How do you pass data between parent and child components in LWC?

In LWC, data can be passed between parent and child components using different approaches depending on the direction of data flow:

Parent to Child (using @api): To pass data from a parent component to a child component, we use the @api decorator on the child component. This decorator makes the property public, allowing the parent component to set the value of the property.

Example:

Parent Component (HTML):

html

Copy code

<template>

<c-child-component message="Hello, World!"></c-child-component>

</template>

Child Component (JavaScript):

javascript

Copy code

import { LightningElement, api } from 'lwc';

export default class ChildComponent extends LightningElement {

@api message;

}

Child to Parent (using Custom Events): To send data from a child component to a parent component, you can use Custom Events. The child component creates an event, dispatches it, and the parent listens for it. The event can carry data by passing it as part of the event detail.

Example:

Child Component (JavaScript):

javascript

Copy code

import { LightningElement } from 'lwc';

export default class ChildComponent extends LightningElement {

handleClick() {

const event = new CustomEvent('messagechange', {

detail: { message: 'Data from child' }

});

this.dispatchEvent(event);

}

}

Parent Component (HTML and JavaScript):

html

Copy code

<template>

<c-child-component onmessagechange={handleMessageChange}></c-child-component>

<p>{message}</p>

</template>

javascript

Copy code

import { LightningElement } from 'lwc';

export default class ParentComponent extends LightningElement {

message = '';

handleMessageChange(event) {

this.message = event.detail.message;

}

}

2. Explain the use of @api for parent-to-child communication and @track or @wire for child-to-parent.

@api: The @api decorator is used for parent-to-child communication. It makes a property or method public and allows the parent component to pass data to the child component.

Usage: It allows the parent component to pass data to the child component by setting the value of a property marked with @api.

@track: The @track decorator is used to make a property reactive so that the component re-renders when its value changes. It’s typically used for internal component state, especially when dealing with complex objects or arrays.

Example:

javascript

Copy code

import { LightningElement, track } from 'lwc';

export default class ExampleComponent extends LightningElement {

@track items = ['item1', 'item2', 'item3']; // Any change triggers reactivity

}

@wire: The @wire decorator is used to fetch data from Salesforce or other external data sources. It automatically updates the component when data changes.

Example:

javascript

Copy code

import { wire } from 'lwc';

import getRecords from '@salesforce/apex/RecordController.getRecords';

export default class RecordListComponent extends LightningElement {

@wire(getRecords) records;

}

3. How do you handle events in LWC?

In LWC, events are used to communicate between components. There are two types of events:

Custom Events (Child to Parent): When a child component needs to send data or signal an action to its parent component, it dispatches a custom event.

To create a custom event, use the CustomEvent constructor and dispatch it using dispatchEvent.

The parent listens for the event and can retrieve data sent with the event.

Example (Child to Parent communication):

javascript

Copy code

import { LightningElement } from 'lwc';

export default class ChildComponent extends LightningElement {

handleClick() {

const event = new CustomEvent('datachange', {

detail: { message: 'Hello from child!' }

});

this.dispatchEvent(event);

}

}

Example (Parent component listens for event):

html

Copy code

<template>

<c-child-component ondatachange={handleDataChange}></c-child-component>

</template>

<script>

import { LightningElement } from 'lwc';

export default class ParentComponent extends LightningElement {

handleDataChange(event) {

console.log(event.detail.message); // Logs 'Hello from child!'

}

}

</script>

Standard Events: LWC also supports standard browser events such as click, mouseover, etc., and you can handle them similarly using event listeners.