Overview of Web API in C#

# 🚀 Introduction to Web API

A Web API (Application Programming Interface) in C# typically refers to the creation of HTTP-based services using ASP.NET Core Web API. These APIs allow client applications to communicate with server-side resources over HTTP using RESTful principles.

# 🔄 How Web API Works (Request Pipeline)

ASP.NET Core uses a middleware-based request pipeline:  
1. HTTP Request enters the Kestrel server.  
2. Middleware components process the request.  
3. Routing middleware maps to appropriate controller/action.  
4. Controller processes the logic and returns a response.  
5. Middleware handles the response back to client.

# 📂 Core Components of Web API

• Controllers: Handle HTTP requests and return responses.

• Models: Represent data and define the structure of request/response bodies.

• Routing: Maps URLs to controller actions.

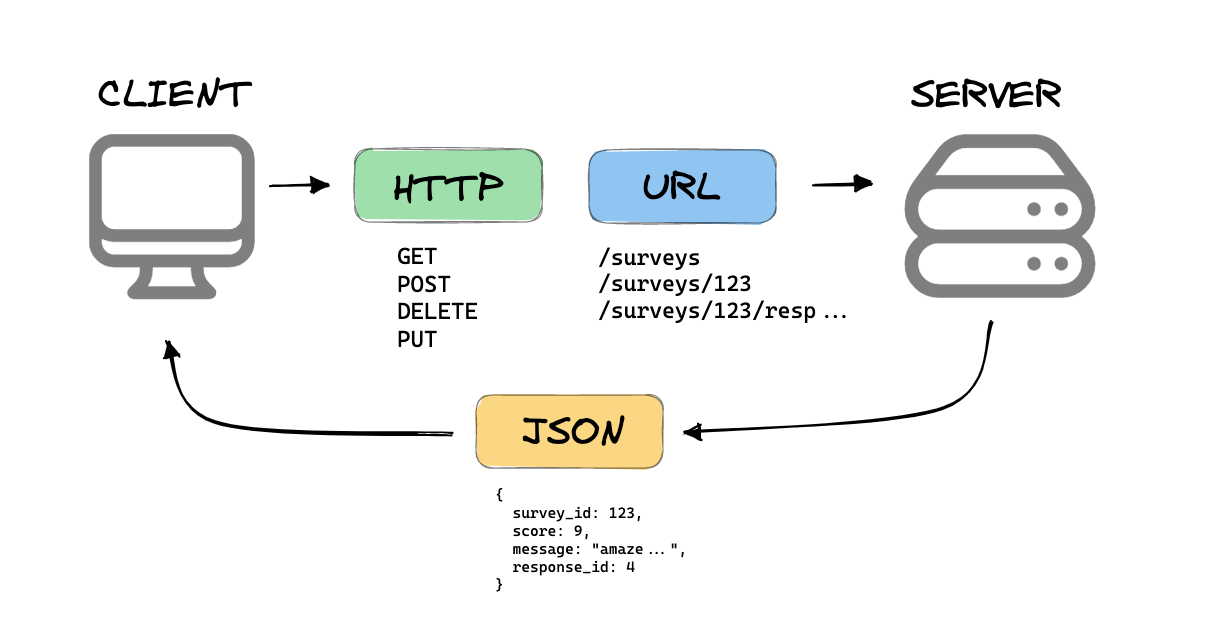
• Services: Encapsulate business logic; used via Dependency Injection.

• Middleware: Software inserted in the pipeline to handle requests/responses globally.

# 🧭 Routing in Web API

• Conventional Routing: Defined in Startup.cs as a pattern.  
• Attribute Routing: Decorate controller methods with route attributes.

[HttpGet("{id}")]  
public IActionResult GetProduct(int id) { ... }



# 🛠️ CRUD Example - ProductsController

[ApiController]  
[Route("api/[controller]")]  
public class ProductsController : ControllerBase  
{  
 private readonly IProductService \_service;  
  
 public ProductsController(IProductService service) => \_service = service;  
  
 [HttpGet] public IActionResult Get() => Ok(\_service.GetAll());  
 [HttpGet("{id}")] public IActionResult Get(int id) => Ok(\_service.Get(id));  
 [HttpPost] public IActionResult Create(Product p) => Ok(\_service.Create(p));  
 [HttpPut("{id}")] public IActionResult Update(int id, Product p) => Ok(\_service.Update(id, p));  
 [HttpDelete("{id}")] public IActionResult Delete(int id) => Ok(\_service.Delete(id));  
}

# 📦 Serialization & Content Negotiation

ASP.NET Core Web API supports automatic serialization of data (usually JSON). It uses formatters to determine how to return responses based on Accept headers sent by the client.  
Popular serializers: System.Text.Json (default), Newtonsoft.Json (optional).

# 🔗 Dependency Injection (DI)

ASP.NET Core has built-in DI. Services are registered in Startup.cs (or Program.cs in .NET 6+) and injected into controllers.  
Example: services.AddScoped<IProductService, ProductService>();

# 🧬 Versioning Web APIs

To support multiple versions:  
• Use URL path (api/v1/products)  
• Use query string (?api-version=1.0)  
• Use custom headers.

# 🔐 Securing Web API (Intro to JWT)

JWT (JSON Web Tokens) are used for securing APIs. After login, the server issues a signed token. The client sends it in the Authorization header for subsequent requests.  
Use middleware like: app.UseAuthentication(); app.UseAuthorization();

# 🧪 Testing & Debugging APIs

• Tools: Postman, Swagger UI, curl  
• Use unit tests with xUnit and integration tests with TestServer  
• Debug using breakpoints and logs (ILogger)

# 🚀 Deployment Options

ASP.NET Core Web API can be deployed on:  
• IIS (Internet Information Services)  
• Linux (via Kestrel and Nginx)  
• Azure App Service  
• Docker containers