

BudgetBuddy: A CRM Application for Managing Personal and Business Expenses

By

Sravani Samayam

samayam.sravani912@gmail.com

Malla Reddy University, Hyderabad

INDEX PAGE

SNO	Topic	Page No
1	Project Overview	4
2	Objectives	5
3	Salesforce Key Features and Concepts Utilized	6 - 7
4	Detailed Steps to Solution Design <ul style="list-style-type: none">● 4.1 Creating the Custom Object: "Expense"● 4.2 Importing Data Using Data Import Wizard● 4.3 Navigating to the Expenses Tab● 4.4 Cloning the LWC Recipes Repository● 4.5 Create Salesforce Project in Vs Code● 4.6 Locating Required Components	8 - 37

	<ul style="list-style-type: none"> ● 4.7 Deploying Components ● 4.8 Renaming Components ● 4.9 Configuring the Flow to Use the Component ● 4.10 Creating the Apex Controller and LWC for BudgetBuddy ● 4.11 Displaying the Data ● 4.12 Create the BudgetBuddy App ● 4.13 Create a lightning app page ● Output Screens 	
5	Testing and Validation	38 - 39
6	Key Scenarios Addressed by Salesforce in the Implementation Project	40 - 41
7	Conclusion	42

1. Project Overview

With personal and business expenses growing more complex, managing finances effectively has become essential for better decision-making. BudgetBuddy is a comprehensive Customer Relationship Management (CRM) application designed to fulfill this need by providing Salesforce users with a reliable CRM solution that brings together data, analytics, and visualization in one place. By automating tracking and offering clear insights into spending habits, this project aims to:

- **Empower users with data-driven insights:** Visual breakdowns of expenses help users make informed financial choices.
- **Improve financial transparency:** The project enables users to identify spending patterns and areas for cost savings.
- **Streamline financial processes within Salesforce:** Integrating expense management into Salesforce minimizes the need for external tools, making BudgetBuddy a seamless, all-in-one solution for users.

The aim is to build a comprehensive expense management tool within the Salesforce platform. Our goal is to develop an intuitive and robust application that helps users track, categorize, and analyze their expenses to gain a clear understanding of their financial health.

This Project uses an existing real-time dataset which is in CSV format i.e Expense_mock_Data.

Based on the data in the CSV file, the project will take the startDate and endDate as input from the user and then it displays the pie chart with expense_type in the Date range.

2. Objectives

1. Date-based Expense Filtering:

- Implement functionality to filter and view expenses based on user-defined date ranges.
- Provide users with control over specific time frames for more targeted financial insights.

2. Visual Analytics:

- Utilize Chart.js to present expense data visually through charts, such as doughnut and bar charts.
- Offer graphical representations of spending by category to enhance data comprehension and user decision-making.

3. Data Accuracy and Real-time Updates:

- Ensure real-time data retrieval and calculations with Salesforce's Lightning framework and Apex.
- Achieve data accuracy and integrity through validations and seamless synchronization of records.

4. Scalability and Integration:

- Develop the application to support future scalability, allowing additional features or integration with other Salesforce modules.
- Ensure compatibility with different Salesforce pages (e.g., App Page, Record Page, Home Page) for easy deployment across various use cases.

Business Goals: Enhance financial awareness and decision-making for users through a streamlined, accessible expense management solution.

Specific Outcomes:

- Simplified expense tracking with detailed categorization
- Dynamic dashboards displaying financial insights

3. Salesforce Key Features and Concepts Utilized

1. Lightning Web Components (LWC):

- Developed the front-end user interface using LWC, enabling a responsive, dynamic, and modular design.
- Leveraged LWC's modern JavaScript framework for efficient client-server communication and seamless user interactions.
- **LWC Recipes** repository is used to provide reliable, pre-built examples and best practices for building Lightning Web Components (LWC). It helps accelerate development, ensures code quality by following Salesforce standards, and serves as a reference for handling data, events, and component communication effectively within the project.

2. Apex Controllers:

- Utilized Apex classes for server-side processing, including querying, aggregating, and filtering expense data based on specified criteria.
- Ensure data security with Apex's "with sharing" keyword to respect user permissions on records.

3. SOQL (Salesforce Object Query Language):

- Used SOQL queries within Apex to retrieve and aggregate expense data, grouping by categories for detailed reporting.
- Applied date filtering in SOQL queries to allow users to view expenses within custom date ranges.

4. Platform Resource Loader:

- Integrated external libraries, such as Chart.js, by loading them via Salesforce's platformResourceLoader.
- Enabled enhanced visualization capabilities within the LWC by using Chart.js for graphical representation of expense data.

5. Chart.js Integration:

- Added rich visual analytics with doughnut and bar charts to present expenses categorized by type.
- Dynamically updated charts based on user input and data retrieved from Salesforce records, improving data interpretation.

6. Custom Metadata Types and Settings:

- Managed settings and configurations to define expense categories, default colors, and other project-specific parameters.
- Allowed flexibility in adding or updating categories without code changes, enhancing the app's adaptability.

7. Lightning App Builder and Community Builder Integration:

- Configured components to be deployable across different Salesforce page types, such as App Page, Record Page, Home Page, and Community Pages.
- Facilitated versatile use cases, allowing BudgetBuddy to function in different Salesforce environments based on user needs.

8. Lightning Flow Integration:

- Integrated with Salesforce Flow to enable flexible input of parameters (e.g., date range selection) for automated processes.
- Allowed the component to be part of complex workflows, enabling seamless interactions across various Salesforce automation tools.

4. Detailed Steps to Solution Design

4.1 Creating the Custom Object: "Expense"

- Object Name: Expense
- Record Name: Expense ID (Auto-numbered)
- Fields:
 - **Expense Type (Picklist):** Defines categories like Travel, Education, Insurance, Utilities, Rent, Entertainment, Miscellaneous, and Food.
 - **Expense Date (Date):** Specifies the date of the expense.
 - **Expense Amount (Currency):** Records the amount spent

4.2 Importing Data Using Data Import Wizard

1. Navigate to the Data Import Wizard in Salesforce Setup.
2. Click on Custom Object
3. Choose the Expense object for data import.
4. Click on Add new Records
5. Map data fields from the CSV file
(<https://drive.google.com/file/d/1wn2vLok6VwszhZQ0hLoF-nTPnO-fKvMe/view?usp=sharing>) to Salesforce fields, ensuring fields like Expense Type, Expense Date, and Expense Amount are properly aligned.
6. Click on Next twice.
7. Click on start import and complete the import process and verify data.

Setup

Home

Object Manager

SETUP > OBJECT MANAGER

Expense

Details

Fields & Relationships

Page Layouts

Lightning Record Pages

Buttons, Links, and Actions

Compact Layouts

Field Sets

Object Limits

Record Types

Related Lookup Filters

Restriction Rules

Scoping Rules

Object Access

Triggers

Flow Triggers

Validation Rules

Fields & Relationships

Quick Find

New Deleted Fields Field Dependencies Set History Tracking

7 Items, Sorted by Field Label

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Created By	CreatedById	Lookup(User)		
Expense Amount	Expense__c	Currency(16, 2)		
Expense Date	Expense_Date__c	Date		
Expense ID	Name	Auto Number		✓
Expense Type	Expense_Type__c	Picklist		
Last Modified By	LastModifiedById	Lookup(User)		
Owner	OwnerId	Lookup(User,Group)		✓

Setup

Home

Object Manager

Getting closer...

Choose data Edit mapping Start import

Import your Data into Salesforce

You can import up to 50,000 records at a time.

Help for this page

What kind of data are you importing?

Standard objects

Custom objects

Book Line Items

Book Orders

Books

Expenses

FlowPersonalConfigurations

FlowTableViewDefinitions

AppLogs

What do you want to do?

Add new records

Match by:

--None--

Which User field in your file designates record owners?

--None--

Trigger workflow rules and processes?

☐ Trigger workflow rules and processes for new and updated records

Update existing records

Add new and update existing records

Where is your data located?

Drag CSV file here to upload

CSV

File

Choose File EXPENSE_...K_DATA.csv

Character Code

ISO-8859-1 (General US & Western European, ISO-LATIN-1)

Values Separated By

Comma

Cancel Previous Next

Setup

Home

Object Manager

Search Setup

Almost done

Choose data

Edit mapping

Start import

Edit Field Mapping: Expenses

Your file has been auto-mapped to existing Salesforce fields, but you can edit the mappings if you wish. Unmapped fields will not be imported.

Help for this page

Edit	Mapped Salesforce Object	CSV Header	Example	Example	Example
Change	Expense Type	expense_type	Rent	Miscellaneous	Entertainment
Change	Expense Date	expense_date	2018-01-01	2018-01-04	2018-01-07
Change	Expense Amount	expense_amount	100	200	300

CancelPreviousNext

Setup

Home

Object Manager

Search Setup

data impo

Integrations

Data Import Wizard

Didn't find what you're looking for?
Try using Global Search.

SETUP

Bulk Data Load Jobs

Help for this Page

Bulk Data Load Job

750NS000003AqgL

View the details of a bulk data load job.

Back to List: Bulk Data Load Jobs

Bulk Data Load Job Detail

Reload

Job ID	750NS000003AqgL	Job Type	Bulk V1	Status	Closed
Submitted By	Stravani Samayam	Operation	Insert	Total Processing Time (ms)	902
Start Time	26/10/2024, 10:46 am IST	Queued Batches	0	API Active Processing Time (ms)	458
End Time	26/10/2024, 10:46 am IST	In Progress Batches	0	Apex Processing Time (ms)	0
Time to Complete (hh:mm:ss)	00:01	Completed Batches	1		
Object	Expense	Failed Batches	0		
External ID Field		Progress	100%		
Content Type	CSV	Records Processed	1000		
Concurrency Mode	Parallel	Records Failed	129		
API Version	62.0	Retries	0		

Reload

Batches

View Request	View Result	Batch ID	Start Time	End Time	Total Processing Time (ms)	API Active Processing Time (ms)	Apex Processing Time (ms)	Records Processed	Records Failed	Retry Count	State Message	Status
View Request	View Result	751NS0000026Cjh	26/10/2024, 10:46 am	26/10/2024, 10:46 am	902	458	0	1,000	129	0		Completed

4.3 Navigating to the Expenses Tab

- Go to the Expenses tab within your org.
- Select All Expenses to view all existing expenses data.
- Click on the Gear and click on “Select Fields to Display”
- Add Columns: Display the following columns by adding them from the display options:
 - Expense Type
 - Expense Date
 - Expense Amount
- Click Save to confirm these column additions.

The screenshot shows the 'Expenses' tab in the Dreamscape Books application. A modal dialog titled 'Select Fields to Display' is open in the center. The dialog has two columns: 'Available Fields' and 'Visible Fields'. The 'Available Fields' column lists: Created By, Created By Alias, Created Date, Last Activity Date, Last Modified By, and Last Modified By Alias. The 'Visible Fields' column lists: Expense ID, Expense Amount, Expense Type, and Expense Date. A blue arrow button points from the 'Available Fields' column to the 'Visible Fields' column. At the bottom of the dialog are 'Cancel' and 'Save' buttons. The background shows a list of expenses with columns for Expense ID, Expense Amount, Expense Type, and Expense Date.

The screenshot shows the 'Expenses' tab in the Dreamscape Books application. The table displays a list of expenses with the following columns: Expense ID, Expense Amount, Expense Type, and Expense Date. The table is sorted by Expense ID. The first 21 items are visible, showing various expense types and amounts.

Expense ID	Expense Amount	Expense Type	Expense Date
E-0001	\$100.00	Rent	01/01/2018
E-0002	\$300.00	Entertainment	07/01/2018
E-0003	\$400.00	Food	10/01/2018
E-0004	\$500.00	Entertainment	13/01/2018
E-0005	\$600.00	Education	16/01/2018
E-0006	\$700.00	Rent	19/01/2018
E-0007	\$900.00	Entertainment	25/01/2018
E-0008	\$1,000.00	Rent	28/01/2018
E-0009	\$1,100.00	Utilities	31/01/2018
E-0010	\$1,200.00	Rent	03/02/2018
E-0011	\$1,300.00	Entertainment	06/02/2018
E-0012	\$1,400.00	Food	09/02/2018
E-0013	\$1,500.00	Rent	12/02/2018
E-0014	\$1,600.00	Food	15/02/2018
E-0015	\$1,700.00	Travel	18/02/2018
E-0016	\$1,800.00	Food	21/02/2018
E-0017	\$1,900.00	Utilities	24/02/2018
E-0018	\$2,000.00	Rent	27/02/2018
E-0019	\$2,100.00	Food	02/03/2018
E-0020	\$2,200.00	Insurance	05/03/2018
E-0021	\$2,300.00	Travel	08/03/2018

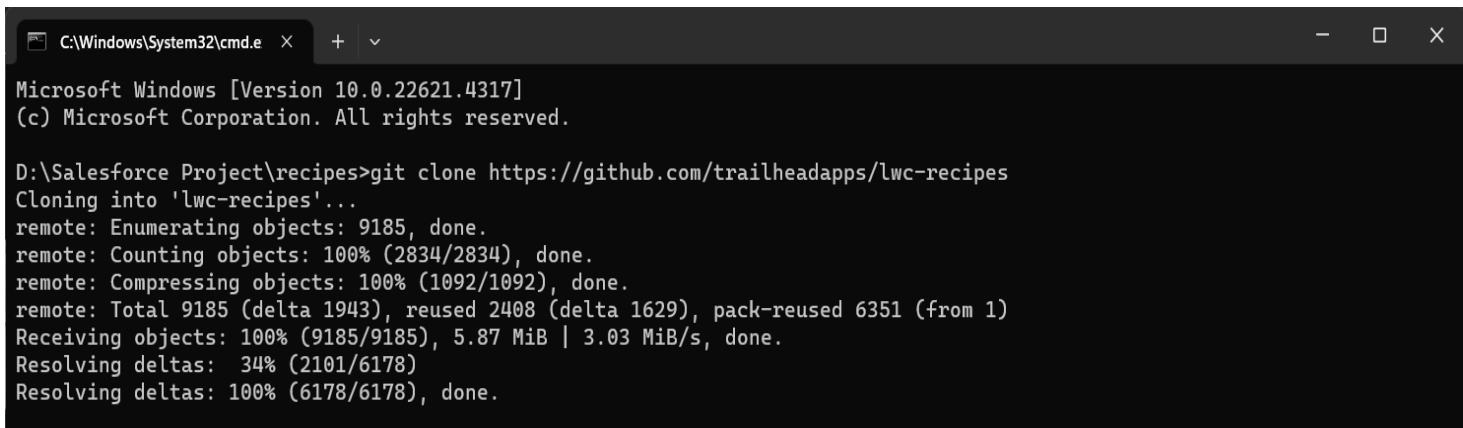
4.4 Cloning the LWC Recipes Repository

- Ensure Git is installed on your machine.
- **Clone Repository:** Use the URL (<https://github.com/trailheadapps/lwc-recipes>) for the LWC Recipes repository.
- Scroll down and copy the path and navigate to your preferred folder where you want to copy this folder.

3. Clone the lwc-recipes repository:

```
git clone https://github.com/trailheadapps/lwc-recipes  
cd lwc-recipes
```

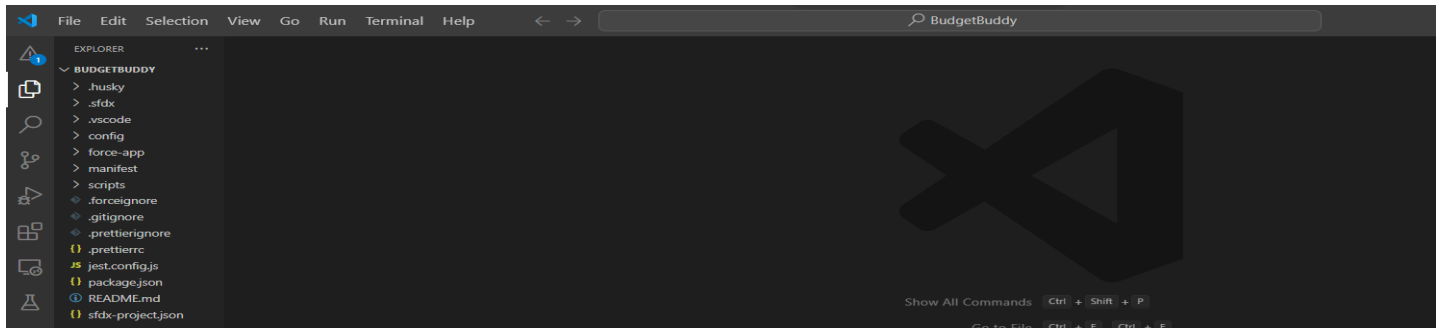
- Open the command prompt and paste the command to clone the repository into the desired folder.



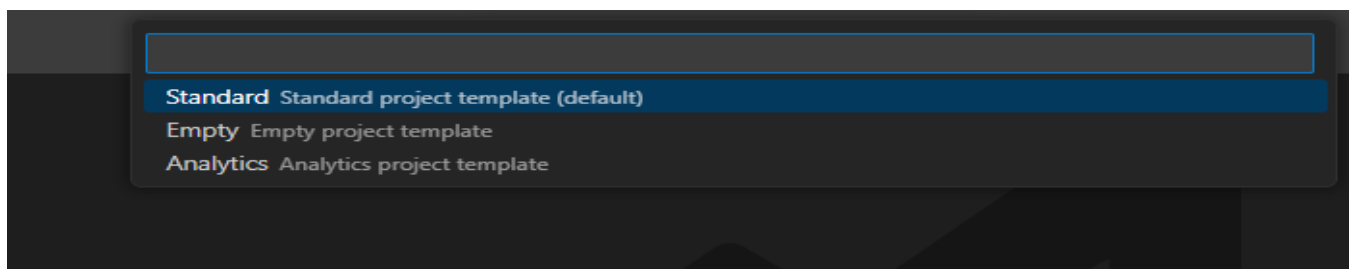
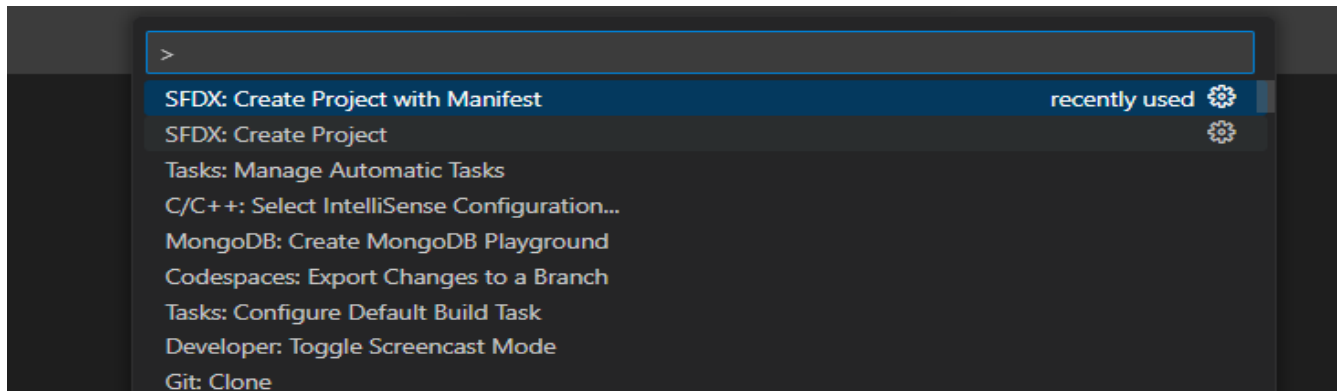
```
C:\Windows\System32\cmd.e  X  +  v  
Microsoft Windows [Version 10.0.22621.4317]  
(c) Microsoft Corporation. All rights reserved.  
  
D:\Salesforce Project\recipes>git clone https://github.com/trailheadapps/lwc-recipes  
Cloning into 'lwc-recipes'...  
remote: Enumerating objects: 9185, done.  
remote: Counting objects: 100% (2834/2834), done.  
remote: Compressing objects: 100% (1092/1092), done.  
remote: Total 9185 (delta 1943), reused 2408 (delta 1629), pack-reused 6351 (from 1)  
Receiving objects: 100% (9185/9185), 5.87 MiB | 3.03 MiB/s, done.  
Resolving deltas: 34% (2101/6178)  
Resolving deltas: 100% (6178/6178), done.
```

4.5 Create Salesforce Project in VS Code

- In VS Code, go to **Extensions**, and search for and install **Salesforce Extension Pack** and **Salesforce Package.xml Generator Extension**
- In VS Code, open the Command Palette (**Ctrl + Shift + P**), type **SFDX: Create Project**, and select it.
- Choose the Standard template.
- Provide a project name and select a location to save the project.

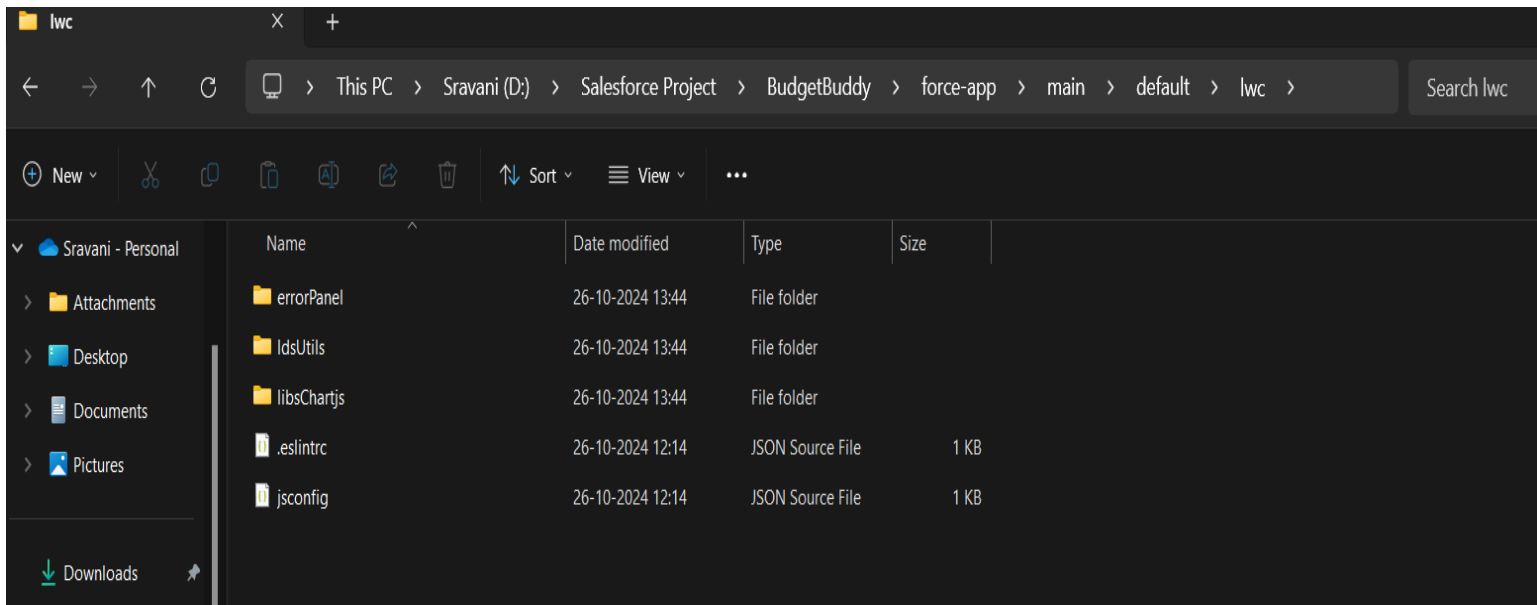


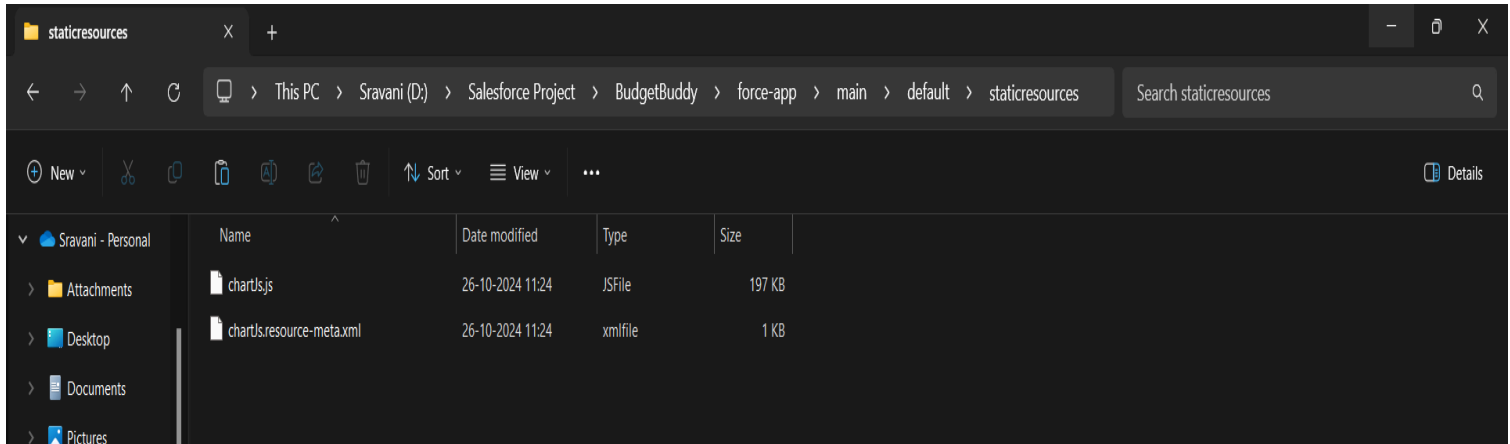
- > Click (**Ctrl + Shift + P**) and enter SFDX: Authorize an org, click on it
- > click on Production or Project Default (as your requirement). Enter the alias for the org
- > It will launch a Window for logging into salesforce and login to your salesforce org. It shows success message once connected to org



4.6 Locating Required Components

- Once cloned, navigate through the following structure in the LWC Recipes folder:
 - force-app > main > default > lwc
 - Copy these files :
 - **errorPanel**
 - **chartjs**
 - **ldsUtils**
- Open your BudgetBuddy org's **LWC folder** in the Explorer.
- Paste the copied components (**errorPanel**, **chartjs**, **ldsUtils**) from the recipes folder into the LWC folder in your project.
- Proceed to copy and paste **static resources** (**chartjs** and **chartjs.resource-meta.xml**) from the recipes' static resources folder into the project's static resources.



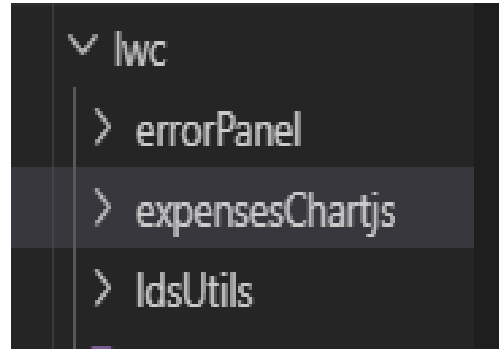
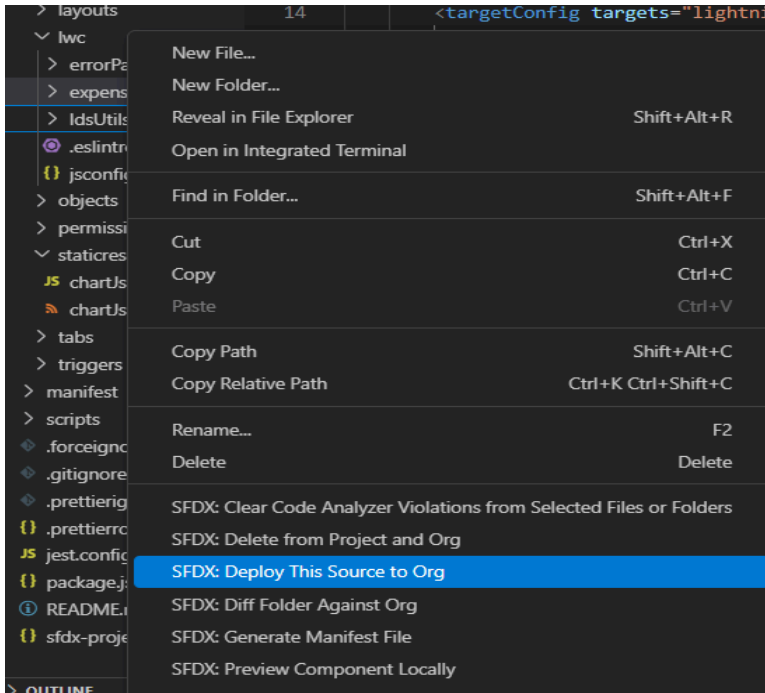


4.7 Deploying Components

- Right-click on the **static resources folder** and choose **Deploy Source to Org** to upload these static files.
- **LWC Components Deployment:** Deploy `ldsUtils` before `errorPanel`, as `errorPanel` depends on `ldsUtils`. Deploy each component individually.

4.8 Renaming Components

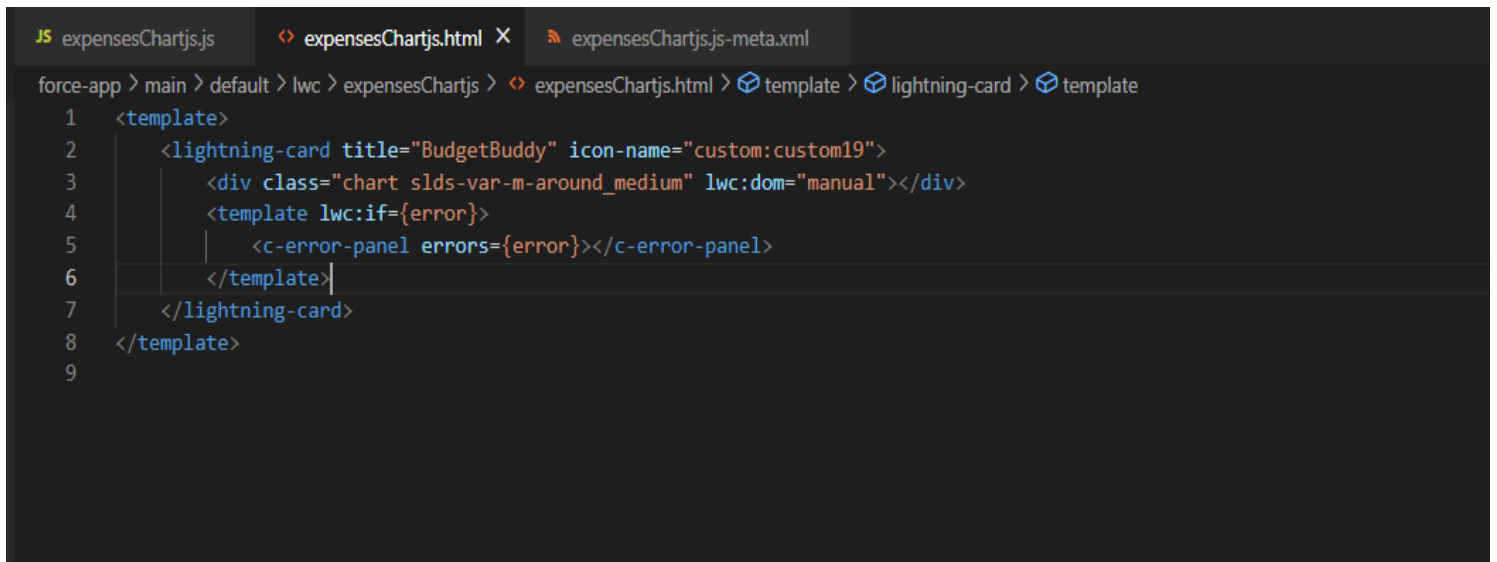
- Rename the `chartjs` component to `expensesChartjs`:
 - Right-click, choose **SFDX Rename Component**, and rename it to `expensesChartjs`



- Navigate to expensesChartjs file and modify the following files

1) HTML Adjustments:

- Set the title as **BudgetBuddy**.
- Ensure the **lwc:dom** directive is set to **manual**.



2) Open expensesChartjs.js file: Set the class name to **ExpensesChartjs**

```
JS expensesChartjs.js X <> expensesChartjs.html expensesChartjs.js-meta.xml
force-app > main > default > lwc > expensesChartjs > JS expensesChartjs.js > ExpensesChartjs
1 import { LightningElement } from 'lwc';
2 import chartjs from '@salesforce/resourceUrl/chartjs';
3 import { loadScript } from 'lightning/platformResourceLoader';
4 /**
5  * When using this component in an LWR site, please import the below custom implementation of 'loadScript' module
6  * instead of the one from 'lightning/platformResourceLoader'
7  *
8  * import { loadScript } from 'c/resourceLoader';
9  *
10 * This workaround is implemented to get around a limitation of the Lightning Locker library in LWR sites.
11 * Read more about it in the "Lightning Locker Limitations" section of the documentation
12 * https://developer.salesforce.com/docs/atlas.en-us.exp_cloud_lwr.meta/exp_cloud_lwr/template_limitations.htm
13 */
14
15 const generateRandomNumber = () => {
16   return Math.round(Math.random() * 100);
17 };
18
19 export default class ExpensesChartjs extends LightningElement {
20   error;
21   chart;
22   chartjsInitialized = false;
23
24   config = {
25     type: 'doughnut'
```

3) open the expensesChartjs.xml file: rewrite the code as below. This code is for taking input from the user.

```
JS expensesChartjs.js <> expensesChartjs.html expensesChartjs.js-meta.xml X
force-app > main > default > lwc > expensesChartjs > expensesChartjs.js-meta.xml > ...
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
3   <apiVersion>61.0</apiVersion>
4   <isExposed>true</isExposed>
5   <targets>
6     <target>lightning__AppPage</target>
7     <target>lightning__RecordPage</target>
8     <target>lightning__HomePage</target>
9     <target>lightningCommunity__Page</target>
10    <target>lightningCommunity__Default</target>
11    <target>lightning__FlowScreen</target>
12  </targets>
13  <targetConfigs>
14    <targetConfig targets="lightning__FlowScreen">
15      <property name="startDate" label="Start Date" type="Date" role="inputOnly" />
16      <property name="endDate" label="end Date" type="Date" role="inputOnly" />
17    </targetConfig>
18  </targetConfigs>
19 </LightningComponentBundle>
20
21
```

4) Add API Decorators: Define **@api** properties for start date and end date so that these dates can be passed to the component from the flow.

```
JS expensesChartjs.js X expensesChartjs.html expensesChartjs.js-meta.xml
force-app > main > default > lwc > expensesChartjs > JS expensesChartjs.js > ExpensesChartjs
1 import { api, LightningElement } from 'lwc';
2 import chartjs from '@salesforce/resourceUrl/chartJs';
3 import { loadScript } from 'lightning/platformResourceLoader';
4 /**
5  * When using this component in an LWR site, please import the below custom implementation of 'loadScript' module
6  * instead of the one from 'lightning/platformResourceLoader'
7  *
8  * import { loadScript } from 'c/resourceLoader';
9  *
10 * This workaround is implemented to get around a limitation of the Lightning Locker library in LWR sites.
11 * Read more about it in the "Lightning Locker Limitations" section of the documentation
12 * https://developer.salesforce.com/docs/atlas.en-us.exp\_cloud\_lwr.meta/exp\_cloud\_lwr/template\_limitations.htm
13 */
14
15 const generateRandomNumber = () => {
16     return Math.round(Math.random() * 100);
17 };
18
19 export default class ExpensesChartjs extends LightningElement {
20     error;
21     chart;
22     chartjsInitialized = false;
23     @api startDate
24     @api endDate
25     config = {
26         type: 'doughnut',
27         data: {
```

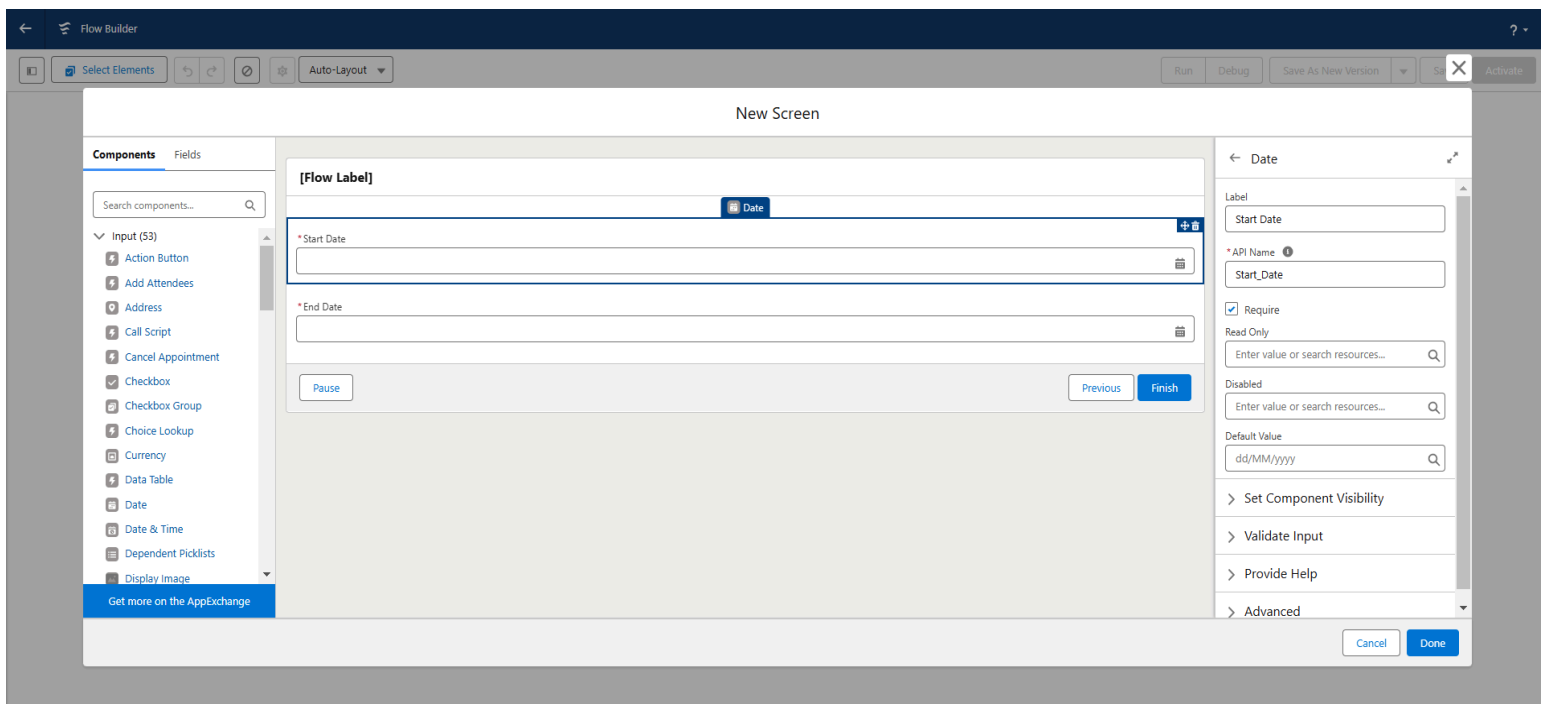
- > Deploy this component to org. Right-click - > SFDX: Deploy this source to Org

4.9 Configuring the Flow to Use the Component

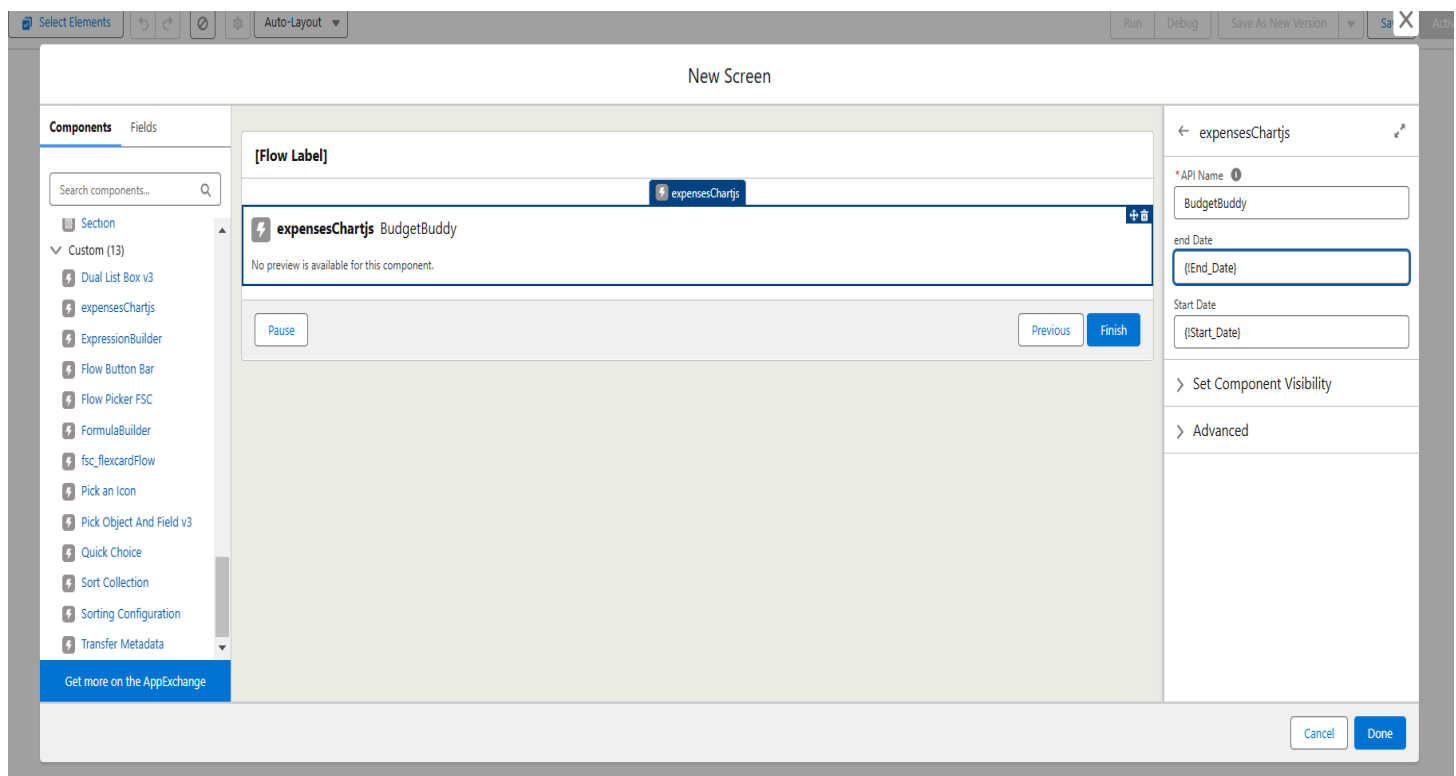
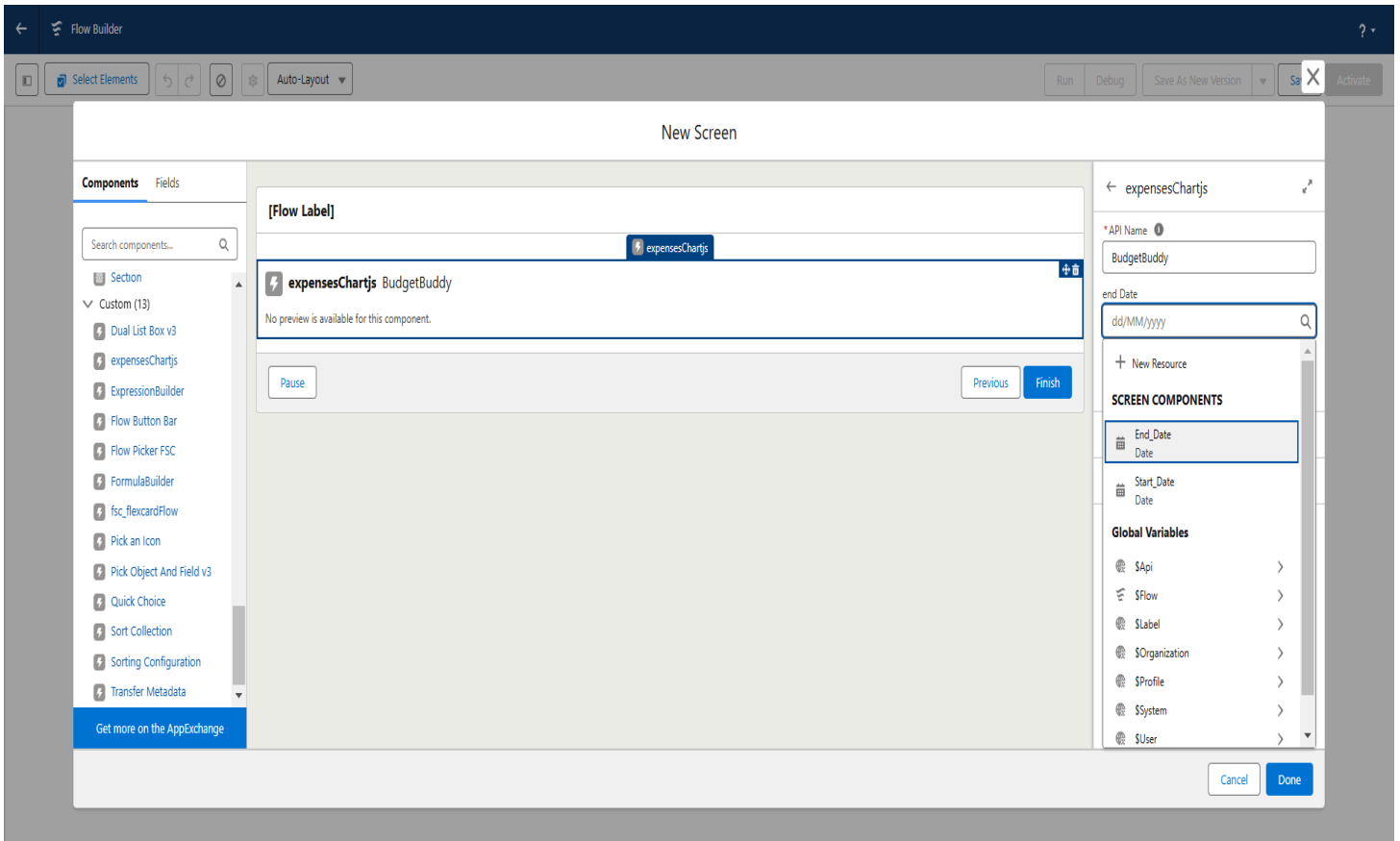
- Go to Setup in Salesforce.
- In the Quick Find box, type Flow Builder and select it.
- Click on New Flow.
- Select Screen Flow and click Create.

Add Screen Element

- In the Flow Builder, drag the Screen element from the left sidebar to the canvas.
- In the Screen Properties pane, for Label, enter "Date range". Add two date fields:
- First Screen, Click on Components and drag the two date fields
 - **Label: Start Date:** Required field.
 - **Label: End Date:** Required field.



- **Second Screen:** Add the custom component, **expensesChartjs**, as follows:
 - In the Screen Properties pane, for Label, enter "Expense Summary".
 - Drag the **expensesChartjs** component onto the screen.
 - Map **startDate** and **endDate** from the input screen to the respective API properties of the component.
 - Click On Done.
 - Save the Flow with **Expense Summary Flow**.

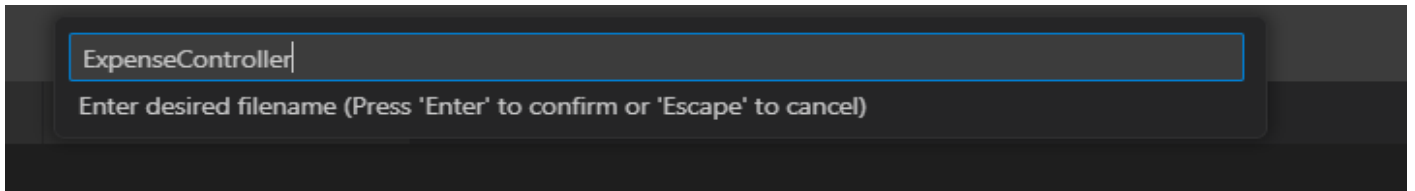


4.10 Creating the Apex Controller and LWC for BudgetBuddy

First, we'll create an Apex class named `ExpenseController`.

Set Up the Controller:

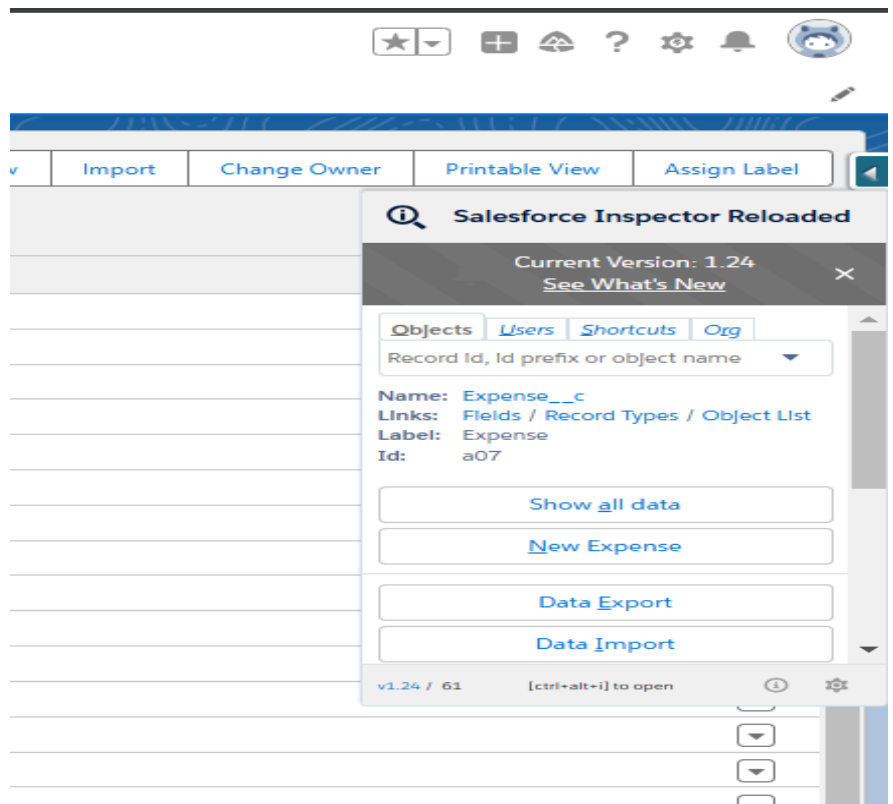
Click (**Ctrl + Shift + P**) and enter SFDX: Create Apex Class, click on it and name it as `ExpenseController` -> Enter



This Apex class retrieves and aggregates expenses from the `Expense__c` object by type within a specified date range, returning the total amounts for each type. It can be called from a Lightning component for dynamic data presentation.

Salesforce Inspector is used in this project to quickly and efficiently access and verify the correct API names of fields in Salesforce objects. It helps developers construct accurate SOQL queries and ensures that the data being queried is aligned with the requirements of the application.

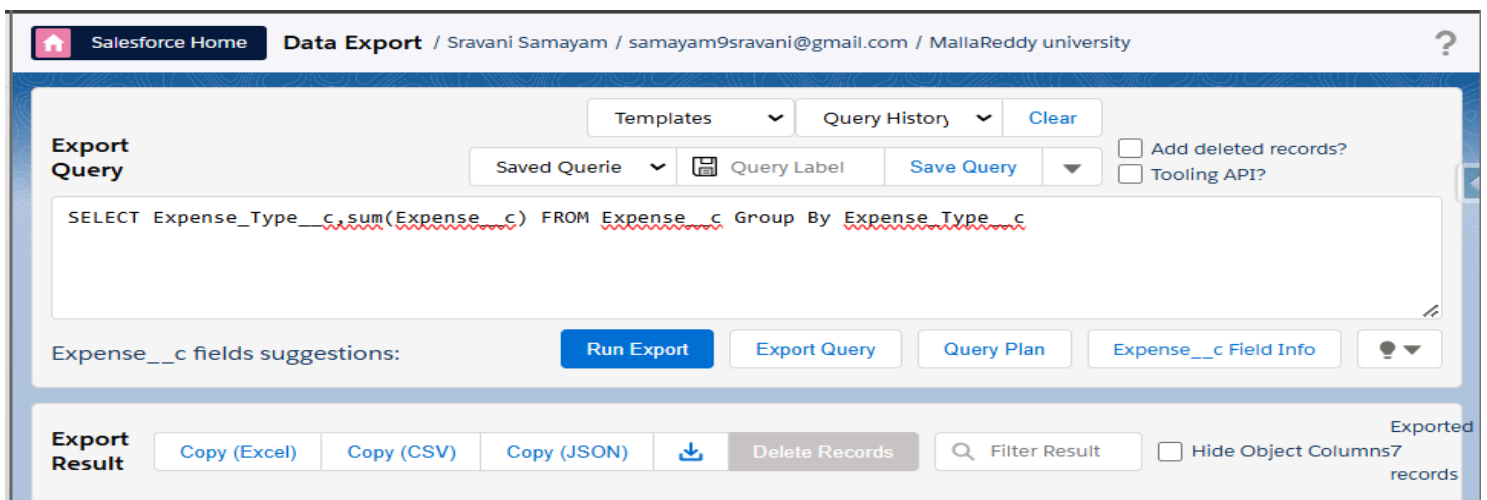
- Add the Salesforce Inspector Extension From Chrome Web Store.
- Now open the Setup in Salesforce
- Click on the Salesforce inspector and click on Data Export.



Enter this Query:

```
SELECT Expense_Type__c,sum(Expense__c) FROM Expense__c
Group By Expense_Type__c
```

- This SOQL query retrieves aggregated data from the Expense__c object in Salesforce. Specifically, it selects the Expense_Type__c field and calculates the sum of the Expense__c field for each unique expense type. The GROUP BY Expense_Type__c clause groups the results by the expense type, providing a total expense amount for each type.



- Now open VS Code and navigate to the [ExpenseController](#) file and Enter the Code Given Below:

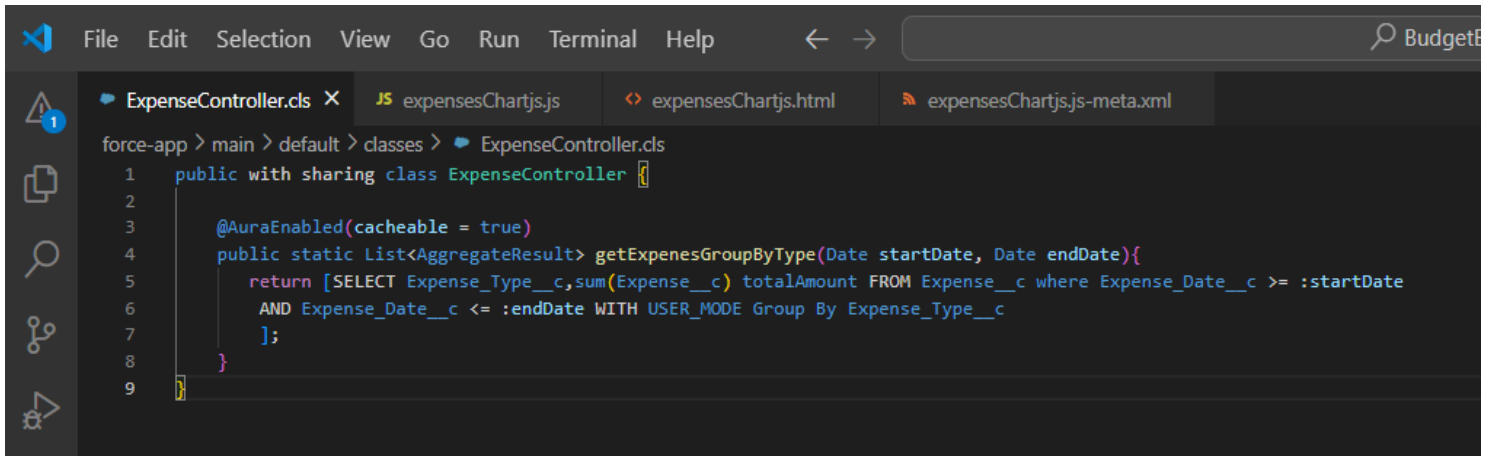
```
public with sharing class ExpenseController {
```

```
    @AuraEnabled(cacheable = true)
```

```
    public static List<AggregateResult> getExpensesGroupedByType(Date
    startDate, Date endDate){
```

```
        return [SELECT Expense_Type__c,sum(Expense__c) totalAmount FROM
        Expense__c where Expense_Date__c >= :startDate AND Expense_Date__c
        <= :endDate WITH USER_MODE Group By Expense_Type__c ];
```

```
    }
```



```
1 public with sharing class ExpenseController {
2
3     @AuraEnabled(cacheable = true)
4     public static List<AggregateResult> getExpensesGroupedByType(Date startDate, Date endDate){
5         return [SELECT Expense_Type__c,sum(Expense__c) totalAmount FROM Expense__c where Expense_Date__c >= :startDate
6                 AND Expense_Date__c <= :endDate WITH USER_MODE Group By Expense_Type__c
7         ];
8     }
9 }
```

Explanation of Code:

Method (**getExpensesGroupedByType**):

- **Annotation `@AuraEnabled(cacheable = true)`:** Makes the method available for use in Lightning components and allows caching of the results for better performance.
- **Parameters:** Accepts two `Date` parameters, `startDate` and `endDate`, representing the range for the expense query.
- **SOQL Query:**
 - Retrieves records from `Expense__c` custom object.
 - Filters based on the `Expense_Date__c` field to include only expenses within the provided date range.
 - Groups by `Expense_Type__c` and calculates the total expense amount per type.
- **Return Type:** Returns a list of `AggregateResult` objects, each containing an expense type and its corresponding total amount.
- This code retrieves and groups expenses by type within a date range, providing a summary of total expenses per type.
- Helps users analyze spending patterns by category over time, making it easier to manage and assess financial data in the application.

Now navigate to the Data Export, where we've written the SOQL query and Click on the Run Export. You get a table summarizing expenses grouped by type.

- **Expense_Type__c**: This column lists the categories or types of expenses (e.g., Entertainment, Food, Education).
- **expr0**: This column shows the total amount spent for each expense type over the specified date range.

The screenshot shows the Salesforce Data Export interface. At the top, there's a navigation bar with 'Salesforce Home', 'Data Export', and user information. Below this, the 'Export Query' section contains a text area with the SOQL query: `SELECT Expense_Type__c, sum(Expense__c) FROM Expense__c Group By Expense_Type__c`. To the right of the query are buttons for 'Templates', 'Query History', 'Clear', 'Saved Query', 'Query Label', and 'Save Query'. There are also checkboxes for 'Add deleted records?' and 'Tooling API?'. Below the query text area are buttons for 'Run Export', 'Export Query', 'Query Plan', and 'Expense__c Field Info'. The 'Export Result' section shows a table with columns 'Expense_Type__c' and 'expr0'. The table contains seven rows of data, each preceded by a blue link labeled 'AggregateResult'. To the right of the table are buttons for 'Copy (Excel)', 'Copy (CSV)', 'Copy (JSON)', 'Delete Records', and 'Filter Result'. There is also a checkbox for 'Hide Object Columns' and a status indicator 'Exported 7 records'.

	Expense_Type__c	expr0
AggregateResult	Entertainment	5082700
AggregateResult	Food	7467000
AggregateResult	Education	6724700
AggregateResult	Rent	5353200
AggregateResult	Utilities	5601200
AggregateResult	Travel	7507900
AggregateResult	Insurance	5663900

- Now , import the `getExpenesGroupByType` method into the JavaScript file to make the **Apex method** available for use in the **Lightning Web Component (LWC)**. By importing this method, JavaScript can call `getExpenesGroupByType` to retrieve data from Salesforce, specifically the grouped expense data based on type and date range, and then display it within the LWC.

- Importing this method enables communication between the Apex backend and the LWC frontend, allowing the component to access and display dynamic Salesforce data.

```

ExpenseController.cls  JS expensesChartjs.js  expensesChartjs.html  expensesChartjsjs-meta.xml
force-app > main > default > lwc > expensesChartjs > JS expensesChartjs.js > ...
1  import { api, LightningElement } from 'lwc';
2  import chartjs from '@salesforce/resourceUrl/chartJs';
3  import { loadScript } from 'lightning/platformResourceLoader';
4  import getExpensesGroupByType from '@salesforce/apex/ExpenseController.getExpensesGroupByType';
5  /**
6   * When using this component in an LWR site, please import the below custom implementation of 'loadScript' module
7   * instead of the one from 'lightning/platformResourceLoader'
8   *
9   * import { loadScript } from 'c/resourceLoader';
10  *
11  * This workaround is implemented to get around a limitation of the Lightning Locker library in LWR sites.
12  * Read more about it in the "Lightning Locker Limitations" section of the documentation
13  * https://developer.salesforce.com/docs/atlas.en-us.exp_cloud_lwr.meta/exp_cloud_lwr/template_limitations.htm
14  */
15
16  const generateRandomNumber = () => {
17    return Math.round(Math.random() * 100);
18  };
19
20  export default class ExpensesChartjs extends LightningElement {
21    error;
22    chart;

```

4.11 Displaying the Data

Enter the Following Code in expensesChartjs.js file:

```

ExpenseController.cls  JS expensesChartjs.js  expensesChartjs.html  expensesChartjsjs-meta.xml
force-app > main > default > lwc > expensesChartjs > JS expensesChartjs.js > generateRandomNumber
1  import { api, LightningElement } from 'lwc';
2  import chartjs from '@salesforce/resourceUrl/chartJs';
3  import { loadScript } from 'lightning/platformResourceLoader';
4  import getExpensesGroupByType from '@salesforce/apex/ExpenseController.getExpensesGroupByType';

```

- **{ api, LightningElement } from 'lwc';**: Imports **api** (for public properties) and **LightningElement** (the base class for LWC components) from the LWC framework.
- **chartjs from '@salesforce/resourceUrl/chartJs';**: Imports the Chart.js library as a static resource from Salesforce, allowing the component to use Chart.js for data visualization.
- **loadScript from 'lightning/platformResourceLoader';**: Imports **loadScript**, a function that loads external scripts like Chart.js into the component.
- **getExpensesGroupByType from '@salesforce/apex/ExpenseController.getExpensesGroupByType';**

Imports the `getExpensesGroupedByType` Apex method to retrieve expense data grouped by type, which will be displayed in the chart.

Chart Configuration (**config**):

- This object sets up the configuration for the doughnut chart.
- The chart is set to be **non-responsive**, meaning it will not adjust its size automatically when the viewport changes.
- **Legend Position:** The legend (expense types) appears on the right.
- **Animation Options:** Enables scale and rotation animations for the chart's appearance.

```
config = {
  type: 'doughnut',

  options: {
    responsive: false,
    plugins: {
      legend: {
        position: 'right'
      }
    },
    animation: {
      animateScale: true,
      animateRotate: true
    }
  }
};
```

renderedCallback() Method:

- This **lifecycle hook** in LWC runs every time the component is rendered.
- The `chartjsInitialized` flag ensures that the code only runs once, preventing duplicate chart creation on re-renders.

```
async renderedCallback() {
  if (this.chartjsInitialized) {
    return;
  }
  this.chartjsInitialized = true;
}
```

Data Object (**data**):

- Defines the data structure for the chart, initially empty.
- `datasets[0].data` will later store expense amounts for each type.
- `labels` will store expense categories/types (like "Food" and "Rent").

```
let data = {
  datasets: [
    {
      data: [
        //expense amount
      ],
      backgroundColor: [
      ],
      label: 'Dataset 1'
    }
  ],
  labels: [] //expense type
}
```

Data Fetching and Processing:

Call the `getExpensesGroupedByType` Apex method to retrieve expense data for the selected date range.

- `result` holds an array of expense data objects grouped by type, with each object containing `Expense_Type__c` (type) and `totalAmount` (total amount).

```
try {
  await loadScript(this, chartjs);
  let result = await getExpensesGroupedByType({
    startDate : this.startDate,
    endDate : this.endDate
  })
}
```

Data Population for Chart:

- Loops through each expense data item from the Apex result and populates `data`.
- **Expense Type and Amount:** `totalAmount` and `Expense_Type__c` are added to `data.datasets[0].data` and `labels`.
- **Background Color:** Random color is assigned to each category by calling `getRandomColor()`.

```
result.forEach(item => {
  data.datasets[0].data.push(item.totalAmount);
  data.labels.push(item.Expense_Type__c);
  data.datasets[0].backgroundColor.push(this.getRandomColor())
});
```

Chart Creation:

- **Canvas Element:** A new `<canvas>` is created and appended to the HTML `<div class="chart">` container to render the chart.
- **Chart.js Instance:** Initializes Chart.js with the populated `config` and `data` on the canvas context (`ctx`), rendering the doughnut chart in the component.

```
this.config.data = data;

const canvas = document.createElement('canvas');
this.template.querySelector('div.chart').appendChild(canvas);
const ctx = canvas.getContext('2d');
this.chart = new window.Chart(ctx, this.config);
```

Random Color Generation (**getRandomColor** method):

Generates a **random hex color** to apply unique colors to each expense type, enhancing chart readability.

```
getRandomColor() {  
  let letters = '0123456789ABCDEF';  
  let color = '#';  
  for (let i = 0; i < 6; i++) {  
    color += letters[Math.floor(Math.random() * 16)];  
  }  
  return color;  
}
```

Complete Code for “expensesChartjs.js” file

```
import { api, LightningElement } from 'lwc';  
import chartjs from '@salesforce/resourceUrl/chartJs';  
import { loadScript } from 'lightning/platformResourceLoader';  
import { getExpensesGroupByType } from '@salesforce/apex/ExpenseController.getExpensesGroupByType';  
  
const generateRandomNumber = () => {  
  return Math.round(Math.random() * 100);  
};  
  
export default class ExpensesChartjs extends LightningElement {  
  error;  
  chart;  
  chartjsInitialized = false;  
  @api startDate  
  @api endDate  
  config = {  
    type: 'doughnut',  
    options: {  
      responsive: false,  
      plugins: {
```

```

        legend: {
            position: 'right'
        }
    },
    animation: {
        animateScale: true,
        animateRotate: true
    }
}
};

async renderedCallback() {
    if (this.chartjsInitialized) {
        return;
    }
    this.chartjsInitialized = true;
    let data = {
        datasets: [
            {
                data: [
                    //expense amount

                ],
                backgroundColor: [
                ],
                label: 'Dataset 1'
            }
        ],
        labels: [] //expense type
    }
    try {
        await loadScript(this, chartjs);
        let result = await getExpenesGroupByType({
            startDate : this.startDate,
            endDate : this.endDate
        })
        result.forEach(item => {

```

```

        data.datasets[0].data.push(item.totalAmount);
        data.labels.push(item.Expense_Type__c);
        data.datasets[0].backgroundColor.push(this.getRandomColor())
    });

    this.config.data = data;
    const canvas = document.createElement('canvas');
    this.template.querySelector('div.chart').appendChild(canvas);
    const ctx = canvas.getContext('2d');
    this.chart = new window.Chart(ctx, this.config);
} catch (error) {
    this.error = error;
}
}

getRandomColor() {
    let letters = '0123456789ABCDEF';
    let color = '#';
    for (let i = 0; i < 6; i++) {
        color += letters[Math.floor(Math.random() * 16)];
    }
    return color;
}
}

```

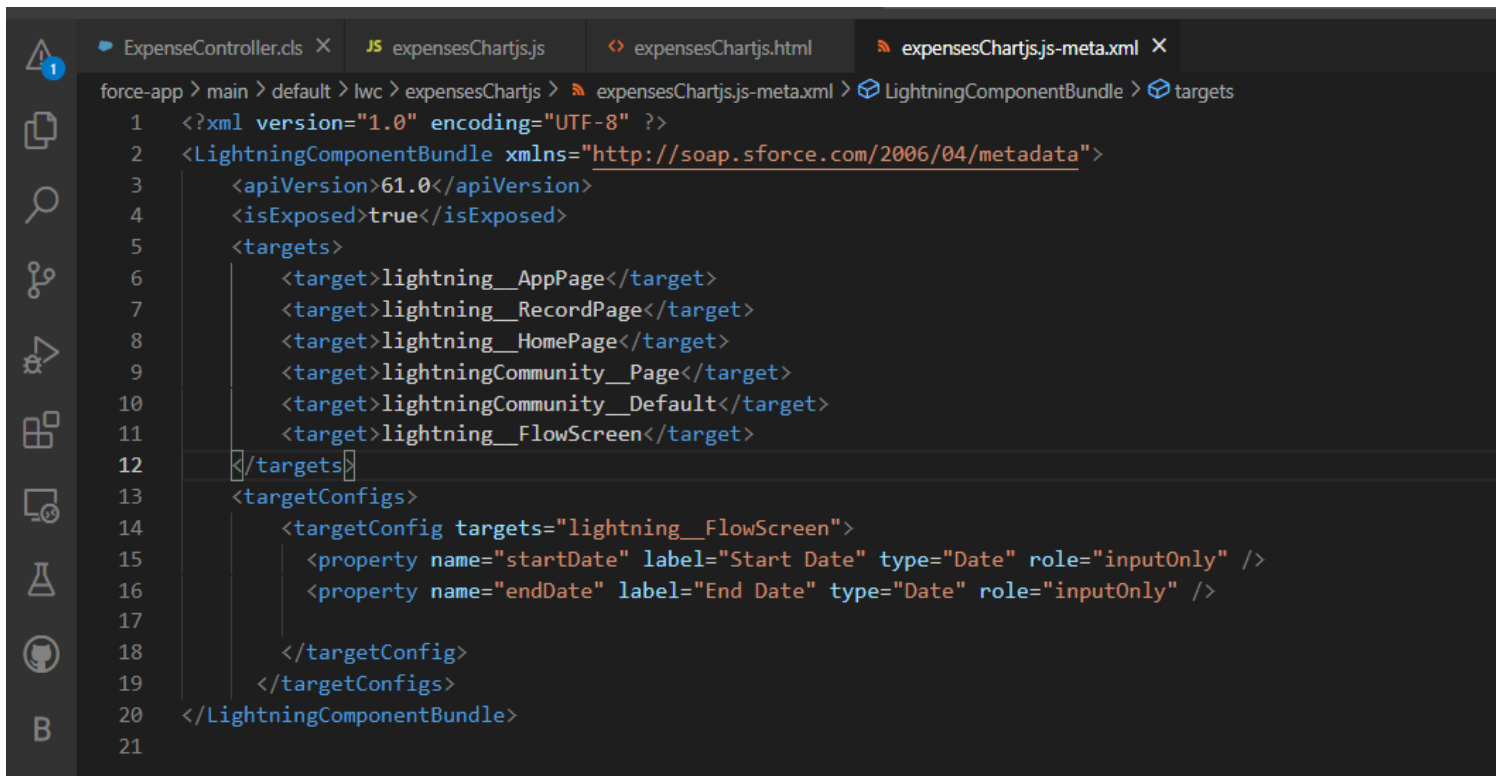
expensesChartjs.js File:

```

ExpenseController.cls  JS expensesChartjs.js  <> expensesChartjs.html X  expensesChartjs.js-meta.xml
force-app > main > default > lwc > expensesCharts > <> expensesChartjs.html > template > lightning-card
1  <template>
2      <lightning-card title="BudgetBuddy" icon-name="custom:custom19">
3          <div class="chart slds-var-m-around_medium" lwc:dom="manual"></div>
4          <template lwc:if={error}>
5              <c-error-panel errors={error}></c-error-panel>
6          </template>
7      </lightning-card>
8  </template>
9

```

expensesChartjs.js-meta.xml file:

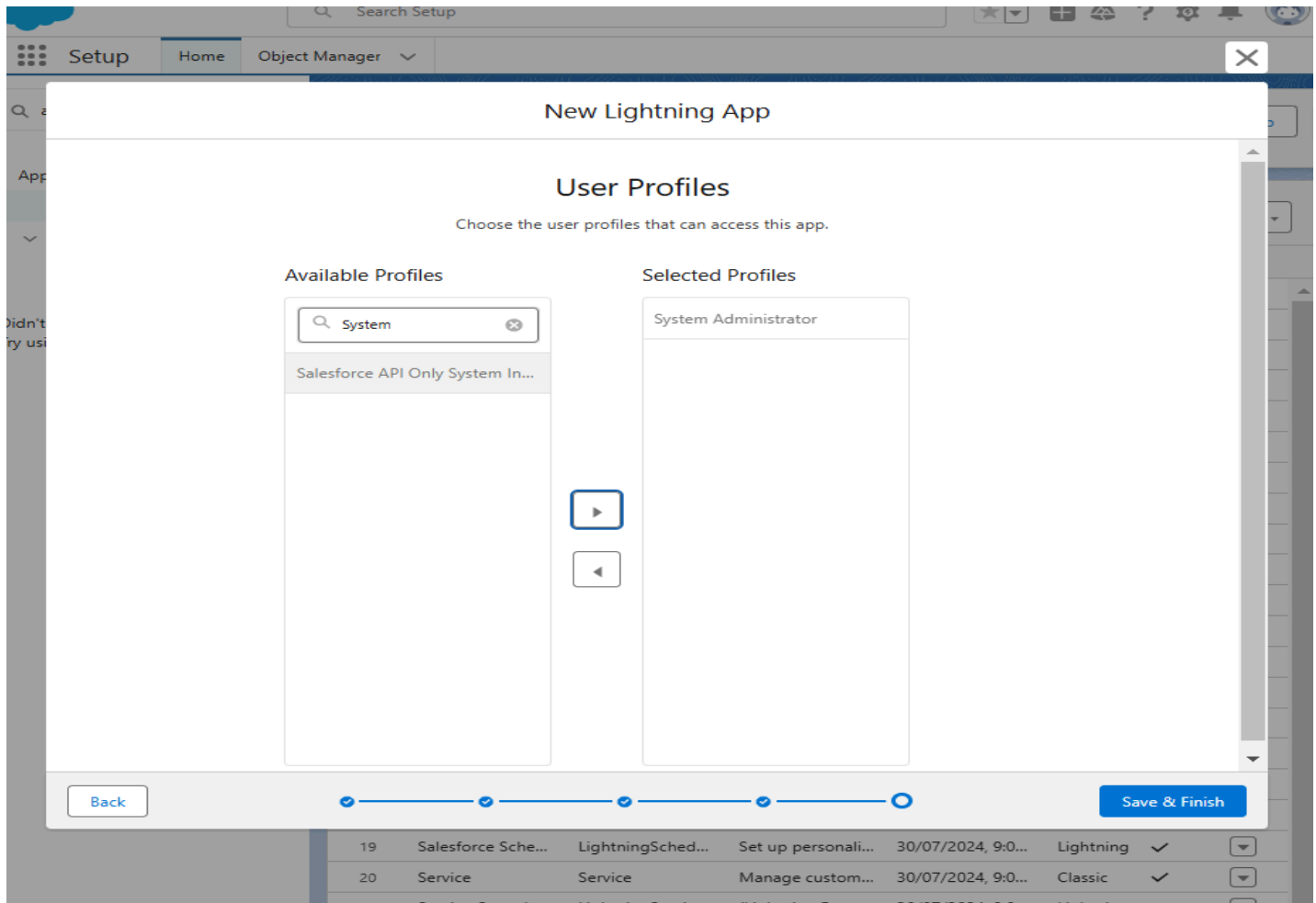


```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
3   <apiVersion>61.0</apiVersion>
4   <isExposed>true</isExposed>
5   <targets>
6     <target>lightning__AppPage</target>
7     <target>lightning__RecordPage</target>
8     <target>lightning__HomePage</target>
9     <target>lightningCommunity__Page</target>
10    <target>lightningCommunity__Default</target>
11    <target>lightning__FlowScreen</target>
12  </targets>
13  <targetConfigs>
14    <targetConfig targets="lightning__FlowScreen">
15      <property name="startDate" label="Start Date" type="Date" role="inputOnly" />
16      <property name="endDate" label="End Date" type="Date" role="inputOnly" />
17    </targetConfig>
18  </targetConfigs>
19 </LightningComponentBundle>
20
21
```

- Make sure to Deploy all the files to the Salesforce Connected Org.


4.12 Create the BudgetBuddy App

- Go to Setup in Salesforce
- Search and Select App Manager
- Click on New Lightning App
- **App name:** BudgetBuddy
- Click on next for 3 times
- Search for Expenses in Available Items and move it to Selected items.
- Search for System Administrator in Available profiles and move it from available profiles to selected profiles.
- Click on Save & Finish.









4.13 Create a lightning app page

- Go to Setup and enter App Builder in the Quick Find box
- Click Lightning App Builder.
- Click New, select App Page, then click Next.
- Name the page BudgetBuddy Dashboard.
- Select the One region Page template.
- Click Done.



Search Setup



SetupHomeObject Manager

app builder


Feature Settings

- Service
 - Field Service
 - Field Service Mobile
 - Field Service Mobile App Builder

User Interface

- Lightning App Builder

Didn't find what you're looking for? Try using Global Search.

SETUPLightning App Builder

The Lightning App Builder provides an easy to use graphical interface for creating custom Lightning pages for Salesforce Lightning Experience and mobile app. Lightning pages are built using Lightning components—compact, configurable, and reusable elements that you can drag and drop into regions of the page in the Lightning App Builder.

View: All [Create New View](#)

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | Other All

Lightning Pages

New

Action	Label ↑	Name	Namespace Prefix	Description	Type	Created By	Last Modified By
Edit Clone Del	Book Order Record Page	Book_Order_Record_Page			Record Page	SSama, 30/07/2024, 9:02 pm	SSama, 30/07/2024, 9:02 pm
Edit Clone Del	BudgetBuddy DashBoard	BudgetBuddy_DashBoard			App Page	SSama, 26/10/2024, 4:14 pm	SSama, 26/10/2024, 4:14 pm
Edit Clone Del	Contact Record Page	Contact_Record_Page			Record Page	SSama, 30/07/2024, 9:02 pm	SSama, 30/07/2024, 9:02 pm

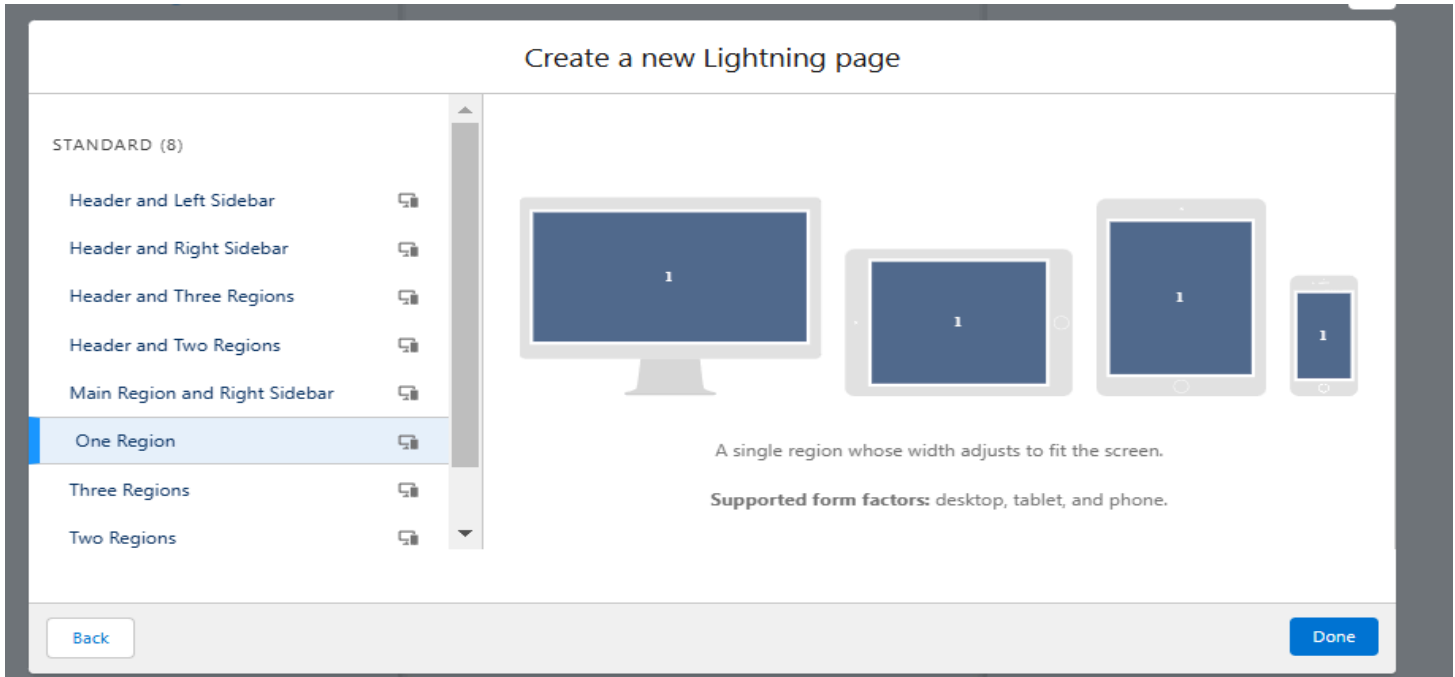
Create a new Lightning page

* Label

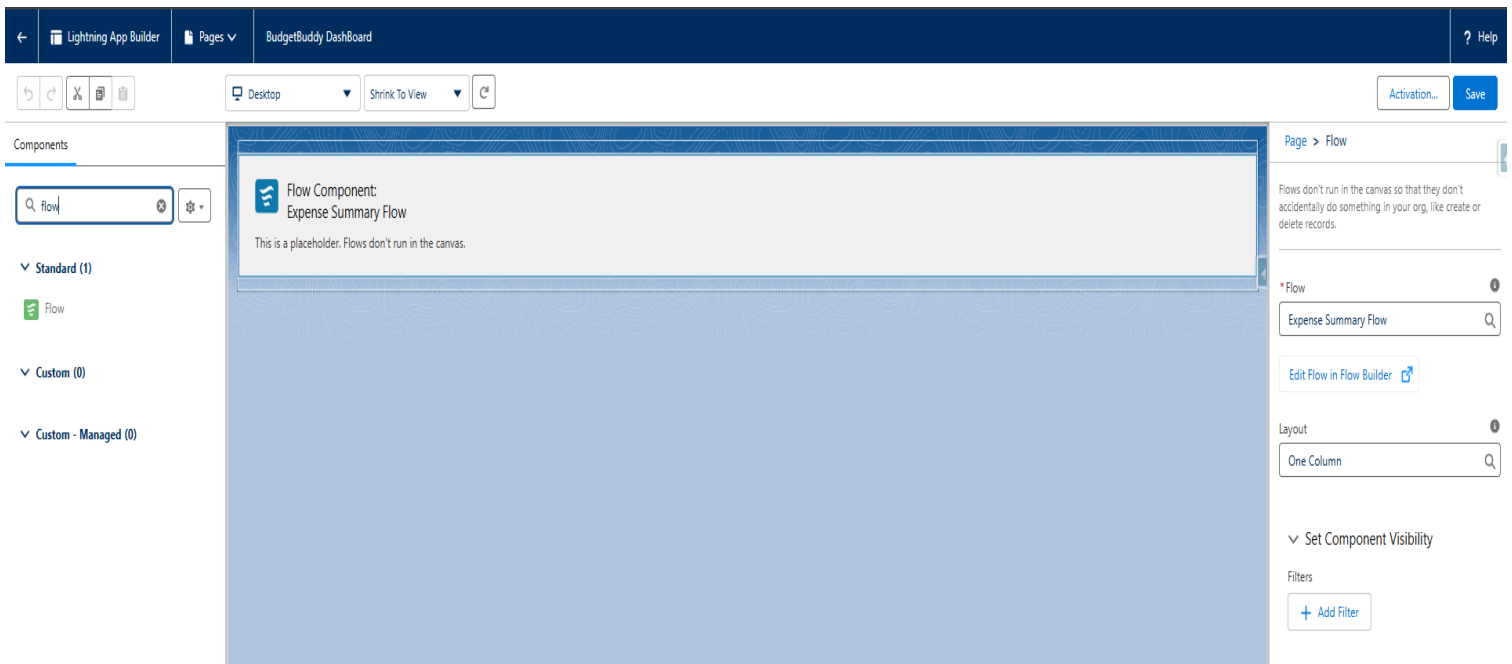
BudgetBuddy DashBoard

Back


Next



- Find Flow in Components in Left-side
- Drag and drop the Flow Component on the region.
- Select the Flow in Right-side
- Click on Save



OUTPUT SCREENS



★


+


🔒

?

⚙️


🔔



 BudgetBuddy


Expenses ▾

BudgetBuddy DashBoard

 BudgetBuddy DashBoard


Expense Summary Flow

* Start Date

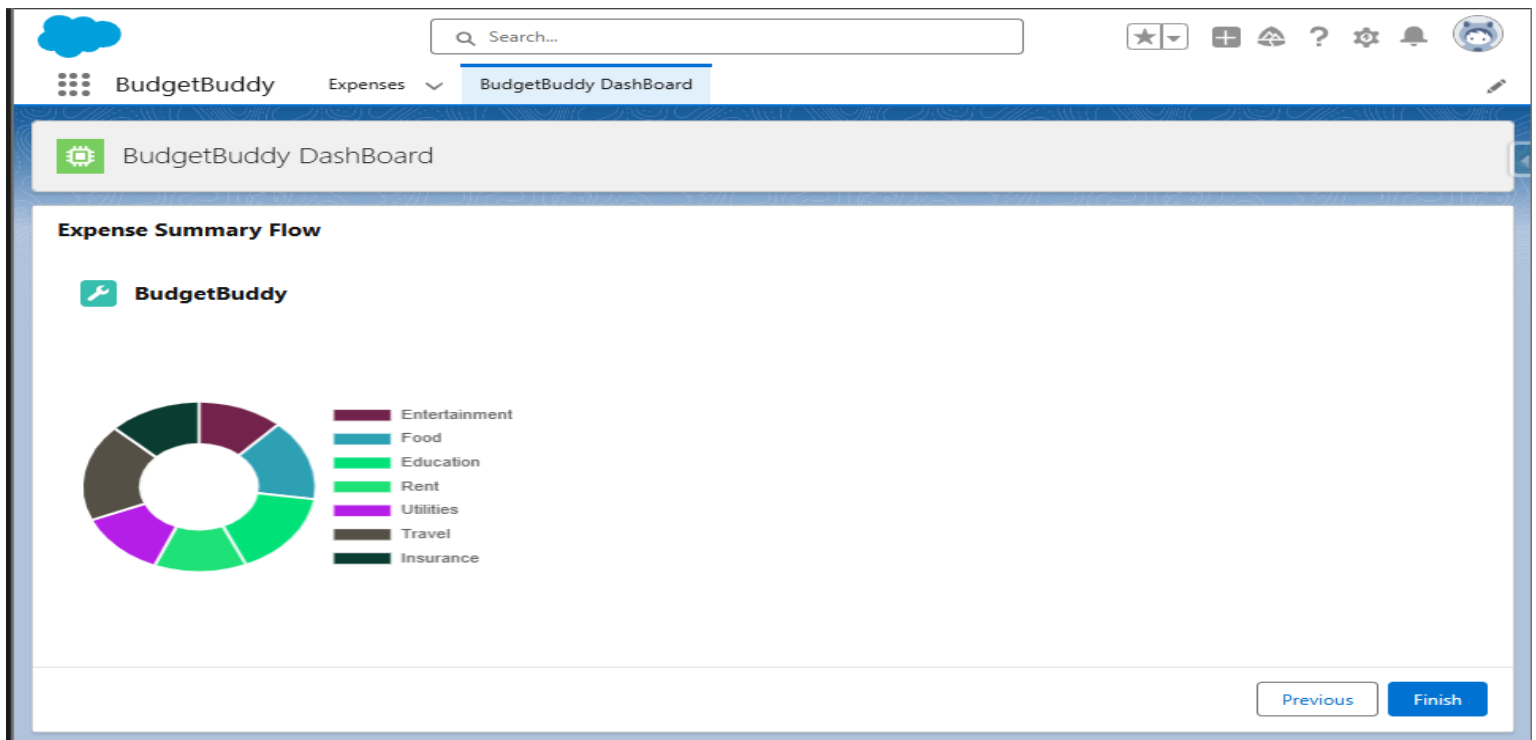


Format: 31-Dec-2024

* End Date



Next



For creating new expense > Click on Expenses tab > New

The screenshot shows the 'New Expense' modal form in the BudgetBuddy application. The form is titled 'New Expense' and includes a close button (X) in the top right corner. It contains the following fields and options:

- Expense ID:** A text input field.
- Owner:** A dropdown menu showing 'Srivani Samayam'.
- Expense Type:** A dropdown menu showing 'Education'.
- Expense Date:** A date picker showing '25/10/2024'.
- Expense Amount:** A text input field showing '50000'.

At the bottom of the modal, there are three buttons: 'Cancel', 'Save & New', and 'Save'. The background shows a table of existing expenses with columns for Expense ID, Expense Amount, Expense Type, and Expense Date.

The screenshot shows the 'Expense E-0872' details page in the BudgetBuddy application. The page is titled 'Expense E-0872' and includes a close button (X) in the top right corner. It contains the following sections:

- Related:** A section showing the expense details, including Expense ID (E-0872), Expense Type (Education), Expense Date (25/10/2024), Expense Amount (₹50,000.00), Created By (Srivani Samayam), and Last Modified By (Srivani Samayam).
- Activity:** A section showing the activity history for this expense. It includes a filter bar with 'All time', 'All activities', and 'All types'. Below the filter bar, it says 'No activities to show. Get started by sending an email, scheduling a task, and more.' and 'No past activity. Past meetings and tasks marked as done show up here.'

5. Testing and Validation

Unit Testing (Apex Classes, Triggers): Validates backend logic by testing Apex classes and triggers. Ensures calculations, like expense categorization and total updates, work accurately under various scenarios, providing reliable data handling.

User Interface (UI) Testing: Confirms that the application's UI is responsive, error-free, and user-friendly. Tests cover data entry, filtering, and visualization (e.g., charts) to ensure the interface is intuitive and displays accurate information.

Debugging and Testing the Flow

- Click **Debug** to run the flow and check for any errors or issues.
- Test with a date range to verify if the static data appears correctly in the chart.

Run Again

Expense Summary Flow

* Start Date

09-Oct-2024

* End Date

01-Oct-2024

Format: 31-Dec-2024

Next

Debug Details

How the Interview Started

Sravani Samayam (005NS000000Ta6X) started the flow interview.
API Version for Running the Flow: 62

Transaction Committed

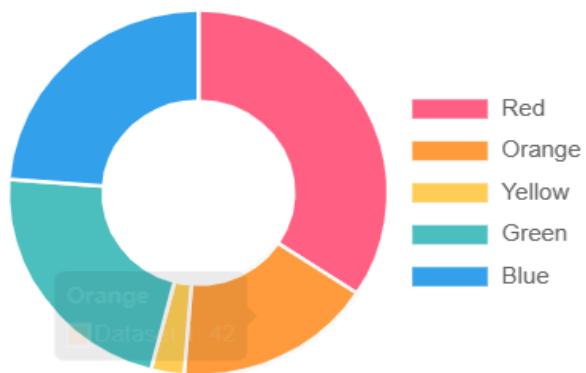
Any records that the flow was ready to create, update, or delete were committed to the database.

Run Again

Expense Summary Flow



BudgetBuddy



Previous

Finish

Debug Details

How the Interview Started

Sravani Samayam (005NS000000Ta6X) started the flow interview.

API Version for Running the Flow: 62

Transaction Committed

Any records that the flow was ready to create, update, or delete were committed to the database.

SCREEN: Date Range

Date: Start_Date

Label: Start Date

Value at run time: 2 October 2024

Date: End_Date

Label: End Date

Value at run time: 6 October 2021

Selected Navigation Button: NEXT

Transaction Committed

Any records that the flow was ready to create, update, or delete were committed to the database.

6. Key Scenarios Addressed by Salesforce in the Implementation Project

1. Expense Tracking and Categorization:

- Scenario: Users need to record expenses under specific categories (e.g., Food, Rent, Utilities) for better organization and reporting.
- Solution: Salesforce's custom objects and fields were utilized to categorize expenses by type, and Apex code was used to query and aggregate data based on these categories.

2. Date-Range-Based Expense Filtering:

- Scenario: Users require the ability to view and analyze expenses within custom date ranges to monitor monthly, quarterly, or yearly spending.
- Solution: Implemented SOQL queries with date filtering in Apex to allow for flexible data retrieval, based on user-specified start and end dates.

3. Data Visualization for Insights:

- Scenario: Users need to visualize spending trends and expense breakdowns to make informed financial decisions.
- Solution: Integrated Chart.js through LWC to display data in interactive charts, providing clear visualizations of expenses by category, period, and other parameters.

4. Customizable Expense Categories:

- **Scenario:** Users need flexibility to add, edit, or remove expense categories based on personal or business requirements.
- **Solution:** Implemented dynamic category management using custom objects and fields in Salesforce, enabling users to tailor expense categories to their needs.

5. Historical Data Analysis:

- **Scenario:** Users require access to historical data to identify trends, compare spending across years, and improve future budget planning.
- **Solution:** Archived historical expense data and enabled comparisons through custom reporting and dashboards that offer year-over-year (YoY) and month-over-month (MoM) analyses.

6. Detailed Expense Reporting by User:

- **Scenario:** Team leads or finance managers need visibility into expenses by individual users to analyze spending patterns or budget adherence.
- **Solution:** Implemented user-specific reporting with SOQL queries and custom reports, enabling managers to filter and view expenses by each team member or department.

7. Conclusion

BudgetBuddy - A CRM Application for Managing Personal and Business Expenses

BudgetBuddy is a Salesforce-based application aimed at simplifying and enhancing the management of both personal and business expenses. The application allows users to categorize, track, and analyze expenses, providing valuable insights that support budgeting and financial planning. By leveraging Salesforce's advanced capabilities, BudgetBuddy automated data processing, enables dynamic data visualization through custom dashboards, and ensures secure, real-time access to financial information.

The primary objectives of the project are to improve financial transparency, streamline expense management, and provide a user-friendly interface. With custom Salesforce objects, Apex controllers, and interactive charts, BudgetBuddy offers a robust system for organizing and understanding expenses. Comprehensive testing, including unit testing for Apex classes and triggers and UI testing, ensures the application's reliability and accuracy.

Future Enhancements:

- **AI-Powered Expense Prediction:** Integrate machine learning algorithms to predict future expenses based on historical data, helping users proactively manage budgets.
- **Mobile App Integration:** Extend BudgetBuddy's functionality to a mobile app, allowing users to record and manage expenses on the go.
- **Automated Expense Insights:** Implement automated insights and recommendations, such as identifying cost-saving opportunities and alerting users of unusual spending patterns.
- **Multi-Currency Support:** Add multi-currency options for global users to support international financial management.
- **Advanced Reporting and Export Options:** Allow users to generate and export detailed reports in various formats, enhancing data accessibility for further analysis.