Create a new ASP.NET Core application

1. Open Visual Studio 2015 Update 3
2. Create a new ASP.NET Core application:
3. File -> New -> Project -> C# -> .NET Core -> ASP.NET Core Web Application (.NET Core) (Empty Template)

VSCode

1. Run `yo aspnet`
2. Select `Empty Web Application` and give it a name of your choosing

Running the application under IIS

1. The application should be setup to run under IIS Express by default.
2. Run the application and navigate to the root. It should show a "Hello World" message (generated by inline middleware in Startup.cs).

VSCode

1. Open VSCode

2. Open the extension view (`Cmd` + `Shift` + `X` / `Ctrl` + `Shift` + `X`, or click the 5th icon on the left)

3. Install the C# extension. Restart VSCode if prompted.

4. Open the folder you created earlier from the `File`-menu (or open the terminal, navigate to the folder and type `code .`)

5. VSCode should prompt you to restore packages, do so. Alternatively, run `dotnet restore` from the terminal.

6. VSCode should also prompt you to install resources needed to build and debug the project. Do so.

7. There should now be a `.vscode` folder with a file `launch.json`. It should look something like this:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": ".NET Core Launch (web)",
      "type": "coreclr",
      "request": "launch",
      "preLaunchTask": "build",
      "program":
        "${workspaceRoot}\\bin\\Debug\\netcoreapp1.0\\EmptyWebApplication.dll",
      "args": [],
      "cwd": "${workspaceRoot}",
      "stopAtEntry": false,
      "internalConsoleOptions": "openOnSessionStart",
      "launchBrowser": {
        "enabled": true,
        "args": "${auto-detect-url}",
        "windows": {
          "command": "cmd.exe",
          "args": "/C start ${auto-detect-url}"
```

```
        },
        "osx": {
          "command": "open"
        },
        "linux": {
          "command": "xdg-open"
        }
      },
      "env": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      },
      "sourceFileMap": {
        "/Views": "${workspaceRoot}/Views"
      }
    },
    {
      "name": ".NET Core Attach",
      "type": "coreclr",
      "request": "attach",
      "processId": "${command.pickProcess}"
    }
  ]
}
```

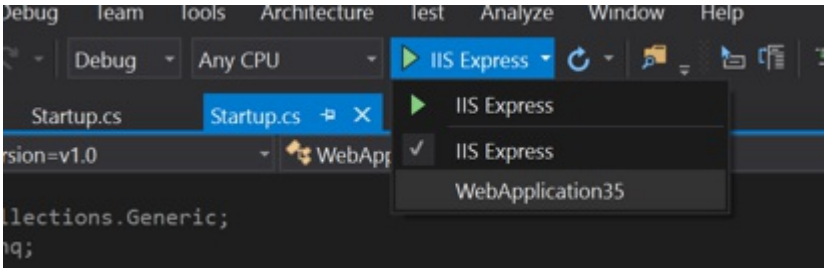
8. Press F5 to run the project from VSCode. If prompted for a platform, select ".NET Core"

9. Alternatively, run `dotnet run` from the terminal instead.

10. Navigate to `http://localhost:5000`. It should show a "Hello World" message (generated by inline middleware in Startup.cs).

Running the application on Kestrel directly (Visual Studio only)

1. Change the Debug drop down in the toolbar to the application name as shown below.



1. Run the application and navigate to the root. It should show the hello world middleware.

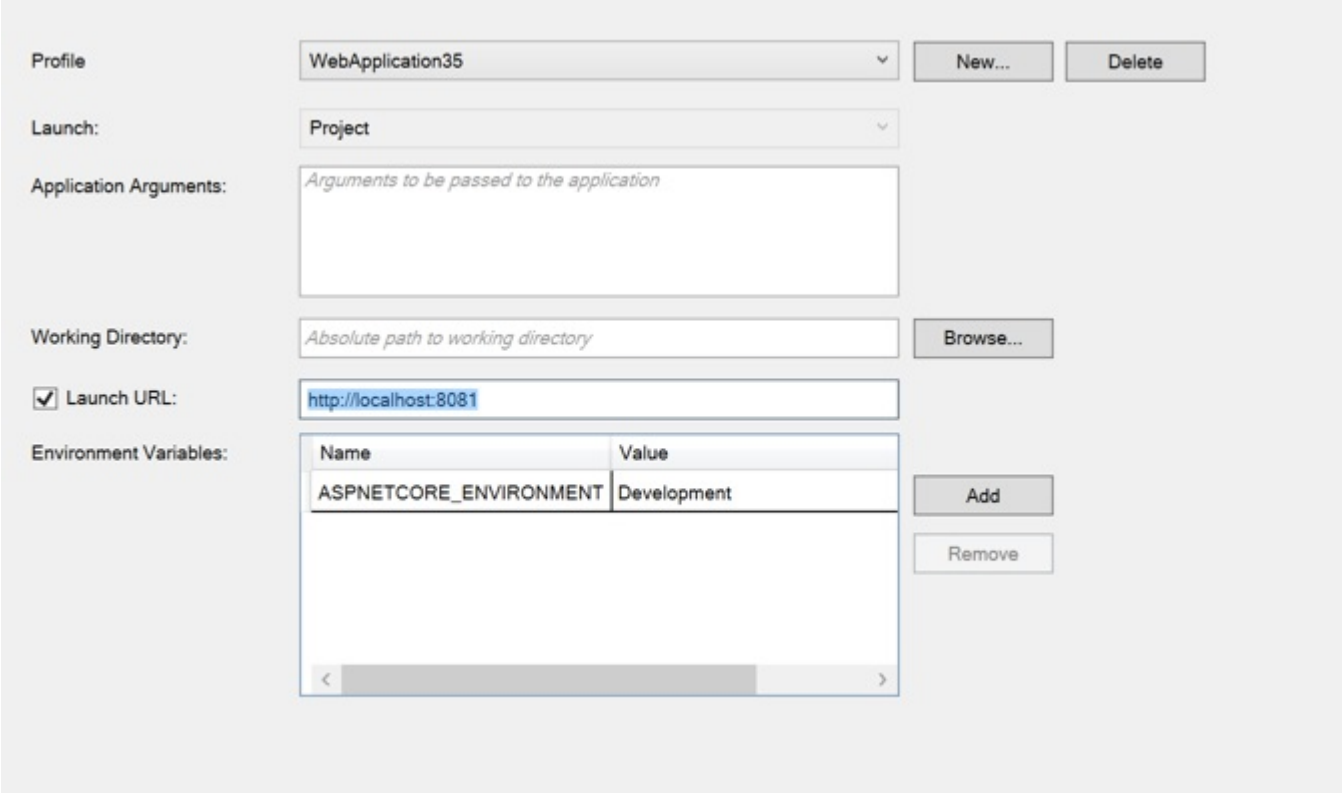
2. Change the port to `8081` by adding a call to `UseUrls` in the `Program.cs`:

```
public class Program
{
    public static void Main(string[] args)
    {
        var host = new WebHostBuilder()
            .UseKestrel()
            .UseUrls("http://localhost:8081")
            .UseContentRoot(Directory.GetCurrentDirectory())
            .UseIISIntegration()
            .UseStartup<Startup>()
            .Build();
        host.Run();

    }
```

```
}
```

3. Navigate to the project properties (by right clicking on the project, and selection `Properties`)

4. Go to the `Debug` tab and change `Launch URL` to `http://localhost:8081`



5. Run the application and navigate to the root. It should show the hello world middleware running on port 8081.

   Note: If the page does not load correctly, verify that the console application host is running and refresh the browser.

Serving static files

1. Add the `Microsoft.AspNetCore.StaticFiles` package to `project.json`:

```
"dependencies": {
  "Microsoft.NETCore.App": {
    "version": "1.0.0",
    "type": "platform"
  },
  "Microsoft.AspNetCore.Diagnostics": "1.0.0",

  "Microsoft.AspNetCore.Server.IISIntegration": "1.0.0",
  "Microsoft.AspNetCore.Server.Kestrel": "1.0.0",
  "Microsoft.Extensions.Logging.Console": "1.0.0",
  "Microsoft.AspNetCore.StaticFiles": "1.0.0"
},
```

2. Save `project.json`. Visual Studio will immediately begin restoring the StaticFiles NuGet package.

3. Go to `Startup.cs` in the `Configure` method and add `UseStaticFiles` before the hello world middleware:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
  ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole();
```

```
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseStaticFiles();

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

4. Create a file called `index.html` with the following contents in the `wwwroot` folder:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title></title>
</head>
<body>
    <h1>Hello from ASP.NET Core!</h1>
</body>
</html>
```

5. Run the application and navigate to the root. It should show the hello world middleware.

6. Navigate to `index.html` and it should show the static page in `wwwroot`.

Adding default document support

1. Change the static files middleware in `Startup.cs` from `app.UseStaticFiles()` to `app.UseFileServer()`.
2. Run the application. The default page `index.html` should show when navigating to the root of the site.

Changing environments

1. The default environment in Visual Studio is Development. In the property pages you can see this is specified by the environment variables section:

2. In VSCode, environment variables can be specified in `launch.json`, under `env`:

```
"env": {
    "ASPNETCORE_ENVIRONMENT": "Development"
},
```

3. If using the terminal instead, environment variables can be specified using `export`

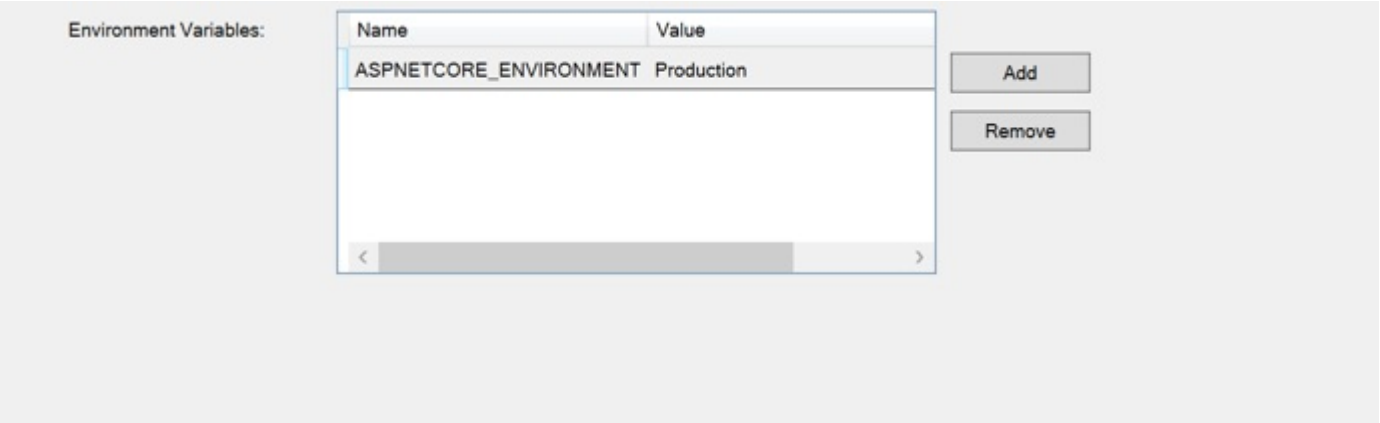
`ASPNETCORE_ENVIRONMENT=Development` in OS X, or `set ASPNETCORE_ENVIRONMENT=Development` in Windows.

4. Add some code to the `Configure` method in `Startup.cs` to print out the environment name. Make sure you comment out the UseFileServer middleware. Otherwise you'll still get the same default static page.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
  ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    //app.UseFileServer();

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync($"Hello World! {env.EnvironmentName}");
    });
}
```

5. Run the application and it should print out `Hello World! Development`.

6. Change the application to run in the `Production` environment by changing the `ASPNETCORE_ENVIRONMENT` environment variable on the `Debug` property page (or the respective method if using VSCode or the terminal):



1. Run the application and it should print out `Hello World! Production`.

Setup the configuration system

1. Add the `Microsoft.Extensions.Configuration.Json` package to `project.json`:

```
"dependencies": {
  "Microsoft.NETCore.App": {
    "version": "1.0.0",
    "type": "platform"
  },
  "Microsoft.AspNetCore.Diagnostics": "1.0.0",

  "Microsoft.AspNetCore.Server.IISIntegration": "1.0.0",
  "Microsoft.AspNetCore.Server.Kestrel": "1.0.0",
  "Microsoft.Extensions.Logging.Console": "1.0.0",
  "Microsoft.AspNetCore.StaticFiles": "1.0.0",
  "Microsoft.Extensions.Configuration.Json": "1.0.0"
```

```
},
```

2. Add a `Configuration` property to `Startup.cs` of type `IConfigurationRoot`:

```
public class Startup
{
    ...
    public IConfigurationRoot Configuration { get; set; }
    ...
}
```

3. Also in `Startup.cs`, add a constructor to the Startup class that configures the configuration system:

```
public Startup()
{
    Configuration = new ConfigurationBuilder()
                        .AddJsonFile("appsettings.json")
                        .Build();
}
```

4. Run the application. It should fail with an exception saying that it cannot find the `'appsettings.json'`.

5. Create a file in the root of the project called `appsettings.json` with the following content:

```
{
  "message": "Hello from configuration"
}
```

6. Modify the `Startup` constructor in `Startup.cs` to inject `IHostingEnvironment` and use it to set the base path for the configuration system to the `ContentRootPath`:

```
public Startup(IHostingEnvironment env)
{
    Configuration = new ConfigurationBuilder()
                        .SetBasePath(env.ContentRootPath)
                        .AddJsonFile("appsettings.json")
                        .Build();
}
```

7. In `Startup.cs` modify the `Configure` method to print out the configuration key in the http response:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
  ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    //app.UseFileServer();

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync($"{Configuration["message"]}");
    });
}
```

8. Run the application and it should print out `Hello from config`.

Extra

- Add support for reloading the configuration without an application restart.
- Replace the JSON configuration provider with the XML configuration provider
- Write a custom configuration provider