Contents

[Explaining Routing: 2](#_Toc202512175)

[1. the origins, 2](#_Toc202512176)

[2. theory for use 2](#_Toc202512177)

[3. how to code in a standard/conventional way … 2](#_Toc202512178)

[…the following in .NET Core Web API and ASP.NET Core Razor Pages: Request Dispatcher which maps HTTP routes and verbs (GET, POST) to code. 2](#_Toc202512179)

[. 2](#_Toc202512180)

[1) ORIGINS — Where Does Request Dispatching Come From? 3](#_Toc202512181)

[roots in **Model-View-Controller (MVC)** patterns from Smalltalk (1970s) 3](#_Toc202512182)

[2) THEORY — Why Use a Request Dispatcher? 4](#_Toc202512183)

[Separation of Concerns 4](#_Toc202512184)

[Flexibility & Scalability 6](#_Toc202512185)

[Uniform Interface 6](#_Toc202512186)

[3) HOW TO CODE IT— Conventional Examples 6](#_Toc202512187)

[A. In .NET Core Web API (Using Controllers) 6](#_Toc202512188)

[Registering Routes (Program.cs) 6](#_Toc202512189)

[Routing with Attributes 6](#_Toc202512190)

[B. In Razor Pages 7](#_Toc202512191)

[Registering Razor Pages 7](#_Toc202512192)

[Razor Page File Structure 7](#_Toc202512193)

[Route Mapping by Convention 7](#_Toc202512194)

[Summary Table 8](#_Toc202512195)

# Explaining Routing:

# the origins,

# theory for use

# how to code in a standard/conventional way …

# …the following in .NET Core Web API and ASP.NET Core Razor Pages: Request Dispatcher which maps HTTP routes and verbs (GET, POST) to code.

**ChatGPT said:**

You're asking a deep and important question that gets at the heart of **routing and request processing** in ASP.NET Core — especially for **Web API** and **Razor Pages**. Let's explore it clearly in **three parts**, covering:

1. **Origins**
2. **Theory**
3. **How to code it conventionally**

# .

# ORIGINS — Where Does Request Dispatching Come From?

## roots in **Model-View-Controller (MVC)** patterns from Smalltalk (1970s)

The concept of **request dispatching** — mapping HTTP requests to code — has its roots in **Model-View-Controller (MVC)** patterns from Smalltalk (1970s), formalized in web frameworks like Ruby on Rails, Django, and ASP.NET MVC.

**🌐 In ASP.NET Core:**

* ASP.NET MVC (2009) introduced **attribute routing** and **convention-based routing**.
* ASP.NET Core (2016+) unified **MVC + Web API** under one model, and introduced **Razor Pages** for more page-focused apps.
* The **routing system** is part of **middleware**, handling request URLs and deciding which handler (Controller action or Razor Page) to invoke.

The dispatcher is essentially a **routing engine** that determines:

"When a request hits /api/books/1, which method (in which controller or Razor Page) should handle it?"

# THEORY — Why Use a Request Dispatcher?

### Separation of Concerns

The routing/dispatching system separates:

* HTTP protocol concerns (verbs, paths)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **🔍 "The routing/dispatching system separates HTTP protocol concerns (verbs, paths)"**  This means that the **routing system in ASP.NET Core** is responsible for **decoupling**  **low-level HTTP mechanics**  — like the **request method** and **URL** —  from the application’s core logic (controllers, handlers, etc.).  Let’s break it down:  **✅ 1. HTTP Protocol Concerns**  These are **technical details of an HTTP request**. Two key parts:   | **Term** | **Example** | **Explanation** | | --- | --- | --- | | **HTTP Verb (request method)** | GET, POST, PUT, DELETE | What action the client is performing | | **Path (URL Route)** | /api/students/5 | What resource the client is requesting |   These are **transport-level** concerns — about **how** the request is made, not what to do with it.  **✅ 2. Separation of Concerns**  In Clean Architecture or good system design, we want to **separate infrastructure from business logic**.  Routing allows us to say:  “If someone sends a **GET** request to /api/students/5:   * call this method * and let the method focus on **logic**, not on **HTTP mechanics**.”   So your method can focus on things like:   * Finding the student by ID * Returning it * Validating it * Not figuring out if it's a **GET** or **POST** or what the **path** is   **🧠 Why This Separation Matters**  Without routing:   * You’d have to parse HTTP manually * Your app logic would be tightly coupled to **request mechanics** * Testing or reusing code would be hard   Instead, in ASP.NET Core, **routing** **abstracts away** **HTTP details** and invokes:   * The correct **method** * On the correct **controller** * With the correct **parameters**   **🛠️ Example**  [HttpGet("students/{id}")]  public async Task<IActionResult> GetStudent(int id)  {  var student = await \_context.Students.FindAsync(id);  if (student == null) return NotFound();  return Ok(student);  }  This method doesn't care:   * That the request was made with **GET** * That the route was **/students/42**   The **dispatcher handles that**, using routing rules.  **🧾 Summary**  When we say:  "The routing/dispatching system separates HTTP protocol concerns (verbs, paths)",  We mean:   * The developer doesn’t have to manually inspect HTTP verbs or URLs. * ASP.NET Core's **routing middleware** does that. * This allows your app logic (controllers, Razor Pages) to focus purely on what needs to be done — not how the request was made. |

* Application logic (controllers, handlers)

This aligns with **Single Responsibility Principle (SRP)**.

### Flexibility & Scalability

* Developers can **map URLs to code** in clear, maintainable ways.
* Easily support RESTful APIs: /api/students/42, /api/students

### Uniform Interface

In REST, routing + verbs give a consistent way to **interact with resources**.

# HOW TO CODE IT— Conventional Examples

We'll break this into:

**✅ A. Web API (Controller-based)**

**✅ B. Razor Pages (Page-focused apps)**

## A. In .NET Core Web API (Using Controllers)

### Registering Routes (Program.cs)

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers(); // Enables attribute routing

var app = builder.Build();

app.MapControllers(); // Hook up controller endpoints

app.Run();

### Routing with Attributes

[ApiController]

[Route("api/[controller]")]

public class BooksController : ControllerBase

{

[HttpGet] // Matches GET api/books

public IActionResult GetAllBooks() => Ok(...);

[HttpGet("{id}")] // Matches GET api/books/5

public IActionResult GetBook(int id) => Ok(...);

[HttpPost] // Matches POST api/books

public IActionResult CreateBook([FromBody] BookDto book) => Created(...);

}

💡 The [HttpGet], [HttpPost], etc. are how .NET maps **verbs to methods**.

## B. In Razor Pages

### Registering Razor Pages

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages(); // Enables Razor Page routing

var app = builder.Build();

app.MapRazorPages(); // Hook up Razor Pages

app.Run();

### Razor Page File Structure

📁 Pages/Students/Details.cshtml  
📁 Pages/Students/Details.cshtml.cs

### Route Mapping by Convention

public class DetailsModel : PageModel

{

public Student Student { get; set; }

public async Task<IActionResult> OnGetAsync(int id)

{

Student = await \_context.Students.FindAsync(id);

return Page();

}

}

* Navigating to /Students/Details?id=3 runs OnGetAsync
* You can also define custom routes:

[BindProperty(SupportsGet = true)]

public int Id { get; set; }

[HttpGet("/students/details/{id}")]

public IActionResult OnGet(int id) => ...

## Summary Table

| **Pattern** | **Dispatcher Component** | **Routing Style** | **Typical Use** |
| --- | --- | --- | --- |
| **Web API** | Controller with [HttpGet], etc. | Attribute routing | RESTful APIs |
| **Razor Pages** | PageModel’s OnGet, OnPost, etc. | File/folder convention or attribute | UI-centric apps |

**🧠 Key Insight**

In both models, ASP.NET Core uses a **middleware pipeline** with a **routing system** that maps **HTTP requests to method invocations**. The dispatcher is highly configurable but designed to be simple and declarative.