Your ultimate guide to mastering Lightning Web Components (LWC)

🟦 Salesforce interviews!

✅ Understand core concepts
✅ Tackle real-world scenarios
✅ Enhance your LWC expertise

Question 1 - What is the structure of LWC?

Answer: LWC is built using these main parts, including HTML, JS, CSS and XML.

Question 2 - Can you explain the purpose of each part of the LWC folder structure?

Answer:

- **HTML** – It defines the markup of the LWC.
- **JS** – It establishes the component logic and behaviours of LWC.
- **CSS** – It defines the styling of LWC.
- **XML** – It includes the component's metadata. Here we defined the attributes such as targets, like where our component will be displayed in the application.

Question 3 -What is the recommended naming convention that should be followed while working with LWC?

Answer: The following are the rules that we should follow while naming our LWC:

1. The component must start with the lowercase i.e. it should follow camel case.
2. It must contain only alphanumeric or underscore characters, not a dash.
3. While referencing components in HTML we should use the kebab case.
4. Must contain only alphanumeric or underscore characters

Below are the samples of the camel case and kebab case

Question 4 - What is SLDS?

Answer: SLDS, or Salesforce Lightning Design System, is a key framework for designing and building consistent user interfaces on the Salesforce platform.

Question 5 - What are the advantages of LWC over Aura components?

Answer:

- Uses native web standards.
- Improved performance and faster rendering.
- Simplified development and debugging.

Question 6 - How do you communicate between two LWC components?

Answer:

- Parent to child: Using @api properties.
- Child to parent: Using custom events.

Question 7 - What are events in LWC?

Answer: Events in Lightning Web Components (LWC) enable communication between components. This communication is crucial for creating dynamic and interactive user experiences.

LWC supports two types of events: standard and custom.

- **Standard events** are built-in browser events like click or change, which handle common interactions.
- **Custom events** are ones you define and dispatch yourself, making them ideal for more complex communication, such as passing data between components or triggering specific actions.

Question 8 - What is the event bubbling mechanism in LWC?

Answer: Event bubbling allows events to propagate from child components up through the DOM tree.

Question 9 - What are the types of decorators in LWC?

Answer: In LWC we have three types of decorators – api, track and wire

Question 10 - Explain each decorator and how we use them.

Answer: First of all to use any decorator we should use @ as a prefix then decorator and then variable or property name.

1. **@api** – It is used to make any property or variable public so that it will be visible to other components for communication.
2. **@track** – It is used to declare any property or method as private. In LWC all fields are reactive but if the field type is array or object, there are some limits to the depth of changes to be tracked. If we declare them with @track, we can observe the changes.
3. **@wire** – It is used to interact with salesforce data.

Are you preparing for the Salesforce Certifications? Check out the Salesforce certification practice set here

Question 11 - Suppose I have a scenario in which I have 2 LWC components and CompA and CompB I. CompA is the parent and CompB is the child. How can we send data from CompA to CompB?

Answer: Here we have to establish the Parent-Child Communication. It will be done through @api decorator. Let's look at the below example.

We have created two components here **childComp** and **parentComp**. We have **messageFromParent** in childComp which is receiving the value from Parent.

childComp.js

```
import { LightningElement,api } from 'lwc';
export default class ChildComp extends LightningElement {
@api messageFromParent
}
```

childComp.html

```
<template>
<lightning-card variant="Narrow" title="Child Component">
<div>{messageFromParent}</div>
</lightning-card>
</template>
```

childComp.js-meta.xml

```
ıl version="1.0"?>
ghtningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
.Version>57.0</apiVersion>
:xposed>true</isExposed>
·gets>
·get>lightning__AppPage</target>
ırgets>
.ghtningComponentBundle>
```

parentComp.js

```
import { LightningElement } from 'lwc';
export default class ParentComp extends LightningElement {
message = ''
handleChange(event) {
this.message = event.detail.value
}
}
```

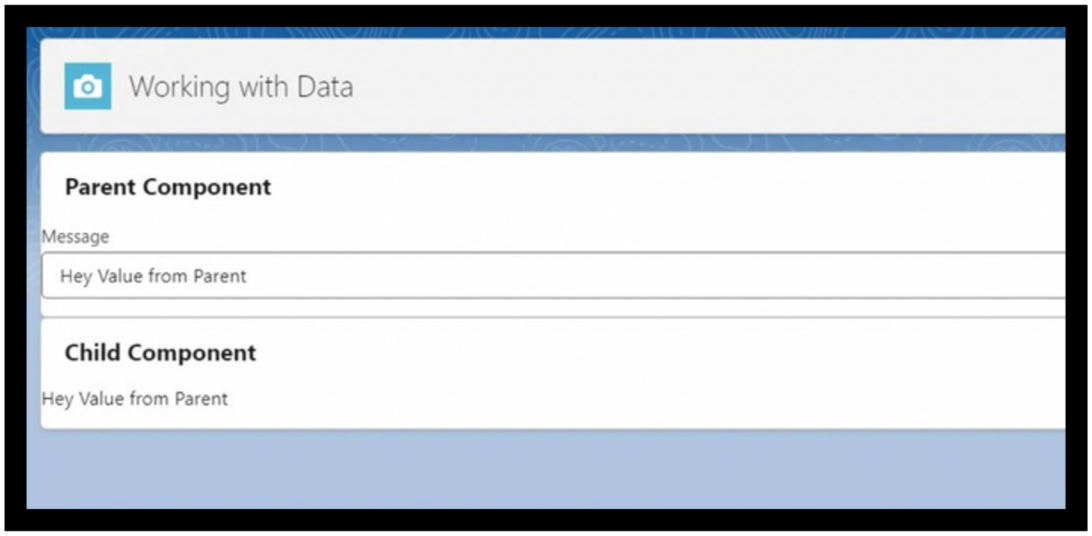
parentComp.html

```
<template>
<lightning-card variant="Narrow" title="Parent Component">
<lightning-input
type="text"
variant="standard"
name="Message"
label="Message"
placeholder="type here..."
value = {message}
onchange={handleChange}></lightning-input>
</lightning-card>
<c-child-comp message-from-parent = {message}></c-child-comp>
</template>
```

parentComp.js-meta.xml

```
?xml version="1.0"?>
_ightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata
apiVersion>57.0</apiVersion>
isExposed>true</isExposed>
targets>
target>lightning__AppPage</target>
/targets>
/LightningComponentBundle>
```

Output



Question 12 - Referring to the above scenario if I want to communicate Child to Parent where CompA is the parent and CompB is the child. How can we send data from CompB to CompA?

Answer: Here we have to show Child to Parent Communication. It will be done through the dispatching of the custom event from the child and then the parent component will receive it. Here we have two components **childComp** and **parentComp**. In the Child component, we have an input box to enter our value and then on clicking of Send Message button, we will dispatch our event.

In Parent component markup, where we defined our child, to catch the event we have to use "**on**" as a prefix just like we do in the standard events like onclick, and onchange. After that, to handle the event we have a handler in the JS **handleMessageFromChild** which is receiving the event and in the detail, we have our value. Below is a sample of Child to Child-to–Parent communication.

childComp.js

```
import { LightningElement, api } from 'lwc';
export default class ChildComp extends LightningElement {
messageFromChild = ''
handleChange(event) {
this.messageFromChild = event.detail.value
}
handleClick() {
const eve = this.dispatchEvent(new CustomEvent('send', { detail:
this.messageFromChild }))
}
}
```

childComp.html

```
<template>
<lightning-card variant="Narrow" title="Child Component">
<lightning-input type="text" variant="standard" name="Message"
label="Message" placeholder="type here..."
value={messageFromChild} onchange={handleChange}></lightning-
input>
<lightning-button label="Send Message" title="Send Message"
onclick={handleClick}></lightning-button>
</lightning-card>
</template>
```

childComp.js-meta.xml

```
<?xml version="1.0"?>
<LightningComponentBundle
xmlns="http://soap.sforce.com/2006/04/metadata">
<apiVersion>57.0</apiVersion>
<isExposed>true</isExposed>
<targets>
<target>lightning__AppPage</target>
</targets>
</LightningComponentBundle>
```

parentComp.js

```
import { LightningElement } from 'lwc';
export default class ParentComp extends LightningElement {
messageRecieved = 'No Message'
handleMessageFromChild(event) {
this.messageRecieved = event.detail
}
}
```

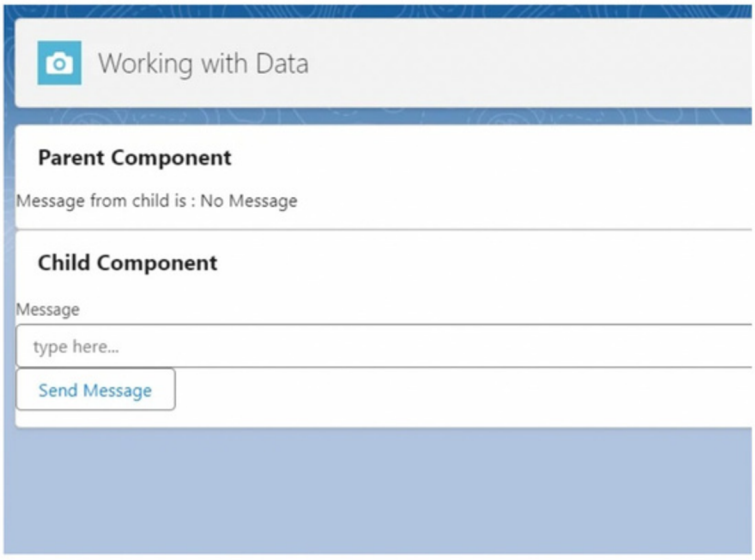
parentComp.html

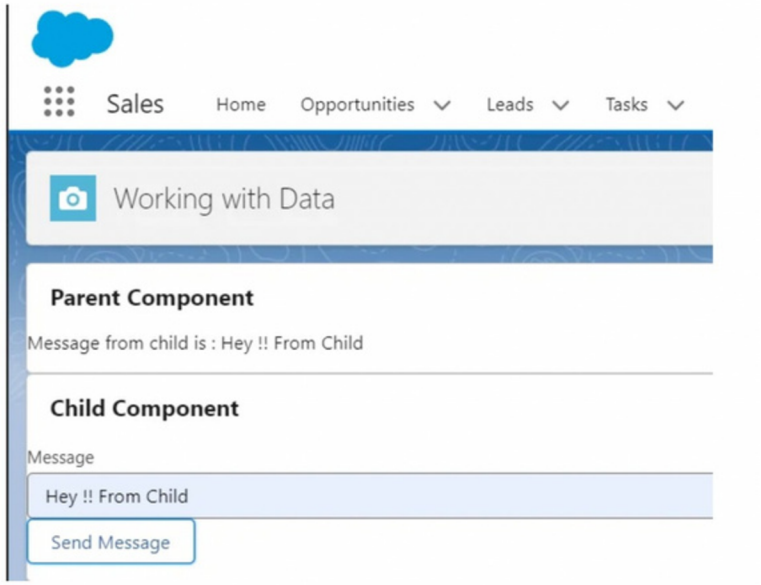
```
<template>
<lightning-card variant="Narrow" title="Parent Component">
<div>Message from child is : {messageRecieved} </div>
</lightning-card>
<c-child-comp message-from-child={messageRecieved} onsend =
{handleMessageFromChild}></c-child-comp>
</template>
```

parentComp.js-meta.xml

```
<?xml version="1.0"?>
<LightningComponentBundle
xmlns="http://soap.sforce.com/2006/04/metadata">
<apiVersion>57.0</apiVersion>
<isExposed>true</isExposed>
<targets>
<target>lightning__AppPage</target>
</targets>
</LightningComponentBundle>
```

Output





Question 13 - What is the promise function of LWC?

Answer: The promise function basically helps us to run a code asynchronously and it returns success and failure events upon completion. It has different states:
Pending: The initial state
Fulfilled: The operation was completed successfully ie resolved.
Rejected: The operation failed.

Question 14 - How can we call apex in LWC?

Answer: There are two ways to call the apex method in LWC –

1. Using wire services
2. Using Imperative method

Call Apex using @wire Decorator:

You can @wire a property or a function to receive the data from Apex. You have to annotate your apex method with **@AuraEnabled(cacheable=true)**. By using the wire services it reduces the server call because it loads the data from the cache. We can't do DML with a wire decorator.

Imperative Call

If we want to invoke our method call on some click or to control our method invocation we have to call the method imperatively. Calling our method in this way is also helpful if we want to perform some DML operations. Here, we don't annotate our method (cacheable=true).

Sample Code for wire and imperative

Here we have the getAccount method that we imported from our Apex controller and returned account data where ID matches to accId. While using wire, to make the variable reactive and receive ID from the

record page, we can annotate it with @api and display it with a $ sign.

However, in the imperative call, it is different, we use it with this.accId and in imperative it will return the promise. Where **.then** is the resolve and **.catch** is the reject.

```
// wire call
@wire(getAccount,{ accId: '$accId' })
wiredAccResult({ data, error }) {
if (data) {
console.log('data coming for accounts are ', data)
console.log('length of data coming for accounts are ', data.length)
}
if (error) {
console.log('Error in fetching account ',
JSON.stringify(error.message));
}
}
// Imperative call
fetchAccount(){
getAccount({ accId: this.accId})
.then(data=>{
//console.log('length of data coming for accounts are ',data.length)
})
.catch(error=>{
console.log('Error occurred while fetching account
',error.body.message)
})
}
```

Question 15 - What are the lifecycle hooks in LWC?

Answer: The lifecycle hooks in LWC provide the ability to control your code at various stages. We have different types of lifecycle hooks in LWC which run in a specific order:

1. constructor()
2. ConnectedCallback()
3. RenderedCallback()
4. DisconnectedCallback()
5. ErrorCallback()

Note: If we also have the child component, Child flow will run after the connectedCallback and before the renderedCallback in the same order.

Question 16 - How do you use conditional rendering in LWC?

Answer: Conditional rendering in Salesforce helps to show content based on specific conditions. We can achieve this by template **if:true** or with new **lwc:if, lwc:elseif, and lwc:else.**

Question 17 - What is the difference between for:each and the map in LWC?

Answer: For:each and Map both are used to handle Arrays. But the major difference between them is that we can update the element in for:each while iterating but it doesn't return the new array. However, Map returns the new array with the modified elements.

Question 18 - Can we call Aura in LWC and Vice versa?

Answer: It is not possible to call Aura inside LWC but vice versa is likely that we can call LWC inside Aura.

Question 19 - What is event bubbling and event capturing in LWC?

Answer: Suppose we have 3 components childComp, parentComp, and grandParentComp and we want to communicate from child to grandparent then while dispatching the event we can set the bubble and composed property as true so that the event can move up to a higher hierarchy And can easily cross the Shadow boundary.

As we see in the event bubbling event is moving from bottom to top. The same in event capturing event is moving from top to bottom.

Question 20 - Suppose I have two independent components CompA and CompB. How can I establish communication between them?

Answer: It shows communication between unrelated components so here we will establish communication via Lightning Message Service.

In this case, we will have to create a message channel, a publisher component from which we have to send the data and a subscriber component from which we have to receive the data.

Are you preparing for the Salesforce Certifications? Check out the Salesforce certification practice set here

Question 21 - Suppose I have a component in which I am displaying the data of Contact in a tabular format but when I deploy that component it is not visible to the user. What is the issue and how can we debug it?

Answer: Here we can have the following issues:

1. We can check if **<isExposed>true</isExposed>** is true or not because if it is false, the component itself is not visible.
2. We can also check if a component is visible to the correct target, if we want to deploy that component in the Lightning App Page then the target should be like this **<target>lightning__AppPage</target>**. Similarly, we have to check for correct targets.
3. If both the above points work fine, then we have to check the Apex class sharing. If the class is using sharing and that user does not have access to those contact records then it might be the issue.
4. Also, we can check whether SOQL is working fine or not. We can check if SOQL is returning to user mode or system mode and if in the user mode then the user should have the correct access.
5. We can also check if the user can access the apex class in the Profile or Permission Set.

Question 22 - Given a scenario, we have a simple requirement to display a form which creates a contact How can we achieve it without apex?

Answer: This scenario can be achieved by using a standard base component viz lightning-record-form. They are useful when we have a simple requirement like that. Below is the code to achieve this.

contactCreator.js

```
import { LightningElement } from 'lwc';
import CONTACT_LastName_FIELD from '@salesforce/schema/Contact.LastName';
import CONTACT_FirstName_FIELD from '@salesforce/schema/Contact.FirstName';
import CONTACT_Email_FIELD from '@salesforce/schema/Contact.Email';
import CONTACT_OBJECT from '@salesforce/schema/Contact';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';

export default class ContactCreator extends LightningElement {
objectApiName = CONTACT_OBJECT
fields = [CONTACT_Email_FIELD, CONTACT_FirstName_FIELD, CONTACT_LastName_FIELD];
handleSuccess(event) {
const toastEvent = new ShowToastEvent({
title: "Contact created",
message: "Record ID: " + event.detail.id,
variant: "success"
});
this.dispatchEvent(toastEvent);
}}
```

contactCreator.html

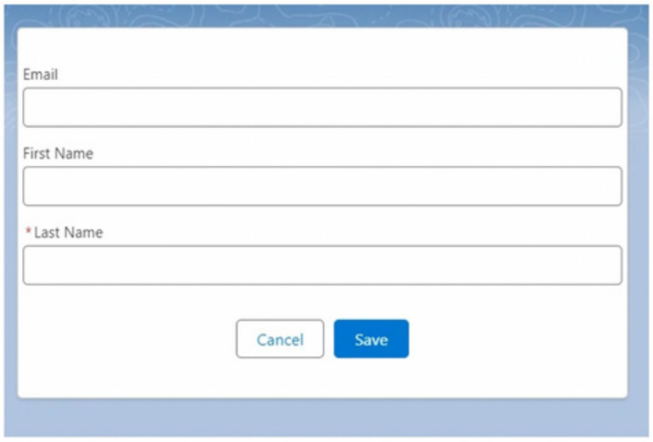
```
<template>
<lightning-card>
<lightning-record-form object-api-name={objectApiName} fields={fields}
onsuccess={handleSuccess}></lightning-record-form>
</lightning-card>
</template>
```

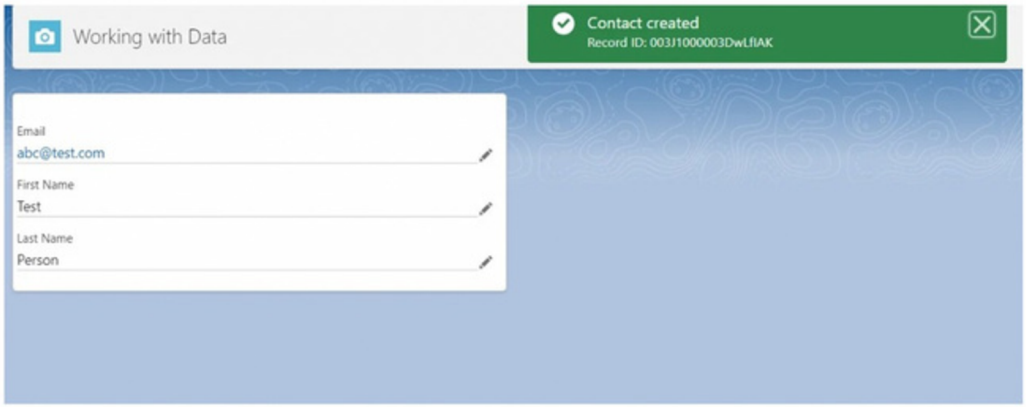
contactCreator..js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata" fqn="helloWorld
<apiVersion>52.0</apiVersion>
<isExposed>true</isExposed>
<targets>
<target>lightning__AppPage</target>
<target>lightning__HomePage</target>
</targets>
</LightningComponentBundle>
```

Here Cancel and Save buttons will come automatically with the record form and clicking on Save button it fire the onsuccess event. If we want to perform some additional things we can handle that.

OUTPUT





Question 23 - Suppose I have one LWC component in which I have a property name "message". I want this message value to be dynamic like I want to display different values on each page. How can I achieve it?

Answer: To achieve this scenario we have to utilize the meta xml file. There we have to define our target config and the property name, with its type, default value and label. Below is the sample code and output. We have created a helloWorld LWC component to demonstrate this. When we deploy this component we will have the option to provide the custom message.

helloWorld.html

```
<template>
<lightning-card title="HelloWorld" icon-name="custom:custom14">
<div class="slds-m-around_medium">
<p>Hello, {greeting}!</p>
</div>
</lightning-card>
</template>
```

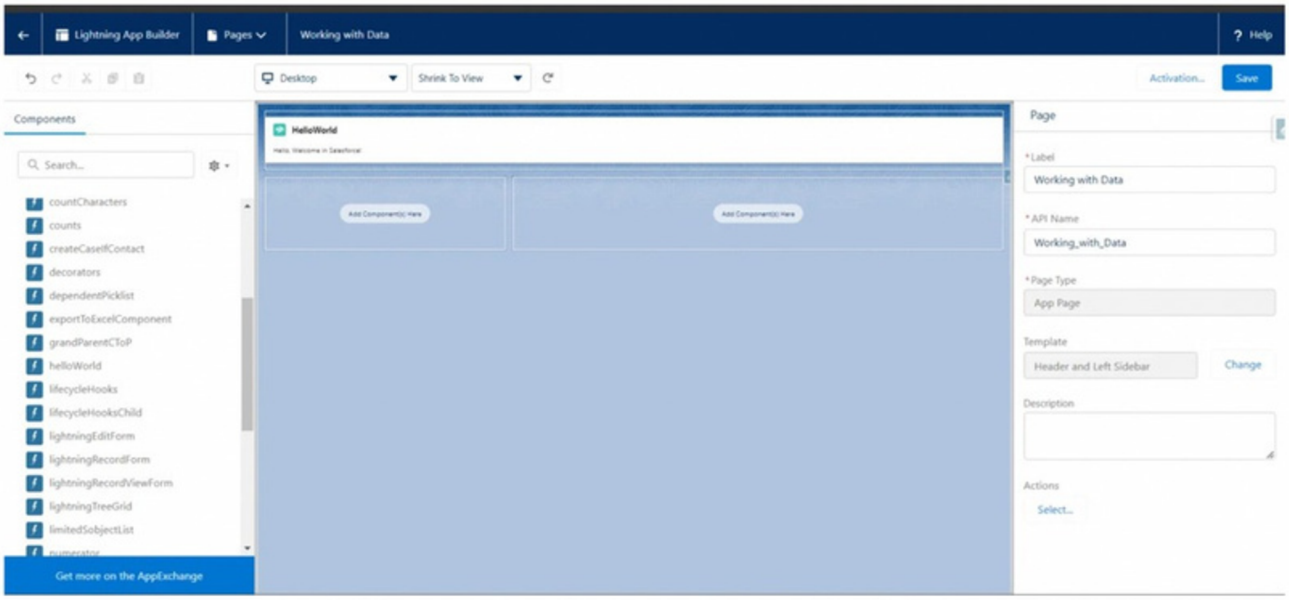
helloWorld.js

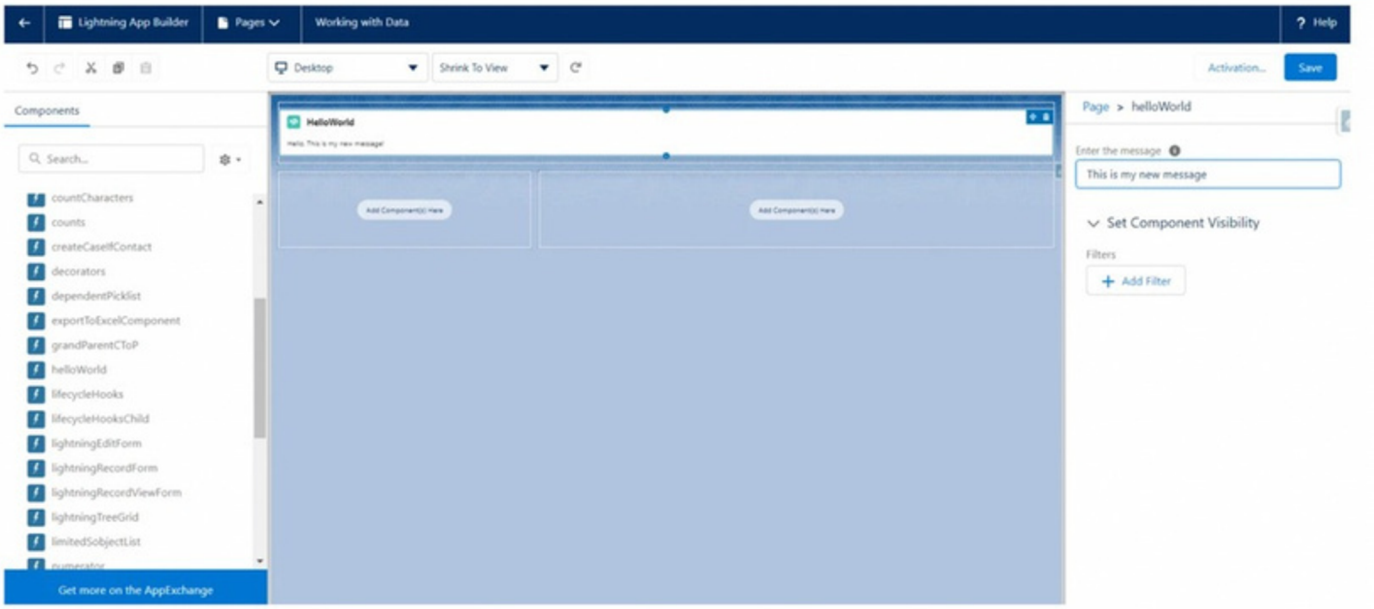
```
import { LightningElement, api } from 'lwc';
export default class HelloWorld extends LightningElement {
@api greeting;
}
```

helloWorld.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle
xmlns="http://soap.sforce.com/2006/04/metadata"
fqn="helloWorld">
<apiVersion>52.0</apiVersion>
<isExposed>true</isExposed>
<targets>
<target>lightning__AppPage</target>
<target>lightning__RecordPage</target>
<target>lightning__HomePage</target>
</targets>
<targetConfigs>
<targetConfig
targets="lightning__HomePage,lightning__AppPage,lightning__Rec
ordPage">
<property name="greeting" type="String" default="Welcome in
Salesforce"
label="Enter the message" />
</targetConfig>
</targetConfigs>
</LightningComponentBundle>
```

OUTPUT





Question 24 - Suppose I have two LWC components parentComp and childComp. How can I access childComp's function in the parentComp's JS?

Answer: Here in the child component we have to annotate our method as @api. so suppose I have method in child component **handleData**

child component

```
@api
handleData(){
// any logic
}
```

parent component

```
handleChange(){
this.template.querySelector('c-child-comp').handleData();
```

Question 25 - What is Lightning Message Service (LMS) in Salesforce?

Answer: Lightning Message Service is used to communicate across different components like between unrelated components, Aura, and VisualForce.

Question 26 - What is Lightning Data Services (LDS) in Salesforce?

Answer: Lightning Data Service (LDS) is a robust feature in Salesforce that enables Lightning Web Components (LWC) to seamlessly interact with Salesforce data, eliminating the need for Apex code.

Functioning on the client side, LDS simplifies data retrieval and management in LWCs, providing key features that enhance the efficiency and ease of working with Salesforce data.

Question 27 - What is the difference between async and await in JS?

Answer: It is a way to write an asynchronous code. Async is a keyword used to define an asynchronous function which returns a promise and await is used to pause that execution until the async returns the promise whether resolved or rejected.

Question 28 - What is the uiRecordApi module?

Answer: uiRecordApi is the module in LWC that provides us with various methods to create, update, and delete records.

Following are the Javascript methods included in the uiRecordApi module:

- createRecord(recordInput)
- createRecordInputFilteredByEditedFields(recordInput, originalRecord)
- deleteRecord(recordId)
- generateRecordInputForCreate(record, objectInfo)
- generateRecordInputForUpdate(record, objectInfo)
- getFieldValue(record, field)
- getFieldDisplayValue(record, field)

Question 29 - Can we perform DML inside the @wire property?

Answer: No It is not possible to perform DML inside wire as it fetches data from the cache.

Question 30 - Why do we use the Super() keyword in LWC?

Answer: When we used the constructor lifecycle hook it was mandatory to use a super() keyword to call the properties of its parent component. It is used to call its parent class in the hierarchy.

Are you preparing for the Salesforce Certifications? Check out the Salesforce certification practice set here

Question 31 - What are component and application events?

Answer: Component events – A component event can be handled by the component that fired the event or by a component in the containment hierarchy that receives the event. They are the events used to communicate across parent and child components.
Application events – They are global events which used to communicate across multiple components. They follow the pub-sub model when an event is fired then all the components handling that event will be notified.

Question 32 - What is the difference between promise.all and promise.race?

Answer: Promise.all take multiple promises and return the promises when all get fulfilled and will reject immediately upon any of the input promises rejected.

Promise.race also takes multiple promises but it returns a single promise whichever settles first or we can say whichever resolves first unlike promise.all.

Question 33 - What are deep copy and shallow copy of objects?

Answer: Shallow copy creates the copy of an object but not for its nested properties. It creates a reference to it. So any changes to the nested properties of a copied object will also change to the original one. Like spread operator creates the shallow copy.
But for a deep copy, it is different, it creates the copy without actually referencing them so any changes in the copied object will not affect the original one and by JSON.stringify(JSON.parse(obj1)) we can create the deep copy of objects.

Question 34 - What does a reduce function do?

Answer: Reduce function iterates on the array's every element and accumulates them into a single value. Below is an example. Here index is basically we can say the collector, currentItem is the current value of the array that is being iterated and 0 is the starting point.

So when it starts to iterate It goes like 0+1 (0 goes into the index as first time and currentItem is the 1) –> 1+2 (now the index value will be 1 and currentItem is 2) it goes on like that.

```
> arr = [1,2,3,4]
  arr.reduce((index,currentItem) => index + currentItem,0);
< 10
>
```

Question 35 - What is Jest in LWC?

Answer: Jest is a fantastic tool designed to simplify writing JavaScript tests. It's ideal for creating unit tests for Lightning web components (LWCs). However, keep in mind that Jest is tailored specifically for LWCs and doesn't support Aura components.

Question 36 - What is the difference between LWC and Aura Component?

Answer:

Aura Components	Lightning Web Components (LWC)
Framework built on top of proprietary concepts.	Uses modern web standards (ES6+, Web Components).
Slower due to the heavy framework overhead.	Faster with lightweight and native browser support.
Component-driven but relies on custom abstractions.	Leverages native JavaScript and browser APIs.
Steeper, requires understanding of Aura-specific syntax.	Easier for developers familiar with modern JavaScript.
Limited cross-platform reuse.	Higher reusability due to standards-based approach.
Two-way binding.	One-way binding with explicit updates for performance.
Relies on polyfills and Salesforce infrastructure.	Built for modern browsers with minimal polyfills.
Requires Salesforce-specific tools.	Supports standard web development tools and practices.

Question 37 - What is data binding in LWC?

Answer: Data binding is a mechanism in LWC that connects the data in the JavaScript controller to the HTML template of a component. This connection ensures synchronization between the underlying data and the user interface, allowing any changes in the data to automatically reflect in the UI.

In LWC, there are two types of data binding mechanisms:

1. One-Way Data Binding
2. Two-Way Data Binding

Question 38 - What is the default binding style in LWC?

Answer: By default Lightning Web Components (LWC) are built with a default one-way data binding approach, meaning data flows in a single direction

Question 39 - What is One-Way Data Binding in LWC?

Answer: One-Way Data Binding

- Data flows in a single direction, from the JavaScript controller to the HTML template.
- This approach ensures better control over data and prevents unintended modifications, as the UI cannot directly update the data in the controller.
- Commonly used for displaying read-only data or passing data to child components.

Question 40 - What is Two-Way Data Binding in LWC?

Answer: Data flows in both directions, allowing updates in the UI to reflect in the JavaScript controller and vice versa.

- Typically achieved using directives like @track or specific event handlers for input fields.
- This approach is useful for interactive components where user input needs to update the underlying data.

Question 41 - What is the 'this' keyword in LWC?

Answer: In Lightning Web Components (LWC), the this keyword is like a bridge to your component's world. It lets you access any property or method defined in your component because it always points to the context where the current code is running.

Example

```
import { LightningElement } from 'lwc';

export default class MyComponent extends LightningElement {
    myProperty = 'Hello, LWC!';

    logProperty() {
        console.log(this.myProperty); // Outputs: Hello, LWC!
    }
}
```

Question 42 - What is Lightning Data Table in LWC?

Answer: A lightning-datatable is a component in LWC that lets you display data in a clean, tabular format. It comes packed with features like column sorting, inline editing, pagination, and custom cell rendering, making it highly versatile. You can populate it with data fetched from Salesforce objects, custom Apex controllers or even static data, giving you flexibility for various business use cases.

Question 43 - What is the use of lightning-record-form?

Answer: The lightning-record-form component is used to quickly create forms for adding, viewing, or updating records. It leverages Lightning Data Service, so no additional Apex controllers are needed for creating or editing record data.

Additionally, it handles field-level security and sharing, ensuring that users only see the data they have access to.

Question 44 - What is Conditional Rendering in LWC?

Answer: Conditional rendering in Lightning Web Components (LWC) lets you dynamically manage what appears on the user interface based on specific conditions. By using directives like **if:true** and **if:false** in the HTML template, you can control the visibility of elements depending on the value of a JavaScript property defined in the component's class.

Example:

```
<template>
    <button onclick={toggleVisibility}>Toggle Visibility</button>
    <template if:true={isVisible}>
        <p>This text appears when isVisible is true.</p>
    </template>
</template>
```

```
import { LightningElement } from 'lwc';

export default class ConditionalRenderingExample extends LightningElement {
    isVisible = false;

    toggleVisibility() {
        this.isVisible = !this.isVisible;
    }
}
```

Question 45 - What is connectedCallback() in LWC?

Answer: connectedCallback() is a lifecycle hook in Lightning Web Components (LWC) that gets executed when a component is inserted into the DOM. It flows from parent to child.

Question 46 - What is renderedCallback() in LWC?

Answer: The renderedCallback() lifecycle hook in Lightning Web Components (LWC) is triggered after the component's template and all its child components are fully rendered in the DOM. This execution follows a flow from child components to the parent component.

Question 47 - What is disconnectedCallback() in LWC?

Answer: In Lightning Web Components (LWC), the disconnectedCallback() is a lifecycle hook that gets triggered when a component is removed from the DOM.

Question 48 - What is event propagation in Salesforce?

Answer: Event propagation in Salesforce is all about how events move from one component to another, especially in Lightning Web Components (LWC). It's like a chain reaction where an event from a child component can pass through the parent components, or the other way around, depending on how the event is set up.

In LWC, events follow two main phases: **bubbling** and **capturing**.

Question 49 - What is Serialization in LWC?

Answer: Serialization in LWC is the process of converting objects into a string format (usually JSON) for easier storage, transmission, or communication between components and systems.

Question 50 - what is getRelatedListCount in LWC?

Answer: The **getRelatedListCount** function in Lightning Web Components (LWC) is a part of Lightning Data Service, and it helps retrieve the count of records in a related list for a specific parent record. This method enables you to efficiently get the number of related records (such as Contacts related to an Account) without the need to fetch the complete set of related records.

Question 51 - Can we write wire in connectedCallBack()?

Answer: No, you cannot directly use @wire inside the connectedCallback() function in Lightning Web Components (LWC).

Question 52 - What is JSON.stringify used for in LWC?

Answer: In Lightning Web Components (LWC), the JSON.stringify() method is used to transform a JavaScript object or value into a JSON string. This comes in handy when you need to work with data in a text format—whether it's for sending it to an external system, displaying it on the user interface, or just for debugging purposes.

Question 53 - How can I get the current user's ID in LWC without Apex?

Answer: Yes, we can retrieve the current user ID without using Apex by simply importing -

```
import Id from '@salesforce/user/Id';
```

Question 54 - How to display toast notifications in Lightning Web Component?

Answer: In LWC, you can display a toast notification by using the ShowToastEvent method from the lightning/platformShowToastEvent module.

Question 55 - What is NavigationmixIn used for in LWC?

Answer: In Salesforce Lightning Web Components (LWC), the NavigationMixin is a handy utility that lets developers programmatically navigate between different pages, records, and apps within Salesforce.

To use the navigation services, we need to import the NavigationMixin from the lightning/navigation module. This allows us to navigate to various destinations, such as list views, object pages, record pages, or even open files, all within our LWC components.

Syntax -

```
import { NavigationMixin } from 'lightning/navigation';
```

Learn more about Navigationmixin here

Question 56 - Can LWC components be used in Visualforce pages?

Answer: Yes, Lightning Web Components (LWC) can be used in Visualforce pages by using the **lightning:container** tag.

Question 57 - How to get the picklist field from a Salesforce object in LWC without using Apex?

Answer: If you want to get picklist field values from a Salesforce object in Lightning Web Components (LWC) without using Apex, you can use the getPicklistValues wire adapter from the lightning/uiObjectInfoApi module. This makes it easy to pull the picklist options for a specific field on any Salesforce object.

Question 58 - Explain the RefreshApex() method in Salesforce.

Answer: refreshApex() extends the capabilities of the standard Lightning Data Service (LDS) by providing greater control and flexibility when fetching data from the server. It allows for manual data refresh, ensuring that the UI always reflects the most accurate and up-to-date information.

Question 59 - What is reportValidity() in LWC?

Answer - reportValidity() is used before submitting a form to ensure all required fields are filled in and meet the validation criteria.

Question 60 - How to retrieve details of a specific Salesforce record in LWC?

Answer: You can use the getRecords wire adapter from the 'lightning/uiRecordApi' module to retrieve data for multiple records in a single call. This allows you to request data for multiple objects or different record types simultaneously.

Question 61 - Does the wire method get called multiple times during the component's lifecycle?

Answer: Yes, the wire method can be called multiple times during a component's lifecycle.

Question 62 - How to import static resources in LWC?

Answer: In Lightning Web Components (LWC), you can import static resources using the **@salesforce/resourceUrl** directive. This lets you access files stored as static resources in Salesforce, including images, stylesheets, JavaScript libraries, and more.

For more valuable content like this

FOLLOW ME ON LINKEDIN

Sandeep Sharma

Be sure to repost to share with your audience