Introduction to ASP.NET Core 9 & MVC Fundamentals

Day 1

Agenda for Day 1

- ASP.NET Core 9 Overview
- Introduction to MVC Pattern
- Project Setup & MVC Basics
- Controllers, Actions, Action Results
- Views: Razor Syntax, Layouts
- Models & Data Passing to Views
- Routing
- Hands-on Demo & Lab Assignment

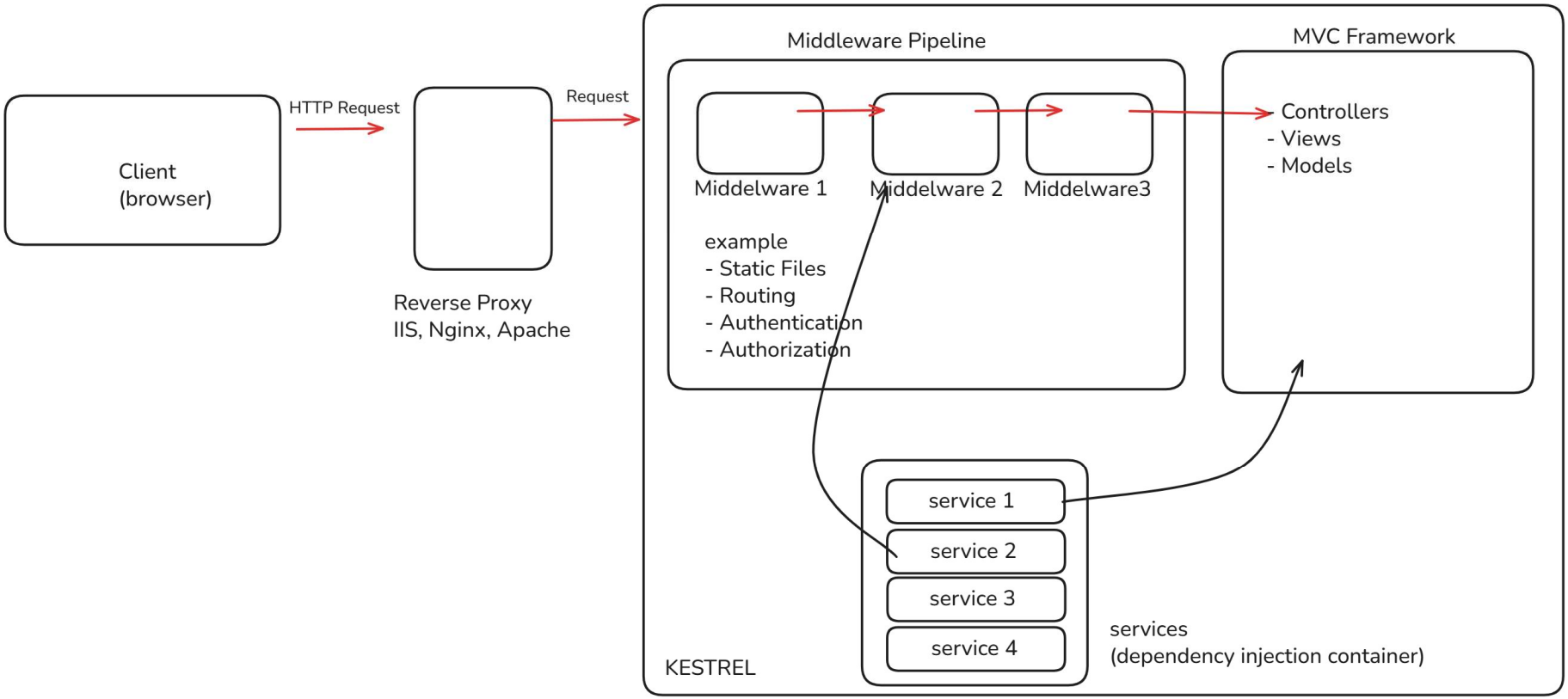
What is ASP.NET Core 9?

- Open-source, cross-platform framework for building modern web applications.
- Runs on .NET 9 (latest version).
- Key features: High performance, cross-platform, unified MVC and Web API, built-in DI, Kestrel web server.

ASP.NET Core Architecture Review

- **Middleware**: Request processing pipeline. Each middleware can process the request, pass it to the next, or short-circuit.
- **Dependency Injection** (DI): Built-in container for managing service dependencies.
- **Hosting**: Kestrel (default web server), can be hosted behind IIS, Nginx, Apache.

ASP.NET Core Architecture



Introduction to MVC Pattern

- **Model-View-Controller (MVC)**: An architectural pattern that separates an application into three main logical components.
- **Benefits**: Separation of Concerns, Testability, Maintainability, Parallel Development.

Roles in MVC

- **Model**:
  - Represents the application's data and business logic.
  - Responsible for data storage, retrieval, and manipulation.
  - Often simple POCOs (Plain Old C# Objects).
- **View**:
  - Responsible for displaying the user interface.
  - Presents data from the Model to the user.
  - In ASP.NET Core, typically Razor (`.cshtml`) files.
- **Controller**:
  - Handles user input and interactions.
  - Interacts with the Model to retrieve/update data.
  - Selects the appropriate View to display.

Project Setup - Visual Studio

- Demo: File -> New -> Project -> "ASP.NET Core Web App (Model-View-Controller)“
- Key files: `Program.cs`, `appsettings.json`, `Controllers` folder, `Views` folder, `Models` folder, `wwwroot` folder.

Project Setup - .NET CLI

- `dotnet new mvc -n MyMvcApp`
- `cd MyMvcApp`
- `dotnet run`
- Cross-platform development.

Controllers

- Classes that handle incoming HTTP requests.
- Inherit from `Microsoft.AspNetCore.Mvc.Controller`.
- **Actions**: Public methods within controllers that respond to specific routes.
- **Action Results**: Methods that return specific types of responses (e.g., `View()`, `RedirectToAction()`, `NotFound()`).

Views - Razor Syntax

- **Razor**: A markup syntax for embedding server-side C# code into HTML.
- **Key Syntax**:
  - `@`: Single C# expression or statement.
  - `@model`: Declares the type of the model passed to the view.
  - `@{ ... }`: Code block.
  - `@()`: Explicit code expression.
  - `@Html.DisplayNameFor()` , `Html.DisplayFor()` : HTML Helpers.

Views - Layouts and Sections

- **Layouts** (**`_Layout.cshtml`**): Defines a common structure for all pages (header, footer, navigation).
- **`@RenderBody()`**: Placeholder for the content of individual views.
- **`@RenderSection()`**: Defines optional content areas that views can fill.
- **`_ViewImports.cshtml`**: Specifies common namespaces and Tag Helpers to be imported into all views.

Models (in MVC context)

- Simple C# classes (POCOs) that represent the data used by the application.
- Can be used for:
  - Displaying data in views.
  - Receiving data from forms.
  - Interacting with data storage (e.g., Entity Framework).

Routing - Conventional Routing

- **Default Route**: Defined in `Program.cs` using `MapControllerRoute`.
- pattern: "`{controller=Home}/{action=Index}/{id?}`"
  - `{controller}`: Maps to controller name (e.g., `HomeController`).
  - `{action}`: Maps to action method name (e.g., `Index()`).
  - `{id?}`: Optional parameter, often used for IDs.

Data Passing to Views

- **Strongly-typed Models (`@model`):**
  - Pass a C# object directly to the `View()` method.
  - Provides compile-time checking and IntelliSense. (Recommended)
- **`ViewData`**:
  - A dictionary-like object (`ViewData["Message"] = "Hello";`).
  - Requires casting in the view.
- **`ViewBag`**:
  - A dynamic property bag (`ViewBag.Message = "Hello";`).
  - No compile-time checking.

Assignment

- **Objective**: Create a new ASP.NET Core 9 MVC project and implement a simple application to manage a list of `Movie` entities using in-memory data.
- **Steps**:
  1. Project Setup (VS or .NET CLI).
  2. Create `Movie` Model (`Id`, `Title`, `Director`, `ReleaseYear`).
  3. Create `MovieController` with in-memory list.
  4. Implement `Index()` action to display all movies.
  5. Implement `Details(int id)` action to display single movie details.
  6. Create `Index.cshtml` and `Details.cshtml` views.
  7. Test the application by navigating through the pages.