**EntityFrameworkSample\_Completed**

---------------------------------------------------------

Тъй като при първо стартиране нямаме картинки за ресторанти, може да свалим няколко предварително.

Примерът е средно голям и съдържа различни функционалности, които няма да бъдат описани подробно тук. Но все пак ще се опитам да обясня как да го стартираме.

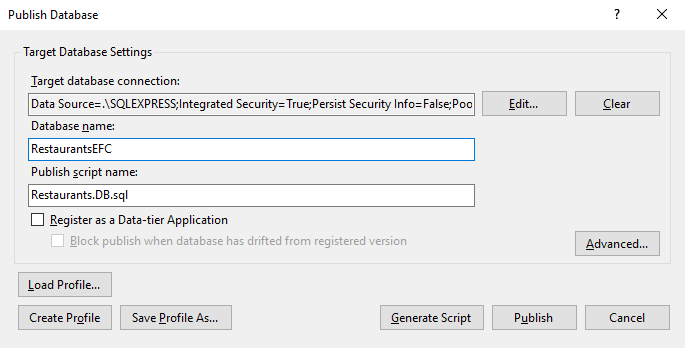
Винаги когато отваряме ново приложение започваме от това да разгледаме базата данни, за да знаем какви данни ще съхраняваме. В примера има проект Restaurants.DB (който е обяснаван как се прави) и може да се ползва за създаване на базата данни ако няма такава.

----------------------------------------------------------------------------------------------------------------------------

Да разгледаме варианта в който нямаме база данни. Ще създадем такава с проекта Restaurants.DB и затова го отваряме.

Даваме десен бутон на проекта и избираме Publish.

От там си избираме нашия сървър : Target database connection и пишем име на новата база данни. Напрмер за име задаваме RestaurantsEFC.



Да обърнем внимание, че с Edit сме избрали нашия сървър за бази данни.

Обикновено Data Source може да бъде в някои случаи само:

**.** –което е localhost или името на нашия хост

или в други случаи:

.\SQLEXPRESS

Когато не сме сигурни кое от двете да използваме, може да отворим SQL Server Managment studio и да копираме полето Server name. За пример полето може да бъде:

DESCTOP-0EI6G\SQLEXPRESS и можем да използваме просто .\SQLEXPRESS

А друг вариант е да е само с едно име DESCTOP-0EI6G и тогава ще използваме само .

---

Преди да изберем Publish е добре да проверим дали Target platform от нашия проект съответства на версията на SQL сървъра. Това може да се провери от : десен бутон на проекта Restaurants.DB-> Properties->Project Settings->Target platform.

Избираме Publish и след извесно време ще бъде създадена базата данни + 3 таблици. Тъй като в проекта Restaurants.DB има и PostDeploymentScript.sql , той ще се изпълни и таблиците ще бъдат попълнени с зададените от този скрипт стойности. Може да се проверим попълнените таблици в SQL Server Managment studio.

Ако има проблем при Publish ще се изведат грешки и трябва отново да проверим дали правилно сме задали пътя до сървъра.

---------------------------------------------------------------------------------------------------

След като създадем база данни ще разгледаме проекта DataAccess. Както подсказва името този проект е за връзката към базата данни. За връзка се използва Entity Framework. В предишен проект е описано как се създава такъв проект когато имаме съществуваща база данни.

Тъй като този проект е създаден не на вашия компютър, то вероятно и пътя до сървъра за базата данни няма да е същия. Затова най-важното е да го редактираме правилно. Почти всичко необходимо за връзка към база данни се описва в така наречения connectionString. Той се намира във с App.Config във тага <connectionStrings> и се задава с атрибут със същото име.

Пример:

connectionString="metadata=res://\*/RestaurantsModel.csdl|res://\*/RestaurantsModel.ssdl|res://\*/RestaurantsModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=localhost\SQLEXPRESS;initial catalog=RestaurantsEFC;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient"

Трябва да проверим дали правилно са зададени името на сървъра и името на базата данни която ще използваме. Може да бъдат нещо от типа:

data source=localhost\SQLEXPRESS

initial catalog=RestaurantsEFC

За да проверим дали този проект ще се връзва към базата данни може да направим следното. Отваряме диаграмата RestaurantsModel.edmx и изтриваме от нея таблицата Restaurant. След това с десен бутон избираме Update Model from DataBase като посочим отметка срещу таблици.

Ако всичко е наред с connectionString таблицата Restaurant ще се добави отново към диаграмата.

Това с изтриването на дадена таблица и повторното и добавяне към диаграмата (.edmx) се прави и когато сме променяли някоя таблица (тип на колона, нова колона) от базата и искаме Entity Framework да отрази тези промени в проекта (DataAccess).

// *За да работи уеб проекта EntityFrameworkSample, трябва да направим същата редакция и в неговия файл Web.config.*

Този проект (DataAccess) е много важен и той ни позволява да ползваме базата данни чрез абстракция. Чрез него няма да правим директни SQL заявки, а ще работим с таблиците подобно на обекти. За целта се използва LINQ синтаксис, който ще ви трябва и ще се наложи постепено да го разучавате. Съответно Entity Framework ще превежда LINQ към SQL заявки (които биха изглеждали много по-сложни ако ги пишем сами – в повечето случаи).

Какво друго е генерирал Entity Framework. Това е контекста (специален клас) и него ще използваме когато искаме да правим заявки към базата. Този контекст се намира във файла RestaurantsModel.Context.cs и в случая е RestaurantsEntities : DbContext. Може да проверим, че той се ползва в прокта Repositories и по специално в класа BaseRepository.cs

В този проект (DataAccess) се намират и автоматично генерираните класове за ентититата:

Category.cs, City.cs, Restaurants.cs – това са базовите класове и представляват начина по който Entity Framework „вижда“ таблиците.

В последствие в някой метод, когато искаме да вземем всички категории, ще направим заявка към контекста и той ще ни върне списък от обекти от тип Category.

Примерно:

public List<Category> GetAll()

{

return Context.Set<Category>().ToList();

}

-------------------------------------------------------------------------------------------------------

След като се разгледа проекта DataAccess следва да разгледаме каква функционалност имаме с дадените таблици от базата данни. Тази функционалност е реализирана в проекта Repositories.

Той съдържа класове за работа с обектите (таблиците).

Тъй като и трите вида обекти (Category, City, Restaurants) ще имат методи с еднакви имена и подобна функционалност може да дефинираме един интерфейс IbaseRepository. За да бъде универсален този интерфейс е generic. Generic означава, че може да го ползваме за различни типове (в случая Category, City или Restaurants).

Например да разгледаме един метод от интерфейса:

List<T> GetAll(); - означава метод който ще ни връща списък от тип, който може да си изберем.

Когато имплементираме този интерфейс за класа City ще имаме готов метод:

List< City > GetAll();

Имаме и един базов клас BaseRepository. Той също за удобстово е generic. В този клас са дефинирани всички методи които имат еднотипна функционалност за трите вида обекти (Category, City, Restaurants). И тъй като е generic само задаваме с кой тип да се работи.

В базовия клас е дефиниран и контекста с който ще работят всички останали репозиторита. Както споменахме по-горе той е от DataAccess проекта и е от тип RestaurantsEntities.

Тъй като метода Save() ще има различни дефиниции в наследниците, той трябва да се дефинира като abstract. Тогава наследниците на базовия клас ще са задължени да override –нат този метод, Когато в един клас има поне един абстрактен метод то този клас става абстрктен.

============================================================================

Накрая разглеждаме уеб проекта EntityFrameworkSample.

Задължително и за този проект трябва да коригираме connectionString за да работи коректно.

Обикновенно уеб прокетите се разглеждат в следната последователност:

* Директорията Models за да видим с какви данни (обекти) ще работим. Също и ViewModels – т.е. обектите които ще се подават от контролера към дадено view и обратно.
* Директорията Controllers – за да видим какви действия имаме. Най-често се започва с HomeController
* Директорията Views – за да разгледаме View-тата. Първо се отваря \_Layout и по необходимост \_ViewStart. После с отделните view-та според action-ите на съответните контролери.

//

Можем да стартираме проекта и да отидем на Restaurants и с бутона Edit да редактираме някои от тях като им наслагаме картинки.

//

--------------------------

Нека разгледаме някои от функционалностите.

Обикновенно за да се ориентираме отваряме \Views\Shared\\_Layout.cshtml за да разгледаме какви менюта имаме. Например може да разгледаме линка:

@Html.ActionLink("Categories", "Index", "Category")

Ако изберем него ще ни отведе до action-а Index в конторлера Category.

Затова отваряме CategoryController и разглеждаме action-а Index.

Тъй като в index-а на категориите искаме да изведем всички, то ще си създадем списък от категории.

Използваме обект CategoryViewModel който ще съдържа информация за една категория. Ние попълваме списък със всички категории и този списък ще го подадем към строго типизирано View (т.е. има за модел този списък от обекти CategoryViewModel) за да бъде визуализиран там.

!! Много е важно да обърнем внимание, че към View-то се подават обекти от типа “CategoryViewModel”, а не от базовия тип Category въпреки че ще върши подобна работа. Това се прави за разграничаване на View модела!!!

Във View-то \Views\Category\Index.cshtml виждаме че с един цикъл foreach се извеждат всички категории като за всяка има по два линка които са всъщност бутони (bootstrap: имат @class = "btn.....“)

Нека сме си избрали бутона:

@Html.ActionLink("Edit", "Edit", "Category", new { id= category.ID }, new { @class = "btn btn-primary btn-sm" })

Той ще ни отведе към контролера Category в action-а Edit и ще има подаден параметър id. Http метода е GET. Затова разглеждаме в контролера метода:

public ActionResult Edit(int id = 0)

там се търси категория с това ID чрез метода от репозиторито categoryRepository.GetByID(id)

След като намерим такава категория използваме обект от CategoryViewModel и го подаваме на View-то. View-то носи името като Action-а, т.е. Edit (това става автоматично при създаване на View).

Затова разглеждаме \Views\Category\Edit.cshtml

Това View e отново строго типизирано и работи с подадения му обект (т.е. model) CategoryViewModel.

Тъй като ще може да се редактира дадена категория ние трябва да знаем нейното ID. Него ще го пазим в едно скрито поле : @Html.HiddenFor(model => model.ID)

В това View има форма Html.BeginForm("Edit", "Category") която съдържа

както една текстова кутия за редактиране на името на категорията

@Html.TextBoxFor(model => model.Name)

така и бутон Submit за самата форма. Когато натиснем Submit бутона, модела model (който е с променено Name, но същото ID ) ще се изпрати към контролера Category и намиращия се там метод Edit. Тъй като формата работи с http метод POST разглеждаме:

[HttpPost]

public ActionResult Edit(CategoryViewModel categoryEdit)

Там намираме категорията с това ID, променяме и името и я запазваме. След това препращаме към View-то "Index". Когато искаме да предадем едно съобщение от едно view към друго може да използваме глобалното пропърти TempData което ще е видимо и от друго View. В случая искаме да изпишем че сме добавили нова категория. Това най често се прави когато имаме метод RedirectToAction() и ViewBag се губи.

-----------------------------

Интересно е да се разгледа как е реализиран DropDownList във View-то \Views\Restaurants\Edit.cshtml

От контролера RestaurantController (метода [HttpGet] ... Edit(...)) може да видим че списъка с градовете и категориите се подават през ViewBag