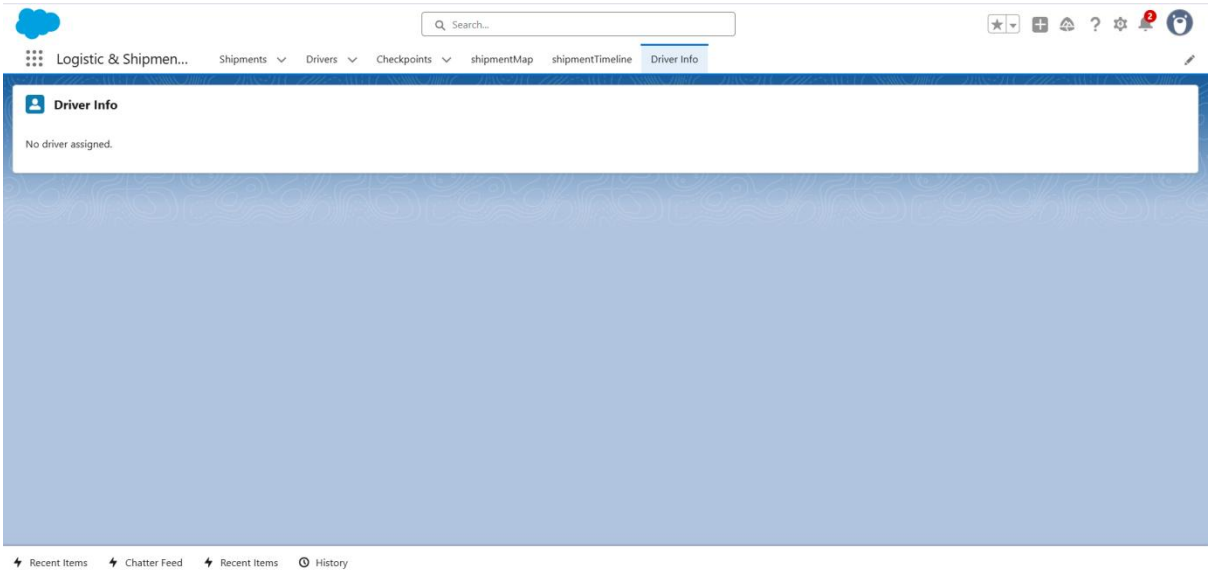
Phase 6: User Interface Development

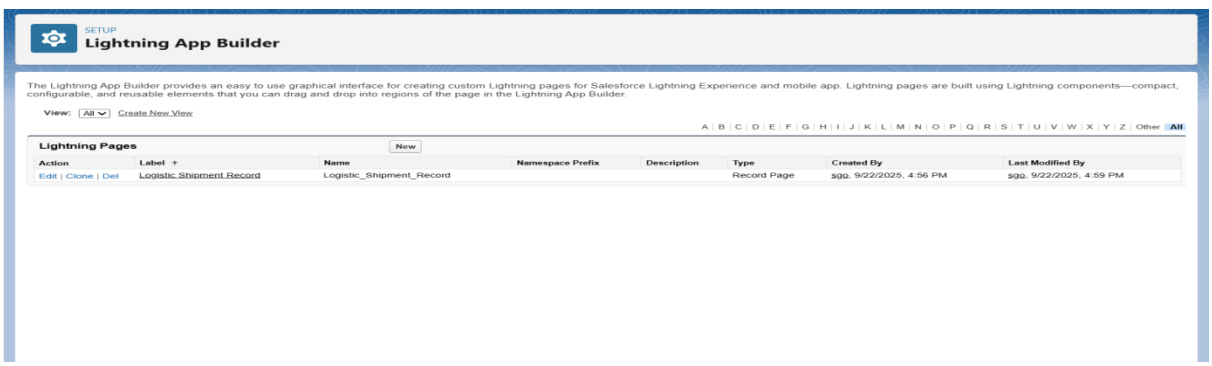
1. *Lightning App Builder*

- **Driver Info Tab:** Added a custom Lightning page to display driver assignment details for a shipment.
- **Dynamic Visibility:** Shows **"No driver assigned"** if no driver is linked to the shipment.
- **Purpose:** Provides users with quick, context-aware shipment + driver insights directly in the UI.



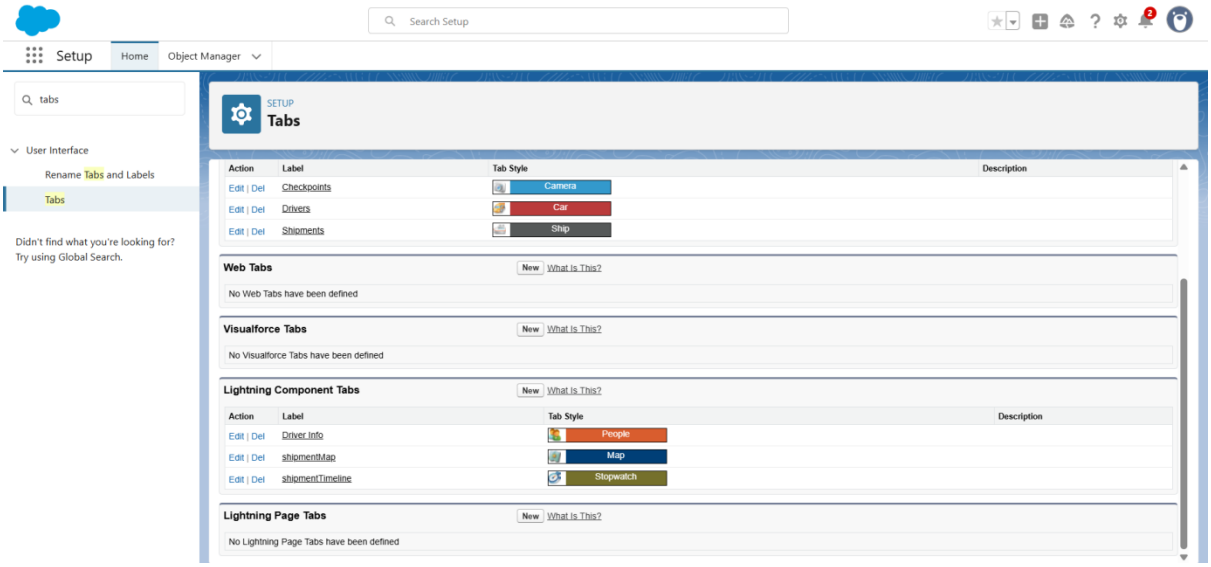
2. *Record Pages*

- **Created a custom Lightning Record Page (`Logistic_Shipment_Record`) in Lightning App Builder** to display shipment-related components in one unified layout.
- **Configured the page for extensibility**, enabling integration of dashboards and real-time components (e.g., shipment notifications, driver info) directly into the record view.



3. *Tabs*

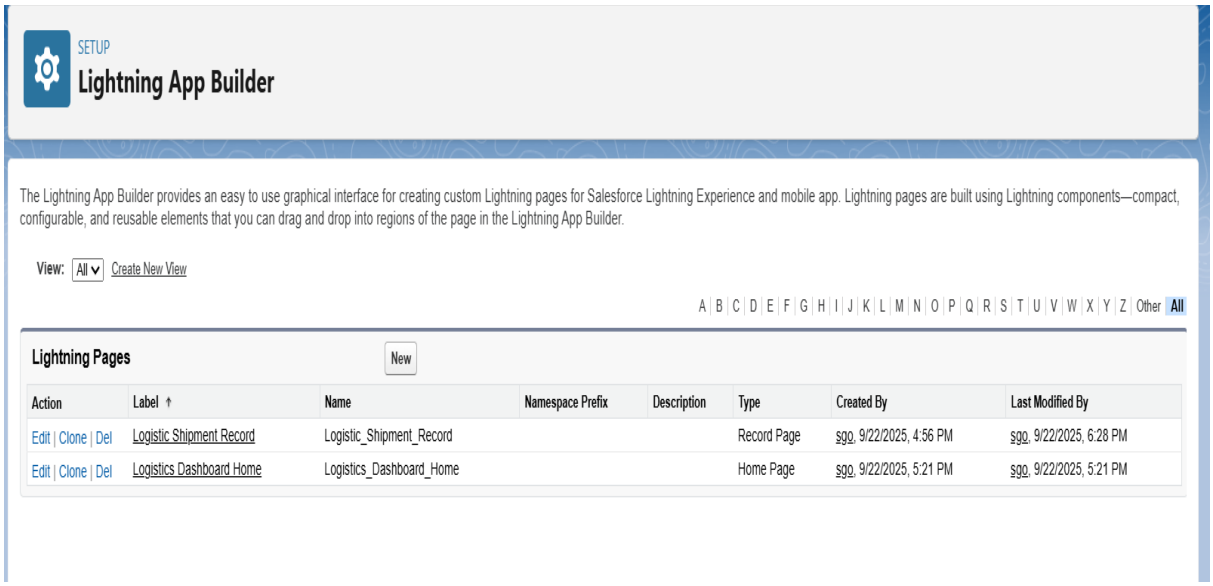
- **Created Custom Tabs** for `Checkpoints`, `Drivers`, and `Shipments` to allow easy navigation and access to respective Salesforce objects.
- • **Configured Lightning Component Tabs** like `Driver Info`, `Shipment Map`, and `Shipment Timeline` to display interactive, real-time data directly within the Lightning App.



4. *Home Page Layouts*

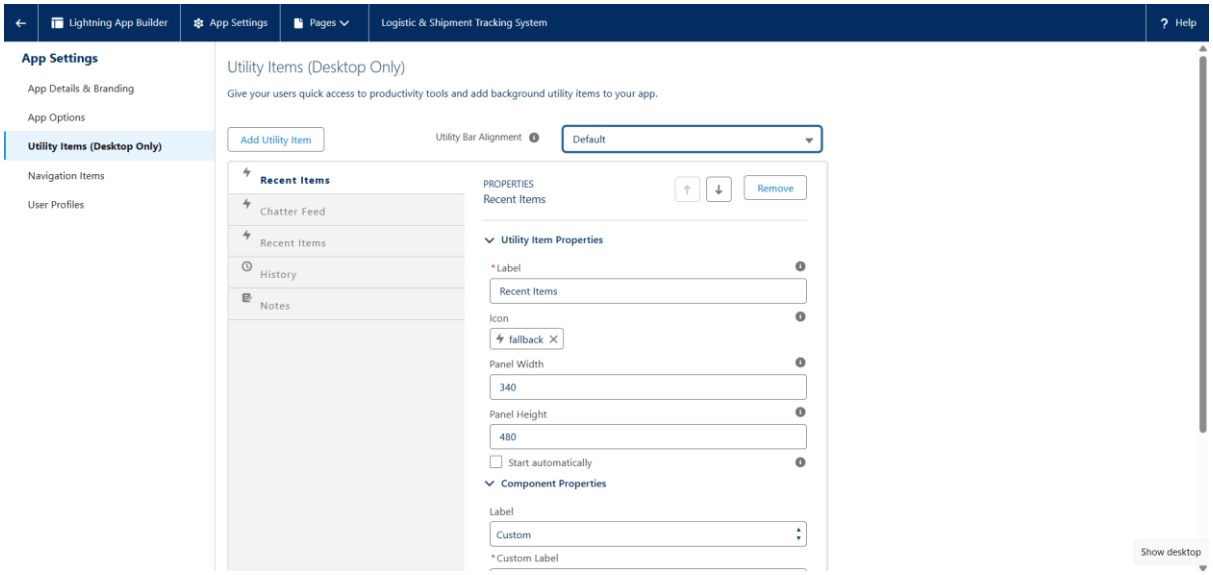
- **Created a custom Home Page (`Logistics Dashboard Home`)** using Lightning App Builder to provide a centralized view for logistics KPIs, dashboards, and recent records.

- • **Configured and activated the Home Page** so that users can directly access shipment insights, driver availability, and performance metrics upon login.



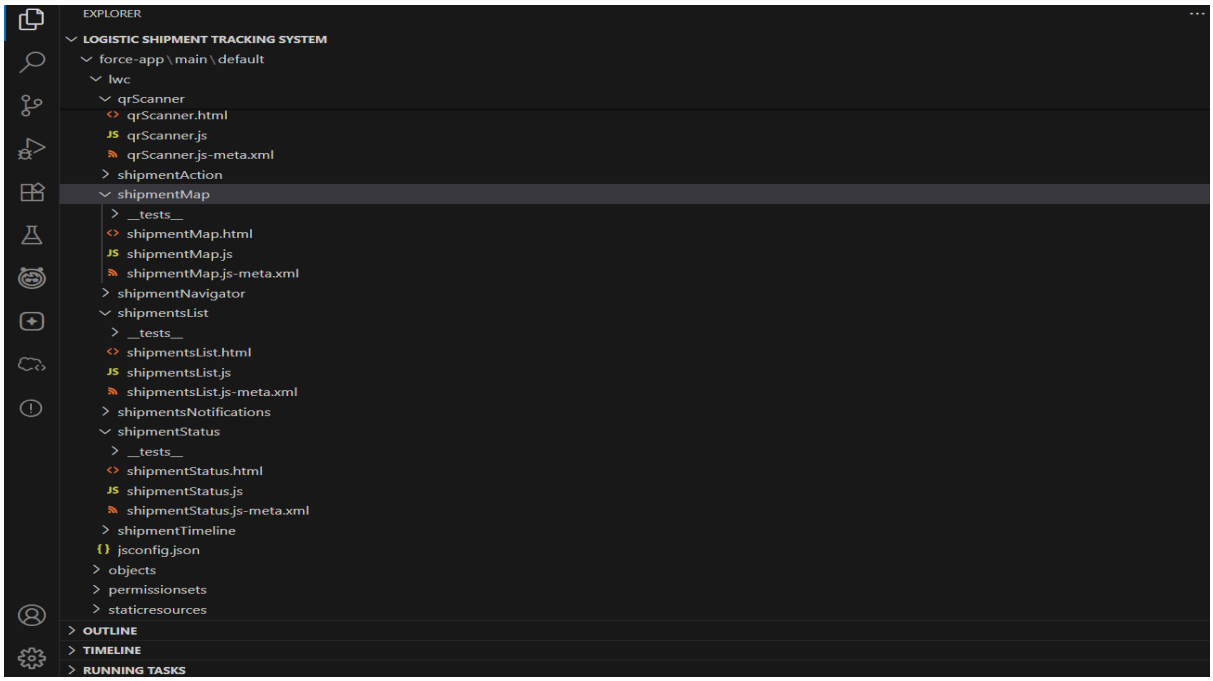
5. *Utility Bar*

- Configured **Utility Items** (Recent Items, Chatter Feed, History, Notes) for quick access within the app.
- Customized properties (panel width, height, labels) to optimize the user experience in the **Logistic & Shipment Tracking System App**.



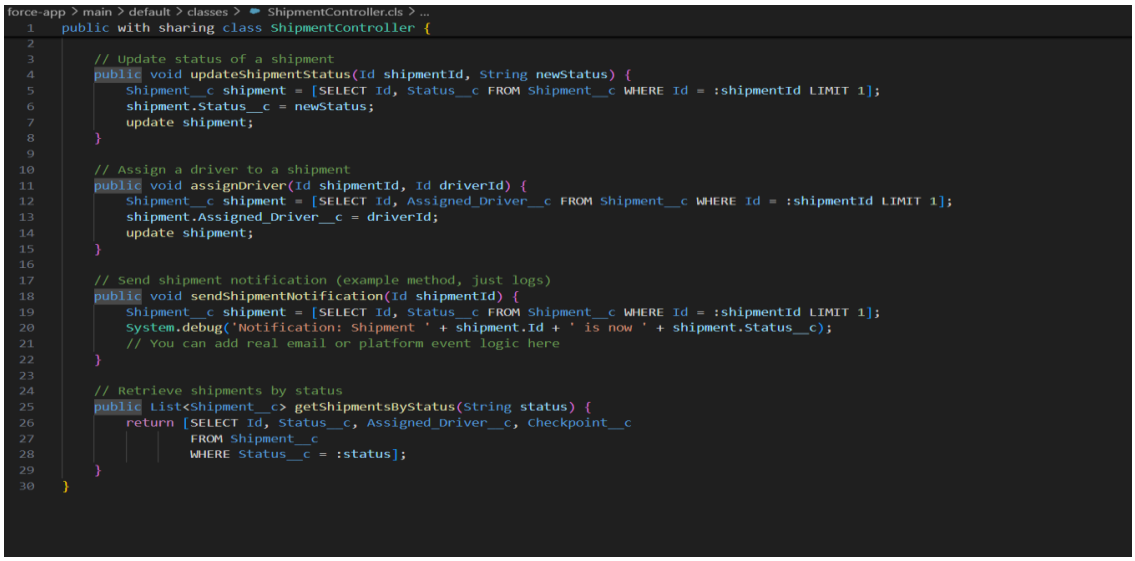
6. *LWC (Lightning Web Components)*

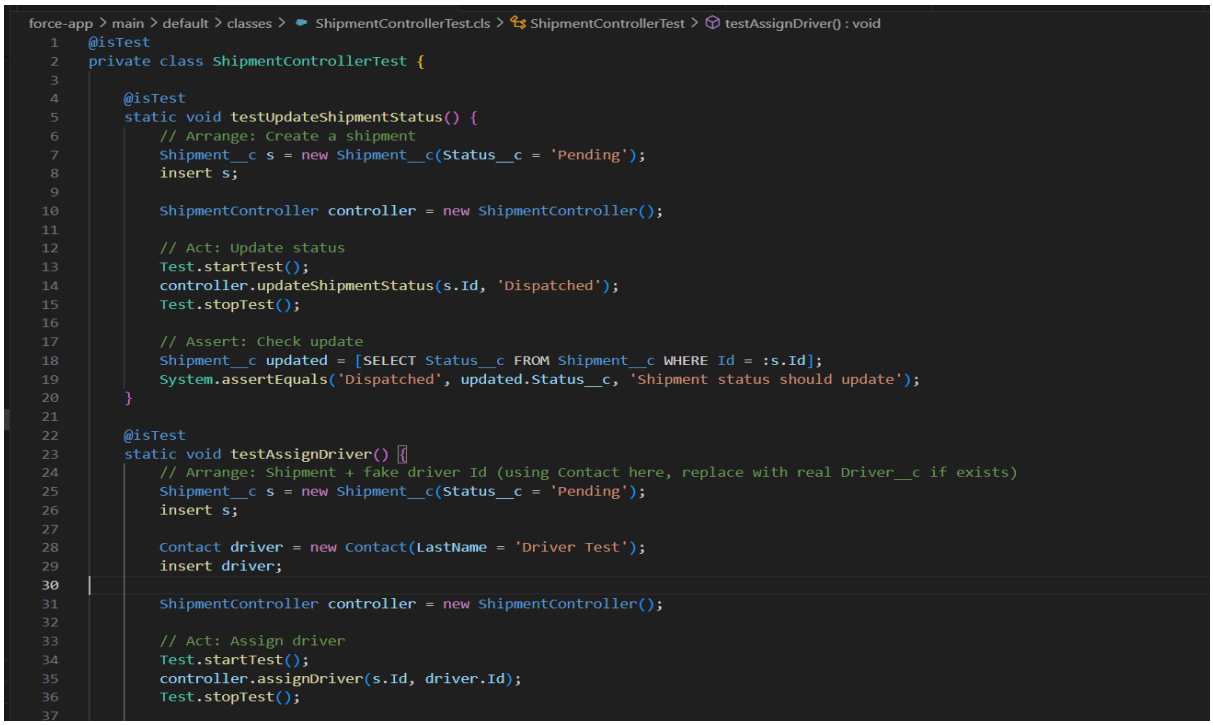
- Developed multiple **Lightning Web Components (LWCs)** such as `qrScanner`, `shipmentMap`, `shipmentNavigator`, `shipmentsList`, `shipmentsNotifications`, `shipmentStatus`, and `shipmentTimeline`.
- Each LWC is designed to handle a specific functionality (e.g., **QR code scanning, live shipment tracking, status updates, timeline visualization**) for the **Logistic Shipment Tracking System**.



7. *Apex with LWC*

- **Apex Controller with LWC :** The `ShipmentController.cls` Apex class exposes methods like `updateShipmentStatus`, `assignDriver`, `sendShipmentNotification`, and `getShipmentsByStatus`.These methods allow LWC components to interact with Salesforce data (e.g., update shipment status, assign drivers, or fetch shipments based on conditions).
- **LWC Components Using Apex** : Custom LWC components (`shipmentMap`, `shipmentStatus`, `shipmentTimeline`, etc.) call the Apex methods using `@AuraEnabled` methods.Example: The `shipmentStatus` LWC retrieves shipment data via `getShipmentsByStatus` and displays the live shipment progress in the UI, creating a seamless integration between backend (Apex) and frontend (LWC).

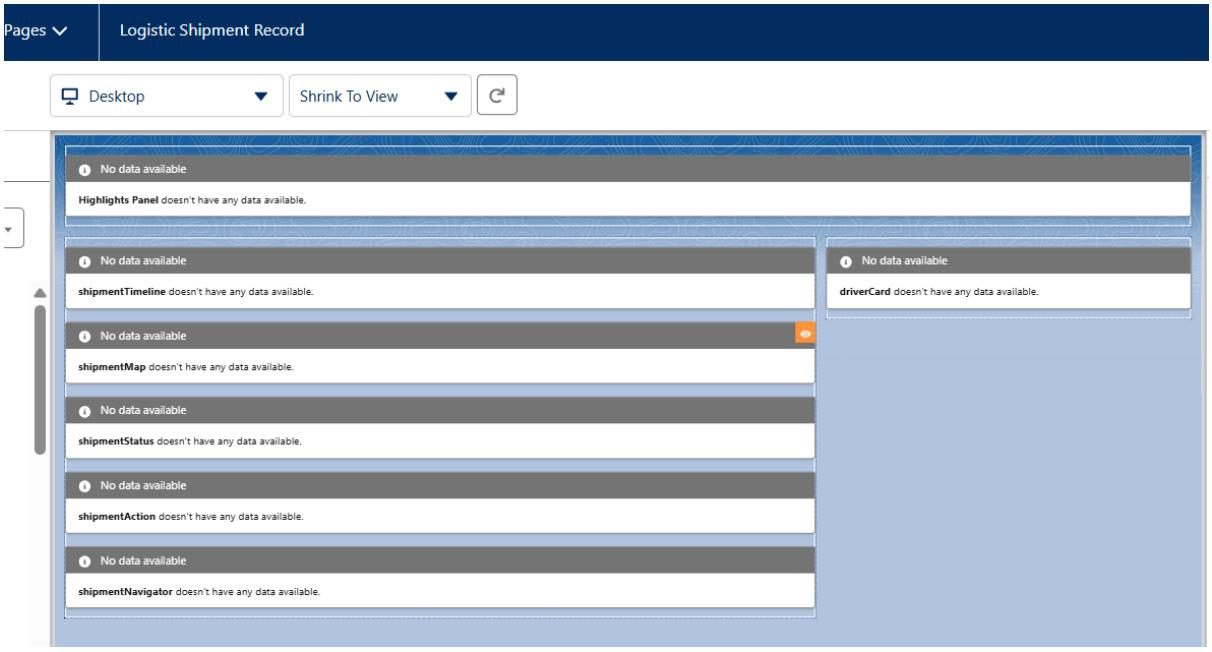




8. *Wire Adapters*

- **Use of Wire Adapters in LWC**
    - Lightning Web Components like `shipmentTimeline`, `shipmentMap`, `shipmentStatus`, and `driverCard` use **`@wire` adapters** to fetch real-time data from Salesforce objects without explicitly writing Apex calls.
    - Example: `@wire(getRecord, { recordId: '$recordId', fields: [...] })` retrieves live shipment or driver details directly from the database.

- **Integration in Lightning Page**
    - These LWC components are embedded into the **Logistic Shipment Record Page** through the Lightning App Builder.
    - When a shipment record is opened, wire adapters automatically fetch and refresh data such as status, assigned driver, checkpoints, and timeline — ensuring up-to-date tracking information.



9. *Imperative Apex Calls*

- **Manual Control of Apex Execution**
- In the `shipmentAction.js` component, the method `handleMarkDelivered()` calls an Apex method **imperatively** instead of using `@wire`.This allows more flexibility, as the call runs **only when triggered** (e.g., button click), not automatically.

- **Error Handling with Toast Messages**
- If the Apex call fails, a `ShowToastEvent` is dispatched to display a user-friendly error notification.Example shown in the screenshot:
- This improves user experience by giving real-time feedback when something goes wrong.

force-app > main > default > lwc > shipmentAction > shipmentAction.js > ...

```
export default class ShipmentAction extends LightningElement {
    handleMarkDelivered() {
            .then(() => {
                );
            })
            .catch(error => {
                // Error toast
                this.dispatchEvent(
                    new ShowToastEvent({
                        title: 'Error',
                        message: error.body ? error.body.message : error.message,
                        variant: 'error'
                    })
                );
            });
    }
}
```

force-app > main > default > lwc > shipmentAction > shipmentAction.html > ...

```
<template>
    <lightning-card title="Shipment Actions">
        <div class="slds-p-around_medium">
            <lightning-button
                label="Mark Delivered"
                onclick={handleMarkDelivered}>
            </lightning-button>
        </div>
    </lightning-card>
</template>
```

force-app > main > default > lwc > shipmentAction > shipmentAction.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>58.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
    </targets>
</LightningComponentBundle>
```

10. Navigation Service

- Exposed LWC to **Record Page, App Page, and Home Page** for flexible use.• Used **Navigation Service** to redirect users to records, lists, dashboards, and external pages.• Improved **user experience** with quick, context-based navigation.