

# Salesforce Developer Interview Kit - v2

## Lightning Web Components & Integration Patterns

\*For developers with 8+ years experience returning to development roles\*

---

## Table of Contents

1. [Lightning Web Components (LWC)](#lightning-web-components-lwc)
2. [Integration & Web Services](#integration--web-services)
3. [Advanced Integration Patterns](#advanced-integration-patterns)
4. [Platform Events & Messaging](#platform-events--messaging)

---

## Lightning Web Components (LWC)

#### Q9: Create an LWC component that displays Account records with search functionality.

\*\*Answer:\*\*

\*\*accountSearch.html:\*\*

```html

<template>

<lightning-card title="Account Search" icon-name="standard:account">

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

<!-- Search Input -->

<lightning-input

type="search"

label="Search Accounts"

value={searchTerm}

onchange={handleSearchChange}

placeholder="Enter account name..."

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

</lightning-input>

<!-- Loading Spinner -->

<div if:true={isLoading} class="slds-text-align\_center slds-p-around\_medium">

<lightning-spinner alternative-text="Loading..." size="small"></lightning-spinner>

</div>

<!-- Error Message -->

<template if:true={error}>

<div class="slds-notify slds-notify\_alert slds-theme\_alert-texture slds-theme\_error">

<span class="slds-assistive-text">Error</span>

{error}

```

        </div>
    </template>

    <!-- Results Table -->
    <template if:true={accounts}>
        <lightning-datatable
            key-field="Id"
            data={accounts}
            columns={columns}
            onrowaction={handleRowAction}
            hide-checkbox-column="true">
        </lightning-datatable>
    </template>

    <!-- No Results Message -->
    <template if:true={showNoResults}>
        <div class="slds-text-align_center slds-p-around_medium">
            <p>No accounts found matching your search criteria.</p>
        </div>
    </template>
</div>
</lightning-card>
</template>
...

**accountSearch.js:**
```javascript
import { LightningElement, track, wire } from 'lwc';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';
import { NavigationMixin } from 'lightning/navigation';
import searchAccounts from '@salesforce/apex/AccountController.searchAccounts';

const COLUMNS = [
    { label: 'Account Name', fieldName: 'Name', type: 'text', sortable: true },
    { label: 'Phone', fieldName: 'Phone', type: 'phone' },
    { label: 'Website', fieldName: 'Website', type: 'url' },
    { label: 'Annual Revenue', fieldName: 'AnnualRevenue', type: 'currency' },
    {
        type: 'action',
        typeAttributes: {
            rowActions: [
                { label: 'View', name: 'view' },
                { label: 'Edit', name: 'edit' }
            ]
        }
    }
]

```

```
    }  
  }  
];
```

```
export default class AccountSearch extends NavigationMixin(LightningElement) {  
  @track accounts = [];  
  @track error;  
  @track isLoading = false;  
  @track searchTerm = "";  
  
  columns = COLUMNS;  
  delayTimeout;  
  
  get showNoResults() {  
    return !this.isLoading && this.accounts.length === 0 && this.searchTerm.length > 0;  
  }  
  
  handleSearchChange(event) {  
    this.searchTerm = event.target.value;  
  
    // Debounce search  
    clearTimeout(this.delayTimeout);  
    this.delayTimeout = setTimeout(() => {  
      this.performSearch();  
    }, 300);  
  }  
  
  performSearch() {  
    if (this.searchTerm.length < 2) {  
      this.accounts = [];  
      this.error = undefined;  
      return;  
    }  
  
    this.isLoading = true;  
    this.error = undefined;  
  
    searchAccounts({ searchTerm: this.searchTerm })  
      .then(result => {  
        this.accounts = result;  
        this.isLoading = false;  
      })  
      .catch(error => {  
        this.error = error.body ? error.body.message : error.message;  
      });  
  }  
}
```

```

        this.accounts = [];
        this.isLoading = false;
        this.showToast('Error', 'Error searching accounts: ' + this.error, 'error');
    });
}

handleRowAction(event) {
    const actionName = event.detail.action.name;
    const row = event.detail.row;

    switch (actionName) {
        case 'view':
            this.navigateToRecord(row.Id);
            break;
        case 'edit':
            this.navigateToRecord(row.Id, 'edit');
            break;
        default:
            break;
    }
}

navigateToRecord(recordId, actionName = 'view') {
    this[NavigationMixin.Navigate]({
        type: 'standard__recordPage',
        attributes: {
            recordId: recordId,
            actionName: actionName
        }
    });
}

showToast(title, message, variant) {
    const event = new ShowToastEvent({
        title: title,
        message: message,
        variant: variant
    });
    this.dispatchEvent(event);
}
...

```

**\*\*accountSearch.js-meta.xml:\*\***

```

``xml
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
  <apiVersion>60.0</apiVersion>
  <isExposed>true</isExposed>
  <targets>
    <target>lightning__RecordPage</target>
    <target>lightning__AppPage</target>
    <target>lightning__HomePage</target>
  </targets>
</LightningComponentBundle>

```

**\*\*AccountController.cls (Apex Controller):\*\***

```

``apex
public with sharing class AccountController {

    @AuraEnabled(cacheable=true)
    public static List<Account> searchAccounts(String searchTerm) {
        try {
            String wildcardSearch = '%' + searchTerm + '%';

            return [
                SELECT Id, Name, Phone, Website, AnnualRevenue, Industry
                FROM Account
                WHERE Name LIKE :wildcardSearch
                ORDER BY Name ASC
                LIMIT 50
            ];

        } catch (Exception e) {
            throw new AuraHandledException('Error searching accounts: ' + e.getMessage());
        }
    }
}

```

**### Q10: Create a parent-child LWC communication example.**

**\*\*Answer:\*\***

**\*\*Parent Component (contactManager.html):\*\***

```

``html
<template>
  <lightning-card title="Contact Manager">

```

```

        <div class="slds-m-around_medium">
            <c-contact-form oncontactsaved={handleContactSaved}></c-contact-form>
            <c-contact-list contacts={contacts}
oncontactselected={handleContactSelected}></c-contact-list>
        </div>
    </lightning-card>
</template>
...

```

```

**contactManager.js:**
```javascript
import { LightningElement, track } from 'lwc';
import getContacts from '@salesforce/apex/ContactController.getContacts';

export default class ContactManager extends LightningElement {
    @track contacts = [];

    connectedCallback() {
        this.loadContacts();
    }

    loadContacts() {
        getContacts()
            .then(result => {
                this.contacts = result;
            })
            .catch(error => {
                console.error('Error loading contacts:', error);
            });
    }

    handleContactSaved(event) {
        // Refresh contact list when new contact is saved
        this.loadContacts();

        // Show success message
        const newContact = event.detail;
        console.log('New contact saved:', newContact);
    }

    handleContactSelected(event) {
        const selectedContact = event.detail;
        console.log('Contact selected:', selectedContact);
        // Handle contact selection logic
    }
}

```

```
}  
}  
...
```

**\*\*Child Component (contactForm.html):\*\***

```
```html  
<template>  
  <lightning-card title="Add New Contact" class="slds-m-bottom_medium">  
    <div class="slds-m-around_medium">  
      <lightning-record-edit-form  
        object-api-name="Contact"  
        onsuccess={handleSuccess}  
        onerror={handleError}>  
  
        <lightning-input-field field-name="FirstName"></lightning-input-field>  
        <lightning-input-field field-name="LastName"></lightning-input-field>  
        <lightning-input-field field-name="Email"></lightning-input-field>  
        <lightning-input-field field-name="Phone"></lightning-input-field>  
        <lightning-input-field field-name="AccountId"></lightning-input-field>  
  
        <div class="slds-m-top_medium">  
          <lightning-button  
            class="slds-m-right_small"  
            variant="brand"  
            type="submit"  
            label="Save Contact">  
          </lightning-button>  
          <lightning-button  
            variant="neutral"  
            label="Cancel"  
            onclick={handleCancel}>  
          </lightning-button>  
        </div>  
      </lightning-record-edit-form>  
    </div>  
  </lightning-card>  
</template>  
```
```

**\*\*contactForm.js:\*\***

```
```javascript  
import { LightningElement } from 'lwc';  
import { ShowToastEvent } from 'lightning/platformShowToastEvent';
```

```

export default class ContactForm extends LightningElement {

  handleSuccess(event) {
    const savedContact = event.detail;

    // Show success toast
    this.showToast('Success', 'Contact saved successfully!', 'success');

    // Dispatch custom event to parent
    const contactSavedEvent = new CustomEvent('contactsaved', {
      detail: savedContact
    });
    this.dispatchEvent(contactSavedEvent);

    // Reset form
    this.template.querySelector('lightning-record-edit-form').reset();
  }

  handleError(event) {
    const errorMessage = event.detail.detail || 'Unknown error occurred';
    this.showToast('Error', errorMessage, 'error');
  }

  handleCancel() {
    // Reset form
    this.template.querySelector('lightning-record-edit-form').reset();
  }

  showToast(title, message, variant) {
    const event = new ShowToastEvent({
      title: title,
      message: message,
      variant: variant
    });
    this.dispatchEvent(event);
  }
}
...

```

---

## ## Integration & Web Services

### Q11: Create a REST API endpoint in Apex for external system integration.



**\*\*Answer:\*\***

```apex

@RestResource(urlMapping='/api/account/\*')  
global with sharing class AccountRestService {

@HttpGet

```
global static AccountResponse getAccount() {  
    try {  
        RestRequest req = RestContext.request;  
        String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/') + 1);  
  
        Account acc = [  
            SELECT Id, Name, Phone, Website, BillingAddress, AnnualRevenue,  
                (SELECT Id, FirstName, LastName, Email FROM Contacts LIMIT 10)  
            FROM Account  
            WHERE Id = :accountId  
            LIMIT 1  
        ];  
  
        return new AccountResponse(true, 'Success', acc);  
  
    } catch (QueryException e) {  
        return new AccountResponse(false, 'Account not found', null);  
    } catch (Exception e) {  
        return new AccountResponse(false, 'Internal server error: ' + e.getMessage(), null);  
    }  
}
```

@HttpPost

```
global static AccountResponse createAccount(AccountRequest accountData) {  
    try {  
        Account newAccount = new Account(  
            Name = accountData.name,  
            Phone = accountData.phone,  
            Website = accountData.website,  
            AnnualRevenue = accountData.annualRevenue,  
            Industry = accountData.industry  
        );  
  
        insert newAccount;  
  
        // Retrieve the created account with all fields  
        Account createdAccount = [  
            SELECT Id, Name, Phone, Website, AnnualRevenue, Industry, CreatedDate
```

```

        FROM Account
        WHERE Id = :newAccount.Id
    ];

    return new AccountResponse(true, 'Account created successfully', createdAccount);

} catch (DmlException e) {
    return new AccountResponse(false, 'Failed to create account: ' + e.getDmlMessage(0),
null);
} catch (Exception e) {
    return new AccountResponse(false, 'Internal server error: ' + e.getMessage(), null);
}
}

@HttpPut
global static AccountResponse updateAccount() {
    try {
        RestRequest req = RestContext.request;
        String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/') + 1);
        String requestBody = req.requestBody.toString();

        AccountRequest accountData = (AccountRequest) JSON.deserialize(requestBody,
AccountRequest.class);

        Account accountToUpdate = [SELECT Id FROM Account WHERE Id = :accountId LIMIT
1];

        if (String.isNotBlank(accountData.name)) {
            accountToUpdate.Name = accountData.name;
        }
        if (String.isNotBlank(accountData.phone)) {
            accountToUpdate.Phone = accountData.phone;
        }
        if (String.isNotBlank(accountData.website)) {
            accountToUpdate.Website = accountData.website;
        }
        if (accountData.annualRevenue != null) {
            accountToUpdate.AnnualRevenue = accountData.annualRevenue;
        }

        update accountToUpdate;

        Account updatedAccount = [
            SELECT Id, Name, Phone, Website, AnnualRevenue, LastModifiedDate

```

```

        FROM Account
        WHERE Id = :accountId
    ];

    return new AccountResponse(true, 'Account updated successfully', updatedAccount);

} catch (QueryException e) {
    return new AccountResponse(false, 'Account not found', null);
} catch (Exception e) {
    return new AccountResponse(false, 'Internal server error: ' + e.getMessage(), null);
}
}

@HttpDelete
global static AccountResponse deleteAccount() {
    try {
        RestRequest req = RestContext.request;
        String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/') + 1);

        Account accountToDelete = [SELECT Id FROM Account WHERE Id = :accountId LIMIT
1];

        delete accountToDelete;

        return new AccountResponse(true, 'Account deleted successfully', null);

    } catch (QueryException e) {
        return new AccountResponse(false, 'Account not found', null);
    } catch (Exception e) {
        return new AccountResponse(false, 'Internal server error: ' + e.getMessage(), null);
    }
}

// Wrapper classes
global class AccountRequest {
    public String name;
    public String phone;
    public String website;
    public Decimal annualRevenue;
    public String industry;
}

global class AccountResponse {
    public Boolean success;
    public String message;

```

```

    public Account data;

    public AccountResponse(Boolean success, String message, Account data) {
        this.success = success;
        this.message = message;
        this.data = data;
    }
}
}
...

```

### Q12: Implement an HTTP callout with proper error handling and retry logic.

**\*\*Answer:\*\***

**```apex**

```

public class ExternalAPIService {

    private static final String BASE_URL = 'https://api.example.com';
    private static final Integer TIMEOUT = 30000;
    private static final Integer MAX_RETRIES = 3;

    public static APIResponse makeCallout(String endpoint, String method, String requestBody) {
        return makeCalloutWithRetry(endpoint, method, requestBody, 0);
    }

    private static APIResponse makeCalloutWithRetry(String endpoint, String method, String
requestBody, Integer retryCount) {
        try {
            HttpRequest request = buildRequest(endpoint, method, requestBody);
            HttpResponse response = new Http().send(request);

            return processResponse(response);

        } catch (CalloutException e) {
            System.debug('Callout exception on attempt ' + (retryCount + 1) + ': ' + e.getMessage());

            if (retryCount < MAX_RETRIES && isRetryableError(e)) {
                // Exponential backoff delay
                Integer delay = (Integer) Math.pow(2, retryCount) * 1000;
                // In a real scenario, you'd implement actual delay mechanism
                return makeCalloutWithRetry(endpoint, method, requestBody, retryCount + 1);
            }

            return new APIResponse(false, 'Callout failed after ' + (retryCount + 1) + ' attempts: ' +
e.getMessage(), null);
        }
    }
}

```

```
    } catch (Exception e) {  
        System.debug('Unexpected error: ' + e.getMessage());  
        return new APIResponse(false, 'Unexpected error: ' + e.getMessage(), null);  
    }  
}
```

```
private static HttpRequest buildRequest(String endpoint, String method, String requestBody) {  
    HttpRequest request = new HttpRequest();  
    request.setEndpoint(BASE_URL + endpoint);  
    request.setMethod(method);  
    request.setTimeout(TIMEOUT);  
    request.setHeader('Content-Type', 'application/json');  
    request.setHeader('Authorization',
```