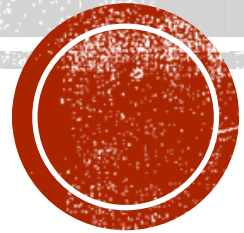
# MAPPING URLS TO ENDPOINTS USING ROUTING



Kamal Beydoun
Lebanese University – Faculty of Sciences I
Kamal.beydoun@ul.edu.lb

# WHAT IS ROUTING?

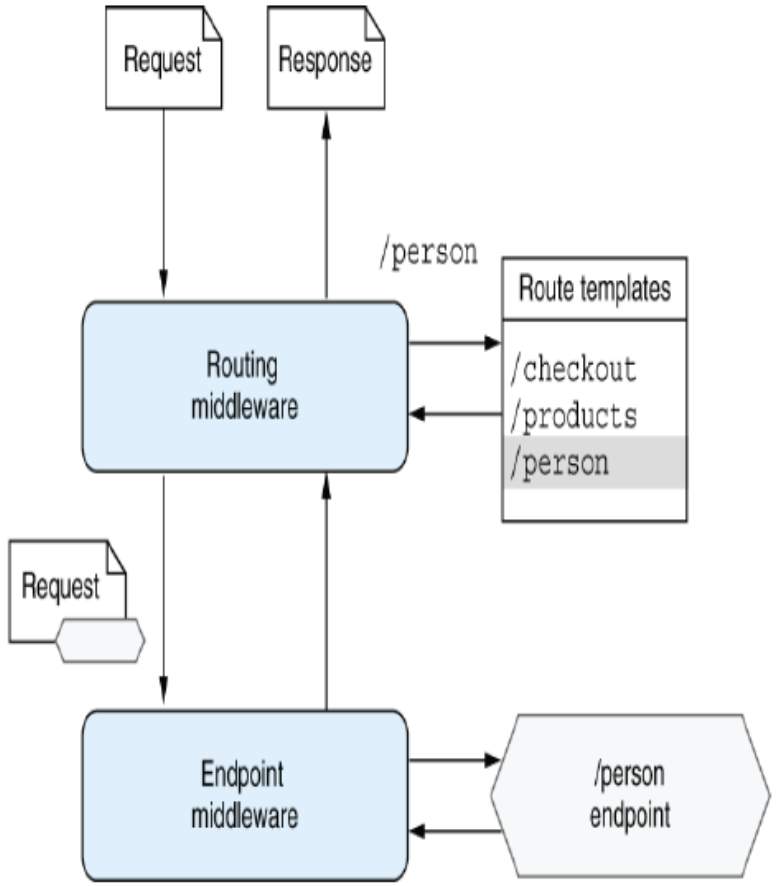
- Routing is the process of **mapping** an incoming request to a **method** that will **handle it**.
- You can use **routing** to control the URLs you expose in your application.
- You can also use routing to enable powerful features like mapping **multiple URLs** to the same Razor Page, and automatically extracting data from a request's URL.

# WHAT IS ROUTING?

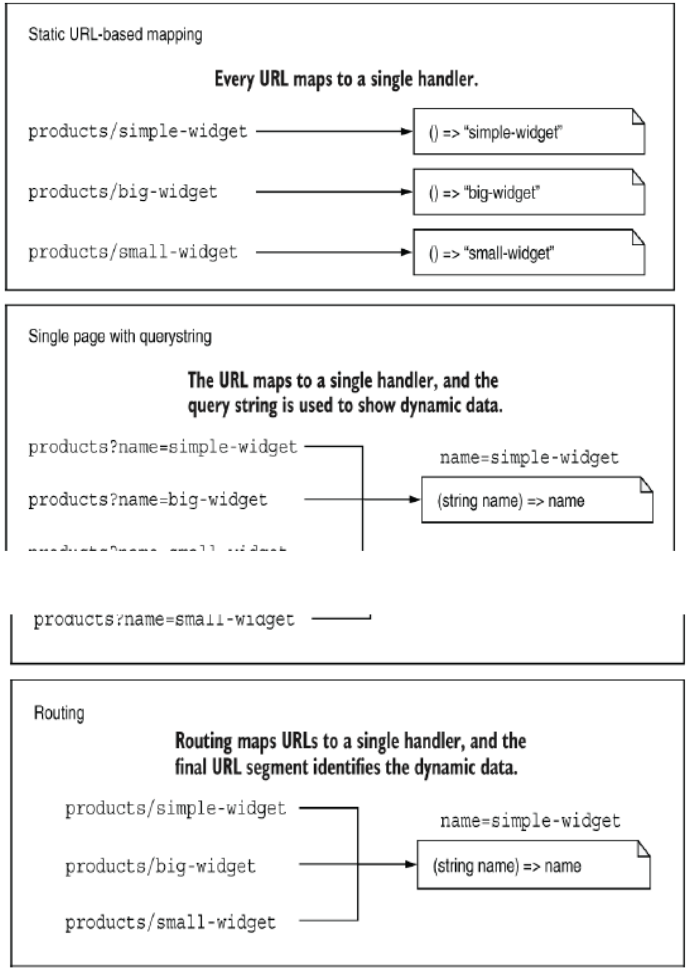
- Routing in ASP.NET Core is the **process of mapping** an incoming HTTP request to a specific **handler**.
  - In minimal APIs, the handler is the endpoint handler associated with a route.
  - In Razor Pages, the handler is a page handler method in a Razor Page.
  - In MVC, the handler is an action method.

# WHAT IS ROUTING?

1. The routing middleware maps the /person URL against a list of templates and selects the associated handler as the endpoint.
2. The routing middleware records the selected endpoint in the request object on the HttpContext.
3. The endpoint middleware executes the selected endpoint and returns the response.



# WHY IS ROUTING?




If you were using a purely file layout-based routing system, you'd only have two options:

- Use a **different handler** for every product in your product range.
  - unfeasible for almost any realistically sized product range.
- Use a **single handler** and use the **query string** to differentiate between products.

# ENDPOINT ROUTING IN ASP.NET CORE

Endpoint routing is implemented using two pieces of middleware :

- **EndpointMiddleware**
  - is to **register** the "**endpoints**" in routing of the system when you start your application.
  - executes one of the endpoints at runtime.
- **EndpointRoutingMiddleware**.
  - chooses **which** of the endpoints registered by the EndpointMiddleware should execute for a given request at runtime.

# REGISTERING MULTIPLE ENDPOINTS

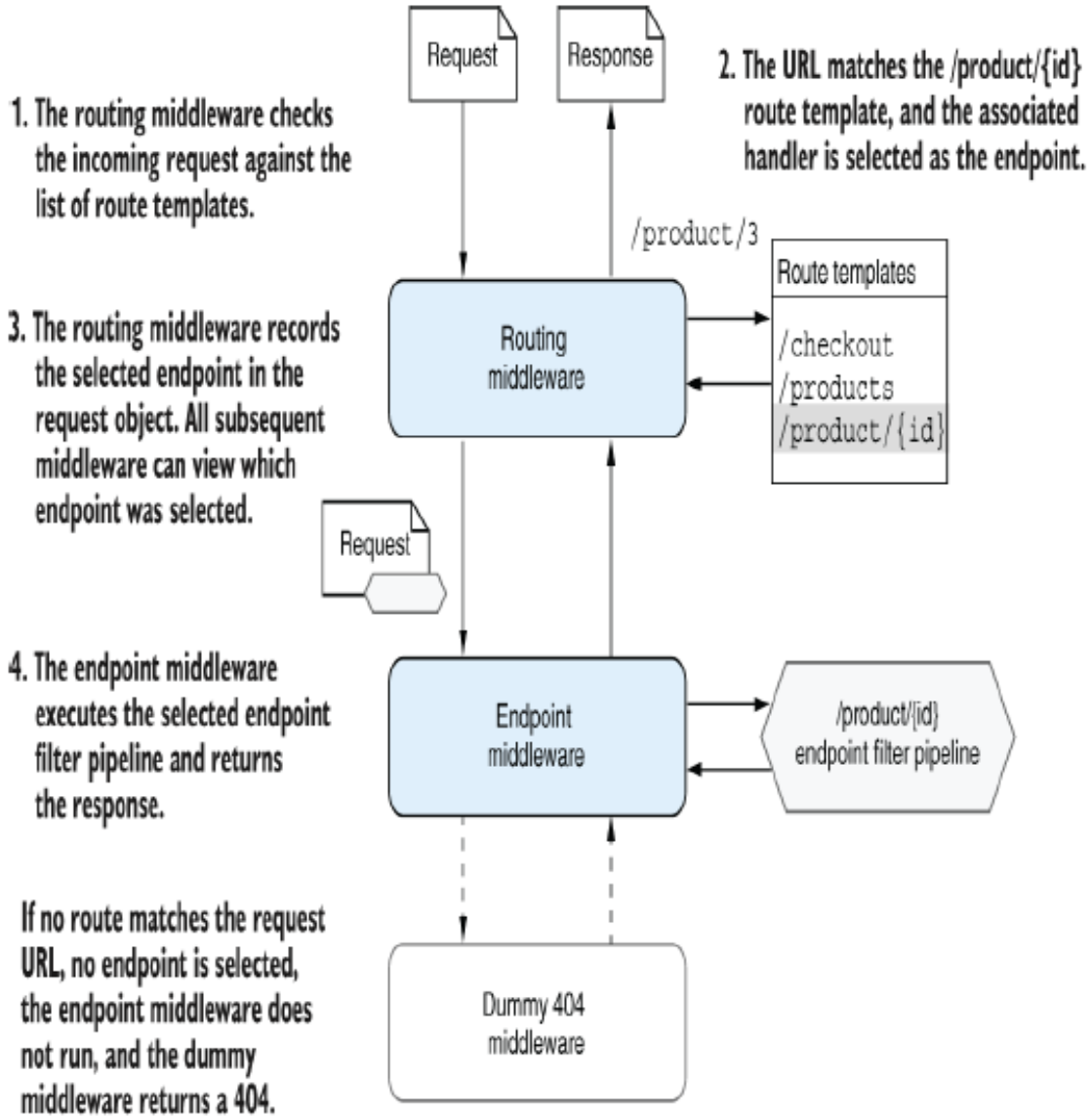
```
WebApplicationBuilder builder = WebApplication.CreateBuilder(args);

builder.Services.AddHealthChecks();          ❶
builder.Services.AddRazorPages();            ❶

WebApplication app = builder.Build();

app.MapGet("/test", () => "Hello world!");   ❷
app.MapHealthChecks("/healthz");             ❸
app.MapRazorPages();                         ❹

app.Run();
```

A **route template** is a URL pattern that is used to match against request URLs, which are strings of fixed values, such as "/test" in the previous listing.
They can also contain placeholders for variables

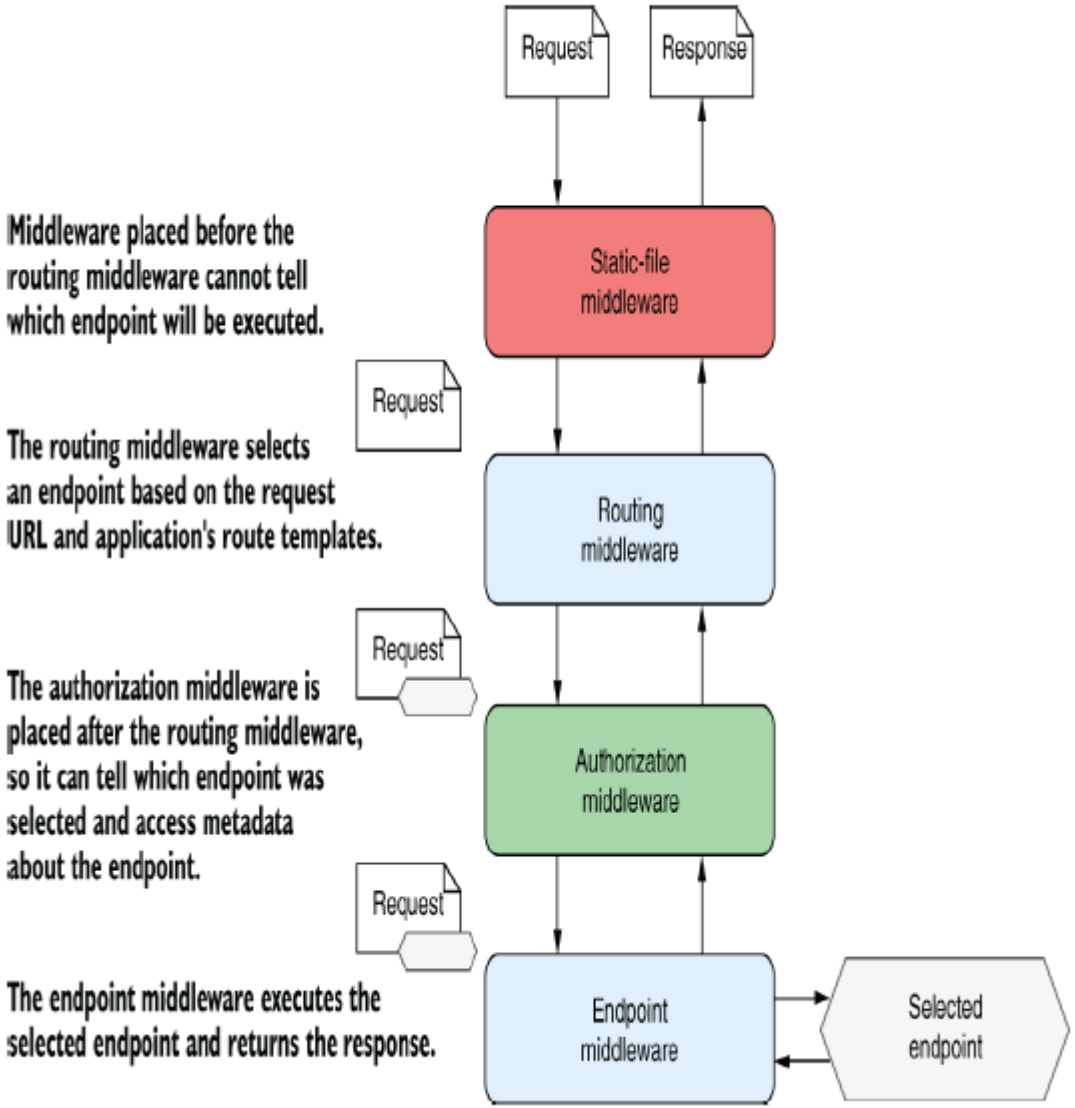
❶ Adds the services required by the health-check middleware and Razor Pages
❷ Registers a minimal API endpoint that returns “Hello World!” at the route /test
❸ Registers a health-check endpoint at the route /healthz
❹ Registers all the Razor Pages in your application as endpoints

# NOTE

- WebApplication **automatically** adds the RoutingMiddleware **to the start of the middleware** and EndpointMiddleware **to the end of** the middleware pipeline,
- Though you can override the location in the pipeline by calling UseRouting() or UseEndpoints().



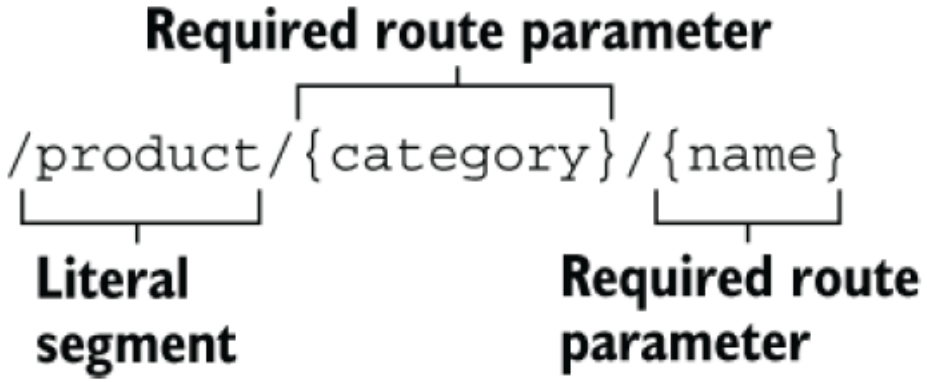

Only middleware placed **after** the RoutingMiddleware can detect which endpoint is going to be executed.



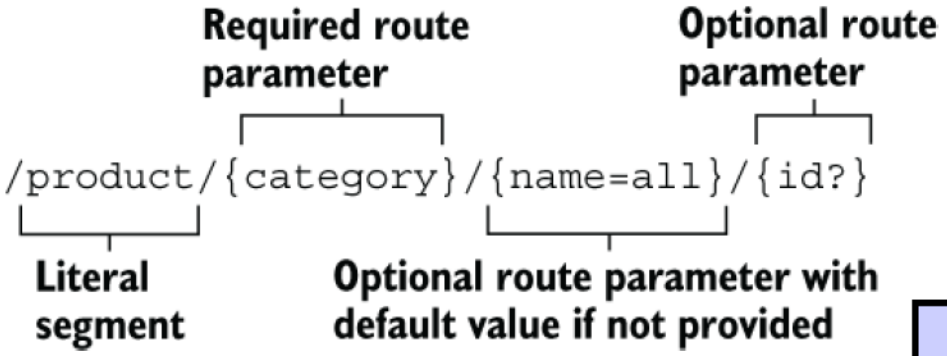

The AuthorizationMiddleware needs to know which endpoint will be executed, so it must be placed after the RoutingMiddleware and **before** the EndpointMiddleware in your middleware pipeline.

# WORKING WITH PARAMETERS AND LITERAL SEGMENTS

- A segment is typically separated by the / character, but it can be any valid character. Each segment is either
- **A literal value**
  - must be matched exactly (ignoring case) by the request URL
- **A route parameter**
  - may vary, but will still be a match for the template.



# USING OPTIONAL AND DEFAULT VALUES




| URL | Route values |
|---|---|
| /product/shoes/formal/3 | category=shoes, name=formal, id=3 |
| /product/shoes/formal | category=shoes, name=formal |
| /product/shoes | category=shoes, name=all |
| /product/bags/satchels | category=bags, name=satchels |
| /product/phones | category=phones, name=all |
| /product/computers/laptops/ABC-123 | category=computers, name=laptops, id=ABC-123 |

# ADDING ADDITIONAL CONSTRAINTS TO ROUTE PARAMETERS

- You can define constraints in a route template for a given route parameter by using **:** (colon).
- {id:int}, for example, would add the IntRouteConstraint to the id parameter.
  - For a given URL to be **considered a match**, the value assigned to the id route value must be convertible to an integer.

# ADDING ADDITIONAL CONSTRAINTS TO ROUTE PARAMETERS

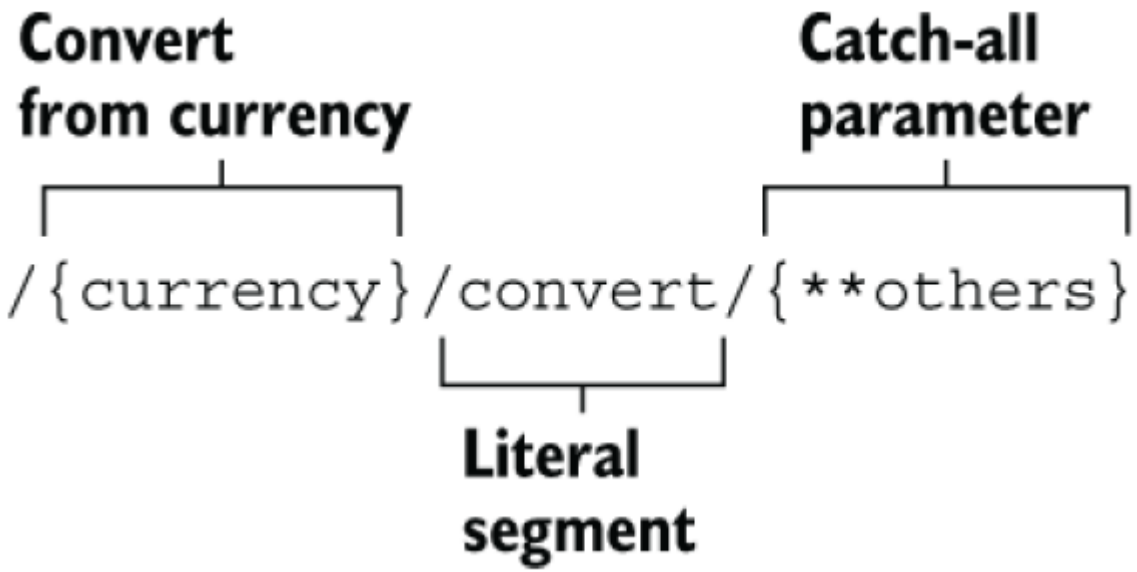
| Constraint | Example | Description | Match examples |
|---|---|---|---|
| `int` | `{qty:int}` | Matches any `integer` | `123, -123, 0` |
| `Guid` | `{id:guid}` | Matches any `Guid` | `d071b70c-a812-4b54-87d2-7769528e2814` |
| `decimal` | `{cost:decimal}` | Matches any `decimal` value | `29.99, 52, -1.01` |
| `min(value)` | `{age:min(18)}` | Matches `integer` values of 18 or greater | `18, 20` |
| `length(value)` | `{name:length(6)}` | Matches `string` values with a length of `6` | `Andrew,123456` |
| `optional int` | `{qty:int?}` | Optionally matches any `integer` | `123, -123, 0, null` |
| `optional int max(value)` | `{qty:int:max(10)?}` | Optionally matches any `integer` of 10 or less | `3, -123, 0, null` |

When the routing middleware matches a URL to a route template, it interrogates the constraints to check that they're all valid. If they aren't valid, the route template isn't considered a match, and the endpoint won't be executed.

# MATCHING ARBITRARY URLS WITH THE CATCH-ALL PARAMETER

- These segments normally split around the slash character, /, so the route parameters themselves won't contain a slash.
- What do you do if you need them to contain a slash, or you don't know how many segments you're going to have?
- Here are some examples:
  - /USD/convert/GBP—Show USD with exchange rate to GBP
  - /USD/convert/GBP/EUR—Show USD with exchange rate to GBP and EUR
  - /USD/convert/GBP/EUR/CAD—Show USD with exchange rate for GBP, EUR, and CAD

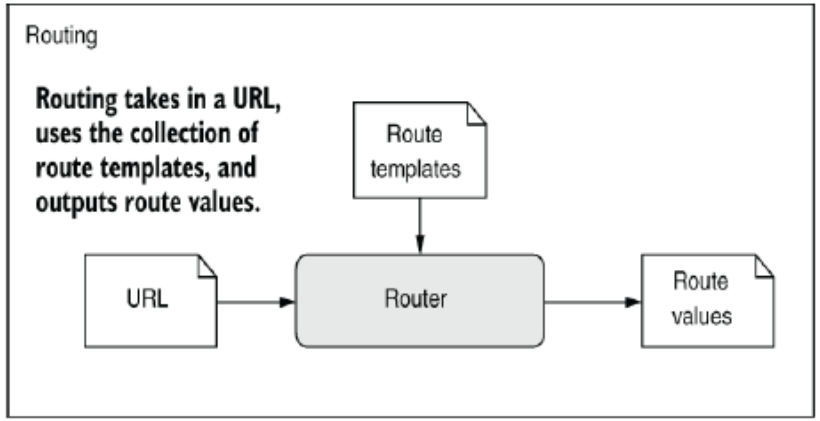
# MATCHING ARBITRARY URLS WITH THE CATCH-ALL PARAMETER

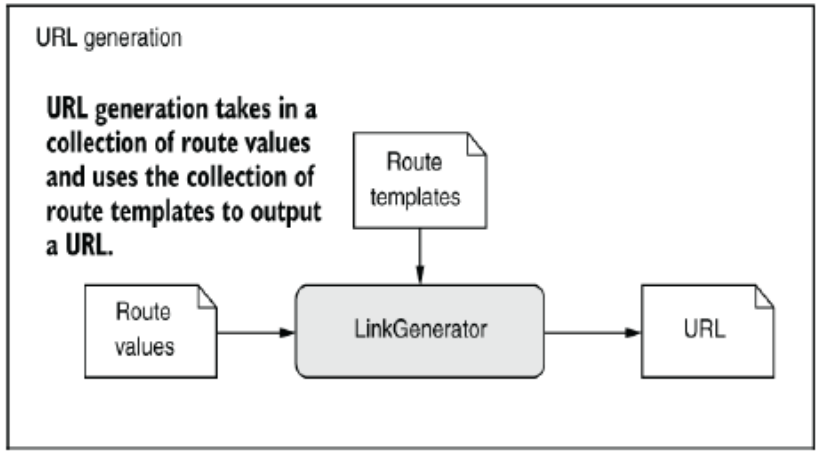


# MATCHING ARBITRARY URLS WITH THE CATCH-ALL PARAMETER

- Catch-all parameters can be declared using either one or two asterisks inside the parameter definition, like {*others} or {**others} .
- These will match the **remaining unmatched portion of a URL**, including any slashes or other characters that aren't part of earlier parameters.
- They can also match an empty string.
- For the USD/convert/GBP/EUR URL, the value of the route value others would be the single string "GBP/EUR".

# GENERATING URLS FROM ROUTE PARAMETERS




You can use the LinkGenerator class to generate URLs for your minimal APIs.

# GENERATING URLS FOR A MINIMAL API ENDPOINT WITH LINKGENERATOR

```
app.MapGet("/product/{name}", (string name) => $"The product is {name}")  ❶
    .WithName("product");                                                  ❷

app.MapGet("/links", (LinkGenerator links) =>                              ❸
{
    string link = links.GetPathByName("product",                           ❹
        new { name = "big-widget"});                                       ❹
    return $"View the product at {link}";                                  ❺
});
```

❶ The endpoint echoes the name it receives in the route template.

❷ Gives the endpoint a name by adding metadata to it

❸ References the LinkGenerator class in the endpoint handler

❹ Creates a link using the route name “product” and provides a value for the route parameter

❺ Returns the value “View the product at /product/big-widget”

The LinkGenerator is a service available anywhere in ASP.NET Core.

# GENERATING URLS FOR A MINIMAL API ENDPOINT WITH LINKGENERATOR

```
links.GetUriByName("product", new { Name = "super-fancy-widget"},
    "https", new HostString("localhost"));
```

Also, some methods available on LinkGenerator take an HttpContext. These methods are often easier to use in an endpoint handler, as they extract ambient values such as the scheme and hostname from the incoming request and reuse them for URL generation.

# GENERATING URLS FOR A MINIMAL API ENDPOINT WITH LINKGENERATOR

```
string link = links.GetPathByName("product", new { name = "big-widget"});
```

- If a selected route **explicitly** includes the defined route value in its definition, such as in the "/product/{name}" route template, the route value will be used in the URL path, resulting in **/product/big-widget**.
- If a route doesn't contain the route value explicitly, as in the "/product" template, the route value is appended to the **query string as additional data**. as in **/product?name=big-widget**.

# GENERATING URLS WITH IRESULTS

```
app.MapGet("/test", () => "Hello world!")
    .WithName("hello");                          ❶

app.MapGet("/redirect-me",
    () => Results.RedirectToRoute("hello"))      ❷
```

❶ Annotates the route with the name “hello”

❷ Generates a response that sends a redirect to the “hello” endpoint

- By default, RedirectToRoute() generates a **302 Found response** and includes the generated URL in the Location response header.

- You can redirect to an arbitrary URL by using the **Results.Redirect()** method.
  - takes **a URL instead of a route name** and can be useful for redirecting to external URLs

# CONTROLLING YOUR GENERATED URLS WITH ROUTEOPTIONS

```
WebApplicationBuilder builder = WebApplication.CreateBuilder(args);
builder.Services.Configure<RouteOptions>(o =>       ❶
{                                                   ❶
    o.LowercaseUrls = true;                         ❶
    o.AppendTrailingSlash = true;                   ❶
    o.LowercaseQueryStrings = false;                ❷
});

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

```
app.MapGet("/HealthCheck", () => Results.Ok()).WithName("healthcheck");
app.MapGet("/{name}", (string name) => name).WithName("product");

app.MapGet("/", (LinkGenerator links) =>
new []
{
    links.GetPathByName("healthcheck"),             ❸
    links.GetPathByName("product",                  ❹
        new { Name = "Big-Widget", Q = "Test"})     ❹
});

app.Run();
```

❶ Configures the RouteOptions used for link generation
❷ All the settings default to false.
❸ Returns /healthcheck/
❹ Returns /big-widget/?Q=Test

# CONTROLLING YOUR GENERATED URLS WITH ROUTEOPTIONS

```
links.GetPathByName("healthcheck",
    options: new LinkOptions
    {
        LowercaseUrls = false,
        AppendTrailingSlash = false,
    });
```

would return the value /HealthCheck. Without the LinkOptions parameter, GetPathByName would return /healthcheck/.