# Building an AI Brand Engine with .NET and MAF

*Josh Morales · RGVDevs.NET · Live Coding Session*

# Where We Left Off

- **`.NET Minimal APIs`**
  All routes in Program.cs. No controllers.
- **`Microsoft.Extensions.AI`**
  IChatClient abstraction — swap models with config.
- **`BrandService`**
  Markdown file → system prompt prefix on every AI call.
- **`5 content endpoints`**
  /social · /blog-outline · /video-idea · /image-prompt · /daily
- **`Blazor Demo app`**
  SharpForge.Demo — calls the API, renders generated content.

***Today: Review the project in detail, then add 4 production-grade extensions to this running app.***

https://qr.pageinit.net/rgvdevsgit1

# Deep Dive: Code Walkthrough

Every class, every prompt — under the hood

# 1 OpenAPI + Scalar

*2 packages · 3 lines · full interactive API explorer*

**Install**

```
dotnet add package Microsoft.AspNetCore.OpenApi
dotnet add package Scalar.AspNetCore
```

**Register + Map**

```
builder.Services.AddOpenApi();
app.MapOpenApi();
app.MapScalarApiReference();
```

**What you get at /scalar/v1**

- Every endpoint documented automatically
- Request / response schemas from your records
- Interactive Try It — no Postman needed
- .WithName() and .WithDescription() already populated
- OpenAPI JSON at /openapi/v1.json for tooling

# 2 API Key Authentication

*Protect /content/* · /status stays public · Scalar shows the lock*

New file + Register

```
Auth/ApiKeyAuthHandler.cs

builder.Services.AddAuthentication(SchemeName)
.AddScheme<Options, Handler>(...);
builder.Services.AddAuthorization();
```

Middleware + Route group

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

var content = app.MapGroup("/content")
  .RequireAuthorization();
```

appsettings.json

```
"Auth": { "ApiKey": "sharpforge-demo-key-2026" }
```

**Demo: hit /content/social → 401. Add X-Api-Key header → 200.**

# Challenge: Fix the Blazor App

*~4 minutes · try it yourself first*

**The problem**

SharpForge.Demo is now getting 401 Unauthorized.

Auth broke the client. Your job: fix it without changing the API.

**The fix**

```
// SharpForge.Demo/Program.cs
var apiKey = config["Api:ApiKey"];
if (!string.IsNullOrEmpty(apiKey))
  client.DefaultRequestHeaders.
Add("X-Api-Key", apiKey);
```

**appsettings.json (Demo)**

```
"Api": {
  "BaseUrl": "https://localhost:44338",
  "ApiKey": "sharpforge-demo-key-2026"
}
```

**The takeaway**

*Clean architecture means a breaking change in one layer is a small, obvious fix in the next. Config + one header.*

# 3 Rate Limiting

*Protect your AI budget · Built into .NET · Zero packages*

**Register**

```
builder.Services.AddRateLimiter(options =>
{
  options.RejectionStatusCode = 429;
  options.AddFixedWindowLimiter("ai", limiter =>
  {
    limiter.PermitLimit = 3;
    limiter.Window = TimeSpan.FromMinutes(1);
    limiter.QueueLimit = 0;
  });
});
```

**Demo: run /content/social 3 times → 429 Too Many Requests on hit 4.**

**Middleware + Route group**

```
app.UseRateLimiter();

var content = app.MapGroup("/content")
  .RequireAuthorization()
  .RequireRateLimiting("ai");
```

**Policies available**

**Fixed Window ← we're using this**
Sliding Window
Token Bucket
Concurrency

# 4 Generation Timing Filter

*Cross-cutting observability · No endpoint changes · IEndpointFilter*

**New file**

Filters/GenerationTimingFilter.cs

```
public class GenerationTimingFilter : IEndpointFilter
{
  public async ValueTask<object?> InvokeAsync(
    EndpointFilterInvocationContext ctx,
    EndpointFilterDelegate next)  {
    var sw = Stopwatch.StartNew();
    var result = await next(ctx);
    sw.Stop();
    ctx.HttpContext.Response.Headers["X-Generation-
Time-Ms"] = sw.ElapsedMilliseconds.ToString();
    return result;
  }
}
```

**Apply to route group**

```
var content = app.MapGroup("/content")
  .RequireAuthorization()
  .RequireRateLimiting("ai")
  .AddEndpointFilter<GenerationTimingFilter>();
```

**The full route group — final state**

*3 cross-cutting concerns*
*applied to 5 endpoints*
*in 3 chained calls.*

# Everything Followed the Same Three Steps.

**1** **Register a service**
*Teach the container about the feature*

```
builder.Services.Add...()
```

**2** **Add middleware**
*Wire it into the request pipeline*

```
app.Use...()
```

**3** **Apply to the route group**
*Scope it precisely — one line, five endpoints*

```
.RequireX() / .AddFilter()
```

***No YAML. No cloud console. No vendor SDKs. Just C# and dotnet run. That's .NET.***

# What Else Can You Do?

*Everything below is built into .NET or is one NuGet package away. Same pattern.*

| Built In | One Package | AI-Specific |
|---|---|---|
| Output Caching | Serilog (structured logs) | Streaming responses |
| Health Checks | Polly (resilience) | Tool / function calling |
| Request Timeouts | Hangfire / Quartz.NET | Multiple agents |
| Endpoint Filters | FluentValidation | Content moderation |
| Exception Middleware | Carter (modules) | Semantic Kernel |
| Named CORS Policies | Scalar custom themes | EF Core multi-brand DB |

*Full list with docs links: phase-3-possibilities.md in the repo*

# What would YOU build next?

*The floor is yours — bring an idea and let's scope it.*

| | |
|---|---|
| `Repo` | *github.com/selaromdotnet/RGVDevs-SharpForge* |
| `phase-3-possibilities.md` | *30+ expansion ideas with docs links* |
| `RGV Devs .NET` | *Next session — we build what you suggest!* |