geeignetes Format für die Repräsentation der Daten. Definieren Sie dazu einfache Klassen (die primär aus Konstruktoren und Properties bestehen), die zum Transport der Daten dienen. Diese Klassen werden häufig als *Domänenklassen* oder *Datentransferklassen* bezeichnet. Im Wesentlichen werden Sie für jede Entität eine entsprechende Domänenklasse benötigen. Die Domänenklassen werden in den Interface-Methoden der Datenzugriffsschicht als Parametertypen verwendet und müssen daher auch der Geschäftslogik bekannt gemacht werden.

Implementieren Sie die Datenzugriffsschicht auf Basis von ADO.NET. Der Einsatz eines OR-Mapping-Werkzeugs (NHibernate, Entity Framework etc.) ist nicht gestattet.

(A1.3) *Unit-Tests:* Testen Sie die Datenzugriffsschicht ausführlich, indem Sie eine umfangreiche Unit-Test-Suite erstellen. Achten Sie auf eine möglichst große Testabdeckung. Erstellen Sie eher mehr, dafür aber kompakte Unit-Tests, die nach Möglichkeit nur einen Aspekt der Funktionalität überprüfen. Für jede Methode sollte zumindest ein Unit-Test existieren. Die Wahl des Testframeworks steht Ihnen frei.

(A1.4) *Dokumentation:* Erstellen Sie für die in den Punkten (A1.2) und (A1.3) angeführten Funktionen eine übersichtliche, gut strukturierte Dokumentation. Ihre Dokumentation sollte zumindest das Datenbankmodell und die Struktur der Datenzugriffsschicht enthalten.

1 Framework Tools

1.1 PostSharp

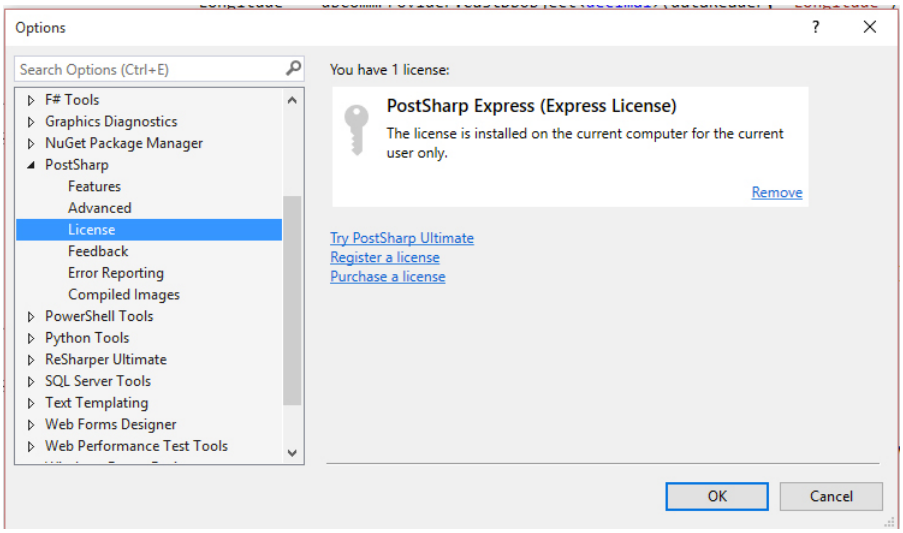
PostSharp (Aspekt orientierte Programmierung): Dieses Tool wird verwendet, um eigene Aspekte bzw. Attribute schreiben zu können. Konkret wurde es für das Exception Handling in der DAL Schnittstelle verwendet. Das Attribut „DaoExceptionhandlerAttribute“ fängt eine auftretende Exception (z.B. bei einer 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.

1.1.1 Installation

Unter folgendem Link kann PostSharp runtergeladen werden.
`https://www.postsharp.net/download`

1.1.2 Registrierung / Account anlegen

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


Options
Search Options (Ctrl+E)
F# Tools
Graphics Diagnostics
NuGet Package Manager
PostSharp
Features
Advanced
License
Feedback
Error Reporting
Compiled Images
PowerShell Tools
Python Tools
ReSharper Ultimate
SQL Server Tools
Text Templating
Web Forms Designer
Web Performance Test Tools
You have 1 license:
PostSharp Express (Express License)
The license is installed on the current computer for the current user only.
Remove
Try PostSharp Ultimate
Register a license
Purchase a license
OK
Cancel

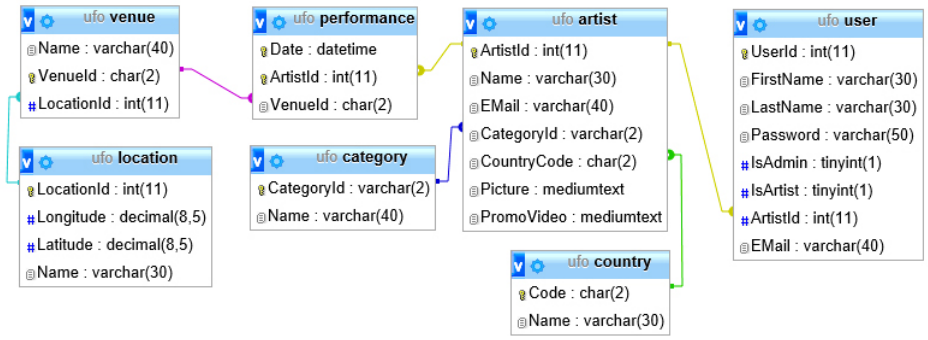
2 Datenbankmodell

2.1 ER-Model

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.



2.2 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.

2.3 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.

2.4 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.

```
private Artist CreateArtistObject(IDataReader dataReader)
{
    var artist = new Artist
    {
        ArtistId = _dbCommProvider.CastDbObject<int>(dataReader, "ArtistId"),
        Name = _dbCommProvider.CastDbObject<string>(dataReader, "ArtistName"),
        EMail = _dbCommProvider.CastDbObject<string>(dataReader, "EMail"),
        PromoVideo = _dbCommProvider.CastDbObject<string>(dataReader, "PromoVideo"),
        Picture = BlobData.CreateBlobData( dbCommProvider.CastDbObject<string>(dataReader, "Picture")),
        Country = new Country
        {
            Code = _dbCommProvider.CastDbObject<string>(dataReader, "CountryCode"),
            Name = _dbCommProvider.CastDbObject<string>(dataReader, "CountryName")
        }
    };
    if (!_dbCommProvider.IsDbNull(dataReader, "CategoryId"))
    {
        artist.Category = new Category
        {
            CategoryId = _dbCommProvider.CastDbObject<string>(dataReader, "CategoryId"),
            Name = _dbCommProvider.CastDbObject<string>(dataReader, "CategoryName")
        };
    }
    return artist;
}
```

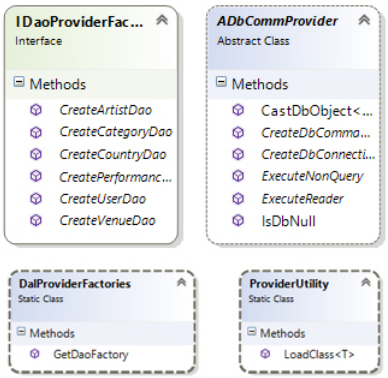
Zusätzlich um Nullwerte aus der Datenbank unterscheiden zu können und zum richtigen Datentyp mappen zu können, werden alle DataReader Zugriffe über die CastDbObject<T> Methode delegiert. Diese sorgt bei Nullwerten für den richtigen Standardwert.

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.

3 Implementation

3.1 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.



3.2 UFO.Server.Dal.Dummy

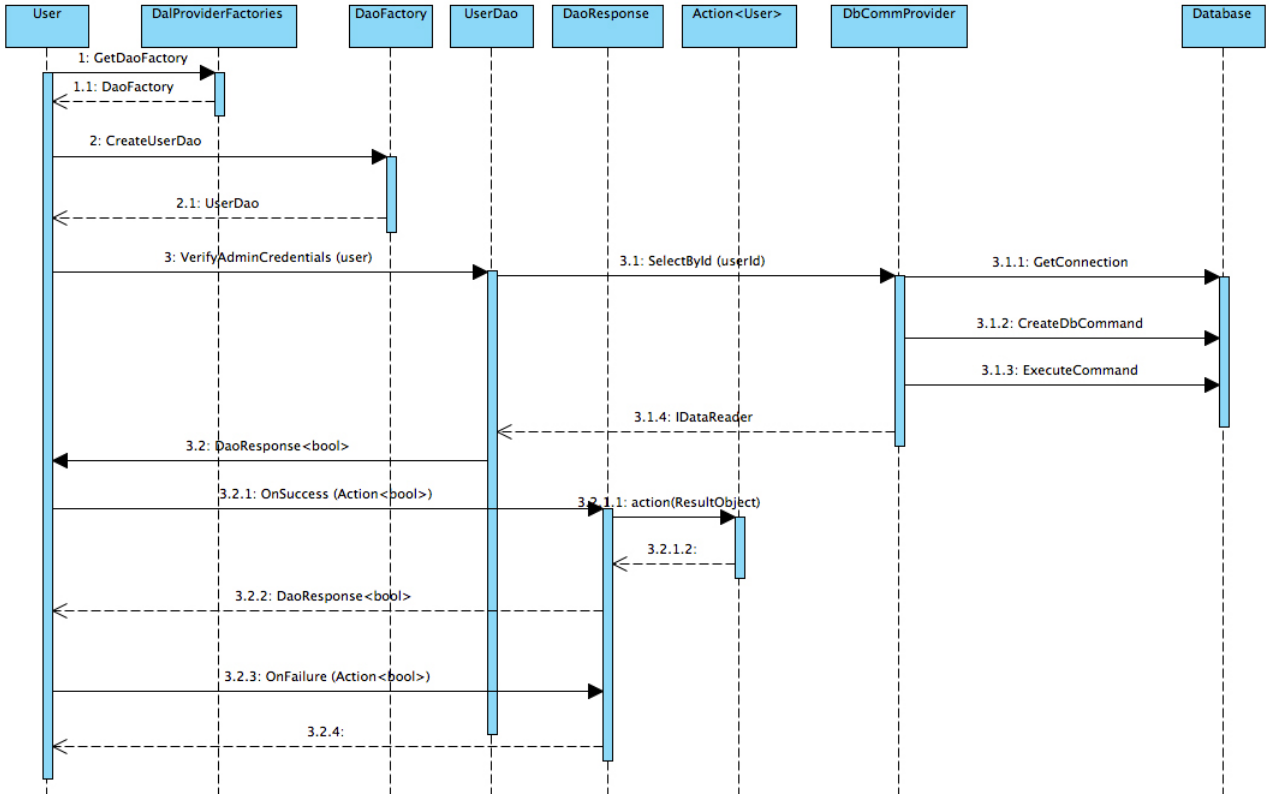
Für die erste Implementierung wurde ein Dummy Assembly erstellt, welches von der IDaoProviderFactory ableitet und demonstrativ den Wechsel der Assemblies veranschaulichen soll. Es wurde jedoch nur ein Bruchteil der DAO Funktionalität implementiert und wird nur noch für Testzwecke verwendet.

3.3 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.

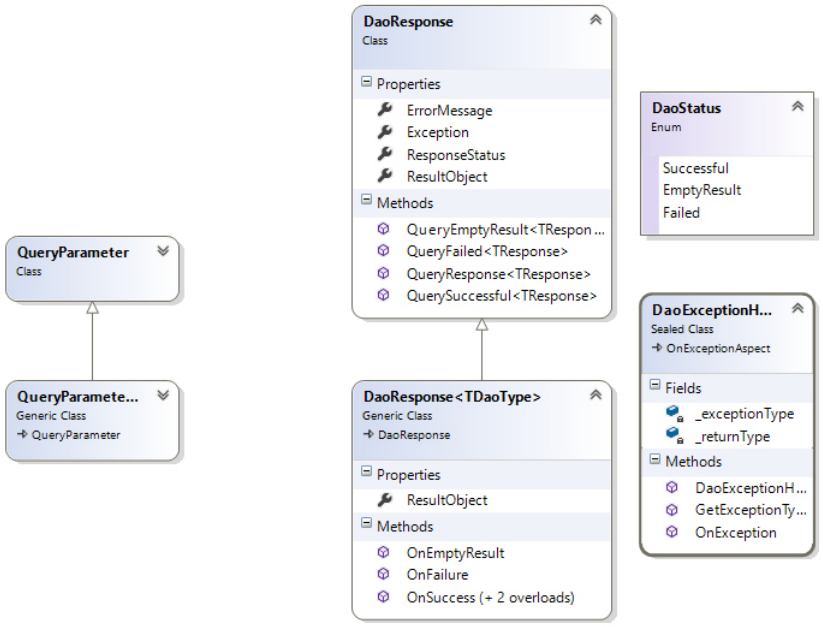
Das Basiskonzept der DAOs beruht auf ein gemeinsamen Responseobjekt, welches als Wrapper für das eigentliche Rückgabeobjekt dient. Dieses bietet zusätzliche Metainformationen und Funktionalitäten. Es wurde nach dem Fluetinterface Modell nachempfunden. Das heißt, das Methoden wie OnSuccessful und OnFailure auch wiederum DAOResponse Objekte (also sich selbst) retournieren und diese per Chaining aufgerufen werden können.
Bsp.:

```
DalProviderFactories.GetDaoFactory()
    .CreateUserDao()
    .VerifyAdminCredentials(user)
    .OnFailure(response => Assert.Fail($"Exception occurred! {response.Exception}"))
    .OnEmptyResult(() => Assert.Fail("User not found!"));
```



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 Status-Code (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.



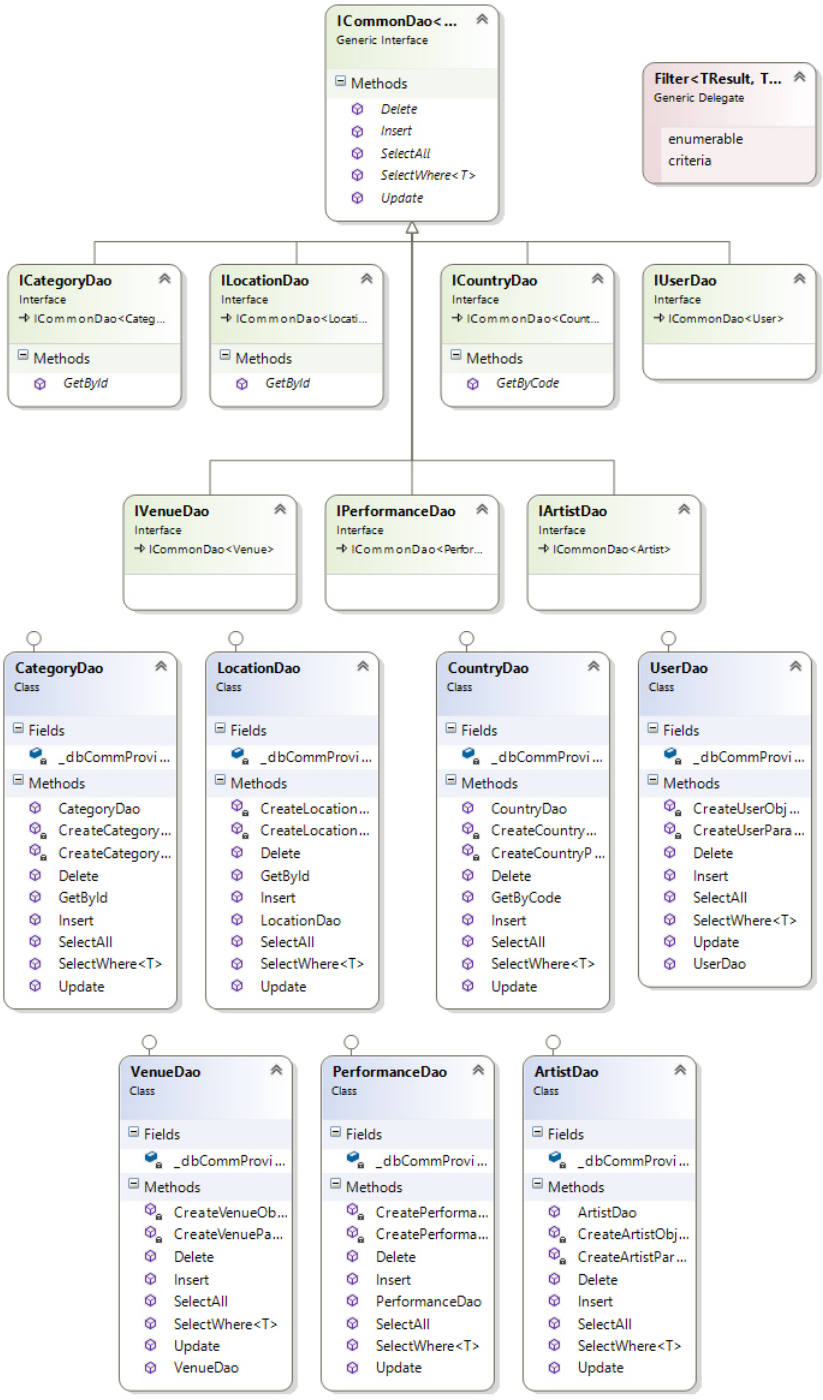
3.4 UFO.Server.Dal.Common

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 abgeän-

dert werden, um somit eine Performance Steigerung zu erzielen. Für Testzwecke wurde auf Basis von SelectWhere auch die Extensionmethods erstellt.

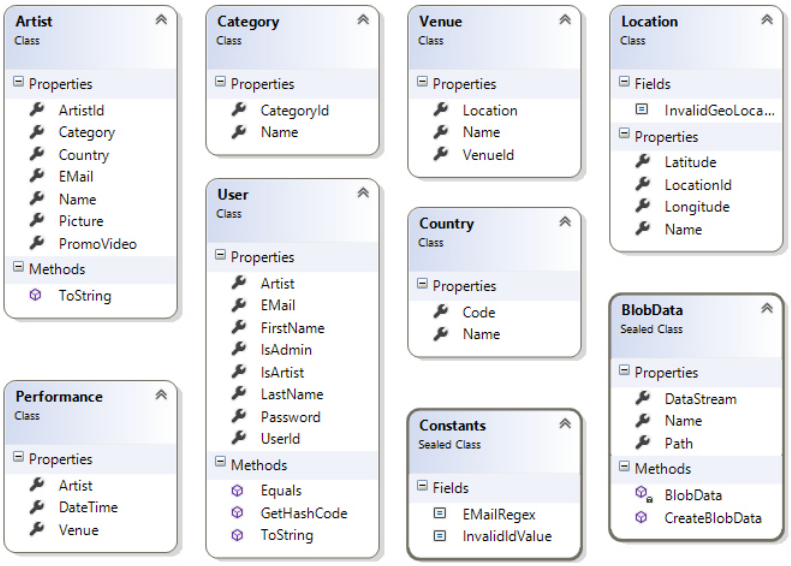


In manchen Fällen ist es Vorteilshaft, wenn zu den DAOs Erweiterungen (Extensions) implementiert werden. So können Interfaces erweitert werden, ohne diese

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.

3.5 UFO.Server.Dal.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.