

1 Ultimate - Festival- Orgainzer

Ultimate Festival Organizer, ist ein Semesterprojekt welches in den Fächern SWK5 und WEA5 als Übung umzusetzen ist. Eine detaillierte Beschreibung der Softwarelösung, kann aus den Dokument Anforderungen entnommen werden.

1.1 Ausbaustufe 1

In der Ausbaustufe 1, ist sowohl die Datenbank, als auch die Datenzugriffsschicht zu Implementieren. Die Datenzugriffsschicht, muss in ADO.NET realisiert werden, und es dürfen keine OR-Mapper verwendet werden.

1.1.1 Systemaufbau

In der Ausbaustufe 1 beschäftigen wir uns lediglich mit der Datenbank, beziehungsweise deren Zugriffsschicht. Aus den Anforderungen können wir jedoch entnehmen, dass wir in einer der nächsten Ausbaustufen ein Webservice realisieren werden, welches unsere Geschäftslogik abwickelt. Diese Schicht, welche die Geschäftslogik enthält, wird in unserem Fall auch sämtliche Prüfungen auf Daten Validität durchführen. In der Datenbankschicht kümmern wir uns um diese nicht. Anhand der ProjektStruktur der Datenbankschicht, können wir gut den Aufbau erkennen:

- UltimateFestivalOrganizer.DAL.Common: Dieses Projekt enthält die POCOS, sowie Interfaces für unsere ein Datenbankobjekt und den DAOS. Weiters enthält es eine DALFactory, welches anhand der Konfiguration die Richtigen DAOS, beziehungsweise Datenbank Objekte erzeugt.
- UltimateFestivalOrganizer.DAL.SqlServer: Dieses Projekt enthält die spezifische Implementierung, der DAL.Common Interfaces für eine SqlServer Datenbank.
- UltimateFestivalOrganizer.DAL.Test: Dieses Projekt enthält die Tests für die Datenzugriffsschicht. Weiters enthält dieses Projekt eine eigene Datenbank, für Integration-Tests, und SQL Scripts zum löschen, und wiederherstellen des Urzustandes der Datenbank.

Als Datenbank wird eine SQLServer LocalDB verwendet.

1.1.2 Datenbank

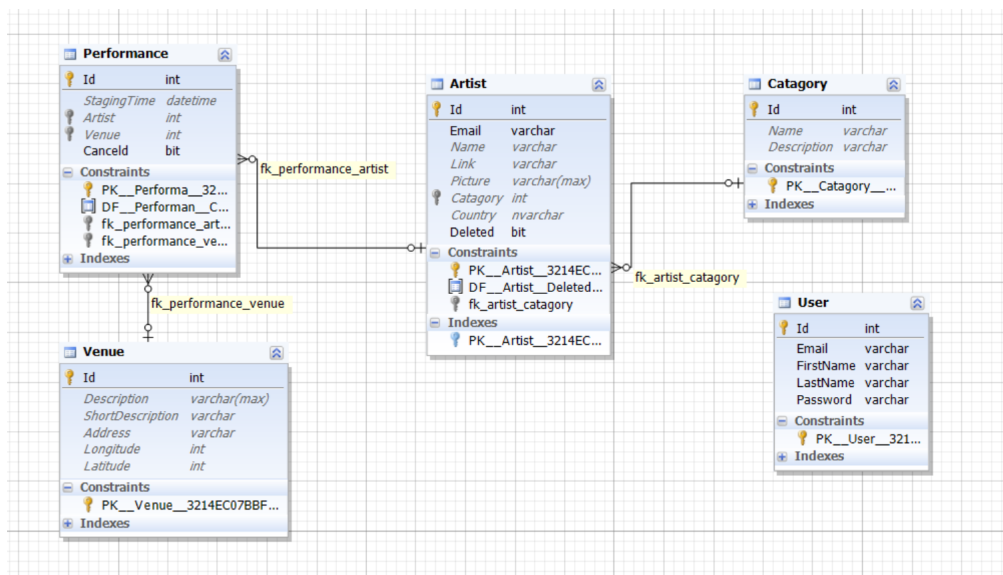


Abbildung 1: Entity Relationship Diagram

Da Performance und Artists, in unserer Geschäftslogik nicht gelöscht werden, wurde ein Flag eingefügt, für Canceled bzw. Deleted. Es wird ein löschen auf Datenbankebene jedoch erlaubt. Die zwei Felder dienen dazu, um eben in der Geschäftslogik nicht löschen zu müssen, sondern nur als Gelöscht markieren zu können. Um die Datenzugriffsschicht zu erleichtern, wurde auf jede Entität ein Surrogate gelegt. Die Tatsächlichen Keys, wurden jedoch mit Unique-Constraints realisiert.

1.1.3 Datenzugriffs-Schicht

- UltimateFestivalOrganizer.DAL.Common

- POCOS: Für jede Entität wurde ein POCO erstellt. Um später einen Dynamischen Dao verwenden zu können, gelten für jedes POCO folgende Konventionen.

- * Jedes POCO leitet von der Basis-Klasse BaseEntity ab
- * Jedes POCO enthält als Klassen-Attribut das Table , mit welchem der Name der Datenbanktabelle angegeben wird.
- * Jedes POCO enthält als PropertyAttribute Id und AutogeneratedId, auf jenes Property welches die Id darstellt.

Da im Laufe der Entwicklung Surrogaten als PrimaryKey eingeführt worden sind, ist es nicht mehr notwendig, in einem abgeleiteten POCO das Attribut Id und AutogeneatedId zu verwenden. Dies wird bereits im BaseEntity richtig gesetzt.

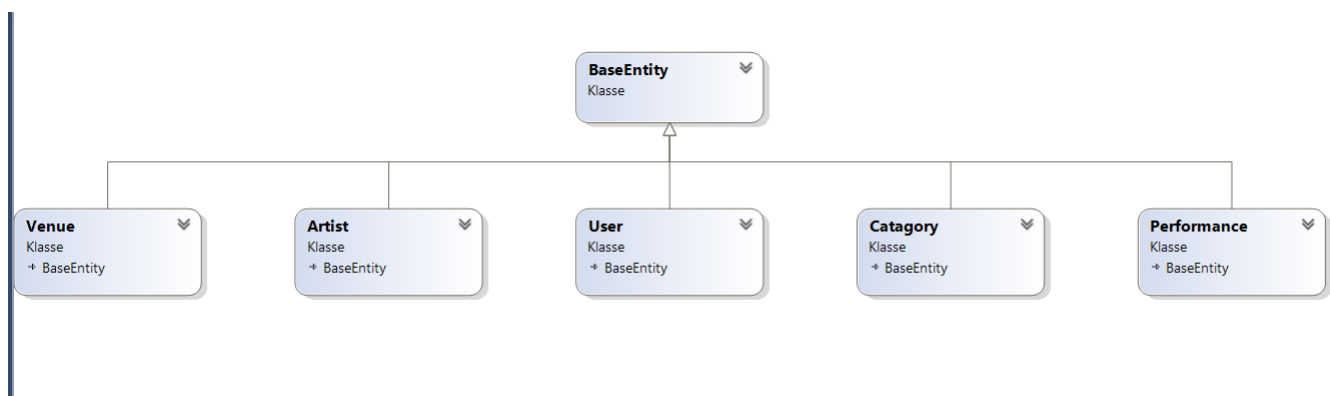


Abbildung 2: Klassendiagramm Pocos

- Dao: Im Package Common, werden lediglich Interfaces für die Daos deklariert. Hier gilt wiederum, jedes neu erstellte Dao, muss vom TemplateDao IBaseDao ableiten. Die Implementierung dieses Abstrakten DAO, wird später so ausgeführt werden, dass Basis Funktionalitäten wie Insert, Update, Delete, findById, findAll zur verfügung gestellt werden. Das Interface IDao wird benötigt, um später dynamisch spezifische Daos erzeugen zu können. Lediglich die Implemtierung vom BaseDao implementiert auch dieses Interface. Alle anderen Daos leiten nur vom BaseDao ab.

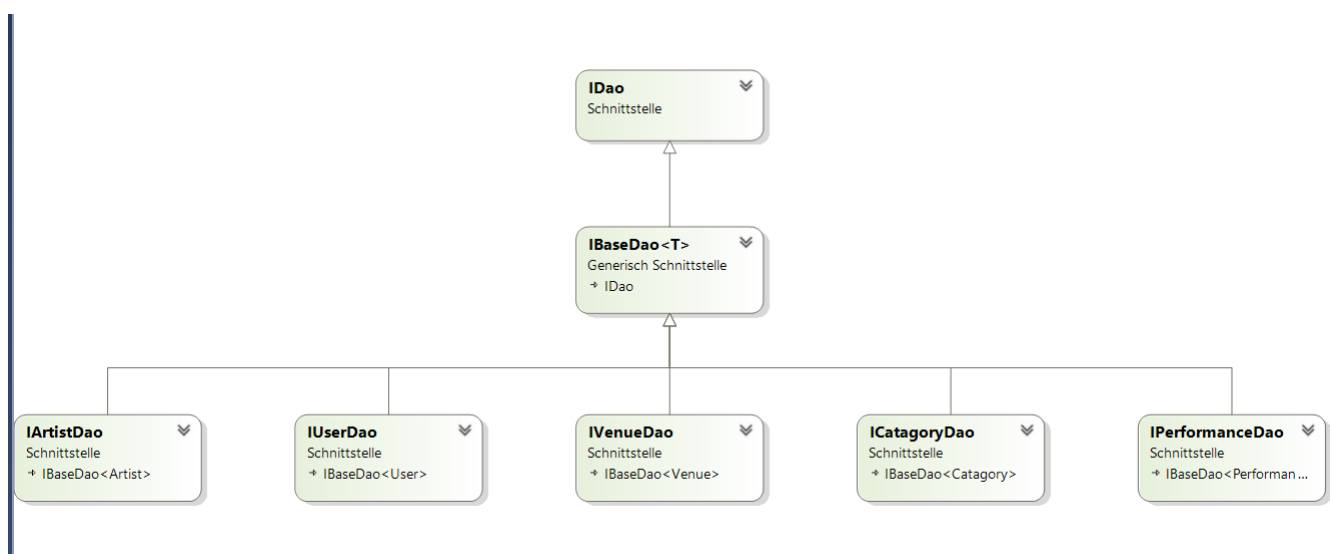


Abbildung 3: Klassendiagramm Dao Interfaces

- Util: Entählt Hilfsmethoden um z.B. die Attribute eines POCOs auslesen zu können.

- IDatabase: Interface, für ein Datenbankobjekt, welches Transaktionen mithilfe von ADO.NET durchführt.
- DALFactory: In der DALFactory werden anhand von der Projektkonfigurationsdatei, DatenbankObjekte aber auch Daos erstellt. Weiters wurde eine Methode erstellt, welche es schafft, anhand eines POCO types den richtigen DAO zurück zu liefern. Um dies realisieren zu können, wird auch das nicht typisierte Interface IDao benötigt.

- UltimateFestivalOrganizer.DAL.SqlServer

- Dao: BaseDao implementiert die Schnittstelle IDao, sowie IBaseDao. Funktionen wie findAll, findById, Insert, Update, Delete werden hier dynamisch per Reflection realisiert. Die Schnittstelle IDao ist so implementiert, das sie lediglich die Funktionalitäten, der BaseDao aufruft.

```
public IList<T> findAll()
{
    DbCommand command = database.CreateCommand(this.GetDefaultSelect());
    using (IDataReader reader = database.ExecuteReader(command))
    {
        return ConvertResultToList(reader);
    }
}

object IDao.findAll()
{
    return findAll();
}
```

Diese Methode der Implementierung wird benötigt, da es beim dynamischen Mappen vom ResultSet sein kann, dass ein ForeignKey aufgelöst werden muss. Dann benötigen wir einen neuen Dao, um dieses ForeignKey Objekt zu holen. Um dann nicht im Generischen Dao entscheiden zu müssen, welchen Typ von Dao man benötigt, kann man so einen Dao von der Factory holen, indem man einfach den Typ der Entität mitgibt. Diese entscheidet welchen Dao man benötigt, und liefert diesen als IDao zurück.

Wichtig, damit diese Basisimplementierungen funktionieren, muss sich im Gesamten an die oben genannten Konventionen gehalten werden. Ist dies nicht der Fall, muss man in den speziellen Dao Ausprägungen die Funktionalitäten überschreiben.

Es wäre angedacht, auch eine generische Suchmethode zu entwickeln. Diese wäre hilfreich um zum Beispiel Datensätze zu Suchen, welche in einem Bestimmten Zeitraum liegen oder ähnliches. Derzeit ist es Zeitlich aber noch nicht möglich gewesen, diese zu realisieren. Man kann, falls unbedingt benötigt, diese ja auch gezielt im jeweiligen Dao aus-programmieren. Da sie für das Webservice jedoch hilfreich sein werden, werden diese in der nächsten Ausbaustufe sicher noch entwickelt werden.

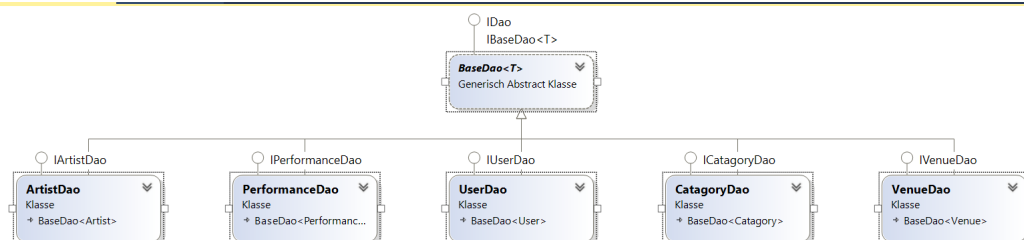


Abbildung 4: Klassendiagramm Implementierung Daos

- UltimateFestivalOrganizer.DAL.Test In diesem Projekt werden lediglich die Datenzugriffsfunktionen getestet. Da es Hier keinen Sinn macht, die Funktionen wirklich Unit zu Testen, wurden hier Integration Tests erstellt. Alle Tests werden von einer Basis-Klasse abgeleitet. Diese Klasse enthält TestInitialize und Cleanup methoden. In dem Projekt existiert eine App.config, in welcher das Datenbankfile, sowie ein Ordern für Delete/Create/Insert files, welche vor jedem Test automatisch von der Basis Testklasse ausgeführt wird.

- BaseTest Der Test ist die Basisklasse einer jeden Testklasse in diesem Projekt. In der Methode TestInitialize wird hier jeweils die TestDatenbank komplett bereinigt. Danach wird die Struktur neu erstellt, und es werden definiert Testdatensätze eingefügt. Daher ist es wichtig, dass jeder Testfall von dieser Methode ableitet, damit sichergestellt ist, dass die Tests immer auf bereinigten Daten ausgeführt werden.

Mit den im Projekt enthaltenen Testfällen, wird folgendes Ergebnis erzielt:

The screenshot displays the Test Explorer and Code Coverage windows in Visual Studio. The Test Explorer shows a list of tests grouped by category, with their execution times and status (all passed). The Code Coverage window shows the coverage percentage for the DAL classes.

Hierarchie	Nicht abgedeckt (Blöcke)	Nicht abgedeckt (% Blöcke)	Abgedeckt (Blöcke)	Abgedeckt (% Blöcke)
Wolfgang_WOLFGANG-PC 2015-11-20 14...	57	7,07 %	749	92,93 %
ultimatefestivalorganizer.dal.c...	13	8,02 %	149	91,98 %
ultimatefestivalorganizer.dal.s...	44	12,61 %	305	87,39 %
ultimatefestivalorganizer.dal.t...	0	0,00 %	295	100,00 %

Abbildung 5: DAL Testabdeckung

1.2 Ausbaustufe 2

In dieser Ausbaustufe wird sowohl die Business-Logik, als auch eine GUI entwickelt.

1.2.1 BusinessLogik

Die BusinessLogik dient zur Abbildung von Speichern, Bearbeiten, holen von Daten. Dazu werden in erster Linie zwei Interfaces erstellt. IAdministrationService, enthält alle Methoden, die für die Applikation benötigt werden, um Daten zu bearbeiten zu kommen. In einem zweiten Interface, werden jene funktionalitäten abgebildet, welche nur lesenden Zugriff erhalten. Die Implementierung ist so erfolgt, dass alle Komponenten keine Abhängigkeiten haben. Das Business Logik Projekt, hat also nur eine Direkte abhängigkeit zum Common, aber nicht zu einer konkreten Implementierung der DAL Klassen. Eine Factory wird benutzt, um eine konkrete Implementierung der AdministrationInterfaces zu bekommen. Aus Zeitlichen gründen, wird die BusinessLogik, welche nur schreibend zugreift, und z.B. für das Webservice verwendet wird, erst dann entwickelt, wenn es auch wirklich benötigt wird. In dem Projekt BusinessLogik.Test befinden sich Unit und Integration Tests für die oben genannten Implementierungen.

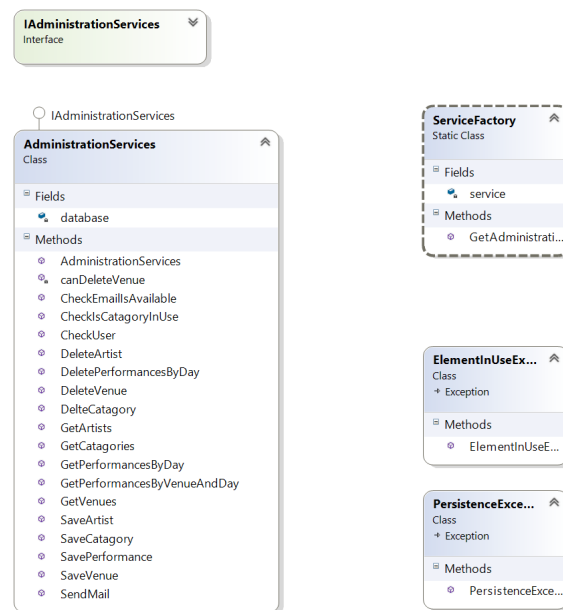


Abbildung 6: Klassendiagramm Business Logik

1.2.2 Commander

In diesem Projekt wird eine WPF-Applikation erstellt. Dieses Projekt hat eine Abhängigkeit zum Common, und zur BusinessLogik. In der MainView wird ein AdministrationService erzeugt, welches dann an alle anderen ViewModels per Konstruktor mit übergeben wird. Die erzeugung erfolgt über die Factory. Schöner wäre es eventuell noch, wenn man die gesamtheit der Objekte über Dependency Injektion lösen würde. In diesem Fall ist es aber denke ich eine ebenfalls praktikable Lösung, wenn man es sich einmal erzeugt, und über den Konstruktor in die restlichen ViewModels mitzieht.

Die Kommunikation zwischen den ViewModels funktioniert per Messaging. Diese Methode liefert MVVM-Light, welches aus diesem Grund, für genau diesen einen Zweck verwendet wird. In der AppMessages Klasse befinden sich alle möglichen Messages. Alle ViewModels können sich danach auf diese Registrieren, und diese werfen. Z.B. Ändert sich eine Catagory wird eine CatagoryChanged message geworfen, und die anderen ViewModels, welche Catagories benötigen, können dies auf aktuellen stand bringen.

Zur Validierung verwende ich das Framework MVVM-Validation. Dazu wurde im BaseModel IDataErrorInfo implementiert, um die Fehler auch korrekt anzuzeigen.

Zur Anzeige generell wurde MahAppsMetro verwendet. Dies bringt einen großen Designvorteil, und zum Beispiel eine vielzahl von Buttons usw..

Es gibt in meiner Implementierung so gut wie keinen Code-Behind. Lediglich in der AdministrationWindow.xaml wird der richtige DataContext einmal gesetzt. Weiters befindet sich auch die Anzeige von zwei MessageBoxen hier. Die BusinessLogik, wird ebenfalls über eine Faktory geholt, und hier beim Anlegen des AdministrationVM zugewiesen.

Der FileDialog wurde ebenfalls über einen Service angesprochen. Diesen Service habe ich jedoch nicht in die BusinessLogik gepackt, sondern im GUI Projekt gelassen. Der Grund ist, dass dieses Service ja ein Fenster öffnet, um eine Datei auszuwählen, wesshalb es für mich eher in das GUI projekt gehört.

Testfälle für die ViewModels wurden nicht erstellt. In der Praxis sollte man aber durchaus die ViewModels Unit-Testen.

Für das Senden von Mails, kann in der App.config konfiguriert werden, dass die Mails nicht direkt versendet werden, sondern einfach in einen Ordner gespeichert werden. Um die Tests zu beschleunigen und zu vereinfachen, wurde dies so konfiguriert.

Die einzelnen Administrationsschritte wurden in einzelne Tabreiter aufgebaut. Jeder Reiter enthält dabei die Gesamte funktionalität um z.B. die Artisten zu bearbeiten. Alle Felder, welche validiert werden müssen, und fehlerhaft werden, werden rot umrahmt. Die GUI wurde wie folgt designed, und stellt folgende Funktionalitäten dar:

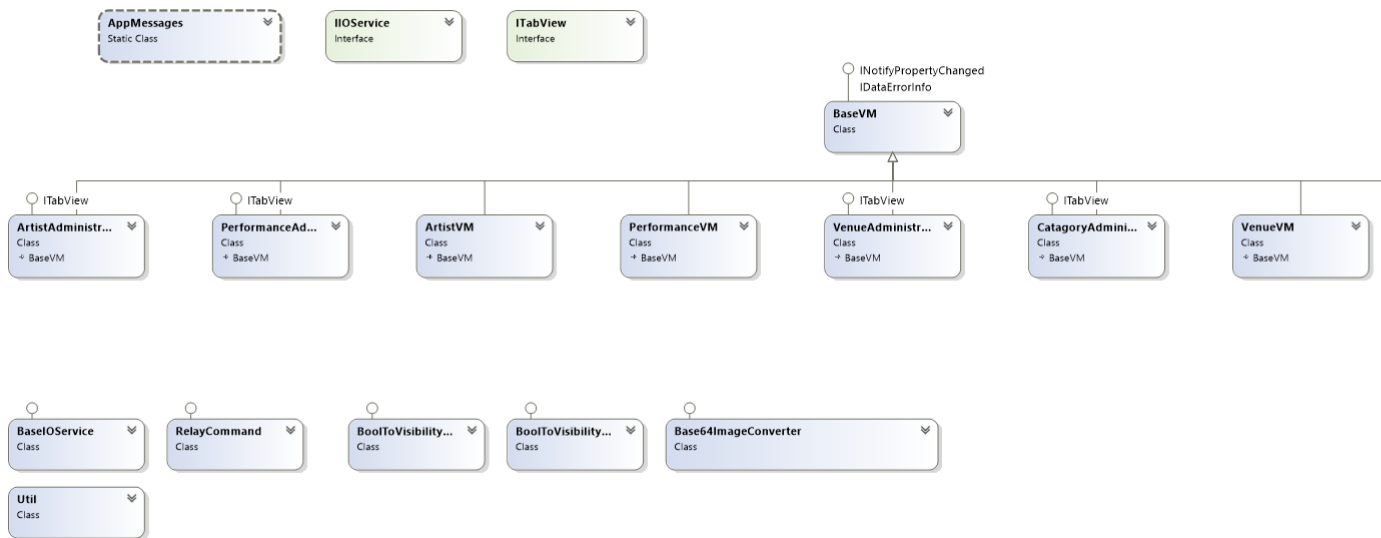
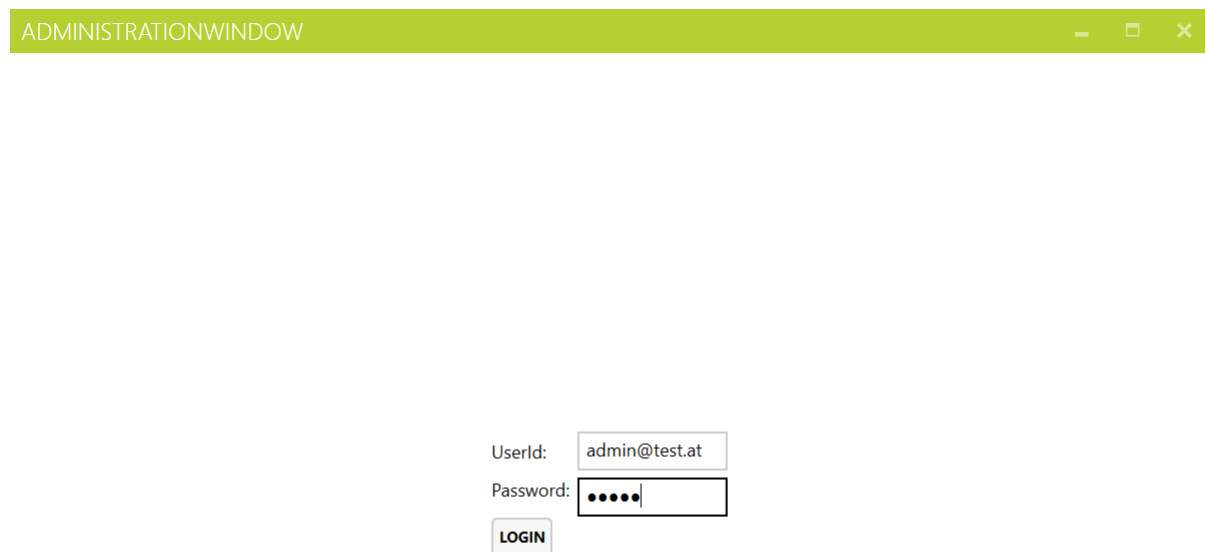


Abbildung 7: Klassendiagramm Dao Interfaces

- Login Gültige Logindaten um sich einzuloggen sind admin@test.at/admin. Gibt man ungültige LoginDaten ein, bleibt man am gleichen Screen und eine Fehlermeldung wird angezeigt.
- Artists Hier können Artisten bearbeitet, gelöscht oder hinzugefügt werden. Es können nur jene Artists gelöscht werden, welche keine aufführung haben.
- Catagories Erklärung siehe Artists.
- Venues Erklärung siehe Artists.
- Performances Der Button Email in der Menü-Leiste sendet, das Tagesprogramm an alle beteiligten User. Möchte man nur einen Artist bescheid geben, so kann der Email-Button am entsprechenden Element verwendet werden. Vor dem Speichern wird das ganze Programm validiert. Mit dem Häkchen kann das gesamte Tagesprogramm validiert werden. Wird ein Artist mit dem Dropdown geändert, so wird im hintergrund sofort eine Validierung durchgeführt.

Die Meldung das die Validierung erfolgreich war, wird nicht angezeigt, wenn die Validierung impliziet im hintergrund läuft. Nur wenn sie mit dem Häkchen gestartet wurde.



The image shows a login interface within a window titled "ADMINISTRATIONWINDOW". The window has a green title bar with standard minimize, maximize, and close buttons. Below the title bar, there are two input fields: "UserId:" containing "admin@test.at" and "Password:" containing five dots. A "LOGIN" button is positioned below the password field.

ADMINISTRATIONWINDOW

UserId: admin@test.at

Password: |

LOGIN

Abbildung 8: Login

ADMINISTRATIONWINDOW

Artists Catagories Venues Performances

+

📁

🗑️

Klassik

Larry Page

sandra riley

audrey hughes

roberto medina

reginald daniels

dddddd

ffffff

asdfasdfdfas

Catagory 3

Marry Page

gilbert phillips

ramona mendoza

tyler lewis

jonathan owens

adam bell

brett robinson

Catagory 2

TO DELETE

beverley turner

alexander gray

ralph sullivan

Name:

Larry Page

Email:

l.page@gmx.com

Link:

asdf

Country:

Catagory:

Klassik

Vorschau-Bild:




Abbildung 9: Administration Artists

ADMINISTRATIONWINDOW

Artists

Catagories

Venues

Performances

+

📁

🗑️

Klassik

Sport

Catagory 1

Catagory 2

Catagory 3

Catagory 4

Catagory 5

Catagory 6

Catagory 7

Catagory 8

Catagory 9

Catagory 10

Catagory 1

Catagory 2

Catagory 3

Catagory 4

Catagory 5

Catagory 6

Catagory 7

Catagory 8

Catagory 9

Name:

Klassik

Description:

Klassik

Abbildung 10: Administration Catagories

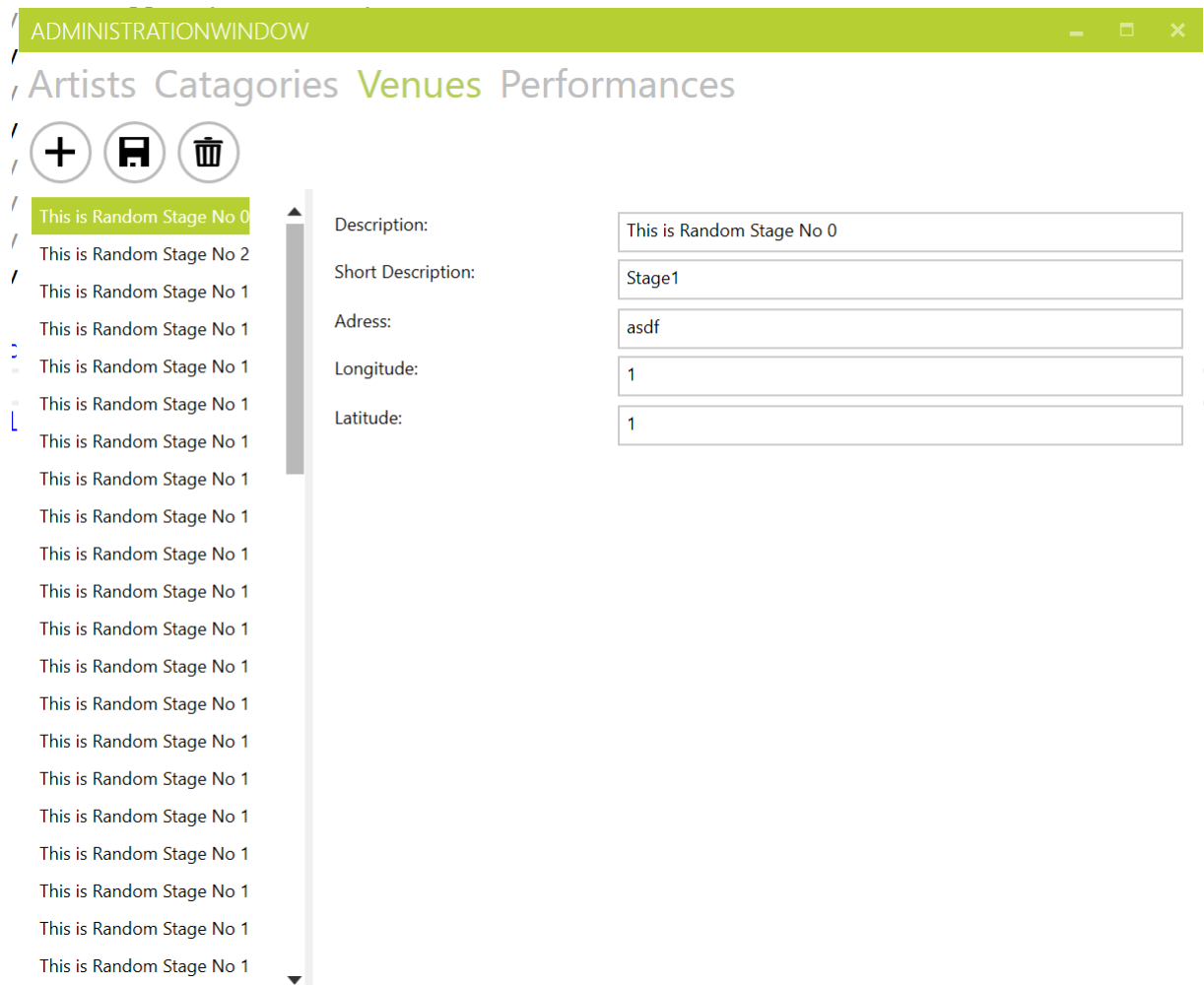


Abbildung 11: Administration Venues

ADMINISTRATIONWINDOW

Artists Catagories Venues Performances

✓

💾

☑

20.12.2015

📅 14

Venues

14-15

15-16

16-17

17-18

18-19

This is Random Stage No 0

ARG
Klassik
ddddd

ddddd

🗑

☑

This is Random Stage No 2

AUT
Catagory 1
isaac sims

isaac sims

🗑

☑

This is Random Stage No 1

AUT
Catagory 6
alicia knight

alicia knight

🗑

☑

This is Random Stage No 1

🗑

☑

Klassik
asdfsdfasdfas

asdfsdfasdfas

🗑

☑

AUT
Catagory 5
ernest moore

ernest moore

🗑

☑

AUT
Klassik
sandra riley

sandra riley

🗑

☑

Klassik
Larry Page

Larry Page

🗑

☑

🗑

☑

🗑

☑

AUT
Catagory 1
leon carroll

leon carroll

🗑

☑

AUT
Klassik
reginald daniels

reginald daniels

🗑

☑

🗑

☑

🗑

☑

🗑

☑

🗑

☑

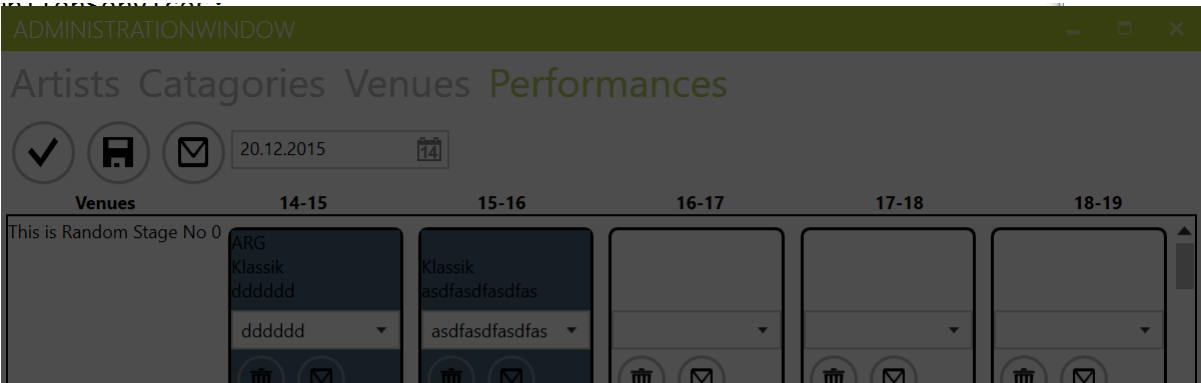
🗑

☑

Abbildung 12: Administration Performances

UltimateFestivalOrganizer

Seite 11



Success

Program is Valid

OK

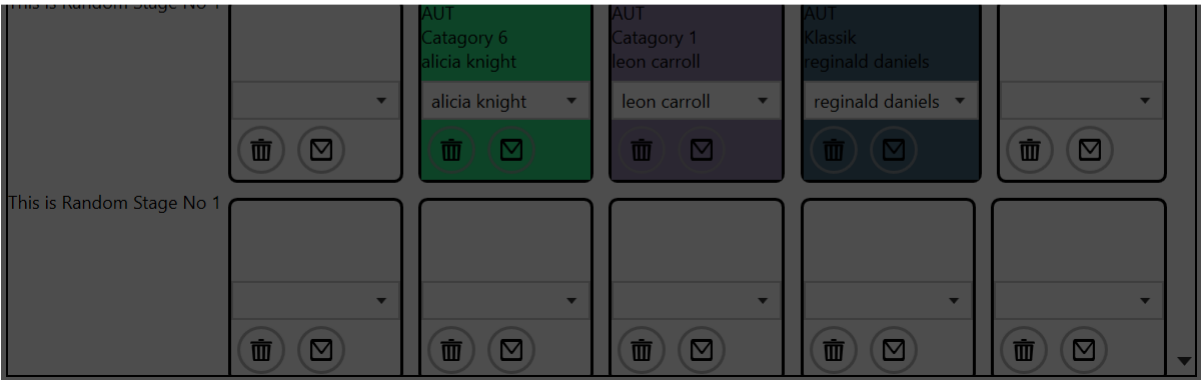


Abbildung 13: Administration Performances

Abbildungsverzeichnis

1	Entity Relationship Diagram	1
2	Klassendiagramm Pocos	2
3	Klassendiagramm Dao Interfaces	2
4	Klassendiagramm Implementierung Daos	3
5	DAL Testabdeckung	4
6	Klassendiagramm Business Logik	5
7	Klassendiagramm Dao Interfaces	6
8	Login	7
9	Administration Artists	8
10	Administration Catagories	9
11	Administration Venues	10
12	Administration Performances	11
13	Administration Performances	12