

در پست های قبلی ([^](#) و [^](#)) با template و ساخت کنترلر و مدل در پروژه های F# MVC آشنا شدید. در این پست به طراحی Repository با استفاده از EntityFramework خواهیم پرداخت. در ادامه مثال قبل، برای تامین داده های مورد نیاز کنترلرها و نمایش آنها در View نیاز به تعامل با پایگاه داده وجود دارد. در نتیجه با استفاده از الگوی Repository، داده های مورد نظر را تامین خواهیم کرد. به صورت پیش فرض با نصب Template جاری (F# MVC4) تمامی اسمبلی های مورد نیاز برای استفاده از EF در پروژه های F# نیز نصب می شود.

پیاده سازی DbContext مورد نیاز

برای ساخت DbContext می توان به صورت زیر عمل نمود:

```
namespace FsWeb.Repositories
open System.Data.Entity
open FsWeb.Models

type FsMvcAppEntities() =
    inherit DbContext("FsMvcAppExample")

    do Database.SetInitializer(new CreateDatabaseIfNotExists<FsMvcAppEntities>())

    [<DefaultValue(>)] val mutable books: IDbSet<Guitar>
    member x.Books with get() = x.books and set v = x.books <- v
```

همان طور که ملاحظه می کنید با ارث بری از کلاس DbContext و پاس دادن Connection String یا نام آن در فایل app.config، به راحتی FsMVCAppEntities ساخته می شود که معادل DbContext پروژه مورد نظر است. با استفاده از دستور do متد SetInitializer برای عملیات migration فراخوانی می شود. در پایان نیز یک DbSet به نام Books ایجاد کردیم. فقط از نظر syntax با حالت C# آن تفاوت دارد اما روش پیاده سازی مشابه است.

اگر syntax زبان F# برایتان نامفهوم است می توانید از این [دوره](#) کمک بگیرید.

پیاده سازی کلاس BookRepository

ابتدا به کدهای زیر دقت کنید:

```
namespace FsWeb.Repositories
type BooksRepository() =
    member x.GetAll () =
        use context = new FsMvcAppEntities()
        query { for g in context.Books do
            select g }
        |> Seq.toList
```

در کد بالا ابتدا تابعی به نام GetAll داریم. در این تابع یک نمونه از DbContext پروژه و هله سازی می شود. نکته مهم این است به جای شناسه let از شناسه use استفاده کردم. شناسه use دقیقاً معادل دستور using(){} در C# است. بعد از اتمام عملیات شی مورد نظر Dispose خواهد شد.

در بخش بعدی یک کوئری از DbSet مورد نظر گرفته می شود. این روش Query گرفتن در F# 3.0 مطرح شده است. در نتیجه در نسخه های قبلی آن (F# 2.0) اجرای این کوئری باعث خطا می شود. اگر قصد دارید با استفاده از F# 2.0 کوئری های خود را ایجاد نماید باید به طریق زیر عمل نمایید:

ابتدا از طریق nuget اقدام به نصب package ذیل نمایید:

```
FSPowerPack.Linq.Community
```

سپس در ابتدا Source File خود، فضای نام Microsoft.FSharp.Linq.Query را باز(استفاده از دستور open) کنید. سپس می‌توانید با اندکی تغییر در کوئری قبلی خود، آن را در F# 2.0 اجرا نمایید.

```
query <@ seq { for g in context.Books -> g } @> |> Seq.toList
```

حال باید Repository طراحی شده را در کنترلر مورد نظر فراخوانی کرد. اما اگر کمی سلیقه به خرج دهیم به راحتی می‌توان با استفاده از [تزریق وابستگی](#) ، BookRepository را در اختیار کنترلر قرار داد. همانند کد ذیل:

```
[<HandleError>]
type BooksController(repository : BooksRepository) =
    inherit Controller()
    new() = new BooksController(BooksRepository())
    member this.Index () =
        repository.GetAll()
        |> this.View
```

در کدهای بالا ابتدا وابستگی به BookRepository در سازنده BookController تعیین شد. سپس با استفاده از سازنده پیش فرض، یک وهله از وابستگی مورد نظر ایجاد و در اختیار سازنده کنترلر قرار گرفت(همانند استفاده از کلمه this در سازنده کلاس‌های C#). با فراخوانی تابع GetAll داده‌های مورد نظر از database تامین خواهد شد.

نکته : تنظیمات مربوط به ConnectionString را فراموش نکنید:

```
<add name="FsMvcAppExample"
    connectionString="YOUR CONNECTION STRING"
    providerName="System.Data.SqlClient" />
```

موفق باشید.

نظرات خوانندگان

نویسنده: Ara

تاریخ: ۱۳۹۲/۱۲/۲۰ ۱:۴

سلام

با تشکر از زحمات شما

به نظر من استفاده از F# در کنار C# به عنوان Library برای حل مسائل خاص خیلی میتونه مفید باشه

اگه ممکنه در مورد صورت مسئله و راه حل های ارائه شده با F# بیشتر بنویسید تا بیشتر مورد استفاده قرار بگیره ،و خیلی کاربردی تر موضوع رو ببینیم

ممنون