**✅ Q1: What is ASP.NET Core? How is it different from ASP.NET MVC?**

**What is ASP.NET Core?**

**ASP.NET Core is:**

* **A modern web framework created by Microsoft.**
* **Cross-platform (runs on Windows, macOS, Linux).**
* **High-performance (much faster than the old ASP.NET MVC).**
* **Open-source and community-driven.**
* **Can build:**
  + **Web APIs**
  + **MVC web apps**
  + **Razor Pages**
  + **Real-time apps (SignalR)**
  + **Microservices**
  + **Cloud-native apps (Azure, AWS, Docker)**

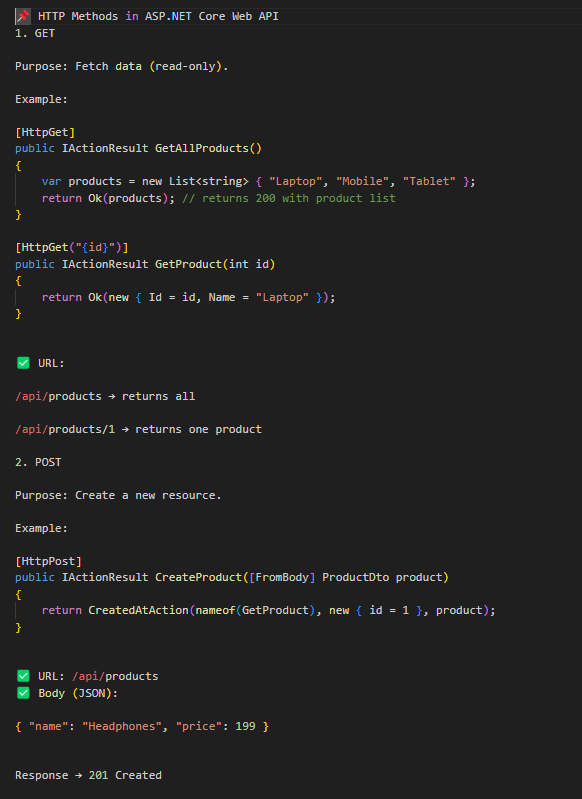
**👉 Think of it as the next generation of ASP.NET that is more flexible, faster, and portable.**

**⚖️ ASP.NET MVC vs. ASP.NET Core (Detailed Comparison)**

| **Feature** | **ASP.NET MVC (Old)** | **ASP.NET Core (New)** |
| --- | --- | --- |
| **Platform** | **Runs only on Windows because it depends on the full .NET Framework.** | **Runs on Windows, Linux, macOS (thanks to .NET Core).** |
| **Web Server** | **Works only with IIS (Internet Information Services).** | **Works with Kestrel (default), but can also integrate with IIS, Nginx, Apache.** |
| **Dependency Injection (DI)** | **No built-in DI. Developers had to use third-party libraries like Autofac, Ninject, Unity.** | **Built-in DI container → no need for third-party libraries.** |
| **Hosting Model** | **Hosted under System.Web (heavy, memory-hungry, tied to IIS).** | **Lightweight self-hosting with Kestrel → can be hosted anywhere (Docker, Cloud, etc.).** |
| **Performance** | **Moderate speed (because of heavy System.Web and IIS).** | **Much faster (benchmarked among the fastest web frameworks in the world).** |
| **Modularity** | **Monolithic: all features are bundled in System.Web (even unused ones load).** | **Highly modular: uses middleware pipeline → you include only what you need.** |
| **Configuration** | **Relies on web.config (XML-based, rigid).** | **Uses appsettings.json, environment variables, and flexible configuration providers.** |
| **Open-source** | **Partially open but mostly Microsoft-controlled.** | **Fully open-source on GitHub with community contributions.** |
| **Deployment** | **Needs IIS and Windows Server.** | **Can deploy to Docker, Linux, Azure, AWS, or even run as a console app.** |

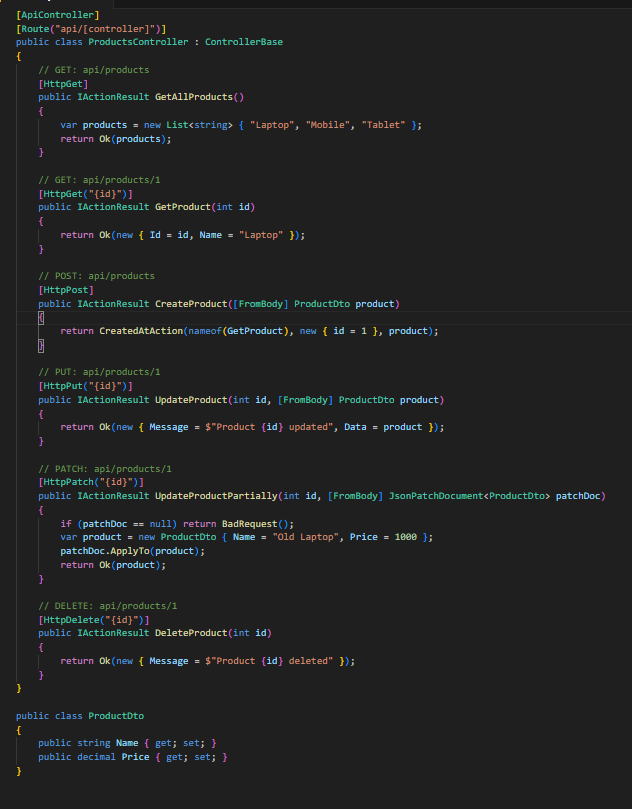
**✅Q2: What are the main components of ASP.NET Core Web API?**

**1. Controllers**

* **What it is:** Controllers are classes that define API **endpoints** (routes) and contain business logic.
* **How it works:** Each controller has **methods (actions)** that respond to HTTP requests like GET, POST, PUT, DELETE.
* 

A screenshot of a computer program

AI-generated content may be incorrect.



2. **What is Routing?**

* **Definition**: Routing is the **process of mapping an incoming HTTP request (URL + HTTP method) to a controller action method**.
* Without routing, ASP.NET Core won’t know **which controller and which method** should handle the request.

👉 Example:

* Request: GET https://localhost:5001/api/products/5
* Routing decides: "This goes to ProductsController.GetProduct(int id)."

**🔹 Types of Routing in ASP.NET Core**

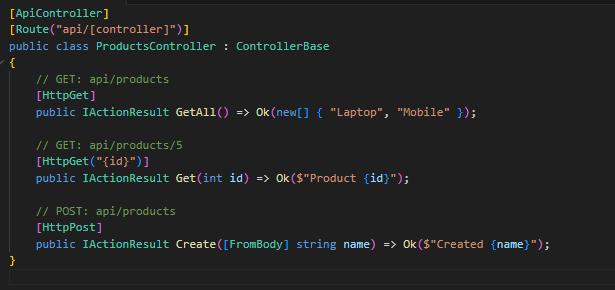
**1. Convention-based Routing (Older style, used in MVC apps)**

* Routes are defined in a **central place** (Program.cs or Startup.cs).
* Uses placeholders like {controller}, {action}, {id?}.
* Example:
* app.MapControllerRoute(
* name: "default",
* pattern: "{controller=Home}/{action=Index}/{id?}");
* **How it works:**
  + /Products/Details/5 → ProductsController.Details(5)
  + /Home/Index → HomeController.Index()
  + If nothing is provided → defaults to HomeController.Index()

✅ Best for **MVC apps with Razor views**, not commonly used for APIs anymore.

**2. Attribute Routing (Most common for Web API)**

* Routes are defined directly **on controllers and actions** using attributes.
* Example:

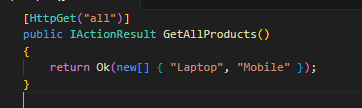


* [Route("api/[controller]")] → ProductsController → /api/products
* [HttpGet("{id}")] → /api/products/5

✅ Best for **Web APIs** (clear, flexible, modern).

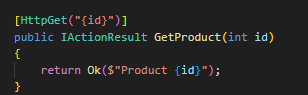
**🔹 Route Templates and Parameters**

**1. Static Route**



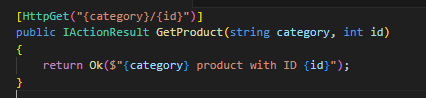
* URL: /api/products/all

**2. Route Parameters**



* URL: /api/products/10
* Maps id = 10

**3. Multiple Parameters**

****

* URL: /api/products/electronics/5
* Maps category = "electronics", id = 5

**4. Optional Parameters**

A screen shot of a computer code

AI-generated content may be incorrect.

* URL: /api/products → "All products"
* URL: /api/products/2 → "Product 2"

**5. Route Constraints**

You can restrict parameters by type/format:

A computer screen shot of text

AI-generated content may be incorrect.

* /api/products/123 → goes to GetById
* /api/products/laptop → goes to GetByName

**🔹 Mixed Routing Example (API + MVC)**

**A screenshot of a computer program

AI-generated content may be incorrect.**

* /Orders/Index → MVC route
* /api/orders/5 → API route

**🔹 Special Tokens in Routing**

* [controller] → replaced with controller name (without "Controller").
* [action] → replaced with method name.
* Example:

A screen shot of a computer code

AI-generated content may be incorrect.

* /api/products/list
* /api/products/details/5

**✅ Summary**

* **Routing** = maps URL → Controller → Action.
* **Convention-based routing** = centralized, good for MVC.
* **Attribute routing** = local to controllers, best for APIs.
* **Features:** route parameters, optional params, constraints, special tokens.

**🔹 Why ProductsController and not ProdController?**

1. **ASP.NET Core convention:**  
   By default, routing uses the **controller class name (without the "Controller" suffix)** when resolving [controller] in routes.

Example:

A screen shot of a computer code

AI-generated content may be incorrect.

* + Class name = **ProductsController**
  + [controller] → "products"
  + URL: /api/products/details/5 → ProductsController.Details(5)

1. **If you name it ProdController:**

A screenshot of a computer

AI-generated content may be incorrect.

* + Class name = **ProdController**
  + [controller] → "prod"
  + URL: /api/prod/details/5 → ProdController.Details(5)

👉 So yes, you **can** name it ProdController, but then your URL will be /api/prod/details/5, not /api/products/details/5.

**🔹 Why do people prefer full names like ProductsController?**

* **Readability:** Clearer meaning → easy for teams to understand.
* **Consistency:** Microsoft docs and templates use plural + Controller (e.g., ProductsController, OrdersController).
* **REST API standard:** Endpoints usually use **plural resource names** (/api/products, /api/orders) instead of abbreviations like /api/prod.

**🔹 Can you override this?**

Yes ✅ You don’t have to depend on [controller]. You can hardcode routes:

A computer code on a white background

AI-generated content may be incorrect.

* Even though the class is ProdController, the URL is /api/products/details/5.

✅ **Summary**

* By convention, [controller] takes the class name minus "Controller".
* ProductsController → /api/products/...
* ProdController → /api/prod/...
* You can **override** it with a custom [Route("api/products")] if you want short class names but meaningful URLs.

**3. Middleware**

* **What it is:** Middleware are components that process requests **in a pipeline** (one after another).
* **Common Middleware:**
  + Logging
  + Exception handling
  + Authentication & Authorization
  + Routing
* **Example:** In Program.cs
* app.UseHttpsRedirection();
* app.UseAuthentication();
* app.UseAuthorization();
* app.MapControllers();

👉 Each middleware decides whether to pass the request forward or stop it.

3. **🔹 What is Dependency Injection (DI)?**

* **Dependency** = A class depends on another class/service to work.
* **Injection** = Instead of creating the dependency manually inside the class, we "inject" it from outside.

👉 ASP.NET Core has **built-in DI container**, so we don’t need third-party frameworks (like Autofac, Unity, etc., used in old ASP.NET MVC).

With DI (good way):

A screen shot of a computer program

AI-generated content may be incorrect.

