## C#::Operator

* Nullable type operator (?) – append to any non-nullable type to make it nullable. Useful for HTTP requests that may or may not have data.
* [async] keyword – indicates that a method is non-blocking and the thread can continue to do other work while the method is completing.
* [await] keyword – pause execution until an asynchronous method is complete (otherwise the method could complete unsuccessfully while the asynchronous method is running).
* [var] keyword – use type inference at compile time. Typically used when the right hand side returns an obvious type and its use improves conciseness, or when the type may change in the future.
* Object Initialization format – used to initialize an object and its properties at the same time. Can improve clarity by making it clear which variables are being initialized.
  + **var obj =** **new className {Prop1 = Val1, Prop2=Val2,};**
  + as opposed to **var obj = new className()**; **obj.Prop1(Val1);**
* Using directive – in the beginning of a file, used to include namespaces which allow you to use types without specifying the full namespace path.
* Using statement - used to ensure that an object that implements the IDisposable interface is properly disposed of once it is no longer needed. Used in one line with a variable declaration so the object will be disposed after the current block or in its own code block.
* [attributes] – applied to classes, methods, properties, parameters to provide additional information or modify behavior.
  + function([Bind(Property1, Property2)] ParamType ParamName) – specify that only listed properties should be included in model binding from HTTP request to prevent over-posting attacks (where attackers set properties on a model they shouldn’t have access to).

## C#::JSON – used to transfer data to/from HTTP requests.

* IActionResult – interface that defines a method ExecuteResultAsync() which represents the result of an action method.
  + ViewResult – Represents an HTML view to be rendered to the response.
  + JsonResult(String) – represents JSON data (a Javascript standard for serializing data) to be written to the response.
  + ContentResult{String content = str1, ContentType = stringFormat,} – represents a string to be written to the response.
  + RedirectResult – represents a redirection to a new URL.

## C#::Encryption – BCRYPT

* Hash – a process to associate strings with sufficiently unique values in a way that can’t be reversed, allowing you to verify a password without storing a copy.
* Salt – randomized data added to a hash to prevent using precomputed values to reverse it.

## C#::Expression Trees – Lambdas

* Entity Framework uses Expression trees to translate LINQ into SQL. Expression trees are used for dynamic queries based on user input or other runtime conditions.
* Expression class – base class for expression trees.
* Lambda expression – can be converted into expression trees.
* Expression types - ConstantExpression, ParameterExpression, MethodCallExpression, etc.

## ASP.NET CORE::Route Templates::Route Template – map incoming HTTP requests to appropriate action methods in your controllers. (System.ComponentModel.DataAnnotations)

* [Route] – used to define the base route for a controller or a specific route for an action method.
* [*Placeholders*] – replaced with the actual values when the route is executed.
  + [controller] – name of the controller class minus the “controller” suffix.
  + [action] – name of the action method.
  + [area] – name of the area. Areas allow you to partition a large application into smaller functional groupings.
  + {variable name} – name of a parameter in your action method that the value is passed to.
* [HttpGet] – function that responds to a request for data, typically with a View, serialized data in JSON or XML, or an IActionResult status code.
* [HttpPost] – function that responds to a request for the server to accept the enclosed data.
* [ApiController] – provides automatic API, routing, and validation behavior. Can be omitted if manual control is required.
* [ValidateAntiForgeryToken] – prevents automated actions that may be malicious be generating a unique anti-forgery token to each user session. Not needed when using the [ApiController] attribute as the anti-forgery validation is automatically performed for all non-safe HTTP methods (like POST, PUT, DELETE, PATCH that modify the state of the server rather than pure information retrieval).
* [controller] – indicates a controller class, automatically removes the “controller” suffix in the route. Add “api/” base folder to route to avoid conflicts according to convention.
* ModelState.IsValid – ensure JS model is valid before safely processing input.

## C#::Entity Framework Core – ORM based on DbContext

* Object-Relational Mapping – maps features of a relational table to functions in an object.
* IConfiguration – saves the connection strings from the appsettings.json file.
* OnConfiguring – takes an optionsBuilder object and passes the connection string to it with UseMySql(IConfiguration.getConnectionString, new MySqlVersion(new Version)))

## C#::MySQL class

* MySqlConnection – Establish a connection to your MySql database.
* MySqlCommand – Execute an SQL query.
* MySqlDataReader – Store the result of the query, which can be displayed in an HTML table with a server-side script.

## C#::Dotnet

* dotnet restore – download any missing packages defined in the project file (.csproj)
* dotnet publish – creates a folder with everything needed to run your application, including the runtime and any package dependencies. (dotnet publish -c Release -r win10-x64 --self-contained true)
* dotnet ef migrations scriptName – generate a script using the script-migrations command to generate tables for an empty database.
* dotnet ef database update – apply any pending migrations. Will fail if tables already exist.
* Service - a class managed by a DI container that performs operations in a .net application. A service must follow the principles of Inversion of Control (IoC) where it does not create instances of its dependencies.
  + Constructor injection – dependencies are passed through as constructor parameters. (Most common)
  + Interface Implementation – dependences are passed through an interface.
  + Lifecycle management – the DI container can create or dispose of the service as necessary.
* Dependency injection (DI) service lifetimes
  + Singleton – only a single instance is created for all requests.
  + Scoped – a new instance is created once per request or per scope.
  + Transient – a new instance is created per request. This can lead to multiple instances being created within the same scope.
* Retrieve services from Dependency Injection (DI) Container
  + using var scope = host.Services.CreateScope() – create a new scope to ensure services are disposed of properly when no longer in use.
  + var services = scope.ServiceProvider() – obtain the service provider, used to retrieve services from the DI container.
  + var controller = services.GetRequiredService<ServiceName>() – obtain a service of the specified type.

## ASP.NET CORE::Route Templates::Placeholders – replaced with the value of the placeholder in the route.

* [controller] – the name of the controller class.
* [action] – the name of the action method.
* [area] – the name of the area, used to partition larger applications into smaller, more manageable sections.
* {parameter} – the name of the associated parameter, used to pass values from the URL to a method.

## Javascript::

* Dynamically typed – a variable can hold any type of value and its type can change over time by reassigning it.
* Prototype-based inheritance – objects can inherit properties directly from other objects.
* Single-threaded – concurrency model is based on an event loop meaning it can handle one operation at a time. Asynchronous operations are run outside the program using a Web API or Node.js (a separate runtime environment), and added to the task queue on completion. The event loop constantly checks if the call stack is empty; if it is, it takes the first task from the task queue and adds it to the stack.

Javascript::Keywords

* Arrow functions – concise function syntax.
  + const add = (a, b) => a + b; - creates a const variable *add* which is a reference to a function. Add braces and a return statement to make it multiline.
* Callback functions – callbacks – a function that is passed as an argument to another function, which is responsible for executing it. Often used in asynchronous operations to execute after the operation completes.
* *var* – declares a variable with function or class scope.
* *let* – declares a variable with block scope.

## Javascript::API

* Promise – a Promise runs a callback function and returns a new Promise. The value of the callback is passed to the parameters of the second Promise.
* Allowed states
  + - Pending – not completed.
    - Fulfilled – Promise has resulting value.
    - Rejected – Promise has fail reason.
  + Methods
    - .then(onFulfilled, onRejected) – takes two optional callbacks and returns a Promise.
    - .catch(onRejected) – equivalent to .then(undefined, OnRejected) – attaches a callback to a Promise failure or rejection and returns a new Promise.
    - .finally(onFinally) – attaches a callback to a settled Promise and returns a new Promise.
* Array.forEach(function(*parameter*) {} ) – calls a function for each element in the list.
* **Fetch API: As you've seen, it provides an interface for fetching resources across the network.**
  + You can use it to manipulate the structure, style, and content of a document.
* Web Storage API: Provides mechanisms for storing data on the user's browser. It includes localStorage and sessionStorage.
* Geolocation API: Allows the user to provide their location to web applications if they so desire.
* Canvas API: Used to draw graphics on a web page via JavaScript.
* Web Audio API: Allows you to incorporate audio into your web apps. You can create, manipulate, and play sounds.
* File API: Provides means to read the contents of files (or raw data buffers).
* History API: Allows manipulation of the browser session history, that is the pages visited in the tab or frame that the current page is loaded in.
* Web Workers API: Runs scripts in the background independently of any user interface scripts. This allows for long tasks to be executed without freezing the user interface.
* Service Workers API: Acts as a proxy between web applications, the browser, and the network (when available). They are designed to be fully asynchronous, and can handle push notifications and background syncs.

## HTML::Elements

* Div – creates block level divisions (paragraphs).
* Span – creates inline divisions.

## HTML::Properties

* Display
  + Block – start on a new line and take the full width available.
    - <div> <p> <h1> <h6>
  + Inline – do not start on a new line and only take up as much space as necessary.
    - <span> <a> <img>
  + Inline-block – similar to inline elements but can have a width and height.
  + Flex – flexible container for one-dimensional layouts.
  + Grid – flexible container for two-dimensional layouts. https://www.w3schools.com/CSS/css\_grid.asp
  + None – does not take up space and are not visible. Different from visibility:hidden which hides an element but still takes up space.