# .NET Core Fundamentals

## 1. .NET Core Overview

.NET Core is an open-source, cross-platform framework developed by Microsoft for building modern applications. It provides high performance, scalability, and flexibility, supporting Windows, macOS, and Linux environments.

## 2. ASP.NET Core

ASP.NET Core is a lightweight, modular, and high-performance framework for building web applications and APIs. It is a re-architected version of ASP.NET, designed for cloud and microservices architectures.

## 3. Project Structure

An ASP.NET Core project includes essential folders and files, such as:

* **Controllers/**: Contains controller classes that handle HTTP requests.
* **Models/**: Defines data structures and business logic.
* **Views/** (for MVC projects): Contains Razor views for rendering UI.
* **wwwroot/**: Stores static files like CSS, JavaScript, and images.
* **Program.cs**: Entry point of the application.
* **appsettings.json**: Configuration file for settings like database connections and logging.

## 4. wwwroot Folder

The wwwroot folder serves as the root for static files, such as JavaScript, CSS, and images. By default, ASP.NET Core serves files from this folder, making it ideal for frontend assets.

## 5. Program.cs

The Program.cs file is the entry point of an ASP.NET Core application. It sets up the WebApplication host, configuring services and middleware components.

## 6. Startup.cs

Prior to .NET 6, Startup.cs handled application startup configuration. It contained methods such as:

* **ConfigureServices()**: Registers services for dependency injection.
* **Configure()**: Defines the request pipeline with middleware components.

## 7. launchSettings.json

This file configures application launch settings for different environments (IIS, console, etc.), defining URLs and profiles for debugging.

## 8. appSettings.json

The appsettings.json file stores configuration settings, such as database connections, logging levels, and third-party service credentials. It supports hierarchical structures and environment-based overrides.

# ASP.NET Core Request Processing Pipeline

## 1. Middleware

Middleware are components that process HTTP requests in a pipeline. Examples include authentication, logging, and routing middleware.

## 2. Routing

Routing directs incoming requests to the appropriate controller action based on URL patterns. It supports attribute-based and conventional routing.

## 3. Filters

Filters provide a way to inject logic at different request processing stages. Common filter types include:

* **Authorization Filters**: Enforce authentication policies.
* **Action Filters**: Modify action execution.
* **Exception Filters**: Handle exceptions centrally.
* **Resource Filters**: Handle request pre-processing and caching.
* **Result Filters**: Modify or process action results.

## 4. Controller Initialization

Controllers in ASP.NET Core are instantiated using dependency injection, ensuring better testability and maintainability.

## 5. Action Method

An action method in a controller handles specific HTTP requests, returning responses in various formats like JSON, XML, or views.

# Dependency Injection

## 1. Built-in IoC Container

ASP.NET Core includes a built-in IoC (Inversion of Control) container to manage dependencies and service lifetimes.

## 2. Registering Application Service

Services are registered in Program.cs using methods like builder.Services.AddScoped<T>().

## 3. Understanding Service Lifetime

Service lifetimes determine how instances are managed:

* **Transient**: New instance created per request.
* **Scoped**: Shared within a single request.
* **Singleton**: Shared across all requests.

## 4. Extension Methods for Registration

Custom extension methods help organize service registration in a structured manner.

## 5. Constructor Injection

Constructor injection allows dependencies to be passed via the controller or service constructor, promoting loose coupling.

# Exception Handling

## 1. UseDeveloperExceptionPage

This middleware provides detailed error information during development mode, aiding in debugging.

## 2. UseExceptionHandler

In production, UseExceptionHandler() is used to handle exceptions gracefully, redirecting users to a generic error page.

# Logging in ASP.NET Core

## 1. Logging API

ASP.NET Core provides a built-in logging API supporting various levels, such as Information, Warning, and Error.

## 2. Logging Providers

Logging providers integrate with third-party logging frameworks like NLog, Serilog, and Application Insights for better log management and storage.