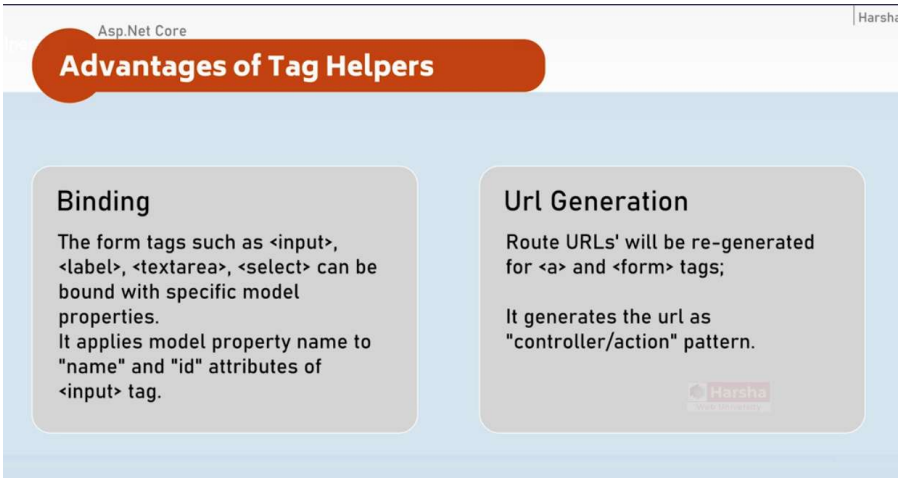
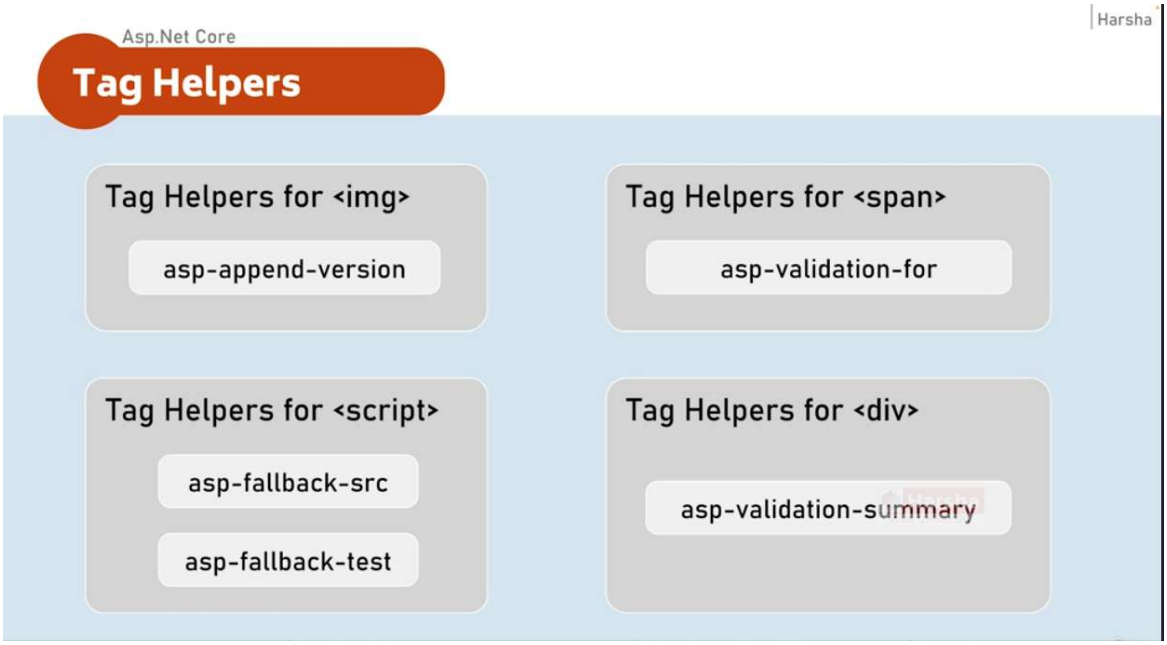
# Section 17: Tag Helpers [MVC]

Sunday, February 2, 2025 2:45 PM






Here's the cleaned-up and spell-checked version of your text:

**What is a Tag Helper?**

A Tag Helper can be invoked as an HTML tag or an HTML attribute in the view.
Tag Helpers provide values for attributes of existing HTML tags or generate new HTML tags.
For example, if you use a predefined Tag Helper called asp-for and specify the model property name for an input tag, it will automatically generate the type, name, id, and value attributes. The values for these attributes are generated on the server side and sent to the browser.
Instead of manually writing the type, name, id, and value attributes, you can simply use the shorthand asp-for. This makes it easier to generate attributes for specific purposes, keeping the code simple, clean, easy to understand, and easy to modify.

**What are the Predefined Tag Helpers?**

For the <a> (anchor) tag and <form> tag, the available Tag Helpers include asp-for, asp-action, and others.
The hyphen (-) in asp-for means a dash, not an underscore. Most predefined Tag Helpers begin with asp-, which is a common prefix for them.
For <input>, <textarea>, and <label> tags, asp-for is the predefined Tag Helper that binds the form element to a model property. We will see this in action in later lectures—don't worry about it now.
Similarly, for the <select> tag, you can use asp-for just like with the <input> tag. Additionally, you can use asp-items to specify a list of items that will generate <option> tags inside the <select> element.

**What Are the Benefits of Using Tag Helpers?**

1. **Simplifies Model Binding**
   - For <input>, <label>, <textarea>, and <select>, asp-for allows you to bind form elements to a model property easily.
   - Instead of manually writing name and id attributes that match the model property name, asp-for generates them automatically.
2. **Automatically Generates URLs**
   - In the case of <form> and <a> (anchor) tags, Tag Helpers generate URLs dynamically.
   - Instead of hardcoding URLs in hyperlinks and form actions, they are generated based on the specified action name and controller name.

These benefits make Tag Helpers a common and default feature in real-world projects. You will rarely see professional

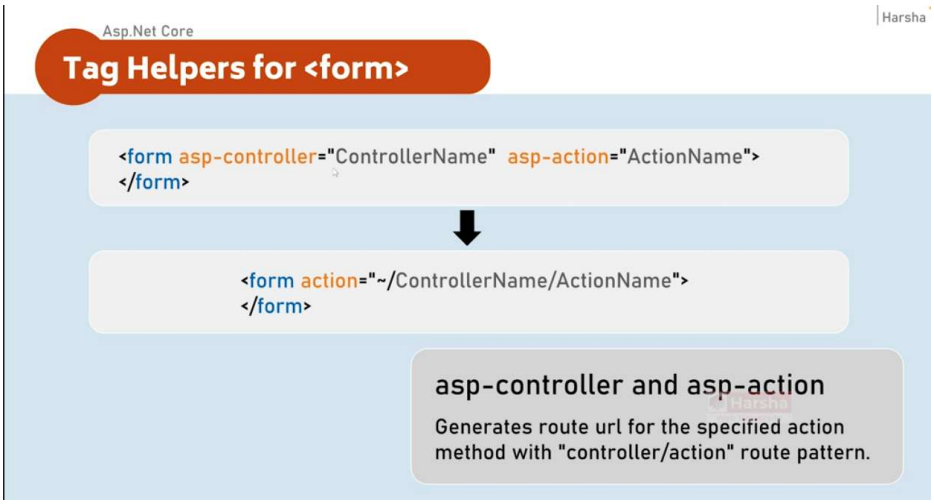
development projects that don't use them.

### Other Predefined Tag Helpers

Just like <input>, <select>, and <a> tags, other HTML elements also have predefined Tag Helpers:

- **<img> tag** → asp-append-version
- **<script> tag** → asp-fallback-src, asp-fallback-test
- ** tag** → asp-validation-for
- ** tag** → asp-validation-summary

We will explore each of these Tag Helpers in detail based on their purpose in this section.


For the <form> tag and <a> (anchor) tag in HTML, you can use two predefined Tag Helpers:
That is asp-controller and asp-action.
In asp-controller, you can specify the controller name, and in asp-action, you specify the action name so that it will generate the URL automatically and place it in the action attribute.
It first takes the controller name and then appends a slash followed by the action name. Based on this pattern, it generates the URL automatically using the specified controller and action names.
Exactly in the same way, for the <a> (anchor) tag, if you specify the controller name and action name in asp-controller and asp-action, the URL will be generated in the href attribute.

### What's the Benefit for You as a Developer?

- The URL is generated automatically based on the specified controller and action name.
- Even if you move the view from one controller to another, it still generates the correct URL.
- The same syntax works for any view in any controller.
- You can supply the controller and action names programmatically using variables.
- For example, if you pass values from the controller through ViewBag, those values can be used in asp-controller and asp-action.

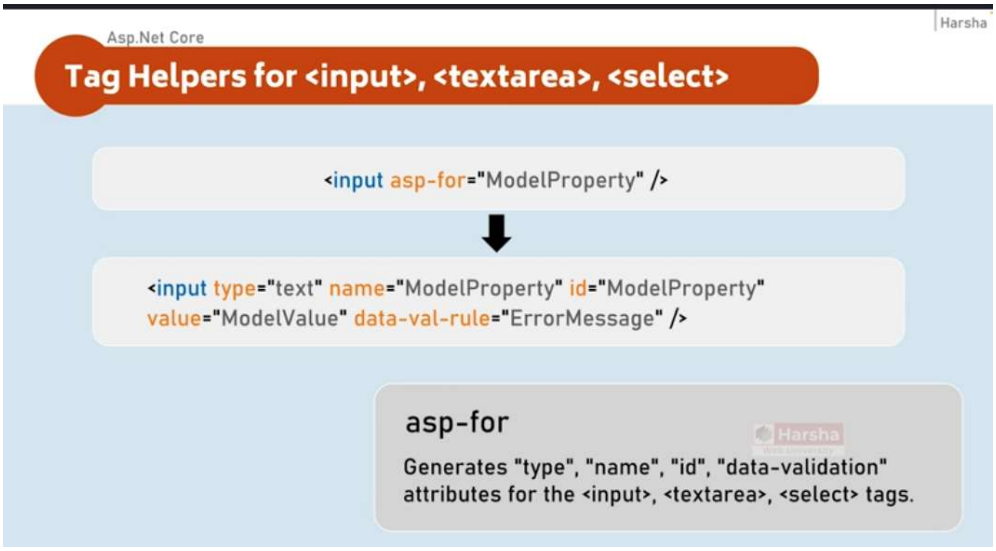
That is the benefit of using Tag Helpers.

Let me demonstrate here.

Asp.Net Core

# Tag Helpers for <input>

Part 1




2

Asp.Net Core

# Tag Helpers for <input>

Part 2

Harsha

Asp.Net Core

## Tag Helpers for <label>

```
<label asp-for="ModelProperty"> </label>
```

⬇

```
<label for="ModelProperty"> </label>
```

**asp-for**

Generates "for" attribute for the <label>.

Asp.Net Core

# Client Side Validations

Harsha

Asp.Net Core

## Client Side Validations

Data annotations on model properties

```
[Required]
public DataType PropertyName { get; set; }
```

"data-*" attributes in html tags
[auto-generated with "asp-for" helper]

```
<input data-val="true" data-
required="ErrorMessage" />
```

Import jQuery Validation Scripts

```
https://cdnjs.cloudflare.com/ajax/libs/
jquery/3.6.0/jquery.min.js
```

```
https://cdnjs.cloudflare.com/ajax/libs/
jquery-validate/1.19.3/jquery.validate.min.js
```

```
https://cdnjs.cloudflare.com/ajax/libs/
jquery-validation-unobtrusive/3.2.12/
jquery.validate.unobtrusive.min.js
```

By default, client-side validations are not enabled in ASP.NET Core applications.
If you want, you have to enable it manually.
See, for example, in this Create Person's view:
I just opened the developer tools with the Network tab and did not enter anything in the person name and other text boxes—all the text boxes are empty.
So I just clicked on the Create button without entering anything.
See, even though there are validation errors (i.e., person name, email, etc., are blank), it makes a POST request to Persons/Create.
It is not correct, right?
In fact, there is no need to make a POST request to Create since there are validation errors. The view should have checked the client-side validations in the browser itself before making the POST request to Persons/Create.
That is exactly what is called client-side validation, and it is not enabled by default.
You have to enable it manually.

**So how to do that?**
These are the three steps to enable client-side validations:
1. **Data Annotations**
2. **Data Attributes**
3. **Importing jQuery Validation Scripts**

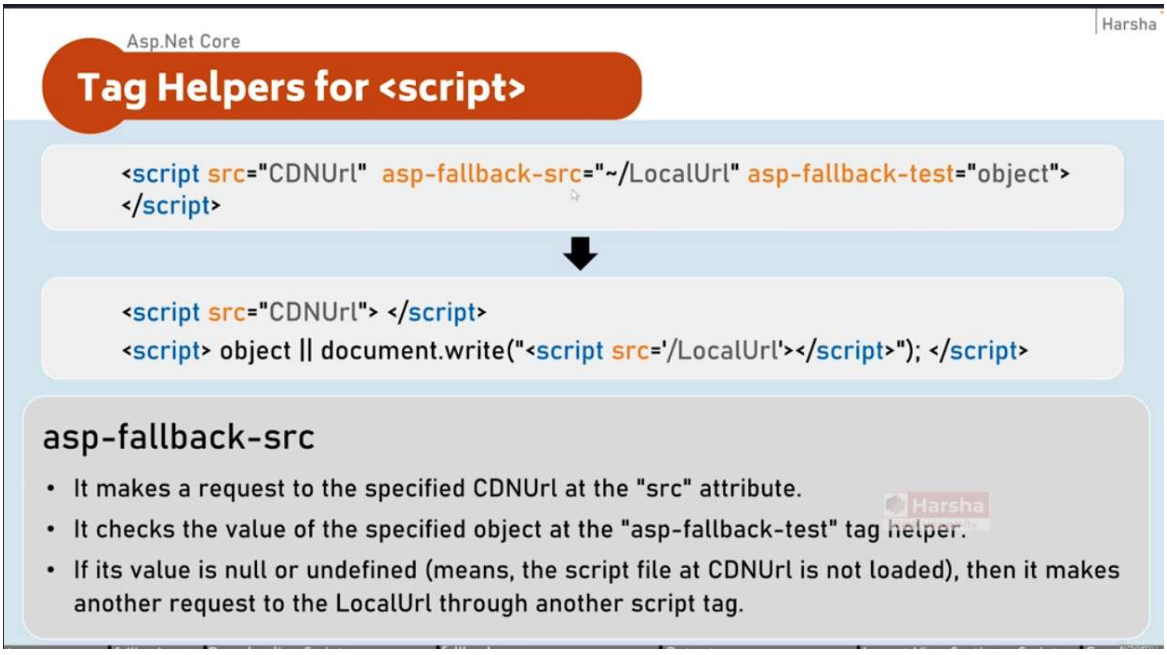
**Step 1: Adding Data Annotations**
Through data annotation attributes, you can specify validation rules in the model class itself.
For example, if you look at the PersonAddRequest, we have already enabled client-side validations with Required, Range, and EmailAddress attributes.
You already know how to apply validation rules for model properties, so we have covered the first step in previous lectures.
**Step 2: Data Attributes**
These are automatically generated when using the Tag Helper asp-for.
For example, if you apply the Required attribute to a property and then use asp-for to specify the model property name in the view, it automatically generates data attributes.
That means attributes beginning with data-.
For example:
data-val="true"
This means the field is mandatory.
data-val-required="Person name cannot be blank"
This represents the actual error message for the Required attribute.
Observe the PersonName property:
- The Required attribute is applied to PersonName.
- In the Create view, the input tag is created using asp-for="PersonName".

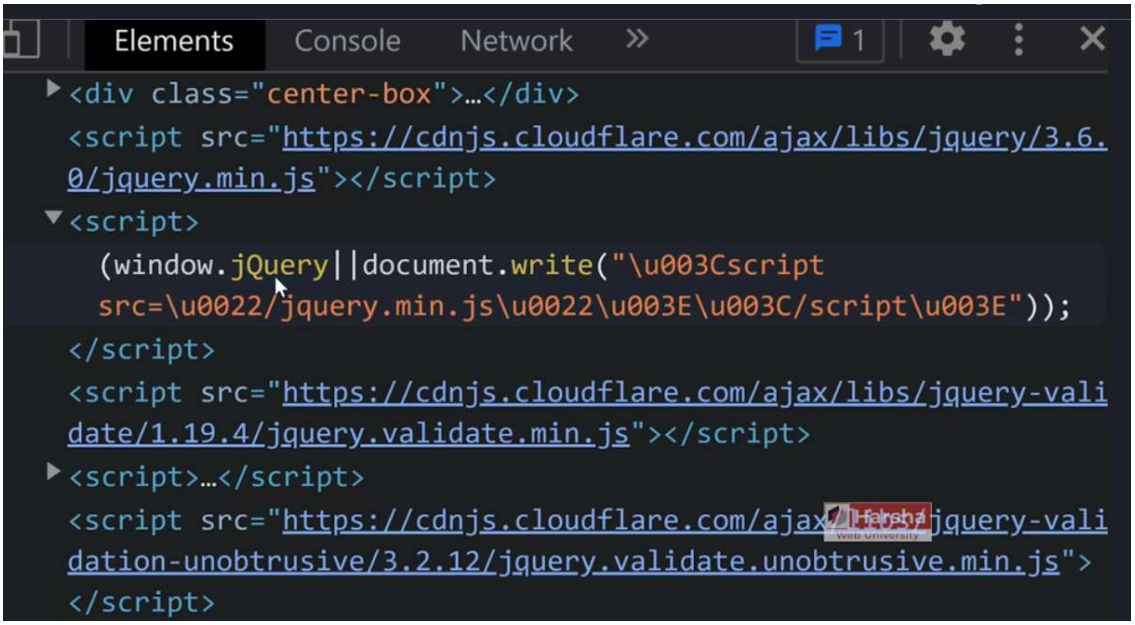
This means the input tag, which represents the PersonName property, will automatically include all validation rules as data-attributes.
Let me show that:
As soon as you use asp-for for the input tag, it automatically generates:
data-val="true"
data-val-required="Person name cannot be blank"
These types of attributes are generated automatically when using data annotations along with asp-for.

**Step 3: Importing jQuery Validation Scripts**
To actually apply client-side validations using these data attributes, you need to load jQuery validation scripts.
1. First, load the jquery.js file.
2. Then, load the jquery.validate.unobtrusive.js file.

Asp.Net Core

# Tag Helpers for <script>



Asp.Net Core

# Tag Helpers for <img>

Harsha

Asp.Net Core

## Tag Helpers for <img>

```
<img src="~/FilePath"  asp-append-version="true" />
```

⬇

```
<img src="/FilePath?v=HashOfImageFile" />
```

### asp-append-version

- Generates SHA512 hash of the image file as query string parameter appended to the file path.
- It REGENERATES a new hash every time when the file is changed on the server.
- If the same file is requested multiple times, file hash will NOT be regenerated.

Asp.Net Core

# Edit View

Asp.Net Core

# Delete View