APIer i .NET

Med Controllers(**Legacy**)

Legacy – Äldre kod eller arkitektur som vi inte ska skapa nytt med men måste underhålla. Se även ”Teknisk Skuld”

Historik

- Webbutveckling med .NET sträcker sig långt tillbaks
- Först hade vi WebForms som skulle vara en variant av WinForms för webben.
- Sen kom Razor-Pages och ASP.NET MVC
  - **M**(odel) **V**(iew) **C**(ontroller)
- MVC är ett designmönster, ungefär som MVVM som syftar till att separera GUI från logik.
- Dessa Controllers blev sedan standard för APIer i .NET
- I .NET 6 fick vi det som kallas Minimal API

Olika typer av APIer

Controllers

- Bygger på MVC-paketet
- Nyttjar Attribut (Decorator Pattern) för att bygga upp ett API
- Används i många befintliga projekt
- Ser ”enklare” ut med mindre boilerplate
- Är beroende av MVC
- Är mer resurskrävande

Minimal API

- Kräver inga paket utom standard ASP.NET
- Nyttjar metodkedjor för att bygga upp ett api
- Nyare sätt att bygga mer effektiva och snabbare APIer
- Ser mer komplext ut pga lamda-funktioner

Controller och ControllerBase

- För att bygga ett API med Controllers behöver vi skapa en klass som ärver av ControllerBase.
- Något som kan vara lurigt om man använder vissa mallar i Visual Studio är att controllern då ärver utav Controller som syftar till att skapa en MVC-applikation
- Controller innehåller alltså extra stöd och kod för MVC som gör ditt API långsammare och större. Använd endast detta för just MVC.
- ControllerBase är en basklass Microsoft tagit fram för att ”banta” Controller-apier. Detta löser dock inte de stora problemen med det

Attributdrivet

- Controllers nyttjar Attribut för att konfigurera endpoints.
  - [ApiController] – Anger att controllern är medad som en API-controller och inte för MVC
  - [Route(”api/[controller]”)] – Anger en bas-path för alla endpoints i controllern
  - [HttpGet] – Indikerar att metoden ska mappa mot en GET-request
  - [HttpPost] – Indikerar att metoden ska mappa mot en POST-request
  - [HttpPut] – Indikerar att metoden ska mappa mot en PUT-request
  - [HttpPatch] – Indikerar att metoden ska mappa mot en Patch-request
  - [HttpDelete] – Indikerar att metoden ska mappa mot en Delete-request
- I alla dessa kan man lägga på en path ex: [HttpGet(”{id}”)]

Demo

Övning 1 – 2 och 2 Konvertera Demo-apiet från sist

- Skapa ett nytt projekt i VisualStudio 2022 av typen ASP.NET WebAPI, kryssa i ”Use Controllers”
- Rensa bort WeatherForecast ur projektet (i Controllern räcker att rensa innehåll i metoder och byta namn)
- Återskapa APIet vi skapade förra veckan (https://github.com/niklas-hjelm/LiveDemoNETAPI) men med Controller
- Testa API-et i Postman

Ytterligare attribut

- Det finns ytterligare attribut vi kan använda i Controllers
  - [FromBody] – Indikerar att en parameter kommer ifrån request-bodyn
  - [FromQuery] – Indikerar att en parameter kommer ifrån Query-string
  - [FromRoute] – Indikerar att en parameter kommer ifrån pathen
  - [FromServices] – Indikerar att en parameter ska hämtas ifrån ServiceCollection
- Det finns några till [FromX] men de gör som exemplen ovan det som de säger.
- Ytterligare Attribut finns för andra funktioner som inte är relevant än. Men alla dessa finns dokumenterade i MS Docs

Controllers Pros and cons

- Pros:
  - Lättläst och tydligt.
  - Mycket bra tillsammans med resten av MVC-paketet. (Även här är det värt att separera MVC och API)
  - Enkelt att gruppera endpoints.
  - Kräver lite för att sätta upp
- Cons:
  - Ineffektivt jämfört med alternativet
  - Resurskrävande jämfört med alternativet
  - Skapar hårt kopplad kod
  - Introducerar beroenden till MVC

Myth-busting

- Många artiklar som förklarar skillnaden mellan MinimalAPI och Controllers pekar på att en av vinsterna med Controllers är att vi kan isolera endpoints i egna klasser. Detta är sant även om MinimalAPI. Alltså inte unikt för Controllers.
- En annan myt är att Controllers är enklare att debugga. Detta stämmer inte heller eftersom båda varianterna är lika lätta att debugga.
- **VIKTIGT!** Microsoft satsar på utvecklingen av MinimalAPI. Branschen håller på att ställa om och Controllers (som de är idag) kommer att fasas ut. Det finns redan idag många jättebra paket för att separera och gruppera endpoints på ett snyggt och enkelt sätt.

Extension metoder

- Det finns en speciell typ av metoder som kallas extension methods
- Dessa metoder utökar beteendet hos en befintlig klass
- Vi kan alltså lägga till och definiera nytt beteende för klasser från paket
- Ett vanligt exempel på detta är att skapa extensionmetoder på WebApplication eller IServiceCollection
- Många av de paket vi kommer att använda nyttjar detta för att förenkla för oss
- För att åstadkomma detta används en statisk klass med en statisk metod som returnerar det man vill utöka och tar inte en instans av samma typ med nyckelordet this

Demo

Övning 2 – 2 och 2 Konvertera Controller till Minimal API

- Utgå från APIet från förra övningen
- Skapa en statisk klass som heter EndpointBuilderExtensions
- I denna, skapa en statisk metod som heter MapMyEndpoints som returnerar void och tar in ”this IEndpointRouteBuilder app”
- I den metoden kör ”var minGrupp = app.MapGroup(”api/test”);”
- Mappa dina endpoints på minGrupp.

- Testa att flytta koden i dina endpoints till egna metoder med hjälp av resharper

Demo

Källor

- Milan Jovanovic - https://youtu.be/gsAuFIhXz3g?si=IGEq_9-sZI4kRvlg
- Nick Chapsas - https://youtu.be/pYl_jnqlXu8?si=iHw-TfCS5XUjD-NM

Dessa två är de bästa resurserna ni kan ha för APIer i .NET.

Milan är mer pedagogisk och noggrann.

Nick är lite mer avancerad och pratar snabbare

EntityFramework Core i ASP.NET Core

Vad behöver vi göra?

- Det finns några saker vi behöver göra.
- Om vi har vår databashantering i ett klassbibliotek (Vilket vi bör ha) och vill skapa migrationer/uppdatera databasen genom vårat API så måste vårat API referera dels till databas-projektet, men också till EntityFrameworkCore.Design
- Vi behöver också registrera DbContext som en service. Antingen som Scoped. Eller genom den inbyggda metoden AddDbContext<T> (som också registrerar den som Scoped).
- Anledningen till att den behöver vara Scoped är för att inte upprätta en anslutning till databasen som ska gälla för alla anrop som vi hade fått genom Singleton. Vi vill heller inte ha en ny anslutning vid varje anrop för att kunna genomföra transaktioner. Scoped låter oss återanvända samma referens till dess att den inte används mer.

Vart ska vi lägga vår ConnectionString?

- Hittils har vi haft vår connection string i OnConfiguring inne i DbContext.
- Detta funkar när vi arbetar med lokala databaser för vår egen skull eftersom ingen utomstående kan komma åt vår lokala databas.
- I produktion ska vi aldrig exponera vår connectionstring publikt läsbart
- Senare under kursen kommer ni att få lära er mer om hur detta går till med hjälp av Miljövariabler/Secrets
- Tills vidare kommer vi att flytta den till en fil som heter applicationsettings.json

Demo

Studiegruppsövning tills V.9 – Crud API med databas

- Genomförande
  - Sitt gemensamt på valfri plats (eller digitalt).
  - En skriver kod och de andra berättar vad som ska skrivas.
  - Rotera vem som skriver kod var tionde minut. Den som skriver kod får inte ge input till vad som ska skrivas.
- Uppgift
  - Börja med att enas om en databasmodell, alltså vilka tabeller och relationer som ska finnas. Det ska finnas minst en ”Many-to-Many”-relation
  - Skapa ett CRUD-API, alltså ett API som har POST,GET,PUT,DELETE med tillhörande databasoperationer för denna databasen.
  - Ni kommer alltså att behöva sätta upp ett projekt för APIet och ett för Databashantering.