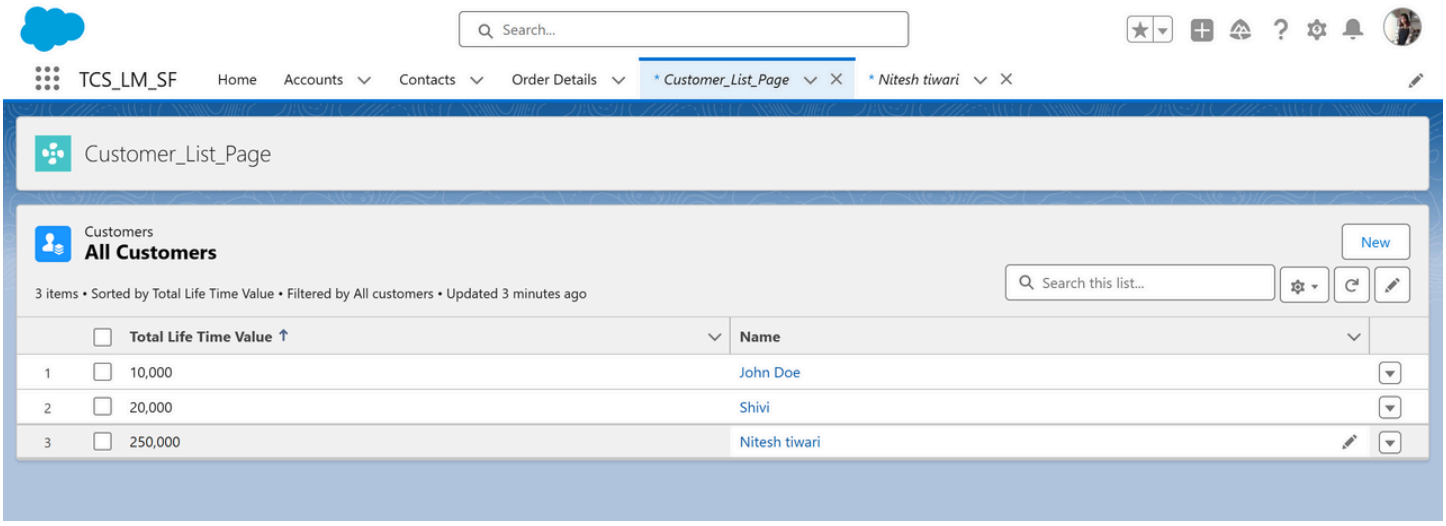
# Phase 6: User Interface Development

## 1. Lightning App Builder

- **Objective:** Create a Lightning App Page to display Customer__c records.
- **Steps Implemented:**

1. Setup → Lightning App Builder → New → **App Page**
2. Page Name: Customer_List_Page
3. Layout: One Region (simple view)
4. **Components Added:**
   - **List View** → Object: Customer__c, List View: “All”
   - **Report Chart** (optional) → linked to existing “Sample Flow Report: Screen Flows”
5. Save & Activate → Assigned to **Org Default** for access
6. **Records Added:**
   - Created 5–6 sample Customer__c records
   - Party → New Individual created (First Name, Last Name)
   - Filled required fields: Name, Customer Status Type, Total Lifetime Value, Age__c

- **Outcome:**
- App Page now shows **Customer List** and **Report Chart**
- Users can view records and monitor basic metrics via List and Charts



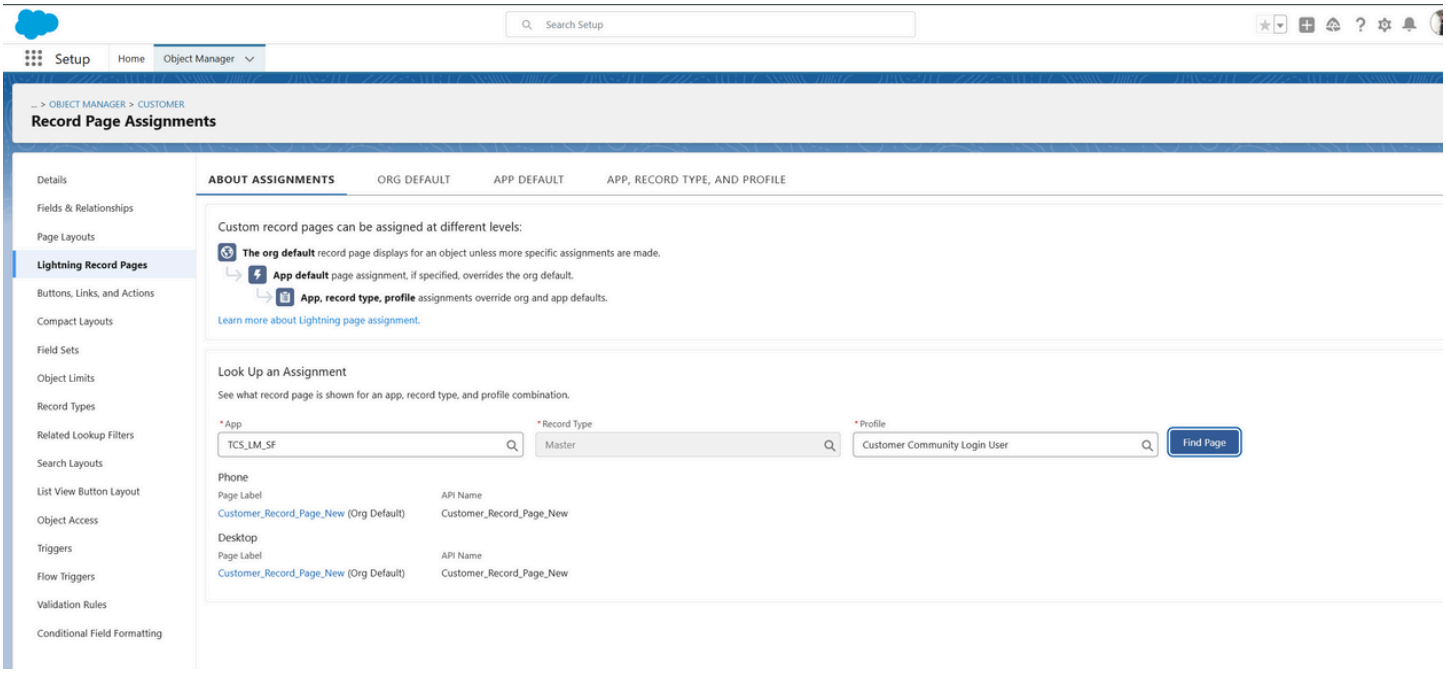

---

## 2. Record type

- **Lightning App Builder – Customer List Page**
- **Objective:** Create a Lightning App Page to display Customer__c records.
- **Steps Implemented:**
  1. Setup → Lightning App Builder → New → **App Page**

2. Page Name: Customer_List_Page
3. Layout: One Region (simple view)
4. **Components Added:**
   - **List View** → Object: Customer__c, List View: “All”
   - **Report Chart** (optional) → linked to existing “Sample Flow Report: Screen Flows”
5. Save & Activate → Assigned to **Org Default** for access
6. **Records Added:**
   - Created 5–6 sample Customer__c records
   - Party → New Individual created (First Name, Last Name)
   - Filled required fields: Name, Customer Status Type, Total Lifetime Value, Age__c

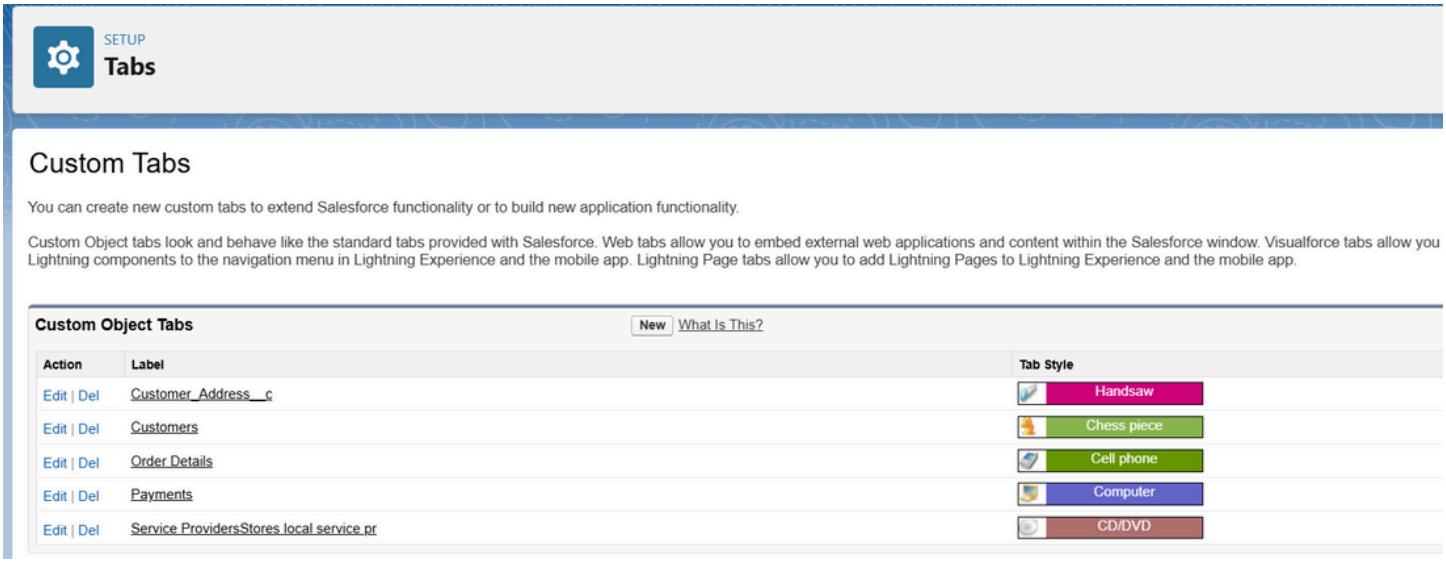
- **Outcome:**
  - App Page now shows **Customer List** and **Report Chart**
  - Users can view records and monitor basic metrics via List and Charts




## 3. Tabs

- **Objective:** Organize multiple sections on a record or app page for easy access.
- **Steps Implemented:**
- Lightning App Builder → Open Record Page → **Add Tabs** component
- Created multiple tabs:
  - **Details** → Record Detail component
  - **Related Lists** → Shows related records
  - **Reports / Charts** → Optional analytics
- **Save & Activate:**

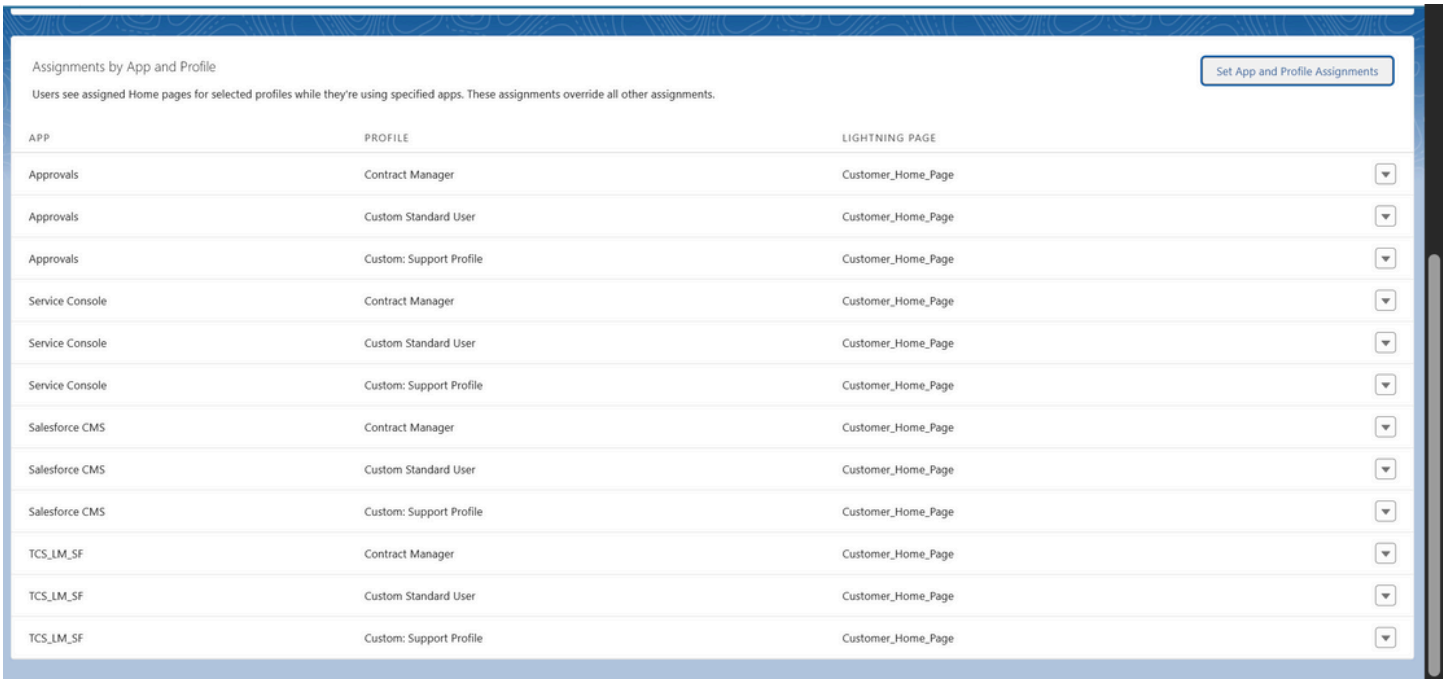
- Tabs layout saved as part of **Customer_Record_Page_New**
- Page assigned as **Org Default**
- **Outcome:**
- Users can switch between sections without scrolling
- Improves page organization and usability




## 4. Home page layouts

- **Objective:** Customize Home Page to show relevant components and information to users.
- **Steps Implemented:**
- Setup → Lightning App Builder → **New** → **Home Page**
- Page Name: e.g., **Customer_Home_Page**
- Layout: Standard or One Region (depending on need)
- **Components Added:**
- **Report Charts** → show key metrics
- **List Views** → quick access to records
- **Rich Text / Images** → announcements or instructions
- **Tabs** (optional) → organize multiple sections
- **Save & Activate:**
- Assign to **Org Default** or specific App Profiles
- **Records / Data Used:**
- Sample Customer__c records created to populate List Views and Charts
- **Outcome:**
- Home Page displays dashboards, lists, and important info

- Users can quickly access metrics and records from one page



Assignments by App and Profile

Users see assigned Home pages for selected profiles while they're using specified apps. These assignments override all other assignments.

Set App and Profile Assignments

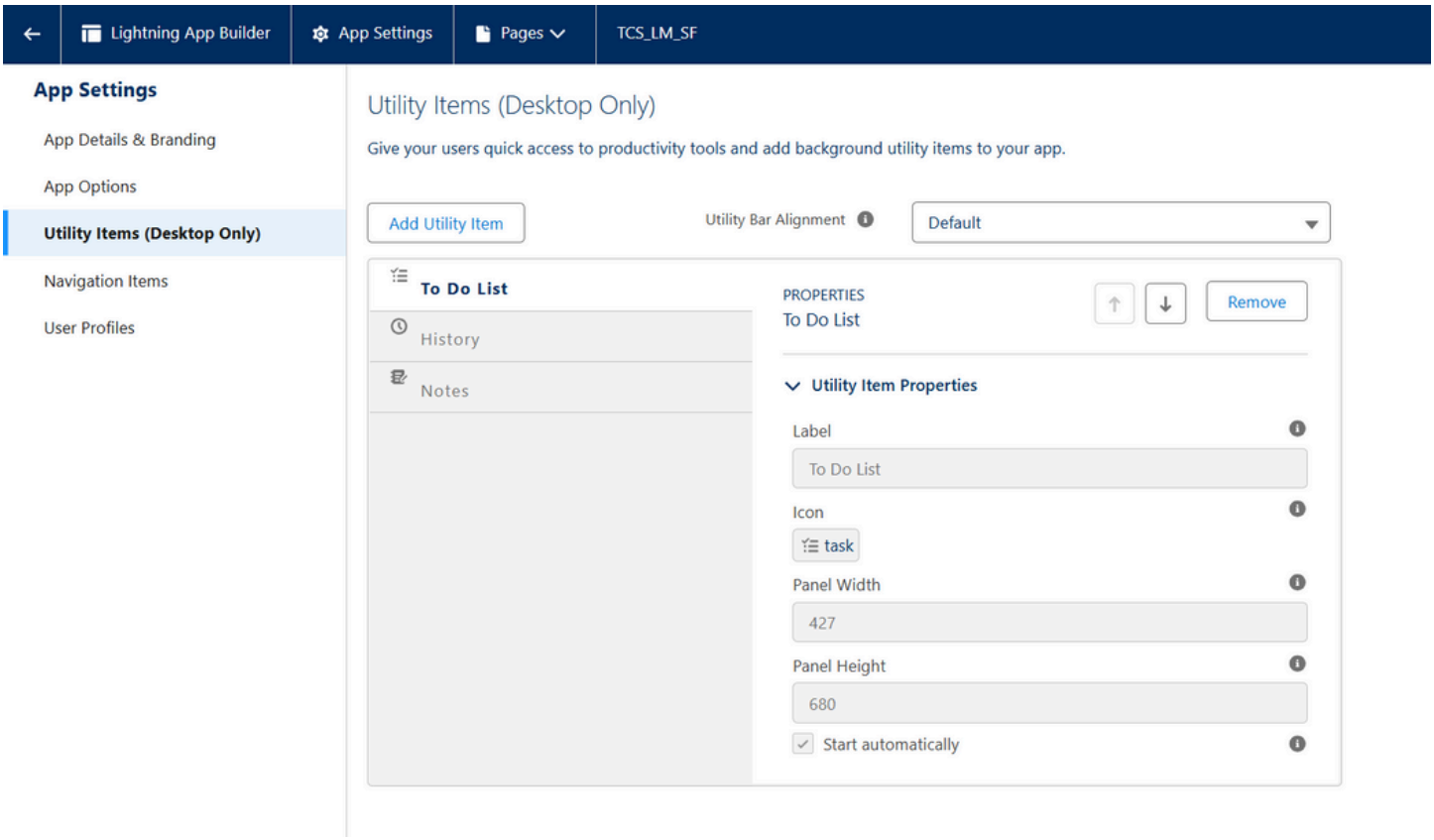
| APP | PROFILE | LIGHTNING PAGE |
|---|---|---|
| Approvals | Contract Manager | Customer_Home_Page |
| Approvals | Custom Standard User | Customer_Home_Page |
| Approvals | Custom: Support Profile | Customer_Home_Page |
| Service Console | Contract Manager | Customer_Home_Page |
| Service Console | Custom Standard User | Customer_Home_Page |
| Service Console | Custom: Support Profile | Customer_Home_Page |
| Salesforce CMS | Contract Manager | Customer_Home_Page |
| Salesforce CMS | Custom Standard User | Customer_Home_Page |
| Salesforce CMS | Custom: Support Profile | Customer_Home_Page |
| TCS_LM_SF | Contract Manager | Customer_Home_Page |
| TCS_LM_SF | Custom Standard User | Customer_Home_Page |
| TCS_LM_SF | Custom: Support Profile | Customer_Home_Page |

## 4. Utility bar

- **Objective:** Customize **TCS_LM_SF** app for Local Business Service CRM without creating a new app.
- **Steps Implemented:**

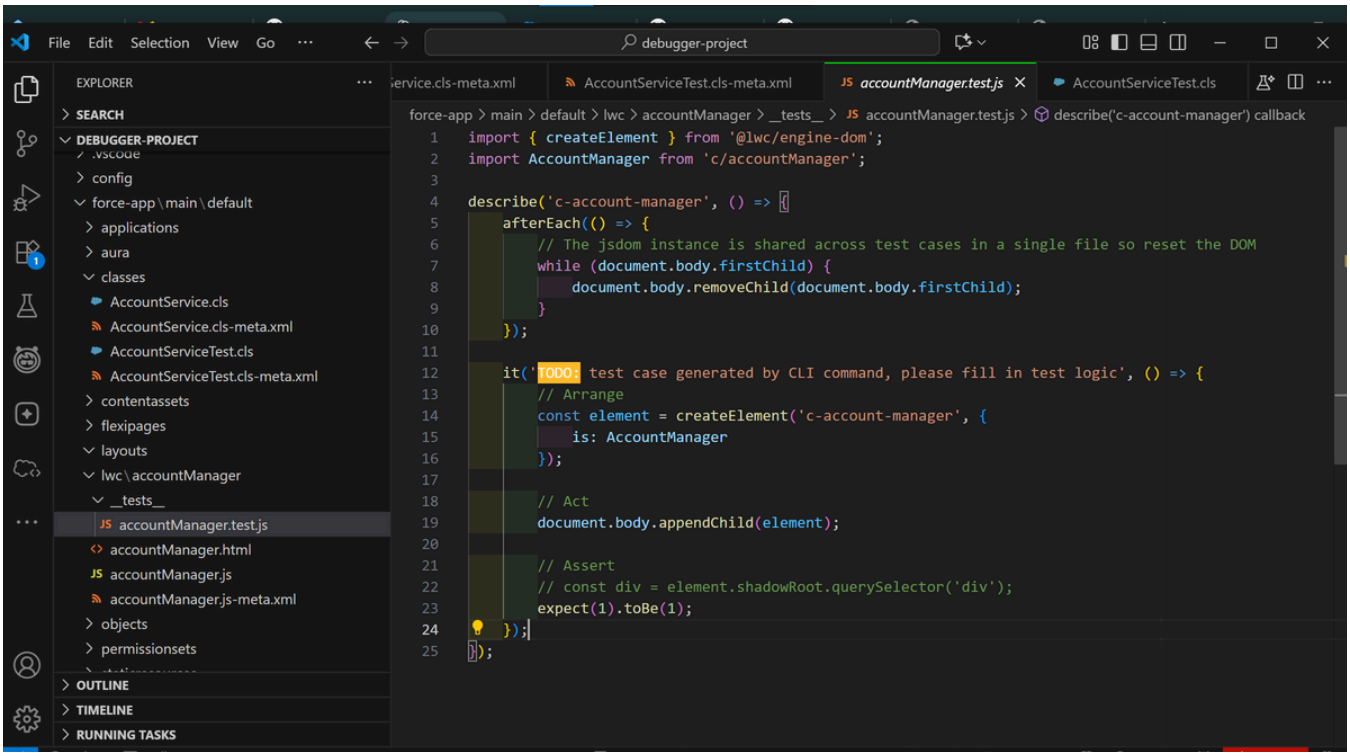
1. **Edit App:** Setup → App Manager → **TCS_LM_SF** → Edit
2. **Update Details (Optional):** Name: Customer CRM App, Description: Manage customers, tasks, service requests
3. **Navigation & Tabs:** Console navigation; Tabs: Customers, Service Requests, Reports/Dashboard
4. **Utility Bar:** Add utilities: Notes, Task List/Jobs, Customer History, Dashboard, Calculator
5. **Utility Properties:** Set Label, Icon, Panel Size/Behavior
6. **Save & Activate:** Assign as App Default / Org Default
7. **Verify:** Open App → Utility Bar visible → Test all tools

- **Outcome:**
- Quick access to tasks, customer history, service requests, and dashboards
- Improves workflow and efficiency for CRM users




## 5. Lightning Web Component (LWC)

- **Purpose:**
- To provide a user interface for managing customer accounts (create, update, and fetch account details).
- To interact with **Apex backend (AccountService)** for CRUD operations.**JavaScript (JS):** Handles user input, calls Apex methods via @AuraEnabled.
- **HTML:** Lightning inputs and buttons for Name, Type, Email, Phone; buttons for Create, Update, Fetch.
- **Meta.xml:** Exposed to Lightning App Builder for **Record Page, App Page, and Home Page**.




## 6. Apex with LWC

- **Purpose:**
- To handle all backend operations for Account management (create, update, fetch account details) that the LWC will call.
- Provides a clear separation between frontend UI (LWC) and backend logic (Apex).
- **Implementation Details: Apex Class – AccountService**
- Methods:
  - createAccount(name, type, phone) → Creates a new Account record.
  - updateAccount(accountId, name, phone) → Updates existing Account record.
  - getAccountDetails(accountId) → Fetches Account details for display.
- @AuraEnabled annotation used to allow **LWC to call these methods**.
- Only standard Account fields used (Name, Type, Phone) to avoid Person Account errors.

Manager

SETUP
**Apex Classes**

**Apex Class Detail** Edit Delete Download Run Test Show Dependencies

| | | | |
|---|---|---|---|
| Name | AccountServiceTest | Status | Active |
| Namespace Prefix | | Created By | Urmi shrivas , 10/9/2025, 9:56 AM |
| Last Modified By | Urmi shrivas , 10/9/2025, 9:56 AM | | |

Class Body | Class Summary | Version Settings | Trace Flags

```
1  @IsTest
2  private class AccountServiceTest {
3
4      @IsTest
5      static void testCreateAndUpdateAccount() {
6          Id accId = AccountService.createAccount('Test Customer', 'Customer', '1234567890');
7          Account acc = [SELECT Id, Name, Phone FROM Account WHERE Id = :accId];
8          System.assertEquals('Test Customer', acc.Name);
9
10         AccountService.updateAccount(accId, 'Updated Name', '0987654321');
11         acc = [SELECT Name, Phone FROM Account WHERE Id = :accId];
12         System.assertEquals('Updated Name', acc.Name);
13     }
14
15     @IsTest
16     static void testGetAccountDetails() {
17         Account acc = new Account(Name='Test Provider', Type='Service Provider', Phone='1112223333');
18         insert acc;
19         Account result = AccountService.getAccountDetails(acc.Id);
20         System.assertEquals('Test Provider', result.Name);
21     }
22 }
```

Manager

SETUP
**Apex Classes**

**Apex Class Detail** Edit Delete Download Run Test Show Dependencies

| | | | |
|---|---|---|---|
| Name | AccountServiceTest | Status | Active |
| Namespace Prefix | | Created By | Urmi shrivas , 10/9/2025, 9:56 AM |
| Last Modified By | Urmi shrivas , 10/9/2025, 9:56 AM | | |

Class Body | Class Summary | Version Settings | Trace Flags

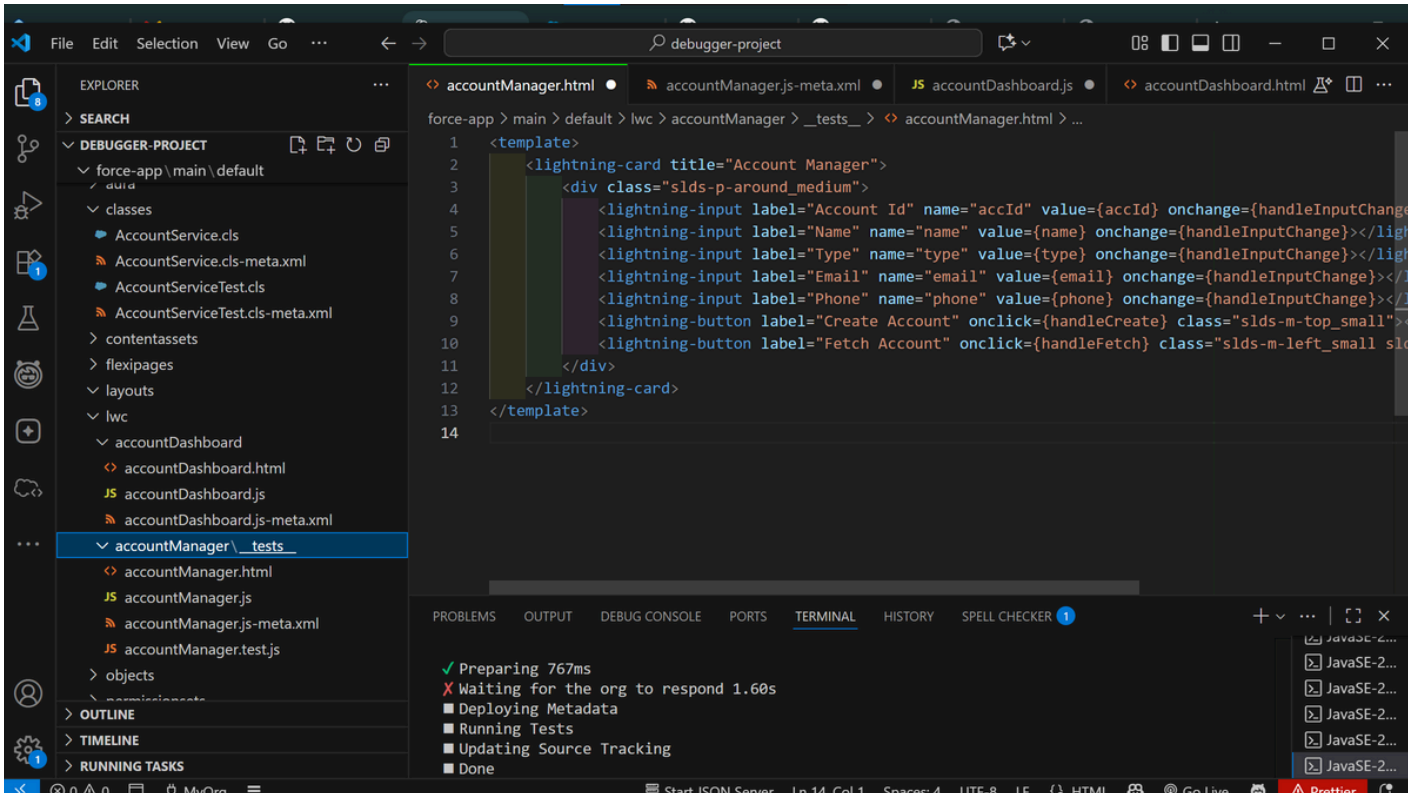
```
1  @IsTest
2  private class AccountServiceTest {
3
4      @IsTest
5      static void testCreateAndUpdateAccount() {
6          Id accId = AccountService.createAccount('Test Customer', 'Customer', '1234567890');
7          Account acc = [SELECT Id, Name, Phone FROM Account WHERE Id = :accId];
8          System.assertEquals('Test Customer', acc.Name);
9
10         AccountService.updateAccount(accId, 'Updated Name', '0987654321');
11         acc = [SELECT Name, Phone FROM Account WHERE Id = :accId];
12         System.assertEquals('Updated Name', acc.Name);
13     }
14
15     @IsTest
16     static void testGetAccountDetails() {
17         Account acc = new Account(Name='Test Provider', Type='Service Provider', Phone='1112223333');
18         insert acc;
19         Account result = AccountService.getAccountDetails(acc.Id);
20         System.assertEquals('Test Provider', result.Name);
21     }
22 }
```


## 7. Events in LWC

- **Purpose:**
- To enable **communication between Lightning Web Components**.
- Specifically, the child component (accountManager) notifies the parent component (accountDashboard) whenever a new account is created.
- Demonstrates **child-to-parent interaction** in LWC.

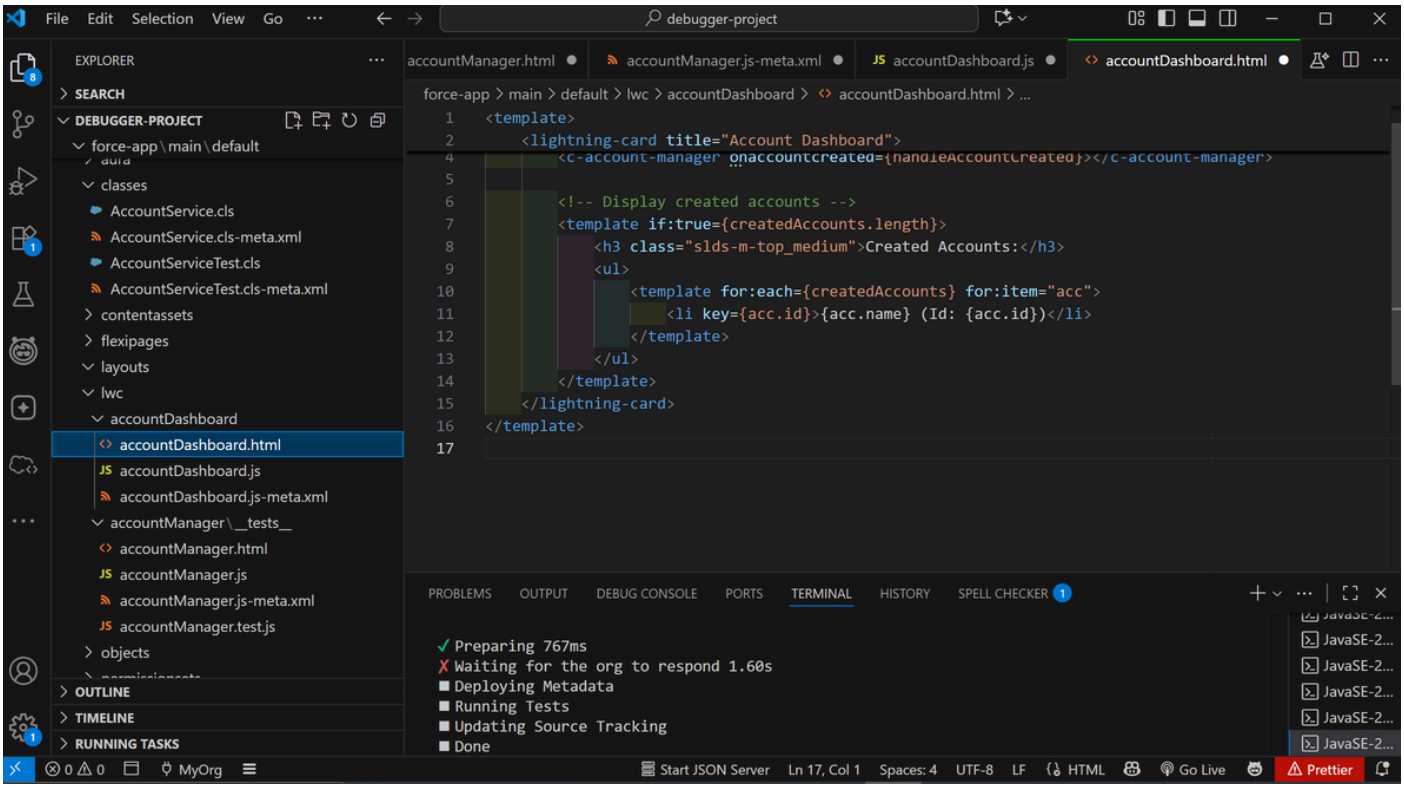
## Implementation Details:

1. **Child Component – accountManager**
   - Handles input fields: Name, Type, Phone.
   - Calls Apex method createAccount() to create an Account.
   - After successful creation, dispatches a **CustomEvent** accountcreated containing account data.



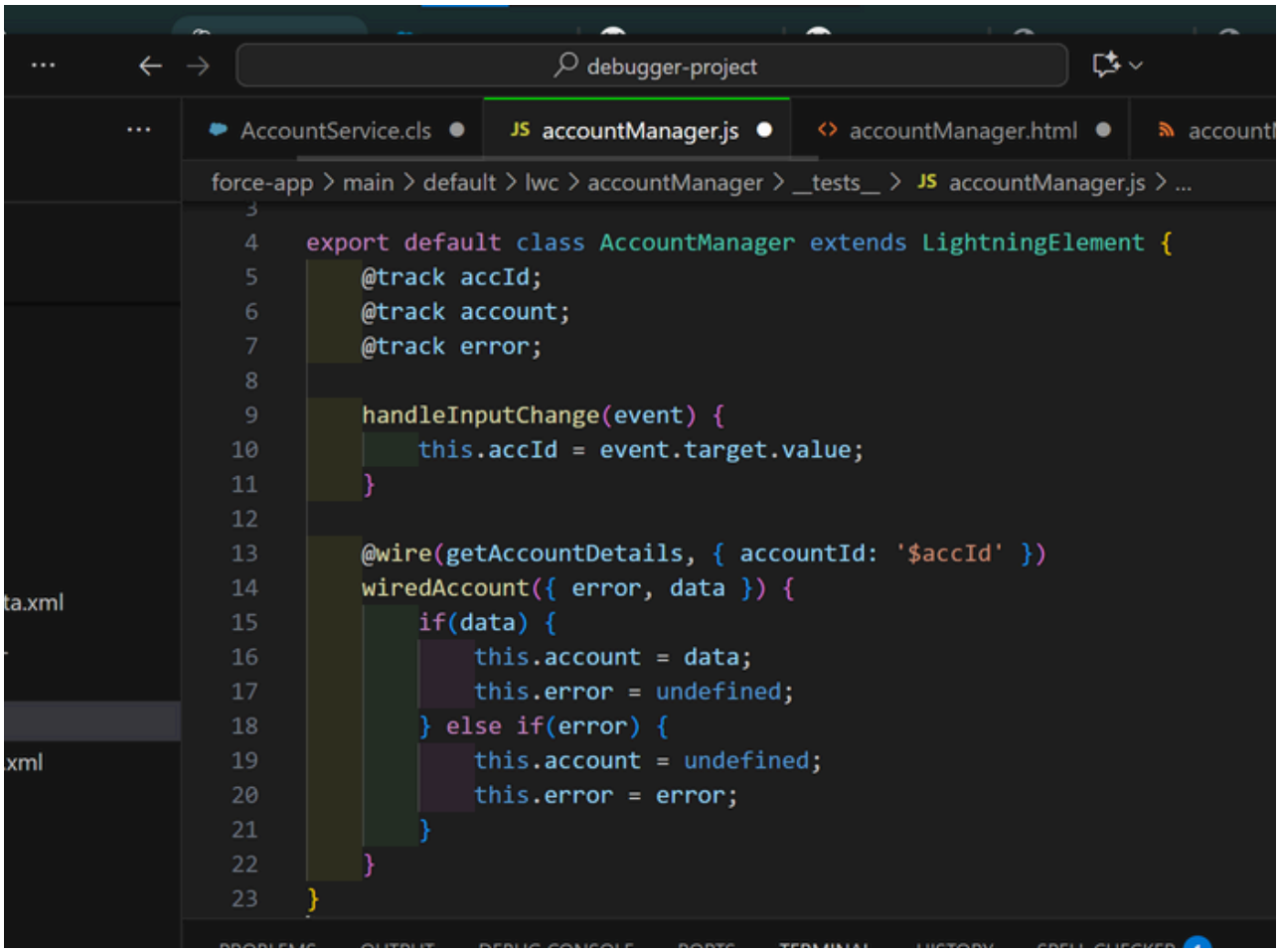

## Parent Component – accountDashboard

- Listens for the accountcreated event from accountManager.
- Updates a list of created accounts dynamically.




## Wire Adapters in LWC

- **Purpose:**
- To fetch Salesforce data **reactively** in LWC without explicitly calling Apex methods imperatively.
- In this project, we use **Wire Adapters** to fetch Account details whenever the user enters an Account Id.
- Demonstrates **automatic data updates and reactive properties** in LWC.




### Imperative Apex Calls

- **Purpose:**
- To call Apex methods **manually from JavaScript** based on user actions.
- In this project, we use imperative calls to **create a new Account** when the user clicks a button.
- Provides **full control** over when and how Apex is invoked, unlike reactive Wire Adapters.

AccountService.cls | accountManager.js | accountManager.html | accountManager.js-meta.xml

force-app > main > default > lwc > accountManager > __tests__ > accountManager.js > AccountManager > hand

```
4   export default class AccountManager extends LightningElement {
12      }
13
14      handleCreateAccount() {
15          // Imperative Apex Call
16          createAccount({ name: this.name, type: this.type, phone: this.phone })
17              .then(result => {
18                  this.createdAccountId = result;
19                  console.log('Account created with Id: ' + result);
20                  // Optionally, dispatch an event to parent
21                  const createdEvent = new CustomEvent('accountcreated', { detail: { id
22                  this.dispatchEvent(createdEvent);
23
24                  // Reset inputs
25                  this.name = '';
26                  this.type = '';
27                  this.phone = '';
28              })
29              .catch(error => {
30                  console.error('Error creating account:', error);
31              });
```