

TSP .NET

Documentație pentru Model

Maxim Paul A6

Martie 2020

1 Introducere

Proiectul 'MyPhotos' cere managementul facil al fotografiilor și filmelor personale, cu ajutorul diverselor tehnologii din .NET Framework. Proiectul este separat în trei foldere: Model, API și GUI. Documentul prezent vizează în special partea de Model și în oarecare măsură și partea de API.

2 Modelul Design First

Primul aspect care a trebuit hotărât a fost mulțimea de entități de care am avut nevoie pentru a descrie cu ușurință și modularitate fotografiile sau filmele. Am stabilit, prin urmare, patru entități: Multimedia, Location, Person și Photo.

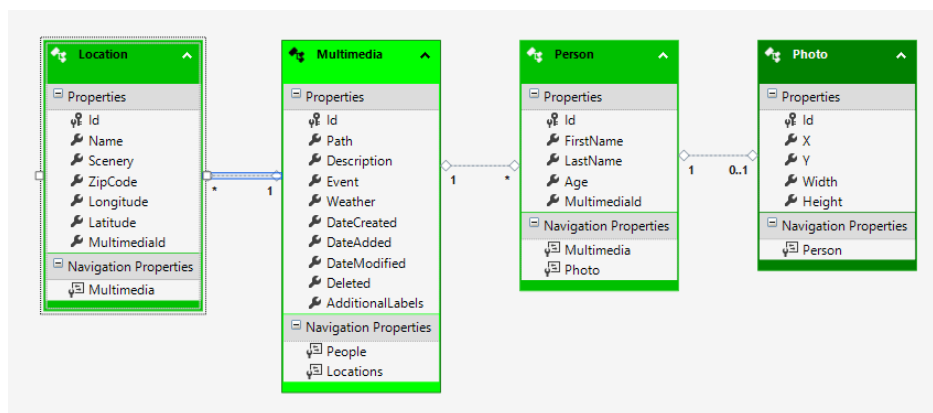


Figure 1: Entitățile din designer

3 Entități

Pentru stocarea pozelor sau filmelor am recurs la o singură entitate, 'Multimedia', ce poate să descrie ambele tipuri de fișiere. În acest fel, evităm duplicarea inutilă, iar de asemenea este ușor să ne dăm seama dacă o instanță este poză sau film. Pur și simplu verificăm extensia acestuia prin intermediul câmpului 'Path'.

Câmpurile ce aparțin 'Multimedia' sunt următoarele:

- **Id** - generat automat și auto-incrementat;
- **Path** - conține calea întreagă a fișierului, nu poate lipsi, maxim 250 de caractere;
- **Description** - evident, conține o scurtă descriere a fișierului, poate lipsi, maxim 400 de caractere;
- **Event** - conține evenimentul prezent din poză / film, poate lipsi, maxim 100 de caractere;
- **Weather** - reprezintă vremea, sub formă de enum cu 13 valori predefinite ("Sunny", "Rainy" ...etc), poate lipsi în sensul că există o valoare în enum "None" ce arată că nu se specifică / nu ne interesează vremea;
- **DateCreated** - este data în care a fost făcută poza / filmul, nu poate lipsi;
- **DateAdded** - semnifică data la care poza / filmul a fost adăugată / adăugat în aplicație, nu poate lipsi;
- **DateModified** - reprezintă ultima dată de modificare asupra entității, nu poate lipsi;
- **Deleted** - arată dacă fișierul este considerat șters în aplicație, reprezintă o valoare booleană, nu poate lipsi;
- **AdditionalLabels** - reprezintă niște etichete ce pot fi adăugate de către utilizator din aplicație, fără ca acesta să intervină în codul de bază, făcând astfel posibilă adăugarea de proprietăți în mod dinamic pentru un fișier;

Relațiile dintre 'Multimedia' și alte entități:

- cu '**Location**'
 - relație One-To-Many, o entitate '**Multimedia**' poate avea zero sau mai multe locații;
 - de ce această relație? Deoarece mai ales în filme, dar și în anumite poze, este posibil să fie prezente mai multe locații, fiecare cu diferențele lor. De asemenea, o poză sau un film poate opta să nu aibă nicio locație, pentru că este posibil ca aceasta să nu fie cunoscută;

- cu **'Person'**
 - relație One-To-Many, o entitate **'Multimedia'** poate avea zero sau mai multe persoane;
 - de ce această relație? Deoarece persoanele sunt cele care apar cel mai des în poze sau filme și de aceea am considerat că este important de a avea și o entitate ce îndeplinește acest fapt.

Pentru stocarea locațiilor am creat entitatea **'Location'**, ce conține proprietăți de bază precum:

- **Id** - generat automat și auto-incrementat;
- **Name** - conține numele locației, mai exact orașul sau adresa unei străzi, maxim 100 de caractere, poate lipsi;
- **Scenery** - descrie peisajul / peisajele din poză / filme, maxim 100 de caractere, poate lipsi;
- **ZipCode** - conține codul poștal al zonei din poză / film, maxim 16 caractere, poate lipsi;
- **Longitude** - reprezintă longitudinea, poate lipsi;
- **Latitude** - reprezintă latitudinea, poate lipsi;
- **MultimediaId** - conține Id-ul multimediei de care aparține;

Pentru stocarea persoanelor am creat entitatea **'Person'** și are proprietățile următoare:

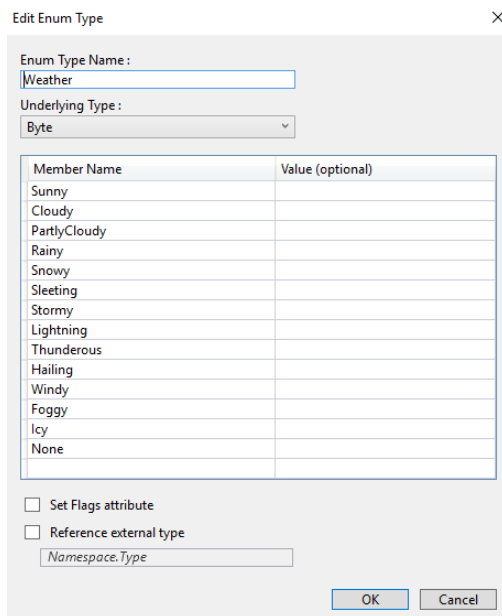
- **Id** - generat automat și auto-incrementat;
- **FirstName** - conține numele de familie a persoanei, maxim 40 de caractere, poate lipsi;
- **LastName** - conține prenumele persoanei, maxim 40 de caractere, poate lipsi;
- **Age** - conține vârsta persoanei la momentul când a fost făcută poza / filmul, reprezintă un număr între 0 și 255, poate lipsi;
- **MultimediaId** - conține Id-ul multimediei de care aparține;

Între entitatea **'Person'** și **'Photo'** există o relație One-To-Zero or One. Am decis ca fiecare persoană din poză / film, să aibă, după propria alegere (deci aceasta poate lipsi), propria lui poză decupată din poza cea mare.

4 Alte tipuri

Am decis ca funcțiile de interogare din baza de date să se afle în API, deoarece acestea sunt în mare măsură de tip CRUD, deci nu efectuează niște operații complexe sau de lungă durată.

Pe lângă entitățile și asocierile descrise anterior, modelul conține și un Enum Type și anume 'Weather'. Acesta descrie vremea dintr-o poză sau film și are un total de treisprezece valori, printre care se numără și o valoare "None" ce arată faptul că nu se dorește a specifica ce vreme se află în poză / film.



The screenshot shows the 'Edit Enum Type' dialog box. The 'Enum Type Name' is 'Weather'. The 'Underlying Type' is 'Byte'. Below these is a table with 13 rows, each with a 'Member Name' and a 'Value (optional)' column. The member names are Sunny, Cloudy, PartlyCloudy, Rainy, Snowy, Sleetng, Stormy, Lightning, Thunderous, Hailing, Windy, Foggy, Icy, and None. The 'Value (optional)' column is empty for all members. At the bottom, there are two checkboxes: 'Set Flags attribute' and 'Reference external type', both of which are unchecked. Below these is a text box containing 'Namespace.Type'. At the bottom right are 'OK' and 'Cancel' buttons.

Member Name	Value (optional)
Sunny	
Cloudy	
PartlyCloudy	
Rainy	
Snowy	
Sleetng	
Stormy	
Lightning	
Thunderous	
Hailing	
Windy	
Foggy	
Icy	
None	

Figure 2: Enum-ul 'Weather'

5 Baza de date

Pentru a începe modelarea am creat un nou 'ADO.NET Entity Data Model' cu 'Empty EF Designer model', iar apoi, după ce am terminat cu modelarea, pentru a genera baza de date, am folosit 'Generate Database from Model...' din 'Model Browser'. Ca server de bază de date am folosit SqlServer, iar pentru managementul bazei de date am folosit SSMS.

Maparea datelor o puteți vedea în următoarele imagini:

Maps to Multimedia		
<Add a Condition>		
Column Mappings		
Id : int	↔	Id : Int32
Path : nvarchar	↔	Path : String
Description : nvarchar	↔	Description : String
Event : nvarchar	↔	Event : String
Weather : tinyint	↔	Weather : MyPhotos.Weather
DateCreated : datetime	↔	DateCreated : DateTime
DateAdded : datetime	↔	DateAdded : DateTime
DateModified : datetime	↔	DateModified : DateTime
Deleted : bit	↔	Deleted : Boolean
AdditionalLabels : nvarchar(max)	↔	AdditionalLabels : String
<Add a Table or View>		

Figure 3: Maparea pentru entitatea 'Multimedia'

Maps to Locations		
<Add a Condition>		
Column Mappings		
Id : int	↔	Id : Int32
Name : nvarchar	↔	Name : String
Scenery : nvarchar	↔	Scenery : String
ZipCode : nvarchar	↔	ZipCode : String
Longitude : decimal	↔	Longitude : Decimal
Latitude : decimal	↔	Latitude : Decimal
Multimediald : int	↔	Multimediald : Int32
<Add a Table or View>		

Figure 4: Maparea pentru entitatea 'Location'

Maps to People		
<Add a Condition>		
Column Mappings		
Id : int	↔	Id : Int32
FirstName : nvarchar	↔	FirstName : String
LastName : nvarchar	↔	LastName : String
Age : smallint	↔	Age : Int16
Multimediald : int	↔	Multimediald : Int32
<Add a Table or View>		

Figure 5: Maparea pentru entitatea 'Person'

Maps to Photos		
<Add a Condition>		
Column Mappings		
Id : int	↔	Id : Int32
X : int	↔	X : Int32
Y : int	↔	Y : Int32
Width : int	↔	Width : Int32
Height : int	↔	Height : Int32
Person_Id : int	↔	
<Add a Table or View>		

Figure 6: Maparea pentru entitatea 'Photo'

6 Clasele POCO și fișierul SQL generat

S-au generat automat clasele POCO, cât și contextul modelului. Acestea au fost nemodificate, adică lăsate așa cum au fost generate, pe durata dezvoltării aplicației. De asemenea, fișierul SQL rezultat a ajutat fără probleme la generarea bazei de date.

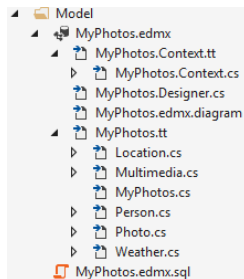


Figure 7: Modelul generat, așa cum este văzut din Visual Studio

7 API

API-ul este un Class Library și este cel care oferă o platformă de interogare a bazei de date și care oferă rezultatele aplicației printr-un pattern foarte bine cunoscut și anume **Unit Of Work** (învățat semestrul trecut la opționalul de .NET). Deoarece modelul conține mai multe entități am optat atât pentru **Unit Of Work**, cât și **Repository Pattern**. Practic, pentru că domeniul de entități este foarte mic, iar operațiile sunt de bază, API-ul va avea și el o funcționalitate destul de simplă.

Structura API-ului este simplă: un folder cu 'repositories', un folder cu 'unit of work', un folder cu 'factories' și un fișier .cs denumit API. Clasa 'Repository' este o clasă din care clasele concrete moștenesc funcționalitățile de bază. Acestea se rezumă la adăugarea, ștergerea, modificarea entităților din context. Această clasă la rândul ei implementează interfața 'IRepository'. Fiecare 'repository' concret implementează de asemenea interfețele specifice lor (însă în cazul nostru, acestea nu conțin funcționalități în plus deoarece n-a fost nevoie). Am optat pentru interfețe pentru a modulariza codul și a îndeplini cât mai bine principiile **SOLID**.

8 Încheiere

Subsemnatul **Maxim Paul** declar pe propria raspundere ca acest cod nu a fost copiat din Internet sau din alte surse. Pentru documentare am folosit urmatoarele surse:

Cărți: -

Link-uri:

- <https://profs.info.uaic.ro/~iasimin/Laborator%20C%20S%20H/Laborator1-2016.pdf>
- https://profs.info.uaic.ro/~iasimin/csharp/C13_14_2012_Entity_Framework.pdf
- <https://www.entityframeworktutorial.net/model-first-with-entity-framework.aspx>
- <https://www.c-sharpcorner.com/UploadFile/abhikumarvatsa/model-first-approach-in-entity>
- <https://www.c-sharpcorner.com/UploadFile/b1df45/unit-of-work-in-repository-pattern/>
- <https://www.programmingwithwolfgang.com/repository-and-unit-of-work-pattern/>
- <https://en.wikipedia.org/wiki/SOLID>