Предния проект (DatabaseRestaurants) демострираше използването на ADO.NET за работа с базата данни. Там заявките се пишеха ръчно.

В този пример ще разгледаме Entity Framework, който ни дава абстракция за работа с базата данни по такъв начин, че работата с таблиците се свежда до работа с класове.

Създаване на нов проект в който ще ползва EntityFramework (EF) и базата е от проекта DatabaseRestaurants\_DB\_Only

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

Създаваме нов, празен MVC проект с името EntityFrameworkSample

Копираме проекта Restaurants.DB в папката на новия solution и го добавяме като съществуващ проект

Проекта Restaurants.DB ще ни послужи да си създадем базата данни ако я нямаме. Това става като от контекстното меню на Restaurants.DB изберем Publish…

* От Target database connections: избираме съществуваща или посочваме нашия сървър (примерно “.” или „.\SQLEXPRESS“)
* От Database name: може да напишем ново, примерно RestaurantsEF и ще създадем нова.

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

Създаваме нов проект DataAccess oт тип Class Library

Добавяме нов Item от тип Data -> “ADO.NET Entity Data Model” с име RestaurantsModel; Избираме EF Designer from database;

В Entity Data Model Wizard:

Which data connection should your application use to connect to the database?

-> New Connection -> Server name: .\SQLEXPRESS,

Select or enter a database name: RestaurantsEF

Вижда се че е дадена отметката „Save connection settings in App.Config as“. Във файлът App.Config ще бъде записан connection string, който ще го копираме после за MVC проекта.

-> Next -> EF 6.x -> Next

Избираме таблиците които искаме да изполваме в проекта си (всички) и избираме отметката “Pluralize or singularize generated object names”

-> Finish (+ 2 OK)

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

Обяснение на edmx диаграмата и показване на генерираните класове

/////////////

Ако след време се правят промени по базата данни и това трябва да се отрази в EF, то трябва да му дадем Update. Важно е да се отбележи, че ако се пряват промени по дадена таблица като смяна на тип (int -> bigint), смяна на дължината на varchar и др., след Update на EF тези промени няма да се отрзят, затова най-лесния начин е да изтрием таблицата от .edmx диаграмата (Delete from Model) и да я добавим чрез Update.

Update на EF се изпълнява чрез: Десен бутон на .edmx диаграмата и избираме “Update Model From Database…”

/////////////

В Category.cs има property от тип ICollection<Restaurant>, защото са 1:n връзка

Разглеждаме и генерирания клас RestaurantsEFEntities

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

Създаваме нов проект Repositories oт тип Class Library

Създаваме нов клас CategoryRepository в новия проект

Добавяме референция в Repositories към DataAccess за да достъпваме класовете му

От tool -> nuget package manager, добавяме EntityFramework към проектите EntityFrameworkSample и Repositories

Добавяме private property: ‘private RestaurantsEntities Context’

// Тъй като базата ни данни се казва RestaurantsEF, в класа CategoryRepository, контекста ще бъде RestaurantsEFEntities

Създаваме default constructor и там инициираме стойността на Context

Добавяме CRUD методи които ще ни трябват и ги разписаме:

public class CategoryRepository

{

private RestaurantsS1Entities Context;

public CategoryRepository()

{

Context = new RestaurantsS1Entities();

}

public List<Category> GetAll()

{

return Context.Categories.ToList();

}

public Category GetByID(int id)

{

return Context.Categories.Find(id);

}

public void Create(Category category)

{

Context.Categories.Add(category);

Context.SaveChanges();

}

public void Update(Category category)

{

var local = Context.Set<Category>()

.Local //access the date that is currently

//in-memory using the DbSet.Local property

.FirstOrDefault(f => f.ID == category.ID);

if (local != null)

{

Context.Entry(local).State = EntityState.Detached;

}

Context.Entry(category).State = EntityState.Modified;

//Context.Entry(category).State = EntityState.Modified;

//var entry = Context.Entry(category);

//Context.Categories.Attach(category);

//entry.State = EntityState.Modified;

Context.SaveChanges();

}

public void DeleteByID(int id)

{

Category category = Context.Categories.Find(id);

if (category != null)

{

Context.Categories.Remove(category);

Context.SaveChanges();

}

}

}

Накрая трябва да билднем проектите DataAccess и Repository (десен бутон на проекта и избираме Build).

Добавяме нов контролер CategoryController в EntityFrameworkSample и го правим той да се зарежда по подразбиране (променяме в App\_Start/RouteConfig.cs)

Копираме connectionstring-a от DataAccess/App.Config в EntityFrameworkSample /Web.Config

Connection string-а може да го сложим в секцията <configuration> след затварящия таг </configSections>

Connection string-а се копира заедно с:

<connectionStrings>

<add name="RestaurantsEFEntities" connectionString="........" providerName="System.Data.EntityClient" />

</connectionStrings>

Тестваме само с debug (F5) дали работят новите методи и проверка в базата дали има промени.

Пробваме и когато липсва connection string.

public class CategoryController : Controller

{

// GET: Category

public ActionResult Index()

{

CategoryRepository categoryRepository =

new CategoryRepository();

var x = categoryRepository.GetAll();

Category y = categoryRepository.GetByID(1);

categoryRepository.DeleteByID(9);

Category y2 = new Category();

y2.ID = 8;

y2.Name = "new name 111111";

categoryRepository.Update(y2);

return View();

}

}

* //не Създаване на Index View за визуализиране на всички категории извлечени с GetAll()