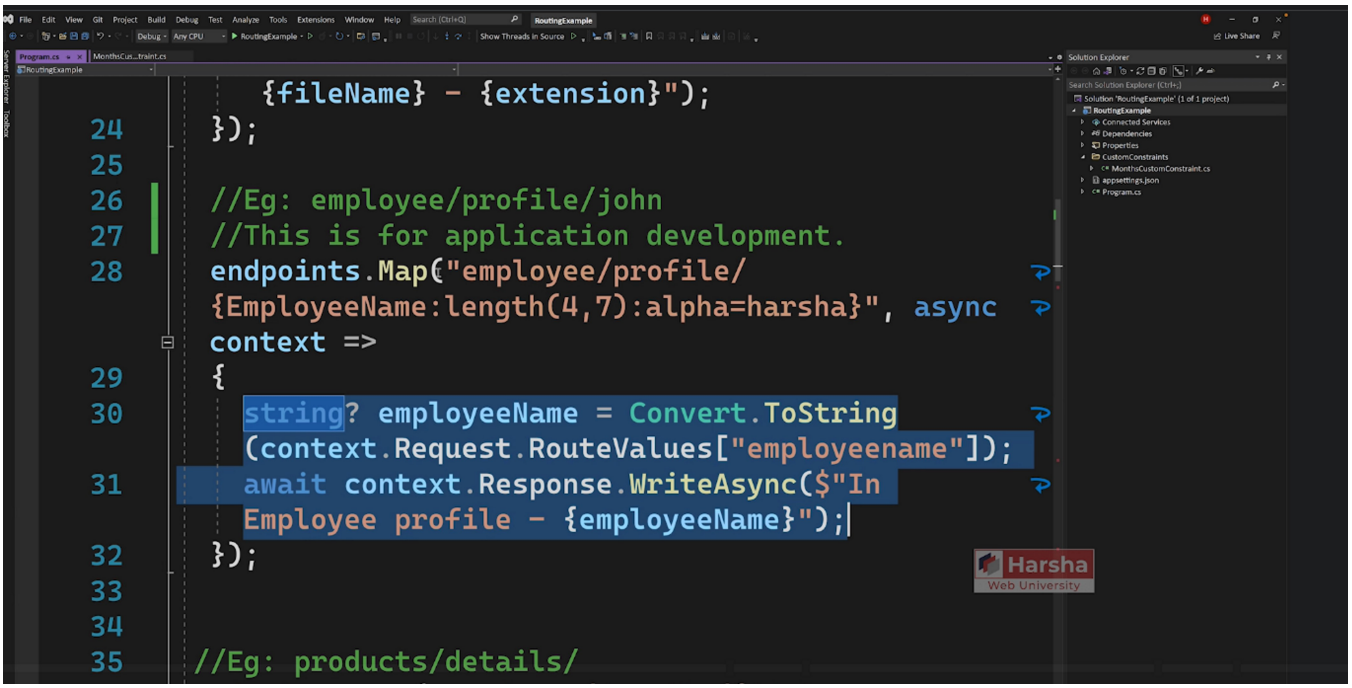
Section 6: Controllers & IActionResult [MVC]

01 January 2025 22:13



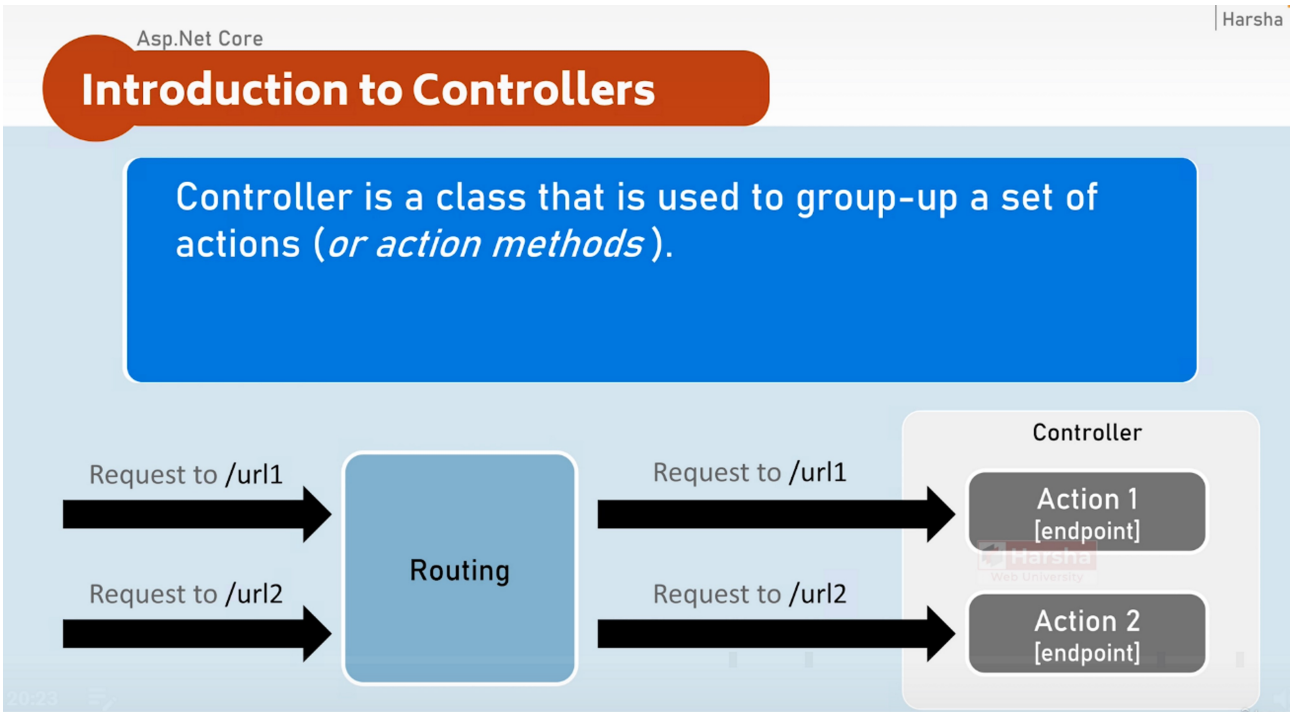
In the existing application, we created one endpoint for each URL and wrote the corresponding code in the same middleware.

But think about real-world projects—can you follow this approach in large-scale applications?

In real-world scenarios, there will be thousands of URLs with a significant amount of code. Writing all the endpoints in the same file is not good programming practice.

If the code grows in length and everything is in the same file without logical grouping, it becomes difficult to identify issues, debug errors, and maintain the codebase.

So, what's the solution? The solution is to group middlewares based on their logical purpose, and this is exactly what controllers provide.



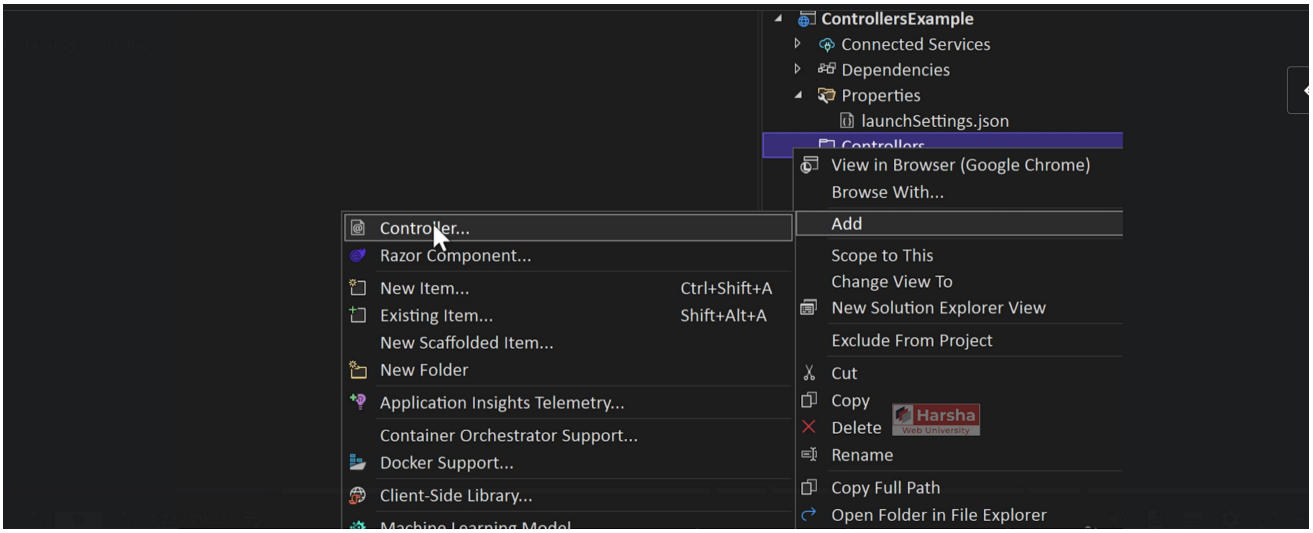
A controller is a class that contains a set of action methods. Each action method acts as an endpoint that can be requested via a specific URL.
For example, if you send a request to URL1, Action 1 will execute, and similarly, URL2 will trigger Action 2. These actions are grouped into a controller class, meaning there is a logical connection between them.
For instance, Action 1 may represent the user registration process, and Action 2 may represent the sign-in process. Both are related to user account maintenance, which forms the logical connection.
By definition, a controller is a collection of action methods (or actions for short), where each action performs a specific operation based on the inputs supplied.

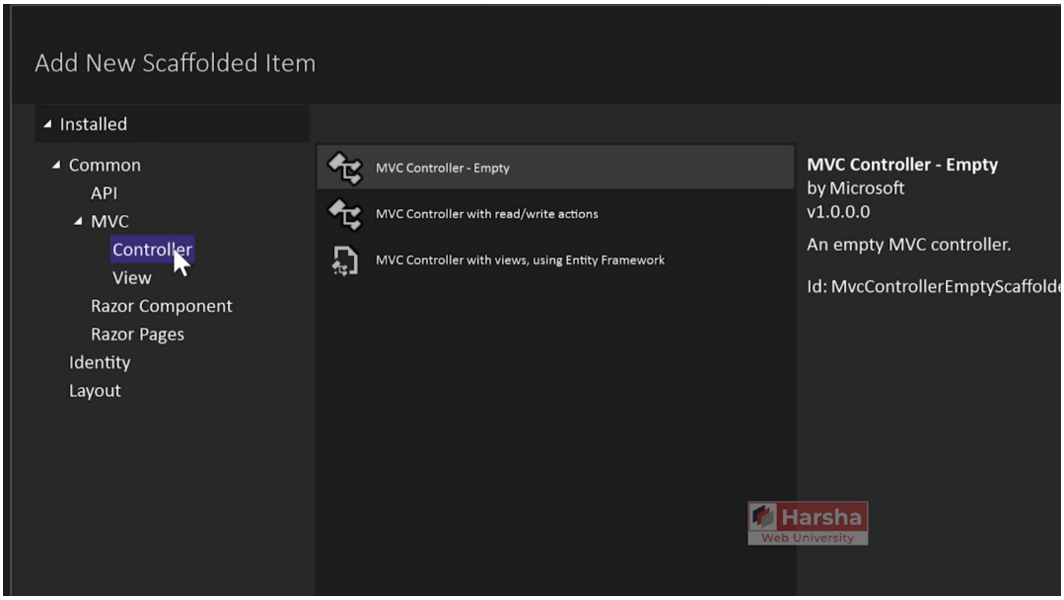
For example:

- Supplying details like username, password, confirm password, and date of birth to Action 1 initiates the registration process.
- Supplying email and password to Action 2 handles the login process.

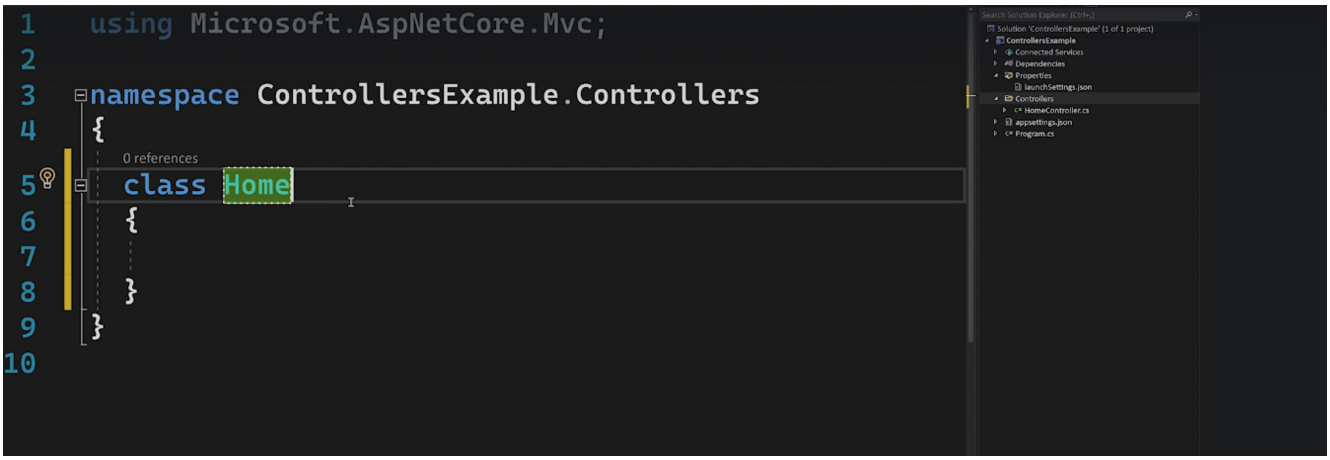
The primary reason to create controllers is to group actions based on their purpose.
Now, how can you create and enable controllers? Let’s try it practically!

create a folder called 'Controllers' (not mandatory) in visual studio.

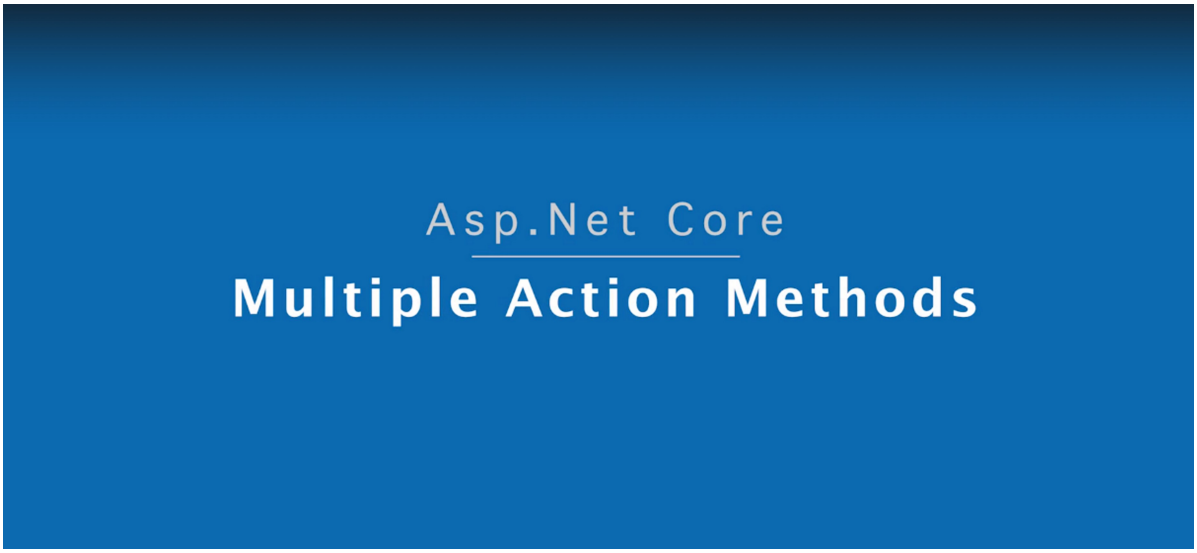




just a normal class, not a controller class

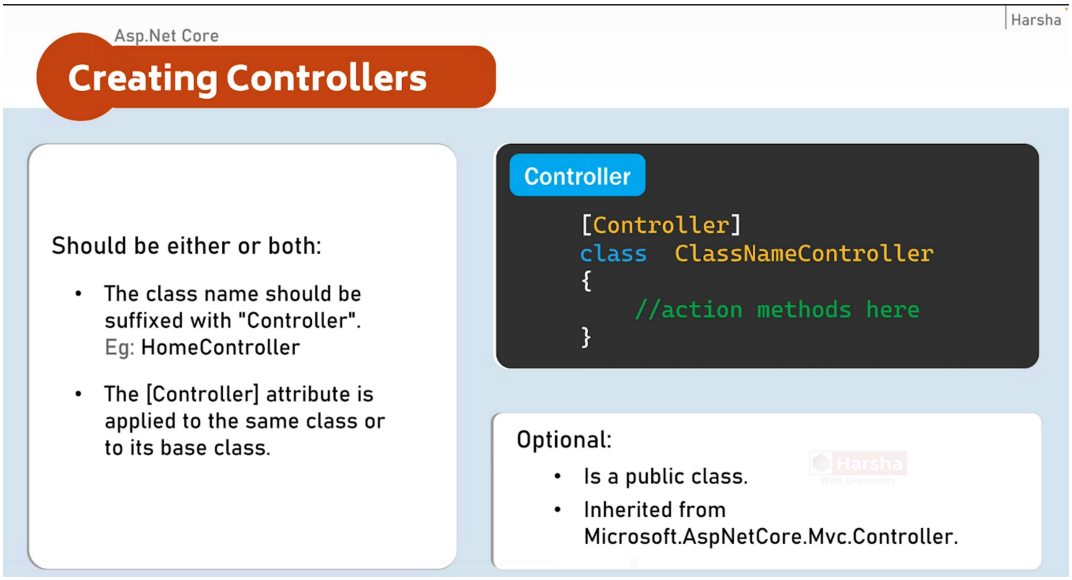


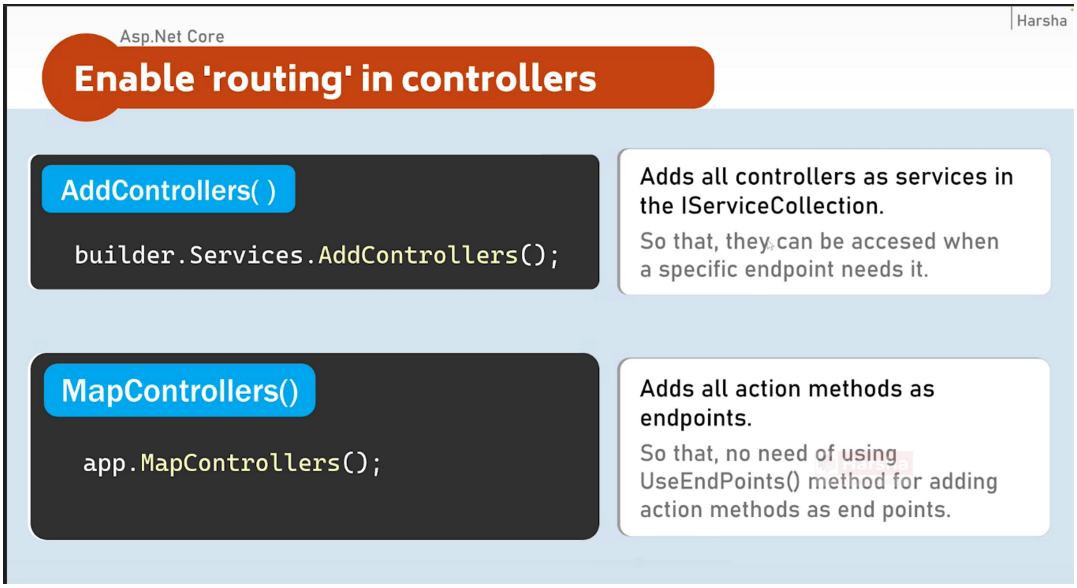
In order to make that class as a Controller class, you have to suffix the same with 'Controller'. It's the fixed pre-defined name.

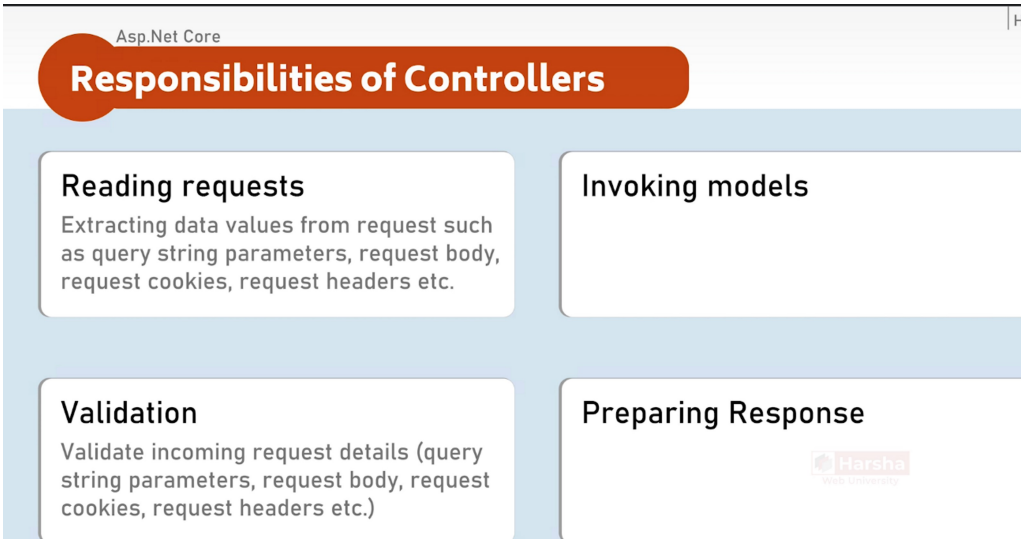


Asp.Net Core

Takeouts about Controllers

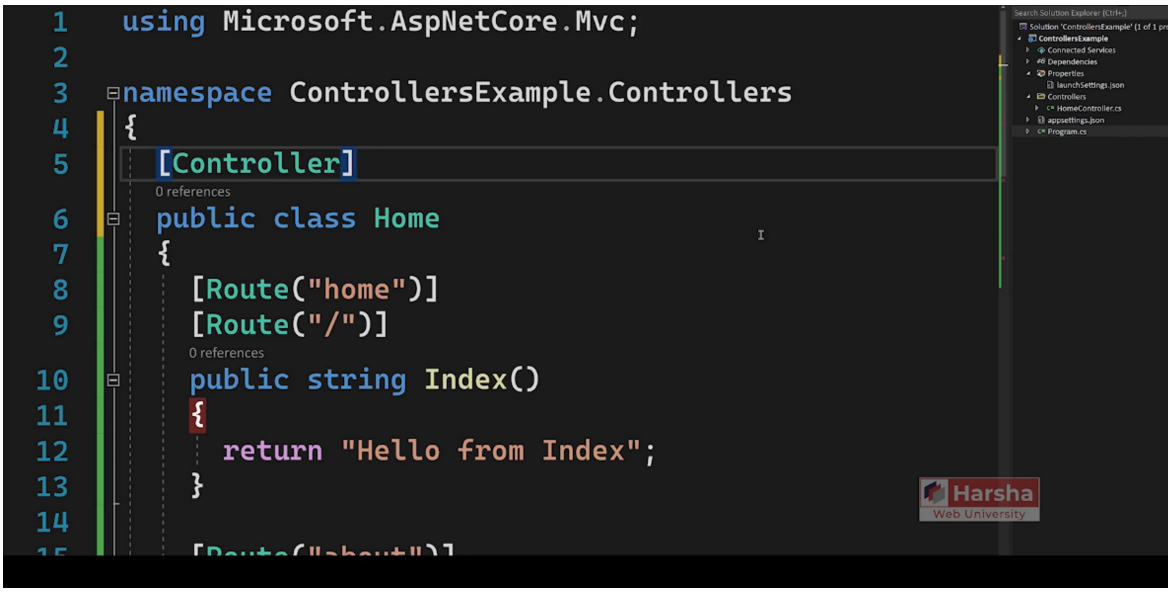






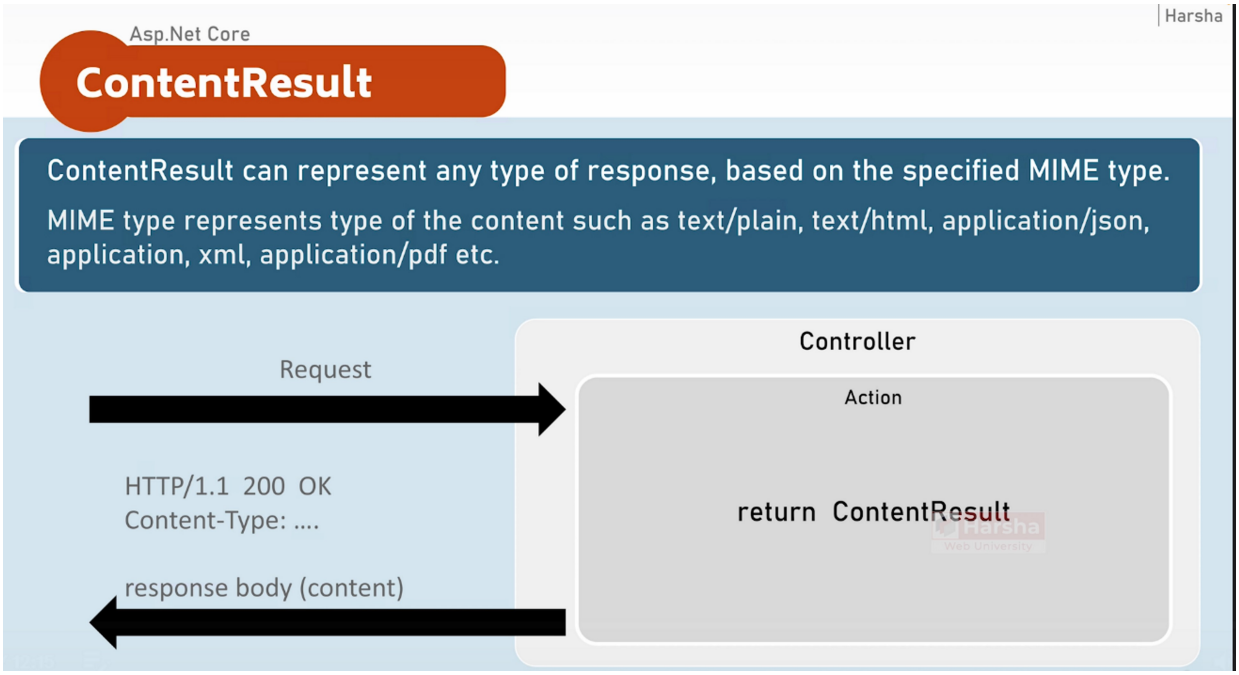
either use Suffix 'Controller' . ex: HomeController

or



Asp.Net Core

ContentResult



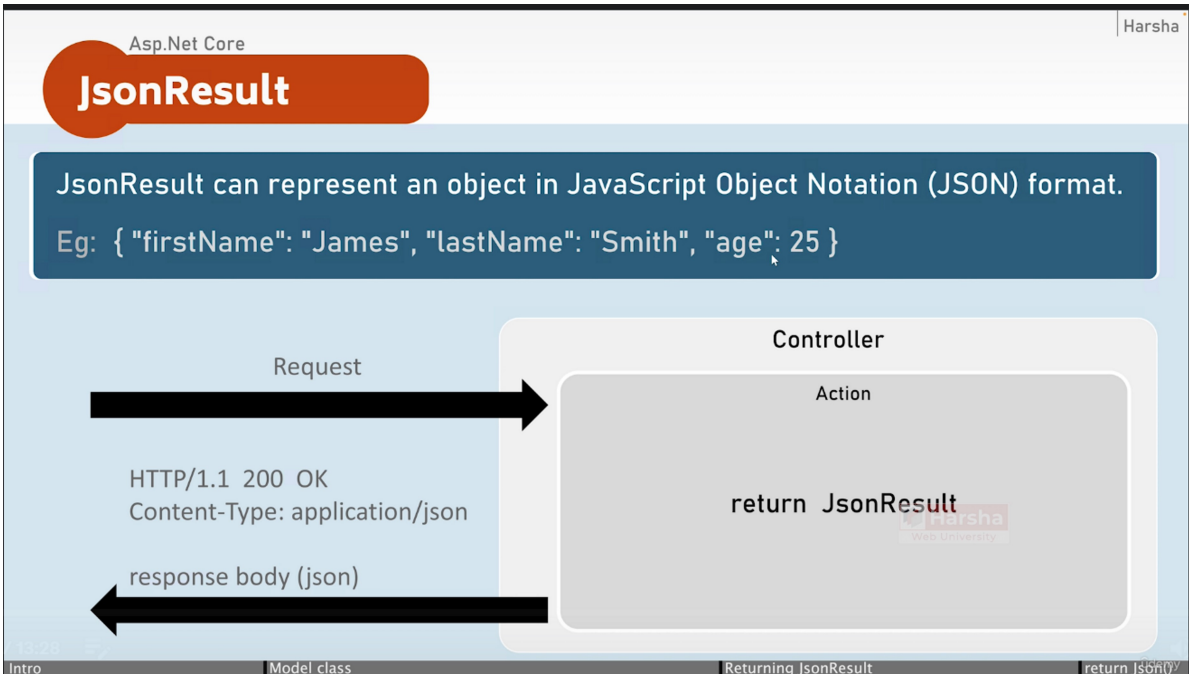
The ContentResult is a type of action result that can be returned from an action method. It can represent almost any type of content.
For example, you can return plain text, HTML, XML, JSON, PDF files, or nearly any type of content using ContentResult.

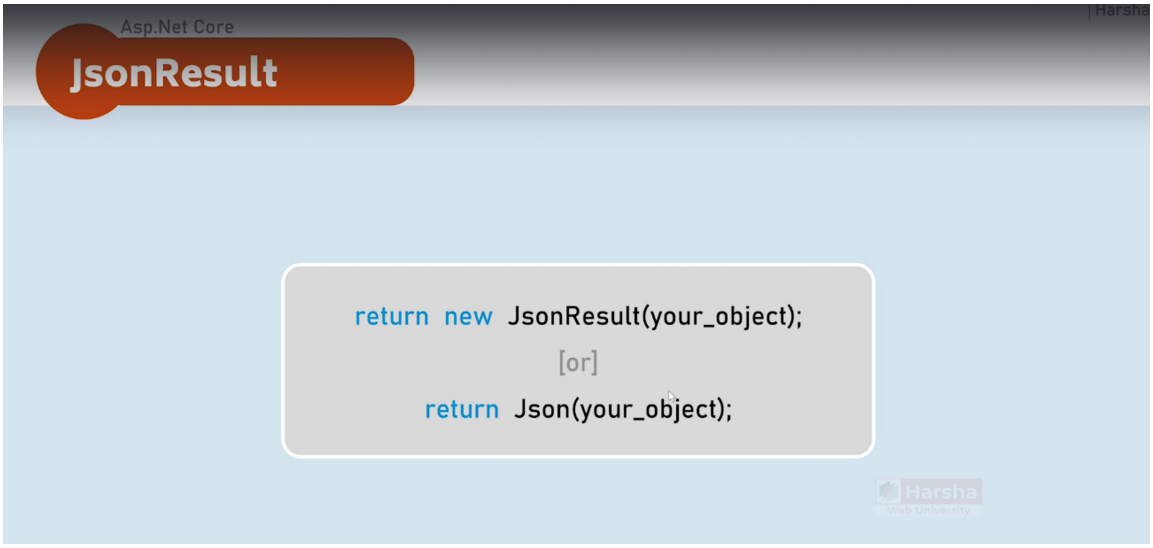
When creating an object for the ContentResult class, you need to specify two pieces of information:
Content: The actual response body you want to return, such as a message.
ContentType: The MIME type of the content, which is added to the response headers.

For example:
- If the response body is plain text, the ContentType would be text/plain.

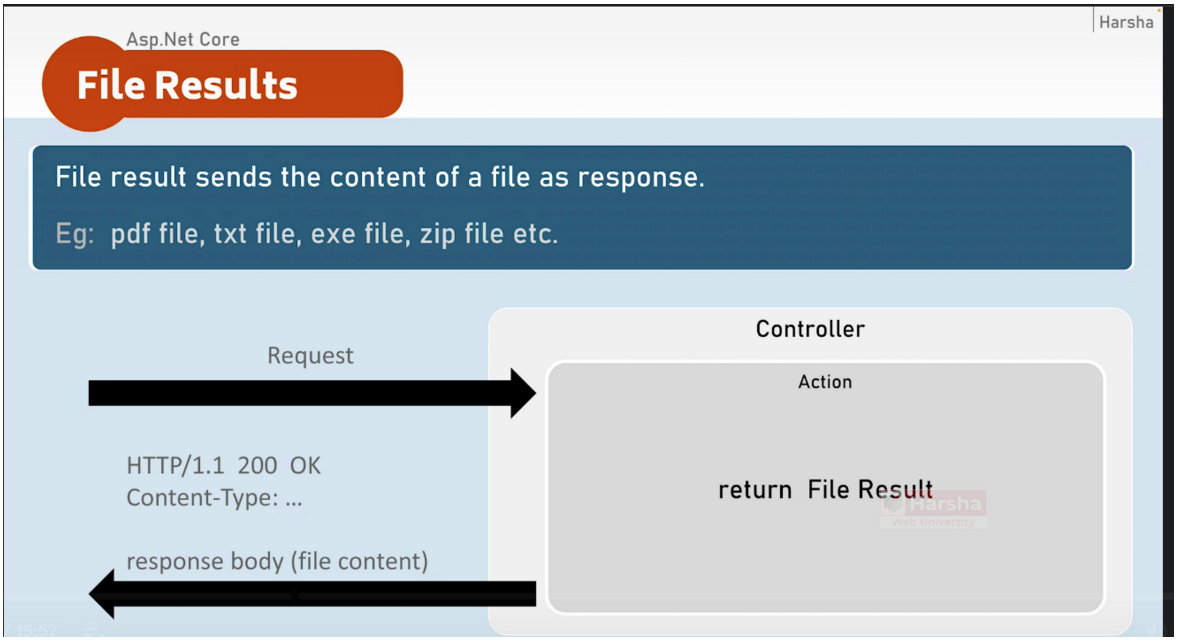
We have already seen an overview of the ContentType while learning about request and response

objects earlier.
Now, let's see how to use it in this existing application.

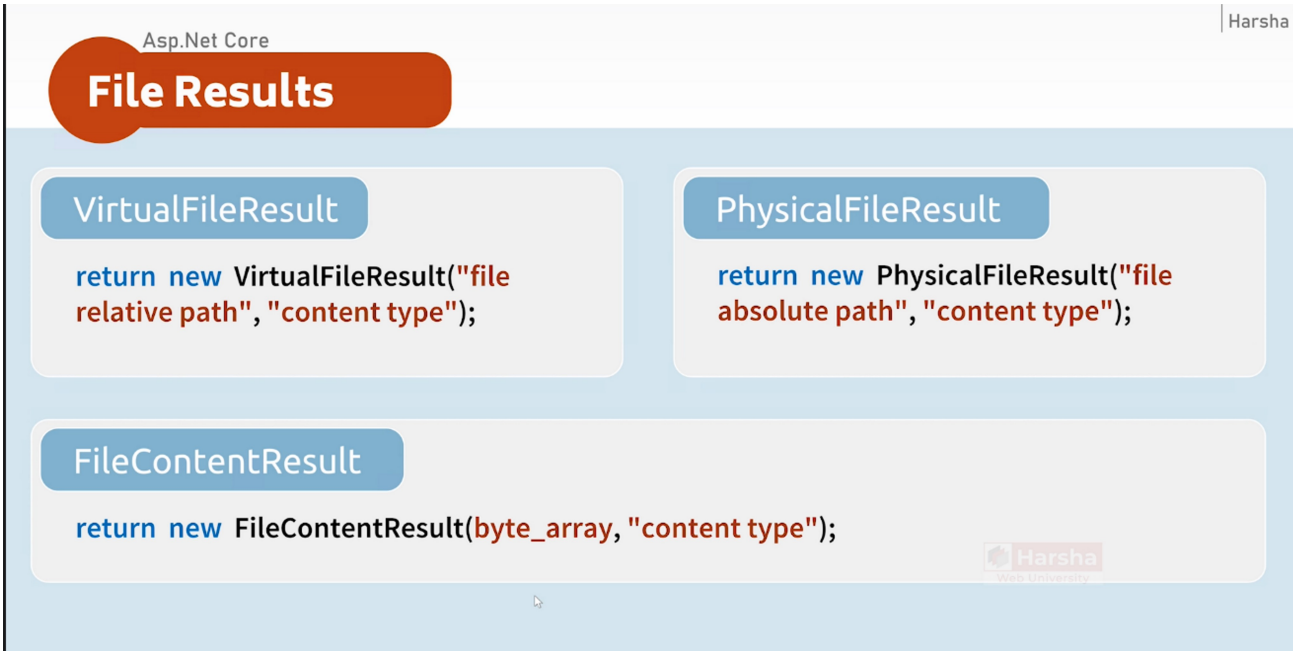


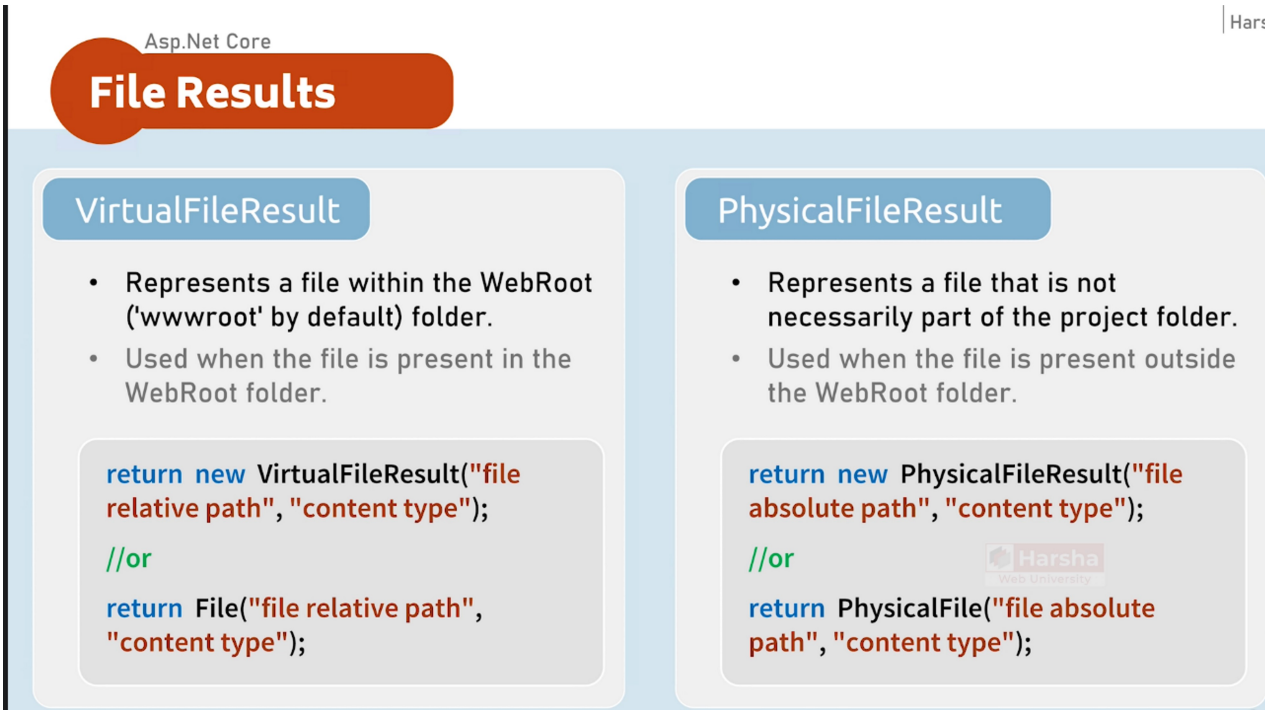

Asp.Net Core
JsonResult
return new JsonResult(your_object);
[or]
return Json(your_object);

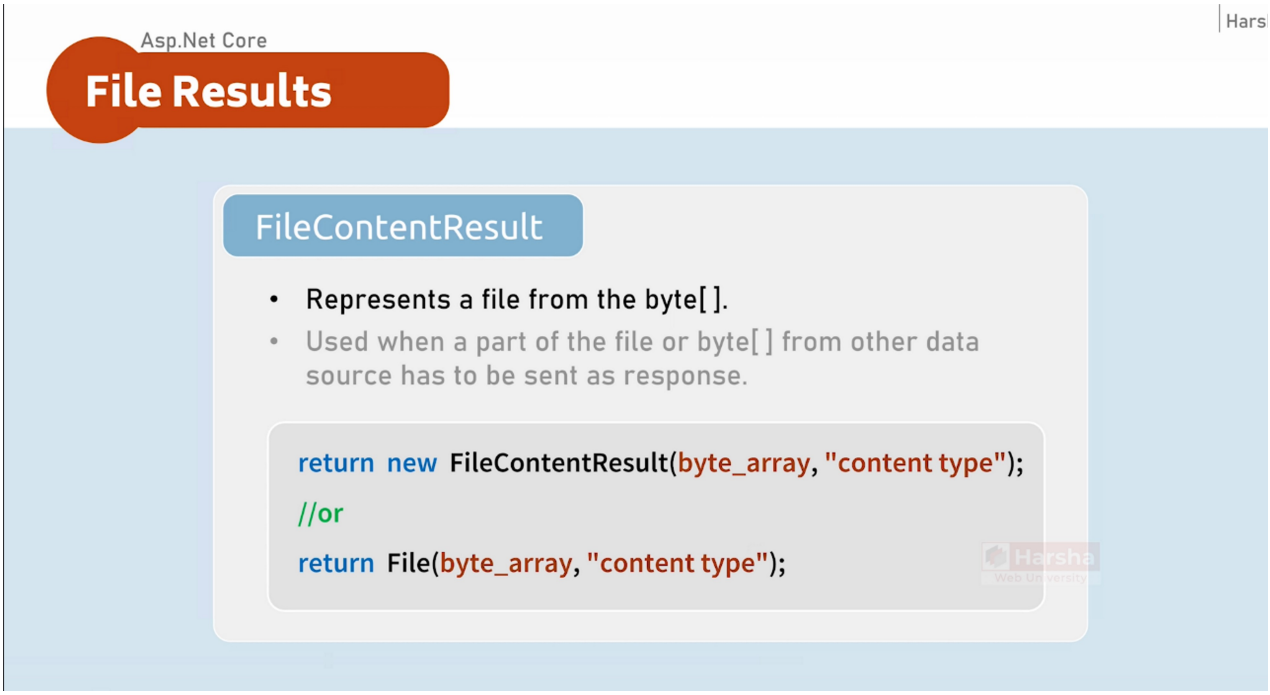

Asp.Net Core
File Results


Asp.Net Core
File Results
File result sends the content of a file as response.
Eg: pdf file, txt file, exe file, zip file etc.
Request
Controller
Action
return File Result
HTTP/1.1 200 OK
Content-Type: ...
response body (file content)

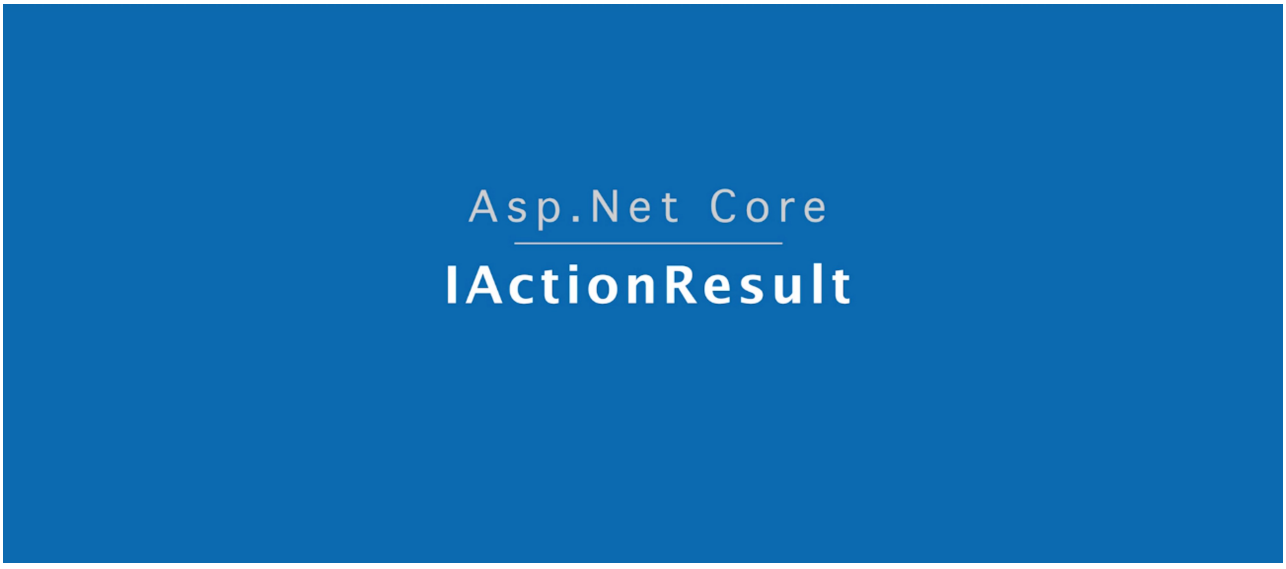
Did you notice sometimes when you open some webpages It gives you a file as download.



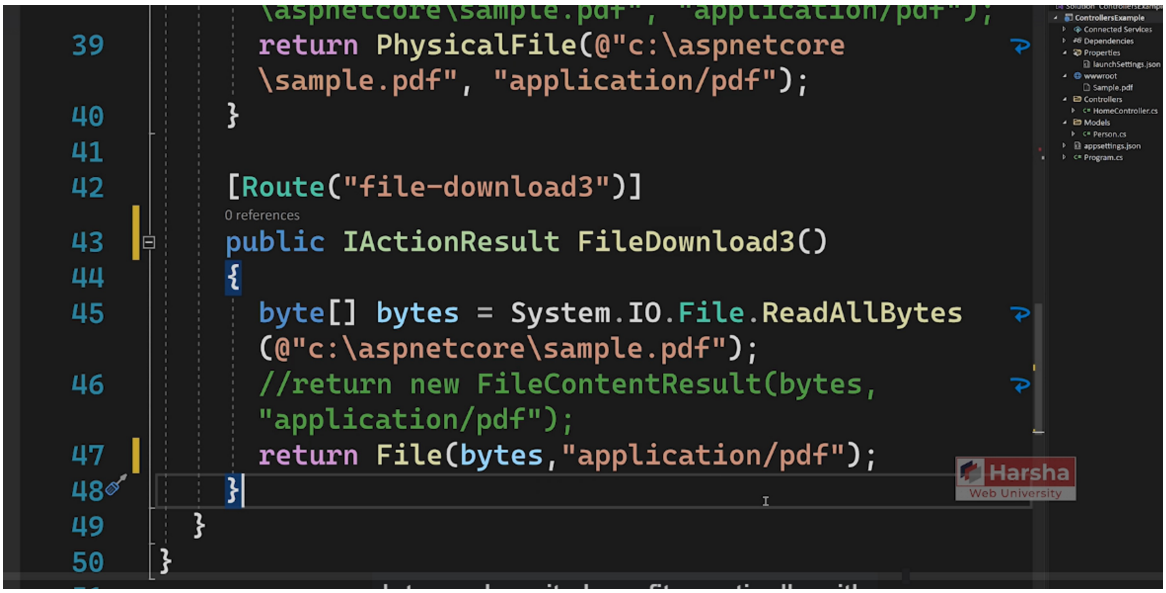


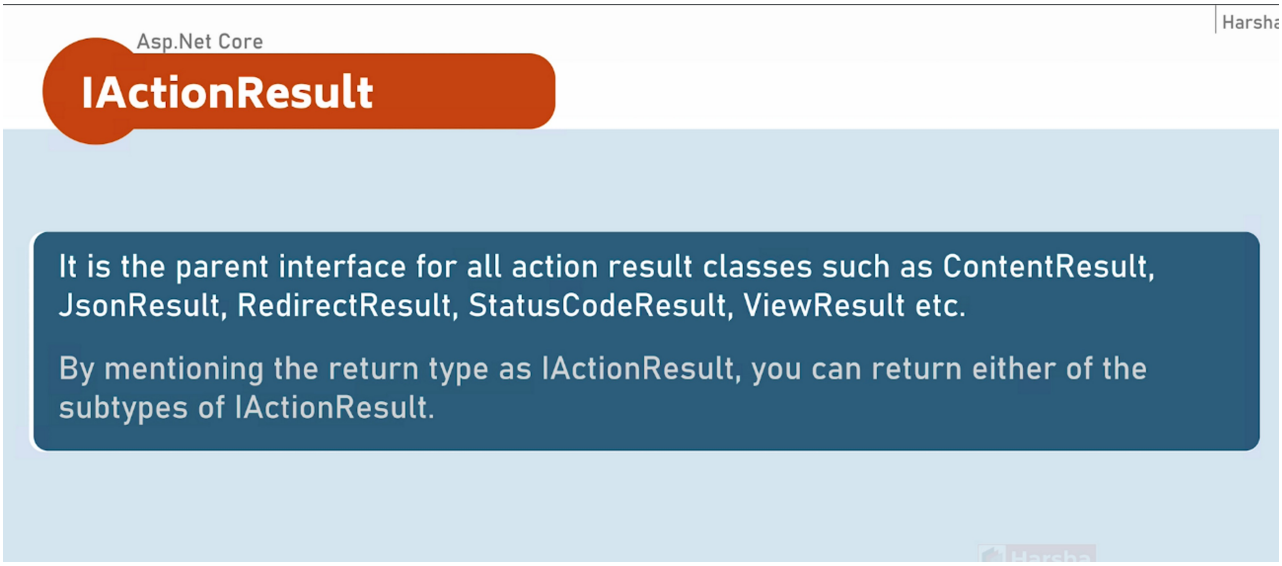


It is useful when you are reading images form databases



In ASP.NET Core, it is required to mention the return type of all the action methods as IActionResult, like this:
In this case, you can return any type of action result such as ContentResult, JsonResult, FileResult, etc.
But here, my question is: how is that possible?





But why is the IActionResult even necessary?

suppose my requirement is to pass 'isLoggedin' and 'bookid' value.

010/book?isloggedin=true&bookid=1

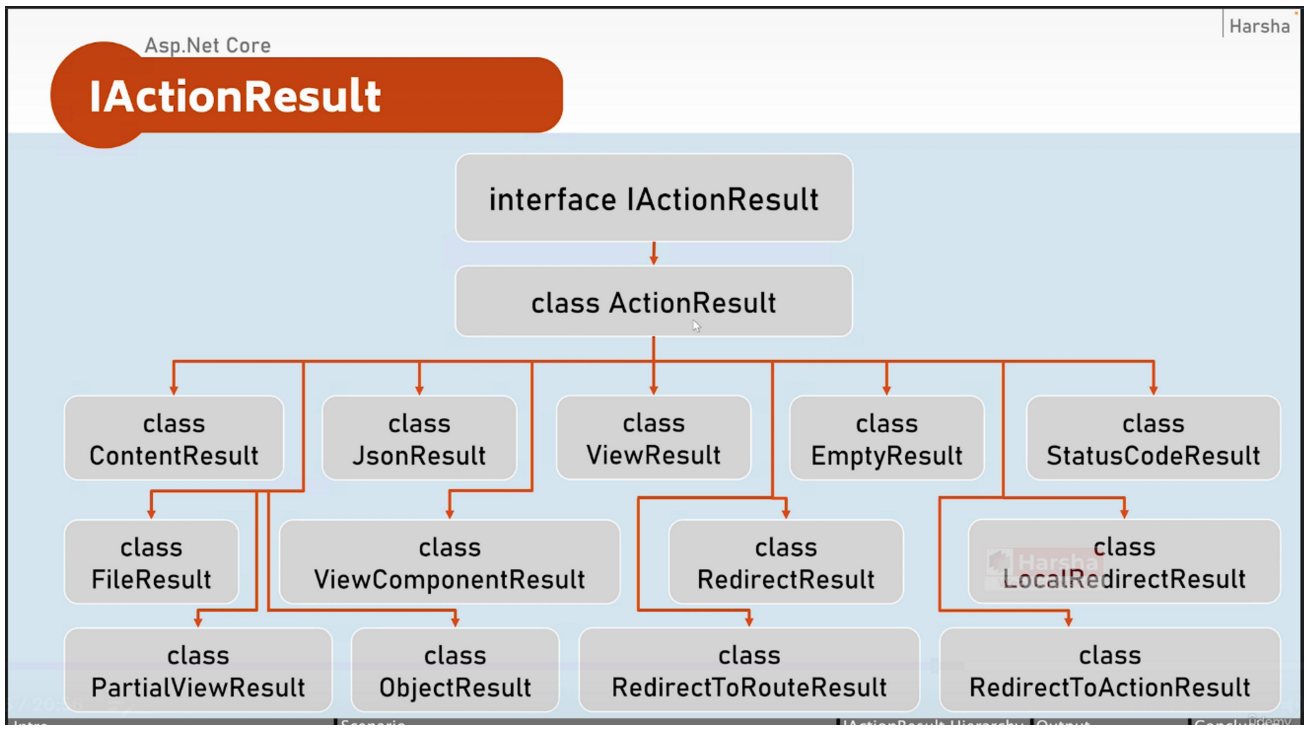
's (6) Body Pre-request Script Tests Settings

	VALUE
	true
	1

010/book?isloggedin=true

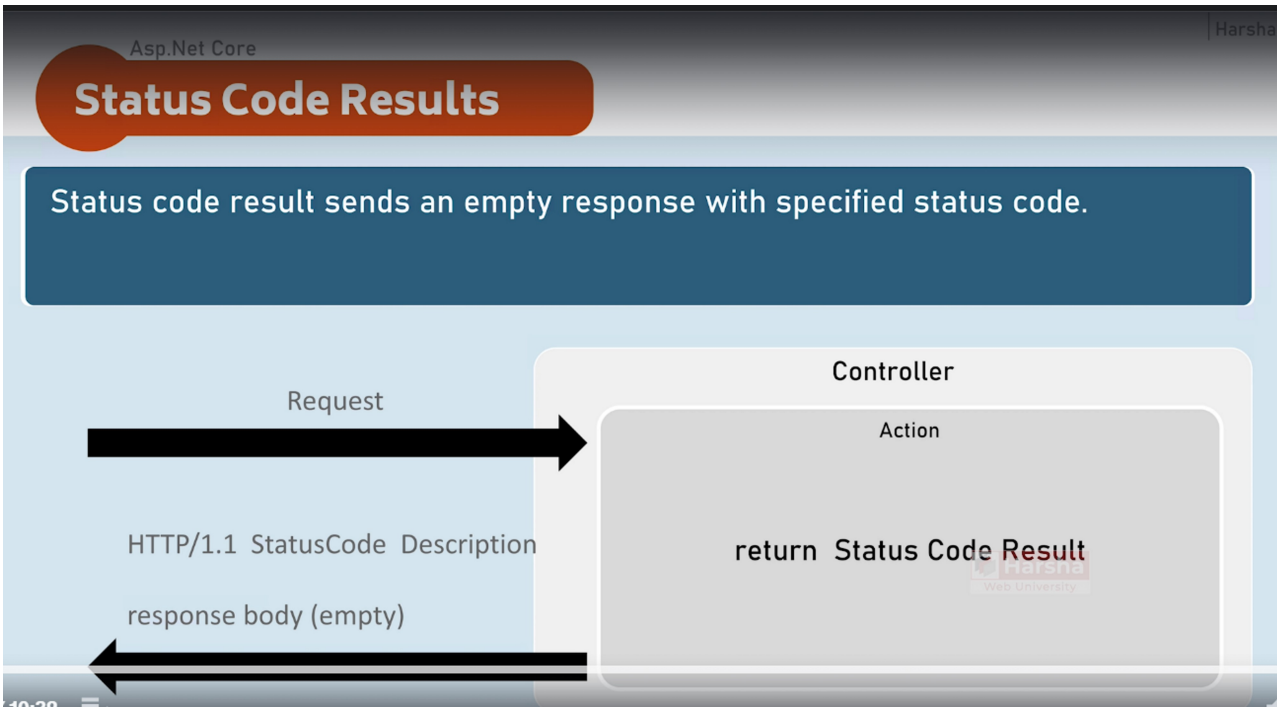
's (6) Body Pre-request Script Tests Settings

	VALUE



Asp.Net Core

Status Code Results



Asp.Net Core

Harsha

Status Code Results

StatusCodeResult

```
return new StatusCodeResult(status_code);
```

UnauthorizedResult

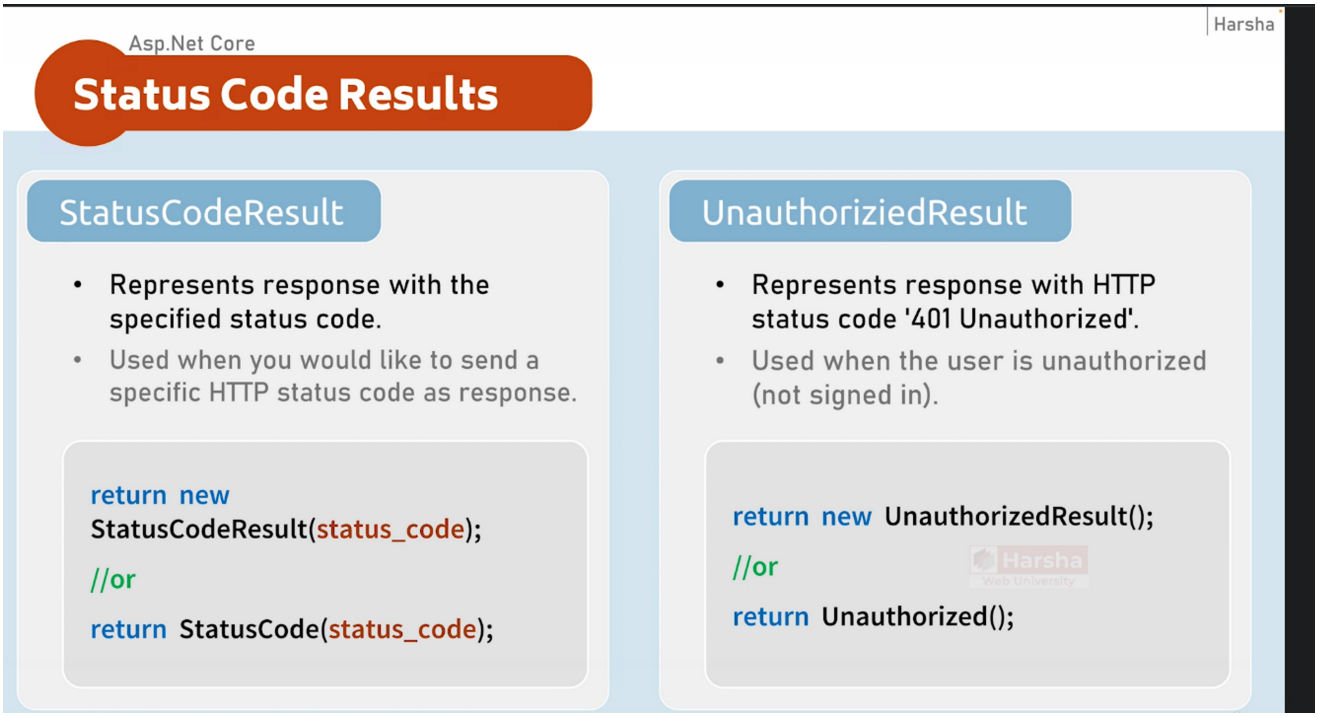
```
return new UnauthorizedResult();
```

BadRequestResult

```
return new BadRequestResult();
```

NotFoundResult

```
return new NotFoundResult();
```

Asp.Net Core

Status Code Results

BadRequestResult

- Represents response with HTTP status code '400 Bad Request'.
- Used when the request values are invalid (validation error).

```
return new BadRequestResult();
//or
return BadRequest();
```

NotFoundResult

- Represents response with HTTP status code '404 Not Found'.
- Used when the requested information is not available at server.

```
return new NotFoundResult();
//or
return NotFound();
```


Asp.Net Core

Redirect Results

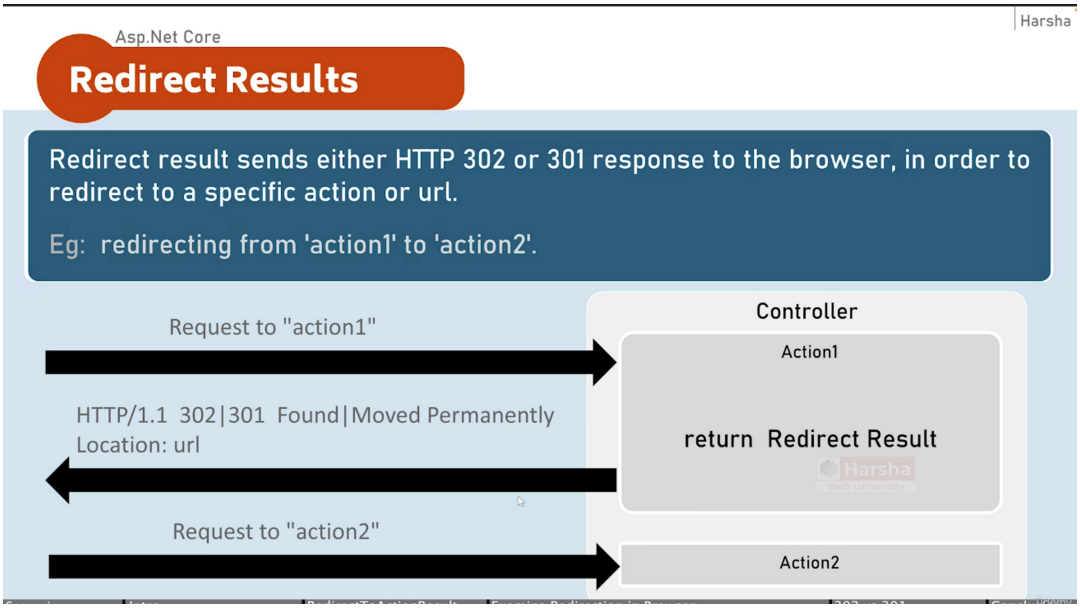
Part 1

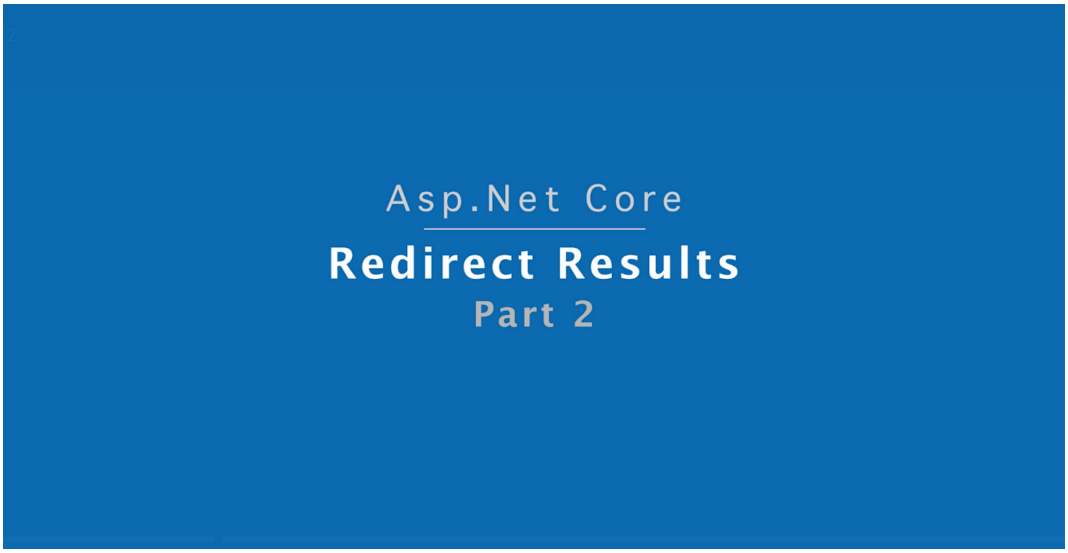
```
Redirect:

 /bookstore

 to

 /store/books
```





Asp.Net Core

Harsha

Redirect Results

RedirectToActionResult

```
return new RedirectToActionResult("action", "controller", new { route_values }, permanent);
```

LocalRedirectResult

```
return new LocalRedirectResult("local_url", permanent);
```

RedirectResult

```
return new RedirectResult("url", permanent);
```


Asp.Net Core

Harsha

RedirectToActionResult

- Represents response for redirecting from the current action method to another action method, based on action name and controller name.

302 - Found

```
return new RedirectToActionResult("action", "controller", new { route_values });
//or
return RedirectToAction("action", "controller", new { route_values });
```

301 - Moved Permanently

```
return new RedirectToActionResult("action", "controller", new { route_values }, true);
//or
return RedirectToActionPermanent("action", "controller", new { route_values });
```