* <https://radixweb.com/blog/web-api-with-asp-dot-net-core-process-and-practices>
  + [Basic Structure of C# Program](https://dotnettutorials.net/lesson/basic-structure-of-csharp-program/)
  + [Methods and Properties of Console Class in C#](https://dotnettutorials.net/lesson/console-class-methods-properties-csharp/)
  + [Data Types in C#](https://dotnettutorials.net/lesson/data-types-in-csharp/)
  + [Literals in C#](https://dotnettutorials.net/lesson/literals-in-csharp/)
  + [Type Casting in C#](https://dotnettutorials.net/lesson/type-casting-in-csharp/)
  + [Variables in C#](https://dotnettutorials.net/lesson/variables-in-csharp/)
  + [Operators in C#](https://dotnettutorials.net/lesson/operators-in-csharp/)
  + [Control Flow Statements in C#](https://dotnettutorials.net/lesson/control-flow-statements-in-csharp/)
  + [If-Else Statements in C#](https://dotnettutorials.net/lesson/if-else-statements-in-csharp/)
  + [Switch Statements in C#](https://dotnettutorials.net/lesson/switch-statements-in-csharp/)
  + [Loops in C#](https://dotnettutorials.net/lesson/loops-in-csharp/)
  + [While Loop in C#](https://dotnettutorials.net/lesson/while-loop-in-csharp/)
  + [Do While Loop in C#](https://dotnettutorials.net/lesson/do-while-loop-in-csharp/)
  + [For Loop in C#](https://dotnettutorials.net/lesson/for-loop-in-csharp/)
  + [Break Statement in C#](https://dotnettutorials.net/lesson/break-statement-in-csharp/)
  + [Continue Statement in C#](https://dotnettutorials.net/lesson/continue-statement-in-csharp/)
  + [Goto Statement in C#](https://dotnettutorials.net/lesson/goto-statement-in-csharp/)
  + [Functions in C#](https://dotnettutorials.net/lesson/functions-in-csharp/)
  + [User-Defined Functions in C#](https://dotnettutorials.net/lesson/user-defined-functions-in-csharp/)
  + [Call By Value and Call By Reference in C#](https://dotnettutorials.net/lesson/call-by-value-and-call-by-reference-in-csharp/)
  + [Recursion in C#](https://dotnettutorials.net/lesson/recursion-in-csharp/)
  + [User Input and Output in C#](https://dotnettutorials.net/lesson/input-and-output-in-csharp/)
  + [Command Line Arguments in C#](https://dotnettutorials.net/lesson/command-line-arguments-in-csharp/)
  + [String in C#](https://dotnettutorials.net/lesson/string-in-csharp/)
  + [Static Keyword in C#](https://dotnettutorials.net/lesson/static-in-csharp/)
  + [Static vs Non-Static Members in C#](https://dotnettutorials.net/lesson/static-and-non-static-members-csharp/)
  + [Const and Read-Only in C#](https://dotnettutorials.net/lesson/const-and-read-only-csharp/)
  + [Properties in C#](https://dotnettutorials.net/lesson/properties-csharp/)
  + [Why we Should Override ToString Method in C#](https://dotnettutorials.net/lesson/why-we-should-override-the-tostring-method/)
  + [Override Equals Method in C#](https://dotnettutorials.net/lesson/why-we-should-override-equals-method/)
  + [Difference Between Convert.ToString and ToString Method in c#](https://dotnettutorials.net/lesson/difference-between-convert-tostring-and-tostring-csharp/)
  + [Checked and Unchecked Keywords in C#](https://dotnettutorials.net/lesson/checked-and-unchecked-keyword/)
  + [Stack and Heap Memory in .NET](https://dotnettutorials.net/lesson/stack-and-heap-dotnet/)
  + [Boxing and Unboxing in C#](https://dotnettutorials.net/lesson/boxing-and-unboxing-in-csharp/)
* **OOPs in C#**
  + [Object Oriented Programming (OOPs) in C#](https://dotnettutorials.net/lesson/object-oriented-programming-csharp/)
  + [Class and Objects in C#](https://dotnettutorials.net/lesson/class-and-objects-csharp/)
  + [Constructors in C#](https://dotnettutorials.net/lesson/constructors-csharp/)
  + [Types of Constructors in C#](https://dotnettutorials.net/lesson/types-of-constructors-in-csharp/)
  + [Why We Need Constructors in C#](https://dotnettutorials.net/lesson/why-do-we-need-constructors-in-csharp/)
  + [Static vs Non-Static Constructors in C#](https://dotnettutorials.net/lesson/static-vs-non-static-constructors-in-csharp/)
  + [Private Constructors in C#](https://dotnettutorials.net/lesson/private-constructors-in-csharp/)
  + [Destructors in C#](https://dotnettutorials.net/lesson/destructor-csharp/)
  + [Garbage Collection in .NET Framework](https://dotnettutorials.net/lesson/garbage-collector/)
  + [Differences Between Finalize and Dispose in C#](https://dotnettutorials.net/lesson/differences-between-finalize-and-dispose-in-csharp/)
  + [Access Specifiers in C#](https://dotnettutorials.net/lesson/access-specifiers-csharp/)
  + [Encapsulation in C#](https://dotnettutorials.net/lesson/encapsulation-csharp/)
  + [Abstraction in C#](https://dotnettutorials.net/lesson/abstraction-csharp-realtime-example/)
  + [Inheritance in C#](https://dotnettutorials.net/lesson/inheritance-c-sharp/)
  + [Types of Inheritance in C#](https://dotnettutorials.net/lesson/types-of-inheritance-in-csharp/)
  + [How to use Inheritance in Application Development](https://dotnettutorials.net/lesson/how-to-use-inheritance-in-application-development/)
  + [IsA and HasA Relationship in C#](https://dotnettutorials.net/lesson/isa-and-hasa-relationship-in-csharp/)
  + [Generalization and Specialization in C#](https://dotnettutorials.net/lesson/generalization-and-specialization-in-csharp/)
  + [Abstract Class and Abstract Methods in C#](https://dotnettutorials.net/lesson/abstract-class-abstract-methods-csharp/)
  + [Abstract Class and Abstract Methods Interview Questions in C#](https://dotnettutorials.net/lesson/abstract-class-and-abstract-methods-interview-questions-in-csharp/)
  + [How to Use Abstract Classes and Methods in C# Application](https://dotnettutorials.net/lesson/how-to-use-abstract-classes-and-methods-in-csharp/)
  + [Interface in C#](https://dotnettutorials.net/lesson/interface-c-sharp/)
  + [Interface Interview Questions and Answers in C#](https://dotnettutorials.net/lesson/interface-interview-questions-and-answers-in-csharp/)
  + [Interface Realtime Examples in C#](https://dotnettutorials.net/lesson/interface-realtime-examples-in-csharp/)
  + [Multiple Inheritance in C#](https://dotnettutorials.net/lesson/multiple-inheritance-csharp/)
  + [Multiple Inheritance Realtime Example in C#](https://dotnettutorials.net/lesson/multiple-inheritance-realtime-example-in-csharp/)
  + [Polymorphism in C#](https://dotnettutorials.net/lesson/polymorphism-csharp/)
  + [Method Overloading in C#](https://dotnettutorials.net/lesson/function-overloading-csharp/)
  + [Operator Overloading in C#](https://dotnettutorials.net/lesson/operator-overloading-in-csharp/)
  + [Method Overriding in C#](https://dotnettutorials.net/lesson/function-overriding-csharp/)
  + [Method Hiding in C#](https://dotnettutorials.net/lesson/function-hiding-csharp/)
  + [Partial Class and Partial Methods in C#](https://dotnettutorials.net/lesson/partial-classes-partial-methods-csharp/)
  + [Sealed Class and Sealed Methods in C#](https://dotnettutorials.net/lesson/sealed-class-methods-csharp/)
  + [Extension Methods in C#](https://dotnettutorials.net/lesson/extension-methods-csharp/)
  + [Static Class in C#](https://dotnettutorials.net/lesson/static-class-in-csharp/)
  + [Variable Reference and Instance of a Class in C#](https://dotnettutorials.net/lesson/variable-reference-and-instance-of-a-class-in-csharp/)
* **OOPs Real-Time Examples**
  + [Real-time Examples of Encapsulation Principle in C#](https://dotnettutorials.net/lesson/real-time-examples-of-encapsulation-principle-in-csharp/)
  + [Real-Time Examples of Abstraction Principle in C#](https://dotnettutorials.net/lesson/real-time-examples-of-abstraction-principle-in-csharp/)
  + [Real-Time Examples of Inheritance Principle in C#](https://dotnettutorials.net/lesson/real-time-examples-of-inheritance-principle-in-csharp/)
  + [Real-Time Examples of Polymorphism Principle in C#](https://dotnettutorials.net/lesson/real-time-examples-of-polymorphism-principle-in-csharp/)
  + [Real-Time Examples of Interface in C#](https://dotnettutorials.net/lesson/real-time-examples-of-interface-in-csharp/)
  + [Real-Time Examples of Abstract Class in C#](https://dotnettutorials.net/lesson/real-time-examples-of-abstract-class-in-csharp/)
* **Exception Handling**
  + [Exception Handling in C#](https://dotnettutorials.net/lesson/exception-handling-csharp/)
  + [Multiple Catch Blocks in C#](https://dotnettutorials.net/lesson/multiple-catch-blocks-csharp/)
  + [Finally Block in C#](https://dotnettutorials.net/lesson/finally-block-in-csharp/)
  + [How to Create Custom Exceptions in C#](https://dotnettutorials.net/lesson/create-custom-exception-csharp/)
  + [Inner Exception in C#](https://dotnettutorials.net/lesson/inner-exception-csharp/)
  + [Exception Handling Abuse in C#](https://dotnettutorials.net/lesson/exception-handling-abuse-csharp/)
* **Events, Delegates and Lambda Expression in C#**
  + [Course Structure of Events, Delegates and Lambda Expression](https://dotnettutorials.net/lesson/course-structure-of-events-delegates-and-lambda-expression-csharp/)
  + [Roles of Events, Delegates and Event Handler in C#](https://dotnettutorials.net/lesson/events-delegates-and-event-handler-in-csharp/)
  + [Delegates in C#](https://dotnettutorials.net/lesson/delegates-csharp/)
  + [Multicast Delegates in C#](https://dotnettutorials.net/lesson/multicast-delegate-csharp/)
  + [Delegates Real-Time Example in C#](https://dotnettutorials.net/lesson/delegates-real-time-example-csharp/)
  + [Generic Delegates in C#](https://dotnettutorials.net/lesson/generic-delegates-csharp/)
  + [Anonymous Method in C#](https://dotnettutorials.net/lesson/anonymous-method-c-sharp/)
  + [Lambda Expressions in C#](https://dotnettutorials.net/lesson/lambda-expression-csharp/)
  + [Events in C# with Examples](https://dotnettutorials.net/lesson/events-in-csharp/)
* **Multi-Threading**
  + [Multithreading in C#](https://dotnettutorials.net/lesson/multithreading-in-csharp/)
  + [Thread class in C#](https://dotnettutorials.net/lesson/constructors-of-thread-class-csharp/)
  + [How to Pass Data to Thread Function in Type Safe Manner in C#](https://dotnettutorials.net/lesson/how-to-pass-data-to-the-thread-function-in-a-type-safe-manner/)
  + [How to Retrieve Data from a Thread Function in C#](https://dotnettutorials.net/lesson/how-to-retrieve-data-from-a-thread-function/)
  + [Join Method and IsAlive Property of Thread Class in C#](https://dotnettutorials.net/lesson/join-method-of-thread-class/)
  + [Thread Synchronization in C#](https://dotnettutorials.net/lesson/thread-synchronization-in-csharp/)
  + [Lock in C#](https://dotnettutorials.net/lesson/locking-in-multithreading/)
  + [Monitor Class in C#](https://dotnettutorials.net/lesson/multithreading-using-monitor/)
  + [Mutex Class in C#](https://dotnettutorials.net/lesson/mutex-in-multithreading/)
  + [Semaphore Class in C#](https://dotnettutorials.net/lesson/semaphore-in-multithreading/)
  + [SemaphoreSlim Class in C#](https://dotnettutorials.net/lesson/semaphoreslim-class-in-csharp/)
  + [Deadlock in C#](https://dotnettutorials.net/lesson/deadlock-in-csharp/)
  + [Performance Testing of a Multithreaded Application](https://dotnettutorials.net/lesson/performance-testing-of-a-multithreaded-application/)
  + [Thread Pool in C#](https://dotnettutorials.net/lesson/thread-pooling/)
  + [Foreground and Background Threads in C#](https://dotnettutorials.net/lesson/foreground-and-background-threads-in-csharp/)
  + [AutoResetEvent and ManualResetEvent in C#](https://dotnettutorials.net/lesson/autoresetevent-and-manualresetevent-in-csharp/)
  + [Thread Life Cycle in C#](https://dotnettutorials.net/lesson/thread-life-cycle-in-csharp/)
  + [Threads Priorities in C#](https://dotnettutorials.net/lesson/threads-priorities-in-csharp/)
  + [How to Terminate a Thread in C#](https://dotnettutorials.net/lesson/how-to-terminate-a-thread-in-csharp/)
  + [Inter Thread Communication in C#](https://dotnettutorials.net/lesson/inter-thread-communication-in-csharp/)
  + [How to Debug a Multi-threaded Application in C#](https://dotnettutorials.net/lesson/how-to-debug-a-multi-threaded-application-in-csharp/)
* **Collections in C#**
  + [Arrays in C#](https://dotnettutorials.net/lesson/arrays-csharp/)
  + [2D Arrays in C#](https://dotnettutorials.net/lesson/two-dimensional-array-in-csharp/)
  + [Advantages and Disadvantages of Arrays in C#](https://dotnettutorials.net/lesson/advantages-and-disadvantages-of-arrays-csharp/)
  + [Collections in C#](https://dotnettutorials.net/lesson/collections-csharp/)
  + [ArrayList in C#](https://dotnettutorials.net/lesson/arraylist-collection-csharp/)
  + [Hashtable in C#](https://dotnettutorials.net/lesson/hashtable-csharp/)
  + [Non-Generic Stack in C#](https://dotnettutorials.net/lesson/stack-collection-csharp/)
  + [Non-Generic Queue in C#](https://dotnettutorials.net/lesson/queue-collection-class-csharp/)
  + [Non-Generic SortedList in C#](https://dotnettutorials.net/lesson/sortedlist-collection-class-in-csharp/)
  + [Advantages and Disadvantages of Non-Generic Collection in C#](https://dotnettutorials.net/lesson/advantages-and-disadvantages-of-collection/)
  + [Generic Collections in C#](https://dotnettutorials.net/lesson/generic-collections-csharp/)
  + [Generics in C#](https://dotnettutorials.net/lesson/generics-csharp/)
  + [Generic Constraints in C#](https://dotnettutorials.net/lesson/generic-constraints-in-csharp/)
  + [Generic List Collection in C#](https://dotnettutorials.net/lesson/list-collection-csharp/)
  + [How to Sort a List of Complex Type in C#](https://dotnettutorials.net/lesson/sorting-a-list-of-complex-type-csharp/)
  + [Comparison Delegate in C#](https://dotnettutorials.net/lesson/comparison-delegate-csharp/)
  + [Dictionary Collection Class in C#](https://dotnettutorials.net/lesson/dictionary-generic-collection-csharp/)
  + [Conversion Between Array List and Dictionary in C#](https://dotnettutorials.net/lesson/conversion-between-array-list-and-dictionary-csharp/)
  + [List vs Dictionary in C#](https://dotnettutorials.net/lesson/dictionary-vs-list-csharp/)
  + [Generic Stack Collection Class in C#](https://dotnettutorials.net/lesson/generic-stack-csharp/)
  + [Generic Queue Collection Class in C#](https://dotnettutorials.net/lesson/generic-queue-collection-class-csharp/)
  + [Foreach Loop in C#](https://dotnettutorials.net/lesson/foreach-loop-in-csharp/)
  + [Generic HashSet Collection Class in C#](https://dotnettutorials.net/lesson/generic-hashset-collection-class-in-csharp/)
  + [Generic SortedList Collection Class in C#](https://dotnettutorials.net/lesson/generic-sortedlist-collection-class-in-csharp/)
  + [Generic SortedSet Collection Class in C#](https://dotnettutorials.net/lesson/generic-sortedset-collection-class-in-csharp/)
  + [Generic SortedDictionary Collection Class in C#](https://dotnettutorials.net/lesson/generic-sorteddictionary-collection-class-in-csharp/)
  + [Generic LinkedList Collection Class in C#](https://dotnettutorials.net/lesson/generic-linkedlist-collection-class-in-csharp/)
  + [Concurrent Collection in C#](https://dotnettutorials.net/lesson/concurrent-collection-in-csharp/)
  + [ConcurrentDictionary Collection Class in C#](https://dotnettutorials.net/lesson/concurrentdictionary-collection-class-in-csharp/)
  + [ConcurrentQueue Collection Class in C#](https://dotnettutorials.net/lesson/concurrentqueue-collection-class-in-csharp/)
  + [ConcurrentStack Collection Class in C#](https://dotnettutorials.net/lesson/concurrentstack-collection-class-csharp/)
  + [ConcurrentBag Collection Class in C#](https://dotnettutorials.net/lesson/concurrentbag-collection-class-in-csharp/)
  + [BlockingCollection in C#](https://dotnettutorials.net/lesson/blockingcollection-class-in-csharp/)
* **File Handling**
  + [File Handling in C#](https://dotnettutorials.net/lesson/file-handling-in-csharp/)
  + [FileStream Class in C#](https://dotnettutorials.net/lesson/filestream-class-in-csharp/)
  + [StreamReader and StreamWriter in C#](https://dotnettutorials.net/lesson/streamreader-and-streamwriter-in-csharp/)
  + [File Class in C#](https://dotnettutorials.net/lesson/file-class-in-csharp/)
  + [TextWriter and TextReader in C#](https://dotnettutorials.net/lesson/textwriter-and-textreader-in-csharp/)
  + [BinaryWriter and BinaryReader in C#](https://dotnettutorials.net/lesson/binarywriter-and-binaryreader-in-csharp/)
  + [StringWriter and StringReader in C#](https://dotnettutorials.net/lesson/stringwriter-and-stringreader-in-csharp/)
  + [FileInfo Class in C#](https://dotnettutorials.net/lesson/fileinfo-class-in-csharp/)
  + [DirectoryInfo Class in C#](https://dotnettutorials.net/lesson/directoryinfo-class-in-csharp/)
  + [Export and Import Excel Data in C#](https://dotnettutorials.net/lesson/export-and-import-excel-data-in-csharp/)
* **Asynchronous Programming**
  + [Introduction to Concurrency](https://dotnettutorials.net/lesson/introduction-to-parallel-and-asynchronous-programming-in-csharp/)
  + [Async and Await in C#](https://dotnettutorials.net/lesson/async-and-await-operator-in-csharp/)
  + [Task in C#](https://dotnettutorials.net/lesson/task-in-csharp/)
  + [How to Return a Value from Task in C#](https://dotnettutorials.net/lesson/return-a-value-from-a-task-in-csharp/)
  + [How to Execute Multiple Tasks in C#](https://dotnettutorials.net/lesson/how-to-execute-multiple-tasks-in-csharp/)
  + [How to Limit Number of Concurrent Tasks in C#](https://dotnettutorials.net/lesson/how-to-limit-number-of-concurrent-tasks-in-csharp/)
  + [How to Cancel a Task in C# using Cancellation Token](https://dotnettutorials.net/lesson/how-to-cancel-a-task-in-csharp/)
  + [How to Create Synchronous Method using Task in C#](https://dotnettutorials.net/lesson/how-to-create-synchronous-method-using-task-in-csharp/)
  + [Retry Pattern in C#](https://dotnettutorials.net/lesson/retry-pattern-in-csharp/)
  + [Only One Pattern in C#](https://dotnettutorials.net/lesson/only-one-pattern-in-csharp/)
  + [How to Control the Result of a Task in C#](https://dotnettutorials.net/lesson/how-to-control-the-result-of-a-task-in-csharp/)
  + [Task-Based Asynchronous Programming in C#](https://dotnettutorials.net/lesson/asynchronous-programming-in-csharp/)
  + [Chaining Tasks by Using Continuation Tasks](https://dotnettutorials.net/lesson/chaining-tasks-by-using-continuation-tasks/)
  + [How to Attached Child Tasks to a Parent Task in C#](https://dotnettutorials.net/lesson/how-to-attached-child-tasks-to-a-parent-task-in-csharp/)
  + [ValueTask in C#](https://dotnettutorials.net/lesson/valuetask-in-csharp/)
  + [How to Cancel a Non-Cancellable Task in C#](https://dotnettutorials.net/lesson/how-to-cancel-a-non-cancellable-task-in-csharp/)
  + [Asynchronous Streams in C#](https://dotnettutorials.net/lesson/asynchronous-streams-csharp/)
  + [How to Cancel Asynchronous Stream in C#](https://dotnettutorials.net/lesson/how-to-cancel-asynchronous-stream-in-csharp/)
* **Parallel Programming**
  + [Task Parallel Library in C#](https://dotnettutorials.net/lesson/task-parallel-library-overview/)
  + [Parallel For in C#](https://dotnettutorials.net/lesson/parallel-for-method-csharp/)
  + [Parallel Foreach Loop in C#](https://dotnettutorials.net/lesson/parallel-foreach-method-csharp/)
  + [Parallel Invoke in C#](https://dotnettutorials.net/lesson/parallel-invoke-method-csharp/)
  + [Maximum Degree of Parallelism in C#](https://dotnettutorials.net/lesson/maximum-degree-of-parallelism-in-csharp/)
  + [How to Cancel Parallel Operations in C#](https://dotnettutorials.net/lesson/cancel-parallel-operations-in-csharp/)
  + [Atomic Methods Thread Safety and Race Conditions in C#](https://dotnettutorials.net/lesson/atomic-methods-thread-safety-and-race-conditions-in-csharp/)
  + [Interlocked vs Lock in C#](https://dotnettutorials.net/lesson/interlocked-vs-lock-in-csharp/)
  + [Parallel LINQ in C#](https://dotnettutorials.net/lesson/parallel-linq-in-csharp/)
  + [Multithreading vs Asynchronous Programming vs Parallel Programming in C#](https://dotnettutorials.net/lesson/multithreading-vs-asynchronous-programming-vs-parallel-programming-in-csharp/)
* **AutoMapper**
  + [AutoMapper in C#](https://dotnettutorials.net/lesson/automapper-in-c-sharp/)
  + [AutoMapper Complex Mapping in C#](https://dotnettutorials.net/lesson/automapper-with-nested-types/)
  + [How to Map Complex Type to Primitive Type using AutoMapper in C#](https://dotnettutorials.net/lesson/mapping-complex-type-to-primitive-type-using-automapper/)
  + [AutoMapper Reverse Mapping in C#](https://dotnettutorials.net/lesson/reverse-mapping-using-automapper/)
  + [AutoMapper Conditional Mapping in C#](https://dotnettutorials.net/lesson/automapper-conditional-mapping-dotnet/)
  + [AutoMapper Ignore Method in C#](https://dotnettutorials.net/lesson/ignore-using-automapper-in-csharp/)
  + [Fixed and Dynamic Values in Destination Property in AutoMapper](https://dotnettutorials.net/lesson/usevalue-resolveusing-and-null-substitution-using-automapper/)
* **Optional Parameter, Indexers and Enums**
  + [How to make Optional Parameters in C#](https://dotnettutorials.net/lesson/how-to-make-method-parameters-optional-csharp/)
  + [Indexers in C#](https://dotnettutorials.net/lesson/indexers-csharp/)
  + [Indexers Real-Time Example in C#](https://dotnettutorials.net/lesson/indexers-real-time-example-csharp/)
  + [Enums in C#](https://dotnettutorials.net/lesson/enums-csharp/)
* **.NET Framework Architecture**
  + [DOT NET Framework](https://dotnettutorials.net/lesson/dotnet-framework/)
  + [Common Language Runtime in .NET Framework](https://dotnettutorials.net/lesson/common-language-runtime-dotnet/)
  + [.NET Program Execution Process](https://dotnettutorials.net/lesson/dotnet-program-execution-process/)
  + [Intermediate Language (ILDASM & ILASM) Code in C#](https://dotnettutorials.net/lesson/intermediate-language/)
  + [Common Type System in .NET Framework](https://dotnettutorials.net/lesson/common-type-system/)
  + [Common Language Specification in .NET Framework](https://dotnettutorials.net/lesson/common-language-specification/)
  + [Managed and Unmanaged Code in .NET Framework](https://dotnettutorials.net/lesson/managed-and-unmanaged-code/)
  + [Assembly DLL EXE in .NET Framework](https://dotnettutorials.net/lesson/assembly-dll-exe/)
  + [App Domain in .NET Framework](https://dotnettutorials.net/lesson/app-domain-dot-net-framework/)
  + [Strong and Weak Assemblies in .NET Framework](https://dotnettutorials.net/lesson/strong-weak-assemblies-in-net/)
  + [How to Install an Assembly into GAC in .NET Framework](https://dotnettutorials.net/lesson/how-to-install-an-assembly-into-gac-in-net-framework/)
  + [DLL Hell Problem and Solution in .NET Framework](https://dotnettutorials.net/lesson/dll-hell-problem-and-solution-in-net-framework/)
* **Var, Dynamic and Reflection**
  + [Reflection in C#](https://dotnettutorials.net/lesson/reflection-in-csharp/)
  + [Dynamic Type in C#](https://dotnettutorials.net/lesson/dynamic-type-in-csharp/)
  + [Var Keyword in C#](https://dotnettutorials.net/lesson/var-keyword-in-csharp/)
  + [Var vs Dynamic in C#](https://dotnettutorials.net/lesson/var-vs-dynamic-in-csharp/)
  + [Dynamic vs Reflection in C#](https://dotnettutorials.net/lesson/dynamic-vs-reflection-in-csharp/)
  + [Volatile Keyword in C#](https://dotnettutorials.net/lesson/volatile-keyword-in-csharp/)
  + [Ref vs Out in C#](https://dotnettutorials.net/lesson/ref-vs-out-in-csharp/)
  + [Named Parameters in C#](https://dotnettutorials.net/lesson/named-parameters-in-csharp/)
* **C# 7.X new Features**
  + [C# 7 New Features](https://dotnettutorials.net/lesson/csharp-7-new-features/)
  + [Enhancement in Out Variables in C# 7](https://dotnettutorials.net/lesson/out-variables-csharp7/)
  + [Pattern Matching in C#](https://dotnettutorials.net/lesson/pattern-matching-csharp/)
  + [Digit Separators in C# 7](https://dotnettutorials.net/lesson/digit-separators-csharp/)
  + [Tuples in C# 7](https://dotnettutorials.net/lesson/tuples-csharp/)
  + [Splitting Tuples in C# 7](https://dotnettutorials.net/lesson/splitting-tuples-csharp/)
  + [Local Functions in C# 7](https://dotnettutorials.net/lesson/local-functions-csharp/)
  + [Ref Returns and Ref Locals in C# 7](https://dotnettutorials.net/lesson/ref-locals-and-ref-returns-chsarp/)
  + [Generalized Async Return Types in C# 7](https://dotnettutorials.net/lesson/generalized-async-return-types/)
  + [Expression Bodied Members in C#](https://dotnettutorials.net/lesson/expression-bodied-members-csharp/)
  + [Thrown Expression in C#](https://dotnettutorials.net/lesson/thrown-expressions-csharp/)
  + [Async Main in C#](https://dotnettutorials.net/lesson/async-main-csharp/)
* **C# 8 New Features**
  + [C# 8 New Features](https://dotnettutorials.net/lesson/csharp-8-new-features/)
  + [ReadOnly Structs in C#](https://dotnettutorials.net/lesson/readonly-structs-in-csharp-8/)
  + [Default Interface Methods in C#](https://dotnettutorials.net/lesson/default-interface-methods-csharp-8/)
  + [Pattern Matching in C#](https://dotnettutorials.net/lesson/pattern-matching-in-csharp-8/)
  + [Using Declarations in C#](https://dotnettutorials.net/lesson/using-declarations-csharp-8/)
  + [Static Local Functions in C#](https://dotnettutorials.net/lesson/static-local-functions-in-csharp-8/)
  + [Disposable Ref Structs in C#](https://dotnettutorials.net/lesson/disposable-ref-structs-in-csharp-8/)
  + [Nullable Reference Types in C# 8](https://dotnettutorials.net/lesson/nullable-reference-types-in-csharp-8/)
  + [Asynchronous Streams in C#](https://dotnettutorials.net/lesson/asynchronous-streams-in-csharp-8/)
  + [Asynchronous Disposable in C#](https://dotnettutorials.net/lesson/asynchronous-disposable-in-csharp-8/)
  + [Indices and Ranges in C#](https://dotnettutorials.net/lesson/indices-and-ranges-in-csharp-8/)
  + [Null-Coalescing Assignment Operator in C#](https://dotnettutorials.net/lesson/null-coalescing-assignment-operator-in-csharp-8/)
  + [Unmanaged Constructed Types in C#](https://dotnettutorials.net/lesson/unmanaged-constructed-types-in-csharp-8/)
  + [Stackalloc in in C#](https://dotnettutorials.net/lesson/stackalloc-in-nested-expressions-in-csharp-8/)
* **Entity Framework – Basics**
  + [Introduction to Entity Framework](https://dotnettutorials.net/lesson/introduction-to-entity-framework/)
  + [Entity Framework Architecture](https://dotnettutorials.net/lesson/entity-framework-architecture/)
  + [Entity Framework Context Class](https://dotnettutorials.net/lesson/context-class-in-entity-framework/)
  + [Entities in Entity Framework](https://dotnettutorials.net/lesson/entities-in-entity-framework/)
  + [Types of Entities in Entity Framework](https://dotnettutorials.net/lesson/types-of-entities-in-entity-framework/)
  + [Entity States in Entity Framework](https://dotnettutorials.net/lesson/entity-state-in-entity-framework/)
  + [Development Approach with Entity Framework](https://dotnettutorials.net/lesson/development-approach-with-entity-framework/)
* **Entity Framework Database First Approach**
  + [Introduction to Entity Framework Database First Approach](https://dotnettutorials.net/lesson/entity-framework-database-first-approach/)
  + [Entity Data Model in Entity Framework](https://dotnettutorials.net/lesson/entity-data-model-in-entity-framework/)
  + [Model Browser in Entity Framework](https://dotnettutorials.net/lesson/model-browser-in-entity-framework/)
  + [DbContext Class in Entity Framework](https://dotnettutorials.net/lesson/dbcontext-in-entity-framework/)
  + [DbSet in Entity Framework](https://dotnettutorials.net/lesson/dbset-in-entity-framework/)
  + [Relationships Between Entities in Entity Framework](https://dotnettutorials.net/lesson/relationships-between-entities-in-entity-framework/)
  + [CRUD Operations in Entity Framework](https://dotnettutorials.net/lesson/crud-operations-in-entity-framework/)
  + [Querying in Entity Framework](https://dotnettutorials.net/lesson/querying-in-entity-framework/)
  + [LINQ to Entities Queries in Entity Framework](https://dotnettutorials.net/lesson/linq-to-entities-queries-in-entity-framework/)
  + [Eager Loading in Entity Framework](https://dotnettutorials.net/lesson/eager-loading-in-entity-framework/)
  + [Lazy Loading in Entity Framework](https://dotnettutorials.net/lesson/lazy-loading-in-entity-framework/)
  + [Lazy Loading vs Eager Loading in Entity Framework](https://dotnettutorials.net/lesson/lazy-loading-vs-eager-loading-in-entity-framework/)
  + [Explicit Loading in Entity Framework](https://dotnettutorials.net/lesson/explicit-loading-in-entity-framework/)
  + [Disconnected Entities in Entity Framework](https://dotnettutorials.net/lesson/disconnected-entities-in-entity-framework/)
  + [Saving Disconnected Entity in Entity Framework](https://dotnettutorials.net/lesson/disconnected-entity-in-entity-framework/)
  + [Asynchronous Programming with Entity Framework](https://dotnettutorials.net/lesson/asynchronous-programming-with-entity-framework/)
  + [Bulk Insert, Update and Delete in Entity Framework](https://dotnettutorials.net/lesson/bulk-insert-update-and-delete-inentity-framework/)
  + [BulkInsert in Entity Framework](https://dotnettutorials.net/lesson/bulkinsert-in-entity-framework/)
  + [BulkUpdate in Entity Framework](https://dotnettutorials.net/lesson/bulkupdate-in-entity-framework/)
  + [BulkDelete in Entity Framework](https://dotnettutorials.net/lesson/bulkdelete-in-entity-framework/)
  + [Stored Procedure in Entity Framework](https://dotnettutorials.net/lesson/stored-procedure-in-entity-framework/)
  + [Stored Function in Entity Framework](https://dotnettutorials.net/lesson/stored-function-in-entity-framework/)
  + [Views in Entity Framework](https://dotnettutorials.net/lesson/views-in-entity-framework/)
  + [Transactions in Entity Framework](https://dotnettutorials.net/lesson/transactions-in-entity-framework/)
  + [Validate Entity in Entity Framework](https://dotnettutorials.net/lesson/validate-entity-in-entity-framework/)
  + [DbEntityEntry Class in Entity Framework](https://dotnettutorials.net/lesson/dbentityentry-class-in-entity-framework/)
  + [Change Tracking in Entity Framework](https://dotnettutorials.net/lesson/change-tracking-in-entity-framework/)
  + [Logging Database Commands in Entity Framework](https://dotnettutorials.net/lesson/logging-database-commands-in-entity-framework/)
* **Entity Framework Code First Approach**
  + [Entity Framework Code First Approach](https://dotnettutorials.net/lesson/introduction-to-entity-framework-code-first-approach/)
  + [Default Code-First Conventions in Entity Framework](https://dotnettutorials.net/lesson/code-first-conventions-in-entity-framework/)
  + [Database Initialization in Entity Framework Code First](https://dotnettutorials.net/lesson/database-initialization-in-entity-framework-code-first/)
  + [Database Initialization Strategies in Entity Framework Code-First](https://dotnettutorials.net/lesson/database-initialization-strategies-in-entity-framework-code-first/)
  + [Configure Domain Classes in Entity Framework](https://dotnettutorials.net/lesson/configure-domain-classes-in-entity-framework/)
  + [Data Annotation Attributes in Entity Framework](https://dotnettutorials.net/lesson/data-annotation-attributes-in-entity-framework/)
  + [Table Attribute in Entity Framework](https://dotnettutorials.net/lesson/table-attribute-in-entity-framework/)
  + [Column Attribute in Entity Framework](https://dotnettutorials.net/lesson/column-attribute-in-entity-framework/)
  + [Key Attribute in Entity Framework](https://dotnettutorials.net/lesson/key-attribute-in-entity-framework/)
  + [ForeignKey Attribute in Entity Framework](https://dotnettutorials.net/lesson/foreignkey-attribute-in-entity-framework/)
  + [Index Attribute in Entity Framework](https://dotnettutorials.net/lesson/index-attribute-in-entity-framework/)
  + [InverseProperty Attribute in Entity Framework](https://dotnettutorials.net/lesson/inverseproperty-attribute-in-entity-framework/)
  + [NotMapped Attribute in Entity Framework](https://dotnettutorials.net/lesson/notmapped-attribute-in-entity-framework/)
  + [Required Attribute in Entity Framework](https://dotnettutorials.net/lesson/required-attribute-in-entity-framework/)
  + [MaxLength and MinLength Attribute in Entity Framework](https://dotnettutorials.net/lesson/maxlength-and-minlength-attribute-in-entity-framework/)
  + [StringLength Attribute in Entity Framework](https://dotnettutorials.net/lesson/stringlength-attribute-in-entity-framework/)
  + [DatabaseGenerated Attribute in Entity Framework](https://dotnettutorials.net/lesson/databasegenerated-attribute-in-entity-framework/)
  + [TimeStamp Attribute in Entity Framework](https://dotnettutorials.net/lesson/timestamp-attribute-in-entity-framework/)
  + [ConcurrencyCheck Attribute in Entity Framework](https://dotnettutorials.net/lesson/concurrencycheck-attribute-in-entity-framework/)
  + [Fluent API Configurations in Entity Framework](https://dotnettutorials.net/lesson/fluent-api-configurations-in-entity-framework/)
  + [Entity Configurations using Entity Framework Fluent API](https://dotnettutorials.net/lesson/entity-configurations-using-entity-framework-fluent-api/)
  + [Property Configurations using Entity Framework Fluent API](https://dotnettutorials.net/lesson/property-configurations-using-entity-framework-fluent-api/)
  + [One-to-One Relationships in Entity Framework Fluent API](https://dotnettutorials.net/lesson/one-to-one-relationships-in-entity-framework-fluent-api/)
  + [One-to-Many Relationships in Entity Framework Fluent API](https://dotnettutorials.net/lesson/one-to-many-relationships-in-entity-framework-fluent-api/)
  + [Many-to-Many Relationships in Entity Framework Fluent API](https://dotnettutorials.net/lesson/many-to-many-relationships-in-entity-framework-fluent-api/)
  + [Existing Database using Entity Framework Code First Approach](https://dotnettutorials.net/lesson/existing-database-using-entity-framework-code-first/)
  + [Stored Procedures in Entity Framework Code-First](https://dotnettutorials.net/lesson/stored-procedures-in-entity-framework-code-first/)
  + [Cascade Delete in Entity Framework Code First Approach](https://dotnettutorials.net/lesson/cascade-delete-in-entity-framework-code-first/)
  + [Seed Data in Entity Framework Code-First](https://dotnettutorials.net/lesson/seed-data-in-entity-framework-code-first/)
  + [Automated Database Migration in Entity Framework Code-First](https://dotnettutorials.net/lesson/automated-database-migration-in-entity-framework-code-first/)
  + [Code-Based Migration in Entity Framework Code First](https://dotnettutorials.net/lesson/code-based-migration-in-entity-framework-code-first/)
  + [Introduction to ASP.NET Core Web API](https://dotnettutorials.net/lesson/introduction-to-asp-net-core-web-api/)
  + [HTTP (HyperText Transport Protocol)](https://dotnettutorials.net/lesson/hypertext-transport-protocol/)
  + [Environment Setup for ASP.NET Core Web API Development](https://dotnettutorials.net/lesson/environment-setup-asp-net-core-web-api/)
  + [Creating ASP.NET Core Web API Project using .NET Core CLI](https://dotnettutorials.net/lesson/creating-asp-net-core-web-api-project-using-net-core-cli/)
  + [**How to Test ASP.NET Core Web API Using Postman**](https://dotnettutorials.net/lesson/how-to-test-asp-net-core-web-api-using-postman/)
  + [Creating ASP.NET Core Web API Project in Visual Studio](https://dotnettutorials.net/lesson/asp-net-core-web-api-project-in-visual-studio-2019/)
  + [Default ASP.NET Core Web API Files and Folders](https://dotnettutorials.net/lesson/asp-net-core-web-api-files-and-folders/)
  + [Swagger API in ASP.NET Core Web API](https://dotnettutorials.net/lesson/swagger-api-in-asp-net-core-web-api/)
  + [Controllers in ASP.NET Core Web API](https://dotnettutorials.net/lesson/controllers-in-asp-net-core-web-api/)
  + [Models in ASP.NET Core Web API](https://dotnettutorials.net/lesson/models-in-asp-net-core-web-api/)
* **ASP.NET Core Web API – Routing**
  + [Routing in ASP.NET Core Web API](https://dotnettutorials.net/lesson/routing-in-asp-net-core-web-api/)
  + [Route Parameters and Query Strings in Routing](https://dotnettutorials.net/lesson/variables-and-query-strings-in-routing/)
  + [Multiple URLs for a Single Resource in ASP.NET Core Web API](https://dotnettutorials.net/lesson/multiple-urls-for-a-single-resource-using-routing/)
  + [Token Replacement in ASP.NET Core Web API Routing](https://dotnettutorials.net/lesson/token-replacement-asp-net-core-routing/)
  + [Route Prefix in ASP.NET Core Web API Routing](https://dotnettutorials.net/lesson/base-route-asp-net-core-web-api-routing/)
  + [Route Constraints in ASP.NET Core Web API](https://dotnettutorials.net/lesson/route-constraints-asp-net-core-web-api/)
* **ASP.NET Core Web API – Model Binding**
  + [Model Binding in ASP.NET Core Web API](https://dotnettutorials.net/lesson/model-binding-in-asp-net-core-web-api/)
  + [Model Binding using FromForm in ASP.NET Core Web API](https://dotnettutorials.net/lesson/fromform-in-asp-net-core-web-api/)
  + [Model Binding using FromQuery in ASP.NET Core Web API](https://dotnettutorials.net/lesson/fromquery-in-asp-net-core-web-api/)
  + [Model Binding Using FromRoute in ASP.NET Core Web API](https://dotnettutorials.net/lesson/fromroute-in-asp-net-core-web-api/)
  + [Model Binding Using FromHeader in ASP.NET Core Web API](https://dotnettutorials.net/lesson/fromheader-in-asp-net-core-web-api/)
  + [Model Binding Using FromBody in ASP.NET Core Web API](https://dotnettutorials.net/lesson/frombody-inasp-net-core-web-api/)
  + [Custom Model Binding in ASP.NET Core Web API](https://dotnettutorials.net/lesson/custom-model-bindingin-asp-net-core-web-api/)
  + [How to Apply Binding Attributes to Model Properties in ASP.NET Core Web API](https://dotnettutorials.net/lesson/how-to-apply-binding-attributes-to-model-properties-in-asp-netcore-web-api/)
  + [Content Negotiation in ASP.NET Core Web API](https://dotnettutorials.net/lesson/content-negotiation-in-asp-net-core-web-api/)
  + [Include and Exclude Properties from Model Binding in ASP.NET Core Web API](https://dotnettutorials.net/lesson/include-and-exclude-properties-from-model-binding-in-asp-net-core-web-api/)
* **Automapper in ASP.NET Core Web API**
  + [How to Use Automapper in ASP.NET Core Web API](https://dotnettutorials.net/lesson/automapper-in-asp-net-core-web-api/)
  + [Automapper Complex Mapping in ASP.NET Core Web API](https://dotnettutorials.net/lesson/automapper-complex-mapping-in-asp-net-core-web-api/)
  + [Mapping Complex Type to Primitive Type using AutoMapper in ASP.NET Core Web API](https://dotnettutorials.net/lesson/mapping-complex-type-to-primitive-type-using-automapper-in-asp-net-core-web-api/)
  + [Automapper Reverse Mapping in ASP.NET Core Web API](https://dotnettutorials.net/lesson/automapper-reverse-mapping-in-asp-net-core-web-api/)
  + [Automapper Pre-Condition Mapping in ASP.NET Core Web API](https://dotnettutorials.net/lesson/automapper-pre-condition-mapping-in-asp-net-core-web-api/)
  + [Automapper Post-Condition Mapping in ASP.NET Core Web API](https://dotnettutorials.net/lesson/automapper-post-condition-mapping-in-asp-net-core-web-api/)
  + [Automapper Conditional Mapping in ASP.NET Core Web API](https://dotnettutorials.net/lesson/automapper-conditional-mapping-in-asp-net-core-web-api/)
  + [Automapper Condition, PreCondition, and PostCondition in ASP.NET Core Web API](https://dotnettutorials.net/lesson/automapper-condition-precondition-and-postcondition-in-asp-net-core-web-api/)
  + [Ignore Property Mapping using Automapper in ASP.NET Core Web API](https://dotnettutorials.net/lesson/ignore-property-mapping-using-automapper-in-asp-net-core-web-api/)
  + [Automapper Null Substitution in ASP.NET Core Web API](https://dotnettutorials.net/lesson/automapper-null-substitution-in-asp-net-core-web-api/)
  + [Fixed and Dynamic Values in Destination Property using Automapper in ASP.NET Core Web API](https://dotnettutorials.net/lesson/fixed-and-dynamic-values-in-destination-property-using-automapper-inasp-net-core-web-api/)
* **HTTP Methods in ASP.NET Core Web API**
  + [HTTP Methods in ASP.NET Core Web API](https://dotnettutorials.net/lesson/http-methods-in-asp-net-core-web-api/)
  + [HTTP GET Method in ASP.NET Core Web API](https://dotnettutorials.net/lesson/http-get-method-in-asp-net-core-web-api/)
  + [HTTP Post Method in ASP.NET Core Web API](https://dotnettutorials.net/lesson/http-post-method-in-asp-net-core-web-api/)
  + [HTTP PUT Method in ASP.NET Core Web API](https://dotnettutorials.net/lesson/http-put-method-in-asp-net-core-web-api/)
  + [HTTP PATCH Method in ASP.NET Core Web API](https://dotnettutorials.net/lesson/http-patch-method-in-asp-net-core-web-api/)
  + [HTTP DELETE Method in ASP.NET Core Web API](https://dotnettutorials.net/lesson/http-delete-method-in-asp-net-core-web-api/)
  + [HTTP HEAD Method in ASP.NET Core Web API](https://dotnettutorials.net/lesson/http-head-method-in-asp-net-core-web-api/)
  + [HTTP OPTIONS Method in ASP.NET Core Web API](https://dotnettutorials.net/lesson/http-options-method-in-asp-net-core-web-api/)
* **Logging – ASP.NET Core Web API**
  + [Logging in ASP.NET Core Web API](https://dotnettutorials.net/lesson/logging-in-asp-net-core-web-api/)
  + [Default Logging in ASP.NET Core Web API](https://dotnettutorials.net/lesson/default-logging-in-asp-net-core-web-api/)
  + [How to Implement Logging using Serilog in ASP.NET Core Web API](https://dotnettutorials.net/lesson/logging-using-serilog-in-asp-net-core-web-api/)
  + [Logging to Database using Serilog in ASP.NET Core Web API](https://dotnettutorials.net/lesson/logging-to-database-using-serilog-in-asp-net-core-web-api/)
  + [How to Implement Logging using nLog in ASP.NET Core Web API](https://dotnettutorials.net/lesson/logging-using-nlog-in-asp-net-core-web-api/)
  + [Logging to Database using NLog in ASP.NET Core Web API](https://dotnettutorials.net/lesson/logging-to-database-using-nlog-in-asp-net-core-web-api/)
  + [Serilog vs NLog in ASP.NET Core Web API](https://dotnettutorials.net/lesson/serilog-vs-nlog-in-asp-net-core-web-api/)
* **Caching – ASP.NET Core Web API**
  + [Caching in ASP.NET Core Web API](https://dotnettutorials.net/lesson/caching-in-asp-net-core-web-api/)
  + [How to Implement In-Memory Caching in ASP.NET Core Web API](https://dotnettutorials.net/lesson/how-to-implement-in-memory-caching-in-asp-net-core-web-api/)
  + [How to Create Custom In-Memory Cache in ASP.NET Core Web API](https://dotnettutorials.net/lesson/how-to-create-custom-in-memory-cache-in-asp-net-core-web-api/)
  + [How to Implement Redis Cache in ASP.NET Core Web API](https://dotnettutorials.net/lesson/how-to-implement-redis-cache-in-asp-net-core-web-api/)
  + [In-Memory Caching vs Distributed Caching in ASP.NET Core](https://dotnettutorials.net/lesson/difference-between-in-memory-caching-and-distributed-caching-in-asp-netcore/)
  + [NCache Cluster and Its Types in Distributed Caching](https://dotnettutorials.net/lesson/ncache-cluster-and-its-types-in-distributed-caching/)
  + [How to Download and Install NCache in Windows](https://dotnettutorials.net/lesson/how-to-download-andinstall-ncache-in-windows/)
  + [How to Implement NCache in ASP.NET Core Web API](https://dotnettutorials.net/lesson/how-to-implement-ncache-in-asp-net-core-web-api/)
* **Security – ASP.NET Core Web API**
  + [How to Store Password in Hash Format in ASP.NET Core Web API](https://dotnettutorials.net/lesson/how-to-store-password-in-hash-format-in-asp-net-core-web-api/)
  + [HMAC Authentication in ASP.NET Core Web API](https://dotnettutorials.net/lesson/hmac-authentication-in-asp-net-core-web-api/)
  + [Encryption and Decryption in ASP.NET Core Web API](https://dotnettutorials.net/lesson/encryption-and-decryption-in-asp-net-core-web-api/)
  + [Asymmetric Encryption in ASP.NET Core Web API](https://dotnettutorials.net/lesson/asymmetric-encryption-in-asp-net-core-web-api/)
  + [Basic Authentication in ASP.NET Core Web API](https://dotnettutorials.net/lesson/basic-authentication-in-asp-net-core-web-api/)
  + [Role-Based Basic Authentication in ASP.NET Core Web API](https://dotnettutorials.net/lesson/role-based-basic-authentication-in-asp-net-core-web-api/)
  + [JWT Authentication in ASP.NET Core Web API](https://dotnettutorials.net/lesson/jwt-authentication-in-asp-net-core-web-api/)
  + [Refresh Token in ASP.NET Core Web API using JWT Authentication](https://dotnettutorials.net/lesson/refresh-token-in-asp-net-core-web-api-using-jwt-authentication/)
  + [Role Based JWT Authentication in ASP.NET Core Web API](https://dotnettutorials.net/lesson/role-based-jwt-authentication-in-asp-net-core-web-api/)
  + [Client Validation using JWT Authentication in ASP.NET Core Web API](https://dotnettutorials.net/lesson/client-validation-using-jwt-authentication-in-asp-net-core-web-api/)
  + [CORS in ASP.NET Core Web API](https://dotnettutorials.net/lesson/cors-in-asp-net-core-web-api/)
* **SSO Implementation**
  + [SSO Authentication in ASP.NET Core Web API](https://dotnettutorials.net/lesson/sso-authentication-in-asp-net-core-web-api/)
  + [Authentication Server using ASP.NET Core Web API](https://dotnettutorials.net/lesson/authentication-server-using-asp-net-core-web-api/)
  + [Resource Server using ASP.NET Core Web API](https://dotnettutorials.net/lesson/resource-server-using-asp-net-core-web-api/)
  + [Client Application One using ASP.NET Core MVC](https://dotnettutorials.net/lesson/client-application-one-using-asp-net-core-mvc/)
  + [Client Application Two using ASP.NET Core MVC](https://dotnettutorials.net/lesson/client-application-two-using-asp-net-core-mvc/)
* **API Versioning in ASP.NET Core**
  + [ASP.NET Core Web API Versioning](https://dotnettutorials.net/lesson/asp-net-core-web-api-versioning/)
  + [ASP.NET Core Web API Versioning using Query String](https://dotnettutorials.net/lesson/asp-net-core-web-api-versioning-using-query-string/)
  + [ASP.NET Core Web API Versioning using URL Path](https://dotnettutorials.net/lesson/asp-net-core-web-api-versioning-using-url-path/)
  + [Header Versioning in ASP.NET Core Web API](https://dotnettutorials.net/lesson/header-versioning-in-asp-net-core-web-api/)
  + [Media Type Versioning in ASP.NET Core Web API](https://dotnettutorials.net/lesson/media-type-versioning-in-asp-net-core-web-api/)
* **Real-Time Application using ASP.NET Core Web API**
  + [E-Commerce Real-Time Application Development using ASP.NET Core Web API](https://dotnettutorials.net/lesson/e-commerce-real-time-application-using-asp-net-core-web-api/)
* **Unit Testing – ASP.NET Core Web API**
  + [Introduction to Unit Testing in ASP.NET Core](https://dotnettutorials.net/lesson/introduction-to-unit-testing-in-asp-net-core/)
  + [Unit Testing in ASP.NET Core Web API using xUnit Framework](https://dotnettutorials.net/lesson/unit-testing-in-asp-net-core-web-api-using-xunit-framework/)
  + [xUnit Framework Theory Attribute in ASP.NET Core](https://dotnettutorials.net/lesson/xunit-framework-theory-attribute-in-asp-net-core/)
  + [Asynchronous Tests using xUnit Framework in ASP.NET Core](https://dotnettutorials.net/lesson/asynchronous-test-using-xunit-framework-in-asp-net-core/)
  + [Fluent Assertions with xUnit Framework in ASP.NET Core](https://dotnettutorials.net/lesson/fluent-assertions-with-xunit-framework-in-asp-net-core/)
* **Minimal API – ASP.NET Core**
  + [Minimal API in ASP.NET Core](https://dotnettutorials.net/lesson/minimal-api-in-asp-net-core/)
  + [Error Handling and Logging in ASP.NET Core Minimal API](https://dotnettutorials.net/lesson/error-handling-and-logging-in-asp-net-core-minimal-api/)
  + [Asynchronous Programming in ASP.NET Core Minimal API](https://dotnettutorials.net/lesson/asynchronous-programming-in-asp-net-core-minimal-api/)
  + [ASP.NET Core Minimal API using Entity Framework Core](https://dotnettutorials.net/lesson/asp-net-core-minimal-api-using-entity-framework-core/)
  + [Endpoint Filters in ASP.NET Core Minimal API](https://dotnettutorials.net/lesson/endpoint-filters-in-asp-net-core-minimal-api/)
  + [JWT Authentication in ASP.NET Core Minimal API](https://dotnettutorials.net/lesson/jwt-authentication-in-asp-net-core-minimal-api/)
  + [API Versioning in ASP.NET Core Minimal API](https://dotnettutorials.net/lesson/api-versioning-in-asp-net-core-minimal-api/)
* **ASP.NET Core Web API – Middleware**
  + [Middleware in ASP.NET Core Web API](https://dotnettutorials.net/lesson/middleware-in-asp-net-core-web-api/)
  + [Run, Use, and Next Method in ASP.NET Core](https://dotnettutorials.net/lesson/run-next-use-methods-in-asp-net-core/)
  + [Map Method in ASP.NET Core](https://dotnettutorials.net/lesson/map-method-in-asp-net-core/)
  + [Custom Middleware in ASP.NET Core](https://dotnettutorials.net/lesson/custom-middleware-in-asp-net-core/)
* **Microservices in ASP.NET Core**
  + [Microservices using ASP.NET Core](https://dotnettutorials.net/lesson/microservices-using-asp-net-core/)
* **Social Media Integration:** When a website integrates Facebook login, it uses Facebook’s API to authenticate users.
* **Payment Gateways:** E-commerce sites use APIs from payment processors like PayPal or Stripe to handle transactions.
* **Weather Data:** Apps that provide weather updates use APIs from weather services to fetch real-time weather data.
* **Google Maps API:** Allows developers to embed maps and location services into their applications.
* **Twilio API:** Enables applications to send SMS, make phone calls, and manage communications.

**Key Characteristics of Web APIs:**

* **HTTP-Based Communication:** Web APIs are designed to work over HTTP, the same protocol used for Web Browsing. This means APIs can be accessed using standard HTTP methods like GET, POST, PUT, DELETE, etc. The API endpoints are typically represented as URLs (Uniform Resource Locators).
* **Data Exchange Formats:** Web APIs use standardized data exchange formats such as JSON (JavaScript Object Notation) and XML (Extensible Markup Language) to structure and transmit data between the client and server. JSON has become the most popular format due to its simplicity and ease of use.
* **RESTful Architecture:** Web APIs are designed to follow Representational State Transfer (REST) principles. A RESTful API is stateless, uses standard HTTP methods, and organizes resources into a hierarchy with unique URLs for each resource.
* **Authentication and Authorization:** Web APIs implement security mechanisms for authentication and authorization to ensure that only authorized clients can access resources or perform specific actions. Common authentication methods include API keys, OAuth, and JWT (JSON Web Tokens).

**What is ASP.NET Core Web API?**

ASP.NET Core Web API is a framework for building scalable and high-performance Restful Web Services (APIs) using the ASP.NET Core platform. It allows developers to create robust and flexible APIs that various clients can consume, such as web applications, mobile apps, desktop applications, and third-party services.

**Prerequisites to Learn ASP.NET Core Web API**

Learning to develop with ASP.NET Core Web API involves understanding general development concepts and specific technologies related to Web API development. Here are the prerequisites that are good if you know before learning ASP.NET Core Web API:

* **Basic Knowledge of C#:** ASP.NET Core is built on C#, so a solid understanding of [**C# Programming**](https://dotnettutorials.net/course/csharp-dot-net-tutorials/) is essential. You should be comfortable with C# syntax, basic programming constructs like loops and conditionals, classes and objects, and more advanced concepts such as LINQ, async/await, and exception handling. This is Mandatory.
* **Understanding .NET Core Basics:** Familiarity with the .NET Core framework is important. This includes understanding the .NET Core CLI, the structure of .NET Core applications, basic concepts like dependency injection, and how to use NuGet packages. We have already discussed this in our [**ASP.NET Core Basic**](https://dotnettutorials.net/course/asp-net-core-tutorials/) course, and it is mandatory.
* **Familiarity with Entity Framework Core:** [**Entity Framework Core (EF Core)**](https://dotnettutorials.net/lesson/entity-framework-core/)is the recommended ORM for data access in ASP.NET Core applications. Understanding EF Core for performing CRUD operations with databases is highly beneficial. This is mandatory.
* **Basic Database Knowledge:** Basic knowledge of databases, especially relational databases like [**SQL Server**](https://dotnettutorials.net/course/ms-sql-server/), [**MySQL**](https://dotnettutorials.net/course/mysql-tutorials-for-beginners-and-professionals/), or [**Oracle**](https://dotnettutorials.net/course/oracle-tutorials/), is important. You should know how to design databases, write basic SQL queries, and understand concepts like tables, keys, and relationships. This is mandatory.

**Why Do We Need Web API?**

Suppose you have an idea for developing and launching a product. To do this, you need to develop a website and launch this product. Then what will you do? You will develop a website using any web technologies like ASP.NET MVC, PHP, ASP.NET Core, JSP, etc., that are available on the market. Of course, you will need a database, such as MySQL, Oracle, SQL Server, etc., to store your product’s business data.

So, by combining the website and the database, you will have a fully functional, dynamic website that interacts with the database. Now, after some time, your business grows. Now, along with the website, you also want Android and iOS apps. That means you want three different applications (Website, Android, and iOS) for your business. But remember, you only have one database in the backend, which stores all the business data. So, we have three different applications and one database. So, all three applications have to interact with the same database, as shown in the image below.

If all three applications interact directly with the database, we have some problems. Let us understand the problems first, and then we will see how to overcome the above problems.

**Problems Without Web APIs:**

1. **Duplicate logic for each Application:** The business should have some business logic. We will write the same logic for each application type, which means repeating the same logic for each type of application. This will duplicate our code.
2. **Error-Prone Code:** The business logic is written for each type of application. We have to write the code in three different applications in our example. So, you might miss some code or logic in some applications. This will add more errors to your application.
3. **Some Front-end frameworks cannot communicate directly with the Database:** If you are developing the website (i.e., front-end) using the angular framework, then the angular framework cannot communicate directly with the database. Angular is a front-end framework.
4. **Hard to Maintain:** This type of structure is hard to maintain. We have written the code in many places, and if we want to improve our application, we need to do the same thing in many places.

There are also many other problems that we face in this structure. Let’s see how to overcome the above problems, or we can say why we need Web APIs.

**Need for Web APIs:**

As you can see in the image below, we have three applications on the left and the database on the right.

* **Platform Independence:** Web APIs can be consumed by any client that understands standard web protocols like HTTP, regardless of the underlying programming language.
* **Low Bandwidth:** APIs typically exchange data using JSON or XML, which is much lighter than full HTML views, reducing the amount of data transmitted.
* **Reusability:** APIs promote code reuse. Once an API is defined and implemented, it can be used by multiple clients or applications without requiring significant changes. That means we write the logic in one place, i.e., in our Web API project, and all applications will use the same logic.
* **Security:** APIs can be secured using authentication and authorization mechanisms, ensuring that only Authenticated and authorized clients can access sensitive data or perform certain actions.
* **Extend Application Functionality:** Suppose, first, we develop the website. Then, we can extend and develop an Android App. Again, in the future, if you want to add another type of application, such as iOS, we don’t have to write any logic.

So, the Website, Android, and iOS applications do not have direct access to the database. They only need to communicate with the Web API Project, and it is the Web API project’s responsibility to interact with the database. The entire business logic will be written in the Web API project only, so we need Web API for our project. So, Web API acts as a mediator between the Front-End and Back-End.

ASP.NET Core Web API is cross-platform, supporting Windows, macOS, and Linux, and it uses the features of ASP.NET Core, such as dependency injection, middleware pipeline, versioning, asynchronous programming, and routing to handle HTTP requests and responses efficiently.

##### ****Why We Need ASP.NET Core Web API?****

Nowadays, a web application is not sufficient to reach all customers. People are becoming very smart; they use different types of devices, such as mobile phones, iPhones, tablets, etc., in their daily lives. These devices have many apps that make their lives easy. In simple words, we can say that we are moving toward the app world from the web.

So, if we want to expose our data (business data) to browsers and modern device apps in a fast, secure, and simple way, then we should have an API that is compatible with browsers and all these modern devices.

The ASP.NET Core Web API is a great framework for building HTTP services that can be consumed by a broad range of clients, including browsers, mobiles, iPhones, and tablets. ASP.NET Core Web API provides several benefits:

* **Performance:** ASP.NET Core is known for its high performance and scalability, making it suitable for building high-traffic APIs. It is faster than its predecessor, ASP.NET.
* **Cross-platform:** ASP.NET Core runs on Windows, Linux, and macOS, allowing developers to deploy their APIs on different operating systems.
* **Modularity:** The framework is built with modularity in mind, allowing developers to include only the necessary components, which keeps the application light and responsive.
* **Modern Features:** It supports modern web development features like dependency injection, middleware, and asynchronous programming.
* **Security:** It includes built-in features for securing APIs, such as authentication and authorization mechanisms.
* **Integration:** It easily integrates with various front-end frameworks, databases, and third-party services.

##### ****What are the REST Principles?****

The REST architectural pattern specifies a set of constraints that a system should adhere to. Here are the REST constraints or principles.

##### ****Client-Server Architecture:****

The client and server should be separate entities that communicate over a standardized interface. REST applications are built around the separation of concerns between client and server. Client-server architecture separates the user interface (client) from data storage (database) and processing (server). This separation allows both sides to develop and deploy independently.

For example, in a social media platform, the client (a mobile app or web browser) interacts with the server (RESTful API) to retrieve user profiles (GET /users/{userId}), post updates (POST /posts), or upload media (POST /media). The server processes these requests and returns appropriate responses, maintaining the separation of concerns between client-side and server-side operations.

##### ****Stateless:****

The stateless constraint specifies that client-server communication must be stateless between requests. That means the server should not store any information related to the client on the server. The client’s request should contain all the necessary information so that the server can identify the client and process that request. This ensures that each request can be treated independently by the server.

For example, a client querying a user’s profile would send a request like GET /users/456, including an authentication token. The server responds using only the information in the request.

##### ****Cacheable:****

In real-time applications, some data provided by the server is not changed that frequently, like the list of Countries, States, Cities, Products, etc. RESTful APIs can take advantage of HTTP caching mechanisms. Responses from the server should explicitly indicate whether they can be cached by clients or not. Caching improves performance and reduces server load.

For example, when a client requests a frequently accessed resource (GET /products/{productId}), the server can include caching headers like Cache-Control or Expires in the response. Subsequent requests for the same resource can be served from the client’s cache, reducing the need to fetch data from the server again until the cache expires or is invalidated.

##### ****Resource Identification****

Resource Identification means that each resource (e.g., a user, a book, an order) is uniquely identifiable by a URI (Uniform Resource Identifier). This allows each resource to be accessed and manipulated independently using standard HTTP methods.

For example, consider a blog platform where each blog post is a resource. Each post can be uniquely identified by a URI like https://example.com/posts/{postId}. Here, {postId} represents the unique identifier for each blog post.

##### ****Uniform Interface:****

The Uniform Interface Constraint defines an interface between the client and the server to communicate. The four principles of the uniform interface are:

1. **Resource-based URIs:** Individual resources are identified in the request using URIs.
2. **HTTP Methods:** Standard methods (GET, POST, PUT, DELETE) are used to perform operations on resources.
3. **Self-Descriptive Messages:** Each message includes enough information to describe how to process the request, i.e., whether to fetch the data, update the data, add a new entity, delete an entity, etc.
4. **Hypermedia as the Engine of Application State (HATEOAS):** Clients interact with the application entirely through hypermedia provided, such as body contents, query-string parameters, request headers, and the requested URI (the resource name).

For example, using HTTP methods such as GET, POST, PUT, and DELETE, clients can interact uniformly:

* GET /books to retrieve a list of books.
* GET /books/123 retrieves the book with ID 123.
* POST /books to create a new book.
* PUT /books/123 updates the book with ID 123.
* DELETE /books/123 deletes the book with ID 123.

##### ****Content Negotiation:****

Clients and servers should be able to negotiate the representation of a resource based on client preferences or capabilities. The client specifies its preferences using the Accept header, and the server responds with the appropriate format if it supports it.

For example, a RESTful API for a news website can provide news articles in both JSON and XML formats. When a client sends a request (GET /articles/{articleId}), it can include an Accept header (**Accept: application/json**) to specify that it prefers JSON format. The server then responds with the requested article in JSON format (**Content-Type: application/json**) if it supports it, ensuring flexibility in how clients consume resources.

##### ****Layered System:****

A Layered System architecture allows an API to be composed of multiple layers, such as security, caching, load balancing, and data storage. Each layer is unaware of the other layers except for the one it directly interacts with. For example, we deploy the APIs in server A, store data on server B, and authenticate requests in server C. A client cannot simply tell whether it is connected directly to the server or an intermediary.

For example, consider an e-commerce application where clients interact with a web server that communicates with a database server. The client is unaware of the database server; it only interacts with the web server via HTTP requests. The web server, in turn, interacts with the database server to fetch and store data.

##### ****Code on Demand (Optional)****

Code on Demand allows servers to extend the functionality of clients by sending executable code (like JavaScript) to clients, which the client can execute locally. This is an optional constraint and is used to add flexibility to the client.

For example, a RESTful service might provide filtering logic as JavaScript code, which a client can execute locally to dynamically adjust the interface based on certain criteria, reducing server load and improving user experience.

##### ****Difference Between ASP.NET Web API and ASP.NET Core Web API****

ASP.NET Web API and ASP.NET Core Web API represent two different approaches to building Web APIs (Restful Services) within the ASP.NET framework, with several key distinctions:

###### **Platform Support:**

* ASP.NET Web API: It is built on the .NET Framework, which means it runs only on Windows operating systems.
* ASP.NET Core Web API: It is part of the ASP.NET Core framework, which runs on .NET Core. This allows it to be cross-platform, operating on Windows, Linux, and macOS. This is a significant advantage for developers aiming for broader application deployment across various environments.

###### **Performance**

* ASP.NET Web API: While efficient, it does not match the performance improvements that have been realized in ASP.NET Core, mainly due to the older architecture and its dependence on the .NET Framework.
* ASP.NET Core Web API: Designed to be lightweight and high-performance. ASP.NET Core has been optimized from the ground up to be faster and more modular. It can handle more requests per second and uses fewer resources because of its ability to support asynchronous programming more extensively.

###### **API Design Features**

* ASP.NET Web API: Supports RESTful services but does not have native support for features like versioning and response compression without additional packages or libraries.
* ASP.NET Core Web API: Includes built-in support for advanced API design features, such as API versioning, response caching, and response compression, making it easier to build robust APIs.

###### **Configuration and Hosting**

* ASP.NET Web API: Typically hosted in IIS (Internet Information Services), it heavily relies on system configuration through web.config files for both application settings and dependency configurations.
* ASP.NET Core Web API: Offers more flexibility in hosting options. It can be hosted in IIS, Kestrel (a cross-platform web server built for ASP.NET Core), or even in Docker containers. Configuration can be achieved through various sources like JSON files, environment variables, command-line arguments, etc., without relying on web.config.

###### **Dependency Injection**

* ASP.NET Web API: Supports dependency injection, but it is not built into the framework. Developers often need to use third-party libraries like Unity or Ninject to achieve dependency injection.
* ASP.NET Core Web API: Dependency injection is a first-class citizen in ASP.NET Core. The framework has built-in support for dependency injection, which promotes more loosely coupled code and better manageability.

###### **Middleware Support**

* ASP.NET Web API: This does not have middleware support the way ASP.NET Core does. Custom handlers and modules can be created, but they are generally more complex to implement and integrate.
* ASP.NET Core Web API: Uses a middleware pipeline that is easy to customize and configure. Middleware components can be used to execute code before and after your application handles a request, enabling scenarios like authentication, error handling, and logging with less complexity.

###### **Community and Future Support**

* ASP.NET Web API: While still supported, it is part of the older .NET Framework, which does not receive new feature updates; it only receives bug fixes and security updates.
* ASP.NET Core Web API: Being part of ASP.NET Core, it benefits from active development and new features. The community and Microsoft’s focus are more on ASP.NET Core due to its modern architecture and flexibility.

##### ****Choosing Between ASP.NET Web API and ASP.NET Core Web API:****

* **Platform Requirements:** If you need cross-platform compatibility or want to deploy on Linux or macOS, ASP.NET Core Web API is the better choice.
* **Performance:** For applications requiring high performance and scalability, ASP.NET Core Web API is generally preferred due to its lightweight nature and optimized performance.
* **Legacy Systems:** If you are maintaining or integrating with older systems built on .NET Framework, ASP.NET Web API might still be necessary due to compatibility requirements.

##### ****What is HTTP?****

HTTP stands for Hypertext Transfer Protocol. It is the foundation of data communication for the World Wide Web. It defines how messages are formatted and transmitted and how web servers and browsers should respond to various commands. HTTP operates as a request-response protocol between a client (typically a web browser) and a server.

For example, when we type a website URL like **https://www.example.com** into our browser and press the Enter button, our browser sends an HTTP request to the server hosting example.com and then receives an HTTP response that includes the website’s HTML content.

##### ****Why Do We Need to Know HTTP?****

Understanding HTTP is essential because it is the primary protocol for web browsing and data communication over the internet. Knowledge of HTTP enables developers to build and troubleshoot web applications effectively, understand the flow of data across the internet, optimize web performance, and ensure secure data transmission. So, understanding HTTP is essential for web development and networking for several reasons, such as:

* **Web Development:** Knowing how data is exchanged allows you to build more efficient web applications.
* **Troubleshooting:** Knowledge of HTTP helps diagnose and solve web-related issues.
* **Security:** Understanding HTTP is crucial for implementing web security measures, such as HTTPS (HTTP Secure).

HTTP is used for communication between the client and server. Let us understand what it means by client and server with an example. Suppose you open your web browser, type the URL in the browser, and press the enter button. When you press the enter button, a request is going to the server (called a web server where the application is hosted). Whatever data you send from the web browser to the web server is called a Request, and whatever data you receive from the web server is called a Response. This is how the browser and web server communicate with each other in the form of Requests and Responses. This type of communication is only possible through the HTTP protocol. So, the request can be termed an HTTP Request, and the response can be called HTTP Response.

###### **User Initiates a Request:**

The process begins when a user enters a URL into a web browser or clicks on a hyperlink. The browser interprets this URL to determine which server to contact and what resource to request. For example, the user enters https://www.example.com.

###### **DNS Resolution:**

Before the browser can send an HTTP request, it needs to determine the IP address of the server that hosts the URL. This is done through a Domain Name System (DNS) lookup. The browser checks its cache to see if it has recently requested the IP address for this domain name (e.g., www.example.com). If not found, it makes a DNS query to resolve the domain name into an IP address. Let’s assume the DNS server returns 192.0.2.1 as the IP address for www.example.com.

###### **Establishing a TCP Connection:**

The browser establishes a TCP (Transmission Control Protocol) connection to the server using the resolved IP address and port (typically port 80 for HTTP and port 443 for HTTPS). This involves a TCP handshake to set up the connection parameters. For example, the browser initiates a TCP connection to 192.0.2.1 on port 443.

###### **Sending an HTTP Request:**

Through the established TCP connection, the browser sends an HTTP request to the server. This request includes:

* An HTTP method (e.g., GET, POST).
* The path to the resource.
* HTTP headers (e.g., User-Agent, Accept types).
* Optionally, a body (for methods like POST).

**For example:**

* GET /index.html HTTP/1.1
* Host: www.example.com
* User-Agent: Mozilla/5.0
* Accept: text/html

###### **Server Processes Request:**

The server receives the HTTP request and processes it:

* **Request Parsing:** The server parses the HTTP method, URL, headers, and any data in the request body.
* **Handle Request:** Depending on the URL and method, the server retrieves or modifies resources. For instance, it might fetch a file, query a database, or execute server-side scripts.

###### **Server Send an HTTP Response:**

After processing the request, the server sends an HTTP response back to the browser. This response includes:

* Status Line: Includes the HTTP version, a status code (e.g., 200 OK, 404 Not Found), and a status message.
* Headers: Provide additional information (e.g., Content-Type, Content-Length).
* Response Body: Contains the actual data being sent back, such as HTML content, images, etc.

**For example:**

HTTP/1.1 200 OK

Content-Type: text/html

Content-Length: 137

**<html>**

**<body>**

**<h1>**Welcome to Example.com!**</h1>**

**</body>**

**</html>**

###### **Browser Displays Content**

Upon receiving the HTTP response, the browser reads the status code to understand the result of the request (e.g., success, redirection, error). If the response is successful (e.g., status code 200), the browser interprets the body of the response—typically HTML—and renders it on the screen. It may also process additional resources like CSS, JavaScript, or media files linked within the HTML. For example, the browser renders the HTML content and displays “Welcome to Example.com!” on the webpage.

###### **Closing the Connection**

Depending on the HTTP headers (e.g., Connection: keep-alive), the TCP connection might be closed after the response is delivered. The connection might stay open for a period, allowing subsequent requests to the same server to reuse it, which speeds up those transactions.

##### ****Start Line or Request Line****

The Request Line of an HTTP request contains three parts:

* **HTTP Method:** This indicates the action the client wants to perform. Common methods include GET (retrieve data), POST (submit data to be processed), PUT (update data), DELETE (remove data), and others like PATCH, HEAD, and OPTIONS.
* **Request Target/URL:** This specifies the resource being requested. It can be a full URL or just a path to a resource on the server that the client wants to interact with, such as/index.html.
* **HTTP Version:** This specifies the version of the HTTP protocol being used (e.g., HTTP/1.1, HTTP/2).

Example of Start Line or Request Line: GET /index.html HTTP/1.1

##### ****Request Headers:****

Each HTTP Request can contain one or more Request Headers. Headers provide additional information about the request. They are key-value pairs separated by a colon and are sent one per line after the request line. Common request headers include:

* Host: Specifies the domain name of the server.
* User-Agent: Identifies the client software initiating the request (e.g., the browser or application).
* Accept:  Tells the server what content types the client can handle.
* Content-Type:  When the request includes a body (like a POST, PUT, or PATCH request), this header indicates the media type of the body.
* Authorization: Credentials for authenticating the client.
* Cookie: Includes any cookies that the client has for this domain. This is used for state management.
* Cache-Control: Directives for caching mechanisms in both requests and responses.

#### ****HTTP Response Components:****

An HTTP response is what a server sends back to the client after processing an HTTP request. The response consists of several key components that convey the status of the request and any data or resources the server is sending back. The HTTP response contains the following key components.

1. **HTTP Status Code:** It must have a Status Code indicating the status of the HTTP Request. 200 Indicates successful, 500 indicates internal server error, 404 indicates resource not found, etc.
2. **Response Headers:** It can have one or more response headers.
3. **Data:**Response can have data, i.e., return to the client.

###### **Status Line:**

The status line of an HTTP response contains three parts:

* **HTTP Version:** Specifies the version of the HTTP protocol that the server is using (e.g., HTTP/1.1, HTTP/2).
* **Status Code:** A three-digit number that indicates the outcome of the request. These codes are categorized into various classes (e.g., 2xx for success, 4xx for client errors, 5xx for server errors).
* **Status Text:** A brief, human-readable phrase provides a description of the status code (e.g., “OK”, “Not Found”).

Example Status Line: HTTP/1.1 200 OK

##### ****Response Headers:****

Each HTTP Response can have one or more Response Headers. HTTP response headers provide additional information about the response and also about the server itself. They are key-value pairs separated by a colon and are sent one per line after the status line. Some common response headers include:

* Content-Type: Specifies the media type of the response body (e.g., text/html, application/json).
* Content-Length: The length of the response body in bytes.
* Server: Information about the server software handling the request.
* Set-Cookie: Used to send cookies from the server to the client browser.
* Cache-Control: Specifies directives for caching mechanisms in both requests and responses.
* Date: The date and time at which the message was sent.

##### ****Summary of HTTP Response Status Codes:****

1. **1xx Informational:** The request was received and is continuing.
2. **2xx Successful:** The request was successfully received, understood, and accepted.
3. **3xx Redirection:** Further action is needed to complete the request.
4. **4xx Client Error:** There was an error in the request sent by the client.
5. **5xx Server Error:** The server failed to fulfill a valid request.

<https://dotnettutorials.net/lesson/405-http-status-code-in-asp-net-core-web-api/>

###### **PUT HTTP Method:**

The HTTP PUT request is used to update all the properties of the current resource in the database. What does it mean? For example, we have a table called Product in our database. If we want to update all properties of a particular product, then we need to use

###### **PATCH HTTP Method:**

In some situations, you don’t want to update all the properties of an existing resource; instead, you want to update a few of the properties of an existing resource. In that case, you need to use the PATCH method. So, the PATCH method is similar to the PUT method but is used to update a few resource properties.

For example, if you want to update a few properties (columns) of an existing product, then you need to use the PATCH method. That means if your Product table contains 10 columns, and you want to update only four columns of an existing product, then you need to use the PATCH method. So, it applies partial modifications to a resource.

##### ****Install Postman:****

As we will work with the Web API, we also need one Web API client tool. And for the client tool, we are going to use Postman. To install Postman, click on the below link.

[**https://www.postman.com/downloads/**](https://www.postman.com/downloads/)

Once you click the above link, it will open the webpage below. From the below page, click on the Download the App link to download Postman on your machine. Depending upon your operating system, you can choose either 32-bit or 64-bit. I have installed a 64-bit operating system, so I chose Windows 64-bit, as shown in the image below.

Once you download it, then install it. A postman is an API tool. Postman provides a complete and nice interface to work with APIs. That’s it. We are ready with our environment to develop ASP.NET Core Web API Applications.

In the next article, I will discuss [**Creating an ASP.NET Core Web API Project using .NET Core CLI**](https://dotnettutorials.net/lesson/creating-asp-net-core-web-api-project-using-net-core-cli/). In this article, I explain the **Environment Setup for ASP.NET Core Web API Application Development,** and I hope you enjoy this ASP.NET Core Web API Environment setup for Development article.

#### ****Creating ASP.NET Core Web API Project****

In .NET Core, there are two ways to create a project. They are as follows:

* **Using .NET Core CLI (Command Line Interface)**
* **Using Visual Studio**

<https://dotnettutorials.net/lesson/creating-asp-net-core-web-api-project-using-net-core-cli/>

As you can see in the above image, our ASP.NET Core Web API Application is running on the following port.

**http://localhost:5008**

Now, open the above URL in any of your browsers, and you will get a 404 error.

The swagger will display the details of all the Web APIs available in your project. As you can see in the above image, it shows one API, i.e., /WeatherForecast, and the type is Get. Now click on the /WeatherForecast API to see the details shown in the image below.

**Note:**Swagger is also a client API Tool, and using Swagger, we can also test the Web APIs. If you are using the default ASP.NET Core Web API project, then by default, swagger is installed into the project.