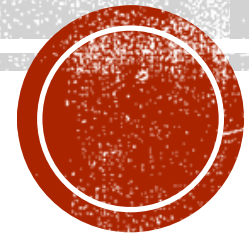


CREATING A WEBSITE WITH RAZOR PAGES

Kamal Beydoun

Lebanese University – Faculty of Sciences I

Kamal.beydoun@ul.edu.lb



YOUR FIRST RAZOR PAGES APPLICATION


Ensure that .NET 7.0 is selected.

Ensure that Authentication Type is set to None.

Ensure that HTTPS is checked and Enable Docker is unchecked.

Ensure that Do Not Use Top-level Statements is unchecked.

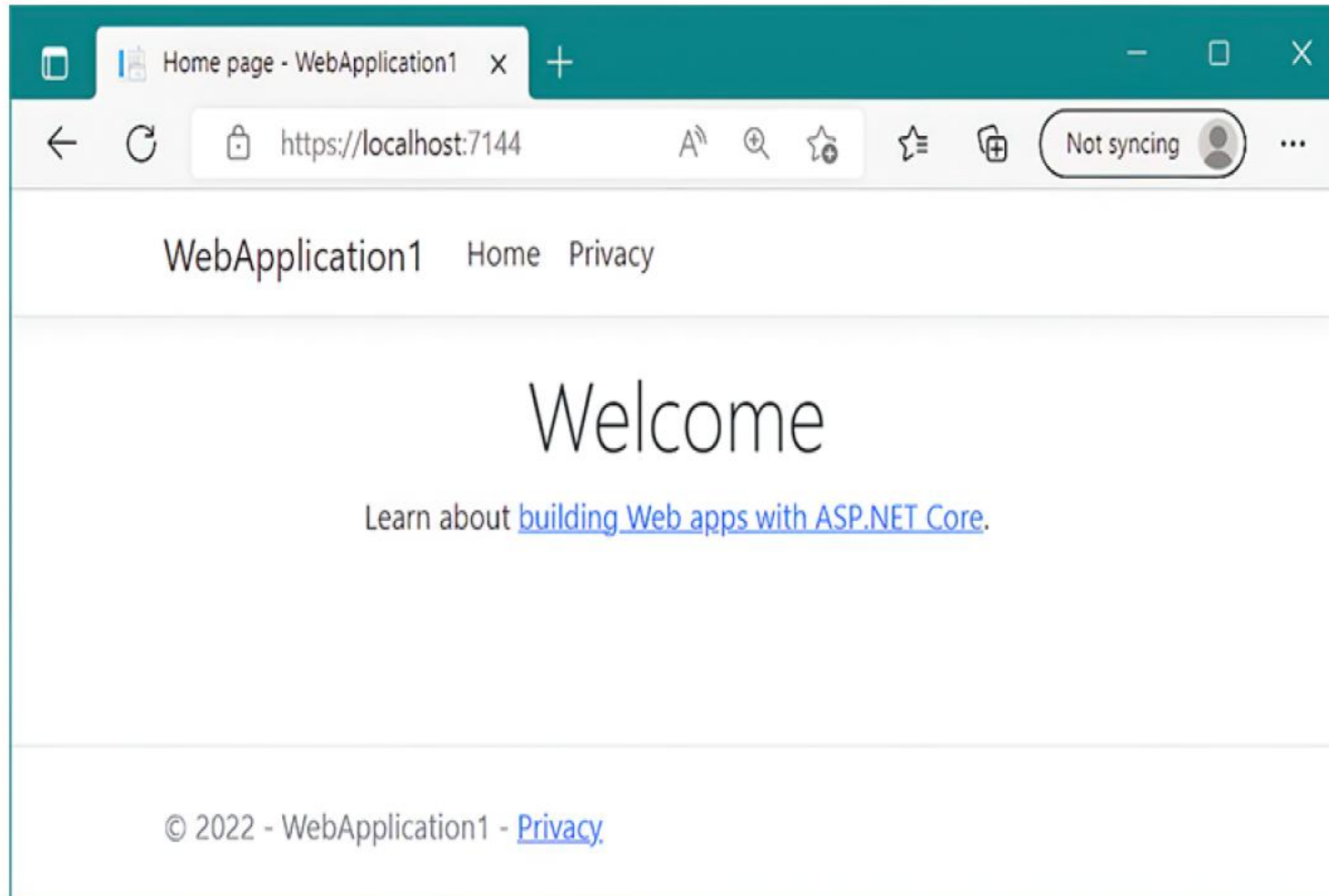
Click Create to generate the app from the selected template.



```
dotnet new sln -n WebApplication1 ❶  
dotnet new razor -o WebApplication1 ❷  
dotnet sln add WebApplication1 ❸
```

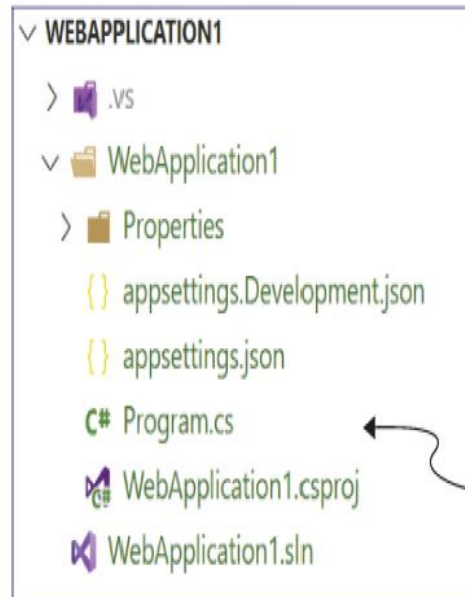
- ❶ Creates a solution file called WebApplication1 in the current folder
- ❷ Creates an ASP.NET Core Razor Pages project in a subfolder, WebApplication1
- ❸ Adds the new project to the solution file

YOUR FIRST RAZOR PAGES APPLICATION



YOUR FIRST RAZOR PAGES APPLICATION

Folder view for
minimal API application

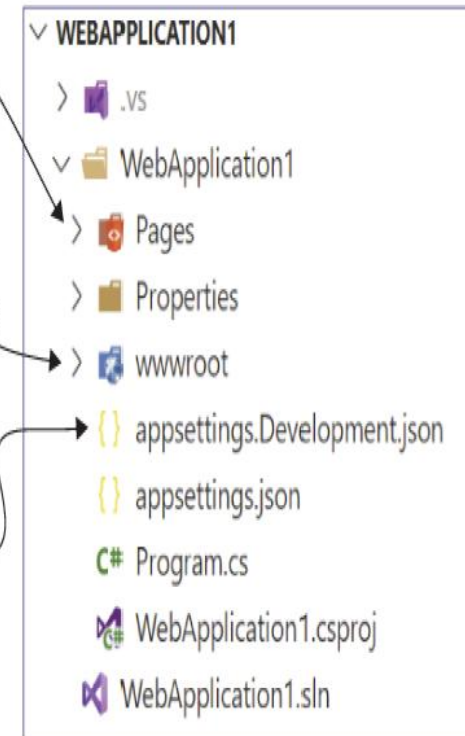


The Pages folder contains
the Razor Page files.

The wwwroot folder contains
static files like css, js, and
html files that are served
as is to the browser.

The majority of the app
structure is the same in
both cases.

Folder view for
Razor Pages app



ADDING AND CONFIGURING SERVICES

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

builder.Services.AddRazorPages(); ❶

WebApplication app = builder.Build();

if (!app.Environment.IsDevelopment()) ❷
{
    app.UseExceptionHandler("/Error"); ❷
    app.UseHsts() ❷
}

app.UseHttpsRedirection(); ❸
app.UseStaticFiles(); ❸
app.UseRouting(); ❸
app.UseAuthorization(); ❸

app.MapRazorPages(); ❹

app.Run();
```

- ❶ Registers the required services to use the Razor Pages feature
- ❷ Conditionally adds middleware depending on the runtime environment
- ❸ Additional middleware can be added to the middleware pipeline.
- ❹ Registers each Razor Page as an endpoint in your application

GENERATING HTML WITH RAZOR PAGES

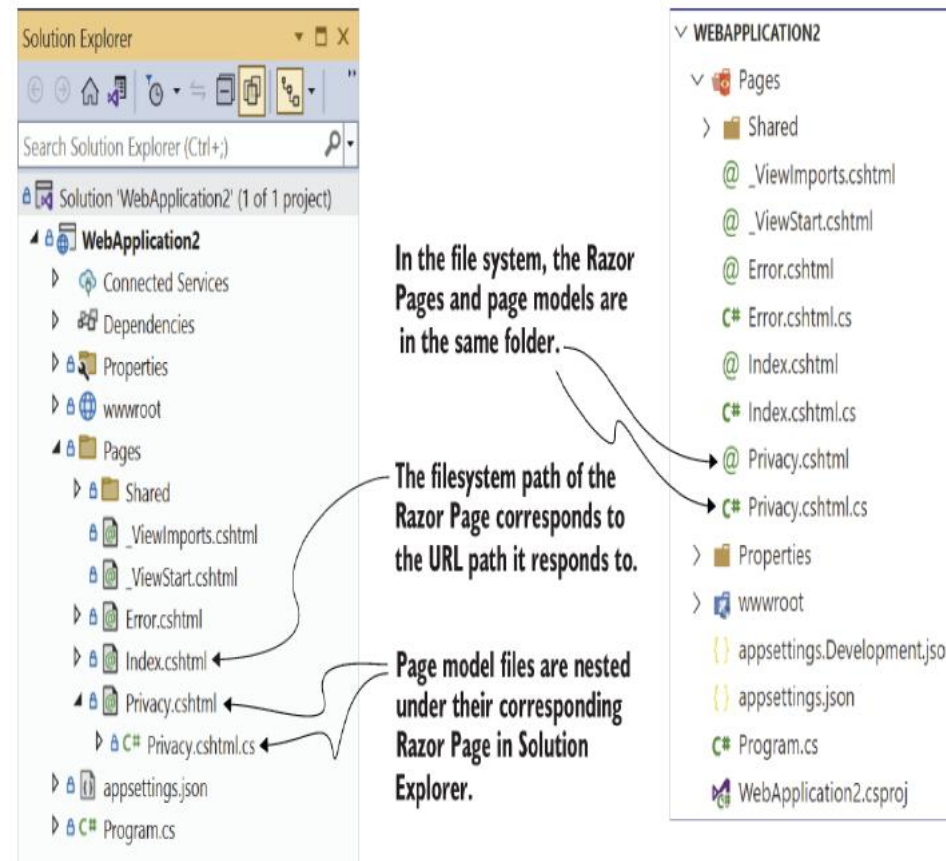
- Razor Pages are stored in .cshtml files (a hybrid of .cs and .xhtml) within the Pages folder of your project.
- In general, the routing middleware maps request URL paths to a **single Razor Page** by looking in the Pages folder of your project for a Razor Page with the same path.

```
@page ❶
@model PrivacyModel ❷
@{
    ViewData["Title"] = "Privacy Policy"; ❸
}
<h1>@ViewData["Title"]</h1> ❹

<p>Use this page to detail your site's privacy policy.</p> ❺
```

- ❶ Indicates that this is a Razor Page
- ❷ Links the Razor Page to a specific PageModel
- ❸ C# code that doesn't write to the response
- ❹ HTML with dynamic C# values written to the response
- ❺ Standalone, static HTML

GENERATING HTML WITH RAZOR PAGES



GENERATING HTML WITH RAZOR PAGES

- Razor Pages have the concept of **layouts**, which are base templates that define the common elements of your application, such as **headers and footers**.
- The HTML of the layout **combines** with the Razor Page template to produce the final HTML that's sent to the browser.
- Layouts prevent you from having to duplicate code for the header and footer in every page, and mean that if you need to tweak something, you'll need to do it in only one place.

HANDLING REQUEST LOGIC WITH PAGE MODELS AND HANDLERS

- The **@page** directive in a .cshtml file marks the page as a Razor Page, but most Razor Pages also have an **associated page model**.
- By convention, this page model is placed in a file commonly known as **a code-behind file** that has a .cs extension.
- Page models **should** derive from the PageModel base class, and they typically contain one or more methods called **page handlers** that define how to handle requests to the Razor Page

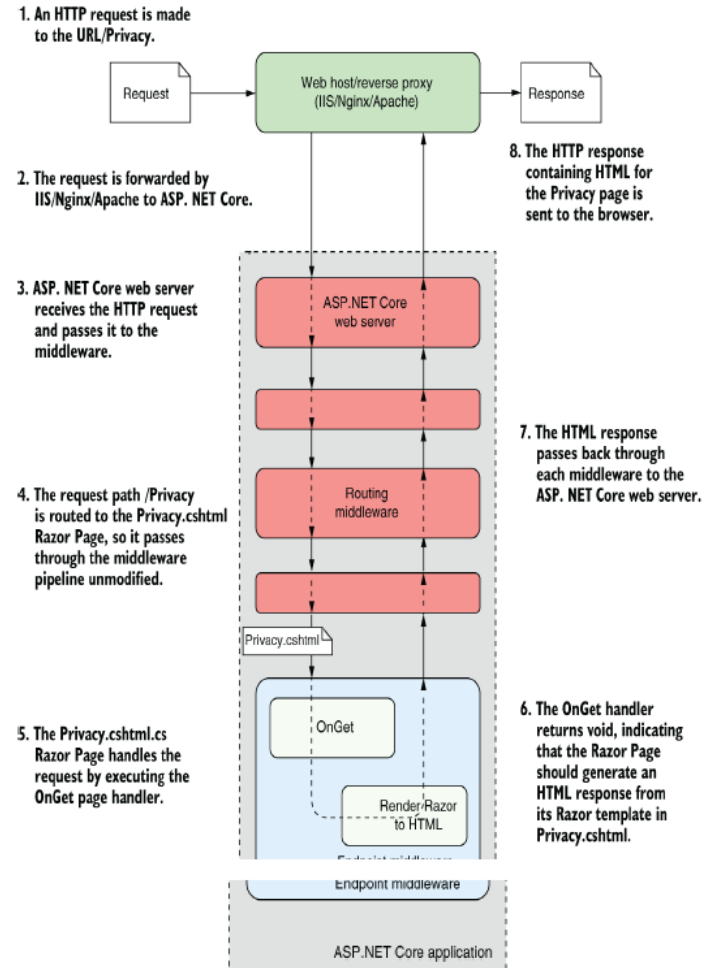
HANDLING REQUEST LOGIC WITH PAGE MODELS AND HANDLERS

```
public class PrivacyModel: PageModel ❶
{
    private readonly ILogger<PrivacyModel> _logger; ❷
    public PrivacyModel(ILogger<PrivacyModel> logger) ❷
    {
        _logger = logger; ❷
    } ❷

    public void OnGet() ❸
    {
    }
}
```

- ❶ Razor Pages must inherit from PageModel.
- ❷ You can use dependency injection to provide services in the constructor.
- ❸ The default page handler is OnGet. Returning void indicates HTML should be generated.

HANDLING REQUEST LOGIC WITH PAGE MODELS AND HANDLERS



EXPLORING A TYPICAL RAZOR PAGE

```
public class CategoryModel : PageModel
{
    private readonly ToDoService _service;
    public CategoryModel(ToDoService service)
    {
        _service = service;
    }

    public ActionResult OnGet(string category)
    {
        Items = _service.GetItemsForCategory(category);
        return Page();
    }
}
```

```
    public List<ToDoListModel> Items { get; set; }
}
```

- ❶ The `ToDoService` is provided in the model constructor using DI.
- ❷ The `OnGet` handler takes a parameter, `category`.
- ❸ The handler calls out to the `ToDoService` to retrieve data and sets the `Items` property.
- ❹ Returns a `PageResult` indicating the Razor view should be rendered
- ❺ The Razor View can access the `Items` property when it's rendered.

UNDERSTANDING THE MVC DESIGN PATTERN

■ Ebook

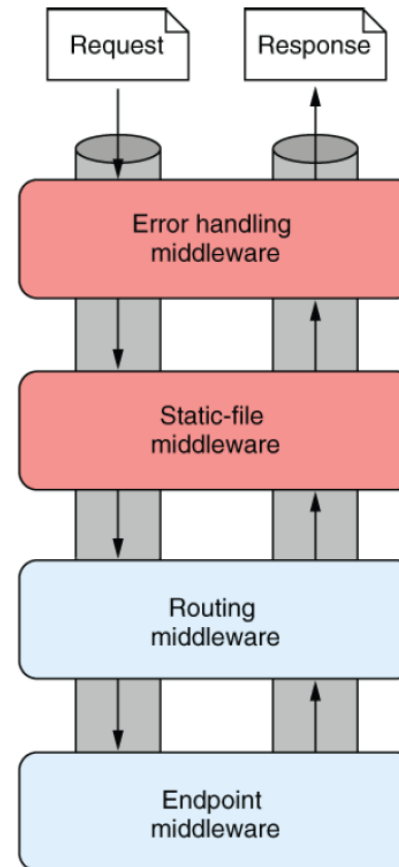
APPLYING THE MVC DESIGN PATTERN TO RAZOR PAGES

The request passes through each middleware in the pipeline.

Each middleware gets an opportunity to handle the request.

The routing middleware attempts to find an endpoint that will handle the request.

The endpoint middleware is the last in the pipeline. The MVC pattern is implemented entirely by individual Razor Page endpoints.



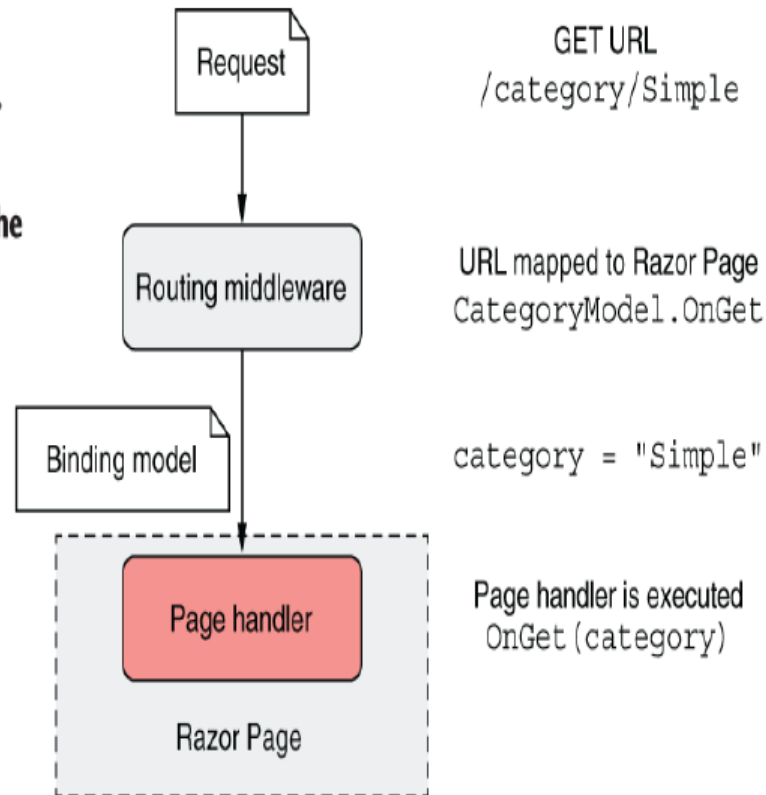
DIRECTING A REQUEST TO A RAZOR PAGE AND BUILDING A BINDING MODEL

1. A request is received and passes through the middleware pipeline.

2. The routing middleware directs the request to a specific Razor Page and page handler.

3. A binding model is built from the details provided in the request.

4. The Razor Page is passed the binding model, and the page handler method is executed.



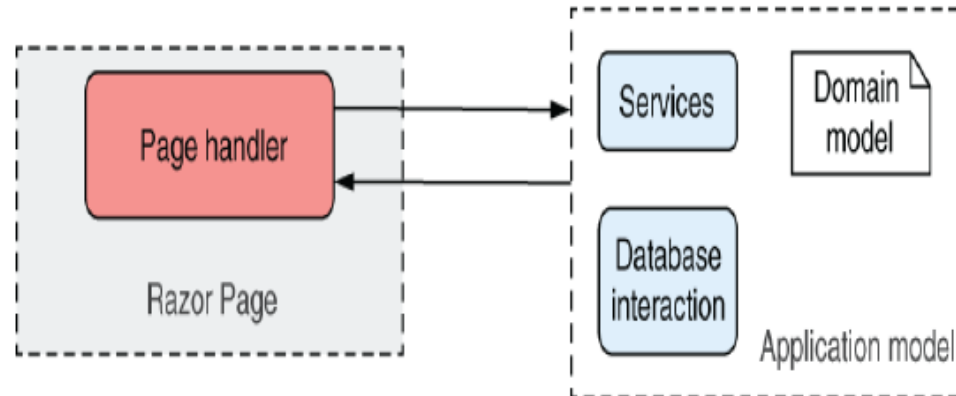
EXECUTING A HANDLER USING THE APPLICATION MODEL

It should perform only a limited number of actions :

- **Validate** that the data contained in the binding model is valid for the request.
- Invoke the appropriate actions on the application model using services.
- Select an appropriate response to generate based on the response from the application model.

EXECUTING A HANDLER USING THE APPLICATION MODEL

1. The page handler uses the category provided in the binding model to determine which method to invoke in the application model.
2. The page handler method calls into services that make up the application model. This might use the domain model to determine whether to include completed ToDo items, for example.
3. The services load the details of the ToDo items from the database and return them to the action method.



BUILDING HTML USING THE VIEW MODEL

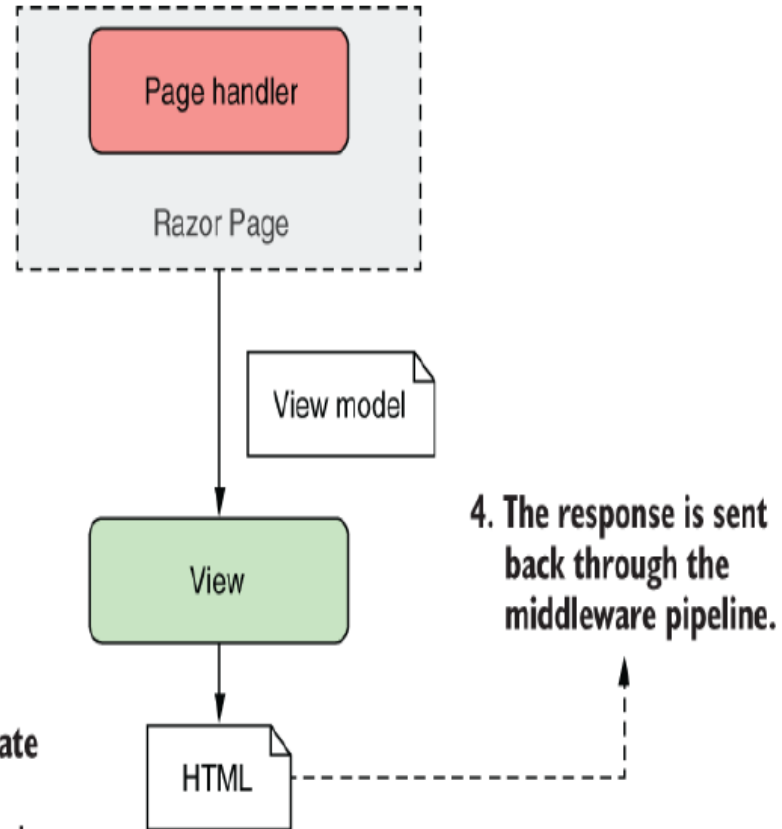
- When the page handler has called out to the application model that contains the application business logic, it's time to generate a response.
- A view model captures the details necessary for the Razor view to generate a response.
- Razor Pages use the **PageModel class itself** as the **view model** for the Razor view by exposing the required data as properties.
- The Razor view uses the **data exposed** (properties) in the page model to generate the final HTML response.

BUILDING HTML USING THE VIEW MODEL

1. The page handler builds a view model from the data provided by the application model by setting properties on the PageModel.

2. The page handler indicates that a view should be rendered.

3. The Razor view uses the provided view model to generate an HTML response containing the details of the Todos to display.



4. The response is sent back through the middleware pipeline.

PUTTING IT ALL TOGETHER: A COMPLETE RAZOR PAGE REQUEST

