ASP.NET Core - True Ultimate Guide

Section 17: Tag Helpers - Notes

Tag Helpers

Tag helpers are a feature in ASP.NET Core MVC that allow you to extend HTML elements with server-side capabilities. They are C# classes that modify the behavior and output of HTML elements during the rendering process.

Benefits of Tag Helpers

- **HTML-Friendly Syntax:** Tag helpers look like standard HTML elements, making them easier to read and write than traditional HTML helpers.
- **Strong Typing:** Tag helpers offer compile-time type safety and IntelliSense support, catching errors early in development.
- **Code Reuse:** They can be easily reused across different views and projects.
- **Reduced Server Roundtrips:** Tag helpers execute on the server, allowing you to perform complex logic and data binding before the page is sent to the client.
- **Extensibility:** You can create your own custom tag helpers to meet specific needs.

When to Use Tag Helpers

- **Form Handling:** Create forms and bind them to your models easily.
- **Links and URLs:** Generate links with correct routing information.
- **Caching:** Control how your views are cached.
- **Conditional Rendering:** Show or hide content based on conditions.
- **Custom Elements:** Build reusable custom UI components.

Best Practices

- **Namespace Management:** Keep tag helper namespaces organized (e.g., @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers).
- **Readability:** Keep tag helper attributes concise and self-explanatory.
- **Performance:** Be mindful of the number of tag helpers used in a view, as they can impact rendering performance.
- **Testing:** Write unit tests for your custom tag helpers to ensure their correct behavior.

Things to Avoid

- **Overusing Tag Helpers:** Use them for appropriate tasks, not for every HTML element.
- **Excessive Nesting:** Avoid deeply nested tag helpers, as it can make your code difficult to read.
- **Mixing Tag Helpers and HTML Helpers:** Try to use either tag helpers or HTML helpers consistently within a view to maintain a cleaner code structure.

Important Tag Helpers with Examples

1. **Anchor Tag Helper (<a>):**

   <a asp-controller="Home" asp-action="Index">Home

   - Generates a link to the Index action method in the HomeController.
   - Automatically handles routing and URL generation.

2. **Form Tag Helper (<form>):**

   <form asp-controller="Products" asp-action="Create" method="post">

   </form>

   - Creates a form that submits data to the Create action in the ProductsController.
   - Handles anti-forgery tokens automatically for better security.

3. **Input Tag Helper (<input>):**

   <input asp-for="ProductName" class="form-control" />

   - Binds the input field to the ProductName property of your model.
   - Automatically sets the input type (e.g., text, email, password) based on the property type.

4. **Select Tag Helper (<select>):**

   <select asp-for="CategoryId" asp-items="Model.Categories"></select>

   - Creates a dropdown list bound to the CategoryId property.
   - asp-items takes a collection of items to populate the dropdown.

5. **Label Tag Helper (<label>):**

   <label asp-for="ProductName"></label>

   - Generates a label for the ProductName input field, automatically setting the for attribute to match the input's ID.

6. **Cache Tag Helper (<cache>):**

   <cache expires-after="@TimeSpan.FromMinutes(10)">

   Content to cache

   </cache>

   - Caches the enclosed content for the specified duration.
   - Improves performance for content that doesn't change frequently.

7. **Environment Tag Helper (<environment>):**

   <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />

   - The asp-append-version attribute automatically adds a version query string to the URL in non-development environments, which helps with cache busting when you deploy updates.

Controllers

- **Index (Read):**
  - HTTP Verb: GET
  - Purpose: Displays a list or table of entities (e.g., persons).
  - Logic:
    1. Retrieves data from the PersonsService using methods like GetFilteredPersons and GetSortedPersons.
    2. Populates ViewBag with:

        - SearchFields: A dictionary of searchable fields and their display names.
        - CurrentSearchBy: The currently selected search field.
        - CurrentSearchString: The current search term.
        - CurrentSortBy: The current sorting field.
        - CurrentSortOrder: The current sort order (ASC or DESC).
    3. Returns the Index view with the filtered and sorted data.

- **Create (Create):**
    - HTTP Verbs: GET (Display form), POST (Process submission)
    - Purpose: Creates a new entity.
    - Logic:
        - **GET:**
            1. Retrieves a list of countries from CountriesService for populating the "Country" dropdown in the form.
            2. Returns the Create view.
        - **POST:**
            1. Receives PersonAddRequest via model binding.
            2. Validates the model state.
            3. If valid, calls _personsService.AddPerson to create the new person and redirects to the Index action.
            4. If invalid, repopulates ViewBag.Countries and ViewBag.Errors and returns the Create view with error messages.

- **Edit (Update):**
    - HTTP Verbs: GET (Display form), POST (Process submission)
    - Purpose: Updates an existing entity.
    - Logic:
        - **GET:**
            1. Retrieves the person to edit using _personsService.GetPersonByPersonID.

2. Retrieves a list of countries from CountriesService for the dropdown.
3. Returns the Edit view with the person's data in a PersonUpdateRequest.

- **POST:**
  1. Receives PersonUpdateRequest via model binding.
  2. Validates the model state.
  3. If valid, calls _personsService.UpdatePerson and redirects to the Index action.
  4. If invalid, repopulates ViewBag.Countries and ViewBag.Errors and returns the Edit view with error messages.

- **Delete (Delete):**
  - HTTP Verbs: GET (Display confirmation), POST (Perform deletion)
  - Purpose: Deletes an existing entity.
  - Logic:
    - **GET:**
      1. Retrieves the person to delete using _personsService.GetPersonByPersonID.
      2. Returns the Delete view to confirm the deletion.
    - **POST:**
      1. Receives PersonUpdateRequest (containing the PersonID) via model binding.
      2. Calls _personsService.DeletePerson and redirects to the Index action.

Views

- **Index.cshtml (Read):**
  - Displays a table of persons.
  - Uses tag helpers (asp-controller, asp-action, etc.) to generate links.

    - Includes a form for searching and sorting.
    - Renders a partial view (_GridColumnHeader) to create sortable table headers.

- **Create.cshtml:**
    - Renders a form for creating a new person.
    - Uses tag helpers for model binding and validation.
    - Displays validation errors using asp-validation-summary and asp-validation-for.

- **Edit.cshtml:**
    - Similar to Create.cshtml but for editing an existing person.

- **Delete.cshtml:**
    - Displays a confirmation message and a form to confirm the deletion.
    - Uses tag helpers for binding the PersonID.

Client-Side Validations

Client-side validations are enabled in these views through the inclusion of jQuery, jQuery Validate, and jQuery Unobtrusive Validation libraries in the @section scripts block. These libraries work together to provide:

- **Instant Feedback:** Validation messages appear immediately when the user interacts with the form fields.
- **Reduced Server Load:** Validations are performed on the client-side, reducing the number of round trips to the server.

HttpPost Action Method Submission Process

1. **Form Submission:** The user fills out the form and clicks the submit button.
2. **Client-Side Validation (Optional):** If enabled, JavaScript validation checks are performed before the form is submitted to the server. If there are errors, they are displayed immediately, and the submission is prevented.
3. **Request Sent to Server:** If there are no client-side errors, the form data is sent to the server via a POST request.

4. **Model Binding:** ASP.NET Core's model binding system extracts the form data and attempts to create a model object (PersonAddRequest or PersonUpdateRequest) based on the form field names.
5. **Model Validation:** Data annotations and custom validation rules are applied to the model object. If errors are found, they are added to the ModelState object.
6. **Controller Action Logic:**
    - If ModelState.IsValid is true, the action performs the appropriate CRUD operation (create, update, delete) using the service layer.
    - If ModelState.IsValid is false, the action typically returns the view again, repopulating the form with the user's input and displaying error messages.
7. **Redirect (Optional):** After a successful POST request, the action often redirects to another page (e.g., the "Index" view) to prevent accidental re-submissions.

Key Points to Remember

Tag Helpers

- **Purpose:** Server-side code that modifies HTML elements to include server-side logic.
- **Benefits:**
    - HTML-friendly syntax
    - Strong typing and IntelliSense
    - Code reuse
    - Reduced server round-trips
- **Common Tag Helpers:**
    - a: Creates links (e.g., asp-controller, asp-action).
    - form: Generates HTML forms (e.g., asp-controller, asp-action, method).
    - input, textarea, select: Bind to model properties (e.g., asp-for).
    - label: Creates labels for form fields (asp-for).
    - cache: Caches a portion of the view.
    - partial: Renders a partial view.
    - environment: Conditionally renders content based on the environment.

CRUD Operations with Tag Helpers

Index (Read)

- **a (Anchor):** Create links to Create, Edit, and Delete actions.

  ```
  <a asp-action="Create">Create</a>
  <a asp-action="Edit" asp-route-id="@item.Id">Edit</a>
  <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
  ```

- **form (Form):** Create a form for filtering or searching.

  ```
  <form asp-action="Index" method="get">
    <input type="text" name="searchString" />
    <button type="submit">Search</button>
  </form>
  ```

Create & Edit

- **form:** Create the form for submitting data.

  ```
  <form asp-action="Create" method="post">
  </form>
  ```

- **input, textarea, select:** Bind to model properties.

  ```
  <input asp-for="Name" />
  <textarea asp-for="Description"></textarea>
  <select asp-for="CategoryId" asp-items="ViewBag.Categories"></select>
  ```

- **label:** Generate labels for input fields.

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

- **span (Validation):** Display validation messages.

  ```
  <span asp-validation-for="Name"></span>
  ```

- **div (Validation Summary):** Summarize validation errors.

  ```
  <div asp-validation-summary="All"></div>
  ```

Delete

- **form:** Create a form that submits a delete request.HTML

  ```
  <form asp-action="Delete" asp-route-id="@Model.Id" method="post">
    <button type="submit">Delete</button>
  </form>
  ```

Key

- **Understanding:** Explain the benefits of tag helpers over traditional HTML helpers.
- **Usage:** Demonstrate how to use common tag helpers in CRUD scenarios.
- **Model Binding and Validation:** Show how to use tag helpers to bind to models and display validation errors.
- **Best Practices:** Discuss how to write clean, maintainable, and reusable code with tag helpers.
- **Performance Considerations:** Explain how tag helpers can impact performance and how to mitigate potential issues (e.g., caching).
- **Security:** Emphasize the importance of input validation and output encoding to prevent XSS vulnerabilities.