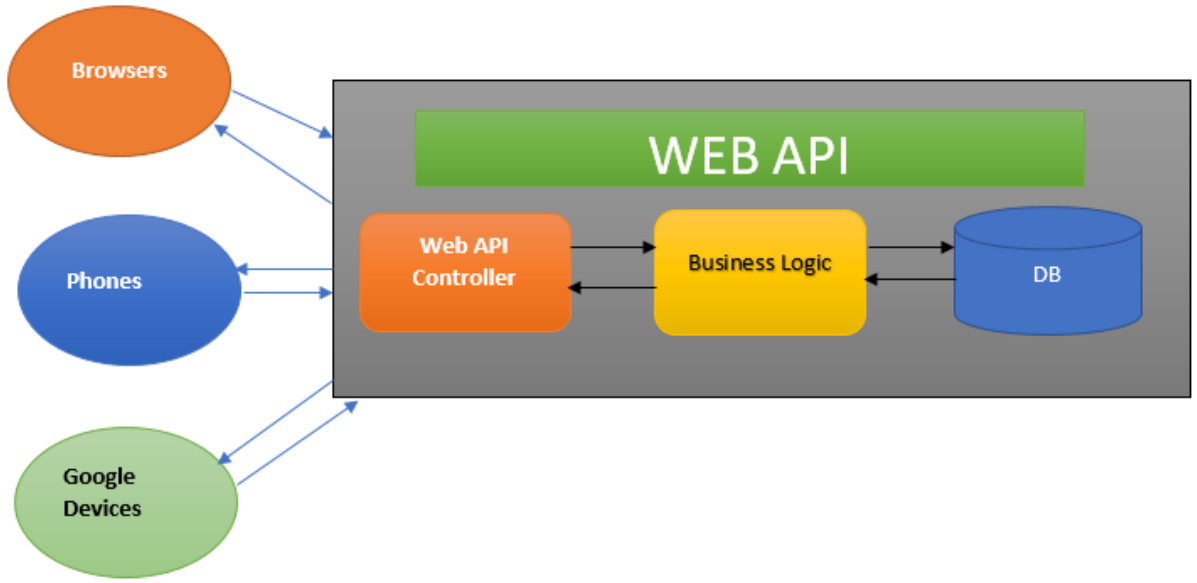
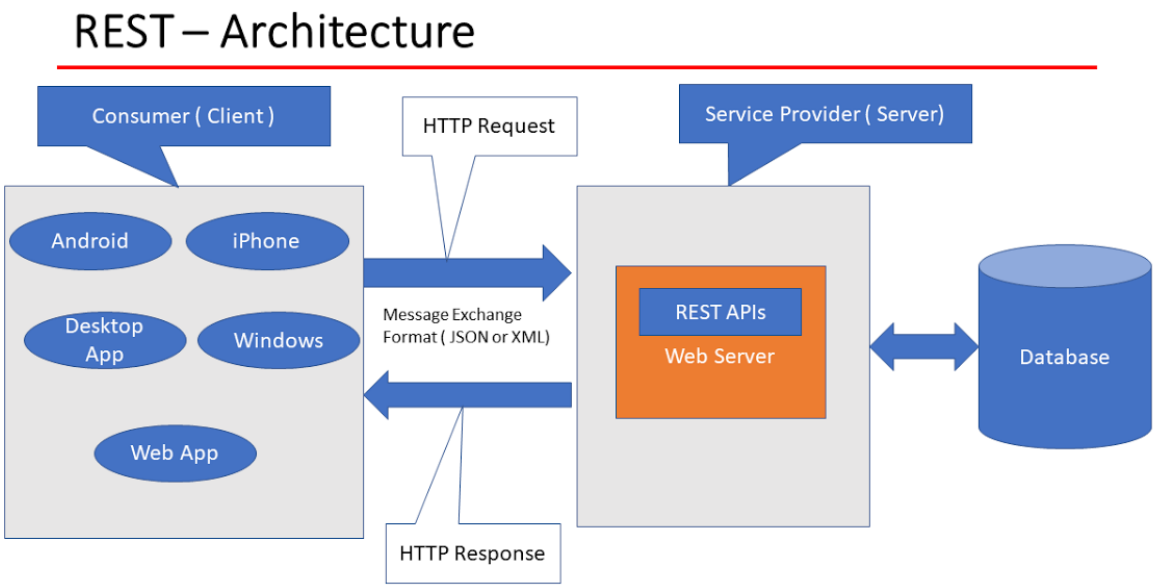
✅ ASP.NET Core REST API — Basics Explained (Instructor Style)





🧩 1. What is a REST API?

REST stands for **Representational State Transfer** — a set of rules/constraints for building scalable web services.

A **REST API**:

- Works over **HTTP/HTTPS**
- Exchanges data usually in **JSON format**
- Uses **HTTP verbs** (GET, POST, PUT, DELETE)
- Is **stateless** — server does not store client session
- Returns appropriate **HTTP status codes**

Examples:

- GET /api/products → returns all products
- POST /api/products → creates a new product

🧩 2. What is ASP.NET Core REST API?

ASP.NET Core is a **cross-platform**, **open-source**, high-performance framework for building:

- Web APIs
- Web applications
- Microservices
- Real-time apps (SignalR)

A **Web API in ASP.NET Core** is built using:

- **Controllers**
- **Routing**
- **Model binding**
- **Dependency Injection (DI)**
- **Middleware pipeline**

🏗️ 3. ASP.NET Core API Project Structure

When you create *ASP.NET Core Web API*, it generates:

Controllers/

WeatherForecastController.cs

Program.cs

appsettings.json

Properties/

🔹 **Important Files**

File	Purpose
Program.cs	Startup file; configures services & middleware
appsettings.json	Configuration values (DB string, JWT, Logging)
Controllers folder	Contains API endpoints
Models folder (you create)	DTOs / Entities / ViewModels
Services & Repositories (optional layers)	Business logic/data access

---

🧩 **4. Routing in ASP.NET Core API**

Routing decides **which URL maps to which action method**.

Example:

```
[Route("api/[controller]")]
[ApiController]
public class ProductsController : ControllerBase
{
    [HttpGet]
    public IActionResult GetAll() => Ok("All products");

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

How routes work:

- GET /api/products → GetAll()
- GET /api/products/10 → GetById(10)

🧩 5. HTTP Methods (CRUD)

Operation	HTTP Verb	Example
Read all	GET	/api/products
Read one	GET	/api/products/1
Create	POST	/api/products
Update	PUT	/api/products/1
Delete	DELETE	/api/products/1

ASP.NET Core maps these using attributes:

[HttpPost]

[HttpPut]

[HttpDelete]

🧩 6. Action Results

Actions return:

- **Data + Status code**
- Example:

```
return Ok(product);          // 200
return NotFound();           // 404
return BadRequest("Invalid");  // 400
return Created(...);         // 201
```

🧩 7. Model Binding & Validation

User input automatically binds to C# objects.

Example model:

```
public class ProductDto
{
   [Required]
   public string Name { get; set; }

   [Range(1,100000)]
   public decimal Price { get; set; }
}
```

Controller:

```
[HttpPost]
public IActionResult Create(ProductDto dto)
{
   if (!ModelState.IsValid)
      return BadRequest(ModelState);

   return Ok(dto);
}
```

🧩 8. Dependency Injection (DI)

ASP.NET Core has **built-in DI**.

Example:

```
builder.Services.AddTransient<IProductService, ProductService>();
```

Controller:

```
public class ProductsController : ControllerBase
{
    private readonly IProductService _service;

    public ProductsController(IProductService service)
    {
        _service = service;
    }
}
```

🧩 9. Middleware Pipeline

Every request flows through middleware:

- Authentication
- Authorization
- Routing
- Exception Handling
- Logging

Example in Program.cs:

```
app.UseHttpsRedirection();
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();
```

🧩 10. Simple Working Example

Program.cs

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

```
builder.Services.AddControllers();

var app = builder.Build();

app.MapControllers();

app.Run();
```

ProductsController.cs

```
[ApiController]
[Route("api/[controller]")]
public class ProductsController : ControllerBase
{
    private static readonly List<string> products = new() { "Pen", "Book" };

    [HttpGet]
    public IActionResult GetAll() => Ok(products);

    [HttpPost]
    public IActionResult Add(string name)
    {
        products.Add(name);
        return Created("", name);
    }
}
```

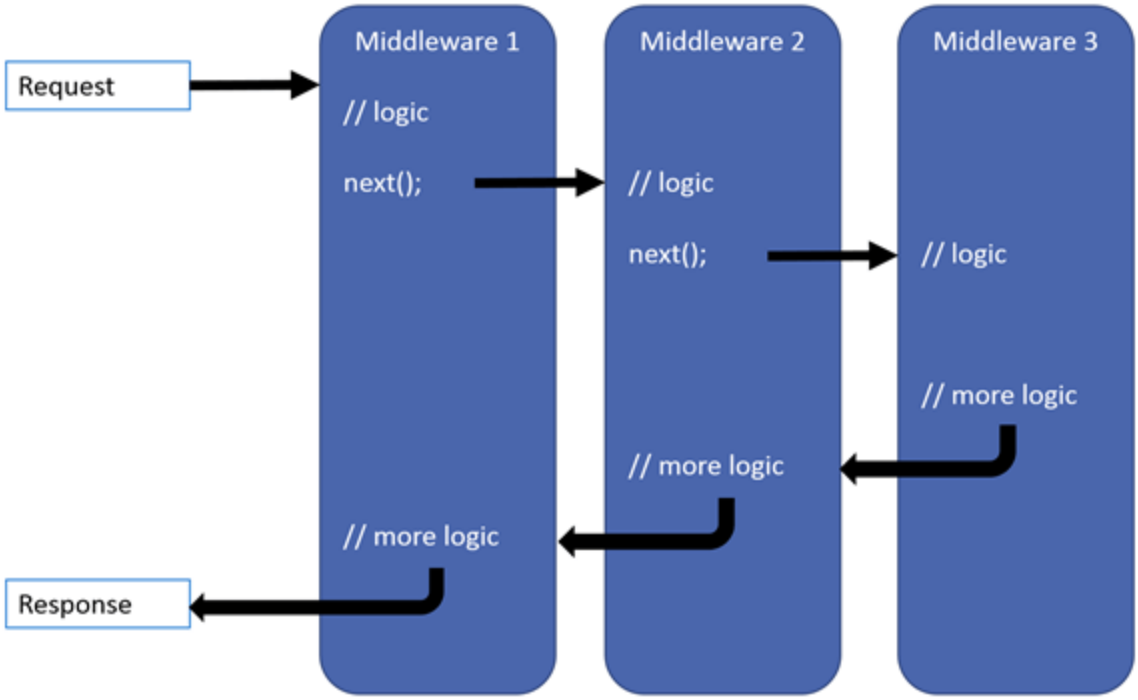
🧩 11. When to Use Web API?

Use ASP.NET Core Web API when building:

- Mobile app backend (Android/iOS)
- Angular/React frontend backend
- Microservices
- Third-party integrations
- IoT devices
- Enterprise applications

🎯 Summary — ASP.NET Core REST API Essentials

Concept	Description
REST	Architecture style using HTTP
Web API	Service that exposes data over HTTP
Controllers	Handle API requests
Routing	Maps URL to action
DI	Loosely-coupled design
Model Binding	Convert JSON → C# object
Validation	Ensures correct data
Middleware	Pipeline for processing


Request
Middleware 1
// logic
next();
// more logic
Response
Middleware 2
// logic
next();
// more logic
Middleware 3
// logic
// more logic