

INVENTORY AND ORDER TRACKING DASHBOARD

Phase 6: User Interface Development

1. Lightning App Builder — create the app

What / Why: A Lightning app bundles navigation items (Products, Orders, Dashboards). Users will launch this app.

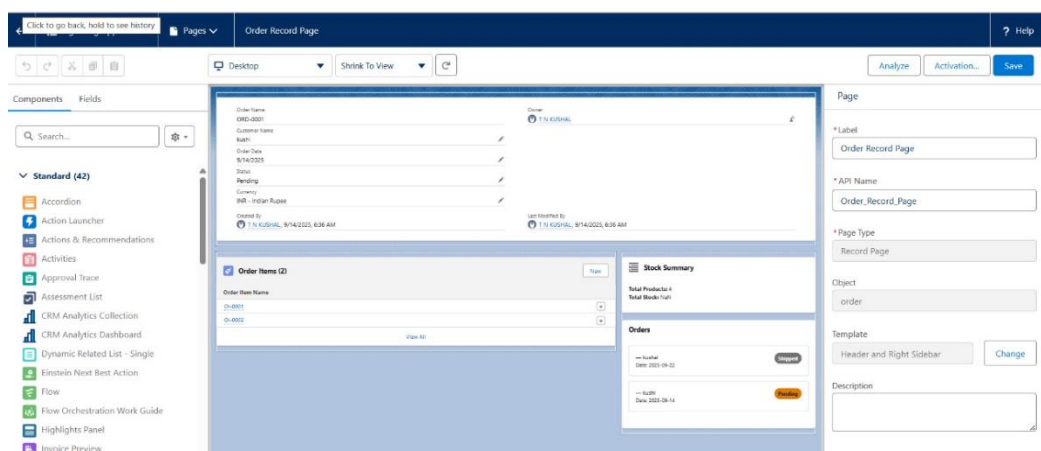
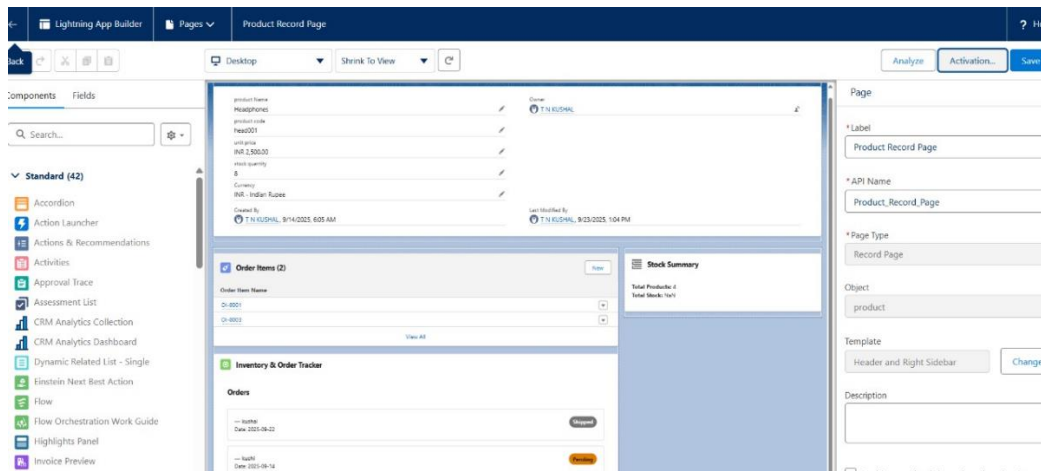
- Steps (Setup UI):
 - i. In Salesforce: Setup → App Manager → New Lightning App.
 - ii. Name: Inventory & Order Tracker.
 - iii. Navigation: Standard
 - iv. Add Navigation Items: Products, Orders, Reports, Dashboards

The screenshot shows the 'App Details & Branding' configuration page in the Salesforce Lightning App Builder. The page is divided into two main sections: 'App Details' and 'App Branding'. The 'App Details' section includes fields for 'App Name' (Inventory & Order Tracker), 'Developer Name' (KUSHAL_tn7), and 'Description' (To track our items). The 'App Branding' section includes an 'Image' field with a clipboard icon, a 'Primary Color Hex Value' field with a blue color picker and the value #0070C2, and a checkbox for 'Org Theme Options' (Use the app's image and color instead of the org's custom theme). Below these sections is an 'App Launcher Preview' showing the app icon and name.

2. Record Pages — customize Product_c / Order_c record pages

What / Why: Control how each product or order record looks.

- Steps (Setup UI):
 - i. Setup → Object Manager → Product__c → Lightning Record Pages → New.
 - ii. Choose template (Header + 2 columns recommended).
- Drag components:
 - Record Details (standard)
 - Related Lists (order items, orders)
 - Custom LWCs (e.g., c-inventory-dashboard-product-details — optional)
 - Click Save → Activate and set as org default or app default (Inventory app).



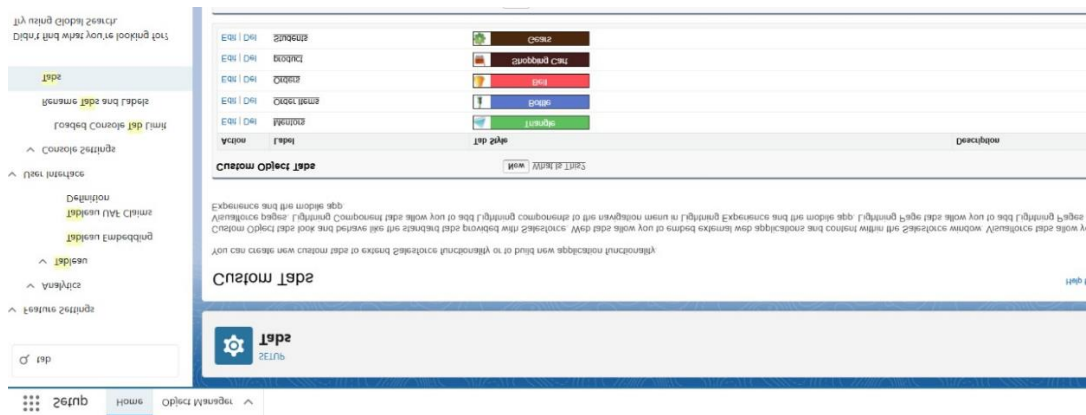
3. Tabs — make LWC or object tabs available in your app

- What / Why: Users need easy navigation to key objects and components.
- Steps (Setup UI):

Setup → Tabs → Object Tabs → New → choose Product c and create a tab.

- For LWC pages, create Lightning Page Tabs via App Page then add to app navigation:

Lightning App Builder → New → App Page → add your LWC → Save → Activate → add to App Manager nav items.

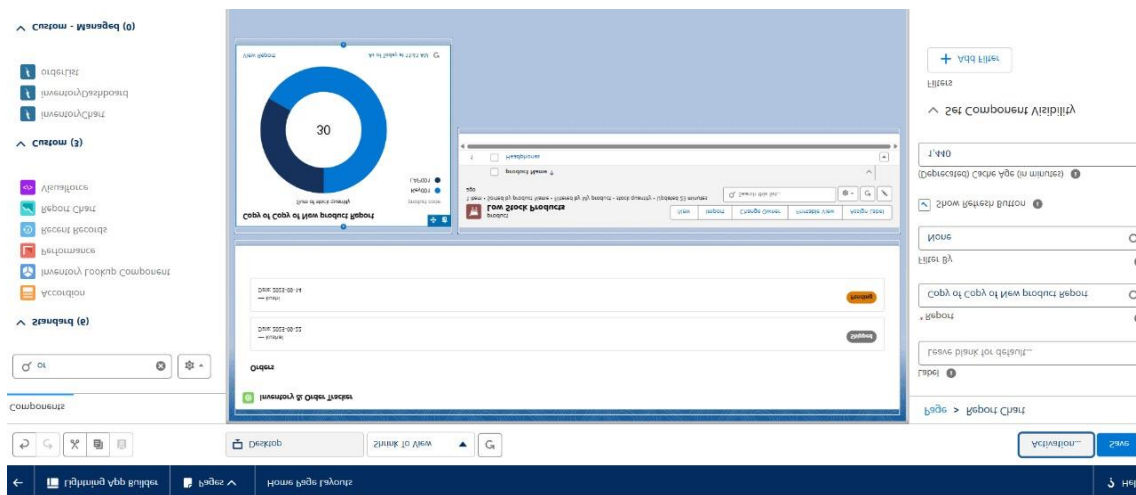


4. Home Page Layouts — a dashboard landing experience

- What / Why: A home page shows summary (stock alerts, top orders, charts).
- Steps (Setup UI):

Setup → Lightning App Builder → Home Page → New.

- Add components:
 - Report Chart (stock levels)
 - List View (Recently Updated Products)
 - Custom LWC (inventory Dashboard)
 - Save → Activate (assign to profiles or app).



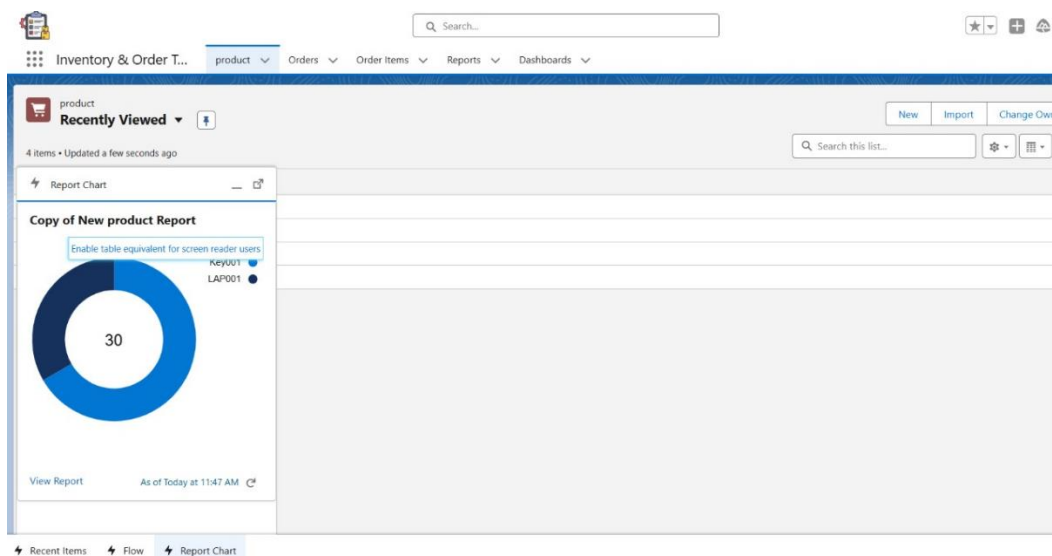
5. Utility Bar — quick tools available across the app

- What / Why: Quick access to frequently used actions: Quick Create Product, Recent Orders, Stock Alerts.

- Steps (Setup UI):

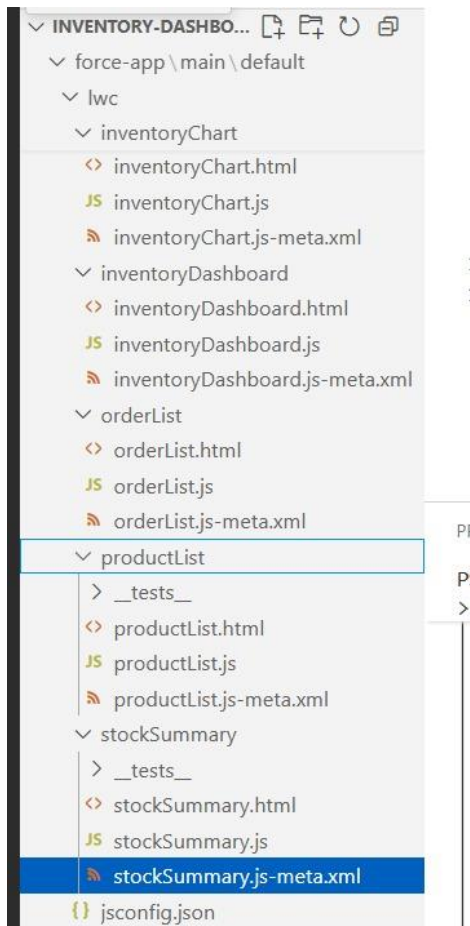
Setup → App Manager → find Inventory & Order Tracker → Edit → Utility Bar.

- Add items:
- Recent Items (built-in)
- Custom Lightning Component (for quick stock add)
- Flow or Report (Stock Alerts)
- Set icon, width, behavior → Save.



6. LWC (Lightning Web Components) — create the components

- What / Why: Build interactive UI pieces (product List, order List, inventory Dashboard, inventory Chart).



7. Apex with LWC — Server Methods

- What we used:
 - Apex class: InventoryController
- Server methods:
 - `getProducts()` → fetches all `Product__c` records
 - `getOrders()` → fetches all `Order__c` records
- Read methods: marked `@AuraEnabled(cacheable=true)` → allows caching and efficient read-only access from LWCs
- Imperative method (example for updates): `updateStock()` → manually called from a button to update product stock

- How it's used in LWCs:
 - productList and orderList call getProducts() / getOrders() to display data
 - stockSummary calls getProducts() to calculate total stock and total number of products
- Why we use it:
 - Provides a secure server-side connection to Salesforce data
 - Lets LWCs work dynamically with real Salesforce records



8. Wire Adapters — Reactive Data Loading

- What we used:

@wire decorator in LWCs:

@wire(getProducts)

```

wiredProducts({ error, data }) {
  if (data) this.products = data;
  else if (error) this.error = error;
}

```

- Which components use it:
 - productList → lists all products automatically when the page loads

- stockSummary → calculates total stock and total products
- orderList → lists all orders automatically
- Why it's used:
 - Automatically fetches server-side data from Apex
 - Reactive → updates component UI whenever underlying data changes
 - Reduces need for manual refreshes or calls

```

force-app > main > default > lwc > stockSummary > JS stockSummary.js > ...
1  import { LightningElement, wire } from 'lwc';
2  import getProducts from '@salesforce/apex/InventoryController.getProducts';
3
4  export default class StockSummary extends LightningElement {
5      totalProducts = 0;
6      totalStock = 0;
7
8      @wire(getProducts)
9      wiredProducts({ error, data }) {
10         if (data) {
11             this.totalProducts = data.length;
12             this.totalStock = data.reduce((sum, product) => sum + product.Stock_Quantity__c, 0);
13         } else if (error) {
14             console.error('Error fetching products:', error);
15         }
16     }
17 }
18

```

9. Imperative Apex Calls — On-demand Actions

- What / Why:
 - Call Apex methods manually from an LWC when you need a specific action triggered by the user (like a button click).
 - Useful for update, insert, or delete operations.
 - Unlike @wire, this does not load automatically — you control when it runs.
- Current Project Use:
 - Your project does not currently have any buttons or user-triggered actions that update Salesforce records.

- All your LWCs (productList, orderList, stockSummary) use wire adapters to load data automatically.
- Therefore, imperative Apex calls are optional for now.

10. Events in LWC — Child → Parent Communication

- What / Why:
 - Allows a child LWC to send data or trigger actions in a parent LWC.
 - Useful when components are nested: child → parent.
- Current Project Use:
 - You don't have nested LWCs yet — all your LWCs are independent (productList, orderList, stockSummary, inventoryDashboard).
 - Therefore, child → parent events are not needed in your current setup.

This can be added later if you split components into smaller pieces (e.g., a single product item child component emits an event when selected).