Unit 3 – Routing, Versioning & Fluent Validation

🧭 1. Routing

What is Routing?

- Routing is the process that decides **which controller and action method** should handle an incoming **HTTP request**.
- It connects the **URL** entered by the client (browser/app) to the correct method in your API.

Key Concepts:

- **HTTP Request:** A request sent by a client (browser or mobile app) to the server with URL, headers, and data.
- **Routing Middleware:** A built-in part of ASP.NET Core that reads incoming URLs and matches them with routes defined in the app.
- **Route Templates:** Define URL patterns (like /api/products/{id}) using placeholders {controller}, {action}, {id}.
- **Route Values:** Extracted from the URL (for example, in /products/123, id=123).

Types of Routing

ASP.NET Core has **two main routing types:**

1. **Convention-Based Routing**
    - Follows a predefined rule pattern, like /controller/action/id.
    - Routes are usually defined in Program.cs or Startup.cs.
    - Example:
    - endpoints.MapControllerRoute(
    - name: "default",
    - pattern: "{controller=Home}/{action=Index}/{id?}");
2. **Attribute Routing**
    - Routes are directly defined **using attributes** on controller or action methods.
    - Example:
    - [Route("api/products/{id}")]
    - public IActionResult GetProduct(int id) { ... }

🏷 Attribute Routing (Detailed)

1. Basic Attribute Routing

You can define route patterns using the [Route()] attribute.

Example:

```
[Route("Home")]
public class HomeController : Controller
{
   [Route("Index")]
   [Route("")]
   [Route("Home/Index")]
   public IActionResult Index() => View();
}
```

Now, the Index() method can be accessed with:

- http://localhost:5280/
- http://localhost:5280/Home
- http://localhost:5280/Home/Index

---

2. Attribute Routing with Parameters

- You can add route parameters to the URL.
  Example:

```
[Route("Home/Details/{id}")]
public IActionResult Details(int id) => View();
```

- Here, {id} will be taken from the URL (e.g., /Home/Details/10).

This process of taking values from URL and mapping to parameters is called **Model Binding**.

---

3. Attribute Routing with Optional Parameters

- You can make route parameters optional using ?.
  Example:

```
[Route("Home/Details/{id?}")]
public IActionResult Details(int? id) => View();
```

- URL /Home/Details and /Home/Details/10 both work.

---

4. Attribute Routing at Controller Level

- You can set a base route for a controller and then define action routes relative to it. Example:

```
[Route("api/[controller]")]
public class ProductController : Controller
{
   [Route("All")]
   public IActionResult GetAll() { ... }

   [Route("{id}")]
   public IActionResult GetById(int id) { ... }
}
```

- Here URLs will be:
    - /api/product/all
    - /api/product/1

---

5. Ignoring Controller-Level Route

If you don't want to use the controller-level route for a specific action, start its route with / or ~/. Example:

```
[Route("~/About")]
public IActionResult About() { ... }
```

- This ignores the controller-level route and directly maps to /About.

⚙️ **2. API Versioning**

Why Do We Need Versioning?

- In software development, **change is constant** — we add new features or fix issues.
- API versioning helps maintain **old versions** of the API while creating **new versions**.
- This ensures:
    - Existing clients (users of old version) still work.
    - New clients get improved features.

In short:
Versioning = Keeping old and new API versions side-by-side without breaking existing apps.

What is API Versioning?

- It's the process of managing different versions of an API.
- It helps developers to:
    - Fix bugs and security issues.
    - Add new features.
    - Avoid breaking existing apps that depend on older APIs.

If changes affect users, they are called **breaking changes**.
Versioning helps to handle them smoothly.

Types of API Versioning

1. **URL Versioning**
    - The version number is included in the URL.
      Example:
      https://example-api.com/v1/products
      https://example-api.com/v2/products
      ➜ Most popular method.
2. **Query Parameter Versioning**
    - Version number is passed as a query parameter.
      Example:
      https://example-api.com/products?version=v1
3. **Header Versioning**
    - Version number is passed inside the **request header**, not in the URL.
      Example:
      api-version: 1.0
4. **Consumer-Based Versioning**
    - Each user or consumer is bound to a version when they first use the API.
    - Their future requests always use that version unless they manually change it.

✅ **3. Fluent Validation**

What is Fluent Validation?

- Fluent Validation is a **.NET library** used to write **strong and flexible validation rules** for models.
- It gives more control and flexibility than traditional **Data Annotations** ([Required], [EmailAddress], etc.).

When to Use It?

- When you need **complex conditions** for validation.
- When you want to **separate validation logic** from models.
- When you want **cleaner code** and **better client-side validation**.

Steps to Apply Fluent Validation

Step 1: Install Package

FluentValidation.AspNetCore

Step 2: Create Validator Class

Example:

```
public class PersonValidator : AbstractValidator<PersonModel>
{
   public PersonValidator()
   {
      RuleFor(x => x.Name).NotEmpty().WithMessage("Name is required");
      RuleFor(x => x.Email).EmailAddress();
      RuleFor(x => x.Age).InclusiveBetween(18, 60);
   }
}
```

- **AbstractValidator<T>** – Base class from FluentValidation.
- **RuleFor(x => x.Property)** – Defines rule for that property.

Step 3: Register in Program.cs

```
builder.Services.AddControllers()
   .AddFluentValidation(fv => fv.RegisterValidatorsFromAssemblyContaining<PersonValidator>());
```

Step 4: Run Project and Test

- Check validations using Swagger or any form — it will show error messages automatically.

Basic Validators

Validator	Description
.NotEmpty()	Field must not be empty
.Length(min, max)	Length limit

Validator	Description
.EmailAddress()	Must be valid email
.InclusiveBetween(min, max)	Number must be in range
.Matches(regex)	Must match pattern

Example:

```
RuleFor(x => x.Password)
   .NotEmpty()
   .Length(8, 15)
   .WithMessage("Password must be 8–15 characters");
```

Number Validators

Rule	Example
.GreaterThan(0)	Must be positive
.LessThan(100)	Must be under 100
.Equal(10)	Must be exactly 10
.NotEqual(0)	Cannot be zero

Date Time Validation Example

Rules:

- Event date must be **in the future**.
- It must be **within next 30 days**.
- It must **not fall on weekends (Sat/Sun)**.
- It must be **within next month**.

Conditional Validation

Used when validation depends on another property's value.
Methods:

- **When()** – Apply rule only when condition is true.
- **Unless()** – Apply rule unless condition is true.

Example:

```
RuleFor(x => x.Discount)
    .GreaterThan(0)
    .When(x => x.IsPremiumUser);
```

ForEach() Method

- Used to apply validation to every element in a collection.
  Example:

```
RuleForEach(x => x.Emails).EmailAddress();
```

HashSet

- A **collection that stores unique values** (no duplicates).
- Faster than List when checking if an element already exists.

Cascade Modes

Determines how multiple rules are checked:

Mode	Description
Continue (default)	Runs all rules even if one fails.
Stop	Stops checking after first failure.

Advantages of Fluent Validation

✅ **High Flexibility** – You can create any custom rule.
✅ **Testability** – Easy to write unit tests.
✅ **Cleaner Code** – Validation logic is separate from model.
✅ **Easy Maintenance** – Change rules in one place.
✅ **Better User Experience** – Shows custom, detailed error messages.

📜 **Summary**

Topic	Key Idea
Routing	Maps URL to specific controller and action.
Attribute Routing	Define routes directly on methods with attributes.
API Versioning	Handle multiple API versions (URL, query, header, consumer-based).

Topic	Key Idea
Fluent Validation	Build complex validation rules using C# code, with clean structure and flexibility.