Data Annotation

Data Annotations are attributes that can be added to c# model properties to provide additional metadata for validation, formatting, and other purposes. These annotations help to improve the quality of data passes in the application ensure consistency, and reduce errors.

using System;

using System.ComponentModel.DataAnnotations;

public class Employee

{

```
    [Required]
    [EmailAddress(ErrorMessage = "Invalid email address")]
    public string email { get; set; }
```

}

In this Example, we used different Data Annotations:

1. [Required] - this attribute indicates that the property is mandatory and cannot be empty.

2. [EmailAddress] - this attribute ensures that the property value is valid email address.

Benefits of using Data Annotation in Webapi c# code:

1. Improved quality and consistency.
2. Simplified validation code.
3. Better Documentation.
4. Maintainability.

Action Filters

Action filters are executed before or after the execution of an action method in your API controller. Action filters can be used to implement authentication, exception handling, an dother concerns in your API.

Three types of action filters in Web API:

1. Authentication filter - to implement authentication and authorization in your API.

2. Action Filter - to implement cross-cutting concerns like logging, exception handling.

3. Exception Filter - to handle exception thrown by API.

Benefits of using action filters in Web API:

Reusability, consistency.

Exception Filter

Exception Filter is a mechanism to to handle exceptions that occur during the processing of webapi request.

When an unhandled exception occurs in your Web API application, it can cause the application to crash or return an error response that is not helpful to the client. Exception filters provide a way to catch and handle these exceptions in a more graceful and controlled manner.

Exception filters are classes that implement one of the following interfaces: IExceptionFilter, IExceptionLogger, or IExceptionHandler. These interfaces define methods that allow you to catch and handle exceptions, log exceptions, or generate error responses, respectively.

In C# ASP.NET Web API, there are four types of exception filters that you can use to handle exceptions in your application:

1.ExceptionFilterAttribute: This is the most general-purpose exception filter. It is executed whenever an unhandled exception occurs during the processing of a Web API request. You can use this filter to log the exception, send an email to the administrator, or return a custom error response to the client.

2.IExceptionFilter: This interface allows you to define a custom exception filter that handles specific types of exceptions. You can use this filter to catch and handle exceptions that are thrown by a specific controller or action method.

3.IExceptionLogger: This interface allows you to define a custom logger that logs exceptions when they occur in your Web API application. You can use this filter to log exceptions to a file, a database, or a third-party logging service.

4.IExceptionHandler: This interface allows you to define a custom handler that generates error responses when exceptions occur in your Web API application. You can use this filter to create custom error messages or return error responses in a specific format (e.g., JSON, XML) based on the exception type

Inversion of Control (IoC)

- IoC is a design pattern that allows for loosely coupled dependencies between components in a software system.
- In traditional approach, where a central module controls the flow and calls other components.
- Instead,here the application flow is inverted with the flow of control being managed by a separate container that manages the dependencies between components.
- STEP 1 - we create a simple interface to represent a data access object.
- STEP 2 - we create a class that implements the interface.
- STEP 3 - Now, we'll create a controller class that depends on interface instance to get the data.
- Instead of instaintiating the class within controller, we define dependency through constructor. This way, we can swap the implementation of the interface at runtime without changing the controller's code.

Error Models

- In a WebAPI project in C#, error models are used to represent the errors that can occur during the execution of an API call.
- An error model is a C# class that contains information about the error. Typically, an error model includes properties like the error message, error code, and any other information that can help developers understand what went wrong.
- Using error model in a WebAPI project, we can create an instance of the ErrorModel class whenever an error occurs, and then return it as part of the HTTP response.
- Using error models helps with API documentation and testing. By clearly defining the expected error responses, documentation can be more accurate.

Automated testing can also be used to verify that error responses are generated correctly.

Executing Regular Expression Separately

- Executing regular expressions separately in an API project, we can create a separate endpoint that accepts a regular expression as a parameter and returns the matched results.

1. Define an endpoint in your API project that accepts a regular expression as a parameter. For example, you can define a route like '/api/regex/{pattern}' where {pattern} is a placeholder for the regular expression.
2. In the implementation of the endpoint, retrieve the regular expression pattern from the route parameter and compile it into a regular expression object using the Regex class.
3. Use the Regex object to match the regular expression pattern against some input data. You can use the Match or Matches method to do this.
4. Return the matched results as a response from the API endpoint. You can use a custom model or class to define the structure of the response.

- Executing regular expressions separately can be beneficial in certain scenarios, such as, Testing, Reusability, Code organization.