Web API Development with ASP.NET Core 6

Slides

https://www.accelebrate.com
(877) 849-1850 ❖ info@accelebrate.com

Customized Technical Training

Accelebrate
ACCELERATED LEARNING, CELEBRATED RESULTS®

On-Site, Customized Private Training

Don't settle for a one-size-fits-all class! Let Accelebrate tailor a private class to your group's goals and experience. Classes can be delivered at your site or online (or a combination of both) worldwide. Visit our web site at **https://www.accelebrate.com** and contact us at **sales@accelebrate.com** for details.

Public Online Training

Need to train just 1-3 people? Attend one of our regularly scheduled, live, instructor-led public online classes. Class sizes are small (typically 3-6 attendees) and you receive just as much hands-on time and individual attention from your instructor as our private classes. For course dates, times, outlines, pricing, and registration, visit **https://www.accelebrate.com/public-training-schedule.**

Newsletter

Want to find out about our latest class offerings? Subscribe to our newsletter **https://www.accelebrate.com/newsletter**.

Blog

Get insights and tutorials from our instructors and staff! Visit our blog, **https://www.accelebrate.com/blog** and join the discussion threads and get feedback from our instructors!

Learning Resources

Get access to learning guides, tutorials, and past issues of our newsletter at the Accelebrate library, **https://www.accelebrate.com/library**.

Call us for a training quote!

877 849 1850

Accelebrate, Inc. was founded in 2002 with the goal of delivering **private training** that rapidly achieves participants' goals. Each year, our experienced instructors deliver hundreds of classes online and at client sites all over the US, Canada, and abroad. We pride ourselves on our **instructors' real-world experience** and ability to **adapt the training** to your team and their objectives. We offer a wide range of topics, including:

- Angular, React, and Vue
- JavaScript
- Data Science using R, Python, & Julia
- Excel Power Query & Power BI
- Tableau
- .NET & VBA programming
- SharePoint & Microsoft 365
- DevOps
- iOS & Android Development
- PostgreSQL, Oracle, and SQL Server
- Java, Spring, and Groovy
- Agile
- Web/Application Server Admin
- HTML5 & Mobile Web Development
- AWS & Azure
- Adobe & Articulate software
- Docker, Kubernetes, Ansible, & Git
- IT leadership & Project Management
- AND MORE *(see back)*

"It's not often that everything goes according to plan and you feel you really got full value for money spent, but in this case, I feel the investment in the Articulate training has already paid off in terms of employee confidence and readiness."

— Paul, St John's University

Like us on Facebook + Follow us on Twitter + Watch us on YouTube

Visit our website for a complete list of courses!

Adobe & Articulate
Adobe Captivate
Adobe Presenter
Articulate Storyline / Studio
Camtasia
RoboHelp

AWS, Azure, & Cloud
AWS
Azure
Cloud Computing
Google Cloud
OpenStack

Big Data
Alteryx
Apache Spark
Teradata
Snowflake SQL

Data Science and RPA
Blue Prism
Django
Julia
Machine Learning
MATLAB
Python
R Programming
Tableau
UiPath

Database & Reporting
BusinessObjects
Crystal Reports
Excel Power Query
MongoDB
MySQL
NoSQL Databases
Oracle
Oracle APEX
Power BI
PivotTable and PowerPivot
PostgreSQL
SQL Server
Vertica Architecture & SQL

DevOps, CI/CD & Agile
Agile
Ansible
Chef
Diversity, Equity, Inclusion
Docker
Git
Gradle Build System
Jenkins
Jira & Confluence
Kubernetes
Linux
Microservices
Red Hat
Software Design

Java
Apache Maven
Apache Tomcat
Groovy and Grails
Hibernate
Java & Web App Security
JavaFX
JBoss
Oracle WebLogic
Scala
Selenium & Cucumber
Spring Boot
Spring Framework

JS, HTML5, & Mobile
Angular
Apache Cordova
CSS
D3.js
HTML5
iOS/Swift Development
JavaScript
MEAN Stack
Mobile Web Development
Node.js & Express
React & Redux
Svelte
Swift
Xamarin
Vue

Microsoft & .NET
.NET Core
ASP.NET
Azure DevOps
C#
Design Patterns
Entity Framework Core
IIS
Microsoft Dynamics CRM
Microsoft Exchange Server
Microsoft 365
Microsoft Power Platform
Microsoft Project
Microsoft SQL Server
Microsoft System Center
Microsoft Windows Server
PowerPivot
PowerShell
VBA
Visual C++/CLI
Web API

Other
C++
Go Programming
IT Leadership
ITIL
Project Management
Regular Expressions
Ruby on Rails
Rust
Salesforce
XML

Security
.NET Web App Security
C and C++ Secure Coding
C# & Web App Security
Linux Security Admin
Python Security
Secure Coding for Web Dev
Spring Security

SharePoint
Power Automate & Flow
SharePoint Administrator
SharePoint Developer
SharePoint End User
SharePoint Online
SharePoint Site Owner

SQL Server
Azure SQL Data Warehouse
Business Intelligence
Performance Tuning
SQL Server Administration
SQL Server Development
SSAS, SSIS, SSRS
Transact-SQL

Teleconferencing Tools
Adobe Connect
GoToMeeting
Microsoft Teams
WebEx
Zoom

Web/Application Server
Apache httpd
Apache Tomcat
IIS
JBoss
Nginx
Oracle WebLogic

*Visit **www.accelebrate.com/newsletter** to sign up and receive our **newsletters** with information about **new courses**, **free webinars**, **tutorials**, and **blog articles**.*

Call us for a training quote! 877 849 1850 (US/Canada) or +1 678 648 3133

Web APIs with ASP.NET Core 6

Agenda

- Introduction
- .NET SDK
- What's New in C#
- Application Architecture
- Application Configuration
- Request Routing
- Models
- Controllers
- Web APIs

Web APIs with ASP.NET Core 6

Agenda

- Application State
- Input Validation
- Error Handling
- Logging
- Testing
- Security
- gRPC
- Blazor
- Deployment

Web APIs with ASP.NET Core 6

Introduction

- Evolution of the .NET Platform
- .NET SDKs and Runtimes
- Visual Studio and Visual Studio Code

2002 .NET 1.0, C# 1.0, Visual Studio .NET

2003 .NET 1.1, Visual Studio 2003

2004 Mono 1.0

2005 .NET 2.0, C# 2.0, Visual Studio 2005
Generics, Nullable Value Types

2006 .NET 3.0, Mono 1.2
WPF, WCF, WF

2007 .NET 3.5, C# 3.0, Visual Studio 2008
LINQ, Anonymous Types, Lambda Expressions, Extension Methods, Implicit Typing

2008 Entity Framework 1.0

2009 ASP.NET MVC 1.0

2010 .NET 4.0, C# 4.0, ASP.NET MVC 2, Visual Studio 2010
Named / Optional Arguments, Dynamic Binding

2012 .NET 4.5, C# 5.0, Mono 3.0, ASP.NET MVC 4, Visual Studio 2012
Asynchronous Members (async / await)

2013 .NET 4.5.1, ASP.NET MVC 5, Visual Studio 2013
SignalR 1.0

2015 .NET 4.6, C# 6.0, Mono 4.0, Visual Studio 2015, Visual Studio Code 1.0
Expression Bodied Members, Null Propagator, String Interpolation

2016 Xamarin Acquisition, .NET Core 1.0, .NET Standard 1.0
Entity Framework Core 1.0

2017 .NET 4.7, .NET Core 2.0, C# 7.0, Visual Studio 2017
ASP.NET Razor Pages, Out Variables, Tuples, Ref Locals and Returns

2018 GitHub Acquisition, .NET Standard 2.0
Blazor Server

2019 .NET 4.8, .NET Core 3.0, C# 8.0, Visual Studio 2019
gRPC, Default Interface Methods, Using Declarations, Nullable Reference Types

2020 .NET 5, C# 9.0
Blazor WebAssembly, Records, Init Only Setters, Top-Level Statements

2021 .NET 6, C# 10.0, Visual Studio 2022
.NET MAUI

Introduction

Evolution of the .NET Platform

- .NET 1.0 included ASP.NET Web Forms
  - Had the potential to be cross-platform but was only officially supported on Windows
- Current version of this variant is 4.8 and now referred to as ".NET Framework"
- Will be supported for many years to come (end of support for .NET 3.5 SP1 is October 2028)

Introduction

Evolution of the .NET Platform

- The ASP.NET MVC web application framework was introduced in 2009
- Initially presented as an alternative to Web Forms (not a replacement)
- Accompanied by a related framework for building services called Web API

Introduction

Evolution of the .NET Platform

- In 2016, Microsoft introduced a new variant of .NET called .NET Core
- Many components were completely rewritten
- Fully supported on Windows, macOS, and Linux
- Included a subset of the functionality provided by .NET Framework
  - Focused on web-based workloads (web UIs and services)
- Merged MVC and Web API into the core framework

Introduction

Evolution of the .NET Platform

- The version of .NET Core after 3.1 became the "main line" for .NET and was labeled .NET 5.0
- Supports development of Windows Forms and WPF applications that run on Windows
- The ASP.NET framework in .NET still includes the name "Core" to avoid confusion with previous versions of ASP.NET MVC

Introduction

Evolution of the .NET Platform

- The entire .NET platform is made available as open-source
- Community contributions are encouraged via pull requests
  - Thoroughly reviewed and tightly controlled by Microsoft

`github.com/dotnet`

Introduction

.NET SDKs and Runtimes

- .NET Runtime
  - Different version for each platform
  - Provides assembly loading, garbage collection, JIT compilation of IL code, and other runtime services
  - Includes the dotnet tool for launching applications
- ASP.NET Core Runtime
  - Includes additional packages for running ASP.NET Core applications
  - Reduces the number of packages that you need to deploy with your application

Introduction

.NET SDKs and Runtimes

- .NET SDK
  - Includes the .NET runtime for the platform
  - Additional command-line tools for compiling, testing, and publishing applications
  - Contains everything needed to develop .NET applications (with the help of a text editor)

Introduction

.NET SDKs and Runtimes

- Each version of .NET has a lifecycle status
  - Current – Includes the latest features and bug fixes but will only be supported for a short time after the next release
  - LTS (Long-Term Support) – Has an extended support period
  - Preview – Not supported for production use
  - Out of support – No longer supported

`dotnet.microsoft.com/download`

Introduction

Visual Studio and Visual Studio Code

- Visual Studio is available for Windows and macOS
  - Full-featured IDE
- Visual Studio Code is available for Windows, macOS, and Linux
  - Includes IntelliSense and debugging features
  - Thousands of extensions are available for additional functionality

`visualstudio.microsoft.com`

Introduction

Visual Studio and Visual Studio Code

- JetBrains also offers an IDE for .NET development called Rider
- Available for Windows, macOS, and Linux
- Includes advanced capabilities in the areas of refactoring, unit testing, and low-level debugging

`www.jetbrains.com/rider`

Web APIs with ASP.NET Core 6

.NET SDK

- Installation
- Version Management
- Command-Line Interface (CLI)

.NET SDK

Installation

- The .NET SDK is distributed using each supported platform's native install mechanism
- Requires administrative privileges to install
- A list of installed SDK versions is available by using the .NET Command Line Interface (CLI)

```
dotnet --list-sdks
```

- A complete list of all installed runtimes and SDKs (as well as the default version) is also available

```
dotnet --info
```


.NET SDK

Version Management

- By default, CLI commands use the newest installed version of the SDK
  - This behavior can be overridden with a global.json file

```
{
  "sdk": {
    "version": "3.1.415"
  }
}
```

  - Will be in effect for that directory and all sub-directories

.NET SDK

Version Management

- Use of global.json files can allow developers to experiment with newer versions of the SDK while ensuring consistency for specific projects
- Include a global.json file in a source control repository to ensure every member of the team is using the same version of the SDK
  - Will generate an error if the specified SDK version is not present on the system

.NET SDK

Version Management

- While the SDK version (tooling) is specified using a global.json file, the runtime version is specified within the project file

```
<PropertyGroup>
  <TargetFramework>net6.0</TargetFramework>
</PropertyGroup>
```

.NET SDK

Version Management

- When an application is launched, it will automatically use the newest available runtime with the same major and minor version number
  - For example, if version 6.0 is specified, the application will use automatically use the 6.0.8 runtime but will not automatically use version 6.1 of the runtime
- Allows for system administrators to apply security patches and runtime bug fixes without the need to recompile and re-deploy the application
- Behavior can be overridden by specifying a RollForward policy value

.NET SDK

Version Management

- The target framework for a project can be an older version than the version of the SDK that you are using
  - For example, you can use version 6 of the SDK to build an application that targets the .NET Core 3.1 runtime

```
<PropertyGroup>
  <TargetFramework>netcoreapp3.1</TargetFramework>
</PropertyGroup>
```

- Recommended approach – Use the newest version of the tools possible and choose a runtime target based on your deployment environment

.NET SDK

Command-Line Interface (CLI)

- Many higher-level tools and IDEs use the CLI "under-the-covers"
- CLI commands consist of the driver (“dotnet”), followed by a “verb” and then possibly some arguments and options

.NET SDK

Command-Line Interface (CLI)

- dotnet new
  - Create a new project from an available template
- dotnet restore
  - Restore the dependencies for a project (download missing NuGet packages)
- dotnet build
  - Build a project and all its dependencies
- dotnet run
  - Run an application from its source code (performs a build if necessary)

.NET SDK

Command-Line Interface (CLI)

- dotnet test
  - Execute unit tests for a project
- dotnet publish
  - Pack an application and its dependencies into a folder for deployment
- And many more…

Lab 1

.NET SDK

- Create and run a .NET 6 console application using the CLI
- Create and run an ASP.NET Core application using the CLI

Web APIs with ASP.NET Core 6

What's New in C#

- Introduction
- Record Types
- Init Only Setters
- Nullable Reference Types
- Global Using Directives
- File-Scoped Namespace Declarations
- Top-Level Statements

What's New in C#

Introduction

- C# 9 introduced with .NET 5
- C# 10 introduced with .NET 6
- Several new features and improvements
  - Complete list available in the online documentation

What's New in C#

Record Types

- Every type in .NET is either a value type or a reference type
  - Struct is a value type
  - Class is a reference type
- Values types are recommended to be defined as immutable and are copied on assignment
  - Use value semantics for equality
  - Supports additional safety and optimizations especially for concurrent programming with shared data

What's New in C#

Record Types

- The record type introduced in C# 9 allows you to easily define an immutable reference type that supports value semantics for equality

```
public record Person
{
    public string LastName { get; }
    public string FirstName { get; }

    public Person(string first, string last) =>
        (FirstName, LastName) = (first, last);
}
```

What's New in C#

Record Types

- None of the properties of a record can be modified once it's created
- Records do support inheritance
- It is easy to create a new record from an existing one via the with keyword

```
var person = new Person("Joe", "Smith");

Person brother = person with { FirstName = "Bill" };
```


What's New in C#

Record Types

- Record types can be a very good fit for things like ViewModels and Data Transfer Objects (DTOs)

What's New in C#

Init Only Setters

- It is very convenient to initialize the properties of an object by using object initialization syntax

```
var product = new Product { Name = "Bread", Price = 2.50 }
```

- However, in the past, this was only possible by defining the properties as writable

What's New in C#

Init Only Setters

- In C# 9, it is now possible to define properties with init only setters
- Properties can be set as part of object initialization but become read-only after that

```
public class Product
{
  public string Name { get; init; }
  public double Price { get; init; }
}
```

What's New in C#

Nullable Reference Types

- By default, value types in .NET cannot be set to null
  - A variable can be defined as a nullable value type so that it can store a null value

```
int? num = null;
```

- Reference types can store null and default to null if not provided with an initial value

```
Product p;  // p is null
```


What's New in C#

Nullable Reference Types

- The most common exception encountered during .NET development is the NullReferenceException
  - Occurs when attempting to access the member of an object that is null
- Safety can be significantly improved by using types that cannot be null unless explicitly identified to allow it

What's New in C#

Nullable Reference Types

- C# 8 introduced the idea of nullable reference types
  - Like values types, reference types are not allowed to be null unless the variable is defined as nullable
- Because of the impact on existing code, this feature was not enabled by default
- Could be enabled via the Nullable annotation in the project file

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

- In .NET 6 project templates, this is now included by default

What's New in C#

Nullable Reference Types

- If enabled, compiler warnings will be generated when…
  - Setting a non-nullable reference type to null
  - Defining a reference type that does not initialize all non-nullable reference type members as part of construction
  - Dereferencing a possible null reference without checking for null (or using the null-forgiving operator)

```
string fn = person!.FirstName;
```

What's New in C#

Nullable Reference Types

- It is a good idea to enable nullable reference types for new projects
- Refactoring an existing application to use nullable reference types could require a significant amount of effort

What's New in C#

Global Using Directives

- C# 10 introduces global using directive support
- If the global keyword is present, the using directive will be in effect for every file in the project

```
global using EComm.Core;
```

- Can be in any file but a good practice is to have a separate cs file for all the project's global using directives

What's New in C#

Global Using Directives

- In .NET 6, global using directives for common system namespaces can be included implicitly via a project setting

```
<ImplicitUsings>enable</ImplicitUsings>
```

- This setting is included in new projects by default
- For an ASP.NET project, there are a total of 16 namespaces that will be implicitly referenced

What's New in C#

File-Scoped Namespace Declarations

- Typically, code within a namespace is defined within curly braces

```
namespace Acme.Models
{
  ...
}
```

- C# 10 allows for a namespace declaration to specify that all the code within a file belongs to a specific namespace

```
namespace Acme.Models;

...
```

What's New in C#

Top-Level Statements

- A .NET application requires an entry point function named Main defined within a static class

```
class Program {
  static void Main(string[] args) {
    Console.WriteLine(“Hello, World!“);
  }
}
```

- The C# 10 compiler can recognize executable code that is outside of a class as the code for the entry point and generate the necessary function and static class for you

```
Console.WriteLine(“Hello, World!“);
```


What's New in C#

Top-Level Statements

- ASP.NET Core 6 project templates combine the implicit using feature with top-level statements to minimize the amount of code required in Program.cs

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

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

app.Run();
```

Web APIs with ASP.NET Core 6

Application Architecture

- Introduction
- NuGet Packages
- Application Startup
- Hosting Environments
- Middleware and the Request Pipeline
- Services and Dependency Injection

Application Architecture

Introduction

- Single stack for Web UI and Web APIs
- Modular architecture distributed as NuGet packages
- Flexible, environment-based configuration
- Built-in dependency injection support
- Support for using an MVC-based architecture or a more page-focused architecture by using Razor Pages
- Blazor allows for the implementation of client-side functionality using .NET code

Application Architecture

NuGet Packages

- NuGet is a package manager for .NET
  - `www.nuget.org`
- All the libraries that make up .NET 6 (and many 3rd-party libraries) are distributed as NuGet packages
- NuGet package dependencies are stored in the project file

```
<PackageReference Include="Microsoft.EntityFrameworkCore" Version="6.0.0" />
```


Application Architecture

NuGet Packages

- The dotnet restore command will fetch any referenced NuGet packages that are not available locally
- Uses nuget.org as the package source by default
- Additional or alternative package sources (remote or local) can be specified by using a nuget.config file

Application Architecture

NuGet Metapackages

- Metapackages are a NuGet convention for describing a set of packages that are meaningful together
- Every .NET Core project implicitly references the Microsoft.NETCore.App package
  - ASP.NET Core projects also reference the Microsoft.AspNetCore.App package
- These two metapackages are included as part of the runtime package store
  - Available anywhere the runtime is installed

Application Architecture

Application Startup

- When an ASP.NET Core application is launched, the first code executed is the application's Main method
  - Generated by the compiler if using top-level statements
- Code in the Main method is used to…
  - Create a WebApplication object
  - Configure application services
  - Configure the request processing pipeline
  - Run the application

Application Architecture

Application Startup

- WebApplication's CreateBuilder method is typically used to create the WebApplicationBuilder object
- When the WebApplicationBuilder is created, it loads configuration information from…
  - appsettings.json and appsettings.{Environment}.json
  - User secrets (when running in Development environment)
  - Environment variables and command-line arguments

Application Architecture

Application Startup

- After the WebApplicationBuilder has been initialized, application services can be added
- WebApplicationBuilder's Build method is used to construct the WebApplication object and initialize the dependency injection system
- The WebApplication object is used to configure the request processing pipeline

Application Architecture

Application Startup

- A collection of framework services are automatically registered with the dependency injection system
  - IHostApplicationLifetime
    - Used to handle post-startup and graceful shutdown tasks
  - IHostEnvironment / IWebHostEnvironment
    - Has many useful properties (ex. EnvironmentName)
  - ILoggerFactory
  - IServer
  - And many others…

Application Architecture

Application Startup

- The environment for local machine development can be set in the launchSettings.json file
  - Overrides values set in the system environment
  - Only used on the local development machine
  - Is not deployed
  - Can contain multiple profiles

Application Architecture

Application Startup

- Project templates include HTTPS redirection middleware by default
- ApplicationBuilder also loads configuration information from a section named Kestrel

```
"Kestrel": {
  "Endpoints": {
    "Http": {
      "Url": "http://localhost:5000"
    },
    "Https": {
      "Url": "https://localhost:5001",
      "Certificate": {
        "Path": "<path to .pfx file>",
        "Password": "<certificate password>"
```


Application Architecture

Application Startup

- By default, clients can use HTTP/2 when selected during the TLS handshake; otherwise, HTTP/1.1 is used
- Additional Kestrel configuration options are described in the documentation

Application Architecture

Hosting Environments

- EnvironmentName property can be set to any value
- Framework-defined values include:
  - Development
  - Staging
  - Production (default if none specified)
- Typically set using the ASPNETCORE_ENVIRONMENT environment variable
- Can also be configured via launchSettings.json

Application Architecture

Middleware

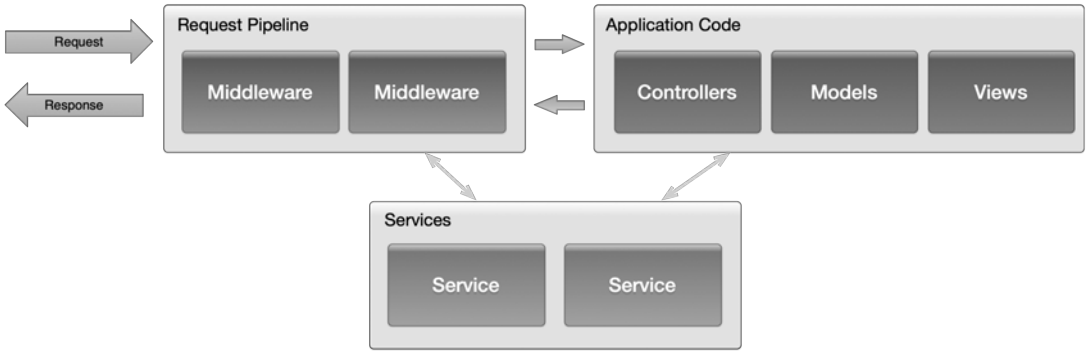
- ASP.NET uses a modular request processing pipeline
- The pipeline is composed of middleware components
- Each middleware component is responsible for invoking the next component in the pipeline or short-circuiting the chain
- Examples of middleware include…
  - Request routing
  - Handling of static files
  - User authentication
  - Response caching
  - Error handling

Application Architecture

Services

- ASP.NET Core also includes the concept of services
- Services are components that are available throughout an application via dependency injection
- An example of a service would be a component that accesses a database or sends an email message

Application Architecture



Application Architecture

Pipeline

- The last piece of middleware in the pipeline is typically the routing middleware
- Routes the incoming request to a controller
- Instead of using controllers, the new minimal API framework in .NET 6 can be used (more on this later)

Lab 2

Application Architecture

- Create a new ASP.NET Core Web API project in Visual Studio
- Run and test the application

Web APIs with ASP.NET Core 6

Application Configuration

- Middleware
- Services
- Configuration Providers and Sources
- Configuration API
- Options Pattern

Application Configuration

Middleware

- A middleware component typically adds an extension method to IApplicationBuilder for adding it to the pipeline
  - By convention, these methods start with the prefix "Use"

```
app.UseHttpsRedirection();
app.UseAuthorization();
```

- The order in which middleware is added to the pipeline can be important
  - Determines the order of execution
  - As an example, it would be very important for authentication middleware to execute before some caching middleware that could return a cached response

Application Configuration

Services

- Services are components that are available throughout an application via dependency injection
- The lifetime of a service can be…
  - Singleton (one instance per application)
  - Scoped (one instance per web request)
  - Transient (new instance each time component requested)
- An example of a service would be a component that accesses a database or sends an email message

Application Configuration

Services

- Services are typically added via extension methods available on IServiceCollection

```
builder.Services.AddDbContext<ApplicationDbContext>(...);
builder.Services.AddScoped<IEmailSender, MyEmailSender>();
builder.Services.AddScoped<ISmsSender, MySmsSender>();
```

- Most methods include the service lifetime as part of the method name (e.g., AddScoped)
- The AddDbContext method is a custom method specifically for adding an Entity Framework DbContext type as a service

Application Configuration

Services

- Services are available throughout the application via dependency injection
- A common practice is to follow the Explicit Dependencies Principle
  - Controllers include all required services as constructor parameters
  - System will provide an instance or throw an exception if the type cannot be resolved via the DI system

```
public class ProductController : ControllerBase
{
  public ProductController(IEmailSender emailSender) {
    ...
  }
}
```


Application Configuration

Services

- If using controllers (and not the minimal API feature), some services must be added during startup

```
builder.Services.AddControllers();
```

- Other methods are available if using views of razor pages

Application Configuration

Configuration Providers and Sources

- Before ASP.NET Core, application settings were typically stored in an application's web.config file
- ASP.NET Core introduced a completely new configuration infrastructure
  - Based on key-value pairs gathered by a collection of configuration providers that read from a variety of different configuration sources

Application Configuration

Configuration Providers and Sources

- Available configuration sources include:
  - Files (INI, JSON, and XML)
  - System environment variables
  - Command-line arguments
  - In-memory .NET objects
  - Azure Key Vault
  - Custom sources

Application Configuration

Configuration Providers and Sources

- The default WebApplicationBuilder adds providers to read settings (in the order shown) from:
  - appsettings.json
  - appsettings.{Environment}.json
  - User secrets
  - System environment variables
  - Command-line arguments
- Values read later override ones read earlier

Application Configuration

Configuration API

- The configuration API provides the ability to read from the constructed collection of name-value pairs
- An object of type IConfiguration is available to be used via dependency injection

```
public class HomeController : ControllerBase
{
  public HomeController(IConfiguration configuration)
  {
    _emailServer = configuration["EmailServer"];
  }
}
```

Application Configuration

Configuration API

- Hierarchical data is read as a single key with components separated by a colon

```
{
  "Email": {
    "Server": "gmail.com",
    "Username": "admin"
  }
}
```

```
public class HomeController
{
  public HomeController(IConfiguration configuration)
  {
    _emailServer = configuration["Email:Server"];
  }
}
```


Application Configuration

Options Pattern

- The options pattern can be used to provide configuration information to other components within your application as strongly-typed objects via dependency injection

```
public class EmailOptions
{
  public string Server { get; set; }
  public string Username { get; set; }
}
```

```
builder.Services.Configure<EmailOptions>(Configuration.GetSection("Email"));
```

```
public HomeController(IOptions<EmailOptions> emailOptions)
{
  _emailOptions = emailOptions;
}
```

Web APIs with ASP.NET Core 6

Request Routing

- RESTful Services
- Endpoint Routing
- Route Attributes
- Route Templates
- Route Constraints
- Route Template Precedence

Request Routing

RESTful Services

- When configuring request routing, you should try to maintain a RESTful API
- Clean, extension-less URLs that identify resources
- Use of the correct HTTP verbs within an API
- Avoid query string parameters except for ancillary data that is related to the presentation of the information
  - Sorting key, current page number, etc.

Request Routing

Endpoint Routing

- Routing is responsible for mapping request URIs to endpoints and dispatching incoming requests to those endpoints
- Routing can also be used to generate URLs that map to endpoints
  - Eliminates hardcoded URLs that would need to be updated when the routing configuration changes

Request Routing

Endpoint Routing

- A collection of extension methods on ApplicationBuilder are available for adding different types of endpoints
  - All start with the word "Map"
- Verb-based methods make it easy to configure simple endpoints
  - MapGet, MapPost, MapPut, MapDelete, etc.
- MapControllers will configure and add an endpoint for each controller action defined in the application

Request Routing

Route Attributes

- If using controllers, attributes can be used to define the routing information used to construct the endpoints
- The Route attribute will create an endpoint for all HTTP verbs

```
[Route("products")]
public IEnumerable<Product> GetAllProducts() { ... }
```

- Verb-specific attributes should be used to define an endpoint for a specific HTTP verb

```
[HttpGet("products")]
public IEnumerable<Product> GetAllProducts() { ... }
```


Request Routing

Route Attributes

- A controller-level attribute can be used to specify a prefix for all the actions of the controller

```
[Route("[controller]")]
public class ProductController : ControllerBase
{
  [HttpGet("{id}")]
  public Product Get(int id) { ... }
}
```

- In the example above, a request for a product would use a URL of https://example.com/product/6

Request Routing

Route Templates

- Tokens within curly braces define route value parameters which will be bound if the route is matched
  - You can define more than one route value parameter in a route segment, but they must be separated by a literal value

```
site/{name}/{id}
```

```
{language}-{region}/library/{topic}
```

```
{language}{region}/{topic}
```


Request Routing

Route Templates

- Route value parameters can have default values
  - The default value is used if no value is present in the URL for the parameter

```
products/{sort=price}
```

- Route value parameters may also be marked as optional
  - When bound to an action parameter, the value will be null (reference type) or zero (value type)

```
product/{id?}
```

Request Routing

Route Templates

- The catch-all parameter (identified using an asterisk) allows for a route to match a URL with an arbitrary number of parameters

```
query/{category}/{*path}
```

```
http://localhost/query/people/hr/managers
```

```
public IActionResult Query(string category, string path)
{
  // category = "people"
  // path = "hr/managers"
}
```


Request Routing

Route Constraints

- A route value parameter can include an inline constraint
- URLs that do not match the constraint are not considered a match
- Multiple constraints can be specified for one parameter

```
products/{id:int}
```

```
products/{id:range(100, 999)}
```

```
employees/{ssn:regex(d{3}-d{2}-d{4})}
```

```
products/{id:int:range(100, 999)}
```

Request Routing

Route Constraints (Partial List)

Constraint	Example Route	Example Match
int	{id:int}	123
bool	{active:bool}	true
datetime	{dob:datetime}	2016-01-01
guid	{id:guid}	7342570B-44E7-471C-A267-947DD2A35BF9
minlength(value)	{username:minlength(5)}	steve
length(min, max)	{filename:length(4, 16)}	Somefile.txt
min(value)	{age:min(18)}	19
max(value)	{age:max(120)}	91
range(min, max)	{age:range(18, 120)}	91
alpha	{name:alpha}	Steve
regex(expression)	{ssn:regex(d{3}-d{2}-d{4})}	123-45-6789

Request Routing

Route Constraints

- Route constraints should be used to help determine the route that should be used but should not be used for the validation of input values
- If a matching route is not found, the response from the server will be a 404 (resource not found)
- Invalid input should typically result in a different response (e.g., 400 with an appropriate error message)

Request Routing

Route Template Precedence

- Each route template is assigned a value by the system based on how specific it is
- Literal segments are considered more specific than parameter segments

`/hello/customer` `/hello/{name}`

- A parameter segment with a constraint is considered more specific than one without a constraint
- The Order property of an endpoint can be used to override the default precedence behavior
- If a request matches multiple endpoints with the same precedence, an AmbiguousMatchException is thrown at runtime

Web APIs with ASP.NET Core 6

Models

- Introduction
- Persistence Ignorance
- Dependency Inversion
- Clean Architecture
- Asynchronous Data Access
- Object-Relational Mapping
- Entity Framework Core
- Dapper ORM

Lab 3

Models

- Create a database with some sample data

Models

Introduction

- Models represent "real world" objects the user is interacting with
- Entities are the objects used during Object-Relational Mapping and provide a way to obtain and persist model data
- The term Data Transfer Object (DTO) is often used to describe an object that carries data between different processes or subsystems
  - A single DTO may contain multiple different entities, exclude some entity properties, or use different property names
  - In a Web API application, the object that get serialized into JSON is often a DTO

Models

Persistence Ignorance

- The model data typically comes from an external source (database, web service, file, etc.)
- For better maintainability and testability, it is a best practice to use a data access component to encapsulate the details about where the model data comes from
- In ASP.NET, data access should be performed by a service made available via dependency injection
  - Makes it easy to test components independently with hard-coded data (no database)

Models

Dependency Inversion

- One of the SOLID design principles is Dependency Inversion
- "High-level modules should not depend on low-level models. Both should depend on abstractions."
  - The web application/service should not be built based on how the data access library was designed
  - An interface should be used to define the functionality that the web application requires
  - The data access library should provide a component that implements the required interface

Models

Clean Architecture

- To help facilitate reuse and a clear separation of concerns, we will apply some principles from a design philosophy known as clean architecture
- Our models (business/domain objects) and data access interface (abstraction) will be defined in a core library
- Our technology-specific data access implementation will be defined in an infrastructure library

Models

Clean Architecture

- The extra work necessary to achieve clean separation will provide more flexibility when it comes to…
  - Unit testing
  - Comparing different data access technologies
  - Switching to a different persistence mechanism (e.g., microservices)

Models

Asynchronous Data Access

- When performing IO-bound operations (database access, web service calls, etc.), it is a best practice to perform that work asynchronously
- Allows for the efficient use of thread resources
  - Thread pool threads can be used to handle other incoming requests while the IO-bound operation is in progress
  - Improves the scalability of a web application

```
public async Task<IEnumerable<Product>> GetAllProducts()
{
  return await _repository.GetProductsAsync();
}
```


Lab 4

Models

- Add a class library for the entity types and data access abstraction
- Add a class library for the data access implementation

Models

Object-Relational Mapping

- If a data access component communicates with a relational database, a necessary task will be to convert between relational data and C# objects
- This can be done manually by with ADO.NET, or several frameworks exist that can help with this task
  - Entity Framework Core
  - Dapper (3rd-party micro-ORM)
  - AutoMapper (mapping one object to another)

Models

Entity Framework Core

- Modeling based on POCO entities
- Data annotations
- Relationships
- Change tracking
- LINQ support
- Built-in support for SQL Server and Sqlite (3rd-party support for Postgres, MySQL, and Oracle)

Models

Entity Framework Core

- By creating a subclass of DbContext, EF Core can populate your entity objects and persist changes

```
public class ECommContext : DbContext
{
  public DbSet<Product> Products { get; set; }
}
```

- The DbContext can be used to create a new database based on the definition of your model objects or it can work with a database that already exists (as we will do)
- The Migrations feature of EF Core can be used to incrementally apply schema changes to a database (beyond the scope of this course)

Models

Entity Framework Core

- EF Core will make certain assumptions about your database schema based on your entity objects
- For example, EF Core will assume the database table names will match the name of each DbSet property

```
public class ECommContext : DbContext
{
  public DbSet<Product> Products { get; set; }
}
```

Models

Entity Framework Core

- To specify different mappings, you can use data annotations on your entities or use EF's fluent API

```
[Table("Product")]
public class Product
{
  [Column("Name")]
  public string ProductName { get; set; }
}
```

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
  modelBuilder.Entity<Product>().ToTable("Product");

  modelBuilder.Entity<Product>().Property(p => p.ProductName)
                                .HasColumnName("Name");
}
```


Models

Entity Framework Core

- Objects retrieved from the context are automatically tracked for changes
- Those changes can be persisted with a call to SaveChanges

```
Product product = _context.Products(p => p.Id == id);
product.ProductName = "Something else";
_context.SaveChanges();
```

Models

Entity Framework Core

- EF Core will not automatically load related entities
- The Include method can be used to perform "eager loading" of one or more related entities

```
_dataContext.Products.Include(p => p.Supplier)
  .SingleOrDefault(p => p.Id == id);
```


Models

Entity Framework Core

- EF Core sends the SQL it generates to the logging system when executed
- Interception API can also be used to obtain or modify the SQL

```
public class HintCommandInterceptor : DbCommandInterceptor
{
  public override InterceptionResult ReaderExecuting(DbCommand command,
      CommandEventData eventData, InterceptionResult result)
  {
    command.CommandText += " OPTION (OPTIMIZE FOR UNKNOWN)";
    return result;
  }
}
```

```
services.AddDbContext(b => b.UseSqlServer(connStr)
    .AddInterceptors(new HintCommandInterceptor()));
```

Models

Entity Framework Core

- EF Core can also be used to execute custom SQL or call a stored procedure

```
var products = context.Products
    .FromSqlRaw("SELECT * FROM dbo.Products")
    .ToList();
```

```
var product = context.Products
    .FromSqlRaw("EXECUTE dbo.GetProduct {0}", id)
    .SingleOrDefault();
```

- In the example above, EF Core uses an ADO.NET parameterized query and SQL injection is not a concern
  - Still an issue if the entire string is constructed first and then passed to FromSqlRaw

Models

Entity Framework Core

- The entity classes can be defined manually, or the code for them can be automatically generated
  - CLI tools
  - Package Manager Console tools in Visual Studio

Models

Entity Framework Core

- EF Core is a large topic and in-depth coverage is beyond the scope of this course
  - Inheritance
  - Shadow Properties
  - Cascading Updates and Deletes
  - Transactions
  - Concurrency Conflicts
  - Migrations

`docs.microsoft.com/en-us/ef/core/`

Models

Dapper ORM

- Dapper is an open-source ORM framework that has become a very popular alternative to Entity Framework

`github.com/StackExchange/Dapper`

- Has less features than EF but provides a good high-performance "middle-ground" between ADO.NET and EF
- Dapper is not specifically covered in this course
  - The labs include alternate data access implementations that use Dapper as well as raw ADO.NET

Lab 5

Data Access

- Create an EF-based implementation of IRepository

Web APIs with ASP.NET Core 6

Controllers

- Responsibilities
- Requirements and Conventions
- Dependencies
- Action Results

Controllers

Responsibilities

- The action executed for a particular endpoint is typically a method of a controller
  - The new minimal API framework provides an alternative approach
- A controller may need to retrieve or make modifications to model data
- The controller also often determines the appropriate type of response to return
  - HTML, JSON, XML, redirection, error, etc.

Controllers

Responsibilities

- Controller methods that are reachable via the routing system are referred to as controller actions
- Any public method of a controller can be an action if a valid route to that action exists

Controllers

Requirements and Conventions

- For a class to act as a controller, it must…
  - Be defined as public
  - Have a name that ends with Controller or inherit from a class with a name that ends with Controller
- Common conventions (not requirements) are...
  - Place all controllers in a root-level folder named Controllers
  - Inherit from a system class called Controller (or its subclass ControllerBase)
    - Provides many helpful properties and methods

Controllers

Dependencies

- It is a recommended best practice for controllers to follow the Explicit Dependencies Principle
- Specify required dependencies via constructor parameters that can be supplied via dependency injection

```
public class HomeController : Controller
{
  private IEmailSender _emailSender;

  public HomeController(IEmailSender es) {
    _emailSender = es;
  }
}
```

Controllers

Action Results

- IActionResult implemented by a variety of different return types
- Framework uses the ExecuteResultAsync when creating the HTTP response

```
public IActionResult Index()
{
  var result = new ContentResult()
  result.content = "Hello, World!";
  return result;
}
```

- Writing directly to the response should be avoided
  - Adds a dependency to the HTTP context
  - Make things like unit testing more difficult

Controllers

Action Results

- When the return type does not implement IActionResult, a content result will be created implicitly

```
public int Sum(int x, int y)
{
  return x + y;
}
```

```
public IActionResult Sum(int x, int y)
{
  int retVal = x + y;
  return Content(retVal.ToString());
}
```

Controllers

Action Results

- API controllers will typically return an entity type or a DTO

```
[HttpGet("products")]
public IEnumerable<Product> GetAllProducts()
```

- System will create an IActionResult and look at the incoming request to support content negotiation
- IActionResult is still useful when an action can return different return types

```
public IActionResult GetProduct(int id)
{
    if (!_repository.TryGetProduct(id, out var product)) {
        return NotFound();
    }
    return Ok(product);
}
```


Controllers

Asynchronous Controller Actions

- It is common for a controller action to invoke an asynchronous method to perform an IO-bound operation
  - Database access, web service call, etc.
- The action should be marked as async with a return type of Task<T> and await used with the asynchronous method

```
public async Task<IEnumerable<Product>> Index()
{
  var products = await _repository.GetAllProducts();
  return products;
}
```

Controllers

Asynchronous Controller Actions

- Making an action asynchronous does not change the experience for the client
  - No response is sent until the entire action is complete
- Can improve application scalability by allowing the thread pool thread to handle other incoming requests while waiting for the IO-bound operation to complete
- It is also possible to accept a CancellationToken that can be used to handle the cancellation of a long-running request

```
public async Task<IActionResult> Index(CancellationToken token)
{
  var products = await _repository.GetAllProducts(token);
  return Ok(products);
}
```


Lab 6

Controllers

- Register a service that returns an IRepository
- Modify a controller to accept a dependency
- Return a response that includes database data

Web APIs with ASP.NET Core 6

Web APIs

- API Controllers
- OpenAPI / Swagger
- Testing APIs
- Retrieval Operations
- Model Binding
- Update, Create, and Delete Operations
- Cross-Origin Request Sharing (CORS)

Web APIs

API Controllers

- ASP.NET Core includes a class named ControllerBase
  - Includes many properties and methods for handling HTTP requests
- The Controller class inherits from ControllerBase and adds support for views
- If creating a controller that does not have any views, you should inherit directly from ControllerBase

Web APIs

API Controllers

- An API controller should be decorated with the ApiController attribute

```
[ApiController]
public class ProductApiController : ControllerBase
```

  - Automatic HTTP 400 responses for validation failures
  - Problem details for error status codes

Web APIs

OpenAPI / Swagger

- OpenAPI is a specification for describing REST APIs
- Swagger is a collection of tools that work with OpenAPI
  - SwaggerDocument objects expose data about the API in JSON format (openapi.json)
  - Swagger UI is a dynamically generated web-based UI that can be used to view and test API methods

Web APIs

OpenAPI / Swagger

- There are two main OpenAPI implementations for .NET
  - Swashbuckle
  - NSwag
- Both include an embedded version of Swagger UI
  - Made available via middleware

Web APIs

OpenAPI / Swagger

- By default, the API project templates include a reference to Swashbuckle.AspNetCore
- SwaggerDocument generation is handled by a service

```
services.AddSwaggerGen();
```

- Document availability and Swagger UI is configured via middleware components

```
app.UseSwagger();
app.UseSwaggerUI();
```

Web APIs

OpenAPI / Swagger

- The ProducesResponseType attribute should be used when defining Web API actions

```
[HttpPost]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public ActionResult<Product> Create(Product product)
```

- Used by tools like Swagger to generate more descriptive documentation

Web APIs

OpenAPI / Swagger

- Actions (or entire controllers) can be omitted from the Swagger document generation process by using the ApiExplorerSettings attribute

```
[ApiExplorerSettings(IgnoreApi = true)]
public class ErrorController : Controller
```

Web APIs

Testing APIs

- API endpoints that are exposed via GET are easy to test using a web browser
- For other verbs, it can be helpful to have a tool that can be used to craft custom HTTP requests
  - Postman application is very popular (getpostman.com)
  - Many other options are available

Web APIs

Testing APIs

- Microsoft recently introduced a new tool called the HTTP REPL

```
dotnet tool install -g Microsoft.dotnet-httprepl
```

- Command-line tool for making HTTP requests
- Supports most of the HTTP verbs
- Can use Swagger documents to discover the endpoints

```
> https://localhot:5001/~ ls

Products   [get|post]
Customers  [get|post]
```

docs.microsoft.com/en-us/aspnet/core/web-api/http-repl

Web APIs

Retrieval Operations

- In a Web API, retrieval operations are performed with an HTTP GET request
- If successful, the response should use an HTTP 200 status code

```
[HttpGet("{id}")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
public async Task<IActionResult>> GetProduct(int id)
{
  var product = await _repository.GetProduct(id, includeSuppliers: true);
  if (product == null) return NotFound();
  return Ok(product);
}
```


Web APIs

Retrieval Operations

- There are several options available for altering the format of the JSON returned
  - Attributes
  - Custom formatter
  - Data projection

```
public async Task<IActionResult> GetProduct(int id)
{
  var product = await _repository.GetProduct(id, true);
  if (product == null) return NotFound();
  var retVal = new {
    Id = product.Id, Name = product.ProductName,
    Price = product.UnitPrice,
    Supplier = product.Supplier.CompanyName
  };
  return Ok(retVal);
}
```

Lab 7

Web API

- Add an API method for retrieving an individual product

Web APIs

Model Binding

- When a controller action is invoked, the model binding system will attempt to populate the parameters of the action with values from the request
  - Request body
  - Route value
  - Query strings
- Items above are listed in priority order (i.e., body values will take precedence over other values)

Web APIs

Model Binding

- If an action accepts an object parameter, the model binding system will create an instance of that type and attempt to populate its public properties with values from the request
- If validation errors occur during the model binding process, the IsValid property of the ModelState property will return false

```
public ActionResult Edit(ProductViewModel vm)
{
  if (ModelState.IsValid) { ... }
  ...
}
```


Web APIs

Model Binding

- It is important to ensure the model binding system does not alter values that you do not intend to be modified
  - Can lead to a security vulnerability known as over-posting
- Attributes can be used to define properties that should not participate in model binding

```
[BindNever]
public int EmployeeId { get; set; }
```

- Alternatively, use a DTO that only includes properties that are intended to participate in model binding

Web APIs

Update Operations

- In a Web API, update operations are performed with…
  - HTTP PUT – Replaces an existing resource
  - HTTP PATCH – Modifies part of an existing resource
- If successful, the response should be HTTP 204 (no content)

```
public async Task<IActionResult> PutProduct(int id, Product product)
{
  if (id != product.Id) return BadRequest();

  var existingProduct = await _repository.GetProduct(id);
  if (existingProduct == null) return NotFound();

  await _repository.SaveProduct(product);
  return NoContent();
}
```


Web APIs

Create Operations

- In a Web API, create operations are performed with an HTTP POST request
- If successful, the response should use an HTTP 201 (created) status code with a Location header set to the URI of the newly created resource
- The CreatedAtAction and CreatedAtRoute methods can be used to generate a correctly formatted response

```
return CreatedAtAction("GetProduct", new { id = product.Id }, product);
```

Web APIs

Delete Operations

- In a Web API, delete operations are performed with an HTTP DELETE request
- If successful, the response should use an HTTP 204 (no content) status code

```
return new NoContentResult();
```


Web APIs

Saving Changes with EF Core

- EF Core has a change tracking system
- A call to SaveChanges (or SaveChangesAsync) will cause EF to submit the necessary SQL to persist all pending changes
- An entity can be attached to a DbContext and then marked as modified

```
Products.Attach(product);
Entry(product).State = EntityState.Modified;
await SaveChangesAsync();
```

- Requires only one trip to the database but updates all columns of the entity

Web APIs

Saving Changes with EF Core

- If the entity is fetched via EF first and then modified, EF can determine which properties are different and then update only those columns

```
var product = Products.Single(p => p.Id == id);
product.ProductName = "Something Else";
product.UnitPrice = 15.99;
// set other properties
await SaveChangesAsync();
```

- Update query is more efficient, but complete operation requires two trips to the database

Lab 8

Web API

- Add an API method to update a product
- Add an API method to create a new product
- Add an API method to delete a product

Web APIs

Cross-Origin Resource Sharing (CORS)

- Browser security prevents a web page from making Ajax requests to another domain
- CORS is a W3C standard that allows a server to relax this policy
- A server can explicitly allow some cross-origin requests
- CORS is configured in ASP.NET Core via a service and middleware

```
services.AddCors();
```

```
app.UseCors(builder =>
    builder.WithOrigins("https://example.com"));
```


Web APIs with ASP.NET Core 6

Input Validation

- Introduction
- Data Annotations
- Model Binding

Input Validation

Introduction

- Whenever any data from the client is being used to perform an action, it is important to have data validation in place
  - Don't skip validation for HTTP header values, cookies, etc. (all are easy to modify)
- Client-side validation provides a good user experience and improved application scalability (less trips to the server)
- Server-side validation must also be provided
  - Client-side validation is easy to circumvent or may not be supported on the client

Input Validation

Data Annotations

- A variety of data annotations can be added to the type being used by the model binding system
- Data annotations are used by during model binding to perform server-side validation

```
public class ProductDto
{
  [Required]
  public string ProductName { get; set; }
```

- If a value is considered to be invalid, an error is added to ModelState and ModelState.IsValid will return false

Input Validation

Data Annotations

Attribute	Purpose
[Required]	Property value is required (cannot allow nulls)
[StringLength]	Specifies a maximum length for a string property
[Range]	Property value must fall within the given range
[RegularExpression]	Property value must match the specified expression
[Compare]	Property value must match the value of another property
[EmailAddress]	Property value must match the format of an email address
[Phone]	Property value must match the format of a phone number
[Url]	Property value must match the format of a URL
[CreditCard]	Property value must match the format of a credit card number

Input Validation

IValidatableObject

- For custom server-side validation, you can implement the IValidatableObject interface for the type being populated by the model binder
- Any errors returned are automatically added to ModelState by the model binder

```
public IEnumerable<ValidationResult> Validate(ValidationContext
                                                validationContext)
{
  var retVal = new List<ValidationResult>();
  if (BirthDate > HireDate) {
    retVal.Add(new ValidationResult("Employee cannot be
                                     hired before they were born"));
  }
  return retVal;
}
```

Lab 9

Input Validation

- Add input validation for when editing a product

Web APIs with ASP.NET Core 6

Error Handling

- Best Practices
- HTTP Error Status Codes
- Exception Handling Middleware

Error Handling

Best Practices

- Handle errors as best you can when they occur
- Record the error information and/or send a notification
- Provide the user with an appropriate response
  - Do not reveal information that a malicious user could potentially use against you (e.g., database schema information)
  - Use caution when generating content for an error response to avoid an error response that itself produces an error

Error Handling

HTTP Error Status Codes

- The HTTP protocol defines a range of status codes that signify an error
  - 4xx = client error (not found, bad request)
  - 5xx = server error
- It is a best practice to define an appropriate customized response that will be returned to the client for both of these cases

Error Handling

Exception Handling

- If an exception occurs during initial application startup…
  - The hosting layer logs a critical exception
  - The dotnet process crashes
- If running on IIS or IIS Express, the ASP.NET Core Module will return a 502.5 – Process Failure response for an application that is unable to start

Error Handling

Exception Handling Middleware

- The Exception Handling Middleware can be used to generate a customized response before a 500 leaves the server

```
if (app.Environment.IsProduction()) {
  app.UseExceptionHandler("/error");
}
```

Error Handling

Exception Handling Middleware

- Use IExceptionHandlerPathFeature to access information about the exception and original request path

```
var exceptionHandlerPathFeature =
    HttpContext.Features.Get<IExceptionHandlerPathFeature>();

if (exceptionHandlerPathFeature?.Error is FileNotFoundException)
  ExceptionMessage = "File error thrown";

if (exceptionHandlerPathFeature?.Path == "/index")
  ExceptionMessage += " from home page";
```


Lab 10

Error Handling

- Add a controller for centralized error handling
- Use the ExceptionHandler middleware
- Use IExceptionHandlerPathFeature to provide a custom response

Web APIs with ASP.NET Core 6

Logging

- Introduction
- Configuration
- ILogger
- Serilog and Seq

Logging

Introduction

- Just as important as error handling is the ability to record information about events that occur
- Logging of error information is essential for tracking down an issue that occurs in production
- It is sometimes helpful to record information about events that are not errors
  - Performance metrics
  - Authentication audit logs

Logging

Introduction

- ASP.NET Core has a logging API that works with a variety of logging providers
- Built-in providers allow you to log to the console and the Visual Studio Debug window
- Other 3rd-party logging frameworks can be used to provide many other logging options
  - Serilog
  - NLog
  - Log4Net
  - Loggr
  - elmah.io

Logging

ILogger

- Any component that wants to use logging can request an ILogger<T> as a dependency

```
public class ProductController : Controller
{
  public ProductController(IRepository repository,
    ILogger<ProductController> logger) { }
}
```

Logging

ILogger

- ILogger defines a set of extension methods for different verbosity levels
  - Trace (most detailed)
  - Debug
  - Information
  - Warning
  - Error
  - Critical

```
_logger.LogInformation("About to save department {0}", id);
```


Logging

ILogger

- The highest verbosity level written to the log is typically set in appsettings

```
"Logging": {
  "LogLevel": {
    "Default": "Debug",
    "System": "Information",
    "Microsoft": "Information"
  }
}
```

Logging

Serilog

- Serilog has become a popular choice for ASP.NET Core
  - Wide variety of destinations and formats
  - Can record structured event data

github.com/serilog/serilog-aspnetcore

Logging

Seq

- In many ASP.NET Core applications, the log data needs to be off-host and centralized (e.g., load-balanced environment)
- Seq is an open-source server that can accept logs via HTTP
  - Integrates with .NET Core, Java, Node.js, Python, Ruby, Go, Docker, and more

Logging

Maintainability

- One important note is that the logging framework(s) that you choose should not change how you write to the log (ILogger)
  - The only code that changes is in Program.cs

Web APIs with ASP.NET Core 6

Testing

- Introduction
- Unit Testing
- xUnit
- Testing Controllers
- Integration Testing

Testing

Introduction

- Testing your code for accuracy and errors is at the core of good software development
- Testability and a loosely-coupled design go hand-in-hand
- Even if not writing tests, keeping testability in mind helps to create more flexible, maintainable software
- The inherit separation of concerns in MVC applications can make them much easier to test

Testing

Introduction

- Unit testing
  - Test individual software components or methods
- Integration testing
  - Ensure that an application's components function correctly when assembled together

Testing

Unit Testing

- A unit test is an automated piece of code that involves the unit of work being tested, and then checks some assumptions about a noticeable end result of that unit

Testing

Unit Testing

- A unit of work is the sum of actions that take place between the invocation of a public method in the system and a single noticeable end result by a test of that system
- A noticeable end result can be observed without looking at the internal state of the system and only through its public API
  - Public method returns a value
  - Noticeable change to the behavior of the system without interrogating private state
  - Callout to a third-party system over which the test has no control

Testing

Unit Testing

- Good unit tests are…
  - Automated and repeatable
  - Easy to implement
  - Relevant tomorrow
  - Easy to run
  - Run quickly
  - Consistent in its results
  - Fully isolated (runs independently of other test)

Testing

Unit Testing

- A unit test is typically composed of three main actions
  - Arrange objects, creating and setting them up as necessary
  - Act on the object
  - Assert that something is as expected

Testing

Unit Testing

- Often, the object under test relies on another object over which you have no control (or doesn't work yet)
- A stub is a controllable replacement for an existing dependency in the system
  - By using a stub, you can test your code without dealing with the dependency directly
- A mock object is used to test that your object interacts with other objects correctly
  - Mock object is a fake object that decides whether the unit test has passed or failed based on how the mock object is being used by the object under test

Testing

xUnit

- A test project is a class library with references to a test runner and the projects being tested
- Several different testing frameworks are available for .NET
  - Visual Studio includes project templates for the MSTest, xUnit, and NUnit frameworks
- xUnit has steadily been gaining in popularity both inside and outside of Microsoft

Testing

xUnit

- Fact attribute is used to define a test that represents something that should always be true
- Theory attribute is used to define a test that represents something that should be true for a particular set of data

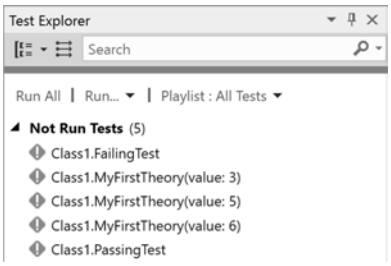
```
[Fact]
public void PassingTest()
{
  Assert.Equal(4, Add(2, 2));
}
```

```
[Theory]
[InlineData(3)]
[InlineData(5)]
[InlineData(6)]
public void MyFirstTheory(int value)
{
  Assert.True(IsOdd(value));
}
```


Testing

xUnit

- Tests can be run using the Visual Studio Test Explorer



- Tests can also be run by using the .NET Core command line interface

```
> dotnet test
```

Testing

Testing Controllers

- When looking to test a controller, ensure that all dependencies are explicit so that stubs and mocks can be used when needed
- When testing a controller action, check for things like…
  - What is the type of the response returned?
  - If a view result, what is the type of the model?
  - What does the model contain?

Lab 11

Unit Testing

- Create a new xUnit test project
- Define and run a simple test
- Create a stub object
- Define and run a test for a controller action

Testing

Integration Testing

- Integration tests check that an app functions correctly at a level that includes the app's supporting infrastructure
  - Request processing pipeline
  - Database
  - File system

Testing

Integration Testing

- The Microsoft.AspNetCore.Mvc.Testing package provides a collection of components to help with integration testing
  - Test web host
  - In-memory test server
  - WebApplicationFactory

Testing

Integration Testing

- ASP.NET Core's WebApplicationFactory class can used to create a host for the application

```
public class ECommAppFactory<TStartup> :
  : WebApplicationFactory<TStatup>where TStartup: class
{
  protected override void ConfigureWebHost(IWebHostBuilder builder)
  {
    builder.ConfigureServices(services =>
    {
      // configure app services here like in the app's
      // ConfigureServices method
    }
  }
}
```


Testing

Integration Testing

- Test classes can use the custom WebApplicationFactory to create an HttpClient

```
public class PageTests :
    IClassFixture<ECommApplicationFactory<EComm.Web.Startup>>
{
  private readonly HttpClient _client;
  private readonly CustomWebApplicationFactory<EComm.Web.Startup> _factory;

  public PageTests(CustomWebApplicationFactory<EComm.Web.Startup> factory)
  {
    _factory = factory;
    _client = factory.CreateClient(new WebApplicationFactoryClientOptions
    {
      AllowAutoRedirect = false
    });
}
```

Testing

Integration Testing

- Your HttpClient can issue the same HTTP requests that a browser would and receive the same HTTP responses

```
[Theory]
[InlineData("/")]
[InlineData("/Index")]
[InlineData("/About")]
public async Task Get_EndpointsSuccessAndContentType(string url)
{
  // Act
  var response = await _client.GetAsync(url);

  // Assert
  response.EnsureSuccessStatusCode(); // Status Code 200-299
  Assert.Equal("text/html; charset=utf-8",
    response.Content.Headers.ContentType.ToString());
}
```


Web APIs with ASP.NET Core 6

Security

- Authentication
- ASP.NET Core Identity
- Authorization
- Web API Authentication
- Secrets Management

Security

Authentication

- Authentication is the process of discovering the identity of an entity through an identifier and verifying the identity by validating the credentials provided by the entity
- It is common to validate the credentials once and generate a temporary unique token that is used to authorize other actions
  - The token can contain other information (claims) about the entity including group membership and privileges
  - Care must be taken to ensure the token is sent in a secure manner and the integrity of the token can be verified

Security

Authentication

- Authentication in ASP.NET Core is handled by the IAuthenticationService
  - Uses registered authentication handlers (IAuthenticationHandler)
  - Triggered by authentication middleware

Security

Authentication

- A configured authentication handler is also called a scheme
  - Registered in ConfigureServices as part of the call to AddAuthentication
  - Multiple schemes can be registered but a default scheme must be identified, or the scheme specified in the Authorize attribute

```
services.AddAuthentication(CookieAuthenticationDefaults.
        AuthenticationScheme).AddCookie();
```


Security

Authentication

- The RemoteAuthenticationHandler type is used for authentication that requires a remote authentication step
- Uses a callback to complete the authentication process
- Used for OAuth 2.0 and OpenID Connect (OIDC) based authentication
  - Facebook, Twitter, Google, Microsoft, etc.

Security

Authentication

- The authentication middleware is added by calling UseAuthentication in the Configure method
- If a request is not authenticated, authorization invokes a challenge using the default (or specified) authentication scheme
- Authentication challenge examples:
  - Redirection to a login page for a Web UI
  - 401 response with www-authentication: bearer header for a Web API

Security

Authentication

- If a request is authenticated but the action is not permitted for the authenticated user, a forbid action is triggered
- Forbid action examples:
  - Redirection to a page explaining the situation
  - 403 response

Security

ASP.NET Core Identity

- ASP.NET Core Identity is a complete, full-featured authentication provider for creating and managing logins
- Supports username/password login as well as external login providers such as Facebook, Google, Microsoft Account, Twitter and more
- Can use SQL Server or a custom credential store
- There is extensive documentation and several example projects

```
docs.microsoft.com/en-us/aspnet/core/
  security/authentication/identity
```


Security

Authorization

- The Authorize attribute can be used to authorize access to specific functionality
- Can be applied at the action or controller level
- If nothing else is specified, the attribute simply ensures the current user has been authenticated
- The AllowAnonymous attribute can be used to opt-out an action

```
[Authorize]
public class AdminController : Controller
{
  public IActionResult Dashboard() { ... }

  [AllowAnonymous]
  public IActionResult Login() { ... }
}
```

Security

Authorization

- With claims-based authorization, you can define a policy that specifies what claims must be present for a user to be authorized

```
services.AddAuthorization(options => {
  options.AddPolicy("AdminsOnly", policy =>
    policy.RequireClaim(ClaimTypes.Role, "Admin"));
});
```

- The Authorize attribute can then be used to enforce the policy

```
[Authorize(Policy="AdminsOnly")]
public IActionResult Dashboard() { ... }
```


Security

Authorization

- Programmatic authorization checks can also be performed within an action to enforce function-level access control

```
public IActionResult Delete(int id)
{
  if (User.HasClaim("SomeClaim")) {
    //
  }
}
```

```
public IActionResult Dashboard()
{
  foreach (var claim in User.Claims)
  {
    // decide what the user should see
  }
}
```

Security

Web API Authentication

- For API authentication, cookie-based authentication is typically not used, and it is not possible to directly present a UI to gather credentials
- In some scenarios, a simple form of authentication where the credentials are passed with each request is sufficient
  - HTTP Basic
  - Pre-shared key (PSK) authentication
- In a microservice architecture, an API Gateway can be used to perform authentication and proxy requests to other services

Security

Web API Authentication

- A modern flexible approach to API authentication is to use bearer token authentication
  - ASP.NET Core supports OAuth 2.0 and OpenID Connect

```
services.AddAuthentication(options => {
  options.DefaultAuthenticateScheme = AspNetCore.Authentication
    .JwtBearer.JwtBearerDefaults.AuthenticationScheme;

  options.DefaultChallengeScheme = AspNetCore.Authentication
    .JwtBearer.JwtBearerDefaults.AuthenticationScheme;
})
  .AddJwtBearer(options => {
    options.Authority = identityUrl;
    options.Audience = "EComm";
  });
```

Security

Web API Authentication

- Authority property is the address of the token-issuing authentication server
  - Used by the middleware to obtain the public key for validation of the token signature
- Audience property represents the resource the token grants access to
  - Value must match the parameter in the token

Security

Web API Authentication

- The token issuing server can be an external identity provider (Facebook, Twitter, Microsoft, etc.) or you can create your own using ASP.NET Core
- If issuing your own security tokens, it is recommended to use a third-party library to handle many of the security-related details
  - IdentityServer4
  - OpenIddict

Security

Web API Authentication

- It is also possible to create a custom authentication scheme / handler
  - Can create a subclass AuthenticationHandler and override HandleAuthenticateAsync
- When developing a custom authentication scheme, extreme care should be taken to ensure security vulnerabilities are not introduced

Lab 12

API Authentication

- Define a custom authentication scheme
- Secure an API method using the custom scheme

Security

Secrets Management

- Secrets are configuration values that are in some way sensitive
  - Connection strings
  - API keys
- It is a good practice for secrets not to be in source code or config files (things that end up in source control)
- Secrets should be made available to the production environment through a controlled means
  - Environment variables
  - Azure Key Vault
  - AWS Secrets Manager

Security

Secrets Management

- For local development, the Secret Manager tool can be used
- Secrets are stored in a JSON configuration file in a system-protected user profile folder on the local machine
- Use the init command or select Manage User Secrets in Visual Studio to enable user secrets for a project

```
dotnet user-secrets init
```

  - Adds an entry into the project file

```
<PropertyGroup>
  <TargetFramework>netcoreapp3.1</TargetFramework>
  <UserSecretsId>79a3edd0-2092-40a2-a04d-dcb46d5ca9ed</UserSecretsId>
</PropertyGroup>
```

Security

Secrets Management

- Secrets can be added via the command-line or by using the Manage User Secrets command in Visual Studio

```
dotnet user-secrets set "Movies:ServiceApiKey" "12345"
```

- CreateDefaultBuilder calls AddUserSecrets when the EnvironmentName is Development

Web APIs with ASP.NET Core 6

gRPC

- Introduction
- Protobuf
- Server
- Client
- Limitations

gRPC

Introduction

- gRPC is a language-agnostic, high-performance Remote Procedure Call (RPC) framework
  - Contract-first API development
  - Tooling available for many languages to generate strongly-typed clients
  - Support for streaming on the client and server
- In-depth coverage of gRPC is beyond the scope

gRPC

Protobuf

- By default, gRPC uses Protocol Buffers (protobuf)
- High-performance binary serialization format
- Services are defined in .proto files that are present on the server and client

```
syntax = "proto3";

service Greeter {
  rpc SayHello (HelloRequest) returns (HelloReply);
}

message HelloRequest {
  string name = 1;
}

message HelloReply {
  string message = 1;
}
```

gRPC

Protobuf

- Code generation is provided by the Grpc.Tools NuGet package
- .NET Core will then generate the necessary .NET types based on .proto files included in the project file

```
<ItemGroup>
  <Protobuf Include="Protos\greet.proto" />
</ItemGroup>
```


gRPC

Server

- To create the server, inherit from the base class generated by the tooling and override the relevant methods
  - If a method is not overridden, the server will return an HTTP 501 – Not Implemented response

```
public class GreeterService : Greeter.GreeterBase
{
  public override Task<HelloReply> SayHello(HelloRequest request,
    ServerCallContext context)
  {
    return Task.FromResult(new HelloReply {
      Message = "Hello " + request.Name
    });
  }
}
```

gRPC

Server

- An endpoint must be created to expose the service

```
app.UseEndpoints(endpoints =>
{
  endpoints.MapGrpcService<GreeterService>();
});
```


gRPC

Client

- To call a gRPC service, a channel is required

```
var channel = GrpcChannel.ForAddress("https://localhost:5001");
```

- A client can then be created (uses HttpClient under-the-covers)

```
var client = new ECommGrpc.ECommGrpcClient(channel);
```

- Calls can then be made using the strongly-typed client

```
var reply = await client.GetProductAsync(new ProductRequest { Id = 5 });
```

gRPC

Limitations

- gRPC depends on features implemented by HTTP/2
- Service cannot be called by JavaScript in the browser
  - Because of some HTTP/2 features that are not yet supported
- Will not work if a proxy exists between the client and server that does not fully support HTTP/2
- Does work very well as a means of high-performance communication between back-end services

`docs.microsoft.com/en-us/aspnet/core/grpc`

Web APIs with ASP.NET Core 6

Blazor

- Introduction
- Razor Components
- Blazor Server
- Blazor WebAssembly
- Hands-On Lab Exercise

Blazor

Introduction

- Blazor is a framework for building interactive client-side web UI with .NET code
- Provides an alternative to frameworks such as Angular and React
- Can be used without .NET on the server-side
  - Server just needs to deliver the runtime, assemblies, and static resources to the client

Blazor

Razor Components

- Blazor apps are based on razor components
  - UI element such as a dialog or data entry form
  - Written using the same Razor syntax used by traditional views and Razor Pages
- Components render into an in-memory representation of the browser's Document Object Model (DOM) called a render tree
  - Used to determine what updates should be applied to the UI

Blazor

Razor Components

```
@page "/counter"

<h1>Counter</h1>

<p>Current count: @currentCount</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    int currentCount = 0;

    void IncrementCount()
    {
        currentCount++;
    }
}
```


Blazor

Razor Components

- Razor Components can be used in one of two different configurations
  - Blazor Server
  - Blazor WebAssembly

Blazor

Blazor Server

- With Blazor Server, a JavaScript file (blazor.server.js) is sent to the browser
  - Establishes a persistent connection between the browser and the server using SignalR (WebSockets)
- Message is sent to the server when events occur in the browser
- C# code in the Razor Component is executed on the server
- UI updates are sent to the browser over the SignalR connection
- JavaScript interop allows for…
  - JavaScript functions to be invoked from C#
  - C# code to be triggered from JavaScript

Blazor

Blazor Server

- Blazor Server does maintain state information for each connected user
- Scalability of the application is therefore limited by available memory on the server
- Scalability varies based on the application, but Microsoft has tested this and found…
  - 5,000 concurrent users on instance with 1 vCPU and 3.5 GB of memory
  - 20,000 concurrent users on instance with 4 vCPU and 14 GB of memory

Blazor

Blazor WebAssembly

- Blazor WebAssembly (WASM) has been getting a lot of attention since its announcement
- Uses the same Razor Components as Blazor Server but the C# code executes on the client
- Microsoft has created a version of the .NET Runtime (CLR) that is written in WebAssembly
  - Allows .NET assemblies (that contain IL code) to be downloaded and executed on the client
  - Code executes in the same browser sandbox as JavaScript
  - Does not require the user to install something ahead of time (browser extension or plug-in)

Blazor

Blazor WebAssembly

- The user must be using a browser that supports WebAssembly
  - Firefox 52 (March 2017)
  - Chrome 57 (March 2017)
  - Edge 16 (September 2017)
  - Safari 11 (September 2017)

Blazor

Blazor WebAssembly

- When visiting a site that uses Blazor WebAssembly, a small JavaScript file is downloaded (blazor.webassembly.js) that…
  - Downloads the WebAssembly version of the .NET runtime (about 1 MB in size), the app, and all dependencies
- No state is maintained on the server
- .NET classes like HttpClient can be used to communicate with the server
  - Preconfigured in a new Blazor WebAssembly project

Lab 13

Blazor

- Create a new Blazor WebAssembly project
- Call the EComm Web API and display products

Web APIs with ASP.NET Core 6

Deployment

- dotnet publish
- Kestrel
- IIS
- Docker

Deployment

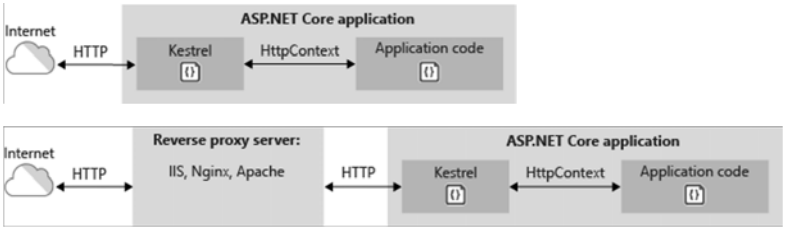
dotnet publish

- The dotnet publish command compiles an app and copies the files required for deployment into a publish folder
  - Used by Visual Studio's [Build > Publish] wizard
- When preforming a publish, the app can be…
  - Framework-dependent
    - Does not include the .NET runtime – correct version must already be present on the deployment machine
  - Self-contained
    - Does include the .NET runtime
    - Must choose the target architecture at publish time

Deployment

Kestrel

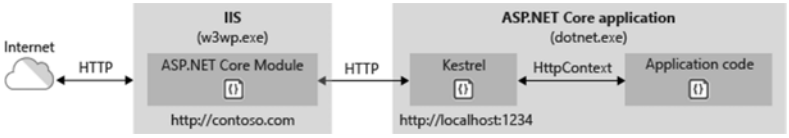
- The Kestrel web server is supported on all platforms where .NET is supported
- You can use Kestrel by itself or with a reverse proxy server



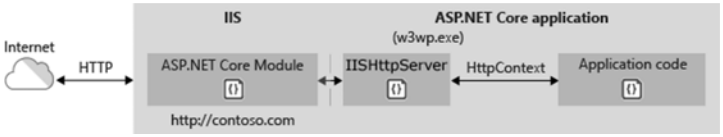
Deployment

IIS

- Out-of-process hosting



- In-process hosting



Deployment

Documentation

- There is extensive documentation for a wide variety of deployment scenarios
  - Apache httpd, Nginix, etc.

`docs.microsoft.com/en-us/aspnet/core/host-and-deploy`

Deployment

Docker

- Docker is an open platform that enables developers and administrators to build images, ship, and run applications in a loosely isolated environment called a container

`www.docker.com`

Deployment

Docker

- Developer – "Helps me to eliminate the 'works on my machine' problem"
  - Application is executed and debugged in the same container that is deployed to production
- Administrator – "Allows me to treat hardware instances less like 'pets' and more like 'cattle'"
  - Hardware resources come-and-go with minimal configuration requirements – just need the ability to run a container

Deployment

Docker

- The Docker platform uses the Docker engine to build and package apps as Docker images
- Docker images are created using files written in the Dockerfile format

Deployment

Docker

- Microsoft provides a collection of official images to act as the starting point for your own images
  - Have the .NET Core runtime pre-installed
  - Some have the .NET Core SDK installed and can be used as a build server
- Available on Docker Hub
  - hub.docker.com/_/microsoft-dotnet-sdk
  - hub.docker.com/_/microsoft-dotnet-aspnet

Deployment

Docker

- A container is a running instance of an image
- When running an ASP.NET Core application in a container, it is necessary to map the internal container port to a port on the host machine

```
docker run –d -p 80:80 ecomm/api
```

- The Microsoft images are preconfigured (via an environment variable) to run Kestrel on port 80
- Documentation has extensive information about configuring Kestrel (HTTPS, HTTP/2, host filtering, etc.)

Deployment

Docker

- Visual Studio supports building, running, and debugging containerized ASP.NET Core applications
  - Must have Docker for Windows installed
- You can enable Docker support when creating a project
- You can also add Docker support to an existing app by selecting [Add > Docker Support]

Deployment

Docker

- Docker containers can run on any machine with the Docker engine installed
- Both Amazon AWS and Microsoft Azure have extensive support for hosting containers
  - AWS EC2 Container Service and Container Registry
  - Azure Container Service and Container Registry