بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

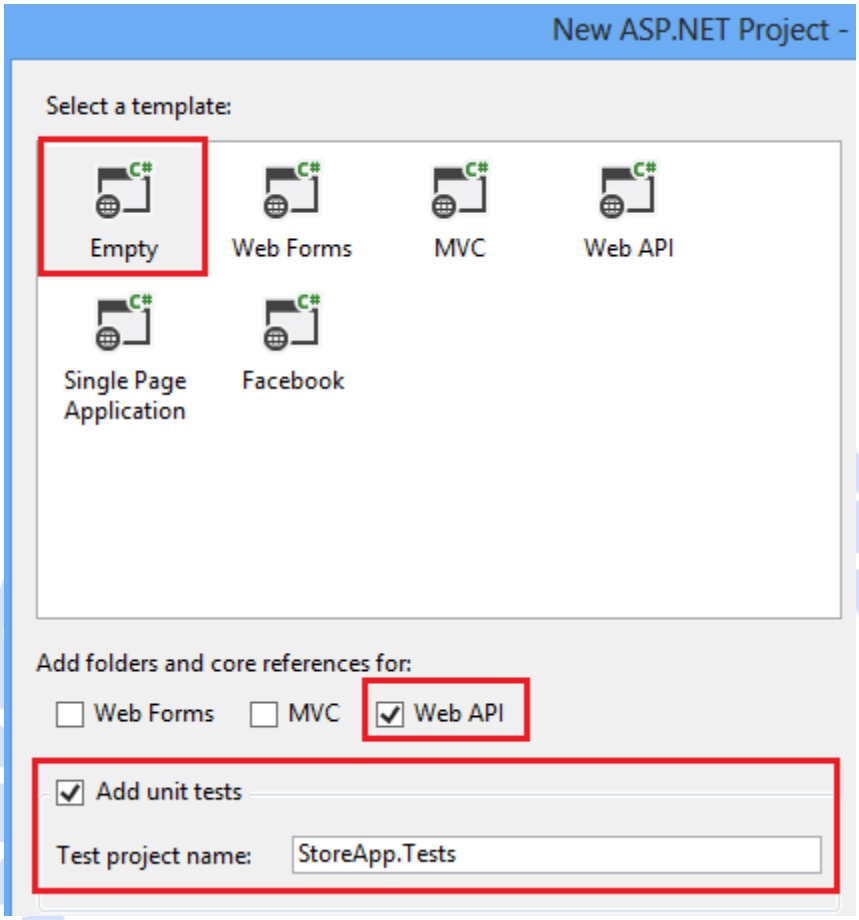
Mock کردن Entity Framework در زمان unit Test کردن ASP.NET Web API 2

مدرس : مهندس افشین رفوآ

دوره آموزشی Web API

ساختن یک برنامه با unit test project

یک برنامه Web application به اسم StoreApp بسازید. در پنجره New Asp.net Project روی قالب Empty کلیک کنید و در Add folders and core references گزینه Web API را انتخاب کنید. گزینه Add unit tests را انتخاب کنید. Unit test project به صورت خودکار نام آن را StoreApp.Tests قرار می دهد و شما می توانید با همین نام ادامه دهید.



بعد از ساخت برنامه، شما ۲ پروژه را می بینید که یکی StoreApp و دیگری StoreApp.Tests می باشد.

ساختن یک کلاس model

در پروژه StoreApp، یک کلاس به فولدر Models به اسم Product.cs اضافه کنید. محتوای فایل را با کد های زیر عوض کنید.

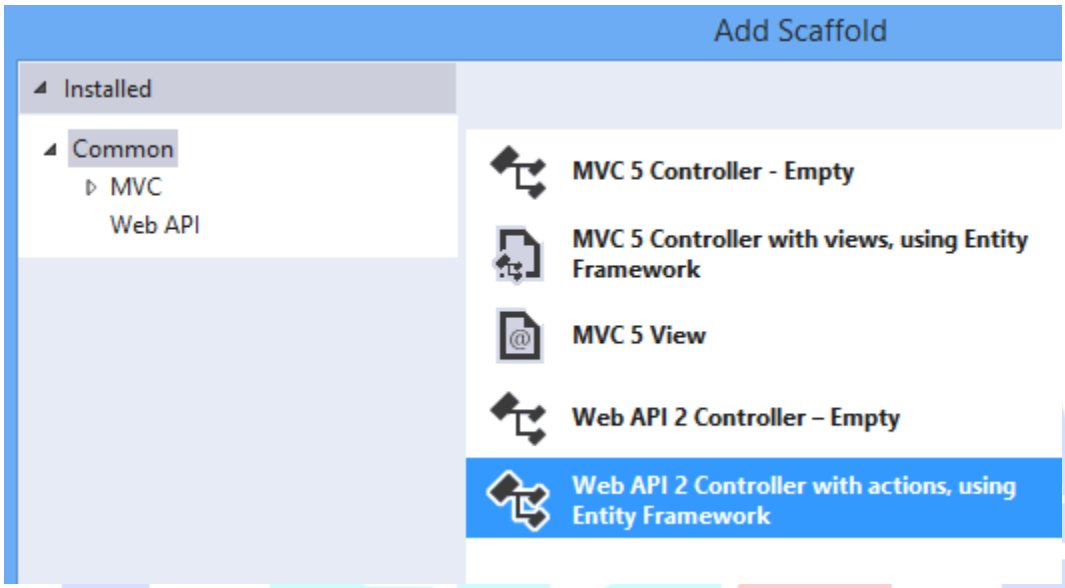
```
using System;
namespace StoreApp.Models
{
    public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public decimal Price { get; set; }
    }
}
```

برنامه را Build کنید.

اضافه کردن Controller

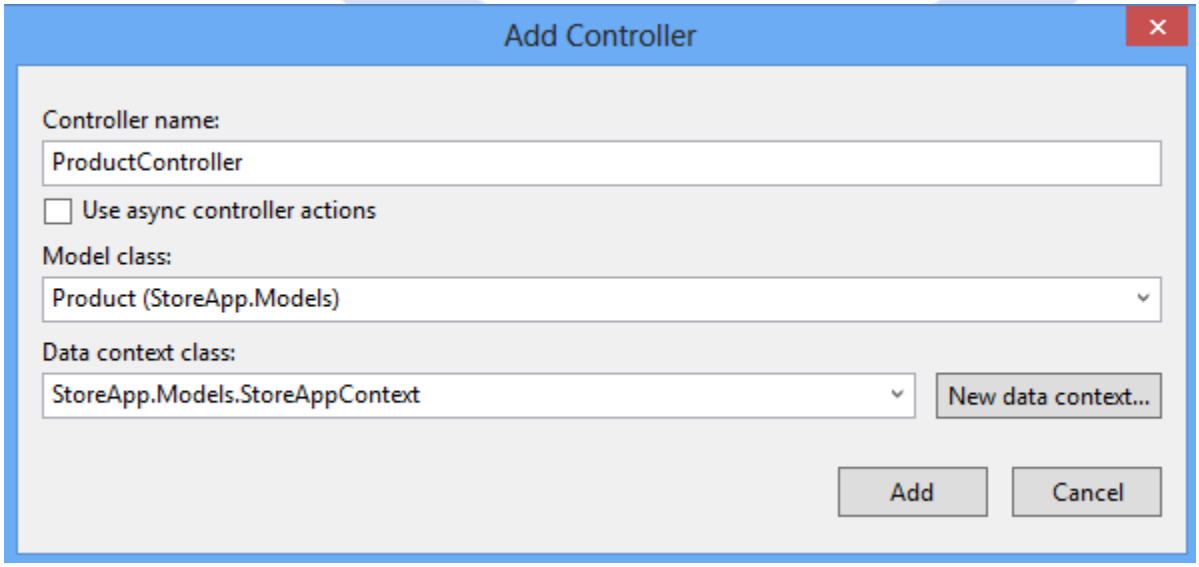
روی فولدر Controllers کلیک راست کنید و ابتدا Add و سپس New Scaffolded Item را انتخاب کنید.

Web API 2 Controller with actions, using Entity Framework را انتخاب کنید.



مقادیر زیر را تنظیم کنید:

- نام Controller: ProductController
- کلاس model: Product
- کلاس Data Context: دکمه New data context را کلیک کنید.



برای ساخت خودکار controller روی Add کلیک کنید. این کد شامل متدهای ساختن، بازیابی کردن، به روز رسانی و حذف کردن مقادیر کلاس Product می شود. کدهای زیر متدی برای اضافه کردن یک Product نشان می دهد. توجه کنید که این متد یک مورد IHttpActionResult بر می گرداند.

```
// POST api/Product
[ResponseType(typeof(Product))]
public IHttpActionResult PostProduct(Product product)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    db.Products.Add(product);
    db.SaveChanges();
    return CreatedAtRoute("DefaultApi", new { id = product.Id }, product);
}
```

IHttpActionResult یکی از ویژگی های جدید در Web API 2 می باشد و باعث ساده تر شدن فرآیند unit test می شود. در قسمت بعدی کد های ساخته شده را برای ساده تر کردن سفارشی سازی عملیات test به Controller اضافه می کنیم.

اضافه کردن dependency injection

در حال حاضر، کلاس ProductController برای استفاده از یک کلاس StoreAppContext سخت رمزگذاری(hard-coded) شده است. شما می توانید از الگوی dependency injection برای تغییرات برنامه خود و حذف کردن hard-coded dependency استفاده کنید. با شکسته شدن این dependency، شما می توانید object ها را در زمان test کردن mock کنید.

روی فولدر models کلیک راست کنید و یک رابط جدید به نام IStoreAppContext اضافه کنید.

کد های آن را به کد های زیر عوض کنید:

```
namespace StoreApp.Models
{
    public interface IStoreAppContext : IDisposable
    {
        DbSet<Product> Products { get; }
        int SaveChanges();
        void MarkAsModified(Product item);
```

```
    }
}
```

فایل StoreAppContext.cs را باز کنید و مقادیر مشخص شده را تغییر دهید. مهمترین تغییراتی که باید صورت گیرد عبارت اند از:

- کلاس StoreAppContext رابط IStoreAppContext را پیاده سازی کند.
- متد MarkAsModified پیاده سازی شود.

```
using System;
using System.Data.Entity;
using System;
using System.Data.Entity;
namespace StoreApp.Models
{
    public class StoreAppContext : DbContext, IStoreAppContext
    {
        public StoreAppContext()
            : base("name=StoreAppContext")
        {
        }
        public DbSet<Product> Products { get; set; }
        public void MarkAsModified(Product item)
        {
            Entry(item).State = EntityState.Modified;
        }
    }

  }
```

فایل ProductController.cs را باز کنید.قسمت های مشخص شده در کد زیر را با کد های درون فایل عوض کنید. این تغییرات ، dependency را در StoreAppContext می شکند و کلاس های دیگر را برای گذاشتن در یک object متفاوت برای کلاس context فعال می سازد.

```
public class ProductController : ApiController
{
    // modify the type of the db field
    private IStoreAppContext db = new StoreAppContext();
    // add these contructors
    public ProductController() { }
    public ProductController(IStoreAppContext context)
    {
        db = context;
    }
    // rest of class not shown
}
```

یک تغییر دیگر باید در ProductController انجام دهید. در متد PutProduct ، خطی که حالت موجودیت (entity state) را به (modified) تغییر داده با متد MarkAsModified عوض می کنیم.

```
// PUT api/Product/5
public IHttpActionResult PutProduct(int id, Product product)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    if (id != product.Id)
    {
        return BadRequest();
    }
    //db.Entry(product).State = EntityState.Modified;
    db.MarkAsModified(product);
    // rest of method not shown
}
```

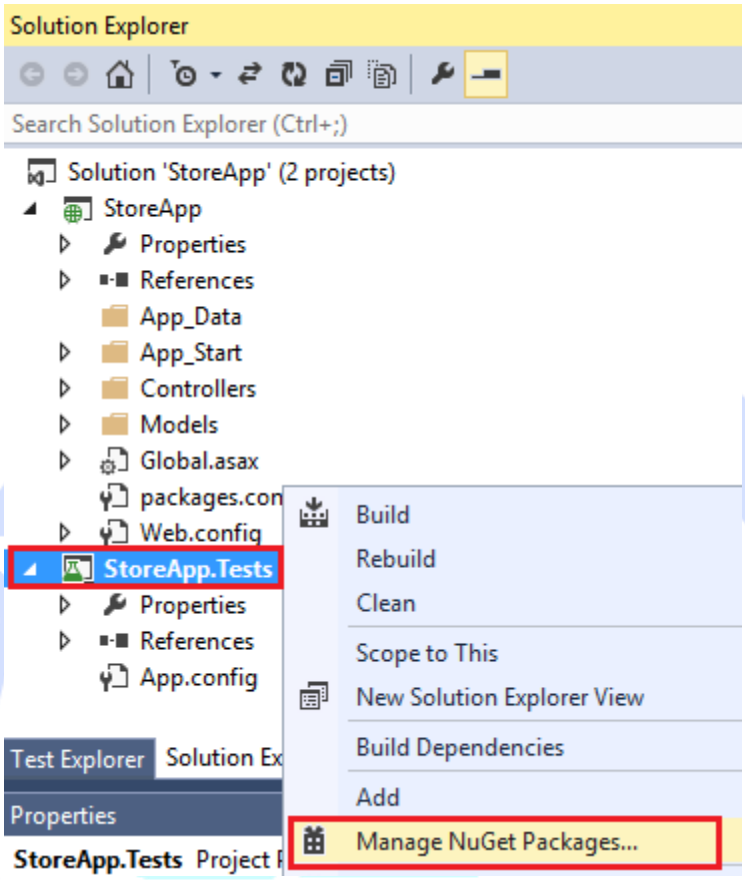
برنامه را build کنید.

حالا برنامه شما برای تنظیمات test project آماده است.

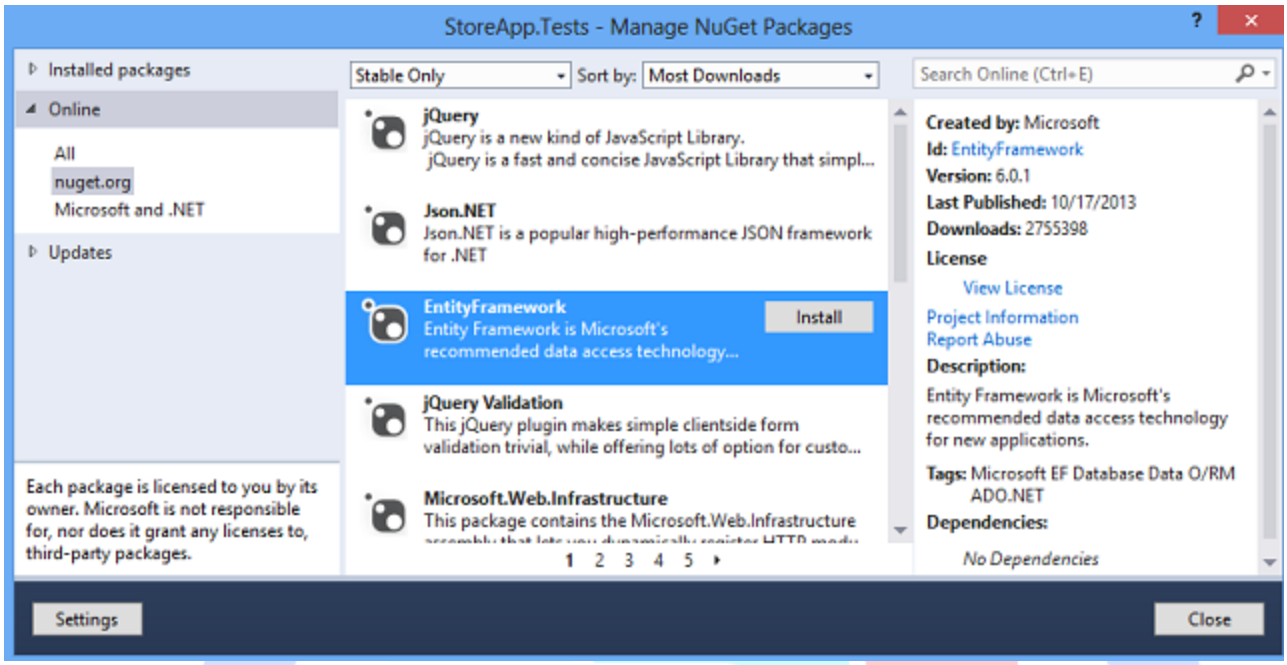
نصب بسته های NuGet در test project

وقتی از قالب empty برای ساختن یک برنامه استفاده کنید، unit test مربوطه (StroreApp.Tests) بسته های نصب شده ی NuGet را شامل نمی شود. قالب های دیگر مثل Web API Template شامل بعضی بسته های NuGet در unit test می شوند. برای این مقاله، شما حتما باید بسته های Entity Framework و Microsoft ASP.NET Web API 2 Core را به test project اضافه کنید.

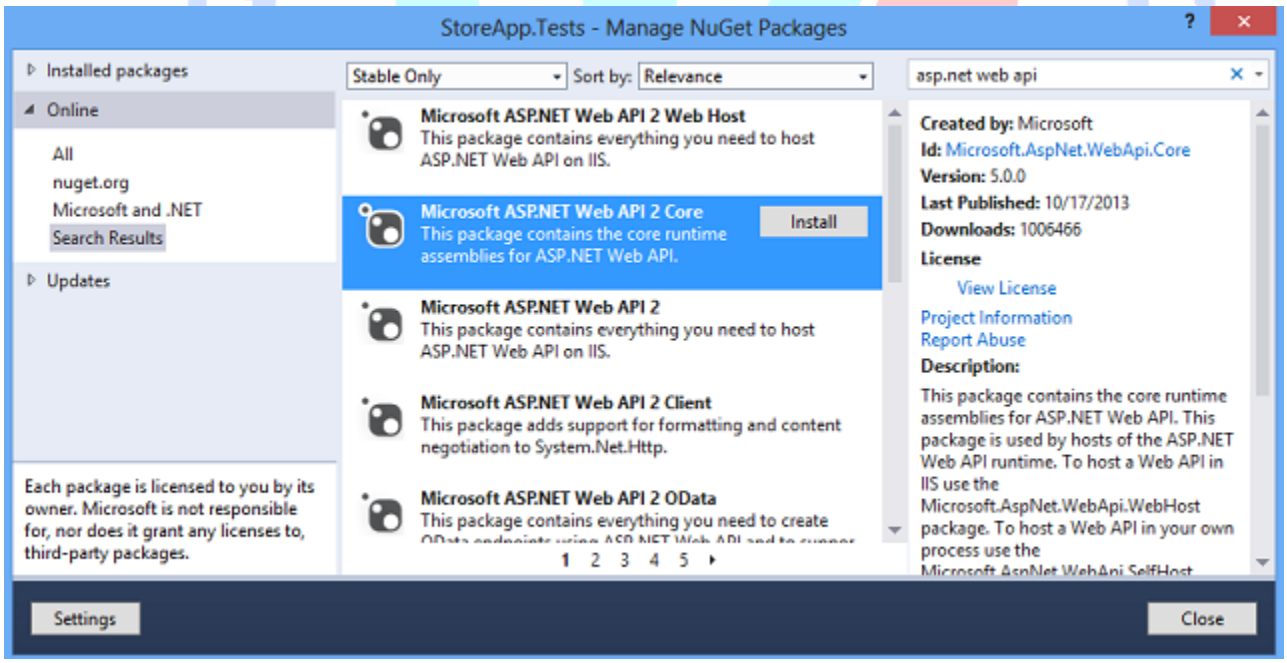
روی StoreApp.Tests کلیک راست کنید و Manage NuGet Packages را انتخاب کنید. شما باید برای اضافه کردن بسته ها به پروژه StoreApp.Tests را انتخاب کنید.



در قسمت Online مطابق تصویر زیر، بسته EntityFramework را انتخاب کنید. اگر پیامی مبنی بر اینکه بسته EntityFramework در حال خاضر نصب می باشد نمایش داد، شما باید StoreApp را به جای StoreApp.Tests انتخاب کنید.



بسته Microsoft ASP.NET Web API 2 Core را پیدا و نصب کنید.



پنجره Manage NuGet Packages را ببندید.

ساخت محتوا (context) test

کلاسی به نام TestDbSet به test project اضافه کنید. این کلاس به عنوان کلاس اصلی test data شما قرار می گیرد. کد های درون آن را با کد های زیر عوض کنید.

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Data.Entity;
using System.Linq;
namespace StoreApp.Tests
{
    public class TestDbSet<T> : DbSet<T>, IQueryable, IEnumerable<T>
        where T : class
    {
        ObservableCollection<T> _data;
        IQueryable _query;
        public TestDbSet()
        {
            _data = new ObservableCollection<T>();
            _query = _data.AsQueryable();
        }
        public override T Add(T item)
        {
            _data.Add(item);
            return item;
        }
        public override T Remove(T item)
        {
            _data.Remove(item);
            return item;
        }
        public override T Attach(T item)
        {
            _data.Add(item);
            return item;
        }
        public override T Create()
        {
            return Activator.CreateInstance<T>();
        }
        public override TDerivedEntity Create<TDerivedEntity>()
        {
            return Activator.CreateInstance<TDerivedEntity>();
        }
        public override ObservableCollection<T> Local
        {
            get { return new ObservableCollection<T>(_data); }
        }
        Type IQueryable.ElementType
        {
            get { return _query.ElementType; }
        }
        System.Linq.Expressions.Expression IQueryable.Expression
        {
            get { return _query.Expression; }
        }
```

```
        IQueryProvider IQueryable.Provider
        {
            get { return _query.Provider; }
        }
        System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
        {
            return _data.GetEnumerator();
        }
        IEnumerator<T> IEnumerable<T>.GetEnumerator()
        {
            return _data.GetEnumerator();
        }
    }
}
```

کلاسی با نام TestProductDbSet درون test project بسازید و کد های زیر را به آن اضافه کنید:

```
using System;
using System.Linq;
using StoreApp.Models;
namespace StoreApp.Tests
{
    class TestProductDbSet : TestDbSet<Product>
    {
        public override Product Find(params object[] keyValues)
        {
            return this.SingleOrDefault(product => product.Id ==
(int)keyValues.Single());
        }
    }
}
```

کلاسی با نام TestStoreAppContext اضافه کنید و کد های داخل آن را با کد های زیر عوض کنید:

```
using System;
using System.Data.Entity;
using StoreApp.Models;
namespace StoreApp.Tests
{
    public class TestStoreAppContext : IStoreAppContext
    {
        public TestStoreAppContext()
        {
            this.Products = new TestProductDbSet();
        }
        public DbSet<Product> Products { get; set; }
        public int SaveChanges()
        {
            return 0;
        }
        public void MarkAsModified(Product item) { }
        public void Dispose() { }
    }
}
```

ساختن test ها

به صورت پیش فرض، project test شامل یک فایل خالی به نام UnitTest1.cs می باشد. این فایل attribute هایی که شما برای ساخت متد های test استفاده می کنید نشان می دهد. برای این مقاله، شما می توانید این فایل را حذف کنید زیرا شما یک کلاس test جدید می سازید.

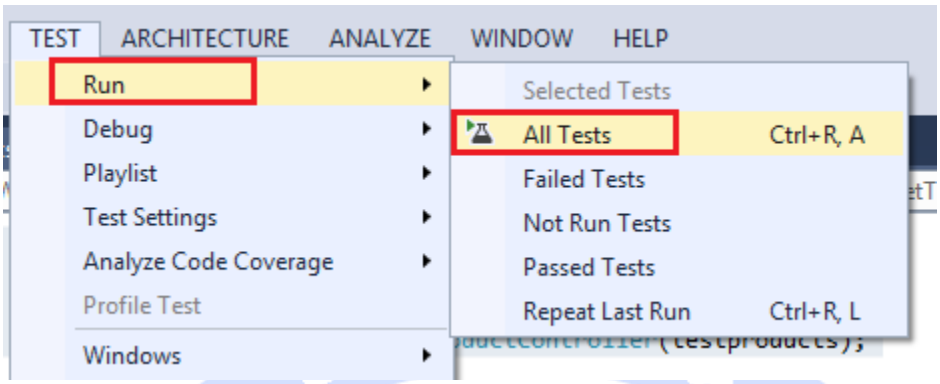
کلاسی با نام testProductController به test project اضافه کنید و کد های زیر را با آن عوض کنید.

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Web.Http.Results;
using System.Net;
using StoreApp.Models;
using StoreApp.Controllers;
namespace StoreApp.Tests
{
    [TestClass]
    public class TestProductController
    {
        [TestMethod]
        public void PostProduct_ShouldReturnSameProduct()
        {
            var controller = new ProductController(new TestStoreAppContext());
            var item = GetDemoProduct();
            var result =
                controller.PostProduct(item) as
CreatedAtRouteNegotiatedContentResult<Product>;
            Assert.IsNotNull(result);
            Assert.AreEqual(result.RouteName, "DefaultApi");
            Assert.AreEqual(result.RouteValues["id"], result.Content.Id);
            Assert.AreEqual(result.Content.Name, item.Name);
        }
        [TestMethod]
        public void PutProduct_ShouldReturnStatusCode()
        {
            var controller = new ProductController(new TestStoreAppContext());
            var item = GetDemoProduct();
            var result = controller.PutProduct(item.Id, item) as StatusCodeResult;
            Assert.IsNotNull(result);
            Assert.IsInstanceOfType(result, typeof(StatusCodeResult));
            Assert.AreEqual(HttpStatusCode.NoContent, result.StatusCode);
        }
        [TestMethod]
        public void PutProduct_ShouldFail_WhenDifferentID()
        {
            var controller = new ProductController(new TestStoreAppContext());
            var badresult = controller.PutProduct(999, GetDemoProduct());
            Assert.IsInstanceOfType(badresult, typeof(BadRequestResult));
        }
        [TestMethod]
        public void GetProduct_ShouldReturnProductWithSameID()
        {
            var context = new TestStoreAppContext();
            context.Products.Add(GetDemoProduct());
```

```
            var controller = new ProductController(context);
            var result = controller.GetProduct(3) as OkNegotiatedContentResult<Product>;
            Assert.IsNotNull(result);
            Assert.AreEqual(3, result.Content.Id);
        }
        [TestMethod]
        public void GetProducts_ShouldReturnAllProducts()
        {
            var context = new TestStoreAppContext();
            context.Products.Add(new Product { Id = 1, Name = "Demo1", Price = 20 });
            context.Products.Add(new Product { Id = 2, Name = "Demo2", Price = 30 });
            context.Products.Add(new Product { Id = 3, Name = "Demo3", Price = 40 });
            var controller = new ProductController(context);
            var result = controller.GetProducts() as TestProductDbSet;
            Assert.IsNotNull(result);
            Assert.AreEqual(3, result.Local.Count);
        }
        [TestMethod]
        public void DeleteProduct_ShouldReturnOK()
        {
            var context = new TestStoreAppContext();
            var item = GetDemoProduct();
            context.Products.Add(item);
            var controller = new ProductController(context);
            var result = controller.DeleteProduct(3) as
OkNegotiatedContentResult<Product>;
            Assert.IsNotNull(result);
            Assert.AreEqual(item.Id, result.Content.Id);
        }
        Product GetDemoProduct()
        {
            return new Product() { Id = 3, Name = "Demo name", Price = 5 };
        }
    }
}
```

اجرا کردن test ها

حالا شما می توانید test ها را اجرا کنید. تمامی متد هایی که با attribute به نام testMethod هستند، آزمایش خواهند شد. از منو test می توانید برنامه را اجرا کنید:



Test Explorer را باز کنید و به نتایج test ها توجه کنید: