**MainWindow.xaml.cs**

*Konstruktor der MainWindow Komponente*



Das Programm „PicDB“, ein Bilder-Datenbank, beginnt grundsätzlich in der MainWindow.xaml.cs Datei, da sie als erstes aufgerufen wird. Dabei wird vom gleichnamigen ViewModel eine Instanz erstellt. Darauf folgt die Initialisierung der View bzw. der UI durch die InitializeComponent() Methode und damit das Binding für diese Komponente funktioniert, wird dem DataContext fürs MainWindow.xaml.cs der controller überwiesen.

**Strukturierung**

Das Projekt ist in mehrere logische Teile bzw. Ebenen/Layer aufgeteilt. Das sorgt zum einen für Übersichtlichkeit und zum anderen für Modularität.

Diese wären folgende:

* Data Access Layer
* Business Layer
* Models
* ViewModels
* Views/UI

**Data Access Layer**

Die Data Access Ebene (DAL) ist für den Zugriff auf die Daten zuständig. Hier werden die Daten von der Datenbank bzw. andere Formen der Datenspeicherung abgefragt. Durch diese Kapselung ist es möglich im Falle einer Äderung der Datenspeicherung, diese zu ändern, ohne dass es Auswirkung auf den restlichen Code hat.

**Business Layer**

Die Business Logic Ebene (BL) beinhaltet die Logik für den Zugriff auf Daten. Sie ist für die Kommunikation zwischen der Datenbank, sprich der Data Access Ebene und Ebenen, die diese Daten benötigen zuständig. In unserem Falle wären das die ViewModels.

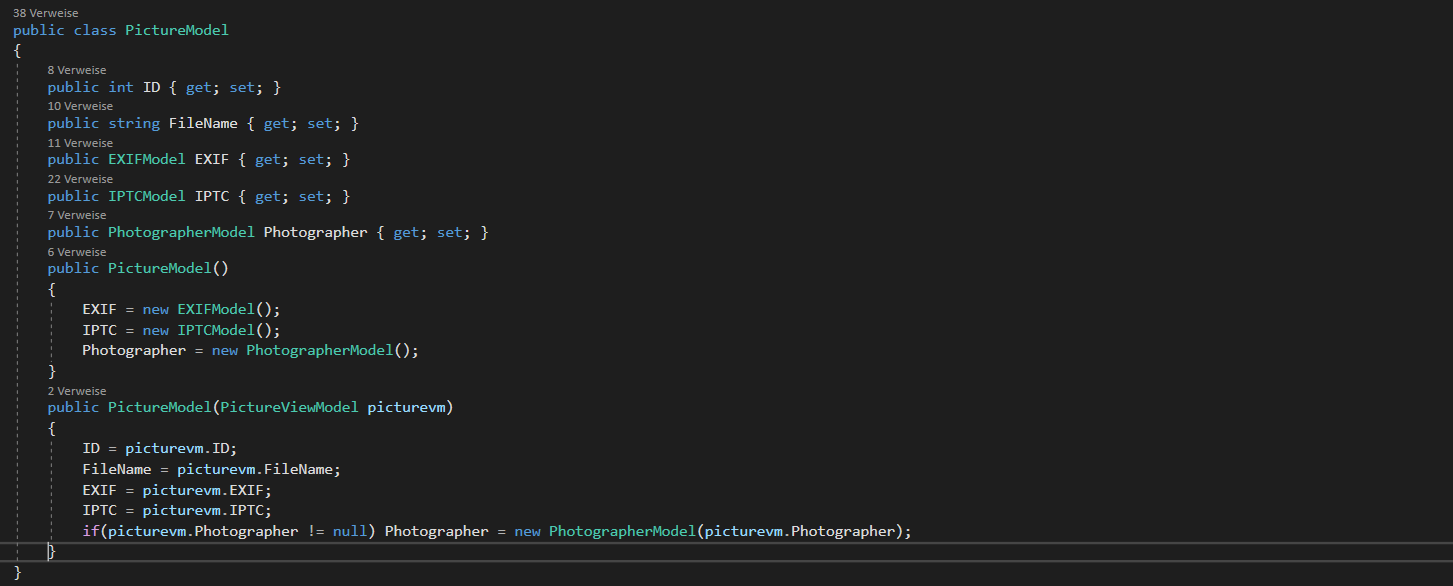
**Models**

Als Models werden Klassen spezifiziert, die die logische Struktur der Daten beinhaltet. Sprich die Objekte die für die Anwendung notwendig sind.

Diese wären folgende:

* Picture
* Photographer
* EXIF
* IPTC

*Beispiel der Picture Klasse*



**ViewModels**

Die ViewModels oder auch Controller genannt, repräsentieren das jeweilige Model, für das sie zuständig sind. Sie sorgen für die Kommunikation zwischen den Models und der View bzw UI. Alle Models haben einen ViewModel. Zusätzlich haben auch wichtige Teile der UI/View ebenfalls ViewModels, wie etwa eine ViewModel für die PictureList oder PhotographerList.

**Views/UI**

Die Views bzw. die UI ist wie der Name bereits erahnen lässt die graphische Benutzeroberfläche. Das ist der Teil, den der User oder die Userin letztendlich sieht bzw. mit dem interagiert wird. Hier werden die Daten von den Models dargestellt. Die Views selbst enthalten keine Programmierlogik, sondern nehmen nur Benutzerinteraktionen entgegen.

**DAL.cs**

Die DAL.cs Datei ist wie der Name schon verrät, die Datei für die Data Access Ebene.

Prinzipiell sind hier zwei wichtige Verbindungen. Zum einem logischerweise den ConnectionString für die Datenbank und zum anderen der Pfad für den Ordner, welcher mit den Bildern befüllt ist, die in der Anwendung angezeigt werden sollen.

Die verwende Datenbank ist eine MySQL Datenbank. Es wurde extern eine MySQL Datenbank mit XAMPP erstellt. Diese wird ebenfalls extern gestartet. Um die Anwendung mit der Datebank zu verbinden, war eine Extension für mysql notwendig. Diese bzw. mehrere benötigten Extensions wurden über den NuGet Paket Manager installiert.

Ebenfalls wurde für jede Aktion, welche für die Anwendung nötig ist, eine eigene Funktion geschrieben. Ein Beispiel für eine Aktion ist das Holen aller Bilder. Das wird in DAL.cs Datei mit der Funktion getPictures() bzw. getPictures(term) bewerkstelligt. Dabei wird in den einzelnen Funktionen eine neue MySqlConnection mit dem bereits definierten ConnectionString aufgebaut. Danach wird das SQL Statement definiert, welche die benötigten Daten suchen soll und je nachdem, ob im Statement Parameter benötigt werden, wird das Statement prepared um SQL Injections zu verhindern.

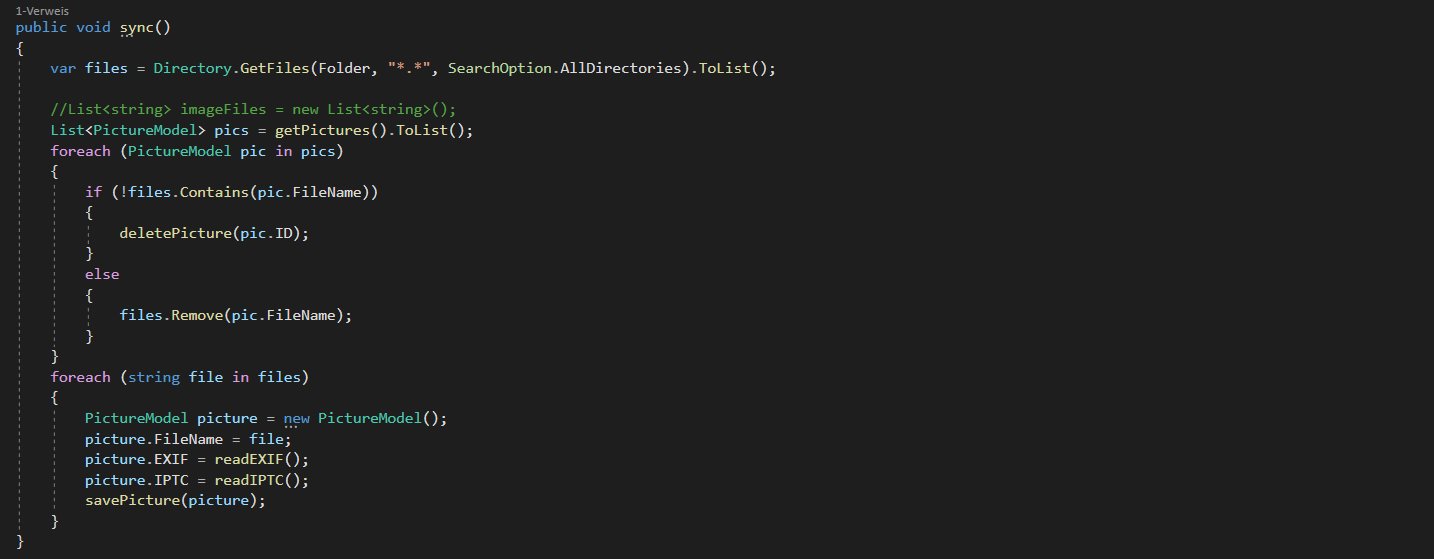
Die zurückgelieferten Daten werden dann ausgelesen und in entsprechende Klassen instanziiert und zurückgegeben. Beim bereits genannten Beispiel von getPictures() wird somit eine Liste von Picture Objekten zurückgeliefert.

Diesen Aufbau besitzen grundsätzlich alle Funktionen in der DAL.cs Klasse. Darunter zählen das Holen, Hinzufügen, Bearbeiten und Löschen für die 4 Models Picture, Photographer, EXIF, IPTC.

Lediglich die sync() Funktion stellt eine Ausnahme da. Diese Funktion ist für die Synchronisierung der Bilder zwischen dem Bilder-Ordner und der Datenbank zuständig.

Die Funktionsweise ist folgendermaßen: Es werden alle Files aus dem im Konstruktor definiertem Pfad ausgelesen und in einer List names files gespeichert. Danach werden alle Bilder, welche in der Datenbank gespeichert sind, mit der getPictures() Methode gefetcht und als List von Picture Elementen gespeichert. Darauffolgend wird mit einer foreach Schleife jene Bilder aus der Datenbank entfernt, welche sich nicht mehr im Bilder-Ordner befinden. Existieren die Bilder doch in der Datenbank, werden sie aus der files List entfernt. Somit bleiben dann nur noch jene Bilder übrig, welche sich im Bilder-Ordner befinden, aber nicht in der Datenbank gespeichert sind. Diese werden dann wiederum mit einer foreach Schleife in der Datenbank abgespeichert. Dabei wird auch das Auslesen der EXIF und IPTC Informationen simuliert.

*sync() Methode in der DAL.cs Klasse*



**PhotographerEdit**

Die PhotographerEdit Komponente ist ein eigener Teil in der Anwendung. Ähnlich wie die PictureListComponent Komponente ist sie von den anderen UI-Elementen getrennt definiert. Sie wird über das Menü aufgerufen. Über den Menüpunkt Edit -> Edit Photographer wird ein kleines Fenster geöffnet. In diesem Fenster werden alle in der Datenbank vorhandenen Fotografen angezeigt. Klickt man auf einen dieser Fotografen bzw. Fotografinnen, werden alle Informationen, wie Name, Notes, Geburtstag usw. in TextBoxen angezeigt. Innerhalb dieser TextBoxen können die Werte dann verändert bzw. bearbeitet werden. Klickt man danach auf den Button „Save“, werden die Änderungen in der Datenbank gespeichert und auf der View aktualisiert.

**PhotographerAdd**

Die PhotographerAdd Komponente ist prinzipiell die gleiche wie die PhotographerEdit Komponente. Ihre Aufgabe ist das Erstellen von neuen Fotografen bzw. Fotografinnen. Die Komponente wir ebenfalls über das Menü unter dem Menüpunkt Edit -> Add Photographer aufgerufen. Es wird ein Fenster geöffnet und man kann dann durch TextBoxes die Attribute, wie Vorname, Nachname, Notes usw., für den Fotograf bzw. der Fotografin festlegen und dann mittels des „Save“ Button speichern. Darauf wird das Photographer Objekt in der Datenbank gespeichert.

**PictureListComponent**

Die PictureListComponent Komponente ist eine externe View. Im Gegensatz zu den anderen UI-Elementen, ist dieses Element separat und kann somit wiederverwendet werden.

Dabei besteht sie zu einem aus einem UI-Teil, sprich die Benutzeroberfläche, welche mittels xaml in der PictureListComponent.xaml erstellt wird und dem Logik-Teil PictureListComponent.xaml.cs

Im UI-Teil wird eine Liste von Images als ListBox implementiert und alle Bilder, welche durch die getPictures() Methode zurückgeliefert wurden, werden dargestellt und das aktuell ausgewählte Bild, sprich jenes das in der ListBox angeklickt bzw. defaultmäßig das erste, wird in der ListBox als ausgewähltes Bild gekennzeichnet. Dabei ist kein Code notwendig, da im xaml Bindings eingebunden sind.

Dabei wird im Logik-Teil eine neue Instanz des MainWindowViewModel erstellt. Danach wird ein EventHandler „PictureSelection\_OnSelectionChanged“ implementiert, welcher bei Änderung bzw. Klicken eines Bildes in der Listbox aufgerufen wird und mit dem Controller des MainWindowViewModels das CurrentPicture, sprich jenes welches in der Anwendung groß und mit EXIF und IPTC Informationen angezeigt wird, dementsprechend anpasst.