

ASP.NET Core Fundamentals

BUILDING YOUR FIRST ASP.NET CORE APPLICATION



Scott Allen

@OdeToCode - <http://OdeToCode.com>



تاریخ شروع: ۹۶/۹/۲

تاریخ پایان: ۹۶/۱۰/۲

گردآوری: سیروان عفیفی

Contents

Building Your First ASP.NET Core Application	5
Introduction.....	5
Setup	5
A New Project.....	6
Command Line Tools and Code	9
The ASP.NET Core Project Structure.....	11
Creating a Greeting Service.....	14
Startup and Middleware	17
Introduction.....	17
How Middleware Works.....	17
Using IApplicationBuilder	18
Showing Exception Details.....	20
Middleware to Match the Environment.....	22
Serving Files.....	25
Setting up ASP.NET MVC Middleware	28
Summary	30
Controllers in the MVC Framework.....	31
The Model View Controller Design Pattern.....	31
Routing	32
Conventional Routes	32
Attribute Routes.....	35
Action Results.....	37
Rendering Views	40
A Table Full of Restaurants.....	42
Models in the MVC Framework	45
Models and View Models	45
Detail a Restaurant.....	48
Create a Restaurant	53

Accepting Form Input	54
POST Redirect GET Pattern	55
Model Validation with Data Annotations	57
Using the Entity Framework.....	63
Introduction.....	63
SQL Server LocalDB	63
Installing the Entity Framework	65
Implementing a DbContext.....	67
Configuring the Entity Framework Services.....	70
Entity Framework Migrations.....	71
Razor Views	73
Layout Views	73
_ViewStart	75
_ViewImports	76
Razor Pages	77
An Edit Form.....	82
Partial Views.....	85
View Components	88
ASP.NET Identity	90
Authentication Services and Middlewares	90
Using the Authorize Attribute	90
User Registration.....	91
Creating a User	95
Log in and Log Out.....	97
Identities and Claims.....	101
Front End Frameworks and Tools	102
Introduction.....	102
Front End Tools	102
Command Line vs. Visual Studio	102

Setting up npm	103
Serving File from node_modules	106
Enabling Client-side Validation	108
Using CDNs and Fallbacks	109

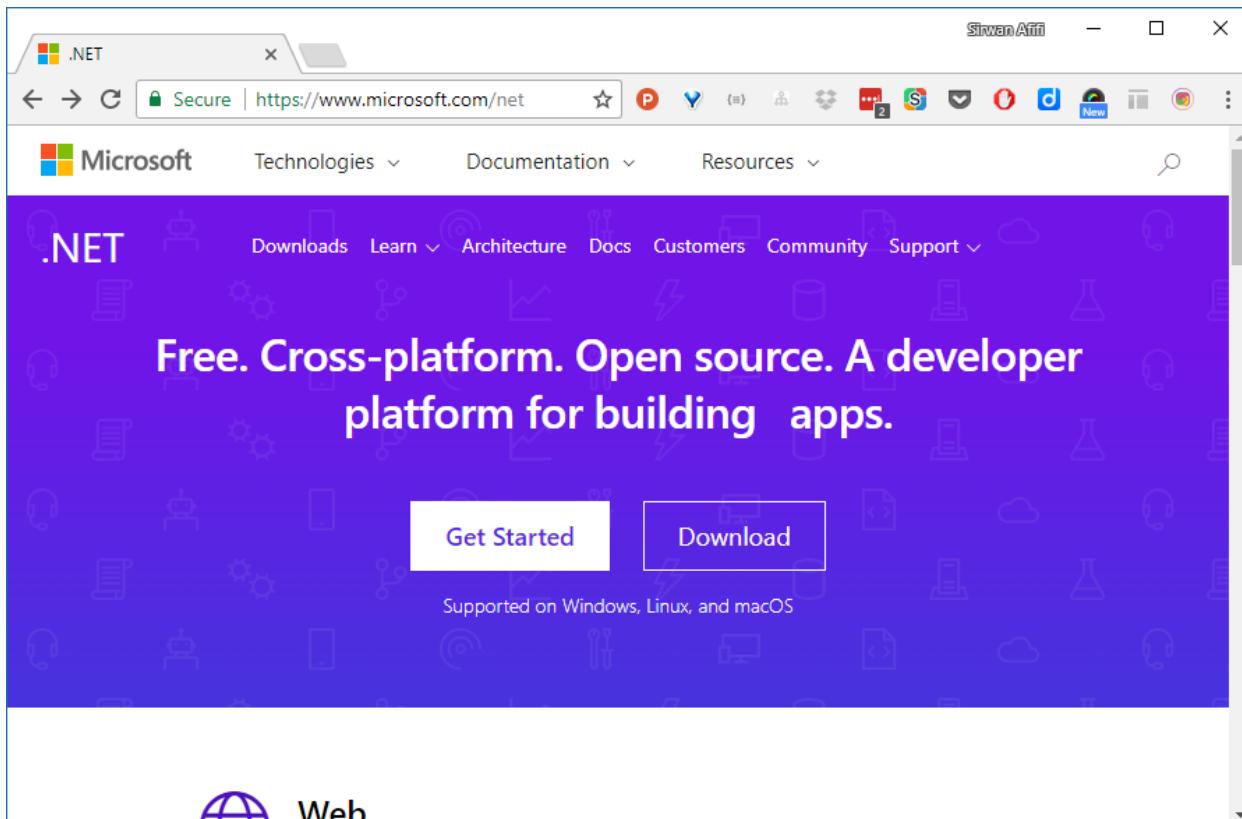
Building Your First ASP.NET Core Application

Introduction

در این دوره به ایجاد یک پروژه با استفاده از ASP.NET Core خواهیم پرداخت. این پروژه قرار است اطلاعات یک رستوران را نمایش دهد.

Setup

برای اولین قدم نیاز است ابتدا مطمئن شویم که .NET Core Framework می‌توانید لیستی از نسخه‌های مختلف .NET Framework را مشاهده نمائید. در صفحه [microsoft.com/net](https://www.microsoft.com/net) موردنظرمان را دانلود کنیم؛ SDK Download حاوی تمامی ابزارهای موردنیاز برای ایجاد یک پروژه .NET می‌باشد:



قدم بعدی انتخاب یک text editor برای نوشتن کدها می‌باشد. در واقع برای پروژه‌های .NET Core از هر ادیتوری می‌توانید استفاده کنید:

A screenshot of a Microsoft Edge browser window displaying the Visual Studio website (<https://www.visualstudio.com>). The page features a hero section with the text "Best-in-class tools for any developer" and a background image of two people working on a computer. Below this, there are two main sections: "Visual Studio IDE" and "Visual Studio Code". Each section includes a brief description, a "Download for Windows" button with a download icon, and a "Get started for free" button with a right-pointing arrow. A "Feedback" button is located on the right side of the page. The top navigation bar includes links for Microsoft, Technologies, Documentation, Resources, Sign in, and a "Free Visual Studio" button.

Visual Studio IDE, Code E

Secure | https://www.visualstudio.com

Microsoft Technologies Documentation Resources Sign in

Visual Studio Products Downloads Marketplace Support Free Visual Studio

Best-in-class tools for any developer

Visual Studio IDE
Rich IDE, advanced debugging

Visual Studio Team Services
Agile tools, Git, continuous integration

[Download for Windows](#) [Get started for free](#)

[Learn More >](#) [Learn More >](#)

Visual Studio Code
Editing and debugging on any OS

Visual Studio App Center
Continuous integration, delivery & learning

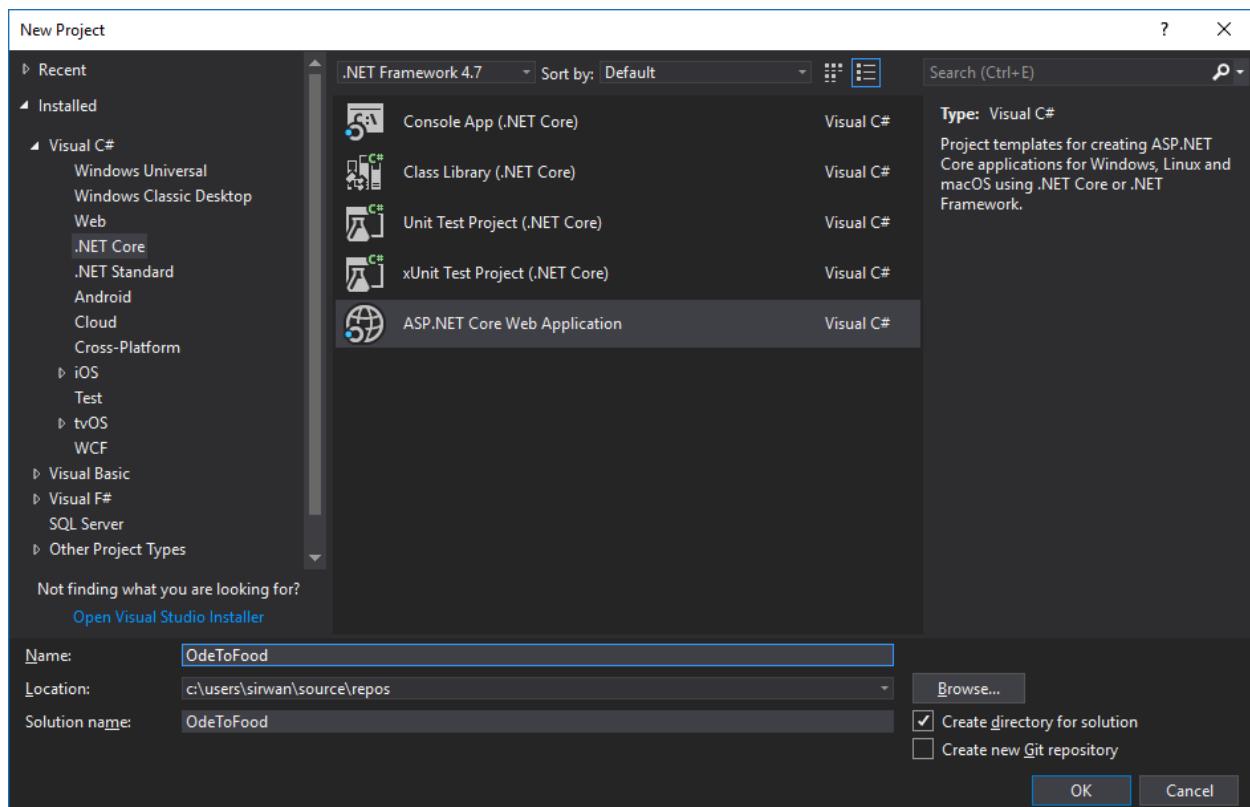
[Download for Windows](#) [Get started for free](#)

[Learn More >](#) [Learn More >](#)

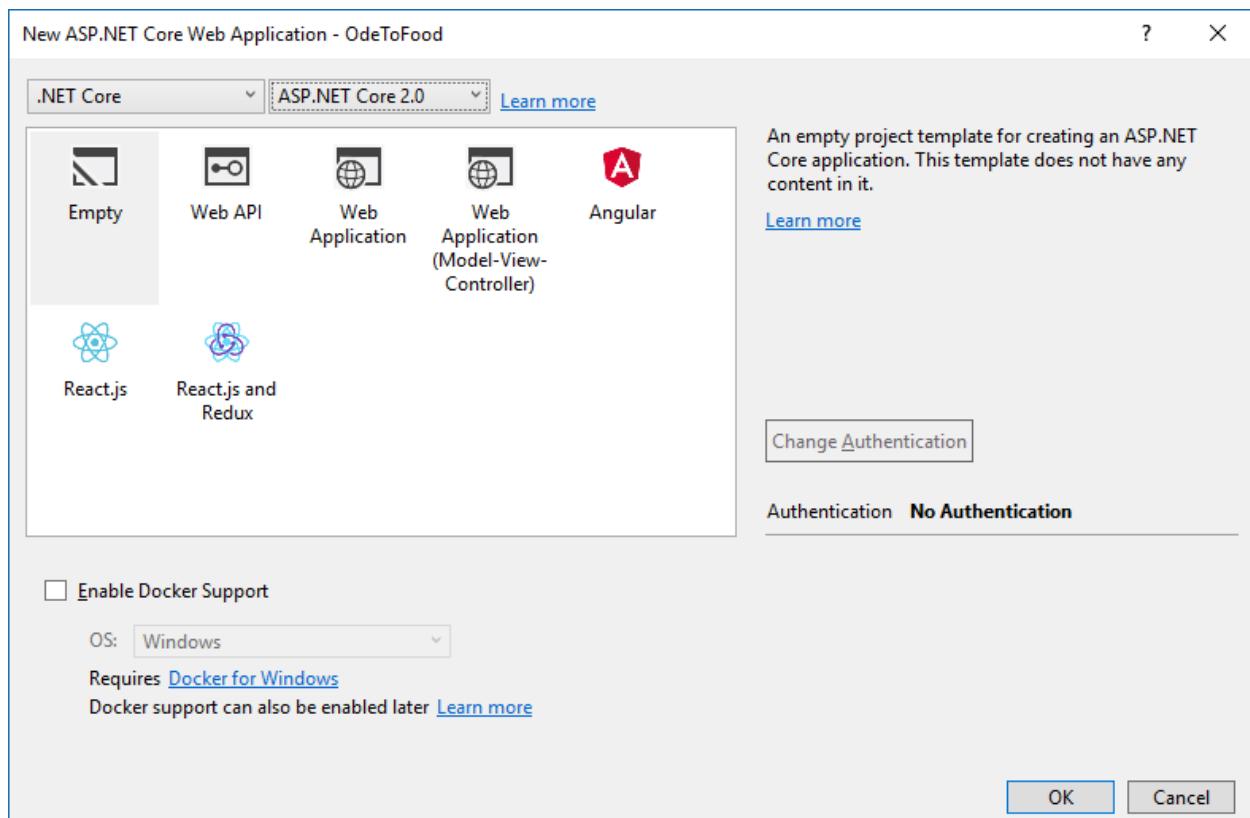
Feedback

A New Project

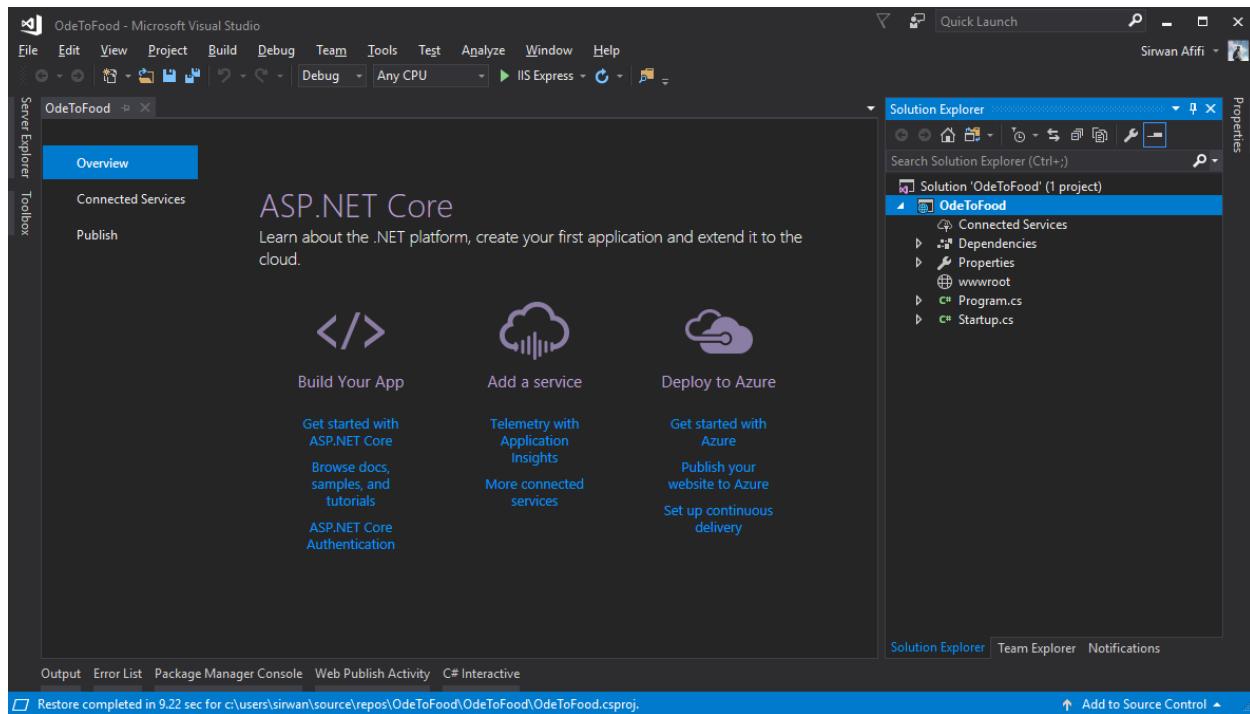
در ادامه یک پروژه جدید ایجاد خواهیم کرد:



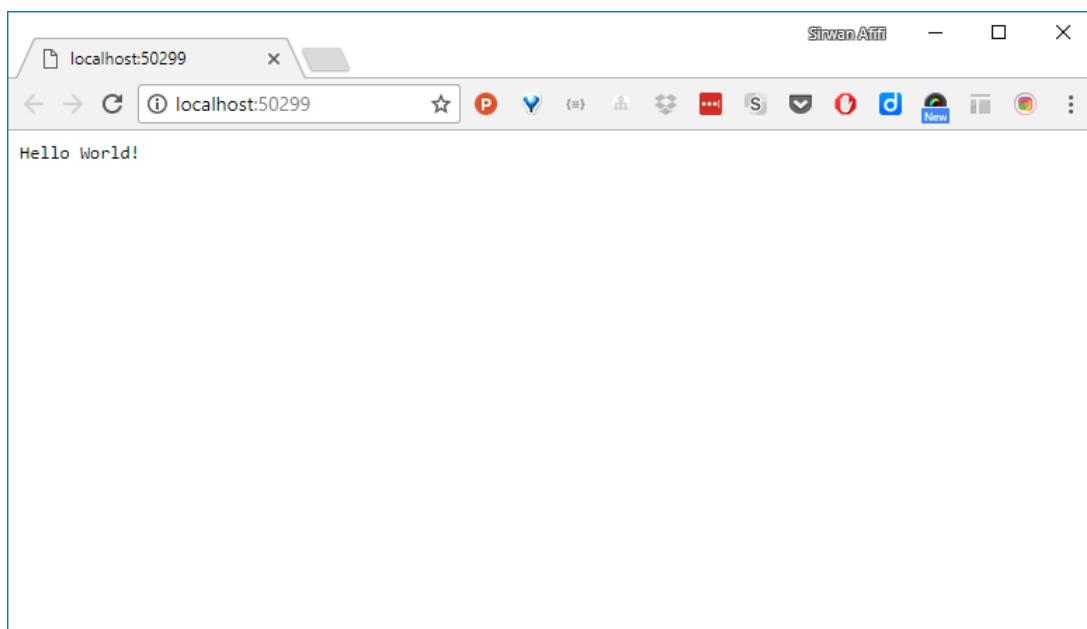
قدم بعدي:



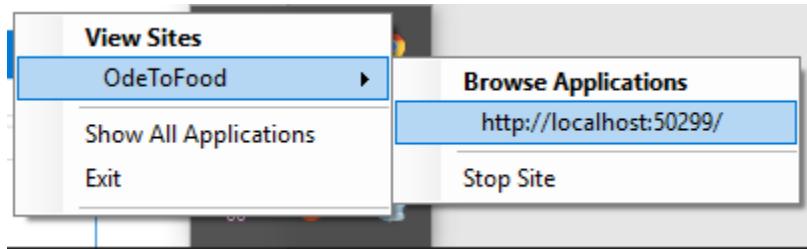
بعد از تائید دیالوگ فوق، پروژه برایمان ایجاد خواهد شد:



اکنون اگر پروژه را اجرا کنیم : (Ctrl + F5)



اگر به قسمت System tray توجه کنید آیکن IIS Express را خواهید دید، این در واقع یک است که همراه با Visual Studio Development Web Server عرضه می‌شود. با کلیک بر روی آن خواهید دید که اپلیکیشن‌مان توسط IIS بر روی آدرس localhost به همراه یک پورت تصادفی در حال اجرا می‌باشد:



Command Line Tools and Code

یک روش دیگر ایجاد پروژه از طریق خط فرمان می‌باشد؛ اینکار را می‌توانید توسط dotnet CLI انجام دهید؛ این دستور حاوی پارامترهای زیادی می‌باشد:

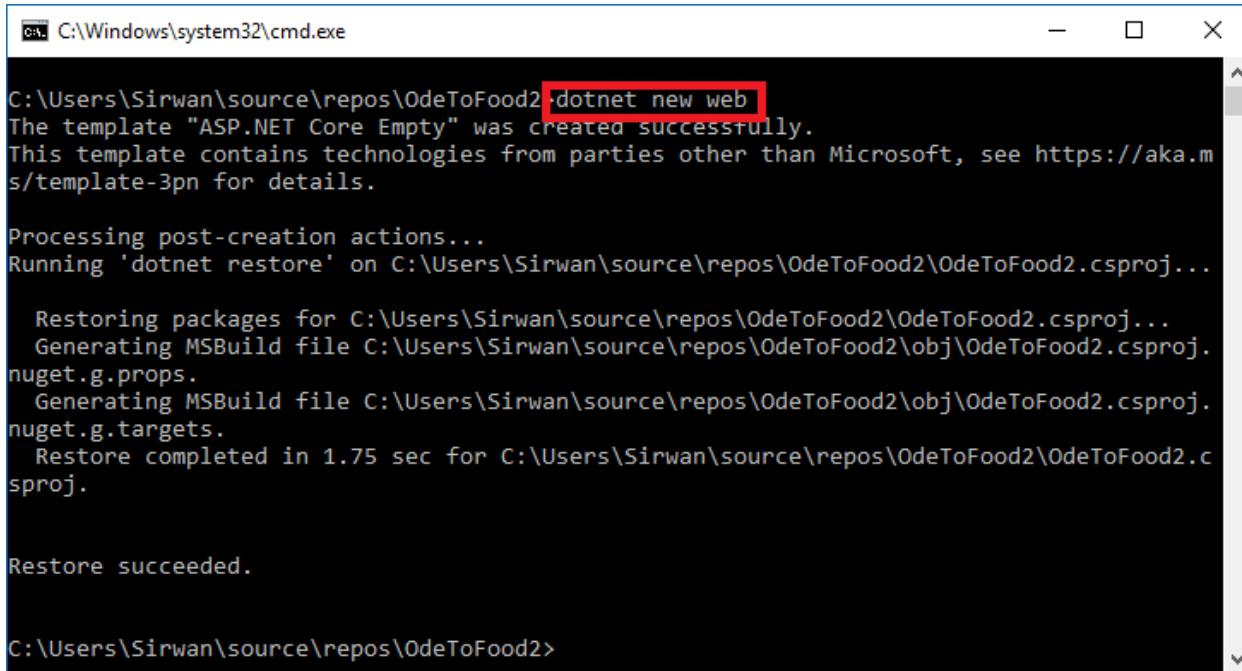
```
C:\Windows\system32\cmd.exe
The path to an application .dll file to execute.

SPK commands:
new Initialize .NET projects.
restore Restore dependencies specified in the .NET project.
run Compiles and immediately executes a .NET project.
build Builds a .NET project.
publish Publishes a .NET project for deployment (including the runtime).
test Runs unit tests using the test runner specified in the project.
pack Creates a NuGet package.
migrate Migrates a project.json based project to a msbuild based project.
clean Clean build output(s).
sln Modify solution (SLN) files.
add Add reference to the project.
remove Remove reference from the project.
list List reference in the project.
nuget Provides additional NuGet commands.
msbuild Runs Microsoft Build Engine (MSBuild).
vstest Runs Microsoft Test Execution Command Line Tool.

Common options:
-v|--verbosity Set the verbosity level of the command. Allowed values are q[uiet], v[erbose], vv[ery verbose], and vvv[ery verbose].
```

در واقع هر کاری که توسط Visual Studio می‌توان انجام داد را نیز توسط دستور فوق می‌توانیم انجام دهیم؛ که در واقع یک ابزار cross platform می‌باشد.

روش ایجاد پروژه قسمت قبل توسط خط فرمان:



```
C:\Windows\system32\cmd.exe
C:\Users\Sirwan\source\repos\OdeToFood2>dotnet new web
The template "ASP.NET Core Empty" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/template-3pn for details.

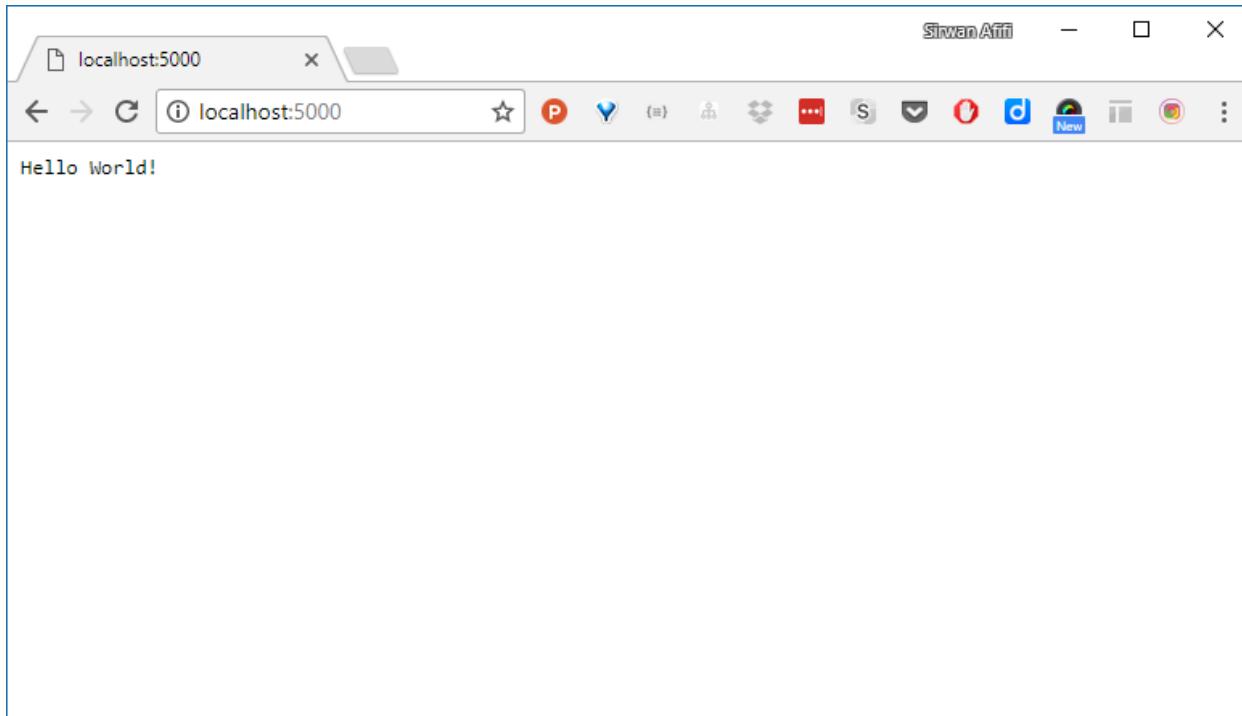
Processing post-creation actions...
Running 'dotnet restore' on C:\Users\Sirwan\source\repos\OdeToFood2\OdeToFood2.csproj...

Restoring packages for C:\Users\Sirwan\source\repos\OdeToFood2\OdeToFood2.csproj...
Generating MSBuild file C:\Users\Sirwan\source\repos\OdeToFood2\obj\OdeToFood2.csproj.nuget.g.props.
Generating MSBuild file C:\Users\Sirwan\source\repos\OdeToFood2\obj\OdeToFood2.csproj.nuget.g.targets.
Restore completed in 1.75 sec for C:\Users\Sirwan\source\repos\OdeToFood2\OdeToFood2.csproj.

Restore succeeded.

C:\Users\Sirwan\source\repos\OdeToFood2>
```

در نهایت توسط دستور dotnet run می‌توانیم پروژه را اجرا کنیم:



توضیح VSCode؛ باز کردن پروژه، نحوه دیبیگ و...

The ASP.NET Core Project Structure

در ادامه می خواهیم یکسری Configuration data به اپلیکیشنمان اضافه کنیم اما قبل از آن می خواهیم محتوای فایل `Program.cs` را بررسی نمائیم بخصوص متدهای `BuildWebHost`:

```
public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .Build();
```

یک `WebHostBuilder` در واقع شیءی است که می داند چطور `web server environment` را تنظیم کند؛ شیء پیشفرض آن محیط موردنیازمان را به یک روش خاص تنظیم خواهد کرد؛ اینها تنظیماتی هستند که ما می توانیم توسط نوشتن کدهای `BUILDER` خودمان تغییر دهیم. اما حالت پیشفرض این چنین است:

1- `Kestrel` Default web host builder : اپلیکیشن را به گونه ایی تنظیم خواهد کرد که از `Kestrel` به عنوان وب سرور استفاده کند؛ `Kestrel` نام وب سروری است که به عنوان بخشی از `ASP.NET Core` عرضه شده است؛ این وب سرور به صورت کراس پلات فرم می باشد یعنی بر روی تمامی سیستم عامل های `.NET Core` پشتیبانی شده توسط `Http` اجرا می باشد. این همان وب سروری است که به `connections` پروسه هی ما گوش فرا خواهد داد و همچنین وب سروری است که ما از طریق خط فرمان آن را استفاده می کنیم.

2- `IIS Integration` : بیlder همچنین این بخش را تنظیم خواهد کرد؛ اگر اپلیکیشنمان پشت `IIS` اجرا شود، این همان بخشی است که به `IIS` اجازه عبور از `Windows credentials` را خواهد داد.

3- `Logging` : بیlder همچنین تعدادی لایگینگ را تنظیم خواهد کرد. وقتی پروژه را اجرا کنید `log` ها را می توانید مشاهده نمایید.

4- `IConfiguration service` : `IConfiguration` را در دسترس قرار خواهد داد – بیlder پیش فرض شیءی که اینترفیس اپلیکیشنمان در هر جایی استفاده کند را وله سازی خواهد کرد. این شیء را می توانیم درون اپلیکیشنمان پیاده سازی کرده باشد را وله سازی خواهد کرد. کار این سرویس فراهم کردن فایل های پیکربندی در اپلیکیشن است، سورس های پیش فرض این سرویس:

a. `appsettings.json`

b. `User secrets`

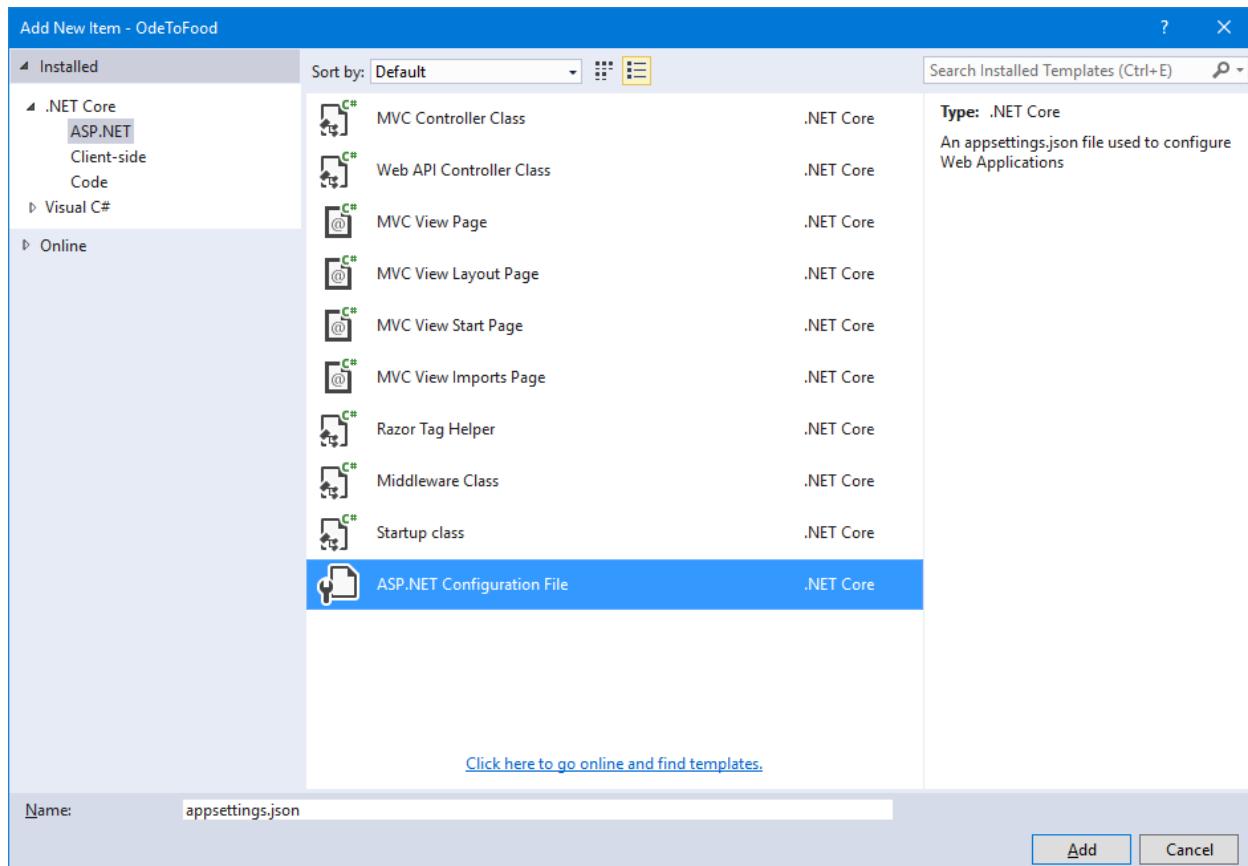
c. `Environment variables`

d. `Command line arguments`

: `IConfiguration` توسط سرویس `appsettings.json` خواندن محتوای

میخواهیم متن Configuration شدهی **Hello World** در مثال قبل را به درون یک فایل **hardcode** منتقل کرده و از طریق آن این متن را نمایش دهیم. قبل تنظیمات را درون فایل **web.config** ذخیره میکردیم؛ این فایل در نسخهی **ASP.NET Core** تنها برای تعیین تنظیمات برای **IIS** مورداستفاده قرار خواهد گرفت.

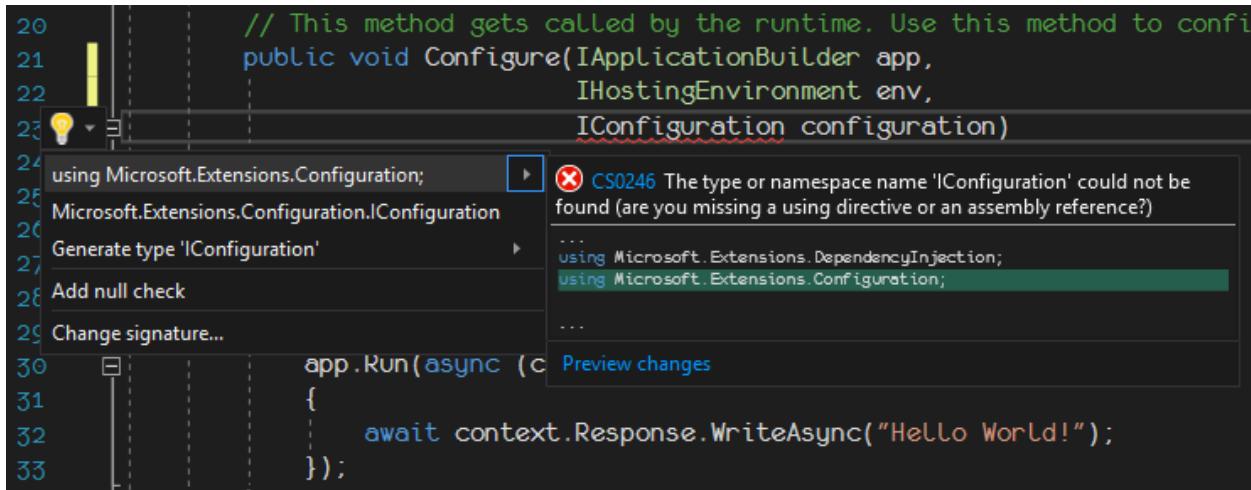
در ادامه فایل **appsettings.json** را ایجاد خواهیم کرد:



درون این فایل میتوانیم تنظیمات موردنظرمان را اضافه کنیم:

```
{  
  "Greeting": "Hello!!",  
  //....  
}
```

افزودن سرویس عنوان شده:



A screenshot of Visual Studio showing the code editor with the following C# code:

```
20 // This method gets called by the runtime. Use this method to config
21 public void Configure(IApplicationBuilder app,
22                         IHostingEnvironment env,
23                         IConfiguration configuration)
24
25     using Microsoft.Extensions.Configuration;
26     Microsoft.Extensions.Configuration.IConfiguration
27     Generate type 'IConfiguration'
28     Add null check
29     Change signature...
30     app.Run(async (c
31     {
32         await context.Response.WriteAsync("Hello World!");
33     });

```

The code editor shows a tooltip for the `IConfiguration` type, which includes the message: "CS0246 The type or namespace name 'IConfiguration' could not be found (are you missing a using directive or an assembly reference?)". Below the tooltip, there is a suggestion list with the following items:

- using Microsoft.Extensions.DependencyInjection;
- using Microsoft.Extensions.Configuration;
- ...

A "Preview changes" button is also visible at the bottom of the tooltip.

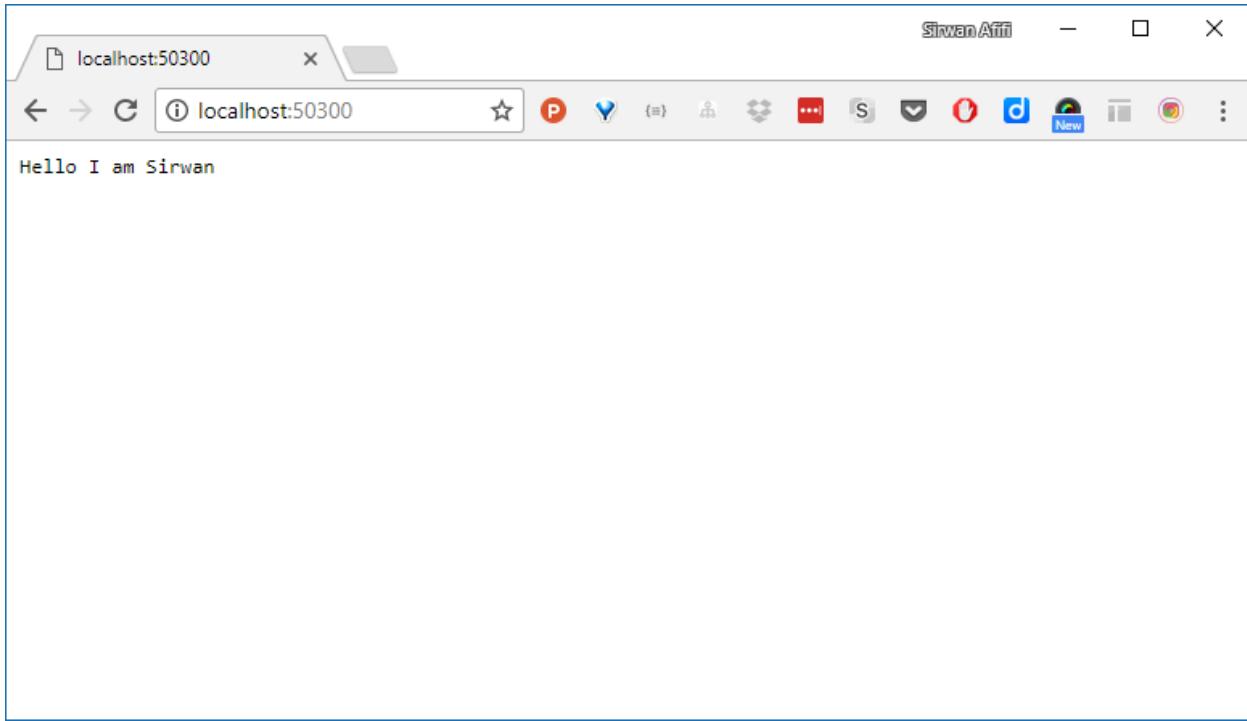
خواندن تنظیمات توسط سرویس اضافه شده:

```
app.Run(async (context) =>
{
    var greeting = configuration["Greeting"];
    await context.Response.WriteAsync(greeting);
});
```

نکته: اگر `IConfiguration service` کلید مورد جستجو را درون `appsettings.json` پیدا کند از آن استفاده خواهد کرد در غیراینصورت به دنبال دیگر سورس‌هایی که عنوان شد خواهد رفت. همانطور که عنوان شد آخرین سورس `Command line arguments` است؛ بنابراین اگر پروژه را اینچنین اجرا کنیم مقدار `Greeting` از فایل `appsettings.json` در نظر گرفته نخواهد شد زیرا توسط سورس آخر `override` شده است:

`dotnet run Greeting="Hello I am Sirwan"`

خروجی:



Creating a Greeting Service

در واقع نمی‌خواهیم به شیء Configuration به صورت مستقیم درون متدهای دسترسی داشته باشیم؛ می‌خواهیم اینکار را به یک سرویس محول کنیم.

قدم اول: ایجاد سرویس:

```
public interface IGreeter
{
    string GetMessageOfDay();
}

public class Greeter : IGreeter
{
    private IConfiguration _configuration;

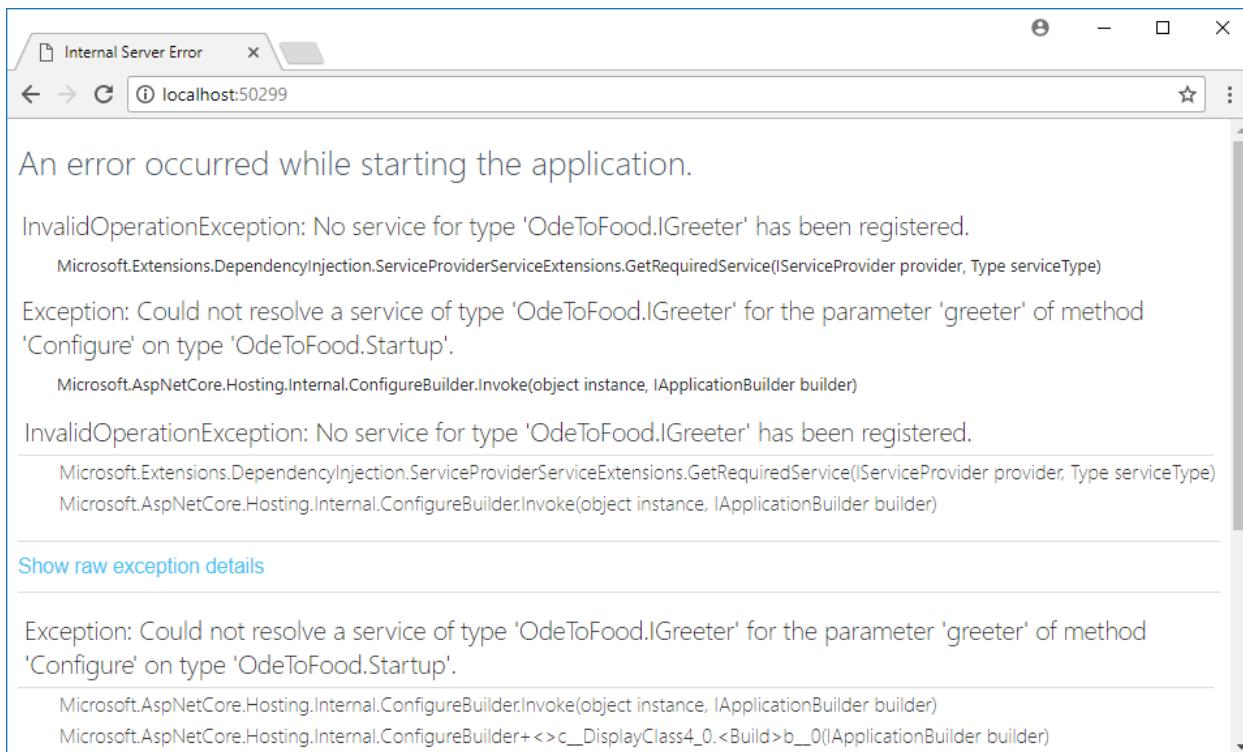
    public Greeter(IConfiguration configuration)
    {
        _configuration = configuration;
    }
    public string GetMessageOfDay()
    {
        return _configuration["Greeting"];
    }
}
```

```
        }  
    }
```

قدم دوم: استفاده از سرویس فوق:

```
public void Configure(IApplicationBuilder app,  
                      IHostingEnvironment env,  
                      IGreeter greeter)  
{  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
  
    app.Run(async (context) =>  
    {  
        var greeting = greeter.GetMessageOfDay();  
        await context.Response.WriteAsync(greeting);  
    });  
}
```

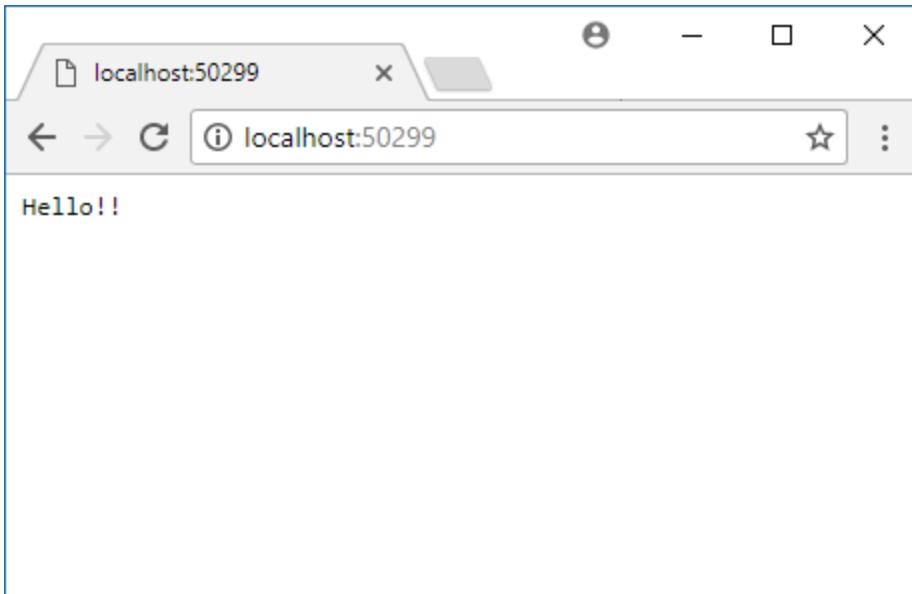
اکنون اگر اپلیکیشن را اجرا کنیم؛ خطای زیر را دریافت خواهیم کرد:



همانطور که مشاهده می‌کنید سرویس **IGreeter** ریجستر نشده است، برای اینکار باید این سرویس را درون متده **ConfigureServices** ریجستر کنیم:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddScoped<IGreeter, Greeter>();
}
```

خروجی:



طول عمر سرویس‌ها

- تنها به یک وله از سرویس در طول کل حیات اپلیکیشن نیاز خواهیم داشت. **AddSingleton**
- هر کسی به وله‌ایی از سرویس نیاز داشت یک وله جدید برای او ایجاد کن. **AddTransient**
- به ازای هر درخواست **Http** یک وله از سرویس ایجاد کن. **AddScoped**

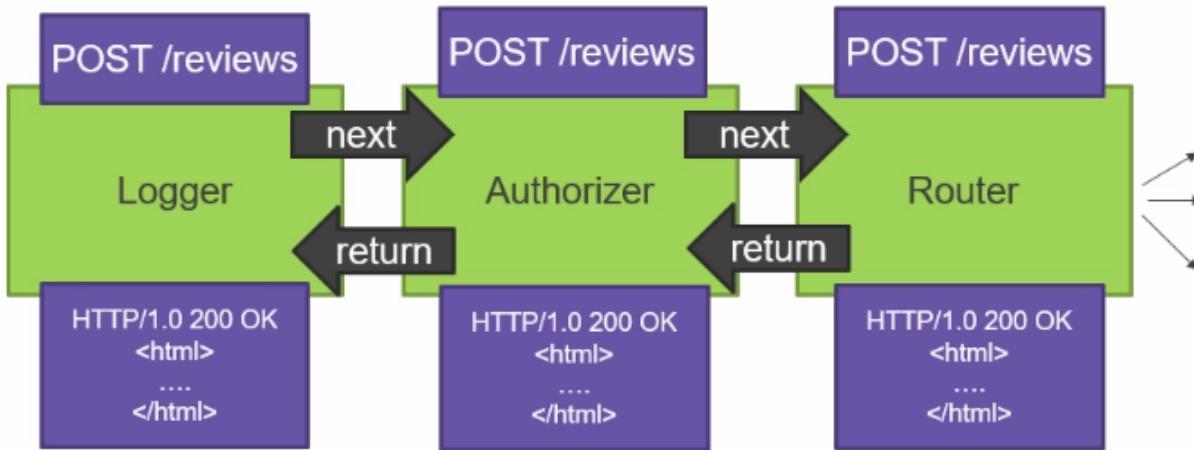
Startup and Middleware Introduction

با استفاده از **ASP.NET middleware** می‌توانیم نحوه پاسخ‌گویی درخواست‌های **HTTP** را کنترل کنیم.

How Middleware Works

وقتی یک درخواست **HTTP** به سمت سرور می‌رسد، مثلًاً فرض کنید یک درخواست از نوع **POST** داریم. نرم‌افزار ما باید به نوعی به این درخواست پاسخ دهد. در **ASP.NET Core** این **middleware** است که تعیین می‌کند این درخواست چگونه باید پردازش شود. فرض کنید برای هر درخواست می‌خواهیم اطلاعات آن را لگ کنیم؛ در نتیجه به یک کامپوننت **logging** نیاز خواهیم داشت؛ این کامپوننت بعد از انجام وظیفه‌ی خودش، درخواست را به **middleware** بعدی هدایت می‌کند. ممکن است **authorizer middleware** بعدی یک **authorizer** باشد که دنبال یک کوکی یا اکسس توکن خاص درخواست جستجو خواهد کرد؛ اگر این **authorizer** توکن

و یا کوکی را پیدا کند به درخواست اجازه خواهد داد تا به middleware بعدی هدایت شود. بعدی ممکن است یک روتر باشد.



Using IApplicationBuilder

برای راهاندازی middleware در یک اپلیکیشن ASP.NET Core نیاز به تغییر کدهای درون متدهای `Configure` است.

به عنوان مثال توسط متدهای `Run` و `Use` امکان اجرای middleware‌های low-level را خواهیم داشت. درون این متدها با استفاده از شیء `context` به تمامی قسمت‌های یک درخواست دسترسی خواهیم داشت:

```

app.Run(async (context) =>
{
    var message = await context.Response.WriteAsync("Hello World!");
});

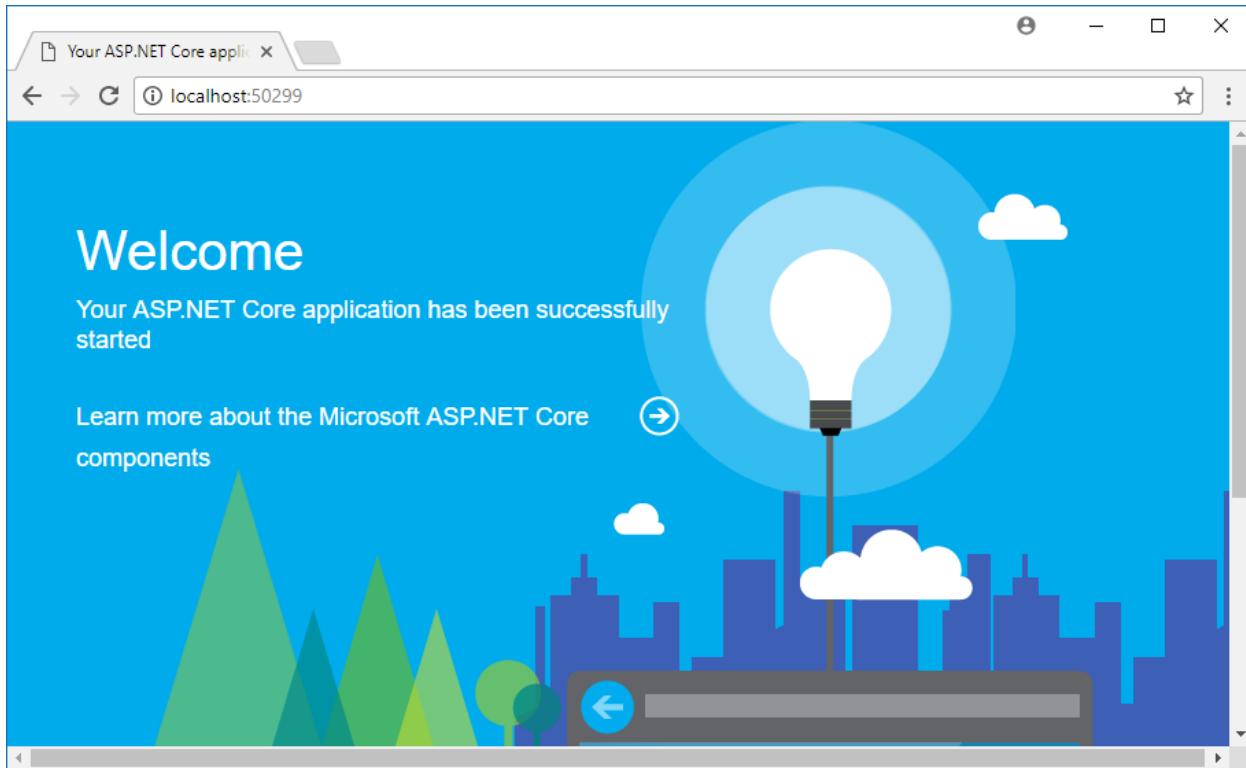
```

The screenshot shows a tooltip for the `context` parameter, which is of type `Microsoft.AspNetCore.Http.HttpContext`. The tooltip details the `AuthenticationManager` property, which facilitates authentication for the request.

Besides the `Run` method, there are extension methods for middleware such as `Use` and `UseXXX` for setting up middleware components.

```
app.UseWelcomePage();
```

کد فوق یک middleware ساده برای نمایش صفحه خوشآمدگویی است؛ که در حالت پیشفرض به هر درخواستی پاسخ خواهد داد:



نکته‌ایی که باید به آن دقت داشته باشید؛ اولویت در نصب middleware‌ها می‌باشد. به عنوان مثال middleware فوق به هر درخواستی با هر آدرسی پاسخ خواهد داد و صفحه فوق را نمایش می‌دهد؛ در اینحالت middleware‌های بعد از آن شناس نمایش داده شدن را نخواهد داشت. به هر حال می‌توانید با پاس دادن یکسری آپشن، مسیر نمایش صفحه فوق را تغییر دهید:

```
app.UseWelcomePage(new WelcomePageOptions
{
    Path = "/wp"
});
```

اکنون middleware فوق زمانی نمایش داده خواهد شد که آدرس /wp را وارد نمائیم.

توسط app.Run می‌توانیم middleware‌های low-level بنویسیم؛ این متدها یک تابع از ورودی دریافت خواهد کرد که ورودی و خروجی این تابع یک RequestDelegate می‌باشد؛ یک HttpContext از ورودی دریافت کرده و در نهایت یک تابع از خروجی خواهد داشت:

```
app.Use(next =>
```

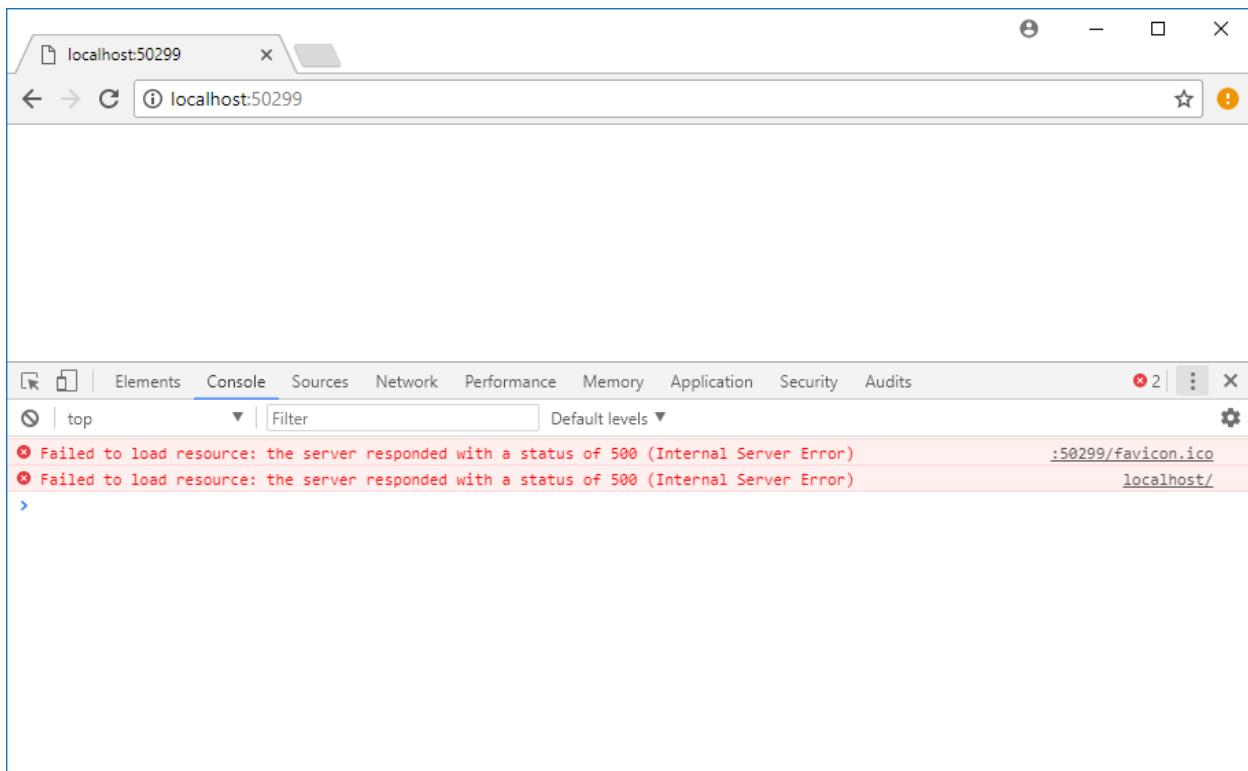
```
{  
    return async context =>  
    {  
        logger.LogInformation("Request incoming");  
        if (context.Request.Path.StartsWithSegments("/mym"))  
        {  
            await context.Response.WriteAsync("Hit!!");  
        }  
        else  
        {  
            await next(context);  
            logger.LogInformation("Response outgoing");  
        }  
    };  
});
```

Showing Exception Details

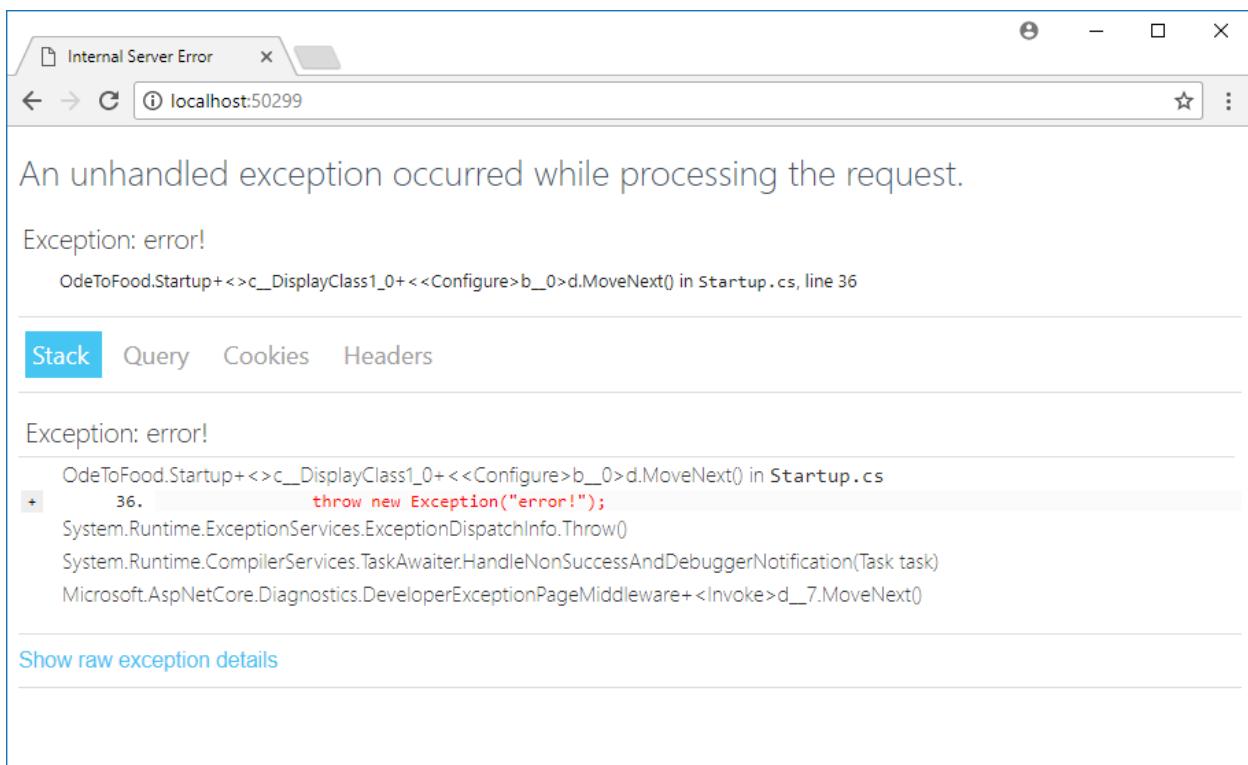
در ادامه می‌خواهیم میان‌افزار UseDeveloperExceptionPage کامنت کنیم؛ و به صورت عمد استثناء زیر را صادر نمائیم:

```
app.Run(async (context) =>  
{  
    throw new Exception("error!");  
  
    var greeting = greeter.GetMessageOfDay();  
    await context.Response.WriteAsync(greeting);  
});
```

به محض بروز یک استثناء در pipeline این خروجی را خواهیم داشت:



اما اگر مجدداً نصب کنیم (کامنت را برداریم) این خروجی را خواهیم داشت:

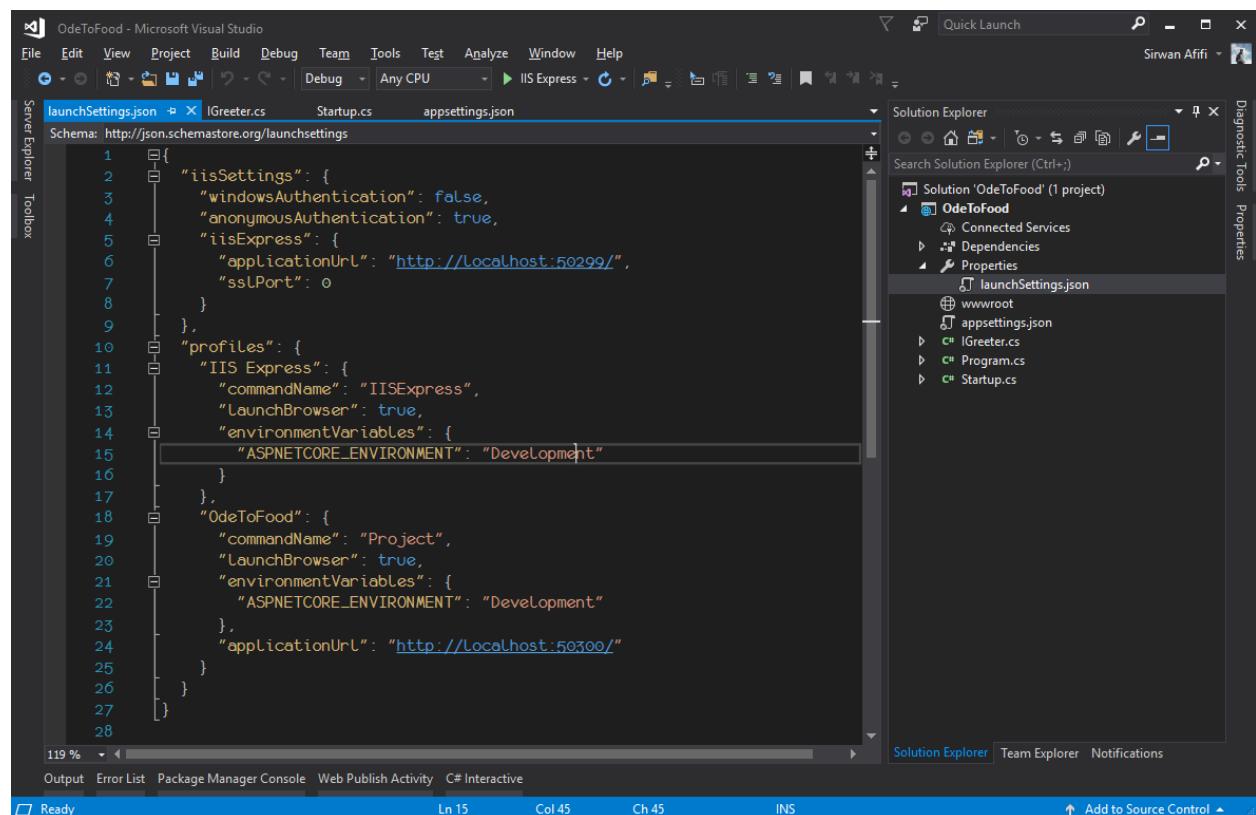


این middleware اجازه می‌دهد درخواست به middleware‌های بعدی هدایت شود. اگر در جایی دیگر در pipeline خطایی رخ دهد، یک UI همراه با اطلاعات کاملی از وضعیت خطا به کاربر نمایش داده خواهد شد (شکل فوق)

Middleware to Match the Environment

اینترفیس `IHostingEnvironment` اطلاعات کاملی را درباره hosting environment در اختیارمان قرار خواهد داد. از طریق این اینترفیس است که نوع `environment` را تعیین خواهیم کرد.

زمانیکه یک پروژه از نوع ASP.NET Core را ایجاد خواهیم کرد، تعدادی پراپرتی را به پروژه جهت پیکربندی اپلیکیشن برای اجرا در محیط development اضافه خواهد کرد؛ برای ست کردن این پراپرتی‌ها می‌توانیم با مراجعه فایل `launchSettings.json` اینکار را انجام دهیم:



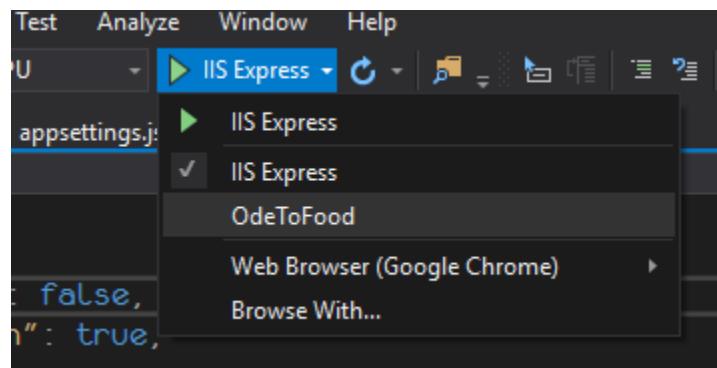
```

1 {
2   "iisSettings": {
3     "windowsAuthentication": false,
4     "anonymousAuthentication": true,
5     "iisExpress": {
6       "applicationUrl": "http://localhost:50299/",
7       "sslPort": 0
8     }
9   },
10  "profiles": {
11    "IIS Express": {
12      "commandName": "IISExpress",
13      "LaunchBrowser": true,
14      "environmentVariables": {
15        "ASPNETCORE_ENVIRONMENT": "Development"
16      }
17    },
18    "OdeToFood": {
19      "commandName": "Project",
20      "LaunchBrowser": true,
21      "environmentVariables": {
22        "ASPNETCORE_ENVIRONMENT": "Development"
23      },
24      "applicationUrl": "http://localhost:50300/"
25    }
26  }
27}
28

```

دیگر ابزارها نیز مانند dotnet CLI تنظیمات درون این فایل را در نظر خواهند گرفت. همانطور که مشاهده می‌کنید درون فایل فوق قسمتی تحت عنوان `profiles` داریم؛ اگر از ویژوال استودیو استفاده شود برای اجرای

پروژه پروفایل IIS Express در نظر گرفته خواهد شد. این همان نامی است که در قسمت toolbar موقع اجرای پروژه وجود دارد:

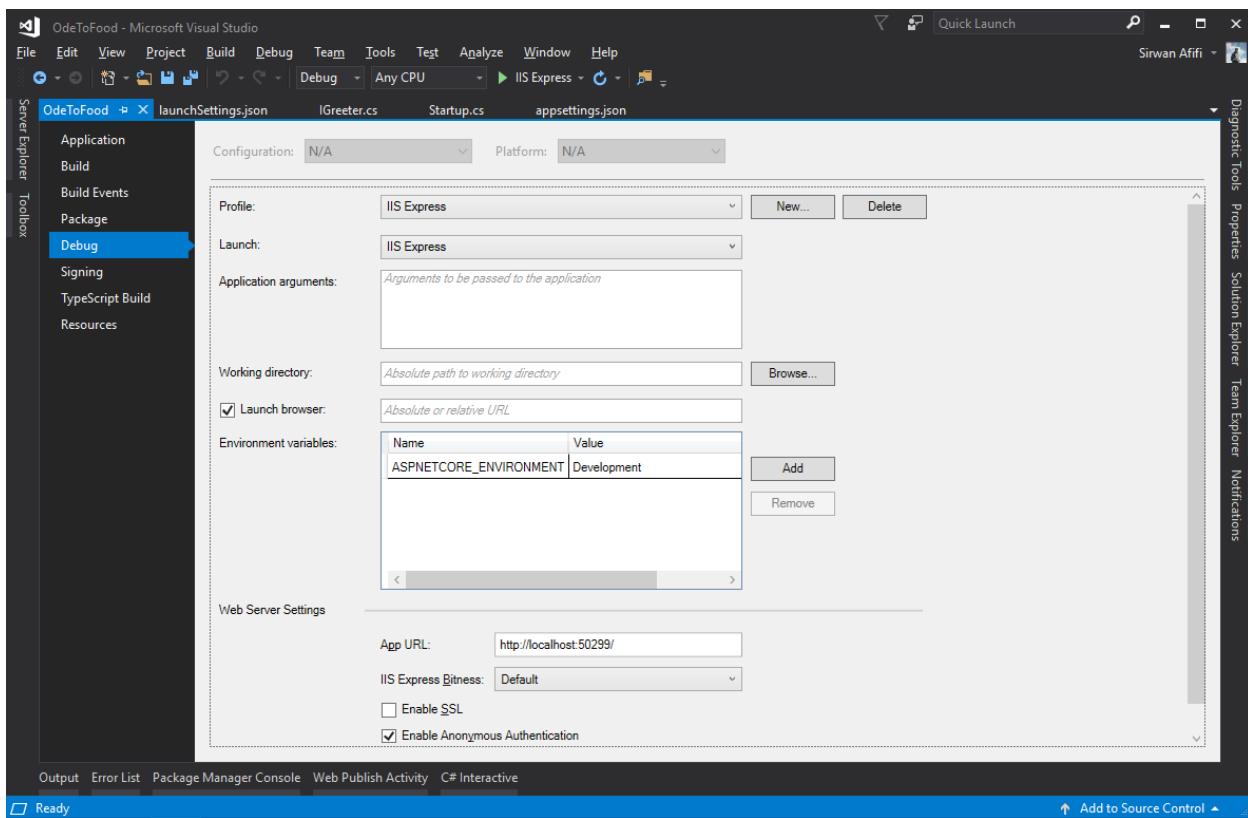


همچنین امکان انتخاب پروفایل OdeToFood نیز وجود دارد؛ این پروفایل نیز همچنین درون فایل launchSettings.json وجود دارد:

```
"profiles": {  
    "IIS Express": {  
        "commandName": "IISExpress",  
        "launchBrowser": true,  
        "environmentVariables": {  
            "ASPNETCORE_ENVIRONMENT": "Development"  
        }  
    },  
    "OdeToFood": {  
        "commandName": "Project",  
        "launchBrowser": true,  
        "environmentVariables": {  
            "ASPNETCORE_ENVIRONMENT": "Development"  
        },  
        "applicationUrl": "http://localhost:50300/"  
    }  
}
```

اما این پروفایل از command با نام Project برای اجرای پروژه استفاده خواهد کرد. منظور این است که در پشت صفحه از دستور dotnet run برای اجرای پروژه استفاده شود. همانطور که در کد فوق مشاهده می‌کنید برای هر پروفایل امکان تعریف یکسری متغیر محیطی را داریم.

یک روش friendly‌تر نیز استفاده از UI زیر جهت تغییر متغیرهای محیطی و همچنین پروفایل جاری پروژه می‌باشد:

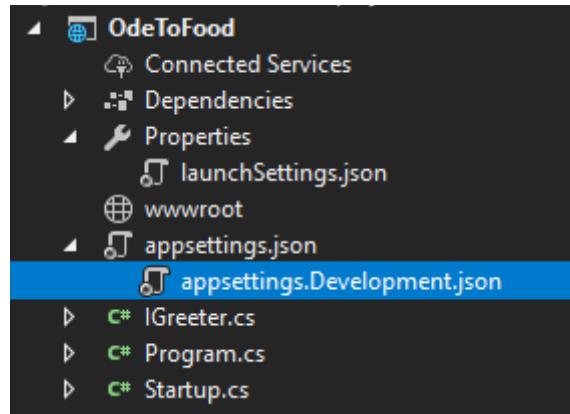


ASP.NET محیطی است که به صورت پیشفرض برای آن جستجو خواهد کرد. اگر مقدار آن برابر با `Development` باشد، فلاگ `IsDevelopment` برابر با `true` خواهد شد. اگر این پراپرتی وجود نداشته باشد ASP.NET فرض را بر این خواهد گذاشت که در محیط `Production` هستیم.

لازم به ذکر است که امکان داشتن چندین فایل `appsettings` را درون پروژه داریم؛ در حالت پیشفرض اپلیکیشن از فایل `appsettings.json` استفاده می‌کند؛ اما ASP.NET Core فایل دیگری با این الگوی نامگذاری را جستجو خواهد کرد:

appsettings.**EnvironmentName**.json

با ایجاد فایل این الگو ساختار نمایش آن اینچنین خواهد بود:



مزیت این فایل به عنوان مثال امکان داشتن connection string متفاوت برای محیط‌های متفاوتی می‌باشد.

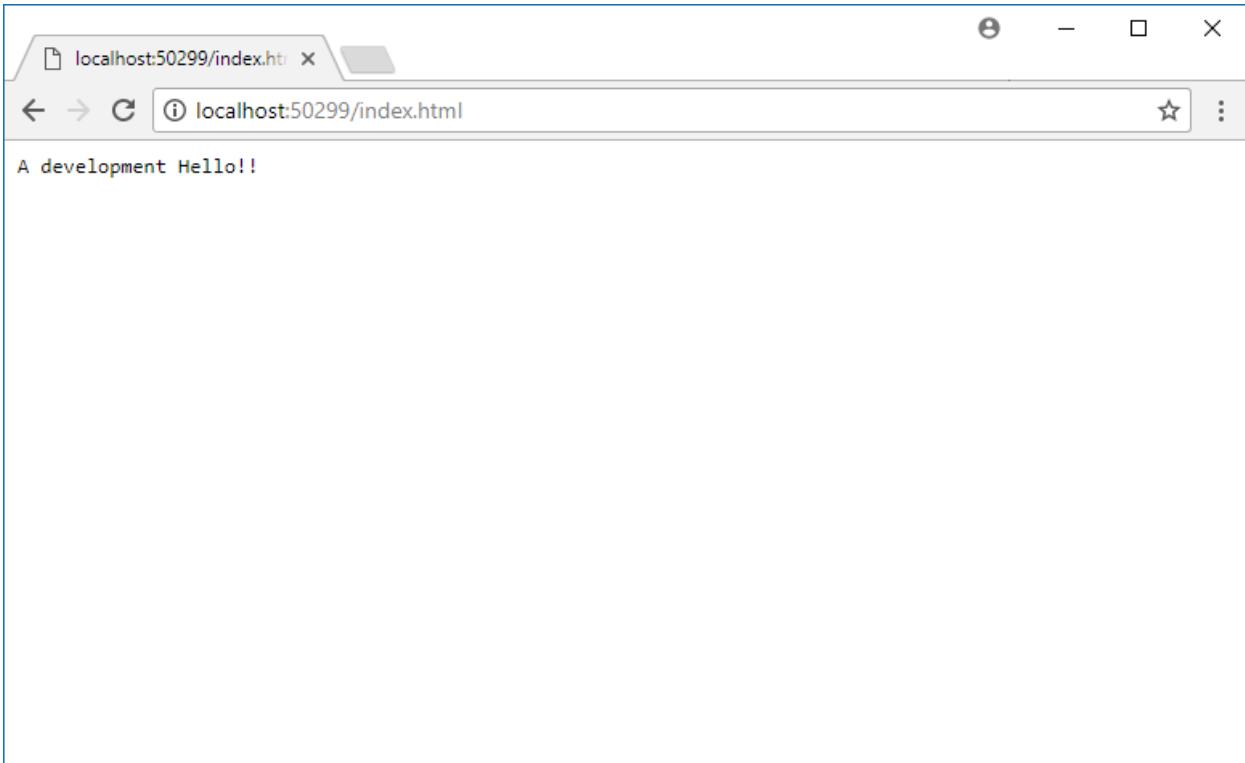
Serving Files

یکی از ویژگی‌های مهم هر وب‌اپلیکیشن داشتن قابلیت serve کردن فایل از فایل‌سیستم است.

اگر به پروژه‌ی مثال قبل یک فایل HTML اضافه کنید:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title></title>
</head>
<body>
    <h1>This is index.html!</h1>
</body>
</html>
```

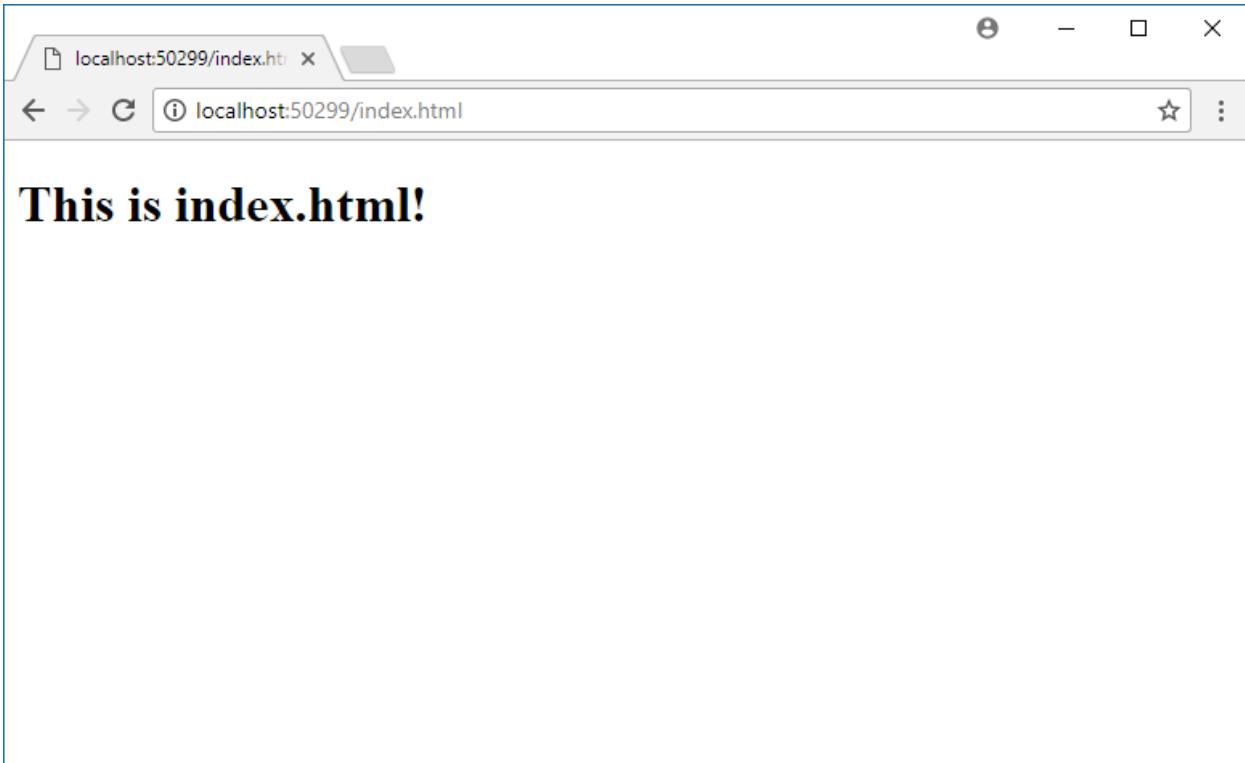
اکنون اگر سعی کنید از طریق مرورگر به آن دسترسی داشته باشید، این امکان را نخواهید داشت. (لازم به ذکر است که فایل‌های عمومی باید درون پوشه‌ی **wwwroot** قرار گیرند.)



دلیل این است middleware‌ی جهت رسیدگی به درخواست index.html درون pipeline‌مان هنوز ثبت نشده است. برای نصب این middleware باید پکیج زیر را نصب کنید:

```
app.UseStaticFiles();
```

خروجی:



در حالت پیشفرض `StaticFiles` به دنبال فایل‌های استاتیک درون پوشه‌ی `wwwroot` جستجو خواهد کرد. اگر بخواهیم `index.html` به عنوان پیشفرض در هنگام اجرای پروژه اجرا شود باید یک `middleware` دیگر با نام `UseDefaultFiles` را نصب کنیم:

```
app.UseDefaultFiles();
app.UseStaticFiles();
```

ترتیب قرار دادن `middleware`‌های فوق خیلی مهم است، اگر `UseDefaultFiles` را بعد از `UseStaticFiles` قرار دهیم، نتیجه مورد انتظار را دریافت نخواهیم کرد زیرا `UseDefaultFiles` در واقع کار `serve` کردن فایل‌های استاتیک را انجام نمی‌دهد؛ یک `middleware` دیگر با نام `UseFileServer` وجود دارد که کار هر دوی این `middleware`‌ها را انجام می‌دهد:

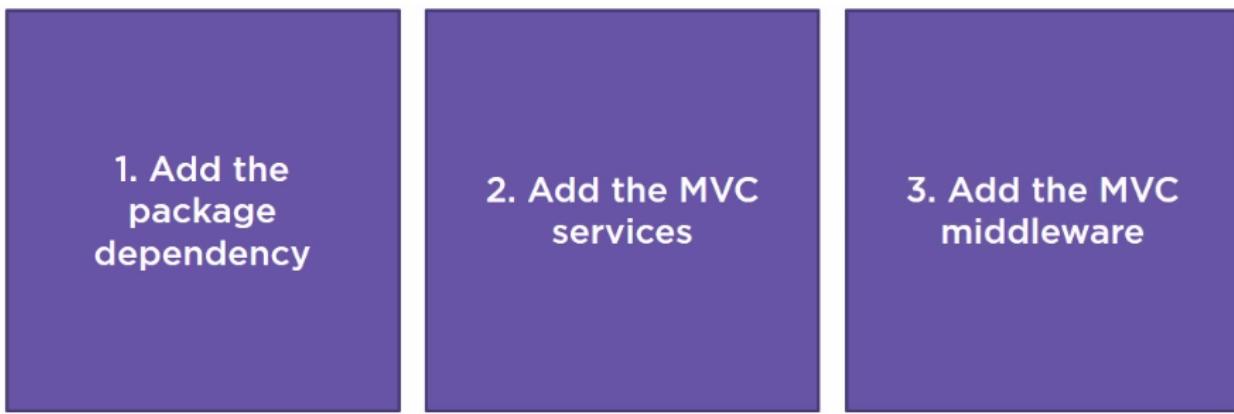
```
app.UseFileServer();
```

حتی می‌توانیم برای این `middleware` نیز یک آپشن ارسال کنیم؛ توسط این آپشن می‌توانیم کارهایی مانند `Directory Browsing` را انجام دهیم.

Setting up ASP.NET MVC Middleware

در نهایت می‌خواهیم یک اپلیکیشن بر فراز ASP.NET ایجاد کنیم؛ به طور دقیق‌تر ایجاد یک اپلیکیشن برفزار ASP.NET MVC؛ در واقع می‌توانیم یک اپلیکیشن را به صورت کامل با استفاده از middleware‌ها ایجاد کنیم؛ این قابلیت را در اختیارمان خواهد گذاشت تا به سادگی صفحات HTML و همچنین HTTP APIs را ایجاد کنیم. اینکار توسط آسان است زیرا می‌توانیم درخواست‌های واردہ را به متدهایی درون کلاس نگاشت کنیم؛ درون این متدها کارهایی مانند کوئری زدن به دیتابیس، خواندن فایل و ... آسان خواهد بود.

در این قسمت می‌خواهیم MVC را نصب کنیم؛ نصب MVC در یک پروژه خالی با سه قدم انجام خواهد شد:



- 1 نصب پکیج ASP.NET MVC
- 2 ریجستر کردن سرویس‌های MVC
- 3 افزودن middleware مربوط به MVC

قدم اول: نصب پکیج :MVC

وقتی یک پروژه خالی ASP.NET Core را آغاز کنید، پکیجی تحت عنوان متابکیج به صورت پیش‌فرض برای پروژه نصب خواهد شد:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>
  <TargetFramework>netcoreapp2.0</TargetFramework>
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.3" />
</ItemGroup>

</Project>
```

همانطور که از نام آن پیداست این پکیج تمامی پکیج‌های ASP.NET Core را تحت یک پکیج با نام **Microsoft.AspNetCore.All** نصب خواهد کرد؛ این پکیج حاوی MVC نیز می‌باشد.

قدم دوم: **MVC** ریجستر کردن سرویس‌های

```
app.UseStaticFiles();

app.UseMvcWithDefaultRoute();
```

نکته: اینبار از **UseStaticFiles** استفاده کرده‌ایم زیرا نمی‌خواهیم نام پیش‌فرضی برای فایل‌های استاتیک در نظر گرفته شود (**UseDefaultFiles**)؛ می‌خواهیم کاربر خودش فایل موردنظر را درخواست کند. در ادامه کار هندل کردن درخواست را به میان‌افزار **UseMvcWithDefaultRoute** واگذار کرده‌ایم.

قدم سوم: **MVC** مربوط به **middleware** افزودن

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddScoped<IGreeter, Greeter>();

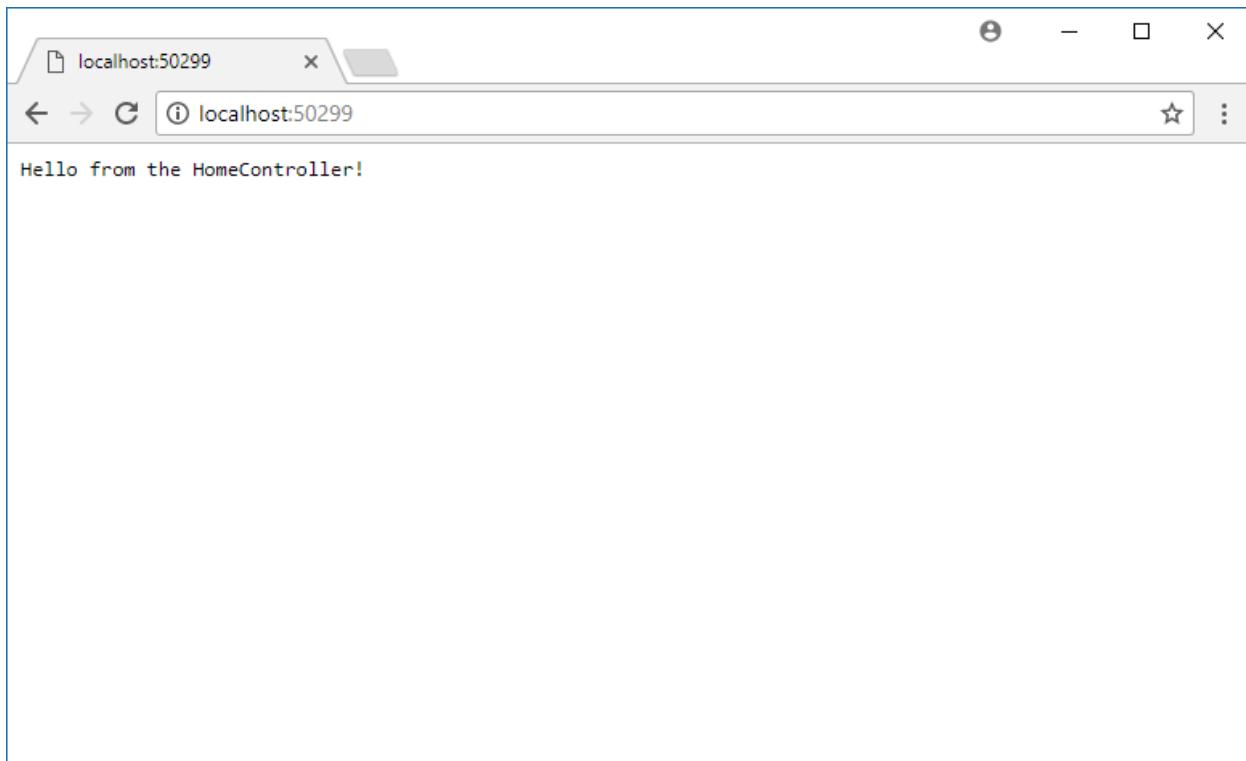
    services.AddMvc();
}
```

اکنون قبل از اجرای پروژه یک کنترلر ساده به صورت زیر ایجاد کنید:

```
namespace OdeToFood.Controllers
{
    public class HomeController
    {
```

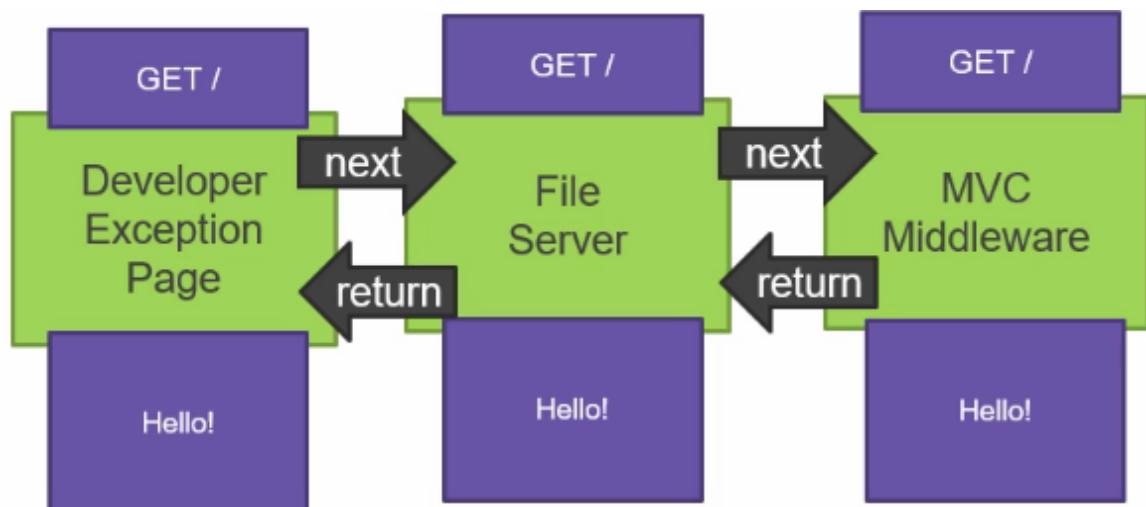
```
public string Index()
{
    return "Hello from the HomeController!";
}
```

خروجی:



Summary

در این مژول بیشتر درباره middleware‌ها و همچنین نحوه‌ی پیکربندی middleware‌ها درون متدهای Configure را بررسی کردیم. اکنون ما یک processing pipeline داریم؛ در این حالت یک درخواست از تعدادی middleware عبور خواهد کرد.

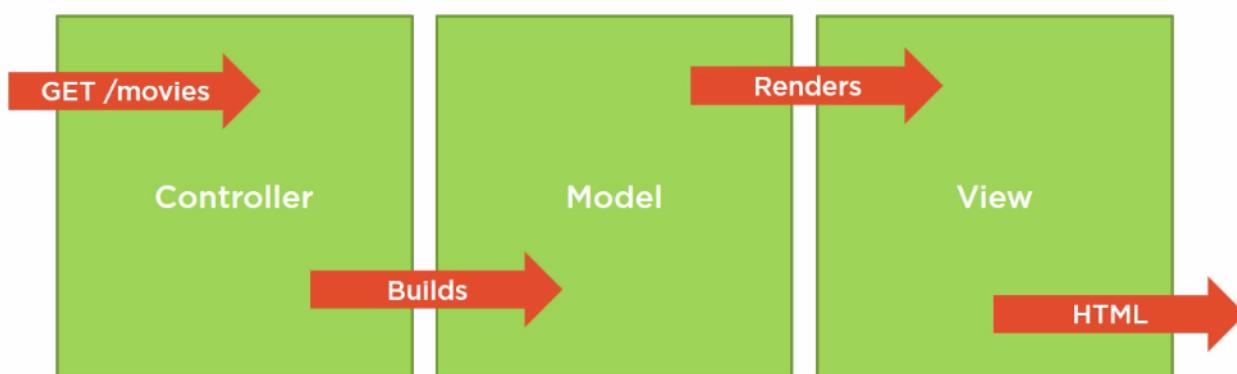


Controllers in the MVC Framework

The Model View Controller Design Pattern

نام فریمورک MVC برگرفته از الگوی طراحی MVC است؛ این الگو یکی از الگوهای محبوب در طراحی UI است. در اپلیکیشن‌هایی با مقیاس بزرگتر، معمولاً MVC را با دیگر الگوهای طراحی ترکیب خواهیم کرد؛ الگوهایی مانند Data Access, Messaging.

در MVC کنترلر یک درخواست HTTP را دریافت خواهد کرد، سپس کنترلر تصمیم خواهد گرفت که چگونه به این درخواست پاسخ دهد؛ اینکار را با populate کردن یک مدل انجام خواهد داد سپس این مدل را به ویو جهت نمایش ارسال خواهد کرد؛ سپس ویو اطلاعات را گرفته و سپس از آنها جهت ساختن یک صفحه HTML استفاده خواهد کرد سپس این HTML به عنوان یک HTTP response به کلاینت ارسال خواهد شد:



Routing

ASP.NET middleware که در ماژول قبلی نصب کردیم، به روشه جهت تعیین اینکه یک درخواست HTTP باید به یک کنترلر جهت پردازش ارسال شود یا نه، نیاز دارد. MVC middleware این تصمیم را براساس URL و یکسری تنظیمات (configuration information) اتخاذ خواهد کرد؛ یک روش برای تعیین این تنظیمات، تعریف routes برای کنترلرها درون فایل **Startup.cs** است؛ به این رهیافت convention گفته می‌شود:

```
routeBuilder.MapRoute("Default",
    "{controller=Home}/{action=Index}/{id?}");
```

با این روش به MVC یک تمپلیت را معرفی خواهد کرد که براساس آن URL را بررسی کند.

روش دیگر استفاده از attribute based routing است:

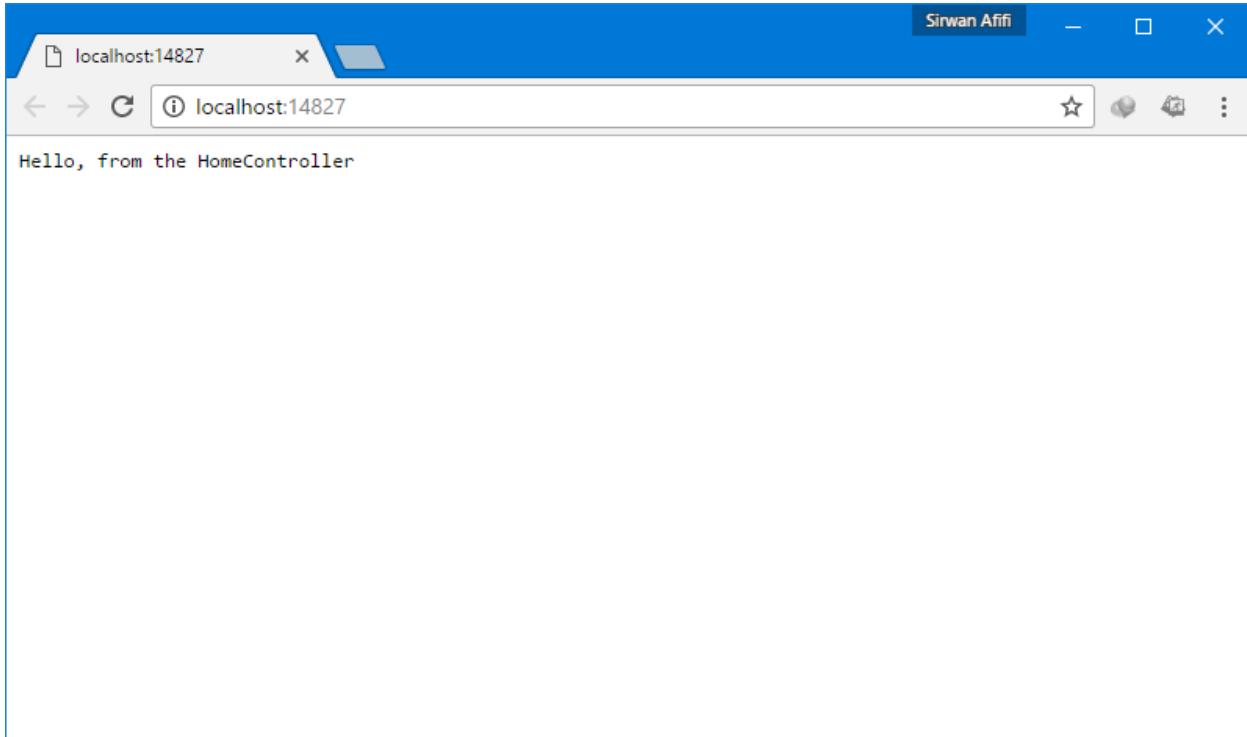
```
[Route("[controller]/[action]")]
```

Conventional Routes

در قسمت قبل یک کنترلر ایجاد کردیم:

```
namespace OdeToFood.Controllers
{
    public class HomeController
    {
        public string Index()
        {
            return "Hello from the HomeController!";
        }
    }
}
```

همانطور که مشاهده می‌کنید این کنترلر نیازی به ارث بری از کلاس پایه‌ی **Controller** ندارد. در عوض یک کلاس ساده‌ی C# همراه با یک متده درون آن داریم. در اینحالت خروجی زیر را خواهیم داشت:



دلیل این است که مسیریابی پیش‌فرض برای **MVC** فعال شده است:

```
app.UseMvcWithDefaultRoute();
```

برای داشتن مسیریابی سفارشی باید از متده **UseMvc** به اینصورت استفاده کنیم:

```
app.UseMvc(RouteConfig.ConfigureRoute);
private void ConfigureRoute(IRouteBuilder routeBuilder)
{
    // /Home/Index/4

    routeBuilder.MapRoute("Default",
        "{controller=Home}/{action=Index}/{id?}");
}
```

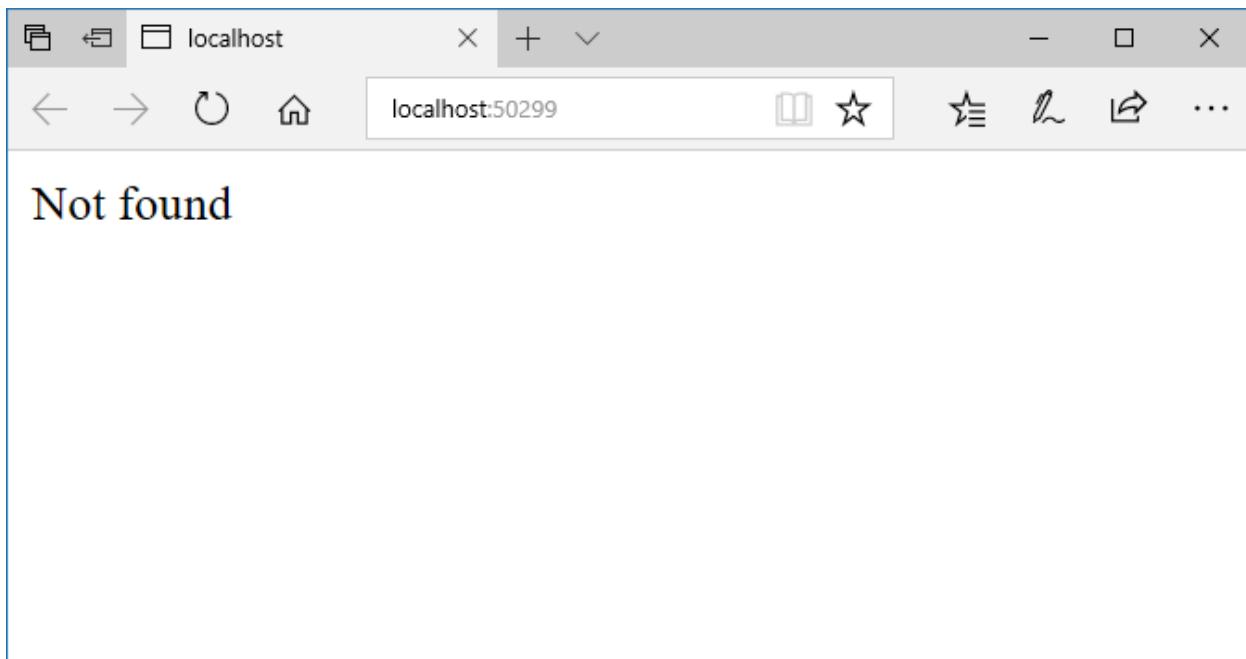
نکته: در حالت استفاده از متده **Run** باید نوع **MIME Type** خروجی را نیز تعیین کنیم. به عنوان مثال در کد زیر تنها یک رشته ساده را به خروجی فرستاده‌ایم:

```
app.UseMvc();
```

```
app.Run(async (context) =>
{
    var greeting = greeter.GetMessageOfDay();

    await context.Response.WriteAsync("Not found");
});
```

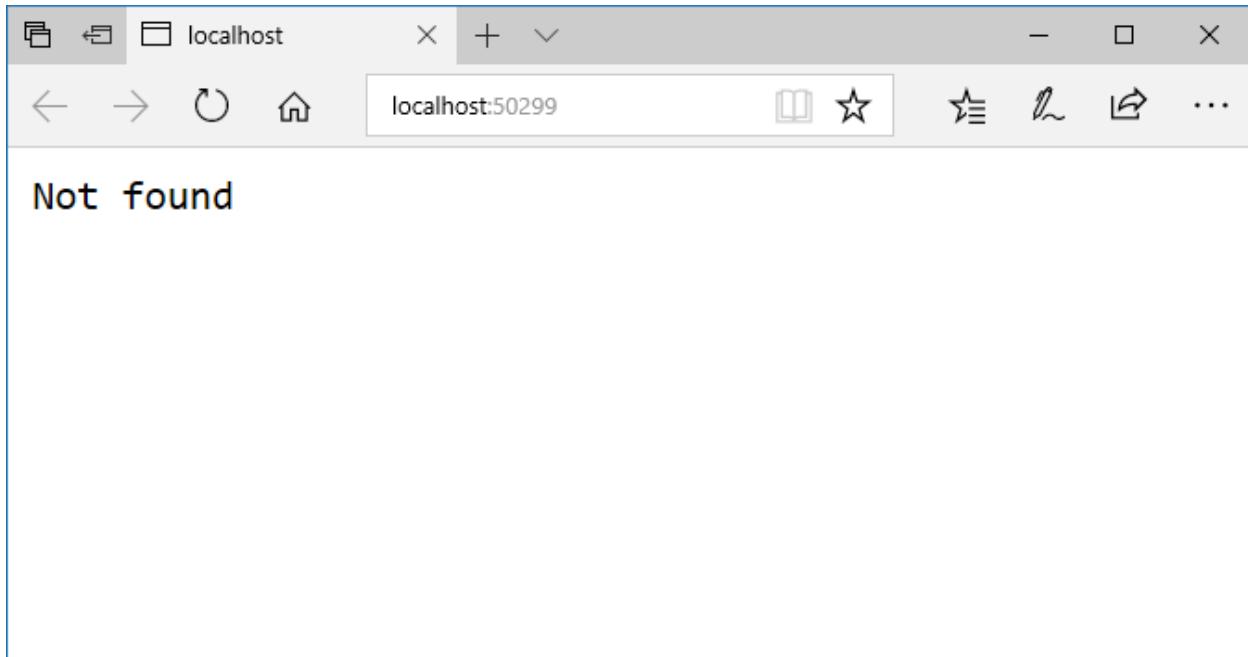
در اینحالت مرورگر وب تلاش خواهد کرد نوع محتوایی دریافتی برای مشاهده تشخیص دهد؛ بنابراین چون درون **Response Header** نمی‌تواند اطلاعاتی پیدا کند آن را به صورت زیر نمایش خواهد داد:



اما اگر **MIME Type** مناسب را برای خروجی تعیین کنیم؛ مرورگر به راحتی می‌تواند محتوای دریافتی را تشخیص دهد:

```
app.Run(async (context) =>
{
    var greeting = greeter.GetMessageOfDay();
    context.Response.ContentType = "text/plain";
    await context.Response.WriteAsync("Not found");
});
```

خروجی:



Attribute Routes

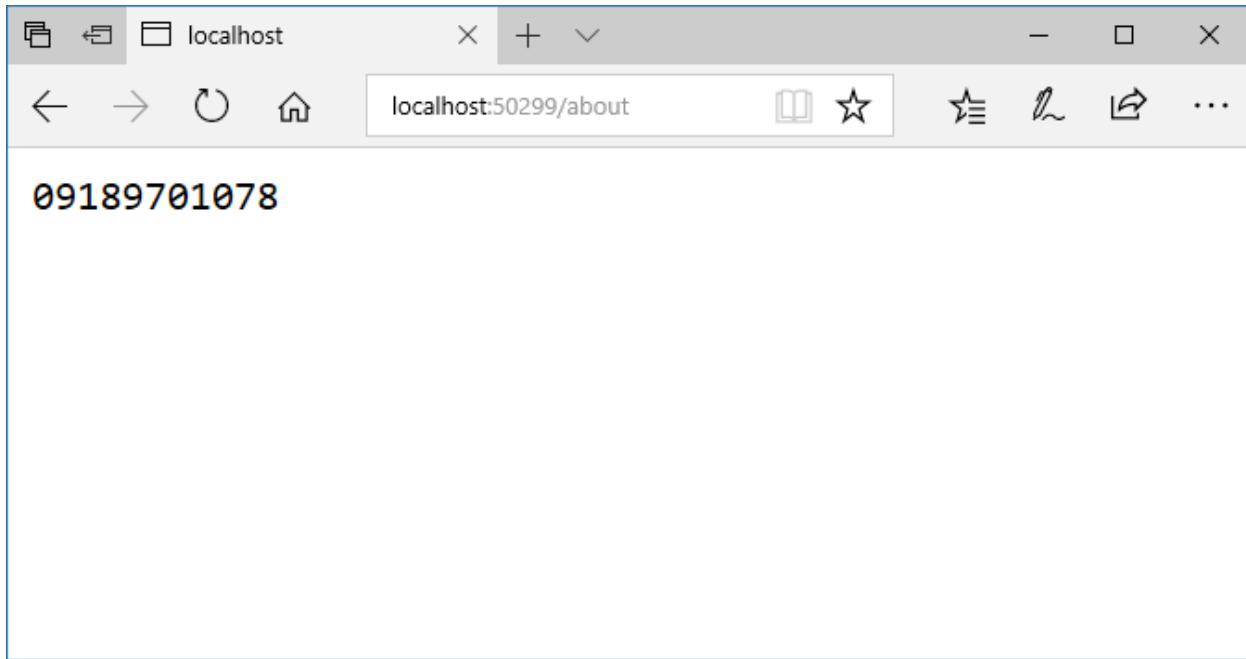
روش دیگر تعریف route در MVC استفاده از attribute routing است:

```
using Microsoft.AspNetCore.Mvc;

namespace OdeToFood.Controllers
{
    //about
    [Route("about")]
    public class AboutController
    {
        [Route("")]
        public string Phone()
        {
            return "09189701078";
        }

        [Route("address")]
        public string Address()
        {
            return "Krdistan, Sanandaj";
        }
    }
}
```

عدم تعیین Route template برای Route به معنای دیفالت بودن آن است (همانند Index) - در حالت فوق
حالت پیشفرض است: Phone



از توکن‌ها نیز می‌توانید استفاده کنید، در اینحالت دیگر نیازی نیست نام کنترلر یا اکشن متده را بنویسید:

```
using Microsoft.AspNetCore.Mvc;

namespace OdeToFood.Controllers
{
    //about
    [Route("company/[controller]/[action]")]
    public class AboutController
    {
        public string Phone()
        {
            return "09189701078";
        }

        public string Address()
        {
            return "Krdistan, Sanandaj";
        }
    }
}
```

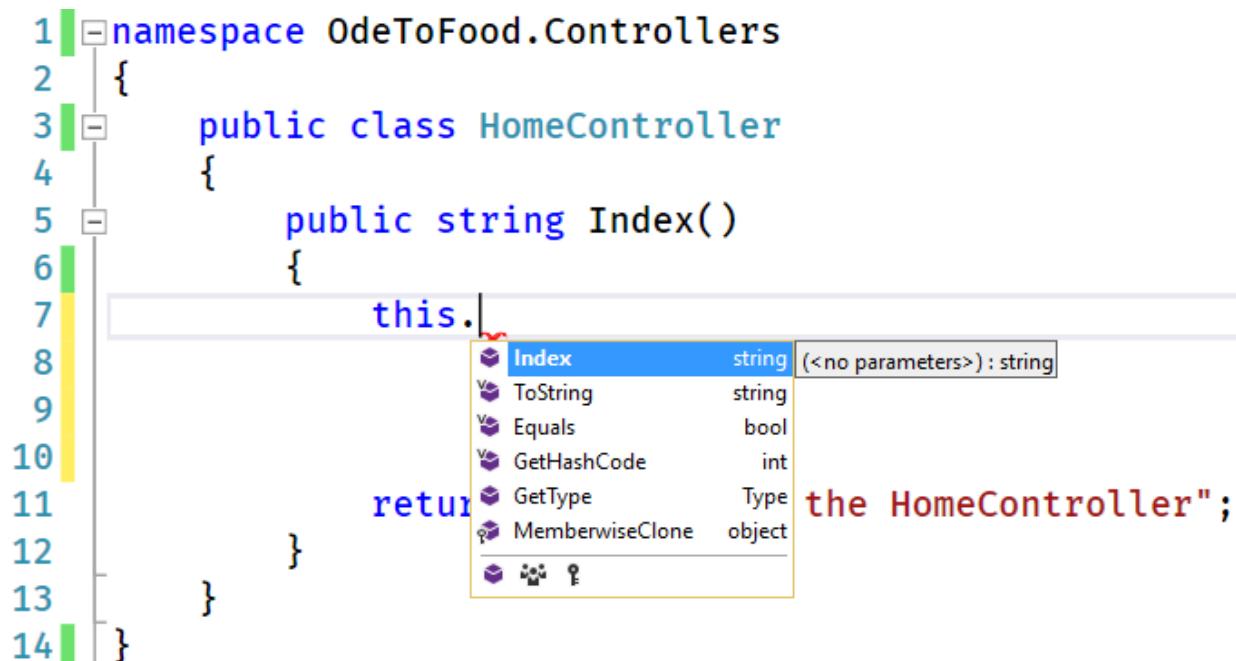
مزیت توکن این است که اگر بعداً نام کنترلر را تغییر دهیم؛ خود فریمورک آن را تشخیص خواهد داد زیرا به صورت صریح نام **about** را ننوشته‌ایم.

Action Results

تا اینجا از Plain simple C# classes مرسوم است که از کلاس پایه‌ی Controller ارث بری کنیم همچنین خروجی اکشن‌های آن نوع IActionResult باشند:

```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return Content("Hello, from a controller!");
    }
}
```

ارث بری از کلاس Controller قابلیت‌های زیادی را درون کنترلر در اختیارمان قرار خواهد داد؛ در حالت عدم ارث بری از این کلاس درون یک کنترلر خواهیم داشت:



```
1 namespace OdeToFood.Controllers
2 {
3     public class HomeController
4     {
5         public string Index()
6         {
7             this.|
```

The screenshot shows a tooltip for the 'Index' method of the 'HomeController'. The tooltip lists the following members:

Index	string	(<no parameters>) : string
ToString	string	
Equals	bool	
GetHashCode	int	
GetType	Type	
MemberwiseClone	object	

```
8         return "the HomeController";
9     }
10 }
```

اما اگر از Controller ارث بری کنیم:

```

1  using Microsoft.AspNetCore.Mvc;
2
3  namespace OdeToFood.Controllers
4  {
5      public class HomeController : Controller
6      {
7          public string Index()
8          {
9              this.|  

10             Response  

11             ControllerBase  

12             HttpContext  

13             MetadataProvider  

14             ModelBinderFactory  

15             ModelState  

16             ObjectValidator  

17             Request  

18             RouteData  

19             TempData  

20             Url
21         }
22     }
23 }

```

The screenshot shows an IDE's Intellisense feature. A tooltip is open over the code at line 9, showing a list of properties starting with 'this.'. The list includes: Response, ControllerBase, HttpContext, MetadataProvider, ModelBinderFactory, ModelState, ObjectValidator, Request, RouteData, TempData, and Url. To the right of the list, their corresponding interfaces are listed: HttpResponse, ControllerBase, HttpContext, IMetadataProvider, IModelBinderFactory, ModelStateDictionary, IObjectModelValidator, HttpRequest, RouteData, ITempDataDictionary, and IUrlHelper.

همانطور که مشاهده می‌کنید در اینحالت به contextual property‌های زیادی دسترسی خواهیم داشت؛
به عنوان مثال:

```

public JsonResult Index()
{
    var headers = this.HttpContext.Request.Headers;
    return Json(headers);
}

```

نکته: بهتر است از استفاده مستقیم از **HttpContext** خودداری کنید، زیرا تست کردن را سخت خواهد کرد.
در ادامه از یک مکانیزم **friendlier** توکار در **MVC** استفاده خواهیم کرد.

اجازه دهید یک **Model** به پروژه اضافه کنیم:

```

namespace OdeToFood.Models
{
    public class Restaurant
    {
        public int Id { get; set; }
    }
}

```

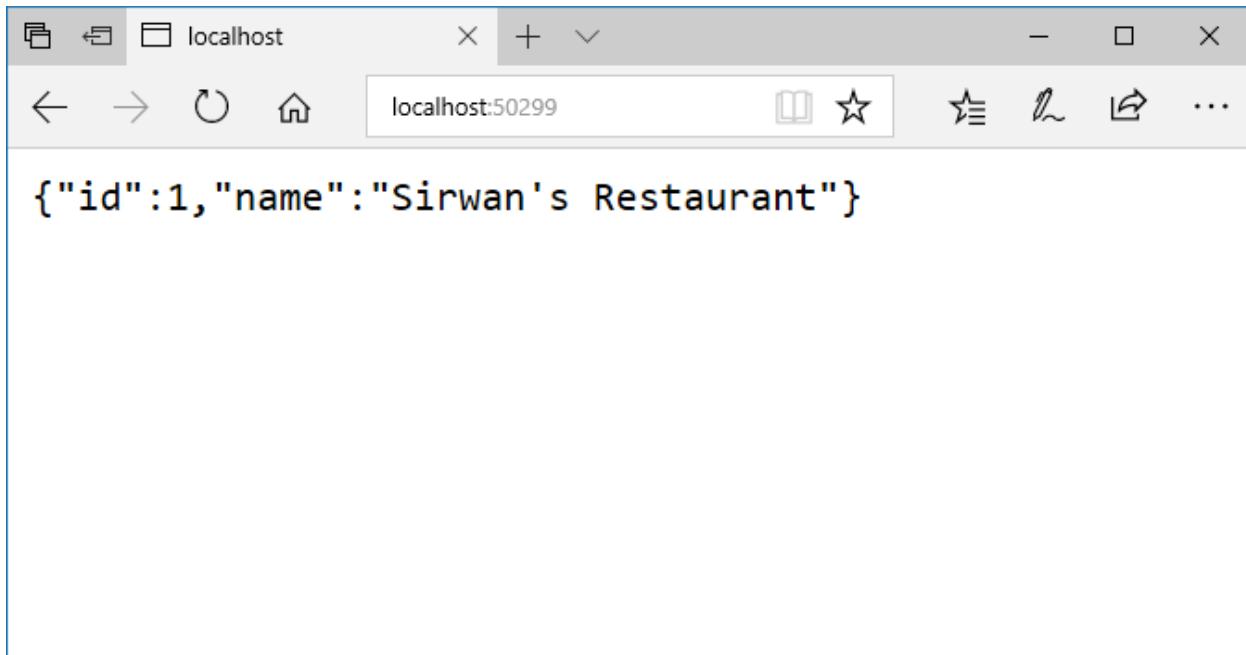
```
        public string Name { get; set; }
    }
}
```

اکنون میخواهیم اطلاعات رستوران را به عنوان خروجی اکشن متدها قبل تعیین کنیم:

```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        var model = new Restaurant { Id = 1, Name = "Sirwan's Restaurant" };

        return new ObjectResult(model);
    }
}
```

تا اینجا کنترلر دیدی ندارد که خروجی این **ObjectResult** دقیقاً چه چیزی خواهد بود. تنها چیزی که می‌داند این است که اطلاعاتی که میخواهیم در خروجی ظاهر شوند، اطلاعات یک رستوران است. مولفه‌ی دیگر در قرار است نوع این **ObjectResult** را تشخیص دهد. اگر پروژه را اجرا کنید خواهد دید که خروجی به صورت JSON می‌باشد:



نکته: کار کنترلر: درخواست به اکشن متده وارد خواهد شد، اطلاعات از منبع داده (دیتابیس، وب سرویس و...) دریافت خواهند شد، از این اطلاعات جهت ایجاد مدل استفاده خواهد شد، در نهایت خروجی را توسط یکی از **IActionResult**ها مشخص خواهیم کرد.

Rendering Views

در MVC یکی از رایج‌ترین روش‌های تولید HTML استفاده از razor view engine است. برای استفاده از این قابلیت اکشن متد باید یک شیء **IActionResult** (این شیء اینترفیس **ViewResult** را پیاده‌سازی کرده است) تولید کند. **ViewResult** نیز نام یک ویوی خاص را به همراه دارد؛ این ویو بر روی دیسک ذخیره شده است؛ همچنین یک مدل را برای استفاده در ویو نیز منتقل خواهد کرد.

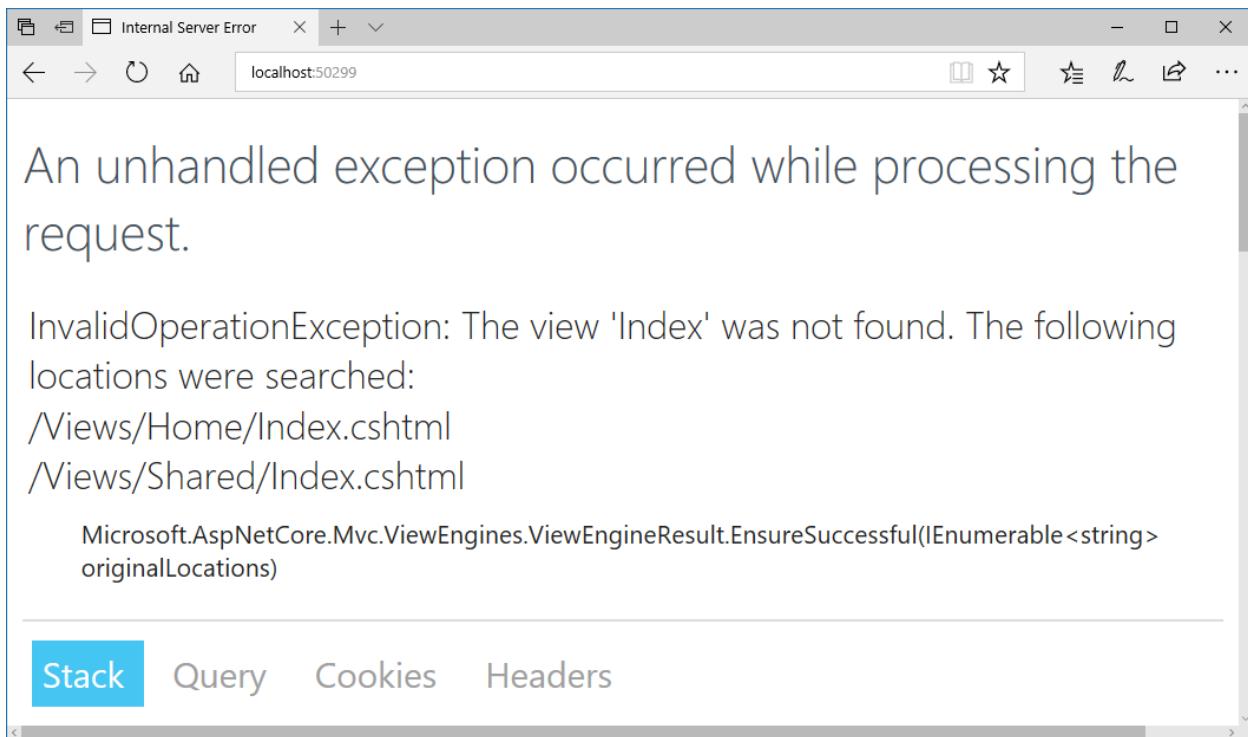


به عنوان مثال با داشتن اکشن متد زیر:

```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        var model = new Restaurant { Id = 1, Name = "Sirwan's Restaurant" };

        return View(model);
    }
}
```

خطای زیر را دریافت خواهیم کرد؛ زیرا **Index** با نام **View** ای باشد؛ اکشن متد فوق وجود ندارد:

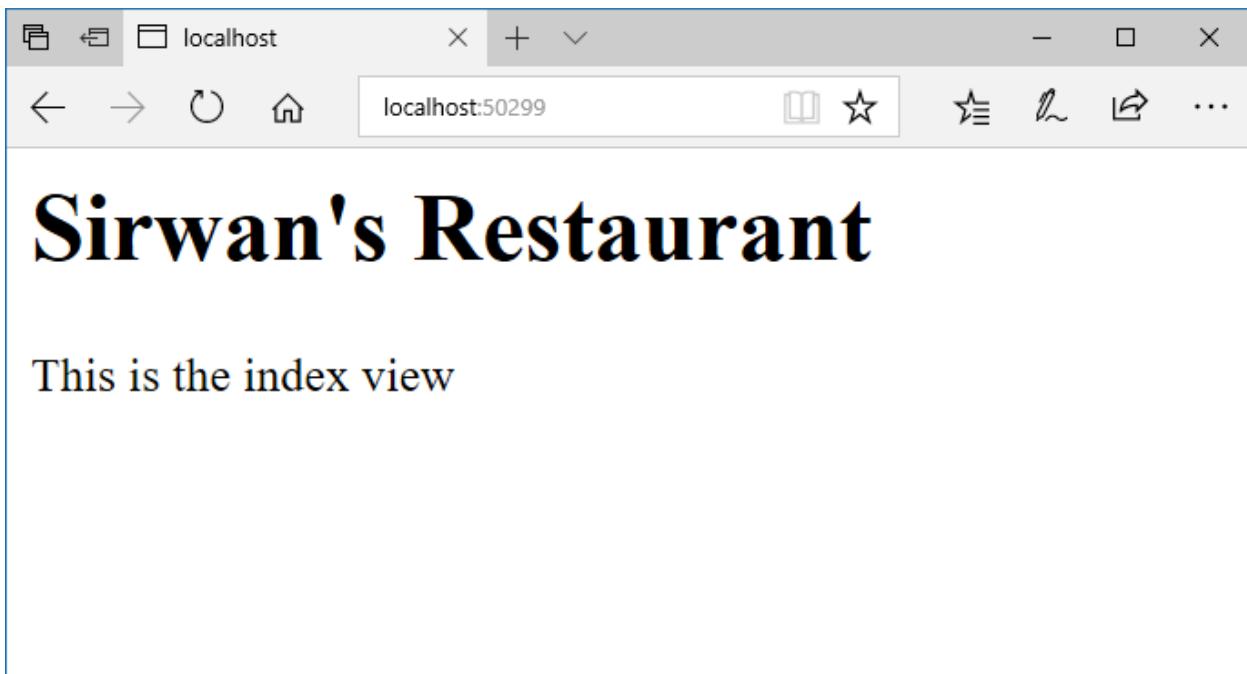


برای حل این مشکل باید ویوی موردنظر را در یکی از مکان‌های اشاره شده در خطای فوق ایجاد کنیم.

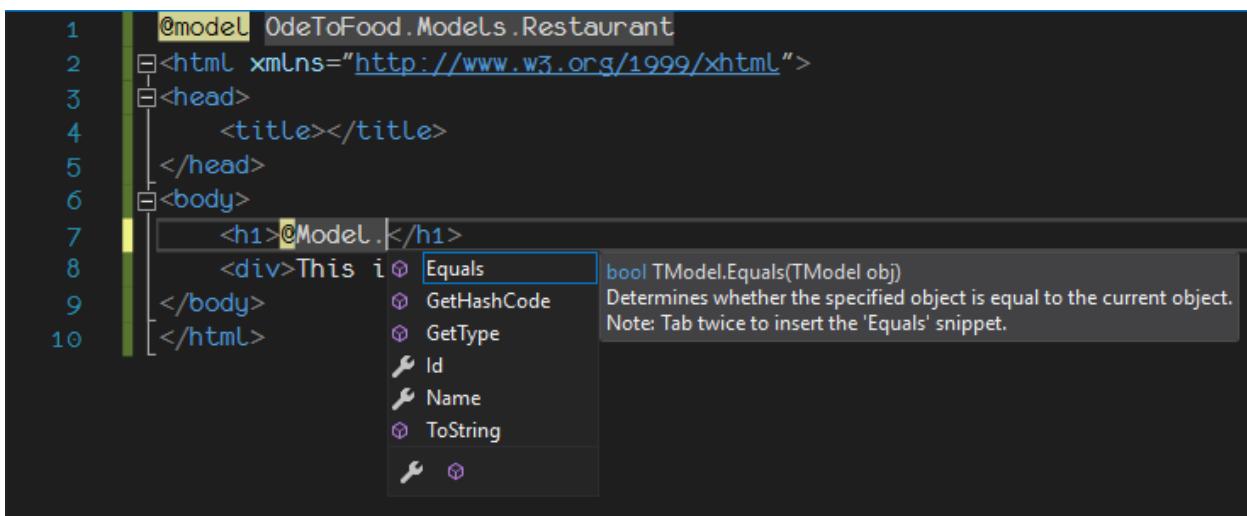
:Index ویوی

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <h1>@Model.Name</h1>
    <div>This is the index view</div>
</body>
</html>
```

خروجی:



نکته: درون ویو اگر دقت کرده باشد Intellisense برای نمایش پراپرتی های مدل وجود ندارد. برای داشتن این قابلیت باید از دایرکتیو **@model** و همچنین تعیین نوع مدل درون ویو استفاده کنیم:



A Table Full of Restaurants

می خواهیم درون ویو Index لیستی از رستوران ها را نمایش دهیم. برای اینکار نیاز به یک سرویس داریم؛ این سرویس می تواند دیتابیس باشد اما برای سادگی می خواهیم از یک لیست درون حافظه ایی برای اینکار استفاده کنیم:

```

namespace OdeToFood.Services
{
    public interface IRestaurantData
    {
        IEnumerable<Restaurant> GetAll();
    }
    public class InMemoryRestaurantData : IRestaurantData
    {
        public InMemoryRestaurantData()
        {
            _restaurants = new List<Restaurant>
            {
                new Restaurant { Id = 1, Name = "Sirwan's Pizza Place" },
                new Restaurant { Id = 2, Name = "Tersiguels" },
                new Restaurant { Id = 3, Name = "King's Contrivance" }
            };
        }
        List<Restaurant> _restaurants;

        public IEnumerable<Restaurant> GetAll()
        {
            return _restaurants.OrderBy(x => x.Name);
        }
    }
}

```

نکته: هر زمانیکه از لیست استفاده می کنید دقت داشته باشید که **List<T>** به صورت **thread-safe** نیست؛ بنابراین باید در حین استفاده از آن در یک اپلیکیشن وب مواظب باشید؛ به خصوص زمانیکه می خواهیم این سرویس را در بین چندین درخواست به اشتراک بگذارید.

اکنون از سرویس فوق می توانیم درون کنترلرهایمان استفاده کنیم اما نمی خواهیم این سرویس را به صورت مستقیم و هله سازی کنیم. یعنی نمی خواهیم کنترلرمان اطلاعی از وجود **concrete class**؛ یعنی کلاس **InMemoryRestaurant** داشته باشد. در عوض می خواهیم کنترلرمان با اینترفیس **IRestaurant** در ارتباط باشد (**programming to interfaces**). برای اینکار، کار و هله سازی سرویس را به عهده **IoC** خواهیم گذاشت:

```
services.AddScoped<IRestaurantData, InMemoryRestaurantData>();
```

اکنون درون کنترلر خواهیم داشت:

```
public class HomeController : Controller
{
    private IRestaurantData _restaurantData;

    public HomeController(IRestaurantData restaurantData)
    {
        _restaurantData = restaurantData;
    }
    public IActionResult Index()
    {
        var model = _restaurantData.GetAll();

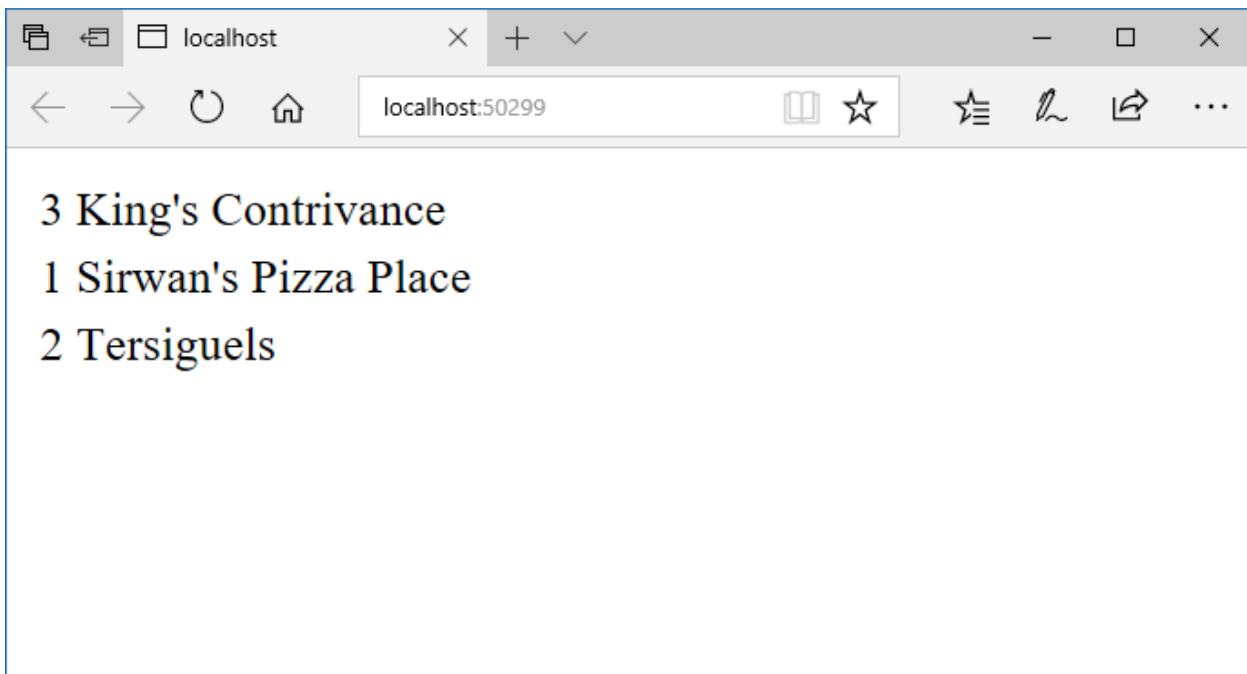
        return View(model);
    }
}
```

در اینحالت `HomeController` هیچ اطلاعی از `concrete service` که اینترفیس `IRestaurantData` را پیاده‌سازی کرده است، ندارد.

ویو:

```
@model IEnumerable<OdeToFood.Models.Restaurant>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <table>
        @foreach (var restaurant in Model)
        {
            <tr>
                <td>@restaurant.Id</td>
                <td>@restaurant.Name</td>
            </tr>
        }
    </table>
</body>
</html>
```

خروجی:



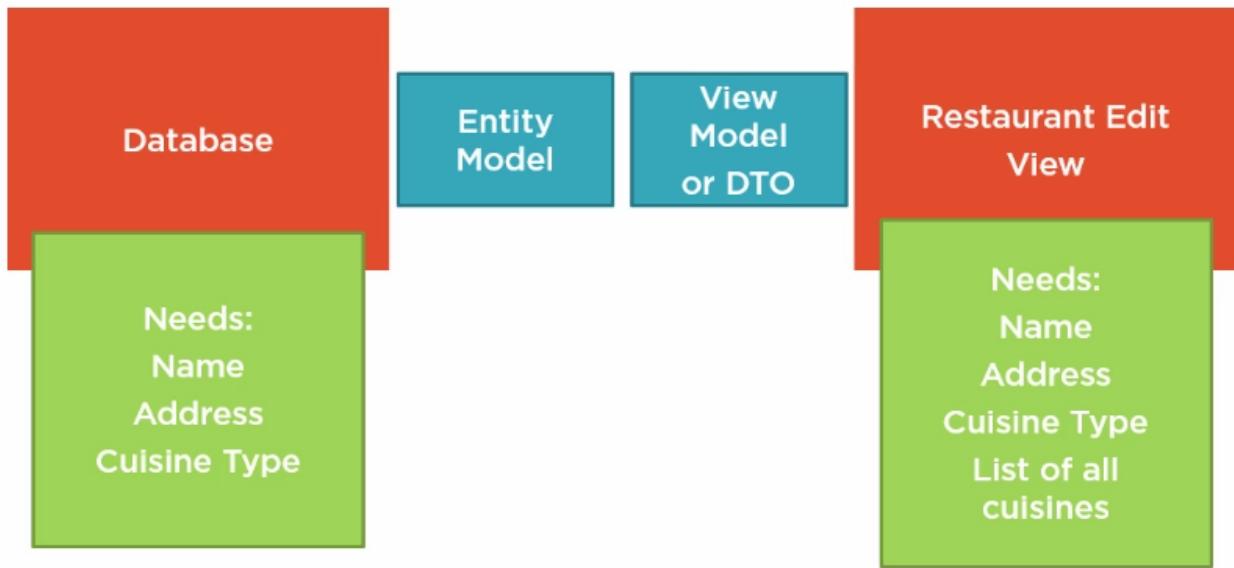
زیبایی این مثال این است که اگر بعداً به استفاده از **SQL Server** سوئیچ کنیم، نیازی به تغییر هیچکدام از ویوها و کنترلرهایمان نخواهیم داشت.

Models in the MVC Framework Models and View Models

فرض کنید یک اپلیکیشن دارید که قرار است یکسری اطلاعات را ذخیره کند. وقتی چیزی مانند یک رستوران را درون دیتابیس ذخیره می‌کنید، دیتابیس نیاز به یکسری اطلاعات خاص در مورد رستوران دارد؛ نام – آدرس – نوع غذا و...

در جایی درون اپلیکیشن تان نیز نیاز خواهد داشت که اطلاعات یک رستوران خاص را ویرایش کنید. ویوی **edit** نیز نیاز به یکسری اطلاعات دارد؛ نام – آدرس – نوع غذا – لیست تمامی غذاها و...

همانطور که مشاهده می‌کنید ویوی **edit** نیاز به اطلاعات بیشتری نیست به فیلدهای درون دیتابیس دارد.



برای مثال در ویوی edit ممکن است یک لیست برای انتخاب نوع غذا داشته باشیم؛ بنابراین در حین کوئری گرفتن از دیتابیس برای ویرایش یک رستوران، اطلاعات یک رستوران همراه با نوع غذای فعلی دریافت خواهیم کرد نه تمام غذاهای احتمالی. در این شرایط از یکی از دو model objects زیر می‌توانیم استفاده کنیم:

Entity Model -
یک موجودیت یا اینتیتی در واقع شیءی است که درون دیتابیس ذخیره خواهد شد.
این مدل معمولاً شبیه به اسکیمای دیتابیس است.

View Model or DTO -
شیء است که اطلاعات را از کنترلر به ویو منتقل خواهد کرد. این شیء نه تنها حاوی اطلاعات اینتیتی است بلکه حاوی اطلاعات موردنیاز ویو نیز خواهد بود. به این دلیل است که معمولاً به ویو مدل (DTO) (Data Transfer Object) گفته می‌شود. درون دیتابیس ذخیره نخواهد شد، بلکه کار انتقال اطلاعات را از ویو به مدل و یا بر عکس انجام خواهد داد.

مثال: فرض کنید برای مثال قبل، صفحه اصلی علاوه بر لیست رستوران‌ها می‌خواهیم پیام امروز را نیز نمایش دهیم؛ که در واقع از دیتاسورس دیگری خواهد آمد؛ در حال حاضر صفحه Index تنها تعدادی رستوران را به ویو انتقال می‌دهد اما اکنون می‌خواهیم همراه با این لیست یک مقدار دیگر را نیز به ویو جهت نمایش ارسال کنیم. در اینحالت به یک special purpose view model class استفاده خواهیم داشت؛ کلاسی که به صورت اختصاصی برای اکشن Index استفاده خواهد شد و تمامی اطلاعاتی که ویوی آن نیاز خواهد داشت را با خود ارسال می‌کند.

قدم اول: ایجاد ویو مدل:

```
using OdeToFood.Models;
```

```
using System.Collections.Generic;

namespace OdeToFood.ViewModels
{
    public class HomeIndexViewModel
    {
        public IEnumerable<Restaurant> Restaurants { get; set; }
        public string CurrentMessage { get; set; }
    }
}
```

قدم دوم: استفاده از ویو مدل فوق درون اکشن متدها:

```
using Microsoft.AspNetCore.Mvc;
using OdeToFood.Services;
using OdeToFood.ViewModels;

namespace OdeToFood.Controllers
{
    public class HomeController : Controller
    {
        private IRestaurantData _restaurantData;
        private IGreeter _greeter;

        public HomeController(IRestaurantData restaurantData
            , IGreeter greeter)
        {
            _restaurantData = restaurantData;
            _greeter = greeter;
        }

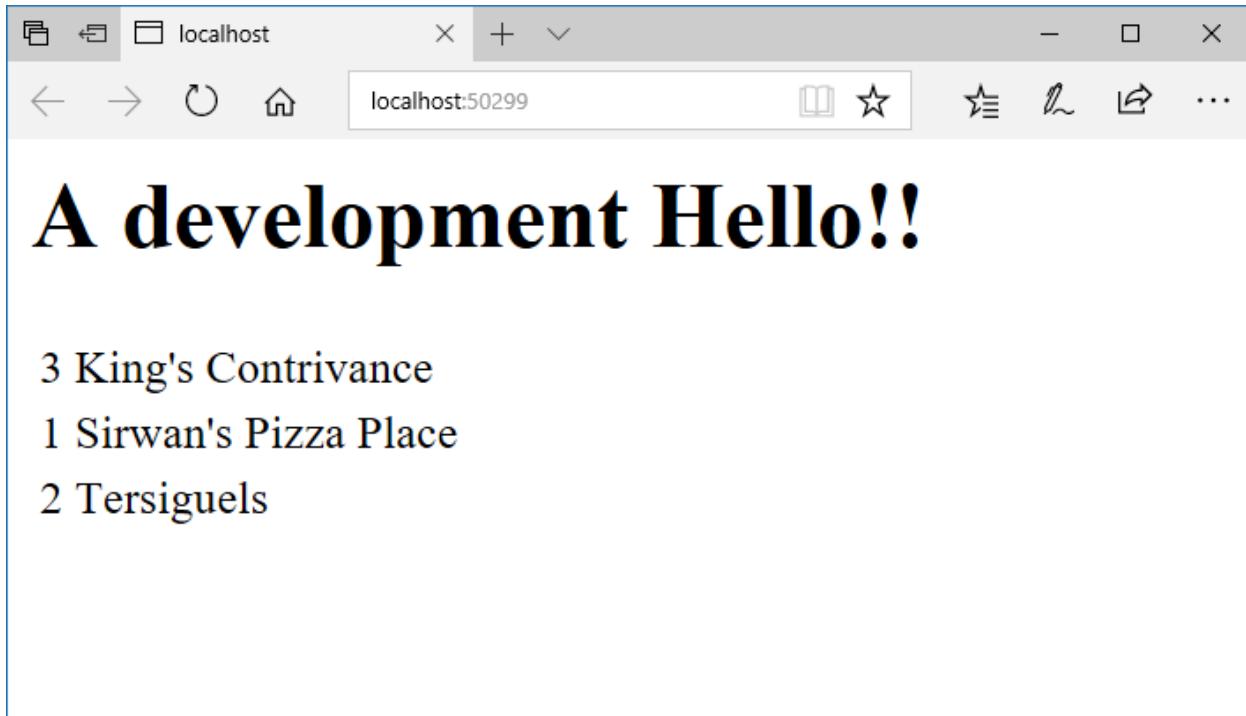
        public IActionResult Index()
        {
            var model = new HomeIndexViewModel
            {
                Restaurants = _restaurantData.GetAll(),
                CurrentMessage = _greeter.GetMessageOfDay()
            };

            return View(model);
        }
    }
}
```

قدم دوم: استفاده از ویو مدل فوق درون ویو:

```
@model OdeToFood.ViewModels.HomeIndexViewModel
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <h1>@Model.CurrentMessage</h1>
    <table>
        @foreach (var restaurant in Model.Restaurants)
        {
            <tr>
                <td>@restaurant.Id</td>
                <td>@restaurant.Name</td>
            </tr>
        }
    </table>
</body>
</html>
```

خروجی:



Detail a Restaurant

قدم اول: تعریف اکشن متدها:

```
public IActionResult Details(int id)
{
    var model = _restaurantData.Get(id);
    return View(model);
}
```

:Get تعریف متده

```
public interface IRestaurantData
{
    IEnumerable<Restaurant> GetAll();
    Restaurant Get(int id);
}
public class InMemoryRestaurantData : IRestaurantData
{
    // as before

    public Restaurant Get(int id)
    {
        return _restaurants.FirstOrDefault(r => r.Id == id);
    }
}
```

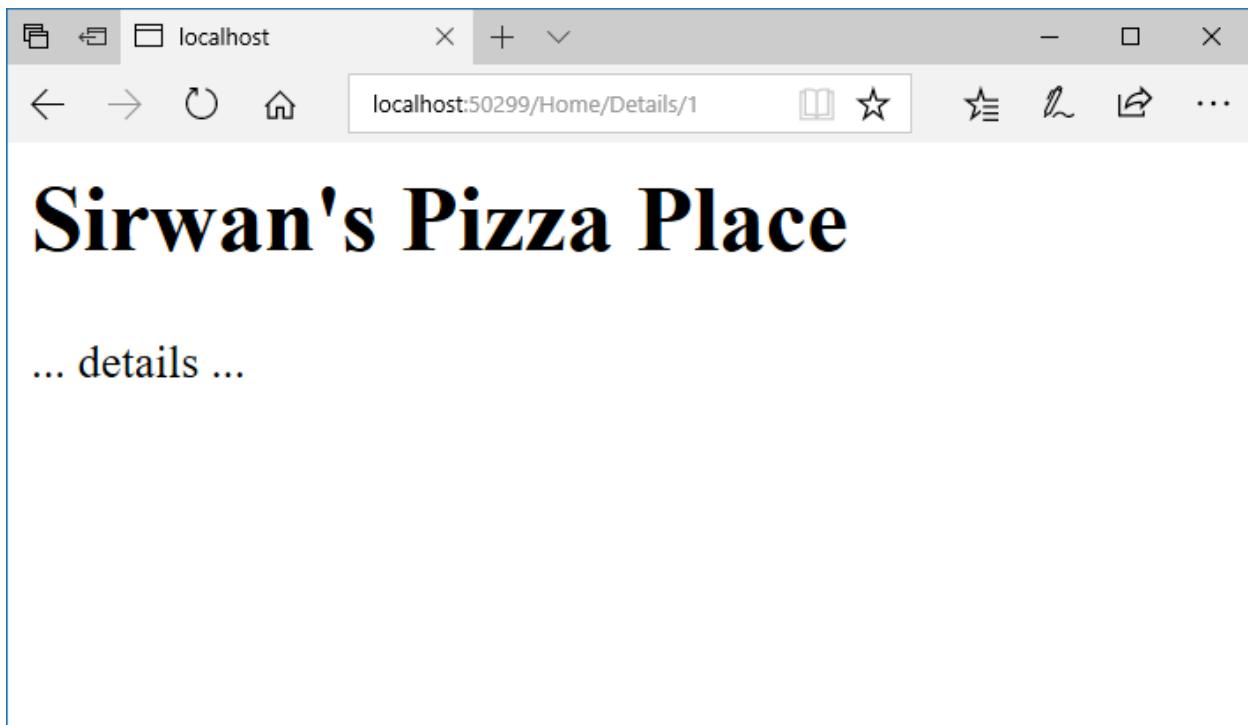
:Details تعریف ویوی سوم قدم

```
@model OdeToFood.Models.Restaurant

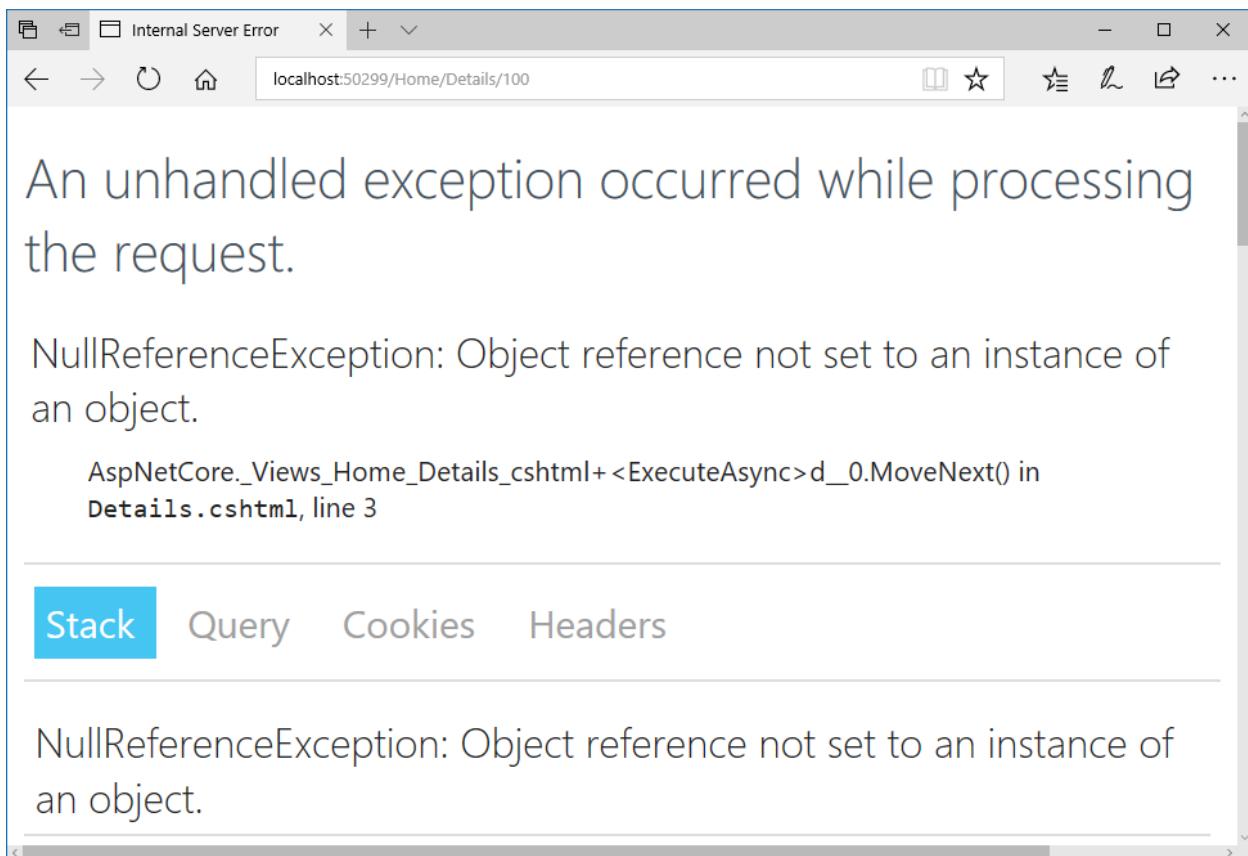
<h1>@Model.Name</h1>

<div> ... details ... </div>
```

خروجی:



اگر یک آی‌دی را وارد کنیم که وجود نداشته باشد خطای زیر را دریافت خواهیم کرد:



زیرا خروجی **FirstOrDefault** در صورتیکه آی دی مورد نظر را پیدا نکند null خواهد بود. برای حل این مشکل چندین استراتژی وجود دارد:

- بررسی null بودن Model درون ویو:

```
- @model OdeToFood.Models.Restaurant  
-  
- @if (Model == null)  
- {  
-     <div>Not found</div>  
- }  
- else  
- {  
-     <h1>@Model.Name</h1>  
-  
-     <div> ... details ... </div>  
- }
```

- بررسی null بودن درون اکشن متدها:

```
- public IActionResult Details(int id)
- {
-     var model = _restaurantData.Get(id);
-     if (model == null)
-     {
-         return RedirectToAction("Index");
-     }
-     return View(model);
- }
```

برای دسترسی سریع‌تر به صفحه‌ی Details می‌توانیم لینک آن را درون صفحه‌ی اصلی قرار دهیم؛ به چند روش می‌توانیم اینکار را انجام دهیم:

HTML ساده -

```
- @foreach (var restaurant in Model.Restaurants)
{
    <tr>
        <td>@restaurant.Id</td>
        <td>@restaurant.Name</td>
        <td>
            <a href="/home/details/@restaurant.Id"></a>
        </td>
    </tr>
}
```

استفاده از Razor -

```
- @foreach (var restaurant in Model.Restaurants)
{
    <tr>
        <td>@restaurant.Id</td>
        <td>@restaurant.Name</td>
        <td>
            @Html.ActionLink("Details", "Details", new { Id =
    restaurant.Id })
        </td>
    </tr>
}
```

استفاده از **ActionLink** کمی جالب نیست زیرا خوانایی آن ساده نیست؛ پس یک روش سوم را معرفی خواهیم کرد؛ اما برای استفاده از این روش باید ابتدا فایلی تحت عنوان `_ViewImports.cshtml` را به پوشه اضافه کنیم:

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

با اینکار **TagHelper** به تمامی ویوهایمان اضافه خواهد شد و می‌توانیم از آن استفاده کنیم.

استفاده از **TagHelper** -

```
- @foreach (var restaurant in Model.Restaurants)
- {
-     <tr>
-         <td>@restaurant.Id</td>
-         <td>@restaurant.Name</td>
-         <td>
-             <a asp-action="Details" asp-route-
- id="@restaurant.Id">Details</a>
-         </td>
-     </tr>
- }
```

Create a Restaurant

در ادامه نحوه ایجاد یک رستوران را بررسی خواهیم کرد.



قدم اول: تعریف اکشن متدها:

```
public IActionResult Create()
{
    return View();
}
```

قدم دوم: تعریف ویو Create

```
@using OdeToFood.Models  
@model OdeToFood.Models.Restaurant  
  
<h1>Create</h1>  
  
<form method="post">  
  
    <input asp-for="Name" />  
  
    <select asp-for="Cuisine"  
            asp-items="@Html.GetEnumSelectList<Cuisine>()">  
    </select>  
  
    <input type="submit" name="save" value="Save" />  
</form>
```

Accepting Form Input

پر اپری ها براساس ویژگی Name مپ خواهند شد:



دریافت داده های ارسالی از فرم:

قدم اول: تعریف اکشن متدها:

```
[HttpPost]
public IActionResult Create(RestaurantEditModel model)
{
    var newRestaurant = new Restaurant();
    newRestaurant.Name = model.Name;
    newRestaurant.Cuisine = model.Cuisine;

    newRestaurant = _restaurantData.Add(newRestaurant);

    return View("Details", newRestaurant);
}
```

قدم دوم: افزودن متده Add:

```
public Restaurant Add(Restaurant restaurant)
{
    restaurant.Id = _restaurants.Max(r => r.Id) + 1;
    _restaurants.Add(restaurant);
    return restaurant;
}
```

نکته: در سناریوی فوق کاربر با ایجاد یک آیتم جدید و کلیک بر روی کلید Create باید آیتم جدید را درون صفحه Details مشاهده کند؛ اما اگر پروژه را اجرا کنید خواهید دید که بعد از ایجاد آیتم جدید و مراجعه به لیست آیتم، آیتم جدیدی وجود ندارد. دلیل آن نیز این است که برای سرویس IRestaurantData از حالت Scoped استفاده کرده‌ایم؛ اینحالت یعنی یک وله از InMemoryRestaurantData را برای هر درخواست HTTP ایجاد کن؛ در طول درخواست از این وله استفاده خواهد شد اما با پایان درخواست این وله از حافظه پاک خواهد شد. برای حل این مشکل باید طول عمر را به Singleton تغییر دهیم:

```
services.AddSingleton<IRestaurantData, InMemoryRestaurantData>();
```

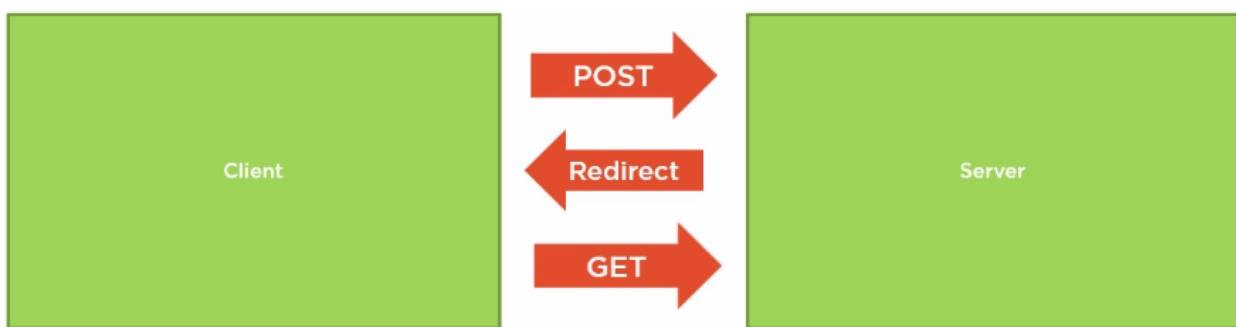
POST Redirect GET Pattern

تا اینجا کلاینت؛ یعنی مرورگر ما توسط یک فرم با اکشن post اطلاعات را (یک رستوران) به سمت سرور ارسال خواهد کرد؛ در نهایت سرور بلافصله اطلاعات رستوران جدید را در یک صفحه دیگر نمایش خواهد داد. اما پاسخ دادن به یک عملیات POST درون یک صفحه‌ی HTML می‌تواند باعث بروز مشکلاتی در اپلیکیشن شود؛ زیرا

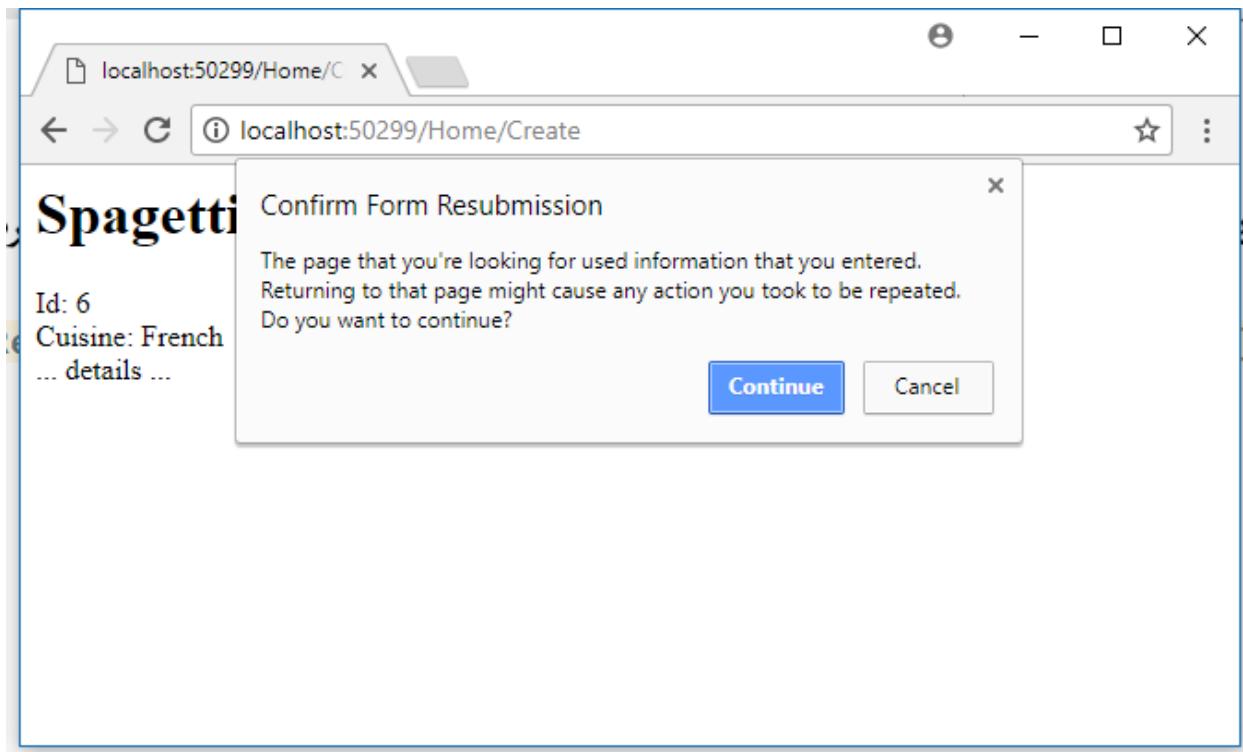
اگر کاربر تصمیم بگیرد مرورگر را ریفرش کند، مرورگر سعی خواهد کرد یک درخواست دیگر به سرور ارسال کند؛
که اینکار باعث ثبت یک رستوران جدید در دیتابیس خواهد شد.

به عنوان یک best practice در وب اپلیکیشن‌ها بهتر است بعد از یک عملیات POST به این عملیات با یک GET request پاسخ دهید؛ یعنی به مرورگر بگوئید که یک http status redirect code را برای خواندن داده‌ها ارسال کند. بخاطر داشته باشید که POST برای عملیات نوشتن است و GET برای عملیات خواندن است.

الگوی **POST Redirect GET** در نهایت یک صفحه را در اختیار کاربر قرار خواهد داد که توسط کاربر خوانده شود؛ یعنی صفحه‌ایی که قابلیت ریفرش شدن و بوکمارک کردن را داشته باشد:



همانطور که در مثال قبل دیدید بعد از ثبت رستوران جدید ویوی Details را در خروجی رندر خواهیم کرد؛ در این صفحه اگر ریفرش کنیم با پیغام زیر مواجه خواهیم شد که با کلیک بر روی Continue اطلاعات تکراری درون دیتابیس ذخیره خواهد شد:



برای حل این مشکل به جای return کردن یک ویو، کاربر را به یک اکشن متدهای خواهیم کرد:

```
return RedirectToAction("Details", new { id = newRestaurant.Id });
```

Model Validation with Data Annotations

تعدادی از Data Annotations موجود در .NET

Name	Purpose
MinLength / MaxLength	Enforce length of strings
Range	Enforce min and max numbers
RegularExpression	Make a string match a pattern
Display	Set the name and formatting string
DataType	Render as password or email input
Required	Model value is mandatory
Compare	Commonly used for password validation

افزودن Data Annotation به مدل و مجبور کردن Model Validation

برای اینکه دیگر annotations به خوبی کار کنند، بهتر است در ویو از هلپر EditorFor که یک نسخه‌ی جنریک‌تر از TextBoxFor است، استفاده کنیم. این هلپر براساس annotations یک تگ مناسب را تولید خواهد کرد.

```
public class Restaurant
{
    public int Id { get; set; }

    [Display(Name = "Restaurant Name")]
    [Required, MaxLength(80)]
    public string Name { get; set; }
    public Cuisine Cuisine { get; set; }
}
```

خروجی:

The screenshot shows a browser window with the URL `localhost:50299/Home/Create`. The page title is 'Create'. There is a form with two fields: 'Restaurant Name' (text input) and 'Cuisine' (dropdown menu). The 'Name' field is highlighted with a red border and contains the placeholder 'None'. Below the form is a 'Save' button. The browser's developer tools are open, specifically the Elements tab, which displays the HTML structure of the page. The 'input' element for the 'Name' field has several validation attributes: `data-val="true"`, `data-val-maxlength="80"`, and `data-val-maxlength-max="80"`. A red box highlights these attributes. A message 'The Name field is required.' is displayed above the input field. The developer tools also show a red box highlighting the message 'The Name field must be a string or array type with a maximum length of 80.' and another red box highlighting the message 'The Restaurant Name field is required.' in the error span element.

این اتربیوت‌های خاص برای اعمال client side validation مورد استفاده قرار می‌گیرند؛ چیزی که مورد استفاده ما است server side validation می‌باشد زیرا ما نباید هیچ وقت به کلاینت اعتماد کنیم.

MVC وقتی ویومدلمان را با داده‌های ارسالی توسط فرم validation rules کرد، populate را بر روی ویومدل اعمال خواهد کرد و بررسی خواهد کرد که آیا مدل دریافتی معتبر است یا خیر. وضعیت مدل را می‌توانیم توسط پرپری ModelState.IsValid بررسی کنیم:

```
[HttpPost]
public IActionResult Create(RestaurantEditModel model)
{
    if (ModelState.IsValid)
    {
        var newRestaurant = new Restaurant();
        newRestaurant.Name = model.Name;
        newRestaurant.Cuisine = model.Cuisine;

        newRestaurant = _restaurantData.Add(newRestaurant);

        return RedirectToAction("Details", new { id = newRestaurant.Id });
    }
    return View();
}
```

نمایش خطاهای در ویو:

```
@using OdeToFood.Models
@model OdeToFood.Models.Restaurant

<h1>Create</h1>

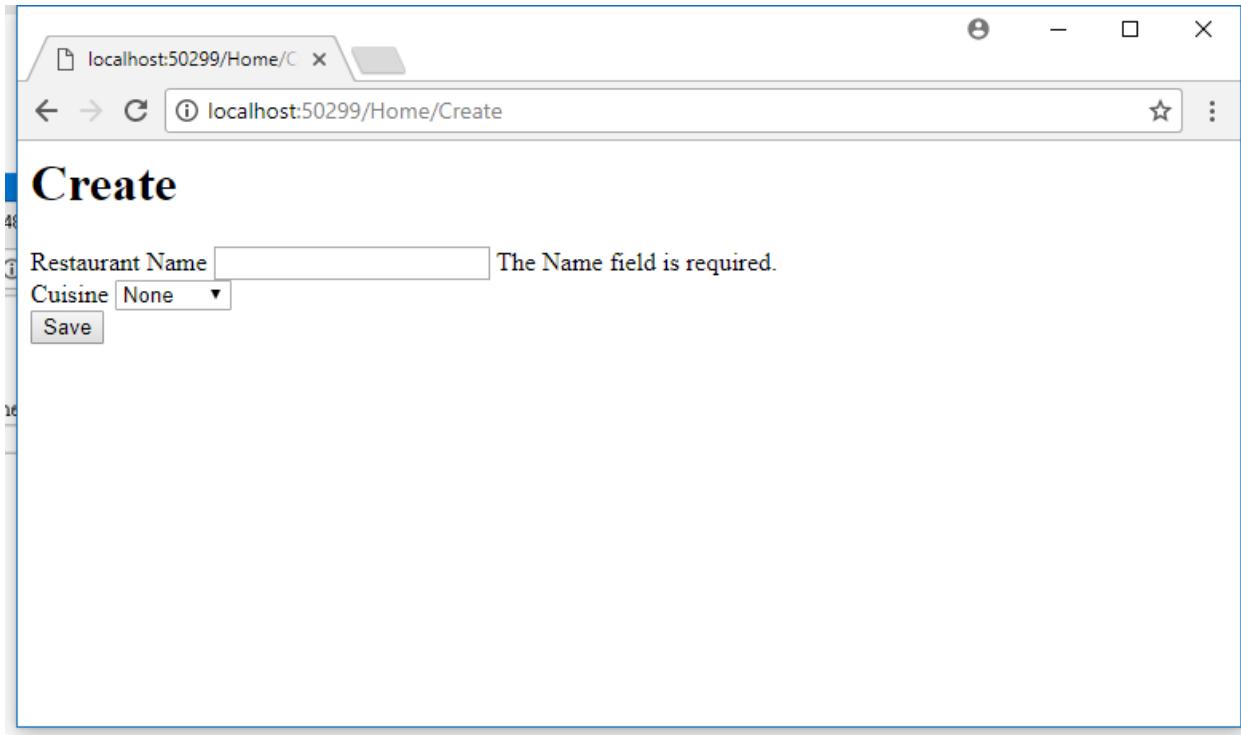
<form method="post">

    <div>
        <label asp-for="Name"></label>
        <input asp-for="Name" />
        <span asp-validation-for="Name"></span>
    </div>

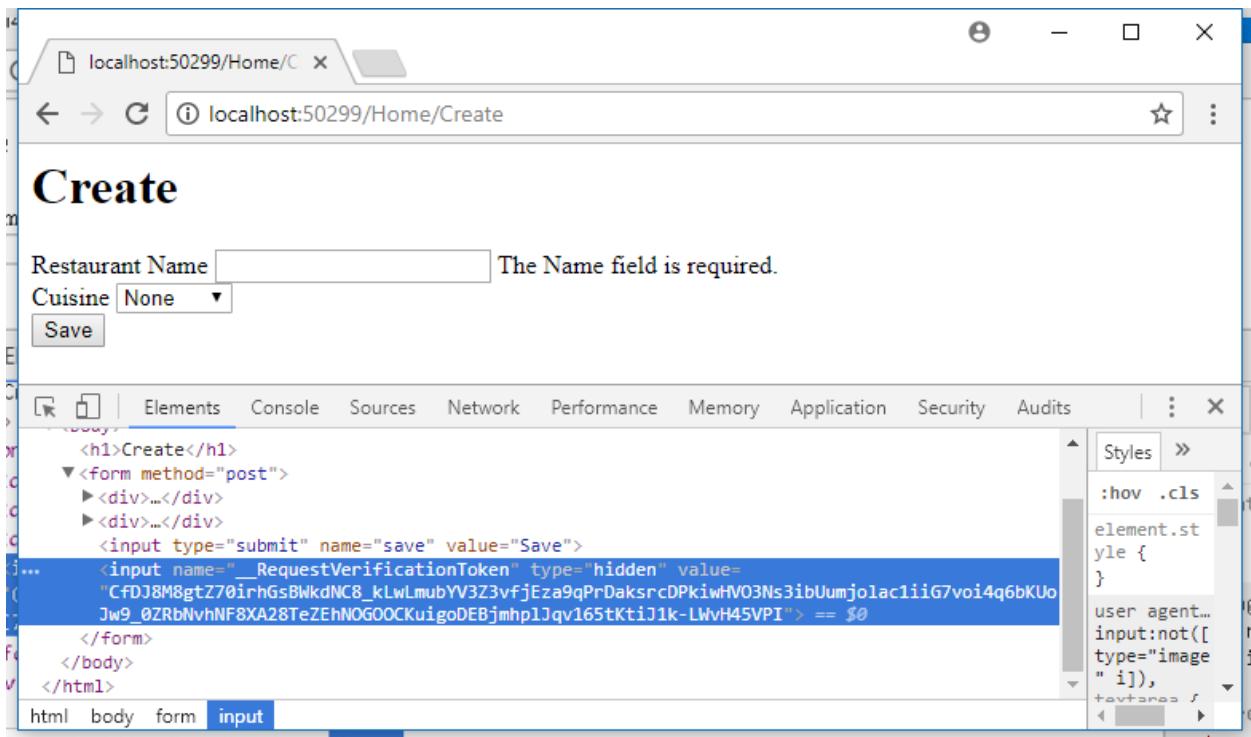
    <div>
        <label asp-for="Cuisine"></label>
        <select asp-for="Cuisine"
                asp-items="@Html.GetEnumSelectList<Cuisine>()"></select>
        <span asp-validation-for="Cuisine"></span>
    </div>
```

```
<input type="submit" name="save" value="Save" />  
</form>
```

خروجی:



وظیفه‌ی فیلد مخفی درون فرم با نام **RequestVerificationToken** چیست؟



مقدار این فیلد مخفی تضمین خواهد کرد که اطلاعاتی که کاربر برایمان ارسال کرده است از طریق فرمی است
که در اختیار کاربر قرار داده ایم نه از جایی دیگر یعنی از وقوع حمله‌ی **Forgery (CSRF)** جلوگیری خواهد کرد.

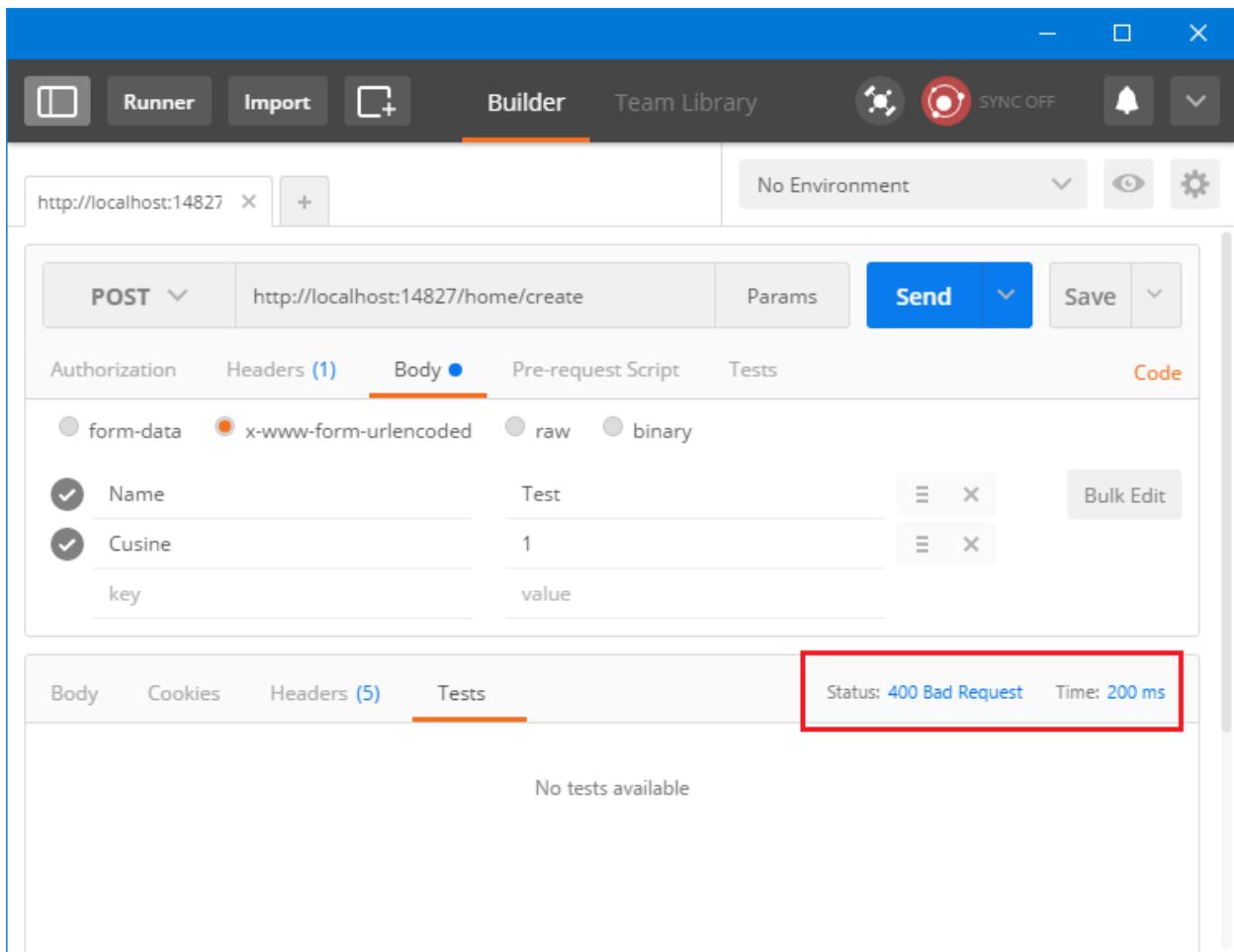
برای استفاده از این قابلیت باید فرم‌های POST‌تان را با اtribut زیر مزین نمائید:

```
[ValidateAntiForgeryToken]
```

مثال: اگر از ویژگی فوق استفاده نکنید به راحتی توسط هر ابزاری می‌توانید یک فرم را به سرور ارسال کنید:

The screenshot shows the Postman application interface. At the top, there are tabs for Runner, Import, and Builder, with Builder being the active tab. Below the tabs, the URL is set to `http://localhost:14827`. The request method is set to POST, and the endpoint is `http://localhost:14827/home/create`. The Body tab is selected, showing two form-data fields: "Name" (Test: 1) and "Cusine" (Test: 1). Other tabs include Headers (1), Pre-request Script, Tests, and Code. The response section shows a status of 200 OK and a time of 6077 ms, with a note that no tests are available.

اما با استفاده از ویژگی **ValidateAntiForgeryToken** هیچ کسی جز فرمی که در اختیار کاربر قرار دادهایم، قادر به ارسال اطلاعات به سرور نخواهد بود؛ به نوعی ارسال اطلاعات تنها توسط فرم خودمان قابل انجام است:



Using the Entity Framework Introduction

در این مژول می‌خواهیم اپلیکیشن‌مان اطلاعات را درون دیتابیس درج و بازیابی نمایید.

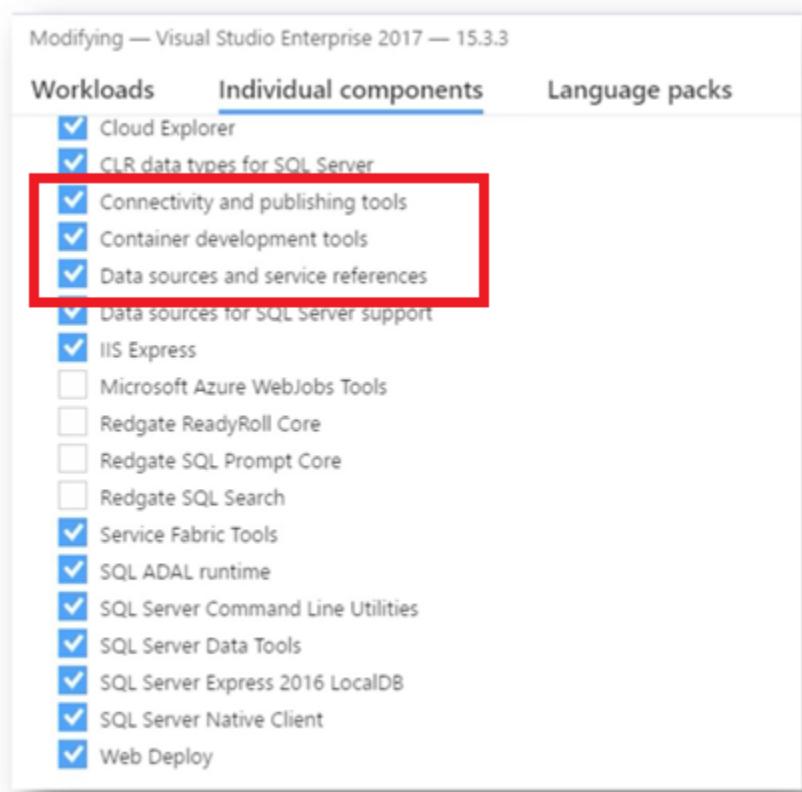
SQL Server LocalDB

یک نسخه از SQL Server LocalDB است که برای توسعه‌دهندگان بهینه شده است. این نسخه از SQL Server همراه با Visual Studio 2017 نصب خواهد شد. برای اینکه مطمئن شوید نصب شده است می‌توانید از دستور زیر استفاده کنید:

```
C:\Developer Command Prompt for VS 2017
C:\Users\Sirwan\source>sqllocaldb info
MSSQLLocalDB

C:\Users\Sirwan\source>
```

اگر LocalDB بر روی سیستم شما نصب نشده است؛ می‌توانید Visual Studio Installer را اجرا و آن را نصب کنید:



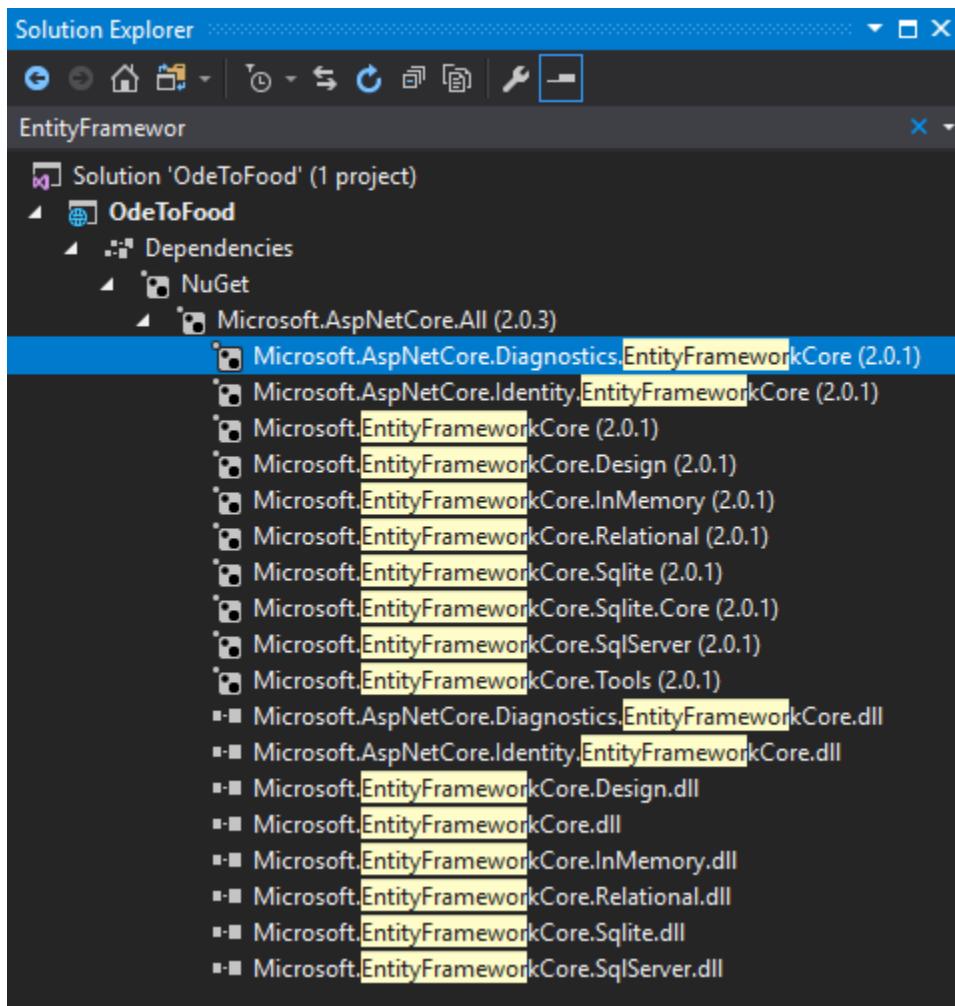
کانشکن استرینگ:

```
Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=master;
Integrated Security=True;Connect Timeout=30;
Encrypt=False;TrustServerCertificate=True;
ApplicationIntent=ReadWrite;MultiSubnetFailover=False
```

Installing the Entity Framework

:EF نصب

از آنجاییکه پروژه ما از متابکریج **Microsoft.AspNetCore.All** استفاده می‌کند، در نتیجه نیز درون این پکیج قرار دارد و نیازی به نصب مجدد آن نیست:



برای فعالسازی فرمان `dotnet ef` باید `package reference` زیر را نیز به پروژه اضافه کنیم:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

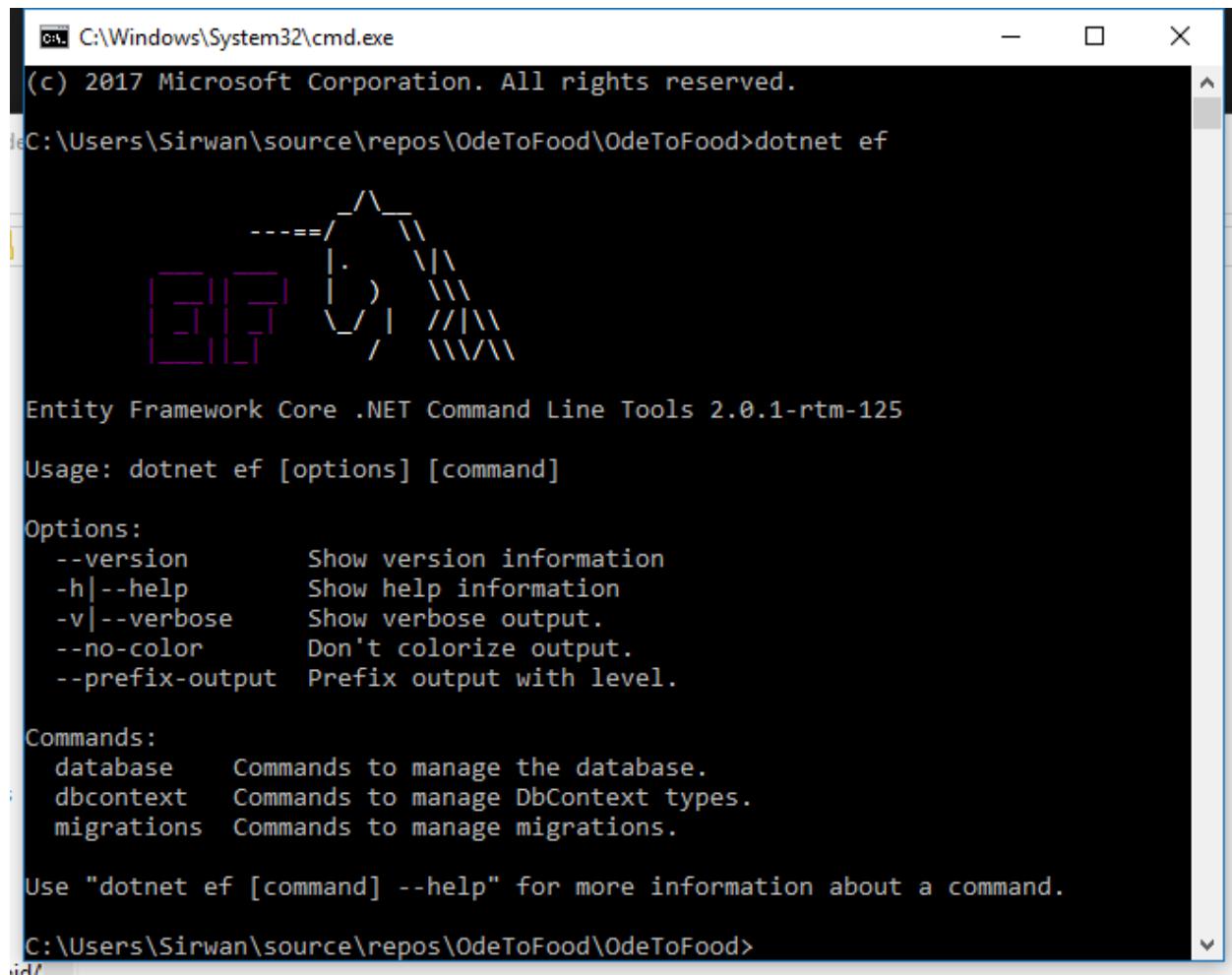
<PropertyGroup>
  <TargetFramework>netcoreapp2.0</TargetFramework>
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.3" />
</ItemGroup>

<ItemGroup>
  <DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet"
Version="2.0.1">
    </DotNetCliToolReference>
</ItemGroup>

</Project>
```

اکنون فرمان `dotnet ef` از طریق خط فرمان برای پروژه در دسترس می‌باشد:



```
C:\Windows\System32\cmd.exe
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Sirwan\source\repos\OdeToFood\OdeToFood>dotnet ef

Entity Framework Core .NET Command Line Tools 2.0.1-rtm-125

Usage: dotnet ef [options] [command]

Options:
  --version      Show version information
  -h|--help      Show help information
  -v|--verbose   Show verbose output.
  --no-color     Don't colorize output.
  --prefix-output Prefix output with level.

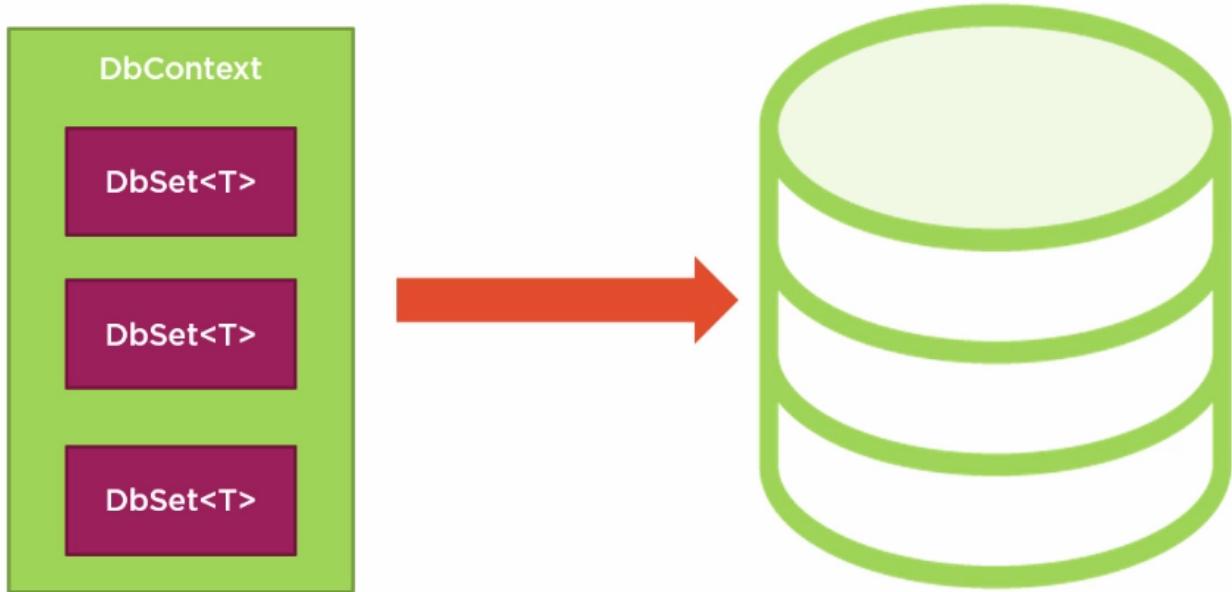
Commands:
  database    Commands to manage the database.
  dbcontext   Commands to manage DbContext types.
  migrations  Commands to manage migrations.

Use "dotnet ef [command] --help" for more information about a command.

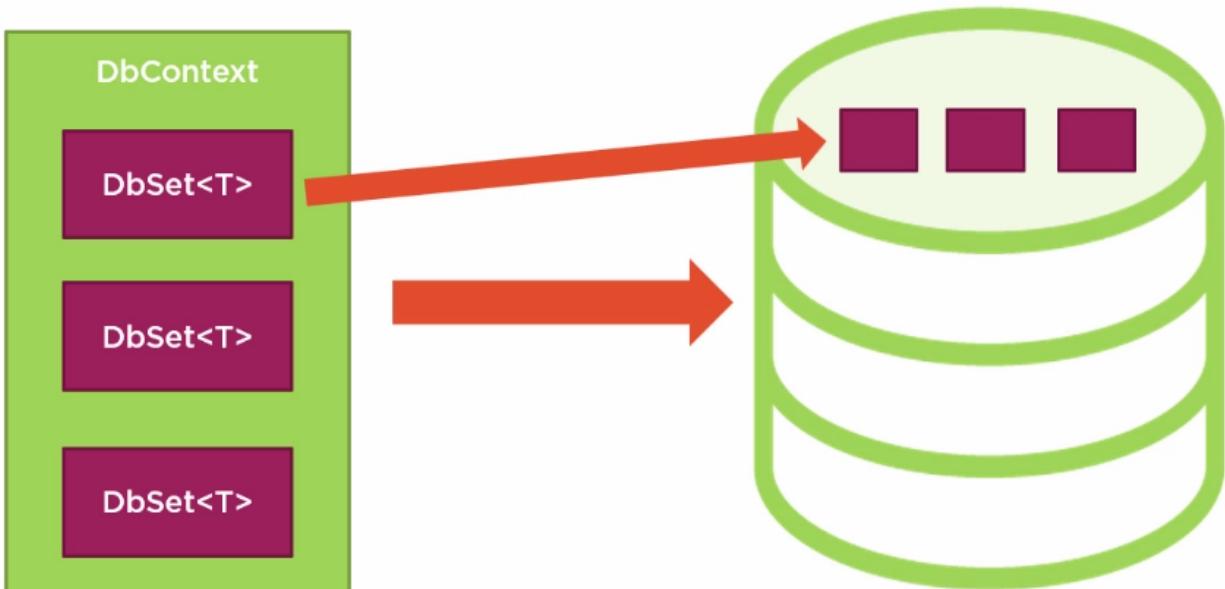
C:\Users\Sirwan\source\repos\OdeToFood\OdeToFood>
```

Implementing a DbContext

برای استفاده از EF باید یک کلاس ایجاد کنیم که از کلاس **DbContext** ارث بری کند. هر که ایجاد می‌کنیم اجازه‌ی دسترسی به یک دیتابیس را به ما می‌دهد. با تغییر `connection string` کلاس `context` مان می‌تواند به دیتابیس‌های فیزیکی مختلفی دسترسی داشته باشد.



درون کلاس `context` مان می‌توانیم یکسری `DbSet<T>` تعریف کنیم. هر `DbSet` درون کانتکست به یک جدول در دیتابیس `T` در واقع نوع `Restaurant` مانند `entity` ... است. در نگاشت داده خواهد شد:



قدم اول: ایجاد کلاس کانتکست:

```
using Microsoft.EntityFrameworkCore;
using OdeToFood.Models;

namespace OdeToFood.Data
{
    public class OdeToFoodDbContext : DbContext
    {
        public OdeToFoodDbContext(DbContextOptions options)
            : base(options)
        {

        }

        public DbSet<Restaurant> Restaurants { get; set; }
    }
}
```

قدم دوم: ایجاد یک پیاده‌سازی دیگر برای **:IRestaurantData**

```
public class SqlRestaurantData : IRestaurantData
{
    private OdeToFoodDbContext _context;

    public SqlRestaurantData(OdeToFoodDbContext context)
    {
        _context = context;
    }

    public Restaurant Add(Restaurant restaurant)
    {
        _context.Restaurants.Add(restaurant);
        _context.SaveChanges();
        return restaurant;
    }

    public Restaurant Get(int id)
    {
        return _context.Restaurants.FirstOrDefault(x => x.Id == id);
    }

    public IEnumerable<Restaurant> GetAll()
    {
```

```
        return _context.Restaurants.OrderBy(x => x.Name);  
    }  
}
```

نکته: فراخوانی متدهای **Add**, **Update**, **Delete** صورت بگیرد؛ در شرایطی ممکن است چندین دستور **Add**, **Update**, **Delete** را برای اشیاء دیگر طی یک درخواست نیاز داشته باشد؛ در این حالت بهتر است بعد از اتمام تمامی عملیات بر روی اشیاء متدهای **SaveChanges()** را فراخوانی کنید. (مطالعه بیشتر - الگوی واحد کار)

Configuring the Entity Framework Services

برای داشتن یک **DbContext** با قابلیت استفاده مجدد، باید مقداری **Configuration** اپلیکیشن را تغییر دهیم؛ باید به فریمورک بگوئیم که از چه **connection string**, **server** قرار است استفاده کنیم؛ از آنجائیکه همیشه تغییر می‌کند آن را درون **appsettings.json** ذخیره خواهیم کرد:

```
{  
  "Greeting": "Hello!!",  
  "ConnectionStrings": {  
    "DefaultConnection":  
      "Server=(localdb)\\MSSQLLocalDB;Database=OdeToFood;Trusted_Connection=True;MultipleActiveResultSets=true"  
  }  
}
```

در ادامه باید به جای سرویس **InMemoryRestaurantData** از سرویس **SqlRestaurantData** استفاده کنیم:

```
services.AddScoped<IRestaurantData, SqlRestaurantData>();
```

همانطور که ملاحظه می‌کنید برای این سرویس از **AddScoped** استفاده کرده‌ایم؛ زیرا **DbContext** در واقع به صورت **thread-safe** نیست؛ در اینحالت مطمئن خواهیم شد که **DbContext** تنها درون یک ترد در دسترس قرار خواهد گرفت.

در قدم بعدی یاد سرویس‌های موردنیاز EF را ریجستر کنیم:

```

private IConfiguration _configuration;

public Startup(IConfiguration configuration)
{
    _configuration = configuration;
}

public void ConfigureServices(IServiceCollection services)
{
    services.AddScoped<IGreeter, Greeter>();
    services.AddDbContext<OdeToFoodDbContext>(options =>
    {

options.UseSqlServer(_configuration.GetConnectionString("DefaultConnection"));
    });
    services.AddScoped<IRestaurantData, SqlRestaurantData>();
    services.AddMvc();
}

```

Entity Framework Migrations

یک روش برای ایجاد دیتابیس، وادار کردن EF به ساخت دیتابیس است؛ این فرآیند با دو مرحله انجام خواهد شد:

dotnet ef migrations add

dotnet ef database update

- اضافه کردن کدهای مهاجرت به پروژه

○ در واقع منظور از کدهای مهاجرت اجرای یکسری کد سی‌شارپ جهت ایجاد دیتابیس است.

می‌تواند کدهای مهاجرت را برایمان تولید کند؛ ابتدا به دیتابیس مراجعه کرده و در صورت وجود

آن را با اینتیلنی مطابقت خواهد داد و تغییرات اسکیما را از روی مدل به دیتابیس اعمال خواهد

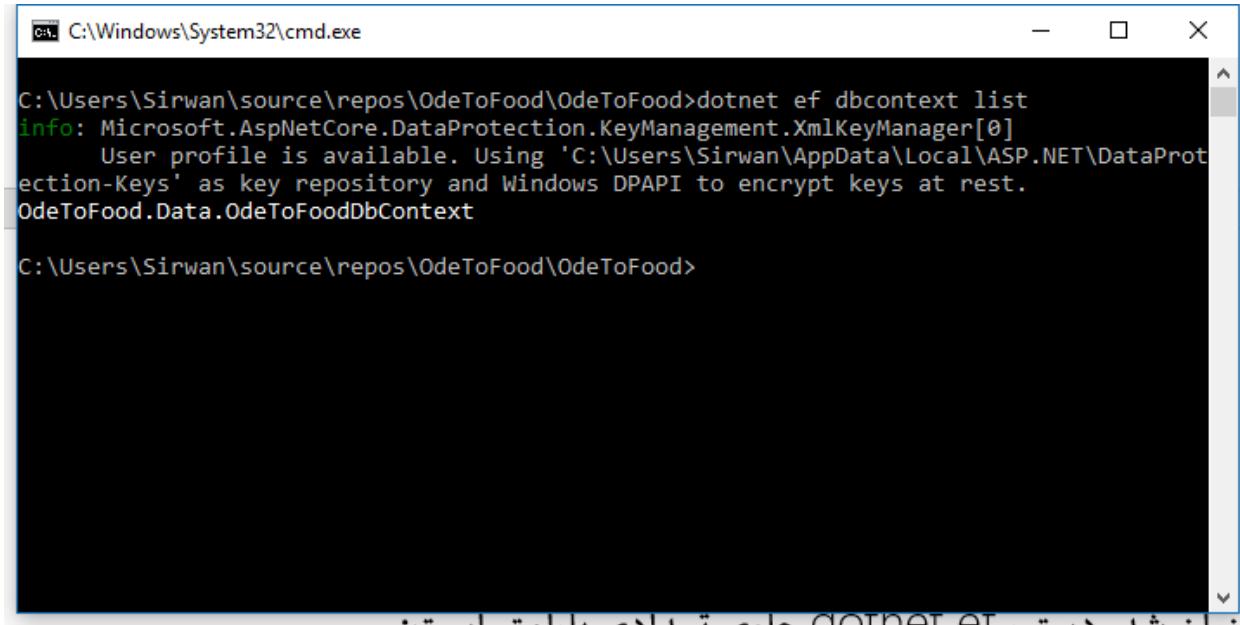
کرد. برای اضافه کردن migration باید از دستور **dotnet ef migrations add**

استفاده کنیم.

- اعمال دستور **dotnet ef database update** جهت اعمال تغییرات و بروزرسانی دیتابیس

همانطور که قبل نیز عنوان شد، دستور `dotnet ef` حاوی تعدادی پارامتر است:

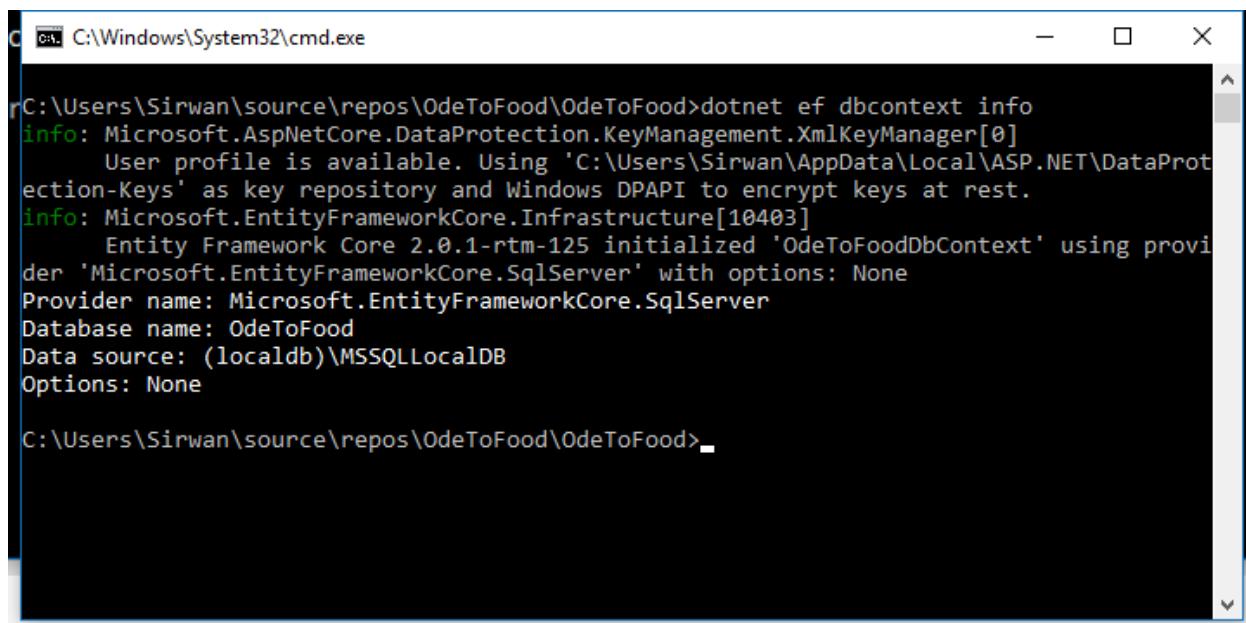
لیست `context` های برنامه:



```
C:\Windows\System32\cmd.exe
C:\Users\Sirwan\source\repos\OdeToFood\OdeToFood>dotnet ef dbcontext list
info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[0]
      User profile is available. Using 'C:\Users\Sirwan\AppData\Local\ASP.NET\DataProtection-Keys' as key repository and Windows DPAPI to encrypt keys at rest.
OdeToFood.Data.OdeToFoodDbContext

C:\Users\Sirwan\source\repos\OdeToFood\OdeToFood>
```

اطلاعاتی در مورد `context` های پروژه جاری:



```
C:\Windows\System32\cmd.exe
C:\Users\Sirwan\source\repos\OdeToFood\OdeToFood>dotnet ef dbcontext info
info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[0]
      User profile is available. Using 'C:\Users\Sirwan\AppData\Local\ASP.NET\DataProtection-Keys' as key repository and Windows DPAPI to encrypt keys at rest.
info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 2.0.1-rtm-125 initialized 'OdeToFoodDbContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: None
Provider name: Microsoft.EntityFrameworkCore.SqlServer
Database name: OdeToFood
Data source: (localdb)\MSSQLLocalDB
Options: None

C:\Users\Sirwan\source\repos\OdeToFood\OdeToFood>
```

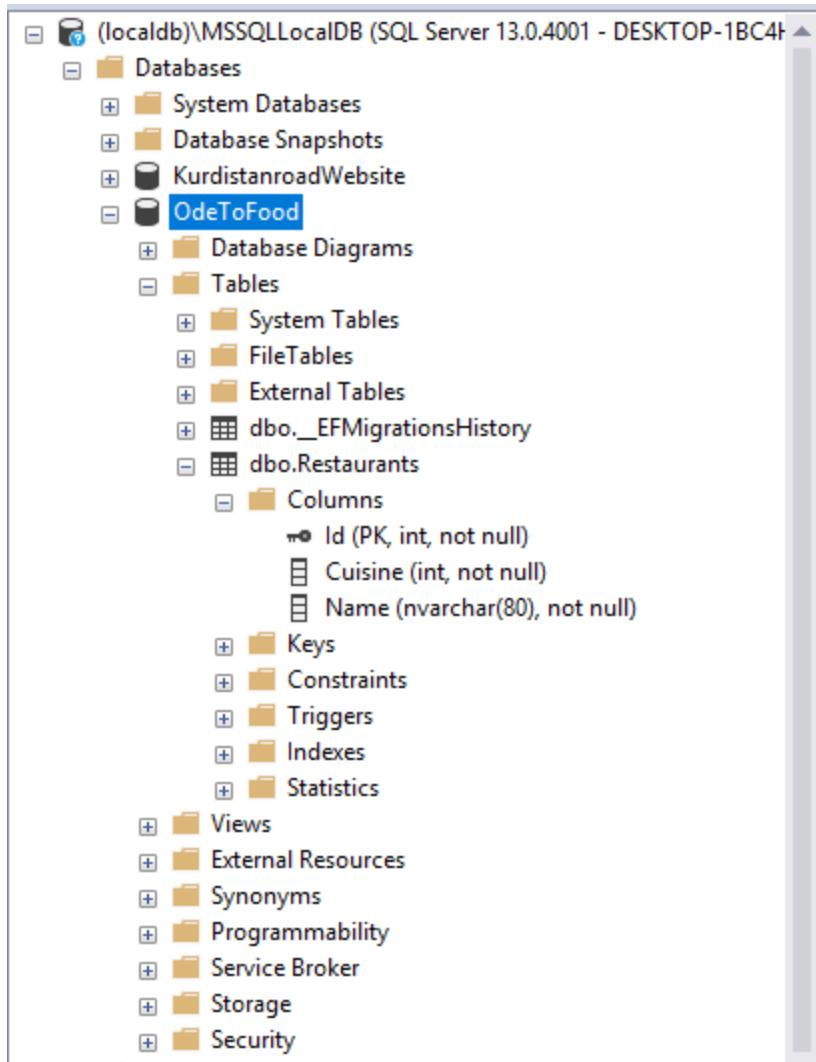
قدم اول: اضافه کردن کدهای مهاجرت:

```
dotnet ef migrations add InitialCreate -v
```

قدم دوم: اعمال تغییرات به دیتابیس:

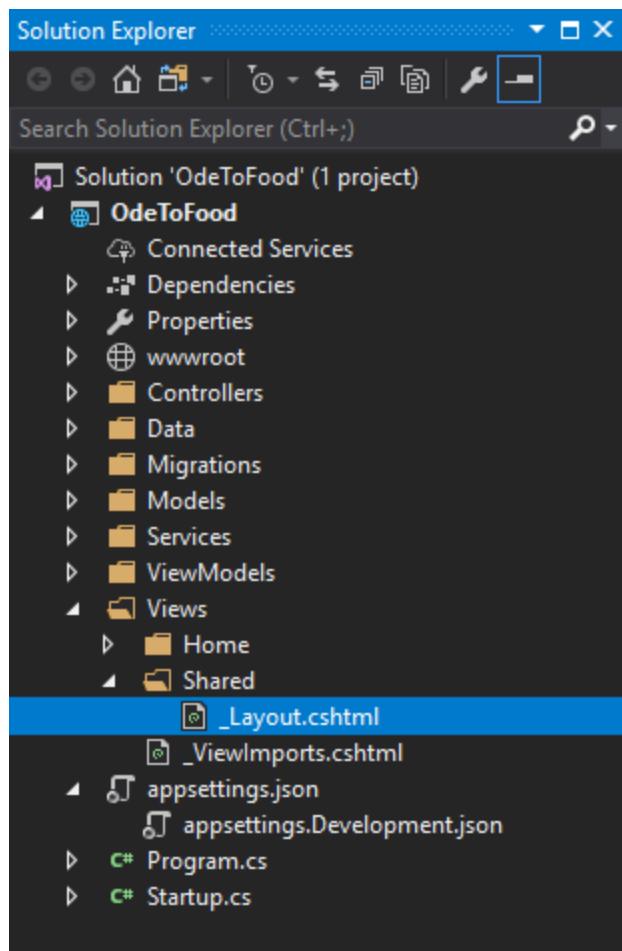
```
dotnet ef database update
```

خروجی:

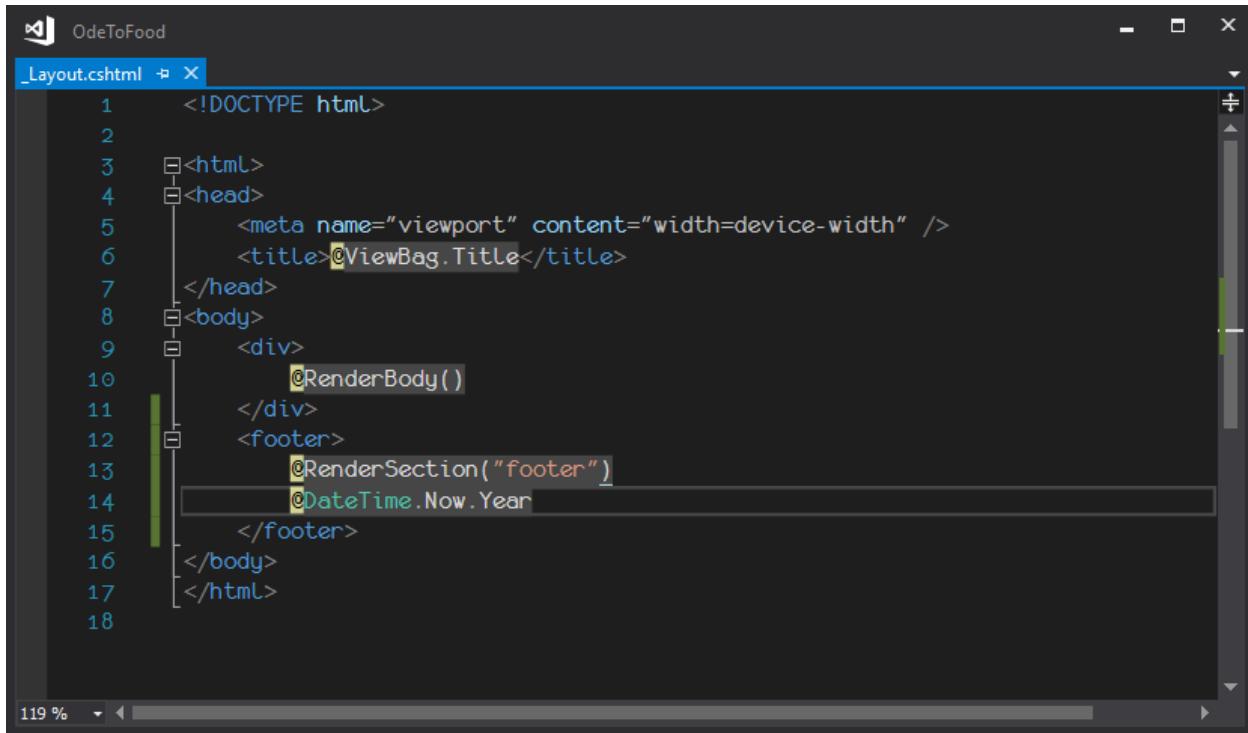


Razor Views Layout Views

لی اوت باید در مسیر زیر ایجاد شود:



محتويات لی اوت:



```
1  <!DOCTYPE html>
2
3  <html>
4  <head>
5      <meta name="viewport" content="width=device-width" />
6      <title>@ViewBag.Title</title>
7  </head>
8  <body>
9      <div>
10         @RenderBody()
11     </div>
12     <footer>
13         @RenderSection("footer")
14         @DateTime.Now.Year
15     </footer>
16 </body>
17 </html>
18
```

برای ایجاد **section** سفارشی نیز می‌توانیم از هلپر **RenderSection** استفاده کنیم.
اکنون درون ویوها دیگر نیاز به نوشتتن کامل ساختار **HTML** نیست. در قدم بعدی باید دو کار انجام دهیم:

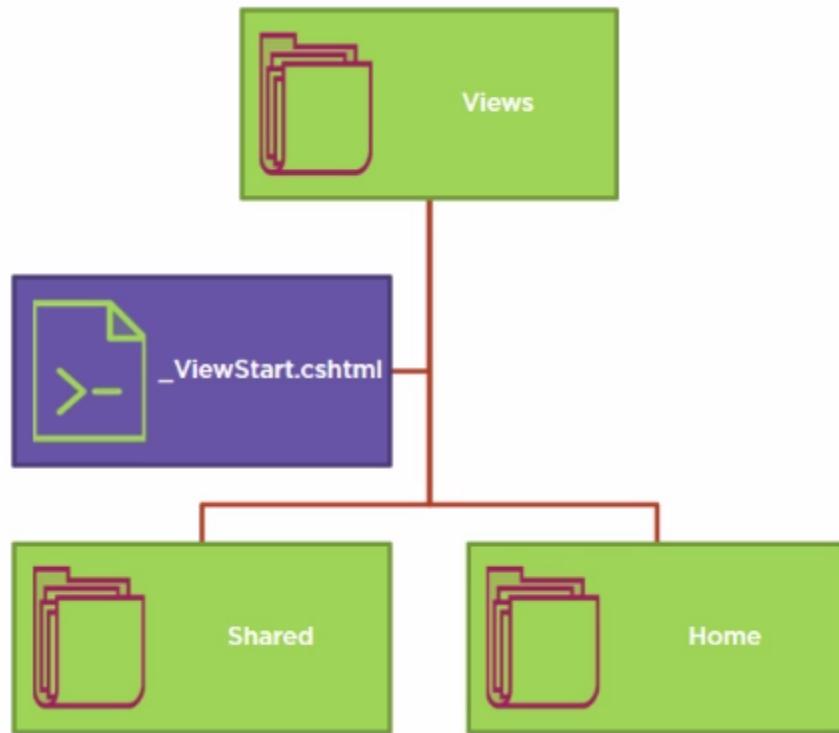
- استفاده از لی اوت فوق برای ویوها

- ست کردن **ViewBag** مربوط به عنوان صفحه

```
- @{
-     Layout = "_Layout";
-     ViewBag.Title = "Home Page";
- }
```

ViewStart

فریمورک **MVC** قبل از اجرای کدهای ویوها ابتدا محتویات ویوی‌ایی با نام **_ViewStart.cshtml** را اجرا خواهد کرد:



محتويات و محل قرار دادن فایل _ViewStart.cshtml

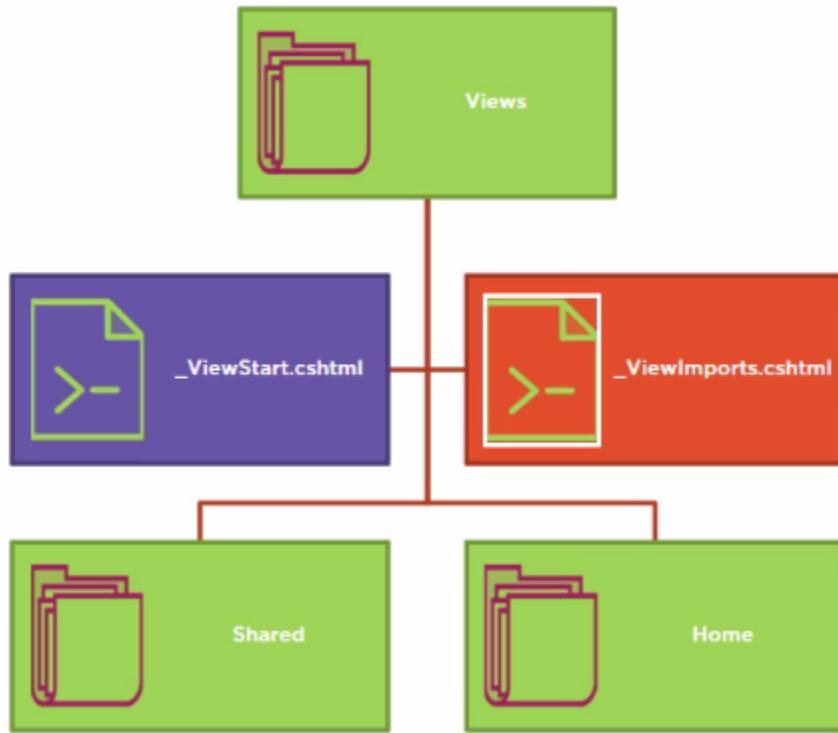
```
@{
    Layout = "_Layout";
}
```

اکنون دیگر درون ویوها نیازی به تعیین لی اوت نیست.

نکته: اگر فایل `_ViewStart.cshtml` را درون پوشه‌ی `Home` قرار دهیم؛ لی اوت تنها برای ویوها در نظر گرفته خواهد شد. اما با قرار دادن این فایل درون پوشه‌ی `Views` لی اوت برای تمامی ویوها تنظیم خواهد شد.

_ViewImports

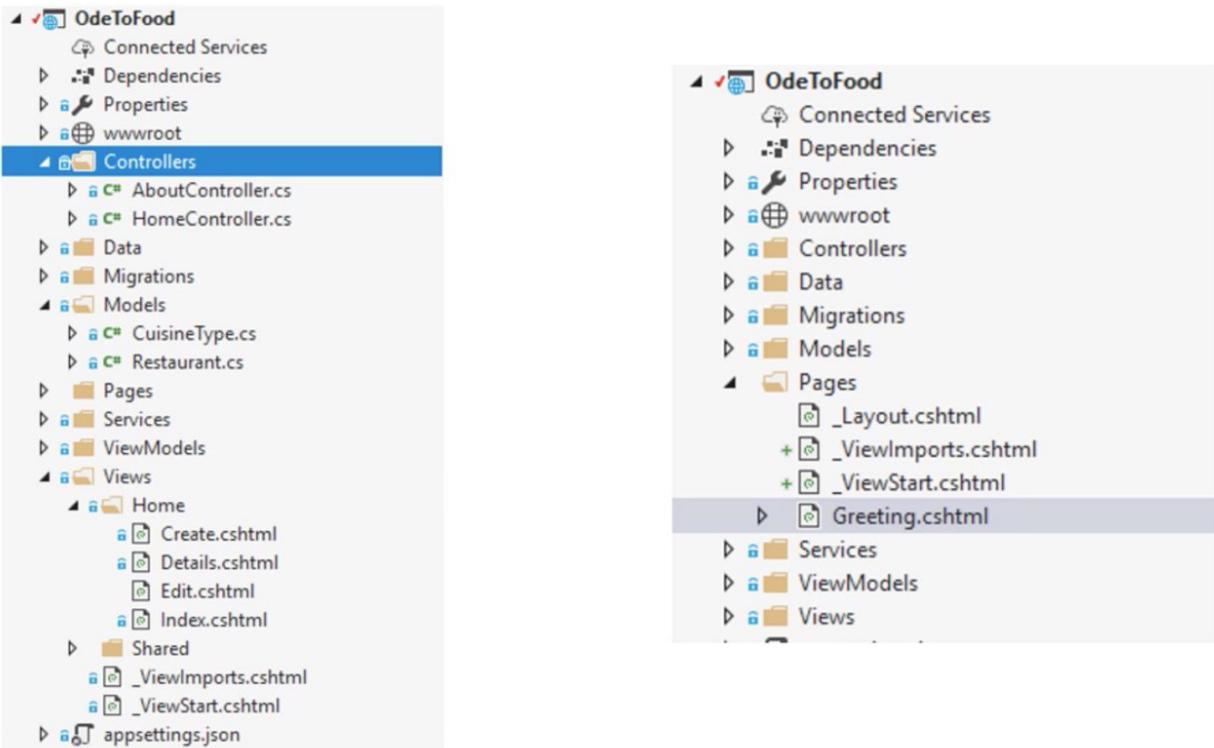
علاوه بر قرار دادن فایل `_ViewImports.cshtml` می‌توانیم از فایل `_ViewStart.cshtml` نیز جهت افزودن `namespace`‌های موردنیاز برای ویوها نیز استفاده کنیم:



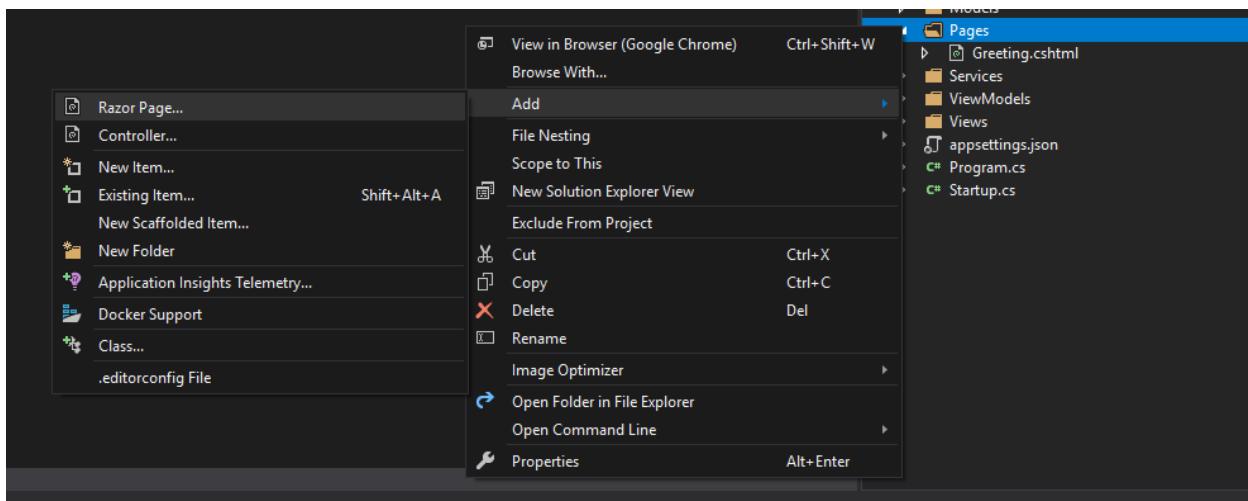
در ورژن‌های قبلی MVC از یک فایل XML برای اینکار استفاده می‌شد.

Razor Pages

Razor Pages یک قابلیت جدید است که به نسخه ASP.NET Core 2.0 اضافه شده است. اگر هدفتان ایجاد اپلیکیشنی است که عموماً کارش تولید HTML است؛ می‌توانید از این قابلیت استفاده کنید. این قابلیت یک روش جایگزین برای الگوی MVC HTML جهت تولید HTML است. در MVC این کنترلر است که با دریافت درخواست یک مدل ایجاد کرده و آن را به ویو جهت رender شدن ارائه می‌دهد. اما دریافت درخواست یک صفحه Razor Pages که درون پوشه Pages واقع در ریشه پروژه، هدایت خواهد شد؛ این صفحه نیز می‌تواند مدل موردنظر را تولید و خروجی مناسب برای کاربر را تولید کند. بنابراین اگر می‌خواهید وب‌اپلیکیشن‌های مبتنی بر سرویس‌های مبتنی بر HTTP یا REST ایجاد کنید از الگوی MVC استفاده کنید، اما هدف از Razor Pages ایجاد صفحات HTML است.



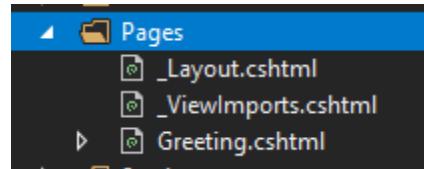
قدم اول ایجاد یک :Razor Page



محفویات فایل :Greeting.cshtml

```
@page
@{
}
```

اکنون هر درخواستی به `/greeting` باید، به صورت خودکار و یو فوق به عنوان خروجی نمایش داده می‌شود. در **Razor Pages** واقع دایرکتیو `@page` است که اینکار را انجام خواهد داد. همچنین لازم به ذکر است که نیز می‌توانند دارای فایل‌های لی‌اوتن و `_ViewImports` باشند:



همچنین می‌شود درون **Razor Pages** از سرویس‌هایمان نیز استفاده کرد:

```
@page
@inject IGreeter Greeter
<div>
    @Greeter.GetMessageOfDay()
</div>
```

برای **Razor Pages** امکان تعریف مدل را نیز خواهیم داشت؛ تعریف مدل را می‌توانیم درون کد `code-` مربوط به **Razor Page** behind انجام دهیم:

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace OdeToFood.Pages
{
    public class GreetingModel : PageModel
    {
        public void OnGet()
        {

        }
    }
}
```

بنابراین کلاس `GreetingModel` به کلاسی و هلمسازی شده‌ایی تبدیل خواهد شد که درون **Razor Page** مان در دسترس خواهد بود.

:مثال

```
using Microsoft.AspNetCore.Mvc.RazorPages;
using OdeToFood.Services;

namespace OdeToFood.Pages
{
    public class GreetingModel : PageModel
    {
        private IGreeter _greeter;

        public string CurrentGreeting { get; set; }

        public GreetingModel(IGreeter greeter)
        {
            _greeter = greeter;
        }

        public void OnGet()
        {
            CurrentGreeting = _greeter.GetMessageOfDay();
        }
    }
}
```

نکته: کار متد **OnGet** چیست؟

وقتی **Razor Page** مان یک درخواست HTTP GET را دریافت کند، فریمورک **MVC** مدل آن را (در اینجا **GreetingModel**) را وله‌سازی کرده و تمامی سرویس‌های موردنیاز را تزریق خواهد کرد، سپس متد **OnGet** را فراخوانی خواهد کرد. بنابراین درون این متد می‌توانید تمامی اطلاعات موردنیاز مدل را **populate** کنیم.

استفاده از مدل فوق درون **Razor Page**:

```
@page
@model GreetingModel

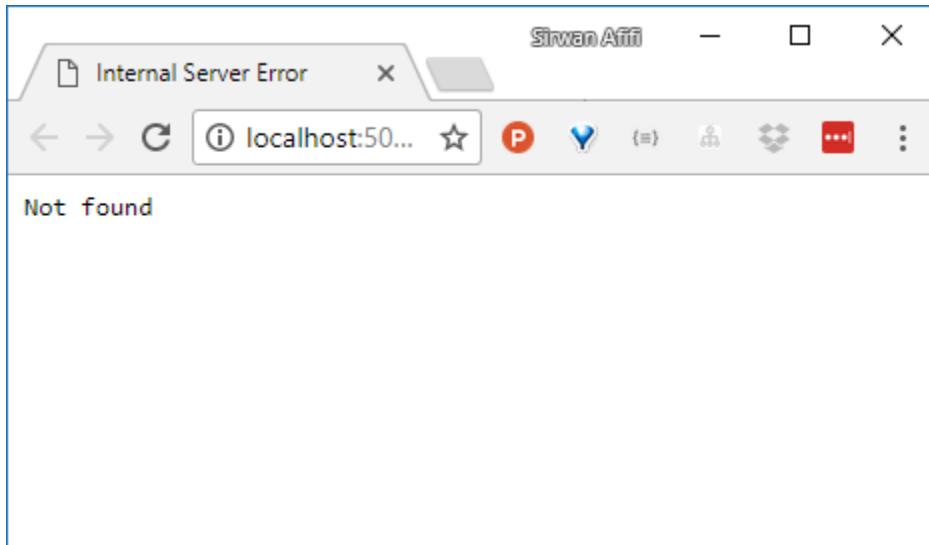
<div>@Model.CurrentGreeting</div>
```

توسط **Razor Pages** می‌توانیم **GET** را نیز برای درخواست‌های **named parameter** تعیین کنیم:

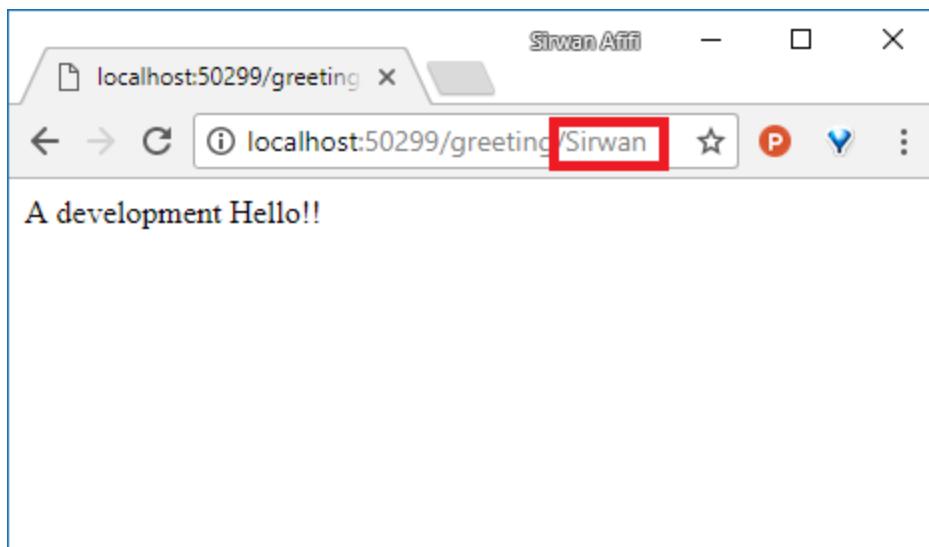
```
@page "{name}"
@model GreetingModel

<div>@Model.CurrentGreeting</div>
```

بنابراین اکنون با مراجعه به `page` فوق خواهیم داشت:



در واقع اکنون گفته‌ایم که این `page` باید دارای یک پارامتر `name` باشد:



اکنون می‌توانیم از این پارامتر درون متده استفاده کنیم:

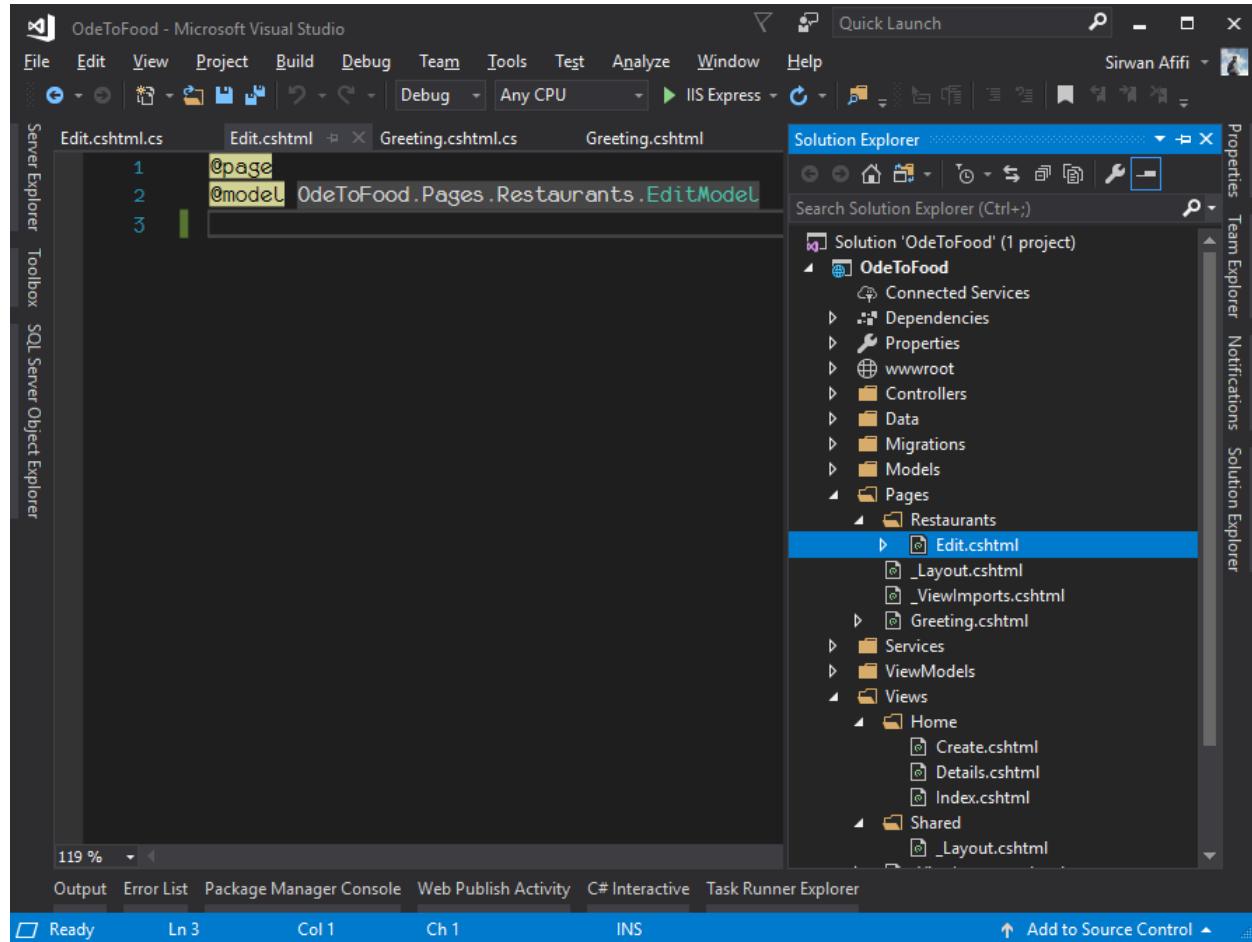
```
public void OnGet(string name)
{
    CurrentGreeting = $"{name} {_greeter.GetMessageOfDay()}";
```

```
}
```

An Edit Form

اضافه کردن قابلیت ویرایش رستوران توسط Razor Pages

قدم اول: افزودن پوشه Restaurants درون پوشه Pages



قدم دوم: تعریف پارامتر id برای page ایجاد شده:

```
@page "{id}"
@model OdeToFood.Pages.Restaurants.EditModel
```

قدم سوم: افزودن لینک Edit

```
<a asp-page="/Restaurants/Edit" asp-route-id="@restaurant.Id">Edit</a>
```

قدم چهارم: درون متدهای `OnGet` باید آیتم موردنظر را از دیتابیس استخراج کنیم و درون صفحه نمایش دهیم:

```
using Microsoft.AspNetCore.Mvc.RazorPages;
using OdeToFood.Models;
using OdeToFood.Services;

namespace OdeToFood.Pages.Restaurants
{
    public class EditModel : PageModel
    {
        private IRestaurantData _restaurantData;

        public Restaurant Restaurant { get; set; }

        public EditModel(IRestaurantData restaurantData)
        {
            _restaurantData = restaurantData;
        }

        public void OnGet(int id)
        {
            Restaurant = _restaurantData.Get(id);
        }
    }
}
```

اگر شیء `ActionResult` نال بود چکار باید بکنیم؟ در واقع خروجی متدهای `OnGet` یک `Restaurant` است؛

در نتیجه میتوانیم یک `redirect action` استفاده کنیم:

```
public IActionResult OnGet(int id)
{
    Restaurant = _restaurantData.Get(id);
    if (Restaurant == null)
    {
        return RedirectToAction("Index", "Home");
    }
    return Page();
}
```

قدم پنجم: نمایش آیتم انتخاب شده درون پیج `Edit`

```
@page "{id}"
```

```

@model OdeToFood.Pages.Restaurants.EditModel

<h1>@Model.Restaurant.Name</h1>

<form method="post">
    <input type="hidden" asp-for="Restaurant.Id"/>
    <div>
        <label asp-for="Restaurant.Name"></label>
        <input asp-for="Restaurant.Name" />
        <span asp-validation-for="Restaurant.Name"></span>
    </div>
    <div>
        <label asp-for="Restaurant.Cuisine"></label>
        <select asp-for="Restaurant.Cuisine"
                asp-items="@Html.GetEnumSelectList<Cuisine>()"></select>
        <span asp-validation-for="Restaurant.Cuisine"></span>
    </div>
    <input type="submit" name="save" value="Save" />
</form>

```

همانطور که مشاهده می‌کنید برای سورس استفاده کرده‌ایم؛ در واقع **HTML Helpers** از **asp-items** استفاده کردیم؛ در بیشتر اوقات از **HTML Helpers** به همراه **Tag Helper** استفاده می‌کنیم.

قدم ششم: نسخهی Edit اکشن متدهی Post

```

public IActionResult OnPost()
{
    if (ModelState.IsValid)
    {
        _restaurantData.Update(Restaurant);
        return RedirectToAction("Details", "Home", new { id = Restaurant.Id });
    }
    return Page();
}

```

نکته: همانطور که مشاهده می‌کنید برخلاف حالت اکشن متدهی **MVC**، در اینجا شیء موردنظر را به متدهی **OnPost** ارسال نکرده‌ایم، زیرا این شیء را در حال حاضر در اختیار داریم (پراپرتی پابلیک **Restaurant**) اما در اینحالت باید اتربیوت **BindProperty** را به این پراپرتی اضافه کنیم؛ با کمک این اتربیوت گفته‌ایم که اطلاعات وارد شده از طریق درخواست **POST** را دریافت کرده و آن را به پراپرتی موردنظر بایند کن:

```
[BindProperty]
```

```
public Restaurant Restaurant { get; set; }
```

همانطور که مشاهده می‌کنید برای ثبت تغییرات یک متده با نام **Update** را به اینترفیس **IRestaurantData** اضافه کرده‌ایم:

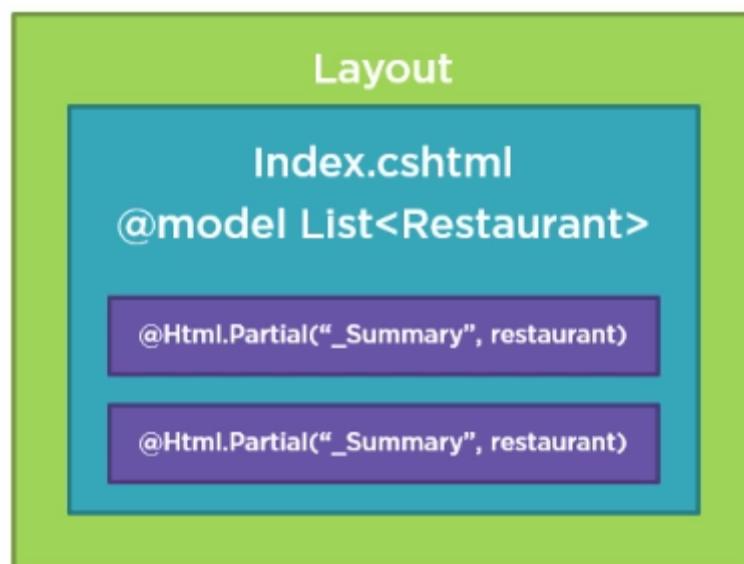
```
Restaurant Update(Restaurant restaurant);
```

پیاده‌سازی متده فوق:

```
public Restaurant Update(Restaurant restaurant)
{
    _context.Attach(restaurant).State =
        Microsoft.EntityFrameworkCore.EntityState.Modified;
    _context.SaveChanges();
    return restaurant;
}
```

Partial Views

دو حالت برای استفاده از **Partial Views** وجود دارد؛ فرض کنید یک **Layout View** دارید؛ این ویو توسط **ویوی Index** جهت رender شدن انتخاب خواهد شد؛ ویوی **Index** نیز یک مدل را از کنترلر دریافت خواهد کرد؛ مدل نیز به **Partial View** جهت رender شدن ارسال خواهد شد:



وقتی از `Html.Partial` جهت رندر کردن یک `partial view` استفاده می‌کنید، این `partial view` برای `parent View` نمی‌تواند به صورت مستقل اطلاعات را دریافت کند.

اگر نیاز داشته باشید مدلی دیگری را جهت رندر شدن به `partial view` ارسال کنید می‌توانید از `View Components` استفاده کنید:



از `@Component.InvokeAsync()` در هر جایی می‌توانیم استفاده کنیم. بخلاف `Partial View`، ویو کامپوننت‌ها متکی بر `parent view` نیستند.

ایجاد یک `Partial View` – درون صفحه `Index` می‌خواهیم لیست رستوران‌ها توسط `Partial Views` رندر کنیم:

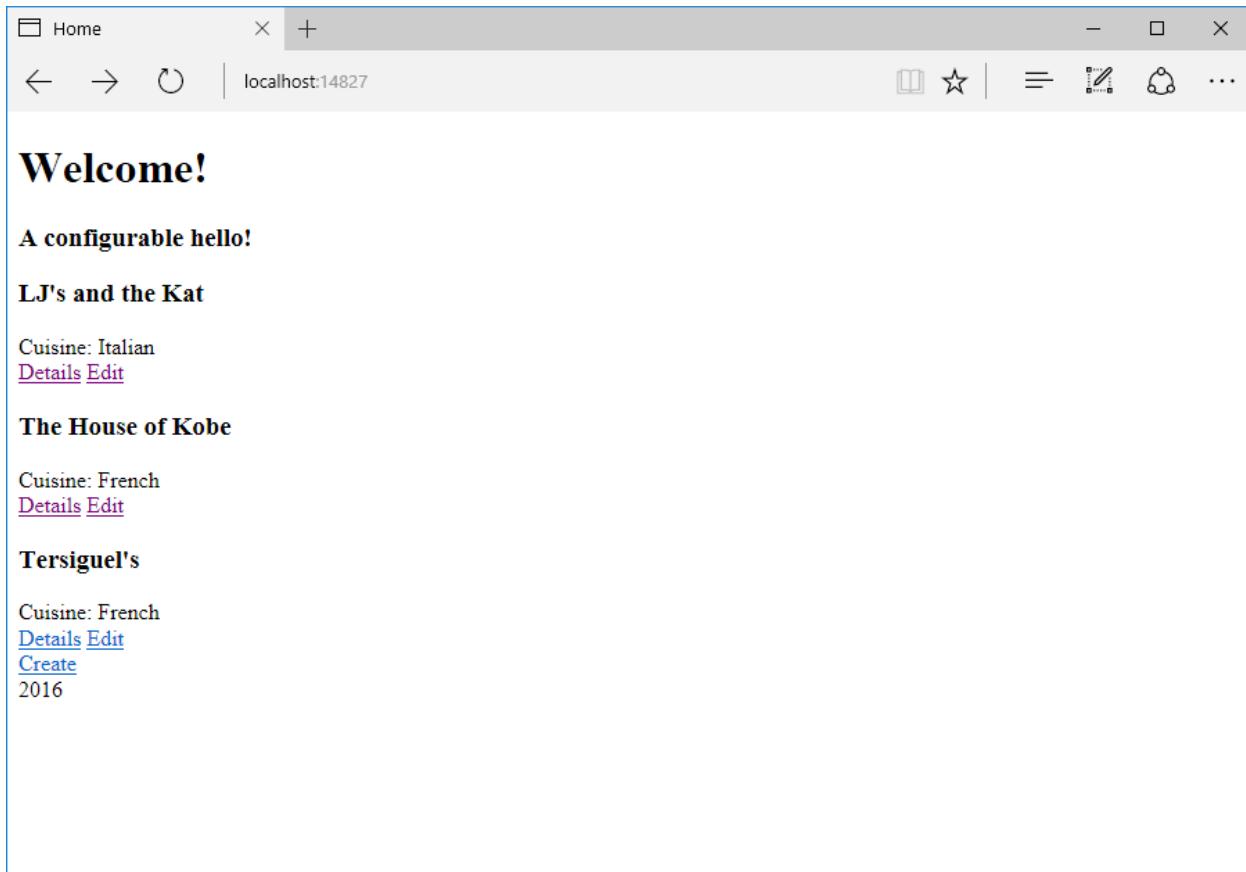
```
@using OdeToFood.Models  
@model Restaurant  
  
<section>  
    <h3>@Model.Name</h3>  
    <div>  
        Cuisine: @Model.Cuisine  
    </div>  
    <div>  
        <a asp-action="Details" asp-route-id="@Model.Id">Details</a>  
        <a asp-page="/Restaurants/Edit" asp-route-id="@Model.Id">Edit</a>  
    </div>
```

```
</div>  
</section>
```

نحوه استفاده:

```
@foreach (var restaurant in Model.Restaurants)  
{  
    @Html.Partial("_Summary", restaurant)  
}
```

خروجی:



نکته: همانطور که عنوان شد، **Partial Views** همیشه به یک مدل نیاز خواهند داشت، اما **View Components** نیازی به مدل ندارند.

View Components

ویو کامپوننت به صورت کاملاً مستقل از کنترلر و ویو عمل می‌کند. به عنوان مثال درون `_Layout` می‌توانیم اینگونه یک ویو کامپوننت را فراخوانی کنیم:

```
<footer>
    @await Component.InvokeAsync("Greeter")
</footer>
```

نکته: می‌توانیم همچنین از سینتکس زیر برای نمایش ویوکامپوننت استفاده کنیم:

```
<vc:greeter></vc:greeter>
```

برای اینکه حالت فوق کار کند باید یک تگ‌هالپر دیگر را به فایل `_ViewImports` اضافه کنیم:

```
@addTagHelper *, OdeToFood
```

اینکار باعث خواهد شد که تمامی المنشاهی‌های سفارشی پروژه‌ی `OdeToFood` برای تمامی ویوها در دسترس باشند.

نکته: قبل از `@Html.Action` برای نمایش خروجی یک اکشن متده استفاده می‌کردیم؛ اما در این نسخه از فریم‌ورک این متده وجود ندارد و بهتر است از **View Components** به جای آن استفاده شود.

:Greeting تعریف ویو کامپوننت

```
using Microsoft.AspNetCore.Mvc;
using OdeToFood.Services;

namespace OdeToFood.ViewComponents
{
    public class GreeterViewComponent : ViewComponent
    {
        private IGreeter _greeter;

        public GreeterViewComponent(IGreeter greeter)
        {
            _greeter = greeter;
        }
    }
}
```

```

    }

    public IViewComponentResult Invoke()
    {
        var model = _greeter.GetMessageOfDay();
        return View("Default", model);
    }
}

```

نکته: اگر ورودی تابع **View** یک رشته باشد، **MVC** فرض خواهد کرد که ما نام **View** را ارسال کدهایم؛ بنابراین برای حل این مشکل باید بگوئیم که منظورمان از رشته مدل‌مان است؛ بنابراین پارامتر اول را نام ویو قرار خواهیم داد و پارامتر دوم را برابر با مدل قرار خواهیم داد.

طبق convention محل ویو فوق باید در مسیر **/Shared/Components/Greeting/Default.cshtml** است.

محتویات ویو کامپوننت **Greeting** :

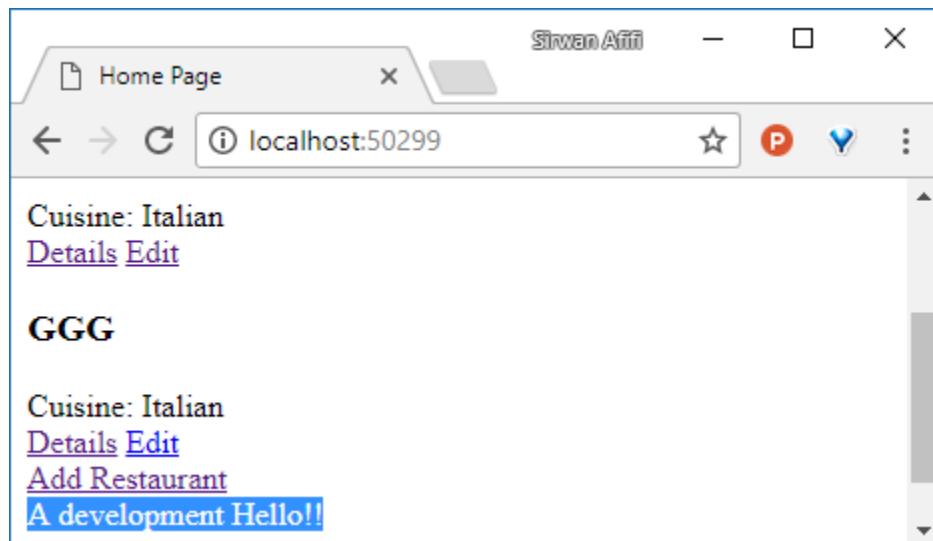
```

@model string

<div>@Model</div>

```

خروجی:



ASP.NET Identity

Authentication Services and Middlewares

قدم اول: نصب Middleware – این میان افزار را بعد از StaticFiles و قبل از Mvc اضافه کنید:

```
app.UseStaticFiles();  
  
app.UseAuthentication();  
  
app.UseMvc(ConfigureRoute);
```

قدم دوم: افزودن سرویس های Identity

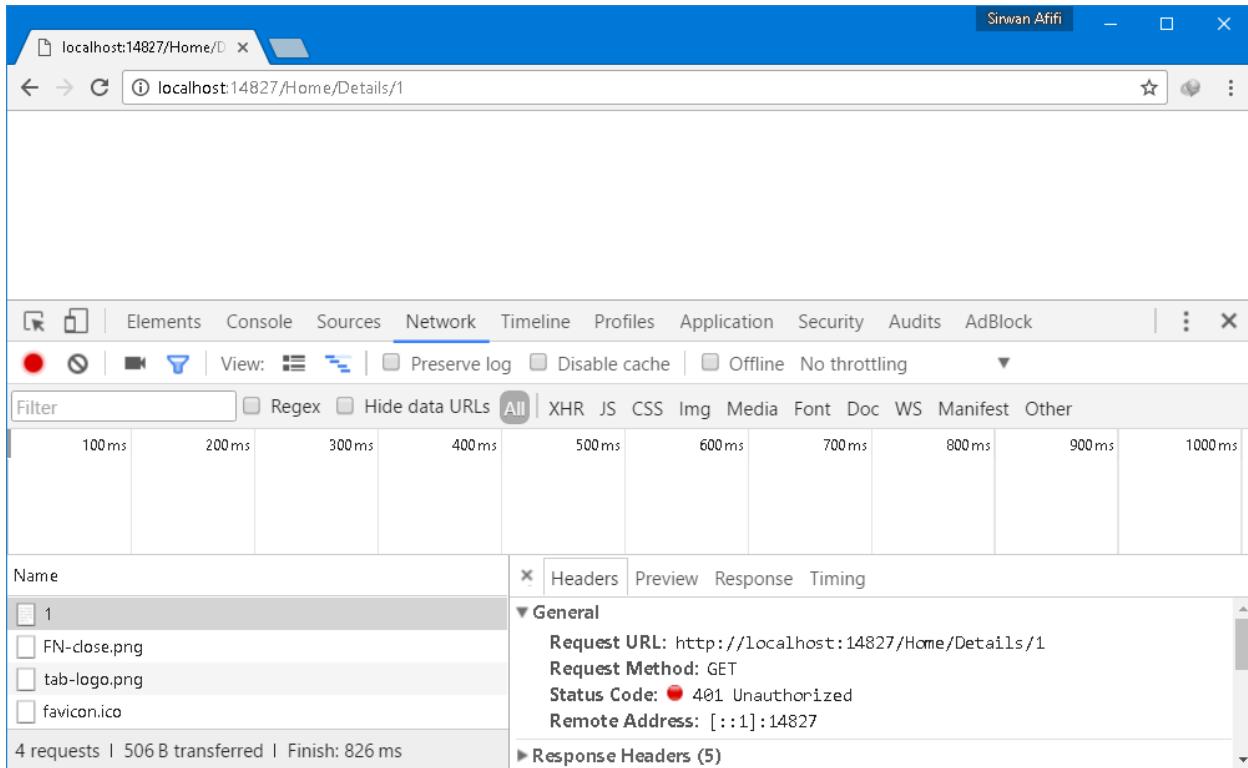
```
services.AddAuthentication(options =>  
{  
    options.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;  
})  
.AddCookie();
```

نکته: توسط **DefaultScheme** نحوه authenticate کردن کاربران را تعیین کرده ایم؛ این نوع می تواند کوکی، توکن و ... باشد. در اینجا ما از کوکی استفاده کرده ایم.

Using the Authorize Attribute

```
[Authorize]  
public class ProtectedController : Controller  
{  
    // ....  
}  
  
[Authorize]  
public class ProtectedController : Controller  
{  
    [AllowAnonymous]  
    public IActionResult UnprotectedAction()  
    {  
        return Content("Hi!");  
    }  
  
    // ... All other actions require sign-in  
}
```

در این مرحله اگر به اکشن متدهی که با ویژگی `Authorize` مزین شده باشد مراجعه کنیم، خروجی زیر را دریافت خواهیم کرد:

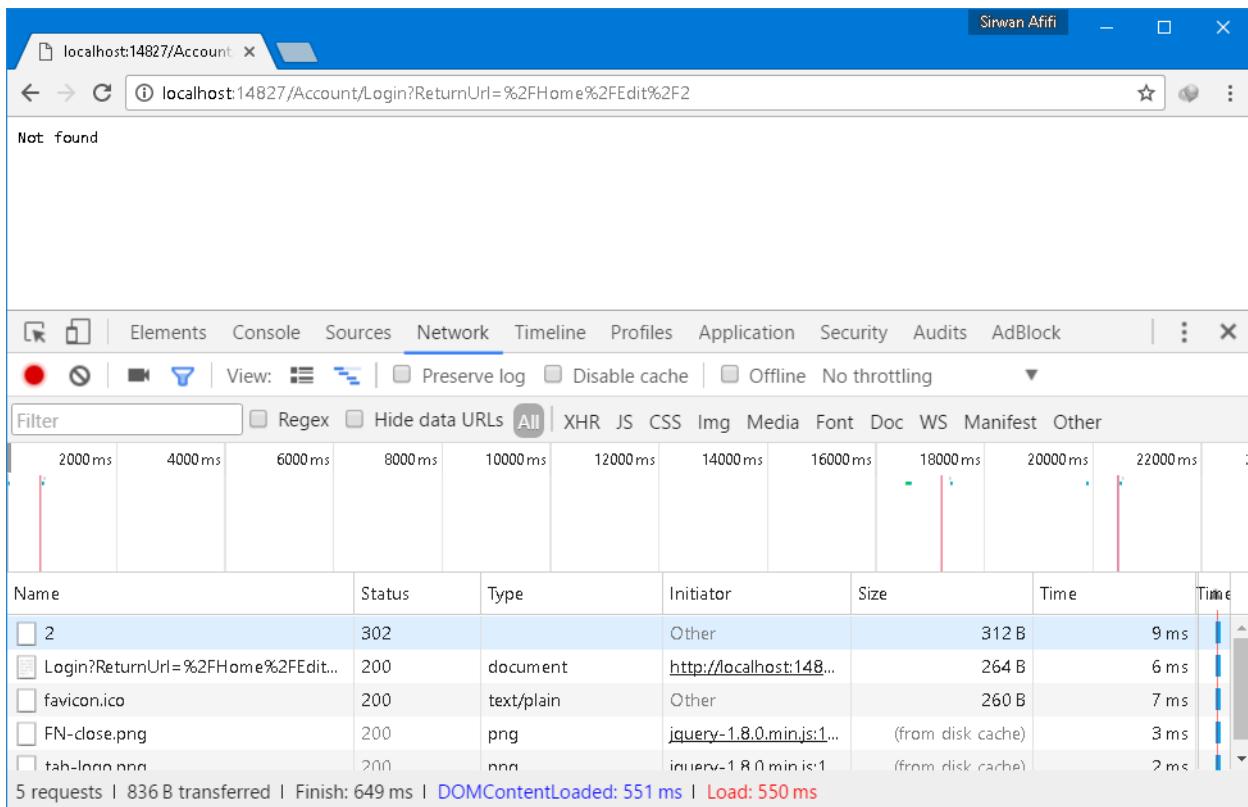


در اینحالت است که **Identity Framework** به ما کمک خواهد کرد؛ این فریمورک خروجی 401 را در اپلیکیشن شناسایی کرده و آن را تبدیل به یک صفحه لگین خواهد کرد؛ سپس کاربر می‌تواند از طریق این صفحه لگین از مانع فوق (خطای 401) عبور کند.

برای اتریبیوت `Authorize` می‌توانیم پراپرتبی ای با نام `Policy` را نیز ست کنیم، توسط این پراپرتبی می‌توانیم یک `named policy` تعریف کنیم، فرض کنید یک پالیسی با نام `IsAdmin` تعریف کرده‌ایم، این پالسی می‌تواند حاوی تمامی چک‌های موردنیاز برای کاربر جاری باشد. نه تنها می‌توان بررسی کرد که کاربر `authenticate` شده است بلکه می‌توانیم اطلاعاتی دیگری را نیز چک کنیم.

User Registration

اکنون که اسکیمای دیتابیس‌مان آپدیت شده است، با مراجعه به صفحه‌ی `Authorize` شده، خروجی `not found` را مشاهده خواهیم کرد؛ همچنین آدرس صفحه نیز تغییر خواهد کرد:



همانطور که مشاهده می‌کنید اینبار به جای 404 status code 302 status code را دریافت کرده‌ایم؛ یعنی صفحه ردآیروکت شده است. همچنین آدرس پیش‌فرض برای لگین نیز `/Account/Login` است که می‌توانیم آن را تغییر دهیم. از آنجائیکه این کنترلر و اکشن متدهای نداریم، این درخواست توسط MVC هندل نخواهد شد؛ در نتیجه به آخرین middleware ثبت شده درون pipeline که خروجی not found را نمایش می‌دهد رسیده‌ایم.

ایجاد کنترلر Account

```
using Microsoft.AspNetCore.Mvc;

namespace OdeToFood.Controllers
{
    public class AccountController : Controller
    {
        [HttpGet]
        public IActionResult Register()
        {
            return View();
        }
    }
}
```

ایجاد ویو مدل ثبت نام کاربران

```
using System.ComponentModel.DataAnnotations;

namespace OdeToFood.ViewModels
{
    public class RegisterUserViewModel
    {
        [Required, MaxLength(256)]
        public string Username { get; set; }

        [Required, DataType(DataType.Password)]
        public string Password { get; set; }

        [DataType(DataType.Password), Compare(nameof(Password))]
        public string ConfirmPassword { get; set; }
    }
}
```

ایجاد ویو ثبت نام

```

@model RegisterUserViewModel

 @{
    ViewBag.Title = "Register";
 }

<h1>Register</h1>

<form method="post" asp-antiforgery="true">

    <div asp-validation-summary="ModelOnly"></div>

    <div>
        <label asp-for="Username"></label>
        <input asp-for="Username" />
        <span asp-validation-for="Username"></span>
    </div>

    <div>
        <label asp-for="Password"></label>
        <input asp-for="Password" />
        <span asp-validation-for="Password"></span>
    </div>

    <div>
        <label asp-for="ConfirmPassword"></label>
        <input asp-for="ConfirmPassword" />
        <span asp-validation-for="ConfirmPassword"></span>
    </div>

    <div>
        <input type="submit" value="Register" />
    </div>

</form>

```

نکته: همانطور که مشاهده می‌کنید در ویو فوق از عبارت `using` برای ایمپورت کردن `namespace` حاوی ویو مدل استفاده نکرده‌ایم؛ زیرا اینکار را یکبار برای تمامی ویوها درون فایل `_ViewImports.cshtml` انجام داده‌ایم:

```

@using OdeToFood.Entities
@using OdeToFood.ViewModels
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

```

نکته: یکی از سه نوع `enum` زیر را می‌پذیرد:



: تمامی خطاهای مدل نمایش داده شوند. -

: خطاهای مربوط به یک پرپرتوی خاص را نمایش نمی‌دهد. -

: هیچ خطایی نمایش داده نشود. -

ایجاد نمونه Post اکشن متده ثبتنام

```
[HttpPost, ValidateAntiForgeryToken]
public IActionResult Register(RegisterUserViewModel model)
{
}
```

در قسمت بعد به نحوه پیاده‌سازی اکشن متده فوق خواهیم پرداخت.

Creating a User

اکشن متده Register

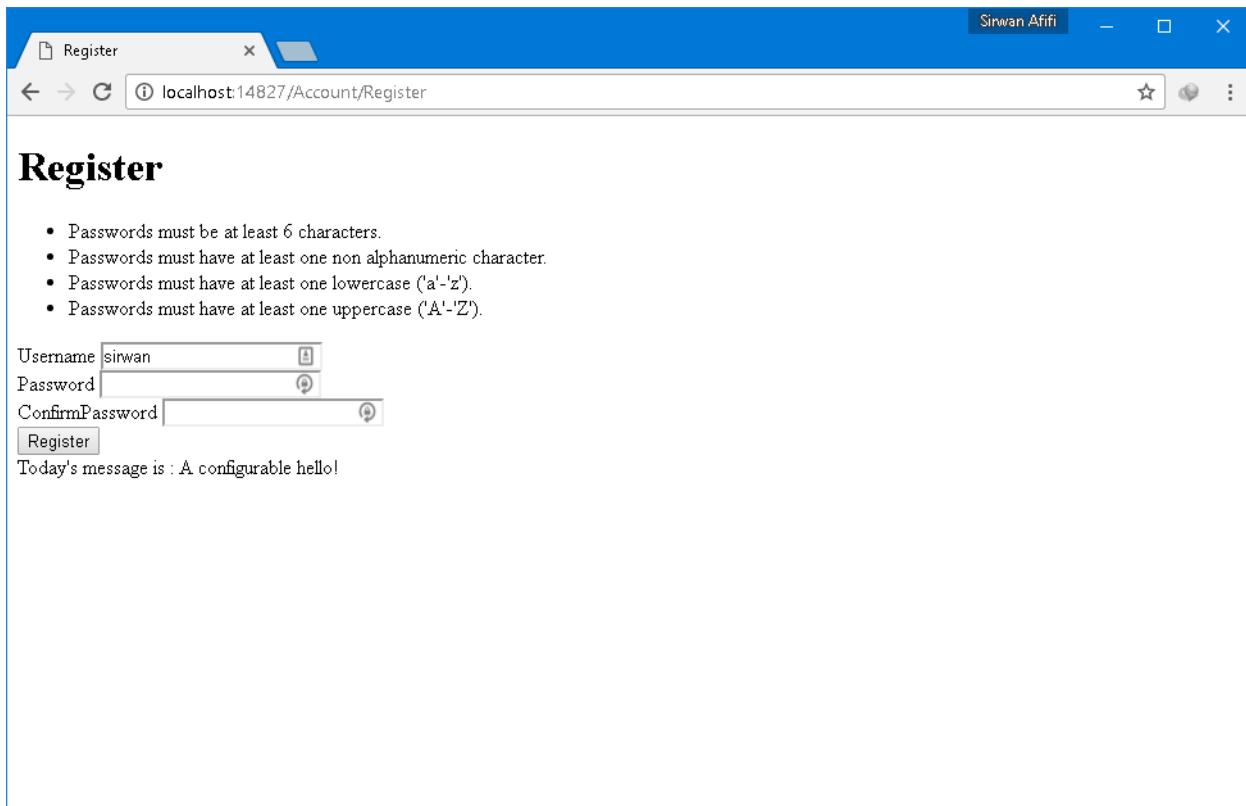
```
[HttpPost, ValidateAntiForgeryToken]
public async Task Register(RegisterUserViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new User { UserName = model.Username };
        var result = await _userManager.CreateAsync(user, model.Password);
        if (result.Succeeded)
        {
            await _signInManager.SignInAsync(user, false);
            return RedirectToAction("Index", "Home");
        }
        else
        {
            foreach (var error in result.Errors)
            {
                ModelState.AddModelError("", error.Description);
            }
        }
    }
    return View();
}
```

نکته: اگر نتیجه‌ی یک کاربر `false` برابر با `Succeed` بود؛ یعنی تمامی خطاهای درون پراپرتی `AddModelError` را به `ModelState` اضافه کرده‌ایم. پراپرتی اول متدهای `ModelState` نام پراپرتی‌ایی است که می‌توانیم خطای خطا را به آن مرتبط (associate) کنیم:

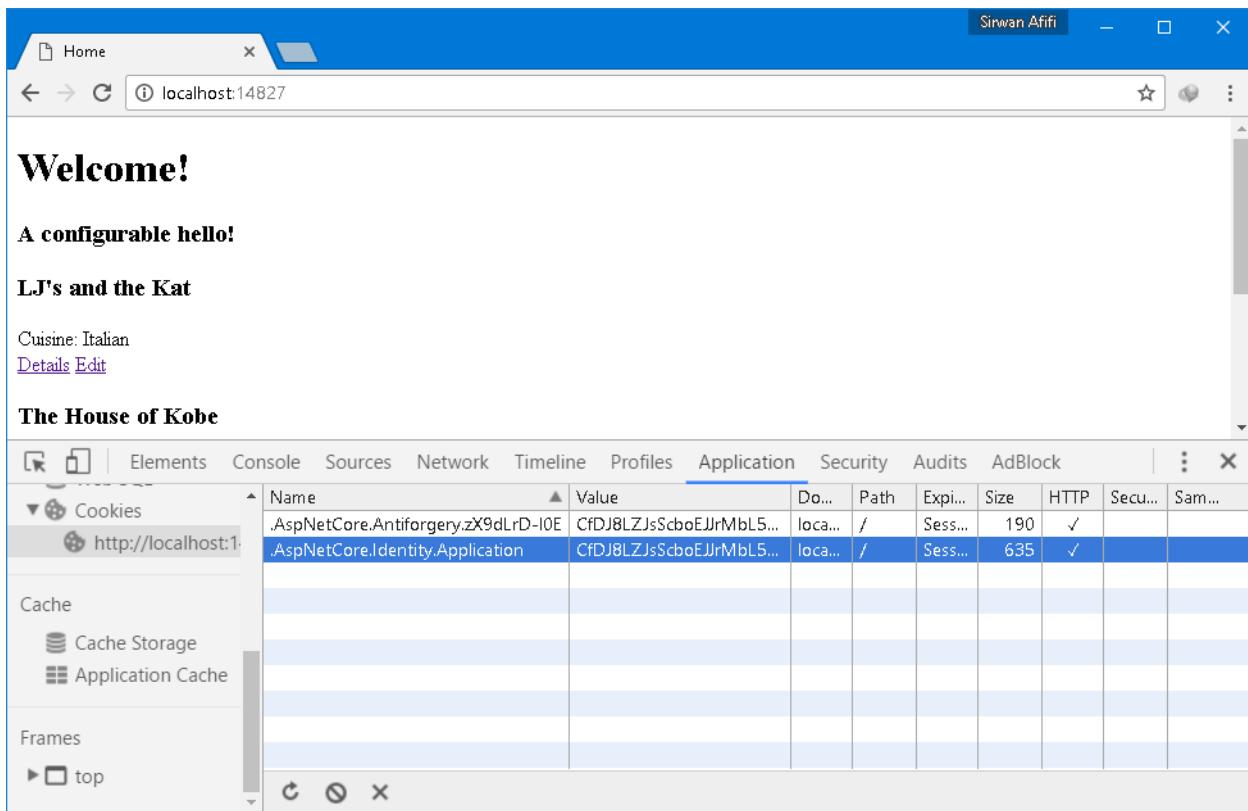
```
ModelState.AddModelError("Username", error.Description);
```

اما چون نمی‌خواهیم اینکار را انجام دهیم؛ آن را خالی رها کرده‌ایم؛ در اینحالت خطای خطا مربوط به مدل است نه یک پراپرتی خاص از مدل:

```
<div asp-validation-summary="ModelOnly"></div>
```



با وارد کردن یک پسورد معتبر کاربر ایجاد شده و کوکی موردنظر نیز ذخیره خواهد شد:



همانطور که مشاهده می‌کنید authentication ticket با موفقیت ایجاد شده است؛ مرورگر نیز این تیکت را به همراه تمامی درخواست‌ها ارسال خواهد کرد؛ بنابراین اینگونه مطمئن خواهد شد که به سایت لاگین کرده‌ایم.

Log in and Log Out

درون فایل `_Layout.cshtml` به کامپوننتی جهت لاگین و لاگ‌اوت نیاز داریم:

```

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
</head>
<body>
<div>
    @await Component.InvokeAsync("LoginLogout")
</div>
<div>
    @RenderBody()
</div>
<footer>
    @RenderSection("footer", false)
    @await Component.InvokeAsync("Greeting")
</footer>
</body>
</html>

```

تعريف کامپوننت LoginLogout

```

using Microsoft.AspNetCore.Mvc;

namespace OdeToFood.ViewComponents
{
    public class LoginLogoutViewComponent : ViewComponent
    {
        public IViewComponentResult Invoke()
        {
            return View();
        }
    }
}

```

ویو کامپوننت فوق

```

@if (User.Identity.IsAuthenticated)
{
    <span>Hello, @User.Identity.Name</span>
    <form method="post" asp-controller="Account" asp-action="Logout" asp-
antiforgery="true">
        <input type="submit" value="Logout" />
    </form>
}
else
{
    <a asp-controller="Account" asp-action="Register">Register</a>
    <a asp-controller="Account" asp-action="Login">Login</a>
}

```

اکشن متد Logout

```
[HttpPost, ValidateAntiForgeryToken]
public async Task<IActionResult> Logout()
{
    await _signInManager.SignOutAsync();
    return RedirectToAction("Index", "Home");
}
```

ایجاد نسخه‌ی GET اکشن متد Login

```
[HttpGet]
public IActionResult Login()
{
    return View();
}
```

ایجاد View Model لایگین

```
using System.ComponentModel.DataAnnotations;

namespace OdeToFood.ViewModels
{
    public class LoginViewModel
    {
        [Required]
        public string Username { get; set; }

        [Required, DataType(DataType.Password)]
        public string Password { get; set; }

        [Display(Name = "Remember Me")]
        public bool RememberMe { get; set; }

        public string ReturnUrl { get; set; }
    }
}
```

ویوی اکشن متد Login

```
@model LoginViewModel
 @{
     ViewBag.Title = "Login";
 }

<h2>Login</h2>

<form method="post" asp-antiforgery="true">
    <div asp-validation-summary="ModelOnly"></div>
    <div>
        <label asp-for="Username"></label>
        <input asp-for="Username" />
        <span asp-validation-for="Username"></span>
    </div>
    <div>
        <label asp-for="Password"></label>
        <input asp-for="Password" />
        <span asp-validation-for="Password"></span>
    </div>
    <div>
        <label asp-for="RememberMe"></label>
        <input asp-for="RememberMe" />
        <span asp-validation-for="RememberMe"></span>
    </div>
    <input type="submit" value="Login" />
</form>
```

ایجاد نسخهی POST آکشن متده لگین

```

[HttpPost, ValidateAntiForgeryToken]
public async Task<IActionResult> Login(LoginViewModel model)
{
    if (ModelState.IsValid)
    {
        var loginResult = await
.signInManager.PasswordSignInAsync(model.Username, model.Password,
model.RememberMe, false);
        if (loginResult.Succeeded)
        {
            if (Url.IsLocalUrl(model.ReturnUrl))
            {
                return Redirect(model.ReturnUrl);
            }
            else
            {
                return RedirectToAction("Index", "Home");
            }
        }
    }
    ModelState.AddModelError("", "Could not login");
    return View(model);
}

```

نکته: دلیل استفاده از متدها `IsLocalUrl` و `Redirect` از آسیب‌پذیری `open redirect` جلوگیری می‌کنند.

Identities and Claims

پر اپراتی `User` علاوه بر کنترلر درون ویوها نیز قابل دسترسی است، فرض کنید می‌خواهیم تمامی `claim`‌های کاربر را نمایش دهیم؛ منظور از `claims` اطلاعات اضافه‌تری از کاربر است:

```

@if (User.Identity.IsAuthenticated)
{
    foreach (var identity in User.Identities)
    {
        <h3>@identity.Name</h3>
        <ul>
            @foreach (var claim in identity.Claims)
            {
                <li>@claim.Type - @claim.Value</li>
            }
        </ul>
    }
}
else
{
    <div>You are anonymous</div>
}

```

}

Front End Frameworks and Tools

Introduction

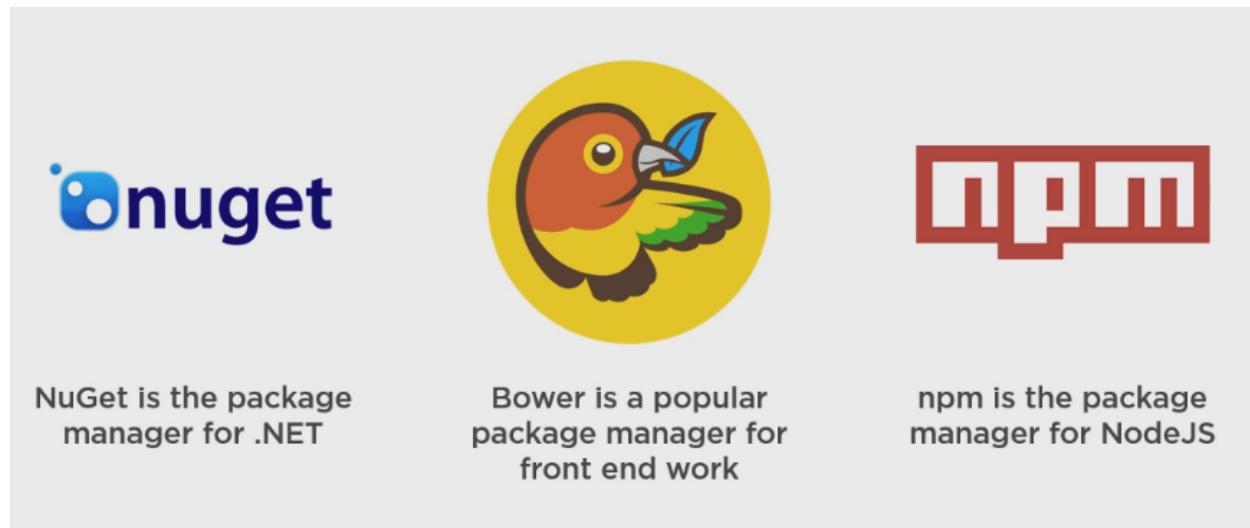
در این مازول به بررسی کتابخانه‌های CSS and JavaScript خواهیم پرداخت؛ در واقع به بررسی ابزارهایی که برای نصب این کتابخانه‌ها استفاده می‌شوند خواهیم پرداخت.

Front End Tools

در سالیان اخیر نیوگت یک پکیج منیجر برای داتنوت بوده است. برای مقطع خاصی نیوگت در ASP.NET Core تنها برای مدیریت پکیج‌های داتنوتی استفاده می‌شد اما تیم وعده داده است که Content Packages را نیز پشتیبانی کند.

برای نصب پکیج‌های فرنتاند bower گزینه‌ی خوبی است اما به دلیلی که در ادامه توضیح خواهیم داد نمی‌خواهیم از آن استفاده کنیم.

Because one tool with one configuration file is easier than using two tools with two configuration files.



Command Line vs. Visual Studio

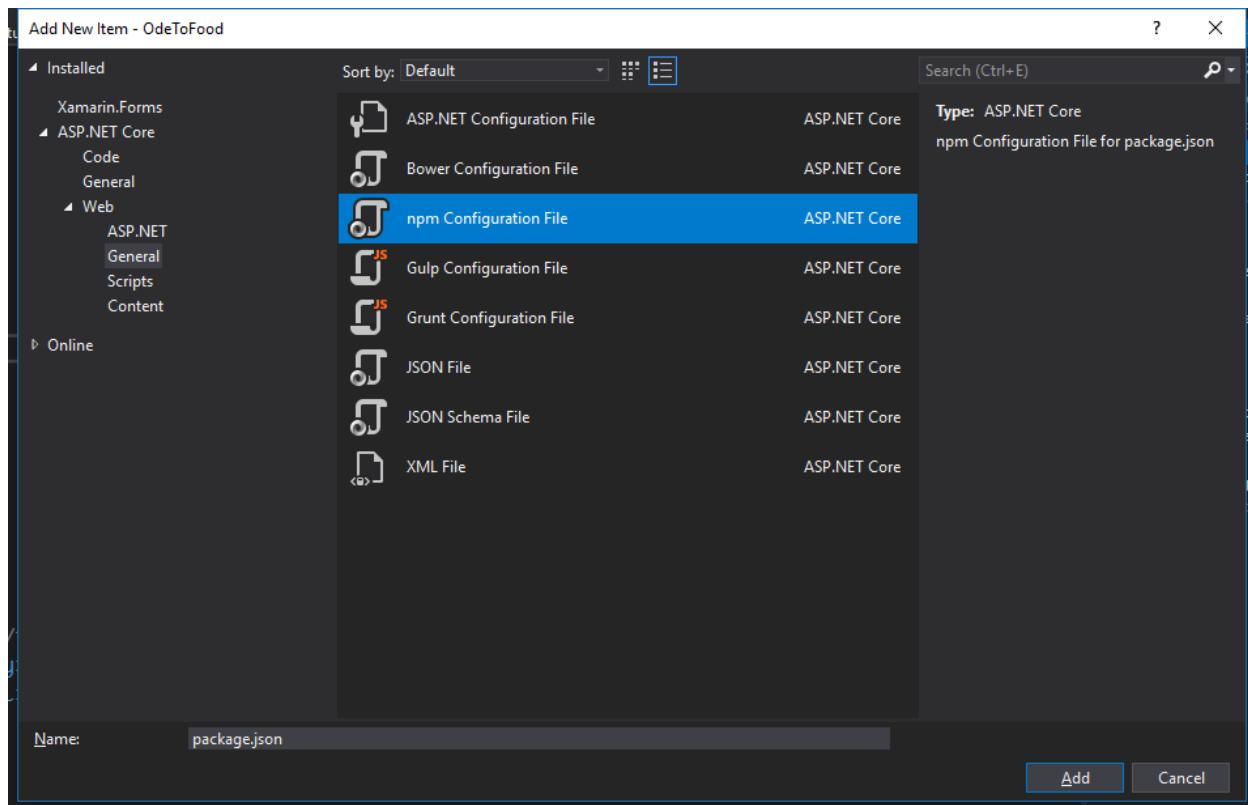
در یک پروژه‌ی ASP.NET دو روش برای کار با NPM وجود دارد.

- Use tooling and support provided by Visual Studio

- Use from the Command Line

Setting up npm

درن ویندوز استودیو کار با NPM به سادگی کار با project.json است؛ کار مدیریت وابستگی‌های سمت سرور را برایمان انجام می‌دهد. فایل package.json نیز کار مدیریت پکیج‌های سمت کلاینت توسط NPM را برایمان انجام خواهد داد:



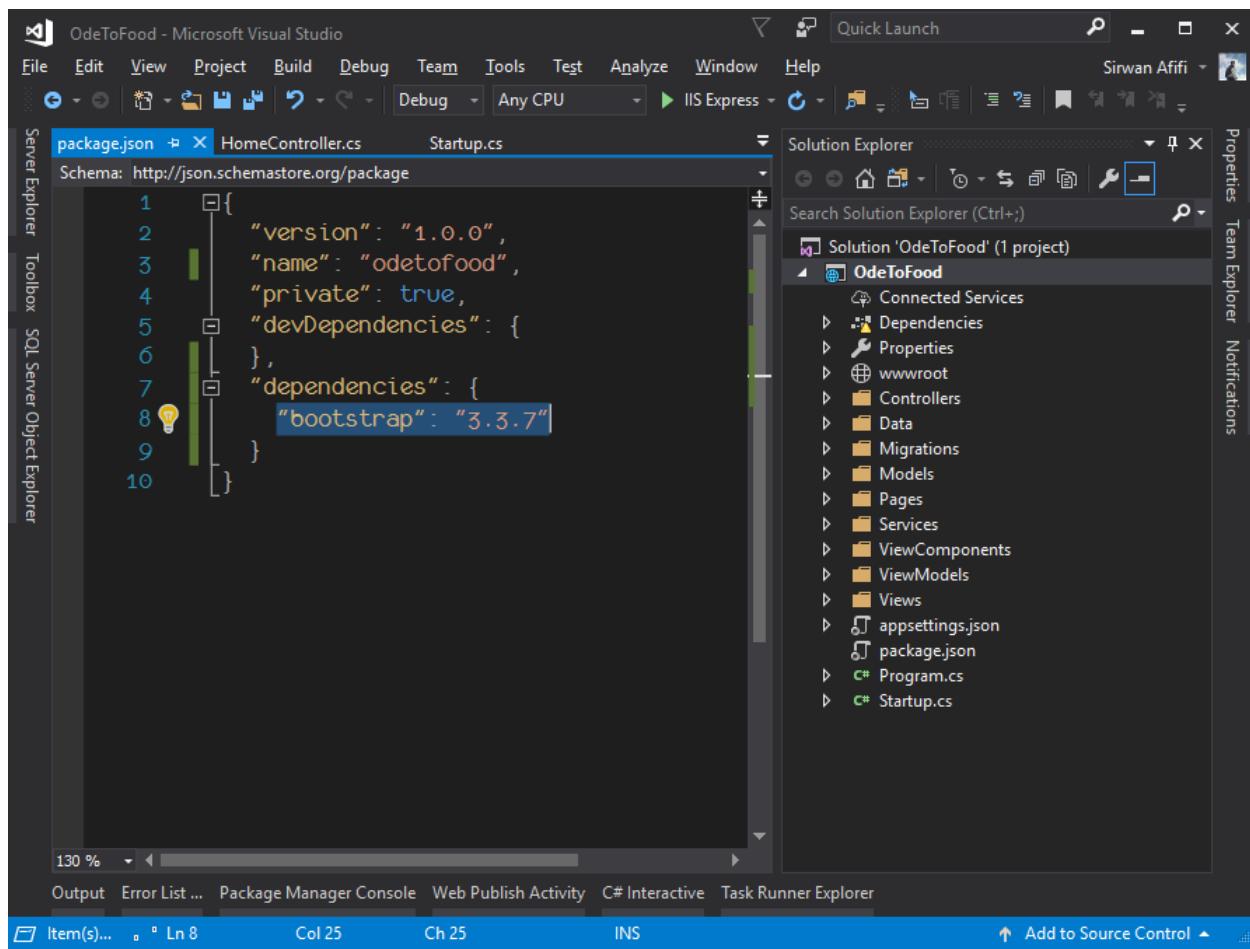
با تغییر این فایل و ذخیره آن، پوششی npm به قسمت Dependencies اضافه خواهد شد؛ ویژال استودیو تشخیص داده است که از npm می‌خواهیم استفاده کنیم:

The screenshot shows the Microsoft Visual Studio interface with the following details:

- Title Bar:** OdeToFood - Microsoft Visual Studio
- Menu Bar:** File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, Help
- Toolbars:** Standard, Debug, IIS Express
- Solution Explorer:** Shows the project 'OdeToFood' with its structure:
 - Connected Services
 - Dependencies
 - Properties
 - wwwroot
 - Controllers
 - Data
 - Migrations
 - Models
 - Pages
 - Services
 - ViewComponents
 - ViewModels
 - Views
 - appsettings.json
 - package.json
 - Program.cs
 - Startup.cs
- Properties Tab:** Team Explorer, Notifications
- Code Editor:** The file 'package.json' is open, showing the following JSON content:

```
1 {  
2   "version": "1.0.0",  
3   "name": "odetofood",  
4   "private": true,  
5   "devDependencies": {}  
6 }  
7 }
```
- Bottom Status Bar:** 130%, Output, Error List..., Package Manager Console, Web Publish Activity, C# Interactive, Task Runner Explorer, Add to Source Control

برای نصب پکیج‌های مورد نظر می‌توانیم همانند فایل‌های csproj عمل کنیم:



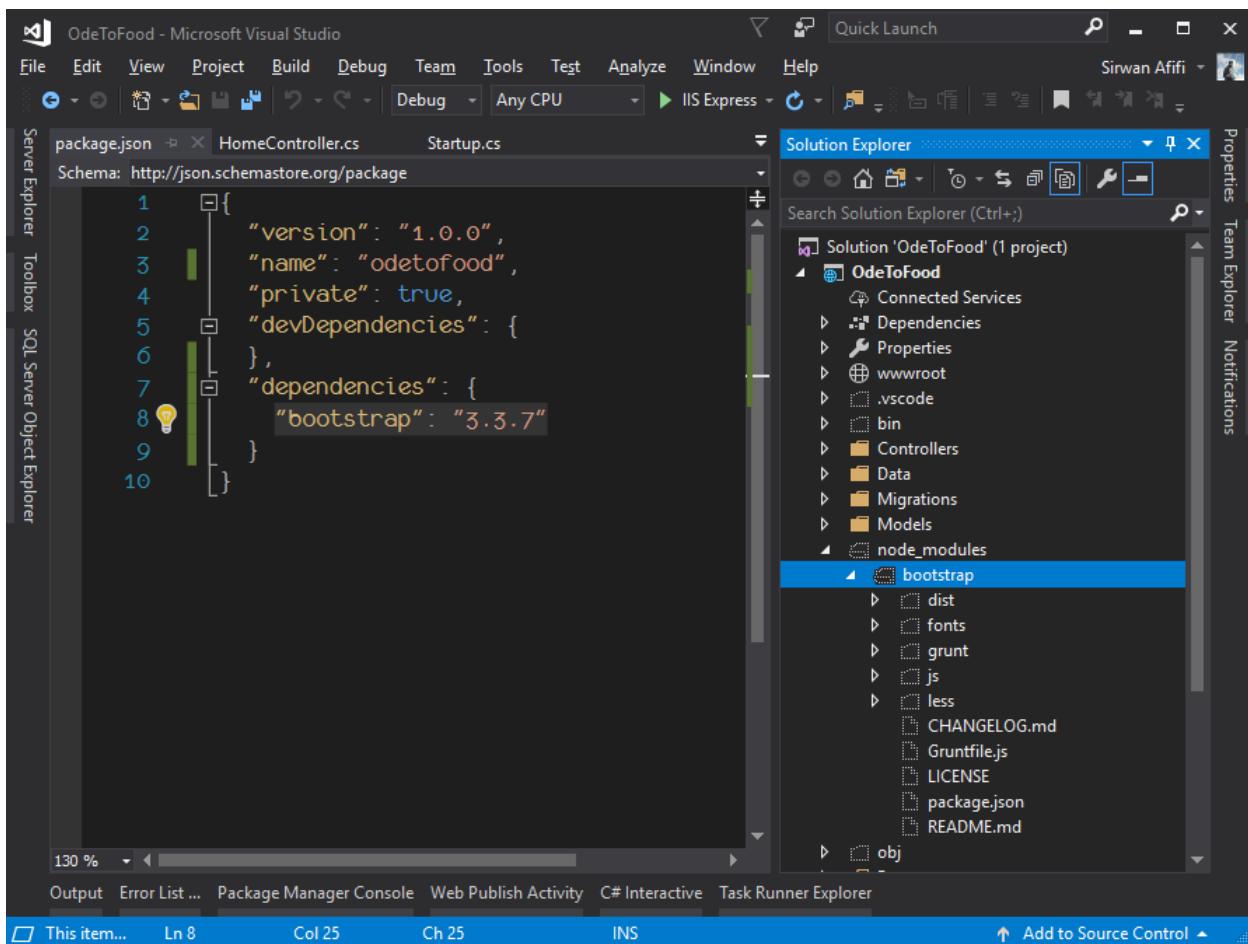
```
1 {  
2     "version": "1.0.0",  
3     "name": "odetofood",  
4     "private": true,  
5     "devDependencies": {  
6     },  
7     "dependencies": {  
8         "bootstrap": "3.3.7"  
9     }  
10 }
```

The screenshot shows the Microsoft Visual Studio interface with the 'OdeToFood' project open. The 'package.json' file is selected in the Solution Explorer. The code editor displays the following JSON content:

```
1 {  
2     "version": "1.0.0",  
3     "name": "odetofood",  
4     "private": true,  
5     "devDependencies": {  
6     },  
7     "dependencies": {  
8         "bootstrap": "3.3.7"  
9     }  
10 }
```

The 'bootstrap' dependency at line 8 is highlighted with a yellow lightbulb icon, indicating a potential issue or suggestion. The Solution Explorer pane shows the project structure, including 'Connected Services', 'Dependencies', 'Properties', 'wwwroot', 'Controllers', 'Data', 'Migrations', 'Models', 'Pages', 'Services', 'ViewComponents', 'ViewModels', 'Views', 'appsettings.json', 'Program.cs', and 'Startup.cs'. The 'Properties' tab is selected in the ribbon.

اما سوال اینجاست که فایل bootstrap در کجا ذخیره شده است؟



در قسمت بعد نحوه serve کردن دایرکتوری node_modules را بررسی خواهیم کرد.

Serving File from node_modules

تا اینجا تنها محتویات درون دایرکتوری wwwroot serve توسط مرورگر خواهند شد؛ برای کردن محتویات داخل دایرکتوری node_modules باید به صورت زیر عمل کنیم.

اگر به خاطر داشته باشید توسط middleware زیر توانستیم یک File Server را جهت serve کردن محتویات داخل دایرکتوری wwwroot ایجاد کنیم:

```
app.UseFileServer();
```

میان افزار فوق در پشت صحنه از StaticFiles استفاده خواهد کرد. در ادامهی کد فوق serve کردن دایرکتوری wwwroot پیکربندی خواهیم کرد؛ برای جلوگیری از شلوغ شدن (cluttering up) فایل Startup کلاس زیر را درون یک دایرکتوری جدید با نام Middleware در ریشهی پروژه قرار خواهیم داد:

```
using Microsoft.AspNetCore.Builder;
```

```

using Microsoft.Extensions.FileProviders;
using System.IO;

namespace Microsoft.AspNetCore.Builder
{
    public static class ApplicationBuilderExtension
    {
        public static IApplicationBuilder UseNodeModules(
            this IApplicationBuilder app, string root)
        {
            var path = Path.Combine(root, "node_modules");

            var provider = new PhysicalFileProvider(path);

            var options = new StaticFileOptions();
            options.RequestPath = "/node_modules";
            options.FileProvider = provider;

            app.UseStaticFiles(options);

            return app;
        }
    }
}

```

همانطور که مشاهده می کنید فضای نام را به **Microsoft.AspNetCore.Builder** تغییر داده ایم؛ در واقع مرسوم است زمانیکه یک متدهایی برای **IApplicationBuilder** طراحی می کنید، بهتر است آن را در همان فضای نامی قرار دهید که این اینترفیس در آن واقع شده است.

برای استفاده از متدهای فوق:

```
app.UseNodeModules(env.ContentRootPath);
```

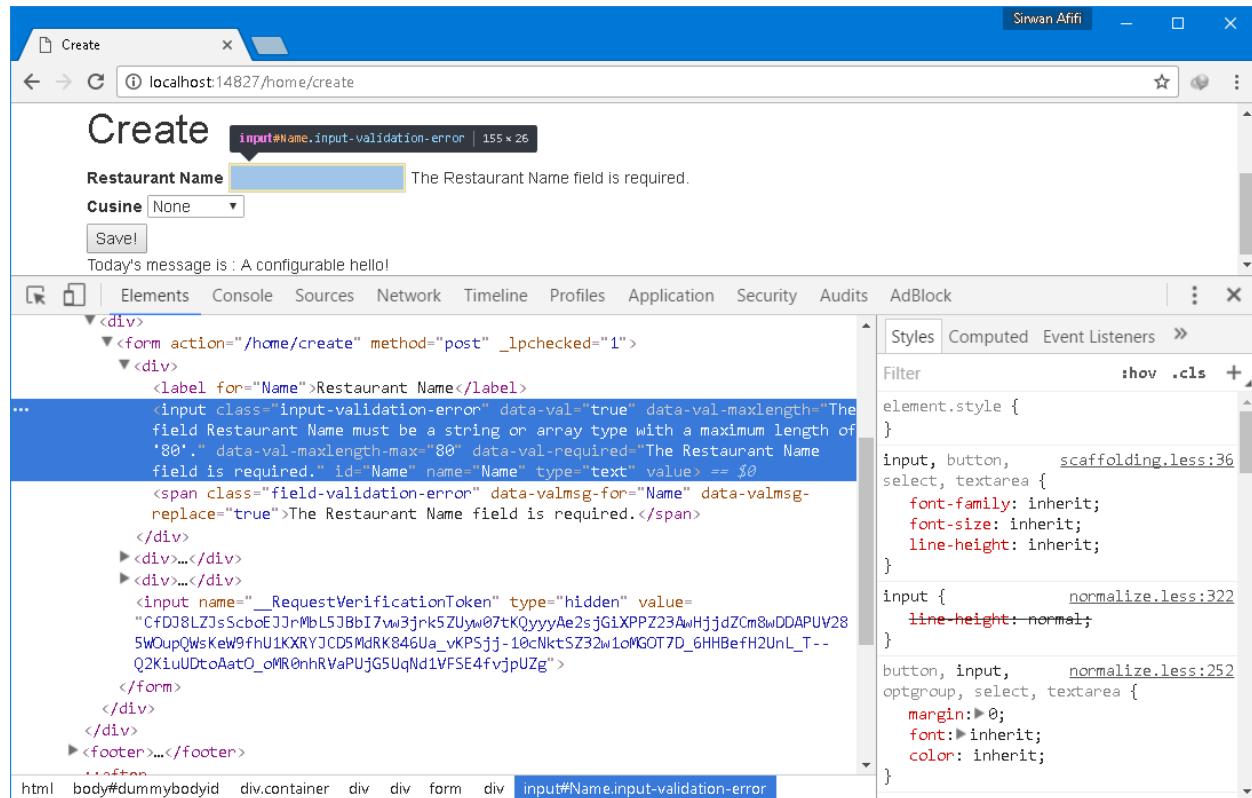
البته بهتر است از یک **build system** یا **task runner** مانند **grunt** یا **gulp** به عنوان یک جهت انتقال فایل ها از **wwwroot** به **node_modules** استفاده کنید.

اکنون می توانیم از فایل های داخل دایرکتوری **node_modules** استفاده کنیم:

```
<link href="/node_modules/bootstrap/dist/css/bootstrap.css"
rel="stylesheet"/>
```

Enabling Client-side Validation

در حالت وجود خطای مدل اگر صفحه را inspect کنید خواهد دید که تعدادی اtribut سفارشی به input ها اضافه شده است:



اینها در واقع توسط فریمورک MVC براساس Data Annotations که برای مدل تعیین کردیم، تولید شده‌اند. مرورگر نمی‌داند با این اtribut‌های تولید شده چه کاری انجام دهد، اما ایده اصلی این است که این اtribut‌ها توسط جاوا اسکریپت مورداستفاده قرار گیرند. اینکار را می‌توانیم توسط کتابخانه‌ایی با نام jQuery Validation انجام دهیم.

پیش‌نیاز لازم برای استفاده از jQuery Validation

قدم اول: نصب وابستگی‌های زیر از طریق (NPM(package.json))



```
package.json
```

```
Schema: http://json.schemastore.org/package
```

```
1 {  
2   "version": "1.0.0",  
3   "name": "odetofood",  
4   "private": true,  
5   "devDependencies": {  
6     },  
7   "dependencies": {  
8     "bootstrap": "3.3.7",  
9     "jquery": "3.1.0",  
10    "jquery-validation": "1.15.1",  
11    "jquery-validation-unobtrusive": "3.2.6"  
12  }  
13}
```

ارجاع به وابستگی‌های فوق درون فایل لی اوت

```
_Layout.cshtml
```

```
22 <footer>  
23   @RenderSection("footer", false)  
24   @await Component.InvokeAsync("Greeting")  
25 </footer>  
26 </div>  
27   <script src="/node_modules/jquery/dist/jquery.js"></script>  
28   <script src="/node_modules/jquery-validation/dist/jquery.validate.js"></script>  
29   <script src="/node_modules/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js"></script>  
30 </body>  
31 </html>
```

اکنون با کلیک بر روی دکمه **Save** جهت نمایش خطاهای مدل، نیاز به **post back** نیست؛ بلکه اعتبارسنجی در سمت کلاینت انجام خواهد شد.

Using CDNs and Fallbacks

برای اپلیکیشن‌هایی که حجم زیادی از جاوا اسکریپت استفاده می‌کنند، داشتن یک **build process** جاوا اسکریپت است؛ یعنی استفاده از یک **task runner** مانند **gulp** و... به طوریکه فایل‌های جاوا اسکریپت را به صورت **minify** شده تحویل کلاینت دهیم. روش دیگر قرار دادن اسکریپت‌ها در **CDNs** است.

The screenshot shows a Microsoft browser window displaying the Microsoft Ajax Content Delivery Network page at <https://www.asp.net/ajax/cdn>. The page lists various JavaScript and CSS files available on the CDN, categorized under 'Table of Contents'.

- ajax.microsoft.com renamed to ajax.aspnetcdn.com
- Visual Studio .vsdoc Support
- Using ASP.NET Ajax from the CDN
- Using jQuery from the CDN
- Using jQuery UI from the CDN
- Third-Party File on the CDN
- jQuery Releases on the CDN
- jQuery Migrate Releases on the CDN
- jQuery UI Releases on the CDN
- jQuery Validation Releases on the CDN
- jQuery Mobile Releases on the CDN
- jQuery Templates Releases on the CDN
- jQuery Cycle Releases on the CDN
- jQuery DataTables Releases on the CDN
- Modernizr Releases on the CDN
- JSHint Releases on the CDN
- Knockout Releases on the CDN
- Globalize Releases on the CDN
- Respond Releases on the CDN
- Bootstrap Releases on the CDN
- Bootstrap TouchCarousel Releases on the CDN
- Hammer.js Releases on the CDN

اما در بیشتر اوقات از اسکریپت‌های لوکال برای **development** و در حالت **production** از اسکریپت‌های موجود بر روی **CDN** استفاده می‌شود. اینکار را می‌توانیم توسط **Tag Helper** زیر مدیریت کنیم:

The screenshot shows the `_Layout.cshtml` file in the Visual Studio code editor. It demonstrates the use of `<environment>` blocks to serve different scripts based on the deployment environment.

```

23     @RenderSection("footer", false)
24     @await Component.InvokeAsync("Greeting")
25   </div>
26 <environment names="Development">
27   <script src="/node_modules/jquery/dist/jquery.js"></script>
28   <script src="/node_modules/jquery-validation/dist/jquery.validate.js"></script>
29   <script src="/node_modules/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js"></script>
30 </environment>
31 <environment names="Staging, Production">
32   <script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-3.1.1.js"
33     asp-fallback-src="/node_modules/jquery/dist/jquery.js"
34     asp-fallback-test="windows.jQuery">
35   </script>
36   <script src="http://ajax.aspnetcdn.com/ajax/jquery.validate/1.15.1/jquery.validate.js"
37     asp-fallback-src="/node_modules/jquery-validation/dist/jquery.validate.js"
38     asp-fallback-test="windows.jQuery & windows.jQuery.validator">
39   </script>
40   <script src="http://ajax.aspnetcdn.com/ajax/mvc/5.2.3/jquery.validate.unobtrusive.min.js"
41     asp-fallback-src="/node_modules/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js"
42     asp-fallback-test="windows.jQuery & windows.jQuery.validator & windows.jQuery.validator">
43   </script>
44 </environment>
45 </body>

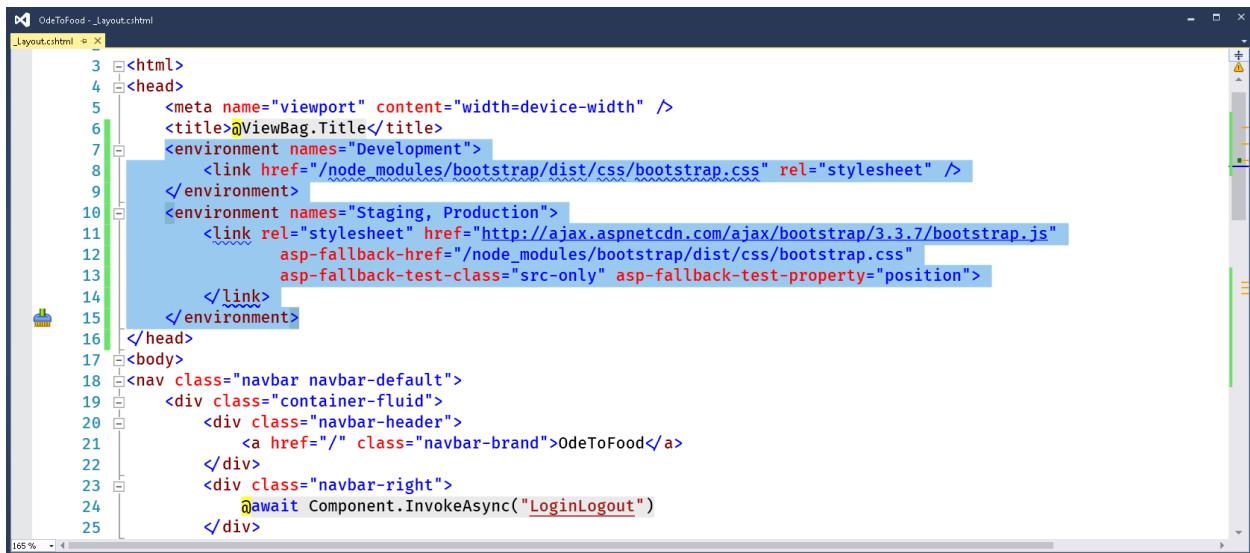
```

همانطور که مشاهده می‌کنید در حالت **Production** و **Development** اسکریپت‌های لازم را بارگذاری کرده‌ایم. در حالت‌های **Staging** و **Production** دو اتریبیوت داریم:

asp-fallback-src -
asp-fallback-test -

اگر تحت شرایطی **CDN** داون شود؛ اسکریپت موردنظر از آدرس تعیین شده در **asp-fallback-src** بارگذاری خواهد شد. توسط **asp-fallback-test** نیز شروط لازم از اینکه اسکریپت موردنظر با موفقیت بارگذاری شده است، انجام شده است. یعنی اگر شروط تعیین شده **fail** شود، اسکریپت از آدرس تعیین شده در **asp-fallback-src** بارگذاری خواهد شد.

لازم به ذکر است که برای استایل‌ها نیز می‌توانیم همینکار را انجام دهیم:



The screenshot shows the Visual Studio code editor with the file `_Layout.cshtml` open. The code defines a head section with environment-specific CSS and JavaScript links. It includes logic to fall back to local files if the CDN fails. The code is as follows:

```
1<html>
2  <head>
3    <meta name="viewport" content="width=device-width" />
4    <title>@ViewBag.Title</title>
5    <environment names="Development">
6      <link href="/node_modules/bootstrap/dist/css/bootstrap.css" rel="stylesheet" />
7    </environment>
8    <environment names="Staging, Production">
9      <link rel="stylesheet" href="http://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/bootstrap.css"
10        asp-fallback-href="/node_modules/bootstrap/dist/css/bootstrap.css"
11        asp-fallback-test-class="src-only" asp-fallback-test-property="position">
12      </link>
13    </environment>
14  </head>
15 <body>
16   <nav class="navbar navbar-default">
17     <div class="container-fluid">
18       <div class="navbar-header">
19         <a href="/" class="navbar-brand">OdeToFood</a>
20       </div>
21       <div class="navbar-right">
22         @await Component.InvokeAsync("LoginLogout")
23       </div>
24     </nav>
25   </body>
```

در اینحالت اپلیکیشنمان باید سریع‌تر از حالت قبل باشد.