Comprehensive Guide to Lightning Web Component (LWC) Interview Questions

From Basics to Advance

1.What is lifecycle hook?

1. Constructor is called
2. Public property are set on parent
3. Parent is inserted to DOM
4. Once parent is inserted to DOM, Connected callback is called on parent
5. From this connected callback as parent is already inserted, you can reference different DOM elements
6. Parent is rendered
7. Once parent is rendered, Constructor is called on the child
8. Public property of child are set
9. Child is inserted to DOM
10. Once child is inserted to DOM, Connected callback is called on child
11. Child is rendered
12. Once child is rendered, rendered callback is called
13. Once child components are rendered on the screen, then parents rendered callback is called.

constructor()

- Hook that fires when a component instance is created.
- Don’t use a return statement inside the constructor body, unless it is a simple early-return (return or return this).

- At that point, the component properties won’t be ready yet.
- It flows from parent to child.

```
import { LightningElement } from 'lwc';
export default class App extends LightningElement {
constructor() {
super();
console.log('In Constructor');
}
```

connectedCallback()

- Fires when a component is inserted into the DOM.
- The component properties will be ready, but the child elements won’t be yet.
- It can fire more than once. For example, if you remove an element and then insert it into another position, such as when you reorder a list, the hook fires several times. If you want code to run one time, write code to prevent it from running twice.
- It flows from parent to child.
- The equivalent in Aura is the init() event.

```
import { LightningElement } from 'lwc';
export default class App extends LightningElement {


connectedCallback(){
console.log('In connectedCallback');
}
}
```

render()

- Hook that overrides the standard rendering functionality.

- Gets invoked after connectedCallback() and must return a valid HTML template.
- It can fire more than once.
- It flows from parent to child.

```
import { LightningElement } from 'lwc';
export default class App extends LightningElement {

render(){
console.log('In render');
}
}
```

renderedCallback()

- Fires when a component rendering is done.
- It can fire more than once.
- It flows from child to parent.

The renderedCallback() is unique to Lightning Web Components.

```
import { LightningElement } from 'lwc';
export default class App extends LightningElement {

renderedCallback(){
console.log('In renderedCallback');
}
}
```

disconnectedCallback()

- Fires when a component is removed from the DOM.
- It can fire more than once.

- It flows from parent to child.
- The equivalent in aura was the unrender() methods of the custom renderer file.

When this component removed form DOM, it will fire disconnectedCallback and clear array.

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

export default class App extends LightningElement

{

disconnectedCallback(){

console.log('In disconnectedCallback');

}

}
```

errorCallback(error, stack)

Captures errors that may happen in all the descendant components lifecycle hooks.

```
errorCallback(error, stack) {

alert(error);

}
```

2.Why we Use of lifecycle hooks ?

Dynamic rendering of components: we can use the render() hook to render different templates according to some conditions.

Component that is an error boundary for the descendent components lifecycle hooks: we can use the errorCallback() to implement such component.

3.How to iterate over multiple items in LWC?

To iterate over list we can use two directives

for:each

Iterator

Whenever we use for:each or Iterator we need to use key directive on the element on which we are doing iteration. Key gives unique id to each item.

4.Explain Component Communication In LWC?

lightning web component can be nested and there are 4 possibilities for communication between components.

- Parent to child communication
- Child to parent communication
- Communication between two separate components.
- Lightning Web Component to aura component

4.How to pass data from Parent to child communication?

To pass data down in the component hierarchy, the **child component must declare a public API**. There are two forms in the public API of a component.

- public properties
- public methods

5.Define parent to child communication using Public Property?

If you want to make the property to be called from another component, you need to declare this property with **@api decorator in the calling component**. Along with decoration you have to import this in your js file as:

import { LightningElement, api } from 'lwc';

6.Define parent to child communication using Public Property?

Created a method and used api decorator before it so as to expose it in parent component.

7. Difference between render and renderedCallback()?

Render

- Hook that overrides the standard rendering functionality.
- Gets invoked after connectedCallback() and must return a valid HTML template.
- It can fire more than once.
- It flows from **parent to child.**

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

export default class App extends LightningElement {

render(){

console.log('In render');

}

}
```

renderedCallback

- Fires when a component rendering is done.
- It can fire more than once.
- It flows from child to parent.

The renderedCallback() is unique to Lightning Web Components. renderedCallback() method in child component will get fired first.

8.**What is the difference between wire and imperative in LWC?**

1. Wire Service (Automatic and Reactive):

When you "wire" a function or property, Salesforce automatically listens for changes that affect the data and retrieves fresh data when necessary. It's

reactive; it updates the component's data whenever the relevant Salesforce data changes.

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

import getPizzas from '@salesforce/apex/Dominos.getPizzas';

export default class PizzaMenu extends LightningElement {

@wire(getPizzas) pizzas;

}
```

In this example, pizzas gets updated automatically if the data in Salesforce changes.

2. Imperative Apex (Manual and More Control):

You call Apex methods directly at the specific times you decide, like in response to a user action (e.g., clicking a button), or a lifecycle event, etc.

Differences:

Reactivity: wire is reactive and automatic, while imperative calls are manual and only happen when specifically invoked.

Control: Imperative calls give you more control over error handling, complex logic, and can be used for more than just fetching data (e.g., sending data, calling multiple methods consecutively, etc.), whereas wire is mainly used for directly wiring data to properties or functions.

9. How to pass data from Child to Parent?

Data must be passed up using Events. Events are just standard JavaScript events that we fire from the child component and listen into the parent.

Note: To communicate **down** the component containment hierarchy, we can pass properties to the **child via HTML attributes, or public methods**. To communicate down the component hierarchy we don't need events.

10.How do we communicate between different components which don't have relation?

To communicate between two different components that do not have any direct relation, we use publish-subscribe utility.

11.How to Expose Apex Methods to Lightning Web Components?

To expose an Apex method to a Lightning web component

- method must be static
- method must be either global or public
- Annotate the method with @AuraEnabled

12.What is the use of Client-Side Caching?

- Annotating the Apex method with @AuraEnabled(cacheable=true) **improves the run time performance.**
- To set cacheable=true, a method must only get data, **it can't mutate (change) data.**
- Marking a method as **cacheable improves component's performance** by quickly **showing cached data** from client-side storage without waiting for a server trip.
- If the cached data is stale, the framework retrieves the latest data from the server. Caching is especially beneficial for users on high latency, slow, or unreliable connections.
- To use @wire to call an Apex method, you must set cacheable=true

13.How to Wire an Apex Method to a Property?

If an Apex method is annotated with @AuraEnabled(cacheable=true), we can invoke it from a component via the wire service.

import apexMethodName from '@salesforce/apex/Namespace.Classname.apexMethodReference';

@wire(apexMethodName,{ apexMethodParams })propertyOrFunction;

14.What is the purpose of the Lightning Data Service in LWC?

The purpose of the Lightning Data Service in LWC is to provide a declarative and efficient way to perform CRUD (Create, Read, Update, Delete) operations on Salesforce records. It is a framework-provided service that works as a data layer between the LWC component and the Salesforce database.

15.What are the benefits of The Lightning Data Service?

- **Declarative data binding:** With the Lightning Data Service, you can declaratively bind your component's UI elements to record data without writing any Apex code or SOQL queries.
- **Caching and automatic record updates:** The Lightning Data Service caches record data locally on the client side and automatically updates it when changes occur in the database. This improves performance and reduces the number of round trips to the server.
- **Automatic CRUD operations:** The Lightning Data Service provides a set of methods to create, read, update, and delete records. These methods are automatically implemented and available for use in your LWC component.
- **Automatic sharing and field-level security:** The Lightning Data Service **automatically enforces sharing rules and field-level security** when reading or modifying records.

16. If parent component A and there are two component B and C as child components. How can we communicate between B and C ?

We should fire up event from child component B to parent A then from A call attribute / function exposed (as @api) and pass data into C component.

17.What is LMS?

LMS is defined as the standard publish-subscribe library that enables communication with DOM across the components be it Visualforce Pages, Aura components, and Lightning Web Components (LWC) all can use it to publish message and listen to messages published by others.

18.Do we have application events in LWC?

Ans: We dont have application event as such in LWC like Aura rather we have LMS in LWC to communicate between components which are not part of same hierarchy.

19.How can we navigate user from LWC component to record detail page?

We can do it using NavigationMixin service

20.Can I get current user ID in LWC without apex?

Yes we can get current user ID without apex by simply importing

import Id from '@salesforce/user/Id'

21.Can i call function annotated with @AuraEnabled(cacheable= true) imperatively ?

Ans: Yes

22.Can we do DML in method annotated with @AuraEnabled(cacheable= true)?

Ans: No we can't do DML inside any method annotated with cacheable = true , you will receive an error as DMLLimit Exception.

23.How to refresh cache when calling method imperatively?

We have to use getRecordNotifyChange(RecordIds) which refreshes the LDS cache providing you the latest data this will work only if cacheable = true was there. Otherwise we will have to call the function again from our js to get the latest data.

24.When do we face error of "Cant assign to Read only property" in LWC?

This error occurs when you are trying to make changes to a property which is marked as @api , ideally you should clone the value then make the changes to it.

25.How do you handle errors and exceptions in LWC?

Error handling in LWC can be done using try-catch blocks or by leveraging the onError lifecycle hook. You can catch exceptions and display appropriate error messages to the user, ensuring a smooth user experience.

<template>
try {
// Code that may throw an exception
} catch (error) {
// Handle the error and display an error message
}
</template>

26.How do you make an HTTP callout from a Lightning Web Component?

You can use the fetch API to make HTTP callouts in LWC . Import the fetch method and use it to send requests to external services.

27.What is the purpose of the lightning-record-edit-form component in LWC?

lightning-record-edit-form is used to create, view, or edit a record's fields using the Salesforce Lightning Data Service.

28.How can you communicate between sibling components in LWC?

You can use custom events and properties defined in a common parent component to facilitate communication between sibling components.

29.How can we track the database changes in the Lightning web component and refresh UI on data updates?

If we need to track the changes done by **Lightning Data Service** then we can use the **getRecordChangeNotifiy()** function from the LDS.

30. Suppose you have written a LWC which has the @wire decorator to get the data from apex. But you are unable to get the data from the server. What could be one of the reason?

check whether you have used **cacheable=true** along with @AuraEnabled annotation in the apex method.

32.Can I call function annotated with @AuraEnabled(cacheable= true) imperatively ?

Yes

33.How to ensure that your LWC respects FLS?

1. Use Standard Components Where Possible: Use Lightning Data Service components

like lightning-record-form, lightning-record-view-form, and lightning-record-edit-form, respect FLS automatically. Whenever possible, use these components to handle data input/display.

<lightning-record-view-form record-id={recordId} object-api-name=”Account”>

<lightning-output-field field-name=”Name”></lightning-output-field>

<!– other fields –>

</lightning-record-view-form>

In this example, FLS is enforced for the “Name” field of the “Account” object, and if a user doesn’t have permission to view the “Name” field, it won’t be displayed.

2. Check FLS in Apex:

When writing Apex controllers that are called from your LWC, always check FLS. Use the **SecurityEnforced** annotation.

public with sharing class AccountController {

@AuraEnabled(cacheable=true)

public static Account getAccount(Id accountId) {

```
return [SELECT Id, Name FROM Account WHERE Id = :accountId
WITH SECURITY_ENFORCED];
}
}
```

The WITH SECURITY_ENFORCED clause ensures that the query respects FLS, throwing an exception if the user lacks the necessary permissions.

3. Use Schema Methods in Apex:

Utilize Schema methods in Apex to check if the current user has read, create, or edit access to a field. This is more manual but allows for fine-grained control.

```
public with sharing class SecurityCheck {
public static Boolean isFieldAccessible() {
return Schema.sObjectType.Account.fields.Name.isAccessible();
}
}
```

34. What do you mean by cacheable = true annotations ?

First of all when you mark function as cacheable = true it can only retrieve data i.e. it cant have any DML.
It is used to improve component performance by quickly showing cached data from client side storage without waiting for server trip.

35. Why do we use $ when passing property in wire function, what does it mean?

Ans: $ prefix tells the wire service to treat values passed as a property of the class and evaluate it as this.propertyName and the property is reactive. If the property's value changes, new data is provisioned and the component renders.

36. **When is the wire method/property called in the lifecycle of a component ?**

Wired service is called right after component is created ie after constructor and is again called when parameter that you are passing is made available.

37. Is wire method called multiple times during lifecycle of component ?

Ans: True

38. What is the difference between event.StopPropogation() and Event.preventDefault()?

Ans: stopPropagation prevents further propagation of the current event in the capturing and bubbling phases. preventDefault prevents the default action the browser makes on that event.

39. How can i reference record ID of page in my component which is fired from quick action.

There are two types of quick actions in LWC :
Screen actions : Normal action which open a modal
Headless actions : Which dont have any html file rather just logic written in javascript and no modal window will come up
In case of screen actions we will be able to get recordID by just defining recordID as public property in the component.
For headless action you have to define @api invoke method which is auto called and recordID will be set after this function is called.

40. What is a promise in async transactions? What are it different stages

Promise is object returned in async transaction which notifies you about completion or failure of transaction.
For example when you call apex imperatively it returns you a promise object on the basis of object returned execution either goes into (then) ie transaction was successful or (catch) which means transaction failed.

Stages of promises are :

Pending: Waiting for result.
Fulfilled: Promise results in success.
Rejected: Promise result in failure.

41. Why do we extend LightningElement?

LightningElement is custom wrapper on HTMLElement which actually contains all the lifecycle hooks methods over which Salesforce did some customization.

42. How to send data from Parent to Child in LWC?

For parent to child communication you just need to expose your function or attribute as @api then in your parent component you can use querySelector to actually query the child component access exposed attribute or method.

43. What does composed = true does in an event ?

These type of events can bubble up inside dom and also able to cross shadow boundary.

44. What is callback hell?

Callback hell occurs when there are many functions you want to call async and you end up putting them one side each another and as the code grows it becomes very difficult to read for example :

```
getData(function(x) {
    getMoreData(x, function(y) {
        getMoreData(y, function(z) { ... });
    });
});
```

45. What are decorators in LWC (Lightning web components) in Salesforce? Latest interview question.

Ans: The Lightning Web Components programming model has three decorators that add functionality to a property or function. There are 3 decorators for LWC
@track , @wire, @api

46. When do I use @track on a property ? Do I still need it considering all properties are by default reactive now?

After Spring 20 all the properties are made by default reactive ie we dont need @track for primitive properties. We still need it for array or object type properties

47. Can I use multiple decorators on one property ?

No we cant use multiple decorators on same property.

48. Can I deploy component with empty css file ?

No we cant do that

49. Can I get current user ID in LWC without apex?

Ans: Yes we can get current user ID without apex by simply importing import Id from '@salesforce/user/Id'

48. Can i call function annotated with @AuraEnabled(cacheable= true) imperatively ?

Ans: Yes

49. Can we do DML in method annotated with @AuraEnabled(cacheable= true)?

Ans: No we cant do DML inside any method annotated with cacheable = true , you will receive an error as DMLLimit Exception.

50. How to refresh cache when calling method imperatively ?

Ans: We have to use getRecordNotifyChange(RecordIds) which refreshes the LDS cache providing you the latest data this will work only if cacheable = true was there.
Otherwise we will have to call the function again from our js to get the latest data.

51. How can you pass data between LWC components?

Answer:

There are several ways to achieve this, depending on the relationship between the components. The most common methods include using public properties, custom events, and the Lightning Message Service (LMS). Understanding these methods is crucial for building robust and maintainable applications.

1. Using Public Properties Public properties are used to pass data from a parent component to a child component. The child component exposes a public property using the @api decorator, and the parent component binds data to this property.

2. Using Custom Events Custom events are used to pass data from a child component to a parent component. The child component dispatches an event with the data, and the parent component listens for this event and handles the data.

the child component dispatches a custom event with a message in the detail property when a button is clicked. The parent component listens for this event and handles the data in the handleEvent method.

3. Using Lightning Message Service (LMS) The Lightning Message Service (LMS) is used to communicate between components that do not have a direct parent-child relationship, even if they are in different parts of the application.

52. How do you handle asynchronous operations in LWC?

Asynchronous operations ensure that the user interface remains responsive while waiting for these tasks to complete. In LWC, asynchronous code can be managed using JavaScript's Promises and the async / await syntax.

Using Promises and async / await JavaScript Promises represent the eventual completion (or failure) of an asynchronous operation and its resulting value.

- **Promises:** Represent asynchronous operations and can be handled using .then() and .catch() methods.

- **async/await:** Syntactic sugar built on top of Promises, allowing for a more readable and concise code structure.

Steps

1. **connectedCallback Method:** The connectedCallback lifecycle hook is used here to fetch data as soon as the component is inserted into the DOM. This ensures the initial data is available when the component is rendered.
2. **async Keyword:** Declares the connectedCallback function as asynchronous, allowing the use of await within it.
3. **await Keyword:** Pauses the execution of the connectedCallback function until the fetchData Promise resolves. This ensures that this.data is assigned the fetched data before the method completes.

Using async / await , the code becomes more readable and easier to maintain compared to traditional Promise chaining with .then() and .catch().

53. Give me a scenario where you will use Lightning Message Service in LWC?

There is a parent component that has two child components. One child component is responsible for displaying a list of items, while the other child component is responsible for displaying the details of the selected item. When the user selects an item from the list, the first child component can send a message on a specific channel with the details of the selected item. The second child component, which has subscribed to the same channel, can receive the message and update its display with the new details.

54. Can you explain the role of the Salesforce Object Query Language (SOQL) in LWC?

SOQL is a query language used in Lightning Web Components (LWC) to retrieve data from the Salesforce database. The wire adapters and imperative Apex calls in LWC provide developers with ways to make server-side calls to retrieve data using SOQL.

55. Explain Bounded and Unbounded Expressions in LWC.

In LWC, expressions are used for dynamically displaying the data in the template. There are two types different types of expressions in LWC:

- **Bounded expressions** are enclosed within double curly braces ({{ }}). They are used to display data values in the template that are derived from the component's JavaScript class or HTML attributes.
- **Let’s understand this through an example -**

```
<template>
  <p>Hello {{name}}, welcome to my website!</p>
  <p>The result of adding 3 and 2 is {{addNumbers(3, 2)}}.</p>
</template>
```

- **Unbounded expressions** are enclosed within single curly braces ({ }). They are used to evaluate JavaScript expressions in the template itself. Unbounded expressions can be used to assign values to HTML attributes or to conditionally render HTML elements.

Let’s understand this also with an example -

```
<template>
  <input type="text" value={inputValue}>
  <template if:true={showMessage}>
    <p>The message is: {message}</p>
  </template>
</template>
```

56. How can you render multiple templates in LWC?

In LWC, we can display multiple templates conditionally based on the component's state using the **if:true** directive.

Here's an example of how to display multiple templates in LWC:

```
<template>
```

```
    <template if:true={showTemplate1}>
        <p>This is template 1</p>
    </template>
    <template if:true={showTemplate2}>
        <p>This is template 2</p>
    </template>
</template>
```

57. How do you handle user input in LWC forms?

We can handle user input in forms using event handlers and data binding.

- **Event Handlers:** LWC provides some built-in event handlers that we can use for handling user inputs, like - **onClick, onChange, onSubmit**, etc. For example, Suppose we want to handle a button click event, we can define an **onClick** event handler on the button element like this:
- **Data Binding:** LWC has some data binding features inbuilt that help in binding the value of an input element to a property in the component state.

58. Explain in detail about @api and @track decorators in LWC.

In LWC, both **@api** and **@track** decorators are used to define properties in a component, but they serve different purposes.

- The **@api** decorator is used to define a public property that can be accessed by other components. This allows passing data between components in a parent-child relationship, or between unrelated components using the Lightning Message Service. Here's an example of how we can use the **@api** decorator to define a public property:

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


export default class MyComponent extends LightningElement {
```

```
    @api message = 'Hello Smriti;
}
```

Now the component is defined. Other components can access this property using dot notation, like this:

```
<c-my-component message="Hello from parent"></c-my-component>
```

- The **@track** decorator is used to define a reactive property, It means suppose there are any changes in the property, then the whole component will be re-rendered. This is useful when we need to update the component's UI based on changes to the property. Here's an example of how we can use the @track decorator to define a reactive property:

58. Explain the concept of data binding in LWC?

Data binding is a core concept in LWC that allows developers to establish a relationship between a component's properties and the values displayed in its HTML template.

In LWC, there are two different data binding used -

- **Property binding:** It is used to bind a component's property to a value in the HTML template.

  ```
  <p>{message}</p>
  ```

Whenever the "message" property in the component is changed, the value displayed in the HTML template will be automatically updated to reflect the new value.

- **Event binding:** This is used to bind an HTML element's event to a method in the component. To do this, we need to use the **on-syntax** followed by the event name and the name of the method to be called whenever the event is triggered. <button on-click={handleClick}>Click me</button>

59. What are the benefits of using LWC over traditional JavaScript frameworks?

- **Modern web standards:** LWC is built using modern web standards like HTML, CSS, and JavaScript. This makes it easier to get started with LWC development.
- **Lightweight and fast:** It can be loaded and executed quickly. This results in faster page load time and a better user experience.
- **Modular architecture:** It allows developers to build reusable and maintainable components, this saves a lot of time and effort in the long run.
- **Compatibility with Salesforce:** LWC is fully compatible with the Salesforce platform.

60. What is the role of the Shadow DOM in LWC, and how does it differ from the traditional DOM?

In LWC, the Shadow DOM is used to encapsulate the component's DOM hierarchy and prevent CSS styles and JavaScript code from bleeding out of the component and interfering with the rest of the page.

The Shadow DOM is a part of the web standards and it is supported by modern browsers. It allows the creation of a separate DOM tree inside the component that is hidden from the outside world. This way, the component's styles, and behaviour are self-contained and it doesn't affect other parts of the page.

In contrast to the traditional DOM, which is global and mutable.

When we create an LWC component, its HTML template, and CSS styles are automatically encapsulated in the Shadow DOM.

61. Can you explain how to use the Lightning Message Service in LWC and give an example?

To use the Lightning Message Service in LWC, we are required to follow these steps:

- **Define a message channel:** A message channel defines the topic or theme of the messages that will be published and subscribed to. We can define a message channel using the “lightning/messageService” module.

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

import { createMessageChannel } from 'lightning/messageService';
```

- **Publish a message:** After defining a message channel, we can publish messages on that channel using the **publish()** method.

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

import { createMessageChannel, publish } from 'lightning/messageService';
```

- **Subscribe to a message:** Finally, to receive and act upon messages published on a message channel, we can use the **subscribe()** method.

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

import { createMessageChannel, subscribe } from
'lightning/messageService';
```

62. How do you use the Salesforce REST API in LWC? What are the best practices for integrating with external systems?

To use the Salesforce REST API in LWC, we can use the built-in fetch method or a third-party library like **axios** or jQuery.

63. Can you explain how to use the LWC component cache and why it is important for performance?

When a user loads a page that contains LWC components, the browser needs to download and render those components. This problem can be solved using the LWC component cache.

This helps speed up this process by storing copies of the components in the browser's cache.

So, in the case when the browser does not need to download the components every time the user accesses the page. Instead of this, the cached components can be quickly retrieved and displayed. This helps in reducing page load time and improves the user experience.

To enable the LWC component cache, we need to use the **@wire** decorator with the **getRecord** or **getListUi** wire adapters. These adapters automatically cache the results of the wire call in the browser's cache, which can help improve performance.

64. How do you create a Lightning App that embeds an LWC component?

To get started, first, we need to create the LWC component that we want to embed.

Once we have the component, then we need to create the Lightning App itself.

Once the app is created, we can then embed the LWC component within it.

```
<aura:application extends="force:slds">
    <c:myLwcComponent />
</aura:application>
```

65. Explain SLDS?

The term SLDS stands for Salesforce Lightning Design System.

Salesforce Lightning Design System is used to develop web applications with great lightning experience, without writing a single line of code in CSS.

66. Is there any possibility of having multiple wire methods in Lightning Web Component?

Yes, there is a possibility of having more than two wire methods in Lightning Web Components.

66. Why does LWC work faster than Aura components?

Normally, the lightning web components work faster than aura components. This is because due to the absence of an abstraction layer, LWC is a lightweight framework built using web stack tools such as HTML and JavaScript.

67. How do Lightning Web components communicate with Aura components?

There is a possibility that Aura components communicate with Lightning Web Components with the help of Public APIs and Events.

68.Can Lightning Web Components can be embedded within Aura Components?

True, Lightning Web Components can be embedded within Aura Components whereas the reverse cannot happen.

69. What is the structure of LWC?

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

1. **HTML** – It defines the markup of the LWC.
2. **JS** – It establishes the component logic and behaviors of LWC.
3. **CSS** – It defines the styling of LWC.
4. **XML** – It includes the component's metadata.

70.What is the uiRecordApi module?

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)

71. 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.

72. What is the difference between promise.all and promise.race?

Answer – **Promise.all** take multiple promises and return the promises when all gets fulfilled and will reject immediately upon any of the input promises rejecting.

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.

73. Explain code reusability in LWC?

Use Pre-Built Components

Instead of building your own button, use the standard <lightning-button> component provided by Salesforce.

<lightning-button label="Click Me" onclick={handleClick}></lightning-button>

Build Custom Components

Create a reusable custom-card component for displaying information in a card format.

Combine Small Components

Share Code with Helper Functions. Create a utility JavaScript file `utils.js` with a common function.

Use Shared Styles

Create a shared CSS file with common styles.

Reuse Backend Code

Create an Apex with a reusable method.

Use Events for Communication

Emit an event from a child component and handle it in the parent component.

74. How to ensure FLS in LWC?

Field-Level Security (FLS) in Lightning Web Components (LWC) is crucial to ensure that users only see and interact with the data they are authorized to access.

Use @wire Adapters with FLS Checks

Salesforce provides wire adapters like getRecord, getObjectInfo, and getFieldValue` that automatically respect FLS. When using these adapters, Salesforce will enforce field-level security, ensuring users do not see fields they do not have permission to view.

import { getRecord, getFieldValue } from 'lightning/uiRecordApi';

Check FLS in Apex Controllers

When LWC interacts with Apex, FLS must be checked on the server-side because Apex runs in system mode, bypassing FLS by default. To enforce

FLS, explicitly check the user's field-level permissions before performing operations.

Lightning Data Service (LDS)

When using LDS (eg lightning-record-view-form, lightning-record-edit-form, lightning-input-field), FLS is automatically enforced.

Custom Logic and FLS

For custom logic or UI elements that need to respect FLS, you can use the UserRecordAccess object in SOQL or Apex methods like isAccessible(), isUpdateable(), and isCreateable() to check field-level permissions.

75.When to Use Lightning Data Service (LDS) Vs Wire Adapter?

When to use LDS

Use LDS when you want to create or display forms, such as viewing or editing records, with standard Salesforce components like lightning-record-view-form or lightning-record-edit-form

LDS automatically handles security, including Field-Level Security (FLS). You don't need to worry about users seeing data they shouldn't.

When to Use @wire Adapter

- For Custom Data Handling: Use @wire adapters when you need more control over the data, like fetching specific fields, combining data from multiple sources, or performing calculations.

They are more flexible and allow you to manipulate the data before displaying it. If you only need specific fields and not a full record, @wire adapters can be more efficient.

76. Difference between using Lightning Data Service (LDS) components and standard @wire adapters with getRecord and getFieldValue?

Suppose you want to display the Name and Industry fields of an Account record in your LWC.

Using Lightning Data Service (LDS) Components

```
<template>
    <lightning-record-view-form record-id={recordId} object-api-name="Account">
        <lightning-output-field field-name="Name"></lightning-output-field>
        <lightning-output-field field-name="Industry"></lightning-output-field>
    </lightning-record-view-form>
```

lightning-output-field handles the display of each field. It automatically formats the field value and applies standard Salesforce styling. LDS takes care of fetching the data, respecting FLS, and displaying it in a consistent format. There's no need for custom data fetching or handling logic.

Using @wire Adapters with getRecord and getFieldValue

```
import { LightningElement, wire } from 'lwc';
import { getRecord, getFieldValue } from 'lightning/uiRecordApi';
import NAME_FIELD from '@salesforce/schema/Account.Name';
import INDUSTRY_FIELD from '@salesforce/schema/Account.Industry';

export default class AccountInfo extends LightningElement {
```

```
    @api recordId;
    @wire(getRecord, { recordId: '$recordId', fields: [NAME_FIELD, INDUSTRY_FIELD] })
    account;
    get name() {
        return getFieldValue(this.account.data, NAME_FIELD);
    }

    get industry() {
        return getFieldValue(this.account.data, INDUSTRY_FIELD);
    }
}
```

Here, getRecord is used to fetch the record data, and getFieldValue is used to extract specific field values.You can decide exactly how to display the data. For example, you can wrap the field values in custom HTML and apply your own styles or formatting.

This approach gives you full control over the presentation and any additional logic you want to apply to the data, such as conditional rendering, formatting, or integrating with other data sources.

LDS is best for straightforward use cases where standard presentation is sufficient, while @wire with getRecord and getFieldValue is suited for more customized and complex data handling and presentation needs.

77. How to use LDS to show data from two objects?

Imagine you want to display information from both the Account and Contact objects in a single component.

Using LDS Components

You can use multiple lightning-record-view-form components, one for each object. Each form can display fields from the respective object.

```
<template>
    <!-- Display Account Information -->
    <lightning-record-view-form record-id={accountId} object-api-name="Account">
        <lightning-output-field field-name="Name"></lightning-output-field>
        <lightning-output-field field-name="Industry"></lightning-output-field>
    </lightning-record-view-form>
    <!-- Display Contact Information -->
    <lightning-record-view-form record-id={contactId} object-api-name="Contact">
        <lightning-output-field field-name="FirstName"></lightning-output-field>
        <lightning-output-field field-name="LastName"></lightning-output-field>
        <lightning-output-field field-name="Email"></lightning-output-field>
    </lightning-record-view-form>
</template>
```

78. How To get records from two different objects and still maintain Field-Level Security (FLS) in Lightning Web Components?

You can use either Lightning Data Service (LDS) components or @wire adapters. Both approaches ensure that FLS is respected, meaning users only see the fields they have permission to access.

Using Lightning Data Service (LDS) Components

With LDS, you can use separate lightning-record-view-form components for each object. Each form will handle FLS and display only the fields that the user is allowed to see.

Using @wire Adapters

Alternatively, you can use @wire adapters to fetch data from different objects. You will need separate wire methods for each object. FLS is automatically respected when using getRecord.

Choose the method that best fits your needs for control, customization, and ease of use. LDS components are easier and more convenient for standard form layouts, while @wire adapters offer more flexibility for custom logic and data handling.

79.In Lightning Web Components (LWC), there isn't a direct equivalent to Aura's refreshView()`method, how can still implement data refresh functionality?

Reactive Properties with @wire

Change a reactive property that is used by the @wire adapter. When this property changes, the wire adapter automatically re-fetches the data. This approach leverages LWC's reactive nature to refresh data.

Imperative Apex Calls

Manually call Apex methods using JavaScript. When you need to refresh the data, simply call the Apex method again. This gives you explicit control over when and how data is fetched.

Using refreshApex

If you are using @wire in combination with imperative Apex calls, you can use refreshApex. This function re-executes the @wire service, refreshing the data.

```
// Re-fetches the data wired to wiredRecordResult

refreshApex(this.wiredRecordResult);
```

80. When and when not to Refresh Data in LDS?

Manual Data Changes Outside of Salesforce UI:

Let's say, If data is changed outside the standard Salesforce UI, such as through an external system integration, API call, or custom Apex operation, and those changes need to be reflected immediately in your LWC.

Use methods like refreshApex to ensure the latest data is displayed.

When Not to Refresh Data in LDS

Standard Data Updates within Salesforce UI

If data is updated through standard Salesforce UI mechanisms (e.g., standard record pages, related lists), LDS typically handles these updates automatically. No manual refresh is needed, as LDS ensures that any such updates are reflected in the components using LDS.

81. Explain the use case of notifyRecordUpdateAvailable(recordIds)?

Imagine a resturant with a digital ordering system. There are two screens:

1. **Order Entry Screen:** Where waitstaff enter and update orders.
2. **Kitchen Display Screen:** Where chefs see the latest orders to prepare.

Using notifyRecordUpdateAvailable(recordIds)

When an order is updated (for example, a customer changes their meal), the kitchen should immediately see this update.

After updating the order, the system needs to inform the Kitchen Display

Screen that the order has changed. This is where notifyRecordUpdateAvailable(recordIds) comes into play. This function acts like a messenger telling the Kitchen Display Screen, "Hey, the order for table 5 has been updated."

Refreshing the Kitchen Display:

Upon receiving this notification, the Kitchen Display Screen fetches the latest data for the updated order.

In the context of Lightning Web Components (LWC), notifyRecordUpdateAvailable(recordIds) helps keep all parts of the application in sync by ensuring that any component displaying data stays up-to-date when that data changes elsewhere in the system. This function is particularly useful in real-time applications where immediate data accuracy is crucial.

```
import { updateRecord, notifyRecordUpdateAvailable } from 'lightning/uiRecordApi';
```

82. Give syntax for wire service?

Use Cases with Syntax Examples

Fetching a Record with getRecord

```
import { LightningElement, wire, api } from 'lwc';
import { getRecord } from 'lightning/uiRecordApi';
import NAME_FIELD from '@salesforce/schema/Account.Name';
import INDUSTRY_FIELD from '@salesforce/schema/Account.Industry';

export default class AccountDetail extends LightningElement {
    @api recordId;

    @wire(getRecord, { recordId: '$recordId', fields: [NAME_FIELD, INDUSTRY_FIELD] })
    account;
}
```

Fetching Object Info with getObjectInfo

Fetches metadata about a Salesforce object, like its label or available fields.

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

import { getObjectInfo } from 'lightning/uiObjectInfoApi';

import ACCOUNT_OBJECT from '@salesforce/schema/Account';


export default class AccountMetadata extends LightningElement {

    @wire(getObjectInfo, { objectApiName: ACCOUNT_OBJECT })

    objectInfo;

}
```

Using a Custom Apex Method

Fetches data using a custom Apex method.

```
import { LightningElement, wire, api } from 'lwc';

import getAccounts from
'@salesforce/apex/AccountController.getAccounts';


export default class AccountList extends LightningElement {

    @wire(getAccounts) accounts;

}
```

Reactive Properties: Use $ before a property name in the @wire configuration to make it reactive. This means the wire service will re-fetch the data whenever the property value changes.

- Data Handling: The data fetched by @wire can be assigned to either a property (for automatic updates) or a function (for custom handling).

- Error Handling: When wired to a property, the data structure includes data and error properties, which can be checked to handle success and error cases accordingly.

83. Syntax for imperative calls in LWC?

Syntax for Imperative Apex Calls

Calling an Apex Method Imperatively

import getAccounts from '@salesforce/apex/AccountController.getAccounts' imports the Apex method.

getAccounts() is called imperatively, usually within a method like loadAccounts

The method returns a promise, allowing you to handle the response with .then() and .catch() for success and error handling, respectively.

Apex Class:

```
public with sharing class AccountController {
    @AuraEnabled(cacheable=true)
    public static List<Account> getAccounts() {
        return [SELECT Id, Name, Industry FROM Account];
    }
}
```

JavaScript Controller:

```
import { LightningElement, track } from 'lwc';
import getAccounts from '@salesforce/apex/AccountController.getAccounts';

export default class AccountList extends LightningElement {
```

```
    @track accounts;
    @track error;

    connectedCallback() {
        this.loadAccounts();
    }

    loadAccounts() {
        getAccounts()
            .then(result => {
                this.accounts = result;
            })
            .catch(error => {
                this.error = error;
            });
    }
}
```

Calling a Standard LDS Service Imperatively

You can also call standard Lightning Data Service (LDS) functions like createRecord ,updateRecord, or deleteRecord imperatively.

84.**Can we put multiple decorators on a single property in LWC?**

No, you cannot put more than one decorator on a single property in LWC. The LWC framework does not allow this because it would create confusion for the execution engine, which wouldn't know which decorator to apply first, potentially leading to performance issues.

// Incorrect usage - will cause an error

@api @track propertyName; // You cannot combine @api and @track

Think of decorators as instructions for a kitchen appliance. If you try to use both "bake" and "grill" modes at the same time on the same oven, the appliance won't know which mode to follow, leading to incorrect cooking results.

85. Can a child component change a property passed from a parent component?

No, properties passed from a parent component to a child component via @api are read-only. To modify it, the child component must clone the property and then make changes to the clone.

Example: The main chef (parent component) sets a recipe (@api property). The assistant chef (child component) can't change the original recipe directly but can create a copy to modify for experiments.

// Child Component

@api parentProperty; // Read-only in child component

// Cloning to modify

let clonedProperty = { ...this.parentProperty };

clonedProperty.someField = 'newValue';

86. Can we use the @wire decorator inside a method?

No, the @wire decorator cannot be used inside methods. It must be used at the class property level or as a method decorator, not inside other methods.

Code Snippet:

```
// Correct usage
@wire(getRecord, { recordId: '$recordId' })
wiredRecord;


// Incorrect - inside a method
someMethod() {
    @wire(getRecord, { recordId: '$recordId' }) // This is incorrect
    wiredRecord;
}
```

87. When does a @wire property get its value?

Answer: A @wire property gets its value after the component's constructor runs and before the connectedCallback lifecycle hook. If data isn't immediately available, the property might initially be undefined and will update when the data is fetched.

This is like setting up ingredients in the kitchen (constructor) before service starts (connectedCallback). If some ingredients aren't available yet, they start with what's available and update as new ingredients arrive.

Code Snippet:

javascript

```
// @wire property gets value after constructor
@wire(getRecord, { recordId: '$recordId' })
```

wiredRecord;

88.**What is the difference between event.stopPropagation() and event.preventDefault()?**

event.preventDefault() stops the default action associated with an event, like preventing a form submission. event.stopPropagation() prevents the event from bubbling up to parent elements, stopping other handlers from executing.

```
// Prevent default form submission
handleSubmit(event) {
    event.preventDefault();
}

// Stop event bubbling
handleClick(event) {
    event.stopPropagation();
}
```

89. **What is the purpose of the export keyword in LWC classes?**

The export keyword is used to make the properties and methods in a class available for import into other modules, allowing the class to be reused across different components.

```
// Exporting a class for reuse
export default class MyComponent extends LightningElement {
    // Class content
}
```

7. What is a callback function in JavaScript?

Answer: A callback function is a function passed as an argument to another function, which can be executed later when needed.

Restaurant Scenario: This is like the chef telling the waiter to notify them when the order is ready (callback function), so they can prepare the meal accordingly.

Code Snippet:

javascript

Copy code

```
function processOrder(callback) {
    // Process the order
    callback(); // Notify when done
}


processOrder(() => {
    console.log('Order is processed');
});
```

90. **What is Promise.all and how does it work?**

Promise.all is a method that takes an array of promises and returns a single promise that resolves when all the promises have resolved. It rejects if any of the promises are rejected.

Promise.all is like waiting for all dishes in an order to be ready before serving them to the customer, ensuring that everything is served together.

```
Promise.all([promise1, promise2]).then(results => {
    console.log(results);
});
```

91. What is Shadow DOM in LWC?

Shadow DOM provides encapsulation for a component's styles and markup, ensuring they do not interfere with the styles and markup of the rest of the document.

Shadow DOM is like having a private dining area in a restaurant where the decor and atmosphere (styles and markup) are different from the main dining room, ensuring a unique experience that doesn't affect the rest of the restaurant.

Code Snippet:

javascript

```
<template>
    <p class="shadow-p">Styled independently</p>
</template>


<style>
    .shadow-p {
        color: red; /* Only affects this component */
```

```
    }
</style>
```

92. Can LWC components be used in record-triggered flows?

No, LWC components are not currently supported in record-triggered flows. They can be used in screen flows, where user interaction is involved.

93. How does @track differ from @wire in LWC?

@track is used to make a property reactive so that changes to its value automatically update the DOM. @wire is used to fetch data from Salesforce services or Apex methods, automatically providing data to the component and updating it when the data changes.

93. Give code snippet of using LWC components inside Aura components?

LWC components can be used inside Aura components by wrapping them in an Aura component.

```
<!-- Aura component wrapping LWC -->
<aura:component>
    <c:myLWCComponent />
</aura:component>
```

94. What tools do you use for deployment in Salesforce, and how do they integrate with LWC?

Tools like GitHub and Copado are used for version control and deployment. In GitHub, developers pull the latest code, commit changes, and push updates. Copado helps manage the deployment pipeline.

95. What are some best practices in LWC development?

1. **Component Reusability:** Write reusable components, such as creating a separate component for displaying toast messages.
2. **Use Salesforce Lightning Design System (SLDS):** For consistent UI design.
3. **Lazy Loading:** Load components or data only when necessary.
4. **Security Practices:** Use Salesforce Locker Service to prevent vulnerabilities like cross-site scripting.

96. What are the different cases used in LWC?

- **Pascal Case:** Used for naming classes (e.g., ReservationForm).
- **Camel Case:** Used for naming properties and methods (e.g., menuItems).
- **Kebab Case:** Used for naming components in HTML (e.g., <c-menu-item>).

```
// Pascal Case for class name
export default class ReservationForm extends LightningElement {
    // Camel Case for properties and methods
    @track menuItems = [];
}

<!-- Kebab Case for component tag -->
<c-menu-item></c-menu-item>
```

97.**What files are included in an LWC component bundle?**

An LWC component bundle typically includes:

- **HTML File:** For markup.

- **JavaScript File:** For logic and handling events.
- **XML Configuration File:** To define metadata like where the component can be used.
- **Optional CSS File:** For styling.
- **Optional SVG File:** For custom icons.

98.What are the limitations of using Shadow DOM in LWC?

Shadow DOM encapsulates component styling and markup, which can sometimes lead to limitations, such as:

- Styles not penetrating the shadow boundary.
- Events needing to be explicitly passed across the boundary.

99. What is refreshApex and when is it used?

refreshApex is used to refresh the cached data provided by an Apex method in a wire service. It is necessary when data may have been changed by an operation, such as a DML action.

Example: After updating the menu on the website backend, refreshApex would be used to refresh the displayed menu items, ensuring that customers see the latest offerings.

Code Snippet:

javascript

```
import { refreshApex } from '@salesforce/apex';


@wire(getMenuItems) menuItems;

handleMenuUpdate() {
    // Perform some update action
    refreshApex(this.menuItems); // Refresh the cached data
```

}

100. What are the different data binding types in LWC compared to Aura?

In Aura, data binding is typically two-way, meaning changes in the data model reflect in the UI and vice versa without extra code. In LWC, data binding is one-way by default, making it more predictable and easier to debug.

101. **What are styling hooks in LWC, and why are they important?**

Styling hooks allow developers to override the default styles of SLDS components to match their branding. They are important for maintaining brand consistency across custom components.

```
:host {
    --slds-c-button-brand-color-background: #ff6600; /* Customize button color */
}
```

102. **What is the purpose of the XML configuration file in an LWC bundle?**

Answer: The XML configuration file in an LWC component bundle defines the component's metadata, including where it can be used (e.g., in an app page, record page, or utility bar) and any design attributes available for customization in the Lightning App Builder.

Code Snippet:

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
```

```
    <apiVersion>53.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
    </targets>
</LightningComponentBundle>
```

103. What does the <isExposed> tag in the XML configuration file signify?

The <isExposed> tag indicates whether the LWC component is exposed to the Lightning App Builder. If set to true, the component can be added to Lightning pages, such as record pages, app pages, and home pages.

Code Snippet:

```
<isExposed>true</isExposed>
```

104. How do you specify where an LWC component can be used in the XML configuration file?

The <targets> and <target> tags define where the component can be used within Salesforce, such as on record pages, app pages, home pages, and in utilities like the utility bar. The <targetConfigs> tag can further specify details for each target, like which objects the component can be associated with.

Code Snippet:

```
<targets>
    <target>lightning__RecordPage</target>
```

```
    <target>lightning__AppPage</target>
    <target>lightning__HomePage</target>
    <target>lightning__UtilityBar</target>
</targets>
```

105. What is the <targetConfig> tag used for in the XML configuration file?

The <targetConfig> tag specifies additional configuration details for each target, such as limiting the component's usage to specific Salesforce objects or setting design attributes that can be configured in the Lightning App Builder.

Code Snippet:

```
<targetConfigs>
    <targetConfig targets="lightning__RecordPage">
        <objects>
            <object>Account</object>
            <object>Contact</object>
        </objects>
    </targetConfig>
</targetConfigs>
```

106. How can you control the visibility of an LWC component in the Lightning App Builder?

Visibility can be controlled using the <isExposed> tag and through <targetConfigs> by specifying criteria such as user profiles, permissions, or specific conditions that must be met for the component to be displayed.

Code Snippet:

xml

```
<targetConfigs>
    <targetConfig targets="lightning__RecordPage">
        <objects>
            <object>Account</object>
        </objects>
        <supportedFormFactors>
            <supportedFormFactor type="Small" />
            <supportedFormFactor type="Large" />
        </supportedFormFactors>
    </targetConfig>
</targetConfigs>
```

107. Explain Lazy Loading in LWC?

Imagine a restaurant website where the homepage features various sections, such as daily specials, customer reviews, and a reservation form. Initially, the website loads with basic information and images. As the user scrolls down or interacts with specific sections, additional data or components are loaded.

Lazy loading is a design pattern used to defer the loading of an object until it is actually needed, which can improve performance and reduce initial load times.

Lazy loading in LWC can be achieved through various techniques, including conditional rendering, event listeners, and using the Intersection Observer API.

Conditional Rendering

In this approach, parts of the component's template are rendered based on certain conditions, such as user actions or state changes.

Intersection Observer API

The Intersection Observer API can be used to load components or data when an element becomes visible in the viewport.

108. What are design attributes in LWC, and how are they configured in the XML file?

Design attributes are customizable properties of an LWC component that can be set in the Lightning App Builder. They allow users to configure component behavior or appearance without changing the code, defined under the <design> tag in the XML configuration file.

Code Snippet:

```
<design>
    <attribute name="backgroundColor" label="Background Color" type="String" />
    <attribute name="title" label="Section Title" type="String" />
</design>
```

109. Can LWC components be created in the Salesforce Developer Console?

No, LWC components cannot be created in the Salesforce Developer Console. Instead, developers use tools like Visual Studio Code or the Code Builder (a web-based IDE currently in beta) to develop and manage LWC components.

110. What are the key properties defined in the XML configuration file of an LWC component?

- **apiVersion**: Specifies the Salesforce API version.

- **isExposed**: Determines if the component is available in the Lightning App Builder.
- **targets**: Lists the areas in Salesforce where the component can be used, such as record pages, app pages, or community pages.
- **targetConfigs**: Provides detailed settings for each target, including object-specific configurations and design attributes.

111. How are SVG files used in LWC, and what is their significance in the component bundle?

Answer: SVG files in LWC are used to define custom icons for the component. These icons appear alongside the component's name in the App Builder or when the component is used in the UI. The SVG file provides a scalable graphic that enhances the component's visual identity.

112. What is Lightning Locker Service, and why is it important?

Lightning Locker Service is a security framework that isolates Lightning components, ensuring that each component's data and DOM are protected from others. It prevents common security vulnerabilities like cross-site scripting (XSS) and ensures secure execution of JavaScript code.

Imagine a large, busy restaurant with several specialized sections: the kitchen, the bar, and the front desk (for reservations and payments). Each section needs to operate independently but also securely within the same environment, ensuring that one section's actions or data do not adversely affect another.

113. How does Locker Service ensure DOM isolation between components?

Locker Service enforces DOM isolation by creating separate namespaces for components, preventing them from accessing each other's DOM elements.

Code Snippet Example:

javascript

```
// Component A cannot access Component B's DOM elements directly
// Component A
<template>
    <div class="component-a">Component A</div>
</template>

// Component B
<template>
    <div class="component-b">Component B</div>
</template>
```

In this example, Component A and Component B cannot manipulate each other's DOM elements directly due to Locker Service's isolation.

114. What are some specific restrictions imposed by Locker Service on JavaScript global objects?

Locker Service restricts access to global objects like window, document, and their properties. For example, direct access to window.document is restricted to prevent potential security risks.

Code Snippet Example:

```
// Allowed: Access to the component's own DOM
this.template.querySelector('div').innerHTML = 'Safe';

// Not Allowed: Direct access to the global document
window.document.querySelector('div').innerHTML = 'Not allowed under Locker Service';
```

115. Can Lightning components communicate with each other under Locker Service? If so, how?

Yes, Lightning components can communicate with each other using public properties (@api), custom events, or Lightning Message Service (LMS).

116. How does Lightning Message Service (LMS) support communication across different Salesforce technologies?

Answer: LMS provides a unified messaging mechanism that allows communication across Lightning Web Components (LWC), Aura components, and Visualforce pages. It uses a publish-subscribe pattern, where messages are published to a channel and any component subscribed to that channel can receive the messages. This enables different parts of a Salesforce application, which might be built using different technologies, to interact seamlessly.

117. What is a message channel in LMS, and how is it defined?

A message channel in LMS is a named resource that defines a communication pathway. It is used to publish and subscribe to messages. A message channel must be created in Salesforce and can be referenced in the code to enable messaging between components.

Think of a message channel like a specific intercom line used by different restaurant sections (like the kitchen or delivery service) to communicate specific types of information (e.g., order statuses).

118**. How can a component unsubscribe from a message channel in LMS?**

A component can unsubscribe from a message channel by calling the unsubscribe function, passing in the subscription object returned when the component initially subscribed to the channel.

```
import { unsubscribe } from 'lightning/messageService';

    disconnectedCallback() {
```

```
        unsubscribe(this.subscription);
        this.subscription = null;
    }
}
```

119. What are the key differences between using LMS and traditional event handling in Salesforce?

Answer: LMS is designed for cross-component and cross-framework communication, allowing LWCs, Aura components, and Visualforce pages to communicate seamlessly. Traditional event handling, such as using custom events, is typically limited to components within the same hierarchy or framework.

120. When should you use LMS over traditional parent-child communication methods in Salesforce?

LMS should be used when the components that need to communicate are not in a direct parent-child relationship, especially if they are different types of components (LWC, Aura, Visualforce) or if they reside on different parts of a Lightning page.

121. What is Lightning Message Service (LMS) and how does it differ from Pub/Sub?

Imagine a large restaurant chain that has multiple systems, like the kitchen display, order-taking system, and delivery tracking. These systems are built

using different technologies. For example, the kitchen display might be a traditional web application, the order-taking system might be an advanced web component, and the delivery tracking might be on a Visualforce page.

LMS acts like a central dispatcher for messages. If an order status changes (e.g., an order is ready for pickup), the LMS can notify all systems, regardless of their technology. So, the kitchen can update the display, the waitstaff can see the status change on their order-taking system, and the delivery tracking system updates for the delivery staff.

In a Pub/Sub model, the system is much more informal and decentralized. Messages are sent out, and any interested listener can receive them. The sender (chef) doesn't need to know who the receivers are (waiters).

Pub/Sub is great for simple, direct messaging where the sender and receiver don't need to be tightly integrated. It's easier to implement for basic needs, but lacks the cross-platform capabilities of LMS.