3.

Step-by-step guide for creating a **.NET 9** Web API (instead of .NET 7) that implements:

1. **Resilience & Transient-Fault Handling** with Polly (retries, circuit breaker, bulkhead, timeout, fallback)
2. **Caching** (In-memory and Redis distributed cache; cache-aside, write-through, expirations)

Prerequisites

- .NET 9.0 SDK installed
- Visual Studio 2022/2024 Preview or VS Code (with C# extension)
- Redis server running locally (Docker container or native)

---

1. Create the Solution & .NET 9 Web API Project

1. Open a terminal / PowerShell.

Make your solution folder and initialize a solution:

```
 mkdir ResilientCachingDemo
cd ResilientCachingDemo
dotnet new sln -n ResilientCachingDemo
```

2.

Create a Web API targeting **net9.0** and add it to the solution:

```
 dotnet new webapi -n Api --framework net9.0
dotnet sln add Api/Api.csproj
```

3.

(Optional) If you want to pin the SDK version, create a `global.json`:

```
{
  "sdk": { "version": "9.0.100" }
}
```

4.

Open the folder in your IDE:

```
code .
```

5.

---

2. Add Required NuGet Packages

In the **Api** folder, run:

```
cd Api
dotnet add package Microsoft.Extensions.Http.Polly
dotnet add package Polly
dotnet add package Microsoft.Extensions.Caching.Memory
dotnet add package Microsoft.Extensions.Caching.StackExchangeRedis
```

---

3. Configure Services in `Program.cs`

Edit **Program.cs** to wire up caching and resilience:

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


// In-Memory Cache
```

```
builder.Services.AddMemoryCache();


// Distributed Redis Cache

builder.Services.AddStackExchangeRedisCache(opts =>

{

   opts.Configuration = builder.Configuration.GetConnectionString("Redis")

                         ?? "localhost:6379";

   opts.InstanceName = "ResilientCacheDemo:";

});


// HttpClient + Polly policies

builder.Services.AddHttpClient<ExternalService>()

   .SetHandlerLifetime(TimeSpan.FromMinutes(5))

   .AddPolicyHandler(Policies.GetRetryPolicy())

   .AddPolicyHandler(Policies.GetCircuitBreakerPolicy())

   .AddPolicyHandler(Policies.GetTimeoutPolicy())

   .AddPolicyHandler(Policies.GetFallbackPolicy());


// Register our demo service

builder.Services.AddScoped<DemoService>();


builder.Services.AddControllers();

var app = builder.Build();
```

```
app.MapControllers();
app.Run();
```

And in **appsettings.json**, add:

```
{
  "ConnectionStrings": {
    "Redis": "localhost:6379"
  },
  // … other settings …
}
```

---

4. Define Your Polly Policies (`Policies.cs`)

Create **Policies.cs** at the project root:

```
using System.Net;
using Polly;
using Polly.Extensions.Http;

public static class Policies
{
    public static IAsyncPolicy<HttpResponseMessage> GetRetryPolicy() =>
        HttpPolicyExtensions
            .HandleTransientHttpError()
            .WaitAndRetryAsync(
                retryCount: 3,

                sleepDurationProvider: retryAttempt => TimeSpan.FromSeconds(Math.Pow(2,
retryAttempt))

            );


    public static IAsyncPolicy<HttpResponseMessage> GetCircuitBreakerPolicy() =>

        HttpPolicyExtensions

            .HandleTransientHttpError()

            .CircuitBreakerAsync(

                handledEventsAllowedBeforeBreaking: 2,

                durationOfBreak: TimeSpan.FromSeconds(30)

            );


    public static IAsyncPolicy<HttpResponseMessage> GetTimeoutPolicy() =>

        Policy.TimeoutAsync<HttpResponseMessage>(TimeSpan.FromSeconds(10));


    public static IAsyncPolicy<HttpResponseMessage> GetFallbackPolicy() =>

        Policy<HttpResponseMessage>

            .Handle<Exception>()

            .FallbackAsync(

                fallbackValue: new HttpResponseMessage(HttpStatusCode.OK)

                {

                    Content = new StringContent("{\"message\":\"fallback\"}")

                },

                onFallbackAsync: async (exception, ctx) =>
                {
                    Console.WriteLine($"[Fallback] {exception.Exception?.Message}");
                }
            );
}
```

---

5. Implement the External HTTP Service (`ExternalService.cs`)

```
public class ExternalService
{
    private readonly HttpClient _http;
    public ExternalService(HttpClient http) => _http = http;

    public async Task<string> GetDataAsync()
    {
        var response = await _http.GetAsync("https://api.example.com/data");
        response.EnsureSuccessStatusCode();
        return await response.Content.ReadAsStringAsync();
    }
}
```

---

6. Build the Demo Service with Caching (`DemoService.cs`)

```
public class DemoService

{

    private readonly IMemoryCache _memCache;

    private readonly IDistributedCache _distCache;

    private readonly ExternalService _external;



    public DemoService(IMemoryCache memCache,

                       IDistributedCache distCache,

                       ExternalService external)

    {

        _memCache  = memCache;

        _distCache = distCache;

        _external  = external;

    }



    // In-Memory Cache-Aside

    public async Task<string> GetMemoryCachedDataAsync(string key)

    {

        if (!_memCache.TryGetValue(key, out string value))

        {

            value = await _external.GetDataAsync();

            _memCache.Set(key, value, new MemoryCacheEntryOptions
            {
                AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(5)
            });
        }
        return value;
    }

    // Redis Cache-Aside
    public async Task<string> GetRedisCachedDataAsync(string key)
    {
        var cached = await _distCache.GetStringAsync(key);
        if (cached is not null)
            return cached;

        var fresh = await _external.GetDataAsync();
        await _distCache.SetStringAsync(key, fresh, new DistributedCacheEntryOptions
        {
            AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(10)
        });
        return fresh;
    }

    // Write-Through Example
    public async Task SaveDataWithWriteThroughAsync(string key, string data)
```

```
    {
        // e.g. await SaveToDatabaseAsync(key, data);
        _memCache.Set(key, data, TimeSpan.FromMinutes(5));
        await _distCache.SetStringAsync(key, data, new DistributedCacheEntryOptions
        {
            AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(10)
        });
    }
}
```

---

7. Expose Endpoints (`DemoController.cs`)

Create **Controllers/DemoController.cs**:

```
[ApiController]
[Route("[controller]")]
public class DemoController : ControllerBase
{
    private readonly DemoService _svc;
    public DemoController(DemoService svc) => _svc = svc;

    [HttpGet("mem/{key}")]
    public async Task<IActionResult> GetFromMemory(string key) =>
        Ok(await _svc.GetMemoryCachedDataAsync(key));
    [HttpGet("redis/{key}")]

    public async Task<IActionResult> GetFromRedis(string key) =>

        Ok(await _svc.GetRedisCachedDataAsync(key));


    [HttpPost("save/{key}")]

    public async Task<IActionResult> Save(string key, [FromBody] string data)

    {

        await _svc.SaveDataWithWriteThroughAsync(key, data);

        return Ok();

    }

}
```

---

8. Run & Verify

Run your API:

```
dotnet run --project Api/Api.csproj
```

1.
2. **Test** with curl/Postman:
   - `GET https://localhost:5001/demo/mem/foo`
   - `GET https://localhost:5001/demo/redis/foo`
   - `POST https://localhost:5001/demo/save/foo` with body `"hello"`
3. **Simulate Faults** by pointing `ExternalService` to a bad URL; watch Polly's retries, breaker trips, and fallback output in the console.

4. **Observe Caching**: first call fetches live, subsequent calls (within TTL) return cached immediately.
