ASP.NET Core

▨ Status	Done
▨ Last edited time	@July 30, 2025 10:23 PM

> Definition: ASP.NET Core is a cross-platform, high-performance, open-source framework used for building modern web, IoT, and cloud-enabled applications.

What We Can Build with ASP.NET Core

- Web apps and services (Web APIs)
- IoT (Internet of Things) apps
- Mobile app backends

Platform Support

OS	Supported
Windows	✅
Linux	✅
macOS	✅

Deployment Options

- **Cloud** (like Azure, AWS)
- **On-premises** (your own server or network)

.NET Core Platform Versions

Year	.NET Version
2022	.NET 8
2024	.NET 9

Two Main Approaches to Build Web APIs

Approach	Description	Suitable For
MVC	Model-View-Controller pattern (structured)	Large applications
Minimal API	Simple syntax, top-level code (faster start)	Small applications

.NET CLI vs Visual Studio Code

Feature	.NET CLI	VS Code Community Edition
Interface	Command Line	Graphical User Interface (GUI)
Speed	Faster for experienced users	Easy for beginners
Usage	dotnet new, run, build	Click and edit in UI

Useful Commands

```
dotnet new webapi        # Create a new Web API project
dotnet run              # Run your project
dotnet build            # Build the project
ctrl + F5              # Run without debugging in VS Code
```

Top-Level Statement Feature (C# 9+)

No need to write `Main()` method or namespace in `Program.cs`.

Example:

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();
```

```
app.MapGet("/", () => "Hello World!");
app.Run();
```

Important ASP.NET Core Components

1. `createBuilder` and `builder.Services`

- Used to configure services and build the app.

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllers(); // Add services
```

2. Middleware & HTTP Request Pipeline

Definition: Middleware are software components that handle HTTP **requests** and **responses**.

Request Pipeline Example

```
Request
   ↓
[Middleware 1]
   ↓
[Middleware 2]
   ↓
[Middleware N]
   ↓
Response
```

Example in Code:

```
app.UseAuthentication();   // Middleware for auth
app.UseAuthorization();    // Middleware for role checks
app.UseRouting();         // Middleware to route requests
```

Swagger – API Documentation

Add to services:

```
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
```

Use in app:

```
app.UseSwagger();
app.UseSwaggerUI();
```

What Happens If You Do This?

```
app.Run(async context ⇒
{
    await context.Response.WriteAsync("Only this response visible");
});
```

- All routes return **only** the above message.
- Swagger UI is not visible anymore.
- Still part of the pipeline, but execution stops after `Run()`.

Development Environment

Check environment with:

```
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

If You Change Environment in `launchSettings.json`

```
"profiles": {
  "YourApp": {
    "environmentVariables": {
      "ASPNETCORE_ENVIRONMENT": "Production"
    }
  }
}
```

Then, the app **won’t show** Swagger (it's only for Development by default).

Summary

Concept	Description
ASP.NET Core	Framework for web/cloud/mobile/IoT apps
MVC	Model-View-Controller approach
Minimal API	Lightweight, fast setup
Top-Level Program	No `Main()` , namespaces, just `Program.cs`
Middleware	Request/Response handlers in pipeline
Swagger	API documentation UI tool
Environments	Development, Staging, Production

Model-View-Controller (MVC) in ASP.NET Core

What is MVC?

MVC is a **software architectural pattern** used to separate the application's data, user interface, and control logic into **3 components**:

M-Tier Architecture

Tier	Role	Code/Example
Model	Represents data and business logic	Classes, DB logic
View	Represents UI - HTML, CSS, Razor	`View.cshtml` files
Controller	Coordinates between Model and View	Controller classes (C# files)

Diagram Description:

```
User → Controller → Model ↔ Database
       ↓
    View (HTML)
```

Resource Representation

Web APIs in MVC often return **JSON** format:

```
{
  "id": 1,
  "name": "Laptop"
}
```

Components in MVC

Controller

- Sits between View and Model
- Handles user input and logic
- Returns response (HTML, JSON, etc.)

- All controller classes:
    - End with `Controller`
    - Inherit from `ControllerBase` or `Controller`

```
public class ProductsController : ControllerBase
{
   [HttpGet]
   public IActionResult GetProduct() => Ok(new { Id = 1, Name = "Pen" });
}
```

Add Controller in Program.cs

```
builder.Services.AddControllers();
app.MapControllers(); // Enables attribute-based routing
```

IActionResult and Common Return Types

Return Type	Description	Code Example
`Ok()`	Returns 200 OK	`return Ok("Success");`
`NotFound()`	Returns 404	`return NotFound("Missing");`
`BadRequest()`	Returns 400	`return BadRequest("Invalid");`
`Created()`	Returns 201 Created	`return Created(...);`
`NoContent()`	Returns 204	`return NoContent();`

Routing

Routing connects incoming URLs to controller actions.

Middleware in Program.cs:

```
app.UseRouting();      // Add routing middleware
app.UseEndpoints(endpoints => { endpoints.MapControllers(); }); // Map route
```

```
s
```

Attribute-Based Routing (Preferred for APIs)

You define routes using attributes directly above methods or classes.

Level	Attribute	URI Example	HTTP Method
Class	`[Route("api/products")]`	`/api/products`	-
Method	`[HttpGet]`	`/api/products`	GET
Method	`[HttpPost]`	`/api/products`	POST
Method	`[HttpGet("{id}")]`	`/api/products/1`	GET

Example:

```
[Route("api/[controller]")]
[ApiController]
public class ProductsController : ControllerBase
{
    [HttpGet]
    public IActionResult GetAll() => Ok(new[] { "Pen", "Book" });

    [HttpGet("{id}")]
    public IActionResult GetById(int id) => Ok("Product " + id);
}
```

Common HTTP Status Codes

Code	Meaning	Description
200	OK	Success
201	Created	Resource created
204	No Content	Success, no body
400	Bad Request	Client-side error

Code	Meaning	Description
404	Not Found	Resource not found
500	Internal Error	Server crashed or failed

IEnumerable & ICollection Return Example

```
[HttpGet]
public IEnumerable<string> GetItems()
{
    return new List<string> { "Apple", "Banana" };
}
```

Use `ICollection` when modifying items like add/remove.

Content Negotiation

Content Negotiation = Selecting best response format based on `Accept` header.

Postman Test:

Key	Value
Accept	application/json
Accept	text/plain

Related:

- **Output Formatter**: Chooses how to return response (JSON/XML)
- **Input Formatter**: Parses incoming request body (Content-Type)

File Handling

Make file visible in root:

- Right-click → Properties → "Copy always" in output

Read File in Controller:

```
var fileBytes = System.IO.File.ReadAllBytes("wwwroot/sample.txt");
return File(fileBytes, "text/plain", "sample.txt");
```

Method	Use
`File.ReadAllBytes(path)`	Reads file as byte array
`File.Exists(path)`	Checks if file exists
`File.ReadAllText(path)`	Reads as text

Full Program.cs Example

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllers(); // Adds controller support

var app = builder.Build();

// Configure middleware pipeline
app.UseRouting(); // Enables routing
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers(); // Maps attribute routes
});

// Enable Swagger if needed
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

```
app.Run();
```

Common Mistakes to Avoid

Mistake	Fix
404 Not Found	Check routes and `[Route]` attribute
500 Internal Server Error	Check null references, debug logs
Missing `MapControllers()`	Add `app.MapControllers()` in Program.cs
Wrong Accept Header	Use `application/json` in Postman

Model Binding Sources

Use binding attributes to tell ASP.NET Core where to take data from.

Attribute	Description	Example Endpoint
`[FromQuery]`	Query string parameters	`/api/items?name=foo`
`[FromRoute]`	Route URL segments	`/api/items/5`
`[FromBody]`	JSON body (POST/PATCH)	JSON payload
`[FromHeader]`	Custom HTTP header	`X-Correlation-ID: abc123`
`[FromForm]`	Form-data / file upload	`multipart/form-data`

Sample Controller:

```
[ApiController]
[Route("api/[controller]")]
public class ItemsController : ControllerBase
{
    [HttpGet("{id}")]
    public IActionResult Get([FromRoute] int id, [FromQuery] string filter) =>
        Ok(new { id, filter });
    [HttpPost]
    public IActionResult Create([FromBody] ItemDto dto) ⇒
        CreatedAtAction(nameof(Get), new { id = 1 }, dto);

    [HttpPost("form")]
    public IActionResult Upload([FromForm] string description, IFormFile file) ⇒
        Ok(new { description, file?.FileName });
}
```

(AlgoLesson, Dot Net Tutorials)

Using DTOs & Resource Methods

- Create separate **DTO classes** to decouple request/response from domain models.

```
public record ItemDto(string Name, decimal Price);
```

- Use **CreatedAtAction** to return newly created resources.

```
return CreatedAtAction(nameof(Get), new { id = newId }, dto);
```

Input Validation & ModelState

Use Data Annotations and check `ModelState`.

```
public record ItemDto([Required] string Name, [Range(0.01, double.MaxValue)] decimal Price);

[HttpPost]
public IActionResult Create([FromBody] ItemDto dto)
{
    if (!ModelState.IsValid)
```

```
        return BadRequest(ModelState);
    // create ...
    return Ok(dto);
}
```

- Provide **custom error messages** using `[Required(ErrorMessage="...")]`.
- Separation of concerns: validation + business logic kept apart.

(dotnetcurry.com)

PATCH Support (JSON Patch)

Partial updates using `PATCH` and `JsonPatchDocument<T>`.

```
[HttpPatch("{id}")]
public IActionResult Patch(int id, [FromBody] JsonPatchDocument<ItemDto>
patch)
{
    if (patch is null) return BadRequest();
    var existing = ... // load your model
    patch.ApplyTo(existing, ModelState);
    if (!ModelState.IsValid) return BadRequest(ModelState);
    // save!
    return Ok(existing);
}
```

- Use operations `replace`, `add`, `remove`, etc.
- Validate and apply in correct order: null check → load → apply → save.(Code Review Stack Exchange)

Example Patch Request (JSON):

```
[
  { "op": "replace", "path": "/Name", "value": "NewName" }
]
```

Test via Postman: select method PATCH, header `Content-Type: application/json-patch+json`.

DELETE Resource

```
[HttpDelete("{id}")]
public IActionResult Delete(int id)
{
    bool found = ...;
    if (!found) return NotFound();
    // delete logic
    return NoContent();
}
```

Use `NotFound()` or `NoContent()` to represent resource deletion state.

File Uploads & IFormFile Safety

```
public class UploadDto
{
    [Required]
    public IFormFile File { get; set; }
}

[HttpPost("upload")]
public async Task<IActionResult> Upload([FromForm] UploadDto dto)
{
    if (!ModelState.IsValid) return BadRequest(ModelState);
    if (dto.File.Length > 5 * 1024 * 1024)
        return BadRequest("Max 5 MB");

    var ext = Path.GetExtension(dto.File.FileName).ToLower();
    if (!new[] { ".jpg", ".png" }.Contains(ext))
        return BadRequest("Invalid file type");

    var path = Path.Combine("Uploads", Path.GetRandomFileName());
    await using var stream = System.IO.File.Create(path);
    await dto.File.CopyToAsync(stream);

    return Ok();
}
```

- Ensure directory has **no execute permissions**.
- Validate size and extensions manually or via custom validation attributes. (Microsoft Learn, Stack Overflow)

Summary

Feature	Key Methods / Attributes
Binding Source	`[FromQuery]`, `[FromRoute]`, `[FromBody]`, `[FromForm]`, `[FromHeader]`
Validation	Data Annotations, `ModelState`
Return Types	`Ok()`, `BadRequest()`, `NotFound()`, `CreatedAtAction()`, `NoContent()`
HTTP Methods	GET (route/query), POST (body/form), PATCH, DELETE
Patch Support	`JsonPatchDocument<T>`
File Upload	`IFormFile`, manual validation
Security	Validate input, deny execute perms, restrict content types