

Debugging and Diagnostics

Wael Kdouh - @waelkdouh

Senior Consultant



Module 5: Debugging and Diagnostics

ASP.NET Core Diagnostics

ASP.NET Core Diagnostics

- Microsoft ASP.NET Core introduces various new diagnostic features, that exist in the “Microsoft.AspNetCore.Diagnostics” NuGet package.
- The ASP.NET Core v2 metapackage “Microsoft.AspNetCore.All” already contains the diagnostics package.
- You need to configure the diagnostics in the Configure() method.
- Order is important in configuring the application pipeline
 - app.Run() should execute after all Diagnostics configuration
- Limit public access to diagnostic information
 - Verify application environment

ASP.NET Core Diagnostics Includes

- Developer Exception Page
- Exception Handling
- Status Code Page
- Welcome Page
- Database Error Page (Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore)

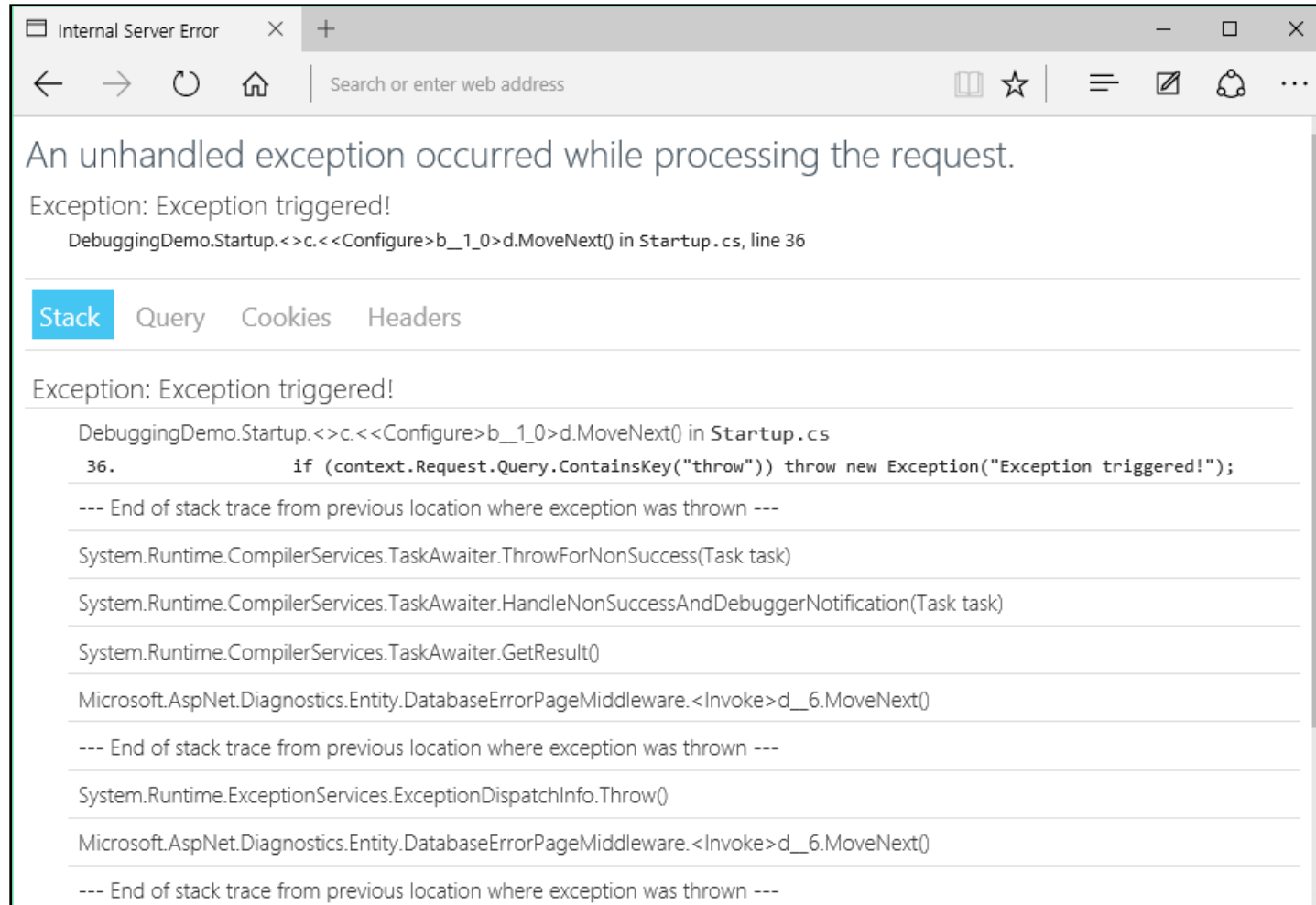
Developer Exception Page

- To use the developer exception page you need to put `UseDeveloperExceptionPage` before any middleware you want to catch exceptions in, such as `app.UseMvc`

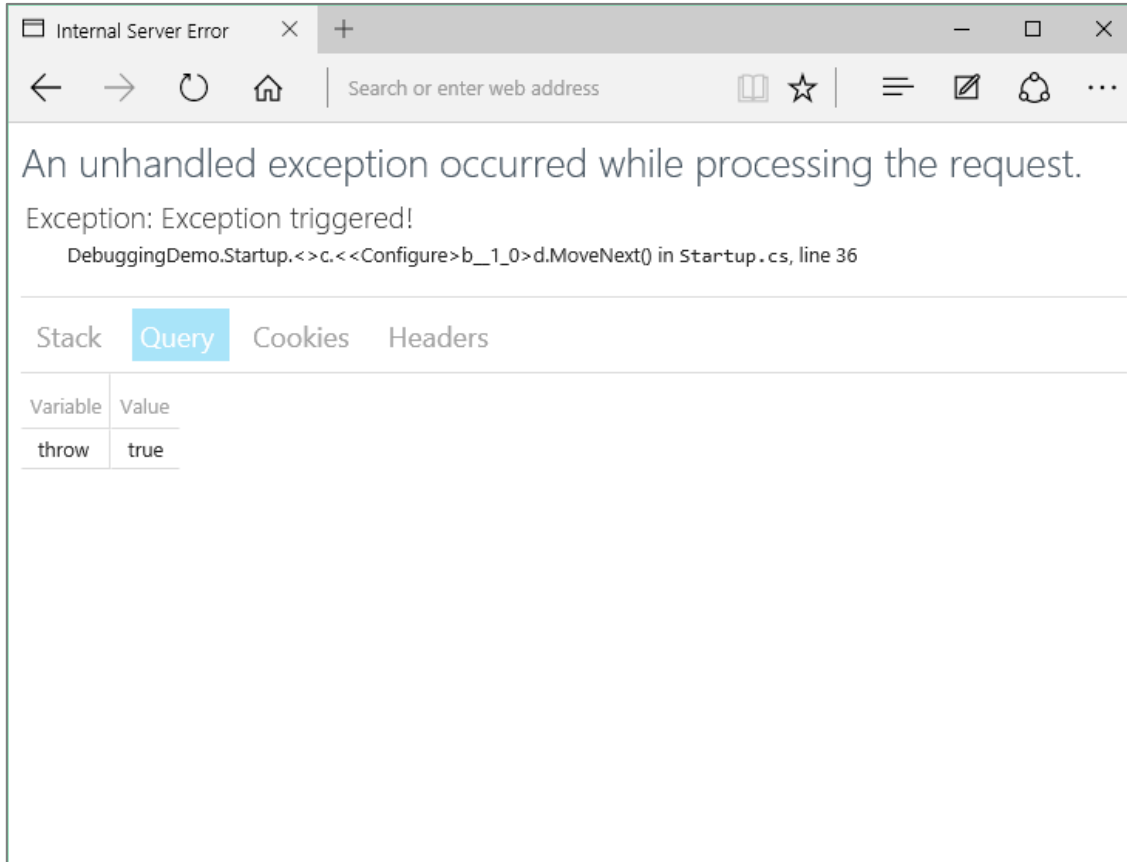
```
public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole();
    env.EnvironmentName = EnvironmentName.Production;
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/error");
    }
}
```

Startup.cs

Developer Exception Page



Developer Exception Page (continued)



Internal Server Error

An unhandled exception occurred while processing the request.

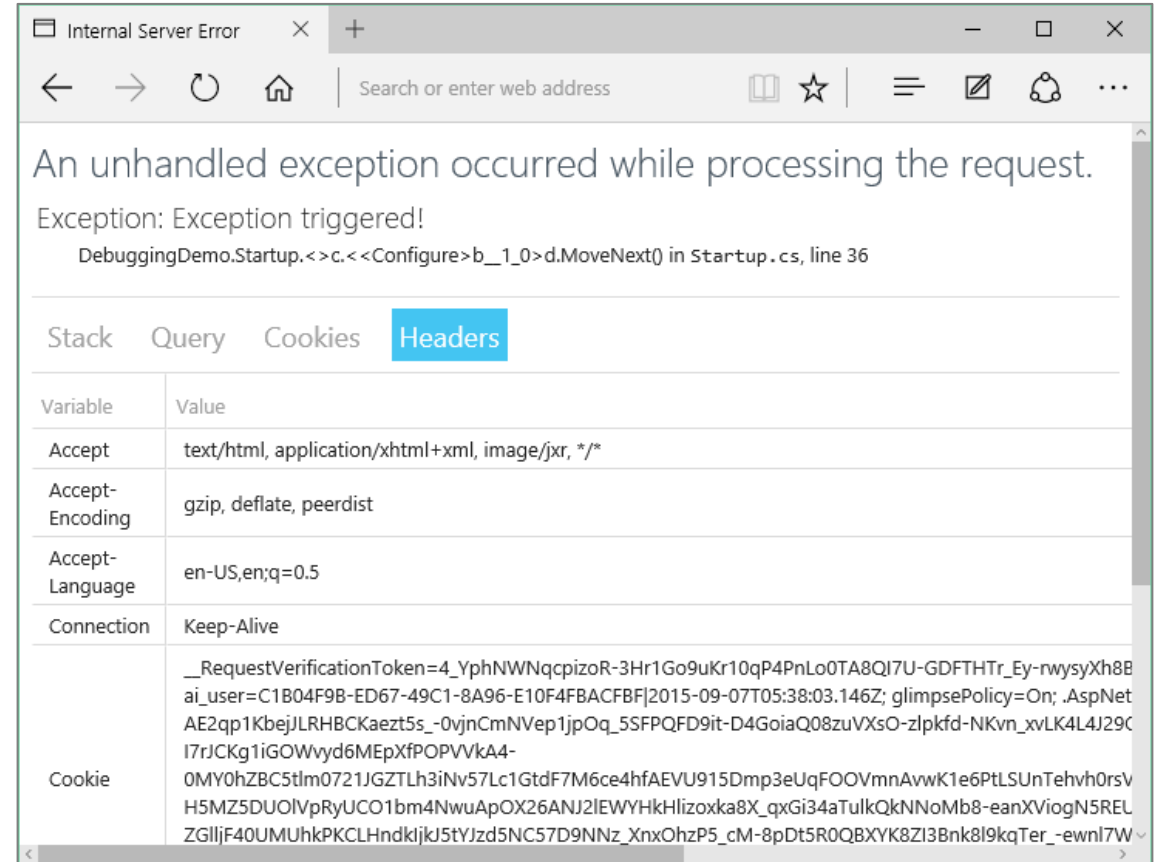
Exception: Exception triggered!

DebuggingDemo.Startup.<>c.<<Configure>b_1_0>d.MoveNext() in Startup.cs, line 36

Stack Query Cookies Headers

Variable	Value
throw	true

Query Parameters



Internal Server Error

An unhandled exception occurred while processing the request.

Exception: Exception triggered!

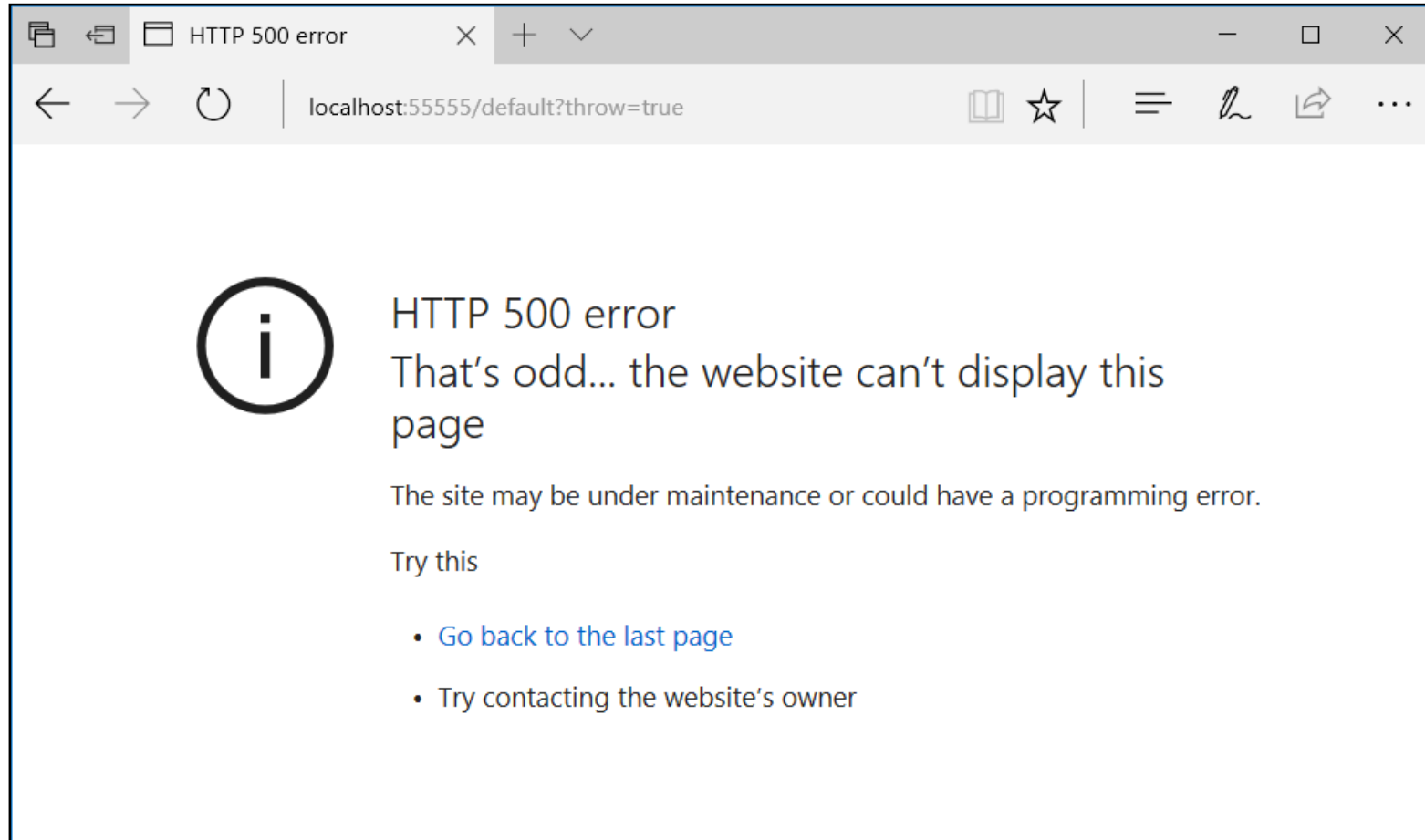
DebuggingDemo.Startup.<>c.<<Configure>b_1_0>d.MoveNext() in Startup.cs, line 36

Stack Query Cookies Headers

Variable	Value
Accept	text/html, application/xhtml+xml, image/jxr, */*
Accept-Encoding	gzip, deflate, peerdist
Accept-Language	en-US,en;q=0.5
Connection	Keep-Alive
Cookie	__RequestVerificationToken=4_YphNWNqcpizoR-3Hr1Go9uKr10qP4PnLo0TA8QI7U-GDFTHTTr_Ey-rwysyXh8Bai_user=C1B04F9B-ED67-49C1-8A96-E10F4FBACFBF]2015-09-07T05:38:03.146Z; glimpsePolicy=On; .AspNetAE2qp1KbejJLRHBCKaezt5s_-0vjinCmNVep1jpOq_5SFPQFD9it-D4GoiaQ08zuVXsO-zlpkfd-NKvn_xvLK4L4J29CI7rJCKg1iGOWvyd6MEpXfPOPVVkA4-0MY0hZBC5tlm0721JGZTLh3iNv57Lc1Gtdf7M6ce4hfAEVU915Dmp3eUqFOOVmnAwvK1e6PtLSUnTehvh0rsvH5MZ5DUOIVpRyUCO1bm4NwuApOX26ANJ2IEWYHkHlizoxka8X_qxGi34aTulkQkNNb8-eanXViogN5RELZGlljF40UMUUhkPKCLHndkljkj5tYJzd5NC57D9NNz_XnxOhzP5_cM-8pDt5R0QBXYK8Zi3Bnk8l9kqTer_-ewnl7W

HTTP Headers

Error Page in Release



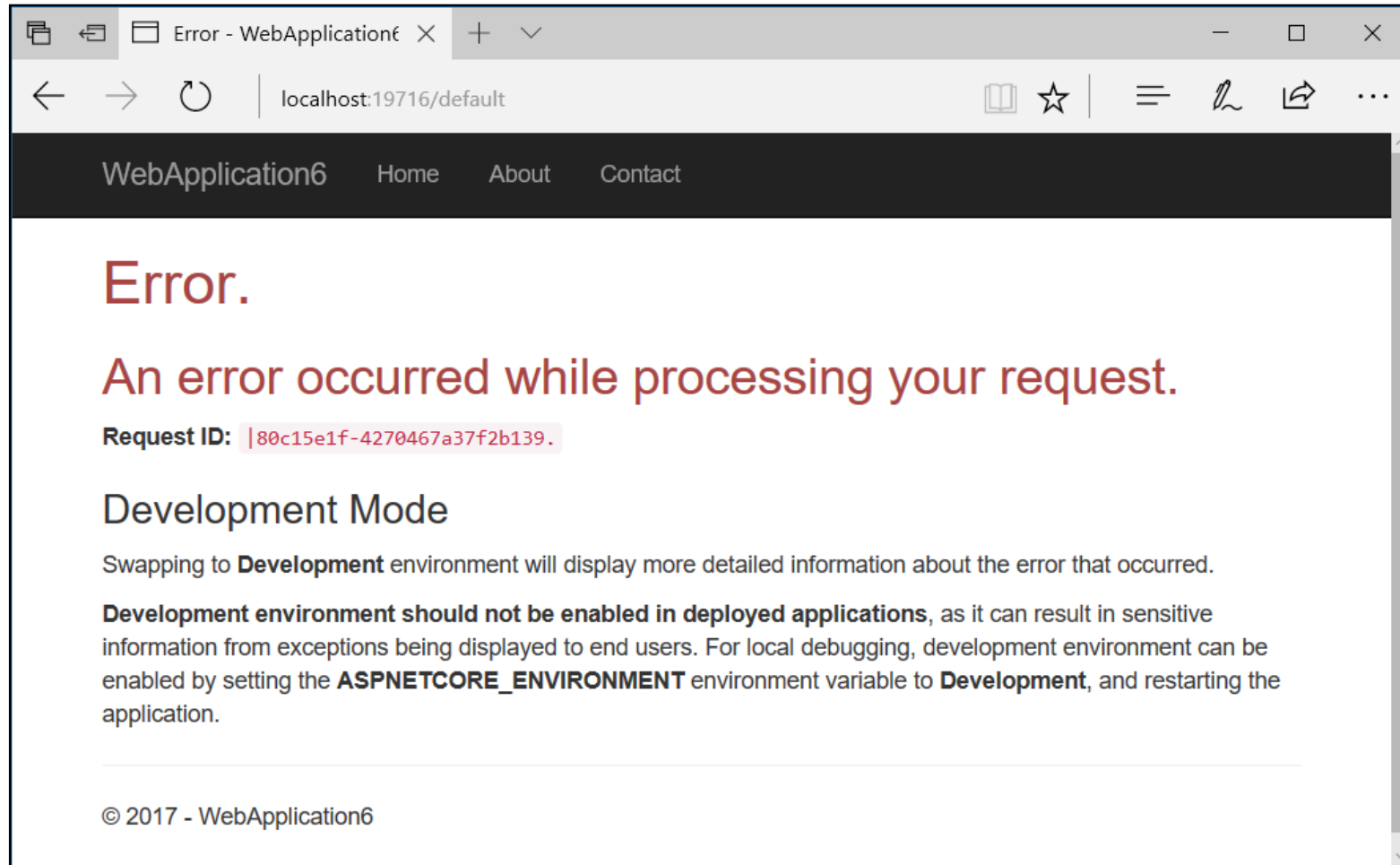
Exception Handler

- It's a good idea to configure an exception handler page to use when the app is not running in the Development environment.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole();
    env.EnvironmentName = EnvironmentName.Production;
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/error");
    }
}
```

Startup.cs

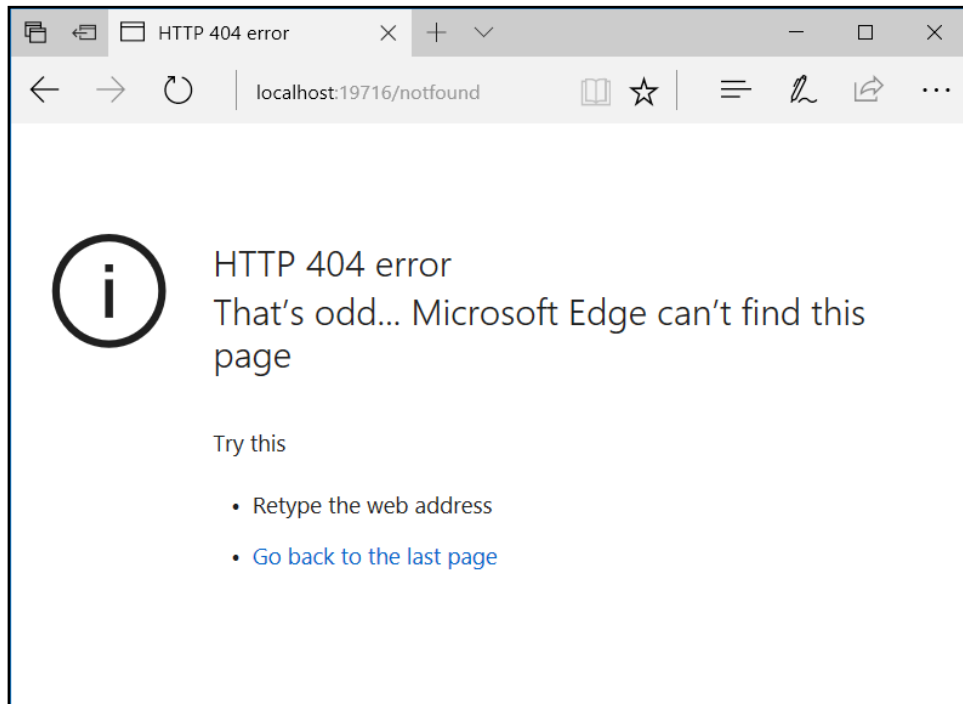
Exception Handler (continued)



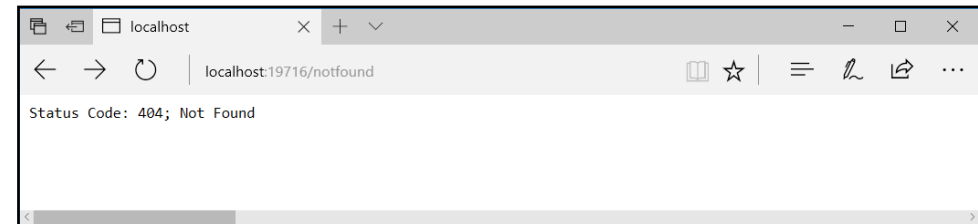
Status Code Page

- By default, your app will not provide a rich status code page for HTTP status codes such as 500 (Internal Server Error) or 404 (Not Found), you can configure it by adding

```
app.UseStatusCodePages();
```



Without middleware



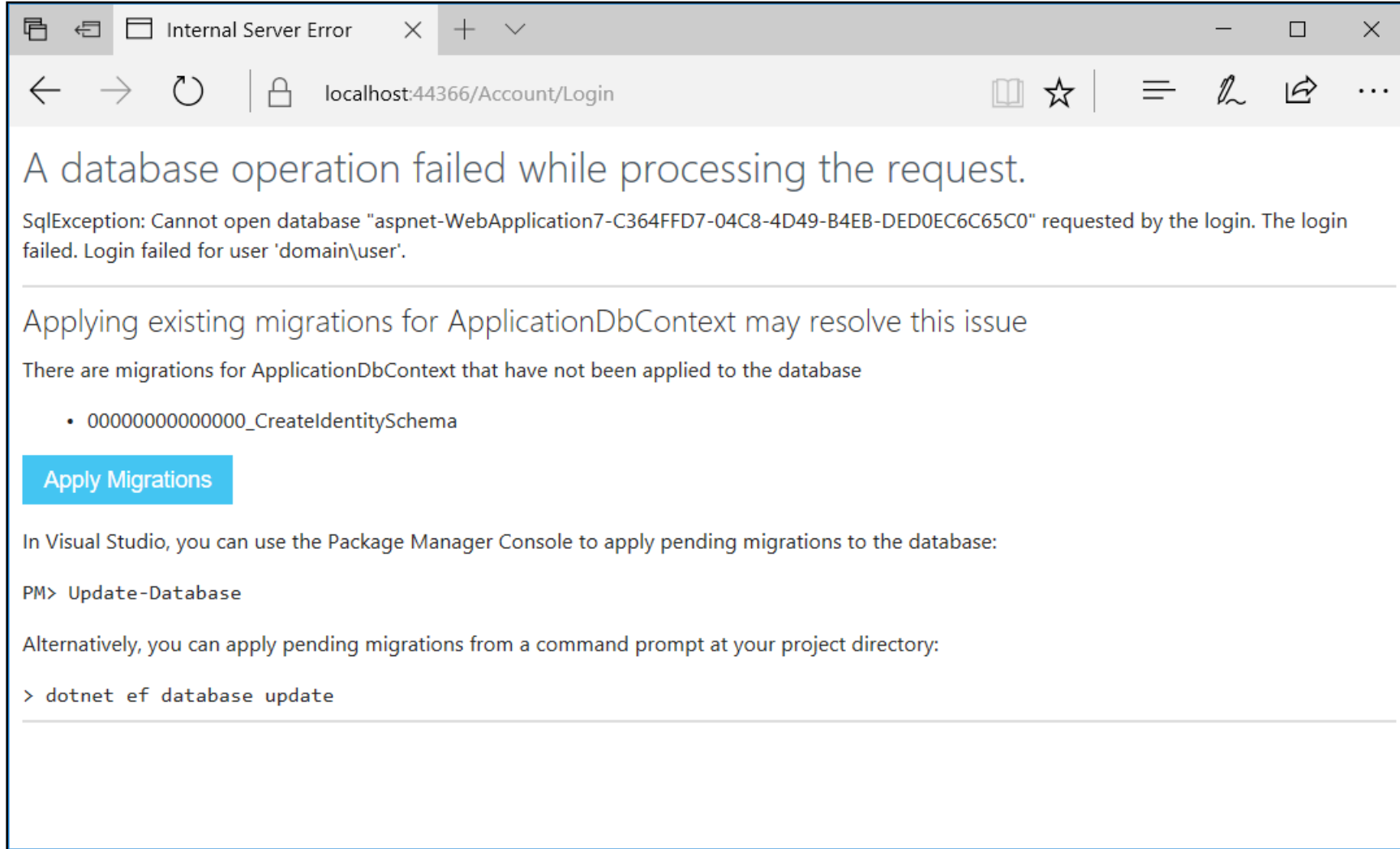
With middleware

Database Error Page

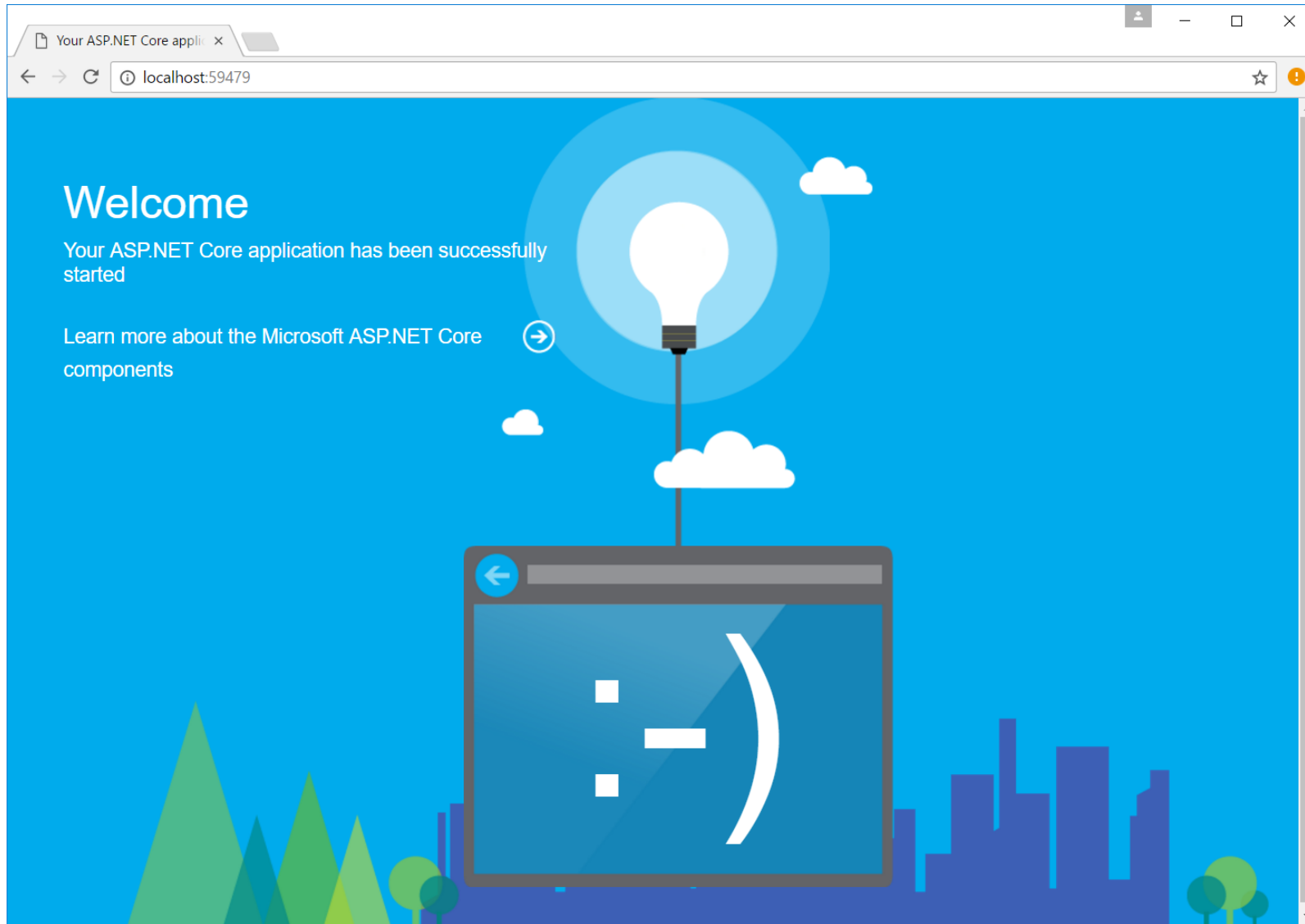
- Captures synchronous and asynchronous database related exceptions from the pipeline that may be resolved using Entity Framework migrations.
- When these exceptions occur an HTML response with details of possible actions to resolve the issue is generated.
- You should use "**Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore**" NuGet package which is already included in the "**Microsoft.AspNetCore.All**" metapackage

```
app.UseDatabaseErrorPage();
```

Database Error Page (continued)



Welcome Page: `app.UseWelcomePage();`

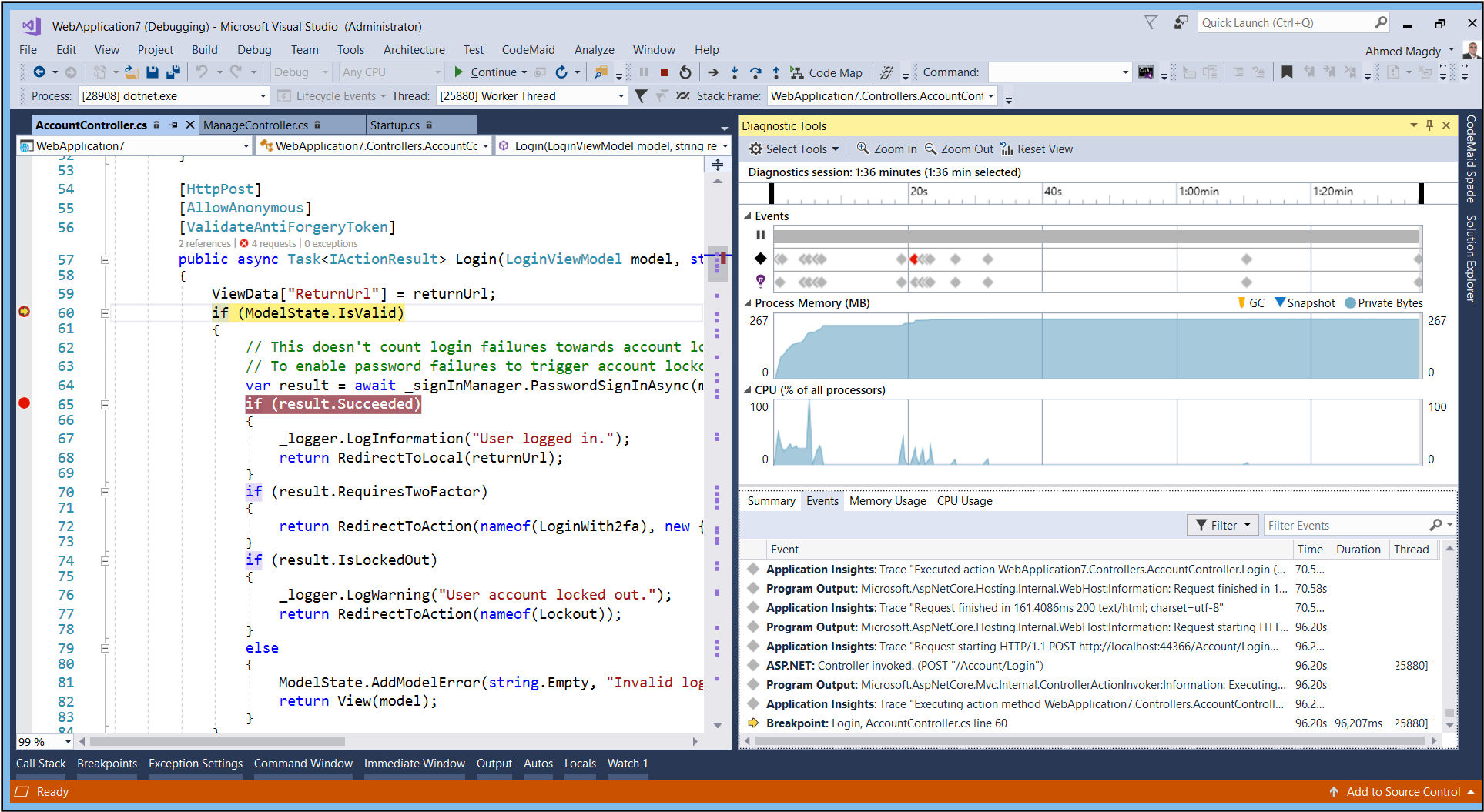


Demo: ASP.NET Core Diagnostics

Module 5: Debugging and Diagnostics

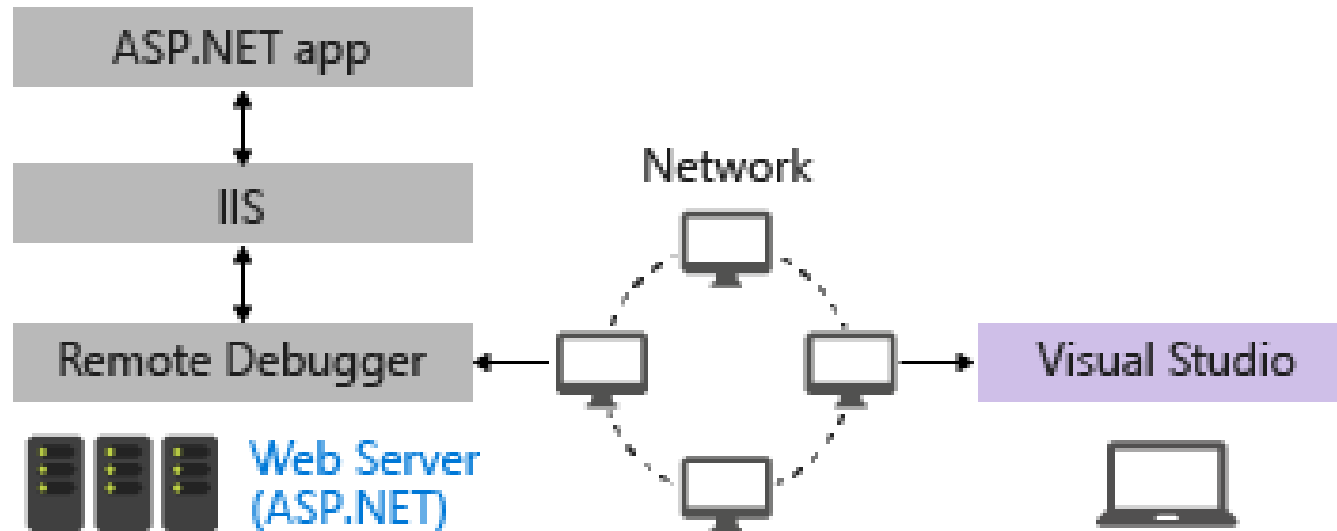
ASP.NET Core Debugging

IntelliTrace



Remote Debugging

- What is remote debugging?
 - It's the ability to debug your application which is hosted on a different/remote computer.
 - Different computers can exist on a remote location and can be either a physical or a virtual VM or a VM running on the cloud (Azure, AWS, Google Cloud)
- To debug an ASP.NET application that has been deployed to IIS, install and run the remote tools on the computer where you deployed your app, and then attach to your running app from Visual Studio.



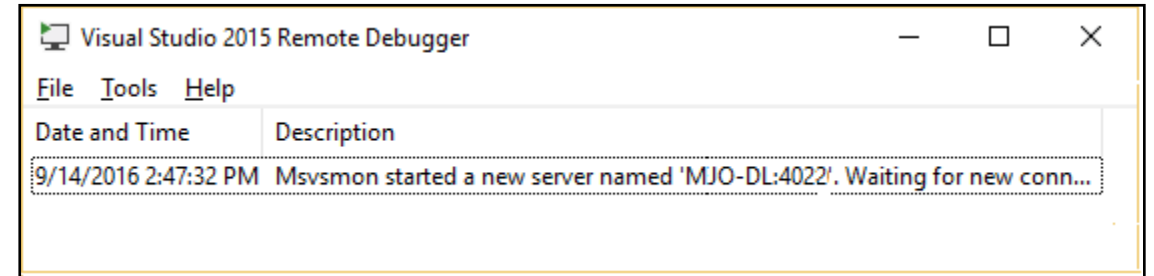
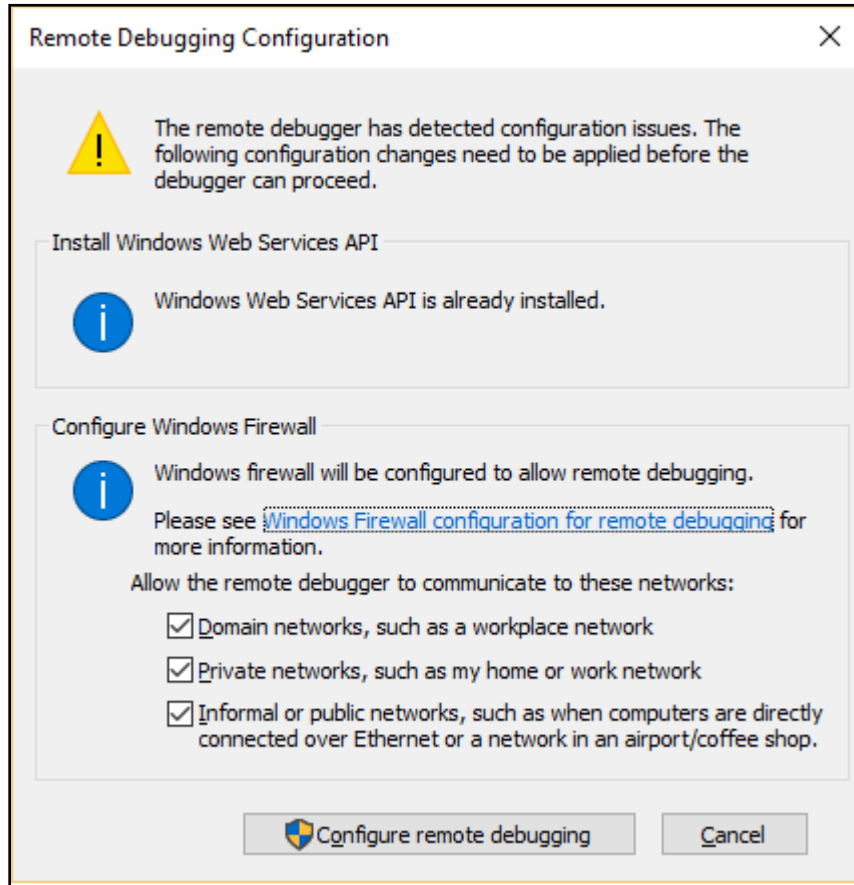
Remote Debugging Step 1

- Download and Install the Remote Tools on the Windows Server
 - On the device or server that you want to debug against, get the correct version of the remote tools.
 - Always download the version matching the OS of the device (x86 or x64)
 - For Visual Studio 2017 you can get it from <https://www.visualstudio.com/downloads/#remote-tools-for-visual-studio-2017>
 - For other version you can get it from

Version	Link	Notes
Visual Studio 2017	Remote Tools	Always download the version matching your device operating system (x86 or x64). For older browsers, use these direct links: Remote Tools (x64) and Remote Tools (x86).
Visual Studio 2015 Update 3	Remote tools	If prompted, join the free Visual Studio Dev Essentials group or you can sign in with a valid Visual Studio subscription. Then reopen the link if necessary.
Visual Studio 2015 (older)	Remote tools	If prompted, join the free Visual Studio Dev Essentials group or you can sign in with a valid Visual Studio subscription. Then reopen the link if necessary.
Visual Studio 2013	Remote tools	Download page in Visual Studio 2013 documentation
Visual Studio 2012	Remote tools	Download page in Visual Studio 2012 documentation

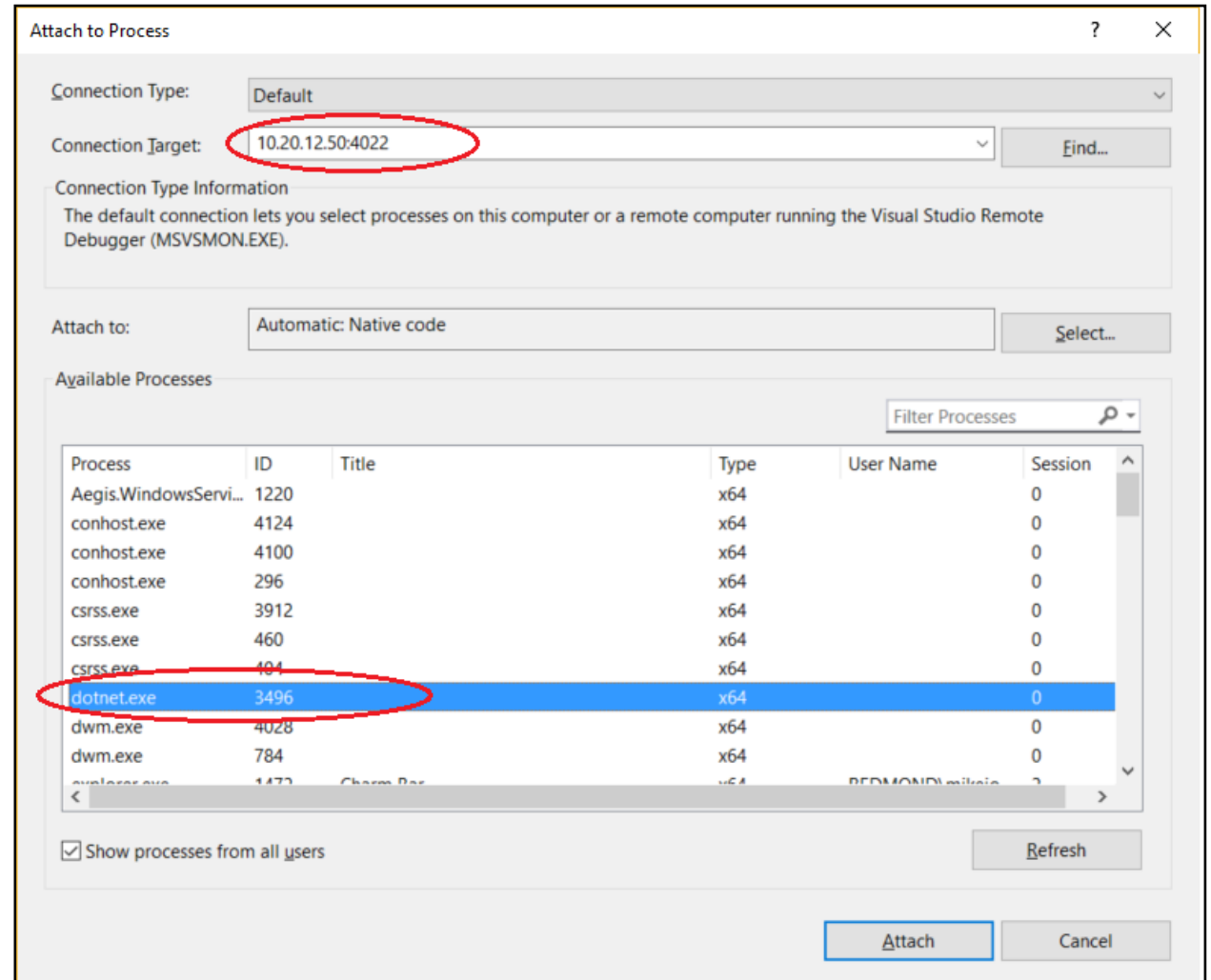
Remote Debugging Step 2

- Set up the remote debugger on Windows Server



Remote Debugging Step 3

- Attach to the ASP.NET application from the Visual Studio computer
 - On the Visual Studio computer, open the MyASPApp solution.
 - In Visual Studio, click Debug > Attach to Process (Ctrl + Alt + P).
 - Set the Qualifier field to <remote computer name>:4022
- Since ASP.NET Core runs inside **dotnet.exe** process you need to select it from the listed processes.



Demo: IntelliTrace

Module 5: Debugging and Diagnostics

Logging

Logging

- ASP.NET Core supports a logging API that works with a variety of logging providers.
- Built-in providers let you send logs to one or more destinations, and you can plug in a third-party logging framework.
- Minimal setup code required
- Loggers
 - Console
 - Debug
 - EventSource
 - EventLog
 - TraceSource
 - Azure App Service
- Implemented via ASP.NET Core Dependency Injection (DI) framework
- Logging is configured in the core service because you need Logging to be available everywhere.

Recommended Practice: Logger Injection in Controller

```
public class TodoController : Controller
{
    private readonly ITodoRepository _todoRepository;
    private readonly ILogger _logger;

    public TodoController(ITodoRepository todoRepository,
        ILogger<TodoController> logger)
    {
        _todoRepository = todoRepository;
        _logger = logger;
    }
}
```

```
public IActionResult GetById(string id)
{
    _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
    var item = _todoRepository.Find(id);
    if (item == null)
    {
        _logger.LogWarning(LoggingEvents.GetItemNotFound, "GetById({ID}) NOT FOUND", id);
        return NotFound();
    }
    return new ObjectResult(item);
}
```

Add logging providers

- A logging provider takes the messages that you create with an ILogger object and displays or stores them.
- For example, the Console provider displays messages on the console, and the Azure App Service provider can store them in Azure blob storage.
- To use a provider, call the provider's Add<ProviderName> extension method in Program.cs: while constructing WebHostBuilder

```
public static void Main(string[] args)
{
    BuildWebHost(args).Run();
}

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


Add logging providers (continued)

```
public static void Main(string[] args)
{
    var webHost = new WebHostBuilder()
        .UseKestrel()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .ConfigureAppConfiguration((hostingContext, config) =>
        {
            var env = hostingContext.HostingEnvironment;
            config.AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
                .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true, reloadOnChange: true);
            config.AddEnvironmentVariables();
        })
        .ConfigureLogging((hostingContext, logging) =>
        {
            logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
            logging.AddConsole();
            logging.AddDebug();
        })
        .UseStartup<Startup>()
        .Build();

    webHost.Run();
}
```

Console Log

Catch-All Logger
placed in Startup.cs

Log Level

Requesting Controller
calling the Logger

```
C:\Users\waziz\.dnx\runtimes\dnx-coreclr-win-x64.1.0.0-beta8\bin\dnx.exe

info : [Microsoft.Net.Http.Server.WebListener] Start
info : [Microsoft.Net.Http.Server.WebListener] Listening on prefix: http://localhost:5000/
Hosting environment: Development
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
info : [Catchall Endpoint] No endpoint found for request /
info : [Catchall Endpoint] No endpoint found for request /BadRequest
info : [TodoApi.Controllers.TodoController] Listing all items
warning : [TodoApi.Controllers.TodoController] Custom warning
info : [TodoApi.Controllers.TodoController] Generating sample items.
```

Logging Severity Levels

#	Level	Description
0	Trace	Logs that contain the most detailed messages. These messages may contain sensitive application data. These messages are disabled by default and should never be enabled in a production environment.
1	Debug	Logs that are used for interactive investigation during development. These logs should primarily contain information useful for debugging and have no long-term value.
2	Information	Logs that track the general flow of the application. These logs should have long-term value.
3	Warning	Logs that highlight an abnormal or unexpected event in the application flow, but do not otherwise cause the application execution to stop.
4	Error	Logs that highlight when the current flow of execution is stopped due to a failure. These should indicate a failure in the current activity, not an application-wide failure.
5	Critical	Logs that describe an unrecoverable application or system crash, or a catastrophic failure that requires immediate attention.
6	None	Not used for writing log messages. Specifies that a logging category should not write any messages.

Log Filters

- The log filtering is a feature of the default LoggerFactory, and is wired up to the registered configuration object.
- All log messages can be run through filters, and they can all be configured via configuration.
- You can specify a minimum log level for a specific provider and category or for all providers or all categories. Any logs below the minimum level aren't passed to that provider, so they don't get displayed or stored.

```
public static void Main(string[] args)
{
    var webHost = new WebHostBuilder()
        .UseKestrel()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .ConfigureAppConfiguration((hostingContext, config) =>
        {
            var env = hostingContext.HostingEnvironment;
            config.AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
                .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true, reloadOnChange: true);
            config.AddEnvironmentVariables();
        })
        .ConfigureLogging((hostingContext, logging) =>
        {
            logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
            logging.AddConsole();
        });
}
```

Log Filters (continued)

- This JSON creates six filter rules, one for the Debug provider, four for the Console provider, and one that applies to all providers. You'll see later how just one of these rules is chosen for each provider when an ILogger object is created.

```
{
  "Logging": {
    "IncludeScopes": false,
    "Debug": {
      "LogLevel": {
        "Default": "Information"
      }
    },
    "Console": {
      "LogLevel": {
        "Microsoft.AspNetCore.Mvc.Razor.Internal": "Warning",
        "Microsoft.AspNetCore.Mvc.Razor.Razor": "Debug",
        "Microsoft.AspNetCore.Mvc.Razor": "Error",
        "Default": "Information"
      }
    },
    "LogLevel": {
      "Default": "Debug"
    }
  }
}
```

Log Scopes

- You can group a set of logical operations within a scope in order to attach the same data to each log that is created as part of that set.
- For example, you might want every log created as part of processing a transaction to include the transaction ID.+

```
public IActionResult GetById(string id)
{
    TodoItem item;
    using (_logger.BeginScope("Message attached to logs created in the using block"))
    {
        _logger.LogInformation(LoggingEvents.GetItem, "Getting item {ID}", id);
        item = _todoRepository.Find(id);
        if (item == null)
        {
            _logger.LogWarning(LoggingEvents.GetItemNotFound, "GetById({ID}) NOT FOUND", id);
            return NotFound();
        }
    }
    return new ObjectResult(item);
}
```


Built-in logging providers

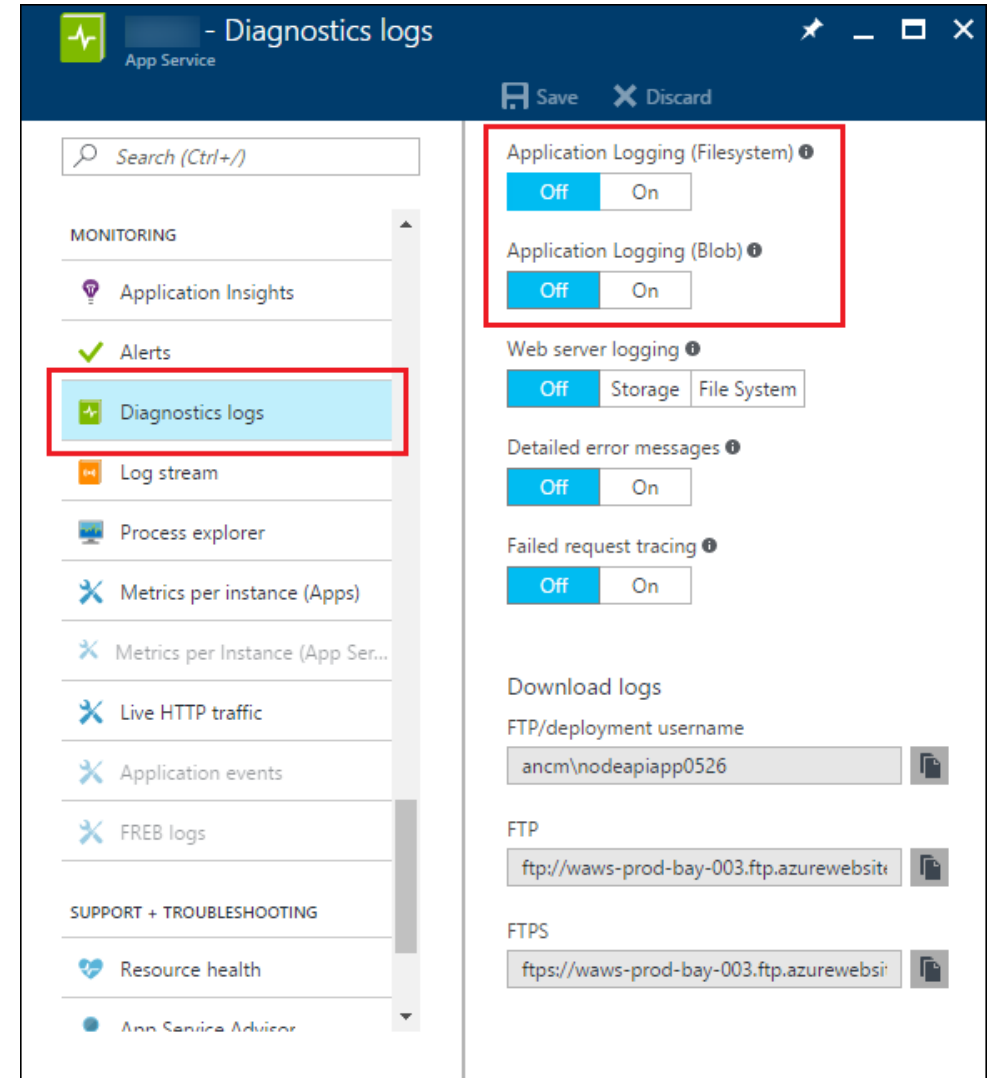
- Console
 - The `Microsoft.Extensions.Logging.Console` provider package sends log output to the console.
 - `logging.AddConsole()`
- Debug
 - The `Microsoft.Extensions.Logging.Debug` provider package writes log output by using the `System.Diagnostics.Debug` class (`Debug.WriteLine` method calls).
 - `logging.AddDebug()`
- EventSource
 - The `Microsoft.Extensions.Logging.EventSource` provider package can implement event tracing.
 - `logging.AddEventSourceLogger()`

Built-in logging providers (continued)

- EventLog
 - The `Microsoft.Extensions.Logging.EventLog` provider package sends log output to the Windows Event Log.
 - `logging.AddEventLog()`
- TraceSource
 - The `Microsoft.Extensions.Logging.TraceSource` provider package uses the `System.Diagnostics.TraceSource` libraries and providers.
 - `logging.AddTraceSource(sourceSwitchName)`
- Azure App Service
 - The `Microsoft.Extensions.Logging.AzureAppServices` provider package writes logs to text files in an Azure App Service app's file system and to blob storage in an Azure Storage account. The provider is available only for apps that target ASP.NET Core 1.1.0 or higher.

The Azure App Service provider

- When you deploy to an App Service app, your application honors the settings in the Diagnostic Logs section of the App Service page of the Azure portal.
- When you change those settings, the changes take effect immediately without requiring that you restart the app or redeploy code to it.
- You don't have to install the provider package or call the `AddAzureWebAppDiagnostics` extension method.
- The provider is automatically available to your app when you deploy the app to Azure App Service.



Third-party logging providers

- [elmah.io](#) - provider for the Elmah.io service
- [JSNLog](#) - logs JavaScript exceptions and other client-side events in your server-side log.
- [Loggr](#) - provider for the Loggr service
- [NLog](#) - provider for the NLog library
- [Serilog](#) - provider for the Serilog library

Logging Best Practices

- Log using the correct LogLevel, that is, Warning, Error, Critical
- Log information that will enable errors to be identified quickly
- Keep log messages concise without sacrificing important information
- Prevent extra method calls and log message setup overhead
- Name your loggers with a distinct prefix
- Use Scopes sparingly
- Application logging code should be related to the business concerns of the application

