**Ques: why controller contructor is called on every request?**

In ASP.NET Core, the **controller constructor** is called on **every request** because of the way **dependency injection (DI)** and **controller instantiation** work in the framework. Let's break this down step by step:

**1. Controller Instantiation in ASP.NET Core:**

Every time an HTTP request is made (whether it's a GET, POST, PUT, etc.), ASP.NET Core needs to create a new instance of the controller to handle the request. When ASP.NET Core creates this instance, it looks at the constructor of the controller and performs **dependency injection** (DI) to inject any services or dependencies that the controller needs.

**Why is this the case?**

ASP.NET Core follows a **per-request** model when creating controller instances. Each HTTP request is considered a distinct unit of work, so for each request, a new instance of the controller is created to ensure that:

* The controller has all the necessary dependencies injected for handling that request.
* The controller can be disposed of properly after the request is handled.

This is important because the lifetime of services (like database contexts, API clients, etc.) is managed by **dependency injection (DI)** and can vary depending on the service's registration type (like **Singleton**, **Scoped**, or **Transient**).

**2. Constructor Injection:**

In ASP.NET Core, **constructor injection** is the preferred method for injecting dependencies into controllers and services. When a controller is instantiated, the **constructor is called** to inject any dependencies specified in the constructor's parameters.

For example:

public class MyController : ControllerBase

{

private readonly IMyService \_myService;

public MyController(IMyService myService)

{

\_myService = myService;

}

// Action methods

}

* **Constructor**: The constructor of MyController is called each time a new instance of MyController is created, and the IMyService dependency is injected.
* This behavior ensures that the controller is provided with the required services to handle the request.

**3. Dependency Injection (DI) Lifetimes:**

ASP.NET Core supports three types of service lifetimes in DI:

* **Singleton**: A single instance of the service is created and shared throughout the application’s entire lifetime.
* **Scoped**: A new instance of the service is created for each **HTTP request**.
* **Transient**: A new instance of the service is created each time it is requested, even within the same request.

**Controller Constructor and Service Lifetime:**

* **Singleton services**: These are created once when the application starts and reused across requests. Even though the controller's constructor is called for each request, if a **Singleton** service is injected into the controller, the same instance of the service is reused across all requests.
* **Scoped services**: These are created once per **HTTP request**. If a **Scoped** service is injected into the controller, a new instance of the service is created for each request, and the controller constructor is called once for that request.
* **Transient services**: These are created each time they are requested. If a **Transient** service is injected into the controller, a new instance of that service will be created each time the controller is instantiated, even within the same HTTP request.

**4. Why Call the Constructor on Every Request?**

* **Fresh Controller Instances**: Each request needs a fresh instance of the controller to ensure that state does not leak between requests. Controllers are generally **stateless**, meaning they don't retain data between requests. If the constructor were called only once (e.g., at the application startup), it would mean that the same controller instance would handle multiple requests, which could introduce issues with data leakage between requests.
* **Request-Specific Data**: Each request may require different data or a different set of dependencies. For example, a service that handles **authentication** or **user context** may need to vary based on the user making the request, so ASP.NET Core creates a fresh instance of the controller (with new dependencies) each time.
* **Dependency Injection**: DI ensures that dependencies are injected and that services like **Scoped** services (e.g., DbContext) are only instantiated per HTTP request. This allows you to cleanly separate concerns and ensures that resources like database connections or services are not shared across requests unnecessarily.

**Example of How the Controller Constructor is Called:**

Consider the following controller:

public class MyController : ControllerBase

{

private readonly IMyService \_myService;

// Constructor is called every time a new request comes in

public MyController(IMyService myService)

{

\_myService = myService;

}

[HttpGet]

public IActionResult Get()

{

return Ok("Hello, World!");

}

}

* **Each Request**: Each time a new HTTP request (like GET /api/my) comes in, ASP.NET Core will:
  1. **Instantiate** the controller.
  2. **Call the constructor** to inject IMyService (or any other dependencies).
  3. **Execute the action method** (like Get()).
  4. Dispose of the controller after the request is processed.
* **Why Constructor is Called**: The constructor is called every time because a new instance of the controller is created for each request. If your controller has dependencies like a service, the DI system will instantiate those dependencies based on the **lifetime** you have defined (Singleton, Scoped, or Transient).

**5. How to Control Constructor Calls:**

If you want to limit when your constructor is called, you can adjust the **lifetime** of the injected services.

* **Singleton**: If you don't want the constructor to be called on every request for a specific dependency, you can register that dependency as a **Singleton**.
* builder.Services.AddSingleton<IMyService, MyService>();

This ensures that IMyService is instantiated only once during the application's lifetime, not on every request.

* **Scoped or Transient Services**: If your service requires a new instance for each request, use **Scoped** or **Transient** services. This ensures that the dependencies are injected into the controller constructor on every request, but the service instance will vary depending on the lifetime.

**Summary:**

* The **controller constructor** is called on every request because ASP.NET Core creates a **new instance** of the controller to handle each request.
* **Constructor injection** is used to inject the required dependencies (like services) into the controller each time it's instantiated.
* Depending on the **lifetime** of the dependencies (Singleton, Scoped, or Transient), the same or new instances of the services are injected into the controller, but the controller itself is instantiated fresh per request to avoid state sharing between requests.

Let me know if you'd like more clarification on any specific point!