Purchase and Feedback System.

User Story: When the user navigates to the product page, He should see an option to enter a referral code or coupon code during the checkout process.Upon entering a valid referral code or coupon code, the system should apply the relevant discount to my order. The checkout process should be seamless, allowing me to review my order and complete the purchase. Customers can buy products at discounted rates by using referral codes and coupons. To enhance customer interaction and satisfaction, email alerts are included to notify customers about their upcoming purchases and encourage them to review their bought items.

Project Overview : The project aims to enhance user experience and customer engagement through seamless checkout processes and email notifications. Components needed include product page integration, discount calculation logic, checkout, feedback system that collects surveys and customer reviews, email notification system, and effective data management. Overall, the project focuses on improving user experience and engagement during the purchase process.

Project Flow:

Milestone 1 : Creation of developer account
Milestone 2 : Object Creation
Milestone 3 : Tabs
Milestone 4 : The Lightning App
Milestone 5 : Fields
Milestone 6 : Creation of Page Layout
Milestone 7 : Creation of Record Types
Milestone 8 : Validation rules
Milestone 9 : SurveyFormLink
Milestone 10 : Apex
Milestone 11 : Creating of Cart LWC component

What you'll learn

1. Real Time Salesforce Project
2. Object & Relationship in Salesforce
3. Validation Rules
4. Survey Form
5. Apex
6. Apex Triggers
7. Lightning Web Components
8. Reports and Dashboards

After Creating a Salesforce Developer org Start with the First Milestone

Milestone 1: Object

What are the objects required ? Salesforce objects are database tables that permit you to store data that is specific to an organization. It consists of fields (columns) and records (rows). Custom objects that are created by users for our case are an Item object that saves the essential information that it holds i.e, price, name etc, similarly the other objects hold the information that is related to Item, Purchase, Feedback. They supply information that is essential to this project.

Activity -1

Create a custom object for Item

To create a custom object, follow these steps :

1. Click on the Gear Icon → From setup click on object manager
2. Click create, select custom object.
3. Fill in the label as "Item".
4. Fill in the plural label as "Item".

5. Record name : “Item Name"
6. Select the data type as "Text".
7. In the Optional Features section, select Allow Reports and Track Field History.
8. In the Deployment Status section, ensure Deployed is selected.
9. In the Search Status section, select Allow Search.
10. In the Object Creation Options section, select select these options:
Add Notes and Attachments related list to default page layout
Launch New Custom Tab Wizard after saving this custom object
11. Leave everything else as is, and click Save.

Activity -2

Create a custom object for Purchase
To create a custom object, follow these steps :
1. Click on the Gear Icon → From setup click on object manager
2. Click create, select custom object.
3. Fill in the label as "Purchase".
4. Fill in the plural label as "Purchase".
5. Record name : "Purchase"
6. Select the data type as "Text".
7. In the Optional Features section, select Allow Reports and Track Field History.
8. In the Deployment Status section, ensure Deployed is selected.
9. In the Search Status section, select Allow Search.
10. In the Object Creation Options section, select select these options:
    Add Notes and Attachments related list to default page layout
    Launch New Custom Tab Wizard after saving this custom object
11. Leave everything else as is, and click Save.

Activity - 3

Create a custom object for Feedback
1. Click on the Gear Icon → From setup click on object manager
2. Click create, select custom object.
3. Fill in the label as “Feedback".

4. Fill in the plural label as "Feedback".
5. Record name : "Feedback"
6. Select the data type as "Text".
7. In the Optional Features section, select Allow Reports and Track Field History.
8. In the Deployment Status section, ensure Deployed is
   selected.
9. In the Search Status section, select Allow Search.
10. In the Object Creation Options section, select select these options:
    Add Notes and Attachments related list to default page layout
    Launch New Custom Tab Wizard after saving this custom object
11. Leave everything else as is, and click Save.

Milestone 2 : Tabs

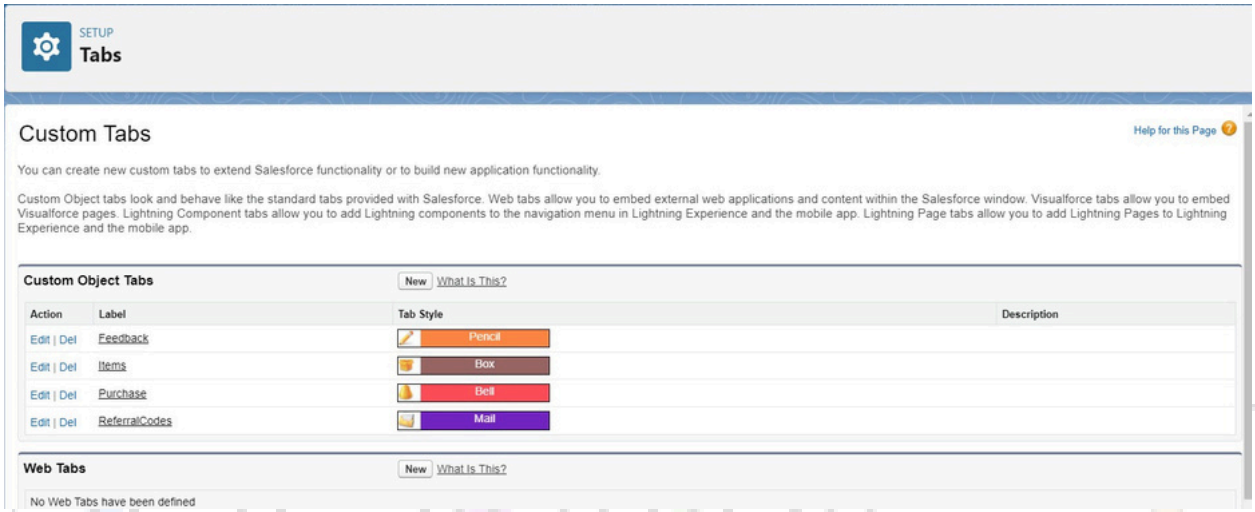
What is the requirement of Tab? Tab is a user interface element that allows users to navigate to different sections of the platform, such as Item, Purchase and Feedback. Tabs are used to access custom objects and custom pages. These are located at the top of the screen and can be customized to fit the needs.

Activity - 1

How to create a tab

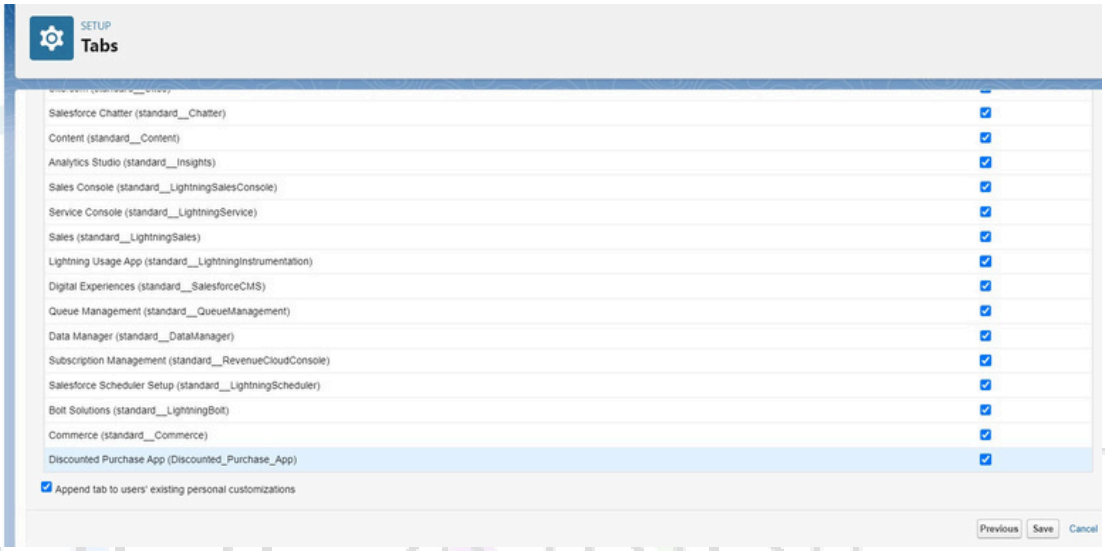
As we selected to launch a custom tab wizard in step 10, a custom tab wizard appears wherein We customize the look of the "Item". object's tab. To do that :

1. To Create the tab → click on the gear Icon → then click Home button → In the quick find box search for the Tabs.



2. Then Click on the new button to create tab, in the Tab style you can select whatever you want to select

3. Then select the Item Object and select the Icon from the search button and click next next and save.



For creating the Remaining tabs follow the above steps

a. ForcreatingatabforFeedbackTabfollowtheabovestepswhicharebeingcreatedinstep1 for Item.

b. For creating a tab for Referral Code Tab follow the above steps which are being created in step1 for customer Item.

Milestone 3:Fields & Relationships

Fields represent the data stored in the columns of a relational database. It can also hold any valuable information that you require for a specific object. Hence, the overall searching, deletion, and editing of the records become simpler and quicker.

Types of Fields

- Standard Fields
- Custom Fields

Standard Fields:

As the name suggests, the Standard Fields are the predefined fields in Salesforce that perform a standard task. The main point is that you can't simply delete a Standard Field until it is a non-required standard field. They are

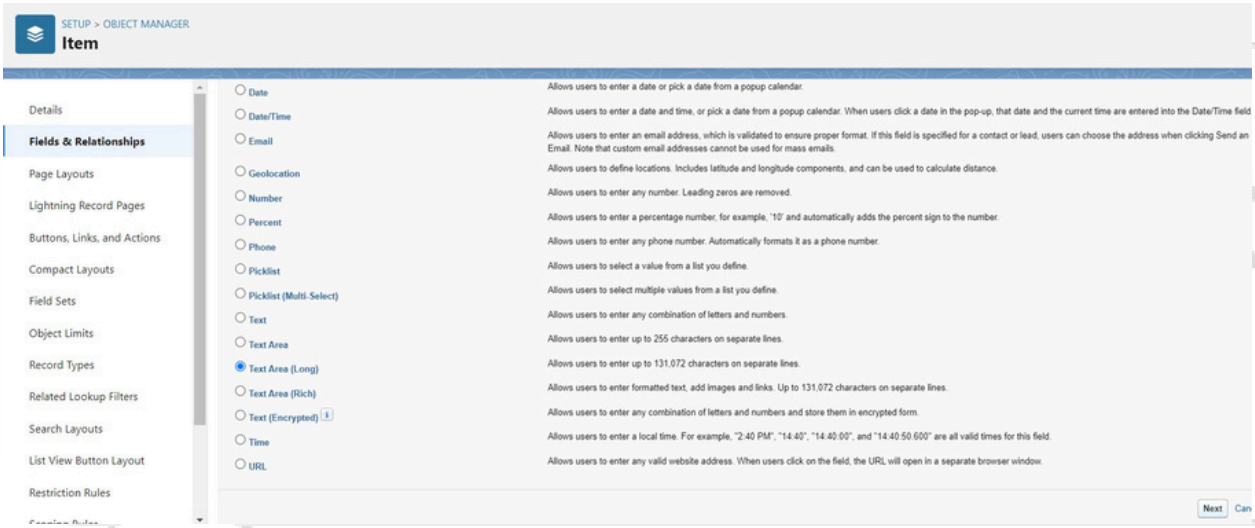
- Created By
- Owner
- Last Modified

Custom Fields:

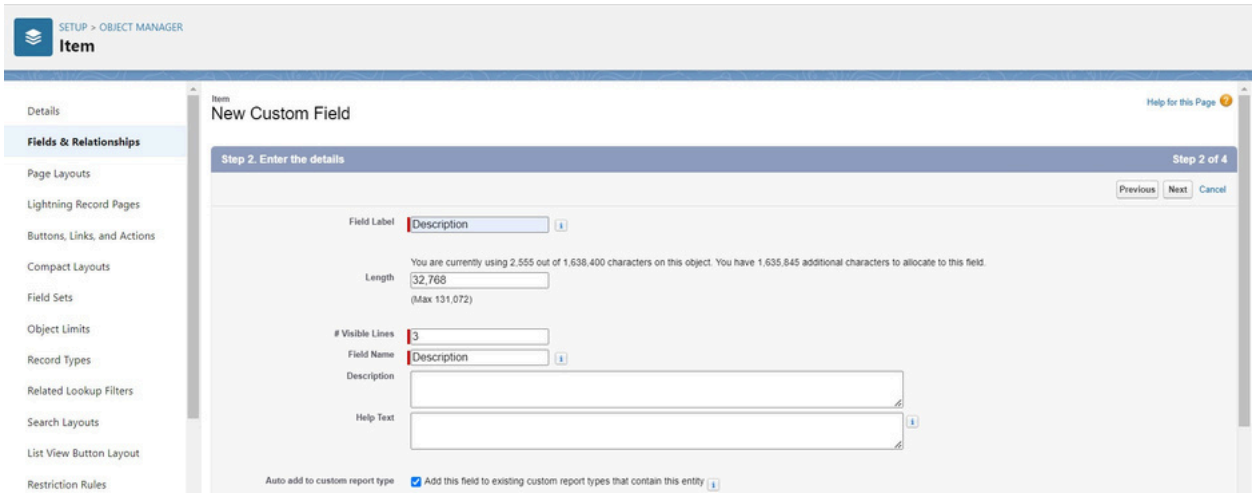
Custom Fields are highly flexible, and users can change them according to requirements. Moreover, each organizer or company can use them if necessary. It means you need not always include them in the records, unlike Standard fields. Hence, the final decision depends on the user, and he can add/remove Custom Fields of any given form.

Activity -1

1. Go to setup → click on Object Manager → type object name in search bar → click on the object→ “Item” → Field and Relationship→ then click on new →Field Data Type(Long Text Area)



2. Click On Next→ Field Label →Description→ next→ next→ Save. Always require a value in this field in order to save a record.

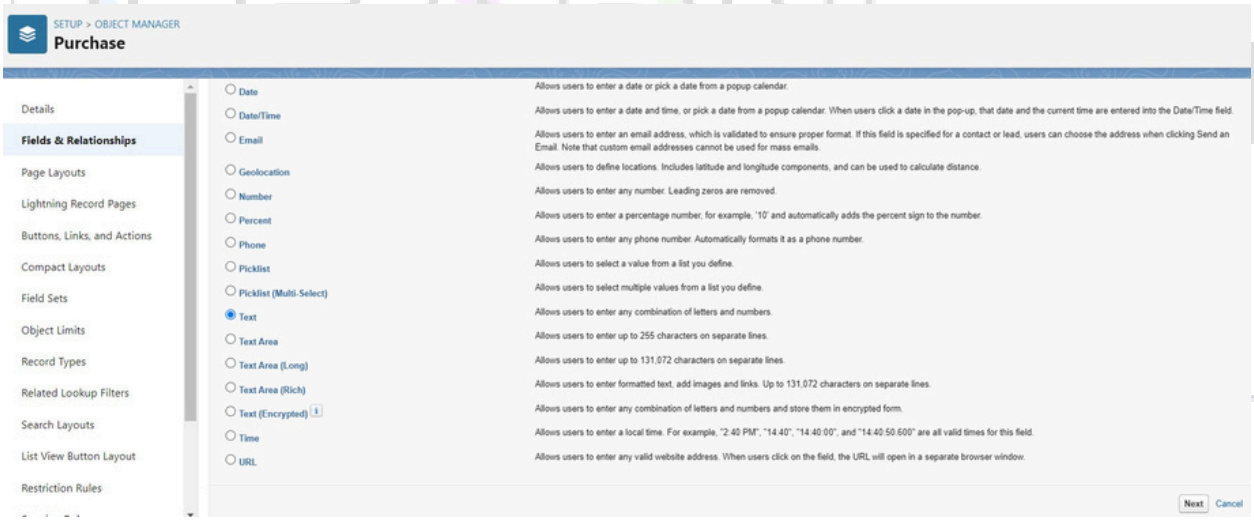


3. Follow the above steps and create the Remaining Fields.

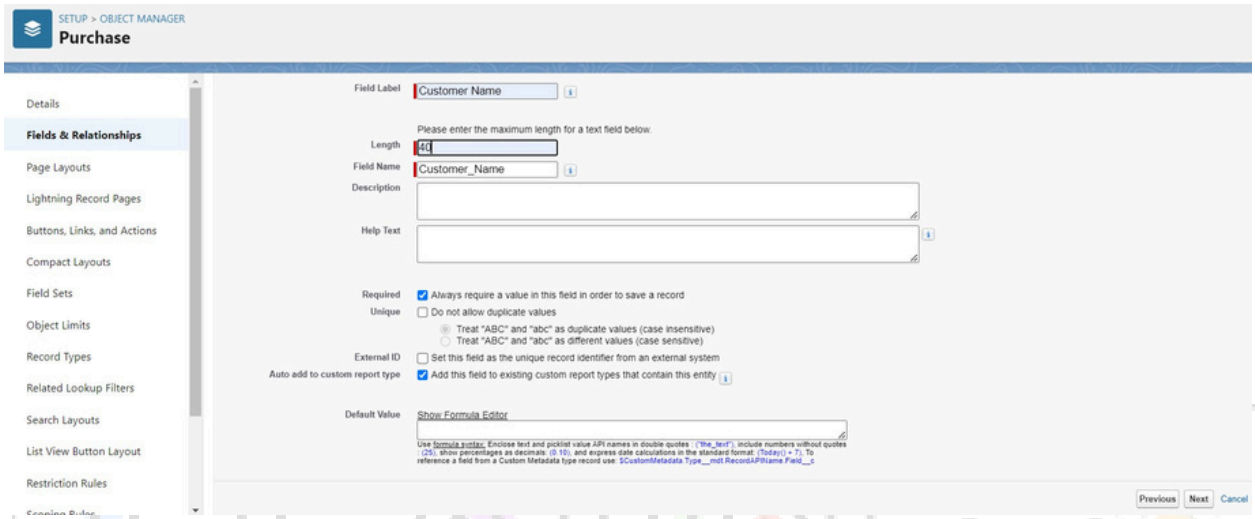
- Item Price (Currency) - Always require a value in this field in order to save a record.
- Quantity Available(Number) - Always require a value in this field in order to save a record.
- Weight (Number) - Always require a value in this field in order to save a record.
- Image(Rich Text Area) - Always require a value in this field in order to save a record.
- Item Category (Picklist) - Add category of items in a new line.

Activity - 2

1. Go to setup → click on Object Manager → type object name in search bar → click on the object → “Purchase” → Field and Relationship→ then click on new →Field Data Type(Text)



2. Click On Next→ Field Label → Customer Name→ next→ next→ Save. Always require a value in this field in order to save a record.



3. Follow the above steps and create the Remaining Fields.
   - OrderAmount(Currency)-Length10Decimal4andsave.
   - CustomerContact(Phone)-Alwaysrequireavalueinthisfieldinordertosavearecord.
   - CustomerEmailId(Email)-Alwaysrequireavalueinthisfieldinordertosavearecord.
   - Address(LongTextArea)- Alwaysrequireavalueinthisfieldinordertosavearecord.
   - City(Text)- Alwaysrequireavalueinthisfieldinordertosavearecord.
   - State(Text)- Alwaysrequireavalueinthisfieldinordertosavearecord.
   - PurchaseDate(Date)-Alwaysrequireavalueinthisfieldinordertosavearecord.

4. Click On Save & New→ Master Detail Relationship → Select “Item” from related to list→ next→Field label Item→Save.

Activity -3

1. Go to setup → click on Object Manager → type object name in search bar → click on the object→ “Feedback” → Field and Relationship→ then click on new →Field Data Type(Text Area Rich).
2. Click On Next→ Field Label→Review/Comments Name→ next→next→Save. Always require a value in this field in order to save a record.
3. Click On New→ Number→Rating→ next→next→Save.
4. Click On Save & New→ Master Detail Relationship → Select “Item” from related to list→ next→Field label Item→Save.
5. Click On Save & New→ Lookup Relationship → Select “Purchase” from related to list→ next→Field label Item→Save.

Milestone 4 : Create App

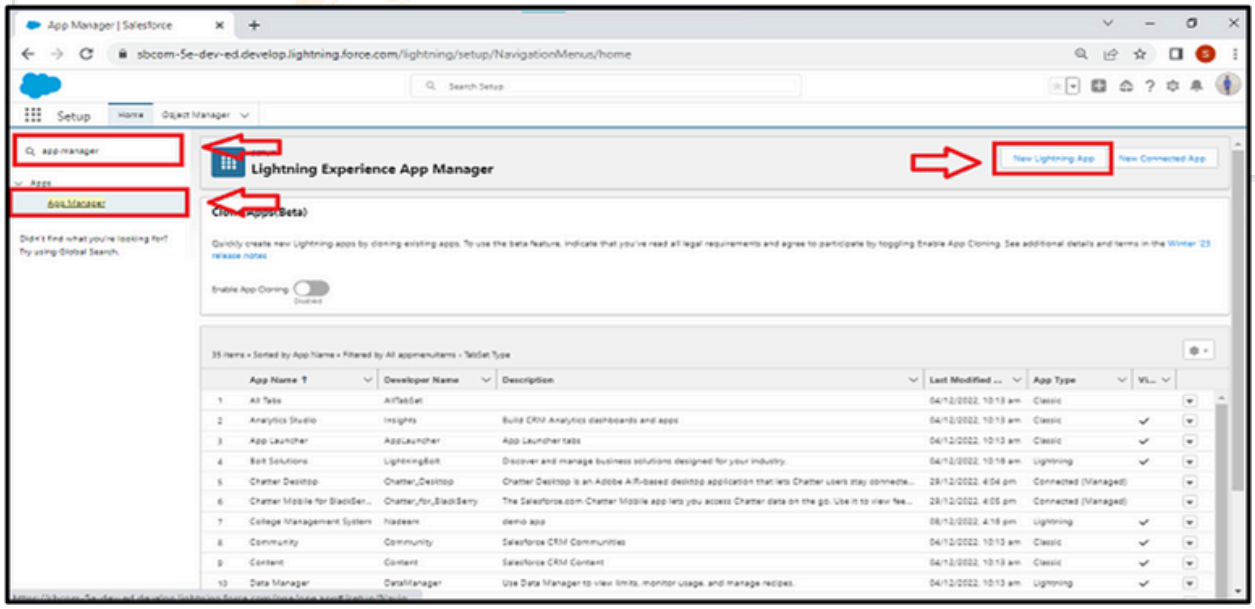
Creating a DiscountedPurchase app involves designing a solution that helps manage and track the customer feedback after purchase. The app is built in Lightning Experience , depending on your organization's preferences and requirements. Below is a brief overview of how you might design such an app :

- DesignaLightningAppusingtheLightningAppBuilder.
- IncludetabsfortheItemobject,standardobjectslikePurchaseetc,andcustom Lightning components.
- Customizetheapp'scolorschemeandlogotogiveitabrandedlook.

Activity -1

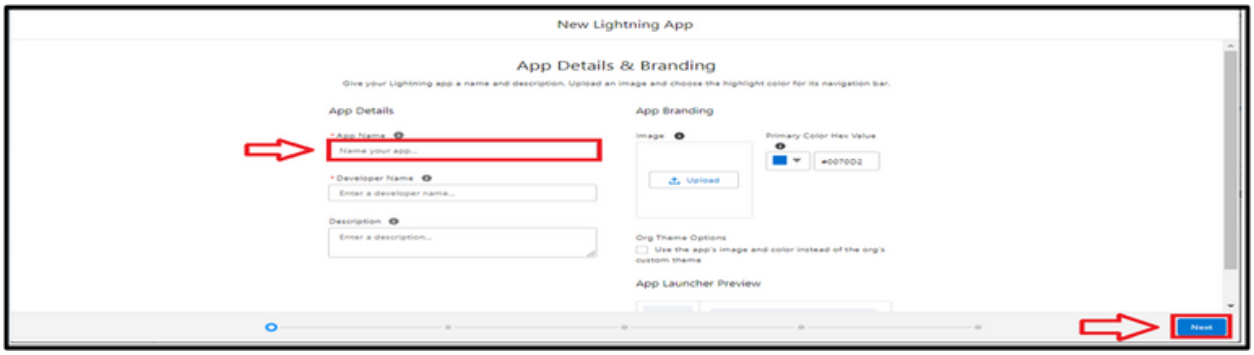
Create the Lightning App

1.Go to the Gear Icon → Click on setup → click on the Home Button → Go to the quick find box and select the App Manager



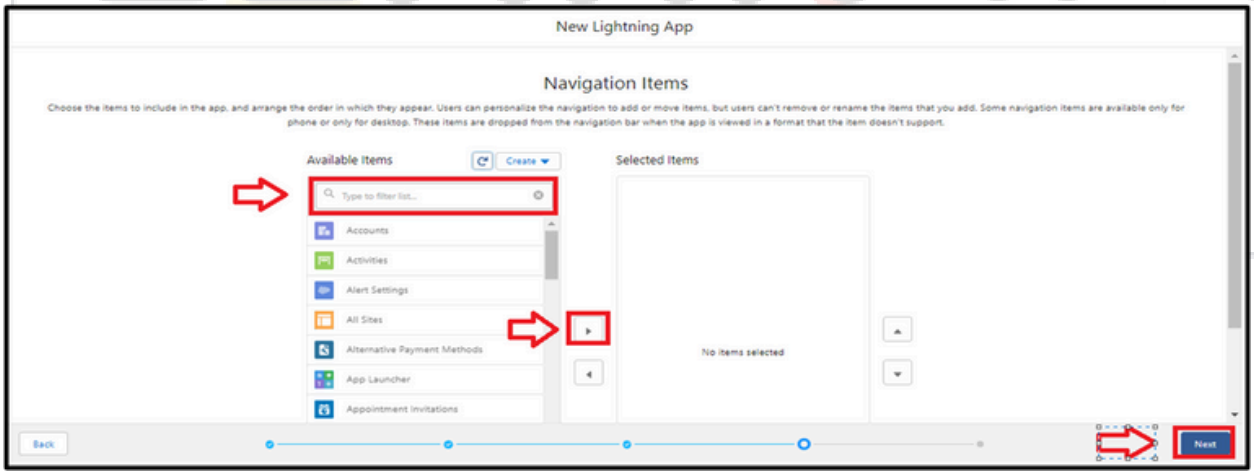
2.Fill the app name as an in app details and branding →Next → (App option page) keep it as default → Next

3. Utility Items keep it as default → Next → (Add Navigation Items)(add tabs Item, Purchase, Feedback, ReferralCode) → Next → (Add User Profile) Add System Administrator, Salesforce platform user, Standard User → Next.



4. To Add Navigation Items:
Select the items from the search bar and move it using the arrow button → Next. select all the tabs which you have created

5. To Add User Profiles:
Search profiles in search bar → click on the arrow button & select Standard user, standard Platform user & System Admin Profile→ save & finish.



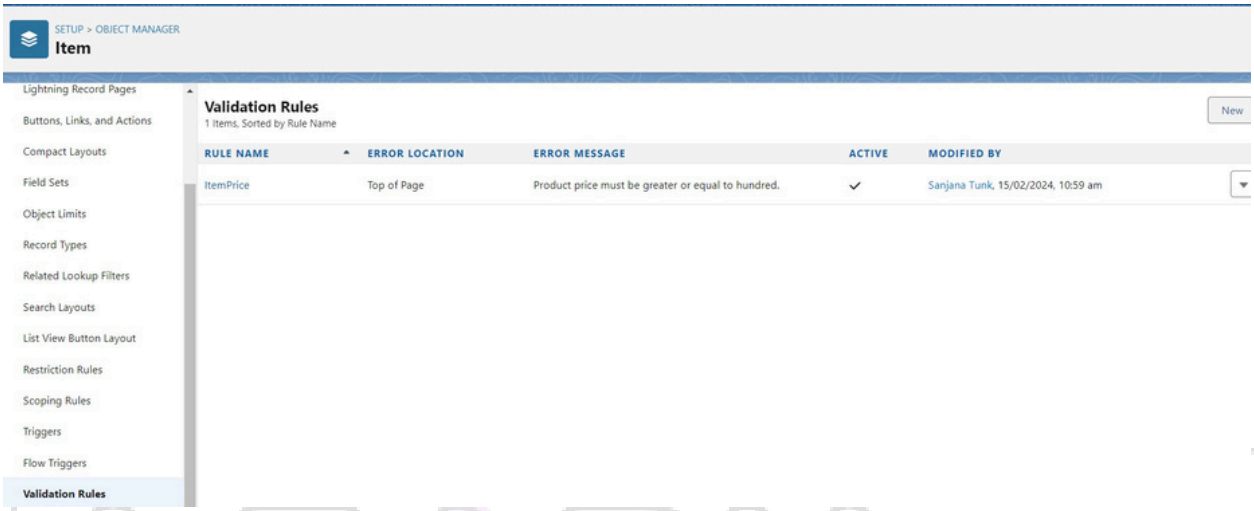
Milestone 5: Validation Rules

Validation rules could ensure that product data meets specific criteria, such as ensuring that the price of a product is non-negative. If feedback is collected as part of the process, validation rules could be used to enforce constraints on feedback data, such as limiting the length of comments or requiring certain fields to be populated.

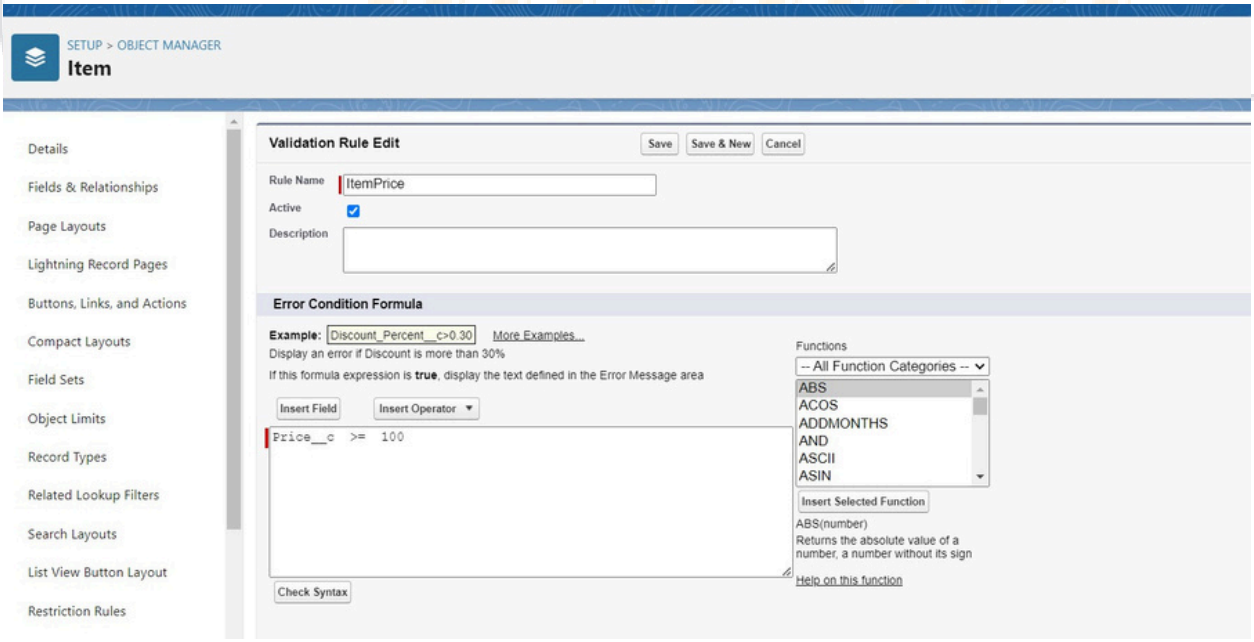
Activity -1

To create a validation rule on Item Object
1. Go to the setup page → click on object manager → From drop down click edit for Item object.

2. Click on the validation rule → click New



3. Enter the Rule name as “IItemPrice”.
4. Insert the Price Condition Formula as :
   Price__c >= 100
5. Enter the Error Message as “ Product price must be greater or equal to hundred. ”, select the Error location as Top of Page and click Save.



6. Create the Validation Rule for Purchase Object named “OrderDateNotInFuture” using the formula [Purchase_Date__c > TODAY()]Similarly by following the Activity 1 Steps.

7. Create the Validation Rule for Feedback Object named “RatingValidation” using the formula [Rating__c < 1 || Rating__c > 5]Similarly by following the Activity 1 Steps.

Milestone 6: Profile

A profile is a group/collection of settings and permissions that define what a user can do in salesforce. Profile controls “Object permissions, Field permissions, User permissions, Tab settings, App settings, Apex class access, Visualforce page access, Page layouts, Record Types, Login hours & Login IP ranges. You can define profiles by the user's job function. For example System Administrator, Developer, Sales Representative.

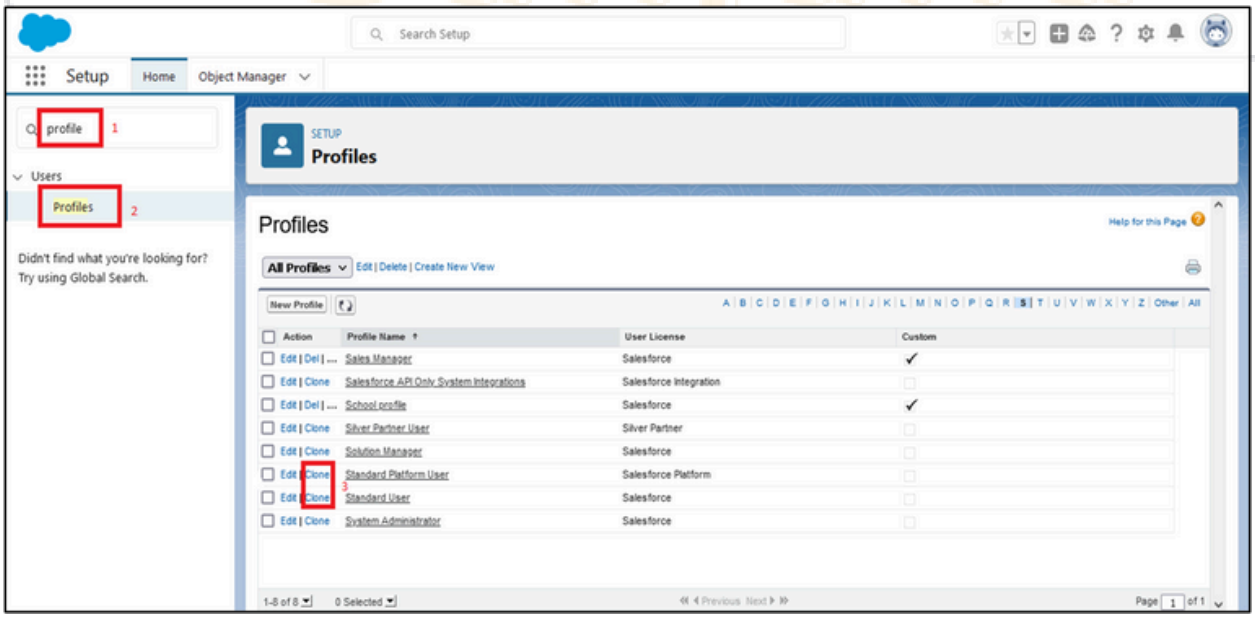
Custom Profiles:

- Custom ones defined by us i.e; Customer Account.
- They can be deleted if there are no users assigned with that particular one

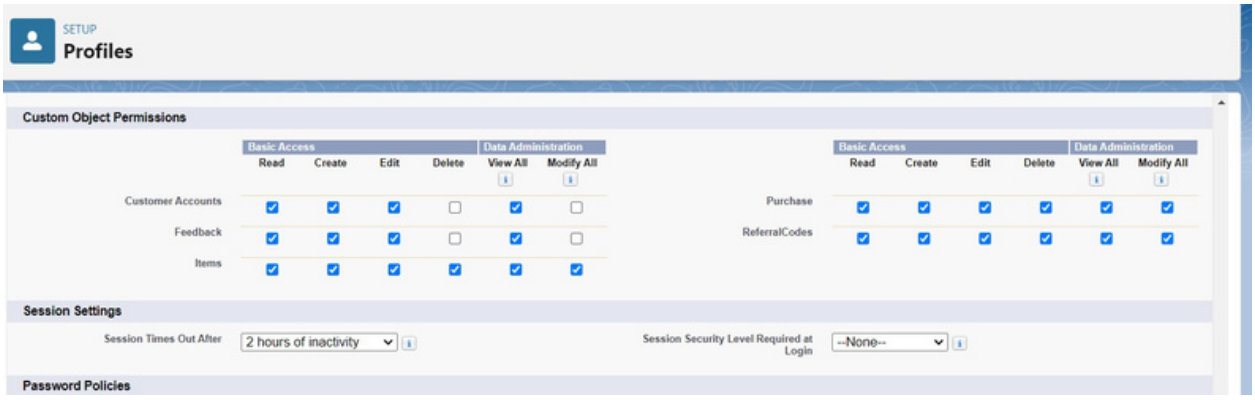
Activity-1

To create a new profile:

1. Go to setup → type profiles in quick find box → click on profiles → clone the desired profile (System Administrator user is pref) and clone that profile



2. Enter a Profile Name(Customer Account) And click on Save
3. Click on the new created profile
4. While still on the profile page, then click Edit.
5.For the Delivery Personnel profile give the following access and save it.



Milestone-7: User Adoption

As a new administrator, you perform user management tasks like creating and editing users, resetting passwords, granting permissions, configuring data access, and much more. In this unit, you will learn about users and how you add users to your Salesforce org. Every user in Salesforce has a user account. The user account identifies the user, and the user account settings determine what features and records the user can access. Each user account contains at least the following:

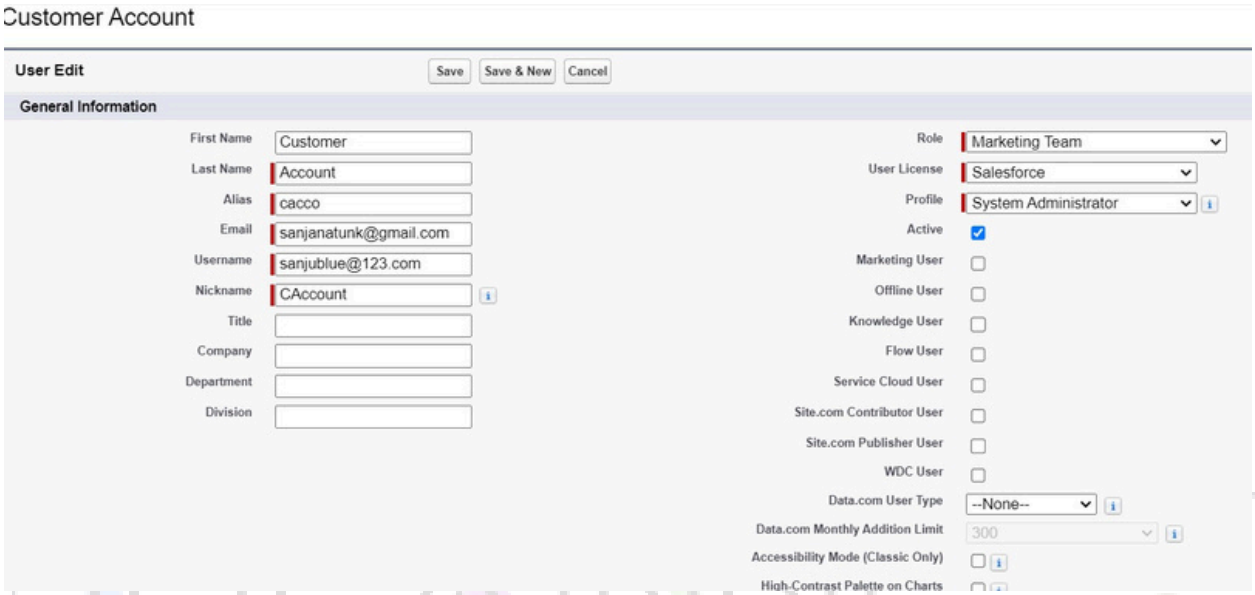
- Username
- EmailAddress
- User'sFirstName(optional)
- User'sLastName
- Alias
- Nickname
- License
- Profile
- Role(optional)

Now create the user with the profile which we have created.

Activity - 1

User: Customer Account

1.Go to the Gear Icon→ Click the setup→click on the home button and in the quick find box search for the user →Click on the user→ then click on new →fill the fields.

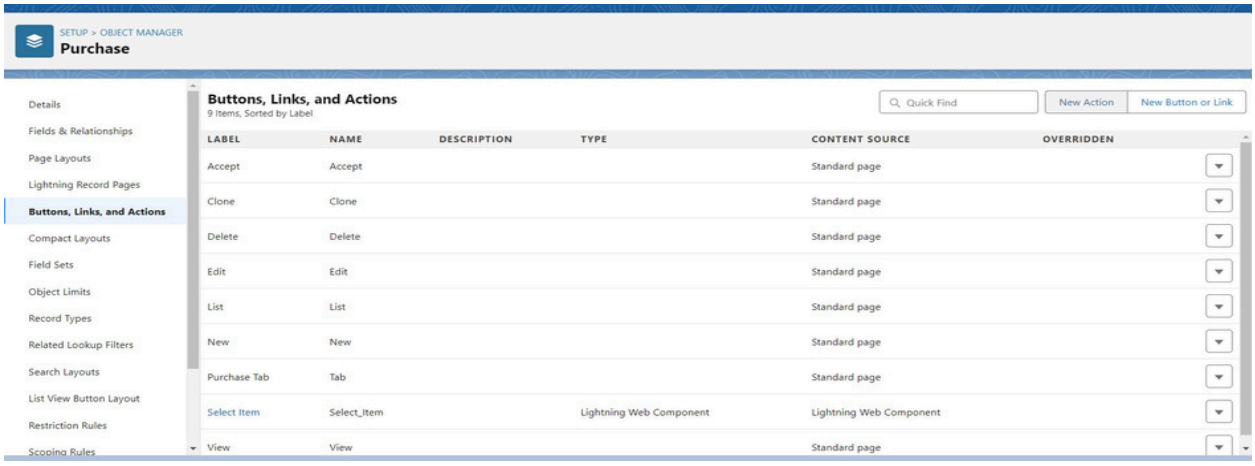


Milestone-8: Creating Buttons, Links and Actions

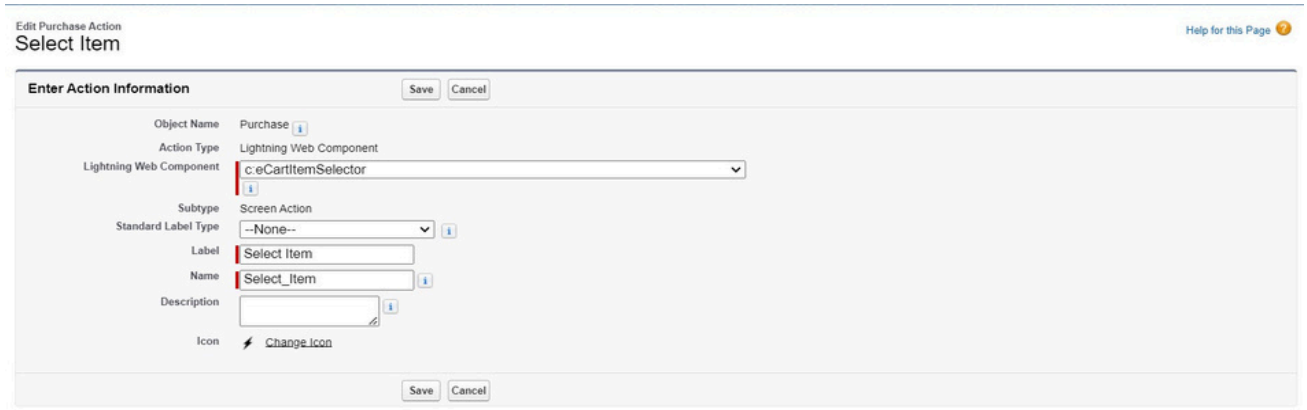
Activity - 1

To create a Action Purchase Object

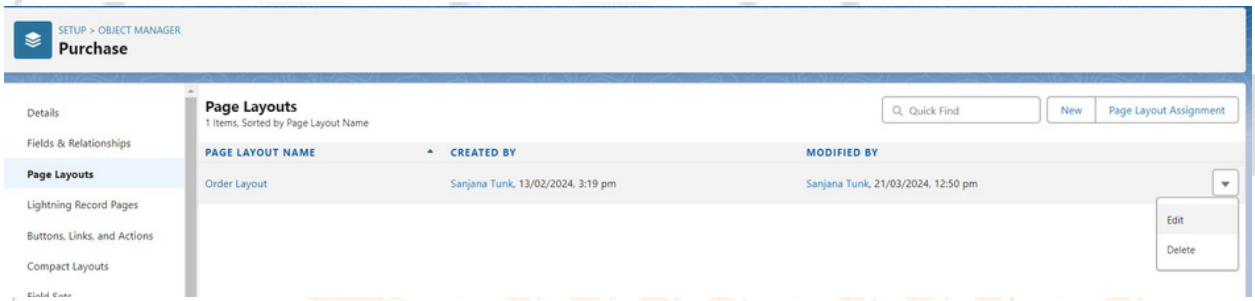
1. Go to the setup page → click on object manager → From drop down click edit for Purchase object.
2. Click on the Buttons, Links and Actions → click New Action.



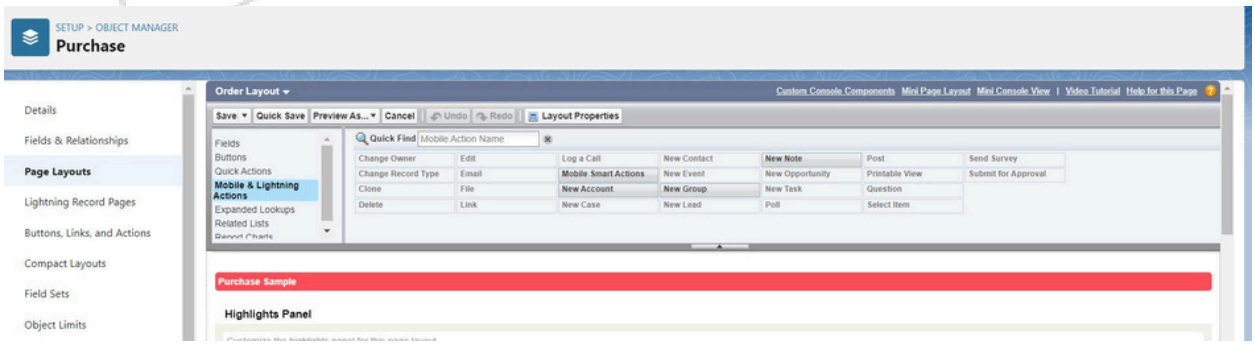
3. Fill in the details as

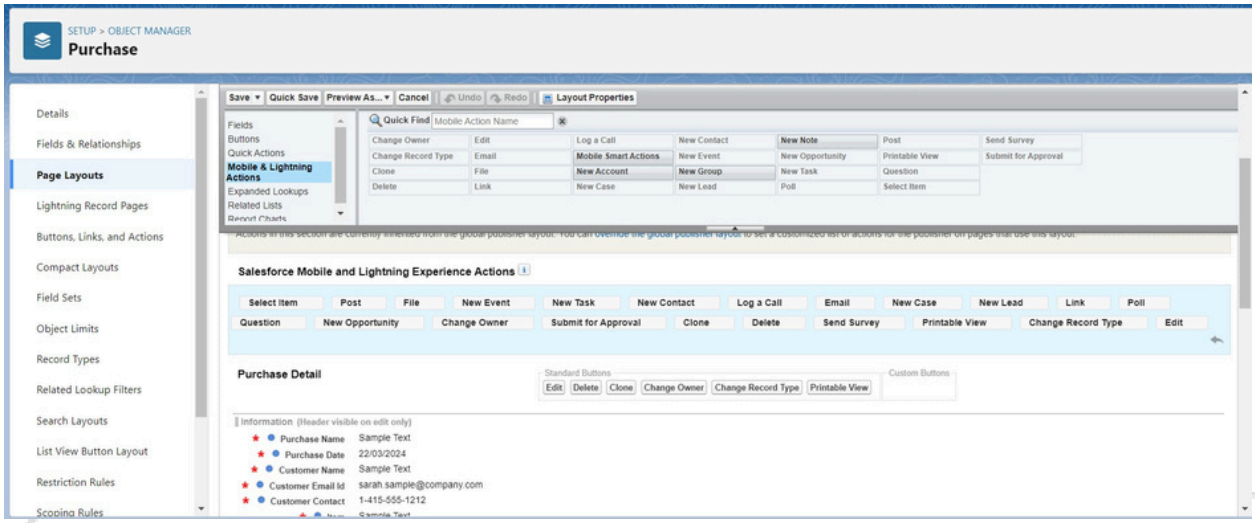


4. Click on Page Layout → click Edit.



5. Click on Mobile and Lightning Actions → You will find the SelectItem Action that you previously created, now drag and drop it in the Salesforce Mobile and Lightning Experience Actions Click Save.



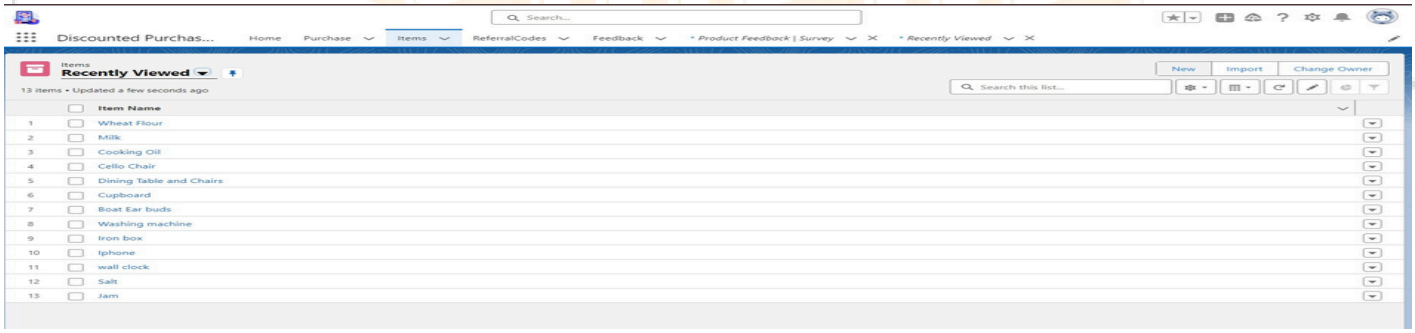


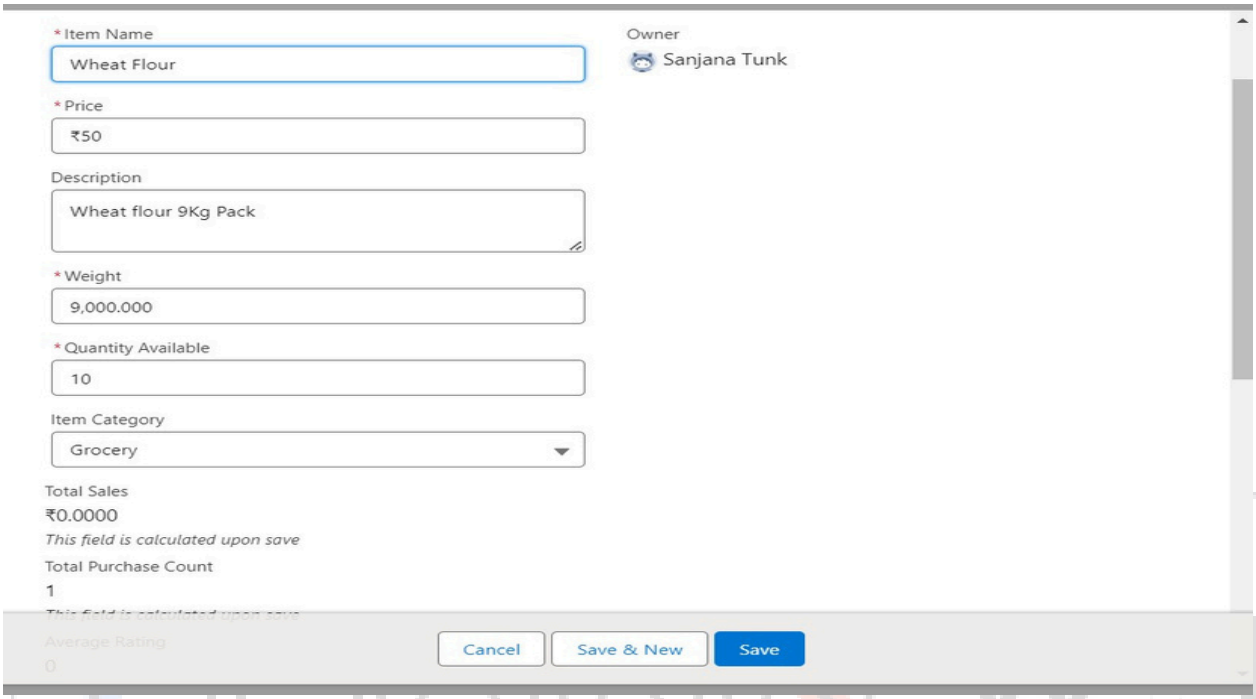
Milestone-9: Create Records

Activity - 1

To create records on Item Object

1.Go to the App Launcher→ Click Discount Purchase App → click on the Items Tab find box search → then click on new →fill the fields





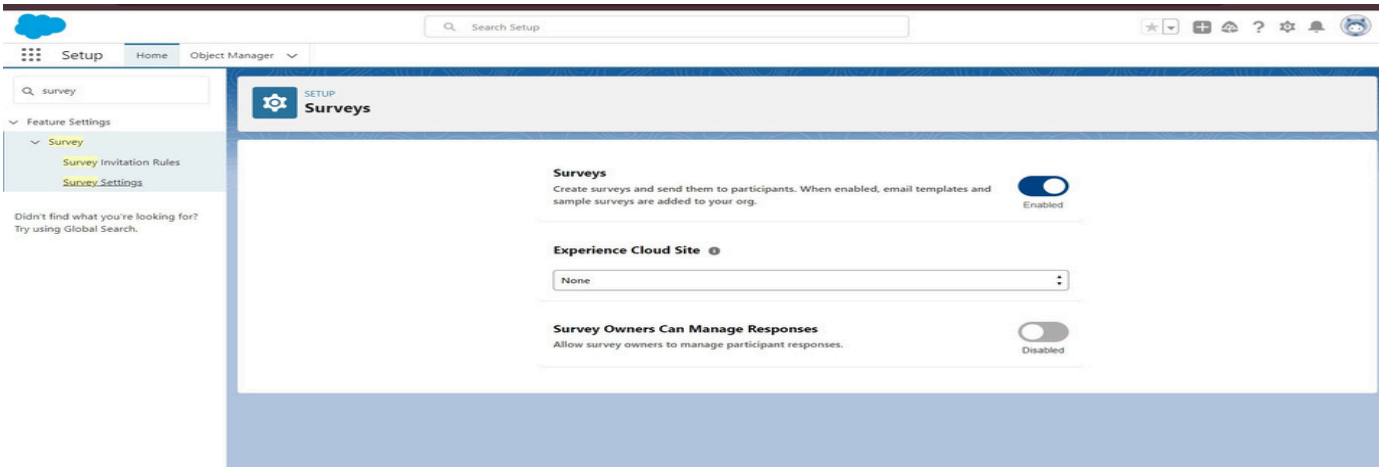
Milestone 10 : SurveyFormLink

Activity -1

Create a Form for Survey

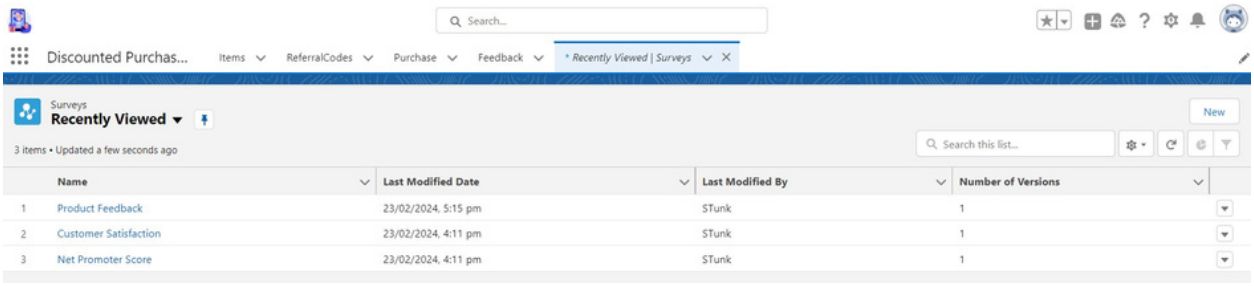
To create a custom object, follow these steps :

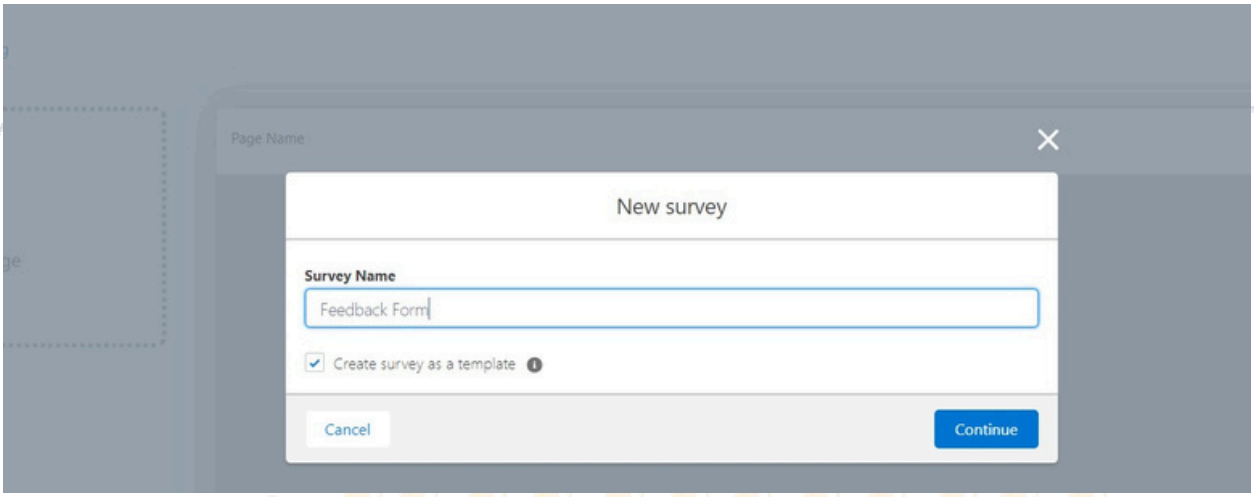
1. Click on the Gear Icon → From setup.
2. Click quick find, Survey settings.



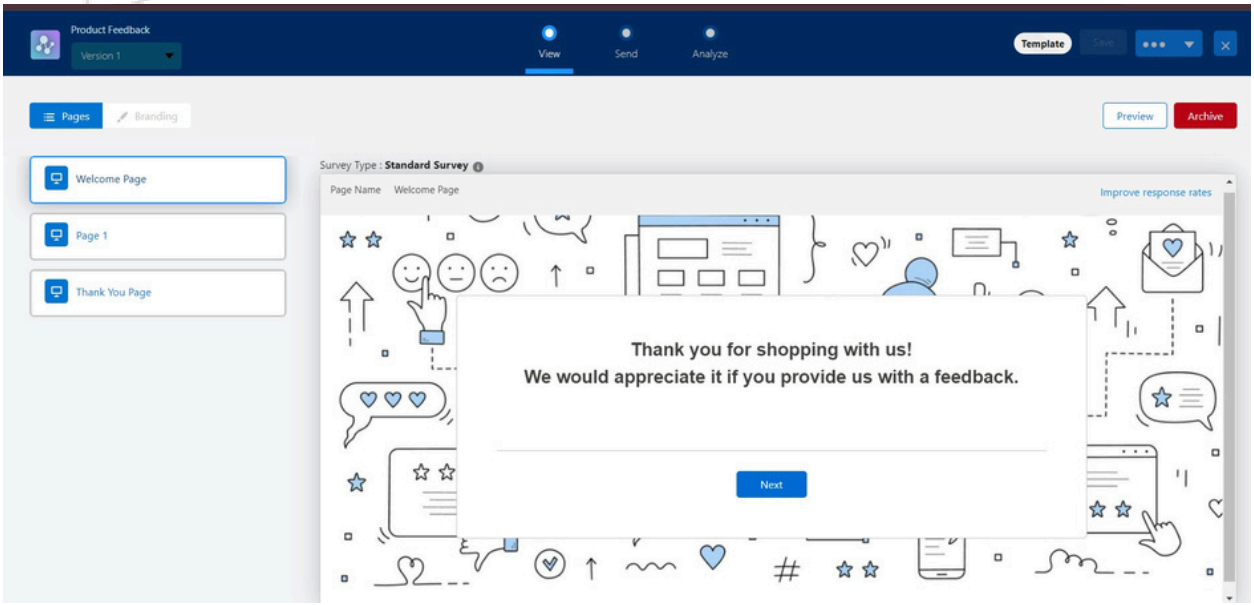
3. Click the App Launcher.
4. Search for "Survey".

5. Click on New to create one.

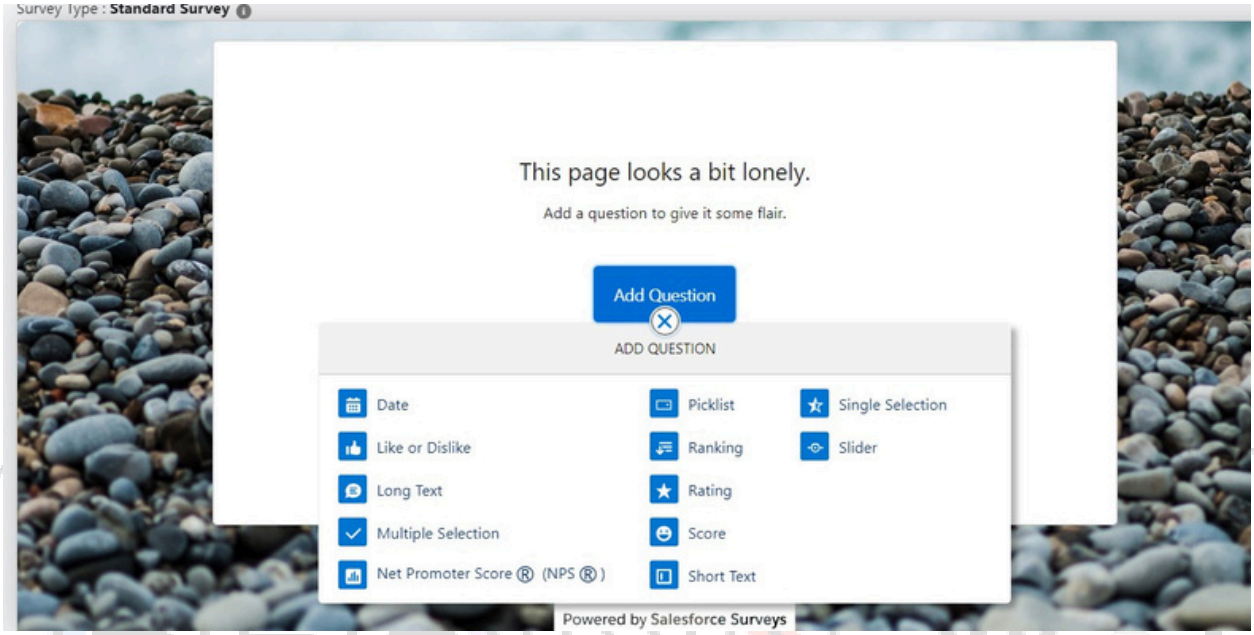




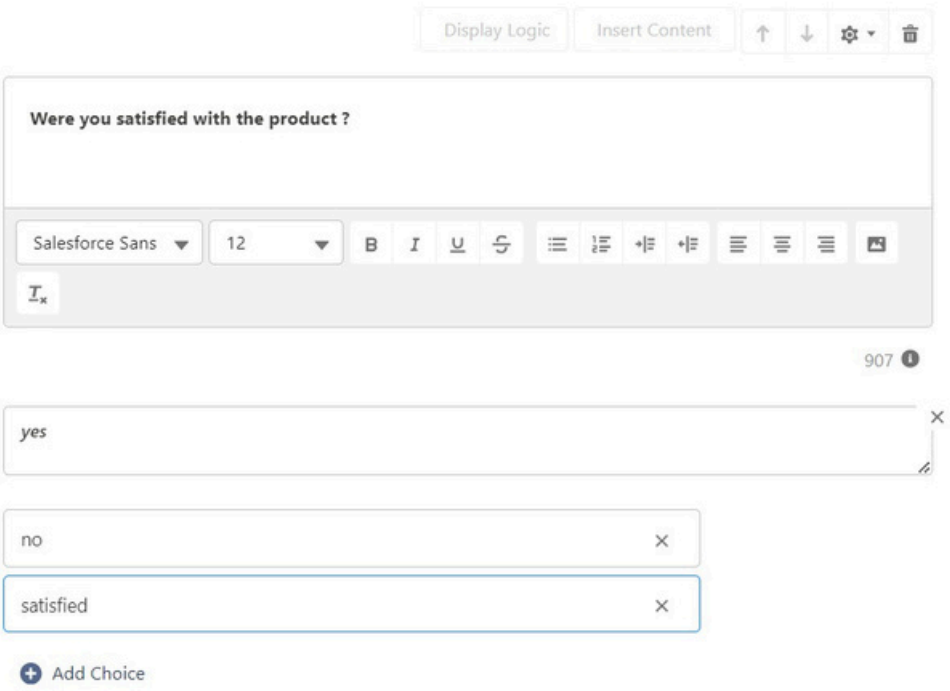
6. Fill the Survey Name as "Feedback Form".
7. Click on Create survey as a Template, then continue.



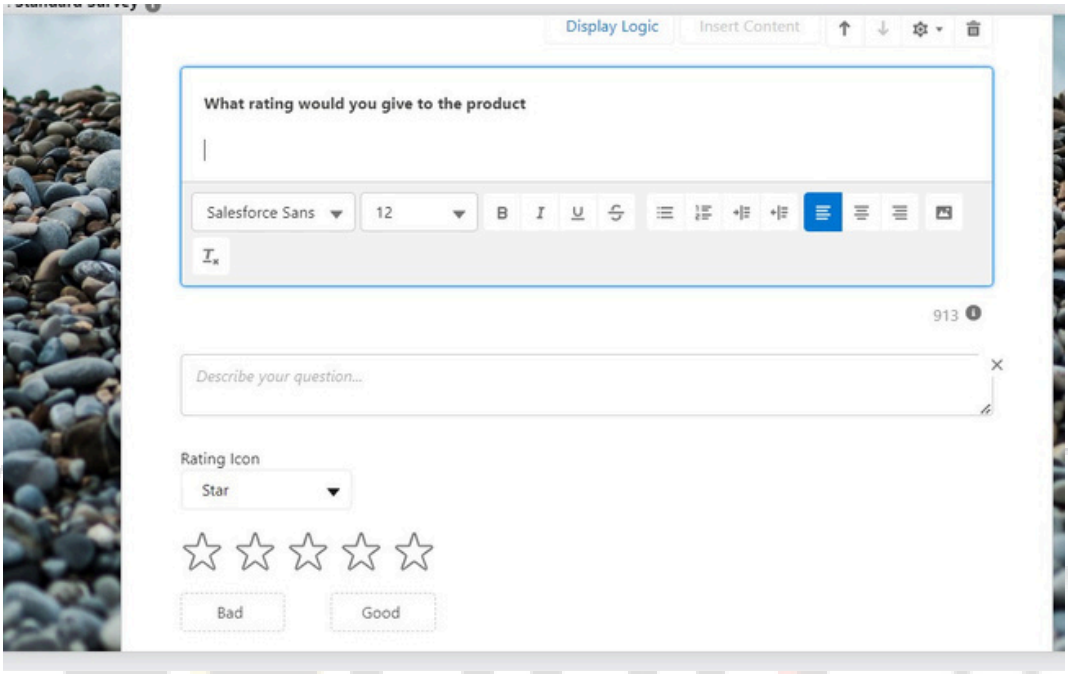
8. Click on Add Page, Then click on add question.



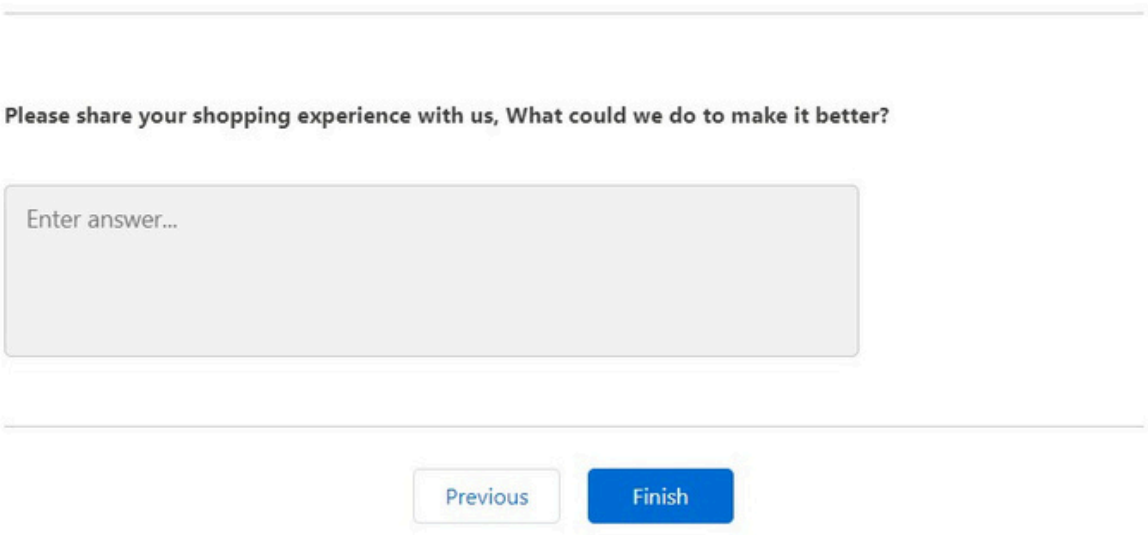
9. Choose multiple selection then enter your desired question



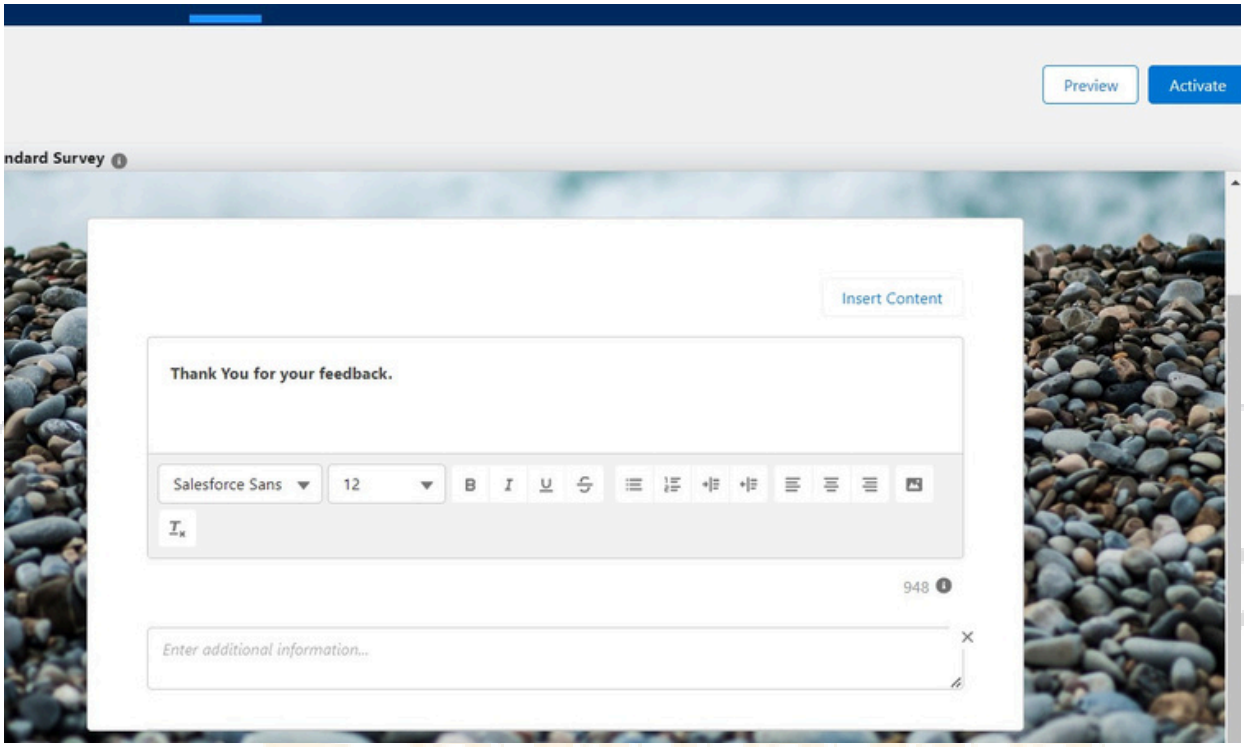
10. Click Add question, Select Rating, define the question



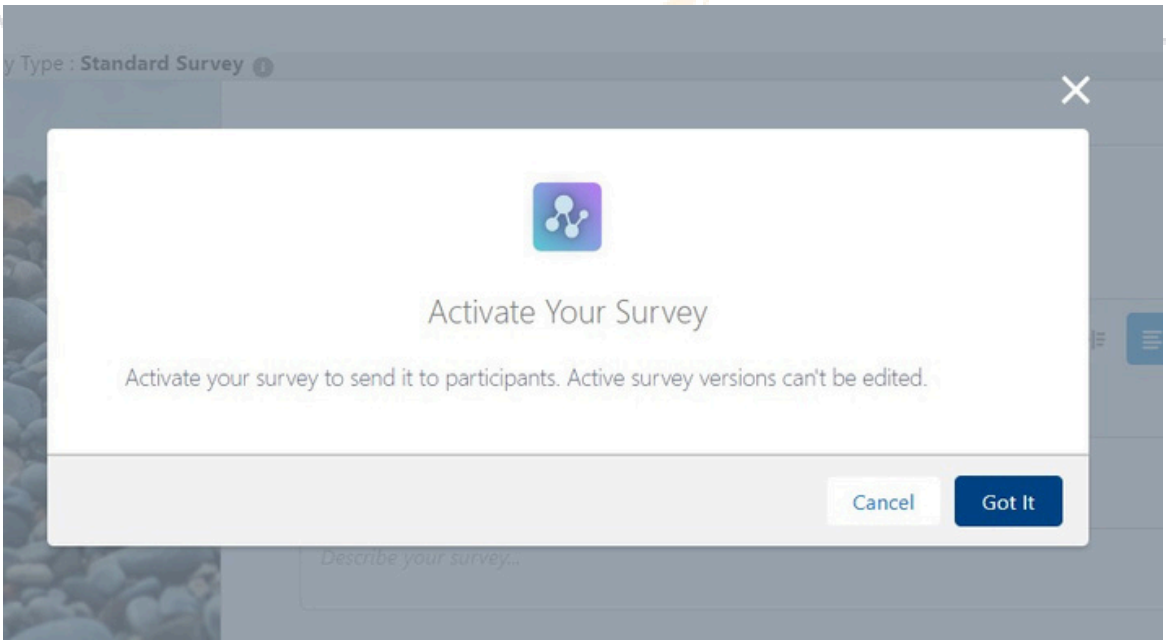
11. Click Add question, Select Long Text, define the question.



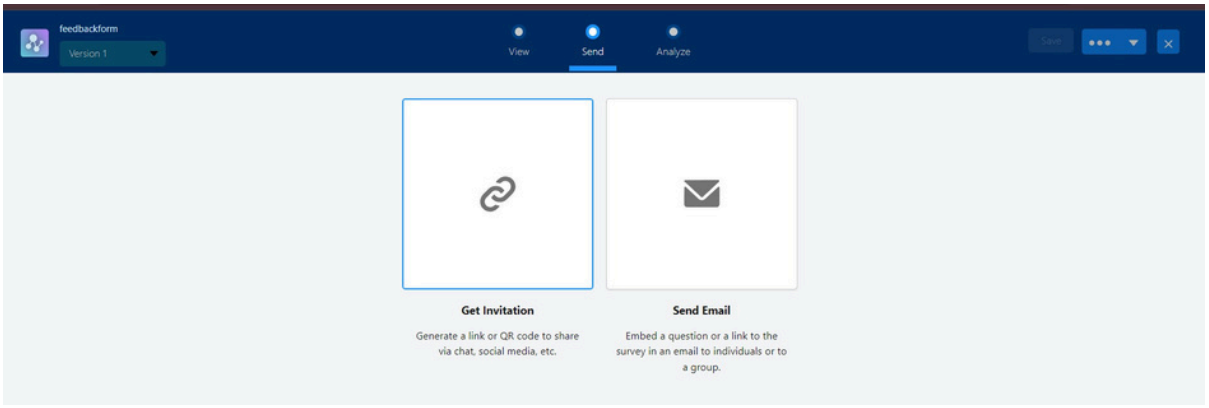
12. Click on Add Page, Thenadd a thankyou message.

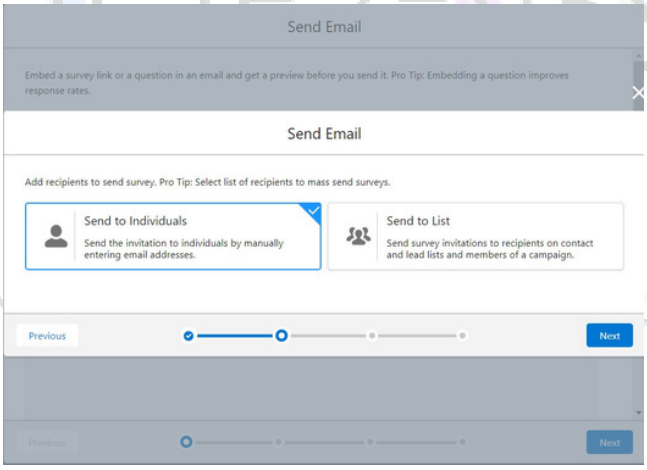


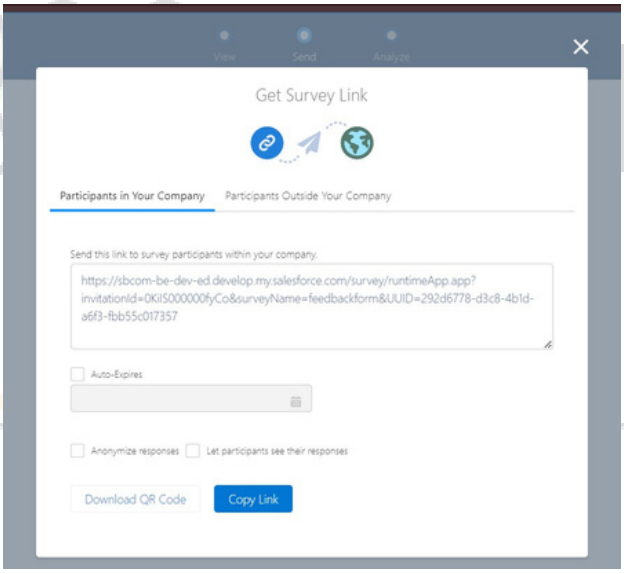
13. Activate the survey.

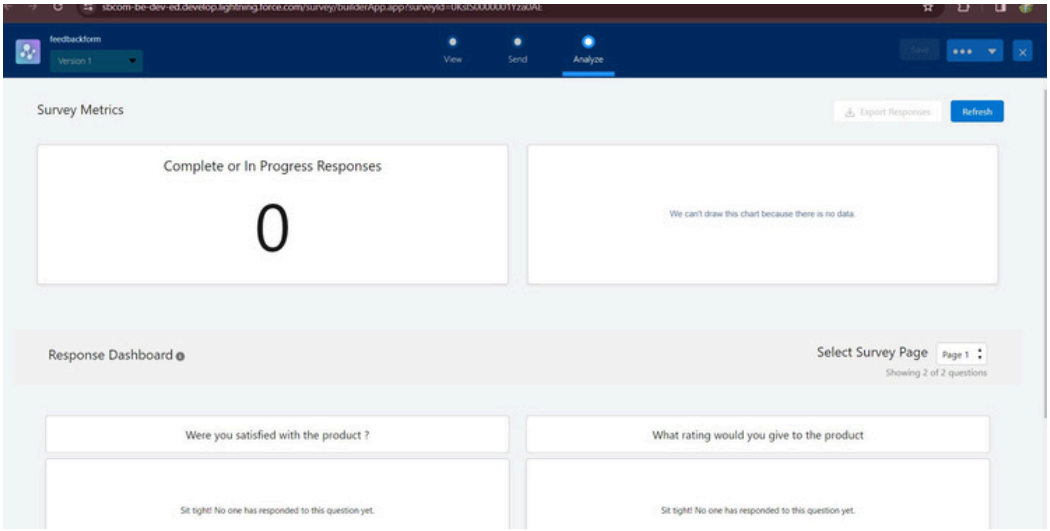


14. Click Send, You can either copy the link or send it through mail. And then click on Analyze to check the response received.


feedbackform
Version 1
View
Send
Analyze
Get Invitation
Generate a link or QR code to share via chat, social media, etc.
Send Email
Embed a question or a link to the survey in an email to individuals or to a group.


Send Email
Embed a survey link or a question in an email and get a preview before you send it. Pro Tip: Embedding a question improves response rates.
Send Email
Add recipients to send survey. Pro Tip: Select list of recipients to mass send surveys.
Send to Individuals
Send the invitation to individuals by manually entering email addresses.
Send to List
Send survey invitations to recipients on contact and lead lists and members of a campaign.
Previous
Next

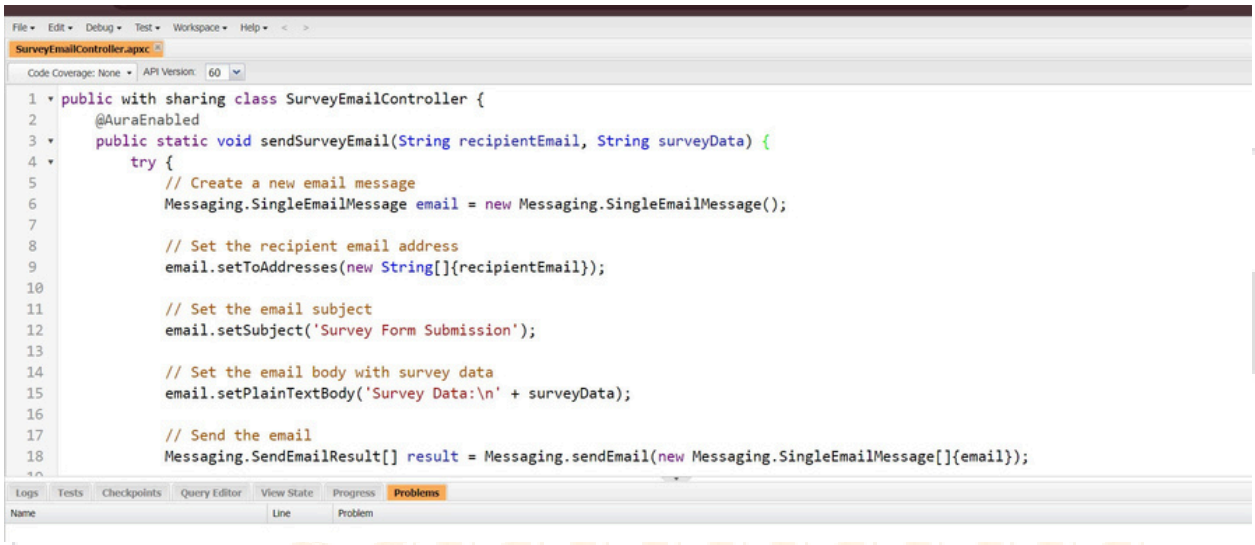

Get Survey Link
Participants in Your Company
Participants Outside Your Company
Send this link to survey participants within your company.
https://sbcom-be-dev-ed.develop.my.salesforce.com/survey/runtimeApp.app?invitationId=0KiS000000fyCo&surveyName=feedbackform&UUID=292d6778-d3c8-4b1d-a6f3-fbb55c017357
Auto-Expires
Anonymize responses
Let participants see their responses
Download QR Code
Copy Link


feedbackform
Version 1
View
Send
Analyze
Survey Metrics
Refresh
Complete or In Progress Responses
0
We can't draw this chart because there is no data.
Response Dashboard
Select Survey Page
Page 1
Showing 2 of 2 questions
Were you satisfied with the product ?
What rating would you give to the product
Sit tight! No one has responded to this question yet.
Sit tight! No one has responded to this question yet.

Milestone-11: Apex Class

Activity- 1 SurveyEmailController Class

Defining the logic for automating actions based on Apex class to handle the email notifications on the survey. It includes methods for sending notifications for these stages.



Apex Class :

```
public with sharing class SurveyEmailController {
        @AuraEnabled
        public static void sendSurveyEmail(String recipientEmail, String surveyData) {
           try{
              // Create a new email message
              Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();

              // Set the recipient email address
```

```
            email.setToAddresses(new String[]{recipientEmail});

            // Set the email subject
            email.setSubject('Survey Form Submission');

            // Set the email body with survey data
            email.setPlainTextBody('Survey Data:\n' + surveyData);

            // Send the email
            Messaging.SendEmailResult[] result = Messaging.sendEmail(new
Messaging.SingleEmailMessage[]{email});

            // Check the result of the email sending operation
            if (result[0].success) {
                System.debug('Email sent successfully');
            } else {
                System.debug('Error sending email: ' + result[0].errors[0].message);
                throw new AuraHandledException('Error sending email: ' +
result[0].errors[0].message);
            }
        } catch (Exception e) {
            // Handle exceptions and provide appropriate error handling
```

```
            System.debug('Error sending email: ' + e.getMessage());
            throw new AuraHandledException('Error sending email: ' + e.getMessage());
        }
    }
}
```

Activity- 2 eCartController Class

Defining the logic for automating actions based on Apex class to handle the Item in the cart ,the logic included are add to cart remove from cart and price details.

```
public with sharing class eCartController {
    @AuraEnabled(cacheable=true)
    public static List<Item__c> getItemsByCategory(String category) {
        try {
            return [SELECT Id, Name, Price__c, Image__c,Quantity_Available__c FROM Item__c WHERE Item_Category__c = :category];
        } catch(Exception e) {
            throw new AuraHandledException('Error retrieving items: ' + e.getMessage());
        }
    }
    @AuraEnabled(cacheable=true)
    public static Map<Id, String> fetchImageUrl() {
        Map<Id, String> imageUrlMap = new Map<Id, String>();

        try {
            List<Item__c> items = [SELECT Id, Image__c FROM Item__c];

            for (Item__c item : items) {
                if (String.isNotBlank(item.Image__c)) {
                    Matcher imgMatcher = Pattern.compile('<img[^>]+src\\s*=\\s*["\']([^"\']+)["\']').matcher(item.Image__c);

                    while (imgMatcher.find()) {
                        String img = imgMatcher.group(1);
                        imageUrlMap.put(item.Id, img.unescapeHtml4());
                    }
                }
            }
```

Apex Class :

```
public with sharing class eCartController {
    @AuraEnabled(cacheable=true)
    public static List<Item__c> getItemsByCategory(String category) {
        try{
            return [SELECT Id, Name, Price__c, Image__c,Quantity_Available__c FROM Item__c WHERE Item_Category__c = :category];
```

```
    } catch(Exception e) {
        throw new AuraHandledException('Error retrieving items: ' + e.getMessage());
    }
}
@AuraEnabled(cacheable=true)
public static Map<Id, String> fetchImageUrl() {
    Map<Id, String> imageUrlMap = new Map<Id, String>();

    try{
        List<Item__c> items = [SELECT Id, Image__c FROM Item__c];

        for (Item__c item : items) {
            if (String.isNotBlank(item.Image__c)) {
                                                                    Matcher imgMatcher =
Pattern.compile('<img[^>]+src\\s*=\\s*["\'][^"\']+["\']').matcher(item.Image__c);

                while (imgMatcher.find()) {
                    String img = imgMatcher.group(1);
                    imageUrlMap.put(item.Id, img.unescapeHtml4());
                }
            }
        }
    } catch (Exception e) {
        // Handle exceptions gracefully
        System.debug('An exception occurred: ' + e.getMessage());
    }

    return imageUrlMap;
}
@AuraEnabled
public static void createPurchaseRecord(String customerName, String email, String phone,
    String address, String city, String state,List<string> items) {
        system.debug('customerName' + customerName);
         system.debug('email' + email);
          system.debug('phone' + phone);
           system.debug('address' + address);
            system.debug('city' + city);
              system.debug('state' + state);
                system.debug('lststring' + items);
```

```
list<Purchase__c> plist = new List<Purchase__c>();
 for(string i: items){
 Purchase__c newPurchase = new Purchase__c();
 newPurchase.Name = customerName;
  newPurchase.Customer_Name__c = customerName;
   newPurchase.Customer_Email_Id__c = email;
   newPurchase.Customer_Contact__c          =
   phone;
   newPurchase.Address__c = address;
    newPurchase.City__c = city;
    newPurchase.Purchase_Date__c = date.today();
    newPurchase.State__c = state;
    newPurchase.Item__c = i;
    plist.add(newPurchase);
        }
     system.debug('plist ' + plist);
     if(!plist.Isempty()){
       insertplist;
     }
   }
}
```

Milestone 12 : Creating LWC component

Use Case upon selecting, the category the item displays a web view with the following details for each item associated with the purchase

Activity- 1

Open a Workspace:

Open Visual Studio Code and create a new workspace or open an existing one. A workspace is a directory that contains your Salesforce projects.

Create a New Salesforce Project:

Open the Command Palette (Ctrl + Shift + P) and run the command "SFDX: Create Project".
Choose the project type.
For LWC development, choose "Standard" for most cases.

Authorize an Org:
Open the Command Palette and run the command "SFDX: Authorize an Org".
Log in to your Salesforce org.

Create a New Lightning Web Component:
Open the Command Palette and run the command "SFDX: Create Lightning Web Component".
Enter a name for your component (Login) and choose the directory to save it.

Edit Your Lightning Web Component:
Open the newly created component in the force-app/main/default/lwc directory.
Edit the HTML, JavaScript, and CSS files as needed.

SurveyFormLWC (JavaScript) code:


```
//surveyFormLWC.js
import { LightningElement, track } from 'lwc';
import sendSurveyEmail from '@salesforce/apex/SurveyEmailController.sendSurveyEmail';

export default class SurveyFormLWC extends LightningElement {
   @track recipientEmail = '';
   @track surveyData = '';
recepientHandle(event){
      if(event.target.label == 'Recipient Email'){
         this.recipientEmail = event.target.value;
```

```
    }else if(event.target.label == 'Survey Data'){
        this.surveyData = event.target.value;
    }
  }
  sendSurvey() {
    // Validate input data if needed
        console.log('email ' + this.recipientEmail);
        console.log('survey ' + this.surveyData);
    // Call the Apex method to send the survey email
    sendSurveyEmail({ recipientEmail: this.recipientEmail, surveyData: this.surveyData })
        .then(result => {
            // Handle success
            console.log('Email sent successfully:', result);
        })
        .catch(error => {
            // Handle errors
            console.error('Error sending email:', error);
        });
  }
}
```


SurveyFormLWC(html) code:

surveyForm > surveyForm.html

```
<!-- surveyFormLWC.html -->
<template>
    <lightning-card title="Survey Form">
        <div class="slds-m-around_medium">
            <!-- Your survey form HTML goes here -->
            <!-- Include fields for survey questions and input elements -->
            <!-- Example: -->
            <label>Email:</label>
            <lightning-input type="email" value={recipientEmail} onchange={recepientHandle} label="Recipient Email"></lightning-input>

            <lightning-textarea name="input2" label="Survey Data" value="initial value" onchange={recepientHandle}></lightning-textarea>


            <lightning-button label="Send Survey" onclick={sendSurvey}></lightning-button>
        </div>
    </lightning-card>
</template>
</lightning-card>
</template>
```

```
<!-- surveyFormLWC.html -->

<template>

   <lightning-card title="Survey Form">

      <div class="slds-m-around_medium">

         <!-- Your survey form HTML goes here -->

         <!-- Include fields for survey questions and input elements -->

         <!-- Example: -->

         <label>Email:</label>

         <lightning-input type="email" value={recipientEmail} onchange={recepientHandle}
label="Recipient Email"></lightning-input>

                        <lightning-textarea name="input2" label="Survey Data" value="initial value"
onchange={recepientHandle}></lightning-textarea>

         <lightning-button label="Send Survey" onclick={sendSurvey}></lightning-button>

      </div>

   </lightning-card>

</template>

</lightning-card>

</template>
```

SurveyFormLWC(Xml) code:

surveyForm › surveyForm.js-meta.xml

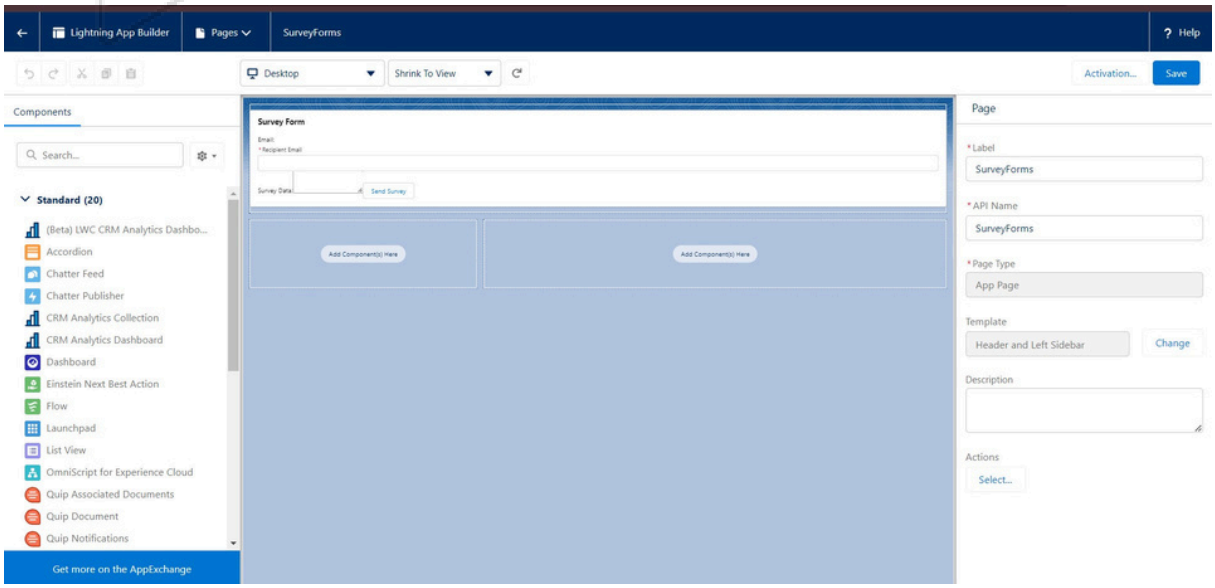
```
<?xml version="1.0"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>57.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__AppPage</target>
        <target>lightning__HomePage</target>

    </targets>
</LightningComponentBundle>
```

<?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__AppPage</target>
<target>lightning__RecordPage</target>
<target>lightning__HomePage</target>
</targets>
</LightningComponentBundle>

Deploy the source code.

Drag the custom component on the App page



Save and activate.

Activity- 2

eCartItemSelector (JavaScript) code:

```
eCartItemSelector > eCartItemSelector.js
import { LightningElement,api, track, wire } from 'lwc';
import { NavigationMixin } from 'lightning/navigation';
import getItemsByCategory from '@salesforce/apex/eCartController.getItemsByCategory';
import createPurchaseRecord from '@salesforce/apex/eCartController.createPurchaseRecord';
import fetchImageUrl from '@salesforce/apex/eCartController.fetchImageUrl';
export default class ECartItemSelector extends NavigationMixin(LightningElement) {
    @api recordId
    lwcIf = false;
    lwcElse = true;
    @track itemCategories = ['Grocery', 'Accessories', 'Electronics'];
    @track selectedCategory = '';
    @track items = [];
    @track ImageUrl = [];
    @api istrue = false;
    @track cartItems =[];
    @track imageMapped = [];
    @wire(getItemsByCategory, { category: '$selectedCategory' })
    wiredItems({ data, error }) {
        if (data) {
            this.items = data;
            console.log(data);
            console.log(this.selectedCategory);
            this.items = this.items.map(currentItem => {
                console.log(JSON.stringify(currentItem));
                const matchingItem = this.imageMapped.find(item => currentItem.Id === item.Id);
                if (matchingItem) {
                    return { ...currentItem, url: matchingItem.value }
                } else {
                    return { ...currentItem }
                }
            });
            console.log(data);
        } else if (error) {
            console.error(error);
            console.log('error on Electronic ' + JSON.stringify(error));
        }
```



```
import { LightningElement,api, track, wire } from 'lwc';
import { NavigationMixin } from 'lightning/navigation';
import getItemsByCategory from '@salesforce/apex/eCartController.getItemsByCategory';
import createPurchaseRecord from
'@salesforce/apex/eCar tController.createPurchaseRecord';
import fetchImageUrl from '@salesforce/apex/eCartController.fetchImageUrl';
exportdefaultclassECartItemSelectorextendsNavigationMixin(LightningElement){
    @api recordId
    lwcIf = false;
    lwcElse = true;
    @track itemCategories = ['Grocery', 'Accessories', 'Electronics'];
    @track selectedCategory = '';
    @track items = [];
    @track ImageUrl = [];
    @api istrue = false;
    @trackcartItems=[];
    @track imageMapped = [];
    @wire(getItemsByCategory, { category: '$selectedCategory' })
    wiredItems({ data, error }) {
```

```
    if (data) {
        this.items = data;
        console.log(data);
        console.log(this.selectedCategory);
        this.items = this.items.map(currentItem => {
            console.log(JSON.stringify(currentItem));
            const matchingItem = this.imageMapped.find(item => currentItem.Id === item.Id);
            if (matchingItem) {
                return { ...currentItem, url: matchingItem.value }
            } else {
                return { ...currentItem }
            }
        });
        console.log(data);
    } else if (error) {
        console.error(error);
        console.log('error on Electronic ' + JSON.stringify(error));
    }
}
HandleCar t(event){
    this.lwcElse = false;
    this.lwcIf = true;
}
@wire(fetchImageUrl)
fetchImageUrlwired({ data, error }) {
    if (data) {
        console.log('data fetchImage Url' + JSON.stringify(data))
        // console.log('parsedata'+JSON.parse(data));
        // this.ImageUrl = JSON.stringify(data);
        for (var key in data) {
            if (data.hasOwnProperty(key)) {
                console.log(key + " -> " + data[key]);
                let obj = { Id: key, value: data[key] }
                this.imageMapped.push(obj);
            }
        }
    } else if (error) {
        console.log('errror on fetching url from salesforce ' + error);
```

```
    }
}

handleChangecategory(evnt){
    this.selectedCategory = evnt.target.value;
}
handleItemClick(event) {
    const selectedItemId = event.target.key;
    const selectedItem = this.items.find(item => item.Id === selectedItemId);
    this.dispatchEvent(new CustomEvent('addtocart', { detail: selectedItem }));
}
get options() {
    return [
        { label: 'Accessories', value: 'Accessories' },
        { label: 'Electronics', value: 'Electronics' },
        {label:'Grocery',value:'Grocery'},
    ];
}
handleRemoveFromCart(event) {
    const itemId = event.detail.itemId;
    constitemIndex=this.cartItems.findIndex(item=>item.Id===itemId);
    if (itemIndex !== -1) {
        this.cartItems.splice(itemIndex,1);
        // Update any other relevant data
    }
}

increaseQuantity(event) {
    const itemId = event.target.dataset.itemId;
    const selectedItemIndex = this.items.findIndex(item => item.Id === itemId);
    if (selectedItemIndex !== -1) {
        this.items[selectedItemIndex].Quantity++;
    }
}
 handleCreatePurchase(event) {
     console.log('event.detail' + event.detail);
     let i = event.detail;
    // Call the Apex method
    console.log('handle Create Purchase is called');
```

```
    createPurchaseRecord({
      customerName: this.customerName,
      email: this.email,
      phone: this.phone,
      address: this.address,
      city: this.city,
      state: this.state,
      items: i // Example list of items
    })
    .then(result => {
      // Handle success
      console.log('Purchase record created successfully');
    })
    .catch(error => {
      // Handle error
      console.error('Error creating purchase record:', error);
    });
}

decreaseQuantity(event) {
    const itemId = event.target.dataset.itemId;
    const selectedItemIndex = this.items.findIndex(item => item.Id === itemId);
    if (selectedItemIndex !== -1 && this.items[selectedItemIndex].Quantity > 0) {
      this.items[selectedItemIndex].Quantity--;
    }
}

handleQuantityChange(event) {
    const { itemId, quantity } = event.detail;
    constselectedItem=this.cartItems.find(item=>item.Id===itemId);
    if (selectedItem) {
      selectedItem.Quantity = quantity;
      // Update any other relevant
    } data


}
handleAddToCart(event) {
    const itemId = event.target.dataset.id;
```

```
    const foundItem = this.items.find(item => item.Id === itemId);
    console.log('found Item' + foundItem);
    //console.log(this.template.querySelector('[data-id="id"]'))
    if(foundItem.Quan)
    console.log(itemId);
    const selectedItem = this.items.find(item => item.Id === itemId);
    if (selectedItem) {
        const newItem = { ...selectedItem, Quantity: 1 }; // Default quantity
        this.cartItems=[...this.cartItems,newItem];
        // Update any other relevant data
    }
    if(this.cartItems.length>0){
        this.istrue = true;
    }
    console.log('cartItems'+this.cartItems);
}
lististrue(event){
    console.log('List is True ');
     this.istrue = false
}
  customerName;
email;
phone;
address;
city;
state;

handleInputChange(event) {
    const fieldId = event.target.dataset.id;
    const value = event.target.value;

    // Assign the value to the corresponding property
    switch (fieldId) {
        case 'customerName':
            this.customerName = value;
            break;
        case 'email':
            this.email = value;
            break;
```

```
            case 'phone':
                this.phone = value;
                break;
            case 'address':
                this.address = value;
                break;
            case 'city':
                this.city = value;
                break;
            case 'state':
                this.state = value;
                break;
            default:
                break;
        }
    }


}
```

eCartItemSelector (HTML) code:

eCartItemSelector > eCartItemSelector.html

```
<template>

    <lightning-card title="Customer Information Form" lwc:if={lwcElse}>
        <!-- Customer Name -->
        <lightning-input
            data-id="customerName"
            type="text"
            label="Customer Name"
            onchange={handleInputChange}
        ></lightning-input>

        <!-- Email -->
        <lightning-input
            data-id="email"
            type="email"
            label="Email"
            onchange={handleInputChange}
        ></lightning-input>

        <!-- Phone -->
        <lightning-input
            data-id="phone"
            type="tel"
            label="Phone"
            onchange={handleInputChange}
        ></lightning-input>

        <!-- Address -->
        <lightning-input
            data-id="address"
            type="text"
            label="Address"
            onchange={handleInputChange}
        ></lightning-input>
```

```
<template>
    <lightning-card title="Customer Information Form" lwc:if={lwcElse}>
        <!-- Customer Name -->
        <lightning-input
            data-id="customerName"
            type="text"
            label="Customer Name"
            onchange={handleInputChange}
        ></lightning-input>

        <!-- Email -->
        <lightning-input
            data-id="email"
            type="email"
            label="Email"
            onchange={handleInputChange}
        ></lightning-input>

        <!-- Phone -->
        <lightning-input
```

```
    data-id="phone"
    type="tel"
    label="Phone"
    onchange={handleInputChange}
></lightning-input>
<!-- Address -->
<lightning-input
    data-id="address"
    type="text"
    label="Address"
    onchange={handleInputChange}
></lightning-input>
<!-- City -->
<lightning-input
    data-id="city"
    type="text"
    label="City"
    onchange={handleInputChange}
></lightning-input>
<!-- State -->
<lightning-input
```

```
        data-id="state"
        type="text"
        label="State"
        onchange={handleInputChange}
      ></lightning-input>
                <lightning-button variant="brand" label="Show Cart" title="titleName"
onclick={HandleCar t}></lightning-button>
 </lightning-card>
        <lightning-card lwc:if={lwcIf}>
        <lightning-layout>
        <lightning-layout-item padding="around-small" class="slds-size_9-of-12">
        <template if:true={items.length}>
              <table>
             <thead>
            <tr>
        <th>Name</th>
        <th>Price</th>
        <th>Quantity</th>
        <th>Image</th>
           </tr>
            </thead>
             <tbody>
```

```
        <template for:each={items} for:item="item">
                <tr key={item.Id}>
                <td>{item.Name}</td>
                <td>{item.Price__c}</td>

<td>{item.Quantity_Available__c}</td>
<td><img src={item.url} alt="Product image" width="100px" height="100px"></td>
<td>
<lightning-button variant="neutral" label="Add to Cart" title="Add to Cart"
onclick={handleAddToCart} data-id={item.Id}>
        </lightning-button>
        </td>
        </tr>
        </template>
       </tbody>
        </table>
        </template>
       </lightning-layout-item>
      <lightning-layout-item padding="around-small" class="slds-size_3-of-12">
     <label for="categorySelect">Select Category:</label>
        <lightning-combobox name="progress" placeholder="Select Progress" options={options}
        onchange={handleChangecategory}></lightning-combobox>
```

```
        </lightning-layout-item>

        </lightning-layout>

<div lwc:if={istrue}>

<c-e-cart-items-list cart-items={cartItems} istrue={istrue} recordid={recordId}
oncreatefire={handleCreatePurchase} onistrue={lististrue}
onremovefromcart={handleRemoveFromCart} onquantitychange={handleQuantityChange}
onaddtocar t={handleAddToCar t}>

                </c-e-car t-items-list>

        </div>

        </lightning-card>

</template>
```

eCartItemSelector (XML) code:

eCartItemSelector > eCartItemSelector.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>
            <!--<target>lightning__Tab</target>-->
            <!--<target>lightning__Inbox</target>-->
            <!--<target>lightning__UtilityBar</target>-->
            <!--<target>lightning__FlowScreen</target>-->
            <!--<target>lightningSnapin__ChatMessage</target>-->
            <!--<target>lightningSnapin__Minimized</target>-->
            <!--<target>lightningSnapin__PreChat</target>-->
            <!--<target>lightningSnapin__ChatHeader</target>-->
            <!--<target>lightningCommunity__Page</target>-->
            <!--<target>lightningCommunity__Default</target>-->
            <!--<target>lightningCommunity__Page_Layout</target>-->
            <!--<target>lightningCommunity__Theme_Layout</target>-->
            <!--<target>lightningStatic__Email</target>-->
            <!--<target>lightning_VoiceExtension</target>-->
            <!--<target>analytics__Dashboard</target>-->
            <target>lightning__RecordAction</target>
        </targets>

</LightningComponentBundle>
```

```
<?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>
        <!--<target>lightning__Tab</target>-->
        <!--<target>lightning__Inbox</target>-->
        <!--<target>lightning__UtilityBar</target>-->
        <!--<target>lightning__FlowScreen</target>-->
        <!--<target>lightningSnapin__ChatMessage</target>-->
        <!--<target>lightningSnapin__Minimized</target>-->
        <!--<target>lightningSnapin__PreChat</target>-->
        <!--<target>lightningSnapin__ChatHeader</target>-->
        <!--<target>lightningCommunity__Page</target>-->
        <!--<target>lightningCommunity__Default</target>-->
        <!--<target>lightningCommunity__Page_Layout</target>-->
        <!--<target>lightningCommunity__Theme_Layout</target>-->
        <!--<target>lightningStatic__Email</target>-->
        <!--<target>lightning_VoiceExtension</target>-->
```

```
        <!--<target>analytics__Dashboard</target>-->
        <target>lightning__RecordAction</target>
    </targets>

</LightningComponentBundle>
```

eCartItemSelector (CSS) code:

eCartItemSelector › eCartItemSelector.css

```
tr {
  border-bottom: 1px solid #ddd;
  object-fit: cover;
}
/* Example styling for buttons */
button {
    background-color: #4CAF50;
    border: none;
    color: white;
    padding: 10px 20px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    margin: 4px 2px;
    cursor: pointer;
    border-radius: 5px;
}

button:hover {
    background-color: #45a049;
}

button:active {
    background-color: #3e8e41;
}

/* Example styling for quantity input fields */
input[type=number] {
    width: 50px;
    padding: 5px;
    border: 1px solid #ccc;
    border-radius: 3px;
```

```
tr{
  border-bottom: 1px solid #ddd;
  object-fit: cover;
}
/* Example styling for buttons */
button {
    background-color: #4CAF50;
```

```
    border: none;
    color: white;
    padding: 10px 20px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    margin: 4px 2px;
    cursor: pointer;
    border-radius: 5px;
}

button:hover {
    background-color: #45a049;
}

button:active {
    background-color: #3e8e41;
}

/* Example styling for quantity input fields */
```

```
input[type=number] {
    width: 50px;
    padding: 5px;
    border: 1px solid #ccc;
    border-radius: 3px;
}

/* Additional styling for table rows */
tr{
    border-bottom: 1px solid #ddd;
    object-fit: cover;
}
.quantity-control {
    display: flex;
    align-items: center;
}

.quantity-button {
    padding: 5px 10px;
    border: 1px solid #ccc;
    background-color: #f0f0f0;
```

```
    cursor: pointer;
}

.quantity-input {
    width: 50px;
    padding: 5px;
    border: 1px solid #ccc;
    border-radius: 3px;
    text-align: center;
}
```

Activity- 3

eCartItemList (JavaScript) code:

eCartItemsList > eCartItemsList.js

```
import { LightningElement, api,track } from 'lwc';

export default class ECartItemsList extends LightningElement {
    @api cartItems = [];
    @api istrue;
    @api recordid ;
    itemids =[];
  @track totalAmount = 0;

    calculateTotalAmount() {
        this.totalAmount = 0;
        console.log("JSON.stringify(this.cartItems)" + JSON.stringify(this.cartItems))
       this.cartItems.forEach(item =>{
          this.itemids.push(item.Id);
           console.log(item.Price__c + item.Quantity)
          let i = item.Price__c * item.Quantity
           this.totalAmount = this.totalAmount + i;
        })
        console.log('total ' + this.totalAmount);
        console.log('this.items ' + this.itemids);
        this.dispatchEvent(new CustomEvent('createfire', { detail: this.itemids }));
    }
    handleRemoveFromCart(event) {
        const itemId = event.target.dataset.id;
        this.dispatchEvent(new CustomEvent('removefromcart', { detail: { itemId } }));
        if (cartItems.length < 1) {
            let i = false;
            this.dispatchEvent(new CustomEvent('onistrue', { detail: { i } }));
        }

    }
```

```
import { LightningElement, api,track } from 'lwc';
```

```
export default class ECartItemsList extends LightningElement {
    @api cartItems = [];
    @api istrue;
    @api recordid ;
    itemids =[];
  @track totalAmount = 0;

  calculateTotalAmount() {
      this.totalAmount = 0;
      console.log("JSON.stringify(this.cartItems)" + JSON.stringify(this.cartItems))
     this.cartItems.forEach(item =>{
        this.itemids.push(item.Id);
          console.log(item.Price__c + item.Quantity)
         let i = item.Price__c * item.Quantity
          this.totalAmount = this.totalAmount + i;
       })
      console.log('total ' + this.totalAmount);
      console.log('this.items ' + this.itemids);
      this.dispatchEvent(new CustomEvent('createfire', { detail: this.itemids }));
    }
    handleRemoveFromCart(event) {
```

```
    const itemId = event.target.dataset.id;
    this.dispatchEvent(new CustomEvent('removefromcart', { detail: { itemId } }));
    if (cartItems.length < 1) {
        let i = false;
        this.dispatchEvent(new CustomEvent('onistrue', { detail: { i } }));
    }
}

handleQuantityChange(event) {
    const itemId = event.target.dataset.id;
    const quantity = parseInt(event.target.value, 10);
    this.dispatchEvent(new CustomEvent('quantitychange', { detail: { itemId, quantity } }));
}


handleAddToCart(event) {
    const itemId = event.target.dataset.id;
    this.dispatchEvent(new CustomEvent('addtocart', { detail: { itemId } }));
}
handleAddToCart(event) {
    const newItem = event.detail;
```

```
        this.cartItems = [...this.cartItems, newItem];
    }
    handleBuyNow(event) {
        const itemId = event.target.dataset.id;
        this.dispatchEvent(new CustomEvent('buynow', { detail: this.totalAmount }));
    }
}
```

eCartItemList (HTML) code:


```
<template>
        <lightning-card>
                <div class="cart-items-container">
                        <p>Cart Items:</p>
                        <table class="cart-items-table">
```

<thead>

<tr>

<th>Item Name</th>

<th>Quantity</th>

<th>price</th>

<th>Action</th>

</tr>

</thead>

<tbody>

<template for:each={cartItems} for:item="item">

<tr key={item.Id}>

<td>{item.Name}</td>

<td>{item.Quantity}</td>

<td>

${item.Price__c}</td>

<td>

<lightning-button label="Remove" onclick={handleRemoveFromCart} data-id={item.Id}>

</lightning-button>

</td>

</tr>

</template>

```
                </tbody>

                </table>

                </div>

                <div class="buy-now-section">

                Total Price: ${totalAmount}

                <!-- Display total price -->&amp;nbsp;&amp;nbsp;&amp;nbsp;

                <lightning-button label="Buy Now" onclick={calculateTotalAmount}
></lightning-button>

            </div>

            </lightning-card>

</template>

</lightning-card>

</template>
```

eCartItemList (XML) code:

eCartItemsList > eCartItemsList.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__AppPage</target>
        <target>lightning__RecordPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>
```

<?xml version="1.0" encoding="UTF-8"?>

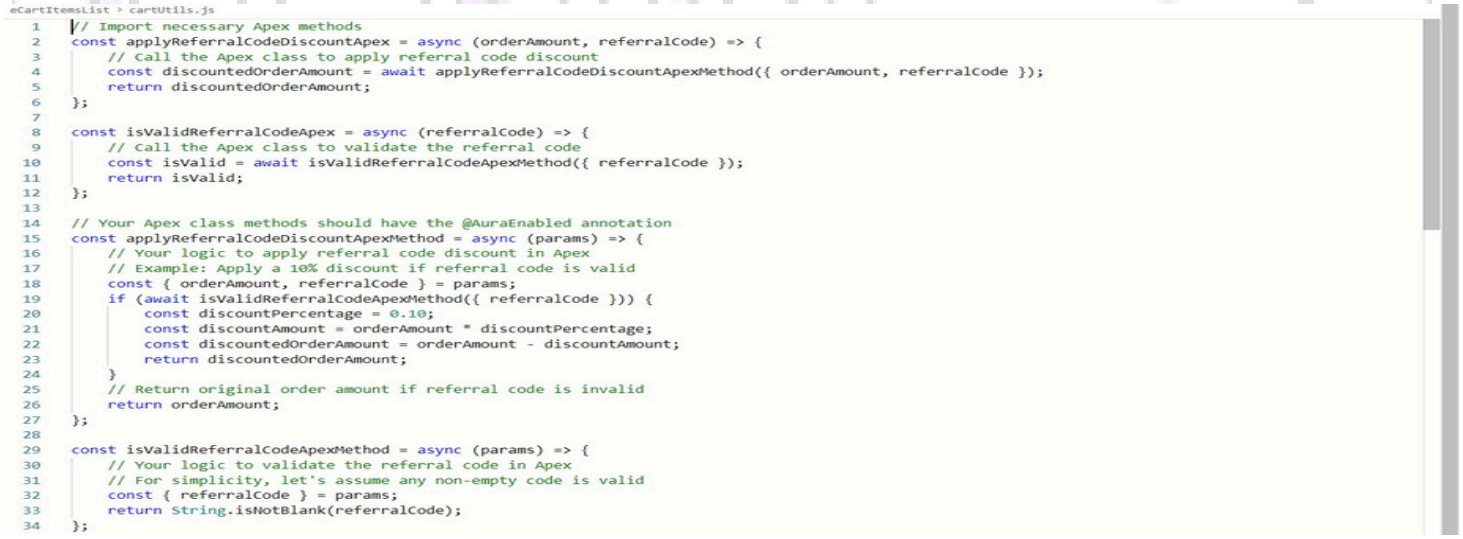
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">

<apiVersion>58.0</apiVersion>

<isExposed>true</isExposed>

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

CartUtils (Js) code:

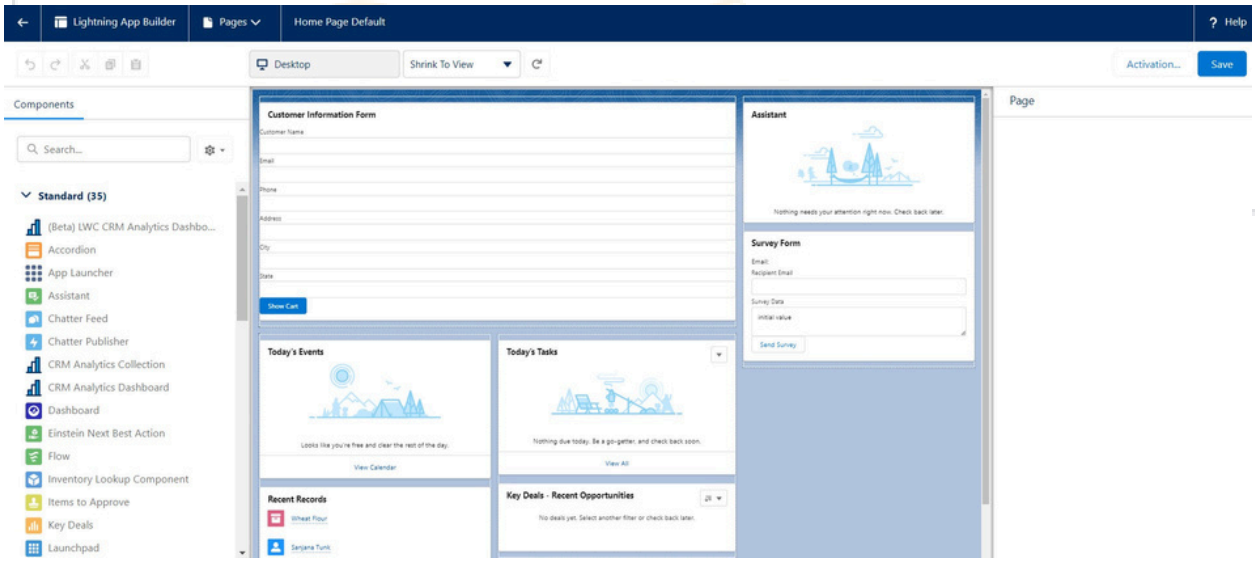


```
// Import necessary Apex methods
const applyReferralCodeDiscountApex = async (orderAmount, referralCode) => {
    // Call the Apex class to apply referral code discount
    const discountedOrderAmount = await applyReferralCodeDiscountApexMethod({ orderAmount, referralCode });
    return discountedOrderAmount;
};

const isValidReferralCodeApex = async (referralCode) => {
    // Call the Apex class to validate the referral code
    const isValid = await isValidReferralCodeApexMethod({ referralCode });
    return isValid;
};

// Your Apex class methods should have the @AuraEnabled annotation
const applyReferralCodeDiscountApexMethod = async (params) => {
    // Your logic to apply referral code discount in Apex
    // Example: Apply a 10% discount if referral code is valid
    const { orderAmount, referralCode } = params;
    if (await isValidReferralCodeApexMethod({ referralCode })) {
        const discountPercentage = 0.10;
        const discountAmount = orderAmount * discountPercentage;
        const discountedOrderAmount = orderAmount - discountAmount;
        return discountedOrderAmount;
    }
    // Return original order amount if referral code is invalid
    return orderAmount;
};

const isValidReferralCodeApexMethod = async (params) => {
    // Your logic to validate the referral code in Apex
    // For simplicity, let's assume any non-empty code is valid
    const { referralCode } = params;
    return String.isNotBlank(referralCode);
};
```

```
const isValidReferralCodeApex = async (referralCode) => {
    // Call the Apex class to validate the referral code
    const isValid = await isValidReferralCodeApexMethod({ referralCode });
    return isValid;
};

// Your Apex class methods should have the @AuraEnabled annotation
const applyReferralCodeDiscountApexMethod = async (params) => {
    // Your logic to apply referral code discount in Apex
    // Example: Apply a 10% discount if referral code is valid
    const { orderAmount, referralCode } = params;
    if (await isValidReferralCodeApexMethod({ referralCode })) {
        const discountPercentage = 0.10;
        const discountAmount = orderAmount * discountPercentage;
        const discountedOrderAmount = orderAmount - discountAmount;
        return discountedOrderAmount;
    }
    // Return original order amount if referral code is invalid
    return orderAmount;
};
```

```
const isValidReferralCodeApexMethod = async (params) => {
    // Your logic to validate the referral code in Apex
    // For simplicity, let's assume any non-empty code is valid
    const { referralCode } = params;
    return String.isNotBlank(referralCode);
};

const addToCart = (item, cartItems, totalAmount) => {
    // Implement logic to add an item to the cart
    // Update the cartItems array and totalAmount accordingly
    //...
};
const removeFromCart = (item, cartItems, totalAmount) => {
    // Implement logic to remove an item from the cart
    // Update the cartItems array and totalAmount accordingly
    //...
};
const updateQuantity = (item, quantity, cartItems, totalAmount) => {
    // Implement logic to update the quantity of an item in the cart
    // Update the cartItems array and totalAmount accordingly
```

```
    //...
};
export default {
    applyReferralCodeDiscount: applyReferralCodeDiscountApex,
    isValidReferralCode: isValidReferralCodeApex,
    addToCart,
    removeFromCart,
    updateQuantity,
};
```



Deploy the source code.

Drag the custom component on the App page