Minimal APIs in the Field

Make ASP.NET Core Minimal APIs work in your LOB apps

Digital Craftsmanship Nordoberpfalz
13.12.2022

Kenny Pflug

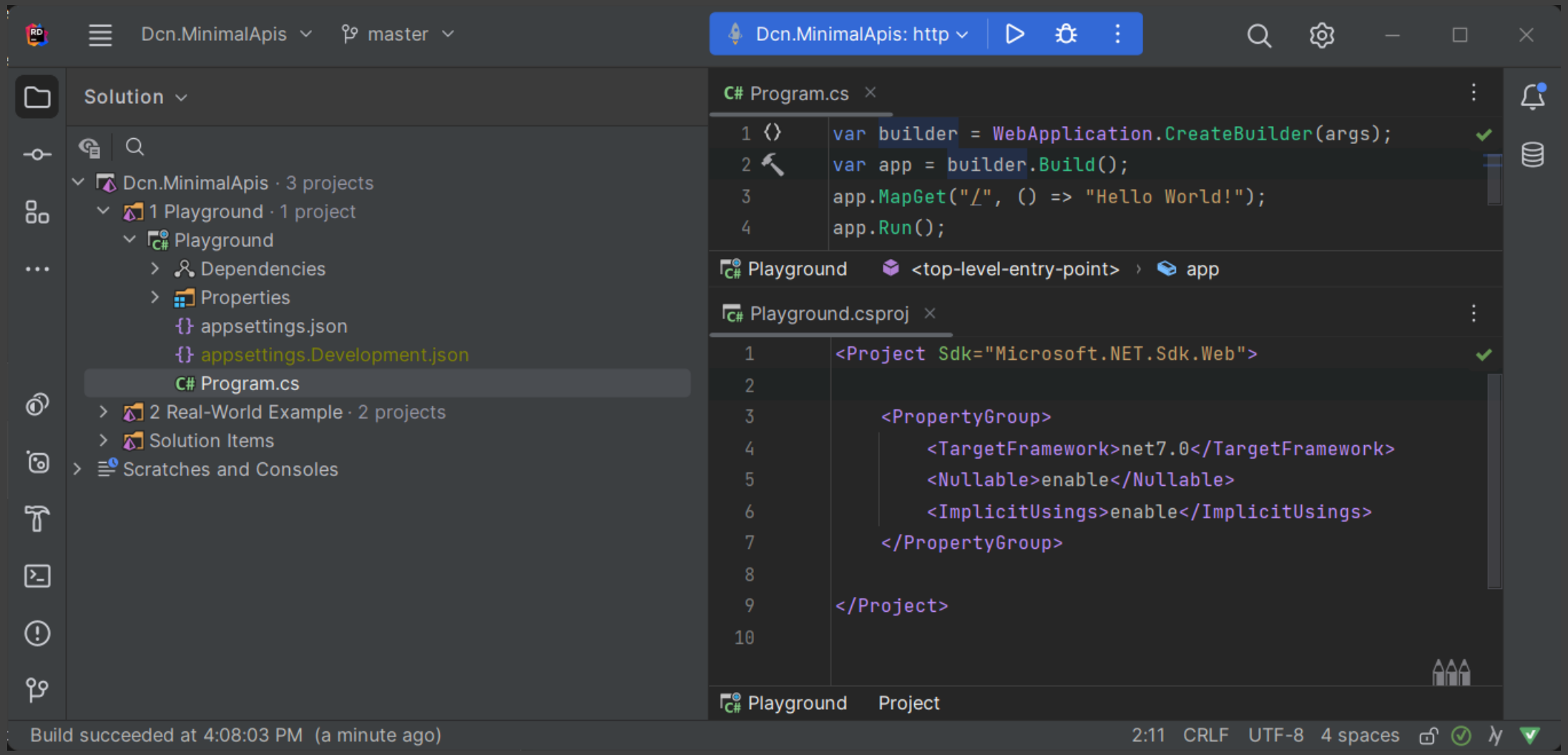
Content

- Where is all my code?
- Structuring HTTP endpoints with Minimal APIs in a real-world project
  - Endpoint Registration
  - Model Binding and Dependency Injection
  - Supporting OpenAPI / Swagger
  - Automated Testing of Minimal API endpoints
  - Validation of parameters and DTOs
  - Patterns and Practices
  - Miscellaneous topics: auth, performance, what's new in .NET 7
- Your questions

Minimal APIs with ASP.NET Core

Where is all my code?

Minimal APIs – Default Template



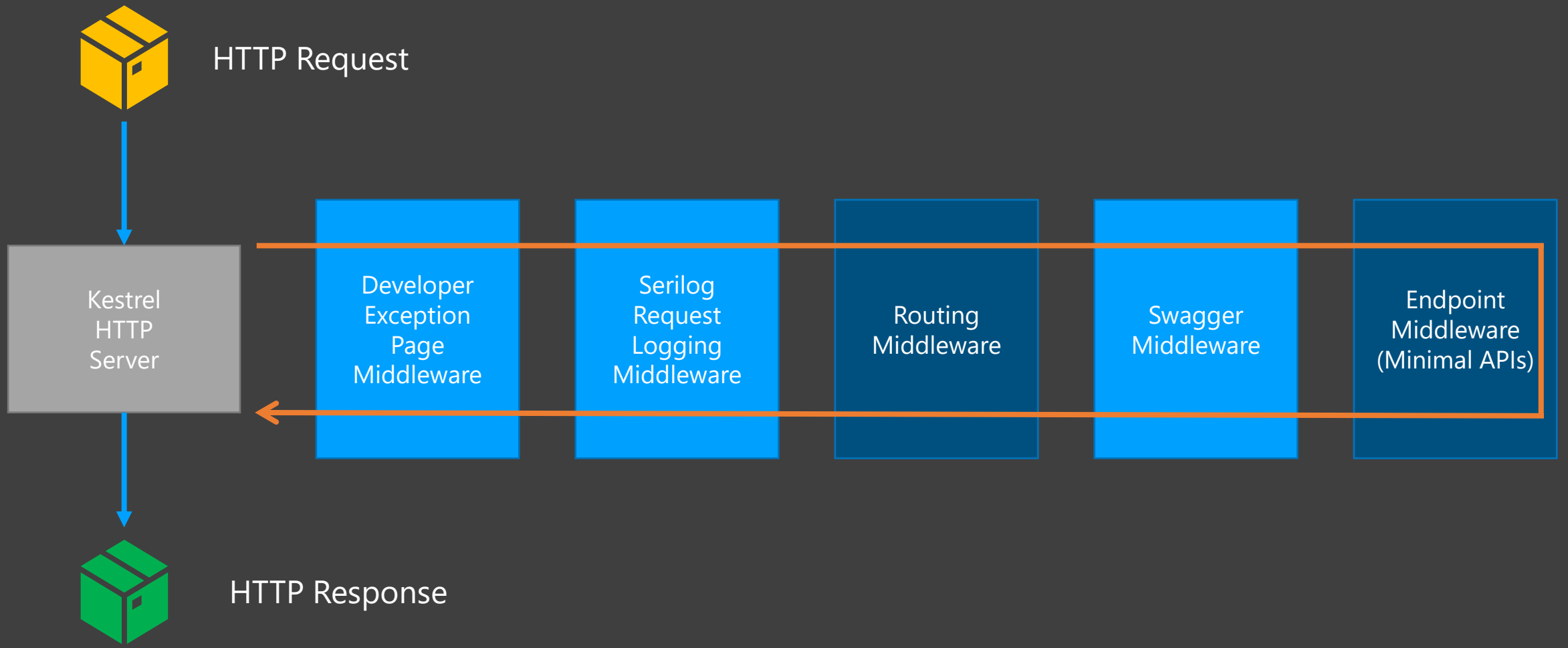
CreateHostBuilder and Startup are gone

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

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

public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddAuthentication();
        services.AddMvc();
    }

    public void Configure(IApplicationBuilder app,
                          IWebHostEnvironment environment)
    {
        if (environment.IsDevelopment())
            app.UseDeveloperExceptionPage();

        app.UseRouting()
           .UseAuthentication()
           .UseEndpoints(builder => builder.MapControllers());
    }
}
```

WebApplicationBuilder

WebApplication

ASP.NET Core Middleware Pipeline



Live Demo

Turning a Minimal API project back to normal

Global Using Directives for Web SDK

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Net.Http.Json;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Routing;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
```

Source: https://github.com/dotnet/aspnetcore/issues/32451#issuecomment-870812382

Structuring HTTP Endpoints

Mapping Endpoints with Delegates

- The MapXXX methods take a delegate that will be called when a corresponding HTTP Request is received
- The delegate can point to an anonymous method / lambda, or any other type of static or instance method
- Internally, the delegate is registered in the endpoint middleware

C# Program.cs

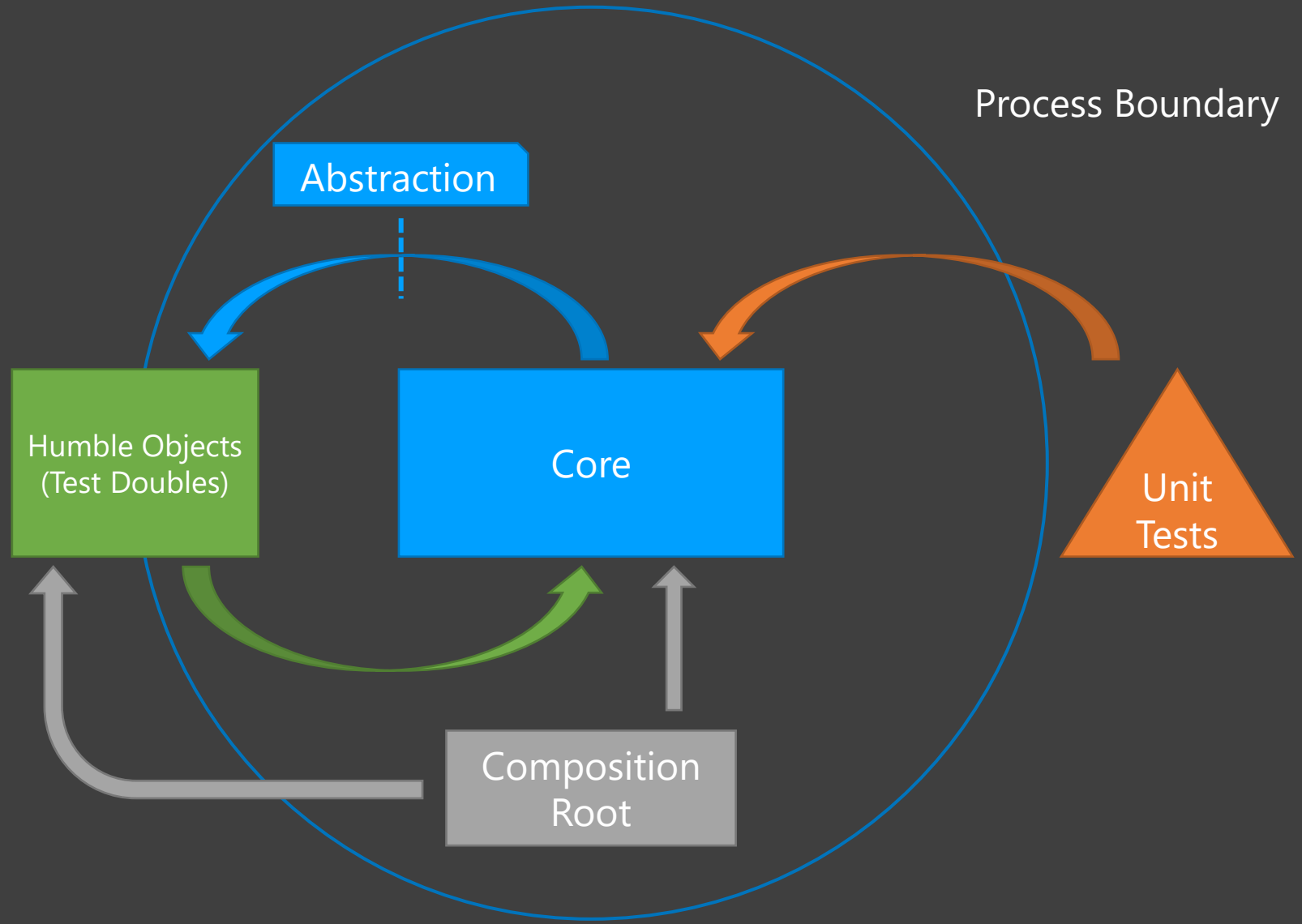
```
1  var builder = WebApplication.CreateBuilder(args);
2  var app = builder.Build();
3
4  app.MapGet("/", () => "Hello World!");
5
6  app.Run();
```

How to organize endpoints?

- The endpoint methods can be called from unit tests without setting up the whole HTTP server
- These methods should not be within a single Program.cs
- I prefer one endpoint per file/class, but you can also achieve several endpoints per file
- Another tip: create namespaces per use case, not for endpoints, data access, interfaces, enums, classes, etc.

```
public static class GetContactsEndpoint
{
    1 usage  Kenny Pflug
    public static WebApplication MapGetContacts(this WebApplication app)
    {
        app.MapGet("/api/contacts", GetContacts)
           .Produces<ContactListDto[]>()
           .Produces<Dictionary<string, string>>(StatusCodes.Status400BadRequest)
           .Produces(StatusCodes.Status500InternalServerError);
        return app;
    }

    /// <summary>
    /// Gets a list of contacts (paged).
    /// </summary>
    /// <param name="validationContext">The validation context that is used to track errors.</param>
    /// <param name="sessionFactory">The factory that creates the session to the database.</param>
    /// <param name="skip">
    /// The number of contacts that will be skipped for paging (optional). The default value is 0.
    /// </param>
    /// <param name="take">
    /// The number of contacts that will be included in the result (optional). The default value is 30.
    /// This value must be between 1 and 100.
    /// </param>
    /// <param name="searchTerm">The search term that is used to filter the contacts (optional). The default value is null.</param>
    /// <response code="400">Occurs when skip is less than 0, or when take is not between 1 and 100.</response>
    2 usages  Kenny Pflug
    public static async Task<IResult> GetContacts(ValidationContext validationContext,
                                                  ISessionFactory<IGetContactsSession> sessionFactory,
                                                  int skip = 0,
                                                  int take = 30,
                                                  string? searchTerm = null)
    {
        if (validationContext.CheckForPagingErrors(skip, take, out var errors))
            return Results.BadRequest(errors);

        searchTerm = searchTerm.NormalizeString();
        await using var session = await sessionFactory.OpenSessionAsync();
        var contacts = await session.GetContactsAsync(skip, take, searchTerm);
        return Results.Ok(ContactListDto.FromContacts(contacts));
    }
}
```

CHUC: Core – Humble Objects – Unit Tests – Composition Root



Live Demo

Real-life project overview

Endpoint Registration

Two ways to register: manual vs. automatic

- By default, endpoints are registered by explicitly calling a MapXXX method
- There is no mechanism to discover and register endpoints automatically, but you can handroll your own
- Be aware: automatic endpoint registration usually involves the use of Reflection – this might hurt you in AoT scenarios
- Automatic registration might also be done via a Source Generator
- I prefer manual registration

```
1 usage   Kenny Pflug
public static WebApplication ConfigureHttpPipeline(this WebApplication app)
{
    if (app.Environment.IsDevelopment())
        app.UseDeveloperExceptionPage();

    app.UseHttpsAndHstsIfNecessary();
    app.UseSerilogRequestLogging();
    app.UseRouting();
    app.UseSwaggerAndSwaggerUi();
    return app.MapEndpoints();
}

1 usage   Kenny Pflug
private static WebApplication MapEndpoints(this WebApplication app)
{
    app.MapHeartbeatEndpoint()
       .MapGetContacts()
       .MapGetContactDetails()
       .AutomaticallyMapEndpoints()
       .MapControllers();
    return app;
}
```

Live Demo

Automatic endpoint registration

Dependency Injection and Model Binding

DI via Method Injection and Model Binding

- By default, dependencies to other objects will be passed via Method Injection next to the regular parameters from the query or body of the request
- These service dependencies reside right next to the arguments that represent query parameters or DTOs deserialized from the JSON body of the request
- Transient or scoped services must be injected this way
- Model binding works in a similar way to ASP.NET Core MVC
- You do not need to decorate parameters with [FromBody] or [FromServices]

```
public static async Task<IResult> GetContactDetails(ISessionFactory<IGetContactDetailsSession> sessionFactory,
                                                    ValidationContext validationContext,
                                                    int id)
{
    if (validationContext.CheckForIdErrors(id, out var errors))
        return Results.BadRequest(errors);

    await using var session = await sessionFactory.OpenSessionAsync();
    var contact = await session.GetContactAsync(id);
    if (contact is null)
        return Results.NotFound();

    return Results.Ok(ContactDetailDto.FromContact(contact));
}
```

Injecting Singletons via the constructor

- If you have an automatic registration mechanism that's based on instantiating objects implementing a certain abstraction, you can use it to inject dependencies via the constructor
- The result looks pretty much like ASP.NET Core MVC Controllers
- Beware: the dependencies become singletons when the mapped endpoint delegate references the object

```
public sealed class UpdateContactEndpoint : IMinimalApiEndpoint
{
    Kenny Pflug
    public UpdateContactEndpoint(UpdateContactDtoValidator validator,
                                 ISessionFactory<IUpdateContactSession> sessionFactory,
                                 ILogger logger)
    {
        Validator = validator;
        SessionFactory = sessionFactory;
        Logger = logger;
    }

    2 usages
    private UpdateContactDtoValidator Validator { get; }
    2 usages
    private ISessionFactory<IUpdateContactSession> SessionFactory { get; }
    2 usages
    private ILogger Logger { get; }

    0+1 usages   Kenny Pflug
    public void MapEndpoint(WebApplication app)
    {
        app.MapPut("/api/contacts/update", UpdateContact)
           .Produces(StatusCodes.Status204NoContent)
           .Produces<Dictionary<string, string>>(StatusCodes.Status400BadRequest)
           .Produces(StatusCodes.Status500InternalServerError);
    }

    /// <summary>
    /// Use this endpoint to update an existing contact.
    /// </summary>
    /// <param name="dto">The DTO describing the ID of the contact and its new values.</param>
    /// <response code="400">Occurs when the DTO is null or when any of the properties is invalid.</response>
    1 usage   Kenny Pflug
    public async Task<IResult> UpdateContact(UpdateContactDto? dto)
    {
        if (Validator.CheckForErrors(dto, out var errors))
            return Results.BadRequest(errors);

        await using var session = await SessionFactory.OpenSessionAsync();
        var contact = await session.GetContactAsync(dto.Id);
        if (contact is null)
            return Results.NotFound();

        contact.FirstName = dto.FirstName;
        contact.LastName = dto.LastName;
        contact.Email = dto.Email;
        await session.UpdateContactAsync(contact);

        var address = contact.Address!;
        address.Street = dto.Street;
        address.ZipCode = dto.ZipCode;
        address.Location = dto.Location;
        await session.UpdateAddressAsync(address);

        await session.SaveChangesAsync();

        Logger.Information("The contact {@Contact} was updated successfully", contact);
        return Results.NoContent();
    }
}
```

Swagger Support

AspNetCore.Swashbuckle with Minimal APIs

- The mapped delegates of Minimal APIs will be added to ASP.NET Core's endpoints API explorer
- When using methods, Swashbuckle can pick up the XML comments of those for documentation purposes
- Configuration of Swagger/OpenAPI is not that different from ASP.NET Core MVC

```
/// <summary>
/// Gets a list of contacts (paged).
/// </summary>
/// <param name="validationContext">The validation context that is used to track errors.</param>
/// <param name="sessionFactory">The factory that creates the session to the database.</param>
/// <param name="skip">
/// The number of contacts that will be skipped for paging (optional). The default value is 0.
/// </param>
/// <param name="take">
/// The number of contacts that will be included in the result (optional). The default value is 30.
/// This value must be between 1 and 100.
/// </param>
/// <param name="searchTerm">The search term that is used to filter the contacts (optional). The default value is null.</param>
/// <response code="400">Occurs when skip is less than 0, or when take is not between 1 and 100.</response>
2 usages  Kenny Pflug
public static async Task<IResult> GetContacts(ValidationContext validationContext,
                                              ISessionFactory<IGetContactsSession> sessionFactory,
                                              int skip = 0,
                                              int take = 30,
                                              string? searchTerm = null)
```

Swagger with anonymous methods in .NET 7

- With the new Microsoft.AspNetCore.OpenApi NuGet package, you can imperatively describe your Web API endpoint
- You can also mix this with the previously shown approach
- https://github.com/dotnet/AspNetCore.Docs/pull/25863

```
app.MapPost("/todo2/{id}", async (int id, Todo todo, TodoDb db) =>
{
    todo.Id = id;
    db.Todos.Add(todo);
    await db.SaveChangesAsync();

    return Results.Created($"/todoitems/{todo.Id}", todo);
})
.WithOpenApi(generatedOperation =>
{
    var parameter = generatedOperation.Parameters[0];
    parameter.Description = "The ID associated with the created Todo";
    return generatedOperation;
});
```

Automated Testing of Minimal API endpoints

IResult was hard – but now, it's fine

- The IResult implementations are internal and cannot be accessed in unit tests
- Damien Edwards create MinimalApis.Extensions which also contains public implementations
- We ended up creating our own because of some discrepancies: https://github.com/Synnotech-AG/Synnotech.AspNetCore.MinimalApis
- Since .NET 7, the classes implementing IResult are public

Getting Started

1. Install the NuGet package into your ASP.NET Core project:

```
> dotnet add package MinimalApis.Extensions --prerelease
```

2. In your project's `Program.cs`, call the `AddEndpointsProvidesMetadataApiExplorer()` method on `builder.Services` to enable enhanced endpoint metadata in `ApiExplorer`:

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddEndpointsProvidesMetadataApiExplorer(); // <-- Add this line
builder.Services.AddSwaggerGen();
...
```

3. Update your Minimal APIs to use the helper binding and result types from this library, e.g.:

```
app.MapPost("/todos", async Task<Results<ValidationProblem, Created<Todo>>> (Validated<Todo> input
{
    if (!input.IsValid)
        return Results.Extensions.ValidationProblem(input.Errors);

    var todo = input.Value;
    db.Todos.Add(todo);
    await db.SaveChangesAsync();

    return Results.Extensions.Created($"/todos/{todo.Id}", todo);
});
```

Live Demo

Unit testing for Minimal APIs

Integration testing for the whole middleware pipeline

- Microsoft suggests using the WebApplicationFactory<T> (where T is either a Program or a Startup class)
- We ended up writing our own base class for better integration with Xunit logging and more control over the configuration of the composition root

```
        ...t = output;

        ...ogger logger = output.CreateTestLogger();

        var builder = WebApplication.CreateBuilder();
        Log.Logger = logger;
        builder.Host.UseSerilog(logger);
        builder.Services.AddSingleton<IAuthenticationService, AuthenticationServiceStub>();
        builder.Services.AddSingleton<IWebHostEnvironment, WebHostEnvironmentStub>();

        App = builder.Build();
        App.Urls.Add(TestServerSettings.GetHostUrlWithPort());
        App.AddStatusCodeResponses()
            .AddObjectResponses()
            .AddRedirectAndNotAllowedResponses()
            .AddFileResponses();
    }

    1 reference
    protected static string Url { get; set; } = TestServerSettings.GetHostUrlWithPort();

    5 references
    protected ITestOutputHelper Output { get; }

    6 references
    private WebApplication App { get; }

    36 references
    protected HttpClient HttpClient { get; } =
        new () { BaseAddress = new Uri(Url, UriKind.Absolute) };

    0 references
    public Task InitializeAsync() => App.StartAsync();

    0 references
    public async Task DisposeAsync()
    {
        try
        {
            await App.StopAsync();
            await App.DisposeAsync();
        }
        catch (Exception exception)
```

Validation of Query Parameters and Data Transfer Objects

No built-in validation

- Minimal APIs don't contain any validation mechanism by default
- MinimalApis.Extensions adapts System.ComponentModel.DataAnnotations with Validated<T>
- FluentValidation has no integration for minimal APIs yet (ValidationResult must be converted into something useful for the response body)
- I ended up creating Light.Validation

```
public sealed class UpdateContactDtoValidator : Validator<UpdateContactDto>
{
    Kenny Pflug
    public UpdateContactDtoValidator(IValidationContextFactory validationContextFactory)
        : base(validationContextFactory) { }

    Kenny Pflug
    protected override UpdateContactDto PerformValidation(ValidationContext context, UpdateContactDto dto)
    {
        context.Check(dto.Id).IsGreaterThan(0);
        context.ValidateContactProperties(dto);
        return dto;
    }
}
```

Live Demo

Validation of parameters and DTOs

Miscellaneous infos and conclusion

Miscellaneous infos

- Minimal APIs is about 5% to 12% faster than ASP.NET Core MVC
- Auth works pretty much the same way as in ASP.NET Core MVC
- .NET 7 introduces
  - Filters
  - Route Groups
  - File Upload
  - OpenAPI support for anonymous methods
  - Public IResult implementations
  - Enhanced model binding support for headers, query strings, and request bodies

Conclusion

- Should you immediately switch to Minimal APIs? Probably not. The performance improvements might be relevant for you in Cloud scenarios.
- Will Minimal APIs become important? Probably yes, it will become the main framework for building Web APIs in the upcoming years.
- Should you use anonymous methods as endpoints? Probably not – no unit testing support. Might make sense in a Microservice environment.
- Should you place all endpoints in Program.cs? No, probably not – that doesn't scale well. You will have more merge conflicts and bloat up the file. Might make sense in a Microservice environment.
- You can run ASP.NET Core MVC alongside Minimal APIs – thus you can transition slowly in existing projects.

Thank you!

Got any questions?