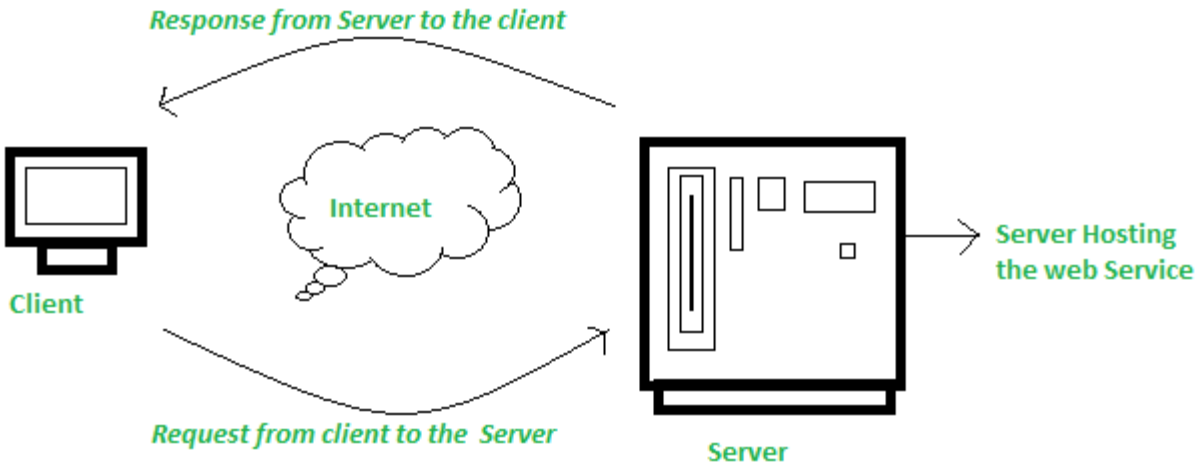
What is Web Service

- A web service is a software module that is intended to carry out a specific set of functions.
- A web service is a standardized method for propagating messages between client and server applications.
- A Web services can be invoked by client over the network.
- The web service would be able to deliver functionality to the client that invoked the web service.
- A web service is a set of open protocols and standards that allow data to be exchanged between different applications or systems.
- Web services can be used by software programs written in a variety of programming languages and running on a variety of platforms to exchange data via computer networks
- Any software, application, or cloud technology that uses standardized web protocols (HTTP or HTTPS) to connect, interoperate, and exchange data messages – commonly XML (Extensible Markup Language) and JSON – across the internet is considered a web service.
- A client invokes a web service by submitting an XML/JSON request, which the service responds with an XML/JSON response.

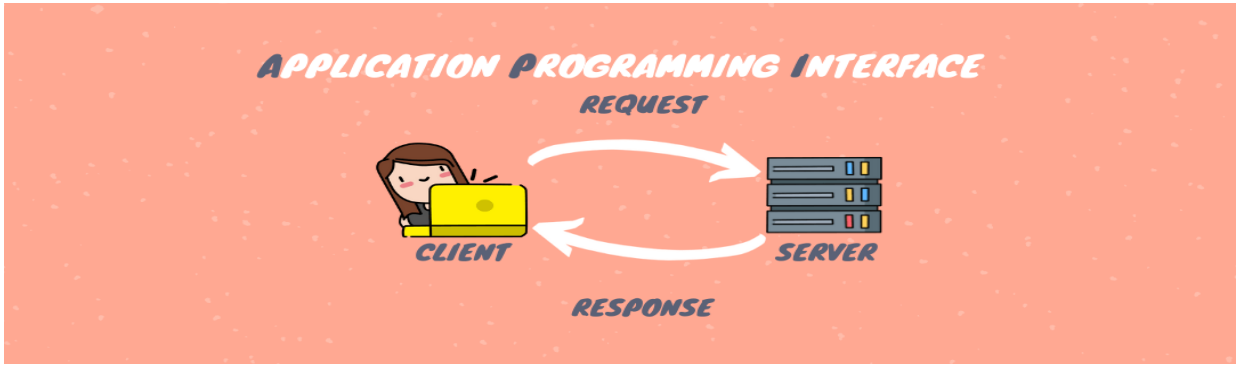
How Does Web Service Work?

- The diagram depicts a very simplified version of how a web service would function. The client would use requests to send a sequence of web service calls to a server that would host the actual web service.



What is Web API

- API stands for Application Programming Interfaces (APIs).
- Application Programming Interfaces (APIs) are basically HTTP services that are used to communicate between applications in a simple and easy way.
- API stands for Application Programming Interface. A Web API, or Web Application Programming Interface, is a set of rules and protocols that allows different software applications to communicate with each other over the internet or a network.
- It enables the exchange of data and functionality between various systems, using HTTP (Hypertext Transfer Protocol) as the communication protocol.



Problems Without Web APIs

- **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.
- **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.
- **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.
- **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.

Advantages of Web API

- **Using Web API, we can avoid code duplication:** We can write the logic in one place, i.e., in our Web API project, and all applications will use that logic.
- **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, we don't have to write any logic.
- **Abstraction:** We have added an extra abstraction layer because we wrote all the business logic in our Web API project. The logic we wrote in the Web API project will not be visible to the front-end developers.
- **Security:** None of the applications can access the database directly, and hence it provides security.

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).

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.
- ASP.NET Core Web API is commonly used for building RESTful APIs that expose data and services over HTTP.
- It's suitable for various scenarios, including building back-end services for web, mobile, and desktop applications, providing data to single-page applications (SPAs), and creating microservices that can be deployed independently.
- Asp.net Core Web API makes it easy to build or crate HTTP services
- Web API services can be consumed by a broad range of clients like
  1. Browsers
  2. Mobile applications
  3. Desktop applications
  4. IOTs(Internet Of Things)
- It can be hosted with in IIS/Tomcat or Windows Azure
- It is very similar to ASP.NET MVC since it contains the MVC features such as routing, controllers, action results, filter, model binders etc.

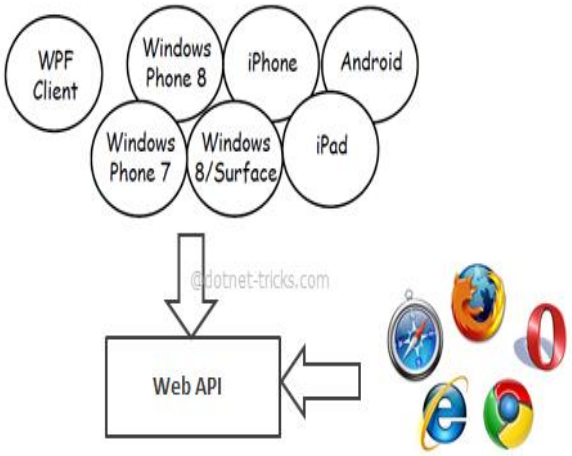
ASP.NET Core Web API Features

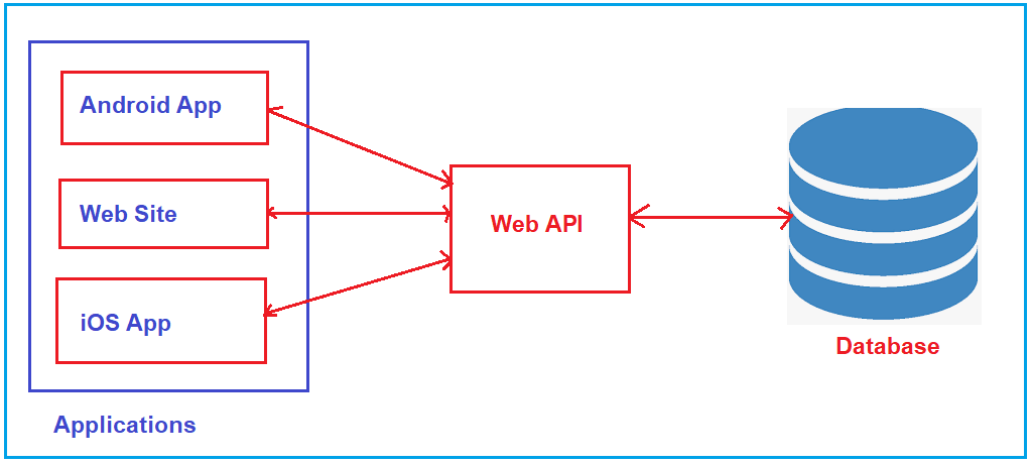
- **Cross-Platform Support:** NET Core Web APIs can run on Windows, Linux, and macOS, making your Web APIs platform-independent.
- **Performance:** NET Core has been designed to have a smaller memory footprint and improved performance compared to its predecessors. It's optimized for modern cloud-based applications.
- **Routing:** Supports attribute routing, which allows the development of -SEO-friendly URLs.
- **Model Binding and Validation:** Automatically maps data from HTTP requests to action method parameters. Model validation is also automatically performed, and any validation errors can be handled and returned to the client.
- **Dependency Injection:** Built-in support for dependency injection (DI). This allows for more modular and testable code.
- **Middleware Support:** NET Core's middleware pipeline enables you to add components that inspect and process HTTP requests/responses, allowing for custom processing like authentication, logging, encryption decryption, etc.
- **Content Negotiation and Serialization:** Automatically serializes your data to and from JSON, XML, or any other format, depending on client preferences and server capabilities.

ASP.NET Core Web API Features Cont..

- **OpenAPI/Swagger Support:** Provides built-in support for generating OpenAPI (formerly Swagger) descriptions of your API, which can then be used to generate beautiful interactive documentation, client SDK generation, and more.
- **Security:** Supports authentication and authorization mechanisms, including OAuth 2.0, Basic Authentication, HMAC, JWT (JSON Web Tokens), and more, to secure your web APIs.
- **Testability:** Designed to support unit testing. You can write tests for your API like any other ASP.NET Core application.
- **Integrated Configuration System:** The Web API can read application configuration data from various sources, like JSON files, environment variables, and more.
- **Environment-based Configuration:** Supports development, staging, and production environments, allowing different configurations for each environment.

Asp.net Core Web API





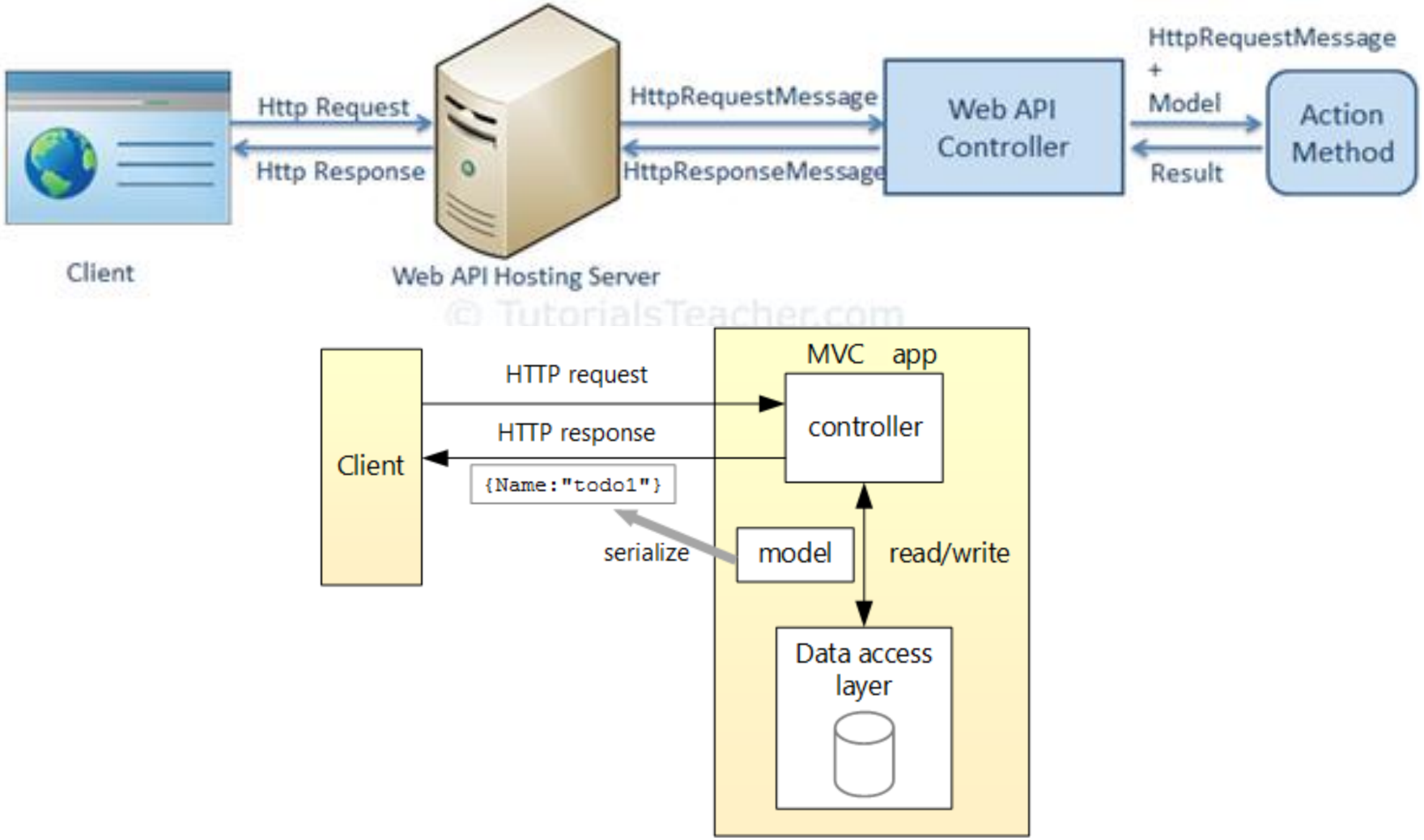
Service Types

- **Web Service**
- It is based on SOAP and return data in XML form.
- It support only HTTP protocol.
- It is not open source but can be consumed by any client that understands xml.
- It can be hosted only on IIS.
- **WCF**
- It is also based on SOAP and return data in XML form.
- It is the evolution of the web service(ASMX) and support various protocols like TCP, HTTP, HTTPS, Named Pipes, MSMQ.
- The main issue with WCF is, its tedious and extensive configuration.
- It is not open source but can be consumed by any client that understands xml.
- It can be hosted with in the applicaion or on IIS or using window service.

Service Types

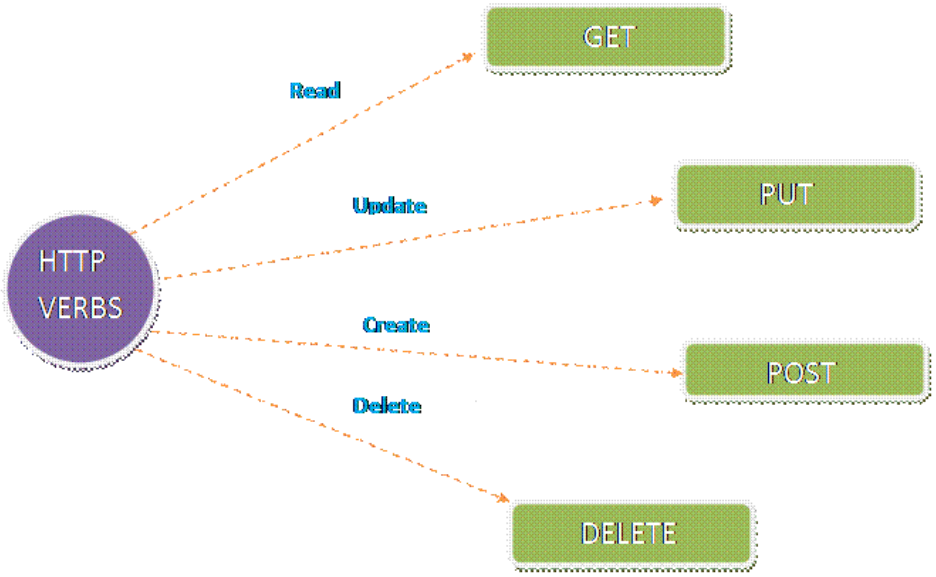
- **Web API**
- This is the new framework for building HTTP services with easy and simple way.
- Web API is open source an ideal platform for building REST-full services over the .NET Core/Framework.
- it use the full features of HTTP (like URIs, request/response headers, caching, versioning, various content formats)
- It also supports the MVC features such as routing, controllers, action results, filter, model binders, IOC container or dependency injection, unit testing that makes it more simple and robust.
- It can be hosted with in the application or on IIS/Azure/AWS.
- It is light weight architecture and good for devices which have limited bandwidth like smart phones.
- Responses are formatted by Web API's MediaTypeFormatter into JSON, XML or whatever format you want to add as a MediaTypeFormatter.

request/response pipeline in Web API



HTTP Verbs in ASP.NET Core Web API

- One of the key features of HTTP is the use of verbs (also known as methods or actions) to perform specific operations on resources.
- In the context of ASP.NET Web API, these HTTP verbs are used to create, read, update, and delete data.
- The following are the five most commonly used HTTP verbs in the ASP.NET Core Web API
- Get
- Post
- Put
- Delete
- Patch



HTTP Verbs in ASP.NET Core Web API

API	Description	Request body	Response body
`GET /api/todoitems`	Get all to-do items	None	Array of to-do items
`GET /api/todoitems/{id}`	Get an item by ID	None	To-do item
`POST /api/todoitems`	Add a new item	To-do item	To-do item
`PUT /api/todoitems/{id}`	Update an existing item	To-do item	None
`DELETE /api/todoitems/{id}`	Delete an item	None	None

HTTP GET

- This verb is used to retrieve data from the server.
- It is a read-only operation that does not change any data on the server.
- A GET request is used to retrieve a representation of a resource from the server.
- Example:
- The following endpoint retrieves a list of all employees.
- GET /api/Employee/ListEmployees

HTTP POST

- This verb is used to create new data on the server.
- It is a write operation that adds a new resource to the server.
- A POST request is used to submit an entity to the specified resource, often causing a change in state on the server.
- Example:
- The following endpoint create or add the employee.
- POST /api/Employee/CreateEmployee .

HTTP PUT

- This verb is used to update existing data on the server.
- It is a write operation that replaces an existing resource with a new one.
- A PUT request is used to update a resource with the client-provided data.
- Example:
- The following endpoint updates employee with ID 1.
- PUT /api/Employee/updateEmployee/1 .

HTTP DELETE

- This verb is used to delete data on the server.
- It is a write operation that removes a resource from the server.
- A DELETE request is used to delete a resource identified by a URI.

- Example:
- The following endpoint deletes employee with ID 1.
- DELETE /api/Employee/deleteEmloyee/1.

HTTP PATCH

- This verb is used to update partial data on the server.
- It is a write operation that modifies a portion of an existing resource.
- A PATCH request is used to update a resource with only the specific changes provided in the request body.
- Example:
- The following endpoint updates a specific property of employee with ID 1.
- PATCH /api/Employee/partialupdate/1

Selecting The Appropriate Method

- A large portion of application functionality can be summed up in the acronym CRUD, which stands for Create, Read, Update, Delete. There are four HTTP methods that correspond to these actions, one for each, like so:
- C - Create - POST
- R - Read - GET
- U - Update - PUT
- D - Delete - DELETE

What is ApiController Attribute in ASP.NET Core Web API?

- The ApiController attribute in ASP.NET Core Web API plays a significant role in developing HTTP API projects.
- It was introduced in ASP.NET Core 2.1 and provides several features that make building robust and well-documented web APIs easier.
- When you apply this attribute to a controller class, it enables various API-specific behaviors and conventions.
- **Attribute Routing Requirement:** The ApiController attribute makes attribute routing a requirement. This means you must use the Route attribute to define the routes. Controllers marked with this attribute won't be accessible through conventional routing.
- **Automatic HTTP 400 Responses:** Model validation errors automatically trigger an HTTP 400 response, which is a big change from earlier versions, where you had to manually check ModelState.IsValid and return a bad request.
- **Required Attribute on Non-Nullable Parameters:** Parameters that are non-nullable are assumed to be required, and if a null value is passed, the API will automatically return a 400 Bad Request.
- **Support for OpenAPI (Swagger) Documentation:** This attribute also facilitates the generation of OpenAPI (Swagger) documentation, as it provides additional metadata that can be useful for generating more descriptive API documentation.

Routing in ASP.NET Core Web API

- Routing in ASP.NET Core Web API application is the process of mapping the incoming HTTP Request (URL) to a particular resource, i.e., controller action method.
- In ASP.NET Core Web API, routing is a fundamental concept that determines how HTTP requests are matched to the actions on controllers.
- There are two primary approaches to routing: Convention-based routing and Attribute routing. Both have their own advantages and use cases.

Convention-based Routing

- Convention-based routing defines routes based on a set of conventions specified in the Program.cs file. Here, routes are defined globally for the application.
- **Advantages:**
- Centralized Configuration: All routes are defined in one place, making it easier to see the big picture of the URL structure of your API.
- Consistency: By following conventions, the routes across different controllers and actions are consistent.
- **Disadvantages:**
- Less Flexible: It is harder to define complex and custom routes that deviate from the established conventions.
- Refactoring Challenges: Changing controller or action names can break routes if not updated in the routing configuration.
- **Use Cases:**
- It is best for applications with a straightforward and uniform URL structure.
- When you want a centralized place to manage routing.

Convention-based Routing

- If you want to use Conventional Routing, please add the MapControllerRoute middleware component to the Program.cs class file. Here, we are configuring the Route Pattern as api/{controller}/{action}/{id?} where id is the optional parameter.

```
app.MapControllerRoute(
name: "default",
pattern: "api/{controller}/{action}/{id?}");
```

Attribute Routing

- Attribute routing uses attributes to define routes directly on controllers and actions. This approach provides more control by allowing custom and complex routes for each action.
- **Advantages:**
- High Flexibility: Enables defining custom routes per action, allowing for complex URL structures.
- Self-documenting: Routes are defined where the action is defined, making it clear what URL maps to what action.
- Support for HTTP Verb Constraints: Easily specify HTTP methods (GET, POST, etc.) for actions.
- **Disadvantages:**
- Scattered Configuration: Routes are spread across controllers and actions, making it harder to get an overview of the API's URL structure.
- Potential for Duplication: There is a possibility of duplicating route patterns, leading to conflicts or confusion.
- Use Cases:
- Ideal for APIs with complex and non-uniform URL patterns.
- When you need fine-grained control over the routing of each action.

Configuring the Routing Middlewares in ASP.NET Core

- To enable Attribute Routing in ASP.NET Core, we must add the following two middleware components to the HTTP Request Processing Pipeline.
- app.UseRouting();
- app.MapControllers();
- **app.UseRouting():** This middleware enables routing capabilities in your ASP.NET Core application. It is responsible for matching incoming HTTP requests to routes that have been defined in your application. UseRouting should be added to the middleware pipeline before any middleware that requires knowledge of the requested endpoint, like authorization or endpoint-specific middleware.
- **app.MapControllers():** This extension method is used to map attribute-routed controllers. It essentially tells the application to look for controllers in your project and creates routes for them based on the attributes you've defined (like [Route], [HttpGet], etc.). This is typically used when you have an API-centric application with controllers handling various HTTP requests.
- In .NET 8, app.UseRouting() and app.MapControllers() are used within the context of an ASP.NET Core application to configure routing and controller endpoints.