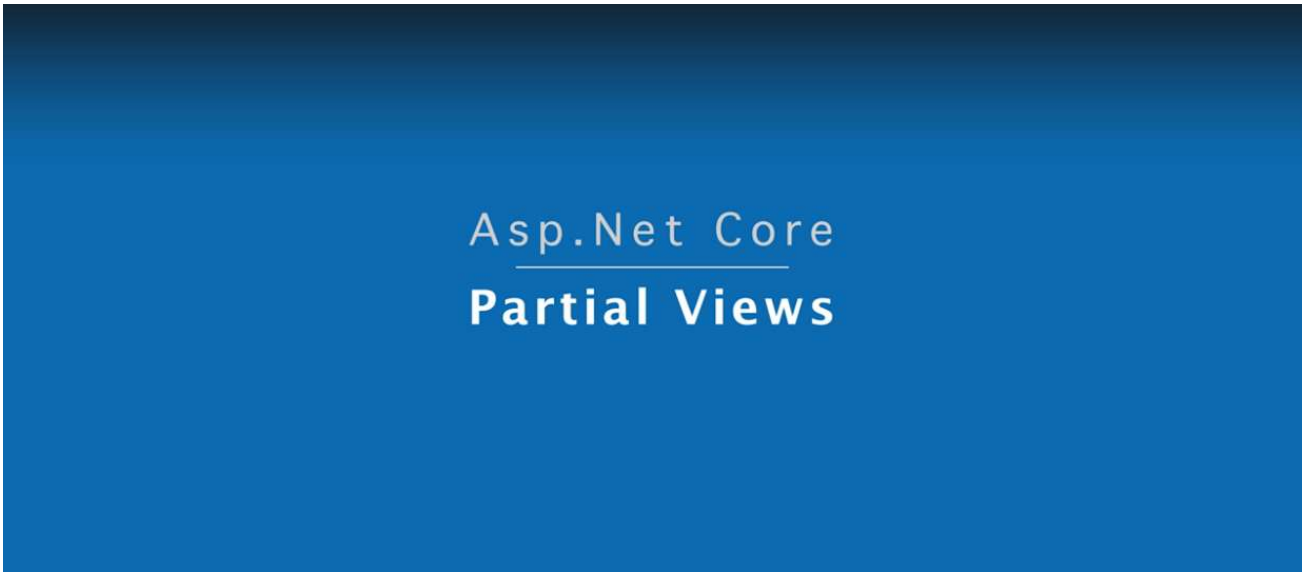
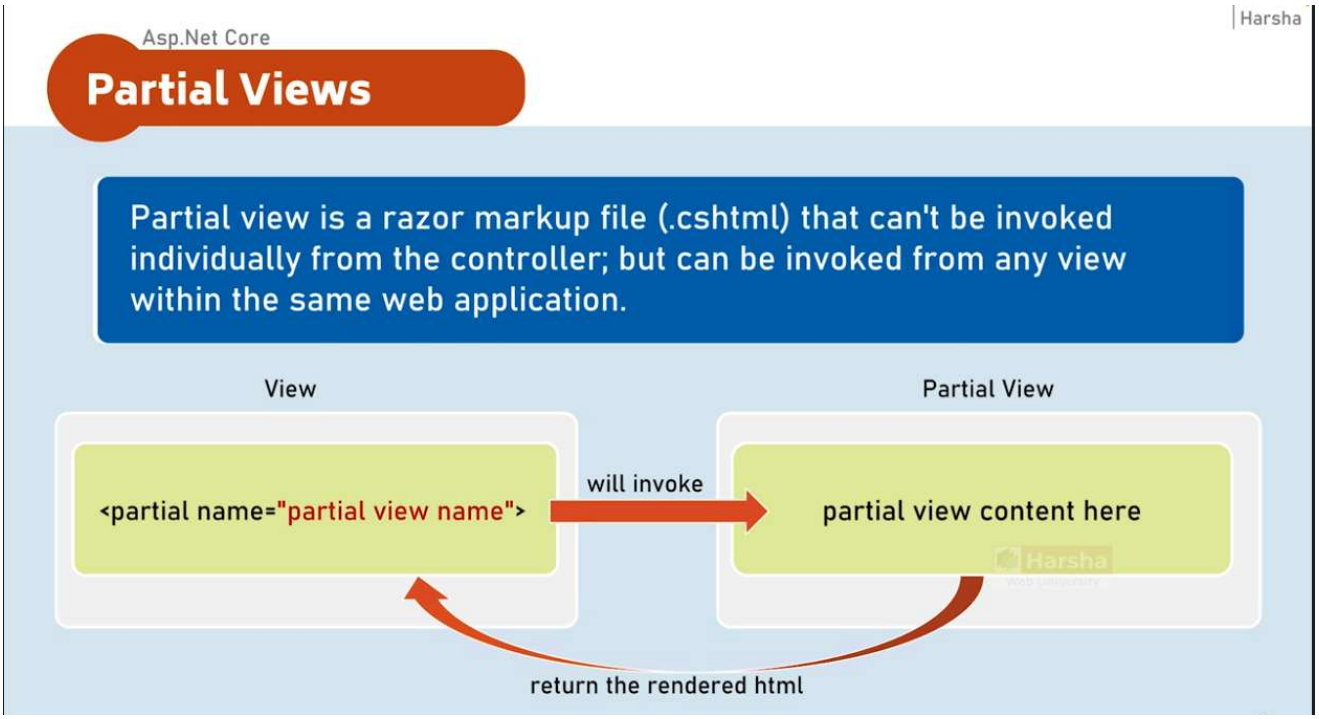
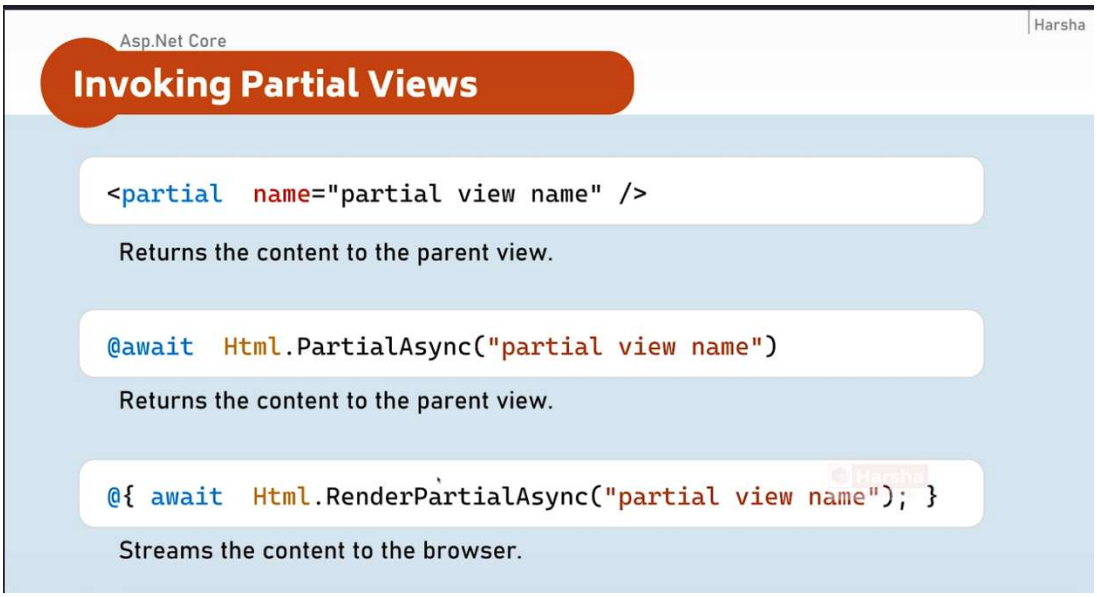
Section 10 : Partial Views [MVC]

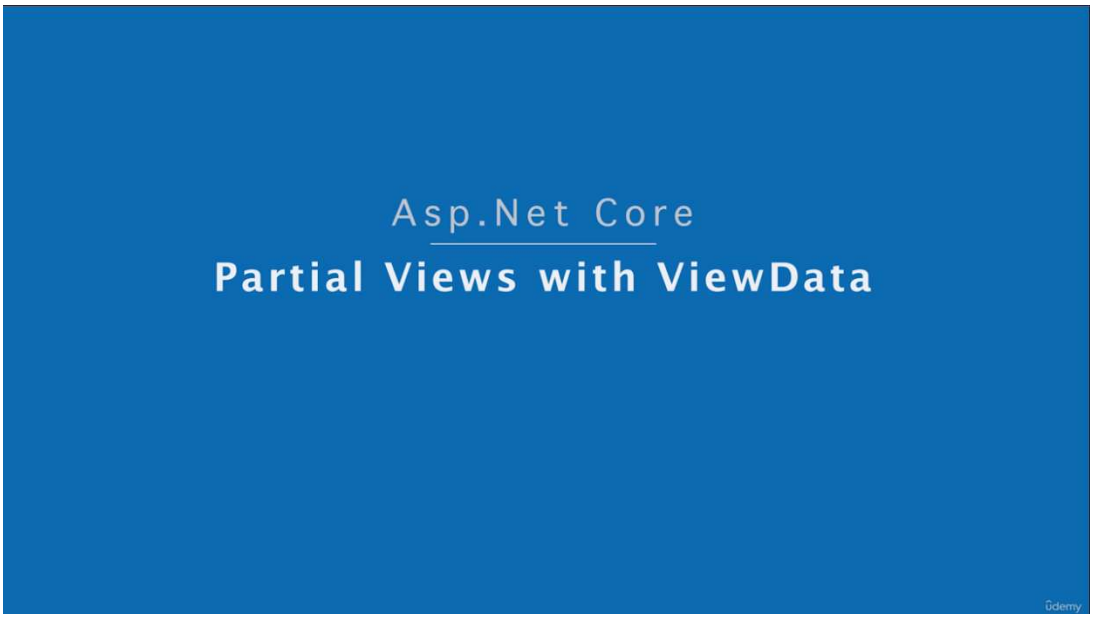
Thursday, January 16, 2025 1:04 PM

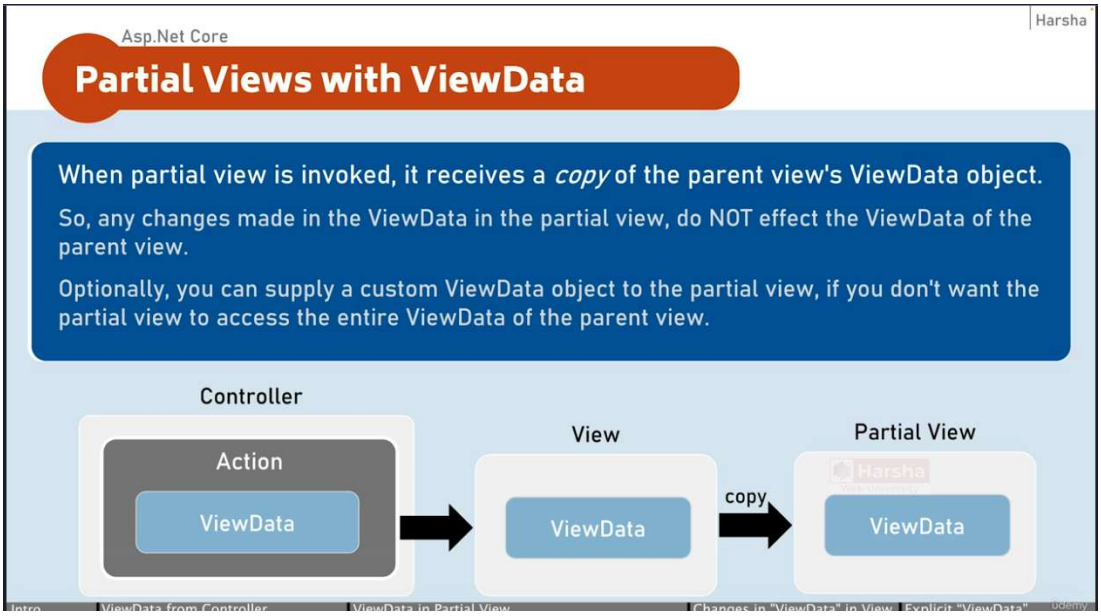


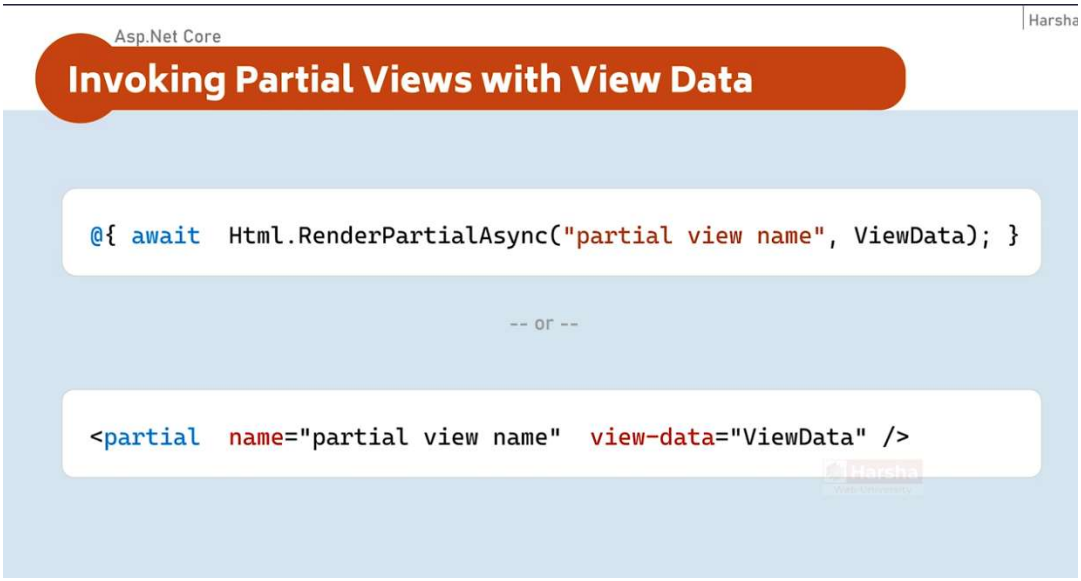


Keep your partial view file at Shared folder so that they can be globally accessible to all view.


Asp.Net Core
Harsha
Invoking Partial Views
<partial name="partial view name" />
Returns the content to the parent view.
@await Html.PartialAsync("partial view name")
Returns the content to the parent view.
@{ await Html.RenderPartialAsync("partial view name"); }
Streams the content to the browser.

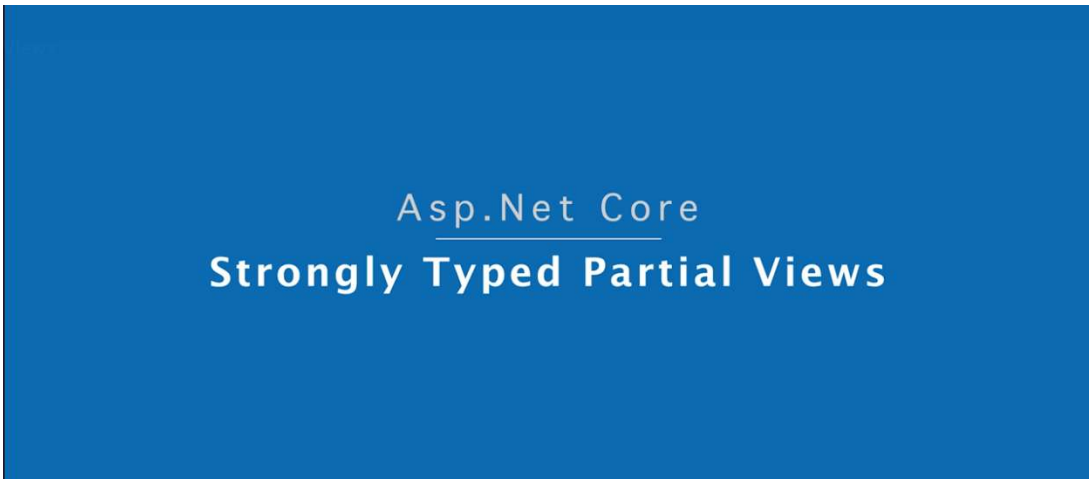

Asp.Net Core
Partial Views with ViewData
Udemy

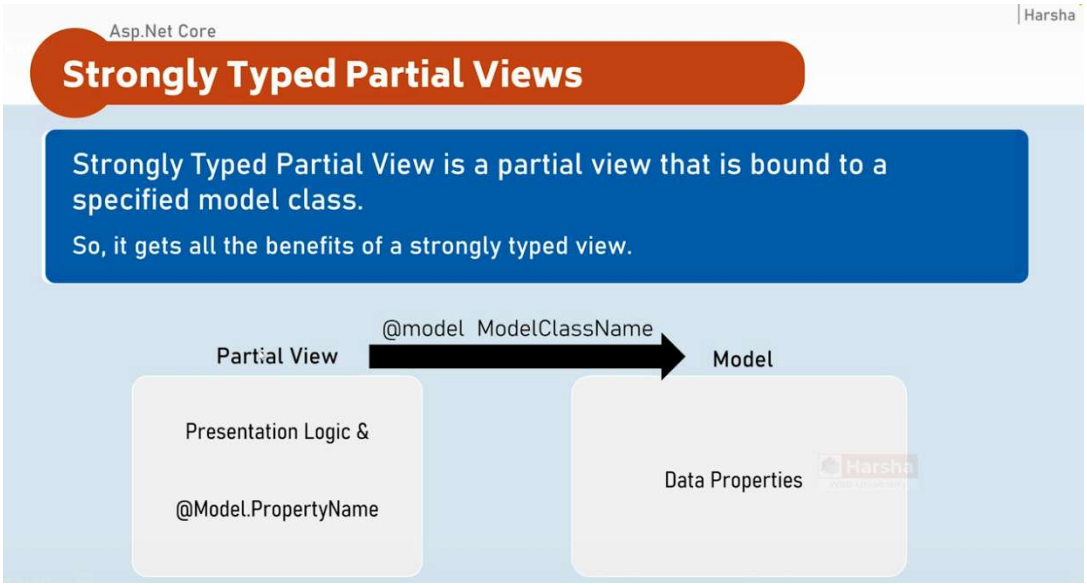




Sending Custom 'ViewData' object to Partial View.

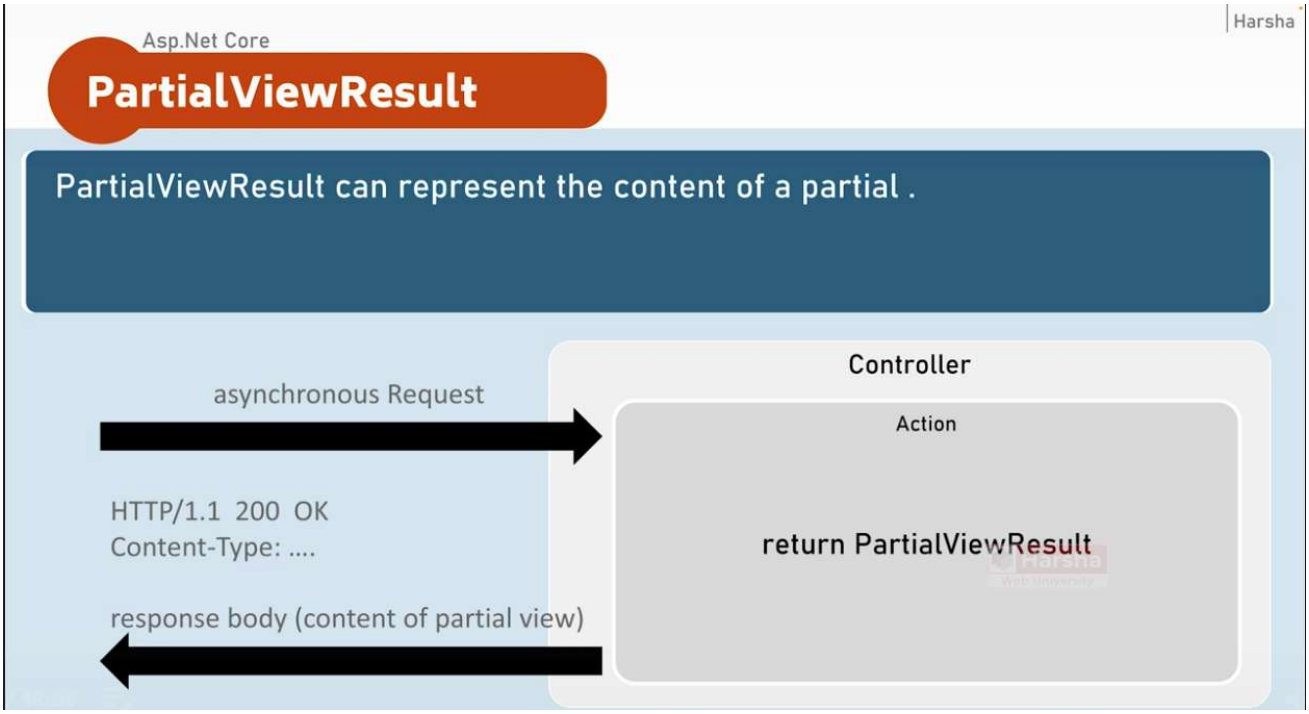
```
@{
 var myViewData = new ViewDataDictionary(ViewData);

 myViewData["ListTitle"] = "Countries";
 myViewData["ListItems"] = new List<string>() {
        "USA",
        "Canada",
        "Japan",
        "Germany",
        "India"
      };
}
<div class="box">
 <partial name="_ListPartialView"></partial>
</div>

<h3>ListTitle in View: @ViewData["ListTitle"]</h3>
```





There is another way to send the model object to the view: you can make the view a strongly typed view.

This means the partial view is tightly coupled with a specific model class or a list of the model class so that it can receive an object of the model class at runtime.
We have already made the normal views strongly typed views, but it is also possible to make the partial view a strongly typed view. Particularly when you have a model class to represent the structure of the data, meaning you have a model object to send to the partial view, it is best to make that view a strongly typed view.
In this case, you will write the @model directive in the partial view. Let me show that.



Already you know that from the controller action method, you can write a statement return View. Similarly, you can also return a partial view result from the action method. But why do we need to do that, and what is the purpose

behind it?
Generally, you return a partial view from the controller for asynchronous requests made from the browser. You can make an asynchronous request from the browser using JavaScript code, either with XMLHttpRequest or the fetch API in JavaScript.
In this case, the controller action method receives the asynchronous request from the browser, creates a model object, and passes that model object to the partial view by writing a statement to return a partial view result. Then, the partial view executes on the server side.
In the partial view, there can be server-side code, meaning C# code, and all that C# code executes on the server side. However, the actual HTML code of the partial view is sent as a response to the browser. This means the executed content of the partial view is added to the response body, and the content type is automatically set as text/html.
From the browser's perspective, you are making an asynchronous request and loading the information from the server asynchronously in the background without refreshing the full page.
So, when you want to load additional or extra content from the server, you can use this technique.

Let's say the page is already running in the browser. I will create a button on the page, and when I click on that button, I want to load a list of countries or some other list dynamically.
In such a scenario, you can use this technique.