# Phase 6 – User Interface Development

## Purpose

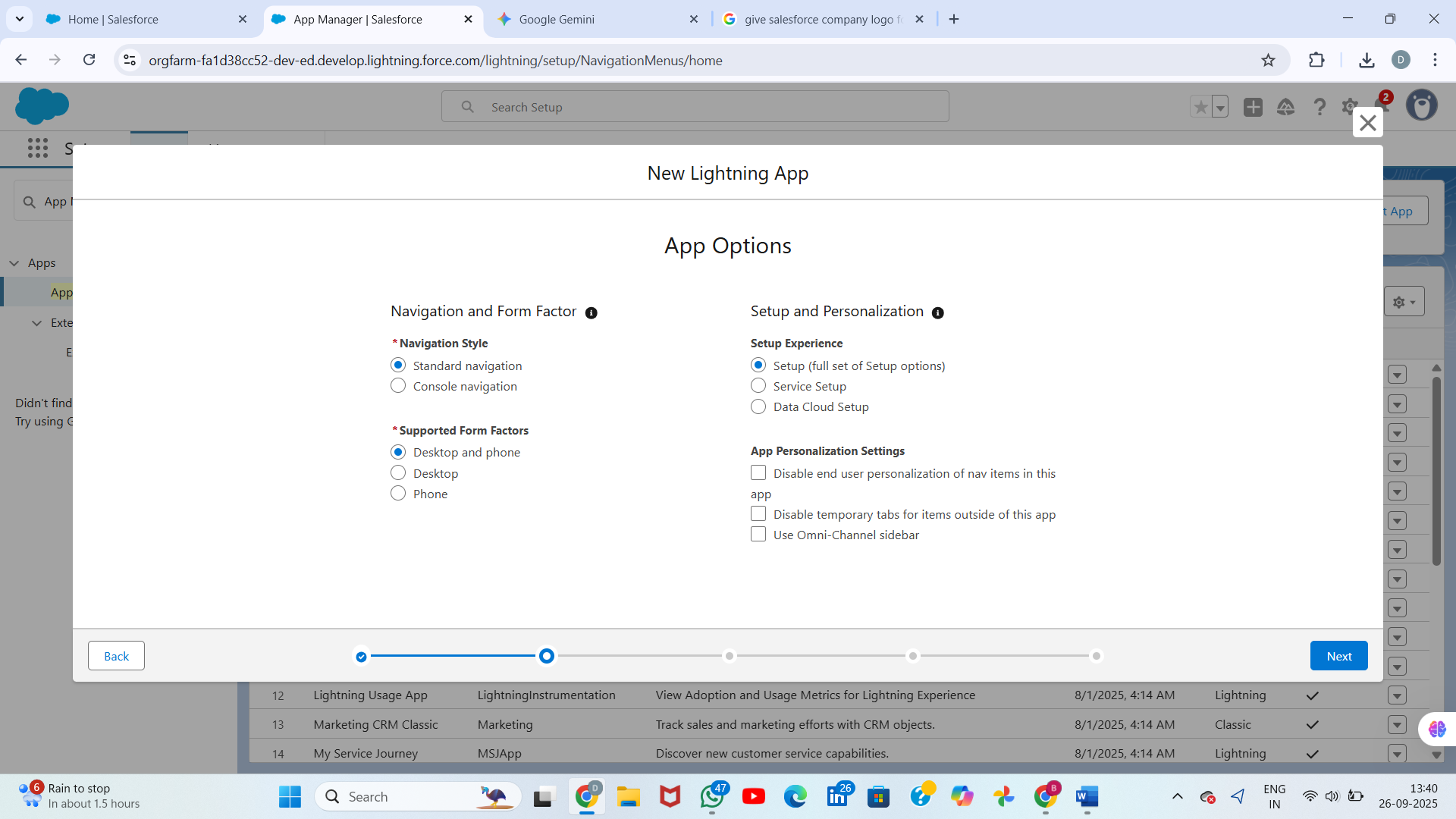
The purpose of Phase 6 is to design and implement an intuitive user interface within Salesforce that allows users to submit tasks, review AI-generated suggestions, and provide feedback. This phase bridges the backend processing with the end-user experience by leveraging both declarative Salesforce tools and programmatic customizations through Lightning Web Components (LWC) and Apex. The goal is to ensure seamless interaction, scalability, and adoption by end users.

## Overview

User Interface Development is a critical phase because it defines how users interact with the AI-enhanced system. The design must focus on accessibility, responsiveness, and simplicity while enabling powerful backend logic to surface seamlessly. Key tools include Lightning App Builder, Home Page Layouts, Record Pages, Tabs, Utility Bar, Lightning Web Components, and advanced LWC capabilities such as events, wire adapters, imperative Apex calls, and navigation service. Each step ensures a modular, interactive, and secure front-end that is consistent across different devices and user profiles..

## Step 1 – Lightning App Builder

* • Navigate to Setup → App Manager → New Lightning App.
* • Define app details (Label, Description, Primary Branding).
* • Choose supported form factors: Desktop, Phone, or Both.
* • Add Navigation Items: Task, AI Suggestions, Feedback, Reports, Dashboards.
* • Configure Utility Bar for global actions (e.g., New Task flow, Quick Access to Suggestions).

• Finalize visibility: Assign app to specific profiles.  


## 

## 

## Step 2 – Home Page Layouts

* • Navigate to Setup → Lightning App Builder → New → Home Page.
* • Select a page template (standard templates or custom).
* • Drag components: Dashboard Charts, Recent Items, or LWC components (AI Overview, Quick Feedback Panel).
* • Add custom components such as AI Insights LWC.
* • Save and Activate for specific apps or org-wide.

## Create a new Home page

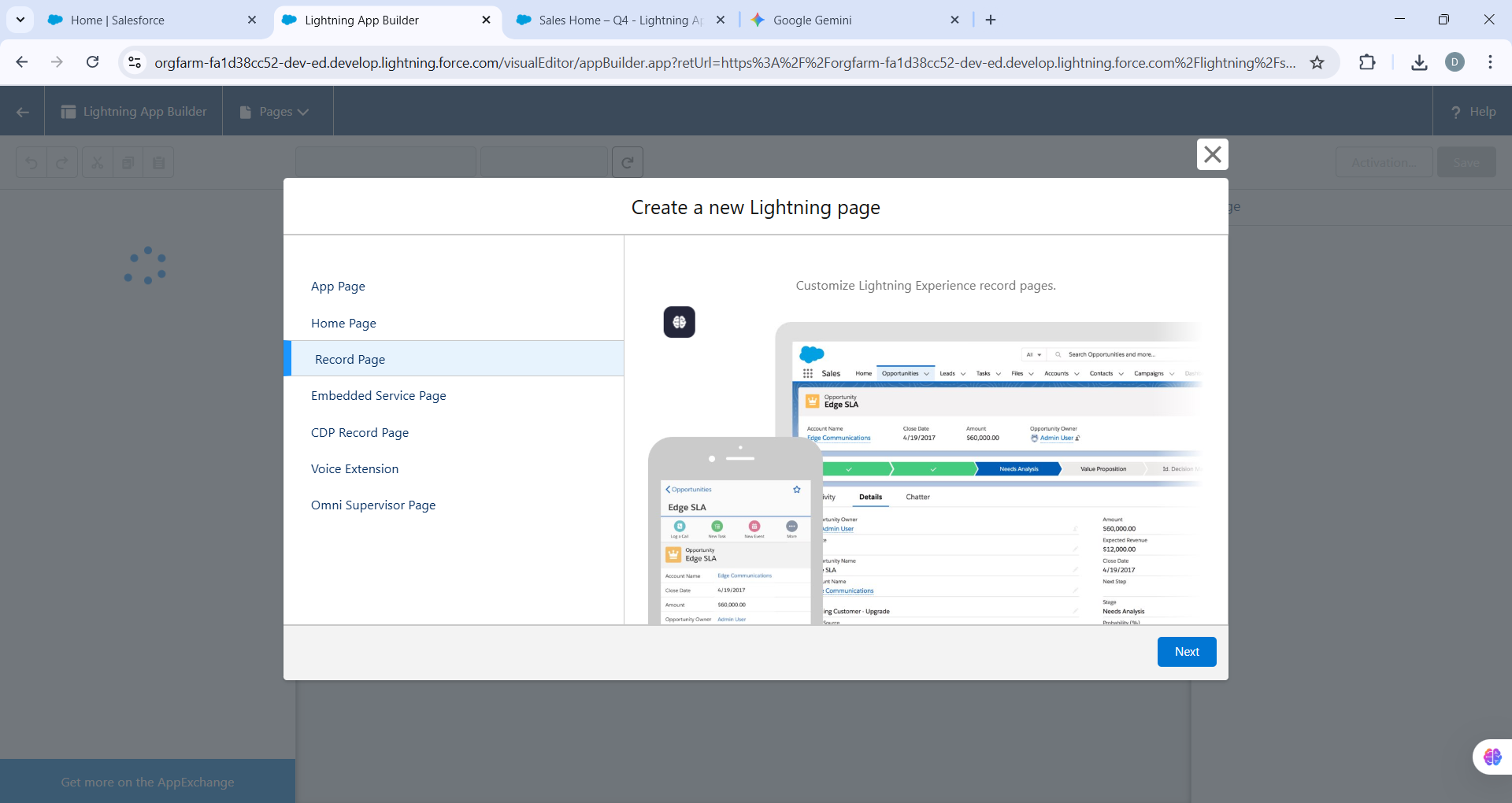
1. In **Lightning App Builder** click **New** (top right).
2. In the popup choose **Home Page** and click **Next**. [Salesforce+1](https://help.salesforce.com/s/articleView?id=platform.lightning_app_builder_create_app_page.htm&language=en_US&type=5&utm_source=chatgpt.com)
3. Give the page a **Name** (e.g., Sales Home – Q4), optionally add a description.
4. Choose a **template** (examples: Header and Left Sidebar, Header and Right Sidebar, Three Regions, or the **Clone Salesforce Default** option). Click **Finish**. Tip: Cloning the standard Home page gives you a copy that Salesforce can auto-upgrade later; creating from scratch gives you full control. [Trailhead](https://trailhead.salesforce.com/content/learn/modules/lightning_app_builder/lightning_app_builder_homepage?utm_source=chatgpt.com)

## Step 3 – Record Pages

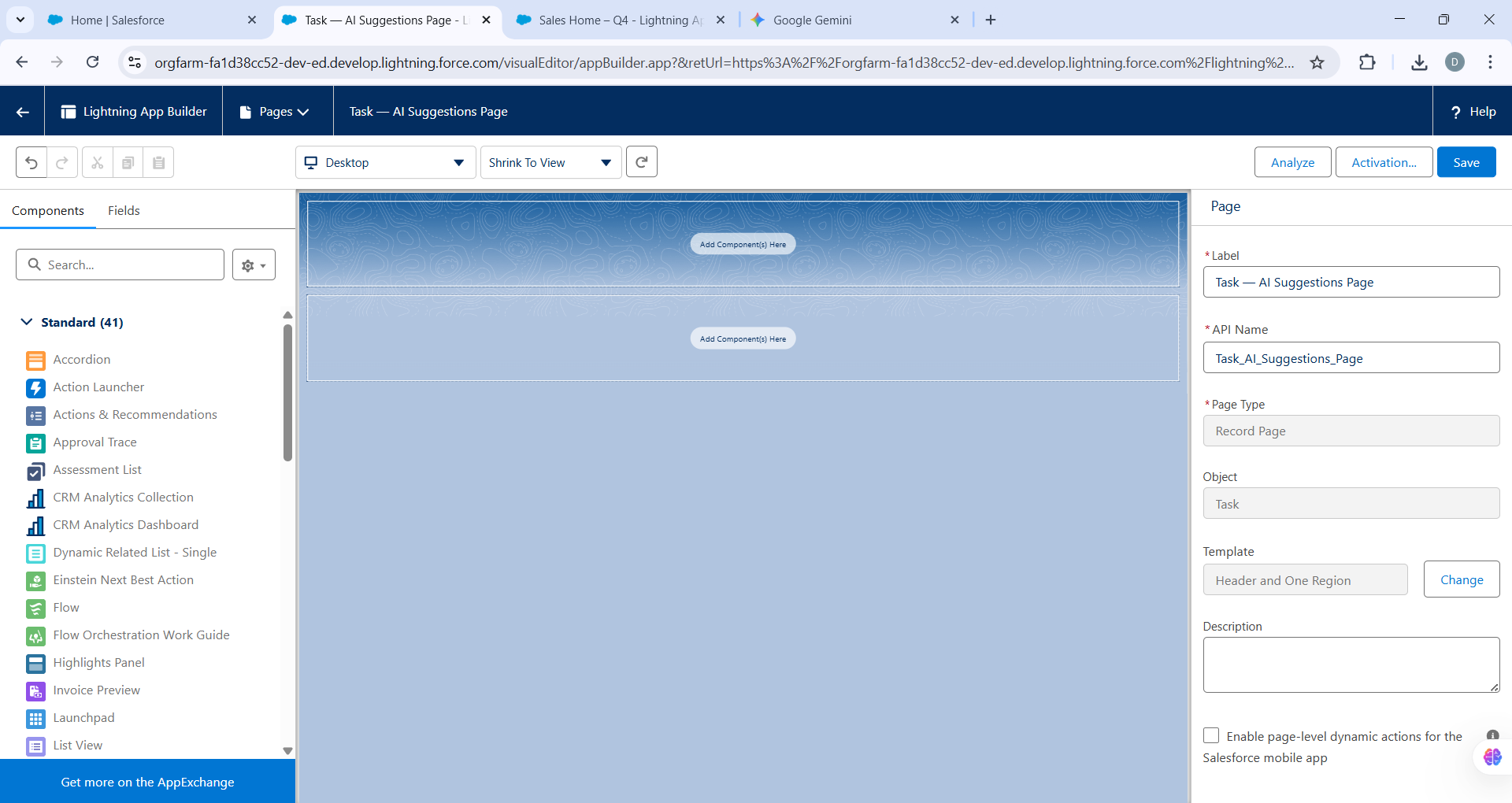
* • Navigate to Object Manager → Task → Lightning Record Pages → New.
* • Use Record Page template: Header, Tabs, and Related lists.
* • Place core details in main region, related lists below.
* • Add a custom LWC 'aiSuggestionsList' to show AI suggestions in context of a task.
* • Add a Flow component if you want to capture approvals or guided input.
* • Save, then Activate and assign for profiles or default app.

## Open ****Lightning Record Pages**** and create a new page

1. In the object menu (left column) click **Lightning Record Pages**.
2. Click **New** (top right). Choose **Record Page** when prompted and click **Next**.
3. Give the page a **Label** (e.g., Task — AI Suggestions Page) and select a **template** (Header + One Column, Two Regions, etc.), then **Finish**. The page opens in the **Lightning App Builder** canvas.

* 

Task — AI Suggestions Page

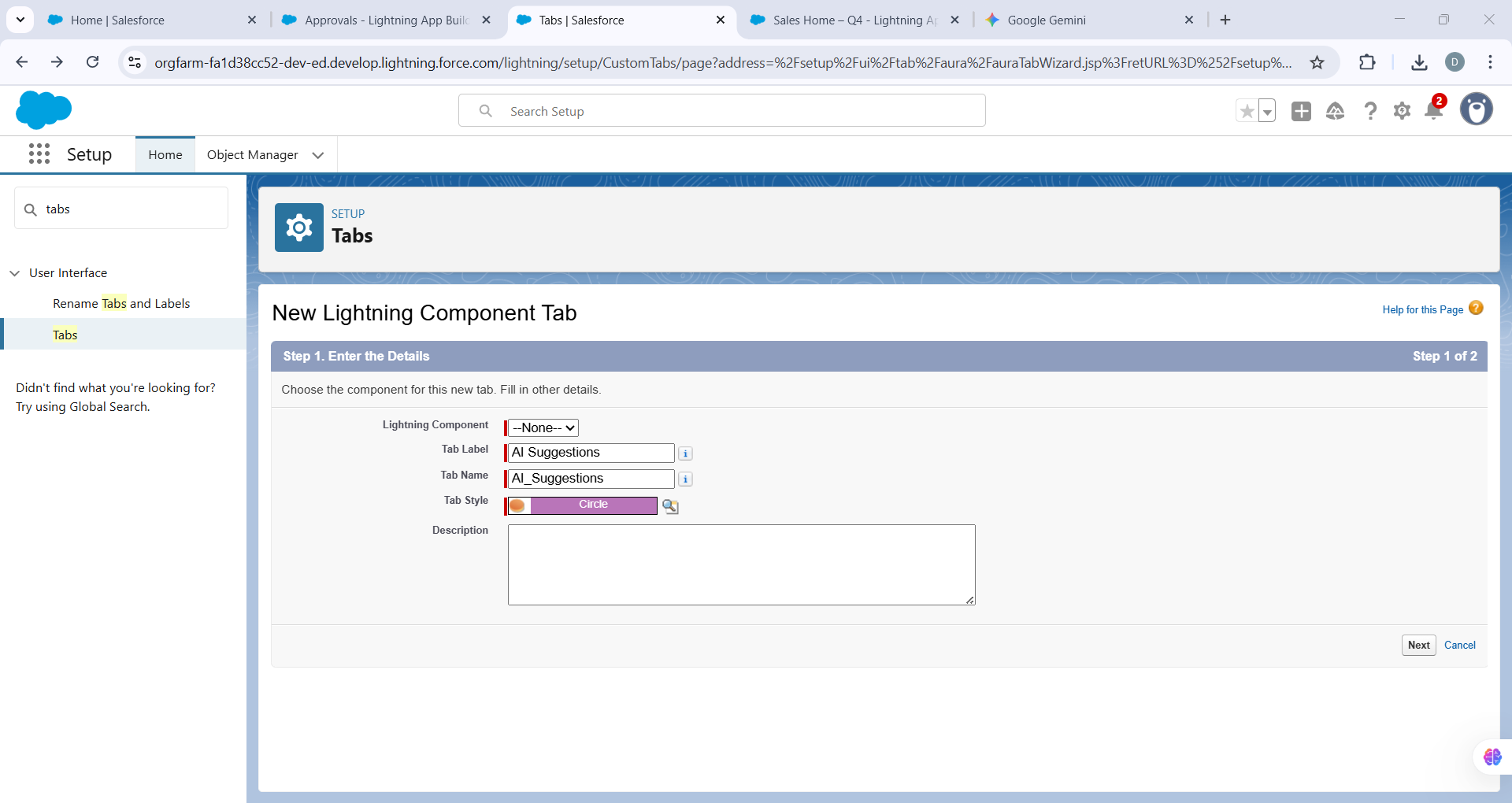
* 

## Step 4 – Tabs

* • Setup → Tabs → New → Lightning Component Tab.
* • Select your LWC (exposed with lightning\_\_Tab target).
* • Provide Tab Label, Icon, and Style for easy identification.
* • Add Tab to Lightning App navigation menu in App Manager.
* • Users can now access LWC-based functionality directly from navigation bar

.Create a new **Lightning Component Tab**

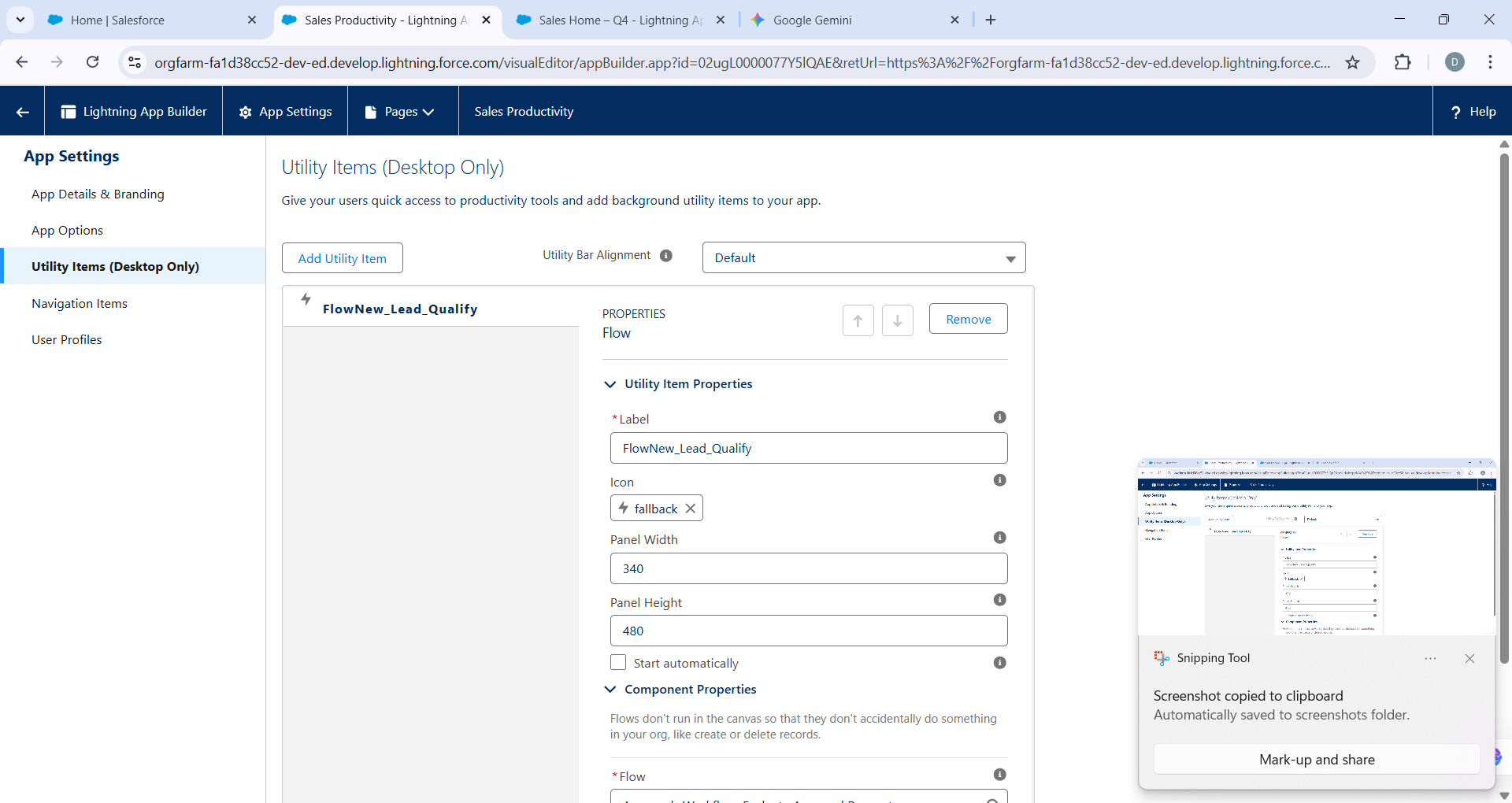
1. Click **New** under **Lightning Component Tabs**.
2. From the **Lightning Component** dropdown, select the **LWC** or **Aura**
3. **Component** you want to expose as a tab.
   * **Important:** For LWCs, ensure the component’s .js-meta.xml file includes the <target>lightning\_\_Tab</target> entry. Example:

* 

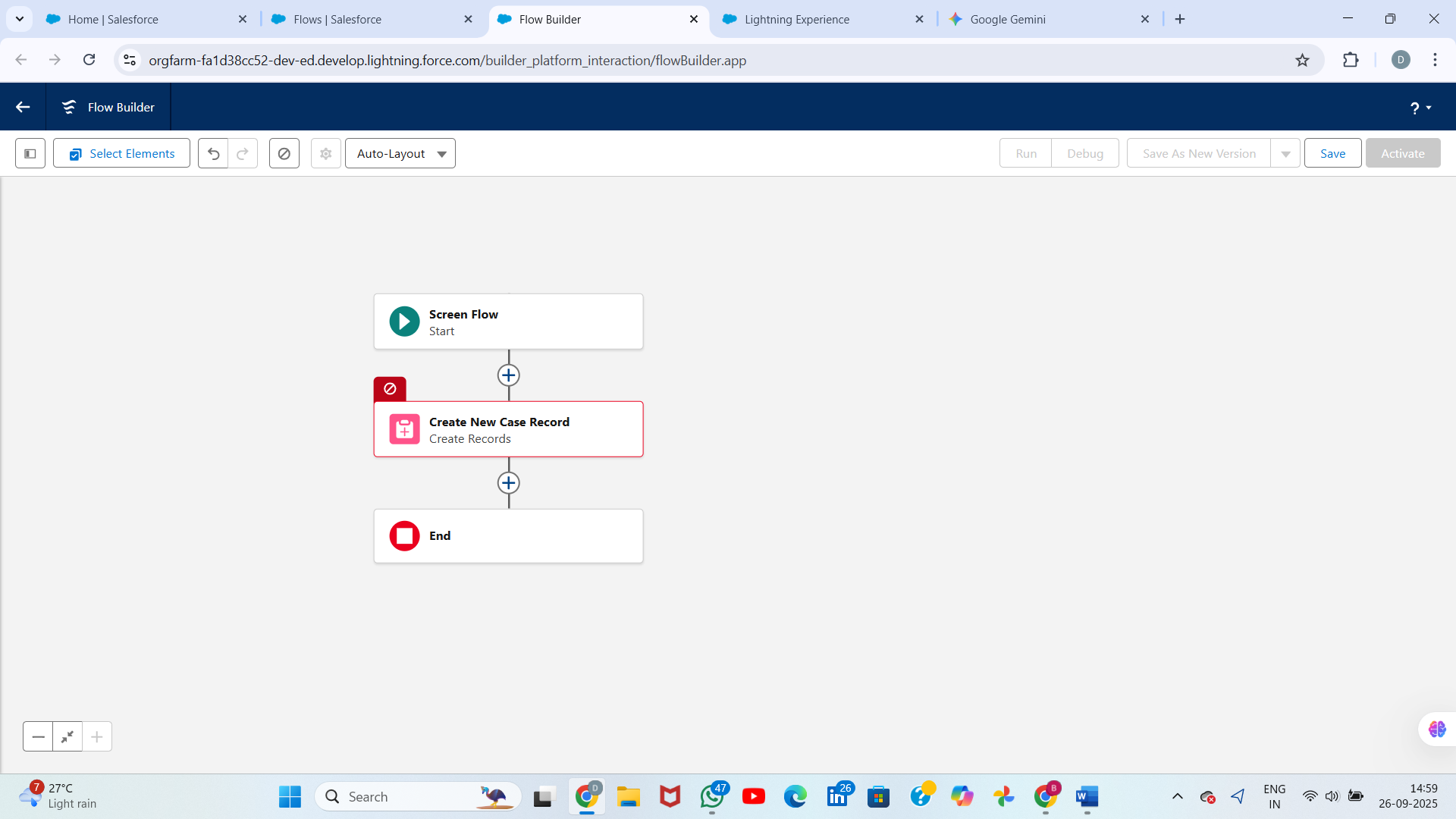
## Step 5 – Utility Bar

* • Utility Bar provides quick global access without leaving the page.
* • Navigate to App Manager → Edit your Lightning App → Utility Items (Desktop only).
* • Add components such as Flows, Notes, Recent Items, or LWCs.
* • Configure panel size, icon, and label.
* • Ensure LWC meta file includes `<target>lightning\_\_UtilityBar</target>`.
* • Example use case: Utility Bar with 'New Task' flow and 'Review AI Suggestions' LWC.
* n **App Manager**, edit the Lightning App → **Utility Items** (Desktop only).
* Add items: **Flow**, **Quick Action**, **Notes**, or a **Lightning Component** (your LWC) as a utility. Configure label/icon and panel size. [Salesforce+1](https://help.salesforce.com/s/articleView?id=platform.apps_lightning_utilities.htm&language=en_US&type=5&utm_source=chatgpt.com)

If you want an LWC in the Utility Bar, ensure the component’s meta XML includes the lightning\_\_UtilityBar target. (Salesforce supports LWC in Utility Bar.) [Salesforce](https://help.salesforce.com/s/articleView?id=release-notes.rn_console_lwc_utility.htm&language=en_US&release=218&type=5&utm_source=chatgpt.com)

* 

## Step 6 – Flows in UI

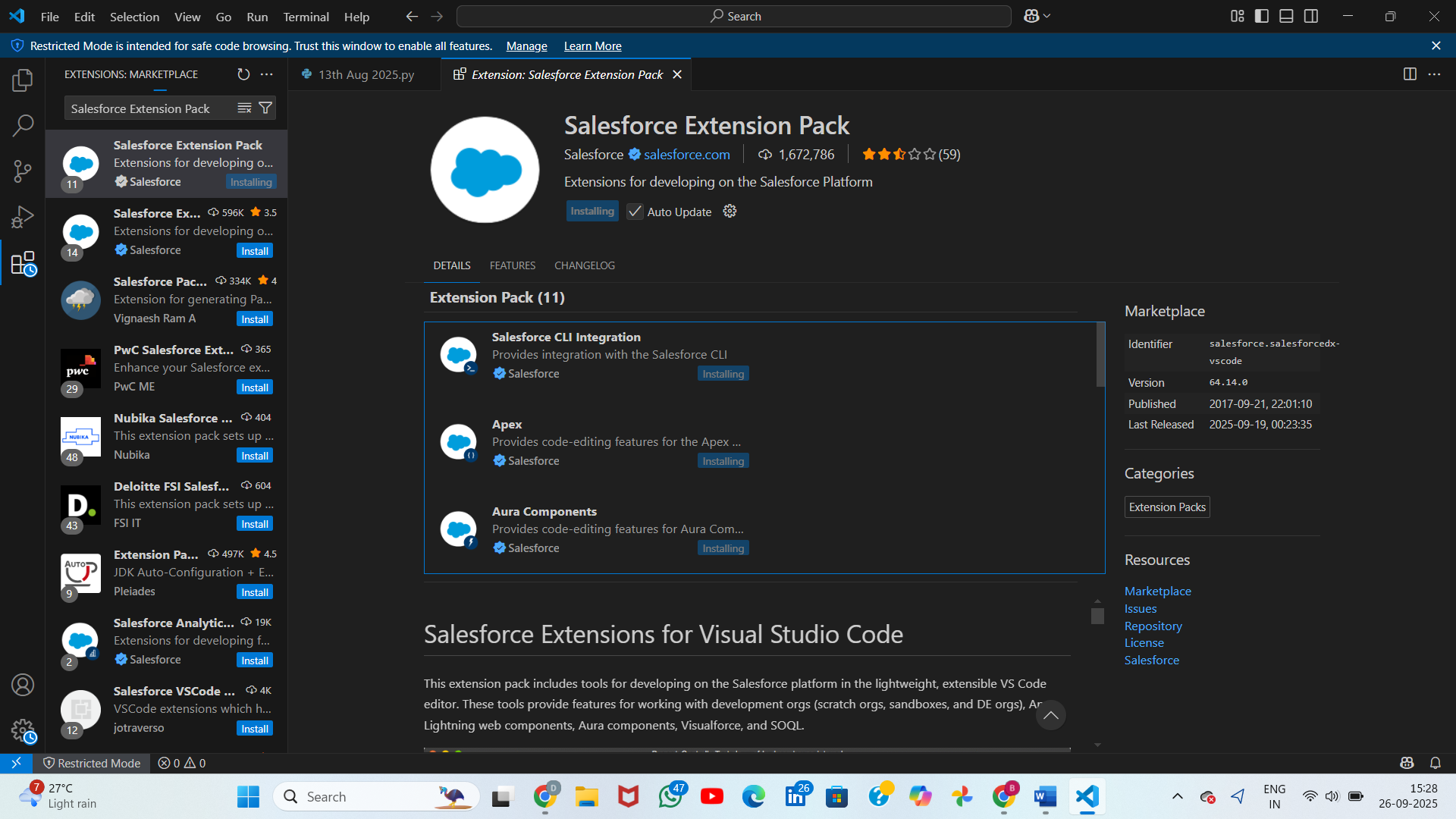
* • Create Flow (e.g., Screen Flow for creating tasks).
* • Embed Flow inside Record Page (for guided updates) or Home Page (for quick task creation).
* • Utility Bar Flows are globally available but don’t auto-pass recordId.
* • Flows can also call Apex actions to perform complex validations or data manipulations.
* • Flows ensure admins can extend functionality without code.
* 

## Step 7 – Lightning Web Components (LWC)

* • Set up Salesforce DX project: `sfdx force:project:create -n MyProject`.
* • Create LWC: `sfdx force:lightning:component:create -n aiSuggestionsList`.
* • Expose component via meta XML to Record Pages, App Pages, Home Page, Utility Bar, and Tabs.
* • Design HTML template for data display and interactivity.
* • Use reactive properties with @track and @api to handle inputs/outputs.
* • Deploy using VS Code or CLI to Salesforce org.

### ****Install Salesforce Extension Pack in VS Code****

1. Open VS Code.
2. Go to **Extensions** (left sidebar icon or press Ctrl+Shift+X).
3. Search for **Salesforce Extension Pack**.
4. Click **Install**.

* 

## Step 8 – Apex with LWC

* • Apex provides server-side logic for LWC interactions.
* • Create @AuraEnabled methods for LWC consumption.
* • Annotate read-only queries with cacheable=true for performance.
* • Handle exceptions and enforce CRUD/FLS compliance.
* • Provide both retrieval (getAISuggestions) and DML (createSuggestion) methods.
* • Unit test Apex with at least 75% coverage before deployment.

## Step 9 – Events in LWC

* • Use CustomEvent for child-to-parent communication within component hierarchy.
* • For unrelated components, use Lightning Message Service (LMS).
* • Events allow modularization and reusable components.
* • Example: Feedback child LWC sends event to parent LWC for saving record.

## Step 10 – Wire Adapters

* • Wire Adapters connect LWCs to Salesforce data and metadata declaratively.
* • Common adapters: getRecord, getObjectInfo, getPicklistValues.
* • Enable real-time sync between UI and Salesforce without writing Apex.
* • Syntax example:

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

* Use built-in wire adapters for record metadata and records (getRecord, getObjectInfo, getPicklistValues, etc.). Use @wire(getRecord, { recordId: '$recordId', fields }). [Salesforce Developers+1](https://developer.salesforce.com/docs/platform/lwc/guide/reference-wire-adapters-record.html?utm_source=chatgpt.com)

Example:

import { getRecord } from 'lightning/uiRecordApi';

@wire(getRecord, { recordId: '$recordId', fields: ['Task.Subject','Task.Status'] })

taskRecord;

## Step 11 – Imperative Apex Calls

* • Use imperative calls when parameters are dynamic or DML is needed.
* • Call Apex with methodName({param:value}) in JS and handle Promise.
* • Example: Insert Feedback\_\_c record via createFeedback Apex method.
* • Combine imperative calls with refreshApex() to refresh wired data.
* Used when you need to pass runtime params or perform DML. Call apex method and handle promise .then/.catch. Call refreshApex() to refresh wired data after changes. [Salesforce Developers](https://developer.salesforce.com/docs/platform/lwc/guide/apex-call-imperative.html?utm_source=chatgpt.com)

## Step 12 – Navigation Service

* • Use NavigationMixin in LWC for navigation to record pages, list views, or object home.
* • Example: Navigate to AI\_Suggestion\_\_c record page after creation.
* • Reduces manual navigation effort for users, improving productivity.
* Use NavigationMixin from lightning/navigation to open records, object home, list views, or custom tabs. Example shown in the LWC JS above

## Step 13 – Activation & Assignments

* • Save and Activate Lightning Pages (Record/Home/App).
* • Assign as Org Default, App Default, or Profile-based.
* • Ensure proper permissions on objects/fields.
* • Test navigation, flow access, and LWC visibility for different profiles.
*  For each Lightning Page you created: **Save** → **Activate** → choose App Default / App & Profile / Org Default.
*  For the App: ensure the Utility Bar items and Nav Items are ordered correctly, and assigned profiles can see the app.
*  Validate that LWC security (FLS/CRUD) and Apex sharing are correct.

## Step 14 – Testing Checklist

* • Confirm all LWCs appear in Lightning App Builder palette.
* • Verify Home, App, and Record Pages render expected components.
* • Run Apex Unit Tests for backend classes.
* • Validate CRUD/FLS access controls.
* • Check Flow execution paths for different input scenarios.
* • Test LMS-based communication between Utility Bar and Record Pages.
*  Confirm LWC appears in Lightning App Builder Components list and drag into pages.
*  Test on different profiles (page visibility/field level security).
*  Test Flow embedded in Page and in Utility Bar; check that flows embedded in Utility Bar cannot read current recordId unless you wire it differently (utility bar is global). [Trailhead](https://trailhead.salesforce.com/content/learn/modules/screen_flow_distribution/screen_flow_distribution_everypage?utm_source=chatgpt.com)
*  Run Apex unit tests and LWC Jest tests (if you add front-end tests).
*  Check performance: use cacheable Apex for read calls and refreshApex where needed

## Step 15 – Deployment

* • Use Change Sets for declarative elements like Pages, Tabs, and Flows.
* • Use SFDX CLI for code deployments (Apex, LWC).
* • Maintain version control in GitHub or Bitbucket for tracking changes.
* • Set up CI/CD pipelines using GitHub Actions or Jenkins for automation.
* • Deploy consistently across Sandbox, UAT, and Production environments.
* Use change sets (Setup) or CI/CD with SFDX (sfdx force:source:deploy / packages). For repeatable delivery, use source control (Git) + pipelines.

## Conclusion

Phase 6 provides a structured approach to building an end-to-end Salesforce user interface for AI-powered task management. By combining declarative tools (Lightning App Builder, Flows, Utility Bar, Tabs) with programmatic tools (LWC, Apex, Events, Wire Adapters), the solution balances flexibility with robustness. Proper testing, documentation, and deployment practices ensure adoption by end-users while maintaining system integrity and scalability.