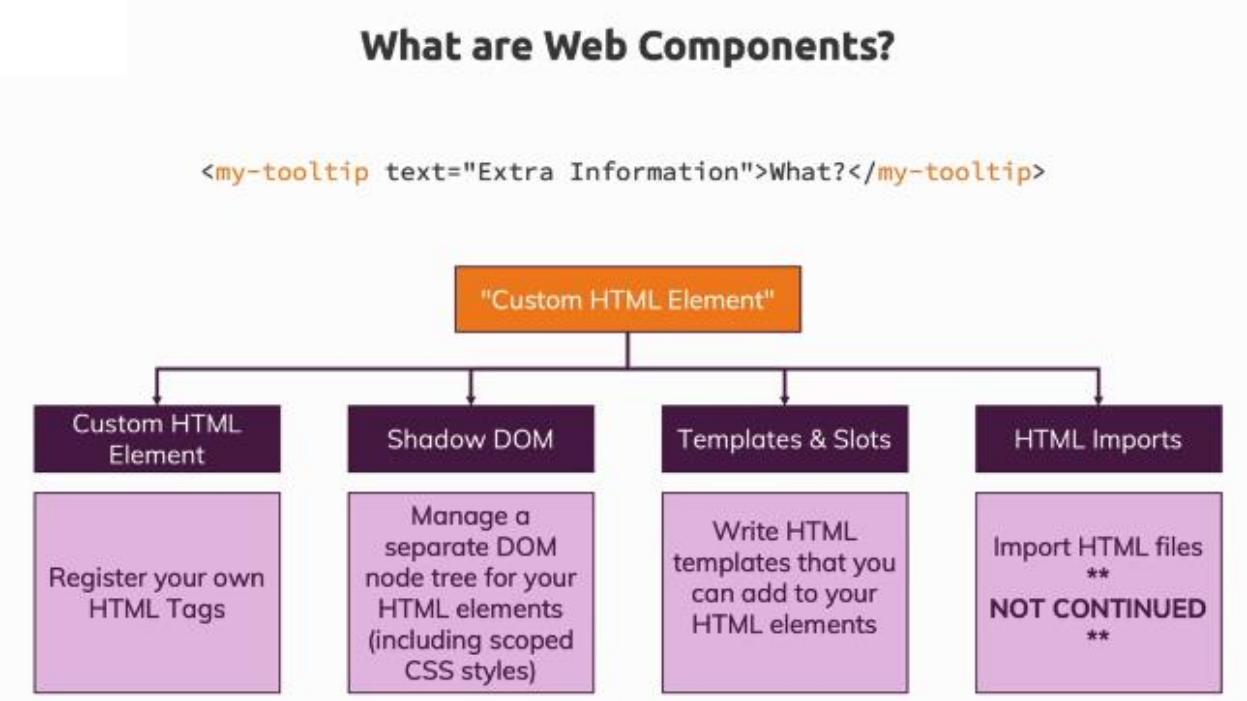
Web Components & Stencil.js

Build Custom HTML Elements

Contents

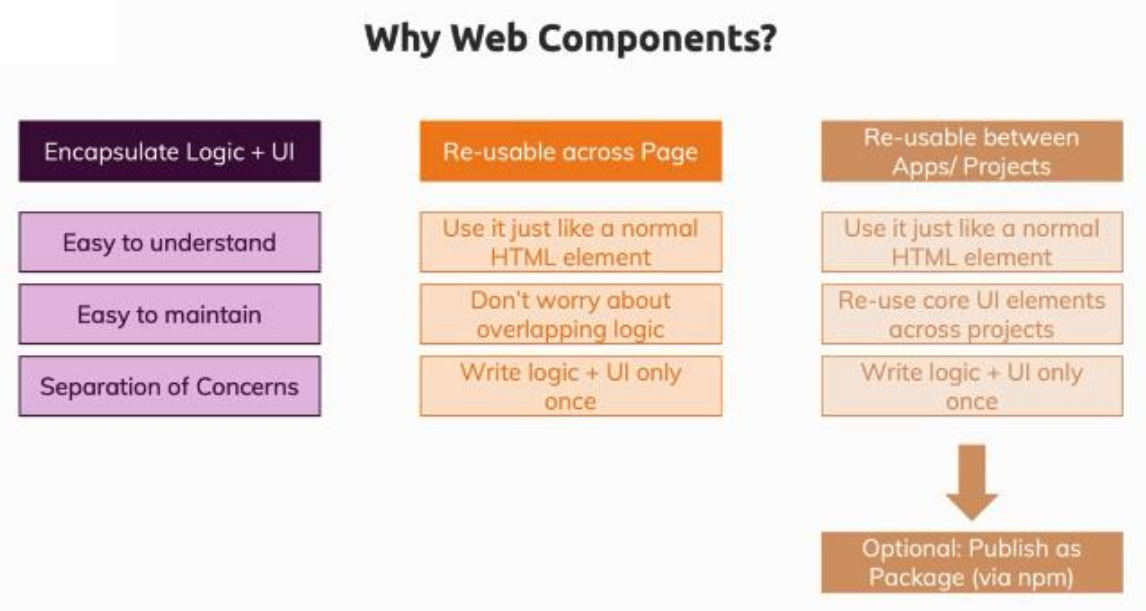
Introduction

What are Web Components

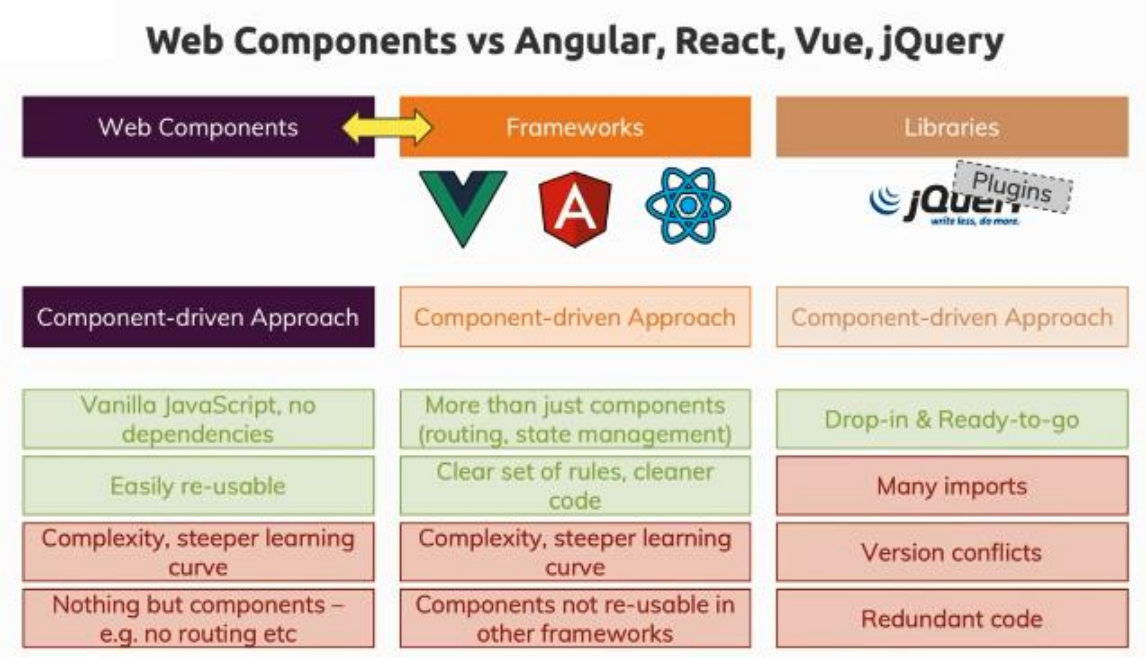


- Web components really are just your own custom HTML elements.
- Web components is a set of specifications
- The **shadow DOM** is all about having a separate DOM, a tree of DOM nodes behind your custom element that is separated from the normal DOM. This helps you for example with scoping your styles to your custom element, so that styling you set up for your custom element doesn't interfere with other parts of your application.
- Your custom element will in the end just be a combination of default HTML elements because in the end, the browser only knows how to handle these but you can compose them and add styling and add Javascript logic to create more powerful widgets.
- **Templates and slots** are another important specification because here you define this core structure behind your element, so you basically define the **template** which is then rendered as a shadow DOM and **slots** are helpful for exposing certain entry points, e.g. title of your modal. Slot allows us to pass dynamic content into our web component so that developers using our web components can use them and the set of features that are baked in. And still if our components wants that, pass extra content into the component.
- The HTML imports specification – That was meant to be a feature which allows you to import HTML files into HTML files, so that you can define these templates for your components in HTML files that are then imported dynamically. But this is discontinued now because the industry moved in a direction where we use Javascript for all of that and where we use modern build tools and Javascript modules to import and export inside of our Javascript files and import the HTML templates as strings there as well.

Why Do We Use Web Components?



Comparing Web Components & Frameworks (Angular, React, Vue, jQuery)

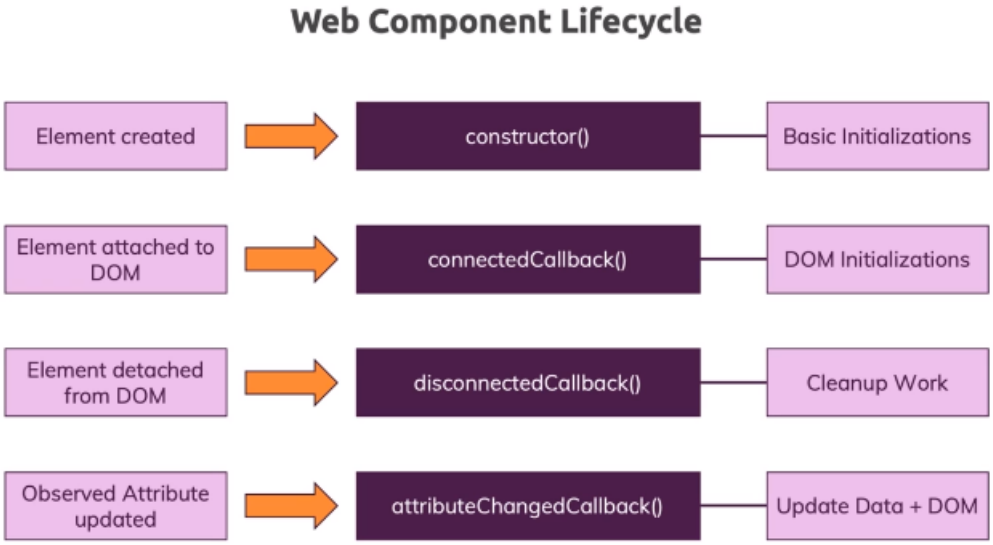


Understanding the Basics

Types of Custom Elements

- **Autonomous elements**
  These are basically your own custom elements that don't depend on anything, that are totally independent of existing HTML elements that are built into the browser.
- **Extending built-in Elements**
  Alternatively, you can also create your own custom elements by extending built-in elements though, like extending the built-in button or the built-in paragraph.

Understanding the Custom Elements Lifecycle



- There is a specific web component lifecycle which the browser follows when instantiating our own custom element objects in the DOM so to say.
- The first thing that gets executed is the **constructor** because this always gets executed when an object gets created based on your class.
- **IMP** – The moment where the element is created is not the moment where the element is then also attached into the real DOM by the browser, instead it's created in memory first and it's not part of the DOM at the beginning. So the constructor is great for some basic initializations, some basic values for the different properties and variables you might be using in your class (component) but it's the wrong place for accessing the DOM because your custom element has not been added to the DOM yet.
- **connectedCallback** method is called when your element has been attached to the DOM and therefore this is the place for DOM initializations. So this is where you can access the DOM.

- There also is a **disconnectedCallback** method which will also be executed automatically for you by the browser whenever your custom element (web component) is detached from the DOM. So this is great place for cleanup work e.g. if you were sending an HTTP request, this would be where you could cancel it.
- There is another method the **attributeChangedCallback** and that will be important for listening to changes to attributes of your custom element. This is important for updating the data and the DOM of your web component when some attributes which can be passed to your component changed.

Shadow DOM

- The shadow DOM and templates – Both these techniques will help us with ensuring that we write a nice, reusable and encapsulated custom web component.
- Basically our custom component should have its own DOM tree which is not mixed or which is not kind of affected by the normal DOM tree and that would be the definition of the **Shadow DOM**. So our custom element should have its own DOM behind it which kind of makes up what we see on the page but which is not directly connected to the real DOM, which is not affected by global styles and the other way around.

Slots

- Whatever passed between opening and closing tag of our custom element is passed to <slot> inside the template of the custom element.
- Slot can have default value.

Diving Deeper Into Web Components

Understanding Shadow DOM Projection

- With slots, you can project content into your shadow DOM, so you got an entry point into your shadow DOM from the main DOM but you don't technically move the element into the shadow DOM, the browser just displays it as if it were part of it but actually, it's rendered differently in the DOM.

Styling "slot" Content Inside the Shadow DOM

- Use ::slotted() CSS selector to style it.
- Styles in main DOM have preference over the styles set in shadow DOM.

Styling Host Component inside Custom Element

- Using :host selector.
- :host can also be used for conditional styling. E.g. :host(.important), it means our custom element having class “important”.

Styling with the host-context in Mind

- Use :host-context inside custom element template to style based on surrounding conditions. E.g. style if out element is inside a paragraph.

Notes:

- More about Templates & Slots: https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_templates_and_slots
- Google Article on Custom Elements: https://developers.google.com/web/fundamentals/web-components/customelements
- Google Article on Shadow DOM: https://developers.google.com/web/fundamentals/web-components/shadowdom

Building More Complex Components

Understanding Named Slots

- Out custom element template can have more than one slots so it is important to name the slots so that proper slots can be targeted from the HTML page where our component is being used.
- E.g. our custom element template will have
  <slot name="**title**">Confirm</slot>
  And it will be targeted from HTML page as
  <uc-modal>
  <h1 slot="**title**">Please Confirm</h1>
  </uc-modal>

slotchange event

- slotchange event will be triggered whenever new content arrives. This will trigger at least once when our shadow DOM is rendered for the first time.
- With this, we get access to the real DOM element content used for a given slot.
- So slotchange combined with assignedNode() can be very helpful if you for example also have an application where the content in your slot might change at runtime too.

Custom Event of our web component

- Using Event constructor to create event object with our event name. and use built-in dispatchEvent() method available on the HTML element to fire our custom event.

Stencil - An Introduction

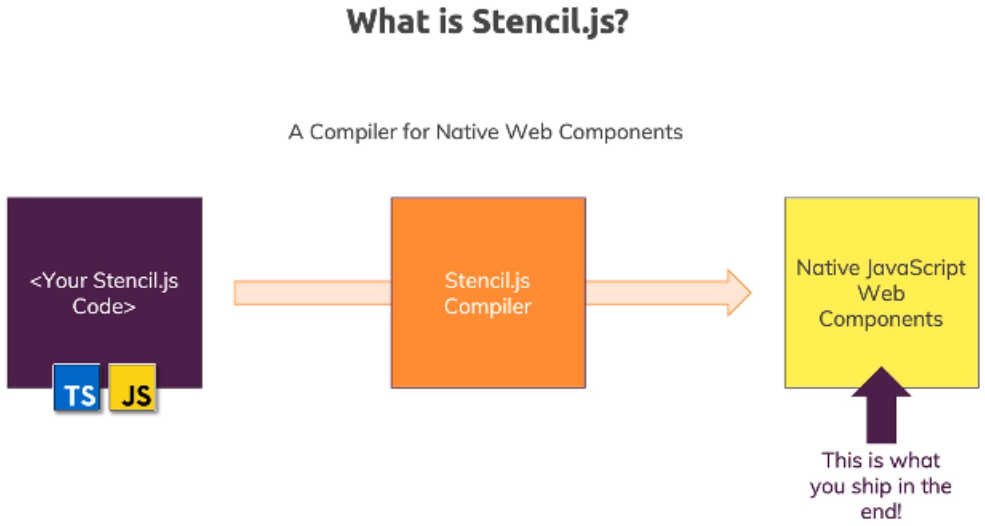
Using Web Components in Modern Browsers

- Web component browser support is one of the reasons why stencil.js, a tool that helps us with building web components, is very appealing.

Browser Support for Web Components

- Browser support – https://www.webcomponents.org
- Polyfills – Refer https://www.webcomponents.org/polyfills
- Stencil.js tool will make his whole deployment and distribution process way easier and will ensure that our web components work in all major browsers out of the box without us needing to do any special extra setup.

What is Stencil?



- It is a tool (neither framework nor library) that runs on your machine which you will never deploy to a website that is then running in the web.
- It's a compiler for native web components and in the end, it will spit out is a native Javascript web component.
- stencil.js gives us a way nicer syntax and a lot of convenience features which we can use to write web components in a more convenient and error safe way using Javascript and also Typescript, and in the end, stencil will then compile that code down to native Javascript web components.
- So we don't need to ship any extra library to make these components work in the browser. It will be the same type of component we build manually, just now being built in a bit of a more convenient way and with the additional feature of adding all polyfills that might be required in a browser automatically so that deploying our web components becomes even easier.

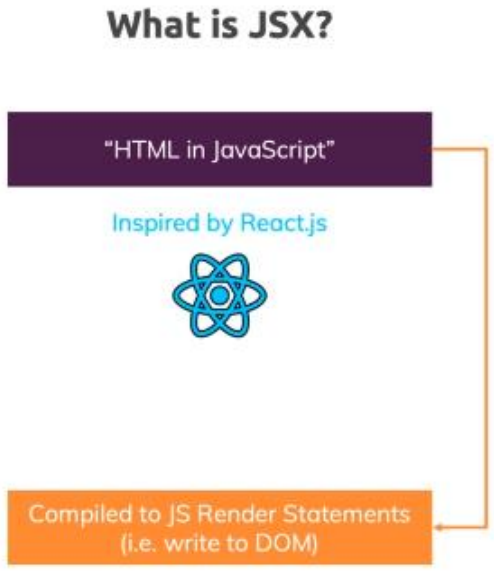
Stencil.js behind the Scenes

- Stencil.js spits out native, vanilla-JS web components.
  BUT: These components have (vanilla) JavaScript added to them that enhances the web component experience by:
  - Loading component code lazily (i.e. source code gets only pulled into the page if it's really needed => This reduces the overall bundle size)
  - Loading required polyfills automatically for browsers that need it
  - Re-rendering the web (component) DOM efficiently (i.e. the DOM gets updated with as minimal impact as possible, to reduce the amount of work JS and the browser have to do)

Creating Stencil Project

- Refer – https://stenciljs.com/
- >npm init stencil
- This will install stencil and will promt for creation of project.
- Select 'component' and then provide name, etc.
- Then install the dependencies by >npm install
- To build the stencil project >npm run build
- This will generate a web component based on this TSX file and if you have multiple components, it would generate component files for all of them.

What is JSX



Using Stencil

Notes

- Official Stencil.js Docs: https://stenciljs.com/docs/introduction
- Refer Code –
    - 03-web-components-stencil
    - 04-advanced-web-components-stencil

Deployment & Publishing

Publishing to NPM

- Signup for NPM https://www.npmjs.com/
- Creating and publishing unscoped public packages https://docs.npmjs.com/creating-and-publishing-unscoped-public-packages
- Once published, we can install it like any other public packages.

Tips and Tricks

- The term Web component and Custom Element are used interchangeably.
- https://stenciljs.com/