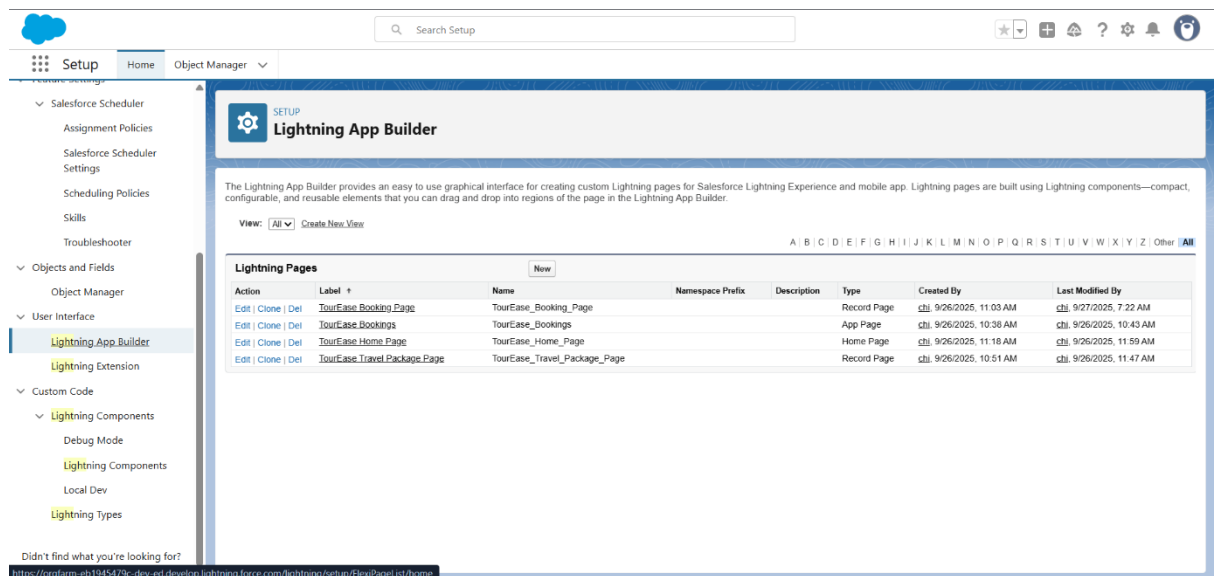


Phase 6: User Interface Development for TourEase

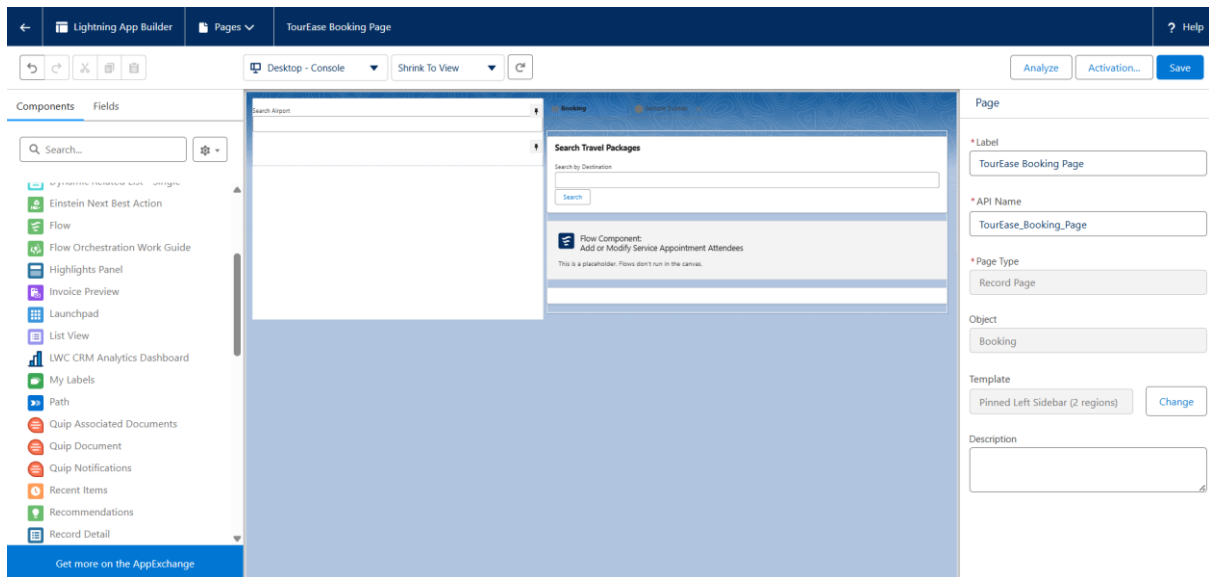
1. Lightning App Builder

- **Purpose:** To create custom apps, record pages, and home pages using drag-and-drop components.
- **Steps for your project:**
 1. Go to **Setup** → **Lightning App Builder**.
 2. Click **New** → **App Page / Record Page / Home Page**.
 3. Give it a name like TourEase Booking App Page.
 4. Drag standard or custom components (like Booking__c list, Contact info) onto the page layout.
 5. Save and activate for users or profiles.



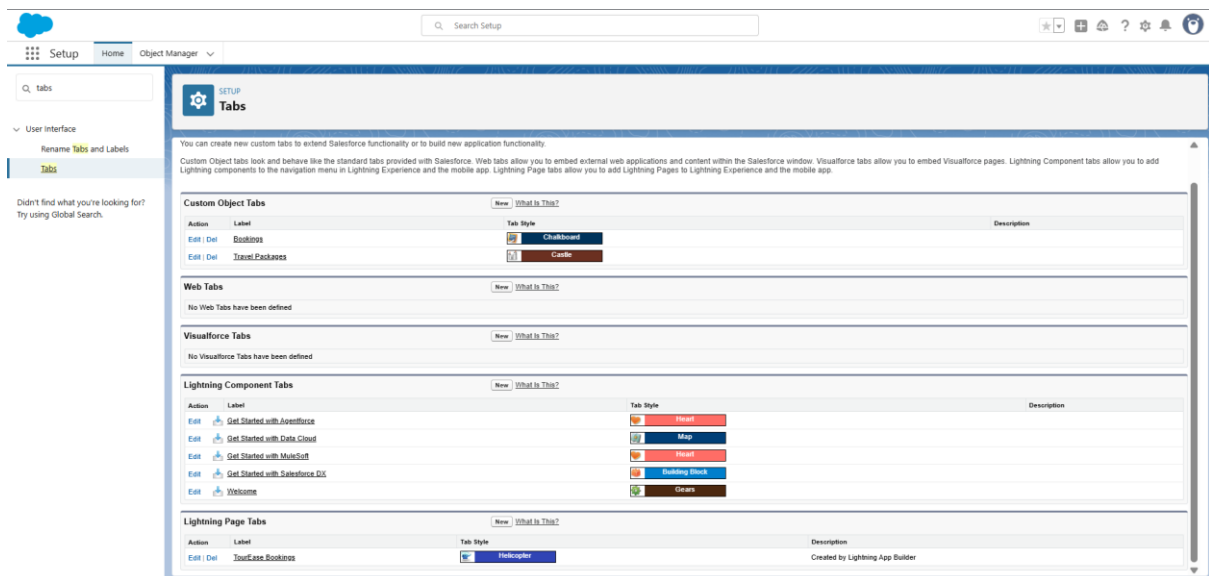
2. Record Pages

- **Purpose:** Customize the layout for a specific object (Booking, Contact, Travel Package).
- **Steps:**
 1. In **Lightning App Builder**, choose **Record Page** → **Booking__c**.
 2. Edit the page:
 - Add **Related Lists**, **Details**, **Custom LWC components**.
 3. Save and activate.



3. Tabs

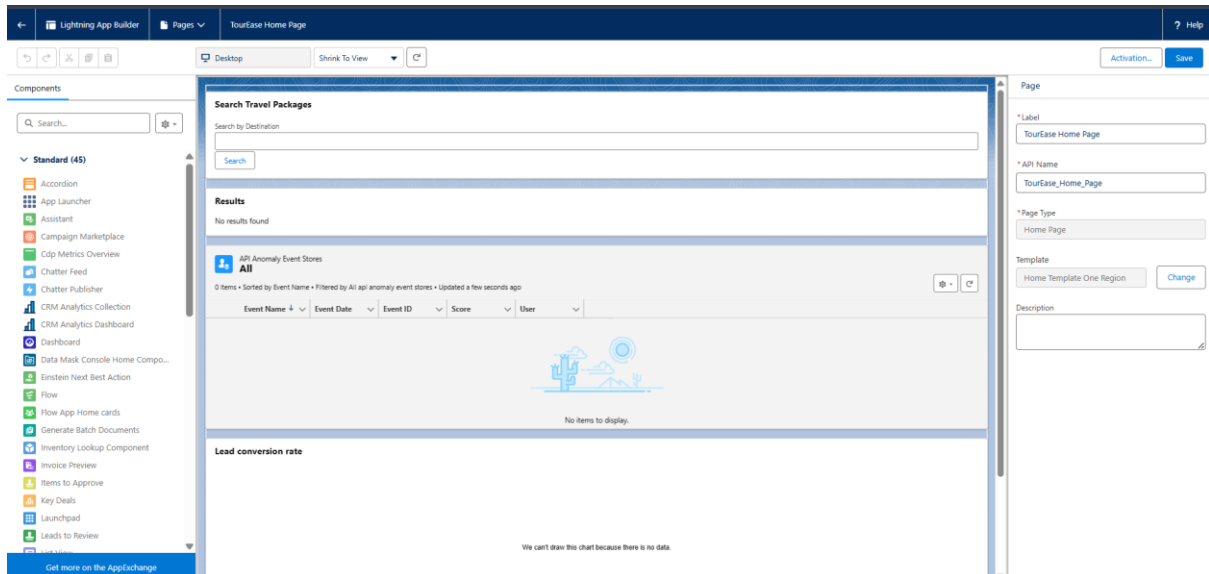
- **Purpose:** Provide quick navigation to objects, apps, or web pages inside Salesforce.
- **Steps:**
 1. Go to **Setup** → **Tabs**.
 2. Click **New** to create a **Custom Object Tab** for Booking, Contact, Travel Package.
 3. Assign tab style and save.



4. Home Page Layouts

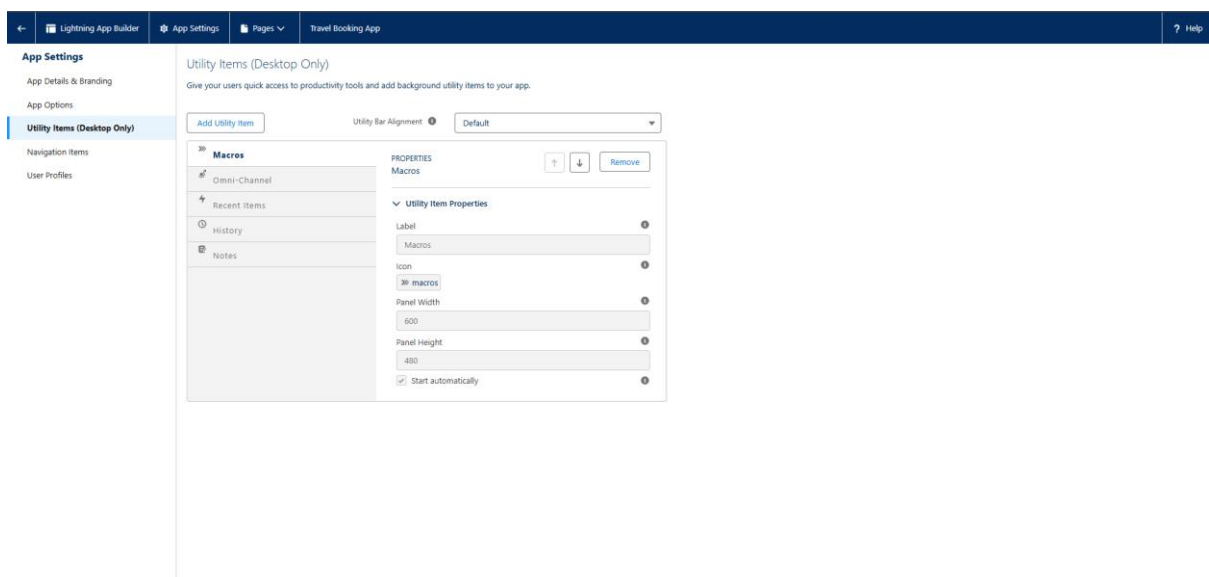
- **Purpose:** Customize the Salesforce Home page for users.
- **Steps:**

1. Go to **Setup** → **Lightning App Builder** → **Home Page**.
2. Drag components like **Reports**, **Tasks**, **TourEase Summary**.
3. Save and activate for user profiles.



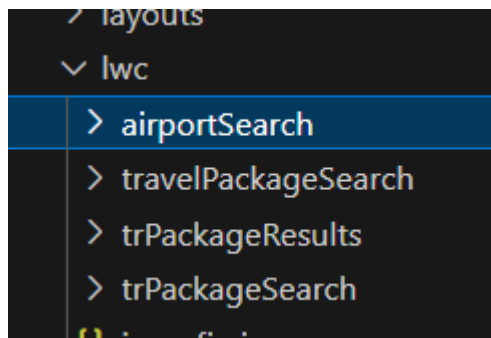
5. Utility Bar

- **Purpose:** Provides persistent tools (like chat, calculator, quick actions) at the bottom of any app.
- **Steps:**
 1. Go to **App Manager** → **Edit the App** → **Utility Bar**.
 2. Add utilities like **Macros**, **History**, **Reports**.
 3. Save and activate.



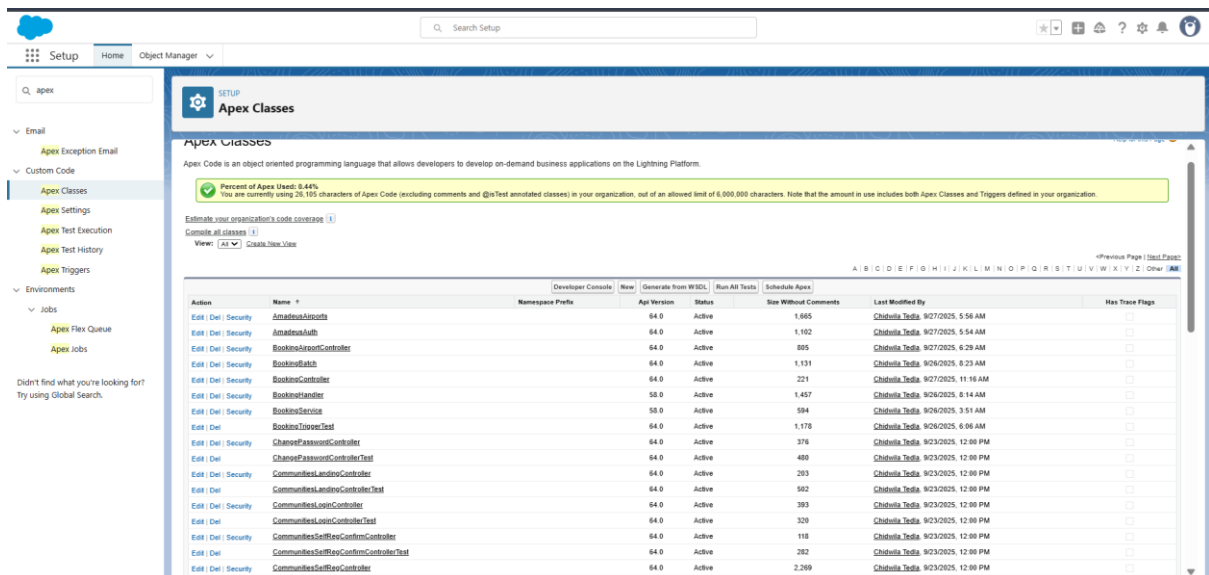
6. LWC (Lightning Web Components)

- **Purpose:** Build modern, reusable components in Salesforce using JavaScript, HTML, CSS.
- **Steps:**
 1. Use **VS Code** with **Salesforce Extension Pack**.
 2. Create LWC:
 3. SFDX: Create Lightning Web Component → Name: BookingList
 4. Deploy to Salesforce.



7. Apex with LWC

- **Purpose:** Fetch or manipulate Salesforce data using server-side Apex methods for LWCs.



Action	Name	Namespace Prefix	API Version	Status	Size Without Comments	Last Modified By	Has Trace Flags
Edit Del Security	AmadeusAirport		64.0	Active	1,665	Chidella, Tedia	
Edit Del Security	AmadeusAuth		64.0	Active	1,162	Chidella, Tedia	
Edit Del Security	AmadeusAirportController		64.0	Active	895	Chidella, Tedia	
Edit Del Security	BookingsBatch		64.0	Active	1,131	Chidella, Tedia	
Edit Del Security	BookingsController		64.0	Active	221	Chidella, Tedia	
Edit Del Security	BookingsHandler		58.0	Active	1,457	Chidella, Tedia	
Edit Del Security	BookingsService		58.0	Active	594	Chidella, Tedia	
Edit Del	BookingsTriggerTest		64.0	Active	1,178	Chidella, Tedia	
Edit Del Security	ChannelPassportController		64.0	Active	376	Chidella, Tedia	
Edit Del	ChannelPassportControllerTest		64.0	Active	480	Chidella, Tedia	
Edit Del Security	CommunicationsController		64.0	Active	293	Chidella, Tedia	
Edit Del	CommunicationsControllerTest		64.0	Active	582	Chidella, Tedia	
Edit Del Security	CommunicationsController		64.0	Active	393	Chidella, Tedia	
Edit Del	CommunicationsControllerTest		64.0	Active	320	Chidella, Tedia	
Edit Del Security	CommunicationsController		64.0	Active	118	Chidella, Tedia	
Edit Del	CommunicationsControllerTest		64.0	Active	282	Chidella, Tedia	
Edit Del Security	CommunicationsController		64.0	Active	2,269	Chidella, Tedia	

8. Events in LWC

- **Purpose:** Communication between LWCs.
 - **Child** → **Parent:** Custom events.

- **Sibling components:** Use **pubsub module**.
 - **Steps:**
 1. Create custom event in child LWC:
 2. `const selectedEvent = new CustomEvent('selected', { detail: bookingId });`
 3. `this.dispatchEvent(selectedEvent);`
 4. Parent listens using:
 5. `<c-child-component onselected={handleSelected}></c-child-component>`
-

9. Wire Adapters

- **Purpose:** Automatically fetch Salesforce data and reactively bind it to LWC.
 - **Example:**
 - `import { LightningElement, wire } from 'lwc';`
 - `import getBookings from '@salesforce/apex/BookingController.getBookings';`
 -
 - `export default class BookingList extends LightningElement {`
 - `@wire(getBookings) bookings;`
 - `}`
-

10. Imperative Apex Calls

- **Purpose:** Call Apex methods on-demand (e.g., button click) instead of reactive binding.
 - **Example:**
 - `import getBookings from '@salesforce/apex/BookingController.getBookings';`
 - `handleClick() {`
 - `getBookings().then(result => { this.bookings = result; });`
 - `}`
-

11. Navigation Service

- **Purpose:** Programmatically navigate users to records, URLs, or pages.
- **Steps in LWC:**
 - `import { NavigationMixin } from 'lightning/navigation';`
 -

- export default class NavigateBooking extends NavigationMixin(LightningElement) {
 - navigateToBooking() {
 - this[NavigationMixin.Navigate]({
 - type: 'standard__recordPage',
 - attributes: {
 - recordId: 'a0Bxxxxxxxxxxxxx',
 - objectApiName: 'Booking__c',
 - actionName: 'view'
 - }
 - });
 - }
 - }
-

Conclusion:

In Phase 6, the TourEase project focused on building a user-friendly and interactive Salesforce interface for seamless travel booking and management, utilizing Lightning App Builder to create custom record pages, home pages, and utility bars for different user profiles. The integration of Lightning Web Components (LWCs) with Apex enabled efficient fetching and display of dynamic data, while wire adapters and imperative calls supported reactive and on-demand interactions. By implementing events and navigation services, components communicated smoothly and navigation became intuitive, enhancing overall user experience. This combination of declarative and programmatic tools ensured both admins and end-users could interact with the system effortlessly, demonstrating that a well-structured Salesforce UI can significantly improve productivity, data accessibility, and usability in a real-world travel booking application like TourEase.