Problem statement: “CPQ & Order Visibility Hub for Industrial Manufacturing”

Phase6 : User Interface Development

- Lightning App Builder

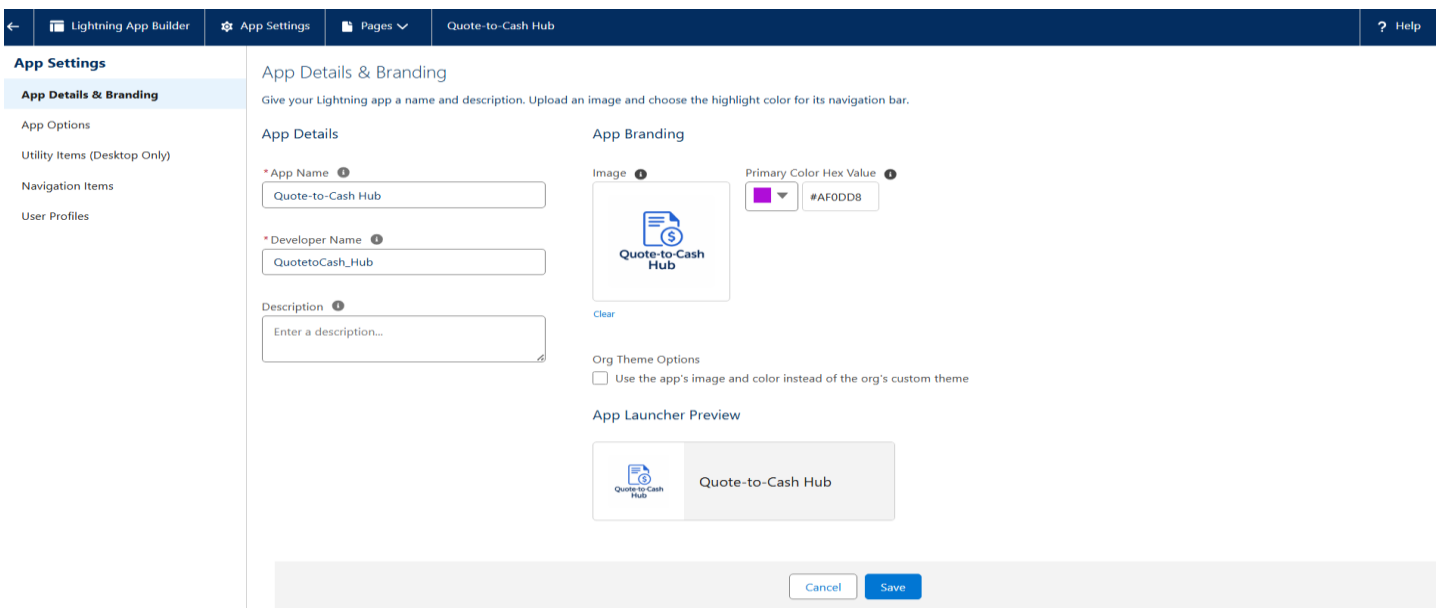
Setup → App Manager → New Lightning App.

Fill: App Name Quote-to-Cash Hub, developer name auto-fills.

Branding: upload logo and voilet app colour.

Choose Standard Navigation (unless you want console behaviour

Finish wizard and Save.

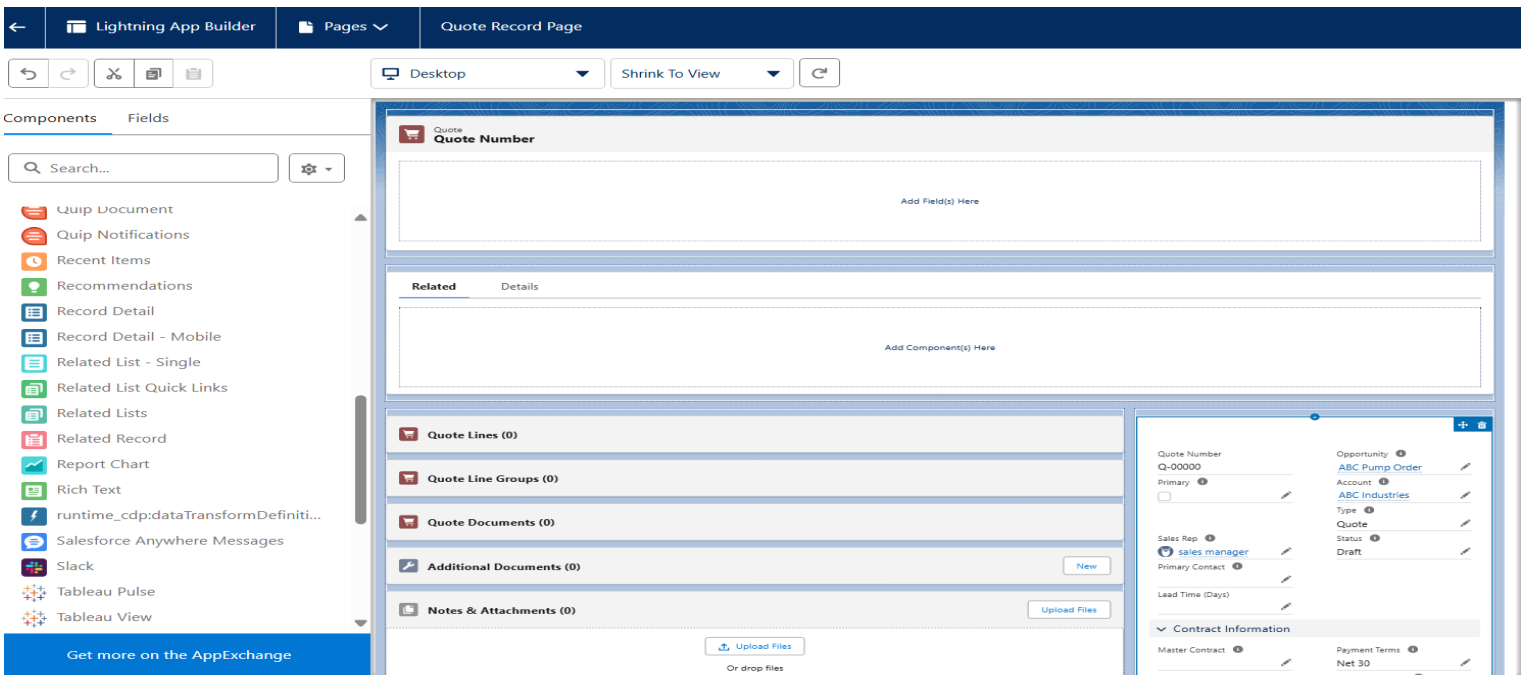


- **Record Pages**

We customize record pages so users see what’s important:

- Quote Record Page
  - Tabs: *Details, Quote Lines, Approvals*.
  - Embedded Flow: *Guided Selling Wizard*.

    - Related List: *Inventory availability*.
- Shipment Record Page
    - Header: *Tracking Number, Status, Carrier*.
    - Related List: *Order it belongs to*.
    - Component: Visual progress timeline (Created → In Production → Shipped).



◆ Tabs

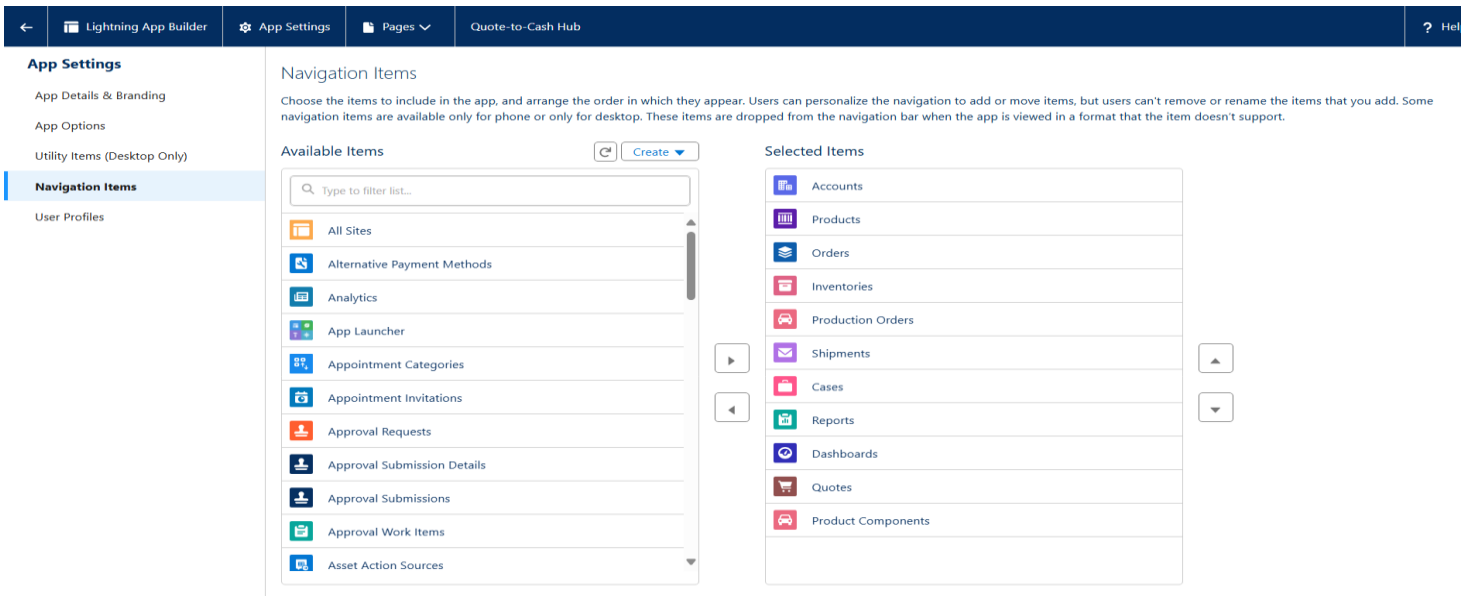
➢ From Available Items, selected tab that want to appear in my app:

➢ Tabs for **Quote-to-Cash Hub**:

- **Accounts** → Lists all customer accounts links to **Quotes, Orders, Shipments and Cases** for that account.
- **Products** → Catalog of products available for quoting and inventory; used in **Quotes and Orders**.
- **Orders** → Represents customer orders; links to **Quote, Shipment, Production Orders**.
- **Inventories** → Tracks stock levels of products; linked to **Quotes, Orders, and Production Orders**.
- **Production Orders** → Represents manufacturing orders for products; links to **Inventory and Orders**.
- **Shipments** → Tracks delivery of products to customers; linked to **Orders**.
- **Cases** → Customer service issues; linked to **Accounts and Shipments**.
- **Reports** → Shows analytics (pipeline, order status, stock levels); no direct payment object purely reporting.
- **Dashboards** → Visual display of reports and key metrics; can include custom LWCs like **Subscription Expiry Alerts**.
- **Quotes** → Represents customer quotes; links to **Accounts, Products, and Orders**.

- **Production Components** → Components/materials used in **Production Orders**; linked to **Inventory and Orders**.

Click Add → Moves it to Selected Items.



◆ Home Page Layouts

Go to Setup → Lightning App Builder → New.

Select Home Page.

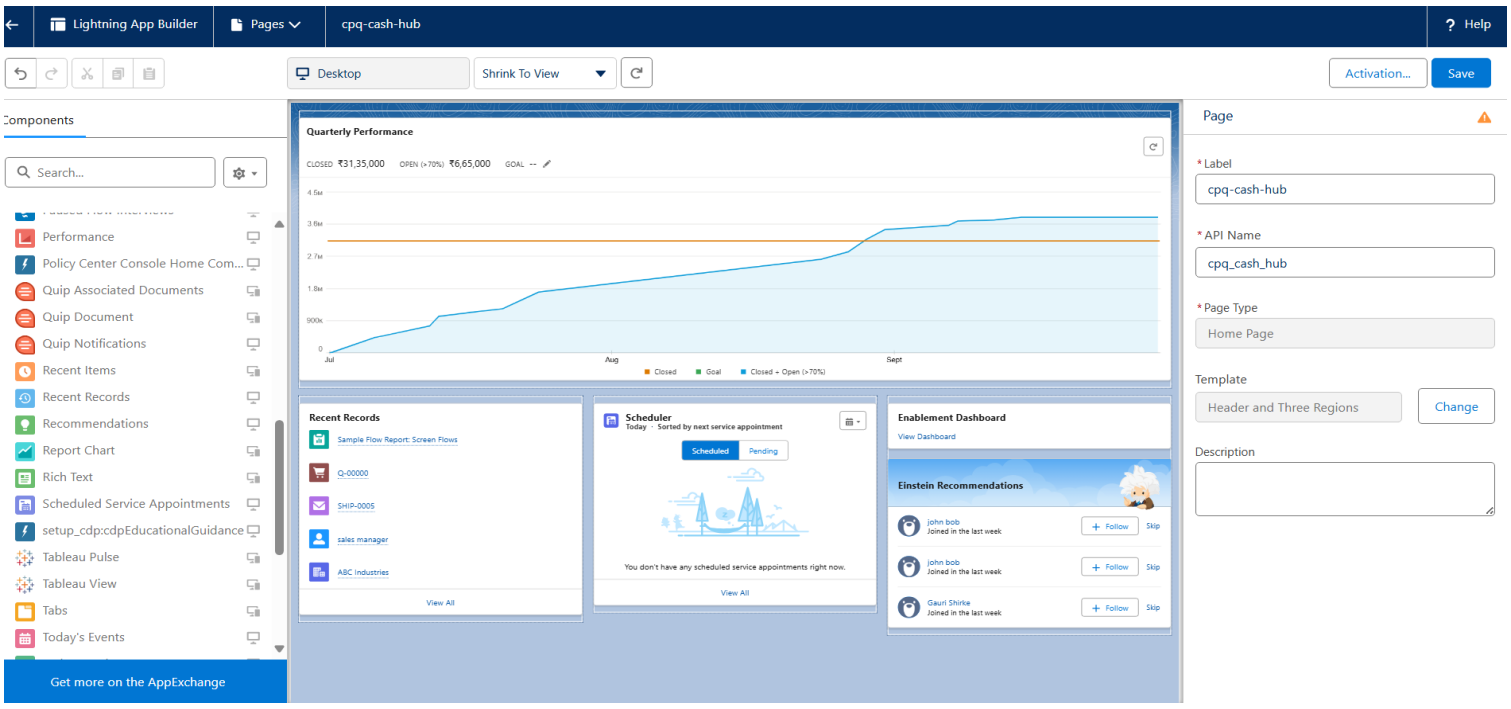
Pick a layout template Header and three region.

Name the page as cpq-cash-hub.

Add Components:

- **Scheduler** → Shows timelines and status.
- **Performance** → Displays account metrics and performance list.
- **Recent Items** → Quick access to recently viewed records (Quotes, Orders, Accounts.).
- **Dashboard** → Visual summary of key metrics; can include alerts like expiry.
- **Recommendation** → Suggests actions, e.g., nearing expiry or renewal opportunities.

- Save & Activate



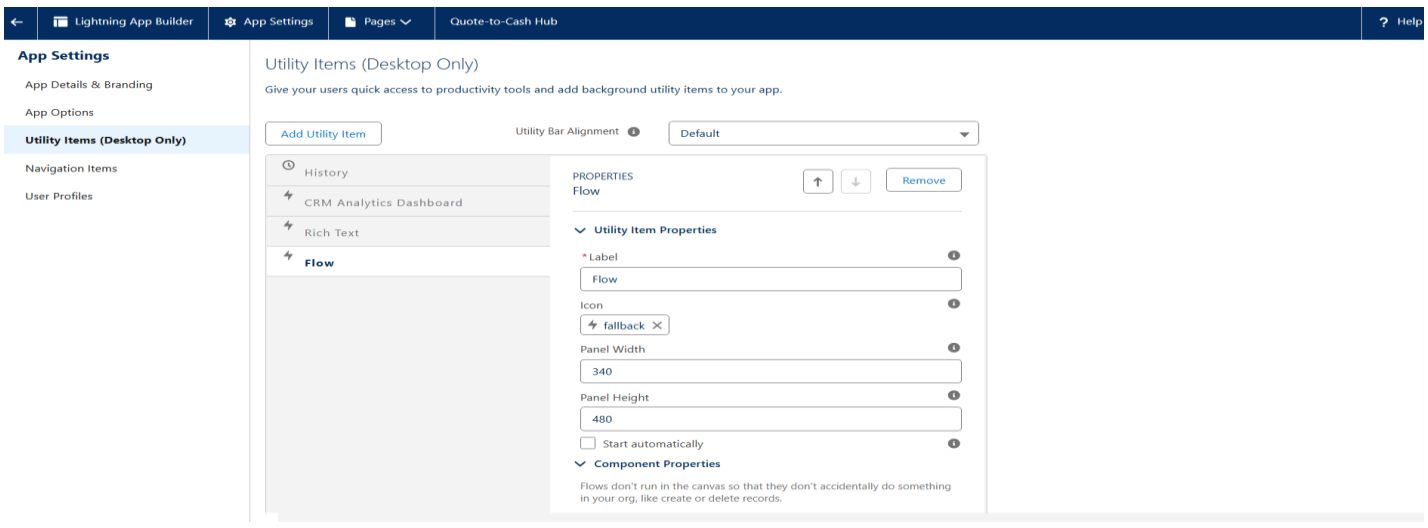
Utility Bar

1. Utility bar items appear in console apps as docked panels.

➢ Setup → App Manager → locate your **Quote-to-Cash Hub**.
➢ Go to **Utility Bar** step → **Add** items such as:

- **Utility Item**: **Flow** (Quick access to Guided Selling or Inventory Search Flow)
- **Utility Item**: **Rich Text** (Process cheat sheet)
- **Utility Item**: **History/Notes**

2. Configure labels, icons, width, panel type (docked). Save App.



LWC(Lightning Web Components)

➢ **Order Status Timeline LWC**

Goal: Show a **visual progress tracker** on the **Order & Shipment record page**.
Stages: *Quote Created → Approved → In Production → Shipped*.

HTML (orderStatusTimeline.html)

```
<template>
    <lightning-card title="Order Status Timeline">
        <div class="slds-p-around_medium slds-grid slds-grid_vertical-align-center">
            <lightning-progress-indicator current-step={status} type="path" variant="base">
                <lightning-progress-step label="Created" value="Created"></lightning-progress-step>
                <lightning-progress-step label="Approved" value="Approved"></lightning-progress-step>
                <lightning-progress-step label="In Production" value="In Production"></lightning-progress-step>
                <lightning-progress-step label="Shipped" value="Shipped"></lightning-progress-step>
            </lightning-progress-indicator>
        </div>
    </lightning-card>
</template>

```

JavaScript (orderStatusTimeline.js)

```
import { LightningElement, api, wire } from 'lwc';
import getOrderStatus from '@salesforce/apex/OrderStatusController.getOrderStatus';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';

export default class OrderStatusTimeline extends LightningElement {
    @api recordId;
    status;
    previousStatus;

    @wire(getOrderStatus, { orderId: '$recordId' })
    wiredOrder({ error, data }) {
        if (data) {
            // Detect status change
            if (this.previousStatus !== data) {
                this.status = data;

                // Fire toast when status becomes Shipped
                if (data === 'Shipped') {
                    this.showShippedToast();
                }

                this.previousStatus = data;
            }
        } else if (error) {
            this.status = 'Error fetching status';
        }
    }

    showShippedToast() {
        const event = new ShowToastEvent({
            title: 'Order Shipped ',
            message: 'This order has been successfully shipped!',
            variant: 'success',
        });
        this.dispatchEvent(event);
    }
}

```

Meta File (orderStatusTimeline.js-meta.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>59.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
    </targets>
    <targetConfigs>
        <targetConfig targets="lightning__RecordPage">
            <objects>
                <object>Order</object>
            </objects>
        </targetConfig>
    </targetConfigs>
</LightningComponentBundle>

```

- **Inventory Checker LWC**

Goal: On a **Quote Line**, show **live stock** from Inventory__c.

HTML (inventoryChecker.html)

```
<template>
    <lightning-card title="Inventory Checker">
        <template if:true={stockLevel}>
            <p>Available Stock: {stockLevel}</p>
        </template>
        <template if:false={stockLevel}>
            <p>No stock data found</p>
        </template>
    </lightning-card>
</template>

```

JS (inventoryChecker.js)

```
import { LightningElement, api, wire } from 'lwc';
import getStockLevel from '@salesforce/apex/InventoryCheckerController.getStockLevel';

export default class InventoryChecker extends LightningElement {
    @api recordId; // inventoryId passed from Quote Line
    stockLevel;

    @wire(getStockLevel, { inventoryId: '$recordId' })
    wiredStock({ error, data }) {
        if (data) {
            this.stockLevel = data;
        } else if (error) {
            this.stockLevel = null;
        }
    }
}

```

Meta File (inventoryChecker.js-meta.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>59.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
    </targets>
    <targetConfigs>
        <targetConfig targets="lightning__RecordPage">
            <objects>
                <object>SBQQ__QuoteLine__c</object>
            </objects>
        </targetConfig>
    </targetConfigs>
</LightningComponentBundle>

```

- Apex with LWC

InventoryCheckerController.cls

```
public with sharing class InventoryCheckerController {
    @AuraEnabled(cacheable=true)
    public static Decimal getStockLevel(Id inventoryId) {
        Inventory__c inv = [
            SELECT Stock_Level__c
            FROM Inventory__c
            WHERE Id = :inventoryId
            LIMIT 1
        ];
        return inv.Stock_Level__c;
    }
}

```

Apex controller OrderStatusController.cls

```apex
public with sharing class OrderStatusController {
    @AuraEnabled(cacheable=true)
    public static String getOrderStatus(Id orderId) {
        Order ord = [
            SELECT Status
            FROM Order
            WHERE Id = :orderId
            LIMIT 1
        ];
        return ord.Status;
    }
}

```

- Events in LWC

Step 1: Child Component → Fire Event

Create a new LWC productSelector.

HTML (productSelector.html)

```html
<template>
    <lightning-combobox
        name="product"
        label="Select Product"
        placeholder="Choose a product"
        options={options}
        onchange={handleSelect}>
    </lightning-combobox>
</template>

```

JS (productSelector.js)

```javascript
import { LightningElement } from 'lwc';

export default class ProductSelector extends LightningElement {
    handleSelect(event) {
        const selectedProduct = event.target.value;

        // Fire a custom event with productId
        const customEvent = new CustomEvent('productselected', {
            detail: { productId: selectedProduct }
        });
        this.dispatchEvent(customEvent);
    }
}

```

Meta File (productSelector.js-meta.xml)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>64.0</apiVersion>
    <isExposed>true</isExposed>
</LightningComponentBundle>
```

Step 2: Parent Component → Listen for Event

Create parent LWC guidedSelling

HTML (guidedSelling.html)

```html
<template>
    <c-product-selector onproductselected={handleProductSelected}></c-product-selector>

    <p if:true={recommendedSeal}>
        Recommended Seal: {recommendedSeal}
    </p>
</template>

```

JS (guidedSelling.js)

```js
import { LightningElement } from 'lwc';

export default class GuidedSelling extends LightningElement {
    recommendedSeal;

    handleProductSelected(event) {
        const productId = event.detail.productId;

        if (productId === 'PumpX') {
            this.recommendedSeal = 'Seal Type Y';
        } else {
            this.recommendedSeal = 'Standard Seal';
        }
    }
}

```

Meta File (guidedSelling.js-meta.xml)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>64.0</apiVersion>
    <isExposed>true</isExposed>
</LightningComponentBundle>
```

- Wire Adapters

Use Case in Project:
On the Shipment record page, show live Shipment Status + Expected Date

without writing Apex.

Step 1: Import Wire Adapter

Use lightning/uiRecordApi → built-in Salesforce adapter

HTML (shipmentStatus.html)

```
<template>
    <lightning-card title="Shipment Info">
        <p>Status: {status}</p>
        <p>Expected Date: {expectedDate}</p>
    </lightning-card>
</template>

```

JS (shipmentStatus.js)

```
import { LightningElement, api, wire } from 'lwc';
import { getRecord } from 'lightning/uiRecordApi';

const FIELDS = [
    'Shipment__c.Status__c',
    'Shipment__c.Expected_Date__c'
];

export default class ShipmentStatus extends LightningElement {
    @api recordId;    // Salesforce automatically passes the current record Id
    status;
    expectedDate;

    @wire(getRecord, { recordId: '$recordId', fields: FIELDS })
    wiredShipment({ error, data }) {
        if (data) {
            this.status = data.fields.Status__c.value;
            this.expectedDate = data.fields.Expected_Date__c.value;
        } else if (error) {
            this.status = 'Error';
        }
    }
}

```

Meta File (shipmentStatus.js-meta.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>64.0</apiVersion>
    <isExposed>true</isExposed>
</LightningComponentBundle>
```

- ## Imperative Apex Calls

  Why need it in project:

- Wire adapters are good for automatic data fetching, but sometimes you want on-demand calls (like pressing a button).
- Example: A Sales Rep wants to check ERP stock availability for a product before finalizing a Quote Line

Apex Class ERPStockController.cls

```
public with sharing class ERPStockController {
    @AuraEnabled
    public static String checkERPStock(Id inventoryId) {
        //  For now, just simulate ERP call
        Inventory__c inv = [
            SELECT Name, Stock_Level__c
            FROM Inventory__c
            WHERE Id = :inventoryId
            LIMIT 1
        ];

        // In real world → callout to ERP system here
        return 'ERP confirms stock for ' + inv.Name + ': ' + inv.Stock_Level
    }
}
```

HTML (erpStockChecker.html)

```
<template>
    <lightning-button label="Check ERP Stock" onclick={handleCheckStock}></lightning-button>
    <template if:true={erpMessage}>
        <p>{erpMessage}</p>
    </template>
</template>
```

JS (erpStockChecker.js)

```
import { LightningElement, api } from 'lwc';
import checkERPStock from '@salesforce/apex/ERPStockController.checkERPStock';

export default class ErpStockChecker extends LightningElement {
    @api recordId; // assume placed on Inventory record page
    erpMessage;

    handleCheckStock() {
        checkERPStock({ inventoryId: this.recordId })
            .then(result => {
                this.erpMessage = result;
            })
            .catch(error => {
                this.erpMessage = 'Error: ' + error.body.message;
            });
    }
}
```

Meta File (erpStockChecker.js-meta.xml)

```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>
    <targetConfigs>
        <targetConfig targets="lightning__RecordPage">
            <objects>
                <object>Inventory__c</object>
            </objects>
        </targetConfig>
    </targetConfigs>
</LightningComponentBundle>
```

Navigation Service

Why need it in the project:

- After automation (e.g., Production Planner creates a Production Order), we want to auto-navigate users to the new record page.
- Helps with smoother UX.
- Navigation Service → after record creation, Salesforce redirects them automatically to the new record page.

HTML (productionOrderCreator.html)

```html
<template>
    <lightning-card title="Create Production Order">
        <div class="slds-p-around_small">
            <lightning-button
                label="Create Production Order"
                variant="brand"
                onclick={handleCreateOrder}>
            </lightning-button>
        </div>
    </lightning-card>
</template>

```

JS (productionOrderCreator.js)

```javascript
import { LightningElement } from 'lwc';
import createProductionOrder from '@salesforce/apex/ProductionOrderController.createProductionOrder';
import { NavigationMixin } from 'lightning/navigation';

export default class ProductionOrderCreator extends NavigationMixin(LightningElement) {
    handleCreateOrder() {
        createProductionOrder()
            .then(result => {
                // Navigate to the Production Order record page
                this[NavigationMixin.Navigate]({
                    type: 'standard__recordPage',
                    attributes: {
                        recordId: result, // returned Production Order Id
                        objectApiName: 'Production_Order__c',
                        actionName: 'view'
                    }
                });
            })
            .catch(error => {
                console.error(error);
            });
    }
}

```

Meta File (productionOrderCreator.js-meta.xml)

```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>
        <target>lightning__AppPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>

```

Apex Class for Production Order

```
public with sharing class ProductionOrderController {
    @AuraEnabled
    public static Id createProductionOrder() {
        Production_Order__c po = new Production_Order__c(
            Name = 'Auto Created Order',
            Status__c = 'Planned',
            Scheduled_Date__c = Date.today().addDays(7)
        );
        insert po;
        return po.Id;
    }
}

```