**Unit 4: Creating ASP.NET Core MVC Applications**

**1. Setting Up the Environment**

**Description:**

* Setting up the environment involves preparing tools and software for developing ASP.NET Core MVC applications.
* Includes installing Visual Studio, .NET SDK, and creating a new project.

**Syntax:**

1. **Creating a New Project**:

dotnet new mvc -n MyMvcApp

**Uses:**

* Prepares a development environment with all required components.
* Allows smooth execution and debugging of ASP.NET Core MVC applications.

**Example:**

1. Install Visual Studio.
2. Select the **ASP.NET Core Web App (Model-View-Controller)** template.
3. Configure the project name and framework.

**2. Controllers and Actions**

**Controllers in ASP.NET Core MVC**

**Description**

Controllers are the central components of an ASP.NET Core MVC application. They handle user input, process requests, and return responses. Controllers interact with models to retrieve or manipulate data and then pass the data to views for rendering.

**Syntax**

1. **Basic Structure**:

using Microsoft.AspNetCore.Mvc;

namespace MyMvcApp.Controllers

{

public class ControllerName : Controller

{

// Action Methods

public IActionResult ActionName()

{

// Logic

return View();

}

}

}

1. **Key Points**:
   * Controllers typically inherit from Controller, but it's not mandatory.
   * They contain methods called **Actions** that handle specific HTTP requests.

**Types of Controllers**

1. **Standard MVC Controllers**:
   * Handle requests and return views or data.
   * Example: HomeController, ProductController.
2. **API Controllers**:
   * Used for building RESTful APIs.
   * Annotated with [ApiController] attribute.
   * Example: WeatherController.
3. **Attribute-Routed Controllers**:
   * Use attributes to define custom routes for actions.
   * Example:

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

public class ProductController : Controller

{

[HttpGet("{id}")]

public IActionResult GetProduct(int id)

{

return Json(new { Id = id, Name = "Sample Product" });

}

}

**Example Program**

**Basic MVC Controller Example**

1. **Controller (HomeController.cs)**:

using Microsoft.AspNetCore.Mvc;

namespace MyMvcApp.Controllers

{

public class HomeController : Controller

{

public IActionResult Index()

{

return View();

}

public IActionResult About()

{

ViewData["Message"] = "Welcome to the About Page.";

return View();

}

}

}

1. **Views**:
   * Views/Home/Index.cshtml:

<h1>Welcome to the Home Page</h1>

* + Views/Home/About.cshtml:

<h1>@ViewData["Message"]</h1>

1. **Startup.cs Configuration**: Ensure the default route is set in Program.cs or Startup.cs:

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

});

**API Controller Example**

1. **Controller (ProductController.cs)**:

using Microsoft.AspNetCore.Mvc;

[ApiController]

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

public class ProductController : ControllerBase

{

[HttpGet("{id}")]

public IActionResult GetProduct(int id)

{

var product = new { Id = id, Name = "Laptop", Price = 1000 };

return Ok(product);

}

}

1. **Test the API**:
   * Access https://localhost:5001/api/product/1.
   * Response:

json

Copy code

{

"Id": 1,

"Name": "Laptop",

"Price": 1000

}

**Use Cases of Controllers**

1. **Routing Requests**:
   * Direct HTTP requests to appropriate actions.
   * Example: /Home/Index routes to Index action in HomeController.
2. **Handling Business Logic**:
   * Process data or interact with the model layer.
   * Example: Fetching data from a database.
3. **Returning Views or Data**:
   * Return HTML views for user interfaces.
   * Example: return View();
   * Return JSON for API responses.
   * Example: return Json(data);
4. **Redirecting**:
   * Redirect users to another action or external URL.
   * Example: return RedirectToAction("About");

**Practical Guide**

**Creating a Controller for a Product Catalog**

1. **Controller**:

public class ProductController : Controller

{

public IActionResult Index()

{

var products = new List<string> { "Laptop", "Phone", "Tablet" };

return View(products);

}

public IActionResult Details(int id)

{

var product = $"Product {id}";

return View("Details", product);

}

}

1. **View for Index**:
   * Views/Product/Index.cshtml:

<h1>Product List</h1>

<ul>

@foreach (var product in Model)

{

<li>@product</li>

}

</ul>

1. **View for Details**:
   * Views/Product/Details.cshtml:

<h1>@Model</h1>

1. **Routes**:
   * /Product/Index: Displays the product list.
   * /Product/Details/1: Displays details for product 1.

Here’s the complete implementation of the **Product Catalog** example:

**1. Using Directives**

using Microsoft.AspNetCore.Mvc; // Provides classes for MVC pattern and web app development

using System.Collections.Generic; // Provides the List<T> class for handling collections

* **Microsoft.AspNetCore.Mvc**:
  + Contains core functionalities for building MVC applications.
  + Provides Controller, IActionResult, JsonResult, etc.
* **System.Collections.Generic**:
  + Provides the List<T> collection used to handle product lists (List<string>).

**2. Controller Class**

public class ProductController : Controller

* **Controller**:
  + Base class for all MVC controllers.
  + Provides methods like View(), RedirectToAction(), and Json() for responding to HTTP requests.
  + Helps in routing and returning views or data in a structured way.

**3. Action Methods**

**Example:**

public IActionResult Index()

{

var products = new List<string> { "Laptop", "Phone", "Tablet" };

return View(products);

}

* **public**:
  + The method must be public to act as an action. Non-public methods (e.g., private) are not exposed as endpoints.
* **IActionResult**:
  + The return type of an action method.
  + Represents different types of responses (e.g., ViewResult, JsonResult, RedirectResult).
* **Index**:
  + By convention, the Index action is used as the default action for a controller when no specific action is provided in the URL.
* **View(products)**:
  + Returns a ViewResult with the products list passed as the model to the view.

**4. View Engine and Razor Syntax**

**View Example: Index.cshtml**

@model List<string> <!-- Specifies the model type passed to the view -->

<h1>Product List</h1>

<ul>

@foreach (var product in Model)

{

<li>@product</li> <!-- Renders each product as a list item -->

}

</ul>

* **@model**:
  + Declares the type of data the view expects.
  + In this case, List<string> specifies that the view receives a list of strings as its model.
* **@foreach**:
  + Razor syntax for looping through a collection (Model in this case, which is a List<string>).
  + Dynamically renders each product in the Model list.
* **@product**:
  + Injects the value of the current product from the loop into the HTML.

**5. Routing**

**Default Routing Configuration (Program.cs):**

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

});

* **endpoints.MapControllerRoute**:
  + Defines a route pattern for matching incoming HTTP requests to controllers and actions.
* **Default Pattern**: {controller=Home}/{action=Index}/{id?}
  + **{controller=Home}**: Routes to the HomeController by default if no controller is specified.
  + **{action=Index}**: Calls the Index action by default if no action is specified.
  + **{id?}**: Optional parameter used for passing additional information (e.g., product ID).

**6. Built-in Methods**

ASP.NET Core MVC provides several predefined methods in the Controller class. Here are the ones used in the example:

| **Method** | **Description** |
| --- | --- |
| View() | Returns a view (HTML page) to the client. |
| View(object model) | Passes a model (data) to the view for rendering. |
| Json(object data) | Returns JSON data to the client (used in APIs). |
| RedirectToAction | Redirects the user to a different action within the same or another controller. |

**7. Razor Pages Structure**

Razor Pages use the .cshtml files to structure views and bind them to the data passed from controllers.

**Razor Syntax Features:**

1. **@Model**:
   * Accesses the data passed from the controller.
2. **Loops and Conditions**:
   * Use @foreach, @if, and other constructs to dynamically generate HTML.
3. **Dynamic Content Injection**:
   * Use @ to inject server-side content directly into HTML.

**8. Program.cs**

**Code:**

var builder = WebApplication.CreateBuilder(args);

// Adds support for controllers and views

builder.Services.AddControllersWithViews();

var app = builder.Build();

// Middleware pipeline

app.UseStaticFiles(); // Serves static files like CSS, JS, images

app.UseRouting(); // Enables routing for incoming requests

// Maps incoming requests to controllers

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

});

// Runs the application

app.Run();

**Key Concepts:**

1. **Middleware**:
   * UseStaticFiles(): Serves static files.
   * UseRouting(): Enables request routing.
2. **Route Mapping**:
   * MapControllerRoute: Maps incoming URLs to controller actions.
3. **Application Start**:
   * app.Run(): Starts the application and listens for incoming HTTP requests.

**How Predefined Code Supports the Example**

1. **Controller Inheritance**:
   * Provides a base class with built-in functionality for handling HTTP requests and generating responses.
2. **Routing**:
   * Automatically maps URLs like /Product/Index to ProductController.Index.
3. **View Engine**:
   * Razor templates (.cshtml) are automatically located and rendered based on conventions (e.g., Views/Product/Index.cshtml).

**Full Directory Structure**

ProductCatalogApp/

├── Controllers/

│ ├── ProductController.cs

├── Views/

│ ├── Product/

│ │ ├── Index.cshtml

│ │ ├── Details.cshtml

├── Program.cs

**Key Takeaways**

* Controllers are pivotal in managing the application flow.
* They interact with models and views to process and display data.
* Flexible routing and action methods make controllers versatile for building MVC and API-based applications.

**Actions in ASP.NET Core MVC**

**Description**

Actions are methods inside a controller that handle incoming HTTP requests. They process the request, interact with the model, and return a response. Actions are the entry points for executing business logic and rendering views.

**Syntax**

public IActionResult ActionName()

{

// Logic

return View(); // Or other result types

}

**Key Points:**

1. **Action Methods**:
   * Must be public.
   * Return an IActionResult (or derived types like ViewResult, JsonResult, etc.).
2. **Action Names**:
   * By default, the method name is used as the action name in routes.
   * You can override the name using the [ActionName] attribute.

**Types of Action Results**

Actions return an **Action Result** that determines the response sent to the client. Common types:

| **Action Result** | **Description** | **Example Code** |
| --- | --- | --- |
| ViewResult | Renders a Razor view as HTML. | return View(); |
| JsonResult | Returns JSON data. | return Json(new { Name = "John" }); |
| ContentResult | Returns plain text or HTML content. | return Content("Hello World"); |
| RedirectResult | Redirects to a different action or URL. | return Redirect("/Home/About"); |
| RedirectToAction | Redirects to another action in the same or another controller. | return RedirectToAction("About"); |
| FileResult | Returns a file for download. | return File("path/to/file.pdf", "application/pdf"); |
| StatusCodeResult | Returns a specific HTTP status code. | return StatusCode(404); |
| EmptyResult | Returns no content (usually for void responses). | return new EmptyResult(); |

**Examples**

**1. Returning a View**

public IActionResult Index()

{

return View();

}

* If a view named Index.cshtml exists in the Views/Home folder, it will be rendered.

**2. Returning JSON**

public IActionResult GetProduct()

{

var product = new { Id = 1, Name = "Laptop", Price = 1200 };

return Json(product);

}

* This returns JSON:

{ "Id": 1, "Name": "Laptop", "Price": 1200 }

**3. Redirecting to Another Action**

public IActionResult RedirectToAbout()

{

return RedirectToAction("About");

}

**4. Returning Plain Text**

public IActionResult HelloWorld()

{

return Content("Hello, World!", "text/plain");

}

**5. Returning Files**

public IActionResult DownloadFile()

{

byte[] fileBytes = System.IO.File.ReadAllBytes("path/to/file.pdf");

return File(fileBytes, "application/pdf", "example.pdf");

}

**6. Returning Status Codes**

public IActionResult CustomNotFound()

{

return StatusCode(404, "Resource not found");

}

**Routing Actions**

Actions are mapped to URLs through routing. By default:

* URL structure: {controller}/{action}/{id?}
* Example:
  + /Home/Index → HomeController.Index
  + /Product/Details/1 → ProductController.Details(int id)

**Custom Route for an Action**

[Route("api/products/{id}")]

public IActionResult GetProduct(int id)

{

return Json(new { Id = id, Name = "Sample Product" });

}

* Access this via /api/products/1.

**Advanced Features**

**Action Parameters**

Actions can accept parameters directly from the route, query string, or form data.

public IActionResult ProductDetails(int id, string category)

{

return Content($"Product ID: {id}, Category: {category}");

}

* URL: /Home/ProductDetails?id=5&category=Electronics

**Using [HttpGet], [HttpPost], etc.**

Specify the HTTP method for an action:

[HttpGet]

public IActionResult GetProduct()

{

return View();

}

[HttpPost]

public IActionResult AddProduct(Product product)

{

// Add product logic

return RedirectToAction("Index");

}

**Example Program**

**HomeController.cs**

using Microsoft.AspNetCore.Mvc;

namespace MyMvcApp.Controllers

{

public class HomeController : Controller

{

public IActionResult Index()

{

return View(); // Returns Index.cshtml

}

[HttpGet]

public IActionResult Details(int id)

{

var product = new { Id = id, Name = "Laptop", Price = 1200 };

return Json(product); // Returns JSON response

}

[HttpPost]

public IActionResult CreateProduct(string name, double price)

{

// Example logic to create a product

return Content($"Product {name} with price {price} created!");

}

}

}

**View: Index.cshtml**

<h1>Welcome to the Home Page</h1>

<a href="/Home/Details/1">View Product Details</a>

**Practical Use of Actions**

1. **Web Applications**:
   * Render views for users to interact with.
   * Example: Display a list of products or order details.
2. **RESTful APIs**:
   * Return JSON or XML responses for API consumers.
   * Example: Mobile apps or third-party integrations.
3. **Custom Responses**:
   * Generate dynamic content, such as reports or file downloads.

**3. Rendering HTML with Views**

**Description:**

* Views use Razor syntax to generate HTML dynamically.
* Views are stored in the Views folder, organized by controller.

**Syntax:**

1. **View**:

@model string

<h1>Hello, @Model!</h1>

**Uses:**

* Views generate dynamic content based on data passed from controllers.
* Razor syntax combines C# and HTML.

**Example:**

* Create a view (Index.cshtml) that displays a list of products:

@model List<string>

<ul>

@foreach (var product in Model)

{

<li>@product</li>

}

</ul>

**Project:**

* Build a **Student List View**:
  + Use Razor syntax to display a list of students dynamically.

**4. Tag Helpers**

**Description:**

* Tag Helpers are used to generate and manipulate HTML elements using server-side logic.

**Syntax:**

1. **Anchor Tag Helper**:

<a asp-controller="Home" asp-action="About">About</a>

**Uses:**

* Simplifies working with HTML forms, links, and elements.
* Provides strong typing and avoids hardcoding URLs.

**Example:**

* Use the Form Tag Helper:

<form asp-controller="Home" asp-action="Submit">

<input type="text" name="Name" />

<button type="submit">Submit</button>

</form>

**Project:**

* Create a **Student Form**:
  + Use the Form Tag Helper to add a new student.

**5. Models: Binding and Validations**

**Description:**

* Models represent data and include validation rules for input data.

**Syntax:**

1. **Model with Validation**:

public class Student

{

public int Id { get; set; }

[Required]

public string Name { get; set; }

}

**Uses:**

* Ensures data integrity by validating user inputs.
* Provides binding between views and models.

**Example:**

1. Create a model with validation attributes:
   * [Required], [Range], etc.
2. Use model binding in the view:

<input asp-for="Name" />

<span asp-validation-for="Name"></span>

**Project:**

* Build a **Student Registration Form**:
  + Validate the name and marks using validation attributes.

**6. URL Routing and Features**

**Description:**

* Routing maps URLs to controllers and actions.

**Syntax:**

1. **Default Route**:

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

**Uses:**

* Simplifies navigation between different parts of the application.
* Allows custom URL patterns.

**Example:**

1. Add a custom route:

app.MapControllerRoute(

name: "custom",

pattern: "MyApp/{controller=Home}/{action=Index}/{id?}");

**Project:**

* Create a **Custom Route**:
  + Map /Student/Details/1 to the Details action of StudentController.

**7. Web API Applications**

**Description:**

* Web APIs allow interaction with data through HTTP methods (GET, POST, etc.).

**Syntax:**

1. **API Controller**:

[ApiController]

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

public class StudentController : ControllerBase

{

[HttpGet]

public IEnumerable<Student> GetAll()

{

return new List<Student>

{

new Student { Id = 1, Name = "Alice" },

new Student { Id = 2, Name = "Bob" }

};

}

}

**Uses:**

* Provides endpoints for external clients.
* Simplifies data exchange using JSON.

**Example:**

* Build an API for managing products:
  + /api/products (GET): Returns all products.
  + /api/products/{id} (GET): Returns a product by ID.

**Project:**

* Build a **Student API**:
  + Create endpoints for listing and adding students.

**8. Dependency Injection and IoC Containers**

**Description:**

* Dependency Injection (DI) provides dependencies at runtime.
* IoC (Inversion of Control) containers manage object lifetimes and dependencies.

**Syntax:**

1. **Register a Service**:

builder.Services.AddScoped<IStudentService, StudentService>();

1. **Use DI in Controller**:

public class HomeController : Controller

{

private readonly IStudentService \_service;

public HomeController(IStudentService service)

{

\_service = service;

}

}

**Uses:**

* Promotes loose coupling.
* Simplifies testing and code maintainability.

**Example:**

1. Create an interface and service:

public interface IStudentService

{

List<Student> GetAllStudents();

}

public class StudentService : IStudentService

{

public List<Student> GetAllStudents()

{

return new List<Student>();

}

}

1. Register and inject the service.

**Project:**

* Build a **Student Service**:
  + Use DI to fetch and manage student data.