**Work with Salesforce data (Connect to Salesforce DB)**

Below are the different ways to connect to SFDC DB

* Lightning Data Service (LDS)
* Wire Services (Refer last section of this doc i. e. more on wire services)
* Call Apex Methods
* Wire a property
* Wire a function
* Call a method imperatively

\*\*\* First try to use LDS. If requires something which can’t be done through LDS then go for Wire Services.   
Lastly, if both ways can’t be achieved then think of Apex methods (here as well try to use wire decorator and avoid imperative calls unless needed).

**LDS (Lightning Data Service)**

| **FEATURE** | **LIGHTNING-RECORD-FORM** | **LIGHTNING-RECORD-VIEW-FORM** | **LIGHTNING-RECORD-EDIT-FORM** |
| --- | --- | --- | --- |
| Create Records | Yes |  | Yes |
| Edit Records | Yes |  | Yes |
| View Records | Yes | Yes |  |
| Read-Only Mode | Yes | Yes |  |
| Layout Types | Yes |  |  |
| Multi Column Layout | Yes |  |  |
| Custom Layout for Fields |  | Yes | Yes |
| Custom Rendering of Record Data |  | Yes | Yes |

Load form –

*<!-- myComponent.html -->*

<template>

<**lightning-record-form**

record-id={recordId}

object-api-name="Account"

layout-type="Compact"

mode="view">

</lightning-record-form>

</template>

*// myComponent.js*

import { LightningElement, api } from 'lwc';

export default class MyComponent extends LightningElement {

@api recordId;

}

Modes can be view or read-only

<template>

<**lightning-record-view-form**

record-id={recordId}

object-api-name="Account">

<div class="slds-grid">

<div class="slds-col slds-size\_1-of-2">

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

<lightning-output-field field-name="Phone"></lightning-output-field>

</div>

<div class="slds-col slds-size\_1-of-2">

<lightning-output-field field-name="Industry"></lightning-output-field>

<lightning-output-field field-name="AnnualRevenue"></lightning-output-field>

</div>

</div>

</lightning-record-view-form>

</template>

import { LightningElement, api } from 'lwc';

export default class MyComponent extends LightningElement{

*// Expose a recordId property.*

@api recordId;

}

Above are sample codes for few tags. Go through dev guide for more samples.

**Calling Apex methods from LWC..**

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

Three ways to call apex method from LWC

1. Wire a property – result will be come in data or error attribute
2. Wire a function – result will be come in data or error attribute
3. Call a method imperatively

**Syntax to Wire a property or a function**

import apexMethod from '@salesforce/apex/Namespace.Classname.apexMethod';

@wire(apexMethod, *{* apexMethodParams *}*)

propertyOrFunction;

**Examples –**

**Wire a property**

// ContactController.cls

public with sharing class ContactController {

@AuraEnabled(cacheable=true)

public static List<Contact> getContactList() {

return [SELECT Id, Name, Title, Phone, Email FROM Contact WHERE Email != null LIMIT 10];

}

}

import { LightningElement, wire } from 'lwc';

import getContactList from '@salesforce/apex/ContactController.getContactList';

export default class ApexWireMethodToProperty extends LightningElement {

@wire(getContactList) contacts;

}

<template>

<lightning-card title="ApexWireMethodToProperty" icon-name="custom:custom63">

<div class="slds-m-around\_medium">

<template if:true={contacts.data}>

<template for:each={contacts.data} for:item="contact">

<p key={contact.Id}>{contact.Name}</p>

</template>

</template>

<template if:true={contacts.error}>

<c-error-panel errors={contacts.error}></c-error-panel>

</template>

</div>

</lightning-card>

</template>

============================================================================================================

// ContactController.cls

public with sharing class ContactController {

@AuraEnabled(cacheable=true)

public static List<Contact> findContacts(String searchKey) {

String key = '%' + searchKey + '%';

return [SELECT Id, Name, Title, Phone, Email FROM Contact WHERE Name LIKE :key AND Email != null LIMIT 10];

}

}

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

import findContacts from '@salesforce/apex/ContactController.findContacts';

*/\*\* The delay used when debouncing event handlers before invoking Apex. \*/*

const DELAY = 300;

export default class ApexWireMethodWithParams extends LightningElement {

@track searchKey = '';

**@wire(findContacts, { searchKey: '$searchKey' })**

contacts;

handleKeyChange(event) {

*// Debouncing this method: Do not update the reactive property as long as this function is*

*// being called within a delay of DELAY. This is to avoid a very large number of Apex method calls.*

window.clearTimeout(this.delayTimeout);

const searchKey = event.target.value;

this.delayTimeout = setTimeout(() => {

this.searchKey = searchKey;

}, DELAY);

}

}

<template>

<lightning-card title="ApexWireMethodWithParams" icon-name="custom:custom63">

<div class="slds-m-around\_medium">

<lightning-input type="search" onchange={handleKeyChange} class="slds-m-bottom\_small" label="Search" **value={searchKey}**></lightning-input>

<template if:true={contacts.data}>

<template for:each={contacts.data} for:item="contact">

<p key={contact.Id}>{contact.Name}</p>

</template>

</template>

<template if:true={contacts.error}>

<c-error-panel errors={contacts.error}></c-error-panel>

</template>

</div>

</lightning-card>

</template>

==============================================================================

**Wire a function**

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

import getContactList from '@salesforce/apex/ContactController.getContactList';

export default class ApexWireMethodToFunction extends LightningElement {

@track contacts;

@track error;

@wire(getContactList)

wiredContacts({ error, data }) {

if (data) {

this.contacts = data;

this.error = undefined;

} else if (error) {

this.error = error;

this.contacts = undefined;

}

}

}

<template>

<lightning-card title="ApexWireMethodToFunction" icon-name="custom:custom63">

<div class="slds-m-around\_medium">

<template if:true={contacts}>

<template for:each={contacts} for:item="contact">

<p key={contact.Id}>{contact.Name}</p>

</template>

</template>

<template if:true={error}>

<c-error-panel errors={error}></c-error-panel>

</template>

</div>

</lightning-card>

</template>

**Call an Apex Method Imperatively**

<template>

<lightning-card title="ApexImperativeMethod" icon-name="custom:custom63">

<div class="slds-m-around\_medium">

<p class="slds-m-bottom\_small">

<lightning-button label="Load Contacts" onclick={handleLoad}></lightning-button>

</p>

<template if:true={contacts}>

<template for:each={contacts} for:item="contact">

<p key={contact.Id}>{contact.Name}</p>

</template>

</template>

<template if:true={error}>

<c-error-panel errors={error}></c-error-panel>

</template>

</div>

</lightning-card>

</template>

import { LightningElement, track } from 'lwc';

import getContactList from '@salesforce/apex/ContactController.getContactList';

export default class ApexImperativeMethod extends LightningElement {

@track contacts;

@track error;

handleLoad() {

getContactList()

.then(result => {

this.contacts = result;

})

.catch(error => {

this.error = error;

});

}

}

LWC => errorPanel component

<template>

<div class="slds-p-around\_medium">

<lightning-icon icon-name="utility:error"></lightning-icon>

<div class="slds-p-around\_small">

<p>{friendlyMessage}</p>

<template if:true={errorMessages.length}>

<div class="slds-p-vertical\_x-small">

<lightning-input

label="Show Details"

type="checkbox"

onchange={handleCheckboxChange}

></lightning-input>

</div>

<template if:true={viewDetails}>

<template for:each={errorMessages} for:item="message">

<p key={message} class="error-message">{message}</p>

</template>

</template>

</template>

</div>

</div>

</template>

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

import { reduceErrors } from 'c/ldsUtils';

export default class ErrorPanel extends LightningElement {

/\*\* Generic / user-friendly message \*/

@api friendlyMessage = 'Error retrieving data';

@track viewDetails = false;

/\*\* Single or array of LDS errors \*/

@api errors;

get errorMessages() {

return reduceErrors(this.errors);

}

handleCheckboxChange(event) {

this.viewDetails = event.target.checked;

}

}

:host > div {

text-align: center;

}

---------------------

**LDSUtils.js**

/\*\*

\* Reduces one or more LDS errors into a string[] of error messages.

\* @param {FetchResponse|FetchResponse[]} errors

\* @return {String[]} Error messages

\*/

export function reduceErrors(errors) {

if (!Array.isArray(errors)) {

errors = [errors];

}

return (

errors

// Remove null/undefined items

.filter(error => !!error)

// Extract an error message

.map(error => {

// UI API read errors

if (Array.isArray(error.body)) {

return error.body.map(e => e.message);

}

// UI API DML, Apex and network errors

else if (error.body && typeof error.body.message === 'string') {

return error.body.message;

}

// JS errors

else if (typeof error.message === 'string') {

return error.message;

}

// Unknown error shape so try HTTP status text

return error.statusText;

})

// Flatten

.reduce((prev, curr) => prev.concat(curr), [])

// Remove empty strings

.filter(message => !!message)

);

}

|  |  |
| --- | --- |
| **Wired** | **Imperative call** |
| Less code and less maintenance. Hence should be used over imperative. | More code and more maintenance. Hence should be avoided. |
| Need Cached methods (cacheable=true) | Can work on cached or non-cached methods. |
| Can’t be used when data is to be changed as methods are cached. | Can be used even if received data is to be changed if method is (cacheable=false) |
| If variable values are getting changed then wired service take care of refreshed data. Do not need to explicitly handle anything. | If variable values are getting changed then code must be written to handle it and get data basis on new variable values. |
| Wire services will be called directly and can’t be called through custom way. To do so extra logic needs to be implemented. | Can be called as per need. |
|  |  |

Don't overload @AuraEnabled Apex methods as any method can be called randomly.

**More on Wire services –**

Wire services and base components,

<template>

<lightning-card

title="Display Contact Address"

icon-name="standard:contact">

<template if:true={contact.data}>

<div class="slds-m-around\_medium">

<lightning-formatted-address

street={street}

city={city}

country={country}

province={state}

postal-code={postal}

show-static-map

></lightning-formatted-address>

</div>

</template>

</lightning-card>

</template>

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

import { getRecord } from 'lightning/uiRecordApi';

import STREET\_FIELD from '@salesforce/schema/Contact.MailingStreet';

import CITY\_FIELD from '@salesforce/schema/Contact.MailingCity';

import STATE\_FIELD from '@salesforce/schema/Contact.MailingState';

import COUNTRY\_FIELD from '@salesforce/schema/Contact.MailingCountry';

import POSTAL\_FIELD from '@salesforce/schema/Contact.MailingPostalCode';

const FIELDS = [STREET\_FIELD, CITY\_FIELD, STATE\_FIELD, COUNTRY\_FIELD, POSTAL\_FIELD];

export default class GetContactAddress extends LightningElement {

@api recordId; *// provided by the contact record page*

@wire(getRecord, { recordId: '$recordId', fields: FIELDS })

contact;

get street() {

return this.contact.data.fields.MailingStreet.value;

}

get city() {

return this.contact.data.fields.MailingCity.value;

}

get state() {

return this.contact.data.fields.MailingState.value;

}

get country() {

return this.contact.data.fields.MailingCountry.value;

}

get postal() {

return this.contact.data.fields.MailingPostalCode.value;

}

}

Create Records with wire services,

<template>

<lightning-card title="LdsCreateRecord" icon-name="standard:record">

<div class="slds-m-around\_medium">

<lightning-input label="Id" disabled value={accountId}></lightning-input>

<lightning-input label="Name" onchange={handleNameChange} class="slds-m-bottom\_x-small"></lightning-input>

<lightning-button label="Create Account" variant="brand" onclick={createAccount}></lightning-button>

</div>

</lightning-card>

</template>

import { LightningElement, track } from 'lwc';

import { createRecord } from 'lightning/uiRecordApi';

import { ShowToastEvent } from 'lightning/platformShowToastEvent';

import ACCOUNT\_OBJECT from '@salesforce/schema/Account';

import NAME\_FIELD from '@salesforce/schema/Account.Name';

export default class LdsCreateRecord extends LightningElement {

@track accountId;

name = '';

handleNameChange(event) {

this.accountId = undefined;

this.name = event.target.value;

}

createAccount() {

const fields = {};

fields[NAME\_FIELD.fieldApiName] = this.name;

const recordInput = { apiName: ACCOUNT\_OBJECT.objectApiName, fields };

createRecord(recordInput)

.then(account => {

this.accountId = account.id;

this.dispatchEvent(

new ShowToastEvent({

title: 'Success',

message: 'Account created',

variant: 'success',

}),

);

})

.catch(error => {

this.dispatchEvent(

new ShowToastEvent({

title: 'Error creating record',

message: error.body.message,

variant: 'error',

}),

);

});

}

}

**Below are the wire adaptors (lightning/ui\*Api)**

1. lightning/uiListApi
2. lightning/uiObjectInfoApi
3. lightning/uiRecordApi

**Lightning Data Table –**

<template>

<lightning-card title = "Search Accounts" icon-name = "custom:custom10">

<div class = "slds-m-around\_medium">

<lightning-input type="search" onchange={findAccountResult} class = "slds-m-bottom\_small" label = "Search"> </lightning-input>

<lightning-datatable key-field="Id" data={accountList} columns={columnList} hide-checkbox-column="true" show-row-number-column="true">

</lightning-datatable>

<template if:true= {noRecordsFound}>

--No Account Records Found--

</template>

</div>

</lightning-card>

</template>

import { LightningElement, track } from 'lwc';

import searchAccounts from '@salesforce/apex/AccountController.searchAccount';

const columnList = [

{label: 'Id', fieldName: 'Id'},

{label: 'Name', fieldName: 'Name'},

{label: 'Website', fieldName: 'Website'},

{label: 'Industry', fieldName: 'Industry'}

];

export default class LightningDataTable extends LightningElement {

@track accountList;

@track columnList = columnList;

@track noRecordsFound = true;

findAccountResult(event) {

const accName = event.target.value;

if(accName) {

searchAccounts ( {accName})

.then(result => {

this.accountList = result;

this.noRecordsFound = false;

})

} else {

this.accountList = undefined;

this.noRecordsFound = true;

}

}

}

public with sharing class AccountController {

@AuraEnabled (cacheable = true)

public static List<Account> searchAccount(String accName) {

string strAccName = '%'+ accName + '%';

return [Select Id, Name, Website, Industry, Phone from Account WHERE Name LIKE: strAccName ];

}

}

==========================

public with sharing class GetAllOpportunities {

@AuraEnabled(cacheable=true)

public static List<Opportunity> getAllOpps() {

return [SELECT Id, Name ,StageName, CloseDate ,Type ,Probability,Account.Name from Opportunity];

}

}

<template>

<h2> Data table Example -1</h2>

<lightning-datatable data={data} columns={columns} key-field="Id">

</lightning-datatable>

</template>

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

import getAllOpps from '@salesforce/apex/GetAllOpportunities.getAllOpps';

export default class DatatableEx12 extends LightningElement {

@track columns = [{

label: 'Opportunity name',

fieldName: 'Name',

type: 'text',

sortable: true

},

{

label: 'Stage Name',

fieldName: 'StageName',

type: 'text',

sortable: true

},

{

label: 'Close date',

fieldName: 'CloseDate',

type: 'date',

sortable: true

}

];

@track error;

@track data ;

@wire(getAllOpps)

wiredOpps({ error, data }) {

if (data) {

this.data = data;

} else if (error) {

this.error = error;

}

}

}

**Wrapper Examples..**

<template>

<lightning-card title="LWC Wrapper Component">

<template if:true={wrappers.data}>

<template for:each={wrappers.data} for:item="wrap">

<p key={wrap.con.Id}>{wrap.con.Name}</p>

</template>

</template>

</lightning-card>

</template>

import { LightningElement, wire } from 'lwc';

import getWrapperList from '@salesforce/apex/ContactController.getWrapperList';

export default class WrapperComponent extends LightningElement {

@wire(getWrapperList) wrappers;

}

public with sharing class ContactController {

@AuraEnabled(cacheable=true)

public static list<Wrapper> getWrapperList() {

list<Wrapper> wrapperList = new List< Wrapper >();

for(Contact con : [SELECT Id, Name, Title, Phone, Email FROM Contact Where Email != null]){

Wrapper obj = new Wrapper (false, con);

wrapperList.add(obj);

}

return wrapperList;

}

}

public class Wrapper{

@AuraEnabled

public Contact con{get;set;}

@AuraEnabled

public boolean selected{get;set;}

public wrapp(boolean selected, Contact con){

this.con = con;

this.selected = selected;

}

}