Microservice Architecture with ASP.NET Core

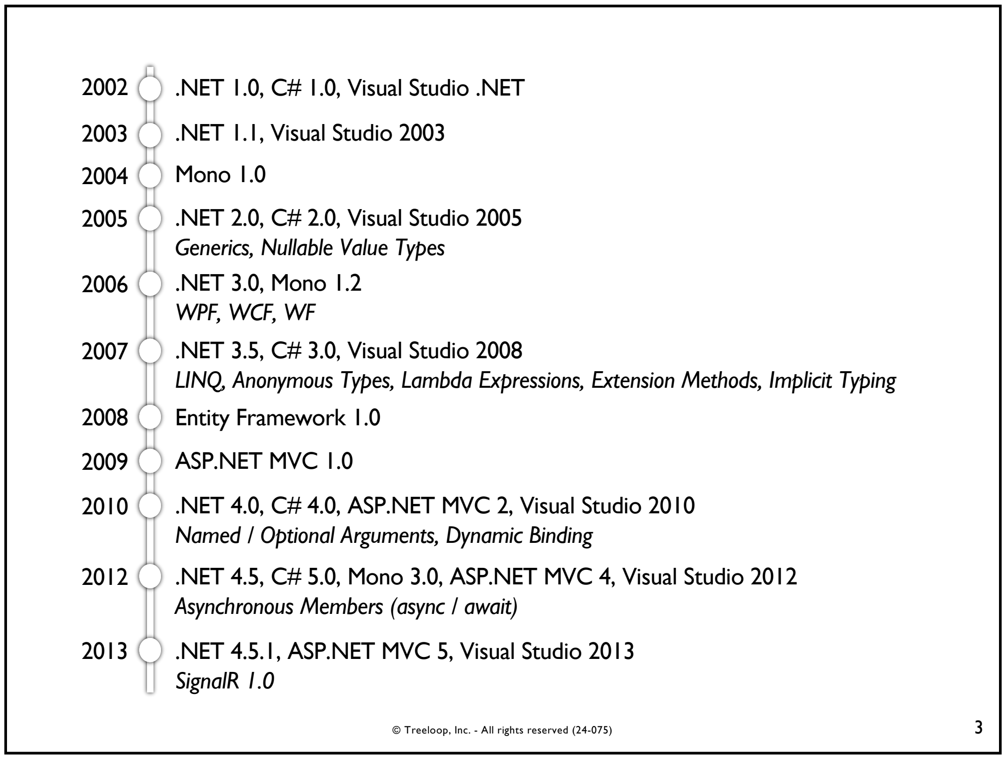
Agenda (Part I)

- Introduction to .NET Core
- .NET 8.0 SDK
- Modern C# and What's New
- ASP.NET Core Application Architecture
- Models
- Controllers
- Request Routing
- Web APIs
- Data Validation
- Error Handling and Logging

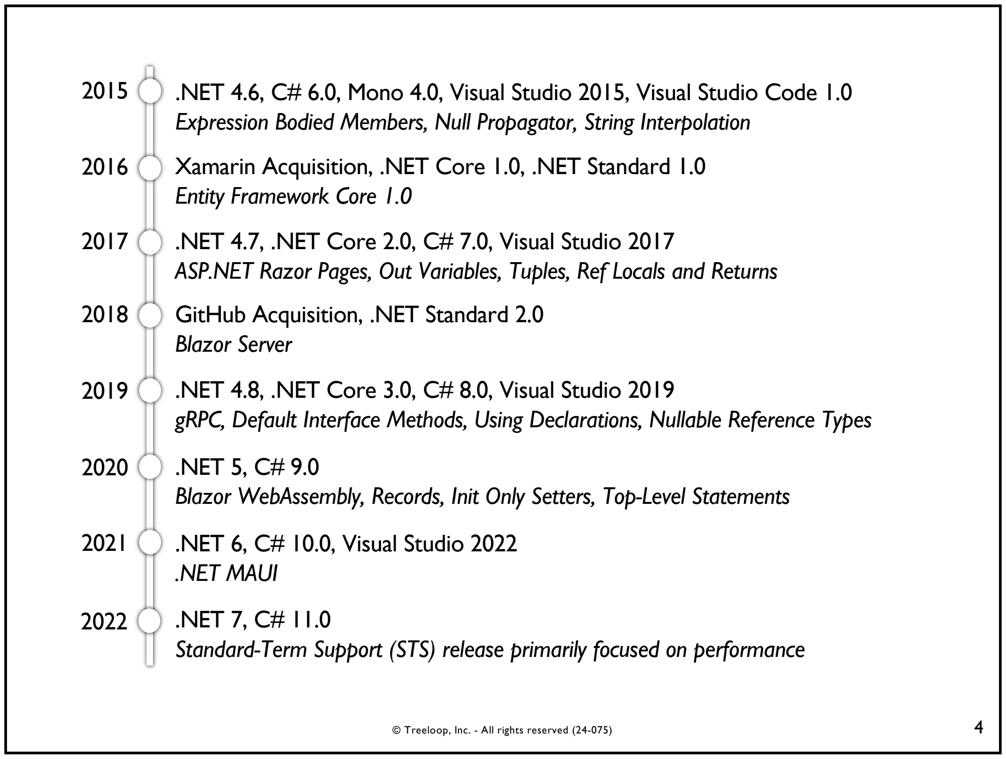
Microservice Architecture with ASP.NET Core

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*
2022	.NET 7, C# 11.0 *Standard-Term Support (STS) release primarily focused on performance*

2023 .NET 8, C# 12

Collection Expressions, ref readonly Parameters, Blazor Render Modes

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
- 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
  - Standard Term Support (STS) – Includes the latest features with a support period of 18 months
  - Long Term Support (LTS) – Has an extended support period of three years
  - 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`

Microservice Architecture with ASP.NET Core

.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": "6.0.14"
  }
}
```

  - 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>net8.0</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 8.0 is specified, the application will automatically use the 8.0.3 runtime but will not automatically use version 8.1 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 8 of the SDK to build an application that targets the .NET 6 runtime

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

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

.NET SDK

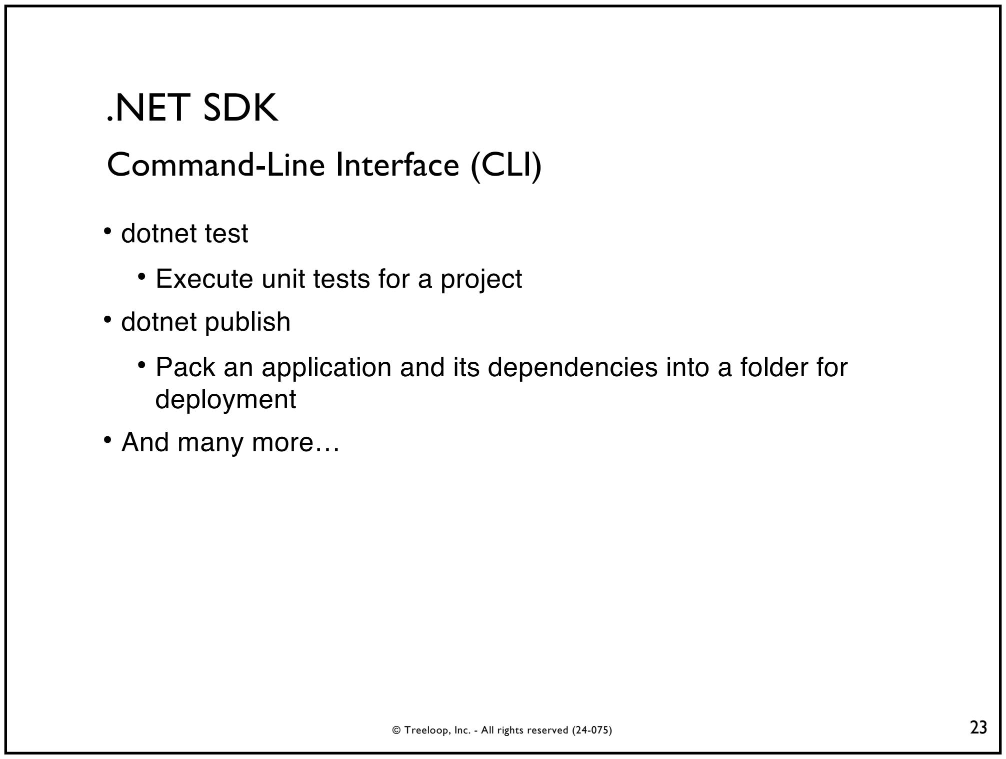
Command-Line Interface (CLI)

- 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)

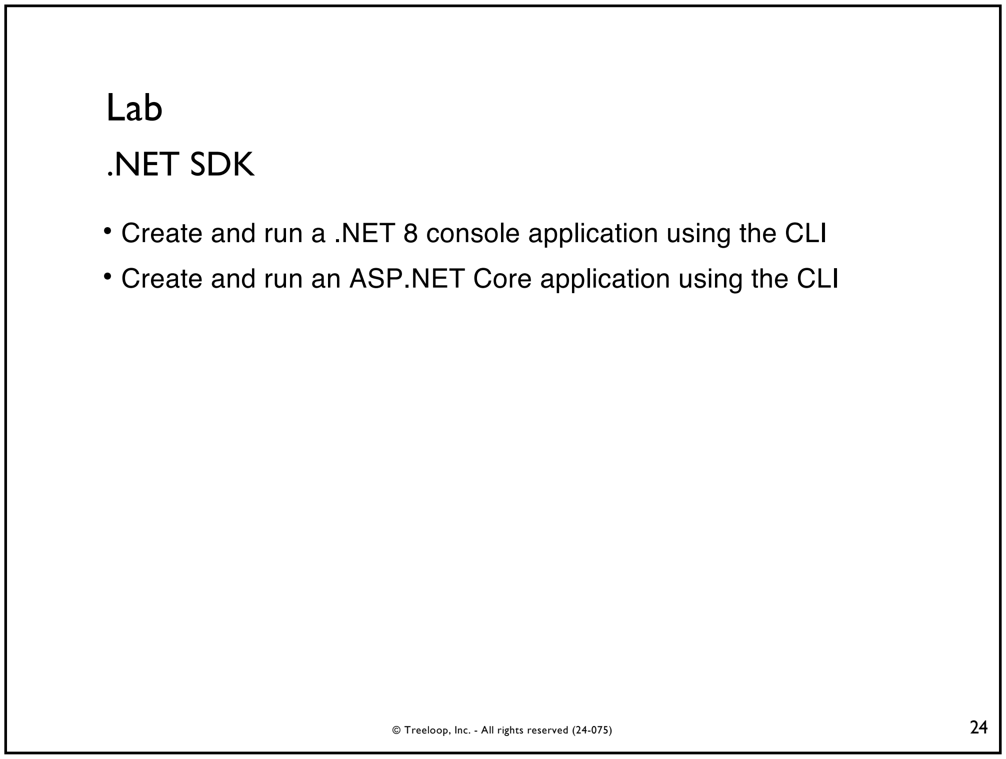
- dotnet new
  - Create a new project from an available template
- dotnet restore
  - Restore the dependencies for a project (download missing NuGet packages)
- dotnet build
  - Build a project and all its dependencies
- dotnet run
  - Run an application from its source code (performs a build if necessary)



.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

.NET SDK

- Create and run a .NET 8 console application using the CLI
- Create and run an ASP.NET Core application using the CLI

Microservice Architecture with ASP.NET Core

Modern C# and What's New

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

Modern C# and What's New

Introduction

- C# 9 introduced with .NET 5
- C# 10 introduced with .NET 6
- C# 11 introduced with .NET 7
- C# 12 introduced with .NET 8

Modern C# and What's New

Global Using Directives

- C# 10 introduced 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

Modern C# and What's New

Global Using Directives

- Starting with .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

Modern C# and What's New

File-Scoped Namespace Declarations

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

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

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

```
namespace Acme.Models;

...
```

Modern C# and What's New

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# compiler (10 and later) 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!“);
```

Modern C# and What's New

Top-Level Statements

- ASP.NET 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();
```

Modern C# and What's New

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
```

Modern C# and What's New

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

Modern C# and What's New

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>
```

- Starting with .NET 6, nullable reference types are enabled by default

Modern C# and What's New

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;
```

Modern C# and What's New

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

Modern C# and What's New

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

Modern C# and What's New

Init Only Setters

- 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; }
}
```

Modern C# and What's New

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

Modern C# and What's New

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);
}
```

Modern C# and What's New

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" };
```

Modern C# and What's New

Record Types

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

Modern C# and What's New

Deferred Execution

- When using LINQ methods in C#, it is important to be aware of deferred execution

Microservice Architecture with ASP.NET Core

ASP.NET Core Application Architecture

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

ASP.NET Core 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

ASP.NET Core Application Architecture

NuGet Packages

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

```
<PackageReference Include="Microsoft.EntityFrameworkCore" Version="8.0.3" />
```

ASP.NET Core 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

ASP.NET Core 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

ASP.NET Core 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

ASP.NET Core 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

ASP.NET Core 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

ASP.NET Core 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…

ASP.NET Core 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

ASP.NET Core Application Architecture

Application Startup

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

ASP.NET Core 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

ASP.NET Core Application Architecture

Middleware

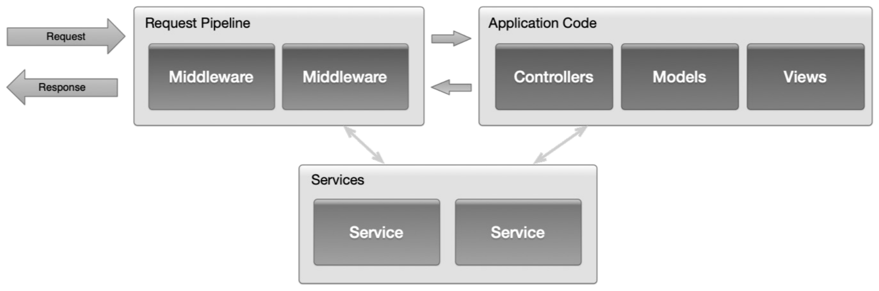
- ASP.NET uses a modular request processing pipeline
- The pipeline is composed of middleware components
- Each middleware component is responsible for invoking the next component in the pipeline or short-circuiting the chain
- Examples of middleware include…
  - Request routing
  - Handling of static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

ASP.NET Core 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

ASP.NET Core Application Architecture



ASP.NET Core 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 can be used

Microservice Architecture with ASP.NET Core

Models

- Introduction
- Persistence Ignorance
- Object-Relational Mapping
- Asynchronous Data Access

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

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

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();
}
```

Microservice Architecture with ASP.NET Core

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

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;
}
```

Microservice Architecture with ASP.NET Core

Controllers

- Responsibilities
- Requirements and Conventions
- Dependencies
- Action Results
- Model Binding

Controllers

Responsibilities

- The action executed for a particular endpoint is typically a method of a controller
- A controller may need to retrieve or make modifications to model data
- The controller also 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 for an API)
    - 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

- ActionResult<T> enables returning a type derived from ActionResult or a specific type
- Framework uses the ExecuteResultAsync when creating the HTTP response

```
public ActionResult<Product> GetProduct(int id)
{
  var product = _repository.GetProduct(id);
  if (product == null) return NotFound();
  return product;
}
```

- 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

- The base class Controller provides helper methods to generate various types of results
  - View `return View(customer);`
  - Serialized object `return Json(customer);`
  - HTTP status code `return NotFound();`
  - Raw content `return Content("Hello");`
  - Contents of a file `return File(bytes);`
  - Several forms of redirection
    - Redirect, RedirectToRoute, RedirectToAction, …
  - And more...

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<IEnumerable<Product>> Index(CancellationToken token)
{
  var products = await _repository.GetAllProducts(token);
  return products;
}
```

Controllers

Model Binding

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

Controllers

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) { ... }
  ...
}
```

Controllers

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

Microservice Architecture with ASP.NET Core

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 IActionResult AllProducts() { ... }
```

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

```
[HttpGet("products")]
public IActionResult AllProducts() { ... }
```

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 : Controller
{
  [HttpGet("{id}")]
  public IActionResult GetProduct(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

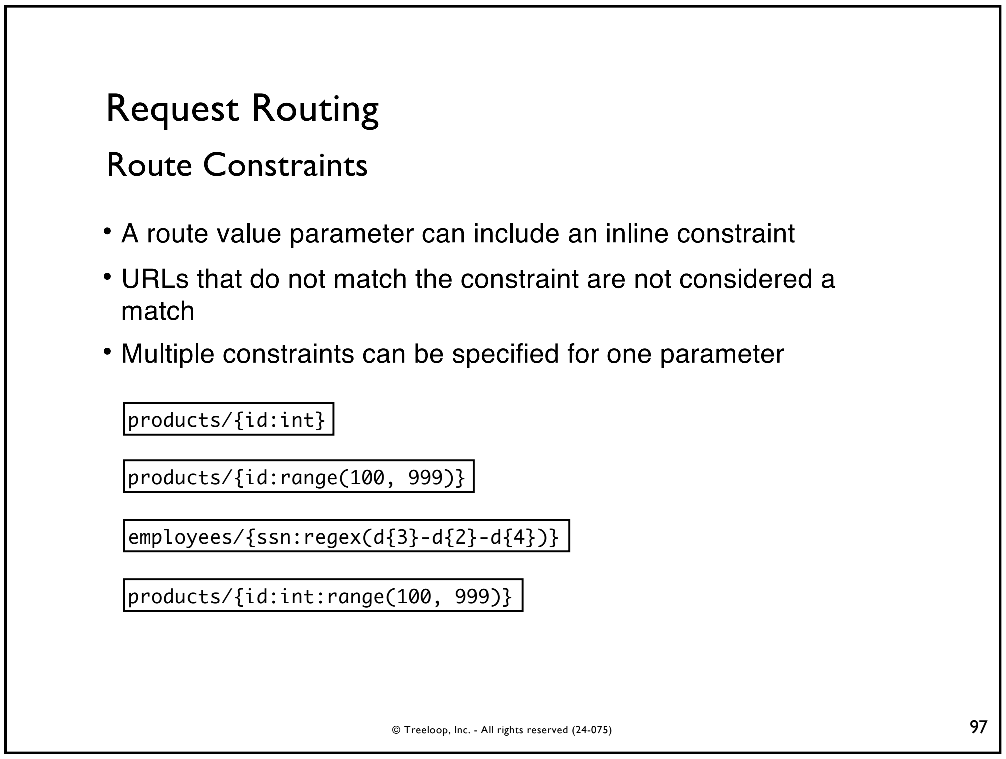
Route Templates

- 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

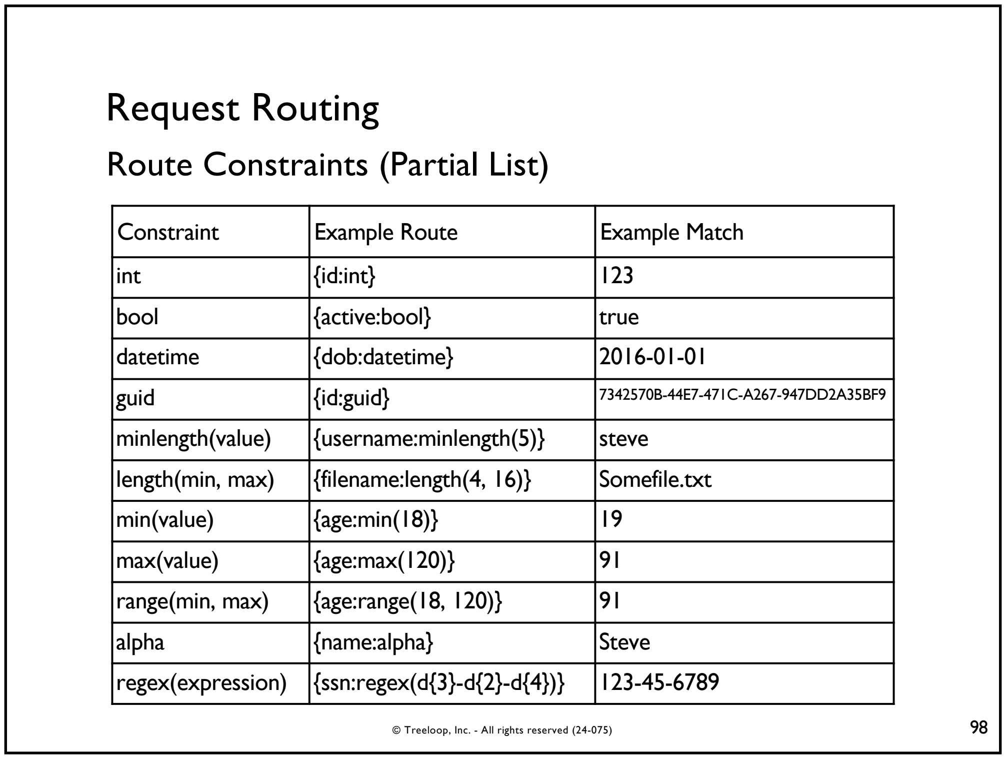
- A route value parameter can include an inline constraint
- URLs that do not match the constraint are not considered a match
- Multiple constraints can be specified for one parameter

```
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

Microservice Architecture with ASP.NET Core

Web APIs

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

101

101

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

102

102

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

- 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 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

Web APIs

Retrieval Operations

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

```
[JsonPropertyName("price")]
public double UnitPrice { get; set; }
```

```
public async Task<ActionResult> 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);
}
```

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

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<ActionResult> 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

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

```
builder.Services.AddCors(options => {
    options.AddPolicy("MyCorsPolicy",
        policy => {
            policy.WithOrigins("https://myapp.com");
        });
});
```

```
app.UseCors("MyCorsPolicy");
```

Microservice Architecture with ASP.NET Core

Data Validation

- Introduction
- Data Annotations
- Model Binding
- IValidatableObject

Data 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 hidden form fields, 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

Data Validation

Data Annotations

- A variety of data annotations can be added to the model (or view model) that is sent to a view
- Data annotations are used during model binding to perform server-side validation

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

Data 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

Data Validation

Model Binding

- If a value is considered to be invalid, an error is added to ModelState and ModelState.IsValid will return false
- ModelState is also used by the helpers in the view when returning a view after a server-side validation error

Data 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;
}
```

Microservice Architecture with ASP.NET Core

Error Handling

- Best Practices
- HTTP Error Status Codes

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)
  - Give the user some options (e.g., link to visit the home page in the case of a 404)
  - Use static content whenever possible to avoid an error page 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 in these cases

Microservice Architecture with ASP.NET Core

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

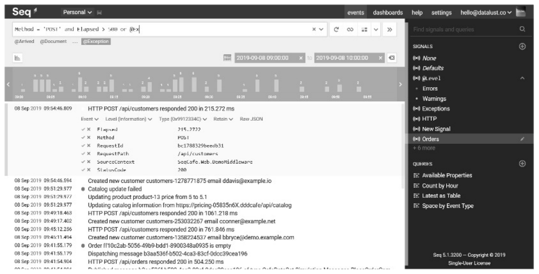
- Serilog has become a popular choice for ASP.NET Core
  - Wide variety of destinations and formats
  - Can record structured event data

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



```
Method = 'POST' and Elapsed > 500 or @Ex
2019-09-08 09:00:00 to 2019-09-08 10:00:00

08 Sep 2019 09:54:46.809   HTTP POST /api/customers responded 200 in 215.272 ms
    Event   Level (Information)   Type (0x9912334C)   Retain   Raw JSON
    Elapsed          215.2722
    Method           POST
    RequestId        bc1788329beedb31
    RequestPath      /api/customers
    SourceContext    SeqCafe.Web.DemoMiddleware
    StatusCode       200
08 Sep 2019 09:54:46.594   Created new customer customers-1278771875 email ddavis@example.io
08 Sep 2019 09:51:29.977 ● Catalog update failed
08 Sep 2019 09:51:29.977   Updating product product-13 price from 5 to 5.1
08 Sep 2019 09:51:29.977   Updating catalog information from https://pricing-05835n6X.dddcafe/api/catalog
08 Sep 2019 09:49:18.463   HTTP POST /api/customers responded 200 in 1061.218 ms
08 Sep 2019 09:49:17.402   Created new customer customers-253032267 email cconner@example.net
08 Sep 2019 09:45:12.256   HTTP POST /api/customers responded 200 in 761.846 ms
08 Sep 2019 09:45:11.494   Created new customer customers-1358224537 email bbryce@demo.example.com
08 Sep 2019 09:41:55.179 ● Order ff10c2ab-5056-49b9-bdd1-8900348a0935 is empty
08 Sep 2019 09:41:55.179   Dispatching message b3aa536f-b502-4ca3-83cf-0dcc39cea196
08 Sep 2019 09:41:54.904   HTTP POST /api/orders responded 200 in 504.250 ms
```

Logging

Maintainability

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