

SWK5 UEG2 SE_v WS15/16 Ausbaustufe 2

Marius-Constantin Dinu (S1310307054)

20. Dezember 2015

Inhaltsverzeichnis

1	Framework Tools	9
1.1	PostSharp	9
1.1.1	Installation	9
1.1.2	Registrierung / Account anlegen	9
2	Datenbankmodell	11
2.1	ER-Model	11
2.2	Performance	11
2.3	User	11
2.4	Artist	12
3	Implementation	14
3.1	UFO.Server	14
3.2	UFO.Server.Dal.Dummy	14
3.3	UFO.Server.Dal.MySQL	14
3.4	UFO.Server.Dal.Common	16
3.5	UFO.Server.Dal.Domain	18
4	Business Logic Layer	20
4.1	Design	20
4.2	WebService	21
5	WPF	22
5.1	ViewModels	23
5.2	Exception Handling	24
6	User Interface	24

<input type="checkbox"/> Gr. 1 Heinzelreiter	<i>Software Engineering</i>	Übungen zu SWK5/WEA5 im WS 2015/16	
<input type="checkbox"/> Gr. 2 Sklenitzka	Name _____	Aufwand in h _____	
<input type="checkbox"/> Gr. 3 Pimminger			
Abgegeben am _____		Punkte _____	korr. _____

Ultimate Festival Organizer (UFO)

Im kommenden Sommer wird das Pflasterspektakel Linz zum 30. Mal stattfinden. Es ist aber noch immer Stand der Technik, dass das jeweilige Tagesprogramm ausschließlich in Form eines PDF-Dokuments veröffentlicht wird (hier ein Beispiel dafür: http://www.pflasterspektakel.at/2015/medien/pdf/Tagesprogramm_Pflasterspektakel_25_07_2015.pdf) Diese Situation ist für eine moderne Industriestadt sehr unbefriedigend, steht doch mit diesem Festival ganz Oberösterreich in der Auslage. Sie sind daher aufgefordert, mit diesem Projekt den Linzern informationstechnologisch unter die Arme zu greifen.

Ziel dieses Projekts ist es, eine Softwareanwendung zu erstellen, mit dem einerseits die Organisatoren des Pflasterspektakels die Tagesprogramme einfach warten können. Andererseits soll das System den Veranstaltungsbesuchern eine Web-Seite zur Verfügung stellen, über die man das aktuelle Tagesprogramm betrachten und nach verschiedenen Kriterien durchsuchen kann.

Das Softwaresystem soll unter möglichst weitreichender Verwendung der in den Lehrveranstaltungen SWK5 und WEA5 behandelten Architekturkonzepte und Technologien umgesetzt werden. Die Entwicklung erfolgt in drei Ausbaustufen, das Ergebnis jeder Ausbaustufe ist separat abzugeben (Details siehe unten). Beachten Sie, dass sich die im Folgenden beschriebenen Anforderungen auf alle drei Ausbaustufen beziehen.

Funktionale Anforderungen

Mit dem Softwaresystem *UFO*, das im Rahmen dieser Projektarbeit zu entwickeln ist, sollen die für das Pflasterspektakel relevanten Stammdaten gepflegt und die Programme für die verschiedenen Veranstaltungstage erstellt und einfach angepasst werden können. Das System muss den gleichzeitigen Zugriff vieler Benutzer auf die Stamm- und Programmdateien, die dabei nach unterschiedlichen Kriterien gefiltert und gruppiert werden können, gewährleisten. Das Softwaresystem besteht aus einer Serverkomponente, die für die Aufbereitung der Daten und die Verwaltung der Daten in einer relationalen Datenbank verantwortlich ist, und zwei Clients. Mit einem Client kann das Tagesprogramm interaktiv gewartet werden, der andere Client erlaubt den lesenden Zugriff auf das Tagesprogramm.

- *UFO.Server* ist die zentrale Komponente zur Verwaltung von Künstlern, Spielorten und Tagesprogrammen. Die Serverkomponente ist auch für die Authentifizierung und Autorisierung der Benutzer zuständig. *UFO.Server* verfügt auch über eine flexible und effiziente Abfragekomponente, welche die erforderlichen Daten an *UFO.Web* liefert.
- *UFO.WebService* exportiert die gesamte Funktionalität von *UFO.Server* in Form eines Web-Dienstes. Über diesen Dienst können die Klienten der Plattform (*UFO.Commander* und *UFO.Web*) mit dem Server kommunizieren.
- *UFO.Commander* wird dazu verwendet, die Stammdaten für Benutzer, Künstler und Spielstätten zu verwalten, Tagesprogramme zu erstellen und zu aktualisieren.

- *UFO.Web*: Mithilfe dieses Clients können sich Besucher jederzeit einen Überblick über das aktuelle Tagesprogramm machen, können Informationen zu den Künstlern einholen und bekommen Hinweise auf die Spielstätten. Zudem können Organisatoren als registrierte Benutzer Veranstaltungen örtlich sowie zeitlich verschieben (z. B. bei Regen).

(a) Datenmodell

Im Folgenden werden die wesentlichen Attribute der wichtigsten Entitäten des Systems definiert. Aus der Anforderungsspezifikation können sich zusätzliche Entitäten und weitere Eigenschaften der angeführten Entitäten ergeben, die entsprechend zu ergänzen sind.

- (a.1) *User (Benutzer)*: Für jeden Administratoren des Systems sind die Zugangsdaten und seine E-Mail-Adresse zu speichern. Administratoren haben Zugang zu *UFO.Commander* und können in *UFO.Web* Verschiebungen von Aufführungen durchführen.
- (a.2) *Artist (Künstler)*: Diese Entität dient zur Verwaltung der Stammdaten eines Künstlers (bzw. einer Künstlergruppe): Name, Land, Kategorie, Bild, E-Mail-Adresse, ev. Promo-Video etc.
- (a.3) *Category (Darbietungskategorie)*: Diese Entität bezeichnet die Art der Darbietung: Akrobatik, Feuershow, Comedy, Musik etc. Jeder Künstler ist genau einer Darbietungskategorie zugeordnet.
- (a.4) *Venue (Spielstätte)*: Die Spielstätte ist ein Ort, an dem Aufführungen stattfinden können. Eine Spielstätte ist durch folgende Eigenschaften definiert: Bezeichnung, Kurzbezeichnung, geographische Lage etc.
- (a.5) *Performance (Aufführung)*: Jedes Tagesprogramm besteht aus einer Liste von Aufführungen. Für jede Aufführung ist zu hinterlegen, wann (Datum und Uhrzeit) und wo (Spielort) diese stattfindet. Jeder Aufführung sind genau eine Spielstätte und ein Künstler zugeordnet.

(b) UFO.Server

Die Serverkomponente stellt den Festivalorganisatoren (Benutzer mit Rolle Administrator) Funktionalität für die Verwaltung der Stammdaten zur Verfügung. Dies umfasst im Wesentlichen die Verwaltung von Künstlern, Spielstätten und Aufführungen, das Erstellen des Tagesprogramms und eine Möglichkeit, diese Daten abzufragen:

- (b.1) *Verwaltung der Künstler*: Es muss zumindest möglich sein, neue Künstler anzulegen und deren Eigenschaften zu aktualisieren. Wird ein Künstler gelöscht, sollen auch alle in der Zukunft liegenden Aufführungen gelöscht werden. Vergangene Aufführungen eines gelöschten Künstlers sollen aber weiterhin im System gespeichert sein. Zusätzliches Material (z. B. Bilder, Promo-Videos etc.) soll ebenfalls über die Datenbank verwaltet werden. Es ist aber zu gewährleisten, dass Clients diese Mediendaten verwenden können, auch wenn sie nicht auf das Dateisystem des Servers Zugriff haben.
- (b.2) *Verwaltung der Spielstätten*: Es soll möglich sein, Spielstätten anzulegen und zu bearbeiten. Das Löschen von Spielstätten ist hingegen optional.
- (b.3) *Sonstige Stammdaten*: Auf Benutzer und Darbietungskategorien wird nur lesend zugegriffen. Die entsprechenden Datenbestände werden direkt in der Datenbank verwaltet.
- (b.4) *Erstellung und Validierung des Festivalprogramms*: Es soll das Festivalprogramm erstellt/gespeichert und auf die Einhaltung folgender Regeln hin validiert werden:
 - Ein Künstler kann nicht zur gleichen Zeit an mehreren Orten auftreten.
 - Ein Künstler muss immer mindestens eine Stunde Pause zwischen seinen Darbietungen haben.

- (b.5) *Benachrichtigung der Künstler über Änderungen am Festivalprogramm:* Wird das Tagesprogramm verändert, sollen alle Künstler per E-Mail darüber informiert werden. In dieser E-Mail sind alle Aufführungen eines Künstlers mit Ort- und Zeitangabe anzuführen. Es ist darauf zu achten, dass die Künstler nicht mit unnötigen E-Mails belästigt werden.



Im Anhang dieser E-Mail soll sich die neue Version des Festivalprogramms im PDF-Format befinden.

- (b.6) *Abfragen:* Es muss möglich sein, Stammdaten der Künstler, Spielstätten und Aufführungen abzufragen. Eine zentrale Rolle nimmt dabei das Festivalprogramm ein, da die beiden Clients nach unterschiedlichen Kriterien auf den Datenbestand zugreifen müssen. Daher soll vor allem in diesem Bereich besonderes Augenmerk auf eine gut strukturierte, flexible Zugriffs-API gelegt werden. Welche Abfragen die Serverkomponente im Detail unterstützen muss, ist aus den Anforderungen der verschiedenen Clients abzuleiten.

(c) UFO.WebService

Die für *UFO.Web* erforderliche Funktionalität ist in Form eines Web-Service zu exportieren.



Die für *UFO.Commander* erforderliche Funktionalität ist ebenfalls als Web-Service zu exportieren.

(d) UFO.Commander

Diese Anwendung erlaubt es den Administratoren, die Organisation des Pflasterspektakels durchzuführen. Konkret sind folgende Funktionen zu realisieren:

- (d.1) *Login:* Ein Administrator muss sich mit seinem Benutzernamen und seinem Passwort anmelden, bevor er Zugang zum System bekommt.
- (d.2) *Verwaltung von Künstlern und Darbietungskategorien:* Alle Künstler, die am Pflasterspektakel teilnehmen, werden mit den unter (a) beschriebenen Daten und Materialien erfasst bzw. bearbeitet. Jeder Künstler muss einer Darbietungskategorie zugeordnet werden.
- (d.3) *Verwaltung von Spielstätten:* Alle zur Verfügung stehenden Spielstätten werden hier hinzugefügt, bearbeitet oder gelöscht.




Die Festlegung der geografischen Position soll über eine interaktive Karte erfolgen.

- (d.4) *Tagesprogramme*: Hier findet die zentrale Organisation des Pflasterspektakels statt – achten Sie daher besonders auf eine benutzerfreundliche und übersichtliche Darstellung. Bei der Planung ist eine Übersicht der gleichzeitig stattfindenden Aufführungen von großer Bedeutung: man versucht, einen möglichst guten Mix aus Darbietungskategorien anzubieten. Gefordert wird nach der Auswahl eines Tages eine Matrix aus Spielorten und Zeitpunkten mit farblicher Unterscheidung der Darbietungskategorien und freien Slots. Das Warten dieser Matrix – also das Hinzufügen, Löschen und Verschieben von Aufführungen – soll möglichst einfach und intuitiv durchgeführt werden können. Aufführungen beginnen immer zur vollen Stunde.

Hauptplatz		14 - 15 Uhr	15 - 16 Uhr	16 - 17 Uhr	17 - 18 Uhr	18 - 19 Uhr
H1	Dreifaltigkeitssäule	EDDY EIGHTY/ C (Spa)	CIA. FRUTILLAS/ C (Chi)	SARSALÉ FLAM./ M,T (Ita/Spa)	LUCA BELLEZZE/ C (Ita)	MAURANGAS/ C,J (Spa/Arg)
H2	Mader Reisen	COMP. MOBIL/ C (Nie)	DJ CAPUZZI/ A,J (Arg)	MAURANGAS/ C,J (Spa/Arg)	KANA/ J (Jap)	LUCA BELLEZZE/ C (Ita)
H3	Haltestelle	RAF. SORRISO/ A,F (Ita/Bra)	IAN DEADLY/ J,C (UK)	LUTREK STATUES/ ST (Pol)	DJ CAPUZZI/ A,J (Arg)	URBAN SAFARI/ W (Nie)
H4	Altes Rathaus	HOOP HOOLIGANS/ J (Neus)	PALLOTTO/ J,A (Ita)	H. HUNDERTPFU/ C (Deu)	DUO MASAWA/ A (Arg/Ita)	ANNE & MITJA/ A (Deu)
H5	Fa. Mammot	HERR KONRAD/ C,J (Deu)	DEREK DEREK/ L,C (USA)	MR. MOSTACHO/ C,A (Chile)	TOLGA TRIO/ M (Nie/Ita)	THEFREAKS AKR./ A (Öst)
H6	Bank Austria	DUO LOOKY/ A (Isr)	DUO KATE & PASI/ A (Fin)	OLA MUCHIN/ OT (Pol)	DIE BUSCHS/ C (Deu)	SARSALÉ FLAM./ M,T (Ita/Spa)
Pfarrplatz und Domgasse						
P1	Keplersalon				FELIPE J. GARCIA/ OT (Chi)	LUCY LOU (Deu)
P2	Adalbert-Stifter-Platz				HERR KONRAD/ C,J (Deu)	NANIROSSI / A,C (Ita)

Beispiel aus dem Veranstaltungsprogramm

- (d.5) *Benachrichtigung per E-Mail*: Sobald das Tagesprogramm fertig ist, kann der Administrator an alle Künstler eine personalisierte E-Mail mit den Aufführungsdetails senden (siehe auch (b.5)).
- (d.6)  *Exportieren des Tagesprogramms*: Das Tagesprogramm kann in einer druckfähigen Variante (z. B. PDF) exportiert werden.

(e) UFO.Web


Mithilfe der Weboberfläche bekommen die Benutzer eine Übersicht über alle Künstler, Veranstaltungsorte und Programmhöhepunkte. Damit können sich Besucher jederzeit einen Überblick über das aktuelle Tagesprogramm machen, können Informationen (z. B. Bilder, Videos etc.) zu den Künstlern einholen, bekommen Hinweise bezüglich Spielstätten (z. B. auf einer Karte) und der dort dargebotenen Aufführungen. Organisatoren können als registrierte Benutzer Veranstaltungen örtlich sowie zeitlich verschieben (z. B. bei Regen). Linz Termine (www.linztermine.at) kann Ihnen als Ideenlieferant dienen.

Bei *UFO.Web* ist besonders auf eine einfache Benutzbarkeit und eine optisch ansprechende Umsetzung zu achten.

- (e.1) *Programmübersicht*: Auf der Einstiegsseite werden alle in der Zukunft liegenden Aufführungen, Künstler und Spielstätten aufgelistet. Über Filter kann die Programmübersicht nach verschiedenen Kriterien angepasst werden (siehe (e.4)). Standardmäßig werden alle am aktuellen Tag anstehenden Aufführungen dargestellt.
- (e.2) *Künstlerinformation*: Zu jedem Künstler sind diverse interessante Informationen (z. B. Link zu seiner Web-Site, Bilder, Videos etc.) anzuzeigen.

- (e.3) *Informationen zu einer Aufführung*: Zu jeder Aufführung sind relevante Informationen (z. B. Zeitpunkt, Ort, Beschreibung, Bilder, Videos etc.) anzuzeigen, sodass der Besucher vorab abklären kann, ob die Aufführung seinem Interesse entspricht.
- (e.4) *Filter*: An geeigneten Stellen in der Benutzerschnittstelle (z. B. bei der Programmübersicht) soll es dem Benutzer möglich sein, über spezielle Filter Künstler, Standorte, Darbietungskategorie, Zeiträume etc. auszuwählen, sodass er leichter den Überblick behält.
- (e.5) *Kartendarstellung*: Aufführungen sowie durchführende Künstler sind per se Orten zugeordnet und sollen daher in einer Kartendarstellung visualisiert werden.
- (e.6) *Organisatoren*: Organisatoren können als registrierte Benutzer (also über ein Login) Veranstaltungen örtlich sowie zeitlich verschieben oder auch ausfallen lassen (z. B. bei Regen).

Ergebnisse

Die in der Anforderungsdefinition festgelegten Funktionen sind in Einer- oder Zweierteams zu realisieren. Anforderungen, die mit dem Symbol  gekennzeichnet sind, müssen nur von Zweierteams ausgearbeitet werden. Die WEA5 zuzuordnende Funktionalität ist in Form von Einzelarbeiten umzusetzen.

Erstellen Sie für alle Dokumente und Quelltext-Dateien eine übersichtliche Verzeichnisstruktur und packen Sie diese in eine ZIP-Datei. Dieses Archiv stellen Sie Ihrem Übungsleiter auf der E-Learning-Plattform in den Kursen zur SWK5-Übung bzw. zu WEA5 zur Verfügung. Achten Sie darauf, dass Sie nur die relevanten (keine generierten) Dateien in das Archiv geben. Große Archive mit unnötigen Dateien vergeuden Speicherplatz und können daher zu Punkteabzügen führen. Große Binärdateien, wie Komponenten von Drittherstellern oder umfangreiche Datenbanken können sie auch über Online-Speicherdienste zugänglich machen. Vermerken Sie dies aber in der Dokumentation Ihres Projekts.

Konzentrieren Sie sich bei dieser Projektarbeit auf die Dokumentation der Architektur und des Designs Ihrer Anwendung. Verwenden Sie dazu gängige Darstellungsformen wie ER- und UML-Diagramme (Use-Case-, Klassen- und Sequenzdiagramme). Zeigen Sie anhand von zwei aussagekräftigen Anwendungsfällen u. a. die Interaktionen der verschiedenen Systemkomponenten mithilfe eines Sequenzdiagramms – auch über Systemgrenzen hinweg (WEA5 und SWK5). Versuchen Sie diesen Teil der Arbeit besonders übersichtlich zu gestalten. Die Dokumente sind ausschließlich in Form von PDF-Dateien abzugeben.

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 ORM-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>

led.

he following NuGet packages

2. Download PostSharp

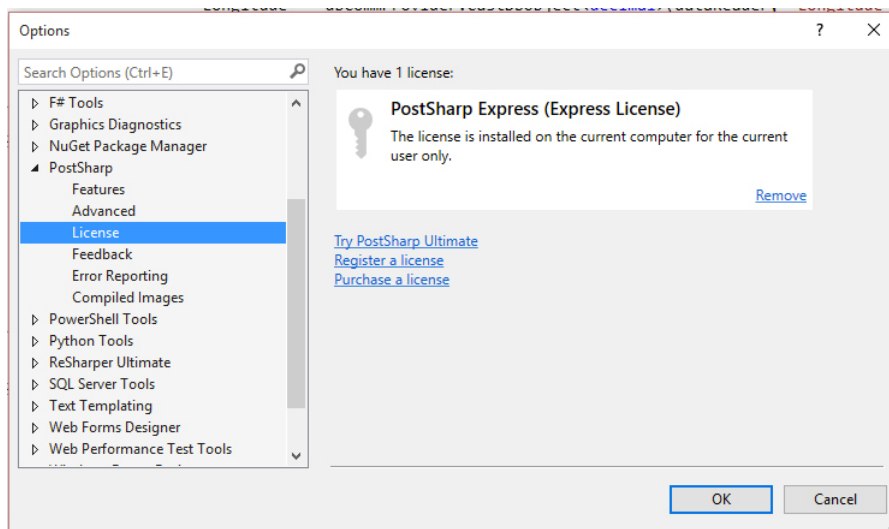
There is a single download for all editions and licenses.
Available features may vary according to the license key you enter.

download PostSharp 4.1.30 (stable)

download PostSharp 4.2.13 (rc)

1.1.2 Registrierung / Account anlegen

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



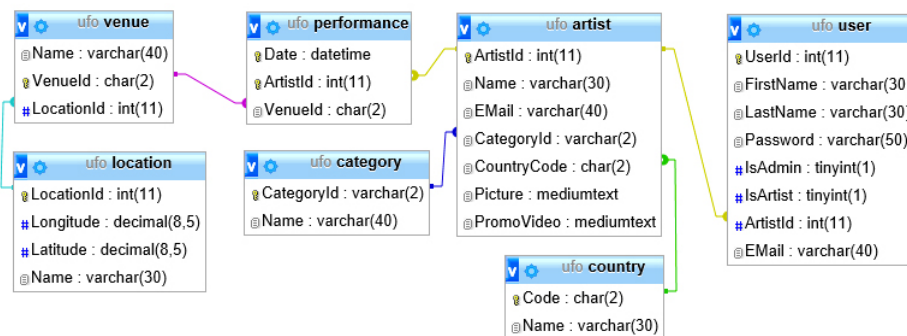
2 Datenbankmodell

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 php-MyAdmin verwaltet und die SQL-Skripte 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 auftritt. 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 abgebildet, 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.

ufo performanceview	ufo userview	ufo artistview	ufo venueview
#ArtistId : int(11) @Date : datetime @VenueId : char(2) @ArtistName : varchar(30) @EMail : varchar(40) @CategoryId : varchar(2) @CategoryName : varchar(40) @CountryCode : char(2) @CountryName : varchar(30) @Picture : mediumtext @PromoVideo : mediumtext @VenueName : varchar(40) #LocationId : int(11) #Longitude : decimal(8,5) #Latitude : decimal(8,5) @LocationName : varchar(30)	#UserId : int(11) @FirstName : varchar(30) @LastName : varchar(30) @UserMail : varchar(40) @Password : varchar(50) #IsAdmin : tinyint(1) #IsArtist : tinyint(1) #ArtistId : int(11) @ArtistName : varchar(30) @ArtistMail : varchar(40) @CategoryId : varchar(2) @CategoryName : varchar(40) @CountryCode : char(2) @CountryName : varchar(30) @Picture : mediumtext @PromoVideo : mediumtext	#ArtistId : int(11) @ArtistName : varchar(30) @EMail : varchar(40) @CategoryId : varchar(2) @CategoryName : varchar(40) @CountryCode : char(2) @CountryName : varchar(30) @Picture : mediumtext @PromoVideo : mediumtext	@VenueId : char(2) @VenueName : varchar(40) #LocationId : int(11) #Longitude : decimal(8,5) #Latitude : decimal(8,5) @LocationName : varchar(30)

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)

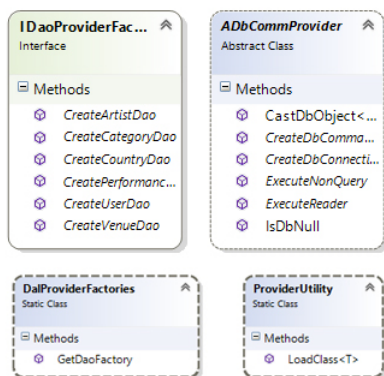
Die jeweiligen SQL Scripten, wurden im Verzeichnis UFO.Database/sql bereitgestellt.

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.

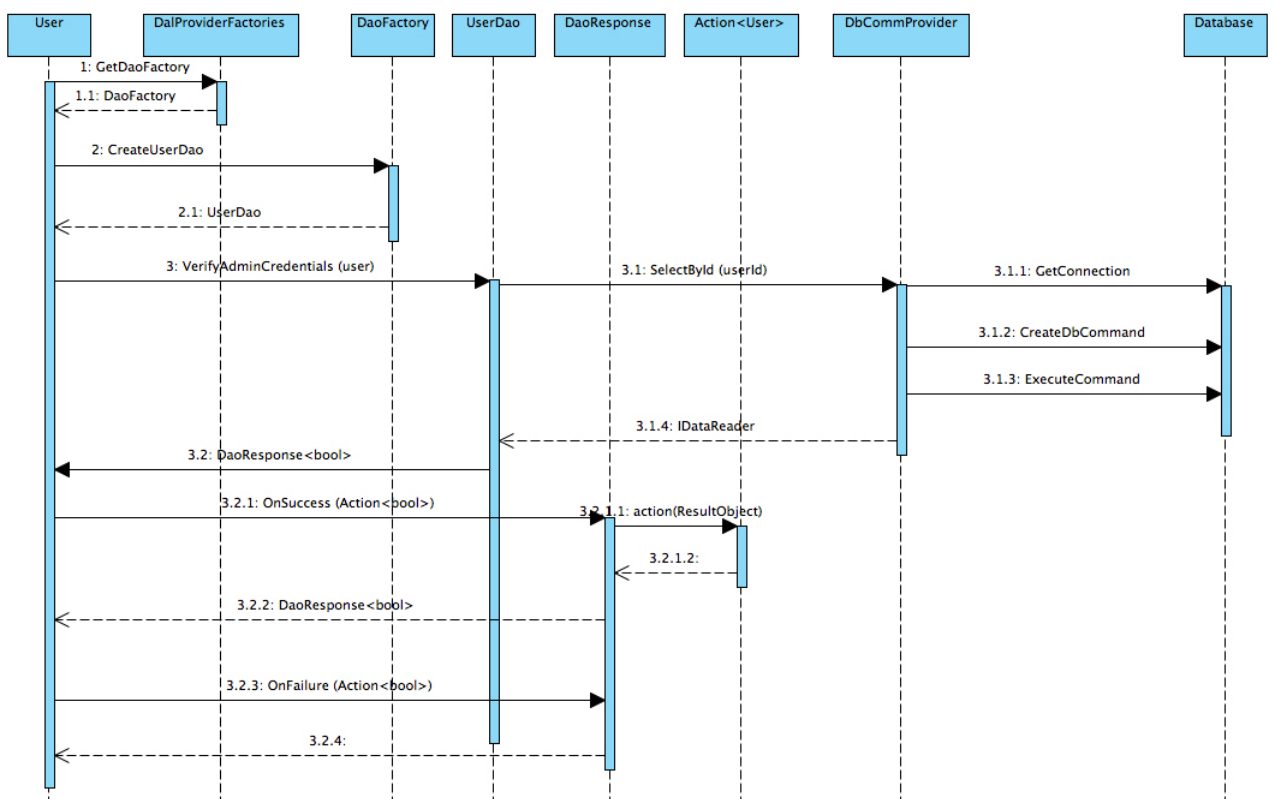
3.3 UFO.Server.Dal.MySQL

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.:

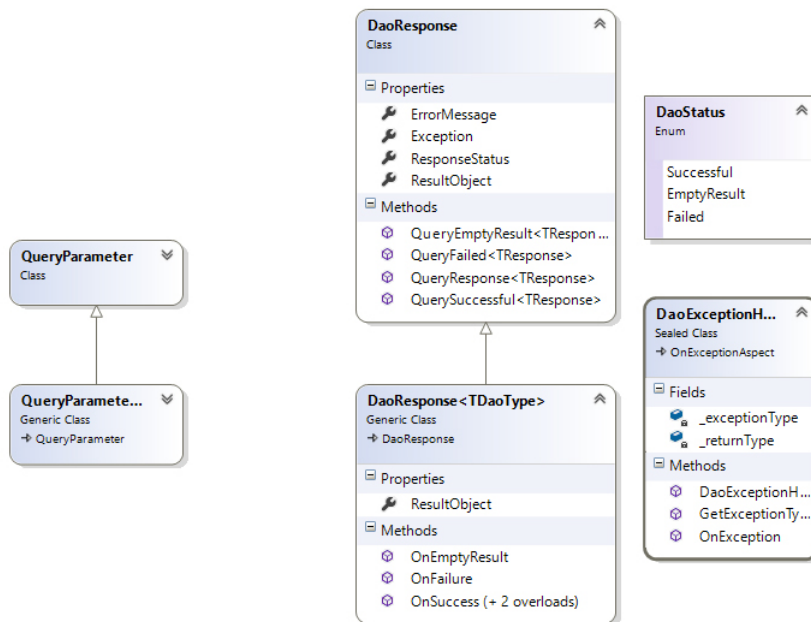
```
DalProviderFactories.GetDaoFactory()
    .CreateUserDao()
    .VerifyAdminCredentials(user)
    .OnFailure(response => Assert.Fail($"Exception occurred! {response.Exception}"))
    .OnEmptyResult(() => Assert.Fail("User not found!"));
```



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**.

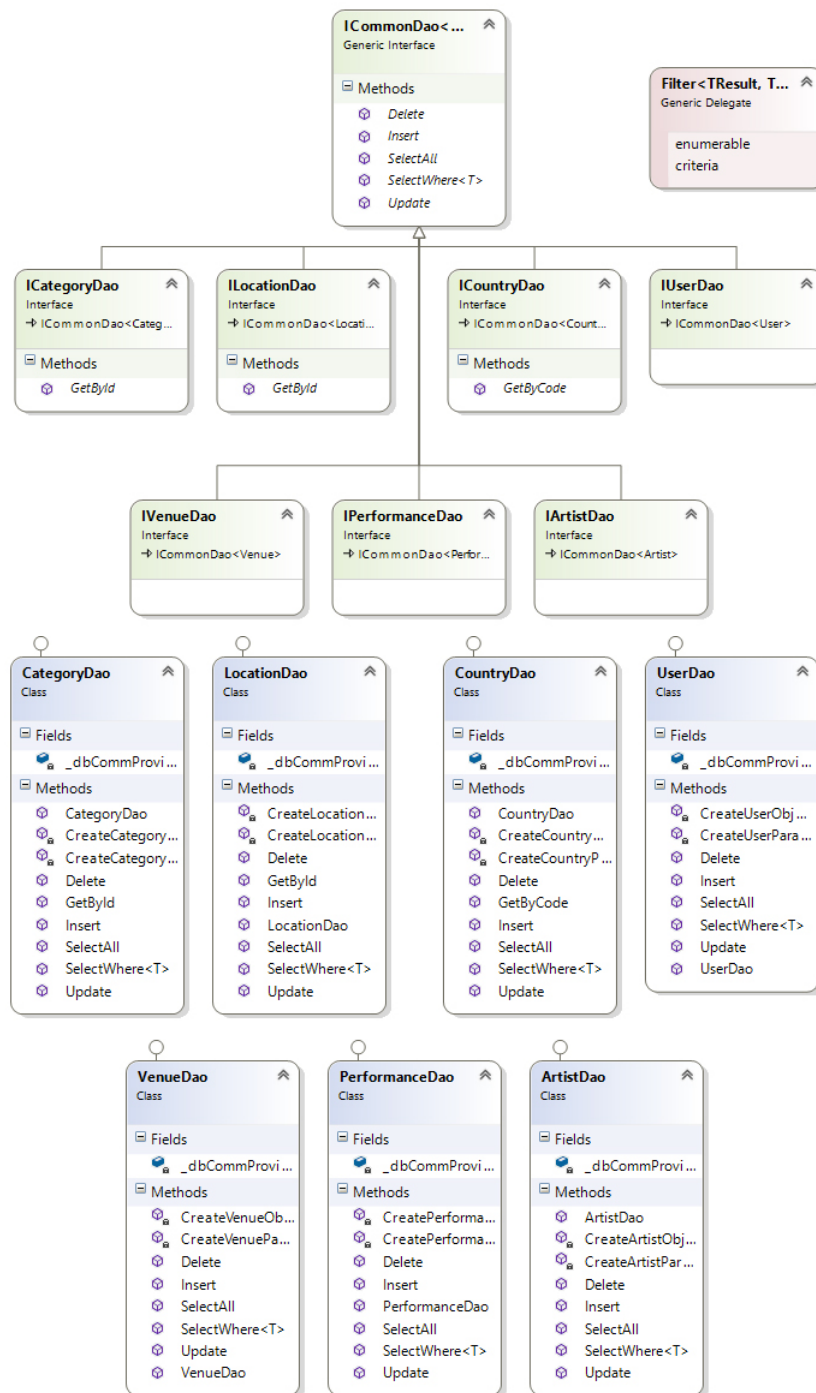
Die Implementierung der DAOs wird nicht direkt vom **ICommonDao** Interface instanziiert, sondern es gibt für jedes DAO noch ein eigenes Interface, welches von **ICommonDao** erbt.

Es kann vorkommen, 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. Desweiteren 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.

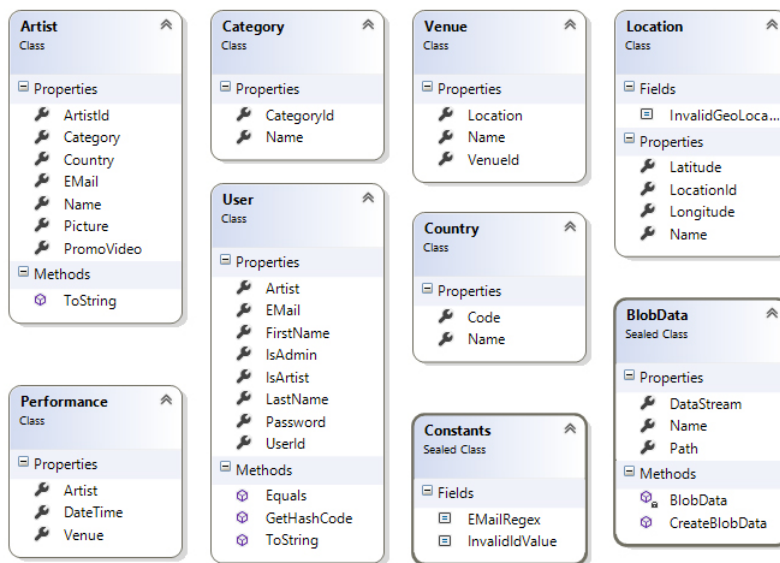


In manchen Fällen ist es Vorteilhaft, wenn zu den DAOs Erweiterungen (Extensions) implementiert werden. So können Interfaces erweitert werden, ohne diese

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.

Ausbaustufe 2

(Abgabe: 20. 12. 2015, 24:00 Uhr)

- (A2.1) *Geschäftslogik*: Implementieren Sie auf Basis der in Ausbaustufe 1 entwickelten Datenzugriffsschicht die gesamte Funktionalität der Komponente *UFO.Server*. Überlegen Sie sich, wie die Klienten auf die (Geschäfts-)Logik zugreifen sollen. Fassen Sie die erforderlichen Funktionen zu logischen Gruppen zusammen und definieren Sie für jede Gruppe ein Interface. Alle Klienten sollen (statisch) ausschließlich von diesen Geschäftslogik-Interfaces abhängig sein.
- (A2.2) *UFO.Commander*: Entwickeln Sie die Benutzeroberfläche von *UFO.Commander* mit Hilfe der WPF. Alternativ kann diese Systemkomponente auch als Windows-Universal-Plattform-App realisiert werden. Koppeln Sie die Benutzeroberfläche mit der Geschäftslogik. Versuchen Sie möglichst weitreichend XAML und andere Konzepte der WPF, wie Ressourcen und Datenbindung, einzusetzen. Sorgen Sie durch Anwendung des Model-View-ViewModel-Musters für eine konsequente Trennung der Logik der Benutzeroberfläche von der Geschäftslogik. Greifen Sie soweit wie möglich auf bestehende Komponenten zurück.
- (A2.3) *Dokumentation*: Führen Sie die notwendigen Ergänzungen in der Dokumentation durch. Erweitern Sie einerseits die Systemdokumentation aus Ausbaustufe 1 und erstellen Sie andererseits eine rudimentäre Benutzerdokumentation. Aus Ihrer Dokumentation soll die Architektur des Gesamtsystems hervorgehen, insbesondere die Struktur der Klassen und Interfaces und die Kopplung der Systemkomponenten sollte übersichtlich dargestellt werden. Hierzu eignen sich besonders gut UML-Klassen- und Paketdiagramme.

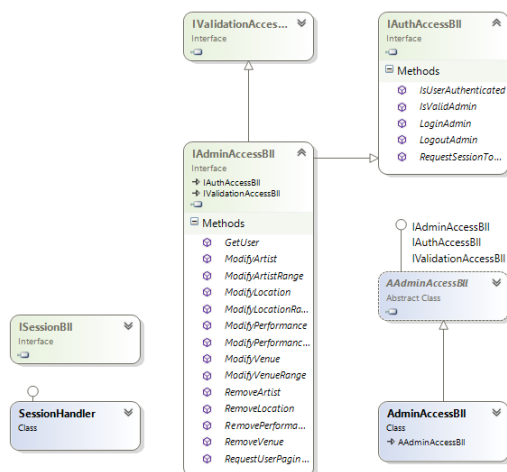
4 Business Logic Layer

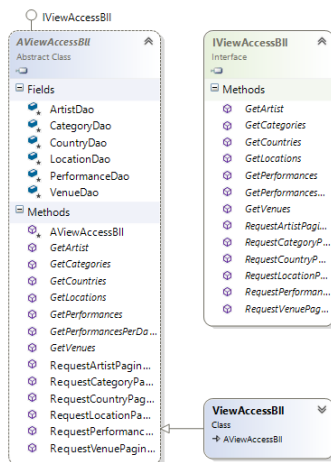
4.1 Design

Der BusinessLogicLayer spiegelt die Kernkomponente der Drei-Schichten-Architektur und bindet die Datenzugriffsschicht mit dem Frontend UI. Hierfür wurden folgende Basis-Interfaces verwendet.

IValidationAccessBll und IAuthAccessBll. Diese werden in ein Rollenkonzept für die Zugriffsberechtigungen wiederverwendet. Hierfür wurden eine administrative und eine öffentliche Rolle definiert, welche von IAdminAccessBll und IViewAccessBll abgeleitet werden. Darauf aufbauend wurde eine abstrakte Basisklasse realisiert, welche kommunale Funktionalitäten implementiert. Von dieser Basis-Klasse werden nun die konkreten Instanzen wie, AdminAccessBll realisiert. Diese bietet die Funktionalität User-, Artist-Daten oder auch andere Domänenobjekte nicht nur anzuzeigen, sondern auch zu modifizieren. Für den öffentlichen Zugriff wurde eine Ableitung von der abstrakten Klasse AViewAccessBll implementiert, welche die öffentlich verfügbaren Daten bereitstellt.

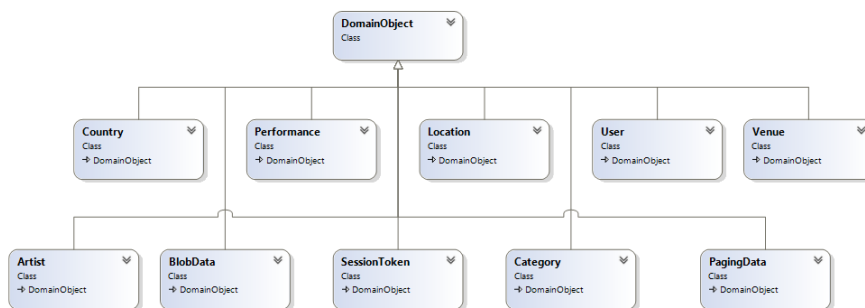
Wie in der Spezifikation gefordert, wurde eine Login Möglichkeit geschaffen, welche Serverseitig verwaltet wird. Die Applikation wurde dahingehend erweitert, dass mehrere Login-Instanzen unterstützt werden das heißt hierfür wird ein serverseitiges Sessionhandling verwendet. Loggt sich beispielsweise ein User mit der Frontendapplikation ein, wird für ihn eine Session am Server angelegt und sofern eine zweite Instanz von einem weiteren Client mit selben Authentifizierungsdaten angefragt wird, wird die erste Session gecancelt.





Die Domaenenklassen 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 Domaenenklassen von DomainObject ab. Der Grund hierfür wird im nachfolgenden Kapitel näher erläutert.



4.2 Webservice

Für die Webservice Realisierung wurde WCF verwendet, da sich Client und Server in derselben Technologiesprache CSharp 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

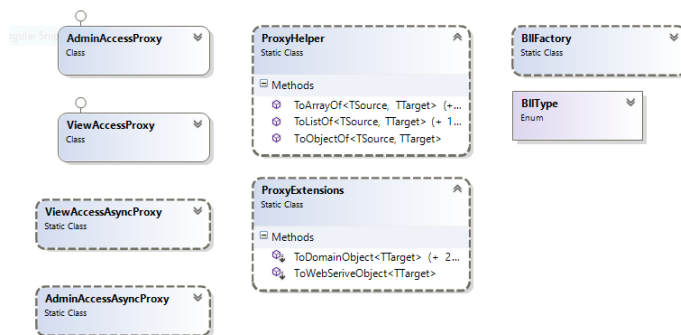
auch eine Erweiterung (Extension Methods) um Domänenklassen zu Webserviceklassen (und vice versa) mappen zu koennen, da der Client nur mit den Domänenobjekten operieren soll, um Technologie unabhaengig zu fungieren. Im folgenden Beispiel wird eine Extensionmethod demonstrativ dargestellt. Diese nutzen den Mechanismus der Reflexion, um von einer Klasse zu einer anderen Klasse mit gleichen Properties (Primitivtyp und Name sind equivalent) Daten zu mappen.

```

24 namespace UFO.Commander.Helper
25 {
26     public static class ViewModelExtensions
27     {
28         public static TTarget ToViewModelObject<TTarget>(this BLL.DomainObject domainObject)
29         {
30             return ProxyHelper.ToObjectOf<BLL.DomainObject, TTarget>(domainObject);
31         }
32     }
33 }
34

```

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



Am Client wird an zentraler Stelle via Singleton auf die BllFactory zugegriffen.

```

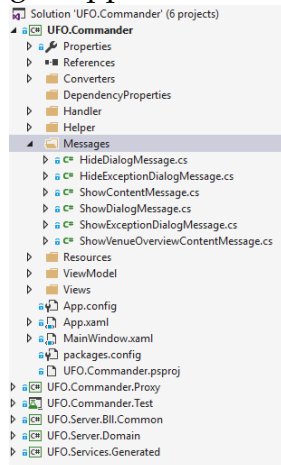
10 namespace UFO.Commander.Helper
11 {
12     class BllAccessHandler
13     {
14         private static IViewAccessBll _viewAccessBll;
15         public static IViewAccessBll ViewAccessBll => _viewAccessBll ?? (_viewAccessBll = BllFactory.CreateViewAccessBll());
16         private static IAdminAccessBll _adminAccessBll;
17         public static IAdminAccessBll AdminAccessBll => _adminAccessBll ?? (_adminAccessBll = BllFactory.CreateAdminAccessBll());
18         public static SessionToken SessionToken { get; set; }
19     }
20 }
21

```

5 WPF

Fuer die Realisierung des MVVM Patterns, wurden zusaetzliche Softwarekomponenten (Frameworks) wie MahApps Metro, MVVM Light und Caliburn.Micro verwendet. Dabei fungiert MahApps Metro als UI-Framework fuer ein ansehnliches Look & Feel des User-Interfaces. Caliburn.Micro hingegen erweitert die XAML Funktionalitaet und bietet beispielsweise ein komfortables Moeglichkeit fuer 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 koennen und per ServiceLocator ansprechen zu koennen. Das heisst konkret, dass Messages gesendet werden koennen, welche das als naechstes

anzuweisende ViewModel beinhaltet. Aus dieser kann die dazugehoerige View gemappt werden.



Alle ViewModels werden vom Service Locator an zentraler Stelle verwaltet.

```

28 public class Locator
29 {
30     public Locator()
31     {
32         ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);
33
34         SimpleIoc.Default.Register<MainViewModel>();
35         SimpleIoc.Default.Register<LoginViewModel>();
36         SimpleIoc.Default.Register<TabControlViewModel>();
37         SimpleIoc.Default.Register<ArtistDialogViewModel>();
38         SimpleIoc.Default.Register<ArtistOverviewViewModel>();
39         SimpleIoc.Default.Register<ExceptionDialogViewModel>();
40         SimpleIoc.Default.Register<CustomViewDialog>();
41         SimpleIoc.Default.Register<CustomExceptionDialog>();
42         SimpleIoc.Default.Register<VenueOverviewViewModel>();
43         SimpleIoc.Default.Register<VenueEditViewModel>();
44         SimpleIoc.Default.Register<LocationEditViewModel>();
45         SimpleIoc.Default.Register<VenueDialogViewModel>();
46         SimpleIoc.Default.Register<LocationDialogViewModel>();
47         SimpleIoc.Default.Register<PerformanceOverviewViewModel>();
48     }
49 }

```

Nun kann per Service Locator beispielsweise im XAML das benoetigte ViewModel fuer das DataContext-Binding verwendet werden.

```

22 <TextBox Text="{Binding Email}" />
23 <Label Content="Category" />
24 <ComboBox ItemsSource="{Binding Path=ArtistOverviewViewModel.Categories, Source={StaticResource Locator}}"
25           SelectedItem="{Binding CategoryViewModel}" />
26 <Label Content="Country" />
27 <ComboBox ItemsSource="{Binding Path=ArtistOverviewViewModel.Countries, Source={StaticResource Locator}}"
28           SelectedItem="{Binding CountryViewModel}" />
29 <Label Content="Video Link" />
30 <TextBox Text="{Binding PromoVideo}" />
31 <Label Content="Picture Link" />
32 <TextBox Text="{Binding Picture.Path}" />
33 </StackPanel>
34
35 <Grid Grid.Row="1" Margin="20">
36     <Image MinWidth="250" MinHeight="250" MaxHeight="250" MaxWidth="250" Source="{Binding Path=Picture.Path}" />
37 </Grid>

```

5.1 ViewModels

Wie in der folgenden Illustrierung erben alle ViewModel-Klassen von ViewModel-Base, welche von MVVM Light bereitgestellt wird.

```

11 namespace UFO.Commander.ViewModel
12 {
13     public class VenueDialogViewModel : ViewModelBase
14     {
15         public RelayCommand CancelCommand { get; set; }
16
17         public VenueDialogViewModel()
18         {
19             CancelCommand = new RelayCommand(() => Messenger.Default.Send(new HideDialogMessage(this)));
20         }
21     }
22 }
23

```

Fuer jede benoetigte Doaenenklasse wurde ein dazugehoeriges ViewModel erstellt. Hierbei wurde von den Domaenenklassen abgeleitet, um via Attributen das NotifyPropertyChanged Event auf die Properties abbilden zu koennen.

```
CategoryViewModel.cs
1 using System;
2 using System.ComponentModel;
3 using System.Runtime.CompilerServices;
4 using PostSharp.Patterns.Model;
5 using UFO.Commander.Annotations;
6 using UFO.Server.Domain;
7
8 namespace UFO.Commander.ViewModel.Entities
9 {
10     [NotifyPropertyChanged]
11     public class CategoryViewModel : Category, INotifyPropertyChanged, IComparable<CategoryViewModel>
12     {
13         public override string CategoryId { get; set; }
14         public override string Name { get; set; }
15         public override string Color { get; set; }
16
17         public int CompareTo(CategoryViewModel other)
18         {
19             return string.CompareOrdinal(Name, other.Name);
20         }
21
22         public override string ToString()
23         {
24             return Name;
25         }
26
27         public event PropertyChangedEventHandler PropertyChanged;
28         [NotifyPropertyChangedInvocator]
29         protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
30         {
31             PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
32         }
33     }
34 }
35
```

5.2 Exception Handling

Das Exceptionhandling wird wie in der ersten Ausbaustufe durch Aspekt orientierte Attribute gesteuert und auf eine User-freundliche View via Messaging projiziert.

```
13 namespace UFO.Commander.Handler
14 {
15     [Serializable]
16     [MulticastAttributeUsage(MulticastTargets.Method | MulticastTargets.InstanceConstructor, AllowMultiple = false)]
17     public class ViewExceptionHandlerAttribute : OnExceptionAspect
18     {
19         protected string Title;
20         protected string Message;
21
22         public ViewExceptionHandlerAttribute(string title = "", string message = "")
23         {
24             this.Title = title;
25             this.Message = message;
26         }
27
28         [Log]
29         public override void OnException(MethodExecutionArgs args)
30         {
31             args.FlowBehavior = FlowBehavior.Continue;
32             var viewModel = Locator.ExceptionDialogViewModel;
33             viewModel.Exception = args.Exception;
34             viewModel.Title = Title;
35             viewModel.Message = Message;
36             Messenger.Default.Send(new ShowExceptionDialogMessage(viewModel));
37         }
38     }
39 }
40
41
```

6 User Interface

Das User Interface verfuegt ueber drei Mainpages (Artists, Venues und Performances). Bevor mit diesen Pages interagiert werden kann, muss sich der User ueber den Login-Dialog authentifizieren. Alle Neu hinzuzufuegenden Eintraege werden auch per Dialog gesteuert. Existierende Eintraege koennen direkt, wie unten zu sehen ist, editiert werden.














UFO Login

e-mail
password
<input type="button" value="Login"/> <input type="button" value="Cancel"/>



UFO

Performances Artists Venues

NEW USER					Name
SAVE					Dead Island
	Candlebox	ewauhuag@vodew.gov	Akrobatik	Argentina	DELETED
	Canned Heat	tu@mip.org	Akrobatik	Argentina	DELETED
	The Babys	fovvogaf@uhnaj.io	Samba	Argentina	DELETED
	The Wallflowers	jazokza@jovocc.co.uk	Samba	Argentina	DELETED
	The Band	for@satobvah.io	Samba	American Samoa	DELETED
	The White Stripes	kud@itgibe.gov	Samba	American Samoa	DELETED
	Dead Island	untitled-no1@mail.com	Musik	Austria	DELETED
	.38 Special	di@tasopjis.edu		Austria	DELETED
	AC/DC	dakag@eje.com	Musik	Austria	DELETED
	Ace	coel@voti.org	Tanz	Austria	DELETED
	Adelitas Way	ed@fiifeisi.gov	Tanz	Austria	DELETED

Name

Dead Island

E-Mail

untitled-no1@mail.com

Category

Musik

Country


Austria

Video Link

<https://www.youtube.com/watch?v=gEIEAHwE2d4>

Picture Link

<http://einfogames.com/news/files/2013/04/zz2.jpg>



UFO
Performances Artists Venues

NEW VENUE
NEW LOCATION

Hauptplatz
Pfarrplatz und Domgasse
Altstadt
Klosterstrasse 7
Landhaus
Altstadt 13
Alter Markt
Hofgasse 13
Schlossmuseum
Landstrasse
Promenade
Straßentheater

Name
Klosterstrasse 7
Location
Altstadt
SAVE
DELETE

UFO
Performances Artists Venues

	14:00	15:00	16:00	17:00	18:00	19:00	20:00	21:00	22:00	23:00	00:00
Altstadt											
Venuel: A1, Venu +	+	+	+	+	+	+	+	Billy Idol	+	+	+
Venuel: A2, Venu +	+	+	+	+	+	+	+	+	Bryan Adams	+	+
Landstrasse											
Venuel: L7, Venu: Black Sabbath	+	+	+	+	+	+	+	+	+	+	+
Venuel: L8, Venu: +	Southside Johnny	+	+	+	+	+	+	+	+	+	+
Pfarrplatz und Domgasse											
Venuel: P1, Venu +	+	Tom Petty & The	+	+	+	+	+	+	+	+	+
Venuel: P3, Venu +	+	+	+	The Lovin Spoonf	+	+	+	+	+	+	+
Venuel: P4, Venu +	+	+	+	+	Saga	+	+	+	+	+	+
Venuel: P5, Venu +	+	+	+	+	+	Steven Stills	+	+	+	+	+