`C:\Program Files\dotnet\` to a position before `C:\Program Files (x86)\dotnet\` on the `PATH`.

Obtain data from an app

If an app is capable of responding to requests, you can obtain the following data from the app using middleware:

- Request: Method, scheme, host, pathbase, path, query string, headers
- Connection: Remote IP address, remote port, local IP address, local port, client certificate
- Identity: Name, display name
- Configuration settings
- Environment variables

Place the following middleware code at the beginning of the `Startup.Configure` method's request processing pipeline. The environment is checked before the middleware is run to ensure that the code is only executed in the Development environment.

To obtain the environment, use either of the following approaches:

- Inject the `IHostingEnvironment` into the `Startup.Configure` method and check the environment with the local variable. The following sample code demonstrates this approach.

- Assign the environment to a property in the `Startup` class. Check the environment using the property (for example, `if (Environment.IsDevelopment())`).

C#

```csharp
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
    IConfiguration config)
{
    if (env.IsDevelopment())
    {
        app.Run(async (context) =>
        {
            var sb = new StringBuilder();
            var nl = System.Environment.NewLine;
            var rule = string.Concat(nl, new string('-', 40), nl);
            var authSchemeProvider = app.ApplicationServices
                .GetRequiredService<IAuthenticationSchemeProvider>();

            sb.Append($"Request{rule}");
            sb.Append($"{DateTimeOffset.Now}{nl}");
            sb.Append($"{context.Request.Method} {context.Request.Path}
{nl}");
            sb.Append($"Scheme: {context.Request.Scheme}{nl}");
            sb.Append($"Host: {context.Request.Headers["Host"]}{nl}");
            sb.Append($"PathBase: {context.Request.PathBase.Value}{nl}");
            sb.Append($"Path: {context.Request.Path.Value}{nl}");
            sb.Append($"Query: {context.Request.QueryString.Value}{nl}
{nl}");

            sb.Append($"Connection{rule}");
            sb.Append($"RemoteIp: {context.Connection.RemoteIpAddress}
{nl}");
            sb.Append($"RemotePort: {context.Connection.RemotePort}{nl}");
            sb.Append($"LocalIp: {context.Connection.LocalIpAddress}{nl}");
            sb.Append($"LocalPort: {context.Connection.LocalPort}{nl}");
            sb.Append($"ClientCert: {context.Connection.ClientCertificate}
{nl}{nl}");

            sb.Append($"Identity{rule}");
            sb.Append($"User: {context.User.Identity.Name}{nl}");
            var scheme = await authSchemeProvider
                .GetSchemeAsync(IISDefaults.AuthenticationScheme);
            sb.Append($"DisplayName: {scheme?.DisplayName}{nl}{nl}");

            sb.Append($"Headers{rule}");
            foreach (var header in context.Request.Headers)
            {
                sb.Append($"{header.Key}: {header.Value}{nl}");
            }
            sb.Append(nl);

            sb.Append($"WebSockets{rule}");
            if (context.Features.Get<IHttpUpgradeFeature>() != null)
            {
                sb.Append($"Status: Enabled{nl}{nl}");
            }
            else
            {
                sb.Append($"Status: Disabled{nl}{nl}");
            }

            sb.Append($"Configuration{rule}");
            foreach (var pair in config.AsEnumerable())
            {
                sb.Append($"{pair.Path}: {pair.Value}{nl}");
            }
            sb.Append(nl);

            sb.Append($"Environment Variables{rule}");
            var vars = System.Environment.GetEnvironmentVariables();
            foreach (var key in vars.Keys.Cast<string>().OrderBy(key => key,
                StringComparer.OrdinalIgnoreCase))
            {
                var value = vars[key];
                sb.Append($"{key}: {value}{nl}");
            }

            context.Response.ContentType = "text/plain";
            await context.Response.WriteAsync(sb.ToString());
        });
    }
}
```

Debug ASP.NET Core apps

The following links provide information on debugging ASP.NET Core apps.

- Debugging ASP Core on Linux
- Debugging .NET Core on Unix over SSH
- Quickstart: Debug ASP.NET with the Visual Studio debugger
- See this GitHub issue for more debugging information.

Troubleshoot ASP.NET Core on Azure App Service and IIS

Article • 09/27/2024

This article provides information on common app startup errors and instructions on how to diagnose errors when an app is deployed to Azure App Service or IIS:

App startup errors
Explains common startup HTTP status code scenarios.

Troubleshoot on Azure App Service
Provides troubleshooting advice for apps deployed to Azure App Service.

Troubleshoot on IIS
Provides troubleshooting advice for apps deployed to IIS or running on IIS Express locally. The guidance applies to both Windows Server and Windows desktop deployments.

Clear package caches
Explains what to do when incoherent packages break an app when performing major upgrades or changing package versions.

Additional resources
Lists additional troubleshooting topics.

App startup errors

In Visual Studio, the ASP.NET Core project default server is Kestrel. Visual studio can be configured to use IIS Express. A *502.5 - Process Failure* or a *500.30 - Start Failure* that occurs when debugging locally with IIS Express can be diagnosed using the advice in this topic.

403.14 Forbidden

The app fails to start. The following error is logged:

```
The Web server is configured to not list the contents of this directory.
```

The error is usually caused by a broken deployment on the hosting system, which includes any of the following scenarios:

- The app is deployed to the wrong folder on the hosting system.
- The deployment process failed to move all of the app's files and folders to the deployment folder on the hosting system.
- The *web.config* file is missing from the deployment, or the *web.config* file contents are malformed.

Perform the following steps:

1. Delete all of the files and folders from the deployment folder on the hosting system.
2. Redeploy the contents of the app's *publish* folder to the hosting system using your normal method of deployment, such as Visual Studio, PowerShell, or manual deployment:

   - Confirm that the *web.config* file is present in the deployment and that its contents are correct.
   - When hosting on Azure App Service, confirm that the app is deployed to the `D:\home\site\wwwroot` folder.
   - When the app is hosted by IIS, confirm that the app is deployed to the IIS **Physical path** shown in **IIS Manager**'s **Basic Settings**.

3. Confirm that all of the app's files and folders are deployed by comparing the deployment on the hosting system to the contents of the project's *publish* folder.

For more information on the layout of a published ASP.NET Core app, see ASP.NET Core directory structure. For more information on the *web.config* file, see ASP.NET Core Module (ANCM) for IIS.

500 Internal Server Error

The app starts, but an error prevents the server from fulfilling the request.

This error occurs within the app's code during startup or while creating a response. The response may contain no content, or the response may appear as a *500 Internal Server Error* in the browser. The Application Event Log usually states that the app started normally. From the server's perspective, that's correct. The app did start, but it can't generate a valid response. Run the app at a command prompt on the server or enable the ASP.NET Core Module stdout log to troubleshoot the problem.

This error also may occur when the .NET Core Hosting Bundle isn't installed or is corrupted. Installing or repairing the installation of the .NET Core Hosting Bundle (for IIS) or Visual Studio (for IIS Express) may fix the problem.

500.0 In-Process Handler Load Failure

The worker process fails. The app doesn't start.

An unknown error occurred loading ASP.NET Core Module components. Take one of the following actions:

- Contact Microsoft Support (select **Developer Tools** then **ASP.NET Core**).
- Ask a question on Stack Overflow.
- File an issue on our GitHub repository.

500.30 In-Process Startup Failure

The worker process fails. The app doesn't start.

The ASP.NET Core Module attempts to start the .NET Core CLR in-process, but it fails to start. The cause of a process startup failure can usually be determined from entries in the Application Event Log and the ASP.NET Core Module stdout log.

Common failure conditions:

- The app is misconfigured due to targeting a version of the ASP.NET Core shared framework that isn't present. Check which versions of the ASP.NET Core shared framework are installed on the target machine.
- Using Azure Key Vault, lack of permissions to the Key Vault. Check the access policies in the targeted Key Vault to ensure that the correct permissions are granted.

500.31 ANCM Failed to Find Native Dependencies

The worker process fails. The app doesn't start.

The ASP.NET Core Module attempts to start the .NET Core runtime in-process, but it fails to start. The most common cause of this startup failure is when the `Microsoft.NETCore.App` or `Microsoft.AspNetCore.App` runtime isn't installed. If the app is deployed to target ASP.NET Core 3.0 and that version doesn't exist on the machine, this error occurs. An example error message follows:

```
The specified framework 'Microsoft.NETCore.App', version '3.0.0' was not
found.
  - The following frameworks were found:
      2.2.1 at [C:\Program Files\dotnet\x64\shared\Microsoft.NETCore.App]
      3.0.0-preview5-27626-15 at [C:\Program
Files\dotnet\x64\shared\Microsoft.NETCore.App]
      3.0.0-preview6-27713-13 at [C:\Program
Files\dotnet\x64\shared\Microsoft.NETCore.App]
      3.0.0-preview6-27714-15 at [C:\Program
Files\dotnet\x64\shared\Microsoft.NETCore.App]
      3.0.0-preview6-27723-08 at [C:\Program
Files\dotnet\x64\shared\Microsoft.NETCore.App]
```

The error message lists all the installed .NET Core versions and the version requested by the app. To fix this error, either:

- Install the appropriate version of .NET Core on the machine.
- Change the app to target a version of .NET Core that's present on the machine.
- Publish the app as a self-contained deployment.

When running in development (the `ASPNETCORE_ENVIRONMENT` environment variable is set to `Development`), the specific error is written to the HTTP response. The cause of a process startup failure is also found in the Application Event Log.

500.32 ANCM Failed to Load dll

The worker process fails. The app doesn't start.

The most common cause for this error is that the app is published for an incompatible processor architecture. If the worker process is running as a 32-bit app and the app was published to target 64-bit, this error occurs.

To fix this error, either:

- Republish the app for the same processor architecture as the worker process.
- Publish the app as a framework-dependent deployment.

500.33 ANCM Request Handler Load Failure

The worker process fails. The app doesn't start.

The app didn't reference the `Microsoft.AspNetCore.App` framework. Only apps targeting the `Microsoft.AspNetCore.App` framework can be hosted by the ASP.NET Core Module.

To fix this error, confirm that the app is targeting the `Microsoft.AspNetCore.App` framework. Check the `.runtimeconfig.json` to verify the framework targeted by the app.

500.34 ANCM Mixed Hosting Models Not Supported

The worker process can't run both an in-process app and an out-of-process app in the same process.

To fix this error, run apps in separate IIS application pools.

500.35 ANCM Multiple In-Process Applications in same Process

The worker process can't run multiple in-process apps in the same process.

To fix this error, run apps in separate IIS application pools.

500.36 ANCM Out-Of-Process Handler Load Failure

The out-of-process request handler, *aspnetcorev2_outofprocess.dll*, isn't next to the *aspnetcorev2.dll* file. This indicates a corrupted installation of the ASP.NET Core Module.

To fix this error, repair the installation of the .NET Core Hosting Bundle (for IIS) or Visual Studio (for IIS Express).

500.37 ANCM Failed to Start Within Startup Time Limit

ANCM failed to start within the provided startup time limit. By default, the timeout is 120 seconds.

This error can occur when starting a large number of apps on the same machine. Check for CPU/Memory usage spikes on the server during startup. You may need to stagger the startup process of multiple apps.

500.38 ANCM Application DLL Not Found

ANCM failed to locate the application DLL, which should be next to the executable.

This error occurs when hosting an app packaged as a single-file executable using the in-process hosting model. The in-process model requires that the ANCM load the .NET Core app into the existing IIS process. This scenario isn't supported by the single-file deployment model. Use **one** of the following approaches in the app's project file to fix this error:

1. Disable single-file publishing by setting the `PublishSingleFile` MSBuild property to `false`.
2. Switch to the out-of-process hosting model by setting the `AspNetCoreHostingModel` MSBuild property to `OutOfProcess`.

502.5 Process Failure

The worker process fails. The app doesn't start.

The ASP.NET Core Module attempts to start the worker process but it fails to start. The cause of a process startup failure can usually be determined from entries in the Application Event Log and the ASP.NET Core Module stdout log.

A common failure condition is the app is misconfigured due to targeting a version of the ASP.NET Core shared framework that isn't present. Check which versions of the ASP.NET Core shared framework are installed on the target machine. The *shared framework* is the set of assemblies (*.dll* files) that are installed on the machine and referenced by a metapackage such as `Microsoft.AspNetCore.App`. The metapackage reference can specify a minimum required version. For more information, see The shared framework.

The *502.5 Process Failure* error page is returned when a hosting or app misconfiguration causes the worker process to fail:

Failed to start application (ErrorCode '0x800700c1')

```
EventID: 1010
Source: IIS AspNetCore Module V2
Failed to start application '/LM/W3SVC/6/ROOT/', ErrorCode '0x800700c1'.
```

The app failed to start because the app's assembly (*.dll*) couldn't be loaded.

This error occurs when there's a bitness mismatch between the published app and the w3wp/iisexpress process.

Confirm that the app pool's 32-bit setting is correct:

1. Select the app pool in IIS Manager's **Application Pools**.

2. Select **Advanced Settings** under **Edit Application Pool** in the **Actions** panel.
3. Set **Enable 32-Bit Applications**:

   - If deploying a 32-bit (x86) app, set the value to `True`.
   - If deploying a 64-bit (x64) app, set the value to `False`.

Confirm that there isn't a conflict between a `<Platform>` MSBuild property in the project file and the published bitness of the app.

Failed to start application (ErrorCode '0x800701b1')

```
EventID: 1010
Source: IIS AspNetCore Module V2
Failed to start application '/LM/W3SVC/3/ROOT', ErrorCode '0x800701b1'.
```

The app failed to start because a Windows Service failed to load.

One common service that needs to be enabled is the "null" service. The following command enables the `null` Windows Service:

Windows Command Prompt

```
sc.exe start null
```

Connection reset

If an error occurs after the headers are sent, it's too late for the server to send a **500 Internal Server Error** when an error occurs. This often happens when an error occurs during the serialization of complex objects for a response. This type of error appears as a *connection reset* error on the client. Application logging can help troubleshoot these types of errors.

Default startup limits

The ASP.NET Core Module is configured with a default *startupTimeLimit* of 120 seconds. When left at the default value, an app may take up to two minutes to start before the module logs a process failure. For information on configuring the module, see Attributes of the aspNetCore element.

Troubleshoot on Azure App Service

ⓘ Important

ASP.NET Core preview releases with Azure App Service

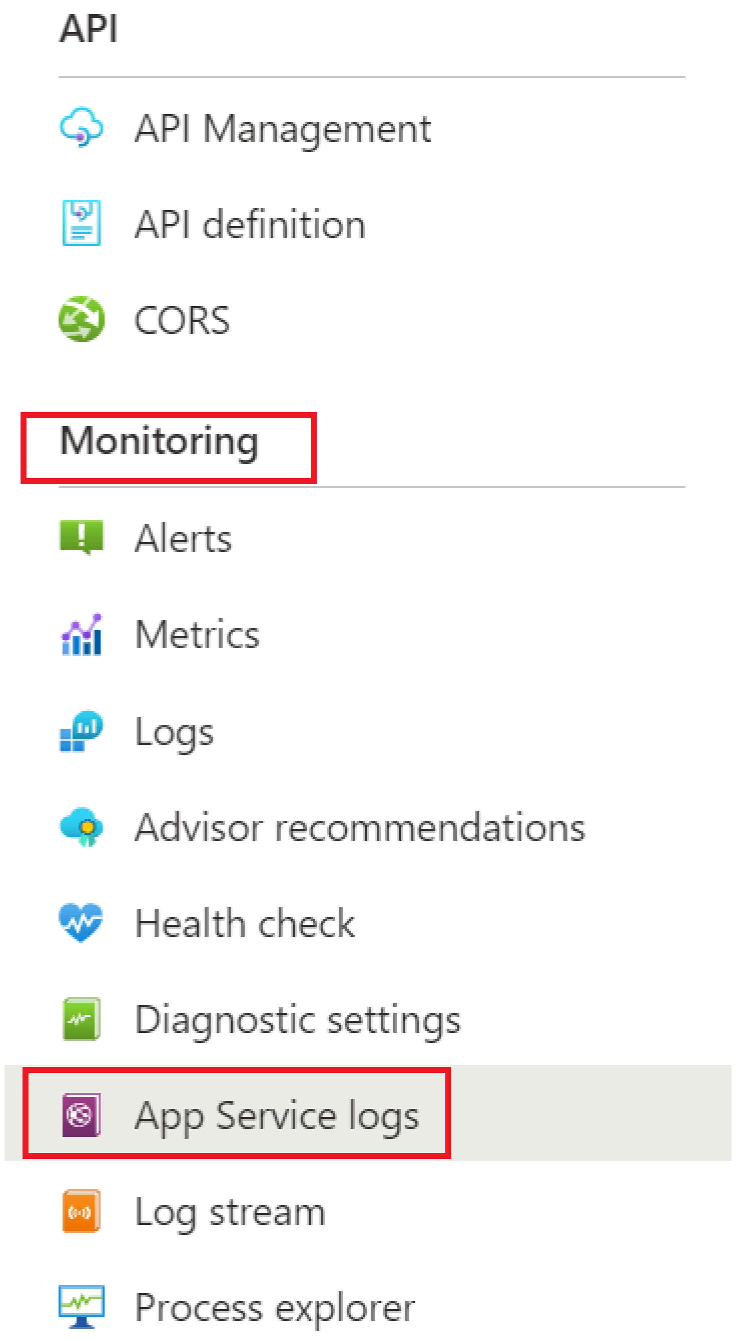
ASP.NET Core preview releases aren't deployed to Azure App Service by default. To host an app that uses an ASP.NET Core preview release, see Deploy ASP.NET Core preview release to Azure App Service.

Azure App Services Log stream

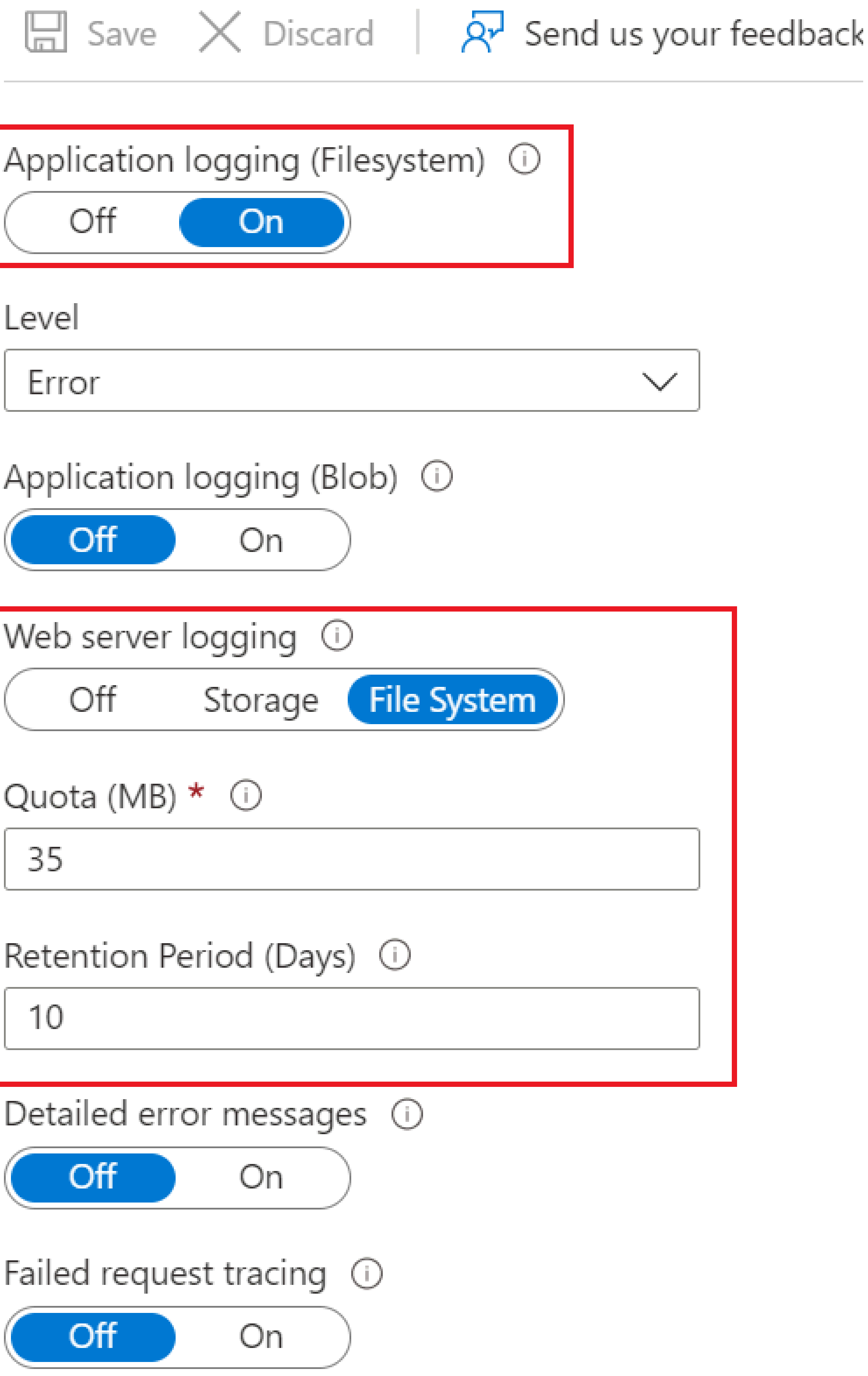
The Azure App Services Log streams logging information as it occurs. To view streaming logs:

1. In the Azure portal, open the app in **App Services**.

2. In the left pane, navigate to **Monitoring** > **App Service Logs**.



3. Select **File System** for **Web Server Logging**. Optionally enable **Application**



Application logging (Filesystem): Off [✓ On]

Level: Error

Application logging (Blob): [✓ Off] On

Web server logging: Off Storage [✓ File System]

Quota (MB) *: 35

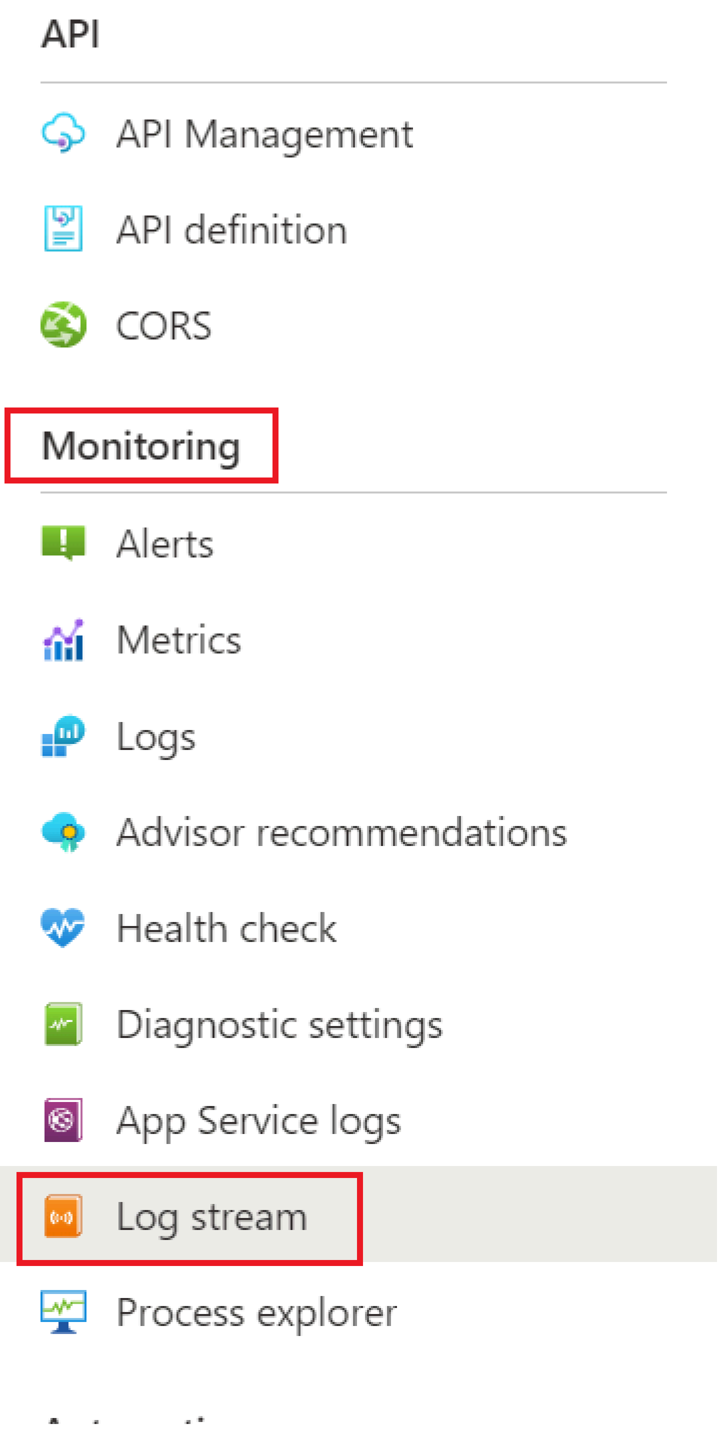
Retention Period (Days): 10

Detailed error messages: [✓ Off] On

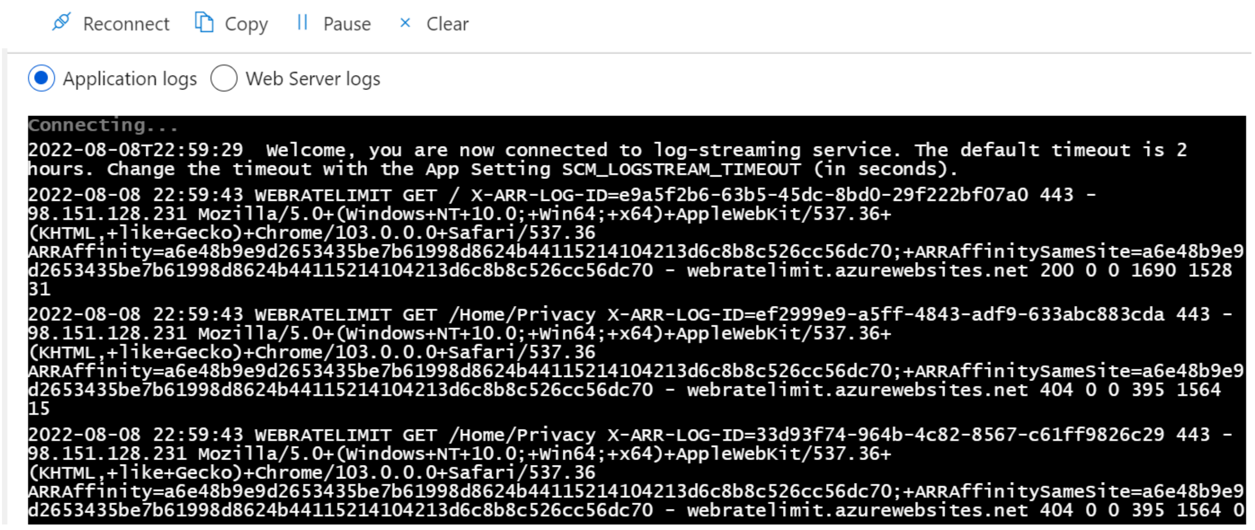
Failed request tracing: [✓ Off] On

logging.

4. In the left pane, navigate to **Monitoring** > **Log stream**, and then select **Application logs** or **Web Server Logs**.



The following images shows the application logs output:



```
Connecting...
2022-08-08T22:59:29  Welcome, you are now connected to log-streaming service. The default timeout is 2
hours. Change the timeout with the App Setting SCM_LOGSTREAM_TIMEOUT (in seconds).
2022-08-08 22:59:43 WEBRATELIMIT GET / X-ARR-LOG-ID=e9a5f2b6-63b5-45dc-8bd0-29f222bf07a0 443 -
98.151.128.231 Mozilla/5.0+(Windows+NT+10.0;+Win64;+x64)+AppleWebKit/537.36+
(KHTML,+like+Gecko)+Chrome/103.0.0.0+Safari/537.36
ARRAffinity=a6e48b9e9d2653435be7b61998d8624b44115214104213d6c8b8c526cc56dc70;+ARRAffinitySameSite=a6e48b9e9
d2653435be7b61998d8624b44115214104213d6c8b8c526cc56dc70 - webratelimit.azurewebsites.net 200 0 0 1690 1528
31
2022-08-08 22:59:43 WEBRATELIMIT GET /Home/Privacy X-ARR-LOG-ID=ef2999e9-a5ff-4843-adf9-633abc883cda 443 -
98.151.128.231 Mozilla/5.0+(Windows+NT+10.0;+Win64;+x64)+AppleWebKit/537.36+
(KHTML,+like+Gecko)+Chrome/103.0.0.0+Safari/537.36
ARRAffinity=a6e48b9e9d2653435be7b61998d8624b44115214104213d6c8b8c526cc56dc70;+ARRAffinitySameSite=a6e48b9e9
d2653435be7b61998d8624b44115214104213d6c8b8c526cc56dc70 - webratelimit.azurewebsites.net 404 0 0 395 1564
15
2022-08-08 22:59:43 WEBRATELIMIT GET /Home/Privacy X-ARR-LOG-ID=33d93f74-964b-4c82-8567-c61ff9826c29 443 -
98.151.128.231 Mozilla/5.0+(Windows+NT+10.0;+Win64;+x64)+AppleWebKit/537.36+
(KHTML,+like+Gecko)+Chrome/103.0.0.0+Safari/537.36
ARRAffinity=a6e48b9e9d2653435be7b61998d8624b44115214104213d6c8b8c526cc56dc70;+ARRAffinitySameSite=a6e48b9e9
d2653435be7b61998d8624b44115214104213d6c8b8c526cc56dc70 - webratelimit.azurewebsites.net 404 0 0 395 1564 0
```

Streaming logs have some latency and might not display immediately.

Application Event Log (Azure App Service)

To access the Application Event Log, use the **Diagnose and solve problems** blade in the Azure portal:

1. In the Azure portal, open the app in **App Services**.
2. Select **Diagnose and solve problems**.
3. Select the **Diagnostic Tools** heading.
4. Under **Support Tools**, select the **Application Events** button.
5. Examine the latest error provided by the *IIS AspNetCoreModule* or *IIS AspNetCoreModule V2* entry in the **Source** column.

An alternative to using the **Diagnose and solve problems** blade is to examine the Application Event Log file directly using Kudu:

1. Open **Advanced Tools** in the **Development Tools** area. Select the **Go→** button. The Kudu console opens in a new browser tab or window.
2. Using the navigation bar at the top of the page, open **Debug console** and select **CMD**.
3. Open the **LogFiles** folder.
4. Select the pencil icon next to the `eventlog.xml` file.
5. Examine the log. Scroll to the bottom of the log to see the most recent events.

Run the app in the Kudu console

Many startup errors don't produce useful information in the Application Event Log. You can run the app in the Kudu Remote Execution Console to discover the error:

1. Open **Advanced Tools** in the **Development Tools** area. Select the **Go→** button. The Kudu console opens in a new browser tab or window.
2. Using the navigation bar at the top of the page, open **Debug console** and select **CMD**.

Test a 32-bit (x86) app

Current release

1. `cd d:\home\site\wwwroot`
2. Run the app:

- If the app is a framework-dependent deployment:

  .NET CLI

  ```
  dotnet .\{ASSEMBLY NAME}.dll
  ```

- If the app is a self-contained deployment:

  Console

  ```
  {ASSEMBLY NAME}.exe
  ```

The console output from the app, showing any errors, is piped to the Kudu console.

Framework-dependent deployment running on a preview release

Requires installing the ASP.NET Core {VERSION} (x86) Runtime site extension.

1. `cd D:\home\SiteExtensions\AspNetCoreRuntime.{X.Y}.x32` (`{X.Y}` is the runtime version)
2. Run the app: `dotnet \home\site\wwwroot\{ASSEMBLY NAME}.dll`

The console output from the app, showing any errors, is piped to the Kudu console.

Test a 64-bit (x64) app

Current release

- If the app is a 64-bit (x64) framework-dependent deployment:

  1. `cd D:\Program Files\dotnet`
  2. Run the app: `dotnet \home\site\wwwroot\{ASSEMBLY NAME}.dll`

- If the app is a self-contained deployment:

  1. `cd D:\home\site\wwwroot`
  2. Run the app: `{ASSEMBLY NAME}.exe`

The console output from the app, showing any errors, is piped to the Kudu console.

Framework-dependent deployment running on a preview release

Requires installing the ASP.NET Core {VERSION} (x64) Runtime site extension.

1. `cd D:\home\SiteExtensions\AspNetCoreRuntime.{X.Y}.x64` (`{X.Y}` is the runtime version)
2. Run the app: `dotnet \home\site\wwwroot\{ASSEMBLY NAME}.dll`

The console output from the app, showing any errors, is piped to the Kudu console.

ASP.NET Core Module stdout log (Azure App Service)

⚠ **Warning**

Failure to disable the stdout log can lead to app or server failure. There's no limit on log file size or the number of log files created. Only use stdout logging to troubleshoot app startup problems.

For general logging in an ASP.NET Core app after startup, use a logging library that limits log file size and rotates logs. For more information, see third-party logging providers.

The ASP.NET Core Module stdout log often records useful error messages not found in the Application Event Log. To enable and view stdout logs:

1. In the Azure Portal, navigate to the web app.
2. In the **App Service** blade, enter **kudu** in the search box.
3. Select **Advanced Tools** > **Go**.
4. Select **Debug console** > **CMD**.
5. Navigate to *site/wwwroot*
6. Select the pencil icon to edit the *web.config* file.
7. In the `<aspNetCore />` element, set `stdoutLogEnabled="true"` and select **Save**.

Disable stdout logging when troubleshooting is complete by setting `stdoutLogEnabled="false"`.

For more information, see ASP.NET Core Module (ANCM) for IIS.

ASP.NET Core Module debug log (Azure App Service)

The ASP.NET Core Module debug log provides additional, deeper logging from the ASP.NET Core Module. To enable and view stdout logs:

1. To enable the enhanced diagnostic log, perform either of the following:

   - Follow the instructions in Enhanced diagnostic logs to configure the app for an enhanced diagnostic logging. Redeploy the app.
   - Add the `<handlerSettings>` shown in Enhanced diagnostic logs to the live app's *web.config* file using the Kudu console:
      a. Open **Advanced Tools** in the **Development Tools** area. Select the **Go→** button. The Kudu console opens in a new browser tab or window.
      b. Using the navigation bar at the top of the page, open **Debug console** and select **CMD**.
      c. Open the folders to the path **site** > **wwwroot**. Edit the *web.config* file by selecting the pencil button. Add the `<handlerSettings>` section as shown in Enhanced diagnostic logs. Select the **Save** button.
2. Open **Advanced Tools** in the **Development Tools** area. Select the **Go→** button. The Kudu console opens in a new browser tab or window.
3. Using the navigation bar at the top of the page, open **Debug console** and select **CMD**.
4. Open the folders to the path **site** > **wwwroot**. If you didn't supply a path for the *aspnetcore-debug.log* file, the file appears in the list. If you supplied a path, navigate to the location of the log file.
5. Open the log file with the pencil button next to the file name.

Disable debug logging when troubleshooting is complete:

To disable the enhanced debug log, perform either of the following:

- Remove the `<handlerSettings>` from the *web.config* file locally and redeploy the app.
- Use the Kudu console to edit the *web.config* file and remove the `<handlerSettings>` section. Save the file.

For more information, see Log creation and redirection with the ASP.NET Core Module.

⚠ **Warning**

Failure to disable the debug log can lead to app or server failure. There's no limit on log file size. Only use debug logging to troubleshoot app startup problems.

For general logging in an ASP.NET Core app after startup, use a logging library that limits log file size and rotates logs. For more information, see third-party logging providers.

Slow or nonresponsive app (Azure App Service)

When an app responds slowly or doesn't respond to a request, see Troubleshoot slow web app performance issues in Azure App Service.

Monitoring blades

Monitoring blades provide an alternative troubleshooting experience to the methods described earlier in the topic. These blades can be used to diagnose 500-series errors.

Confirm that the ASP.NET Core Extensions are installed. If the extensions aren't installed, install them manually:

1. In the **DEVELOPMENT TOOLS** blade section, select the **Extensions** blade.
2. The **ASP.NET Core Extensions** should appear in the list.
3. If the extensions aren't installed, select the **Add** button.
4. Choose the **ASP.NET Core Extensions** from the list.
5. Select **OK** to accept the legal terms.
6. Select **OK** on the **Add extension** blade.
7. An informational pop-up message indicates when the extensions are successfully installed.

If stdout logging isn't enabled, follow these steps:

1. In the Azure portal, select the **Advanced Tools** blade in the **DEVELOPMENT TOOLS** area. Select the **Go→** button. The Kudu console opens in a new browser tab or window.
2. Using the navigation bar at the top of the page, open **Debug console** and select **CMD**.
3. Open the folders to the path **site** > **wwwroot** and scroll down to reveal the *web.config* file at the bottom of the list.
4. Click the pencil icon next to the *web.config* file.
5. Set **stdoutLogEnabled** to `true` and change the **stdoutLogFile** path to: `\\?\%home%\LogFiles\stdout`.
6. Select **Save** to save the updated *web.config* file.

Proceed to activate diagnostic logging:

1. In the Azure portal, select the **Diagnostics logs** blade.
2. Select the **On** switch for **Application Logging (Filesystem)** and **Detailed error messages**. Select the **Save** button at the top of the blade.
3. To include failed request tracing, also known as Failed Request Event Buffering (FREB) logging, select the **On** switch for **Failed request tracing**.
4. Select the **Log stream** blade, which is listed immediately under the **Diagnostics logs** blade in the portal.

5. Make a request to the app.
6. Within the log stream data, the cause of the error is indicated.

Be sure to disable stdout logging when troubleshooting is complete.

To view the failed request tracing logs (FREB logs):

1. Navigate to the **Diagnose and solve problems** blade in the Azure portal.
2. Select **Failed Request Tracing Logs** from the **SUPPORT TOOLS** area of the sidebar.

See Failed request traces section of the Enable diagnostics logging for web apps in Azure App Service topic and the Application performance FAQs for Web Apps in Azure: How do I turn on failed request tracing? for more information.

For more information, see Enable diagnostics logging for web apps in Azure App Service.

⚠ Warning

Failure to disable the stdout log can lead to app or server failure. There's no limit on log file size or the number of log files created.

For routine logging in an ASP.NET Core app, use a logging library that limits log file size and rotates logs. For more information, see third-party logging providers.

Troubleshoot on IIS

Application Event Log (IIS)

Access the Application Event Log:

1. Open the Start menu, search for *Event Viewer*, and select the **Event Viewer** app.
2. In **Event Viewer**, open the **Windows Logs** node.
3. Select **Application** to open the Application Event Log.
4. Search for errors associated with the failing app. Errors have a value of *IIS AspNetCore Module* or *IIS Express AspNetCore Module* in the *Source* column.

Run the app at a command prompt

Many startup errors don't produce useful information in the Application Event Log. You can find the cause of some errors by running the app at a command prompt on the hosting system.

Framework-dependent deployment

If the app is a framework-dependent deployment:

1. At a command prompt, navigate to the deployment folder and run the app by executing the app's assembly with *dotnet.exe*. In the following command, substitute the name of the app's assembly for <assembly_name>: `dotnet .\<assembly_name>.dll`.
2. The console output from the app, showing any errors, is written to the console window.
3. If the errors occur when making a request to the app, make a request to the host and port where Kestrel listens. Using the default host and post, make a request to `http://localhost:5000/`. If the app responds normally at the Kestrel endpoint address, the problem is more likely related to the hosting configuration and less likely within the app.

Self-contained deployment

If the app is a self-contained deployment:

1. At a command prompt, navigate to the deployment folder and run the app's executable. In the following command, substitute the name of the app's assembly for <assembly_name>: `<assembly_name>.exe`.
2. The console output from the app, showing any errors, is written to the console window.
3. If the errors occur when making a request to the app, make a request to the host and port where Kestrel listens. Using the default host and post, make a request to `http://localhost:5000/`. If the app responds normally at the Kestrel endpoint address, the problem is more likely related to the hosting configuration and less likely within the app.

ASP.NET Core Module stdout log (IIS)

To enable and view stdout logs:

1. Navigate to the site's deployment folder on the hosting system.
2. If the *logs* folder isn't present, create the folder. For instructions on how to enable MSBuild to create the *logs* folder in the deployment automatically, see the Directory structure topic.
3. Edit the *web.config* file. Set **stdoutLogEnabled** to `true` and change the **stdoutLogFile** path to point to the *logs* folder (for example, `.\logs\stdout`).

`stdout` in the path is the log file name prefix. A timestamp, process id, and file extension are added automatically when the log is created. Using `stdout` as the file name prefix, a typical log file is named *stdout_20180205184032_5412.log*.

4. Ensure your application pool's identity has write permissions to the *logs* folder.
5. Save the updated *web.config* file.
6. Make a request to the app.
7. Navigate to the *logs* folder. Find and open the most recent stdout log.
8. Study the log for errors.

Disable stdout logging when troubleshooting is complete:

1. Edit the *web.config* file.
2. Set **stdoutLogEnabled** to `false`.
3. Save the file.

For more information, see ASP.NET Core Module (ANCM) for IIS.

⚠ **Warning**

Failure to disable the stdout log can lead to app or server failure. There's no limit on log file size or the number of log files created.

For routine logging in an ASP.NET Core app, use a logging library that limits log file size and rotates logs. For more information, see third-party logging providers.

ASP.NET Core Module debug log (IIS)

Add the following handler settings to the app's *web.config* file to enable ASP.NET Core Module debug log:

XML

```xml
<aspNetCore ...>
  <handlerSettings>
    <handlerSetting name="debugLevel" value="file" />
    <handlerSetting name="debugFile" value="c:\temp\ancm.log" />
  </handlerSettings>
</aspNetCore>
```

Confirm that the path specified for the log exists and that the app pool's identity has write permissions to the location.

For more information, see Log creation and redirection with the ASP.NET Core Module.

Enable the Developer Exception Page

The `ASPNETCORE_ENVIRONMENT` environment variable can be added to web.config to run the app in the Development environment. As long as the environment isn't overridden in app startup by `UseEnvironment` on the host builder, setting the environment variable allows the Developer Exception Page to appear when the app is run.

XML

```xml
<aspNetCore processPath="dotnet"
      arguments=".\MyApp.dll"
      stdoutLogEnabled="false"
      stdoutLogFile=".\logs\stdout"
      hostingModel="InProcess">
  <environmentVariables>
    <environmentVariable name="ASPNETCORE_ENVIRONMENT" value="Development"
/>
  </environmentVariables>
</aspNetCore>
```

Setting the environment variable for `ASPNETCORE_ENVIRONMENT` is only recommended for use on staging and testing servers that aren't exposed to the Internet. Remove the environment variable from the *web.config* file after troubleshooting. For information on setting environment variables in *web.config*, see environmentVariables child element of aspNetCore.

Obtain data from an app

If an app is capable of responding to requests, obtain request, connection, and additional data from the app using terminal inline middleware. For more information and sample code, see Troubleshoot and debug ASP.NET Core projects.

Slow or non-responding app (IIS)

A *crash dump* is a snapshot of the system's memory and can help determine the cause of an app crash, startup failure, or slow app.

App crashes or encounters an exception

Obtain and analyze a dump from Windows Error Reporting (WER):

1. Create a folder to hold crash dump files at `c:\dumps`. The app pool must have write access to the folder.

2. Run the EnableDumps PowerShell script :

   - If the app uses the in-process hosting model, run the script for *w3wp.exe*:

     Console

     ```
     .\EnableDumps w3wp.exe c:\dumps
     ```

   - If the app uses the out-of-process hosting model, run the script for *dotnet.exe*:

     Console

     ```
     .\EnableDumps dotnet.exe c:\dumps
     ```

3. Run the app under the conditions that cause the crash to occur.

4. After the crash has occurred, run the DisableDumps PowerShell script :

   - If the app uses the in-process hosting model, run the script for *w3wp.exe*:

     Console

     ```
     .\DisableDumps w3wp.exe
     ```

   - If the app uses the out-of-process hosting model, run the script for *dotnet.exe*:

     Console

     ```
     .\DisableDumps dotnet.exe
     ```

After an app crashes and dump collection is complete, the app is allowed to terminate normally. The PowerShell script configures WER to collect up to five dumps per app.

⚠ **Warning**

Crash dumps might take up a large amount of disk space (up to several gigabytes each).

App doesn't respond, fails during startup, or runs normally

When an app stops responding but doesn't crash, fails during startup, or runs normally, see User-Mode Dump Files: Choosing the Best Tool to select an appropriate tool to produce the dump.

Analyze the dump

A dump can be analyzed using several approaches. For more information, see Analyzing a User-Mode Dump File.

Clear package caches

A functioning app may fail immediately after upgrading either the .NET Core SDK on the development machine or changing package versions within the app. In some cases, incoherent packages may break an app when performing major upgrades. Most of these issues can be fixed by following these instructions:

1. Delete the *bin* and *obj* folders.
2. Clear the package caches by executing dotnet nuget locals all --clear from a command shell.

   Clearing package caches can also be accomplished with the nuget.exe tool and executing the command `nuget locals all -clear`. *nuget.exe* isn't a bundled install with the Windows desktop operating system and must be obtained separately from the NuGet website.
3. Restore and rebuild the project.
4. Delete all of the files in the deployment folder on the server prior to redeploying the app.

Additional resources

- Debug .NET and ASP.NET Core source code with Visual Studio
- Troubleshoot and debug ASP.NET Core projects
- Common error troubleshooting for Azure App Service and IIS with ASP.NET Core
- Handle errors in ASP.NET Core
- ASP.NET Core Module (ANCM) for IIS

Azure documentation

- Application Insights for ASP.NET Core
- Remote debugging web apps section of Troubleshoot a web app in Azure App Service using Visual Studio
- Azure App Service diagnostics overview
- How to: Monitor Apps in Azure App Service
- Troubleshoot a web app in Azure App Service using Visual Studio
- Troubleshoot HTTP errors of "502 bad gateway" and "503 service unavailable" in your Azure web apps
- Troubleshoot slow web app performance issues in Azure App Service
- Application performance FAQs for Web Apps in Azure
- Azure Web App sandbox (App Service runtime execution limitations)

Visual Studio documentation

- Remote Debug ASP.NET Core on IIS in Azure in Visual Studio 2017
- Remote Debug ASP.NET Core on a Remote IIS Computer in Visual Studio 2017
- Learn to debug using Visual Studio

Visual Studio Code documentation

- Debugging with Visual Studio Code

Common error troubleshooting for Azure App Service and IIS with ASP.NET Core

Article • 07/31/2024

ⓘ **Important**

This information relates to a pre-release product that may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

For the current release, see the .NET 9 version of this article.

This topic describes the most common errors and provides troubleshooting advice when hosting ASP.NET Core apps on Azure Apps Service and IIS.

See Troubleshoot ASP.NET Core on Azure App Service and IIS information on common app startup errors and instructions on how to diagnose errors.

Collect the following information:

- Browser behavior such as status code and error message.
- Application Event Log entries
  - Azure App Service: See Troubleshoot ASP.NET Core on Azure App Service and IIS.
  - IIS
    1. Select **Start** on the **Windows** menu, type *Event Viewer*, and press **Enter**.
    2. After the **Event Viewer** opens, expand **Windows Logs** > **Application** in the sidebar.
- ASP.NET Core Module stdout and debug log entries
  - Azure App Service: See Troubleshoot ASP.NET Core on Azure App Service and IIS.
  - IIS: Follow the instructions in the Log creation and redirection and Enhanced diagnostic logs sections of the ASP.NET Core Module topic.

Compare error information to the following common errors. If a match is found, follow the troubleshooting advice.

The list of errors in this topic isn't exhaustive. If you encounter an error not listed here, open a new issue using the **Content feedback** button at the bottom of this topic with detailed instructions on how to reproduce the error.

> ⓘ **Important**
>
> **ASP.NET Core preview releases with Azure App Service**
>
> ASP.NET Core preview releases aren't deployed to Azure App Service by default. To host an app that uses an ASP.NET Core preview release, see Deploy ASP.NET Core preview release to Azure App Service.

OS upgrade removed the 32-bit ASP.NET Core Module

Application Log: The Module DLL **C:\WINDOWS\system32\inetsrv\aspnetcore.dll** failed to load. The data is the error.

Troubleshooting:

Non-OS files in the **C:\Windows\SysWOW64\inetsrv** directory aren't preserved during an OS upgrade. If the ASP.NET Core Module is installed prior to an OS upgrade and then any app pool is run in 32-bit mode after an OS upgrade, this issue is encountered. After an OS upgrade, repair the ASP.NET Core Module. See Install the .NET Core Hosting bundle. Select **Repair** when the installer is run.

Missing site extension, 32-bit (x86) and 64-bit (x64) site extensions installed, or wrong process bitness set

Applies to apps hosted by Azure App Services.

- **Browser:** HTTP Error 500.0 - ANCM In-Process Handler Load Failure
- **Application Log:** Invoking hostfxr to find the inprocess request handler failed without finding any native dependencies. Could not find inprocess request handler. Captured output from invoking hostfxr: It was not possible to find any compatible framework version. The specified framework 'Microsoft.AspNetCore.App', version '{VERSION}-preview-*' was not found. Failed to start application '/LM/W3SVC/1416782824/ROOT', ErrorCode '0x8000ffff'.

- **ASP.NET Core Module stdout Log:** It was not possible to find any compatible framework version. The specified framework 'Microsoft.AspNetCore.App', version '{VERSION}-preview-*' was not found.
- **ASP.NET Core Module Debug Log:** Invoking hostfxr to find the inprocess request handler failed without finding any native dependencies. This most likely means the app is misconfigured, please check the versions of Microsoft.NetCore.App and Microsoft.AspNetCore.App that are targeted by the application and are installed on the machine. Failed HRESULT returned: 0x8000ffff. Could not find inprocess request handler. It was not possible to find any compatible framework version. The specified framework 'Microsoft.AspNetCore.App', version '{VERSION}-preview-*' was not found.

Troubleshooting:

- If running the app on a preview runtime, install either the 32-bit (x86) **or** 64-bit (x64) site extension that matches the bitness of the app and the app's runtime version. **Don't install both extensions or multiple runtime versions of the extension.**
  - ASP.NET Core {RUNTIME VERSION} (x86) Runtime
  - ASP.NET Core {RUNTIME VERSION} (x64) Runtime

  Restart the app. Wait several seconds for the app to restart.

- If running the app on a preview runtime and both the 32-bit (x86) and 64-bit (x64) site extensions are installed, uninstall the site extension that doesn't match the bitness of the app. After removing the site extension, restart the app. Wait several seconds for the app to restart.
- If running the app on a preview runtime and the site extension's bitness matches that of the app, confirm that the preview site extension's *runtime version* matches the app's runtime version.
- Confirm that the app's **Platform** in **Application Settings** matches the bitness of the app.

For more information, see Deploy ASP.NET Core apps to Azure App Service.

An x86 app is deployed but the app pool isn't enabled for 32-bit apps

- **Browser:** HTTP Error 500.30 - ANCM In-Process Start Failure

- **Application Log:** Application '/LM/W3SVC/5/ROOT' with physical root '{PATH}' hit unexpected managed exception, exception code = '0xe0434352'. Please check the stderr logs for more information. Application '/LM/W3SVC/5/ROOT' with physical root '{PATH}' failed to load clr and managed application. CLR worker thread exited prematurely
- **ASP.NET Core Module stdout Log:** The log file is created but empty.
- **ASP.NET Core Module Debug Log:** Failed HRESULT returned: 0x8007023e

This scenario is trapped by the SDK when publishing a self-contained app. The SDK produces an error if the RID doesn't match the platform target (for example, `win10-x64` RID with `<PlatformTarget>x86</PlatformTarget>` in the project file).

Troubleshooting:

For an x86 framework-dependent deployment (`<PlatformTarget>x86</PlatformTarget>`), enable the IIS app pool for 32-bit apps. In IIS Manager, open the app pool's **Advanced Settings** and set **Enable 32-Bit Applications** to **True**.

Platform conflicts with RID

- **Browser:** HTTP Error 502.5 - Process Failure
- **Application Log:** Application 'MACHINE/WEBROOT/APPHOST/{ASSEMBLY}' with physical root 'C:{PATH}' failed to start process with commandline '"C:{PATH}{ASSEMBLY}.{exe|dll}" ', ErrorCode = '0x80004005 : ff.
- **ASP.NET Core Module stdout Log:** Unhandled Exception: System.BadImageFormatException: Could not load file or assembly '{ASSEMBLY}.dll'. An attempt was made to load a program with an incorrect format.

Troubleshooting:

- Confirm that the app runs locally on Kestrel. A process failure might be the result of a problem within the app. For more information, see Troubleshoot ASP.NET Core on Azure App Service and IIS.
- If this exception occurs for an Azure Apps deployment when upgrading an app and deploying newer assemblies, manually delete all files from the prior deployment. Lingering incompatible assemblies can result in a `System.BadImageFormatException` exception when deploying an upgraded app.

URI endpoint wrong or stopped website

- **Browser:** ERR_CONNECTION_REFUSED **--OR--** Unable to connect
- **Application Log:** No entry
- **ASP.NET Core Module stdout Log:** The log file isn't created.
- **ASP.NET Core Module Debug Log:** The log file isn't created.

Troubleshooting:

- Confirm the correct URI endpoint for the app is in use. Check the bindings.
- Confirm that the IIS website isn't in the *Stopped* state.

CoreWebEngine or W3SVC server features disabled

OS Exception: The IIS 7.0 CoreWebEngine and W3SVC features must be installed to use the ASP.NET Core Module.

Troubleshooting:

Confirm that the proper role and features are enabled. See IIS Configuration.

Incorrect website physical path or app missing

- **Browser:** 403 Forbidden - Access is denied **--OR--** 403.14 Forbidden - The Web server is configured to not list the contents of this directory.
- **Application Log:** No entry
- **ASP.NET Core Module stdout Log:** The log file isn't created.
- **ASP.NET Core Module Debug Log:** The log file isn't created.

Troubleshooting:

Check the IIS website **Basic Settings** and the physical app folder. Confirm that the app is in the folder at the IIS website **Physical path**.

Incorrect role, ASP.NET Core Module not installed, or incorrect permissions

- **Browser:** 500.19 Internal Server Error - The requested page cannot be accessed because the related configuration data for the page is invalid. **--OR--** This page can't be displayed
- **Application Log:** No entry
- **ASP.NET Core Module stdout Log:** The log file isn't created.
- **ASP.NET Core Module Debug Log:** The log file isn't created.

Troubleshooting:

- Confirm that the proper role is enabled. See IIS Configuration.
- Open **Programs & Features** or **Apps & features** and confirm that **Windows Server Hosting** is installed. If **Windows Server Hosting** isn't present in the list of installed programs, download and install the .NET Core Hosting Bundle.

  Current .NET Core Hosting Bundle installer (direct download)

  For more information, see Install the .NET Core Hosting Bundle.
- Make sure that the **Application Pool** > **Process Model** > **Identity** is set to **ApplicationPoolIdentity** or the custom identity has the correct permissions to access the app's deployment folder.
- If you uninstalled the ASP.NET Core Hosting Bundle and installed an earlier version of the hosting bundle, the *applicationHost.config* file doesn't include a section for the ASP.NET Core Module. Open *applicationHost.config* at *%windir%/System32/inetsrv/config* and find the `<configuration><configSections><sectionGroup name="system.webServer">` section group. If the section for the ASP.NET Core Module is missing from the section group, add the section element:

  XML

  ```xml
  <section name="aspNetCore" overrideModeDefault="Allow" />
  ```

  Alternatively, install the latest version of the ASP.NET Core Hosting Bundle. The latest version is backwards-compatible with supported ASP.NET Core apps.

Incorrect processPath, missing PATH variable, Hosting Bundle not installed, system/IIS not restarted, VC++ Redistributable not installed, or dotnet.exe access violation

- **Browser:** HTTP Error 500.0 - ANCM In-Process Handler Load Failure
- **Application Log:** Application 'MACHINE/WEBROOT/APPHOST/{ASSEMBLY}' with physical root 'C:{PATH}' failed to start process with commandline '"{...}" ', ErrorCode = '0x80070002 : 0. Application '{PATH}' wasn't able to start. Executable was not found at '{PATH}'. Failed to start application '/LM/W3SVC/2/ROOT', ErrorCode '0x8007023e'.
- **ASP.NET Core Module stdout Log:** The log file isn't created.
- **ASP.NET Core Module Debug Log:** Event Log: 'Application '{PATH}' wasn't able to start. Executable was not found at '{PATH}'. Failed HRESULT returned: 0x8007023e

Troubleshooting:

- Confirm that the app runs locally on Kestrel. A process failure might be the result of a problem within the app. For more information, see Troubleshoot ASP.NET Core on Azure App Service and IIS.
- Check the *processPath* attribute on the `<aspNetCore>` element in *web.config* to confirm that it's `dotnet` for a framework-dependent deployment (FDD) or `.\{ASSEMBLY}.exe` for a self-contained deployment (SCD).
- For an FDD, *dotnet.exe* might not be accessible via the PATH settings. Confirm that *C:\Program Files\dotnet\* exists in the System PATH settings.
- For an FDD, *dotnet.exe* might not be accessible for the user identity of the app pool. Confirm that the app pool user identity has access to the *C:\Program Files\dotnet* directory. Confirm that there are no deny rules configured for the app pool user identity on the *C:\Program Files\dotnet* and app directories.
- An FDD may have been deployed and .NET Core installed without restarting IIS. Either restart the server or restart IIS by executing **net stop was /y** followed by **net start w3svc** from a command prompt.
- An FDD may have been deployed without installing the .NET Core runtime on the hosting system. If the .NET Core runtime hasn't been installed, run the **.NET Core Hosting Bundle installer** on the system.

Current .NET Core Hosting Bundle installer (direct download)

For more information, see Install the .NET Core Hosting Bundle.

If a specific runtime is required, download the runtime from the .NET Downloads page and install it on the system. Complete the installation by restarting the system or restarting IIS by executing **net stop was /y** followed by **net start w3svc** from a command prompt.

Incorrect arguments of <aspNetCore> element

- **Browser:** HTTP Error 500.0 - ANCM In-Process Handler Load Failure
- **Application Log:** Invoking hostfxr to find the inprocess request handler failed without finding any native dependencies. This most likely means the app is misconfigured, please check the versions of Microsoft.NetCore.App and Microsoft.AspNetCore.App that are targeted by the application and are installed on the machine. Could not find inprocess request handler. Captured output from invoking hostfxr: Did you mean to run dotnet SDK commands? Please install dotnet SDK from: https://go.microsoft.com/fwlink/?LinkID=798306&clcid=0x409 Failed to start application '/LM/W3SVC/3/ROOT', ErrorCode '0x8000ffff'.
- **ASP.NET Core Module stdout Log:** Did you mean to run dotnet SDK commands? Please install dotnet SDK from: https://go.microsoft.com/fwlink/?LinkID=798306&clcid=0x409
- **ASP.NET Core Module Debug Log:** Invoking hostfxr to find the inprocess request handler failed without finding any native dependencies. This most likely means the app is misconfigured, please check the versions of Microsoft.NetCore.App and Microsoft.AspNetCore.App that are targeted by the application and are installed on the machine. Failed HRESULT returned: 0x8000ffff Could not find inprocess request handler. Captured output from invoking hostfxr: Did you mean to run dotnet SDK commands? Please install dotnet SDK from: https://go.microsoft.com/fwlink/?LinkID=798306&clcid=0x409 Failed HRESULT returned: 0x8000ffff

Troubleshooting:

- Confirm that the app runs locally on Kestrel. A process failure might be the result of a problem within the app. For more information, see Troubleshoot ASP.NET Core on Azure App Service and IIS.

- Examine the *arguments* attribute on the `<aspNetCore>` element in *web.config* to confirm that it's either (a) `.\{ASSEMBLY}.dll` for a framework-dependent deployment (FDD); or (b) not present, an empty string (`arguments=""`), or a list of the app's arguments (`arguments="{ARGUMENT_1}, {ARGUMENT_2}, ... {ARGUMENT_X}"`) for a self-contained deployment (SCD).

Missing .NET Core shared framework

- **Browser:** HTTP Error 500.0 - ANCM In-Process Handler Load Failure

- **Application Log:** Invoking hostfxr to find the inprocess request handler failed without finding any native dependencies. This most likely means the app is misconfigured, please check the versions of Microsoft.NetCore.App and Microsoft.AspNetCore.App that are targeted by the application and are installed on the machine. Could not find inprocess request handler. Captured output from invoking hostfxr: It was not possible to find any compatible framework version. The specified framework 'Microsoft.AspNetCore.App', version '{VERSION}' was not found.

Failed to start application '/LM/W3SVC/5/ROOT', ErrorCode '0x8000ffff'.

- **ASP.NET Core Module stdout Log:** It was not possible to find any compatible framework version. The specified framework 'Microsoft.AspNetCore.App', version '{VERSION}' was not found.

- **ASP.NET Core Module Debug Log:** Failed HRESULT returned: 0x8000ffff

Troubleshooting:

For a framework-dependent deployment (FDD), confirm that the correct runtime installed on the system.

Stopped Application Pool

- **Browser:** 503 Service Unavailable

- **Application Log:** No entry

- **ASP.NET Core Module stdout Log:** The log file isn't created.

- **ASP.NET Core Module Debug Log:** The log file isn't created.

Troubleshooting:

Confirm that the Application Pool isn't in the *Stopped* state.

Sub-application includes a <handlers> section

- **Browser:** HTTP Error 500.19 - Internal Server Error
- **Application Log:** No entry
- **ASP.NET Core Module stdout Log:** The root app's log file is created and shows normal operation. The sub-app's log file isn't created.
- **ASP.NET Core Module Debug Log:** The root app's log file is created and shows normal operation. The sub-app's log file isn't created.

Troubleshooting:

Confirm that the sub-app's *web.config* file doesn't include a `<handlers>` section or that the sub-app doesn't inherit the parent app's handlers.

The parent app's `<system.webServer>` section of *web.config* is placed inside of a `<location>` element. The InheritInChildApplications property is set to `false` to indicate that the settings specified within the <location> element aren't inherited by apps that reside in a subdirectory of the parent app. For more information, see ASP.NET Core Module (ANCM) for IIS.

stdout log path incorrect

- **Browser:** The app responds normally.
- **Application Log:** Could not start stdout redirection in C:\Program Files\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll. Exception message: HRESULT 0x80070005 returned at {PATH}\aspnetcoremodulev2\commonlib\fileoutputmanager.cpp:84. Could not stop stdout redirection in C:\Program Files\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll. Exception message: HRESULT 0x80070002 returned at {PATH}. Could not start stdout redirection in {PATH}\aspnetcorev2_inprocess.dll.
- **ASP.NET Core Module stdout Log:** The log file isn't created.
- **ASP.NET Core Module debug Log:** Could not start stdout redirection in C:\Program Files\IIS\Asp.Net Core Module\V2\aspnetcorev2.dll. Exception message: HRESULT 0x80070005 returned at {PATH}\aspnetcoremodulev2\commonlib\fileoutputmanager.cpp:84. Could not stop stdout redirection in C:\Program Files\IIS\Asp.Net Core

Module\V2\aspnetcorev2.dll. Exception message: HRESULT 0x80070002 returned at {PATH}. Could not start stdout redirection in {PATH}\aspnetcorev2_inprocess.dll.

Troubleshooting:

- The `stdoutLogFile` path specified in the `<aspNetCore>` element of *web.config* doesn't exist. For more information, see ASP.NET Core Module: Log creation and redirection.
- The app pool user doesn't have write access to the stdout log path.

Application configuration general issue

- **Browser:** HTTP Error 500.0 - ANCM In-Process Handler Load Failure **--OR--** HTTP Error 500.30 - ANCM In-Process Start Failure
- **Application Log:** Variable
- **ASP.NET Core Module stdout Log:** The log file is created but empty or created with normal entries until the point of the app failing.
- **ASP.NET Core Module Debug Log:** Variable

Troubleshooting:

The process failed to start, most likely due to an app configuration or programming issue.

For more information, see the following topics:

- Troubleshoot ASP.NET Core on Azure App Service and IIS
- Troubleshoot and debug ASP.NET Core projects

Code analysis in ASP.NET Core apps

Article • 07/09/2024

.NET compiler platform analyzers inspect application code for code quality and style issues. This document lists diagnostics for ASP.NET Core. For information on .NET diagnostics, see Overview of .NET source code analysis.

The following list contains the diagnostics for ASP.NET Core. Not all of the diagnostics shown are available in older versions of ASP.NET Core.

Diagnostic ID:

- ASP0000
- ASP0001
- ASP0003
- ASP0004
- ASP0005
- ASP0006
- ASP0007
- ASP0008
- ASP0009
- ASP0010
- ASP0011
- ASP0012
- ASP0013
- ASP0014
- ASP0015
- ASP0016
- ASP0017
- ASP0018
- ASP0019
- ASP0020
- ASP0021
- ASP0022
- ASP0023
- ASP0024
- ASP0025
- ASP0026
- BL0001
- BL0002
- BL0003

- BL0004
- BL0005
- BL0006
- MVC1000
- MVC1001
- MVC1002
- MVC1003
- MVC1004
- MVC1005
- MVC1006

ASP0000: Do not call 'IServiceCollection.BuildServiceProvider' in 'ConfigureServices'

Article • 06/18/2024

Expand table

	Value
Rule ID	ASP0000
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A call to BuildServiceProvider was detected in the application start up code.

Rule description

Calling 'BuildServiceProvider' from application code results in more than one copy of singleton services being created which might result in incorrect application behavior. Consider alternatives such as dependency injecting services as parameters to 'Configure'.

How to fix violations

Remove the call to `BuildServiceProvider` from the application startup code.

When to suppress warnings

It is safe to suppress this rule if updating the application to remove the call to `BuildServiceProvider` is non-trivial and you have thoroughly tested the application to ensure multiple singleton services are not added.

ASP0001: Authorization middleware is incorrectly configured

Article • 06/18/2024

Expand table

	Value
Rule ID	ASP0001
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An out of order call to UseAuthorization was detected in the application start up code.

Rule description

For authorization to be effective for endpoint routes, the call to `UseAuthorization` should appear between the calls to `UseRouting` and `UseEndpoints`. In the absence of this, the fallback policy, if configured, will be used to authorize all requests.

Consider the following code:

```C#
app.UseStaticFiles();
app.UseAuthorization();

app.UseRouting();
app.UseEndpoints(endpoints =>
{
    endpoints.MapDefaultControllerRoute();
});
```

The call to `UseAuthorization` appears before `UseRouting` and consequently is not endpoint aware.

How to fix violations

Change the order in which the call to `UseAuthorization` and `UseRouting` are performed.

```C#
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>
{
    endpoints.MapDefaultControllerRoute();
});
```

When to suppress warnings

It is safe to suppress this rule if the call to `UseAuthorization` is intended to authorize the fallback policy on all outgoing requests, or is meant to authorize resources not routed using endpoint routing.

ASP0003: Do not use model binding attributes with route handlers

Article • 09/28/2022

Expand table

	Value
Rule ID	ASP0003
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A route handler delegate includes a parameter that has a [Bind] attribute.

Rule description

Route handler endpoints only support a subset of MVC attributes such as [FromRoute], [FromBody], etc. Unsupported attributes are ignored and result in unexpected binding behavior. For example, the following code results in an analyzer warning:

C#

```csharp
app.MapGet("/todos/{id}", ([Bind] int id) => new Todo { Id = id });
```

How to fix violations

To fix a violation of this rule, make sure that the endpoint uses one of the allowed model binding attributes.

C#

```csharp
app.MapGet("/todos/{id}", ([FromRoute] int id) => new Todo { Id = id });
```

When to suppress warnings

Do ***not*** suppress a warning from this rule. An incorrect model binding setup can result in unexpected behavior when resolving parameters at runtime.

ASP0004: Do not use action results with route handlers

Article • 09/28/2022

Expand table

	Value
Rule ID	ASP0004
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A route handler delegate returns a value that implements IActionResult.

Rule description

Route handler endpoints do not support executing MVC's `IActionResult` instances. Returning an `IActionResult` that doesn't implement `IResult` results in serializing the result instance rather than executing the result.

```C#
app.MapGet("/todos/{id}", (int id) => new JsonResult(new Todo { .. }));
```

How to fix violations

To fix a violation of this rule, make sure that endpoint's route handler returns an IResult type by using the Results extension methods.

```C#
app.MapGet("/todos/{id}", (int id) => Results.Json(new Todo { .. }));
```

When to suppress warnings

Do ***not*** suppress a warning from this rule. Returning an `IActionResult` that doesn't implement `IResult` results in serializing the result instance rather than executing the result.

ASP0005: Do not place attribute on method called by route handler lambda

Article • 09/28/2022

Expand table

	Value
Rule ID	ASP0005
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An attribute was applied to a method definition instead of the route handler in a route handler endpoint.

Rule description

When an endpoint is declared, attributes should be applied to the delegate parameter in order to be effective. For example, the Authorize attribute in the following code sample isn't set on the registered endpoint:

C#

```
app.MapGet("/todos/{id}", GetTodoById);

[Authorize]
Todo GetTodoById(int id)
{
  ...
}
```

The attribute must be placed on the route handler parameter as shown in the following code:

C#

```
app.MapGet("/todos/{id}", [Authorize] GetTodoById);
```

```
Todo GetTodoById(int id)
{
  ...
}
```

How to fix violations

To fix a violation of this rule, make sure that endpoint attributes are applied to the route handler parameter:

```C#
app.MapGet("/todos/{id}", [Authorize] (int id) => {});
app.MapGet("/users/{id}", [Authorize] GetUserById);
```

When to suppress warnings

Do not suppress a warning from this rule. Misplaced attributes can result in unexpected behavior at runtime.

ASP0006: Do not use non-literal sequence numbers

Article • 10/30/2022

Expand table

	Value
Rule ID	ASP0006
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An invocation on a method in RenderTreeBuilder containing a sequence number that isn't a literal as a parameter.

Rule description

Blazor's UI diffing algorithm relies on sequence numbers to determine which elements have changed. Computing the sequence number dynamically or using a counter can result in poor diffing performance. Instead, use a literal sequence number that maps to the source code line for the element. For example, the following code produces an error:

C#

```csharp
using Microsoft.AspNetCore.Components.Rendering;
var builder = new RenderTreeBuilder();
var seqNum = 1;
builder.OpenElement(seqNum, "div");
builder.CloseElement();
```

How to fix violations

To fix a violation of this rule, make sure that calls to methods on the RenderTreeBuilder class that take a sequence number as a parameter use a literal sequence number.

C#

```
using Microsoft.AspNetCore.Components.Rendering;
var builder = new RenderTreeBuilder();
builder.OpenElement(0, "div");
builder.CloseElement();
```

When to suppress warnings

Do ***not*** suppress a warning from this rule. Using a non-literal sequence number can result in performance degradation.

ASP0007: Route parameter and argument optionality is mismatched

Article • 09/28/2022

Expand table

	Value
Rule ID	ASP0007
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A route parameter is declared as required in the Delegate definition but is marked as optional in the endpoint route.

Rule description

When an endpoint is declared, optionality of parameters can be declared in both the route template and in the route handler arguments. When a parameter is declared as optional in the handler, it must also be declared as optional in the route template. For example, GET `/todos` fails to resolve a match for the following code:

C#

```csharp
app.MapGet("/todos/{id}", (int? id) => {});
```

The preceding code fails to match GET `/todos` because the `id` parameter was not provided, even though it is treated as optional in the route handler.

How to fix violations

To fix a violation of this rule, make sure that the optionality in the route template and the delegate match. For example, for the following code sample:

C#

```
app.MapGet("/todos/{id}", (int? id) => {});
```

If the parameter is intended to be required, make the type non-nullable by removing the `?` from `int?`:

C#

```
app.MapGet("/todos/{id}", (int id) => {});
```

If the parameter is intended to be optional, then the nullable value type operator `?` should be applied:

C#

```
app.MapGet("/todos/{id?}", (int? id) => {});
```

When to suppress warnings

Do ***not*** suppress a warning from this rule. Mismatched parameter optionality can result in unexpected behavior with routing at runtime.

ASP0008: Do not use ConfigureWebHost with WebApplicationBuilder.Host

Article • 09/28/2022

Expand table

	Value
Rule ID	ASP0008
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

`ConfigureWebHost` can't be used with the `Host` property on `WebApplicationBuilder`.

Rule description

The `WebApplicationBuilder` doesn't support configuring the `WebHost` before build using the `ConfigureWebHost` extension method.

```C#
var builder = WebApplication.CreateBuilder(args);

builder.Host.ConfigureWebHost(webHostBuilder => {

webHostBuilder.UseContentRoot(Path.Combine(Directory.GetCurrentDirectory(),
"myContentRoot"));
});

var app = builder.Build();

app.Run();
```

How to fix violations

To fix a violation of this rule, configure the `WebHost` directly on the `WebApplicationBuilder`. For example, instead of setting the content root path via `ConfigureWebHost`.

```C#
var builder = WebApplication.CreateBuilder(args);

builder.Host.ConfigureWebHost(webHostBuilder =>
{

webHostBuilder.UseContentRoot(Path.Combine(Directory.GetCurrentDirectory(),
"myContentRoot"));
});

var app = builder.Build();

app.Run();
```

Configure the content root path directly on the `WebApplicationBuilder.WebHost`.

```C#
var builder = WebApplication.CreateBuilder(args);

builder.WebHost.UseContentRoot(Path.Combine(Directory.GetCurrentDirectory(),
"foobar"));

var app = builder.Build();

app.Run();
```

When to suppress warnings

Do ***not*** suppress a warning from this rule. A misconfigured application can result in unexpected behavior at runtime.

ASP0009: Do not use Configure with WebApplicationBuilder.WebHost

Article • 09/28/2022

	Value
Rule ID	ASP0009
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

`Configure` can't be used with the `WebHost` property on `WebApplicationBuilder`.

Rule description

The `WebApplicationBuilder` doesn't support configuring the `WebHost` before build using the `Configure` extension method.

```C#
var builder = WebApplication.CreateBuilder(args);

builder.WebHost.Configure(webHostBuilder => {

webHostBuilder.UseContentRootPath(Path.Combine(Directory.GetCurrentDirectory
(), "myContentRoot"));
});

var app = builder.Build();

app.Run();
```

How to fix violations

To fix a violation of this rule, configure the `WebHost` directly on the `WebApplicationBuilder`. For example, instead of setting the content root path via

`Configure`.

```C#
var builder = WebApplication.CreateBuilder(args);

builder.WebHost.Configure(webHostBuilder =>
{

webHostBuilder.UseContentRoot(Path.Combine(Directory.GetCurrentDirectory(),
"myContentRoot"));
});

var app = builder.Build();

app.Run();
```

Configure the content root path directly on the `WebApplicationBuilder`.

```C#
var builder = WebApplication.CreateBuilder(args);

builder.WebHost.UseContentRoot(Path.Combine(Directory.GetCurrentDirectory(),
"myContentRoot"));

var app = builder.Build();

app.Run();
```

When to suppress warnings

Do ***not*** suppress a warning from this rule. A misconfigured application can result in unexpected behavior at runtime.

ASP0010: Do not use UseStartup with WebApplicationBuilder.WebHost

Article • 06/18/2024

Expand table

	Value
Rule ID	ASP0010
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

`UseStartup` can't be used with `WebApplicationBuilder.WebHost`.

Rule description

The `WebApplicationBuilder` doesn't support configuration via a `Startup` class.

C#

```csharp
var builder = WebApplication.CreateBuilder(args);

builder.WebHost.UseStartup<Startup>();

var app = builder.Build();

app.Run();
```

How to fix violations

To fix a violation of this rule, leverage the `Configuration` and `Services` properties on the `WebApplicationBuilder` to modify configuration and DI directly, without the need for a startup class.

C#

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddAuthentication();

var app = builder.Build();

app.Run();
```

When to suppress warnings

Do ***not*** suppress a warning from this rule. A misconfigured application can result in unexpected behavior at runtime.

ASP0011: Suggest using builder.Logging over Host.ConfigureLogging or WebHost.ConfigureLogging

Article • 09/28/2022

Expand table

	Value
Rule ID	ASP0011
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

`ConfigureLogging` isn't the recommended strategy for configuring logging in a minimal API application.

Rule description

`ConfigureLogging` isn't the recommended strategy for configuring logging in a minimal API application.

C#

```csharp
var builder = WebApplication.CreateBuilder(args);

builder.Host.ConfigureLogging(logging =>
{
    logging.AddJsonConsole();
})

var app = builder.Build();

app.Run();
```

How to fix violations

To fix a violation of this rule, use the `Logging` property on the `WebApplicationBuilder` to modify the logging configuration directly without the need for an additional `ConfigureLogging` call.

```csharp
var builder = WebApplication.CreateBuilder(args);

builder.Logging.AddJsonConsole();

var app = builder.Build();

app.Run();
```

When to suppress warnings

Do ***not*** suppress a warning from this rule.

ASP0012: Suggest using builder.Services over Host.ConfigureServices or WebHost.ConfigureServices

Article • 09/28/2022

Expand table

	Value
Rule ID	ASP0012
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

`ConfigureServices` isn't the recommended strategy for registering services in DI in a minimal API application.

Rule description

`ConfigureServices` isn't the recommended strategy for configuring logging in a minimal API application.

C#

```csharp
var builder = WebApplication.CreateBuilder(args);

builder.Host.ConfigureServices(services =>
{
    services.AddAntiforgery();
})

var app = builder.Build();

app.Run();
```

How to fix violations

To fix a violation of this rule, use the `Services` property on the `WebApplicationBuilder` to modify the DI container directly without the need for an additional `ConfigureServices` call.

```csharp
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddAntiforgery();

var app = builder.Build();

app.Run();
```

When to suppress warnings

Do ***not*** suppress a warning from this rule.

ASP0013: Suggest switching from using Configure methods to WebApplicationBuilder.Configuration

Article • 02/13/2024

Expand table

	Value
Rule ID	ASP0013
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

Configure isn't the recommended strategy for reading and writing to configuration in a minimal API app. `Configure` was designed to be used with Web Host or .NET Generic Host. In a minimal API app, WebApplicationBuilder.Configuration should be used to modify configuration directly.

Rule description

`Configure` isn't the recommended strategy for configuring logging in a minimal API app.

```C#
var builder = WebApplication.CreateBuilder(args);

builder.Host.ConfigureAppConfiguration(builder =>
{
    builder.AddJsonFile("customAppSettings.json");
})

var app = builder.Build();

app.Run();
```

How to fix violations

To fix a violation of this rule, use WebApplicationBuilder.Configuration to modify application configuration directly without the need for an additional ConfigureAppConfiguration call.

```csharp
var builder = WebApplication.CreateBuilder(args);

builder.Configuration.AddJsonFile("customAppSettings.json");

var app = builder.Build();

app.Run();
```

When to suppress warnings

Do ***not*** suppress a warning from this rule.

ASP0014: Suggest using top level route registrations

Article • 02/23/2023

Expand table

	Value
Rule ID	ASP0014
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

Routes can be registered directly at the top-level of a minimal API application.

Rule description

Routes can be registered directly at the top-level of a minimal API application and don't need to be nested within a `UseEndpoints` call.

```C#
var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.UseEndpoints(endpoints =>
{
    endpoints.MapGet("/", () => "Hello World!");
});

app.Run();
```

How to fix violations

To fix a violation of this rule, register the endpoints directly on the `WebApplication`.

C#

```
var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();
```

When to suppress warnings

Warnings from this rule can be suppressed if the target `UseEndpoints` invocation is invoked without any mappings as a strategy to organize middleware ordering.

```C#
var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

#pragma warning disable ASP0014
app.UseEndpoints(e => {});
#pragma warning restore ASP0014

app.Run();
```

ASP0015: Suggest using IHeaderDictionary properties

Article • 11/28/2022

Expand table

	Value
Rule ID	ASP0015
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

IHeaderDictionary properties are the recommended strategy for accessing headers.

Rule description

`IHeaderDictionary` properties are recommended for accessing headers. Accessing headers using an indexer as in the example below is not recommended.

```csharp
var app = WebApplication.Create();

app.MapGet("/", (HttpContext context) => context.Request.Headers[""content-
type""]);

app.Run();
```

How to fix violations

To fix a violation of this rule, use the property specified in the analyzer message to access the header specified in the message or apply the associated codefix.

```csharp
var app = WebApplication.Create();
app.MapGet("/", (HttpContext context) =>
context.Request.Headers.ContentType);
app.Run();
```

When to suppress warnings

Do not suppress a warning from this rule.

ASP0016: Do not return a value from RequestDelegate

Article • 11/28/2022

Expand table

	Value
Rule ID	ASP0016
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A method used to create a RequestDelegate returns `Task<T>`. `RequestDelegate` discards this value.

Rule description

Do not return a value `Delegate`s provided to APIs that expect `RequestDelegate`. For example, the following sample returns a `Task<string>` where the `string` value of the `Task` will be discarded.

```C#
var app = WebApplication.Create();
app.Use(next =>
{
    return new RequestDelegate((HttpContext context) =>
    {
        return Task.FromResult(""hello world"");
    });
});
```

How to fix violations

To fix a violation of this rule, change the return type to non-generic `Task` or, if the delegate is a route handler, cast it to `Delegate` so the return value is written to the response.

When to suppress warnings

Do not suppress a warning from this rule.

ASP0017: Invalid route pattern

Article • 11/28/2022

Expand table

	Value
Rule ID	ASP0017
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A route pattern is invalid.

Rule description

This diagnostic is emitted when a route pattern is invalid. In the example below, the route pattern contains an invalid token.

```C#
var app = WebApplication.Create();

app.MapGet("/{id", (int id) => ...);
```

How to fix violations

To fix a violation of this rule, correct the error identified in the analyzer message.

When to suppress warnings

Do not suppress a warning from this rule.

ASP0018: Unused route parameter

Article • 06/20/2024

Expand table

	Value
Rule ID	ASP0018
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A route parameter is specified but not used.

Rule description

A route parameter is specified but not used. In the example below, the `id` parameter is defined in the route but not in the route handler.

```C#
var app = WebApplication.Create();

app.MapGet("/{id}", () => ...);
```

How to fix violations

To fix a violation of this rule, remove the route parameter or add code that uses the parameter.

```C#
var app = WebApplication.Create();

app.MapGet("/{id}", (string id) => ...);
```

When to suppress warnings

In general, do ***not*** suppress a warning from this rule without validating the route parameter is used. Currently properties within the bound model for FromRoute attributes may not be analyzed. For more information, see GitHub issue #54212.

ASP0019: Suggest using IHeaderDictionary.Append or the indexer

Article • 11/28/2022

Expand table

	Value
Rule ID	ASP0019
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

`IDictionary.Add` isn't recommended for setting or appending headers. IDictionary.Add throws an `ArgumentException` when attempting to add a duplicate key.

Rule description

`IDictionary.Add` isn't recommended for setting or appending headers.

How to fix violations

To fix a violation of this rule, use `IHeaderDictionary.Append` or the indexer to append or set headers.

When to suppress warnings

Do not suppress a warning from this rule.

ASP0020: Complex types referenced by route parameters must be parsable

Article • 03/27/2023

Expand table

	Value
Rule ID	ASP0020
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A route parameter is a complex type that isn't parsable.

Rule description

This diagnostic is emitted when a route parameter is a complex type that isn't parsable.

How to fix violations

To fix a violation of this rule, define a `bool TryParse(string, IFormatProvider, out T)` method, where `T` is the complex type identified in the error message. As an alternative, implement IParsable<TSelf>.

When to suppress warnings

Do not suppress a warning from this rule.

ASP0021: The return type of the BindAsync method must be `ValueTask<T>`.

Article • 03/27/2023

Expand table

	Value
Rule ID	ASP0021
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An implementation of the BindAsync method has a return type that isn't ValueTask<TResult>.

Rule description

This diagnostic is emitted when an implementation of the `BindAsync` method has a return type that isn't `ValueTask<T>`.

How to fix violations

To fix a violation of this rule, define a `ValueTask<T>` return type for `BindAsync` and consider implementing IBindableFromHttpContext<TSelf> to enforce implementation.

When to suppress warnings

Do not suppress a warning from this rule.

ASP0022: Route conflict detected between route handlers

Article • 11/08/2023

Expand table

	Value
Rule ID	ASP0022
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An HTTP request matches multiple routes, resulting in an ambiguous match error.

Rule description

This diagnostic is emitted when a route conflict is detected.

How to fix violations

To fix a violation of this rule, change the route's pattern, HTTP method, or route constraints.

When to suppress warnings

Do not suppress a warning from this rule.

Notes

ASP0023 warns on route conflicts in ASP.NET Core MVC apps, this analyzer warns on route conflicts in minimal API apps.

This analyzer is intentionally conservative about duplicate routes it reports to avoid false positives:

- The analyzer only reports duplicate routes declared in the same code block in a method. Duplicate routes in different branches of an `if` statement aren't reported because the analyzer can't statically determine which will be used at runtime.
- The analyzer only reports duplicate routes with known metadata. Unknown methods called on a route handler might add new metadata that is then used to customize how the route is matched.

ASP0023: Route conflict detected between route handlers

Article • 03/29/2023

Expand table

	Value
Rule ID	ASP0023
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

An HTTP request matches multiple routes, resulting in an ambiguous match error.

Rule description

This diagnostic is emitted when a route conflict is detected.

How to fix violations

To fix a violation of this rule, change the route's pattern, HTTP method, or route constraints.

When to suppress warnings

Do not suppress a warning from this rule.

Notes

ASP0022 warns on route conflicts in minimal API apps, this analyzer warns on route conflicts in ASP.NET Core MVC apps.

ASP0024: Route handler has multiple parameters with the `[FromBody]` attribute

Article • 03/29/2023

Expand table

	Value
Rule ID	ASP0024
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A route handler has multiple parameters with the [FromBody] attribute or a parameter with an [AsParameters] attribute where the parameter type contains multiple members with `[FromBody]` attributes.

Rule description

Route handler has multiple parameters with the `[FromBody]` attribute.

How to fix violations

To fix a violation of this rule, remove `[FromBody]` attributes from all but one parameter.

When to suppress warnings

Do not suppress a warning from this rule.

ASP0025: Use AddAuthorizationBuilder to register authorization services and construct policies.

Article • 05/15/2023

Expand table

	Value
Rule ID	ASP0025
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

The use of AddAuthorization can be converted to the new AddAuthorizationBuilder.

Rule description

Use `AddAuthorizationBuilder` to register authorization services and construct policies.

How to fix violations

To fix a violation of this rule, replace the usage of `AddAuthorization` with `AddAuthorizationBuilder`.

The code fix converts any usage of the setters for the following properties of AuthorizationOptions:

- DefaultPolicy
- FallbackPolicy
- InvokeHandlersAfterFailure

These setter usages are converted to equivalent method calls on AuthorizationBuilder:

- SetDefaultPolicy
- SetFallbackPolicy
- SetInvokeHandlersAfterFailure

No diagnostic is reported when the configure action passed to `AddAuthorization` uses any of the following members of `AuthorizationOptions`:

- The GetPolicy(String) method
- The DefaultPolicy getter
- The FallbackPolicy getter
- The InvokeHandlersAfterFailure getter

`AuthorizationBuilder` doesn't have equivalents for these members of `AuthorizationOptions`, so they can't be converted.

No diagnostic is reported if the configure action passed to `AddAuthorization` contains operations unrelated to `AuthorizationOptions`. The code fix would not be able to automatically map unrelated operations to the fluent API of `AddAuthorizationBuilder`.

The following example shows code that triggers this diagnostic:

C#

```csharp
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("AtLeast21", policy =>
        policy.Requirements.Add(new MinimumAgeRequirement(21)));
});

var app = builder.Build();

app.UseAuthorization();

app.Run();
```

The following example shows the result of applying the code fix:

C#

```csharp
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddAuthorizationBuilder()
  .AddPolicy("AtLeast21", policy =>
  {
        policy.Requirements.Add(new MinimumAgeRequirement(21));
  });

var app = builder.Build();

app.UseAuthorization();
app.Run();
```

When to suppress warnings

The severity level of this diagnostic is Information. Suppress warnings if you don't want to use the new syntax.

ASP0026: `[Authorize]` is overridden by `[AllowAnonymous]` from "farther away"

Article • 07/09/2024

Expand table

	Value
Rule ID	ASP0026
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

It seems intuitive that an `[Authorize]` attribute placed "closer" to an MVC action than an `[AllowAnonymous]` attribute would override the `[AllowAnonymous]` attribute and force authorization. However, this is not necessarily the case. What does matter is the relative order of the attributes.

The following code shows examples where a closer `[Authorize]` attribute gets overridden by an `[AllowAnonymous]` attribute that is farther away.

C#

```csharp
[AllowAnonymous]
public class MyController
{
    [Authorize] // Overridden by the [AllowAnonymous] attribute on the class
    public IActionResult Private() => null;
}
```

C#

```csharp
[AllowAnonymous]
public class MyControllerAnon : ControllerBase
{
}

[Authorize] // Overridden by the [AllowAnonymous] attribute on
MyControllerAnon
public class MyControllerInherited : MyControllerAnon
{
}

public class MyControllerInherited2 : MyControllerAnon
{
    [Authorize] // Overridden by the [AllowAnonymous] attribute on
MyControllerAnon
    public IActionResult Private() => null;
}
```

C#

```
[AllowAnonymous]
[Authorize] // Overridden by the preceding [AllowAnonymous]
public class MyControllerMultiple : ControllerBase
{
}
```

Rule description

Warning that an `[Authorize]` attribute is overridden by an `[AllowAnonymous]` attribute from "farther away."

How to fix violations

The correct action to take if you see this warning depends on the intention behind the attributes. The farther away `[AllowAnonymous]` attribute should be removed if it's unintentionally exposing the endpoint to anonymous users. If the `[AllowAnonymous]` attribute was intended to override a closer `[Authorize]` attribute, you can repeat the `[AllowAnonymous]` attribute after the `[Authorize]` attribute to clarify the intent.

C#

```
[AllowAnonymous]
public class MyController
{
    // This produces no warning because the second, "closer"
[AllowAnonymous]
    // clarifies that [Authorize] is intentionally overridden.
    // Specifying AuthenticationSchemes can still be useful
    // for endpoints that allow but don't require authenticated users.
    [Authorize(AuthenticationSchemes = "Cookies")]
    [AllowAnonymous]
    public IActionResult Privacy() => null;
}
```

When to suppress warnings

The severity level of this diagnostic is Information. You can suppress warnings if your intention is to override the `[Authorize]` attribute. However, we recommend that you make the intent clear by repeating the `[AllowAnonymous]` attribute after the `[Authorize]` attribute.

ASP0028: Consider using `IPAddress.IPv6Any` instead of `IPAddress.Any`

Article • 11/18/2024

Expand table

	Value
Rule ID	ASP0028
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

On the server machine that supports `IPv6`, IPv6Any is preferred to Any because `Any` can be slower than `IPv6Any`. In some cases, `Any` may not work at all. `Any` can be slower due to the underlying System types implementation.

`127.0.0.1` is the IPv4 loopback address. `::1` is the IPv6 loopback address. `Any` is the wildcard address for IPv4. `IPv6Any` is the wildcard address for IPv6.

Current behavior with with IPv6 when using HTTP/1.x or HTTP/2.0:

- `localhost` resolves to `[::1]`.
- `[::1]` isn't accepted by the server, which forces a retry using `127.0.0.1`, creating a repeated cycle.

Using `Any` with the preceding conditions causes the `ASP0028` diagnostic message. Here's an example of the code that can result in these conditions:

```C#
.UseKestrel().ConfigureKestrel(options =>
{
    options.Listen(IPAddress.Any, ...);
})
```

Rule description

The recommended way to configure Kestrel to listen for incoming connections on all available `IPv6` network interfaces is with `IPv6Any`.

How to fix violations

For the problematic code, replace `Any` with `IPv6Any`.

Use the ListenAnyIP(Int32) method without specifying any arguments:

```diff
.UseKestrel().ConfigureKestrel(options =>
{
-   options.Listen(IPAddress.Any, ...);
+   options.ListenAnyIP(...);
})
```

Or use the IPv6Any field:

```diff
.UseKestrel().ConfigureKestrel(options =>
{
-   options.Listen(IPAddress.Any, ...);
+   options.Listen(IPAddress.IPv6Any, ...);
})
```

When to suppress warnings

The `ASP0028` diagnostic has Information level severity. Suppress this warning if your intention is to disable `IPv6` usage completely on the server, although doing so risks the performance problems mentioned in this article.

`IPv6` can be disabled either system-wide, or for .NET only via the AppCtx switch or environment variable

BL0001: Component parameter should have public setters

Article • 06/03/2022

Expand table

	Value
Rule ID	BL0001
Category	Usage
Fix is breaking or non-breaking	Breaking

Cause

A property on a type deriving from ComponentBase annotated with [Parameter] has a missing or non-public setters.

Rule description

Component parameters are required to have publicly accessible setters to allow the framework to assign values. All of the parameter declarations in the following example result in this diagnostic.

razor

```razor
@code
{
    [Parameter] int Parameter1 { get; set; }

    [Parameter] public int Parameter2 { get; }

    [Parameter] public int Parameter3 { get; private set; }
}
```

How to fix violations

- Make the property and its setter public.

razor

```
@code
{
    [Parameter] public int Parameter1 { get; set; }

    [Parameter] public int Parameter2 { get; set; }

    [Parameter] public int Parameter3 { get; set; }
}
```

- If making the property non-public is not possible, consider implementing SetParametersAsync manually.

When to suppress warnings

Do not suppress a warning from this rule.

BL0002: Component has multiple CaptureUnmatchedValues parameters

Article • 06/03/2022

Expand table

	Value
Rule ID	BL0002
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

More than one parameter on a type deriving from ComponentBase is annotated with `CaptureUnmatchedValues = true`.

Rule description

For a component, exactly one parameter is expected to have the CaptureUnmatchedValues set to `true`.

razor

```
@code
{
    [Parameter(CaptureUnmatchedValues = true)] public Dictionary<string,
object> Parameter1 { get; set; }

    [Parameter(CaptureUnmatchedValues = true)] public Dictionary<string,
object> Parameter2 { get; set; }
}
```

How to fix violations

Limit a single parameter to have `CaptureUnmatchedValues` set.

razor

```
@code
{
    [Parameter(CaptureUnmatchedValues = true)] public Dictionary<string,
object> Parameter1 { get; set; }

    [Parameter] public Dictionary<string, object> Parameter2 { get; set; }
}
```

When to suppress warnings

Do not suppress a warning from this rule.

BL0003: Component parameter with CaptureUnmatchedValues has the wrong type

Article • 06/03/2022

Expand table

	Value
Rule ID	BL0003
Category	Usage
Fix is breaking or non-breaking	Breaking

Cause

A parameter on a type deriving from ComponentBase annotated with CaptureUnmatchedValues `= true` is not assignable from `Dictionary<string, object>`

Rule description

Parameters annotated with `CaptureUnmatchedValues = true` must be able to receive a `Dictionary<string, object>` value.

```razor
@code
{
    [Parameter(CaptureUnmatchedValues = true)] public IDictionary<string,
string> Attributes { get; set; }
}
```

How to fix violations

Change the type of the parameter to either `IDictionary<string, object>` or `Dictionary<string, object>`

razor

```
@code
{
    [Parameter(CaptureUnmatchedValues = true)] public IDictionary<string,
object> Attributes { get; set; }
}
```

When to suppress warnings

Do not suppress a warning from this rule.

BL0004: Component parameter should be public

Article • 06/03/2022

Expand table

	Value
Rule ID	BL0004
Category	Usage
Fix is breaking or non-breaking	Breaking

Cause

A property on a type deriving from ComponentBase annotated with [Parameter] is not public.

Rule description

Component parameters are required to be public and must have a public setter.

razor

```
@code
{
    [Parameter] int Parameter1 { get; set; }
}
```

How to fix violations

- Make the property and its setter public.

razor

```
@code
{
    [Parameter] public int Parameter1 { get; set; }
}
```

- If making the property non-public is not possible, consider implementing SetParametersAsync manually.

When to suppress warnings

Do not suppress a warning from this rule.

BL0005: Component parameter should not be set outside of its component

Article • 06/03/2022

Expand table

	Value
Rule ID	BL0005
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A property on a type deriving from ComponentBase annotated with [Parameter] is being assigned to from outside the component.

Rule description

Component parameters should be assigned to as part of the component initialization or as part of `SetParametersAsync`. Assigning a value to a parameter from an external source results in the value being overwritten the next time the component renders.

How to fix violations

Consider using a distinct property to receive values from other components. Additional code can then be written to decide which of the two values to use in the component.

When to suppress warnings

Do not suppress a warning from this rule.

BL0006: Do not use RenderTree types

Article • 06/03/2022

Expand table

	Value
Rule ID	BL0006
Category	Usage
Fix is breaking or non-breaking	Non-breaking

Cause

A reference to a type in the Microsoft.AspNetCore.Components.RenderTree was found.

Rule description

Types in the `Microsoft.AspNetCore.Components.RenderTree` namespace are part of Blazor's implementation detail and are subject to breaking changes. Using these types is not recommended in user code.

How to fix violations

Remove the reference to the type.

When to suppress warnings

It is safe to suppress this rule if using this type is essential. Note that this these are subject to breaking changes between major releases.

MVC1000: Use of IHtmlHelper.Partial should be avoided

Article • 06/03/2022

Expand table

	Value
Rule ID	MVC1000
Fix is breaking or non-breaking	Non-breaking

Cause

The Partial or RenderPartial method was called or referenced.

Rule description

Rendering a partial using `IHtmlHelper.Partial` or `IHtmlHelper.RenderPartial` extension methods results in blocking calls. This may result in performance degradation and application dead locks issues due to thread pool starvation.

How to fix violations

- Use the PartialTagHelper
- Use the PartialAsync or RenderPartialAsync

When to suppress warnings

It's safe to suppress this rule if updating the application to use the suggested fixes is non-trivial. Before the validation is disabled, be sure to consider the risks of thread pool starvation to the application.

MVC1001: Filters cannot be applied to page handler methods

Article • 06/18/2024

Expand table

	Value
Rule ID	MVC1001
Fix is breaking or non-breaking	Non-breaking

Cause

An attribute implementing IFilterMetadata was applied to a Razor Page handler method.

Rule description

Razor Page handler methods are selected after MVC filter execution has started, and consequently cannot contribute filters to execute. Applying a filter to a Razor Page handler is unsupported and always incorrect.

C#

```csharp
public class IndexModel : PageModel
{
    [MyFilter]
    public IActionResult OnGet() => Page();
}
```

How to fix violations

Remove the filter from the handler and apply it to the page model. If a filter has to be applied to a specific handler, consider using multiple Razor Pages.

C#

```csharp
[MyFilter]
public class IndexModel : PageModel
{
    public IActionResult OnGet() => Page();
}
```

When to suppress warnings

Don't suppress warnings from this rule.

MVC1002: Route attribute cannot be applied to page handler methods

Article • 06/18/2024

Expand table

	Value
Rule ID	MVC1002
Fix is breaking or non-breaking	Non-breaking

Cause

An attribute implementing IRouteTemplateProvider was applied to a Razor Page handler method.

Rule description

Razor Page handler methods are selected after routing is completed, and consequently cannot contribute a route. Applying a route attribute such as `HttpGet` or `HttpPost` to a Razor Page handler is not supported.

```C#
public class IndexModel : PageModel
{
    [HttpGet("/my-url")]
    public IActionResult OnGet() => Page();
}
```

How to fix violations

Remove the route attribute from the handler. Routes can be specified for a Razor Page using an `@page` directive or by using conventions. For more information, see custom routes in Razor Pages.

When to suppress warnings

Don't suppress warnings from this rule.

MVC1003: Route attributes cannot be applied to page models

Article • 06/18/2024

Expand table

	Value
Rule ID	MVC1003
Fix is breaking or non-breaking	Non-breaking

Cause

An attribute implementing IRouteTemplateProvider was applied to a Razor Page model.

Rule description

Razor Page models are identified after routing is completed, and consequently cannot contribute a route. Applying a route attribute such as `Route` to a Razor Page model is not supported.

```C#
[Route("/my-page-route")]
public class IndexModel : PageModel
{
    public IActionResult OnGet() => Page();
}
```

How to fix violations

Remove the route attribute from the page model. Routes can be specified for a Razor Page using an `@page` directive or by using conventions. For more information, see custom routes in Razor Pages.

When to suppress warnings

Don't suppress warnings from this rule.

MVC1004: Rename model bound parameter

Article • 06/03/2022

Expand table

	Value
Rule ID	MVC1004
Fix is breaking or non-breaking	Breaking

Cause

A model bound parameter has the same name as one of its properties.

Rule description

Model binding a complex parameter with a property that has the same name may result in unexpected binding behavior. Consider renaming the parameter, or using a binding attribute to specify a different name.

Consider the following code:

```C#
public class HomeController : Controller
{
    public IActionResult Get(SearchModel search)
    {
        ...
    }
}

public class SearcModel
{
    public string Search { get; set; }
}
```

In this model, the parameter and its property are both named `Search`, which results in model binding attempting to bind the property as `search.Search`. Naming a parameter and its property the same prevents binding to a value without a prefix such as a query that looks like `?search=MySearchTerm`.

How to fix violations

- Rename the parameter if its prefix is not used during binding:

```C#
public IActionResult Get(SearchModel model)
{
    ...
}
```

Renaming a parameter on a public type could be considered a breaking change since it changes a library's public API surface.

- If this is problematic, consider using a model binding attribute such as `Bind` to specify the model binding prefix:

```C#
public IActionResult Get([Bind(Prefix = "")] SearchModel search)
{
    ...
}
```

When to suppress warnings

Warnings can be suppressed if you intend to use the parameter name as a prefix during model binding.

MVC1005: Cannot use UseMvc with Endpoint Routing

Article • 06/03/2022

Expand table

	Value
Rule ID	MVC1005
Fix is breaking or non-breaking	Non-breaking

Cause

UseMvc was invoked as part of startup.

Rule description

Using MVC via UseMvc or UseMvcWithDefaultRoute requires an explicit opt-in inside `Startup.ConfigureServices`. This is required because MVC must know whether it can rely on the authorization and CORS Middleware during initialization.

How to fix violations

If the app requires legacy IRouter support, disable EnableEndpointRoutingusing any of the following approaches in `Startup.ConfigureServices`:

```C#
services.AddMvc(options => options.EnableEndpointRouting = false);
```

If legacy `IRouter` support is not required, replace the call to `UseMvc` with `UseEndpoints`. For more details, see the migration guide.

When to suppress warnings

Do not suppress a warning from this rule.

MVC1006: Methods containing TagHelpers must be async and return Task

Article • 06/18/2024

Expand table

	Value
Rule ID	MVC1006
Fix is breaking or non-breaking	Breaking

Cause

A tag helper was defined inside a Razor function that executes synchronously

Rule description

Tag Helper execution is asynchronous. When used inside a method or a lambda within a Razor Page, the containing function must also be declared to be async.

Consider the following cshtml file:

```razor
void Helper(string controller)
{
    <a asp-controller="@controller">Home</a>
}
```

`asp-controller` is a tag helper and will trigger this rule.

How to fix violations

Declare the function to be async and Task returning:

razor

```
async Task Helper(string controller)
{
    <a asp-controller="@controller">Home</a>
}
```

When to suppress warnings

Do not suppress a warning from this rule.

Razor Pages with Entity Framework Core in ASP.NET Core - Tutorial 1 of 8

Article • 09/27/2024

ⓘ **Important**

This information relates to a pre-release product that may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

For the current release, see the .NET 9 version of this article.

By Tom Dykstra, Jeremy Likness, and Jon P Smith

This is the first in a series of tutorials that show how to use Entity Framework (EF) Core in an ASP.NET Core Razor Pages app. The tutorials build a web site for a fictional Contoso University. The site includes functionality such as student admission, course creation, and instructor assignments. The tutorial uses the code first approach. For information on following this tutorial using the database first approach, see this Github issue.

Download or view the completed app. Download instructions.

Prerequisites

- If you're new to Razor Pages, go through the Get started with Razor Pages tutorial series before starting this one.

Visual Studio

- Visual Studio 2022 with the **ASP.NET and web development** workload.
- .NET 6.0 SDK

Database engines

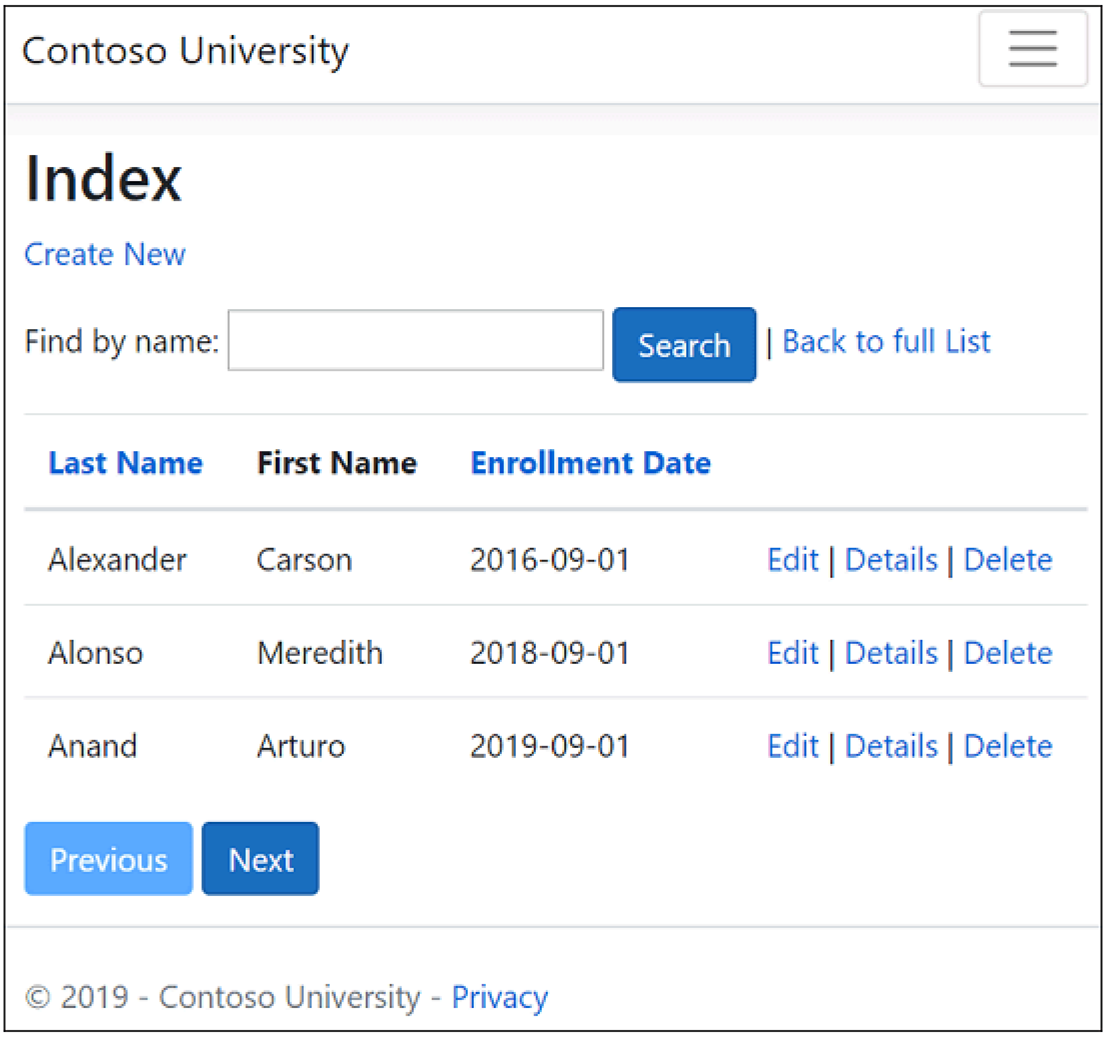
The Visual Studio instructions use SQL Server LocalDB, a version of SQL Server Express that runs only on Windows.

Troubleshooting

If you run into a problem you can't resolve, compare your code to the completed project. A good way to get help is by posting a question to StackOverflow.com, using the ASP.NET Core tag or the EF Core tag.

The sample app

The app built in these tutorials is a basic university web site. Users can view and update student, course, and instructor information. Here are a few of the screens created in the tutorial.


Contoso University

Index

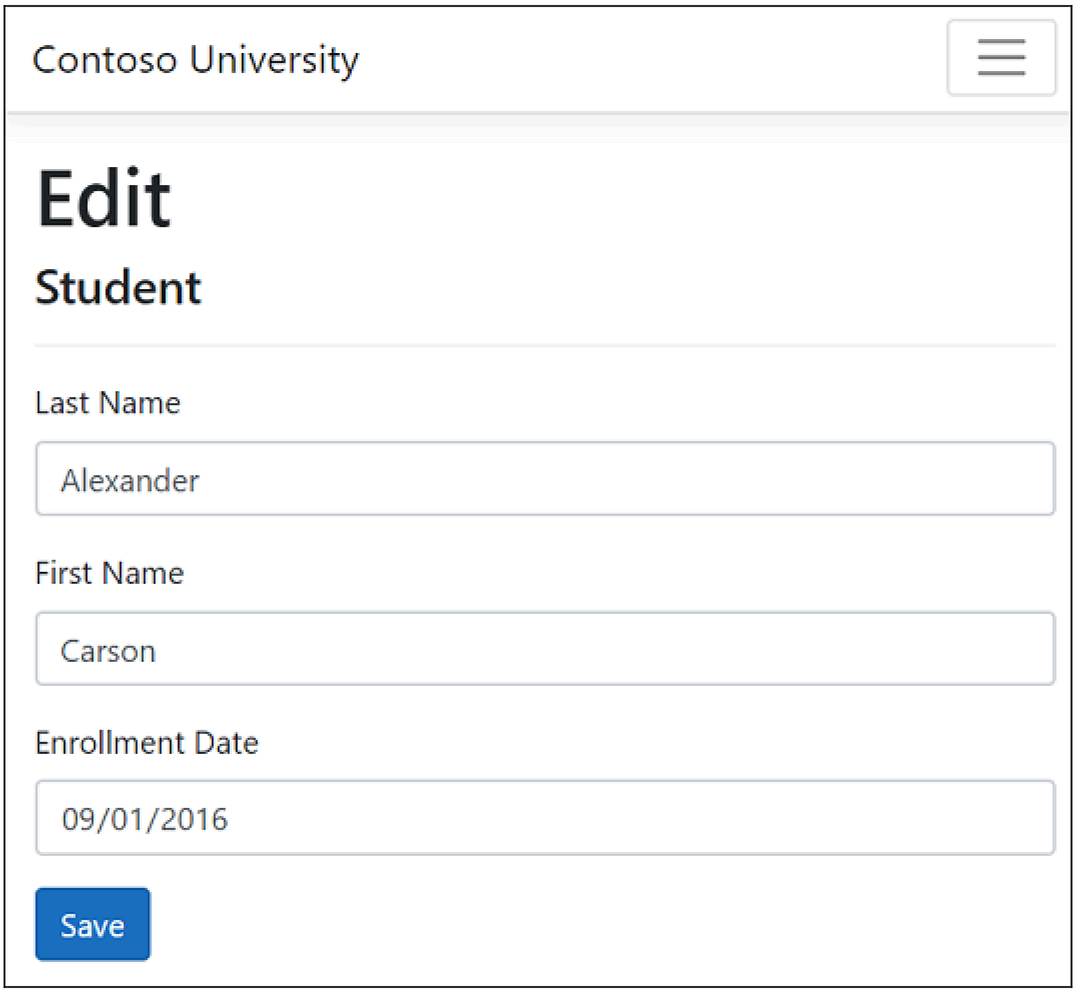
Create New

Find by name: Search | Back to full List

Last Name	First Name	Enrollment Date	
Alexander	Carson	2016-09-01	Edit | Details | Delete
Alonso	Meredith	2018-09-01	Edit | Details | Delete
Anand	Arturo	2019-09-01	Edit | Details | Delete

Previous Next

© 2019 - Contoso University - Privacy


Contoso University

Edit

Student

Last Name

Alexander

First Name

Carson

Enrollment Date

09/01/2016

The UI style of this site is based on the built-in project templates. The tutorial's focus is on how to use EF Core with ASP.NET Core, not how to customize the UI.

Optional: Build the sample download

This step is optional. Building the completed app is recommended when you have problems you can't solve. If you run into a problem you can't resolve, compare your code to the completed project. Download instructions.

Visual Studio

Select `ContosoUniversity.csproj` to open the project.

- Build the project.
- In Package Manager Console (PMC) run the following command:

PowerShell

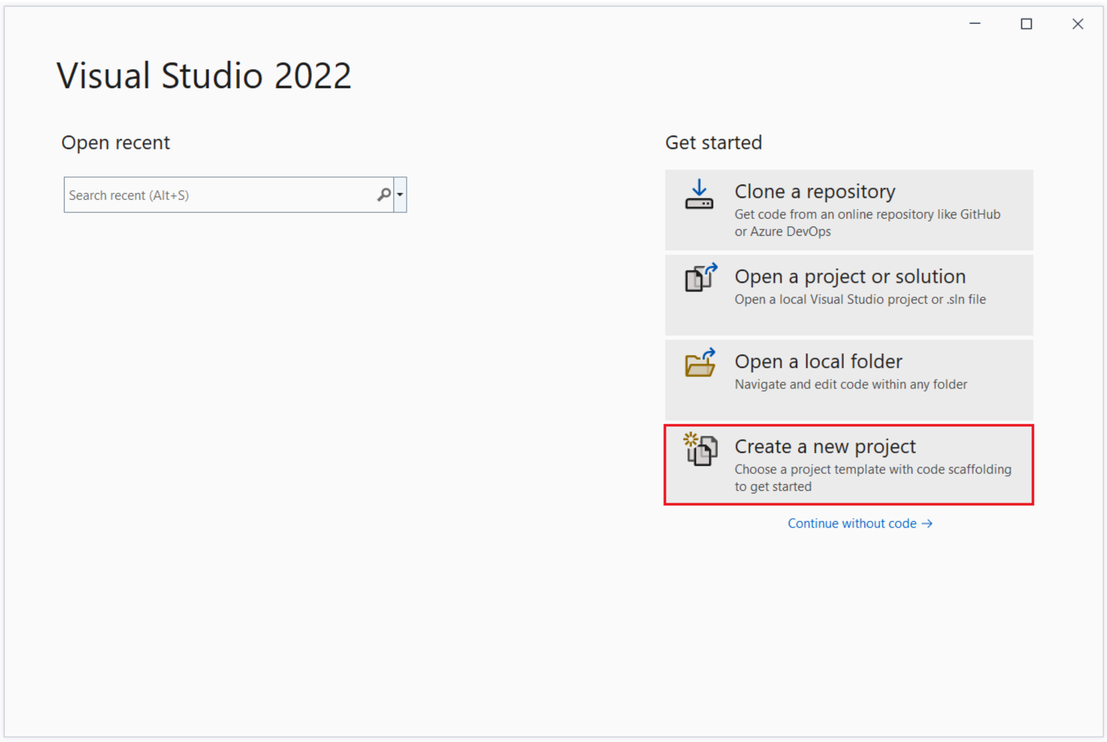
```
Update-Database
```

Run the project to seed the database.

Create the web app project

Visual Studio

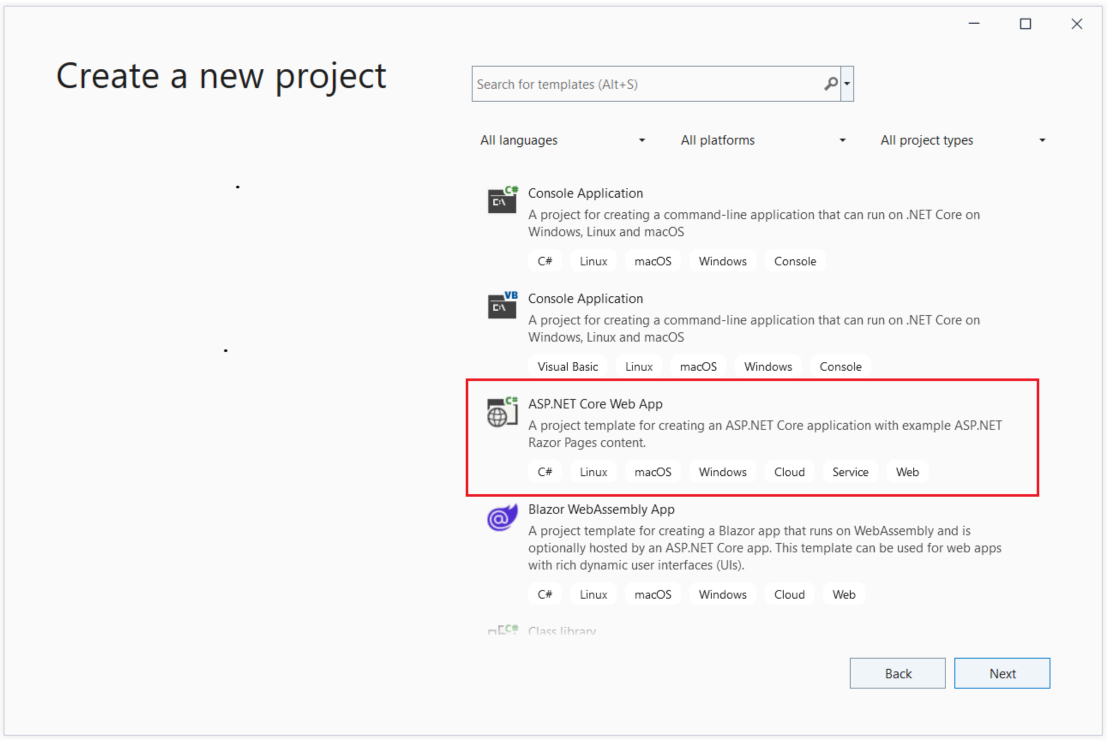
1. Start Visual Studio 2022 and select **Create a new project**.



Visual Studio 2022

Open recent
Search recent (Alt+S)
Get started
Clone a repository
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code →

2. In the **Create a new project** dialog, select **ASP.NET Core Web App**, and then select **Next**.


Create a new project

Search for templates (Alt+S)

All languages | All platforms | All project types

Console Application
A project for creating a command-line application that can run on .NET Core on Windows, Linux and macOS
C# Linux macOS Windows Console

Console Application
A project for creating a command-line application that can run on .NET Core on Windows, Linux and macOS
Visual Basic Linux macOS Windows Console

ASP.NET Core Web App
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
C# Linux macOS Windows Cloud Service Web

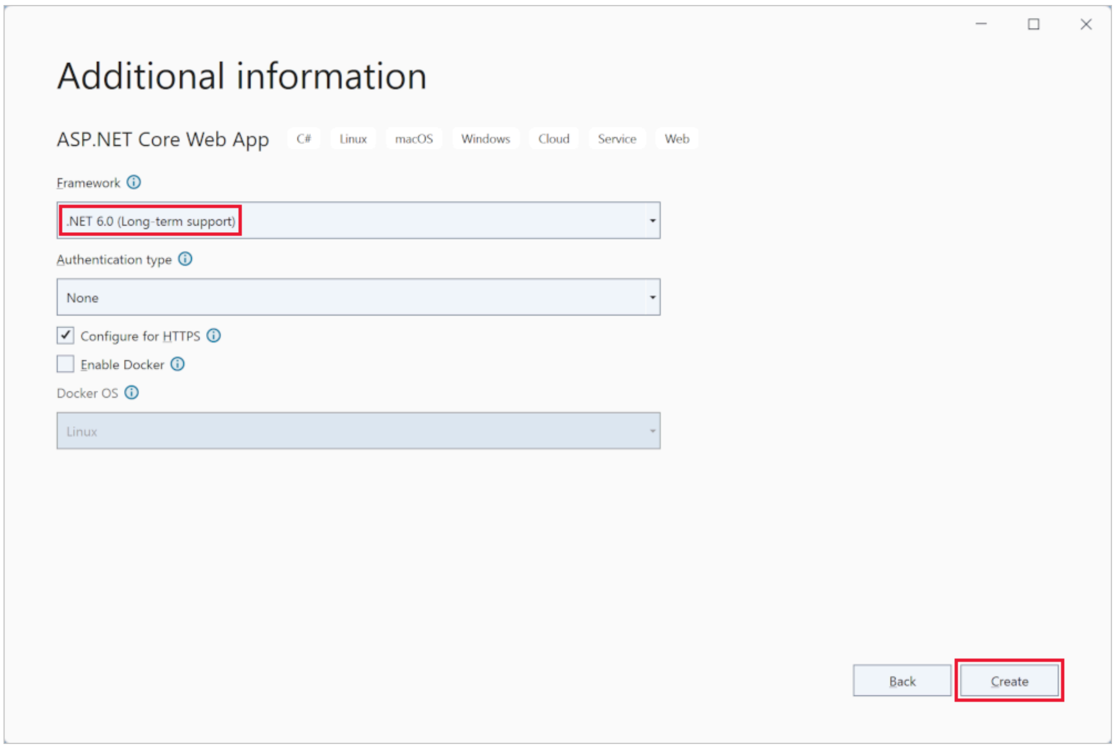
Blazor WebAssembly App
A project template for creating a Blazor app that runs on WebAssembly and is optionally hosted by an ASP.NET Core app. This template can be used for web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Cloud Web

Class library

3. In the **Configure your new project** dialog, enter `ContosoUniversity` for **Project name**. It's important to name the project *ContosoUniversity*, including matching the capitalization, so the namespaces will match when you copy and paste example code.

4. Select **Next**.

5. In the **Additional information** dialog, select **.NET 6.0 (Long-term support)** and then select **Create**.


Additional information

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Framework: .NET 6.0 (Long-term support)

Authentication type: None

☑ Configure for HTTPS

☐ Enable Docker

Docker OS: Linux

Set up the site style

Copy and paste the following code into the `Pages/Shared/_Layout.cshtml` file:

CSHTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Contoso University</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true"
/>
    <link rel="stylesheet" href="~/ContosoUniversity.styles.css" asp-append-
version="true" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-
light bg-white border-bottom box-shadow mb-3">
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-
page="/Index">Contoso University</a>
                <button class="navbar-toggler" type="button" data-bs-
toggle="collapse" data-bs-target=".navbar-collapse" aria-
controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle
navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex
justify-content-between">
                    <ul class="navbar-nav flex-grow-1">
```

```
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
page="/About">About</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
page="/Students/Index">Students</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
page="/Courses/Index">Courses</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
page="/Instructors/Index">Instructors</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
page="/Departments/Index">Departments</a>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
    </header>
    <div class="container">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>

    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2021 - Contoso University - <a asp-area="" asp-
page="/Privacy">Privacy</a>
        </div>
    </footer>

    <script src="~/lib/jquery/dist/jquery.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>

    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

The layout file sets the site header, footer, and menu. The preceding code makes the following changes:

- Each occurrence of "ContosoUniversity" to "Contoso University". There are three occurrences.
- The **Home** and **Privacy** menu entries are deleted.

- Entries are added for **About**, **Students**, **Courses**, **Instructors**, and **Departments**.

In `Pages/Index.cshtml`, replace the contents of the file with the following code:

CSHTML

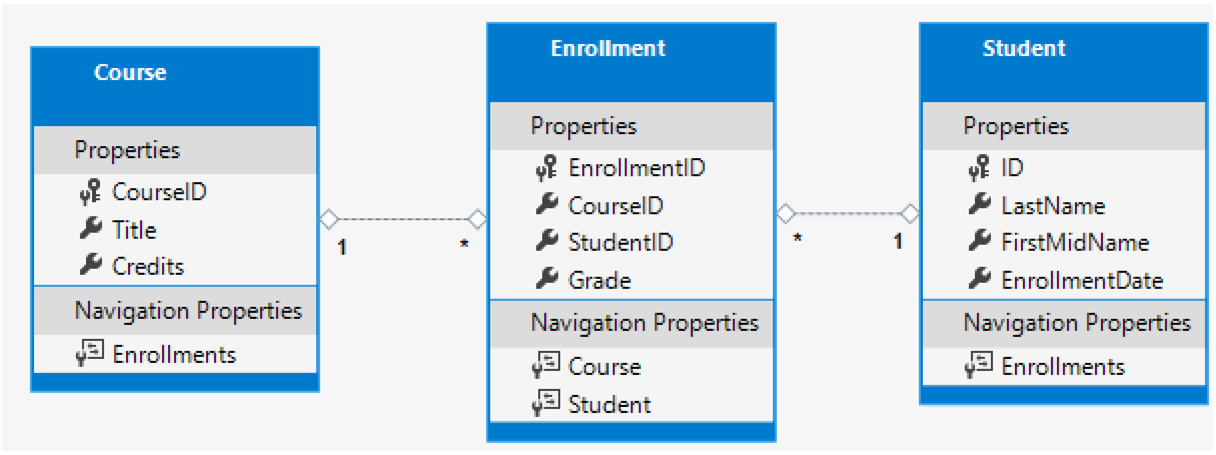
```
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}

<div class="row mb-auto">
    <div class="col-md-4">
        <div class="row no-gutters border mb-4">
            <div class="col p-4 mb-4 ">
                <p class="card-text">
                    Contoso University is a sample application that
                    demonstrates how to use Entity Framework Core in an
                    ASP.NET Core Razor Pages web app.
                </p>
            </div>
        </div>
    </div>
    <div class="col-md-4">
        <div class="row no-gutters border mb-4">
            <div class="col p-4 d-flex flex-column position-static">
                <p class="card-text mb-auto">
                    You can build the application by following the steps in
a series of tutorials.
                </p>
                <p>
@*                      <a
href="https://docs.microsoft.com/aspnet/core/data/ef-rp/intro"
class="stretched-link">See the tutorial</a>
*@                  </p>
            </div>
        </div>
    </div>
    <div class="col-md-4">
        <div class="row no-gutters border mb-4">
            <div class="col p-4 d-flex flex-column">
                <p class="card-text mb-auto">
                    You can download the completed project from GitHub.
                </p>
                <p>
@*                      <a
href="https://github.com/dotnet/AspNetCore.Docs/tree/main/aspnetcore/data/ef
-rp/intro/samples" class="stretched-link">See project source code</a>
*@                  </p>
            </div>
        </div>
    </div>
</div>
```

The preceding code replaces the text about ASP.NET Core with text about this app.

Run the app to verify that the home page appears.

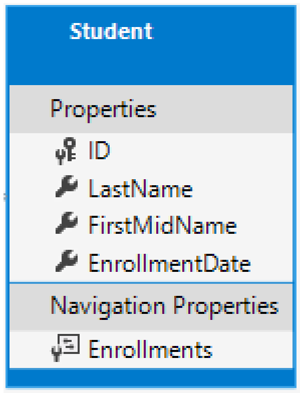
The data model

The following sections create a data model:



A student can enroll in any number of courses, and a course can have any number of students enrolled in it.

The Student entity



- Create a *Models* folder in the project folder.
- Create `Models/Student.cs` with the following code:

C#

```
namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }
        public string LastName { get; set; }
        public string FirstMidName { get; set; }
        public DateTime EnrollmentDate { get; set; }

        public ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

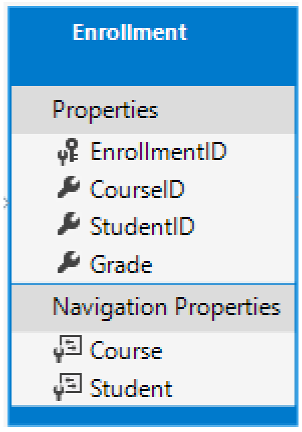
The `ID` property becomes the primary key column of the database table that corresponds to this class. By default, EF Core interprets a property that's named `ID` or `classnameID` as the primary key. So the alternative automatically recognized name for the `Student` class primary key is `StudentID`. For more information, see EF Core - Keys.

The `Enrollments` property is a navigation property. Navigation properties hold other entities that are related to this entity. In this case, the `Enrollments` property of a `Student` entity holds all of the `Enrollment` entities that are related to that Student. For example, if a Student row in the database has two related Enrollment rows, the `Enrollments` navigation property contains those two Enrollment entities.

In the database, an Enrollment row is related to a Student row if its `StudentID` column contains the student's ID value. For example, suppose a Student row has ID=1. Related Enrollment rows will have `StudentID` = 1. `StudentID` is a *foreign key* in the Enrollment table.

The `Enrollments` property is defined as `ICollection<Enrollment>` because there may be multiple related Enrollment entities. Other collection types can be used, such as `List<Enrollment>` or `HashSet<Enrollment>`. When `ICollection<Enrollment>` is used, EF Core creates a `HashSet<Enrollment>` collection by default.

The Enrollment entity



Create `Models/Enrollment.cs` with the following code:

```
C#

using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
    public enum Grade
    {
        A, B, C, D, F
    }

    public class Enrollment
    {
        public int EnrollmentID { get; set; }
        public int CourseID { get; set; }
        public int StudentID { get; set; }
        [DisplayFormat(NullDisplayText = "No grade")]
        public Grade? Grade { get; set; }

        public Course Course { get; set; }
        public Student Student { get; set; }
    }
}
```

The `EnrollmentID` property is the primary key; this entity uses the `classnameID` pattern instead of `ID` by itself. For a production data model, many developers choose one pattern and use it consistently. This tutorial uses both just to illustrate that both work. Using `ID` without `classname` makes it easier to implement some kinds of data model changes.

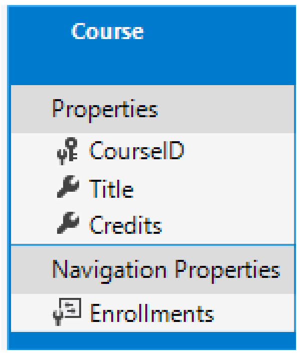
The `Grade` property is an `enum`. The question mark after the `Grade` type declaration indicates that the `Grade` property is nullable. A grade that's null is different from a zero grade—null means a grade isn't known or hasn't been assigned yet.

The `StudentID` property is a foreign key, and the corresponding navigation property is `Student`. An `Enrollment` entity is associated with one `Student` entity, so the property contains a single `Student` entity.

The `CourseID` property is a foreign key, and the corresponding navigation property is `Course`. An `Enrollment` entity is associated with one `Course` entity.

EF Core interprets a property as a foreign key if it's named `<navigation property name><primary key property name>`. For example, `StudentID` is the foreign key for the `Student` navigation property, since the `Student` entity's primary key is `ID`. Foreign key properties can also be named `<primary key property name>`. For example, `CourseID` since the `Course` entity's primary key is `CourseID`.

The Course entity



Create `Models/Course.cs` with the following code:

C#

```
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Course
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int CourseID { get; set; }
        public string Title { get; set; }
        public int Credits { get; set; }

        public ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

The `Enrollments` property is a navigation property. A `Course` entity can be related to any number of `Enrollment` entities.

The `DatabaseGenerated` attribute allows the app to specify the primary key rather than having the database generate it.

Build the app. The compiler generates several warnings about how `null` values are handled. See this GitHub issue, Nullable reference types, and Tutorial: Express your design intent more clearly with nullable and non-nullable reference types for more information.

To eliminate the warnings from nullable reference types, remove the following line from the `ContosoUniversity.csproj` file:

XML

```xml
<Nullable>enable</Nullable>
```

The scaffolding engine currently does not support nullable reference types, therefore the models used in scaffold can't either.

Remove the `?` nullable reference type annotation from `public string? RequestId { get; set; }` in `Pages/Error.cshtml.cs` so the project builds without compiler warnings.

Scaffold Student pages

In this section, the ASP.NET Core scaffolding tool is used to generate:

- An EF Core `DbContext` class. The context is the main class that coordinates Entity Framework functionality for a given data model. It derives from the Microsoft.EntityFrameworkCore.DbContext class.
- Razor pages that handle Create, Read, Update, and Delete (CRUD) operations for the `Student` entity.

Visual Studio

- Create a *Pages/Students* folder.
- In **Solution Explorer**, right-click the *Pages/Students* folder and select **Add** > **New Scaffolded Item**.
- In the **Add New Scaffold Item** dialog:
  - In the left tab, select **Installed** > **Common** > **Razor Pages**
  - Select **Razor Pages using Entity Framework (CRUD)** > **ADD**.
- In the **Add Razor Pages using Entity Framework (CRUD)** dialog:
  - In the **Model class** drop-down, select **Student (ContosoUniversity.Models)**.
  - In the **Data context class** row, select the **+** (plus) sign.
    - Change the data context name to end in `SchoolContext` rather than `ContosoUniversityContext`. The updated context name: `ContosoUniversity.Data.SchoolContext`
    - Select **Add** to finish adding the data context class.
    - Select **Add** to finish the **Add Razor Pages** dialog.

The following packages are automatically installed:

- `Microsoft.EntityFrameworkCore.SqlServer`
- `Microsoft.EntityFrameworkCore.Tools`

- `Microsoft.VisualStudio.Web.CodeGeneration.Design`

If the preceding step fails, build the project and retry the scaffold step.

The scaffolding process:

- Creates Razor pages in the *Pages/Students* folder:
  - `Create.cshtml` and `Create.cshtml.cs`
  - `Delete.cshtml` and `Delete.cshtml.cs`
  - `Details.cshtml` and `Details.cshtml.cs`
  - `Edit.cshtml` and `Edit.cshtml.cs`
  - `Index.cshtml` and `Index.cshtml.cs`
- Creates `Data/SchoolContext.cs`.
- Adds the context to dependency injection in `Program.cs`.
- Adds a database connection string to `appsettings.json`.

Database connection string

The scaffolding tool generates a connection string in the `appsettings.json` file.

Visual Studio

The connection string specifies SQL Server LocalDB:

JSON

```json
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "SchoolContext": "Server=
(localdb)\\mssqllocaldb;Database=SchoolContext-
0e9;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

LocalDB is a lightweight version of the SQL Server Express Database Engine and is intended for app development, not production use. By default, LocalDB creates *.mdf* files in the `C:/Users/<user>` directory.

Update the database context class

The main class that coordinates EF Core functionality for a given data model is the database context class. The context is derived from Microsoft.EntityFrameworkCore.DbContext. The context specifies which entities are included in the data model. In this project, the class is named `SchoolContext`.

Update `Data/SchoolContext.cs` with the following code:

```
C#
```

```csharp
using Microsoft.EntityFrameworkCore;
using ContosoUniversity.Models;

namespace ContosoUniversity.Data
{
    public class SchoolContext : DbContext
    {
        public SchoolContext (DbContextOptions<SchoolContext> options)
            : base(options)
        {
        }

        public DbSet<Student> Students { get; set; }
        public DbSet<Enrollment> Enrollments { get; set; }
        public DbSet<Course> Courses { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Course>().ToTable("Course");
            modelBuilder.Entity<Enrollment>().ToTable("Enrollment");
            modelBuilder.Entity<Student>().ToTable("Student");
        }
    }
}
```

The preceding code changes from the singular `DbSet<Student> Student` to the plural `DbSet<Student> Students`. To make the Razor Pages code match the new `DBSet` name, make a global change from: `_context.Student.` to: `_context.Students.`

There are 8 occurrences.

Because an entity set contains multiple entities, many developers prefer the `DBSet` property names should be plural.

The highlighted code:

- Creates a DbSet<TEntity> property for each entity set. In EF Core terminology:
  - An entity set typically corresponds to a database table.
  - An entity corresponds to a row in the table.
- Calls OnModelCreating. `OnModelCreating`:
  - Is called when `SchoolContext` has been initialized but before the model has been secured and used to initialize the context.
  - Is required because later in the tutorial the `Student` entity will have references to the other entities.

We hope to fix this issue in a future release.

Program.cs

ASP.NET Core is built with dependency injection. Services such as the `SchoolContext` are registered with dependency injection during app startup. Components that require these services, such as Razor Pages, are provided these services via constructor parameters. The constructor code that gets a database context instance is shown later in the tutorial.

The scaffolding tool automatically registered the context class with the dependency injection container.

Visual Studio

The following highlighted lines were added by the scaffolder:

C#

```csharp
using ContosoUniversity.Data;
using Microsoft.EntityFrameworkCore;
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages();

builder.Services.AddDbContext<SchoolContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("SchoolCo
ntext")));
```

The name of the connection string is passed in to the context by calling a method on a DbContextOptions object. For local development, the ASP.NET Core configuration system reads the connection string from the `appsettings.json` or the `appsettings.Development.json` file.

Add the database exception filter

Add AddDatabaseDeveloperPageExceptionFilter and UseMigrationsEndPoint as shown in the following code:

Visual Studio

C#

```
using ContosoUniversity.Data;
using Microsoft.EntityFrameworkCore;
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages();

builder.Services.AddDbContext<SchoolContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("SchoolCo
ntext")));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}
else
{
    app.UseDeveloperExceptionPage();
    app.UseMigrationsEndPoint();
}
```

Add the Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore NuGet package.

In the Package Manager Console, enter the following to add the NuGet package:

PowerShell

```
Install-Package Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore
```

The `Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore` NuGet package provides ASP.NET Core middleware for Entity Framework Core error pages. This middleware helps to detect and diagnose errors with Entity Framework Core migrations.

The `AddDatabaseDeveloperPageExceptionFilter` provides helpful error information in the development environment for EF migrations errors.

Create the database

Update `Program.cs` to create the database if it doesn't exist:

Visual Studio

C#

```csharp
using ContosoUniversity.Data;
using Microsoft.EntityFrameworkCore;
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages();

builder.Services.AddDbContext<SchoolContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("SchoolCo
ntext")));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}
else
{
    app.UseDeveloperExceptionPage();
    app.UseMigrationsEndPoint();
}

using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;

    var context = services.GetRequiredService<SchoolContext>();
    context.Database.EnsureCreated();
    // DbInitializer.Initialize(context);
}
```

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

app.UseRouting();

app.UseAuthorization();

app.MapRazorPages();

app.Run();
```

The EnsureCreated method takes no action if a database for the context exists. If no database exists, it creates the database and schema. `EnsureCreated` enables the following workflow for handling data model changes:

- Delete the database. Any existing data is lost.
- Change the data model. For example, add an `EmailAddress` field.
- Run the app.
- `EnsureCreated` creates a database with the new schema.

This workflow works early in development when the schema is rapidly evolving, as long as data doesn't need to be preserved. The situation is different when data that has been entered into the database needs to be preserved. When that is the case, use migrations.

Later in the tutorial series, the database is deleted that was created by `EnsureCreated` and migrations is used. A database that is created by `EnsureCreated` can't be updated by using migrations.

Test the app

- Run the app.
- Select the **Students** link and then **Create New**.
- Test the Edit, Details, and Delete links.

Seed the database

The `EnsureCreated` method creates an empty database. This section adds code that populates the database with test data.

Create `Data/DbInitializer.cs` with the following code:

C#

```
using ContosoUniversity.Models;

namespace ContosoUniversity.Data
{
    public static class DbInitializer
    {
        public static void Initialize(SchoolContext context)
        {
            // Look for any students.
            if (context.Students.Any())
            {
                return;   // DB has been seeded
            }

            var students = new Student[]
            {
                new
Student{FirstMidName="Carson",LastName="Alexander",EnrollmentDate=DateTime.P
arse("2019-09-01")},
                new
Student{FirstMidName="Meredith",LastName="Alonso",EnrollmentDate=DateTime.Pa
rse("2017-09-01")},
                new
Student{FirstMidName="Arturo",LastName="Anand",EnrollmentDate=DateTime.Parse
("2018-09-01")},
                new
Student{FirstMidName="Gytis",LastName="Barzdukas",EnrollmentDate=DateTime.Pa
rse("2017-09-01")},
                new
Student{FirstMidName="Yan",LastName="Li",EnrollmentDate=DateTime.Parse("2017
-09-01")},
                new
Student{FirstMidName="Peggy",LastName="Justice",EnrollmentDate=DateTime.Pars
e("2016-09-01")},
                new
Student{FirstMidName="Laura",LastName="Norman",EnrollmentDate=DateTime.Parse
("2018-09-01")},
                new
Student{FirstMidName="Nino",LastName="Olivetto",EnrollmentDate=DateTime.Pars
e("2019-09-01")}
            };

            context.Students.AddRange(students);
            context.SaveChanges();

            var courses = new Course[]
            {
                new Course{CourseID=1050,Title="Chemistry",Credits=3},
                new Course{CourseID=4022,Title="Microeconomics",Credits=3},
                new Course{CourseID=4041,Title="Macroeconomics",Credits=3},
                new Course{CourseID=1045,Title="Calculus",Credits=4},
                new Course{CourseID=3141,Title="Trigonometry",Credits=4},
                new Course{CourseID=2021,Title="Composition",Credits=3},
                new Course{CourseID=2042,Title="Literature",Credits=4}
            };

            context.Courses.AddRange(courses);
            context.SaveChanges();

            var enrollments = new Enrollment[]
            {
                new Enrollment{StudentID=1,CourseID=1050,Grade=Grade.A},
                new Enrollment{StudentID=1,CourseID=4022,Grade=Grade.C},
                new Enrollment{StudentID=1,CourseID=4041,Grade=Grade.B},
                new Enrollment{StudentID=2,CourseID=1045,Grade=Grade.B},
                new Enrollment{StudentID=2,CourseID=3141,Grade=Grade.F},
                new Enrollment{StudentID=2,CourseID=2021,Grade=Grade.F},
                new Enrollment{StudentID=3,CourseID=1050},
                new Enrollment{StudentID=4,CourseID=1050},
                new Enrollment{StudentID=4,CourseID=4022,Grade=Grade.F},
                new Enrollment{StudentID=5,CourseID=4041,Grade=Grade.C},
                new Enrollment{StudentID=6,CourseID=1045},
                new Enrollment{StudentID=7,CourseID=3141,Grade=Grade.A},
            };

            context.Enrollments.AddRange(enrollments);
            context.SaveChanges();
        }
    }
}
```

The code checks if there are any students in the database. If there are no students, it adds test data to the database. It creates the test data in arrays rather than `List<T>` collections to optimize performance.

- In `Program.cs`, remove `//` from the `DbInitializer.Initialize` line:

C#

```
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;

    var context = services.GetRequiredService<SchoolContext>();
    context.Database.EnsureCreated();
    DbInitializer.Initialize(context);
}
```

Visual Studio

- Stop the app if it's running, and run the following command in the **Package Manager Console** (PMC):

PowerShell

```
Drop-Database -Confirm
```

- Respond with `Y` to delete the database.

- Restart the app.
- Select the Students page to see the seeded data.

View the database

Visual Studio

- Open **SQL Server Object Explorer** (SSOX) from the **View** menu in Visual Studio.
- In SSOX, select **(localdb)\MSSQLLocalDB > Databases > SchoolContext-{GUID}**. The database name is generated from the context name provided earlier plus a dash and a GUID.
- Expand the **Tables** node.
- Right-click the **Student** table and click **View Data** to see the columns created and the rows inserted into the table.
- Right-click the **Student** table and click **View Code** to see how the `Student` model maps to the `Student` table schema.

Asynchronous EF methods in ASP.NET Core web apps

Asynchronous programming is the default mode for ASP.NET Core and EF Core.

A web server has a limited number of threads available, and in high load situations all of the available threads might be in use. When that happens, the server can't process new requests until the threads are freed up. With synchronous code, many threads may be tied up while they aren't doing work because they're waiting for I/O to complete. With asynchronous code, when a process is waiting for I/O to complete, its thread is freed up for the server to use for processing other requests. As a result, asynchronous code enables server resources to be used more efficiently, and the server can handle more traffic without delays.

Asynchronous code does introduce a small amount of overhead at run time. For low traffic situations, the performance hit is negligible, while for high traffic situations, the potential performance improvement is substantial.

In the following code, the async keyword, `Task` return value, `await` keyword, and `ToListAsync` method make the code execute asynchronously.

```C#
public async Task OnGetAsync()
{
    Students = await _context.Students.ToListAsync();
}
```

- The `async` keyword tells the compiler to:
  - Generate callbacks for parts of the method body.
  - Create the Task object that's returned.
- The `Task` return type represents ongoing work.
- The `await` keyword causes the compiler to split the method into two parts. The first part ends with the operation that's started asynchronously. The second part is put into a callback method that's called when the operation completes.
- `ToListAsync` is the asynchronous version of the `ToList` extension method.

Some things to be aware of when writing asynchronous code that uses EF Core:

- Only statements that cause queries or commands to be sent to the database are executed asynchronously. That includes `ToListAsync`, `SingleOrDefaultAsync`, `FirstOrDefaultAsync`, and `SaveChangesAsync`. It doesn't include statements that just change an `IQueryable`, such as `var students = context.Students.Where(s => s.LastName == "Davolio")`.
- An EF Core context isn't thread safe: don't try to do multiple operations in parallel.
- To take advantage of the performance benefits of async code, verify that library packages (such as for paging) use async if they call EF Core methods that send queries to the database.

For more information about asynchronous programming in .NET, see Async Overview and Asynchronous programming with async and await.

> ⚠ **Warning**
>
> The async implementation of Microsoft.Data.SqlClient has some known issues (#593, #601, and others). If you're seeing unexpected performance problems,

try using sync command execution instead, especially when dealing with large text or binary values.

Performance considerations

In general, a web page shouldn't be loading an arbitrary number of rows. A query should use paging or a limiting approach. For example, the preceding query could use `Take` to limit the rows returned:

C#

```
public async Task OnGetAsync()
{
    Student = await _context.Students.Take(10).ToListAsync();
}
```

Enumerating a large table in a view could return a partially constructed HTTP 200 response if a database exception occurs part way through the enumeration.

Paging is covered later in the tutorial.

For more information, see Performance considerations (EF).

Next steps

Use SQLite for development, SQL Server for production

Next tutorial

Part 2, Razor Pages with EF Core in ASP.NET Core - CRUD

Article • 07/26/2024

ⓘ **Important**

This information relates to a pre-release product that may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

For the current release, see the .NET 9 version of this article.

By Tom Dykstra, Jeremy Likness, and Jon P Smith

The Contoso University web app demonstrates how to create Razor Pages web apps using EF Core and Visual Studio. For information about the tutorial series, see the first tutorial.

If you run into problems you can't solve, download the completed app and compare that code to what you created by following the tutorial.

In this tutorial, the scaffolded CRUD (create, read, update, delete) code is reviewed and customized.

No repository

Some developers use a service layer or repository pattern to create an abstraction layer between the UI (Razor Pages) and the data access layer. This tutorial doesn't do that. To minimize complexity and keep the tutorial focused on EF Core, EF Core code is added directly to the page model classes.

Update the Details page

The scaffolded code for the Students pages doesn't include enrollment data. In this section, enrollments are added to the `Details` page.

Read enrollments

To display a student's enrollment data on the page, the enrollment data must be read. The scaffolded code in `Pages/Students/Details.cshtml.cs` reads only the `Student` data, without the `Enrollment` data:

C#

```csharp
public async Task<IActionResult> OnGetAsync(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    Student = await _context.Students.FirstOrDefaultAsync(m => m.ID == id);

    if (Student == null)
    {
        return NotFound();
    }
    return Page();
}
```

Replace the `OnGetAsync` method with the following code to read enrollment data for the selected student. The changes are highlighted.

C#

```csharp
public async Task<IActionResult> OnGetAsync(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    Student = await _context.Students
        .Include(s => s.Enrollments)
        .ThenInclude(e => e.Course)
        .AsNoTracking()
        .FirstOrDefaultAsync(m => m.ID == id);

    if (Student == null)
    {
        return NotFound();
    }
    return Page();
}
```

The Include and ThenInclude methods cause the context to load the `Student.Enrollments` navigation property, and within each enrollment the

`Enrollment.Course` navigation property. These methods are examined in detail in the Read related data tutorial.

The AsNoTracking method improves performance in scenarios where the entities returned are not updated in the current context. `AsNoTracking` is discussed later in this tutorial.

Display enrollments

Replace the code in `Pages/Students/Details.cshtml` with the following code to display a list of enrollments. The changes are highlighted.

CSHTML

```
@page
@model ContosoUniversity.Pages.Students.DetailsModel

@{
    ViewData["Title"] = "Details";
}

<h1>Details</h1>

<div>
    <h4>Student</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Student.LastName)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Student.LastName)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Student.FirstMidName)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Student.FirstMidName)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Student.EnrollmentDate)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Student.EnrollmentDate)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Student.Enrollments)
        </dt>
        <dd class="col-sm-10">
            <table class="table">
                <tr>
                    <th>Course Title</th>
                    <th>Grade</th>
                </tr>
                @foreach (var item in Model.Student.Enrollments)
                {
                    <tr>
                        <td>
                            @Html.DisplayFor(modelItem => item.Course.Title)
                        </td>
                        <td>
                            @Html.DisplayFor(modelItem => item.Grade)
                        </td>
                    </tr>
                }
            </table>
        </dd>
    </dl>
</div>
<div>
    <a asp-page="./Edit" asp-route-id="@Model.Student.ID">Edit</a> |
    <a asp-page="./Index">Back to List</a>
</div>
```

The preceding code loops through the entities in the `Enrollments` navigation property. For each enrollment, it displays the course title and the grade. The course title is retrieved from the `Course` entity that's stored in the `Course` navigation property of the Enrollments entity.

Run the app, select the **Students** tab, and click the **Details** link for a student. The list of courses and grades for the selected student is displayed.

Ways to read one entity

The generated code uses FirstOrDefaultAsync to read one entity. This method returns null if nothing is found; otherwise, it returns the first row found that satisfies the query filter criteria. `FirstOrDefaultAsync` is generally a better choice than the following alternatives:

- SingleOrDefaultAsync - Throws an exception if there's more than one entity that satisfies the query filter. To determine if more than one row could be returned by the query, `SingleOrDefaultAsync` tries to fetch multiple rows. This extra work is unnecessary if the query can only return one entity, as when it searches on a unique key.

- FindAsync - Finds an entity with the primary key (PK). If an entity with the PK is being tracked by the context, it's returned without a request to the database. This method is optimized to look up a single entity, but you can't call `Include` with `FindAsync`. So if related data is needed, `FirstOrDefaultAsync` is the better choice.

Route data vs. query string

The URL for the Details page is `https://localhost:<port>/Students/Details?id=1`. The entity's primary key value is in the query string. Some developers prefer to pass the key value in route data: `https://localhost:<port>/Students/Details/1`. For more information, see Update the generated code.

Update the Create page

The scaffolded `OnPostAsync` code for the Create page is vulnerable to overposting. Replace the `OnPostAsync` method in `Pages/Students/Create.cshtml.cs` with the following code.

C#

```csharp
public async Task<IActionResult> OnPostAsync()
{
    var emptyStudent = new Student();

    if (await TryUpdateModelAsync<Student>(
        emptyStudent,
        "student",   // Prefix for form value.
        s => s.FirstMidName, s => s.LastName, s => s.EnrollmentDate))
    {
        _context.Students.Add(emptyStudent);
        await _context.SaveChangesAsync();
        return RedirectToPage("./Index");
    }

    return Page();
}
```

TryUpdateModelAsync

The preceding code creates a Student object and then uses posted form fields to update the Student object's properties. The TryUpdateModelAsync method:

- Uses the posted form values from the PageContext property in the PageModel.
- Updates only the properties listed (`s => s.FirstMidName, s => s.LastName, s => s.EnrollmentDate`).
- Looks for form fields with a "student" prefix. For example, `Student.FirstMidName`. It's not case sensitive.
- Uses the model binding system to convert form values from strings to the types in the `Student` model. For example, `EnrollmentDate` is converted to `DateTime`.

Run the app, and create a student entity to test the Create page.

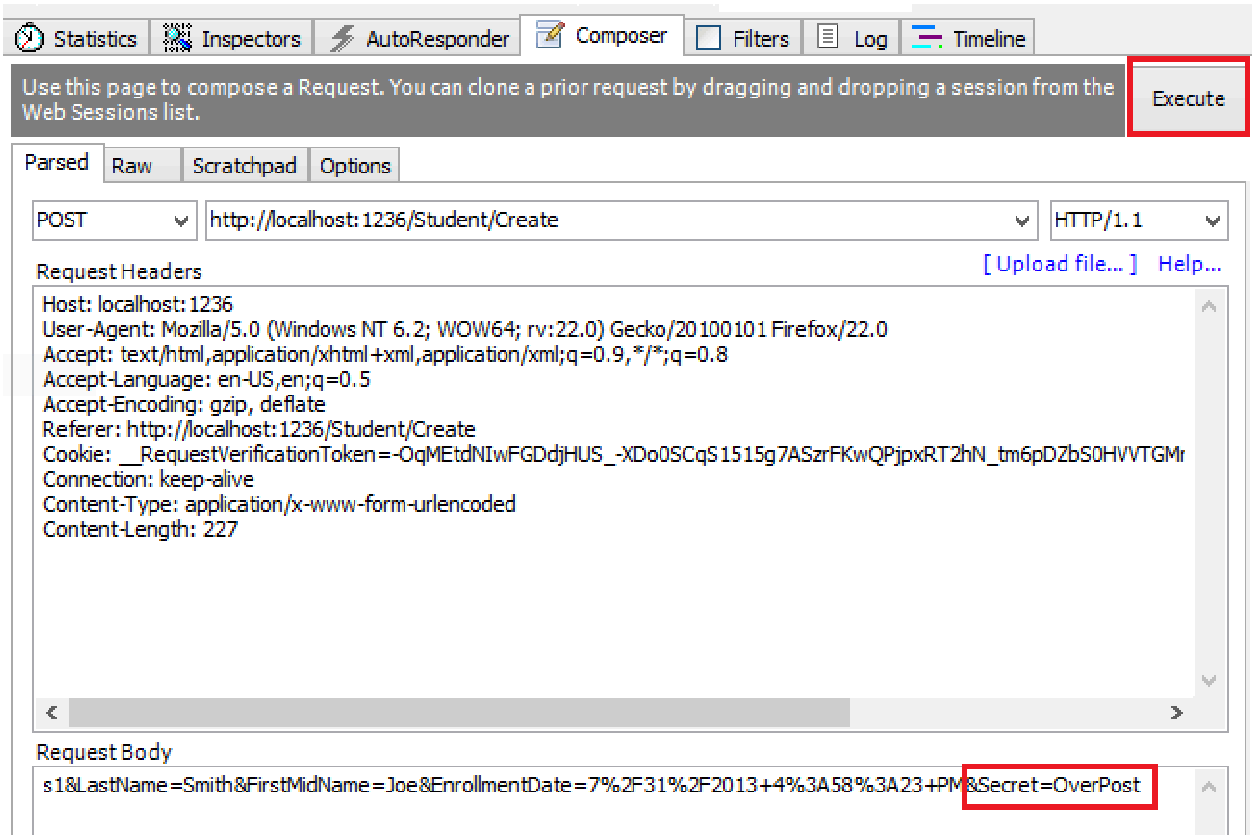
Overposting

Using `TryUpdateModel` to update fields with posted values is a security best practice because it prevents overposting. For example, suppose the Student entity includes a `Secret` property that this web page shouldn't update or add:

```C#
public class Student
{
    public int ID { get; set; }
    public string LastName { get; set; }
    public string FirstMidName { get; set; }
    public DateTime EnrollmentDate { get; set; }
    public string Secret { get; set; }
}
```

Even if the app doesn't have a `Secret` field on the create or update Razor Page, a hacker could set the `Secret` value by overposting. A hacker could use a tool such as Fiddler, or write some JavaScript, to post a `Secret` form value. The original code doesn't limit the fields that the model binder uses when it creates a Student instance.

Whatever value the hacker specified for the `Secret` form field is updated in the database. The following image shows the Fiddler tool adding the `Secret` field, with the value "OverPost", to the posted form values.


Statistics
Inspectors
AutoResponder
Composer
Filters
Log
Timeline
Use this page to compose a Request. You can clone a prior request by dragging and dropping a session from the Web Sessions list.
Execute
Parsed
Raw
Scratchpad
Options
[Upload file...]
Help...

```
POST    http://localhost:1236/Student/Create    HTTP/1.1
```

Request Headers

```
Host: localhost:1236
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:1236/Student/Create
Cookie: __RequestVerificationToken=-OqMEtdNIwFGDdjHUS_-XDo0SCqS1515g7ASzrFKwQPjpxRT2hN_tm6pDZbS0HVVTGMr
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 227
```

Request Body

```
s1&LastName=Smith&FirstMidName=Joe&EnrollmentDate=7%2F31%2F2013+4%3A58%3A23+PM&Secret=OverPost
```

The value "OverPost" is successfully added to the `Secret` property of the inserted row. That happens even though the app designer never intended the `Secret` property to be set with the Create page.

View model

View models provide an alternative way to prevent overposting.

The application model is often called the domain model. The domain model typically contains all the properties required by the corresponding entity in the database. The view model contains only the properties needed for the UI page, for example, the Create page.

In addition to the view model, some apps use a binding model or input model to pass data between the Razor Pages page model class and the browser.

Consider the following `StudentVM` view model:

C#

```csharp
public class StudentVM
{
    public int ID { get; set; }
    public string LastName { get; set; }
    public string FirstMidName { get; set; }
    public DateTime EnrollmentDate { get; set; }
}
```

The following code uses the `StudentVM` view model to create a new student:

```C#
[BindProperty]
public StudentVM StudentVM { get; set; }

public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    var entry = _context.Add(new Student());
    entry.CurrentValues.SetValues(StudentVM);
    await _context.SaveChangesAsync();
    return RedirectToPage("./Index");
}
```

The SetValues method sets the values of this object by reading values from another PropertyValues object. `SetValues` uses property name matching. The view model type:

- Doesn't need to be related to the model type.
- Needs to have properties that match.

Using `StudentVM` requires the Create page use `StudentVM` rather than `Student`:

```CSHTML
@page
@model CreateVMModel

@{
    ViewData["Title"] = "Create";
}

<h1>Create</h1>

<h4>Student</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger">
</div>
            <div class="form-group">
```

```
                <label asp-for="StudentVM.LastName" class="control-label">
</label>
                <input asp-for="StudentVM.LastName" class="form-control" />
                <span asp-validation-for="StudentVM.LastName" class="text-
danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="StudentVM.FirstMidName" class="control-
label"></label>
                <input asp-for="StudentVM.FirstMidName" class="form-control"
/>
                <span asp-validation-for="StudentVM.FirstMidName"
class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="StudentVM.EnrollmentDate" class="control-
label"></label>
                <input asp-for="StudentVM.EnrollmentDate" class="form-
control" />
                <span asp-validation-for="StudentVM.EnrollmentDate"
class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary"
/>
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-page="Index">Back to List</a>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Update the Edit page

In `Pages/Students/Edit.cshtml.cs`, replace the `OnGetAsync` and `OnPostAsync` methods with the following code.

```C#
public async Task<IActionResult> OnGetAsync(int? id)
{
    if (id == null)
    {
        return NotFound();
    }
```

```
        Student = await _context.Students.FindAsync(id);

        if (Student == null)
        {
            return NotFound();
        }
        return Page();
    }

    public async Task<IActionResult> OnPostAsync(int id)
    {
        var studentToUpdate = await _context.Students.FindAsync(id);

        if (studentToUpdate == null)
        {
            return NotFound();
        }

        if (await TryUpdateModelAsync<Student>(
            studentToUpdate,
            "student",
            s => s.FirstMidName, s => s.LastName, s => s.EnrollmentDate))
        {
            await _context.SaveChangesAsync();
            return RedirectToPage("./Index");
        }

        return Page();
    }
```

The code changes are similar to the Create page with a few exceptions:

- `FirstOrDefaultAsync` has been replaced with FindAsync. When you don't have to include related data, `FindAsync` is more efficient.
- `OnPostAsync` has an `id` parameter.
- The current student is fetched from the database, rather than creating an empty student.

Run the app, and test it by creating and editing a student.

Entity States

The database context keeps track of whether entities in memory are in sync with their corresponding rows in the database. This tracking information determines what happens when SaveChangesAsync is called. For example, when a new entity is passed to the AddAsync method, that entity's state is set to Added. When `SaveChangesAsync` is called, the database context issues a SQL `INSERT` command.

An entity may be in one of the following states:

- `Added`: The entity doesn't yet exist in the database. The `SaveChanges` method issues an `INSERT` statement.
- `Unchanged`: No changes need to be saved with this entity. An entity has this status when it's read from the database.
- `Modified`: Some or all of the entity's property values have been modified. The `SaveChanges` method issues an `UPDATE` statement.
- `Deleted`: The entity has been marked for deletion. The `SaveChanges` method issues a `DELETE` statement.
- `Detached`: The entity isn't being tracked by the database context.

In a desktop app, state changes are typically set automatically. An entity is read, changes are made, and the entity state is automatically changed to `Modified`. Calling `SaveChanges` generates a SQL `UPDATE` statement that updates only the changed properties.

In a web app, the `DbContext` that reads an entity and displays the data is disposed after a page is rendered. When a page's `OnPostAsync` method is called, a new web request is made and with a new instance of the `DbContext`. Rereading the entity in that new context simulates desktop processing.

Update the Delete page

In this section, a custom error message is implemented when the call to `SaveChanges` fails.

Replace the code in `Pages/Students/Delete.cshtml.cs` with the following code:

C#

```
using ContosoUniversity.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;
using System;
using System.Threading.Tasks;

namespace ContosoUniversity.Pages.Students
{
    public class DeleteModel : PageModel
    {
        private readonly ContosoUniversity.Data.SchoolContext _context;
        private readonly ILogger<DeleteModel> _logger;

        public DeleteModel(ContosoUniversity.Data.SchoolContext context,
                           ILogger<DeleteModel> logger)
        {
            _context = context;
            _logger = logger;
        }

        [BindProperty]
        public Student Student { get; set; }
        public string ErrorMessage { get; set; }

        public async Task<IActionResult> OnGetAsync(int? id, bool?
saveChangesError = false)
        {
            if (id == null)
            {
                return NotFound();
            }

            Student = await _context.Students
                .AsNoTracking()
                .FirstOrDefaultAsync(m => m.ID == id);

            if (Student == null)
            {
                return NotFound();
            }

            if (saveChangesError.GetValueOrDefault())
            {
                ErrorMessage = String.Format("Delete {ID} failed. Try
again", id);
            }

            return Page();
        }

        public async Task<IActionResult> OnPostAsync(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var student = await _context.Students.FindAsync(id);

            if (student == null)
            {
                return NotFound();
            }
            try
            {
                _context.Students.Remove(student);
                await _context.SaveChangesAsync();
                return RedirectToPage("./Index");
            }
            catch (DbUpdateException ex)
            {
                _logger.LogError(ex, ErrorMessage);

                return RedirectToAction("./Delete",
                                     new { id, saveChangesError = true });
            }
        }
    }
}
```

The preceding code:

- Adds Logging.
- Adds the optional parameter `saveChangesError` to the `OnGetAsync` method signature. `saveChangesError` indicates whether the method was called after a failure to delete the student object.

The delete operation might fail because of transient network problems. Transient network errors are more likely when the database is in the cloud. The `saveChangesError` parameter is `false` when the Delete page `OnGetAsync` is called from the UI. When `OnGetAsync` is called by `OnPostAsync` because the delete operation failed, the `saveChangesError` parameter is `true`.

The `OnPostAsync` method retrieves the selected entity, then calls the Remove method to set the entity's status to `Deleted`. When `SaveChanges` is called, a SQL `DELETE` command is generated. If `Remove` fails:

- The database exception is caught.
- The Delete pages `OnGetAsync` method is called with `saveChangesError=true`.

Add an error message to `Pages/Students/Delete.cshtml`:

CSHTML

```
@page
@model ContosoUniversity.Pages.Students.DeleteModel

@{
    ViewData["Title"] = "Delete";
}
```

```
<h1>Delete</h1>

<p class="text-danger">@Model.ErrorMessage</p>

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Student</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Student.LastName)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Student.LastName)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Student.FirstMidName)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Student.FirstMidName)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Student.EnrollmentDate)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Student.EnrollmentDate)
        </dd>
    </dl>

    <form method="post">
        <input type="hidden" asp-for="Student.ID" />
        <input type="submit" value="Delete" class="btn btn-danger" /> |
        <a asp-page="./Index">Back to List</a>
    </form>
</div>
```

Run the app and delete a student to test the Delete page.

Next steps

Previous tutorial Next tutorial

Part 3, Razor Pages with EF Core in ASP.NET Core - Sort, Filter, Paging

Article • 04/10/2024

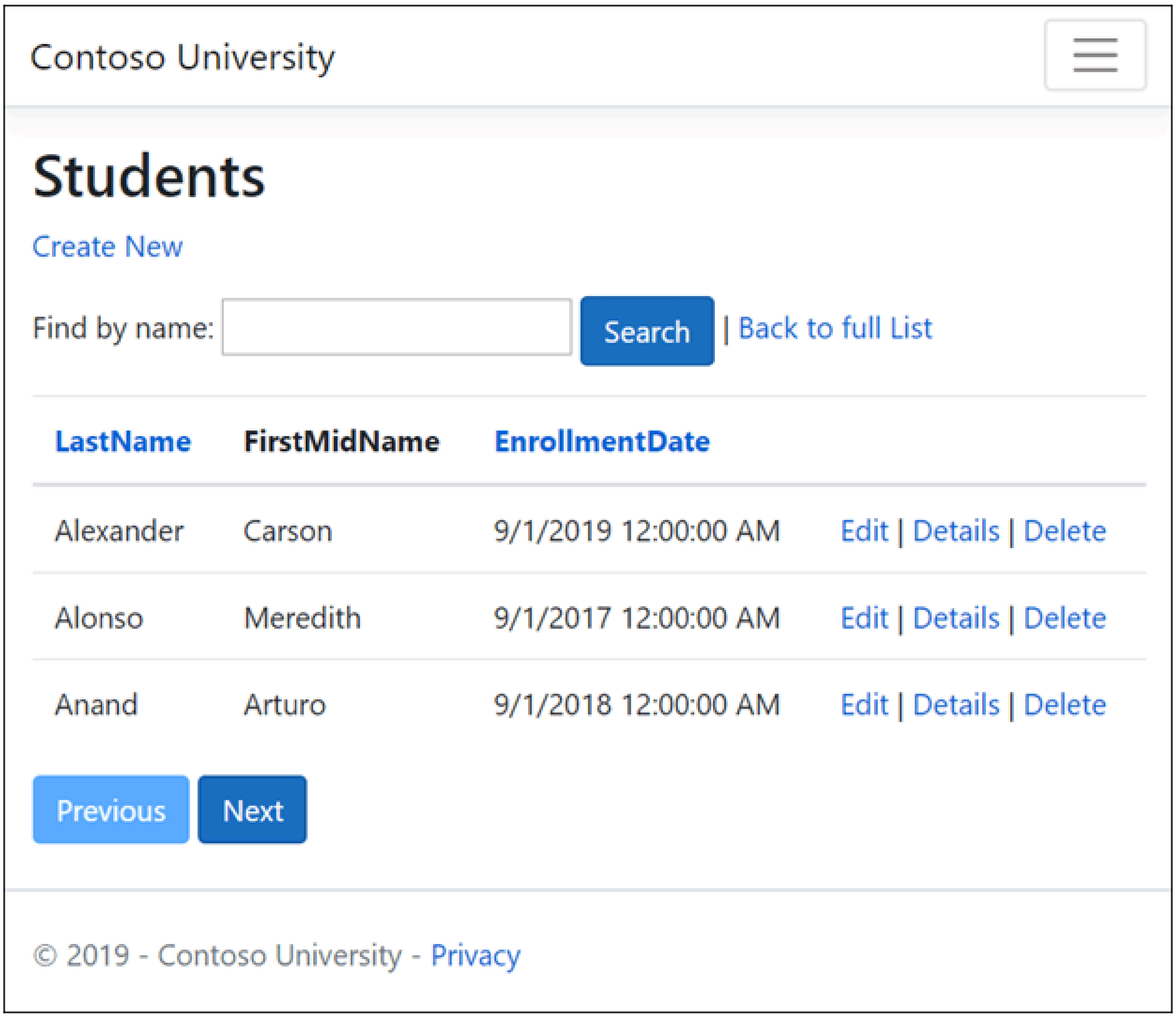
By Tom Dykstra, Jeremy Likness, and Jon P Smith

The Contoso University web app demonstrates how to create Razor Pages web apps using EF Core and Visual Studio. For information about the tutorial series, see the first tutorial.

If you run into problems you can't solve, download the completed app and compare that code to what you created by following the tutorial.

This tutorial adds sorting, filtering, and paging functionality to the Students pages.

The following illustration shows a completed page. The column headings are clickable links to sort the column. Click a column heading repeatedly to switch between ascending and descending sort order.


Contoso University

Students

Create New

Find by name: Search | Back to full List

LastName	FirstMidName	EnrollmentDate	
Alexander	Carson	9/1/2019 12:00:00 AM	Edit | Details | Delete
Alonso	Meredith	9/1/2017 12:00:00 AM	Edit | Details | Delete
Anand	Arturo	9/1/2018 12:00:00 AM	Edit | Details | Delete

Previous Next

© 2019 - Contoso University - Privacy

Add sorting

Replace the code in `Pages/Students/Index.cshtml.cs` with the following code to add sorting.

C#

```
public class IndexModel : PageModel
{
    private readonly SchoolContext _context;
    public IndexModel(SchoolContext context)
    {
        _context = context;
    }

    public string NameSort { get; set; }
    public string DateSort { get; set; }
    public string CurrentFilter { get; set; }
    public string CurrentSort { get; set; }

    public IList<Student> Students { get; set; }

    public async Task OnGetAsync(string sortOrder)
    {
        // using System;
        NameSort = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
        DateSort = sortOrder == "Date" ? "date_desc" : "Date";

        IQueryable<Student> studentsIQ = from s in _context.Students
                                          select s;

        switch (sortOrder)
        {
            case "name_desc":
                studentsIQ = studentsIQ.OrderByDescending(s => s.LastName);
                break;
            case "Date":
                studentsIQ = studentsIQ.OrderBy(s => s.EnrollmentDate);
                break;
            case "date_desc":
                studentsIQ = studentsIQ.OrderByDescending(s =>
s.EnrollmentDate);
                break;
            default:
                studentsIQ = studentsIQ.OrderBy(s => s.LastName);
                break;
        }

        Students = await studentsIQ.AsNoTracking().ToListAsync();
    }
}
```

The preceding code:

- Requires adding `using System;`.
- Adds properties to contain the sorting parameters.
- Changes the name of the `Student` property to `Students`.
- Replaces the code in the `OnGetAsync` method.

The `OnGetAsync` method receives a `sortOrder` parameter from the query string in the URL. The URL and query string is generated by the Anchor Tag Helper.

The `sortOrder` parameter is either `Name` or `Date`. The `sortOrder` parameter is optionally followed by `_desc` to specify descending order. The default sort order is ascending.

When the Index page is requested from the **Students** link, there's no query string. The students are displayed in ascending order by last name. Ascending order by last name is the `default` in the `switch` statement. When the user clicks a column heading link, the appropriate `sortOrder` value is provided in the query string value.

`NameSort` and `DateSort` are used by the Razor Page to configure the column heading hyperlinks with the appropriate query string values:

C#

```
NameSort = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
DateSort = sortOrder == "Date" ? "date_desc" : "Date";
```

The code uses the C# conditional operator ?:. The `?:` operator is a ternary operator, it takes three operands. The first line specifies that when `sortOrder` is null or empty, `NameSort` is set to `name_desc`. If `sortOrder` is ***not*** null or empty, `NameSort` is set to an empty string.

These two statements enable the page to set the column heading hyperlinks as follows:

Expand table

Current sort order	Last Name Hyperlink	Date Hyperlink
Last Name ascending	descending	ascending
Last Name descending	ascending	ascending
Date ascending	ascending	descending
Date descending	ascending	ascending

The method uses LINQ to Entities to specify the column to sort by. The code initializes an `IQueryable<Student>` before the switch statement, and modifies it in the switch statement:

```C#
IQueryable<Student> studentsIQ = from s in _context.Students
                                select s;

switch (sortOrder)
{
    case "name_desc":
        studentsIQ = studentsIQ.OrderByDescending(s => s.LastName);
        break;
    case "Date":
        studentsIQ = studentsIQ.OrderBy(s => s.EnrollmentDate);
        break;
    case "date_desc":
        studentsIQ = studentsIQ.OrderByDescending(s => s.EnrollmentDate);
        break;
    default:
        studentsIQ = studentsIQ.OrderBy(s => s.LastName);
        break;
}

Students = await studentsIQ.AsNoTracking().ToListAsync();
```

When an `IQueryable` is created or modified, no query is sent to the database. The query isn't executed until the `IQueryable` object is converted into a collection. `IQueryable` are converted to a collection by calling a method such as `ToListAsync`. Therefore, the `IQueryable` code results in a single query that's not executed until the following statement:

```C#
Students = await studentsIQ.AsNoTracking().ToListAsync();
```

`OnGetAsync` could get verbose with a large number of sortable columns. For information about an alternative way to code this functionality, see Use dynamic LINQ to simplify code in the MVC version of this tutorial series.

Add column heading hyperlinks to the Student Index page

Replace the code in `Students/Index.cshtml`, with the following code. The changes are highlighted.

CSHTML

```
@page
@model ContosoUniversity.Pages.Students.IndexModel

@{
    ViewData["Title"] = "Students";
}

<h2>Students</h2>
<p>
    <a asp-page="Create">Create New</a>
</p>

<table class="table">
    <thead>
        <tr>
            <th>
                <a asp-page="./Index" asp-route-sortOrder="@Model.NameSort">
                    @Html.DisplayNameFor(model =>
model.Students[0].LastName)
                </a>
            </th>
            <th>
                @Html.DisplayNameFor(model =>
model.Students[0].FirstMidName)
            </th>
            <th>
                <a asp-page="./Index" asp-route-sortOrder="@Model.DateSort">
                    @Html.DisplayNameFor(model =>
model.Students[0].EnrollmentDate)
                </a>
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.Students)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.LastName)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.FirstMidName)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.EnrollmentDate)
                </td>
                <td>
                    <a asp-page="./Edit" asp-route-id="@item.ID">Edit</a> |
                    <a asp-page="./Details" asp-route-
id="@item.ID">Details</a> |
                    <a asp-page="./Delete" asp-route-
id="@item.ID">Delete</a>
                </td>
            </tr>
        }
    </tbody>
</table>
```

The preceding code:

- Adds hyperlinks to the `LastName` and `EnrollmentDate` column headings.
- Uses the information in `NameSort` and `DateSort` to set up hyperlinks with the current sort order values.
- Changes the page heading from Index to Students.
- Changes `Model.Student` to `Model.Students`.

To verify that sorting works:

- Run the app and select the **Students** tab.
- Click the column headings.

Add filtering

To add filtering to the Students Index page:

- A text box and a submit button is added to the Razor Page. The text box supplies a search string on the first or last name.
- The page model is updated to use the text box value.

Update the OnGetAsync method

Replace the code in `Students/Index.cshtml.cs` with the following code to add filtering:

```C#
public class IndexModel : PageModel
{
    private readonly SchoolContext _context;

    public IndexModel(SchoolContext context)
    {
        _context = context;
    }
    public string NameSort { get; set; }
    public string DateSort { get; set; }
    public string CurrentFilter { get; set; }
    public string CurrentSort { get; set; }

    public IList<Student> Students { get; set; }

    public async Task OnGetAsync(string sortOrder, string searchString)
    {
        NameSort = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
        DateSort = sortOrder == "Date" ? "date_desc" : "Date";

        CurrentFilter = searchString;

        IQueryable<Student> studentsIQ = from s in _context.Students
                                        select s;
        if (!String.IsNullOrEmpty(searchString))
        {
            studentsIQ = studentsIQ.Where(s =>
s.LastName.Contains(searchString)
                                   ||
s.FirstMidName.Contains(searchString));
        }

        switch (sortOrder)
        {
            case "name_desc":
                studentsIQ = studentsIQ.OrderByDescending(s => s.LastName);
                break;
            case "Date":
                studentsIQ = studentsIQ.OrderBy(s => s.EnrollmentDate);
                break;
            case "date_desc":
                studentsIQ = studentsIQ.OrderByDescending(s =>
s.EnrollmentDate);
                break;
            default:
                studentsIQ = studentsIQ.OrderBy(s => s.LastName);
                break;
        }

        Students = await studentsIQ.AsNoTracking().ToListAsync();
    }
}
```

The preceding code:

- Adds the `searchString` parameter to the `OnGetAsync` method, and saves the parameter value in the `CurrentFilter` property. The search string value is received from a text box that's added in the next section.
- Adds to the LINQ statement a `Where` clause. The `Where` clause selects only students whose first name or last name contains the search string. The LINQ statement is

executed only if there's a value to search for.

IQueryable vs. IEnumerable

The code calls the Where method on an `IQueryable` object, and the filter is processed on the server. In some scenarios, the app might be calling the `Where` method as an extension method on an in-memory collection. For example, suppose `_context.Students` changes from EF Core `DbSet` to a repository method that returns an `IEnumerable` collection. The result would normally be the same but in some cases may be different.

For example, the .NET Framework implementation of `Contains` performs a case-sensitive comparison by default. In SQL Server, `Contains` case-sensitivity is determined by the collation setting of the SQL Server instance. SQL Server defaults to case-insensitive. SQLite defaults to case-sensitive. `ToUpper` could be called to make the test explicitly case-insensitive:

C#

```
Where(s => s.LastName.ToUpper().Contains(searchString.ToUpper())`
```

The preceding code would ensure that the filter is case-insensitive even if the `Where` method is called on an `IEnumerable` or runs on SQLite.

When `Contains` is called on an `IEnumerable` collection, the .NET Core implementation is used. When `Contains` is called on an `IQueryable` object, the database implementation is used.

Calling `Contains` on an `IQueryable` is usually preferable for performance reasons. With `IQueryable`, the filtering is done by the database server. If an `IEnumerable` is created first, all the rows have to be returned from the database server.

There's a performance penalty for calling `ToUpper`. The `ToUpper` code adds a function in the WHERE clause of the TSQL SELECT statement. The added function prevents the optimizer from using an index. Given that SQL is installed as case-insensitive, it's best to avoid the `ToUpper` call when it's not needed.

For more information, see How to use case-insensitive query with Sqlite provider.

Update the Razor page

Replace the code in `Pages/Students/Index.cshtml` to add a **Search** button.

CSHTML

```
@page
@model ContosoUniversity.Pages.Students.IndexModel

@{
    ViewData["Title"] = "Students";
}

<h2>Students</h2>

<p>
    <a asp-page="Create">Create New</a>
</p>

<form asp-page="./Index" method="get">
    <div class="form-actions no-color">
        <p>
            Find by name:
            <input type="text" name="SearchString"
value="@Model.CurrentFilter" />
            <input type="submit" value="Search" class="btn btn-primary" /> |
            <a asp-page="./Index">Back to full List</a>
        </p>
    </div>
</form>

<table class="table">
    <thead>
        <tr>
            <th>
                <a asp-page="./Index" asp-route-sortOrder="@Model.NameSort">
                    @Html.DisplayNameFor(model =>
model.Students[0].LastName)
                </a>
            </th>
            <th>
                @Html.DisplayNameFor(model =>
model.Students[0].FirstMidName)
            </th>
            <th>
                <a asp-page="./Index" asp-route-sortOrder="@Model.DateSort">
                    @Html.DisplayNameFor(model =>
model.Students[0].EnrollmentDate)
                </a>
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.Students)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.LastName)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.FirstMidName)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.EnrollmentDate)
                </td>
                <td>
                    <a asp-page="./Edit" asp-route-id="@item.ID">Edit</a> |
                    <a asp-page="./Details" asp-route-
id="@item.ID">Details</a> |
                    <a asp-page="./Delete" asp-route-
id="@item.ID">Delete</a>
                </td>
            </tr>
        }
    </tbody>
</table>
```

The preceding code uses the `<form>` tag helper to add the search text box and button. By default, the `<form>` tag helper submits form data with a POST. With POST, the parameters are passed in the HTTP message body and not in the URL. When HTTP GET is used, the form data is passed in the URL as query strings. Passing the data with query strings enables users to bookmark the URL. The W3C guidelines recommend that GET should be used when the action doesn't result in an update.

Test the app:

- Select the **Students** tab and enter a search string. If you're using SQLite, the filter is case-insensitive only if you implemented the optional `ToUpper` code shown earlier.
- Select **Search**.

Notice that the URL contains the search string. For example:

browser-address-bar

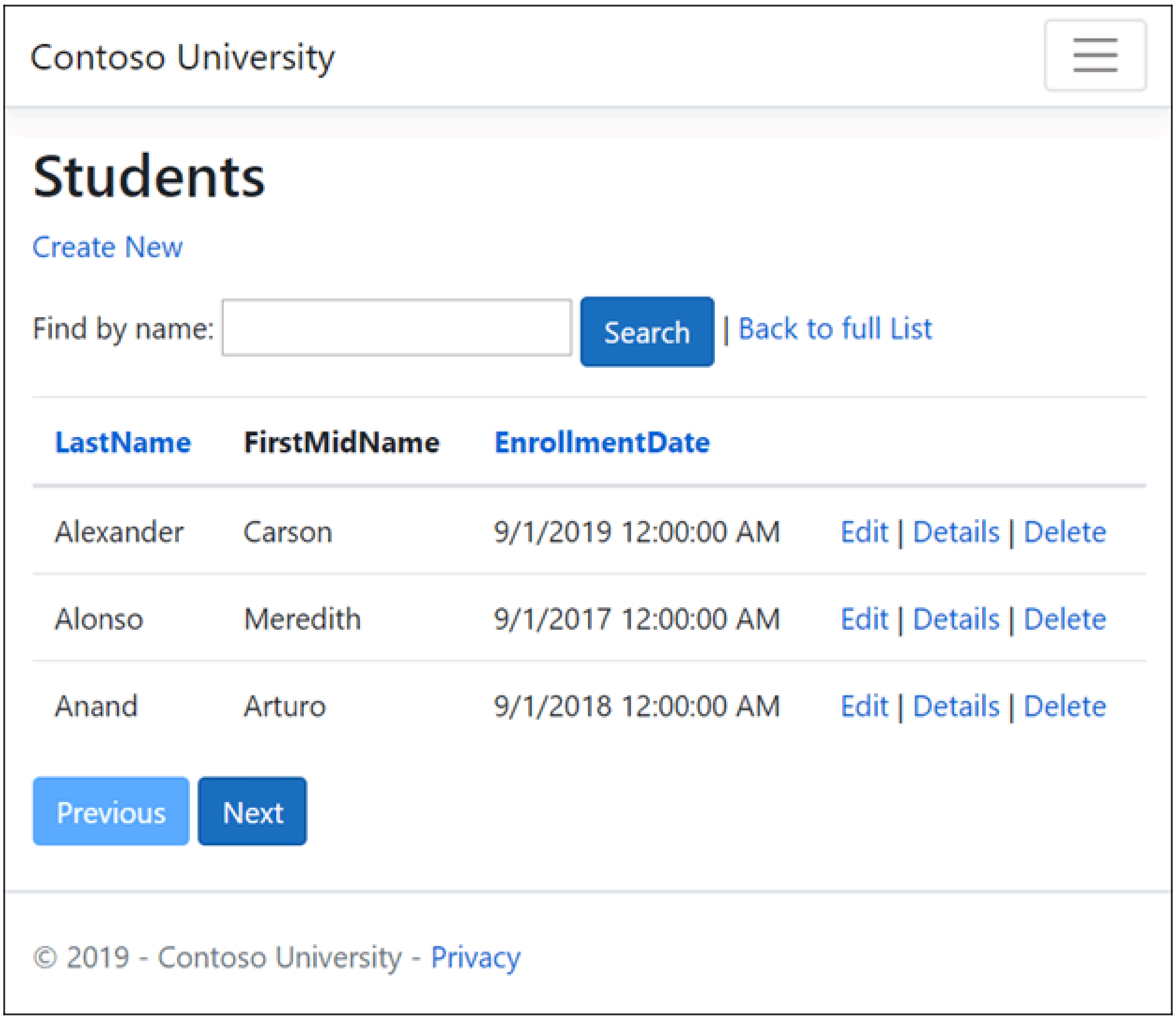
```
https://localhost:5001/Students?SearchString=an
```

If the page is bookmarked, the bookmark contains the URL to the page and the `SearchString` query string. The `method="get"` in the `form` tag is what caused the query string to be generated.

Currently, when a column heading sort link is selected, the filter value from the **Search** box is lost. The lost filter value is fixed in the next section.

Add paging

In this section, a `PaginatedList` class is created to support paging. The `PaginatedList` class uses `Skip` and `Take` statements to filter data on the server instead of retrieving all rows of the table. The following illustration shows the paging buttons.


Contoso University

Students

Create New

Find by name: Search | Back to full List

LastName	FirstMidName	EnrollmentDate	
Alexander	Carson	9/1/2019 12:00:00 AM	Edit | Details | Delete
Alonso	Meredith	9/1/2017 12:00:00 AM	Edit | Details | Delete
Anand	Arturo	9/1/2018 12:00:00 AM	Edit | Details | Delete

© 2019 - Contoso University - Privacy

Create the PaginatedList class

In the project folder, create `PaginatedList.cs` with the following code:

```C#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;

namespace ContosoUniversity
{
    public class PaginatedList<T> : List<T>
    {
        public int PageIndex { get; private set; }
        public int TotalPages { get; private set; }

        public PaginatedList(List<T> items, int count, int pageIndex, int 
pageSize)
        {
            PageIndex = pageIndex;
            TotalPages = (int)Math.Ceiling(count / (double)pageSize);

            this.AddRange(items);
        }

        public bool HasPreviousPage => PageIndex > 1;

        public bool HasNextPage => PageIndex < TotalPages;

        public static async Task<PaginatedList<T>> CreateAsync(
            IQueryable<T> source, int pageIndex, int pageSize)
        {
            var count = await source.CountAsync();
            var items = await source.Skip(
                (pageIndex - 1) * pageSize)
                .Take(pageSize).ToListAsync();
            return new PaginatedList<T>(items, count, pageIndex, pageSize);
        }
    }
}
```

The `CreateAsync` method in the preceding code takes page size and page number and applies the appropriate `Skip` and `Take` statements to the `IQueryable`. When `ToListAsync` is called on the `IQueryable`, it returns a List containing only the requested page. The properties `HasPreviousPage` and `HasNextPage` are used to enable or disable **Previous** and **Next** paging buttons.

The `CreateAsync` method is used to create the `PaginatedList<T>`. A constructor can't create the `PaginatedList<T>` object; constructors can't run asynchronous code.

Add page size to configuration

Add `PageSize` to the `appsettings.json` Configuration file:

JSON

```json
{
  "PageSize": 3,
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "SchoolContext": "Server=(localdb)\\mssqllocaldb;Database=CU-1;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

Add paging to IndexModel

Replace the code in `Students/Index.cshtml.cs` to add paging.

```C#
using ContosoUniversity.Data;
using ContosoUniversity.Models;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using System;
using System.Linq;
using System.Threading.Tasks;

namespace ContosoUniversity.Pages.Students
{
    public class IndexModel : PageModel
    {
        private readonly SchoolContext _context;
        private readonly IConfiguration Configuration;

        public IndexModel(SchoolContext context, IConfiguration configuration)
        {
            _context = context;
            Configuration = configuration;
        }

        public string NameSort { get; set; }
        public string DateSort { get; set; }
        public string CurrentFilter { get; set; }
        public string CurrentSort { get; set; }
        public PaginatedList<Student> Students { get; set; }

        public async Task OnGetAsync(string sortOrder,
            string currentFilter, string searchString, int? pageIndex)
        {
            CurrentSort = sortOrder;
            NameSort = String.IsNullOrEmpty(sortOrder) ? "name_desc" : "";
            DateSort = sortOrder == "Date" ? "date_desc" : "Date";
            if (searchString != null)
            {
                pageIndex = 1;
            }
            else
            {
                searchString = currentFilter;
            }

            CurrentFilter = searchString;

            IQueryable<Student> studentsIQ = from s in _context.Students
                                             select s;
            if (!String.IsNullOrEmpty(searchString))
            {
                studentsIQ = studentsIQ.Where(s =>
s.LastName.Contains(searchString)
                                       ||
s.FirstMidName.Contains(searchString));
            }
            switch (sortOrder)
            {
                case "name_desc":
                    studentsIQ = studentsIQ.OrderByDescending(s =>
s.LastName);
                    break;
                case "Date":
                    studentsIQ = studentsIQ.OrderBy(s => s.EnrollmentDate);
                    break;
                case "date_desc":
                    studentsIQ = studentsIQ.OrderByDescending(s =>
s.EnrollmentDate);
                    break;
                default:
                    studentsIQ = studentsIQ.OrderBy(s => s.LastName);
                    break;
            }

            var pageSize = Configuration.GetValue("PageSize", 4);
            Students = await PaginatedList<Student>.CreateAsync(
                studentsIQ.AsNoTracking(), pageIndex ?? 1, pageSize);
        }
    }
}
```

The preceding code:

- Changes the type of the `Students` property from `IList<Student>` to `PaginatedList<Student>`.
- Adds the page index, the current `sortOrder`, and the `currentFilter` to the `OnGetAsync` method signature.
- Saves the sort order in the `CurrentSort` property.
- Resets page index to 1 when there's a new search string.
- Uses the `PaginatedList` class to get Student entities.
- Sets `pageSize` to 3 from Configuration, 4 if configuration fails.

All the parameters that `OnGetAsync` receives are null when:

- The page is called from the **Students** link.
- The user hasn't clicked a paging or sorting link.

When a paging link is clicked, the page index variable contains the page number to display.

The `CurrentSort` property provides the Razor Page with the current sort order. The current sort order must be included in the paging links to keep the sort order while paging.

The `CurrentFilter` property provides the Razor Page with the current filter string. The `CurrentFilter` value:

- Must be included in the paging links in order to maintain the filter settings during paging.
- Must be restored to the text box when the page is redisplayed.

If the search string is changed while paging, the page is reset to 1. The page has to be reset to 1 because the new filter can result in different data to display. When a search value is entered and **Submit** is selected:

- The search string is changed.
- The `searchString` parameter isn't null.

The `PaginatedList.CreateAsync` method converts the student query to a single page of students in a collection type that supports paging. That single page of students is passed to the Razor Page.

The two question marks after `pageIndex` in the `PaginatedList.CreateAsync` call represent the null-coalescing operator. The null-coalescing operator defines a default value for a nullable type. The expression `pageIndex ?? 1` returns the value of `pageIndex` if it has a value, otherwise, it returns 1.

Add paging links

Replace the code in `Students/Index.cshtml` with the following code. The changes are highlighted:

CSHTML

```
@page
@model ContosoUniversity.Pages.Students.IndexModel

@{
    ViewData["Title"] = "Students";
}

<h2>Students</h2>

<p>
    <a asp-page="Create">Create New</a>
</p>

<form asp-page="./Index" method="get">
    <div class="form-actions no-color">
        <p>
            Find by name:
            <input type="text" name="SearchString"
value="@Model.CurrentFilter" />
            <input type="submit" value="Search" class="btn btn-primary" /> |
            <a asp-page="./Index">Back to full List</a>
        </p>
    </div>
</form>

<table class="table">
    <thead>
        <tr>
            <th>
                <a asp-page="./Index" asp-route-sortOrder="@Model.NameSort"
                   asp-route-currentFilter="@Model.CurrentFilter">
                    @Html.DisplayNameFor(model =>
model.Students[0].LastName)
                </a>
            </th>
            <th>
                @Html.DisplayNameFor(model =>
model.Students[0].FirstMidName)
            </th>
            <th>
                <a asp-page="./Index" asp-route-sortOrder="@Model.DateSort"
                   asp-route-currentFilter="@Model.CurrentFilter">
                    @Html.DisplayNameFor(model =>
model.Students[0].EnrollmentDate)
                </a>
            </th>
```

```
                <th></th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Model.Students)
            {
                <tr>
                    <td>
                        @Html.DisplayFor(modelItem => item.LastName)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.FirstMidName)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.EnrollmentDate)
                    </td>
                    <td>
                        <a asp-page="./Edit" asp-route-id="@item.ID">Edit</a> |
                        <a asp-page="./Details" asp-route-
id="@item.ID">Details</a> |
                        <a asp-page="./Delete" asp-route-
id="@item.ID">Delete</a>
                    </td>
                </tr>
            }
        </tbody>
    </table>

    @{
        var prevDisabled = !Model.Students.HasPreviousPage ? "disabled" : "";
        var nextDisabled = !Model.Students.HasNextPage ? "disabled" : "";
    }

    <a asp-page="./Index"
       asp-route-sortOrder="@Model.CurrentSort"
       asp-route-pageIndex="@(Model.Students.PageIndex - 1)"
       asp-route-currentFilter="@Model.CurrentFilter"
       class="btn btn-primary @prevDisabled">
        Previous
    </a>
    <a asp-page="./Index"
       asp-route-sortOrder="@Model.CurrentSort"
       asp-route-pageIndex="@(Model.Students.PageIndex + 1)"
       asp-route-currentFilter="@Model.CurrentFilter"
       class="btn btn-primary @nextDisabled">
        Next
    </a>
```

The column header links use the query string to pass the current search string to the `OnGetAsync` method:

CSHTML

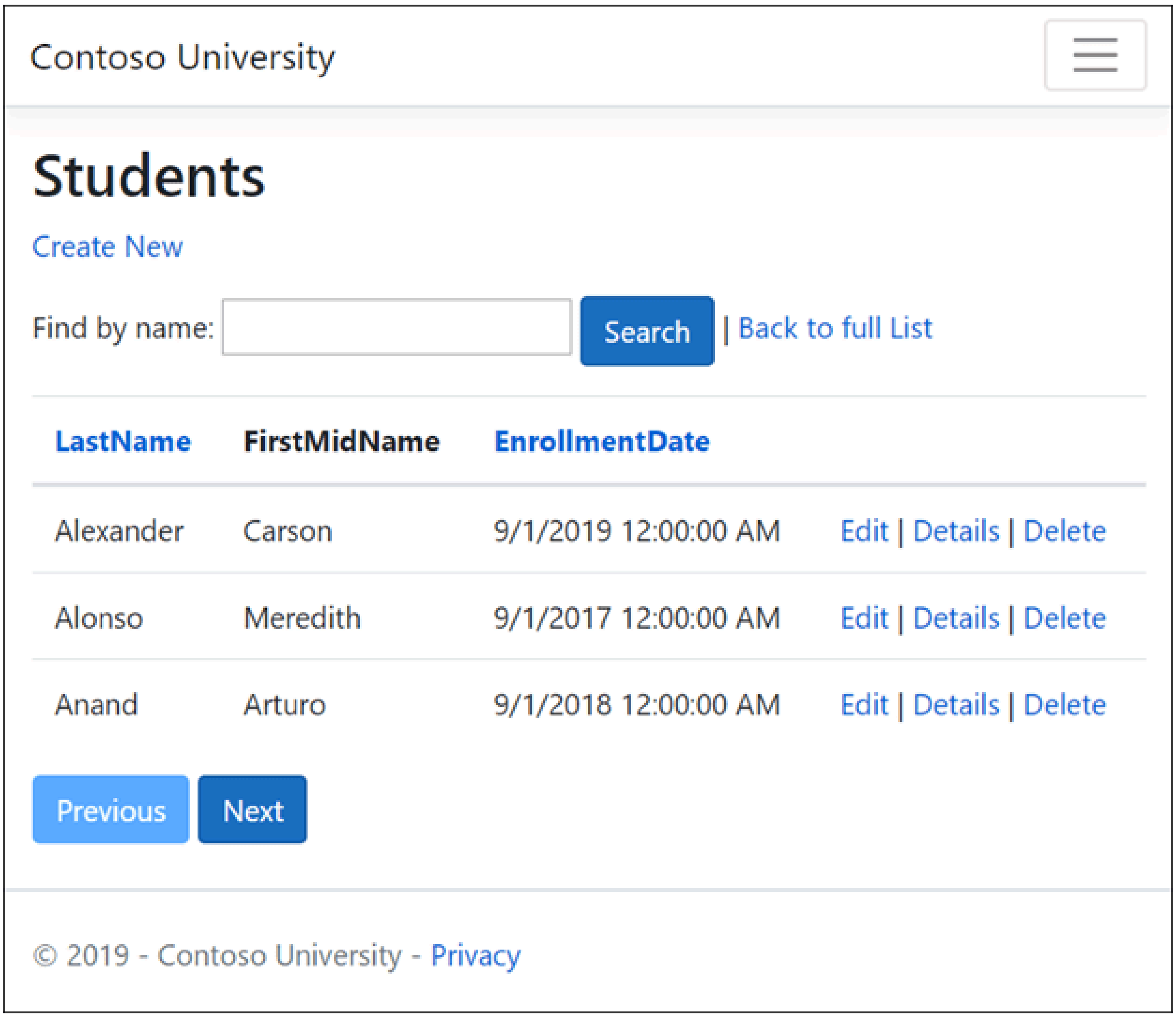
```
<a asp-page="./Index" asp-route-sortOrder="@Model.NameSort"
   asp-route-currentFilter="@Model.CurrentFilter">
    @Html.DisplayNameFor(model => model.Students[0].LastName)
</a>
```

The paging buttons are displayed by tag helpers:

```CSHTML

<a asp-page="./Index"
   asp-route-sortOrder="@Model.CurrentSort"
   asp-route-pageIndex="@(Model.Students.PageIndex - 1)"
   asp-route-currentFilter="@Model.CurrentFilter"
   class="btn btn-primary @prevDisabled">
    Previous
</a>
<a asp-page="./Index"
   asp-route-sortOrder="@Model.CurrentSort"
   asp-route-pageIndex="@(Model.Students.PageIndex + 1)"
   asp-route-currentFilter="@Model.CurrentFilter"
   class="btn btn-primary @nextDisabled">
    Next
</a>
```

Run the app and navigate to the students page.

- To make sure paging works, click the paging links in different sort orders.
- To verify that paging works correctly with sorting and filtering, enter a search string and try paging.


Contoso University

Students

Create New

Find by name: Search | Back to full List

LastName	FirstMidName	EnrollmentDate	
Alexander	Carson	9/1/2019 12:00:00 AM	Edit | Details | Delete
Alonso	Meredith	9/1/2017 12:00:00 AM	Edit | Details | Delete
Anand	Arturo	9/1/2018 12:00:00 AM	Edit | Details | Delete

Previous Next

© 2019 - Contoso University - Privacy

Grouping

This section creates an `About` page that displays how many students have enrolled for each enrollment date. The update uses grouping and includes the following steps:

- Create a view model for the data used by the `About` page.
- Update the `About` page to use the view model.

Create the view model

Create a *Models/SchoolViewModels* folder.

Create `SchoolViewModels/EnrollmentDateGroup.cs` with the following code:

```C#
using System;
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models.SchoolViewModels
```

```
{
    public class EnrollmentDateGroup
    {
        [DataType(DataType.Date)]
        public DateTime? EnrollmentDate { get; set; }

        public int StudentCount { get; set; }
    }
}
```

Create the Razor Page

Create a `Pages/About.cshtml` file with the following code:

CSHTML

```
@page
@model ContosoUniversity.Pages.AboutModel

@{
    ViewData["Title"] = "Student Body Statistics";
}

<h2>Student Body Statistics</h2>

<table>
    <tr>
        <th>
            Enrollment Date
        </th>
        <th>
            Students
        </th>
    </tr>

    @foreach (var item in Model.Students)
    {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.EnrollmentDate)
            </td>
            <td>
                @item.StudentCount
            </td>
        </tr>
    }
</table>
```

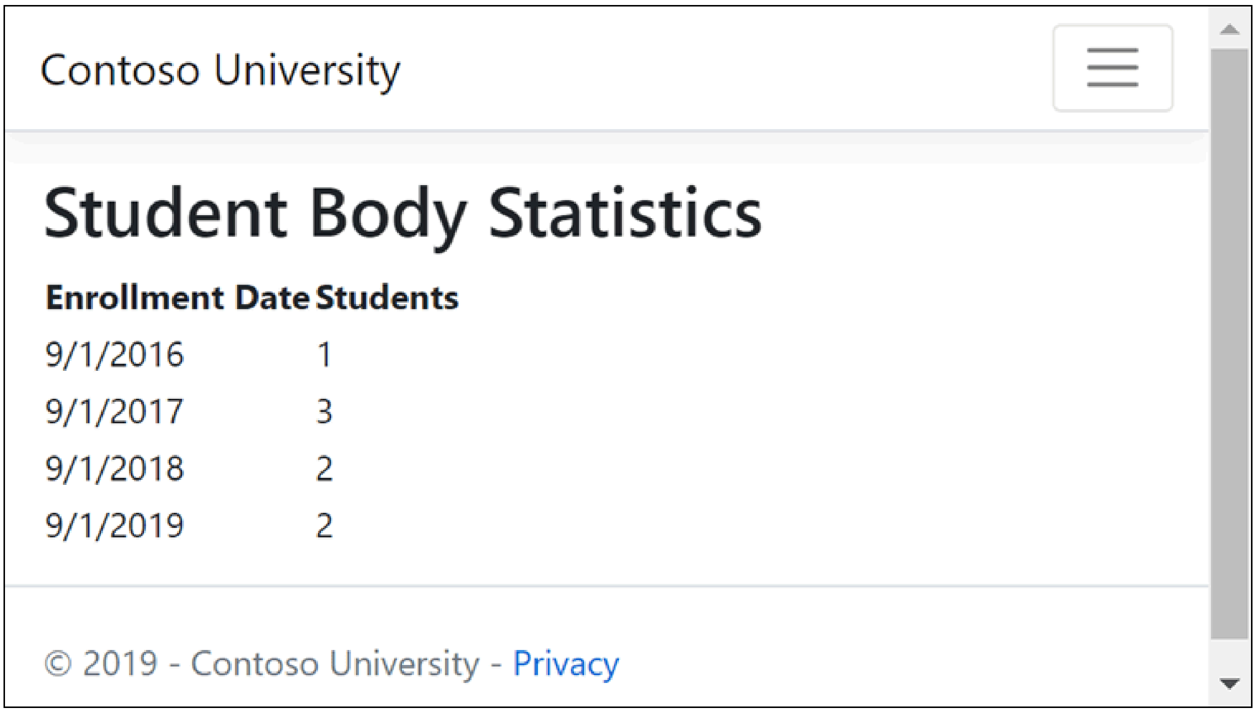
Create the page model

Update the `Pages/About.cshtml.cs` file with the following code:

```C#
using ContosoUniversity.Models.SchoolViewModels;
using ContosoUniversity.Data;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using ContosoUniversity.Models;

namespace ContosoUniversity.Pages
{
    public class AboutModel : PageModel
    {
        private readonly SchoolContext _context;

        public AboutModel(SchoolContext context)
        {
            _context = context;
        }

        public IList<EnrollmentDateGroup> Students { get; set; }

        public async Task OnGetAsync()
        {
            IQueryable<EnrollmentDateGroup> data =
                from student in _context.Students
                group student by student.EnrollmentDate into dateGroup
                select new EnrollmentDateGroup()
                {
                    EnrollmentDate = dateGroup.Key,
                    StudentCount = dateGroup.Count()
                };

            Students = await data.AsNoTracking().ToListAsync();
        }
    }
}
```

The LINQ statement groups the student entities by enrollment date, calculates the number of entities in each group, and stores the results in a collection of `EnrollmentDateGroup` view model objects.

Run the app and navigate to the About page. The count of students for each enrollment date is displayed in a table.



Contoso University

Student Body Statistics

Enrollment Date	Students
9/1/2016	1
9/1/2017	3
9/1/2018	2
9/1/2019	2

© 2019 - Contoso University - Privacy

Next steps

In the next tutorial, the app uses migrations to update the data model.

Previous tutorial Next tutorial

Part 4, Razor Pages with EF Core migrations in ASP.NET Core

Article • 05/31/2024

By Tom Dykstra, Jon P Smith, and Rick Anderson

The Contoso University web app demonstrates how to create Razor Pages web apps using EF Core and Visual Studio. For information about the tutorial series, see the first tutorial.

If you run into problems you can't solve, download the completed app and compare that code to what you created by following the tutorial.

This tutorial introduces the EF Core migrations feature for managing data model changes.

When a new app is developed, the data model changes frequently. Each time the model changes, the model gets out of sync with the database. This tutorial series started by configuring the Entity Framework to create the database if it doesn't exist. Each time the data model changes, the database needs to be dropped. The next time the app runs, the call to `EnsureCreated` re-creates the database to match the new data model. The `DbInitializer` class then runs to seed the new database.

This approach to keeping the DB in sync with the data model works well until the app needs to be deployed to production. When the app is running in production, it's usually storing data that needs to be maintained. The app can't start with a test DB each time a change is made (such as adding a new column). The EF Core Migrations feature solves this problem by enabling EF Core to update the DB schema instead of creating a new database.

Rather than dropping and recreating the database when the data model changes, migrations updates the schema and retains existing data.

ⓘ Note

SQLite limitations

This tutorial uses the Entity Framework Core migrations feature where possible. Migrations updates the database schema to match changes in the data model. However, migrations only does the kinds of changes that the database engine supports, and SQLite's schema change capabilities are limited. For example, adding a column is supported, but removing a column is not supported. If a migration is created to remove a column, the `ef migrations add` command succeeds but the `ef database update` command fails.

The workaround for the SQLite limitations is to manually write migrations code to perform a table rebuild when something in the table changes. The code goes in the `Up` and `Down` methods for a migration and involves:

- Creating a new table.
- Copying data from the old table to the new table.
- Dropping the old table.
- Renaming the new table.

Writing database-specific code of this type is outside the scope of this tutorial. Instead, this tutorial drops and re-creates the database whenever an attempt to apply a migration would fail. For more information, see the following resources:

- SQLite EF Core Database Provider Limitations
- Customize migration code
- Data seeding
- SQLite ALTER TABLE statement

Drop the database

Visual Studio

Use **SQL Server Object Explorer** (SSOX) to delete the database, or run the following command in the **Package Manager Console** (PMC):

PowerShell

```
Drop-Database
```

Create an initial migration

Visual Studio

Run the following commands in the PMC:

PowerShell

```
Add-Migration InitialCreate
Update-Database
```

Remove EnsureCreated

This tutorial series started by using EnsureCreated. `EnsureCreated` doesn't create a migrations history table and so can't be used with migrations. It's designed for testing or rapid prototyping where the database is dropped and re-created frequently.

From this point forward, the tutorials will use migrations.

In `Program.cs`, delete the following line:

C#

```
context.Database.EnsureCreated();
```

Run the app and verify that the database is seeded.

Up and Down methods

The EF Core `migrations add` command generated code to create the database. This migrations code is in the `Migrations\<timestamp>_InitialCreate.cs` file. The `Up` method of the `InitialCreate` class creates the database tables that correspond to the data model entity sets. The `Down` method deletes them, as shown in the following example:

C#

```
using System;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Migrations;

namespace ContosoUniversity.Migrations
{
    public partial class InitialCreate : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Course",
                columns: table => new
                {
                    CourseID = table.Column<int>(nullable: false),
                    Title = table.Column<string>(nullable: true),
                    Credits = table.Column<int>(nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Course", x => x.CourseID);
                });

            migrationBuilder.CreateTable(
                name: "Student",
                columns: table => new
                {
                    ID = table.Column<int>(nullable: false)
                        .Annotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn),
                    LastName = table.Column<string>(nullable: true),
                    FirstMidName = table.Column<string>(nullable: true),
                    EnrollmentDate = table.Column<DateTime>(nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Student", x => x.ID);
                });

            migrationBuilder.CreateTable(
                name: "Enrollment",
                columns: table => new
                {
                    EnrollmentID = table.Column<int>(nullable: false)
                        .Annotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn),
                    CourseID = table.Column<int>(nullable: false),
                    StudentID = table.Column<int>(nullable: false),
                    Grade = table.Column<int>(nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Enrollment", x => x.EnrollmentID);
                    table.ForeignKey(
                        name: "FK_Enrollment_Course_CourseID",
                        column: x => x.CourseID,
                        principalTable: "Course",
                        principalColumn: "CourseID",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_Enrollment_Student_StudentID",
                        column: x => x.StudentID,
                        principalTable: "Student",
                        principalColumn: "ID",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateIndex(
                name: "IX_Enrollment_CourseID",
                table: "Enrollment",
                column: "CourseID");

            migrationBuilder.CreateIndex(
                name: "IX_Enrollment_StudentID",
                table: "Enrollment",
                column: "StudentID");
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "Enrollment");

            migrationBuilder.DropTable(
                name: "Course");

            migrationBuilder.DropTable(
                name: "Student");
        }
    }
}
```

The preceding code is for the initial migration. The code:

- Was generated by the `migrations add InitialCreate` command.
- Is executed by the `database update` command.
- Creates a database for the data model specified by the database context class.

The migration name parameter (`InitialCreate` in the example) is used for the file name. The migration name can be any valid file name. It's best to choose a word or phrase that summarizes what is being done in the migration. For example, a migration that added a department table might be called "AddDepartmentTable."

The migrations history table

- Use SSOX or SQLite tool to inspect the database.
- Notice the addition of an `__EFMigrationsHistory` table. The `__EFMigrationsHistory` table keeps track of which migrations have been applied to the database.
- View the data in the `__EFMigrationsHistory` table. It shows one row for the first migration.

The data model snapshot

Migrations creates a *snapshot* of the current data model in `Migrations/SchoolContextModelSnapshot.cs`. When add a migration is added, EF determines what changed by comparing the current data model to the snapshot file.

Because the snapshot file tracks the state of the data model, a migration cannot be deleted by deleting the `<timestamp>_<migrationname>.cs` file. To back out the most recent migration, use the migrations remove command. `migrations remove` deletes the migration and ensures the snapshot is correctly reset. For more information, see dotnet ef migrations remove.

See Resetting all migrations to remove all migrations.

Applying migrations in production

We recommend that production apps **not** call Database.Migrate at application startup. `Migrate` shouldn't be called from an app that is deployed to a server farm. If the app is scaled out to multiple server instances, it's hard to ensure database schema updates don't happen from multiple servers or conflict with read/write access.

Database migration should be done as part of deployment, and in a controlled way. Production database migration approaches include:

- Using migrations to create SQL scripts and using the SQL scripts in deployment.
- Running `dotnet ef database update` from a controlled environment.

Troubleshooting

If the app uses SQL Server LocalDB and displays the following exception:

```text
SqlException: Cannot open database "ContosoUniversity" requested by the
login.
The login failed.
Login failed for user 'user name'.
```

The solution may be to run `dotnet ef database update` at a command prompt.

Additional resources

- EF Core CLI.
- dotnet ef migrations CLI commands

- Package Manager Console (Visual Studio)

Next steps

The next tutorial builds out the data model, adding entity properties and new entities.

Previous tutorial Next tutorial

Part 5, Razor Pages with EF Core in ASP.NET Core - Data Model

Article • 04/10/2024

By Tom Dykstra, Jeremy Likness, and Jon P Smith

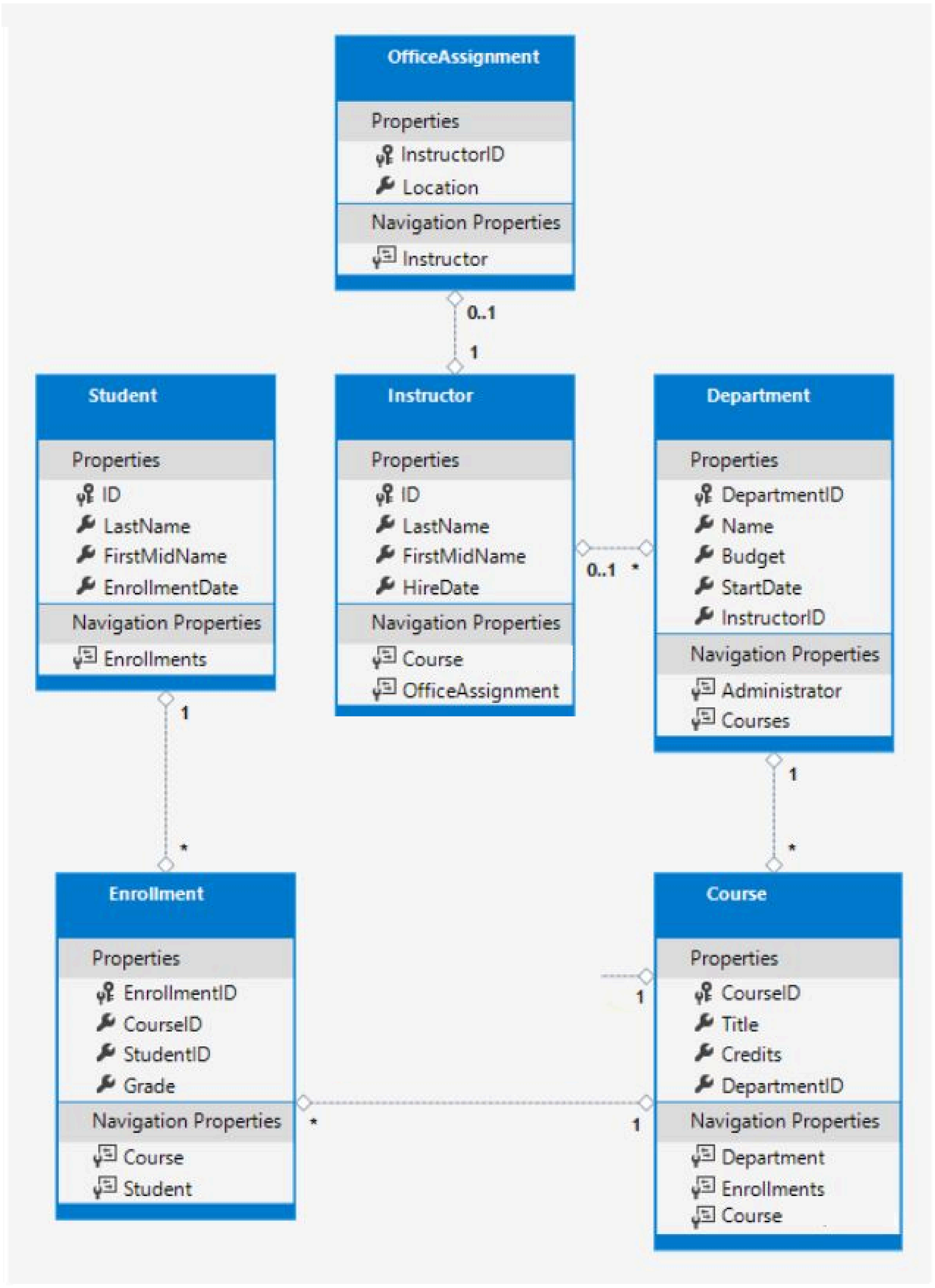
The Contoso University web app demonstrates how to create Razor Pages web apps using EF Core and Visual Studio. For information about the tutorial series, see the first tutorial.

If you run into problems you can't solve, download the completed app and compare that code to what you created by following the tutorial.

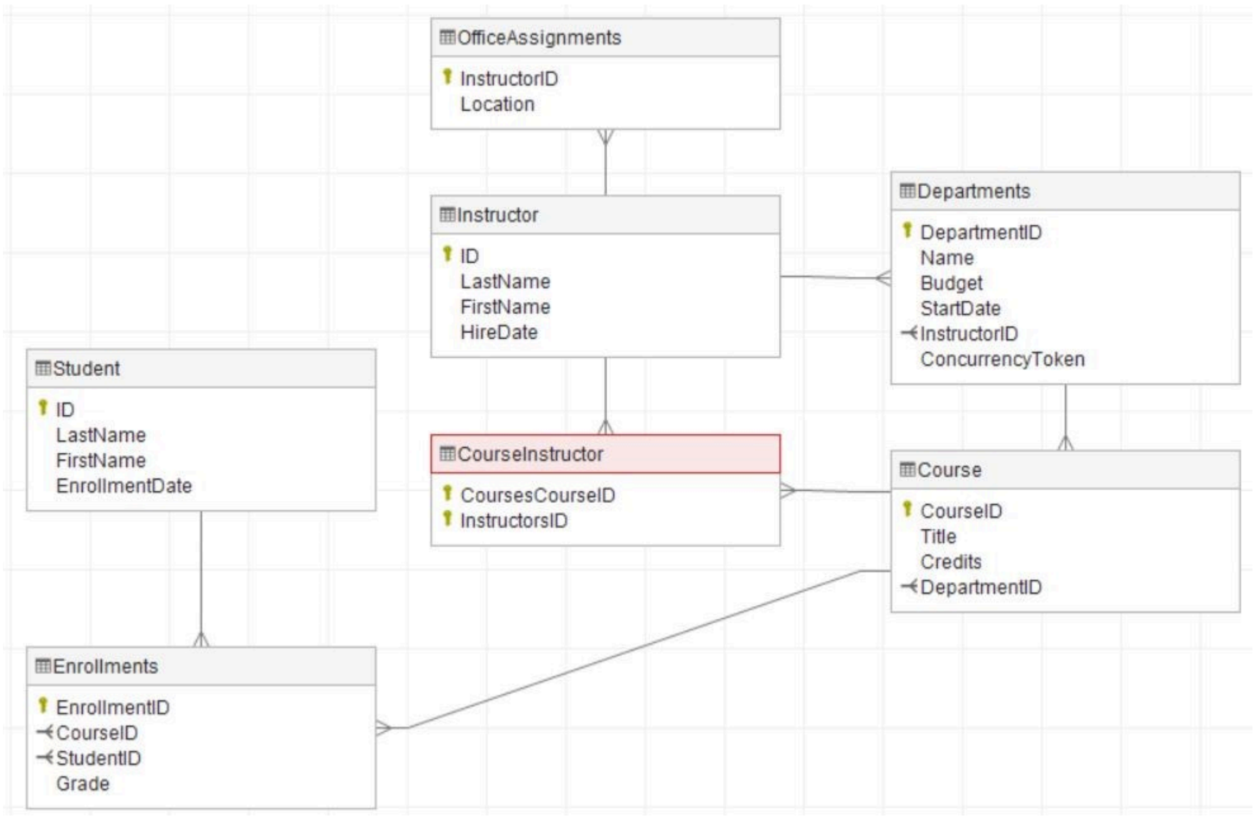
The previous tutorials worked with a basic data model that was composed of three entities. In this tutorial:

- More entities and relationships are added.
- The data model is customized by specifying formatting, validation, and database mapping rules.

The completed data model is shown in the following illustration:



The following database diagram was made with Dataedo:



To create a database diagram with Dataedo:

- Deploy the app to Azure
- Download and install Dataedo on your computer.
- Follow the instructions Generate documentation for Azure SQL Database in 5 minutes

In the preceding Dataedo diagram, the `CourseInstructor` is a join table created by Entity Framework. For more information, see Many-to-many

The Student entity

Replace the code in `Models/Student.cs` with the following code:

```C#
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Student
    {
        public int ID { get; set; }
        [Required]
        [StringLength(50)]
        [Display(Name = "Last Name")]
        public string LastName { get; set; }
        [Required]
        [StringLength(50, ErrorMessage = "First name cannot be longer than 
50 characters.")]
        [Column("FirstName")]
        [Display(Name = "First Name")]
        public string FirstMidName { get; set; }
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", 
ApplyFormatInEditMode = true)]
        [Display(Name = "Enrollment Date")]
        public DateTime EnrollmentDate { get; set; }
        [Display(Name = "Full Name")]
        public string FullName
        {
            get
            {
                return LastName + ", " + FirstMidName;
            }
        }

        public ICollection<Enrollment> Enrollments { get; set; }
    }
}
```

The preceding code adds a `FullName` property and adds the following attributes to existing properties:

- [DataType]
- [DisplayFormat]
- [StringLength]
- [Column]
- [Required]
- [Display]

The FullName calculated property

`FullName` is a calculated property that returns a value that's created by concatenating two other properties. `FullName` can't be set, so it has only a get accessor. No `FullName` column is created in the database.

The DataType attribute

C#

```
[DataType(DataType.Date)]
```

For student enrollment dates, all of the pages currently display the time of day along with the date, although only the date is relevant. By using data annotation attributes, you can make one code change that will fix the display format in every page that shows the data.

The DataType attribute specifies a data type that's more specific than the database intrinsic type. In this case only the date should be displayed, not the date and time. The DataType Enumeration provides for many data types, such as Date, Time, PhoneNumber, Currency, EmailAddress, etc. The `DataType` attribute can also enable the app to automatically provide type-specific features. For example:

- The `mailto:` link is automatically created for `DataType.EmailAddress`.
- The date selector is provided for `DataType.Date` in most browsers.

The `DataType` attribute emits HTML 5 `data-` (pronounced data dash) attributes. The `DataType` attributes don't provide validation.

The DisplayFormat attribute

```C#
[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode =
true)]
```

`DataType.Date` doesn't specify the format of the date that's displayed. By default, the date field is displayed according to the default formats based on the server's CultureInfo.

The `DisplayFormat` attribute is used to explicitly specify the date format. The `ApplyFormatInEditMode` setting specifies that the formatting should also be applied to the edit UI. Some fields shouldn't use `ApplyFormatInEditMode`. For example, the currency symbol should generally not be displayed in an edit text box.

The `DisplayFormat` attribute can be used by itself. It's generally a good idea to use the `DataType` attribute with the `DisplayFormat` attribute. The `DataType` attribute conveys the semantics of the data as opposed to how to render it on a screen. The `DataType` attribute provides the following benefits that are not available in `DisplayFormat`:

- The browser can enable HTML5 features. For example, show a calendar control, the locale-appropriate currency symbol, email links, and client-side input validation.
- By default, the browser renders data using the correct format based on the locale.

For more information, see the <input> Tag Helper documentation.

The StringLength attribute

C#

```
[StringLength(50, ErrorMessage = "First name cannot be longer than 50
characters.")]
```

Data validation rules and validation error messages can be specified with attributes. The StringLength attribute specifies the minimum and maximum length of characters that are allowed in a data field. The code shown limits names to no more than 50 characters. An example that sets the minimum string length is shown later.

The `StringLength` attribute also provides client-side and server-side validation. The minimum value has no impact on the database schema.

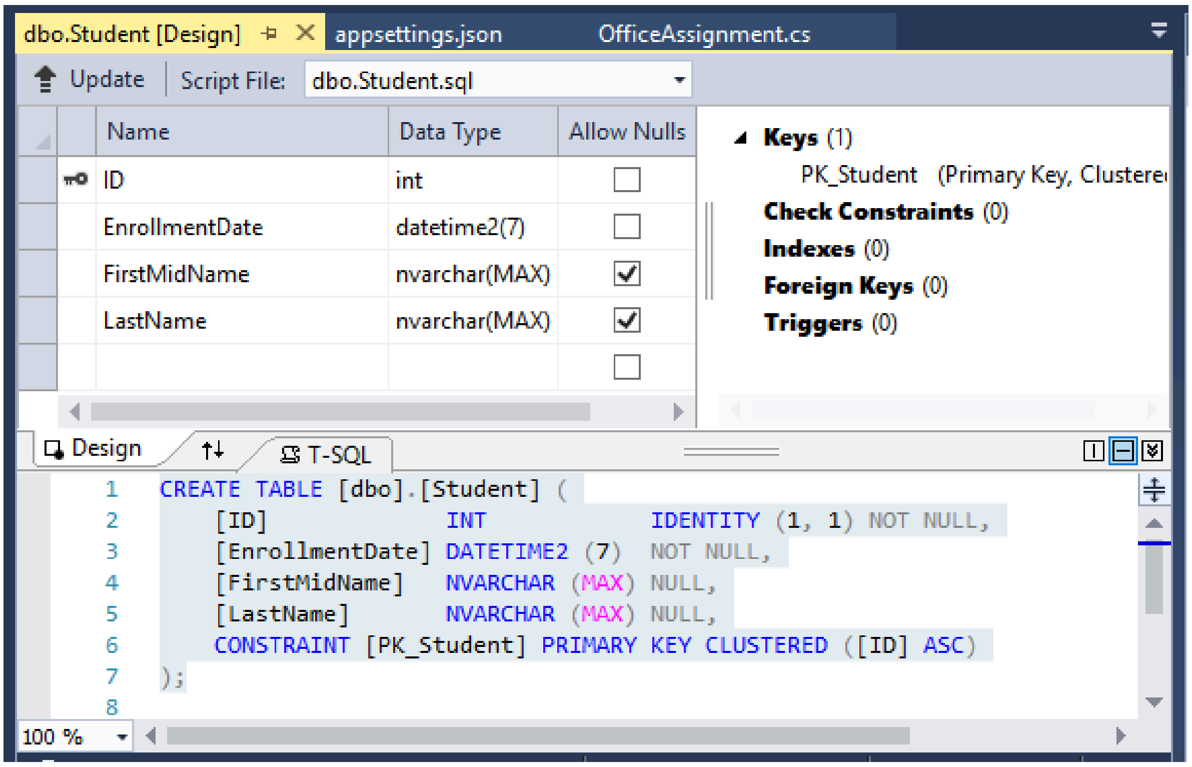
The `StringLength` attribute doesn't prevent a user from entering white space for a name. The RegularExpression attribute can be used to apply restrictions to the input. For example, the following code requires the first character to be upper case and the remaining characters to be alphabetical:

C#

```
[RegularExpression(@"^[A-Z]+[a-zA-Z]*$")]
```

Visual Studio

In **SQL Server Object Explorer** (SSOX), open the Student table designer by double-clicking the **Student** table.



Name	Data Type	Allow Nulls
ID	int	☐
EnrollmentDate	datetime2(7)	☐
FirstMidName	nvarchar(MAX)	✓
LastName	nvarchar(MAX)	✓
		☐

Keys (1)
PK_Student (Primary Key, Clustere
Check Constraints (0)
Indexes (0)
Foreign Keys (0)
Triggers (0)

```sql
CREATE TABLE [dbo].[Student] (
    [ID]             INT            IDENTITY (1, 1) NOT NULL,
    [EnrollmentDate] DATETIME2 (7)  NOT NULL,
    [FirstMidName]   NVARCHAR (MAX) NULL,
    [LastName]       NVARCHAR (MAX) NULL,
    CONSTRAINT [PK_Student] PRIMARY KEY CLUSTERED ([ID] ASC)
);
```

The preceding image shows the schema for the `Student` table. The name fields have type `nvarchar(MAX)`. When a migration is created and applied later in this tutorial, the name fields become `nvarchar(50)` as a result of the string length attributes.

The Column attribute

C#

```csharp
[Column("FirstName")]
public string FirstMidName { get; set; }
```

Attributes can control how classes and properties are mapped to the database. In the `Student` model, the `Column` attribute is used to map the name of the `FirstMidName` property to "FirstName" in the database.

When the database is created, property names on the model are used for column names (except when the `Column` attribute is used). The `Student` model uses `FirstMidName` for the first-name field because the field might also contain a middle name.

With the `[Column]` attribute, `Student.FirstMidName` in the data model maps to the `FirstName` column of the `Student` table. The addition of the `Column` attribute changes the model backing the `SchoolContext`. The model backing the `SchoolContext` no longer matches the database. That discrepancy will be resolved by adding a migration later in this tutorial.

The Required attribute

```C#
[Required]
```

The `Required` attribute makes the name properties required fields. The `Required` attribute isn't needed for non-nullable types such as value types (for example, `DateTime`, `int`, and `double`). Types that can't be null are automatically treated as required fields.

The `Required` attribute must be used with `MinimumLength` for the `MinimumLength` to be enforced.

```C#
[Display(Name = "Last Name")]
[Required]
[StringLength(50, MinimumLength=2)]
public string LastName { get; set; }
```

`MinimumLength` and `Required` allow whitespace to satisfy the validation. Use the `RegularExpression` attribute for full control over the string.

The Display attribute

```C#
[Display(Name = "Last Name")]
```

The `Display` attribute specifies that the caption for the text boxes should be "First Name", "Last Name", "Full Name", and "Enrollment Date." The default captions had no space dividing the words, for example "Lastname."

Create a migration

Run the app and go to the Students page. An exception is thrown. The `[Column]` attribute causes EF to expect to find a column named `FirstName`, but the column name in the database is still `FirstMidName`.

Visual Studio

The error message is similar to the following example:

```
SqlException: Invalid column name 'FirstName'.
There are pending model changes
Pending model changes are detected in the following:

SchoolContext
```

- In the PMC, enter the following commands to create a new migration and update the database:

  PowerShell

  ```powershell
  Add-Migration ColumnFirstName
  Update-Database
  ```

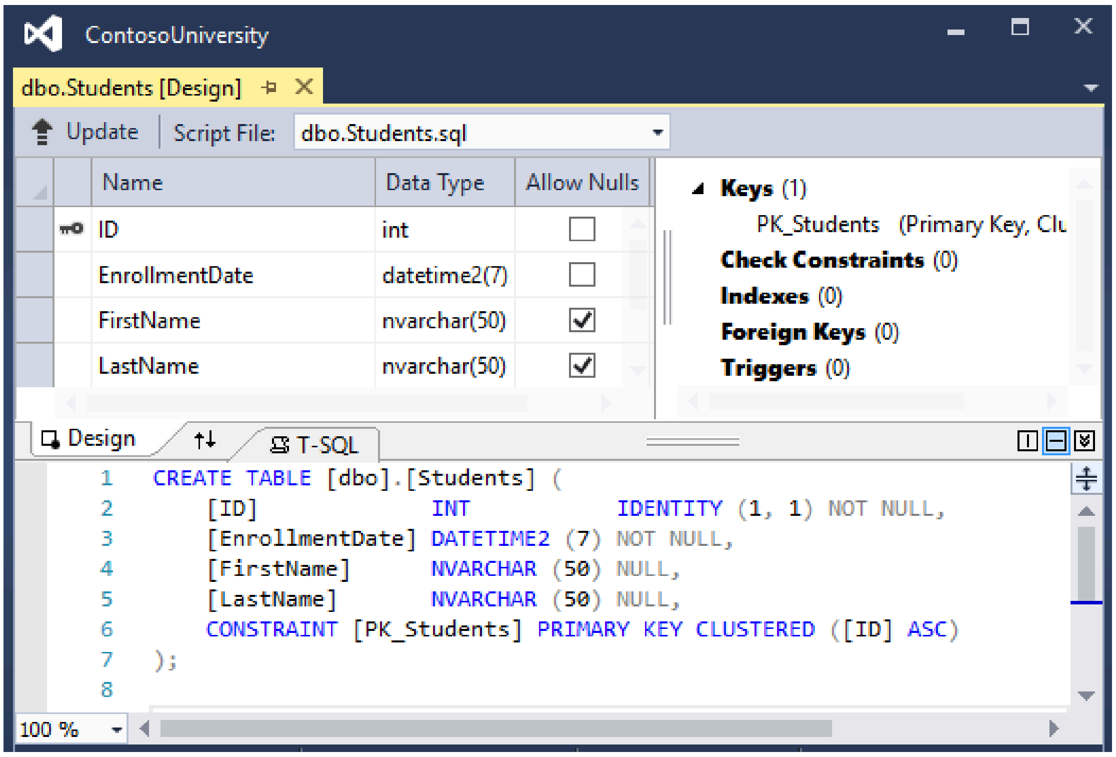
  The first of these commands generates the following warning message:

  text

  ```text
  An operation was scaffolded that may result in the loss of data.
  Please review the migration for accuracy.
  ```

  The warning is generated because the name fields are now limited to 50 characters. If a name in the database had more than 50 characters, the 51 to last character would be lost.

- Open the Student table in SSOX:



Name	Data Type	Allow Nulls
ID	int	☐
EnrollmentDate	datetime2(7)	☐
FirstName	nvarchar(50)	✓
LastName	nvarchar(50)	✓

```
CREATE TABLE [dbo].[Students] (
    [ID]             INT           IDENTITY (1, 1) NOT NULL,
    [EnrollmentDate] DATETIME2 (7) NOT NULL,
    [FirstName]      NVARCHAR (50) NULL,
    [LastName]       NVARCHAR (50) NULL,
    CONSTRAINT [PK_Students] PRIMARY KEY CLUSTERED ([ID] ASC)
);
```

Before the migration was applied, the name columns were of type nvarchar(MAX). The name columns are now `nvarchar(50)`. The column name has changed from `FirstMidName` to `FirstName`.

- Run the app and go to the Students page.
- Notice that times are not input or displayed along with dates.
- Select **Create New**, and try to enter a name longer than 50 characters.

ⓘ **Note**

In the following sections, building the app at some stages generates compiler errors. The instructions specify when to build the app.

The Instructor Entity

Create `Models/Instructor.cs` with the following code:

C#

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
```

```
{
    public class Instructor
    {
        public int ID { get; set; }

        [Required]
        [Display(Name = "Last Name")]
        [StringLength(50)]
        public string LastName { get; set; }

        [Required]
        [Column("FirstName")]
        [Display(Name = "First Name")]
        [StringLength(50)]
        public string FirstMidName { get; set; }

        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}",
ApplyFormatInEditMode = true)]
        [Display(Name = "Hire Date")]
        public DateTime HireDate { get; set; }

        [Display(Name = "Full Name")]
        public string FullName
        {
            get { return LastName + ", " + FirstMidName; }
        }

        public ICollection<Course> Courses { get; set; }
        public OfficeAssignment OfficeAssignment { get; set; }
    }
}
```

Multiple attributes can be on one line. The `HireDate` attributes could be written as follows:

C#

```
[DataType(DataType.Date),Display(Name = "Hire
Date"),DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}",
ApplyFormatInEditMode = true)]
```

Navigation properties

The `Courses` and `OfficeAssignment` properties are navigation properties.

An instructor can teach any number of courses, so `Courses` is defined as a collection.

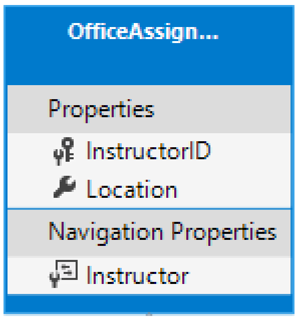
C#

```
public ICollection<Course> Courses { get; set; }
```

An instructor can have at most one office, so the `OfficeAssignment` property holds a single `OfficeAssignment` entity. `OfficeAssignment` is null if no office is assigned.

```C#
public OfficeAssignment OfficeAssignment { get; set; }
```

The OfficeAssignment entity



Create `Models/OfficeAssignment.cs` with the following code:

```C#
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class OfficeAssignment
    {
        [Key]
        public int InstructorID { get; set; }
        [StringLength(50)]
        [Display(Name = "Office Location")]
        public string Location { get; set; }

        public Instructor Instructor { get; set; }
    }
}
```

The Key attribute

The [Key] attribute is used to identify a property as the primary key (PK) when the property name is something other than `classnameID` or `ID`.

There's a one-to-zero-or-one relationship between the `Instructor` and `OfficeAssignment` entities. An office assignment only exists in relation to the instructor it's assigned to. The `OfficeAssignment` PK is also its foreign key (FK) to the `Instructor` entity. A one-to-zero-or-one relationship occurs when a PK in one table is both a PK and a FK in another table.

EF Core can't automatically recognize `InstructorID` as the PK of `OfficeAssignment` because `InstructorID` doesn't follow the ID or classnameID naming convention. Therefore, the `Key` attribute is used to identify `InstructorID` as the PK:

```C#
[Key]
public int InstructorID { get; set; }
```

By default, EF Core treats the key as non-database-generated because the column is for an identifying relationship. For more information, see EF Keys.

The Instructor navigation property

The `Instructor.OfficeAssignment` navigation property can be null because there might not be an `OfficeAssignment` row for a given instructor. An instructor might not have an office assignment.

The `OfficeAssignment.Instructor` navigation property will always have an instructor entity because the foreign key `InstructorID` type is `int`, a non-nullable value type. An office assignment can't exist without an instructor.

When an `Instructor` entity has a related `OfficeAssignment` entity, each entity has a reference to the other one in its navigation property.

The Course Entity

Update `Models/Course.cs` with the following code:

```C#
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Course
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        [Display(Name = "Number")]
        public int CourseID { get; set; }

        [StringLength(50, MinimumLength = 3)]
        public string Title { get; set; }

        [Range(0, 5)]
        public int Credits { get; set; }

        public int DepartmentID { get; set; }

        public Department Department { get; set; }
        public ICollection<Enrollment> Enrollments { get; set; }
        public ICollection<Instructor> Instructors { get; set; }
    }
}
```

The `Course` entity has a foreign key (FK) property `DepartmentID`. `DepartmentID` points to the related `Department` entity. The `Course` entity has a `Department` navigation property.

EF Core doesn't require a foreign key property for a data model when the model has a navigation property for a related entity. EF Core automatically creates FKs in the database wherever they're needed. EF Core creates shadow properties for automatically created FKs. However, explicitly including the FK in the data model can make updates simpler and more efficient. For example, consider a model where the FK property `DepartmentID` is ***not*** included. When a course entity is fetched to edit:

- The `Department` property is `null` if it's not explicitly loaded.
- To update the course entity, the `Department` entity must first be fetched.

When the FK property `DepartmentID` is included in the data model, there's no need to fetch the `Department` entity before an update.

The DatabaseGenerated attribute

The `[DatabaseGenerated(DatabaseGeneratedOption.None)]` attribute specifies that the PK is provided by the application rather than generated by the database.

C#

```
[DatabaseGenerated(DatabaseGeneratedOption.None)]
[Display(Name = "Number")]
public int CourseID { get; set; }
```

By default, EF Core assumes that PK values are generated by the database. Database-generated is generally the best approach. For `Course` entities, the user specifies the PK. For example, a course number such as a 1000 series for the math department, a 2000 series for the English department.

The `DatabaseGenerated` attribute can also be used to generate default values. For example, the database can automatically generate a date field to record the date a row was created or updated. For more information, see Generated Properties.

Foreign key and navigation properties

The foreign key (FK) properties and navigation properties in the `Course` entity reflect the following relationships:

A course is assigned to one department, so there's a `DepartmentID` FK and a `Department` navigation property.

```C#
public int DepartmentID { get; set; }
public Department Department { get; set; }
```

A course can have any number of students enrolled in it, so the `Enrollments` navigation property is a collection:

```C#
public ICollection<Enrollment> Enrollments { get; set; }
```

A course may be taught by multiple instructors, so the `Instructors` navigation property is a collection:

```C#
public ICollection<Instructor> Instructors { get; set; }
```

The Department entity

Create `Models/Department.cs` with the following code:

C#

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ContosoUniversity.Models
{
    public class Department
    {
        public int DepartmentID { get; set; }

        [StringLength(50, MinimumLength = 3)]
        public string Name { get; set; }

        [DataType(DataType.Currency)]
        [Column(TypeName = "money")]
        public decimal Budget { get; set; }

        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}",
                       ApplyFormatInEditMode = true)]
        [Display(Name = "Start Date")]
        public DateTime StartDate { get; set; }

        public int? InstructorID { get; set; }

        public Instructor Administrator { get; set; }
        public ICollection<Course> Courses { get; set; }
    }
}
```

The Column attribute

Previously the `Column` attribute was used to change column name mapping. In the code for the `Department` entity, the `Column` attribute is used to change SQL data type mapping. The `Budget` column is defined using the SQL Server money type in the database:

C#

```csharp
[Column(TypeName="money")]
public decimal Budget { get; set; }
```

Column mapping is generally not required. EF Core chooses the appropriate SQL Server data type based on the CLR type for the property. The CLR `decimal` type maps to a SQL Server `decimal` type. `Budget` is for currency, and the money data type is more appropriate for currency.

Foreign key and navigation properties

The FK and navigation properties reflect the following relationships:

- A department may or may not have an administrator.
- An administrator is always an instructor. Therefore the `InstructorID` property is included as the FK to the `Instructor` entity.

The navigation property is named `Administrator` but holds an `Instructor` entity:

```C#
public int? InstructorID { get; set; }
public Instructor Administrator { get; set; }
```

The `?` in the preceding code specifies the property is nullable.

A department may have many courses, so there's a Courses navigation property:

```C#
public ICollection<Course> Courses { get; set; }
```

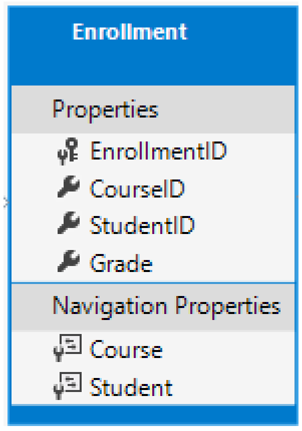
By convention, EF Core enables cascade delete for non-nullable FKs and for many-to-many relationships. This default behavior can result in circular cascade delete rules. Circular cascade delete rules cause an exception when a migration is added.

For example, if the `Department.InstructorID` property was defined as non-nullable, EF Core would configure a cascade delete rule. In that case, the department would be deleted when the instructor assigned as its administrator is deleted. In this scenario, a restrict rule would make more sense. The following fluent API would set a restrict rule and disable cascade delete.

```C#
modelBuilder.Entity<Department>()
    .HasOne(d => d.Administrator)
    .WithMany()
    .OnDelete(DeleteBehavior.Restrict)
```

The Enrollment foreign key and navigation properties

An enrollment record is for one course taken by one student.



Update `Models/Enrollment.cs` with the following code:

```csharp
using System.ComponentModel.DataAnnotations;

namespace ContosoUniversity.Models
{
    public enum Grade
    {
        A, B, C, D, F
    }

    public class Enrollment
    {
        public int EnrollmentID { get; set; }
        public int CourseID { get; set; }
        public int StudentID { get; set; }
        [DisplayFormat(NullDisplayText = "No grade")]
        public Grade? Grade { get; set; }

        public Course Course { get; set; }
        public Student Student { get; set; }
    }
}
```

The FK properties and navigation properties reflect the following relationships:

An enrollment record is for one course, so there's a `CourseID` FK property and a `Course` navigation property:

```csharp
public int CourseID { get; set; }
public Course Course { get; set; }
```

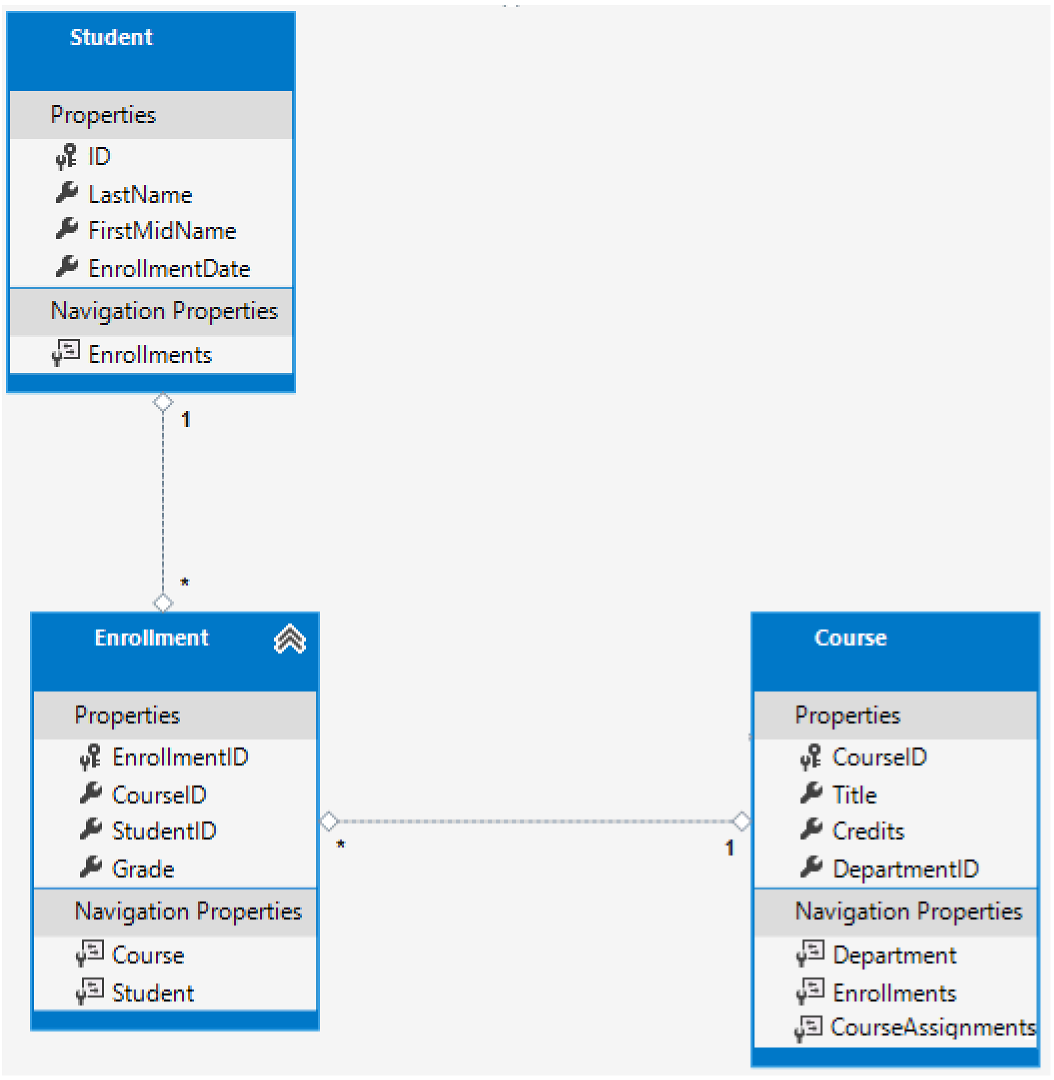
An enrollment record is for one student, so there's a `StudentID` FK property and a `Student` navigation property:

```C#
public int StudentID { get; set; }
public Student Student { get; set; }
```

Many-to-Many Relationships

There's a many-to-many relationship between the `Student` and `Course` entities. The `Enrollment` entity functions as a many-to-many join table ***with payload*** in the database. ***With payload*** means that the `Enrollment` table contains additional data besides FKs for the joined tables. In the `Enrollment` entity, the additional data besides FKs are the PK and `Grade`.

The following illustration shows what these relationships look like in an entity diagram. (This diagram was generated using EF Power Tools for EF 6.x. Creating the diagram isn't part of the tutorial.)



Each relationship line has a 1 at one end and an asterisk (*) at the other, indicating a one-to-many relationship.

If the `Enrollment` table didn't include grade information, it would only need to contain the two FKs, `CourseID` and `StudentID`. A many-to-many join table without payload is sometimes called a pure join table (PJT).

The `Instructor` and `Course` entities have a many-to-many relationship using a PJT.

Update the database context

Update `Data/SchoolContext.cs` with the following code:

C#

```csharp
using ContosoUniversity.Models;
using Microsoft.EntityFrameworkCore;

namespace ContosoUniversity.Data
{
    public class SchoolContext : DbContext
    {
        public SchoolContext(DbContextOptions<SchoolContext> options) :
base(options)
        {
        }

        public DbSet<Course> Courses { get; set; }
        public DbSet<Enrollment> Enrollments { get; set; }
        public DbSet<Student> Students { get; set; }
        public DbSet<Department> Departments { get; set; }
        public DbSet<Instructor> Instructors { get; set; }
        public DbSet<OfficeAssignment> OfficeAssignments { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Course>().ToTable(nameof(Course))
                .HasMany(c => c.Instructors)
                .WithMany(i => i.Courses);
            modelBuilder.Entity<Student>().ToTable(nameof(Student));
            modelBuilder.Entity<Instructor>().ToTable(nameof(Instructor));
        }
    }
}
```

The preceding code adds the new entities and configures the many-to-many relationship between the `Instructor` and `Course` entities.

Fluent API alternative to attributes

The `OnModelCreating` method in the preceding code uses the *fluent API* to configure EF Core behavior. The API is called "fluent" because it's often used by stringing a series of method calls together into a single statement. The following code is an example of the fluent API:

```
C#
```

```csharp
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .Property(b => b.Url)
        .IsRequired();
}
```

In this tutorial, the fluent API is used only for database mapping that can't be done with attributes. However, the fluent API can specify most of the formatting, validation, and mapping rules that can be done with attributes.

Some attributes such as `MinimumLength` can't be applied with the fluent API. `MinimumLength` doesn't change the schema, it only applies a minimum length validation rule.

Some developers prefer to use the fluent API exclusively so that they can keep their entity classes *clean*. Attributes and the fluent API can be mixed. There are some configurations that can only be done with the fluent API, for example, specifying a composite PK. There are some configurations that can only be done with attributes (`MinimumLength`). The recommended practice for using fluent API or attributes:

- Choose one of these two approaches.
- Use the chosen approach consistently as much as possible.

Some of the attributes used in this tutorial are used for:

- Validation only (for example, `MinimumLength`).
- EF Core configuration only (for example, `HasKey`).
- Validation and EF Core configuration (for example, `[StringLength(50)]`).

For more information about attributes vs. fluent API, see Methods of configuration.

Seed the database

Update the code in `Data/DbInitializer.cs`:

```
C#
```

```csharp
using ContosoUniversity.Models;
using System;
using System.Collections.Generic;
using System.Linq;

namespace ContosoUniversity.Data
{
    public static class DbInitializer
    {
        public static void Initialize(SchoolContext context)
        {
            // Look for any students.
            if (context.Students.Any())
            {
                return;   // DB has been seeded
            }

            var alexander = new Student
            {
                FirstMidName = "Carson",
                LastName = "Alexander",
                EnrollmentDate = DateTime.Parse("2016-09-01")
            };

            var alonso = new Student
            {
                FirstMidName = "Meredith",
                LastName = "Alonso",
                EnrollmentDate = DateTime.Parse("2018-09-01")
            };

            var anand = new Student
            {
                FirstMidName = "Arturo",
                LastName = "Anand",
                EnrollmentDate = DateTime.Parse("2019-09-01")
            };

            var barzdukas = new Student
            {
                FirstMidName = "Gytis",
                LastName = "Barzdukas",
                EnrollmentDate = DateTime.Parse("2018-09-01")
            };

            var li = new Student
            {
                FirstMidName = "Yan",
                LastName = "Li",
                EnrollmentDate = DateTime.Parse("2018-09-01")
            };

            var justice = new Student
            {
                FirstMidName = "Peggy",
                LastName = "Justice",
                EnrollmentDate = DateTime.Parse("2017-09-01")
            };

            var norman = new Student
            {
                FirstMidName = "Laura",
                LastName = "Norman",
                EnrollmentDate = DateTime.Parse("2019-09-01")
            };

            var olivetto = new Student
            {
                FirstMidName = "Nino",
                LastName = "Olivetto",
                EnrollmentDate = DateTime.Parse("2011-09-01")
            };

            var students = new Student[]
            {
                alexander,
                alonso,
                anand,
                barzdukas,
                li,
                justice,
                norman,
                olivetto
            };

            context.AddRange(students);

            var abercrombie = new Instructor
            {
                FirstMidName = "Kim",
                LastName = "Abercrombie",
                HireDate = DateTime.Parse("1995-03-11")
            };

            var fakhouri = new Instructor
            {
                FirstMidName = "Fadi",
                LastName = "Fakhouri",
                HireDate = DateTime.Parse("2002-07-06")
            };

            var harui = new Instructor
            {
                FirstMidName = "Roger",
                LastName = "Harui",
                HireDate = DateTime.Parse("1998-07-01")
            };

            var kapoor = new Instructor
            {
                FirstMidName = "Candace",
                LastName = "Kapoor",
                HireDate = DateTime.Parse("2001-01-15")
            };

            var zheng = new Instructor
            {
                FirstMidName = "Roger",
                LastName = "Zheng",
                HireDate = DateTime.Parse("2004-02-12")
            };

            var instructors = new Instructor[]
            {
                abercrombie,
                fakhouri,
                harui,
                kapoor,
                zheng
            };

            context.AddRange(instructors);

            var officeAssignments = new OfficeAssignment[]
            {
                new OfficeAssignment {
                    Instructor = fakhouri,
                    Location = "Smith 17" },
                new OfficeAssignment {
                    Instructor = harui,
                    Location = "Gowan 27" },
                new OfficeAssignment {
                    Instructor = kapoor,
                    Location = "Thompson 304" }
            };

            context.AddRange(officeAssignments);

            var english = new Department
            {
                Name = "English",
                Budget = 350000,
                StartDate = DateTime.Parse("2007-09-01"),
                Administrator = abercrombie
            };

            var mathematics = new Department
            {
                Name = "Mathematics",
                Budget = 100000,
                StartDate = DateTime.Parse("2007-09-01"),
                Administrator = fakhouri
            };
```