Unit 8
Client Side Development in Asp.Net Core

Common Client-Side Web Strategies

- ASP.NET Core applications are web applications and they typically rely on client-side web technologies like HTML, CSS, and JavaScript.
- By separating the content of the page (the HTML) from its layout and styling (the CSS), and its behavior (via JavaScript), complex web apps can leverage the Separation of Concerns principle.
- Future changes to the structure, design, or behavior of the application can be made more easily when these concerns are not intertwined.
- While HTML and CSS are relatively stable, JavaScript, by means of the application frameworks and utilities developers work with to build web-based applications, is evolving at breakneck speed. For example, Angular JS and React JS client-side libraries.

HTML

- HTML is the standard markup language used to create web pages and web applications.
- Its elements form the building blocks of pages, representing formatted text, images, form inputs, and other structures.
- When a browser makes a request to a URL, whether fetching a page or an application, the first thing that is returned is an HTML document.
- This HTML document may reference or include additional information about its look and layout in the form of CSS, or behavior in the form of JavaScript.

CSS

- CSS (Cascading Style Sheets) is used to control the look and layout of HTML elements.
- CSS styles can be applied directly to an HTML element, defined separately on the same page, or defined in a separate file and referenced by the page.
- Styles cascade based on how they are used to select a given HTML element.
- It's best to keep styles in their own separate stylesheet files, and to use selection-based cascading to implement consistent and reusable styles within the application.

JavaScript

- JavaScript is a dynamic, interpreted programming language of the web which is used to enhance the html page.
- HTML pages are fine for displaying static content, e.g. a simple image or text. However, most pages nowadays are rarely static. Many of today's pages have menus, forms, slideshows and even images that provide user interaction. Javascript is the language employed by web developers to provide such interaction.
- Like CSS, JavaScript can be defined as attributes within HTML elements, as blocks of script within a page, or in separate files.
- Just like CSS, it's recommended to organize JavaScript into separate files, keeping it separated as much as possible from the HTML found on individual web pages or application views.
- When working with JavaScript in your web application, there are a few tasks that you'll commonly need to perform:
    - **Selecting an HTML element and retrieving and/or updating its value.**
    - Querying a Web API for data (AJAX Call).
    - Sending a command to a Web API (and responding to a callback with its result).
    - **Performing validation.**
- We can perform all of these tasks with JavaScript alone, but many libraries exist to make these tasks easier.
- One of the first and most successful of these libraries are **jQuery**, which continues to be a popular choice for simplifying these tasks on web pages.
- **For Single Page Applications (SPAs), jQuery doesn't provide many of the desired features that Angular and React offer.**

Legacy web apps with jQuery

- Although ancient by JavaScript framework standards, jQuery continues to be a commonly used library for working with HTML/CSS and building applications that make AJAX calls to web APIs. However, jQuery operates at the level of the browser document object model (DOM), and by default offers only an imperative, rather than declarative, model.
- For example, imagine that if a textbox's value exceeds 10, an element on the page should be made visible. In jQuery, this functionality would typically be implemented by writing an event handler with code that would inspect the textbox's value and set the visibility of the target element based on that value. This process is an imperative, code-based approach. Another framework might instead use databinding to bind the visibility of the element to the value of the textbox declaratively. This approach would not require writing any code, but instead only requires decorating the elements involved with data binding attributes. As client-side behaviors grow more complex, data binding approaches frequently result in simpler solutions with less code and conditional complexity.
- Most of the features jQuery lacks intrinsically can be added with the addition of other libraries. However, a SPA framework like Angular provides these features in a more integrated fashion, since it's been designed with all of them in mind from the start. Also, jQuery is an imperative library, meaning that you need to call jQuery functions in order to do anything with jQuery. Much of the work and functionality that SPA frameworks provide can be done declaratively, requiring no actual code to be written.
- Data binding is a great example of this functionality. In jQuery, it usually only takes one line of code to get the value of a DOM element or to set an element's value. However, you have to write this code anytime you need to change the value of the element, and sometimes this will occur in multiple functions on a page. Another common example is element visibility. In jQuery, there might be many different places where you'd write code to control whether certain elements were visible. In each of these cases, when using data binding, no code would need to be written. You'd simply bind the value or visibility of the elements in question to a viewmodel on the page, and changes to that viewmodel would automatically be reflected in the bound elements.

Forms and Validations

- Before submitting data to the server, it is important to ensure all required form controls are filled out, in the correct format. This is called **client-side form validation**, and helps ensure data submitted matches the requirements set forth in the various form controls.
- Client-side validation is an initial check and an important feature of good user experience; by catching invalid data on the client-side, the user can fix it straight away. If it gets to the server and is then rejected, a noticeable delay is caused by a round trip to the server and then back to the client-side to tell the user to fix their data.
- However, client-side validation *should not be considered* an exhaustive security measure! Your apps should always perform security checks on any form-submitted data on the *server-side* **as well** as the client-side, because client-side validation is too easy to bypass, so malicious users can still easily send bad data through to your server.

Different types of client-side validation

1, **Built-in form validation**:

- It uses HTML5 form validation features.
- This validation generally doesn't require much JavaScript.
- Built-in form validation has better performance than JavaScript, but it is not as customizable as JavaScript validation.
- One of the most significant features of HTML5 form controls is the ability to validate most user data without relying on JavaScript. This is done by using validation attributes on form elements. Some of them are
    1. required: Specifies whether a form field needs to be filled in before the form can be submitted.
       Example:

```
<!doctype html>
<html>
<body>
<form>
  <label for="Name">Full Name</label>
  <input id="Name" name="Name" required>
  <button>Submit</button>
</form>
</body>
</html>
```

    2. minlength and maxlength: Specifies the minimum and maximum length of textual data (strings)

```
<!doctype html>
```

```
<html>
<body>
<form>
  <label for="Name">Full Name</label>
<input type="text" id="Name" name="Name" required minlength="3" maxlength="8">
  <button>Submit</button>
</form>
</body>
</html>
```

3. min and max: Specifies the minimum and maximum values of numerical input types

```
<!doctype html>
<html>
<body>
<form>
  <label for="Roll">Enter Roll Number</label>
<input type="number" id="Roll" name="Roll"min="1" max="36">
  <button>Submit</button>
</form>
</body>
</html>
```

4. type: Specifies whether the data needs to be a number, an email address, or some other specific preset type.

```
<!doctype html>
<html>
<body>
<form>
  <label for="EmailAddress">Enter Email Address</label>
<input type="email" id="EmailAddress" name="EmailAddress">
  <button>Submit</button>
</form>
</body>
</html>
```

5. pattern: Specifies a regular expression that defines a pattern the entered data needs to follow. Some example of regular expressions is listed below

- `a` — Matches one character that is `a` (not `b`, not `aa`, and so on).
- `abc` — Matches `a`, followed by `b`, followed by `c`.
- `ab?c` —Matches `a`, optionally followed by a single `b`, followed by `c`. (`ac` or `abc`)
- `ab*c` —Matches `a`, optionally followed by any number of `b`s, followed by `c`. (`ac` , `abc` , `abbbbbc` , and so on).
- `a|b` — Matches one character that is `a` or `b`.
- `abc|xyz` — Matches exactly `abc` or exactly `xyz` (but not `abcxyz` or `a` or `y`, and so on).

Example:

```
<!doctype html>
<html>
<body>
<form>
  <label for="Name">Day</label>
<input id="Name" name="Name" required
pattern="[Ss]unday|[Mm]onday|[Tt]uesday|[Tt]hursday|[Ff]riday|[Ss]aturday">
  <button>Submit</button>
</form>
</body>
</html>
```

- If the data entered in a form field follows all of the rules specified by the above attributes, it is considered valid. If not, it is considered invalid.

2. **JavaScript**:

- This validation is coded using JavaScript.
- This validation is completely customizable, but you need to create it all (or use a library i.e. jQuery).

Example:

1. Create html form with two text field for entering full name. Validate the form for empty fields before submitting using JavaScript.

```
<!doctype html>
<html>
<head>
<script>
        function validateForm()
        {
                var x = document.forms["myForm"]["Name"].value;
                if (x == "")
                {
                        alert("Name must be filled out");
                        return false;
                }
        }
</script>
</head>
<body>
<form name="myForm">
  <label for="Name">Full Name</label>
  <input id="Name" name="Name">
  <button onclick="validateForm()">Submit</button>
</form>
</body>
</html>
```

Or

```
<!doctype html>
<html>
<head>
<script>
        function validateForm()
        {
                var x = document.forms["myForm"]["Name"].value;
                if (x == "")
                {
                        alert("Name must be filled out");
                        return false;
                }
        }
</script>
</head>
<body>
<form name="myForm" onsubmit="return validateForm()">
  <label for="Name">Full Name</label>
  <input id="Name" name="Name">
 <input type="submit" value="Submit">
</form>
</body>
</html>
```

2. Create html form with one text field for entering roll number. Validate the number to be in the range 1 to 36 before submitting using JavaScript.

```
<!DOCTYPE html>
<html>
<head>
<script>
    function myFunction()
    {
      var x, text;
      // Get the value of the input field with id="numb"
      x = document.getElementById("numb").value;
      // If x is Not a Number or less than one or greater than 10
      if (isNaN(x) || x > 36 || x < 1) {
            alert("Please Provid Valid Roll number");
      }
    }
</script>
</head>
<body>
<label>Enter Roll Number</label>
<input type="number" id="numb"/>
<button type="button" onclick="myFunction()">Submit</button>
</body>
</html>
```

3. Form Validation using JQuery
//will be discussed in lab

Single Page Application

- A single-page application (SPA) is a web application or website that interacts with the user by dynamically rewriting the current web page with new data from the web server, instead of the default method of the browser loading entire new pages.
- The goal is faster transitions that make the website feel more like a native app.
- In a SPA, all necessary HTML, JavaScript, and CSS code is either retrieved by the browser with a single page load, or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions. The page does not reload at any point in the process, nor does it transfer control to another page, although the location hash or the HTML5 History API can be used to provide the perception and navigability of separate logical pages in the application.
- There are various techniques available that enable the browser to retain a single page even when the application requires server communication. Web browser JavaScript frameworks and libraries, such as **AngularJS, React.js,** Knockout.js, Vue.js, and Svelte have adopted SPA principles.

Single Page Application

- A single-page application (SPA) is a web application or website that interacts with the user by dynamically rewriting the current web page with new data from the web server, instead of the default method of the browser loading entire new pages.
- The goal is faster transitions that make the website feel more like a native app.
- In a SPA, all necessary HTML, JavaScript, and CSS code is either retrieved by the browser with a single page load, or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions. The page does not reload at any point in the process, nor does it transfer control to another page, although the location hash or the HTML5 History API can be used to provide the perception and navigability of separate logical pages in the application.
- There are various techniques available that enable the browser to retain a single page even when the application requires server communication. Web browser JavaScript frameworks and libraries, such as **AngularJS, React.js,** Knockout.js, Vue.js, and Svelte have adopted SPA principles.

AngularJS

- Angular remains one of the world's most popular JavaScript frameworks developed and maintained by Google for building a single page applications.

- Angular applications are built from **components.** Components combine HTML templates with special objects and control a portion of the page.
- A simple component from Angular's docs is shown here:

```
<!DOCTYPE html>
<html>
<script src="angular.js"></script>
<body>
<div ng-app="">
  <p>Name : <input type="text" ng-model="name"></p>
  <h1>Hello {{name}}</h1>
</div>
</body>
</html>
```

- AngularJS lets you extend HTML with new attributes called Directives.
- AngularJS directives are extended HTML attributes with the prefix ng-. for example,
  - The ng-app directive initializes an AngularJS application.
  - The ng-model directive binds the value of HTML controls (input, select, textarea) to application data.

- AngularJS is a fully client-side framework.
- AngularJS's templating is based on bidirectional UI data binding. Data-binding is an automatic way of updating the view whenever the model changes, as well as updating the model whenever the view changes.
- In traditional server-side HTML programming, concepts such as controller and model interact within a server process to produce new HTML views.
- In the AngularJS framework, the controller and model states are maintained within the client browser. Therefore, new pages are capable of being generated without any interaction with a server.

Visit: AngularJS Tutorial (w3schools.com) for detail

ReactJS

- React is a JavaScript library for building reusable user interfaces components. It is maintained by Facebook, Instagram and a community of individual developers and corporations.
- Lots of people use React as the V in MVC.
- React abstracts away the DOM from you, offering a simpler programming model and better performance.
- Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM. React finds out what changes have been made, and changes **only** what needs to be changed.
- One of the limitation of this is it covers only the view layer of the app, hence you still need to choose other technologies to get a complete tooling set for development.

Simple Example;

```
<!DOCTYPE html>
<html>
  <script src="production.min.js"></script>
  <script src="react-dom.production.min.js"></script>
  <script src="babel.min.js"></script>
  <body>
    <div id="mydiv"></div>

    <script type="text/babel">
      class Hello extends React.Component {
        render() {
          return <h1>Hello World!</h1>
        }
      }
      ReactDOM.render(<Hello />, document.getElementById('mydiv'))
    </script>
  </body>
</html>
```

- Start by including three scripts, the first two let us write React code in our JavaScripts, and the third, Babel, allows us to write JSX(JavaScript XML) syntax and ES6 in older browsers.
- React renders HTML to the web page by using a function called ReactDOM.render(). The ReactDOM.render() function takes two arguments, HTML code and an HTML element. The purpose of the function is to display the specified HTML code inside the specified HTML element.

Visit: React Tutorial (w3schools.com) For Detail