Interview Questions for - 

By Shubham Shendre | Gradx Academy | Contact - 8767401160

---

1. For:each vs Iterator Code

In Salesforce Lightning Web Components (LWC), the `for:each` directive and iterators are used to render lists of items in templates.

`for:each` Directive: This is a simple way to iterate over an array in the template. It's straightforward and suitable for most use cases.
Example:

```
<template for:each={contacts} for:item="contact">
  <p key={contact.Id}>{contact.Name}</p>
</template>
```

Iterator: The iterator provides more control by exposing the first and last items in the list, which can be useful for adding separators or specific styling.
Example:

```
<template iterator:contact={contacts}>
 <p key={contact.value.Id}>
  {contact.value.Name}
  <template if:true={contact.first}> (First Contact)</template>
  <template if:true={contact.last}> (Last Contact)</template>
 </p>
</template>
```

2. Bubble and Capture Phase Code Example

In JavaScript, event propagation has two main phases: capturing and bubbling.

- **Capturing Phase**: The event starts from the root and travels down to the target element.
- **Bubbling Phase**: After reaching the target, the event bubbles up back to the root.

In LWC, you can control event propagation using the `composed`, `bubbles`, and `cancelable` properties when dispatching events.

Example:

```
const event = new CustomEvent('myevent', {
```

```
  bubbles: true, // Event will bubble up
  composed: true // Event can cross the shadow DOM boundary
});
this.dispatchEvent(event);
```

3. Implement LMS in Example Code

Lightning Message Service (LMS) allows communication between components, including those in separate namespaces.

Example:

Message Channel Definition (`messageChannel.xml`):

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningMessageChannel
xmlns="http://soap.sforce.com/2006/04/metadata">
  <description>Sample message channel</description>
  <isExposed>true</isExposed>
  <lightningMessageFields>
    <fieldName>message</fieldName>
  </lightningMessageFields>
</LightningMessageChannel>
```

Publisher Component:

```
import { LightningElement } from 'lwc';
import { publish, MessageContext } from 'lightning/messageService';
import SAMPLEMC from
'@salesforce/messageChannel/SampleMessageChannel__c';

export default class Publisher extends LightningElement {
  @wire(MessageContext)
  messageContext;

  handleClick() {
    const message = {
```

```
      message: 'Hello, World!'
    };
    publish(this.messageContext, SAMPLEMC, message);
  }
}
```

Subscriber Component:

```
import { LightningElement, wire } from 'lwc';
import { subscribe, MessageContext } from 'lightning/messageService';
import SAMPLEMC from
'@salesforce/messageChannel/SampleMessageChannel__c';

export default class Subscriber extends LightningElement {
  messageReceived;

  @wire(MessageContext)
  messageContext;

  connectedCallback() {
    this.subscription = subscribe(
      this.messageContext,
      SAMPLEMC,
      (message) => this.handleMessage(message)
    );
  }

  handleMessage(message) {
    this.messageReceived = message.message;
  }
}
```

4. What is Lightning Data Service?

Lightning Data Service (LDS) is a standard controller in Lightning Components that provides access to Salesforce data and metadata, handles sharing rules, field-level security, and manages data caching. It simplifies data access and improves performance by reducing the need for Apex controllers.

5. How to Bypass Validation Rules in Salesforce

To bypass validation rules, you can create a custom checkbox field, e.g., `Bypass_Validation__c`, and modify your validation rules to exclude records where this checkbox is checked.

Example:

```
AND(
  NOT($User.Bypass_Validation__c),
  /* Existing validation conditions */
)
```

This allows users with the `Bypass_Validation__c` checkbox selected to bypass the validation rules.

6. In What Context is Manual Sharing Enabled in Salesforce?

Manual sharing is available for records in Salesforce when the object's organization-wide default sharing setting is set to either "Private" or "Public Read Only." This feature allows users to grant specific access to individual records beyond the predefined sharing rules.

7. Can We Delete Records Created by Someone in the Same Role?

The ability to delete records is determined by the user's profile permissions and the sharing model. If a user's profile has the "Delete" permission for the object, they can delete records they own. However, deleting records owned by others, even within the same role, depends on sharing settings and whether the user has the "Modify All" or "Modify All Data" permission.

8. What is User Mode and System Mode?

- **User Mode**: In this mode, code respects the user's permissions, field-level security, and sharing rules. Standard controllers and anonymous blocks run in user mode.
- **System Mode**: Code executed in system mode ignores user permissions and field-level security. However, sharing rules can be enforced if specified. Apex classes and triggers run in system mode by default.

---

9. What are Decorators?

In LWC, decorators are functions that add functionality to properties or functions. Common decorators include:

- `@api`: Exposes a property or method as public, making it accessible to parent components.
- `@track`: Tracks changes to a property's value for reactivity.
- `@wire`: Links a property or method to a wire adapter's data.

10. How Many Ways Can We Share Records?

Records in Salesforce can be shared using the following methods:

1. **Organization-Wide Defaults (OWD)**: Set baseline access at the organization level.
2. **Role Hierarchies**: Share records up the hierarchy automatically.
3. **Sharing Rules**: Share records with specific roles, public groups, or users.
4. **Manual Sharing**: Share individual records with users or groups (enabled for OWD Private or Public Read Only).
5. **Apex Sharing**: Programmatically share records using Apex.
6. **Team Sharing**: Assign access to Account, Opportunity, or Case teams.
7. **Territory Management**: Share records based on territories in Enterprise Territory Management.
8. **Salesforce Managed Sharing**: System-controlled sharing, like ownership-based sharing.

11. Can We Call a Batch Class from a Future Method, and If Not, Why?

No, you cannot call a batch class from a future method because both are asynchronous operations, and Salesforce does not allow one asynchronous job to initiate another. This limitation prevents overloading the system with chained asynchronous processes.

12. Can We Make Callouts from Triggers?

No, callouts are not allowed directly from triggers because triggers are synchronous, whereas callouts are asynchronous operations. To work around this, you can use a future method or Queueable Apex to perform the callout.

Example using Queueable Apex:

```
public class CalloutExample implements Queueable {
    public void execute(QueueableContext context) {
        HttpRequest request = new HttpRequest();
        HttpResponse response = new HttpResponse();
        Http http = new Http();
        request.setEndpoint('https://api.example.com');
        request.setMethod('GET');
        response = http.send(request);
    }
}
```

13. Can We Implement Both Scheduled Apex and Batch Apex in a Single Class?

Yes, you can implement both `Schedulable` and `Database.Batchable` interfaces in the same class. However, you must define separate methods for each implementation.

Example:

```
public class CombinedApexClass implements Schedulable,
Database.Batchable<sObject> {
    public void execute(SchedulableContext context) {
        Database.executeBatch(this);
    }

    public Database.QueryLocator start(Database.BatchableContext
context) {
        return Database.getQueryLocator('SELECT Id FROM Account');
    }

    public void execute(Database.BatchableContext context,
List<sObject> scope) {
        // Batch logic
    }

    public void finish(Database.BatchableContext context) {
        // Final actions
```

```
    }
}
```

14. Can We Make Callouts from Batch Apex?

Yes, callouts can be made from Batch Apex, but you need to use the `Database.AllowsCallouts` interface. Each batch execution allows one callout per record scope.

Example:

```
public class CalloutBatch implements Database.Batchable<sObject>, Database.AllowsCallouts {
    public Database.QueryLocator start(Database.BatchableContext context) {
        return Database.getQueryLocator('SELECT Id FROM Account');
    }

    public void execute(Database.BatchableContext context, List<Account> scope) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://api.example.com');
        request.setMethod('GET');
        HttpResponse response = http.send(request);
    }

    public void finish(Database.BatchableContext context) {
        // Final actions
    }
}
```

15. What Are Lifecycle Hooks in LWC?

Lifecycle hooks in Lightning Web Components (LWC) are callback methods triggered during the component's lifecycle. They help developers execute custom logic at specific stages. The common lifecycle hooks include:

1. **`constructor()`**
   Called when the component is created but before it is added to the DOM.
   Use Case: Initialize state or bind methods.

   ```
   constructor() {
       super();
       console.log('Component created');
   }
   ```

2. **`connectedCallback()`**
   Invoked when the component is inserted into the DOM.
   Use Case: Fetch data or interact with the DOM.

   ```
   connectedCallback() {
       console.log('Component inserted into DOM');
   }
   ```

3. **`renderedCallback()`**
   Called after the component has been rendered and the DOM updated.
   Use Case: Manipulate the DOM or perform post-render actions.

   ```
   renderedCallback() {
       console.log('Component rendered');
   }
   ```

4. **`disconnectedCallback()`**
   Triggered when the component is removed from the DOM.
   Use Case: Cleanup tasks, like removing event listeners.

   ```
   disconnectedCallback() {
       console.log('Component removed from DOM');
   ```

```
}
```

`errorCallback(error, stack)`
Invoked when an error occurs in the component or its child components.
Use Case: Handle errors gracefully.
javascript
Copy code

```
errorCallback(error, stack) {
    console.error(error, stack);
}
```

16. Which Executes First: Decorators or Lifecycle Hooks?

Decorators are processed at compile time before the component lifecycle begins. Lifecycle hooks are invoked during runtime as the component progresses through its lifecycle stages.

17. Advantages and Limitations of Lightning Data Service (LDS)

- **Advantages**:
  - Simplifies CRUD operations without Apex.
  - Automatically respects sharing rules, FLS, and record security.
  - Reduces boilerplate code, improving maintainability.
  - Improves performance by caching data.
- **Limitations**:
  - Works only with Salesforce Standard and Custom objects.
  - Limited to operations supported by the standard `force:recordData` component.
  - Cannot perform complex business logic like Apex controllers.

18. Account and Contact Phone Match Validation Scenario

Validation Rule to ensure the phone number on the Contact matches the related Account:

```
AND(
   NOT(ISBLANK(Account.Phone)),
   NOT(ISBLANK(Phone)),
   Account.Phone <> Phone
```

```
)
```

19. What is the Relationship Between Account and Contact?

The relationship between Account and Contact is a standard **lookup relationship** where:

- A Contact must be associated with one Account (required lookup).
- Accounts can have multiple Contacts (1-to-Many).

20. Roll-Up Summary for Contact from Account—How to Implement It?

Roll-up summary fields cannot be created directly between Account and Contact as the relationship is not Master-Detail. Use one of these alternatives:

1. **Apex Trigger**: Write a trigger to update a custom field on Account when Contact data changes.
2. **Flow**: Create a record-triggered flow to calculate and update the value.
3. **AppExchange Package**: Use declarative roll-up tools like DLRS (Declarative Lookup Rollup Summaries).

21. Making Opportunity and Opportunity Products Read-Only After Closed-Won

This can be achieved using validation rules:

Opportunity Validation Rule:

```
AND(
    ISPICKVAL(StageName, 'Closed Won'),
    ISCHANGED(StageName)
)
```

Opportunity Product Validation Rule:

```
AND(
    ISPICKVAL(Opportunity.StageName, 'Closed Won'),
    ISCHANGED(Opportunity.StageName)
)
```

22. Communication Between Parent-Child and Independent Components in LWC

- **Parent-Child Communication**:
  - Use `@api` decorator to pass data or methods from parent to child.
  - Use custom events to send data from child to parent.

Example:

```
this.dispatchEvent(new CustomEvent('myevent', { detail: data }));
```

-
- **Independent Components Communication**:
  - Use **Lightning Message Service (LMS)** to facilitate communication.

23. Debug Logs Created for Batch Apex with 2000 Records and a Scope of 200

For 2000 records and a batch size of 200:

- **Number of Batches**: 2000 / 200 = 10 batches.
- **Debug Logs**: Each batch execution generates one debug log. Therefore, 10 debug logs will be created.

24. Experience with Deployment Tools

Common tools for Salesforce deployments include:

- **Change Sets** (native, limited to related orgs).
- **SFDX (Salesforce CLI)** (ideal for CI/CD pipelines).
- **ANT Migration Tool** (script-based deployment).
- **Third-party Tools**: Gearset, Copado, AutoRabit.

25. Difference Between "With Sharing" and "Without Sharing"

With Sharing: Enforces sharing rules defined in Salesforce.
Example:

```
public with sharing class SharingExample {
    // Code here respects sharing rules
}
```

-

Without Sharing: Ignores sharing rules but respects object and field-level security.
Example:

```
public without sharing class NoSharingExample {
    // Code here ignores sharing rules
```

```
}
```