* Lightning Web Components (LWC)
  + Lightning component framework (revisited)
    - Term for the salesforce UI framework that is tailored to single page applications
    - Made up of two programming archetypes
      * Lightning aura components
      * Lightning web components
    - Diagram

      Description automatically generated
  + They were introduced in the spring 2019 release of salesforce
    - Because of this a lot of companies have not transitioned yet
  + Can be used within aura components but aura cannot be used in lwc
  + They comply with the world wide web (www) Consortium web component standards
    - Shadow DOM
      * DOM within a DOM
      * Was created to allow encapsulation and componentization on a web platform
      * Is an isolated DOM tree with its own elements and styles
        + Completely isolated from the original DOM
      * We use a shadow host to attach the shadow DOM to
        + The root can then be rendered from there
      * Not based on a full standalone document
      * Shadows the DOM it is attached to
    - Custom elements
    - HTML templates
    - CSS changes
    - JSON, CSS and HTML modules
  + Because it is built to be run natively in a browser, it is lightweight and has excellent efficiency
* The Component (in LWC)
  + A lightning web component that renders some form of user interface must include
    - HTML file
    - JS file
    - Metadata config file
  + You can optionally have more files
  + These files mist use the same name so the framework can autowire them to the view
  + A service component/library must include
    - JS file
    - Metadata file
  + Component folder
    - Holds all files associated with the component
    - The file naming conventions
      * Must begin with a lowercase letter
      * Must contain only alphanumeric or underscore characters
      * Must be unique in the namespace
      * Can’t include any whitespace
      * Can’t end with an underscore
      * Can’t contain two consecutive underscores
      * Can’t contain a hyphen
      * Must use camelCase to name your component that might require a hyphen
  + Component HTML file
    - The markup for your component
    - Each UI component HTML file must begin with the root tag <template>
    - The HTML file follows the naming convention <component>.html using camelCase
    - When the component renders, the template tag is replaced with the name of the component in kebab case
      * Component html file: myComponent.html kebab case: my-Component
  + Component JavaScript file
    - Every component must have a JS file
    - The JS file defines the HTML element
    - In LWC, JS files are ES6 modules
      * Echma script 6
      * Everything is scrited to that module
      * To import a class, function or a variable declared in a module, you import with import keyword
      * To allow other code to use a function, class or variable, we must first export it
    - Third party libraries
      * We cab use third party libraries without JS
      * To do so, download the library from the third party libraries site
      * Upload the library to salesforce as a static resource
      * Import it from resources
      * LIGHTNING LOCKER IS STILL USED WITH THIS
* Lightning Element
  + The core module in LWC is the lightning element from the LWC module
  + It is a wrapper for the standard HTML element
  + We will create a JS class that extends lightning element to create a JavaScript class for a LWC
* HTML templates
  + The power of LWC is the templating system, which uses the virtual DOM to render component smartly and efficiently
  + These are written in the template tag of the component file
  + It can be used and augmented with templates using standard HTML, some directives and data binding that are unique to LWC
  + Data binding
    - Binding properties in the component template to properties in the components JS class
    - In the template you surround the property with {}
    - Variables can be bound to values in the component with {}
    - No operations can be done in the {} so all logic must be done in the controller