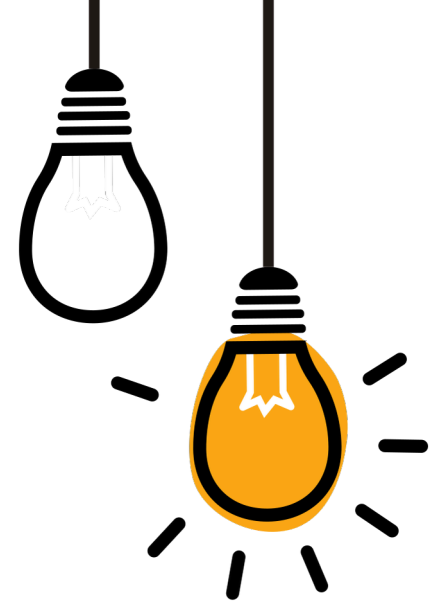
TRAILHEAD IQ



CALLING APEX METHOD USING @WIRE IN LIGHTNING WEB COMPONENTS

@TrailheadIQ

Swipe next →

What is @wire in LWC?

- @wire is a powerful decorator in Lightning Web Components (LWC) that enables seamless integration with Salesforce data and services.

- It allows you to declaratively call Apex methods, Lightning Data Service (LDS), or listen to platform events, all while handling the complexities of data flow between the client and server for you.

- The decorator automatically manages data binding, ensuring that your component stays in sync with Salesforce data without requiring manual intervention.

Swipe next ⟶

@TrailheadIQ

Why Use @wire to Call Apex Methods?

- **Automatic Reactivity:** @wire re-invokes methods whenever reactive properties change, keeping your UI in sync with backend data.

- **Simplified Code:** It eliminates the need for manual async handling, making your code cleaner.

- **Error Handling:** @wire manages both success and error states for you.

- **Performance:** With @AuraEnabled (cacheable= true), it improves performance by leveraging caching to reduce server calls.

@TrailheadIQ

Swipe next ⟶

Steps to Call Apex Methods Using @wire

Expose the Apex Method:

- The method must be static.
- Use the @AuraEnabled(cacheable=true) decorator for methods that only retrieve data.

Use the @wire Decorator in LWC:

- Import the Apex method using @salesforce/apex.
- Decorate the property or function with @wire to bind the method call.

Swipe next ⟶

@TrailheadIQ

Creating Apex Method Example

- This Apex method retrieves a list of accounts.
- It's cacheable since it only reads data and doesn’t modify the database.

Apex

```apex
public with sharing class AccountController {

    @AuraEnabled(cacheable=true)
    public static List<Account> getAllAccounts() {

        return [SELECT Id, Name, Industry FROM Account LIMIT 10];

    }
}
```

Swipe next ⟶

@TrailheadIQ

Importing and Using @wire in LWC

Import the Apex Method in JavaScript and Use @wire Decorator to Bind Apex Call

JS

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

export default class AccountList extends LightningElement {

    @wire(getAllAccounts) accounts;

    get hasAccounts() {

        return this.accounts && this.accounts.data;
    }
}
```

- The @wire decorator binds the result of the Apex method to the accounts property.
- The get hasAccounts() getter checks if data is available.

@TrailheadIQ

Swipe next ⟶

Displaying the Data in HTML

To display the accounts in your template

```html
<template if:true={hasAccounts}>
    <ul>
        <template for:each={accounts.data} for:item="account">
            <li key={account.Id}>{account.Name} - {account.Industry}</li>
        </template>
    </ul>
</template>
<template if:true={accounts.error}>
    <p>Error loading accounts: {accounts.error}</p>
</template>
```

Explanation:

- if:true={hasAccounts} ensures the list is only rendered when data is available.
- The for:each directive loops through the list of accounts to display them.

Swipe next →

@TrailheadIQ

Using Parameters in @wire Decorated Methods

You can pass parameters to Apex methods when using @wire.

Apex Method with Parameters:

```apex
@AuraEnabled(cacheable=true)
public static List<Account> getAccountsByIndustry(String industry) {

    return [SELECT Id, Name FROM Account WHERE Industry = :industry];

}
```

Swipe next ⟶

@TrailheadIQ

Use @wire with Parameters

Js

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

/** The delay used when debouncing event handlers before invoking Apex. */
const DELAY = 300;

export default class FilteredAccounts extends LightningElement {
    industry = 'Technology';

   handleKeyChange(event) {

        window.clearTimeout(this.delayTimeout);
        const industry = event.target.value;
        // eslint-disable-next-line @lwc/lwc/no-async-operation
        this.delayTimeout = setTimeout(() => {
            this.industry = industry;
        }, DELAY);
    }

    @wire(getAccountsByIndustry, { industry: '$industry' }) filteredAccounts;

    get hasFilteredAccounts() {

        return this.filteredAccounts && this.filteredAccounts.data;
    }
}
```

The $industry denotes a reactive property. When industry changes, the @wire method is automatically re-invoked.

@TrailheadIQ

Swipe next ⟶

Key Benefits of Using @wire

Reactive Data Fetching:

- Automatically updates the data when reactive parameters change.

Simplified Syntax:

- Removes the need for manually invoking Apex methods using then() and catch() for basic use cases.

Improved Performance:

- Automatically handles caching of results when using cacheable=true.

Swipe next →

@TrailheadIQ

Key Takeaways

- The @wire decorator is a powerful tool to bind Apex methods to your LWC components.

- Use @AuraEnabled(cacheable=true) in Apex for read-only methods.

- You can pass parameters to the wire service and handle reactivity effortlessly.

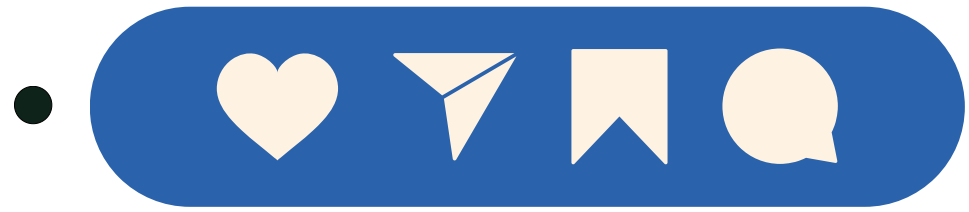
- Error handling in @wire is essential to improve the user experience.

Swipe next ⟶

@TrailheadIQ

THANK YOU

Remember, every like, share, and comment helps us reach more people and make a bigger impact. Together, we can make the Salesforce community stronger and more informed.#TrailheadIQ#SalesforceCommunity



www.trailheadiq.com

TRAILHEAD IQ