**LIGHTNING WEB COMPONENT IN EXPERIENCE BUILDER**

**What is LWC?**

* LWC's are nothing but a component which can be used to build a site page.
* multiple components can be added in one page.
* eg -> Take any online site home page.... what ever we see over there

like header, menu, carousel banner... all are separate components.

* advantage here is that the component can be reused.
* LWC will consist of 4 files (HTML, CSS, JS, XML)

**Lifecycle Hooks**

1. constructor

2. Connected callback

3. rendered callback

4. render

5. disconnected callback

6. errorCallback(error, stack)

**constructor**() -

* fires when a component instance is created
* flows from parent to child. child elements cannot be accessed here as child is not yet instantiated.
* used to set up the initial state and default values for the component’s properties.

constructor() {

super();

this.propertyName = ‘value’;

}

**Connected callback -**

* The connectedCallback lifecycle hook is invoked when a component is connected to the document.
* This callback is invoked after all the public properties are set and can be used to set the state of the component.
* flow from parent to child. You can’t access child elements from the callbacks because they don’t exist yet.
* Perform initialization tasks, such as fetch data, set up caches, or listen for events
* Subscribe and Unsubscribe from a Message Channel (communicating between two components, where it will wait for one event to complete and start the other event).

**disconnectedCallback() -**

* Called when the element is removed from a document. This hook flows from parent to child.
* Use disconnectedCallback() to clean up work done in the connectedCallback()
* use this hook to unsubscribe from a message channel.

**renderedCallback**() -

* Called after every render of the component
* flows from child to parent
* When a component renders, the expressions used in the template are reevaluated.
* use a boolean field like hasRendered to track whether renderedCallback() has been executed
* eg - Elements created using the for:each directive. The decision to reuse these iteration elements depends on the key attribute. If key changes, the element may be rerendered.
* If key doesn’t change, the element isn’t rerendered, because the engine assumes that the iteration element didn’t change.

**render() -**

* The main use case is to conditionally render a template.
* You may want to render a component with more than one look and feel, but not want to mix the HTML in one file.
* https://developer.salesforce.com/docs/platform/lwc/guide/create-render.html

**errorCallback** -

* it captures errors that occur
* captures errors in all the descendent components in its tree.

<template>

<template lwc:if={error}>

<error-view error={error} info={stack}></error-view>

</template>

<template lwc:else>

<healthy-view></healthy-view>

</template>

</template>

import { LightningElement } from "lwc";

export default class Boundary extends LightningElement {

error;

stack;

errorCallback(error, stack) {

this.error = error;

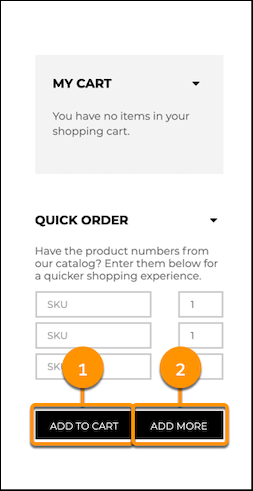
}

}

**Lightning data service**

* + If a page is composed of components showing the same record, all components show the same version of the record.
  + Components accessing the same record see significant performance improvements, because a record is loaded once, no matter how many components are using it
  + If Lightning Data Service detects a change to a record or any data or metadata it supports, all components using a relevant @wire adapter receive the new value
  + Loads record data progressively.
  + Caches results on the client.
  + Invalidates cache entries when dependent Salesforce data and metadata changes.
  + Optimizes server calls by bulkifying and deduping requests
  + To improve performance, Lightning Data Service maintains a client-side cache of record data that has been loaded via a wire adapter
    - LWC -> Lightning data service <-> server

EXAMPLE –



**Component communication using Pub sub/LMS/Events –**

1. **Create and Dispatch events –**

* To create an event, use the CustomEvent()
* To dispatch an event, call the EventTarget.dispatchEvent()
* If your event is called onmessage, the markup would be <c-my-component ononmessage={handleMessage}>
* To pass data up to a receiving component, set a detail property in the CustomEvent constructor.

1. **Handle Events –**

* Two ways, one is through html and other one is through programmatically (even listeners)

this.template.addEventListener("notification", (evt) => {

console.log("Notification event", evt);});

<template>

<input type="text" value={myValue} onchange={handleChange} />

</template>

1. **Pub sub (Lightning message service)–**

* It means publish and subscribe
* Used for communication between two components on same page, even when there is no parent child relationship
* In a publish-subscribe pattern, one component publishes an event. Other components subscribe to receive and handle the event.

**Design Attributes –**

* **targetConfigs** :- Configure the component for different page types and define component properties. For example, a component could have different properties on a record home page than on the Salesforce Home page or on an app page.
* **targetConfig** :- Use a separate targetConfig for each different page type configuration. Specify one or more page types in the targets attribute, such as <targetConfig targets="lightning\_\_RecordPage"> or <targetConfig targets="lightning\_\_RecordPage,lightning\_\_AppPage">.
* **property** :- Specifies a public property of a component that can be set in Lightning App Builder, App Manager, Lightning Flow Builder, or Community Builder. The component author defines the property in the component’s JavaScript class using the @api decorator.
* **datasource** :- Renders a field as a picklist, with static values. Supported only if the type attribute is String.
* **description** :- Displays as an i-bubble for the attribute in the tool.
* **default** :- The default value for the attribute.
* **label** :- Displays as a label for the attribute in the tool.
* **type** :- The attribute’s data type.
* **name** :- Required if you’re setting properties for your component. The attribute name. This value must match the property name in the component’s JavaScript class.

<?xml version="1.0" encoding="UTF-8"?>

<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata" fqn="migrateDesignAttributeToLWC">

<apiVersion>46.0</apiVersion>

<isExposed>true</isExposed>

<masterLabel>Migrate Design Attribute to LWC</masterLabel>

<description>This component shows ability to expose public property to Lightning App Builder, App Manager.</description>

<targets>

<target>lightning\_\_RecordPage</target>

<target>lightning\_\_AppPage</target>

<target>lightning\_\_HomePage</target>

</targets>

<targetConfigs>

<targetConfig targets="lightning\_\_RecordPage,lightning\_\_AppPage,lightning\_\_HomePage">

<property name="currentStep" type="String" datasource="step-1,step-2,step-3,step-4,step-5" />

<property name="type" type="String" datasource="base,path" />

<property name="variant" type="String" datasource="base,shaded" />

</targetConfig>

</targetConfigs>

</LightningComponentBundle>

**Wire Apex Methods –**

* Use @wire in a component’s JavaScript class to specify an Apex method
* To use @wire to call an Apex method, annotate the Apex method with @AuraEnabled(cacheable=true)
* A client-side Lightning Data Service cache is checked before issuing the network call to invoke the Apex method on the server

import apexMethodName from '@salesforce/apex/namespace.classname.apexMethodReference';

@wire(apexMethodName, { apexMethodParams })

propertyOrFunction;

**Lightning locker service overview –**

* It is nothing but lightning web security
* JavaScript, strict mode is enforced in modules
* JavaScript throws errors that would otherwise be suppressed. Examples of unsafe actions include assigning values to non-writable properties and using a variable that hasn’t been declared.
* Lightning web components can’t use the window or document global properties to query for DOM elements. For example, use this.template.querySelector() instead of document.querySelector().
* Avoid passing large arrays of objects more than one level down. For example, pass an array from parent to child but no further.
* Divide your data into multiple smaller arrays in the parent component.