1 Ultimate - Festival- Orgainzer

Ultimate Festival Organizer, ist ein Semsterprojekt 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 sämtlich 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.
- UtlimateFestivalOrganizer.DAL.CreateTestData: Dies ist ein Hilfsprojekt. Es kann ausgeführt werden, um eine Datenbank mit Testdaten zu füllen. Für die Unit-Tests wird es derzeit nicht verwendet, da ich bei diesen derzeit noch nicht mit Dynamisch generierten Testdaten umgehen kann.

Als Datenbank wird eine SQLServer LocalDB verwendet.

1.1.2 Datenbank

Es müssen folgende Entitäten in der Datenbank abgebildet werden.

- Artist: Ein Artist, wird durch einen Namen, entweder Künstlername, oder Name der Gruppe gekennzeichnet. Weiters sollen einige Zusatzinformationen gespeichert werden. Zustäzlich ist ein Artist immer genau einer Kategorie zugeordnet.
- Catagory: Eine Catagory, ist durch ihren Namen, sowie einer kurzen Beschreibung gekennzeichnet.
- Venue Venue stellt die Spielstätte dar. An einer Spielstätte können jeweils mehrere Auftritte pro Tag stattfinden, jedoch nie gleichzeitig.
- Performance Eine Performance ist jeweils ein Auftritt. Dieser enthält zumindest eine Spielstätte, einen Akteur (Artist), sowie einen fixen Zeitpunkt. Eine Performance kann nicht gelöscht werden, jedoch wird ein Artist gelöscht oder als gelöscht gekennzeichnet, besteht die Möglichkeit, die Performance als abgesagt zu Kennzeichnen.
- User Enthält Name, sowie Email und Passwort. Email und Passwort, werden als User Identifikation für einen in einer Späteren Ausbaustufe realisierten Login verwenden.

Um die Datenzugriffschicht zu erleichtern, wurde auf jede Entiät ein Surogate 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.

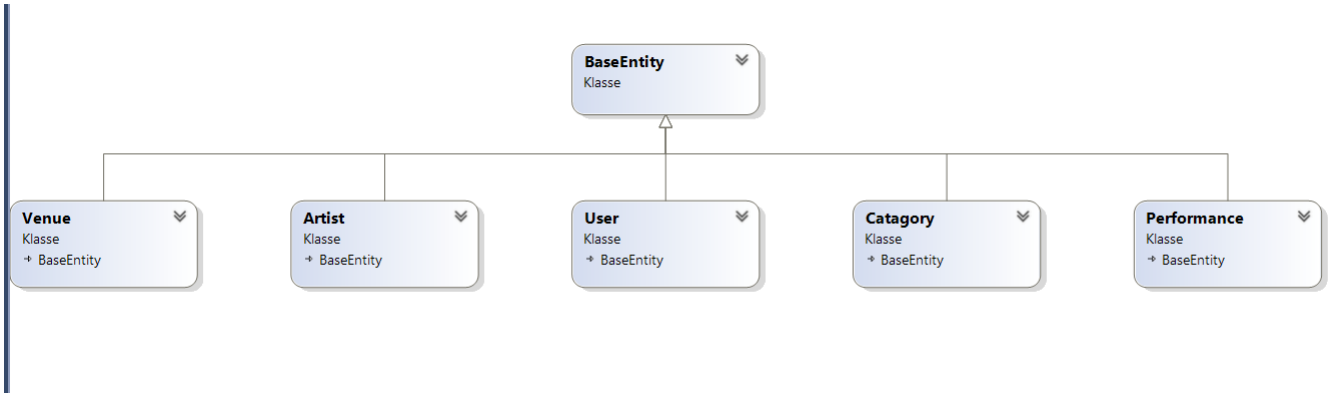

Abbildung 1: Klassendiagramm Pocos

  - 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.

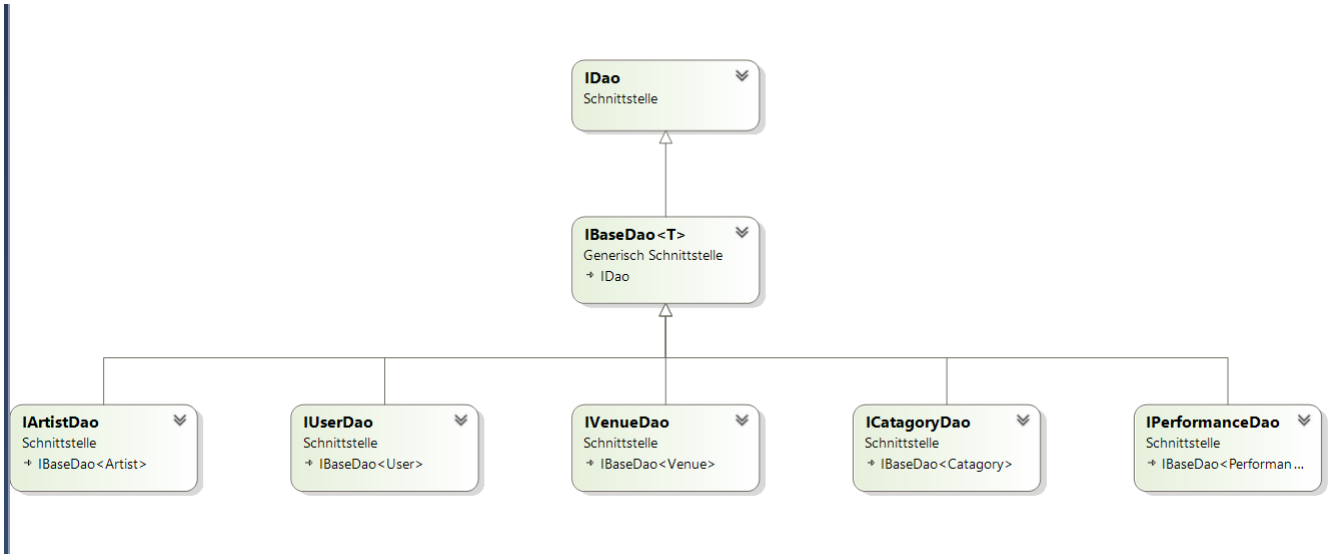

Abbildung 2: Klassendiagramm Dao Interfaces

  - Util: Entählt Hiflsmethoden 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 zuliefern. 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.

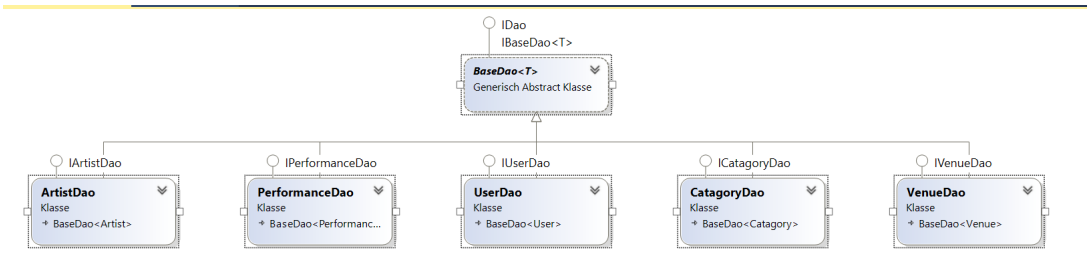

Abbildung 3: Klassendiagramm Implementierung Daos

- 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.
  - BaseTest Der Test ist die Basisklasse einer jeden Testklasse in diesem Projekt. In der Methode TestInitzialize 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, das die Tests immer auf bereinigten Daten ausgeführt werden.

Abbildungsverzeichnis