

1 UFO (Ultimate - Festival - Organizer)

Folgende Dokumentation stellt die Gesamtdokumentation der aufbauenden Übung UFO dar, die im Zuge der Realisierung iterativ über die drei Ausbaustufen hinweg erweitert wird.

1.1 Ausbaustufe 1 (ADO.NET)

Folgender Teil dokumentiert die erste Ausbaustufe der aufbauenden Übung UFO. In diesem Teil wird die Persistenz Schicht in .NET unter Hilfenahme von ADO.NET implementiert. Aufgrund der Analyse der Gesamtaufgabenstellung wurde entschieden das vorerst nur die Persistenz Schicht an sich, also INSERT, UPDATE, DELETE der einzelnen Entitäten realisiert wird, da die Geschäftslogik erst bei der Realisierung der Administration und des Webzugriffs endgültig feststehen wird.

Im Zuge der Realisierung des Webzugriffs wird auch der Web-Service implementiert werden müssen, der die Daten der Web Applikation zur Verfügung stellt. Dieser soll die Daten bereits gefiltert und strukturiert zur Verfügung stellen, daher besteht eine gewisse Abhängigkeit zwischen dem Web-Service und der Web Applikation sowie auch der Client Administration.

Daher werden die Web-Service Implementierung und die Administration die eigentliche Geschäftslogik enthalten, die in einer Transaktion abgearbeitet und im wesentlichen aus den logischen Prüfungen gegen die Datenbank bestehen wird sowie der Speicherung und Löschung von Entitäten über die Administration. Die einzelnen Datenabfragen, die benötigt werden können einfach hinzugefügt werden.

1.1.1 Systemaufbau

Folgend ist der Systemaufbau der Persistenz Schicht dokumentiert.

Die folgende Auflistung illustriert die Projektstruktur der Persistenz Schicht:

1. *UFO.Server.Data.Api*
Dieses Projekt enthält die Spezifikation der Persistenz Schicht in Form von Interfaces, abstrakten Klassen, Exceptions und den Entitäten, die wie bei einem ORM-Mapper DB unabhängig sein sollen (sofern möglich).
2. *UFO.Server.Data.MySql*
Dieses Projekt enthält die MySql spezifische Implementierung der Persistenz Schicht.
3. *UFO.Server.Test.Data.MySql*
Dieses Projekt enthält die MySql spezifischen Tests der Persistenz Schicht.
4. *UFO.Common.Util*
Dieses Projekt enthält die Utilities für die UFO Infrastruktur in .NET, die nicht spezifisch einen Systemteil zuzuordnen sind.
5. *ufo-data-generator*
Ein kleines Java Projekt welches eine Java Main Klasse enthält und die benötigten Ressourcen um das Testdatenskript zu erstellen. Hierzu wurde *freemarker* verwendet.

Alle .NET Systemteile werden unter dem Namensraum *UFO.** zusammengefasst.

Übung 3

Die folgende Auflistung illustriert die verwendeten Technologien und Frameworks:

1. *MySql*
Es wird eine MySql Datenbank verwendet, die Open-Source ist und eine Integration in .NET besitzt.
2. *NUNIT*
Als Test-Framework wird NUNIT verwendet, da es mehr Funktionalität mitbringt als das Standard Test-Framework integriert in .NET.
3. *ADO.NET*
Als Persistenz Provider wird wie gewünscht ADO.NET und kein ORM Mapper verwendet.
4. *freemarker*
Template-Engine in Java mit der das Testdaten SQL Skript erstellt wird.

1.1.2 Datenbank

Es wurde als Datenbank MySQL und Modellierungstool MySQL Workbench gewählt, da diese Datenbank erstens Open-Source, zweitens eine gute .NET Integration vorhanden ist sowie drittens bereits Erfahrungen mit dieser Datenbank vorhanden waren.

Es wurden folgende Skripten generiert die einerseits für die Tests und andererseits für die Generierung der Testdaten genutzt werden. Diese Skripten befinden sich im Projekt *UFO.Server.Data.MySql/Resources*.

1. *createDatabase.sql*
Ein vollständiges Skript für das Anlegen der Datenbank mit allen Constraints und verwendeten Triggern.
2. *deleteDatabase.sql*
Ein Skript welches alle Einträge in der Datenbank löscht.
3. *dropDatabase.sql*
Ein Skript für das Droppen der Datenbank UFO.
4. *createTestData.sql*
Ein Skript welches die Testdaten generiert.

Die Testdaten wurden mit einer Java Applikation mit Hilfe von *Freemarker* erstellt, welches eine Template Engine darstellt. Die Daten werden von **.csv* Dateien zur Verfügung gestellt und anschließend über eine Java Konsolen Applikation verarbeitet. Diese Applikation liest die Daten ein, verpackt diese in Pojos, generiert dynamische Daten, wie z.B.: die Aufführungen mit den Aufführungszeiten und stellt diese Daten einem Template zur Verfügung.

Übung 3

Das folgende ER-Diagramm illustriert das implementierte Datenbank Schema.

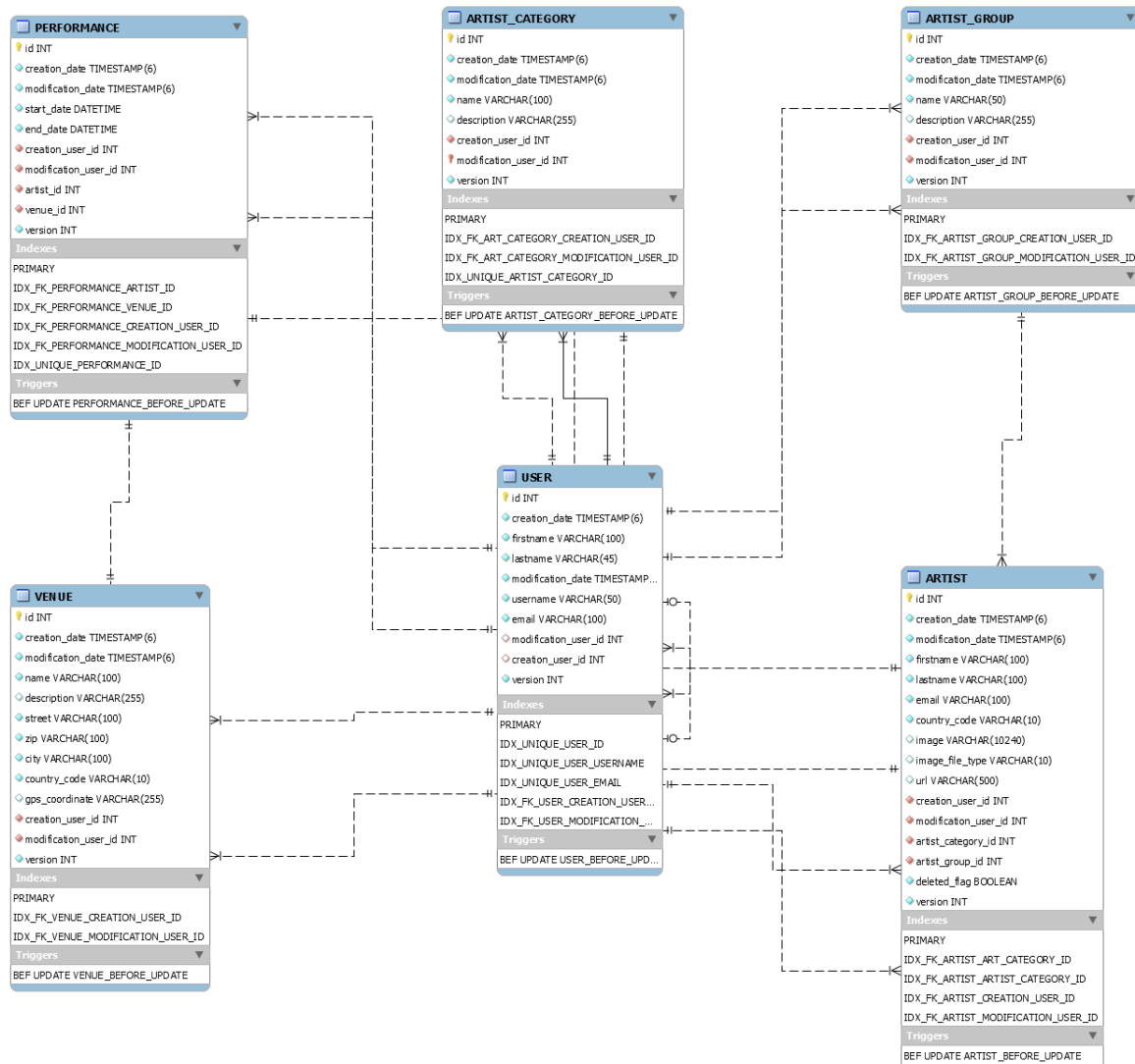


Abbildung 1: ER-Diagramm des Schema 'UFO'

Auf jeder Entität wurde eine Spalte für die Versionierung eingeführt (MySQLDbType.LONG), die über einen Update-Trigger bei jedem Update um eins erhöht wird sowie auch das Modifizierungsdatum. Ein ganzzahliger Datentyp erscheint hier mehr sinnvoll, da es hier mit Sicherheit keine Kollisionen geben kann, nicht so wie bei einem Zeit Datentyp.

Ebenso halten alle Entitäten eine Referenz auf den Benutzer der Sie erstellt sowie zuletzt modifiziert hat. Dies dient der Nachverfolgbarkeit von Änderungen, zumindest wer zuletzt eine Änderung vorgenommen hat.

Übung 3

1.1.3 Klassenhierarchien

Folgend sind die Klassenhierarchien der implementierten Klassen und Interfaces dokumentiert.

IDao

Folgendes Klassendiagramm zeigt die Hierarchie des Interfaces *IDao*, das der Basistyp für alle implementierten DAO Interfaces dient, da es bereits alle Basisaktionen auf eine Entität definiert.

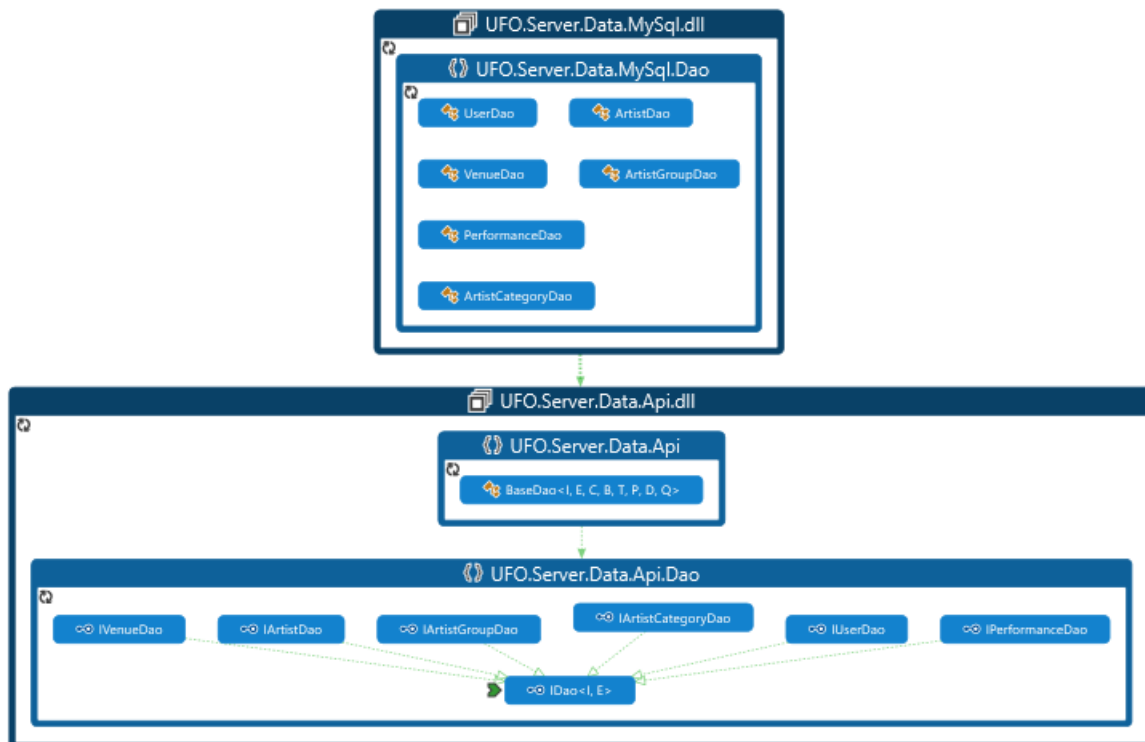


Abbildung 2: Klassenhierarchie IDao Interface

Die Basisklasse *BaseDao* implementiert alle Methoden, die in *IDao* definiert wurden für alle implementieren Entitäten sofern sie *IEntity* implementieren. Dieser generische und abstrakte Ansatz erlaubt es dass die Basisfunktionalität eines DAOs nur einmal für alle Entitätstypen, die *IEntity* implementieren, implementiert werden musste.

Die *DAOs* für die einzelnen Entitäten werden zukünftig Methoden implementieren, die spezifische Datenbankabfragen realisieren, die z.B.: eine komplexe *where clause* aufweisen, die ihrerseits wieder einen Teil der Geschäftslogik enthält, welche noch nicht vollständig bekannt ist.

IEntity

Folgendes Klassendiagramm illustriert die Klassenhierarchie des Interfaces *IEntity*, welches den Basistyp für alle Entitäten darstellt.

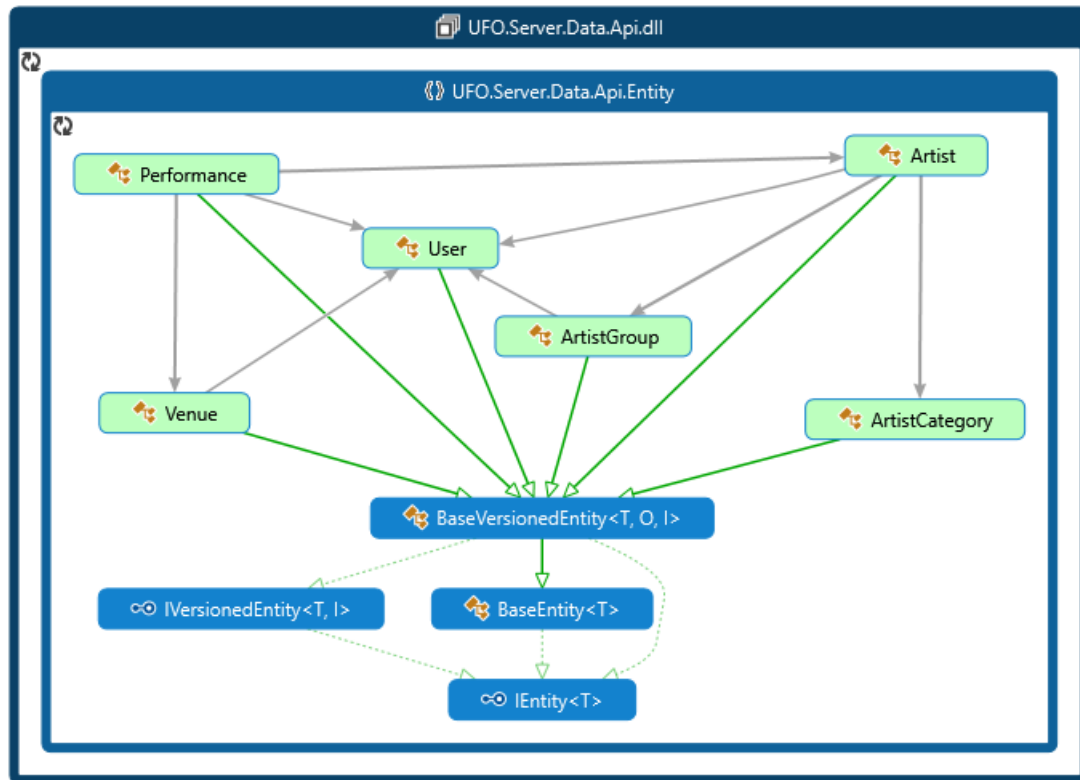


Abbildung 3: Klassenhierarchie IEntity Interface

Es wurden Basisentitäten eingeführt, welche eine Basisstruktur definieren die sich Entitäten unterwerfen müssen wenn sie von diesen Klassen ableiten. Somit wird eine konsistente Struktur der Entitäten bzw. deren Tabellenrepräsentation gewährleistet.

Die Aufteilung von *IEntity* und *IVersionedEntity* wurde eingeführt, da eine Entität nicht zwangsweise versionierbar sein muss. Ebenso wurde mit den abstrakten Klassen verfahren, die jetzt eine Hierarchie abbilden anstatt die gesamte Funktionalität in eine abstrakte Klasse zu packen.

IEntityHelper

Folgendes Klassendiagramm illustriert die Klassenhierarchie des Interfaces *IEntityHelper* welches die Utility Methoden für das generieren von Entitäten definiert.

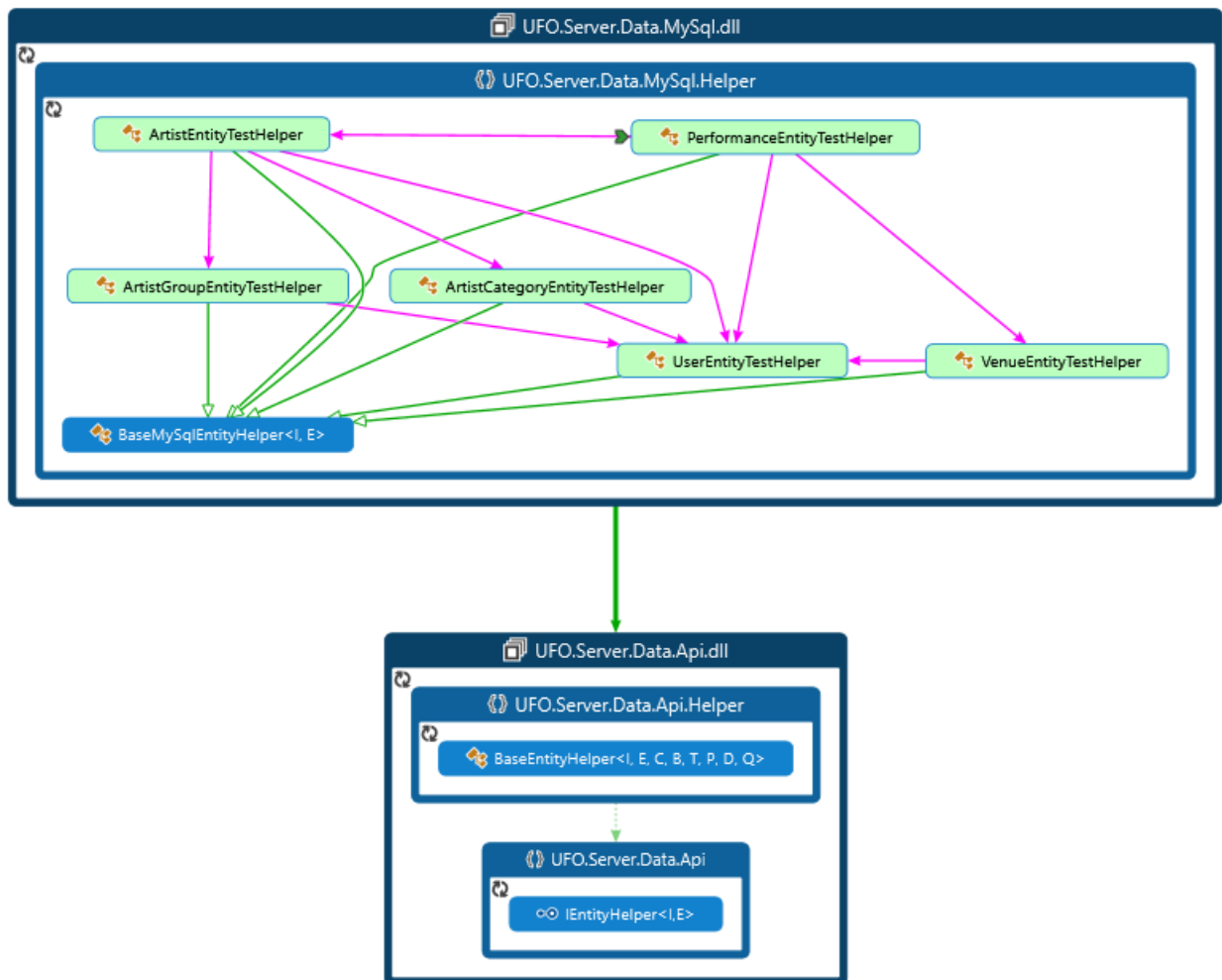


Abbildung 4: Klassenhierarchie IEntityHelper Interface

Dieses Interface und dessen Implementierungen dienen als Hilfestellung für die Test und die Generierung von Testdaten über die Entitäten Modelle. Die abstrakte Basisklasse *BaseEntityHelper* implementiert einige der Interface Methoden und stellt eine Persistenzprovider unabhängige Implementierung für das einfache Speichern von Entitäten zur Verfügung. Diese Hilfsklassen entstanden aufgrund der generischen Testklasse *BaseDaoTest*, die die Entitäten nicht erzeugen kann und daher diese von außen zur Verfügung gestellt werden müssen.

BaseCommandBuilder

Folgendes Klassendiagramm illustriert die abstrakte Klasse *BaseCommandBuilder* die das Handling mit einen *DbCommand* beinhaltet.

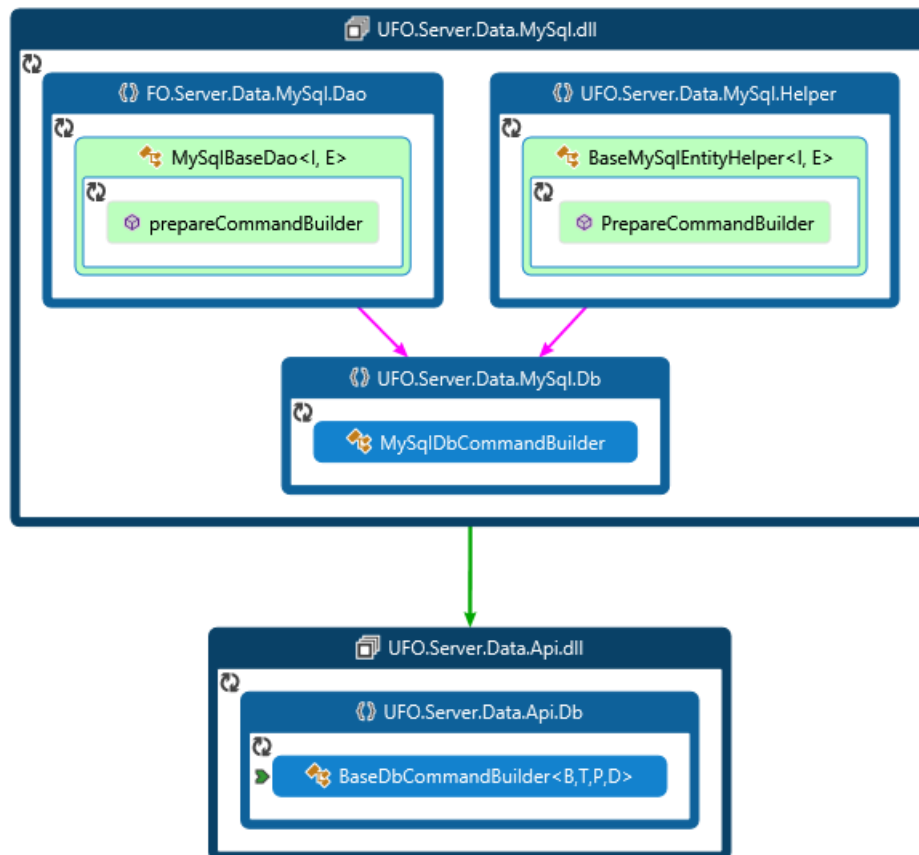


Abbildung 5: Klassenhierarchie der abstrakte Klasse *BaseCommandBuilder*

Um zu vermeiden sich mit dem Code des Erstellens, Modifizierens und Verwerfens eines *DbCommand*, in unserem Fall ein *MySqlDbCommand*, herumschlagen zu müssen, wurde beschlossen eine Hilfsklasse einzuführen, die uns dieses Handling mit einem *DbCommand* abnimmt. Da sich hier eine Fluent-API gut anwenden lässt, wurde diese Funktionalität in Form eines Builder abgebildet.

IQueryCreator

Folgendes Klassendiagramm illustriert das Interface *IQueryCreator*, die die Datenbank spezifischen Statements enthält.

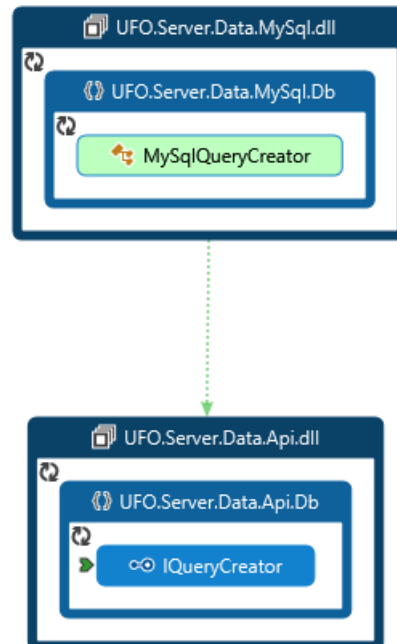


Abbildung 6: Klassenhierarchie des Interface *IQueryCreator*

Diese Interface abstrahiert die Datenbank spezifischen Statements von der Klient Logik. Ebenso erlaubt sie es alle Statements in einem Interface abzubilden und diese an einer Stelle pro Entität zu bündeln.

1.1.4 Hilfsklassen

Im folgenden werden die Hilfsklassen beschrieben, die eingeführt wurden um sich wiederholende und daher immer wiederkehrende Funktionalitäten zu kapseln und zentral zur Verfügung zu stellen.

EntityMetamodel

Diese Klasse löst die Meta-Informationen eines *IEntity* Typs auf und stellt diese aufbereitet nach außen zur Verfügung. Da sich diese Daten zur Laufzeit nicht ändern und daher nur einmalig erzeugt werden sollen, wird eine Factory *EntityMetamodelFactory* eingeführt, die das Caching der *EntityMetamodel* übernimmt.

EntityBuilder

Diese Klasse wurde eingeführt um die Transformation von den Entitäten zur Datenbank und visa versa zu unterstützen, wobei hier einerseits die Werte der Properties, die auf die Datenbank serialisierbar sind, ausgelesen und auf den Property gemapped werden und andererseits die De-Serialisierung vom *IDataReader* zu einer Entität.

IDbTypeResolver

Implementierungen dieses Interface lösen einen C# Typ in einen korrespondierenden Datenbank spezifischen Typ auf.

DbConnectionFactory

Diese Klasse erstellt und verwaltet die verwendeten *DbConnection* Instanzen. Der Typ der zu verwendenden *DbConnection* wird über die *App.config* definiert, sowie der *ConnectionString*.

Übung 3

1.1.5 Tests

Die Tests bestehen aus einer einzigen Testklasse, die das *IDao* Interface bzw. die dessen Ableitungen bzw. dessen Implementierungen, die zurzeit nur aus der Implementierung in *BaseDao* bestehen. Also die Basis Dao Funktionalitäten beinhalten wie.

1. *dao.ById // Throws Exception*
2. *dao.Find // Returns null*
3. *dao.Update*
4. *dao.Persist*
5. *dao.Delete*
6. *dao.CheckIfExists*

Die generische Testklasse *BaseDaoTest* wird mit *NUNIT* Attributen versehen, die die Testklasse mit den zur Verfügung gestellten Typen instanzieren. Danach wird in der Setup Methode die verwendeten Ressourcen über Reflection instanziiert und in der Tear-Down Methode disposed. Hier ist die Typinformation zur Laufzeit sehr Hilfreich. In Java als Beispiel würde hier dies Javas Type Erasure unmöglich machen.

```
[TestFixture(typeof(long?), typeof(User), typeof(UserDao), typeof(UserEntityTestHelper))]
[TestFixture(typeof(long?), typeof(ArtistGroup), typeof(ArtistGroupDao), typeof(ArtistGroupEntityTestHelper))]
[TestFixture(typeof(long?), typeof(ArtistCategory), typeof(ArtistCategoryDao), typeof(ArtistCategoryEntityTestHelper))]
[TestFixture(typeof(long?), typeof(Artist), typeof(ArtistDao), typeof(ArtistEntityTestHelper))]
[TestFixture(typeof(long?), typeof(Venue), typeof(VenueDao), typeof(VenueEntityTestHelper))]
[TestFixture(typeof(long?), typeof(Performance), typeof(PerformanceDao), typeof(PerformanceEntityTestHelper))]
[CreateDatabase]
0 references | Thomas Herzog, 21 hours ago | 1 author, 3 changes
public class BaseDaoTest<I, E, D, C> where E : class, IEntity<I>
    where D : class, IDao<I, E>
    where C : class, IEntityHelper<I, E>
{
    // Here we depend on naming convention of factory method names
    // because here we get the dao from the factory via reflections
    protected D dao;
    protected IEntityHelper<I, E> entityHelper;

    [SetUp]
    0 references | Thomas Herzog, 21 hours ago | 1 author, 2 changes
    public void Init()
    {
        dao = typeof(DaoFactory).GetMethod("Create" + typeof(D).Name).Invoke(null, null) as D;
        entityHelper = Activator.CreateInstance(typeof(C)) as C;
        entityHelper.Init();
        Console.WriteLine("setup called");
    }

    [TearDown]
    0 references | Thomas Herzog, 21 hours ago | 1 author, 2 changes
    public void Dispose()
    {
        Console.WriteLine("tear down called");
        dao?.Dispose();
        entityHelper?.Dispose();
    }
}
```

Abbildung 7: Ausschnitt aus der Testklasse *BaseDaoTest*

1.2 Ausbaustufe 2 (Commander)

Folgender Teil dokumentiert die zweite Ausbaustufe des Projektes *UFO* in dem die Administration mit WPF implementiert werden sollte.

Folgende Projekte wurden der Solution hinzugefügt.

1. *UFO.Commander.ServiceApi*
Dieses Projekt enthält die Schnittstellen Spezifikation für den Service Layer.
2. *UFO.Commander.Service.Impl*
Dieses Projekt enthält die Implementierungen der Service Schnittstellen Spezifikationen
3. *UFO.Commander.Wpf.Administration*
Dieses Projekt enthält die WPF Anwendung, die die Administration abbildet.

Des Weiteren wurde der Wurzelnamensraum auf *UFO* beschränkt unter dem jetzt alle Projekte liegen.

1.2.1 Mögliche Probleme

Mir ist aufgefallen dass Code im statischen Kontext beim Visual Designer zu Problemen geführt hat, da dieser anscheinend evaluiert wird. Sollten Sie also auf Probleme stoßen so liegt es an folgender Klasse *UFO.Commander.Service.Api.Base.ServiceFactory*.

Des Weiteren viel mir auf das MySql anscheinend in meiner Version bei einem Prefix diesen nicht in das ResultSet übernimmt. Also eine Query folgendermaßen aussehen könnte.

SELECT artist., venue.* FROM ...* ein Result liefert wie *id, name, id, name*. Sollten Sie eine neuere Version als *5.6.20* könnte dies auf der Performance View Probleme verursachen, da ich davon ausgehe dass diese Verhalten auftritt.

1.2.2 Klassenhierarchien

Folgender Teil der Dokumentation dokumentiert die definierten Klassenhierarchien der WPF Anwendung und des Service Layers.

Übung 3

IService

Folgendes Klassendiagramm zeigt die Hierarchie des Interfaces *IService*, welche die Wurzel aller Service Interfaces darstellt und *IDisposable* erweitert. Somit ist jeder abgeleiteter Service dazu gezwungen die Methode *Dispose* zu implementieren um dort seine gebundenen Ressourcen freizugeben. Des Weiteren wurde eine Basisklasse namens *BaseService* eingeführt, welche die Datenbankverbindung und Transaktionsmethoden implementiert, sodass die abgeleiteten Klassen diese Nutzen können.

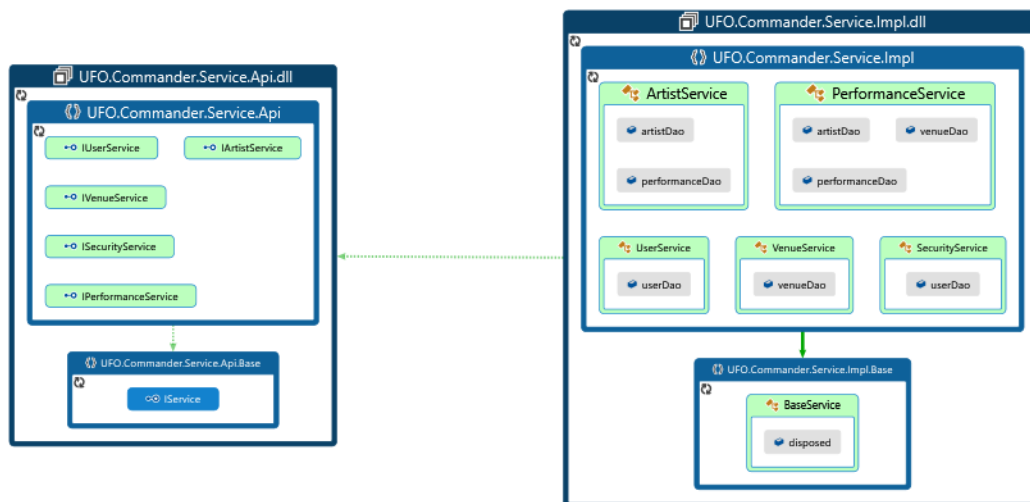


Abbildung 8: Basis Interface für Service Interfaces und Implementierungen

Alle Services bekommen in Ihrem Konstruktor eine *DbConnection* Instanz übergeben, da alle verwendeten *DAO* Implementierungen dieselbe Datenbank Verbindung nutzen müssen, damit alle sich in derselben Transaktion bewegen.

ITabModel

Folgendes Klassendiagramm zeigt die Hierarchien des Interfaces *ITabModel*, die Operationen definiert, die von der Klasse *TabController* verwendet werden. Die Klasse *TabController* wurde eingeführt um *ITabModel* Instanzen zu initialisieren und beim Wechseln eines Tab aufzuräumen.

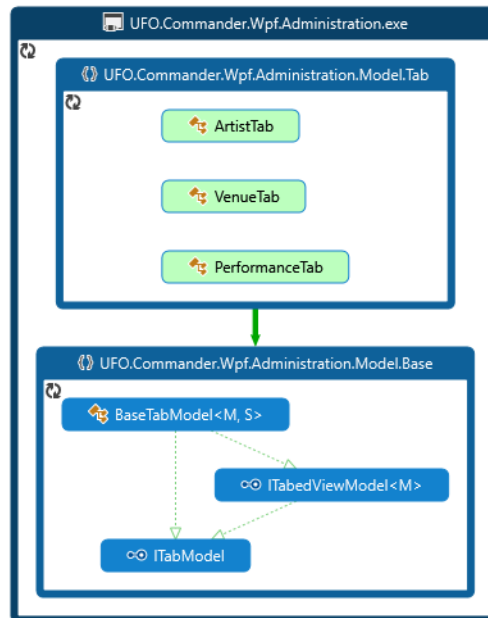


Abbildung 9: Basis Interface für Tab-Model Implementierungen

Das Interface *ITabbedViewModel* definiert die Struktur der Implementierten Tab-Klassen, somit verhält sich jeder Tab gleichermaßen und kann beliebig erweitert werden, je nach seinem View-Content.

BasePropertyChangeViewModel

Folgendes Klassendiagramm zeigt die Hierarchien der Basisklasse *BasePropertyChangeViewModel*, die die Wurzelklasse aller implementierten *View-Models* darstellt, da es immer mindesten einen Property gibt der diesen Event benötigt. Des Weiteren wurde die Klasse *BaseValidationViewModel* eingeführt, die die erste Ableitung von *BasePropertyChangeViewModel* darstellt und die Logik für die Validierung über *System.ComponentModel.DataAnnotations* realisiert. Von dieser Klasse erbt die Basisklasse *BaseEntityViewModel*, die als Wrapper für ViewModels verwendet wird, die eine *IEntity* Instanz für die View wrappen.

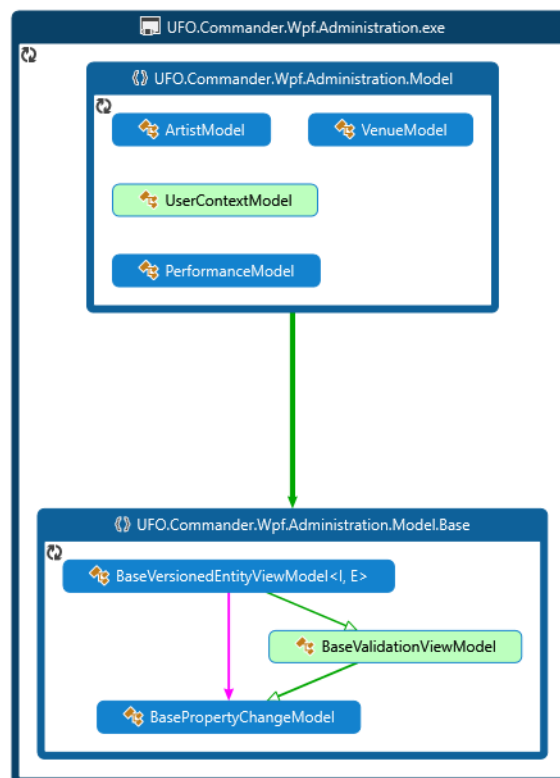


Abbildung 10: Basisklasse für View-Models

Die Klasse *UserContextModel* wurde eingeführt um einen eingeloggten Benutzer zu repräsentieren und wird als statischer Property der Klasse *App* definiert, da es nur einen eingeloggten Benutzer pro gestarteter Anwendung geben kann. Alle Klassen, die auf den *UserContext* angewiesen sind müssen die Instanz der Klasse *App* verwenden.

Übung 3

SimpleObjectModel

Folgendes Klassendiagramm zeigt die Hierarchien der Klasse *SimpleObjectModel*, die eingeführt wurde um in Listen, Comboboxen und dergleichen Entitäten für die Darstellung zu halten. Die Controls verwenden Konverter, in denen aus der String Repräsentation wieder in die Entität konvertiert wird. Diese Klasse hält hierbei eine Object Instanz (z.B.: Artist) und den anzuzeigenden Label. In den Konvertern wird eine Instanz *SimpleObjectModel* aus dem zu verwaltenden Objekt erzeugt und visa versa.

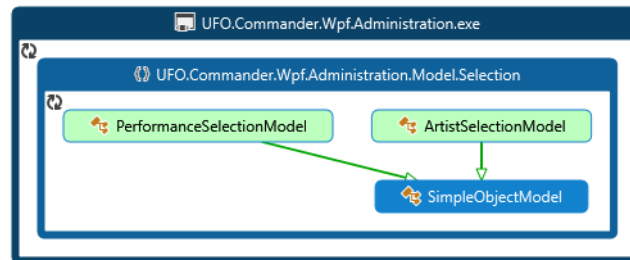


Abbildung 11: Klasse für Controls mit Konverter

Die beiden abgeleiteten Klassen erweitern hierbei die Klasse *SimpleObjectModel* um spezifische Attribute, die in den Controls verwendet werden.

IConverter

Folgendes Klassendiagramm zeigt die Hierarchien der Klasse *IConverter*.

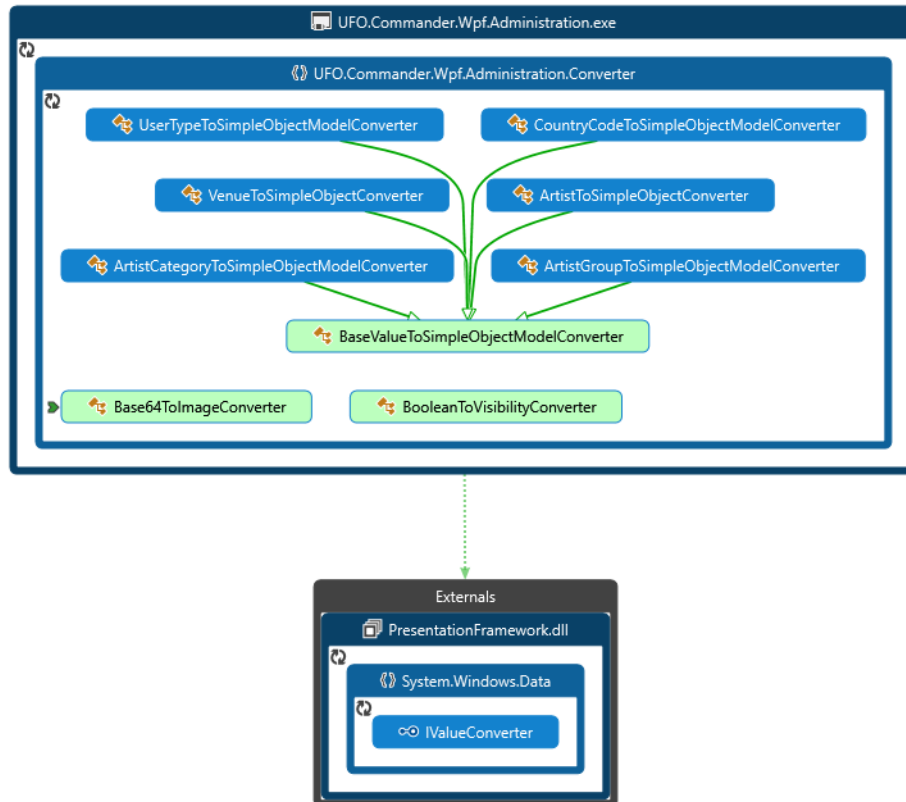


Abbildung 12: Konverter Hierarchie

Die Klasse *BaseValueToSimpleObjectConverter* stellt die Basisklasse aller Konverter dar, die Instanzen von *SimpleObjectModel* konvertieren. In diese Klasse wurden alle gemeinsamen Funktionalitäten wie

1. *Typ-Prüfung*

Es wird geprüft ob der Typ des übergebenen Value dem erwarteten Typ entspricht

2. *ConvertBack*

Da hier nur Instanzen von *SimpleObjectModel* konvertiert werden kann diese Methoden in einer Basisklasse implementiert werden, da hier nur auf den Property *Data* zugegriffen wird.

Es wurden auch Konverter für die Visibility (true=Visibility.VISIBLE, false=Visibility.HIDDEN) und zum dekodieren von Base64 Strings in Image Instanzen eingeführt.

1.2.3 WPF Struktur

Folgende Abbildung zeigt wie die Views innerhalb des Projektes strukturiert wurden.

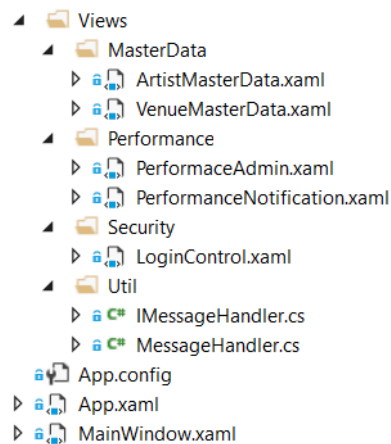


Abbildung 13: Verzeichnisstruktur der Views innerhalb des Projektes

Bis auf die *App.xaml* und *MainView.xaml* wurden alle Views innerhalb eines Verzeichnis names *Views* gebündelt wobei hierbei eine Trennung zwischen den einzelnen Typen der Views durchgeführt wurde.

1. *MasterData*
Alle Views für die Verwaltung der Stammdaten
2. *Performance*
Alle Views für die Verwaltung des Festivalprogramms
3. *Security*
Die Login-View
4. *Util*
Utility Klassen um innerhalb von View-Models mit der View zu interagieren ohne Referenzen auf WPF Namespaces verwenden zu müssen.

Alle diese Views sind als UserControls implementiert worden und werden innerhalb von *MainView.xaml* verwendet (außer *PerformanceNotification.xaml*), die diese UserControls in einem Tab-Control als DataTemplate für die verschiedene Typen der Tab-Models definiert.

Übung 3

Folgende Abbildung zeigt die eingeführte Verzeichnisstruktur um die WPF-Ressourcen innerhalb dieses Projektes zu strukturieren.

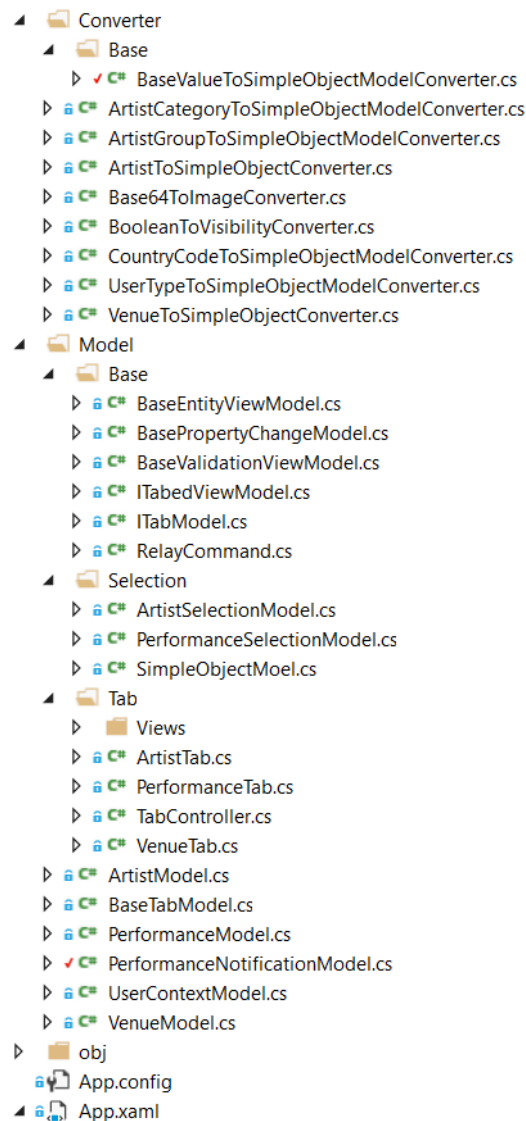


Abbildung 14: Verzeichnisstruktur der WPF-Ressourcen

Die jeweiligen Base-Verzeichnisse beinhalten die eingeführten Basisklassen für den jeweiligen Kontext (z.B.: Converter, Models, ...).

Übung 3

1.2.4 Benutzerdokumentation

Folgend ist die Benutzerdokumentation der WPF-Anwendung *Commander* angeführt.

Login

Die Administration benötigt einen Login bevor mit Ihr interagiert werden kann. Dazu müssen Sie als Benutzer angelegt worden sein.

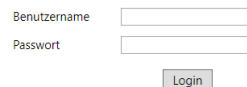


Abbildung 15: Login nach Start der Anwendung

Es wurden keine Beschränkungen beim Login eingeführt, was bedeutet Ihr Zugang wird nach mehrmaligen Fehlversuchen gesperrt.

Mögliche Fehler

Wenn Sie nachdem Start der Anwendung folgende Fehlermeldung erhalten, prüfen Sie bitte ob Sie auf die Datenbank zugreifen können.

1. Fehlende Datenbankverbindung

Sollten sie keine Datenbankverbindung habe so erhalten Sie folgende Fehlermeldung nach dem Sie die Anwendung gestartet haben. In diesem Fall können Sie sich nicht einloggen.

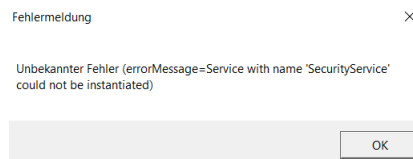


Abbildung 16: Fehlermeldung keine Datenbankverbindung

2. Login fehlgeschlagen

Wenn Sie ein falsches Passwort eingeben erhalten Sie folgende Fehlermeldung

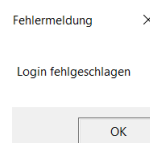


Abbildung 17: Fehlermeldung Login fehlgeschlagen

Artistenverwaltung

Nachdem Sie sich erfolgreich eingeloggt haben starten Sie mit der Ansicht der Artistenverwaltung, wo Sie die Stammdaten der Artisten verwalten können.

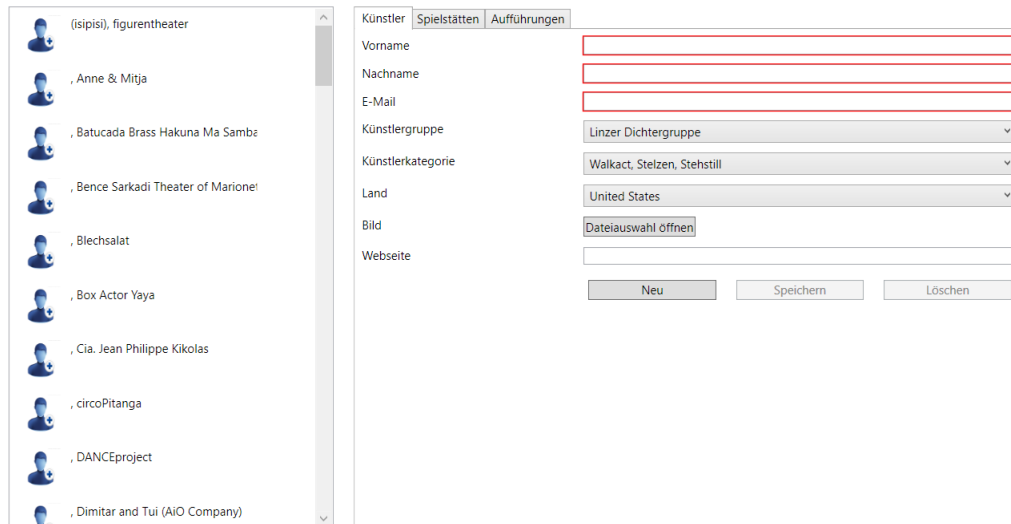


Abbildung 18: Stammdatenverwaltung der Artisten

Alle Ansichten sind nachdem selben Schema aufgebaut. Linker Hand sehen Sie die Auswahl der bereits existierenden Einträge - in diesem Fall die Artisten - die Sie auswählen und rechter Hand - innerhalb des Tabs - bearbeiten können. Die rot markierten Eingabefelder signalisieren, dass hier Validierungsfehler aufgetreten sind. Wenn Sie mit dem Cursor über ein rot markiertes Eingabefeld kommen wird Ihnen ein Tooltip angezeigt, der Ihnen mitteilt welcher Validierungsfehler aufgetreten ist.

Sollten Sie einen Artisten löschen so werden alle in der Zukunft liegenden Aufführungen dieses Artisten ebenfalls gelöscht. Sollten alle Aufführungen gelöscht worden sein, so wird der Artist ebenfalls gelöscht, andererseits wird der Artist als gelöscht markiert. Auf jeden Fall ist der Artist nicht mehr für Sie verfügbar.

Übung 3

Mögliche Fehler

1. *Bild zu groß*

Sollten Sie ein Bild auswählen, welches die maximal erlaubte Größe überschreitet, dann erhalten Sie folgende Fehlermeldung

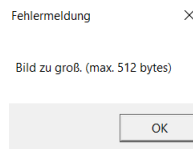


Abbildung 19: Fehlermeldung Bild zu groß

2. *Artist wurde geändert*

Sollten ein anderer Benutzer den Artisten geändert haben so erhalten Sie folgende Fehlermeldung

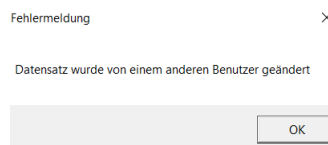


Abbildung 20: Fehlermeldung Artist geändert

3. *Artist nicht mehr vorhanden*

Sollten Sie versuchen einen Artisten zu speichern/löschen und der Artist wurde entweder als gelöscht markiert oder tatsächlich gelöscht, so erhalten Sie folgende Fehlermeldung

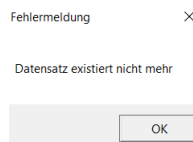


Abbildung 21: Fehlermeldung Artist nicht gefunden

Übung 3

Spielstätten

Auf diesem Tab könne Sie die zur Verfügung stehenden Spielstätten administrieren.

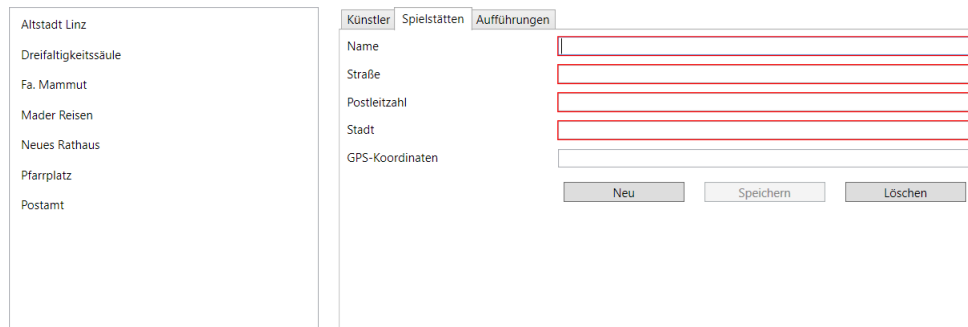


Abbildung 22: Stammdatenverwaltung der Spielstätten

Bitte stellen Sie sicher das die eingegeben GPS-Koordinaten korrekt sind da hier keine Prüfung auf Korrektheit erfolgt. Sollten Sie GPS-Koordinaten angeben, so können diese in der Web-Anwendung für Goolge-Maps verwendet werden. Sollten Sie nur die Adressdaten angeben so kann es passieren dass die Position auf Goggle-Maps nicht korrekt angezeigt wird, falls Google diese Adresse nicht kennt oder falsch zuordnet.

Sollten bereits Aufführungen an dieser Spielstätte stattfinden, so haben Sie nur mehr die Möglichkeit den Name dieser Spielstätte zu ändern. Bei einem Löschen einer in Verwendung befindlichen Spielstätte wird diese lediglich als gelöscht markiert. In jedem Fall sind gelöschte Spielstätten für Sie nicht mehr verfügbar.

Mögliche Fehler

1. *Spielstätte wurde geändert*

Sollte ein anderer Benutzer die Spielstätte geändert haben so erhalten Sie folgende Fehlermeldung

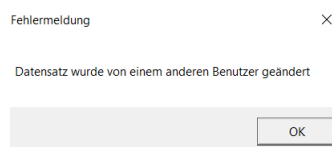


Abbildung 23: Fehlermeldung Spielstätte geändert

2. *Spielstätte nicht mehr vorhanden*

Sollten Sie versuchen eine Spielstätte zu speichern/löschen und die Spielstätte wurde entweder als gelöscht markiert oder tatsächlich gelöscht, so erhalten Sie folgende Fehlermeldung

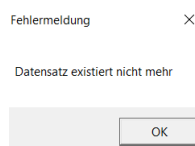


Abbildung 24: Fehlermeldung Spielstätte nicht gefunden

Übung 3

Aufführungen Auf diesem Tab können Sie die Aufführungen verwalten. Diese werden auf die Aufführungstage aufgeteilt und sind in der Auswahl - linker Hand - verfügbar. Hier werden auch Informationen angezeigt wie viele Artisten an wie vielen Spielstätten Aufführungen haben.

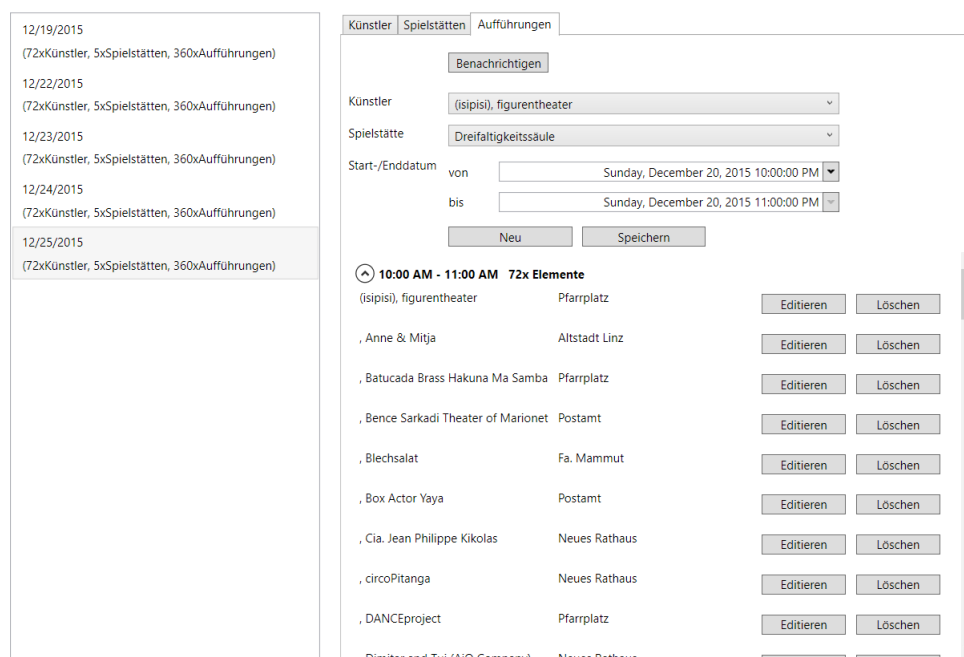


Abbildung 25: Festivalverwaltung

Nach der Auswahl eines Aufführungstages werden unterhalb des Formulars die Aufführungen geordnet nach der Aufführungszeit, Artistenname und Spielstätte angezeigt. Die Buttons neben den Einträgen signalisieren das diese Einträge änderbar und löschtbar sind. Sollten keine Buttons vorhanden sein, so wird diese Aufführung bereits in der Vergangenheit liegen und kann daher weder gelöscht noch geändert werden.

Wenn sie auf Editieren klicken so wird das Formular mit den Aufführungsdaten gesetzt und der Button 'Löschen' wird sichtbar. Ansonsten sind die Formularfelder mit Standardwerten befüllt.

Wenn Sie den Button 'Benachrichtigen' klicken öffnet sich ein Dialog über den Sie die E-Mail angeben können, die an alle Artisten versendet wird.

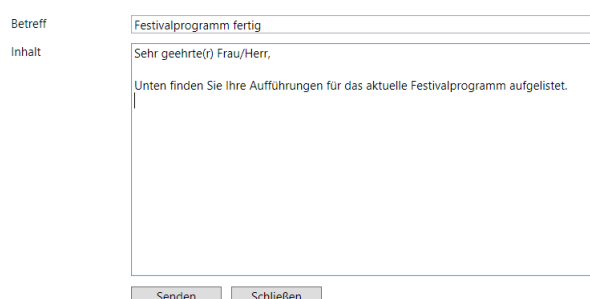


Abbildung 26: Dialog für E-Mail Benachrichtigungen

Diese E-Mail wird an alle Artisten verschickt wobei der E-Mail-Nachricht alle Aufführungen des jeweiligen Artisten angefügt werden.

Übung 3

Wenn die E-Mail-Nachrichten versendet wurden erhalten Sie eine Meldung über die Anzahl der versendeten E-Mail-Nachrichten.

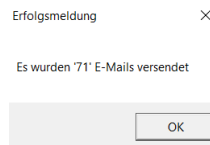


Abbildung 27: Erfolgsmeldung für E-Mail-Versand

Sehen Sie hier ein Beispiel für so eine E-Mail Nachricht.

Sehr geehrte(r) Frau/Herr,

Unten finden Sie Ihre Aufführungen für das aktuelle Festivalprogramm aufgelistet.

Sollten Sie das gesamte Festivalprogramm einsehen wollen, so gehen Sie bitte auf unsere Webseite <http://ufo.at>

12/22/2015:

Altstadt Linz, 08:00 - 09:00

Altstadt Linz, 10:00 - 11:00

Fa. Mammut, 12:00 - 13:00

Postamt, 14:00 - 15:00

Altstadt Linz, 16:00 - 17:00

12/23/2015:

Postamt, 08:00 - 09:00

Altstadt Linz, 10:00 - 11:00

Pfarrplatz, 12:00 - 13:00

Neues Rathaus, 14:00 - 15:00

Neues Rathaus, 16:00 - 17:00

Abbildung 28: Generierte E-Mail

Übung 3

Mögliche Fehler

1. *Aufführung wurde geändert*

Sollte ein anderer Benutzer die Aufführung geändert haben so erhalten Sie folgende Fehlermeldung

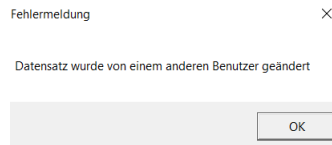


Abbildung 29: Fehlermeldung Spielstätte geändert

2. *Aufführung nicht mehr vorhanden*

Sollten Sie versuchen eine Aufführung zu speichern/löschen und die Aufführung wurde bereits gelöscht so erhalten Sie folgende Fehlermeldung

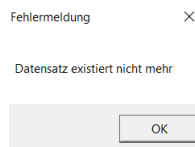


Abbildung 30: Fehlermeldung Spielstätte nicht gefunden

3. *Artist überbucht*

Sollten Sie versuchen eine Aufführung anzulegen/speichern und der Artist hat bereits eine Aufführung zu diesem Zeitpunkt oder die vordefinierte Pause von 1 Stunde vor und nach einer Aufführung wird unterschritten so erhalten Sie folgende Fehlermeldung

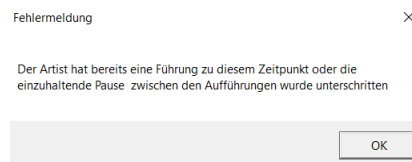


Abbildung 31: Artist überbucht

1.3 Ausbaustufe 3

Folgender Teil dieser Dokumentation dokumentiert den dritten Teil der Ausbaustufe. In diesem Teil der Ausbaustufe musste die Web-Anwendung für *UFO* implementiert werden. Dazu wurden folgende Technologien und Frameworks verwendet:

1. *Myfaces*
Java-Server-Faces Implementierung von Apache
2. *Weld*
Context and Dependency Injection Implementierung von Apache
3. *Deltaspike*
Open-Source CDI-Extension Library, die nützliche Features für eine CDI-Anwendung zur Verfügung stellt.
4. *Google-Maps-Services*
Google Java-API für Geocoding.
5. *Jackson-Json*
Open-Source Framework für JSON Mappings
6. *log4j*
Apache Logging Framework
7. *Apache-Commons*
Eine Sammlung von common Libraries von Apache wie z.B.: common-collections.
8. *Primefaces*
Bekannte Open-Source JSF-Komponentenlibrary

In dieser Anwendung wurde *CDI* verwendet, damit Abhängigkeiten und Ressourcen über Injizierung den Beans zur Verfügung gestellt werden konnten. Diese Abhängigkeiten können vom CDI-Container kontextabhängig (verschiedene Scopes) werden. Daher wurde entschlossen CDI in die Anwendung zu integrieren. Dies stellte sich nicht als schwierig heraus, da neben den notwendigen Abhängigkeiten nur geringfügige Konfiguration in der *web.xml* von Nöten war.

Deltaspike ist eine Erweiterung für CDI und stellt nützliche Features zur Verfügung. Es wurde hierbei das Feature *Type-Safe-Messages* verwendet, dass erlaubt über ein Interface ein *MessageBundle* zu definieren, welches auch über CDI und *EL* verfügbar ist. Hierbei wird über eine CDI-Extension beim Start des Containers dynamisch ein Bean an den Container gebunden, wobei keine explizite Implementierung dieses Beans von Nöten ist.

Google-Maps-Services wurde dazu verwendet um die Koordinaten der Spielstätten aufzulösen, die keine Koordinaten aber Adressen zur Verfügung stellen. Dadurch können auch diese Spielstätten auf der verwendeten Google-Map angezeigt werden. Die Genauigkeit hängt aber hierbei von den zur Verfügung gestellten Adressdaten ab und natürlich von dem gelieferten Resultat des Geocoding.

Übung 3

1.3.1 Konfiguration

Die Konfiguration erfolgte hauptsächlich in der *web.xml*. Es wurde vollständig darauf verzichtet die Navigation über die *faces-config.xml* zu implementieren, da einerseits diese Art der Konfiguration mir zu komplex und unflexibel erscheint und andererseits die Anwendung als Single-Page-Anwendung implementiert wurde und daher keine Navigation erforderlich ist.

In der *web.xml* wurden alle Konfiguration für die SOAP-Services vorgenommen wie die URLs, die verwendete Zeitzone, den verwendeten Namespace und die Authentifizierungsdaten der Web-Anwendung für die SOAP-Zugriffe. Da es Unterschiede bei der Interpretation der Datumsinformationen in *.NET* (UTC) und *Java* (GMT) gab, musste die Zeitzone konfigurierbar gemacht werden, damit bei der Überführung von den SOAP-Models in die View-Models die Datumsinformationen mit der richtigen Zeitzone interpretiert werden konnten. Da die Adressen der Services sich ändern könnten wurden diese Adressen ebenfalls in der *web.xml* für jeden Service einzeln konfigurierbar gemacht.

Listing 1: web.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4      xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5      ↪ http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
6      id="WebApp_ID" version="3.1">
7      <!-- Session timeout config -->
8      <session-config>
9          <session-timeout>30</session-timeout>
10     </session-config>
11     <!-- Bind CDI BeanManager to CDI -->
12     <resource-env-ref>
13         <resource-env-ref-name>BeanManager</resource-env-ref-name>
14         <resource-env-ref-type>javax.enterprise.inject.spi.BeanManager</resource-env-ref-type>
15     </resource-env-ref>
16     <!-- Default locale used by app -->
17     <context-param>
18         <param-name>Locale</param-name>
19         <param-value>de_DE</param-value>
20     </context-param>
21     <!-- Remote Service configuration -->
22     <context-param>
23         <param-name>service.timezone</param-name>
24         <param-value>UTC</param-value>
25     </context-param>
26     <context-param>
27         <param-name>ufo.webservice.username</param-name>
28         <param-value>WebApplication_1</param-value>
29     </context-param>
30     <context-param>
31         <param-name>ufo.webservice.password</param-name>
32         <param-value>j678Zl1HOB1z~/ .9eoZs@8gg,0cl-Bqi</param-value>
33     </context-param>
34     <context-param>
35         <param-name>ufo.webservice.namespace</param-name>
36         <param-value>https://webservice.ufo.swk.ooe.fh.at/</param-value>
37     </context-param>
38     <context-param>
39         <param-name>ufo.webservice.artistService</param-name>
40         <param-value>https://localhost:44300/Soap/ArtistService.asmx</param-value>
41     </context-param>
42     <context-param>
43         <param-name>ufo.webservice.performanceService</param-name>

```

Übung 3

```
44     <param-value>https://localhost:44300/Soap/PerformanceService.asmx</param-value>
45 </context-param>
46 <context-param>
47     <param-name>ufo.webservice.venueService</param-name>
48     <param-value>https://localhost:44300/Soap/VenueService.asmx</param-value>
49 </context-param>
50 <context-param>
51     <param-name>ufo.webservice.securityService</param-name>
52     <param-value>https://localhost:44300/Soap/SecurityService.asmx</param-value>
53 </context-param>
54 <context-param>
55     <param-name>google.map.api.key</param-name>
56     <param-value>AIzaSyBPniHSAOZLz2g1MuQ06N2d4S11GA6r4bA</param-value>
57 </context-param>
58 <!-- Myfaces JSf configuration -->
59 <context-param>
60     <param-name>org.apache.myfaces.PRETTY_HTML</param-name>
61     <param-value>true</param-value>
62 </context-param>
63 <context-param>
64     <param-name>org.apache.myfaces.USE_ENCRYPTION</param-name>
65     <param-value>true</param-value>
66 </context-param>
67 <context-param>
68     <param-name>org.apache.myfaces.CACHE_EL_EXPRESSIONS</param-name>
69     <param-value>alwaysRecompile</param-value>
70 </context-param>
71 <!-- Standard JSf configuration -->
72 <context-param>
73     <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
74     <param-value>server</param-value>
75 </context-param>
76 <context-param>
77     <param-name>javax.faces.PARTIAL_STATE_SAVING</param-name>
78     <param-value>true</param-value>
79 </context-param>
80 <context-param>
81     <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
82     <param-value>xhtml</param-value>
83 </context-param>
84 <context-param>
85     <param-name>javax.faces.PROJECT_STAGE</param-name>
86     <param-value>Development</param-value>
87 </context-param>
88 <context-param>
89     <param-name>javax.faces.FACELETS_REFRESH_PERIOD</param-name>
90     <param-value>1</param-value>
91 </context-param>
92 <!-- Primefaces configuration -->
93 <context-param>
94     <param-name>primefaces.THEME</param-name>
95     <param-value>bootstrap</param-value>
96 </context-param>
97 <context-param>
98     <param-name>primefaces.FONT_AWESOME</param-name>
99     <param-value>true</param-value>
100 </context-param>
101 <!-- Log4j web module context listener -->
102 <listener>
103     <listener-class>org.apache.logging.log4j.web.Log4jServletContextListener</listener-class>
104 </listener>
105 <!-- FacesServlet for *.xhtml files enabled -->
106 <servlet>
```

Übung 3

```

107     <servlet-name>Faces Servlet</servlet-name>
108     <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
109     <load-on-startup>1</load-on-startup>
110 </servlet>
111 <servlet-mapping>
112     <servlet-name>Faces Servlet</servlet-name>
113     <url-pattern>*.xhtml</url-pattern>
114 </servlet-mapping>
115 <!-- Log4j web module filter configuration -->
116 <filter>
117     <filter-name>log4jServletFilter</filter-name>
118     <filter-class>org.apache.logging.log4j.web.Log4jServletFilter</filter-class>
119 </filter>
120 <filter-mapping>
121     <filter-name>log4jServletFilter</filter-name>
122     <url-pattern>*</url-pattern>
123     <dispatcher>REQUEST</dispatcher>
124     <dispatcher>FORWARD</dispatcher>
125     <dispatcher>INCLUDE</dispatcher>
126     <dispatcher>ERROR</dispatcher>
127 </filter-mapping>
128 <!-- Welcome file list -->
129 <welcome-file-list>
130     <welcome-file>index.xhtml</welcome-file>
131 </welcome-file-list>
132
133 <!-- Allow SSL only -->
134 <security-constraint>
135     <web-resource-collection>
136         <web-resource-name>Protected Context</web-resource-name>
137         <url-pattern>*</url-pattern>
138     </web-resource-collection>
139     <!-- auth-constraint goes here if you require authentication -->
140     <user-data-constraint>
141         <transport-guarantee>CONFIDENTIAL</transport-guarantee>
142     </user-data-constraint>
143 </security-constraint>
144 </web-app>

```

Übung 3

Es wurde eine eigene *JsExceptionHandlerFactory* und eine eigener *JsExceptionHandler* implementiert sodass die Exceptions nicht mehr an den Client gesendet werden und diese ebenfalls wie gewünscht behandelt werden können. Diese *JsExceptionHandlerFactory*, die die *JsExceptionHandler* Instanz erstellt wurde in der *faces-config.xml* registriert.

Listing 2: faces-config.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <faces-config xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5     ↪ http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
6   version="2.2">
7   <application>
8     <!-- Deltaspike needs this here otherwise resource bundle is not discoverable -->
9     <message-bundle>at.fh.ooe.swk.ufo.web.application.message.MessageBundle</message-bundle>
10  </application>
11  <factory>
12    <!-- Custom exception handler factory which provides our custom exception
13         handler -->
14    <exception-handler-factory>
15      at.fh.ooe.swk.ufo.web.application.exception.GlobalExceptionHandlerFactory
16    </exception-handler-factory>
17  </factory>
18 </faces-config>
```

Zusätzlich zu diesen Konfigurationen muss beim Start des Tomcat-Servers JVM-Argumente hinzugefügt werden, damit der Zugriff auf die SOAP-Services über SSL funktioniert.

```
javax.net.ssl.trustStore
javax.net.ssl.trustStorePassword
```

Siehe hierzu die Datei *tomcat-config.txt* im Java-Web-Project Root-Verzeichnis. Der Truststore muss natürlich das aktuell von IIS verwendete Zertifikat importieren.

1.3.2 Web-Service-Proxy-Architektur

Folgende Architektur wurde implementiert um die SOAP-Service Instanzen von der View-Logik zu abstrahieren.

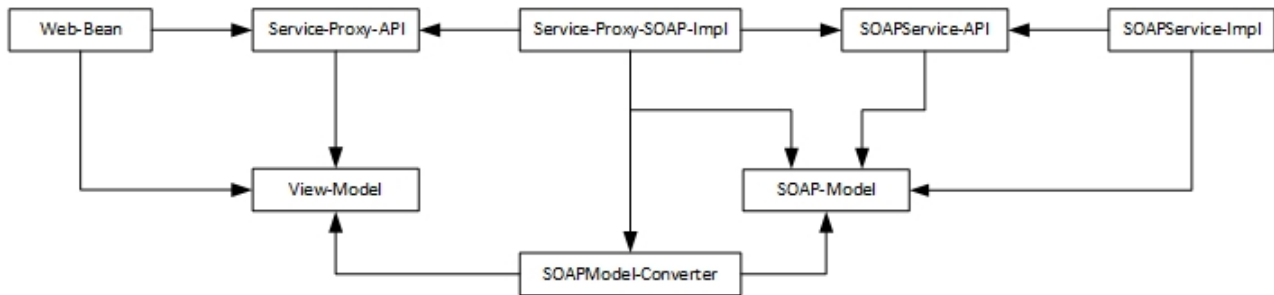


Abbildung 32: Service-Proxy-Architecture

Durch diese Architektur wird gewährleistet, dass die Remote-Service Abhängigkeiten vollständig von den View-Beans abstrahiert sind und diese ausgetauscht werden können ohne diese Beans modifizieren zu müssen. Es wurden Konverter eingeführt, die die Remote-Service-Models in die View-Models überführen und nötige Aufbereitungen und Konvertierungen vornehmen.

Die Service-Proxy Instanzen werden über CDI injiziert und wurden als `javax.enterprise.context.ApplicationScoped` deklariert, was zur Folge hat das diese Instanzen nur einmalig innerhalb der Anwendungslebenszeit erstellt werden. Dies ist als gleichwertig mit einer Singleton-Instanz anzusehen. Mit den Konvertern wurde gleich verfahren.

Im Gegensatz zu den Service-Proxy werden die SOAP-Service-Instanzen über einen Producer erstellt, da hier noch Konfigurationen von Nöten sind und daher die Instanzen nicht von CDI selbst erstellt werden können. Dank *Deltaspike* ist man in der Lage sich den *ServletContext* in CDI-Beans injizieren zu lassen, wobei hier der herkömmliche Ansatz der Konfiguration umgedreht wird. Also nicht ein *ServletContextListener* liest die Init-Parameter aus und konfiguriert ein Bean sondern der Producer liest die Init-Parameter aus und konfiguriert die zu erstellenden Instanzen. Dies ist ein guter Ansatz da für den Client die Factory im Verborgenen bleibt was der Grundsätzliche Gedanke von *IOC* (Inversion-Of-Control) ist.

Die *SOAP-Service* Implementierungen wurden mit dem Standardtool von Eclipse generiert (Apache-Axis), was zwar eine relative alte Implementierung darstellt aber ausreichend für den Anwendungszweck war.

Folgende Paketstruktur wurde für die Service-Proxy Ressourcen eingeführt:

1. `at.fh.ooe.swk.ufo.service.api`
Enthält die Spezifikation für die Service-Proxy Ressourcen.
2. `at.fh.ooe.swk.ufo.service.impl`
Enthält die Implementierung der Service-Proxy-Spezifikation

1.3.3 Web-Application-Resources

Da es über die gesamte Anwendung hinweg gemeinsam genutzte Ressourcen gibt wurden diese in einem Paket *at.fh.ooe.swk.ufo.web.application.** zusammengefasst. Hier sind Ressourcen wie folgt gebündelt:

1. *at.fh.ooe.swk.ufo.web.application.bean*
Enthält die Beans für den Login, Repräsentation eines eingeloggten BenutzerIn sowie das *LanguageBean* für die Verwaltung der verwendeten Sprache.
2. *at.fh.ooe.swk.ufo.web.application.converter*
Enthält die Konverter, die über die gesamte Anwendung hinweg genutzt werden können und keine Referenzen auf spezielle Models beinhalten. Z.B.: Konverter für *SelectItem* Instanzen, die als Value Object, Enum Instanzen halten.
3. *at.fh.ooe.swk.ufo.web.application.model*
Nachdem jedes View-Model eine Entität darstellt und diese auch über eine Id verfügen wurde ein Interface namens *IdHolder<T>* eingeführt, sowie eine abstrakte Klasse *AbstractIdHolderModel<T>* **implements** *IdHolder<T>*, die dieses Interface und die Methoden *hash*, *equals* sowie Getter und Setter für die Id implementiert.
4. *at.fh.ooe.swk.ufo.web.application.constants*
Alle definierten Kontextparameter wurden als Enumeration abgebildet. Somit sind diese Parameter nur an zwei Stellen definiert. Einmal in der *web.xml* und ein zweites Mal in der Enumeration.
5. *at.fh.ooe.swk.ufo.web.application.exceptions*
Enthält die implementierte *JsExceptionHandlerFactory* und den implementierten *JsExceptionHandler* für das Behandeln unbehandelter Exceptions. Dadurch wird das Rendern der Exceptions zum Client unterdrückt.
6. *at.fh.ooe.swk.ufo.web.application.listener*
Enthält den *ServletContextListener*, der die *Default Locale* der VM auf die definierte *Locale* in der *web.xml* setzt
7. *at.fh.ooe.swk.ufo..web.application.message*
Enthält die Deltaspike *MessageBundle* Interfaces und dazugehörigen *RessourceBundles*. Dadurch wird Multilingualität von der Anwendung unterstützt.
8. *at.fh.ooe.swk.ufo.web.application.producer*
Enthält die Producer, die gemeinsam genutzte Ressourcen kontextabhängig produzieren.

Es wurde ein JSF-Template implementiert, welches das Layout und die gemeinsam genutzten View-Ressourcen definiert wie: Loading-Dialog und Message-Komponenten. Diese Ressourcen wurden in das Verzeichnis *WEB-INF/templates* gelegt, da diese nicht von außen verfügbar sein sollen und alle Ressourcen in *WEB-INF* implizit von einem Zugriff geschützt sind.

1.3.4 Performances-Resources

Die gesamten Ressourcen der Performance Ansichten wurden in einem Paket mit folgender Struktur gebündelt.

1. *at.fh.ooe.swk.ufo.web.performance.constants*
Enthält die Konstanten für die bereitgestellten Suchoptionen, die in Form von Enumerationen abgebildet wurden.
2. *at.fh.ooe.swk.ufo.web.performance.model*
Enthält alle *View-Models* sowie *Edit-View-Models*, die in den Performance Ansichten verwendet werden.
3. *at.fh.ooe.swk.ufo.web.performance.page*
Enthält alle *Backing-Beans* für die einzelnen Ansichten, die deren Verhalten steuern bzw, die zur Verfügung gestellten Aktionen bereitstellen. Somit ist *Interaktionslogik* von der *Präsentationslogik* vollständig getrennt.
4. *at.fh.ooe.swk.ufo.web.performance.support*
Enthält das *Support-Bean*, in dem die gesamte Ladelogik implementiert ist, sowie die Producer Methoden, die die Performance spezifischen Ressourcen bereitstellt. Dadurch werden die *Backing-Beans* von der Ladelogik befreit.

Die einzelnen Teile der Ansichten wurden in eigenen **.xhtml* Dateien implementiert, die im den Verzeichnis *WEB-INF/page* gehalten werden. Lediglich die *index.xhtml*, die die Komposition der Anwendung darstellt, wurde im Root-Verzeichnis platziert und somit die einzige Möglichkeit des Zugriffs auf die Anwendung. Wiederverwendbare Teile wurden wiederum als Kompositionen definiert wie z.B.: die Tooltip Informationen, die an mehreren Stellen verwendet werden.

1.3.5 Multilingualität

Die gesamte Anwendung unterstützt Multilingualität, obwohl anzumerken ist, dass nur eine Sprache implementiert wurde. Nicht desto trotz können andere Sprachen implementiert werden. Es wurde hierbei der Mechanismus *Type-Safe-Messages* von *Deltaspike* verwendet, welche eine CDI Extension darstellt. Im folgenden ist ein Ausschnitt des Interfaces `MessagesBundle.java` angeführt.

```
@MessageBundle
@Named("msg")
public interface MessagesBundle extends Serializable {

    @MessageTemplate("{UFO_PROGRAM}")
    String getUfoProgram();
}
```

Listing 3: MessageBundle.java

Folgende Auflistung beschreibt die verwendeten Annotationen im Interface, die dazu verwendet werden um das MessageBundle zu konfigurieren.

1. `@MessageBundle`
Annotation, die der Deltaspike-CDI-Extension mitteilt, dass es sich hierbei um eine Message-Bundle handelt.
2. `@ApplicationScoped`
CDI-Scope-Annotation, die Deltaspike mitteilt eine Bean zu erstellen, welches nur einmalig über die Anwendungslebenszeit existiert.
3. `@Named("msg")`
CDI-Annotation, die dieses Bean über *Expression-Language* zur Verfügung stellt. Wird von der Deltaspike-CDI-Extension bei der Bean-Erstellung berücksichtigt.
4. `@MessageTemplate`
Deltaspike spezifische Annotation, die den Key in der ResourceBundle Datei darstellt.

Die enthaltenen Getter-Methoden müssen der Java-Bean-Convention folgen damit diese in EL-Expressions verwendet werden können. Um andere Sprachen zu unterstützen müssen jetzt lediglich nur noch die ResourceBundle Dateien für die einzelnen Sprachen angelegt werden. Des weiteren sollte vielleicht ein eigener `org.apache.deltaspike.core.api.message.LocaleResolver` implementiert werden, der die Locale vielleicht aus dem FacesContext auflöst. Es wird in dem oben angeführten Beispiel der bereits mitgelieferte `org.apache.deltaspike.core.impl.message.DefaultLocaleResolver` verwendet, der die VM-Default-Locale verwendet.

Folgendes Beispiel zeigt die Verwendung des Interfaces `MessagesBundle.java` in `*.xhtml`.

```
<h:outputText value="#{msg.firstName}" style="font-weight: bold;" />
```

Listing 4: MessageBundle Verwendung in *.xhtml Dateien

1.3.6 Security

Die SOAP-Webservices sind über SSL verfügbar, daher musste dem Tomcat-Server eine Truststore-Datei zur Verfügung gestellt werden, die das trusted Zertifikat des SOAP-Servers enthält. Es wurde hierbei in VisualStudio eingestellt, dass die SOAP-Services SSL unterstützen. Sie sind auch noch über HTTP verfügbar, dies sollte aber bei einem Release unterbunden werden. Ebenfalls ist die Web-Anwendung nur über SSL erreichbar, was durch eine Konfiguration in der Tomcat *server.xml* und in der *web.xml* erreicht wurde.

```
<Connector SSLEnabled="true"
    clientAuth="false"
    keyAlias="ufo"
    keystoreFile="conf/keystore.jks"
    keystorePass="ufo"
    keystoreType="jks"
    maxThreads="150"
    port="8443"
    protocol="org.apache.coyote.http11.Http11NioProtocol"
    scheme="https"
    secure="true"
    sslProtocol="TLS" />
```

Listing 5: server.xml

```
<!-- Allow SSL only -->
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Protected Context</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <!-- auth-constraint goes here if you require authentication -->
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Listing 6: web.xml

Des weiteren sind die SOAP-Services nicht offen verfügbar sondern die einzelnen Anwendung müssen registriert sein, ansonsten liefert der Webservice kein Daten zurück. Man kann darüber streiten ob die Daten offen Verfügbar sein sollen, aber in dieser Anwendung sind sie es nicht. Die Zugangsdaten für die Web-Anwendung sind in der *web.xml* definiert und auf der .NET Seite in der *web.config*, wobei hier angemerkt sei, dass dieser Ansatz nur für dieses Beispiel Anwendung finden sollte. Mann sollte hierbei in der Lage sein die Zugangsdaten der Anwendung dynamisch zum Beispiel aus einer Datenbank zu ziehen und sie nicht statisch in einer Konfigurationsdatei halten. Bei einem Deployment einer Web-Anwendung könnte für diese die Zugangsdaten erstellt und in einer Datenbank gespeichert werden.

```
@ApplicationScoped
public class SoapWebServiceProducer implements Serializable {

    @Inject
    @DeltaSpike
    private ServletContext servletContext;

    private String username;
    private String password;

    @PostConstruct
    public void postConstruct() {
        // Get authentication data from web.xml
        username = servletContext.getInitParameter(
            ContextParameter.WEBSERVICE_USERNAME.key);
        password = servletContext.getInitParameter(
            ContextParameter.WEBSERVICE_PASSWORD.key);
        ...
    }

    ...

    private SOAPHeaderElement createSoapHeader() throws Exception {
        SOAPHeaderElement authentication = new SOAPHeaderElement(
            xmlNamespace,
            "Credentials");
        SOAPElement node = authentication.addChildElement("Username");
        node.addTextNode(username);
        SOAPElement node2 = authentication.addChildElement("Password");
        node2.addTextNode(password);

        return authentication;
    }
}
```

Listing 7: Auszug aus SoapWebServiceProducer.java

Die Authentifizierungsdaten für die Web-Anwendung werden hierbei über einen SOAP-Header an die Soap-Services übermittelt, die diesen Header auswerten und überprüfen ob es sich um eine registrierten Anwendung handelt.

Listing 8: BaseSecureWebservice.cs

```

1  using System;
2  using UFO.Server.Webservice.Soap.Model.Base;
3  using UFO.Server.Webservice.Soap.Soap.Authentication;
4
5  namespace UFO.Server.Webservice.Soap.Soap.Base
6  {
7      public abstract class BaseSecureWebservice
8      {
9
10         public Credentials credentials;
11
12         /// <summary>
13         /// Handles the authentication.
14         /// </summary>
15         /// <returns>the model which holds the error information, or null if authentication
16         ↪ succeeded</returns>
17
18         protected T HandleAuthentication<T>() where T : BaseModel, new()
19         {
20             if ((credentials == null) || (!credentials.IsValid()))
21             {
22                 Console.WriteLine("GetDetails: " + ((credentials == null) ? "no authentication
23                 ↪ data given" : $"Authentication failed for user '{credentials.Username}'"));
24                 return new T
25                 {
26                     ErrorCode = (int)((credentials == null) ? ErrorCode.AUTHENTICATION_MISSING :
27                     ↪ ErrorCode.AUTHENTICATION_FAILED),
28                     Error = ((credentials == null) ? "No credentials were given" : "Authentication
29                     ↪ failed");
30                 };
31             }
32             return null;
33         }
34     }
35 }

```

```
[WebMethod]
[ScriptMethod(UseHttpGet = false)]
[SoapHeader("credentials")]
public ListResultModel<PerformanceModel> GetPerformances(
    PerformanceFilterRequest filter)
{
    IPerformanceDao performanceDao = DaoFactory.CreatePerformanceDao();
    ListResultModel<PerformanceModel> model = null;
    if ((model = HandleAuthentication<ListResultModel<PerformanceModel>>())
        == null)
    {
        ...
    }
    ...
}
```

Listing 9: Auszug auf PerformanceWebservice.cs

`BaseSecureWebservice.cs` beinhaltet den Property, der das SOAP-Header-Model darstellt, sowie die Authentifizierungsmethode, die auch das Resultat im Falle einer fehlgeschlagenen Authentifizierung aufbereitet. Der konkrete Webservice erbt von dieser Klasse und stellt die Daten nur im Falle einer gültigen Authentifizierung zur Verfügung.

Der SOAP-Header wird hierbei vom ASP.NET Framework über das Attribut `[SoapHeader("credentials")]` in den korrespondierenden Property mit dem definierten Namen injiziert.

1.3.7 SOAP-Services

Die ASP.NET Soap-Services resultieren immer eine Instanz von `BaseModel.cs`, von der wiederum zwei Klassen ableiten `SingleResultModel.cs` und `ListResultModel.cs`, die jeweils ein Single oder List Resultat halten. Die Basisklasse hält etwaige Informationen über einen Fehler, der in der Web-Anwendung ausgewertet werden könnte. Dadurch sind die Anwendungen, die die Daten abrufen, in der Lage etwaige Fehler korrekt auszuwerten und darauf zu reagieren, da die Fehlerinformationen mit übermittelt werden, wobei aber der Stacktrace ausgeschlossen wurde und nur die Message der Exception übermittelt wird.

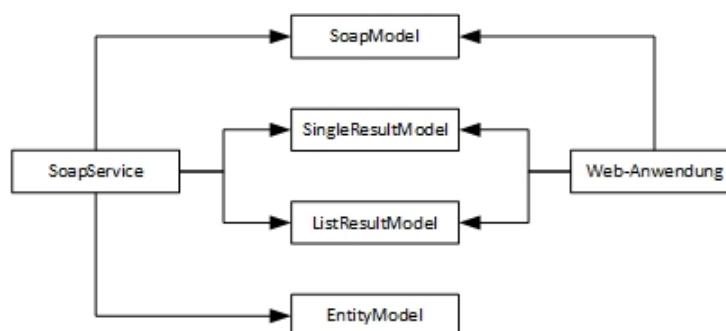


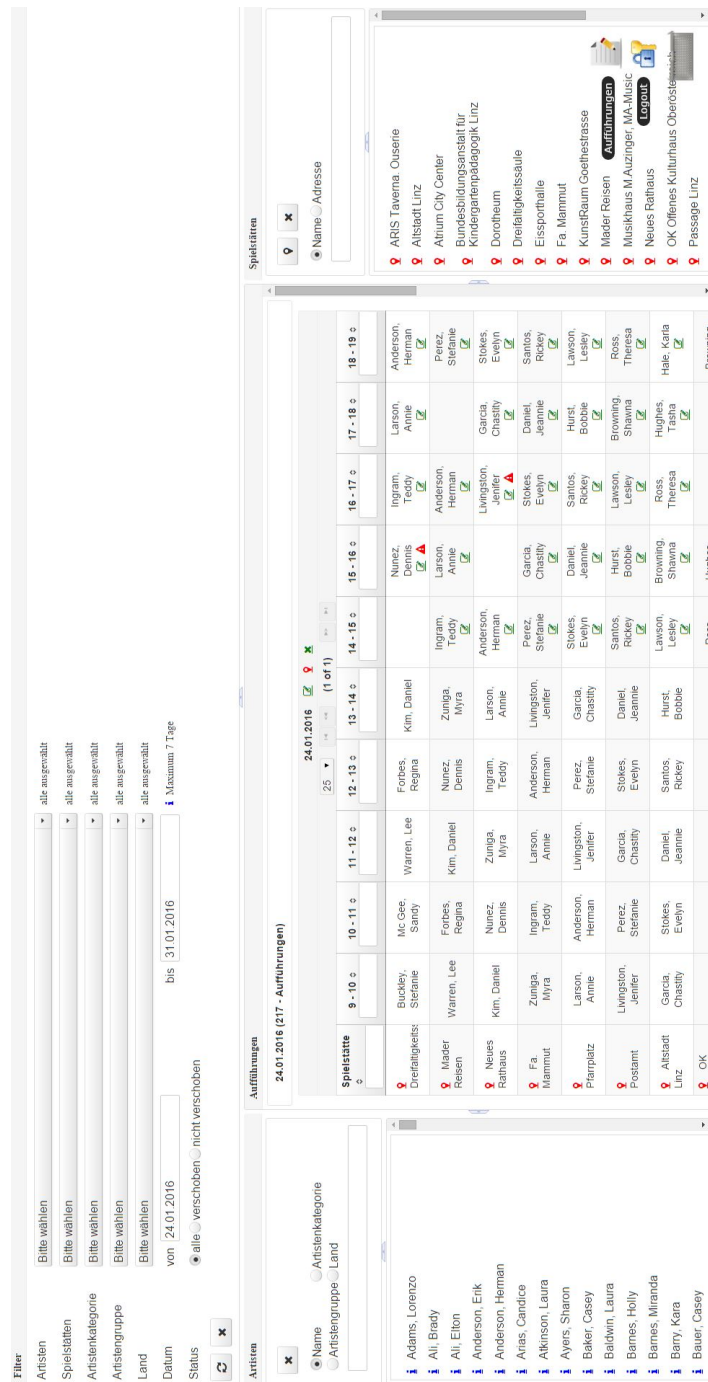
Abbildung 33: SOAP-Model-Referenzen

Wie in der Grafik ersichtlich gibt es keine Referenzen zwischen den Entity-Models und der Web-Anwendung. Es werden alle DAO-Resultate in die entsprechenden SOAP-Models überführt. Somit ist die SOAP-Zugriffsschicht entkoppelt von der Datenbankzugriffsschicht und Änderungen am Datenmodell wirken sich nicht direkt auf die SOAP-Zugriffsschicht aus. Lediglich das Mapping zwischen den Models muss angepasst werden.

Es wird sich auf die implizite Konvertierung, die von ASP.NET bereitgestellt wird verlassen, jedoch wurden auch einige XML-Attribute verwendet, wenn z.B.:SOAP-Model-Properties als nullable markiert bzw. dessen XML-Repräsentation definiert werden musste.

1.3.8 Web-View

Im folgenden wird der Aufbau der Web-View beschrieben. Die Web-View besteht zwar aus mehreren Templates ist jedoch als Single-Page implementiert. Die einzelnen Teile der Seite sind über ein Layout von einander getrennt. Diese Teile lassen sich ein und ausklappen.



The screenshot displays a web application for managing performances. It features a sidebar with filters for Artists, Venues, and Categories, each with a 'select all' option. A date range selector is set to '24.01.2016' to '31.01.2016'. The main area shows a calendar view for the selected date, with a table of performances. The table has columns for date, venue, and artist. The artists listed include Adams, Lorenz, Ali, Brady, Ali, Elton, Anderson, Erik, Anderson, Herman, Atkinson, Laura, Arias, Candice, Ayers, Sharon, Baker, Casey, Baldwin, Laura, Barnes, Holly, Barnes, Miranda, Barry, Kara, and Bauer, Casey.

Abbildung 34: Vollständige View mit allen Layout-Parts aufgeklappt

Übung 3

Natürlich ist nicht ratsam alle Teile auf zu klappen sondern nur die Teile, die die Informationen bereitstellen, die man benötigt. Lediglich der Center-Content lässt sich nicht weg klappen. Dieser enthält die Tabelle mit allen Aufführungen. Im folgenden werden die einzelnen Teile der Webseite beschrieben.

1.3.8.1 Filter

Die folgende Abbildung zeigt die globalen Filter, die die Aufführungsdaten global filtern. Die Daten werden hierbei vom SOAP-Service mit den gesetzten Filtern erneut geladen.



The screenshot shows a web interface for filtering performances. The 'Filter' section includes dropdown menus for 'Artisten', 'Spielstätten', 'Artistenkategorie', 'Artistengruppe', and 'Land', each with a count of selected items. The 'Datum' section has date pickers for 'von' and 'bis' with a 'Maximum 7 Tage' warning. The 'Status' section has radio buttons for 'alle', 'verschoben', and 'nicht verschoben'. Below the filters is a table titled 'Aufführungen' with two rows: '26.01.2016 (2 - Aufführungen)' and '28.01.2016 (1 - Aufführungen)'.

Abbildung 35: Globale Filter

Es stehen hierbei mehrere Filteroptionen zur Verfügung, die alle auf die zu ladenden Daten angewendet werden. Rechts von den Filtern ist die Anzahl der gewählten Optionen angeführt, wobei angemerkt sei, dass wenn keine Optionen ausgewählt wurde es denselben Effekt hat, als wären alle Einträge ausgewählt worden. Der Datumsbereich wurde auf sieben Tage beschränkt. Das Spektakel finden zwar nur an einem Tag statt, jedoch steht es den Entwicklern offen es auch gegebenenfalls für mehrere Tage konfigurieren. Die AnwenderInnen sollten jedoch trotzdem in der Auswahl der Tage eingeschränkt sein, da es einerseits die Performance negativ beeinflussen würde und andererseits es der Übersicht nicht dienlich ist, sich alle tage in einem Jahr anzeigen zu lassen.

Übung 3

1.3.8.2 Artisten

Dieser Teil der Anwendung zeigt alle verfügbaren Artisten an, die aktiv sind. Es wird hierbei keine Vorfilterung vorgenommen. Auch hier stehen mehrere Option für die Filterung zur Verfügung.

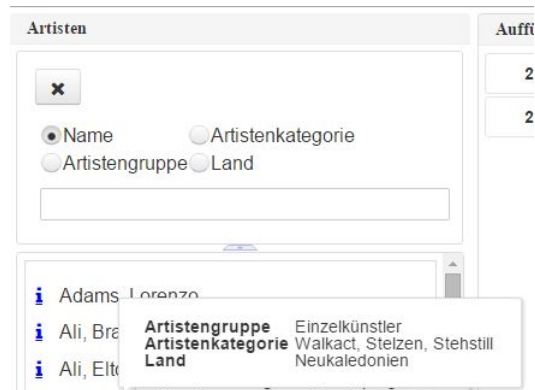


Abbildung 36: Übersicht Artisten

Die Filterung kann entweder nach dem Namen, Artistengruppe, Artistenkategorie und des Landes des Artisten erfolgen. Über einen Tooltip werden die wichtigsten Details des Artisten angezeigt. Über den Info-Icon kann ein Dialog geöffnet werden, der alle Details eines Artisten anzeigt. Der Button setzt den gesetzten Filter auf die Standardeinstellungen zurück.

1.3.8.3 Übersicht Spielstätten

Dieser Teil der Anwendung zeigt alle verfügbaren Spielstätten an. Es wird hierbei keine Vorfilterung vorgenommen. Auch hier stehen mehrere Option für die Filterung zur Verfügung.

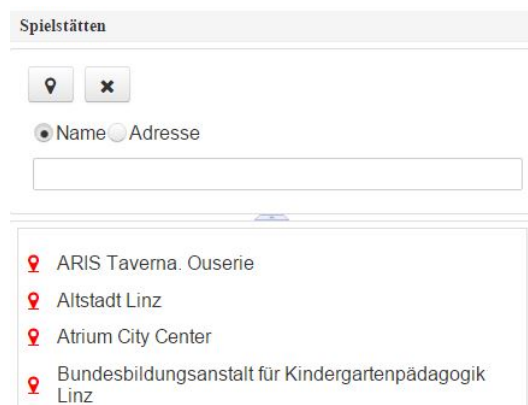


Abbildung 37: Übersicht Spielstätten

Es kann hier nachdem Namen und der Adresse gefiltert werden. Über einen Tooltip wird die Adresse der Spielstätte angezeigt. Über den Marker-Icon oder den Button mit den Marker-Icon kann der Dialog mit den Spielstättendetails geöffnet werden, wobei über den Marker-Icon nur die dazugehörige Spielstätten Details angezeigt werden und beim Button alle aktuell sichtbaren Spielstätten.

Übung 3

1.3.8.4 Übersicht Aufführungen

Der Hauptteil der Seite sind die Aufführungen, die gruppiert nach deren Aufführungsdatum in einem Akkordion angezeigt werden. In der Tabelle werden nur die Aufführungszeiten angezeigt bei denen es mindestens eine Aufführung auf einer Spielstätte gibt. sollte dies nicht der Fall sein, so wird diese Spalte ausgeblendet.

Aufführungen										
24.01.2016 (217 - Aufführungen)										
24.01.2016 (1 of 1)										
Spielstätte	9 - 10	10 - 11	11 - 12	12 - 13	13 - 14	14 - 15	15 - 16	16 - 17	17 - 18	18 - 19
Dreifaltigkeitsgasse	Buckley, Stefanie	Mc Gee, Sandy	Warren, Lee	Forbes, Regina	Kim, Daniel		Nunez, Dennis	Ingram, Teddy	Larson, Annie	Anderson, Herman
Mader Reisen	Warren, Lee	Forbes, Regina	Kim, Daniel	Nunez, Dennis	Zuniga, Myra	Ingram, Teddy	Larson, Annie	Anderson, Herman	Perez, Stefanie	
Neues Rathaus	Kim, Daniel	Nunez, Dennis	Zuniga, Myra	Ingram, Teddy	Larson, Annie	Anderson, Herman		Livingston, Jennifer	Garcia, Chastity	Stokes, Evelyn
Fa. Mammut	Zuniga, Myra	Ingram, Teddy	Larson, Annie	Anderson, Herman	Livingston, Jennifer	Perez, Stefanie	Garcia, Chastity	Stokes, Evelyn	Daniel, Jeannie	Santos, Rickey

Abbildung 38: Übersicht Artisten

Die Aufführungen werden nach dem Spielstätten (Zeilen) und dessen Aufführungszeiten (Spalte) tabellarisch angezeigt.

Folgende Aktionen stehen in dieser Ansicht für nicht eingeloggte Benutzer zur Verfügung.

1. *Dialog Artistendetails öffnen*

Durch das Klicken auf einen Artistennamen in einer Spalte öffnet sich der Dialog mit den Artistendetails

2. *Dialog einzelner Spielstättendetails öffnen*

Durch das Klicken auf einen Marker-Icon, der sich neben den Namen der Spielstätte befindet, öffnet sich ein Dialog mit den Spielstättendetails, wobei hier auch eine einfache Auflistung der Aufführungen an diesem Aufführungstag auf diese Spielstätte angeführt ist.

3. *Dialog aller Spielstättendetails öffnen*

Durch das Klicken auf einen Marker-Icon, der sich im Tabellenkopf befindet, öffnet sich ein Dialog mit den Spielstättendetails, wobei hier alle in der Tabelle befindlichen Spielstätten angezeigt werden so wie auch deren Aufführungen an diesem Aufführungstag.

Folgende Aktionen stehen eingeloggten Benutzern zur Verfügung:

1. *Dialog zum Ändern einer/aller einzelnen der Aufführungen öffnen*

Durch das Klicken auf den Edit-Icon entweder neben dem Artistennamen einer Spalte oder dem Tabellenkopf kann ein Dialog geöffnet werden, über den die Aufführung geändert werden kann.

Hierbei sei angemerkt das der Edit-Icon in den einzelnen Aufführungen nur ersichtlich wenn diese auch in der Zukunft liegen. Aufführungen die in der Vergangenheit liegen oder zu diesem Zeitpunkt stattfinden können nicht bearbeitet werden.

Übung 3

Sollte ein Warn-Icon neben den Artistennamen einer Aufführung angezeigt werden, so wurde dieser verschoben, wobei die jeweils die Informationen der letzten Aufführungsdaten über einen Tooltip angezeigt werden.



Abbildung 39: Tooltip Warnung

Wenn man mit dem Cursor über einen Artistennamen in der Tabelle fährt, so wird ein Tooltip mit den wichtigsten Artistendetails angezeigt.



Abbildung 40: Tooltip Artist

Übung 3

1.3.8.5 Menü

Am unteren rechten Ende der Seite befindet sich ein sogenanntes Stack-Menü welches die Möglichkeit bietet, Aktionen auszuführen.

Sollte kein BenutzerIn eingeloggt sein, so sieht das Menü wie folgt aus.



Abbildung 41: Stack Menü wenn nicht eingeloggt

Sollte ein BenutzerIn bereits eingeloggt sein, so sieht das Menü wie folgt aus.



Abbildung 42: Stack Menü wenn eingeloggt

Durch Klicken auf den Korb kann das Menü auch weg geklappt werden.



Abbildung 43: Stack Menü zusammengeklappt

1.3.8.6 Login

Für registrierte BenutzerInnen steht die Möglichkeit zur Verfügung sich einzuloggen, damit diese BenutzerInnen Aufführungen anlegen und bestehende Aufführungen ändern können. Durch das Klicken auf das Login Menü wird ein Dialog geöffnet, über den man sich einloggen kann. Durch Klicken auf das Loginmenü wird der Login Dialog geöffnet.

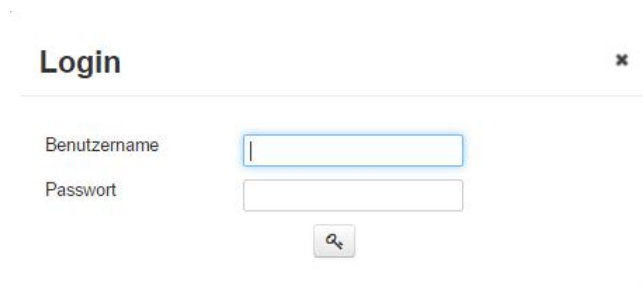


Abbildung 44: Login Dialog

Sollten keine Daten eingegeben worden sein, so werden die Eingabefelder entsprechend markiert.

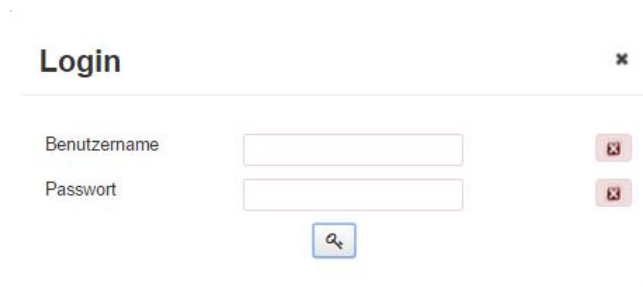


Abbildung 45: Fehlerhafte Eingabe

Sollte der Login aufgrund fehlerhafter Login Daten fehlschlagen, so wird dies über eine Meldung angezeigt.



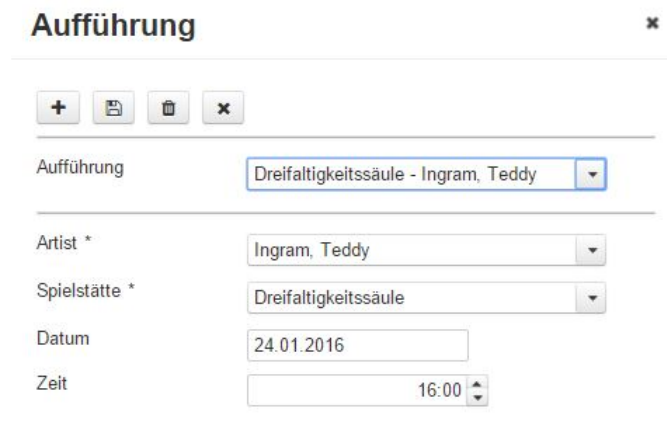
Abbildung 46: Login fehlgeschlagen

Nachdem einem erfolgreichen Login wird wieder auf die Hauptseite zurückgekehrt und der Dialog wird geschlossen. Anschließend stehen im Menü alle Funktionen für einen eingeloggten BenutzerIn zur Verfügung. Ebenso werden die Icons in der Aufführungstabelle angezeigt, über die der Aufführungsdiallog angezeigt werden kann.

Übung 3

1.3.8.7 Aufführung anlegen/bearbeiten

Über einen Dialog können bestehende Aufführungen bearbeitet sowie neue Aufführungen angelegt werden. Sollte der Dialog über den Edit-Icon in einer Spalte der Aufführungstabelle angelegt werden so wird diese Aufführung im Dialog gesetzt. Ansonsten wird ein leeres Formular angezeigt die AnwenderInnen können über eine Auswahl eine Aufführung auswählen. Diese Auswahl beschränkt sich auf die Aufführungen, die einerseits in der Seite zu sehen sind und andererseits auf die Aufführungen, die geändert werden dürfen.



The screenshot shows a dialog box titled 'Aufführung' with a close button (x). Below the title bar are three icons: a plus sign (+), a document with a pencil (edit), and a trash can (delete). The main form contains the following fields:

- Aufführung:** A dropdown menu showing 'Dreifaltigkeitssäule - Ingram, Teddy'.
- Artist *:** A dropdown menu showing 'Ingram, Teddy'.
- Spielstätte *:** A dropdown menu showing 'Dreifaltigkeitssäule'.
- Datum:** A text input field containing '24.01.2016'.
- Zeit:** A time selection field showing '16:00'.

Abbildung 47: Dialog Aufführung anlegen/bearbeiten

Sollten die eingegebene Daten ungültig sein, so werden die Eingabefelder dementsprechend markiert.



The screenshot shows the same 'Aufführung' dialog box, but with the following changes to indicate invalid input:

- Aufführung:** The dropdown menu now shows 'Neuer Eintrag'.
- Artist *:** The dropdown menu shows 'Bitte wählen' and has a red border and a red 'x' icon to its right.
- Spielstätte *:** The dropdown menu shows 'Bitte wählen' and has a red border and a red 'x' icon to its right.
- Datum:** The text input field still contains '24.01.2016'.
- Zeit:** The time selection field now shows '15:00'.

Abbildung 48: Ungültige Eingabe

Übung 3

Sollte die Aufführung aufgrund eines logischen Fehlers fehlschlagen, so wird die eine entsprechende Meldung angezeigt.

Aufführung ✕

 Der Artist hat bereits eine Führung zu diesem Zeitpunkt oder die einzuhaltende Pause zwischen den Aufführungen wurde unterschritten

+ 📄 ✕

Aufführung

Neuer Eintrag ▾

Artist *

Ingram, Teddy ▾

Spielstätte *

Dreifaltigkeitssäule ▾

Datum

24.01.2016

Zeit

17:00 ▴ ▾

Abbildung 49: Fehlgeschlagene logische Prüfung

Übung 3

1.3.8.8 Dialog Spielstätte

Es können die Spielstätten in einem Dialog angezeigt werden wobei es hier zwei Möglichkeiten gibt:

1. Mit Aufführungsdaten

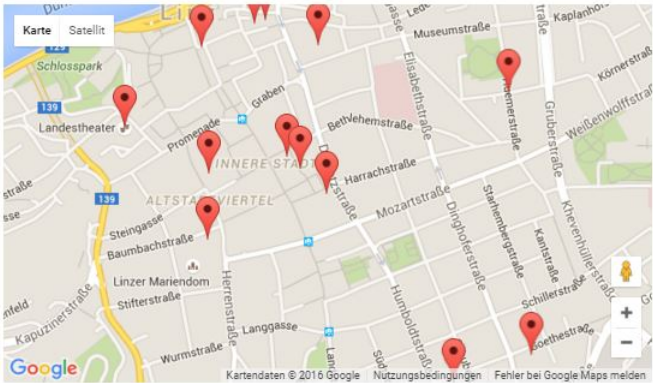
Sollte der Dialog über die Aufführungstabelle geöffnet werden, so sind hier die Aufführungsdetails der Spielstätte(n) am Aufführungstag angeführt. Durch das Öffnen des Dialogs über den Tabellenkopf werden alle Spielstätten dieser Tabelle angeführt, ansonsten nur eine einzeln.

2. Ohne Aufführungsdaten

Sollte der Dialog über die Spielstättenauswahl geöffnet werden, so werden hier lediglich die Spielstättendaten angezeigt.

Die Spielstätten werden hierbei auf einer Google-Maps angezeigt, wobei im Falle mit den Aufführungsdetails die einzelnen Spielstätten auch ausgewählt werden können und die dementsprechenden Ausführungsdetails ausgewählt werden.

Spielstätten - 24.01.2016



Aufführungen 12

Neues Rathaus
(1 of 1)

Zeit ↕	Artist
09:00:00 - 10:00:00	Kim, Daniel
10:00:00 - 11:00:00	Nunez, Dennis
11:00:00 - 12:00:00	Zuniga, Myra
12:00:00 - 13:00:00	Ingram, Teddy
13:00:00 - 14:00:00	Larson, Annie
14:00:00 - 15:00:00	Anderson, Herman
15:00:00 - 16:00:00 ⚠	Livingston, Jenifer
17:00:00 - 18:00:00	Garcia, Chastity
18:00:00 - 19:00:00	Stokes, Evelyn

Abbildung 50: Spielstätten Dialog mit Aufführungsdetails

Auch in dieser Tabelle werden über Tooltips Informationen des Artisten sowie auch von verschobenen Aufführungen angezeigt.



Abbildung 51: Spielstätten Dialog ohne Aufführungsdetails