Josh Morales

Page Init Solutions

https://pageinit.net
https://selarom.net
https://rgvdevs.net

Cyber RGV | .NET | AI

Creating AI Apps with

.NET Minimal APIs and MAF

20 years of evolution → one elegant endpoint

Agenda

01 .NET API Evolution (2002 → Today)

02 Minimal APIs

03 MAF Refresher

04 Demo Teaser

05 April 14 — Full Deep Dive

Why APIs Matter

- APIs are the backbone of modern applications
- .NET has evolved dramatically over 20+ years to reduce developer friction
- Minimal APIs = the most elegant version of .NET API development yet
- Less ceremony → faster AI feature delivery

Era 1

Web Forms + ASMX

.NET 1.0 · 2002

Maximum ceremony. Minimum REST.

Era 1 · .NET 1.0 · 2002

Web Forms + ASMX Overview

- Page-centric, event-driven model — .aspx files everywhere
- SOAP-only APIs via .asmx — XML in, XML out, no JSON whatsoever
- Heavy markup: .aspx, .asmx, code-behind (.cs), designer files
- Web.config required for nearly everything including SOAP protocols
- No REST support — HTTP was just the transport

Era 1 · .NET 1.0 · 2002

Web Forms + ASMX — Code Example

HelloService.asmx

HelloService.asmx

```
<%@ WebService Language="C#" CodeBehind="HelloService.asmx.cs"
Class="MyApp.HelloService" %>
```

HelloService.asmx.cs

HelloService.asmx.cs

```
using System.Web.Services;

namespace MyApp
{
    [WebService(Namespace = "http://mycompany.com/services")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    public class HelloService : WebService
    {
        [WebMethod]
        public string Hello(string name)
        {
            return $"Hello, {name}";
        }
    }
}
```

Web.config

Web.config

```
<configuration>
  <system.web>
    <webServices>
      <protocols>
        <add name="HttpSoap" />
        <add name="HttpPost" />
        <add name="HttpGet" />
      </protocols>
    </webServices>
  </system.web>
</configuration>
```

Project Structure

Typical Project Structure

```
/Default.aspx
/Default.aspx.cs
/Default.aspx.designer.cs
/HelloService.asmx
/HelloService.asmx.cs
/Web.config
```

Era 2

WCF

Windows Communication Foundation · .NET 3.0 · 2006

Powerful. Flexible. Verbose.

Era 2 · .NET 3.0 · 2006

WCF Overview

- Unified communication stack: SOAP, TCP, MSMQ, Named Pipes
- Enterprise-grade: security, transactions, reliable messaging
- Extremely verbose — config files measured in scroll length
- REST support bolted on later, never felt natural
- Steep learning curve; required deep specialist knowledge

Era 2 · .NET 3.0 · 2006

WCF — Code Example

Service Contract (interface)

Service Contract

```
[ServiceContract]
public interface IHelloService
{
    [OperationContract]
    string SayHello(string name);
}
```

Service Implementation

Implementation

```
public class HelloService : IHelloService
{
    public string SayHello(string name) => $"Hello, {name}";
}
```

Self-Hosting (Program.cs)

Hosting

```
var baseAddress = new Uri("http://localhost:8080/HelloService");
var host = new ServiceHost(typeof(HelloService), baseAddress);

host.AddServiceEndpoint(typeof(IHelloService), new WSHttpBinding(), "");
host.Description.Behaviors.Add(new ServiceMetadataBehavior { HttpGetEnabled = true });

host.Open();
Console.ReadLine();
```

app.config — the infamous part

app.config

```
<system.serviceModel>
  <services>
    <service name="HelloService">
      <endpoint address=""
                binding="wsHttpBinding"
                contract="IHelloService" />
      <endpoint address="mex"
                binding="mexHttpBinding"
                contract="IMetadataExchange" />
      <host>
        <baseAddresses>
          <add baseAddress="http://localhost:8080/HelloService" />
        </baseAddresses>
      </host>
    </service>
  </services>
</system.serviceModel>
```

Era 3

ASP.NET MVC

.NET 3.5 / 4.0 · 2009

Routing arrived. Controllers everywhere. JSON emerged, slowly.

Era 3 · .NET 3.5/4.0 · 2009

ASP.NET MVC Overview

- Clean MVC pattern: Models, Views, Controllers
- URL routing replaces page-centric navigation
- Testable controllers — a major step forward
- JSON support emerged but views were still first-class citizens
- Still not API-first — Global.asax, RouteConfig, and scaffolding required

Era 3 · .NET 3.5/4.0 · 2009

ASP.NET MVC — Code Example

Global.asax

Global.asax

```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();
        RouteConfig.RegisterRoutes(RouteTable.Routes);
    }
}
```

RouteConfig.cs

RouteConfig.cs

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Hello", id =
UrlParameter.Optional }
        );
    }
}
```

HomeController.cs

HomeController.cs

```
public class HomeController : Controller
{
    public ActionResult Hello(string name)
    {
        var model = new HelloModel { Name = name };
        return View(model);
    }
}
```

Hello.cshtml

Hello.cshtml

```
@model HelloModel
<h1>Hello, @Model.Name</h1>
```

Era 4

ASP.NET Web API

.NET 4.0 / 4.5 · 2012

True REST at last. JSON-first. Still controller-heavy.

Era 4 · .NET 4.0/4.5 · 2012

ASP.NET Web API Overview

- A true REST framework — finally purpose-built for HTTP
- JSON by default, attribute routing, clean HTTP verb semantics
- Filters, model binding, content negotiation built-in
- Still controller-heavy — ApiController inheritance required
- Startup scattered: Global.asax + WebApiConfig + Web.config

Era 4 · .NET 4.0/4.5 · 2012

ASP.NET Web API — Code Example

Global.asax

Global.asax

```
public class WebApiApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        GlobalConfiguration.Configure(WebApiConfig.Register);
    }
}
```

WebApiConfig.cs

HelloController.cs

HelloController.cs

```
public class HelloController : ApiController
{
    [HttpGet]
    public IHttpActionResult Get(string name)
    {
        return Ok(new { Message = $"Hello, {name}" });
    }
}
```

Web.config (partial)

Web.config

```
<system.webServer>
  <handlers>
    <add name="ExtensionlessUrlHandler-Integrated-4.0"
         path="*."
         verb="*"
         type="System.Web.Handlers.TransferRequestHandler"
         preCondition="integratedMode,runtimeVersionv4.0" />
  </handlers>
</system.webServer>
```

Era 5

ASP.NET Core

.NET Core 1.0 · 2016

Cross-platform. Unified. Modern pipeline. Still needs Startup.cs.

Era 5 · .NET Core 1.0 · 2016

ASP.NET Core Overview

- Cross-platform — Linux, macOS, Windows from day one
- Unified MVC + Web API into a single coherent framework
- Middleware pipeline, built-in DI, modular by design
- Dramatically faster than classic ASP.NET
- Controllers still required — Startup.cs still required

Era 5 · .NET Core 1.0 · 2016

ASP.NET Core — Code Example

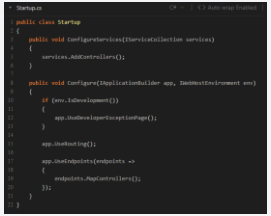
Program.cs

Program.cs

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
}
```

Startup.cs



HelloController.cs

HelloController.cs

```
[ApiController]
[Route("api/[controller]")]
public class HelloController : ControllerBase
{
    [HttpGet("{name}")]
    public IActionResult Get(string name)
    {
        return Ok(new { Message = $"Hello, {name}" });
    }
}
```

appsettings.json

appsettings.json

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

Era 6

Minimal APIs

.NET 6 · 2021 → Present

No controllers. No Startup.cs. No ceremony. Just code.

Era 6 · .NET 6 · 2021 → Present

Minimal APIs Overview

- No controllers — routes defined inline with MapGet / MapPost
- No Startup.cs — Program.cs is the entire application
- No config files — just appsettings.json, and even that's optional
- Single-file APIs possible in under 10 lines of code
- Perfect for microservices, serverless, and AI model endpoints

Era 6 · .NET 6 · 2021 → Present

Minimal API — Code Example

Basic: The Whole API (3 lines)

Complete API

```
var app = WebApplication.Create(args);

app.MapGet("/hello/{name}", (string name) => $"Hello, {name}");

app.Run();
```

Advanced: DI + Auth + Rate Limiting + OpenAPI

Advanced Endpoint

```
app.MapPost("/summarize",
    [Authorize] async (SummaryRequest req, IAIService ai) =>
{
    return await ai.SummarizeAsync(req.Text);
})
.RequireRateLimiting("standard")
.Produces<SummaryResponse>(200)
.WithOpenApi();
```

Why Minimal APIs Shine for AI

Stateless by nature

AI model calls are stateless — perfect match for Minimal API handlers

Small surface area

One route, one concern — no controller inheritance baggage

Fast to build

Idea to working AI endpoint in minutes, not hours

Easy to deploy

Single file — runs in a container, Lambda, or VPS without ceremony

Fluent chaining

Auth, rate limiting, OpenAPI, CORS — all via clean method chaining

Model endpoint ready

Thin wrapper over your AI abstraction: inputs in, outputs out

MAF — Model Abstraction Framework

Encapsulates AI logic so your API layer stays thin.

Encapsulates AI logic

Prompt engineering, model config, and context management live inside MAF — not your endpoint

Standardizes I/O

Typed request/response contracts — your API layer doesn't care which model is running

Swap models easily

Switch from GPT-4o to Claude to a local model behind a single interface change

Keeps endpoints thin

MapPost receives input → calls IModelRunner → returns result. Done.

Minimal API + MAF — The Payoff

Program.cs — The Complete AI Endpoint (MapPost + IModelRunner)

Using MAF

```
app.MapPost("/chat", async (ChatRequest req, IModelRunner model) =>
{
    var result = await model.RunAsync(req);
    return Results.Ok(result);
});
```

🚀 Demo Time

Minimal API + MAF

~40	0	0	∞
Lines of Code	Controllers	Startup Files	Flexibility

April 14 — we go deep. Today, we see it run.

"Minimal" Doesn't Mean Limited

Minimal APIs support nearly everything Web API does — with far less ceremony.

- ✓ Authentication
- ✓ Authorization
- ✓ Rate Limiting
- ✓ Validation
- ✓ Caching
- ✓ OpenAPI / Swagger
- ✓ CORS
- ✓ API Versioning
- ✓ Response Streaming
- ✓ Background Tasks
- ✓ Model Routing
- ✓ Health Checks

What's Next

RGV Devs .NET

- .NET meetups in the Rio Grande Valley
- Talks, demos, and community projects
- All skill levels welcome
- Find us at https://rgvdevs.net

April 14 – Deep Dive

- Complete architecture review
- AI Prompts
- Adding new features
- https://qr.pageinit.net/apimaf

Thank You

Questions + Discussion

Site: rgvdevs.net

Hosted By: RGV Devs .NET | Cyber RGV

Next Event: April 14 — Minimal APIs + MAF Deep Dive

https://qr.pageinit.net/apimaf

Minimal APIs. Maximum impact.