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

* **ASP.NET Core** is Microsoft’s modern, cross-platform framework for building web applications and RESTful APIs.
* A **Web API** (Application Programming Interface) exposes data and operations through HTTP endpoints (GET, POST, PUT, DELETE) that other apps can consume.
* APIs are stateless and mainly return data in **JSON** format.

**Why use ASP.NET Core Web API?**

* Cross-platform (Windows, Linux, macOS)
* Built-in Dependency Injection
* High performance with Kestrel server
* Easy integration with Entity Framework Core and SQL Server
* Excellent for React / Angular / Mobile backends

**2. Project Structure Overview**

When you create a new API project in Visual Studio or VS Code (dotnet new webapi), you get these folders:

|  |  |
| --- | --- |
| **Folder / File** | **Purpose** |
| **Controllers/** | Contains controller classes that define API endpoints |
| **Models/** | Classes representing data structures or database entities |
| **Program.cs** | Entry point — configures middleware, services, routing |
| **appsettings.json** | Stores configuration (connection strings, logging) |
| **Properties/** | Contains launchSettings.json (port, profiles) |
| **wwwroot/** | Static files (optional for APIs) |

**3. Understanding the MVC Pattern in APIs**

Although APIs don’t use Views, they still follow **Model–Controller** logic:

* **Model** – Represents the data (e.g., Student, Course)
* **Controller** – Handles requests and responses
* **View** – Not used here; the client (React App or Postman) acts as the “view”

**4. How ASP.NET Core Handles a Request**

1. Client sends HTTP request (GET /api/students).
2. ASP.NET Core’s **Routing Middleware** matches the URL to a controller/action.
3. The **Controller** executes logic, interacts with a database (via EF Core).
4. The **Action Method** returns a JSON response.

**5. Core Concepts and Terms**

|  |  |
| --- | --- |
| **Term** | **Meaning** |
| **Controller** | Class ending with Controller, e.g., StudentsController, defines endpoints |
| **Action Method** | Method inside a controller that handles an HTTP verb |
| **Route** | URL pattern (api/[controller]) that maps to actions |
| **[HttpGet]/[HttpPost]/[HttpPut]/[HttpDelete]** | Attributes that define which HTTP method triggers the action |
| **IActionResult / ActionResult** | Return types representing HTTP responses |
| **Dependency Injection (DI)** | Technique to provide services (like DbContext) automatically |
| **Middleware** | Components that process requests (Routing, CORS, Authentication, etc.) |

**6. Creating Your First Web API**

**Step 1 – Create the Project**

dotnet new webapi -n StudentApi

cd StudentApi

dotnet run

Default endpoint: https://localhost:5001/weatherforecast

**Step 2 – Create a Model**

namespace StudentApi.Models

{

public class Student

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public string Grade { get; set; }

}

}

**Step 3 – Add a Controller**

using Microsoft.AspNetCore.Mvc;

using StudentApi.Models;

using System.Collections.Generic;

namespace StudentApi.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class StudentsController : ControllerBase

{

private static List<Student> students = new List<Student>();

[HttpGet]

public IActionResult GetAllStudents()

{

return Ok(students);

}

[HttpPost]

public IActionResult AddStudent(Student s)

{

students.Add(s);

return Ok(new { message = "Student added successfully" });

}

}

}

**Step 4 – Run and Test**

dotnet run

Open **Postman** → GET https://localhost:5001/api/students → returns empty list.  
Then send a POST request with JSON:

{

"id": 1,

"name": "Udaya",

"age": 22,

"grade": "A"

}

**7. Understanding appsettings.json**

Example:

{

"ConnectionStrings": {

"DefaultConnection": "Server=.;Database=StudentDB;Trusted\_Connection=True;"

},

"Logging": { "LogLevel": { "Default": "Information" } },

"AllowedHosts": "\*"

}

Later you’ll use this connection string in Entity Framework Core to link SQL Server.

**8. Middleware & Request Pipeline**

Middleware runs sequentially for every request.  
Common ones:

1. **UseRouting()** – matches URL to routes
2. **UseCors()** – allows frontend to access backend
3. **UseAuthorization()** – manages access control
4. **MapControllers()** – executes controller actions

The order matters — it defines how requests flow through the application.

**Testing Tools**

* **Postman** – API testing (send GET, POST, PUT, DELETE requests)
* **Swagger (built-in)** – Visual API documentation UI  
  Launch: https://localhost:5001/swagger/index.html

**Key Takeaways**

|  |  |
| --- | --- |
| **Concept** | **Summary** |
| ASP.NET Core | Framework to build REST APIs |
| Controller | Defines endpoints and logic |
| Model | Represents data structure |
| Routing | Maps URLs to controllers |
| Dependency Injection | Automatically provides services |
| Swagger & Postman | Tools for testing APIs |

**Mini Task for Day 1**

Build a simple Web API with a StudentController supporting:

* GET – Fetch all students
* POST – Add a new student  
  Return JSON responses and verify using Swagger/Postman.

**Snapshots:**

A screenshot of a computer

AI-generated content may be incorrect.

Code : Program.cs

A screenshot of a computer

AI-generated content may be incorrect.

Code : Student.cs

A screenshot of a computer program

AI-generated content may be incorrect.

Code : StudentController.cs

A screenshot of a computer

AI-generated content may be incorrect.

Output : Swagger UI

A screenshot of a computer

AI-generated content may be incorrect.

Output : GET ( All data )

A screenshot of a computer

AI-generated content may be incorrect.

Output : POST

A screenshot of a computer

AI-generated content may be incorrect.

Output : GET (Specific Data)

A screenshot of a computer

AI-generated content may be incorrect.

Output : DELETE