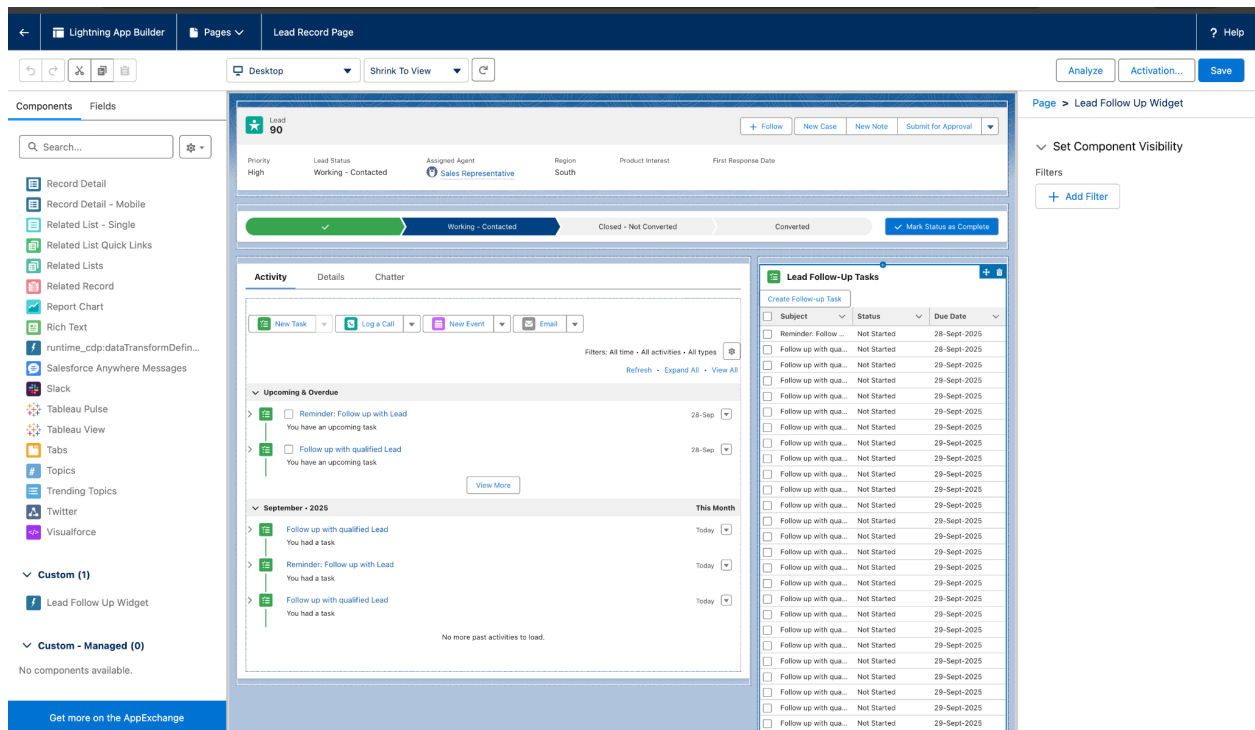


Phase 6 Report

Record Pages are custom layouts that define how specific Salesforce records appear for users in Lightning Experience. In my project, I customized the Lead Record Page to integrate a Lightning Web Component (LWC) that enables users to create and view follow-up tasks directly from the Lead page. This customization enhances the user experience by allowing quick access to key sales activities within the familiar record context.



Tabs

Tabs provide an organized navigation structure within Salesforce Lightning Apps. The Leads tab plays a central role in my project's workflow, serving as the gateway where users access all Lead records. Placing the LWC component within this context ensures that users can efficiently manage follow-up tasks related to individual leads.

[illegible]

Home Page Layouts

Home Page Layouts offer users a customizable dashboard with relevant summaries and quick links upon login. While my project didn't focus extensively on Home Page customization, incorporating related task summaries here in future phases could further boost productivity by providing immediate insights into pending follow-ups.

Sales Home Opportunities ▾ **Leads** ▾ Tasks ▾ Files ▾ Accounts ▾ Contacts ▾ Campaigns ▾ Dashboards ▾ Reports ▾ Chatter Groups ▾ Calendar ▾ People ▾ Cases ▾ More ▾

Leads
My Leads

Created This Quarter Owner Me

Total Leads	No Activity	Idle	No Upcoming	Overdue	Due Today	Upcoming
6	5	0	0	0	0	5

6 Items - Sorted by Lead Source - Filtered by Created Date, Me, Total Leads

Name	Title	Company	Lead Status	Lead Source	Last Activity	Actions
Bm pramo		The smartBrid	Working - Contacted		29/9/2025	[Email] [Phone] [Dropdown]
bm praveen		dknw	Working - Contacted			[Email] [Phone] [Dropdown]
pramod		The smartBr	Open - Not Contacted			[Email] [Phone] [Dropdown]
pramod		The smartBri	Working - Contacted	Web		[Email] [Phone] [Dropdown]
pramod		The smartBridg	Working - Contacted	Web		[Email] [Phone] [Dropdown]
pramod		The smartBridge	Working - Contacted	Phone Inquiry		[Email] [Phone] [Dropdown]

☰ To Do List

Utility Bar

The Utility Bar is a persistent panel offering quick access to common tools across all app pages. Though not part of the current scope, future enhancements might include adding follow-up task shortcuts or notifications to the Utility Bar to facilitate seamless task management.

Lightning Web Components (LWC)

Lightning Web Components are modern, reusable UI building blocks following web standards. My project developed a custom LWC, `leadFollowUpWidget`, which displays pending follow-up tasks on the Lead Record Page and enables the creation of new tasks with a single click. This component significantly streamlines the sales process by embedding task management capabilities directly into the Lead interface.

The screenshot displays the Salesforce Setup interface. The top navigation bar includes the Salesforce logo, a search bar, and utility icons. The left sidebar contains a 'Quick Find' bar and a list of setup categories: Setup Home, Salesforce Go, Service Setup Assistant, Commerce Setup Assistant, Field Service Setup Home (Beta), Hyperforce Assistant, Release Updates, Salesforce Mobile App, Lightning Usage, Optimizer, Sales Cloud Everywhere, ADMINISTRATION (Users, Data, Email), and PLATFORM TOOLS (Subscription Management, Apps, Feature Settings, Slack, Data Cloud, Heroku, MuleSoft, Einstein, Objects and Fields). The main content area is titled 'Lightning Components' and shows the 'Lightning Component Detail' for the 'leadFollowUpWidget'. The detail view includes a 'Lightning Web Component Bundle Detail' section with fields for Name, Label, Description, Created By, and Modified By. Below this is the 'LWC Dependencies' section, which contains a search bar and a table listing dependencies. The table has columns for Name, Label, Type, Namespace Prefix, and Api Version. A single dependency is listed: 'leadFollowUpWidget' with a count of 58.

Name	Label	Type	Namespace Prefix	Api Version
leadFollowUpWidget	leadFollowUpWidget	LWC		58

Apex with LWC

Apex acts as the server-side backbone, handling complex business logic and database interactions. In my project, Apex classes expose methods to create new follow-up tasks and fetch existing tasks, which the LWC consumes. The seamless integration between Apex and LWC allows for dynamic data operations while maintaining robust security and scalability.

The screenshot displays the VS Code editor interface with a project named 'LEAD-MANAGEMENT-SALES-TRACKING'. The file explorer on the left shows the project structure, including files like `LeadController.cls`, `LeadController.cls-meta.xml`, `lwc/leadFollowUpWidget`, and `leadFollowUpWidget.html`. The main editor shows the `leadFollowUpWidget.js` file, which is an LWC component. It imports the `createFollowUpTask` and `getFollowUpTasks` methods from the `LeadController` Apex class. The component defines columns for 'Subject' and 'Due Date', and implements the `getFollowUpTasks` method to fetch tasks from the server. The terminal at the bottom shows the deployment process, which was successful. The 'Deployed Source' table below summarizes the deployment details.

State	Name	Type	Path
Unchanged	LeadController	ApexClass	force-app/main/default/classes/LeadController.cls
Unchanged	LeadController	ApexClass	force-app/main/default/classes/LeadController.cls-meta.xml
Changed	leadFollowUpWidget	LightningComponentBundle	force-app/main/default/lwc/leadFollowUpWidget/leadFollowUpWidget.html
Changed	leadFollowUpWidget	LightningComponentBundle	force-app/main/default/lwc/leadFollowUpWidget/leadFollowUpWidget.js
Changed	leadFollowUpWidget	LightningComponentBundle	force-app/main/default/lwc/leadFollowUpWidget/leadFollowUpWidget.js-meta.xml

Events in LWC

Events in LWC facilitate communication and feedback within components. The project utilizes the ShowToastEvent to provide users with immediate success or error notifications upon task creation. This enhances user experience by giving clear, concise feedback on their actions

[illegible]

Wire Adapters

Wire adapters deliver reactive, declarative data-fetching capabilities from Salesforce Apex or platform data. The @wire decorator is used in my component to retrieve follow-up tasks in real-time, automatically updating the component UI whenever relevant data changes.

The screenshot shows a Visual Studio Code editor with a Salesforce project named 'Lead-Management-Sales-Tracking'. The Explorer pane on the left shows the project structure, including the 'leadFollowUpWidget.js' file. The main editor displays the code for 'leadFollowUpWidget.js', which implements a wire adapter to fetch follow-up tasks. The code includes a 'wiredTasks' property, a 'handleCreateTask' method, and a 'createFollowUpTask' method. The 'createFollowUpTask' method uses the 'createFollowUpTask' Apex method to create a new follow-up task and then refreshes the data. The 'handleCreateTask' method uses the 'createFollowUpTask' method to create a new follow-up task and then refreshes the data. The code is as follows:

```
13 export default class LeadFollowUpWidget extends LightningElement {
22   wiredTasks(result) {
23     this.error = result.error;
24     this.tasks = undefined;
25   }
26 }
27
28 handleCreateTask() {
29   createFollowUpTask({ leadId: this.recordId })
30   .then(() => {
31     // Success toast
32     new ShowToastEvent({
33       title: 'Success', Static string found: 'success'
34       message: 'Follow-up task created', Static string found: 'Follow-up task created'
35       variant: 'Success' Static string found: 'success'
36     });
37   });
38
39   // Refresh the wired data
40   if (this.wiredTasksResult) {
41     return refreshApex(this.wiredTasksResult);
42   }
43 }
44
45 .catch(error => {
46   this.dispatchEvent(
47     new ShowToastEvent({
48       title: 'Error creating task', Static string found: 'Error creating task'
49       message: error.body?.message || error.message,
50       variant: 'error'
51     });
52   });
53 }
54 }
55 }
56 }
57 }
```

The TERMINAL pane at the bottom shows the output of the 'sf project deploy start' command, indicating a successful deployment. The 'Deployed Source' table is as follows:

State	Name	Type	Path
Unchanged	LeadController	ApexClass	force-app/main/default/classes/LeadController.cls
Unchanged	LeadController	ApexClass	force-app/main/default/classes/LeadController.cls-meta.xml
Changed	leadFollowUpWidget	LightningComponentBundle	force-app/main/default/lwc/leadFollowUpWidget/leadFollowUpWidget.html
Changed	leadFollowUpWidget	LightningComponentBundle	force-app/main/default/lwc/leadFollowUpWidget/leadFollowUpWidget.js
Changed	leadFollowUpWidget	LightningComponentBundle	force-app/main/default/lwc/leadFollowUpWidget/leadFollowUpWidget.js-meta.xml

Imperative Apex Calls

Imperative Apex calls execute server-side logic programmatically based on user interaction. The task creation function in the project is implemented as an imperative call, triggered when users click the "Create Follow-up Task" button, demonstrating precise control over Apex method invocation.

The screenshot shows a Visual Studio Code editor with the following components:

- EXPLORER:** Displays the project structure for 'LEAD-MANAGEMENT-SALES-TRACKING'. The file 'leadFollowUpWidget.html' is selected.
- MAIN EDITOR:** Shows the HTML template for 'leadFollowUpWidget.html'. The code includes a Lightning button to create a follow-up task and a Lightning datatable to display the tasks. The Apex logic is embedded within the template using `<template>` and `<lightning-datatable>` tags.
- TERMINAL:** Shows the output of the deployment command `sf project deploy start --source-dir force-app`. The deployment is successful, with a status of 'Succeeded' and a deploy ID of '8Af20000000000000000000000000000'.
- Deployed Source Table:**

State	Name	Type	Path
Unchanged	LeadController	ApexClass	force-app/main/default/classes/LeadController.cls
Unchanged	LeadController	ApexClass	force-app/main/default/classes/LeadController.cls-meta.xml
Changed	leadFollowUpWidget	LightningComponentBundle	force-app/main/default/lwc/leadFollowUpWidget/leadFollowUpWidget.html
Changed	leadFollowUpWidget	LightningComponentBundle	force-app/main/default/lwc/leadFollowUpWidget/leadFollowUpWidget.js
Changed	leadFollowUpWidget	LightningComponentBundle	force-app/main/default/lwc/leadFollowUpWidget/leadFollowUpWidget.js-meta.xml

Conclusion

In this phase, the core focus was on enhancing the Lead management experience by developing an integrated solution using **Salesforce Lightning App Builder**, **Lightning Web Components (LWC)**, and **Apex**. A custom Lead Record Page was created to embed the `leadFollowUpWidget` component, enabling users to create and manage follow-up tasks efficiently within the Lead record interface.

The development leveraged the power of modern web standards through LWCs for a responsive and interactive UI, while Apex provided robust server-side logic for handling data operations securely and effectively. Using **wire adapters** and **imperative Apex calls** ensured seamless integration and real-time data synchronization between the client and server. Event handling via toast notifications enhanced user feedback during interactions.

Though challenges related to data refreshing and notifications were encountered, they reinforced important best practices in Salesforce component development, such as properly managing reactive data and careful event handling.

Overall, this phase successfully showcased the ability to customize Salesforce for business-specific workflows, improving productivity and user experience in sales pipeline management. The foundation laid here opens up avenues for further innovation, including advanced task automation, integrations, and enhanced user interfaces.