Learning About Core Concepts and Reading Resources

Kevin Dockx
Architect

@KevinDockx | www.kevindockx.com

Coming Up

- Dependency injection in minimal APIs
- Routing
- Exposing DTOs instead of entity objects
- Parameter binding
- Improving API responses

Inversion of Control (IoC)

The IoC pattern delegates the function of selecting a concrete implementation type for a class's dependencies to an external component

Dependency Injection in Minimal APIs

The handler's dependencies are provided by an external component

- Don't new them up

Dependency Injection (DI)

The DI pattern uses an object - the container - to initialize other objects, manage their lifetime and provide the required dependencies to objects

Dependency Injection in Minimal APIs

Support via ASP.NET Core's built-in container

```
builder.Services.AddSingleton<TService>(…);
builder.Service.AddSingleton<TService, TImplementation>(…);

builder.Services.AddScoped<TService>(…);
builder.Service.AddScoped<TService, TImplementation>(…);

builder.Services.AddTransient<TService>(…);
builder.Service.AddTransient<TService, TImplementation>(…);
```

Dependency Injection in Minimal APIs

Extension methods for each lifetime, register by type or by contract & implementation

```
builder.Services.AddDbContext<DishesDbContext>(o => o.UseSqlite(
    builder.Configuration["ConnectionStrings:DishesDBConnectionString"]));
```

Dependency Injection in Minimal APIs

Helper extension methods exist for common sets of services

```
public SomeController(DishesDbContext dishesDbContext)
{
    // … store the injected dependency in a field
}
```

Dependency Injection in Minimal APIs

Example of constructor injection

```
app.MapGet("/dishes", (DishesDbContext dishesDbContext) =>
{
    // … implementation
});
```

Dependency Injection in Minimal APIs

Example of injection via handler parameter

Demo

Dependency injection in minimal APIs

Routing

The process of matching an HTTP method & URI to a specific route handler

Learning About Routing

`app.Map`*`Action`* **methods**

- Where *Action* = the HTTP method
- `IEndpointRouteBuilder`

Learning About Routing

HTTP method	Endpoint route builder method	Request payload	Sample URI	Response payload
GET	`MapGet`	-	/dishes /dishes/{dishId}	dish collection single dish
POST	`MapPost`	single dish	/dishes	single dish
PUT	`MapPut`	single dish	/dishes/{dishId}	single dish or empty
DELETE	`MapDelete`	-	/dishes/{dishId}	-

Using HTTP methods as intended improves the overall reliability of your system

Use HTTP Methods as Intended

Different components in your architecture will rely on correct use of the HTTP standard

- Don't introduce potential inefficiencies
- Don't introduce potential bugs

For reference, RFC9110

- `https://datatracker.ietf.org/doc/html/rfc9110`

URI

`https:/localhost:7070/dishes/dab23b28-df91-47e1-948e-76bf3e0c1546`

Route Parameters

Gather input via the URI with route parameters

URI

```
https:/localhost:7070/dishes/dab23b28-df91-47e1-948e-76bf3e0c1546
```

Route pattern

```
app.MapGet("/dishes/{dishId}", …);
```

Route Parameters

In the route template/pattern, use accolades to signify a route parameter

URI

```
https:/localhost:7070/dishes/dab23b28-df91-47e1-948e-76bf3e0c1546
```

Route pattern + handler

```
app.MapGet("dishes/{dishId}", () => (Guid dishId) => { … });
```

Route Parameters

Route parameters from the URI will be bound to same-name parameters in the handler signature

Get dish by id: guid

```
app.MapGet("dishes/{dishId:guid}", () => (Guid dishId) => { … });
```

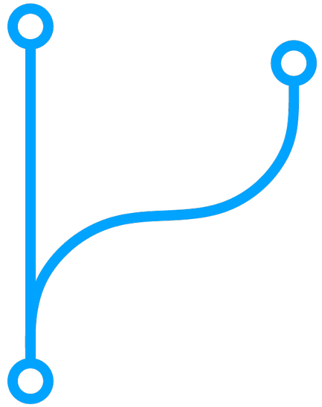
Get dish by name: string (implied constraint)

```
app.MapGet("dishes/{dishName}", () => (string dishName) => { … });
```

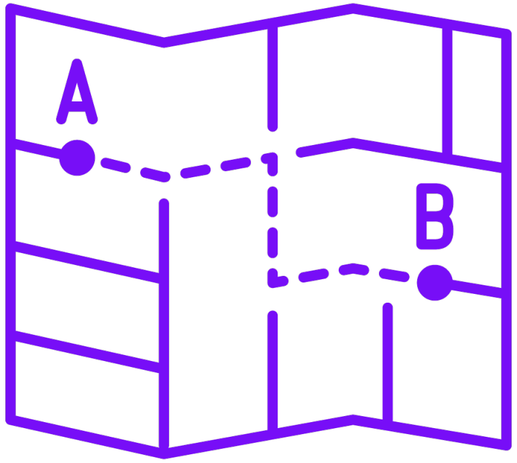
Route Constraints

Route constraints allow constraining the matching behavior of a route

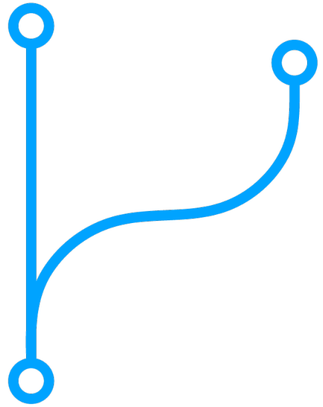
Routing Concepts



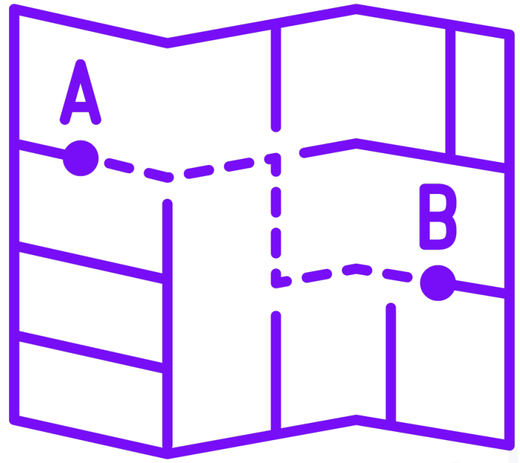
Route builder methods on `IEndpointRouteBuilder`



Route templates or patterns



Route parameters



Route constraints

Demo

Working with routing templates

Why You Shouldn't Expose the Entity Model

Don't expose the entity model to the outside world

Entity model

A means to represent database rows as objects (object graphs)

DTO model

A means to represent the data that’s sent over the wire

Why You Shouldn’t Expose the Entity Model

Potential differences between an entity & a related DTO

Dish (entity)

```
public class Dish
{
    [Key]
    public Guid Id { get; set; }

    public required string Name { get; set; }

    public ICollection<Ingredient> Ingredients
      { get; set; } = new List<Ingredient>();
}
```

Dish (dto)

```
public class Dish
{
    public Guid Id { get; set; }

    public required string Name { get; set; }

    public int IngredientCount { get; set; }
}
```

Why You Shouldn’t Expose the Entity Model

Potential differences between an entity & a related DTO

DishIngredient (entity)

```
public class DishIngredient
{

    public int AmountOfIngredient { get; set; }

    // … additional fields – FK, PK, …
}
```

Ingredient (dto)

```
public class Ingredient
{

    public int AmountOfIngredient { get; set; }

    // … additional fields – PK,
    // ingredient name, description, …
}
```

Keeping the entity and DTO model separate leads to more robust, reliably evolvable code

Demo

Adding the DTO model and using AutoMapper

Parameter binding

The process of converting request data into strongly typed parameters that are expressed by route handlers

Parameter Binding Sources

Route values

Query string

Header

Body (as JSON)

Services provided by DI

Custom

```
app.MapGet(
     "/dishes/{dishId:guid}",
     async (DishesDbContext dishesDbContext,
    IMapper mapper,
    Guid dishId) => { … }
);
```

Parameter Binding Sources

Use binding attributes to be explicit about the binding source

```
app.MapGet(
    "/dishes/{dishId:guid}",
    async ([FromServices] DishesDbContext dishesDbContext,
    [FromServices] IMapper mapper,
    [FromRoute] Guid dishId) => { … }
);
```

Parameter Binding Sources

Use binding attributes to be explicit about the binding source

Parameter Binding Sources

Route values

Query string

Header

Body (as JSON)

Services provided by DI

Custom

Parameter Binding Sources

Route values
`[FromRoute]`

Query string
`[FromQuery]`

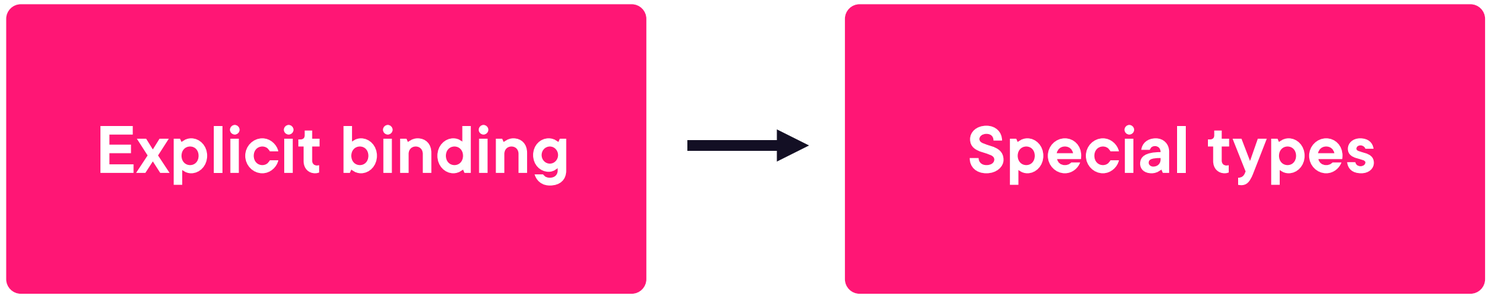
Header
`[FromHeader]`

Body (as JSON)
`[FromBody]`

Services provided by DI
`[FromServices]`

Custom

Rules for Selecting the Binding Source



Special Types for Binding

HttpContext

HttpRequest (bound from HttpContext.Request)

HttpResponse (bound from HttpContext.Response)

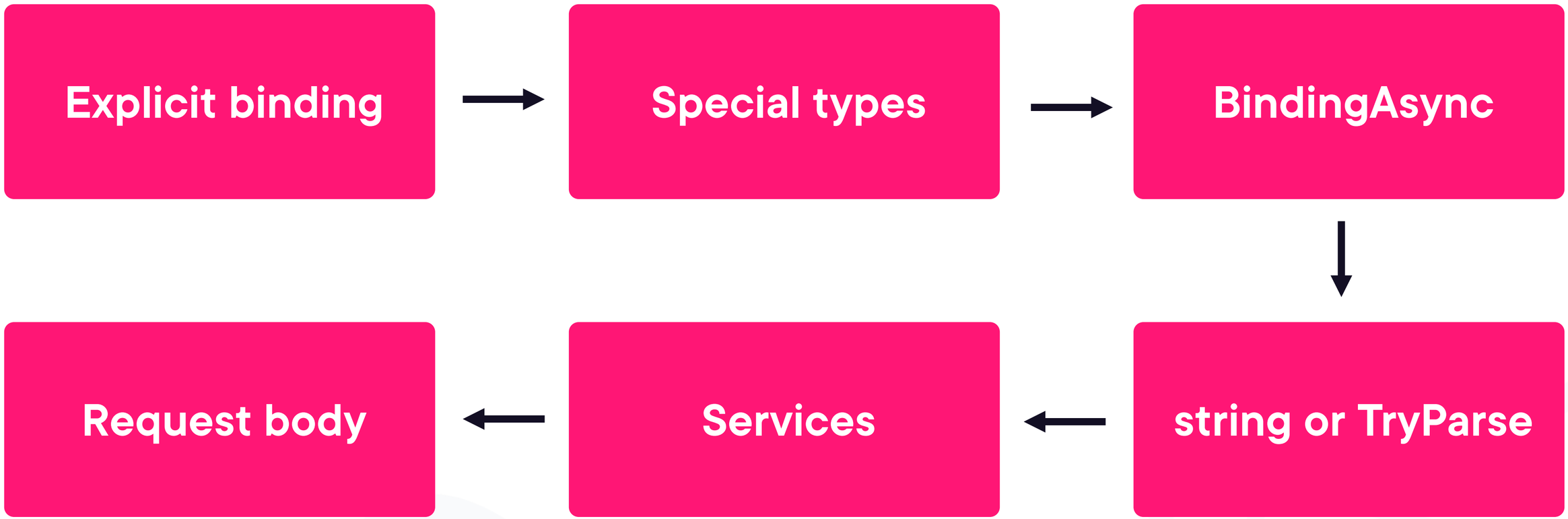
ClaimsPrincipal (bound from HttpContext.User)

CancellationToken (bound from HttpContext.RequestAborted)

Special Types for Binding

- IFormFileCollection (bound from HttpContext.Request.Form.Files)
- IFormFile (bound from HttpContext.Request.Form.Files[paramName]
- Stream (bound from HttpContext.Request.Body)
- PipeReader (bound from HttpContext.Reqeuest.BodyReader)

Rules for Selecting the Binding Source



Demo

Parameter binding

Filtering

Limiting a collection resource, taking into account a predicate

```
GET https://localhost:7070/dishes?name=Rendang
```

Modelling Common API Functionality

Filter via the query string, using name/value combinations for fields to filter on

Searching

Searching for matching items in a collection based on a predefined set of rules

```
GET https://localhost:7070/dishes?searchQuery=stew
```

Modelling Common API Functionality

Search via the query string, passing through a value to search for
It's up to the API to decide what to search through and how to search

```
GET https://localhost:7070/dishes?orderby=name desc, price
```

Modelling Common API Functionality

Sort via the query string, passing through fields and (optional) direction
Allow sorting on properties of the DTO, not on entity properties

```
GET https://localhost:7070/dishes?pageNumber=2&pageSize=5


200 OK
Headers:
X-Pagination :{"pageNumber": 2, "pageSize": 5, "totalCount": 24, … }
```

Modelling Common API Functionality

Page via the query string, passing through page number & page size
Page by default
Limit the page size
Return pagination metadata in a response header

Status codes are inspected by the consumer to know how the request went

Status Codes and Creating Responses

Status codes tell the consumer

- If the request worked out as expected
- If something went wrong, who's responsibility it is

Using the correct status codes is essential

Common Status Codes

200 – Ok

201 – Created

204 – No Content

400 – Bad Request

401 – Unauthorized

403 – Forbidden

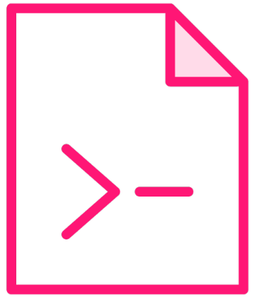
404 – Not Found

405 – Method Not Allowed

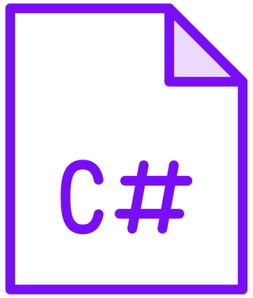
500 – Internal Server Error

Minimal API Endpoint Return Values

`string`



Any type but `string`



`IResult`**-based types:**
`Results.X`,
`TypedResults.X`

Demo

Creating correct API responses

Summary

Inject dependencies via the handler delegate

Routing matches an HTTP method & URI to a specific route handler

- `MapGet(…)`, `MapPost(…)`, …
- Stick to the HTTP standard

Summary

Design URIs with route patterns

- Parameters are bound with { }

Constraint matching behavior with route constraints

Summary

Parameters can be bound from a variety of sources

- Route, query string, headers, request body, ...

Do not expose the entity model to improve the robustness, reliability and evolvability of your codebase

Summary

Return specific responses via `Results.X` or `TypedResults.X`

- The consumer relies on the response status code to learn the outcome of a request

Up Next:

Manipulating Resources