

## **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 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 :

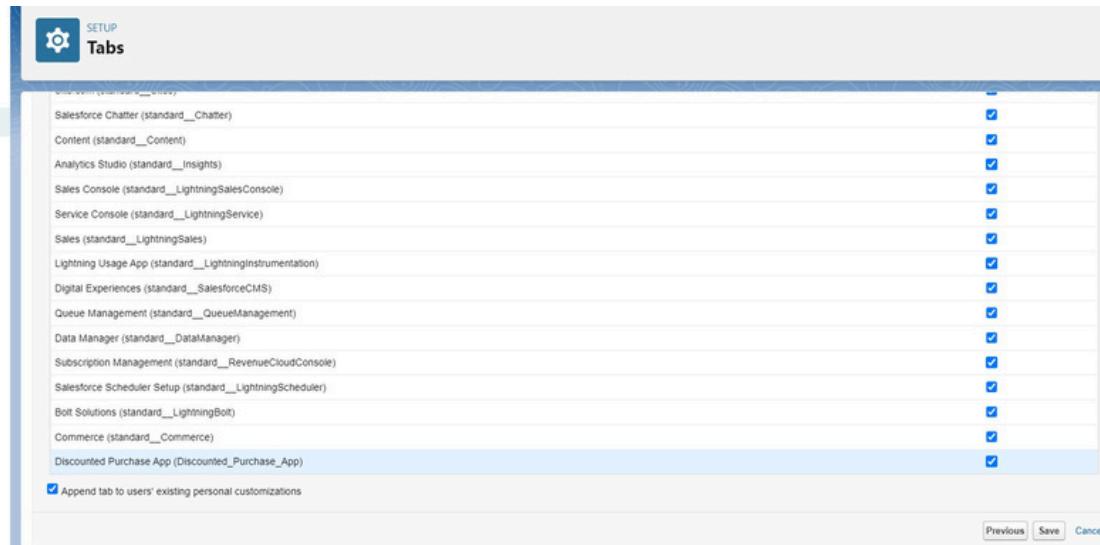
1. To Create the tab → click on the gear Icon → then click Home button → In the quick find box search for the Tabs.

The screenshot shows the Salesforce Setup interface under the 'Tabs' section. It includes two main sections: 'Custom Object Tabs' and 'Web Tabs'. The 'Custom Object Tabs' section lists several tabs with their labels and styles: Feedback (Pencil), Items (Box), Purchase (Bell), and ReferralCodes (Mail). The 'Web Tabs' section indicates 'No Web Tabs have been defined'.

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

The screenshot shows the 'Tab Style Selector' dialog box from Google Chrome. It displays a grid of icons representing various tab styles, such as Airplane, Alarm clock, Apple, Balls, Bank, Bell, Big top, Boat, Books, Bottle, Box[1], Bridge, Building, Building Block, Caduceus, Camera, Can, Car, Castle, CD/DVD, Cell phone, Chalkboard[1], Chess piece, Chip, Circle, Compass[1], Computer, Credit card, CRT TV, Cup, Desk, Diamond, Dice, Factory, Fan, Flag[1], Form, Gears, Globe, Guitar, Hammer, Hands, Handsaw, Headset, Heart, Helicopter, Hexagon, Highway Sign, Hot Air Balloon, Insect, IP Phone, Jewel, Keys, Laptop, Leaf, and Lightning.

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. For creating a tab for Feedback Tab follow the above steps which are being created in step1 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)

The screenshot shows the Salesforce Object Manager interface for the 'Item' object. The left sidebar lists various setup categories like Details, Page Layouts, Lightning Record Pages, etc. The 'Fields & Relationships' tab is selected. In the main pane, there's a list of field types with their descriptions. A callout points to the 'Text Area (Long)' option, which is described as allowing up to 131,072 characters on separate lines.

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

The screenshot shows the 'New Custom Field' setup page, Step 2. The 'Field Label' is set to 'Description'. Other settings include 'Length' (32,768), '# Visible Lines' (3), and 'Field Name' (Description). The 'Description' and 'Help Text' fields are empty. At the bottom, there are checkboxes for 'Auto add to custom report type' and 'Add this field to existing custom report types that contain this entity'.

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)

The screenshot shows the Salesforce Object Manager interface for the 'Purchase' object. On the left, there's a sidebar with various setup options like Details, Fields & Relationships, Page Layouts, etc. The 'Fields & Relationships' tab is active. On the right, a list of field types is shown with their descriptions. The 'Text' field type is selected and highlighted with a blue border. At the bottom right of the list, there are 'Next' and 'Cancel' buttons.

Field Type	Description
Date	Allows users to enter a date or pick a date from a popup calendar.
Date/Time	Allows users to enter a date and time, or pick a date from a popup calendar. When users click a date in the pop-up, that date and the current time are entered into the Date/Time field.
Email	Allows users to enter an email address, which is validated to ensure proper format. If this field is specified for a contact or lead, users can choose the address when clicking Send an Email. Note that custom email addresses cannot be used for mass emails.
Geolocation	Allows users to define location. Includes latitude and longitude components, and can be used to calculate distance.
Number	Allows users to enter any number. Leading zeros are removed.
Percent	Allows users to enter a percentage number, for example, '10' and automatically adds the percent sign to the number.
Phone	Allows users to enter any phone number. Automatically formats it as a phone number.
Picklist	Allows users to select a value from a list you define.
Picklist (Multi-Select)	Allows users to select multiple values from a list you define.
<b>Text</b>	<b>Allows users to enter any combination of letters and numbers.</b>
Text Area	Allows users to enter up to 255 characters on separate lines.
Text Area (Long)	Allows users to enter up to 131,072 characters on separate lines.
Text Area (Rich)	Allows users to enter formatted text, add images and links. Up to 131,072 characters on separate lines.
Text (Encrypted)	Allows users to enter any combination of letters and numbers and store them in encrypted form.
Time	Allows users to enter a local time. For example, "2:40 PM", "14:40", "14:40:00", and "14:40:59:600" are all valid times for this field.
URL	Allows users to enter any valid website address. When users click on the field, the URL will open in a separate browser window.

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)-Always require a value in this field in order to save a record.
- CustomerEmailId>Email)-Always require a value in this field in order to save a record.
- Address(LongTextArea)- Always require a value in this field in order to save a record.
- City(Text)- Always require a value in this field in order to save a record.
- State(Text)- Always require a value in this field in order to save a record.
- PurchaseDate(Date)-Always require a value in this field in order to save a record.

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 :

- Design a Lightning App using the Lightning App Builder.
- Include tabs for the Item object, standard objects like Purchase etc, and custom Lightning components.
- Customize the app's colorscheme and logo to give it a branded look.

### Activity -1

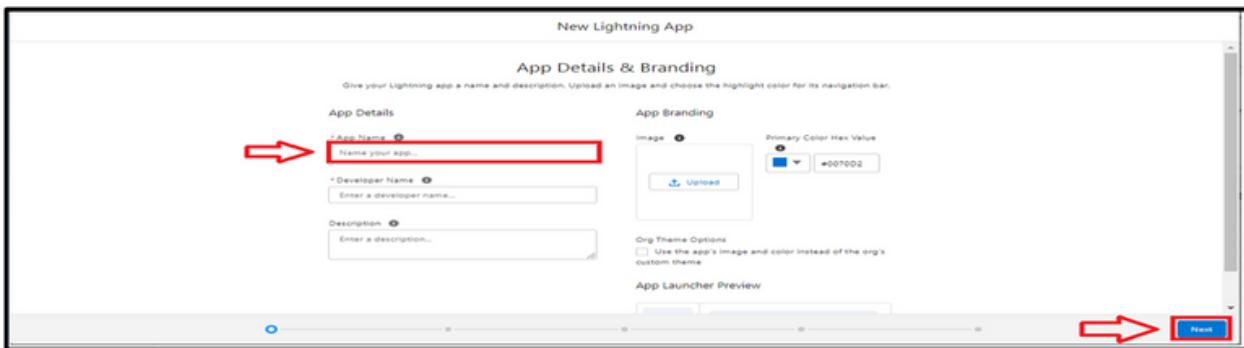
Create the Lightning App

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

The screenshot shows the Salesforce App Manager interface. At the top, there are tabs for 'Setup', 'Home', and 'Object Manager'. A red box highlights the 'Home' tab. Below the tabs, there is a search bar and a 'Quickly create new Lightning apps by cloning existing apps...' section. A red box highlights the 'New Lightning App' button. The main area displays a table of existing apps, with columns for 'App Name', 'Developer Name', 'Description', 'Last Modified...', 'App Type', and 'V...'. A red box highlights the 'Clone App(Beta)' link. The table lists 35 items, including 'All Tabs', 'Analytics Studio', 'App Launcher', 'Bolt Solutions', 'Chatter Desktop', 'Chatter Mobile for BlackBerry', 'College Management System', 'Community', 'Content', and 'Data Manager'. The 'Last Modified...' column shows dates from April 12, 2022, to April 13, 2022. The 'App Type' column shows various types: Classic, Lightning, Connected (Managed), and Connected (Varaged).

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.

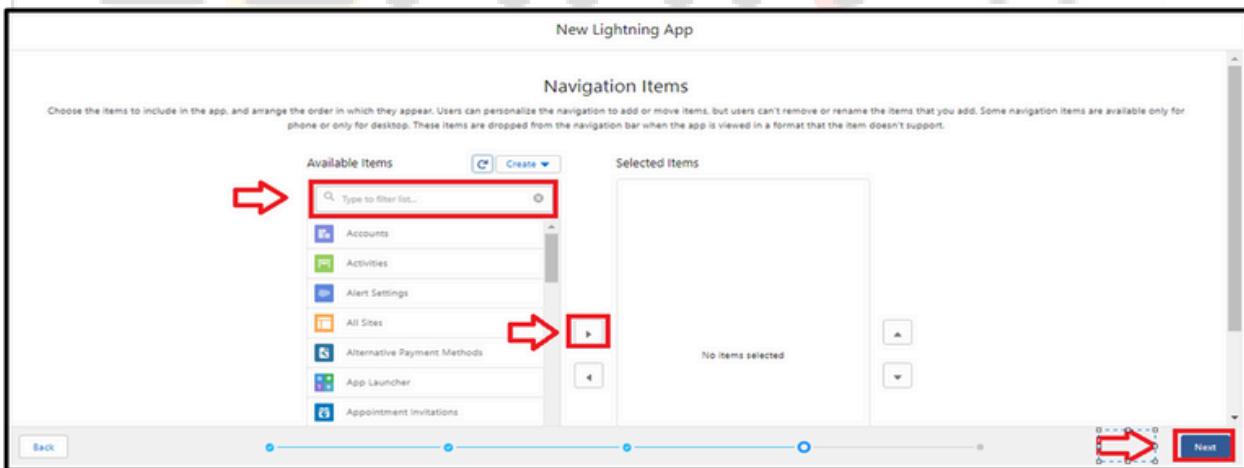


#### 4. To Add Navigation Items:

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

SETUP > OBJECT MANAGER  
Item

Validation Rules

RULE NAME	ERROR LOCATION	ERROR MESSAGE	ACTIVE	MODIFIED BY
ItemPrice	Top of Page	Product price must be greater or equal to hundred.	✓	Sanjana Tunk, 15/02/2024, 10:59 am

Lightning Record Pages  
Buttons, Links, and Actions  
Compact Layouts  
Field Sets  
Object Limits  
Record Types  
Related Lookup Filters  
Search Layouts  
List View Button Layout  
Restriction Rules  
Scoping Rules  
Triggers  
Flow Triggers  
Validation Rules

3. Enter the Rule name as "ItemPrice".

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.

SETUP > OBJECT MANAGER  
Item

Validation Rule Edit

Rule Name: ItemPrice

Active:

Description:

Error Condition Formula

Example: Discount\_Percent\_\_c>0.30 [More Examples...](#)

If this formula expression is true, display the text defined in the Error Message area

Insert Field  Price\_\_c >= 100

Functions

ABS  
ACOS  
ADDMONTHS  
AND  
ASCII  
ASIN

Insert Selected Function  
ABS(number)  
Returns the absolute value of a number, a number without its sign

Check Syntax

Details  
Fields & Relationships  
Page Layouts  
Lightning Record Pages  
Buttons, Links, and Actions  
Compact Layouts  
Field Sets  
Object Limits  
Record Types  
Related Lookup Filters  
Search Layouts  
List View Button Layout  
Restriction Rules

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

Action	Profile Name	User License	Custom
<a href="#">Edit   Del</a> ...	Sales Manager	Salesforce	✓
<a href="#">Edit   Clone</a>	Salesforce API Only System Integrations	Salesforce Integration	□
<a href="#">Edit   Del</a> ...	School profile	Salesforce	✓
<a href="#">Edit   Clone</a>	Silver Partner User	Silver Partner	□
<a href="#">Edit   Clone</a>	Solution Manager	Salesforce	□
<a href="#">Edit   Clone</a>	Standard Platform User	Salesforce Platform	□
<a href="#">Edit   Clone</a>	Standard User	Salesforce	□
<a href="#">Edit   Clone</a>	System Administrator	Salesforce	□

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.

The screenshot shows the Salesforce Setup interface under the 'Profiles' tab. It displays two sections: 'Custom Object Permissions' and 'Session Settings'.

**Custom Object Permissions:**

	Basic Access				Data Administration	
	Read	Create	Edit	Delete	View All	Modify All
Customer Accounts	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Feedback	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Items	<input checked="" type="checkbox"/>					
Purchase	<input checked="" type="checkbox"/>					
ReferralCodes	<input checked="" type="checkbox"/>					

**Session Settings:**

- Session Times Out After: 2 hours of inactivity
- Session Security Level Required at Login: --None--

## 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.

## Customer Account

User Edit

Save Save & New Cancel

General Information

First Name	Customer	Role	Marketing Team
Last Name	Account	User License	Salesforce
Alias	cacco	Profile	System Administrator
Email	sanjanatunk@gmail.com	Active	<input checked="" type="checkbox"/>
Username	sanjublue@123.com	Marketing User	<input type="checkbox"/>
Nickname	CAccount	Offline User	<input type="checkbox"/>
Title		Knowledge User	<input type="checkbox"/>
Company		Flow User	<input type="checkbox"/>
Department		Service Cloud User	<input type="checkbox"/>
Division		Site.com Contributor User	<input type="checkbox"/>
		Site.com Publisher User	<input type="checkbox"/>
		WDC User	<input type="checkbox"/>
		Data.com User Type	-None--
		Data.com Monthly Addition Limit	300
		Accessibility Mode (Classic Only)	<input type="checkbox"/>
		High-Contrast Palette on Charts	<input type="checkbox"/>

## 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.

SETUP > OBJECT MANAGER  
Purchase

Buttons, Links, and Actions  
9 items, Sorted by Label

LABEL	NAME	DESCRIPTION	TYPE	CONTENT SOURCE	OVERRIDDEN
Accept	Accept			Standard page	<input type="button" value="▼"/>
Clone	Clone			Standard page	<input type="button" value="▼"/>
Delete	Delete			Standard page	<input type="button" value="▼"/>
Edit	Edit			Standard page	<input type="button" value="▼"/>
List	List			Standard page	<input type="button" value="▼"/>
New	New			Standard page	<input type="button" value="▼"/>
Purchase Tab	Tab			Standard page	<input type="button" value="▼"/>
Select Item	Select_Item	Lightning Web Component	Lightning Web Component		<input type="button" value="▼"/>
View	View			Standard page	<input type="button" value="▼"/>

### 3. Fill in the details as

Edit Purchase Action  
Select Item

Help for this Page ?

**Enter Action Information**

Object Name	Purchase <a href="#">?</a>
Action Type	Lightning Web Component
Lightning Web Component	c:eCartItemSelector <a href="#">?</a>
Subtype	Screen Action
Standard Label Type	-None-- <a href="#">?</a>
Label	Select Item <a href="#">?</a>
Name	Select_Item <a href="#">?</a>
Description	<input type="text"/> <a href="#">?</a>
Icon	Change Icon

Save Cancel

### 4. Click on Page Layout → click Edit.

SETUP > OBJECT MANAGER  
**Purchase**

Details  
Fields & Relationships  
**Page Layouts**  
Lightning Record Pages  
Buttons, Links, and Actions  
Compact Layouts  
Global Cache

**Page Layouts**  
1 Items, Sorted by Page Layout Name

PAGE LAYOUT NAME	CREATED BY	MODIFIED BY
Order Layout	Sanjana Tunk, 13/02/2024, 3:19 pm	Sanjana Tunk, 21/03/2024, 12:50 pm

[Edit](#) [Delete](#)

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.

SETUP > OBJECT MANAGER  
**Purchase**

Details  
Fields & Relationships  
**Page Layouts**  
Lightning Record Pages  
Buttons, Links, and Actions  
Compact Layouts  
Field Sets  
Object Limits

**Order Layout**

Save Quick Save Preview As... Cancel Undo Redo Layout Properties

Custom Console Components Mini Page Layout Mini Console View Video Tutorial Help for this Page ?

Mobile & Lightning Actions

Fields	Change Owner	Edit	Log a Call	New Contact	New Note	Post	Send Survey
Buttons	Change Record Type	Email	Mobile Smart Actions	New Event	New Opportunity	Printable View	Submit for Approval
Quick Actions	Clone	File	New Account	New Group	New Task	Question	
Mobile & Lightning Actions	Delete	Link	New Case	New Lead	Poll	Select Item	

Purchase Sample  
Highlights Panel

Actions in this section are currently inherited from the global publisher layout. You can override the global publisher layout to set a customized list of actions for this publisher on pages that use this layout.

**Salesforce Mobile and Lightning Experience Actions**

Select Item Post File New Event New Task New Contact Log a Call Email New Case New Lead Link Poll  
Question New Opportunity Change Owner Submit for Approval Clone Delete Send Survey Printable View Change Record Type Edit

**Purchase Detail**

Standard Buttons: Edit, Delete, Clone, Change Owner, Change Record Type, Printable View | Custom Buttons

**Information (Header visible on edit only)**

- \* Purchase Name Sample Text
- \* Purchase Date 22/03/2024
- \* Customer Name Sample Text
- \* Customer Email Id sarah.sample@company.com
- \* Customer Contact 1-415-555-1212

## 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

A screenshot of a Salesforce form for creating a new item record. The form fields include:

- \* Item Name: Wheat Flour
- \* Price: ₹50
- Description: Wheat flour 9Kg Pack
- \* Weight: 9,000.000
- \* Quantity Available: 10
- Item Category: Grocery
- Total Sales: ₹0.0000 (Note: This field is calculated upon save)
- Total Purchase Count: 1

The bottom of the form shows a modal dialog with Average Rating (0), Cancel, Save & New, and Save buttons.

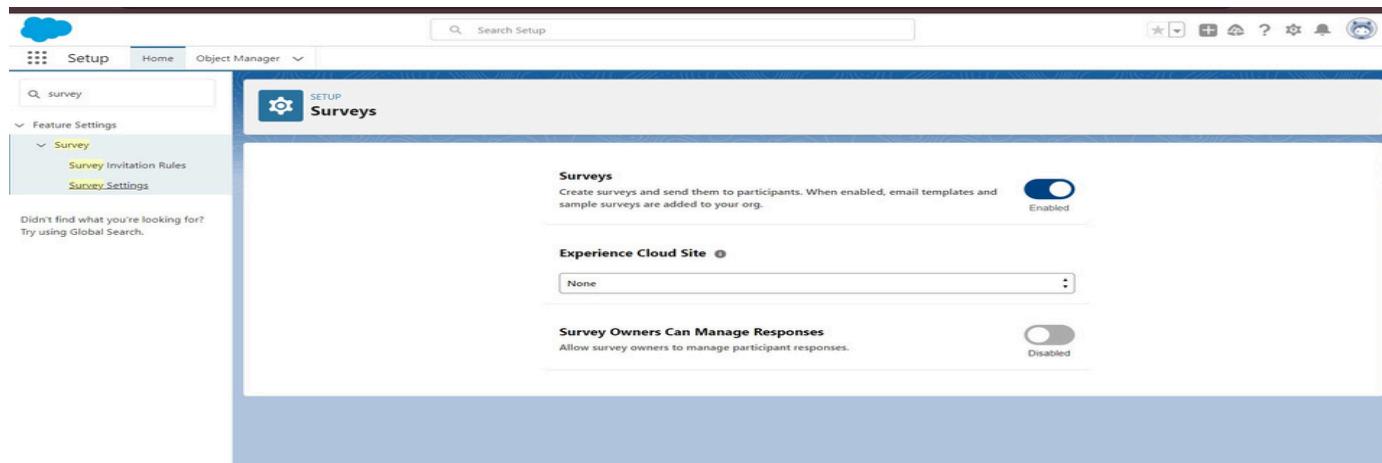
## Milestone 10 : SurveyFormLink

### 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.

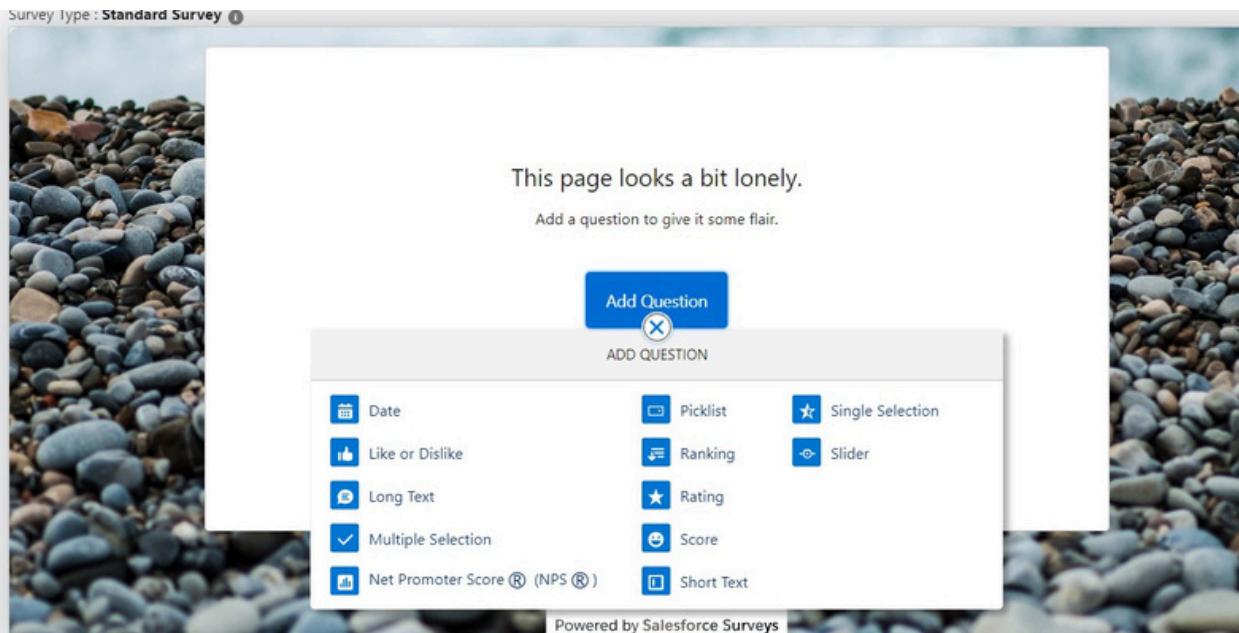
The screenshot shows a software interface for managing surveys. At the top, there's a navigation bar with links like 'Discounted Purchas...', 'Items', 'ReferralCodes', 'Purchase', 'Feedback', and 'Recently Viewed | Surveys'. Below this is a search bar and a toolbar with icons for star, plus, question mark, settings, and notifications. A 'Surveys' section titled 'Recently Viewed' shows three items: 'Product Feedback', 'Customer Satisfaction', and 'Net Promoter Score', each with a last modified date and by whom. A modal window titled 'New survey' is open in the center, prompting for a 'Survey Name' which is currently 'Feedback Form'. There's also a checked checkbox for 'Create survey as a template'. Buttons for 'Cancel' and 'Continue' are at the bottom of the modal.

6. Fill the Survey Name as "Feedback Form".

7. Click on Create survey as a Template, then continue.

The screenshot shows a survey editor interface for a 'Product Feedback' survey. The top navigation bar includes 'View', 'Send', 'Analyze', 'Template', 'Save', and other controls. On the left, there's a sidebar with 'Pages' (selected) and 'Branding'. Under 'Pages', there are three cards: 'Welcome Page' (selected), 'Page 1', and 'Thank You Page'. The main area is titled 'Survey Type : Standard Survey'. It features a cartoon-style illustration of a hand pointing at a smiley face, a computer screen, and a person. Text in the center says 'Thank you for shopping with us! We would appreciate it if you provide us with a feedback.' A 'Next' button is at the bottom. On the right, there's a 'Preview' and 'Archive' button. A 'Branding' tab is visible on the far right.

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



9. Choose multiple selection then enter your desired question

The screenshot shows a survey editor interface. At the top, there are buttons for "Display Logic", "Insert Content", and various layout controls. Below this is a text input field containing the question "Were you satisfied with the product ?". Underneath the question is a rich text editor toolbar with font size dropdowns (12), bold (B), italic (I), underline (U), and other styling options. Below the toolbar is a text input field containing "yes". To the right of this field is a small "907" count and a "More" button. Below the "yes" field are two more fields: "no" and "satisfied", each with a delete "X" icon to its right. At the bottom left, there is a button labeled "+ Add Choice".

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

The screenshot shows a survey builder interface with a floating window for creating a new question. The window title is "What rating would you give to the product". It includes a rich text editor toolbar with font selection (Salesforce Sans), size (12), bold (B), italic (I), underline (U), and other styling options. Below the toolbar is a text input field with the placeholder "Describe your question...". Underneath the text input is a "Rating Icon" section with a dropdown set to "Star" and five star icons below it. At the bottom of this section are two buttons: "Bad" and "Good". The background of the interface features a pebble beach image on the left and a large stylized "th" logo on the right.

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

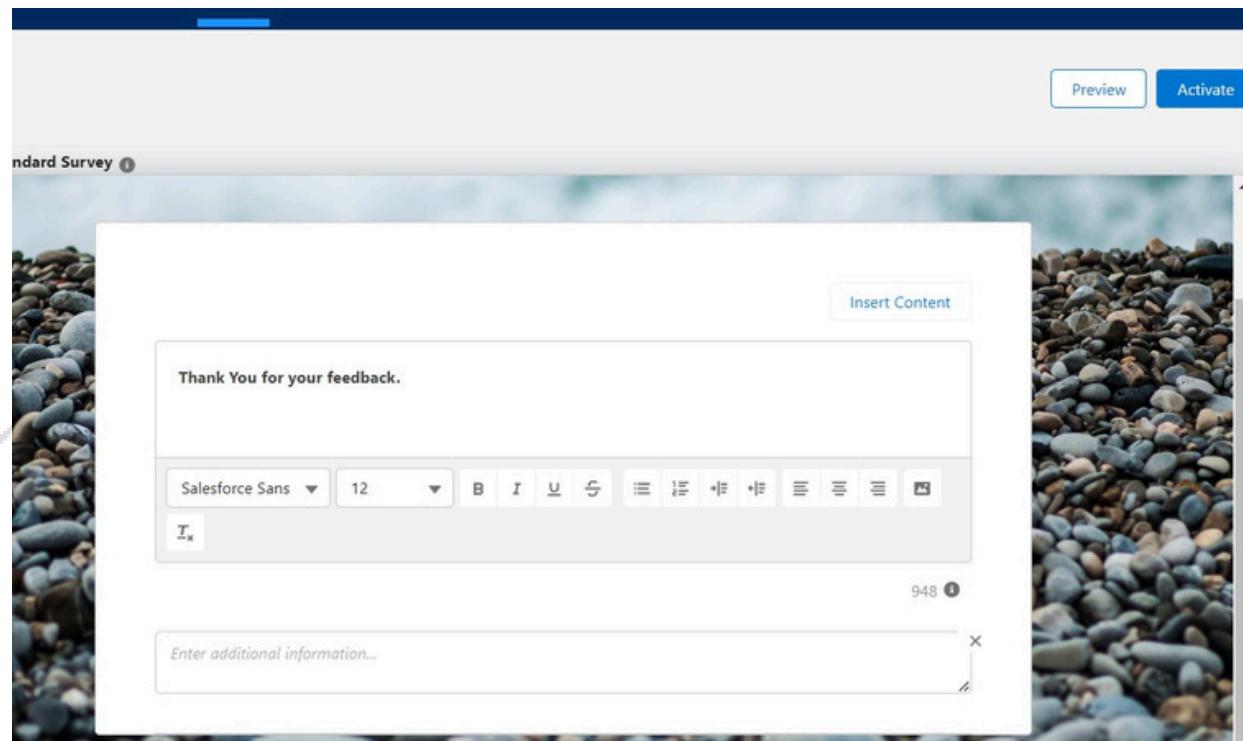
Please share your shopping experience with us, What could we do to make it better?

Enter answer...

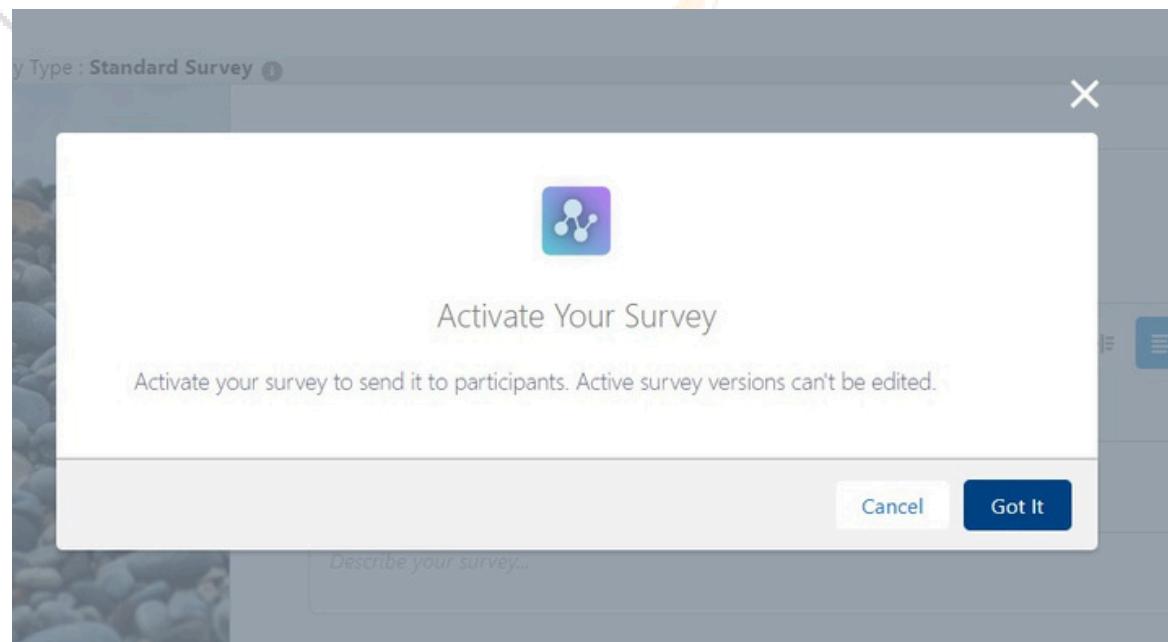
Previous

Finish

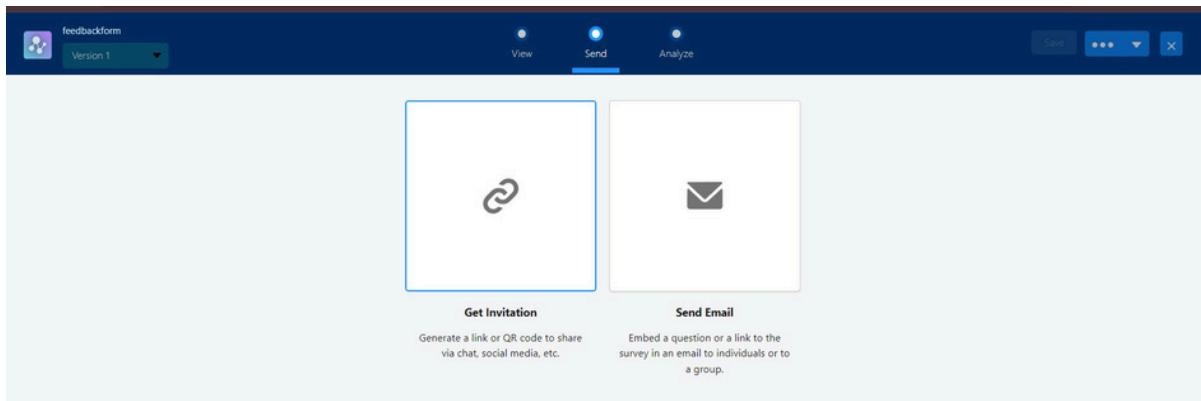
12. Click on Add Page, Then add a thankyou message.



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.



This comparison highlights the differences in the 'Send Email' process between SurveyMonkey and Salesforce Lightning Survey.

**Salesforce Lightning Survey (Left):**

- Header:** 'Send Email'
- Description:** '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.'
- Options:**
  - 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.
- Progress Bar:** A horizontal progress bar with 'Previous' and 'Next' buttons.

**SurveyMonkey (Right):**

- Header:** 'Get Survey Link'
- Participants Options:** 'Participants in Your Company' (selected) and 'Participants Outside Your Company'.
- Link Preview:** A preview of the survey link: <https://socom-be-dev-ed.develop.my.salesforce.com/survey/runtimeApp.app?invitationId=0K15000000fCo&surveyName=feedbackform&UUID=292d6778-d3c8-4b1d-a6f3-fbb55c017357>
- Settings:**
  - Auto-Expires
  - Anonymize responses
  - Let participants see their responses
- Buttons:** 'Download QR Code' and 'Copy Link'.

The screenshot shows the SurveyMonkey Response Dashboard for the 'feedbackform' survey. The 'View' tab is active.

**Survey Metrics:**

- Complete or In Progress Responses:** 0
- Message:** 'We can't draw this chart because there is no data.'

**Response Dashboard:**

- Question 1:** 'Were you satisfied with the product ?'
- Question 2:** 'What rating would you give to the product'
- Feedback:** 'Sis tight! No one has responded to this question yet.'

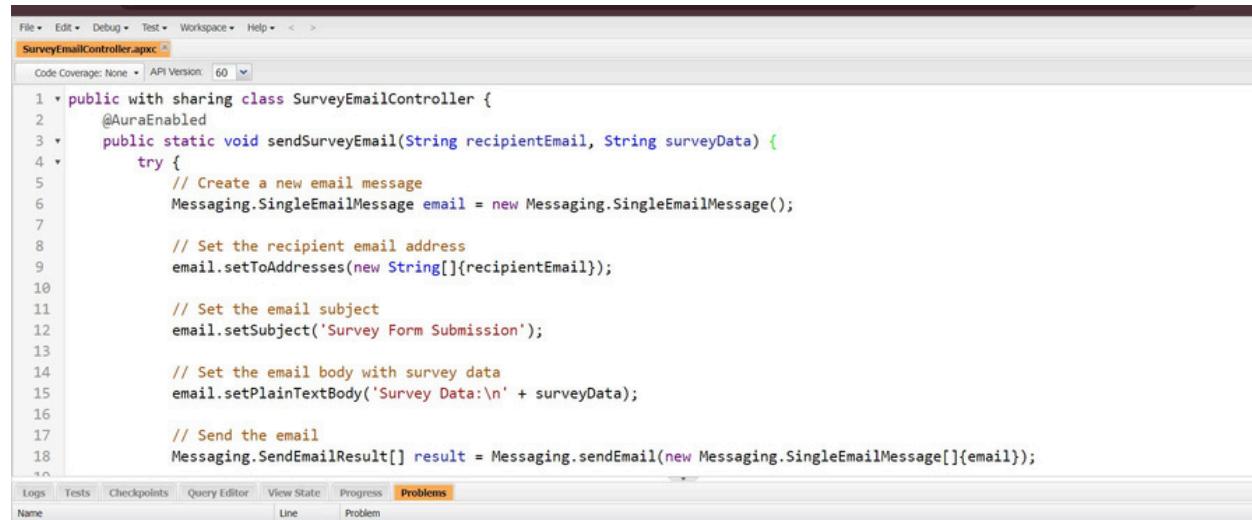
**Analyze Tab:**

- Select Survey Page:** Page 1 / 2
- Message:** 'Showing 2 of 2 questions'

## 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.



The screenshot shows the Salesforce IDE interface with the file 'SurveyEmailController.apxc' open. The code editor displays the following Apex class:

```
1 public with sharing class SurveyEmailController {
2     @AuraEnabled
3     public static void sendSurveyEmail(String recipientEmail, String surveyData) {
4         try {
5             // Create a new email message
6             Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();
7
8             // Set the recipient email address
9             email.setToAddresses(new String[]{recipientEmail});
10
11            // Set the email subject
12            email.setSubject('Survey Form Submission');
13
14            // Set the email body with survey data
15            email.setPlainTextBody('Survey Data:\n' + surveyData);
16
17            // Send the email
18            Messaging.SendEmailResult[] result = Messaging.sendEmail(new Messaging.SingleEmailMessage[]{email});
19        }
20    }
21}
```

#### 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.

```
File Edit Debug Test Workspace Help < >
eCartController.apxc SurveyEmailController.apxc
Code Coverage: None API Version: 60
1 public with sharing class eCartController {
2     @AuraEnabled(cacheable=true)
3     public static List<Item__c> getItemsByCategory(String category) {
4         try {
5             return [SELECT Id, Name, Price__c, Image__c,Quantity_Available__c FROM Item__c WHERE Item_Category__c = :category];
6         } catch(Exception e) {
7             throw new AuraHandledException('Error retrieving items: ' + e.getMessage());
8         }
9     }
10    @AuraEnabled(cacheable=true)
11    public static Map<Id, String> fetchImageUrl() {
12        Map<Id, String> imageUrlMap = new Map<Id, String>();
13
14        try {
15            List<Item__c> items = [SELECT Id, Image__c FROM Item__c];
16
17            for (Item__c item : items) {
18                if (String.isNotBlank(item.Image__c)) {
19                    Matcher imgMatcher = Pattern.compile('<img[^>]+src\\s*=\\s*["\\'](.*?)["\\']').matcher(item.Image__c);
20
21                    while (imgMatcher.find()) {
22                        String img = imgMatcher.group(1);
23                        imageUrlMap.put(item.Id, img.unescapeHtml14());
24                    }
25                }
26            }
27        }
28    }
}
```

## **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:

```
surveyForm > surveyForm.js
1 // surveyFormLWC.js
2 import { LightningElement, track } from 'lwc';
3 import sendSurveyEmail from '@salesforce/apex/SurveyEmailController.sendSurveyEmail';
4
5 export default class SurveyFormLWC extends LightningElement {
6     @track recipientEmail = '';
7     @track surveyData = '';
8     recipientHandle(event){
9         if(event.target.label == 'Recipient Email'){
10             this.recipientEmail = event.target.value;
11         }else if(event.target.label == 'Survey Data'){
12             this.surveyData = event.target.value;
13         }
14     }
15     sendSurvey() {
16         // Validate input data if needed
17         console.log('email ' + this.recipientEmail);
18         console.log('survey ' + this.surveyData);
19         // Call the Apex method to send the survey email
20         sendSurveyEmail({ recipientEmail: this.recipientEmail, surveyData: this.surveyData })
21             .then(result => {
22                 // Handle success
23                 console.log('Email sent successfully:', result);
24             })
25             .catch(error => {
26                 // Handle errors
27                 console.error('Error sending email:', error);
28             });
29     }
30 }
```

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

export default class SurveyFormLWC extends LightningElement {
    @track recipientEmail = "";
    @track surveyData = "";
    recipientHandle(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
1  <!-- SurveyFormLWC.html -->
2  <template>
3      <lightning-card title="Survey Form">
4          <div class="slds-m-around_medium">
5              <!-- Your survey form HTML goes here -->
6              <!-- Include fields for survey questions and input elements -->
7              <!-- Example: -->
8              <label>Email:</label>
9              <lightning-input type="email" value={recipientEmail} onchange={recipientHandle} label="Recipient Email"></lightning-input>
10
11             <lightning-textarea name="input2" label="Survey Data" value="initial value" onchange={recipientHandle}></lightning-textarea>
12
13             <lightning-button label="Send Survey" onclick={sendSurvey}></lightning-button>
14         </div>
15     </lightning-card>
16 </template>
17 </lightning-card>
18 </template>
19 
```

```
<!-- 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
1  <?xml version="1.0"?>
2  <LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
3      <apiVersion>57.0</apiVersion>
4      <isExposed>true</isExposed>
5      <targets>
6          <target>lightning__AppPage</target>
7          <target>lightning__HomePage</target>
8      </targets>
9
10 </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

The screenshot shows the Lightning App Builder interface. At the top, there's a navigation bar with tabs for 'Lightning App Builder', 'Pages', and 'SurveyForms'. Below the navigation is a toolbar with icons for back, forward, search, and other functions. The main area is divided into two sections: 'Components' on the left and the 'Survey Form' component preview on the right. The 'Survey Form' component has fields for 'Email' and 'Recipient Email', a 'Survey Date' input, and a 'Send Survey' button. On the right side of the preview, there are configuration panels for the page. These panels include fields for 'Label' (set to 'SurveyForms'), 'API Name' (set to 'SurveyForms'), and 'Page Type' (set to 'App Page'). There are also dropdowns for 'Template' (set to 'Header and Left Sidebar') and 'Actions'. At the bottom of the preview, there are buttons for 'Activation...' and 'Save'. The 'Components' sidebar on the left lists various standard components like Accordion, Chatter Feed, Flow, and Launchpad, along with some beta components like LWC CRM Analytics Dashboard and OmniScript for Experience Cloud.

Save and activate.

## Activity- 2

### eCartItemSelector (JavaScript) code:

```
eCartItemSelector > eCartItemSelector.js
1 import { LightningElement, api, track, wire } from 'lwc';
2 import { NavigationMixin } from 'lightning/navigation';
3 import getItemsByCategory from '@salesforce/apex/eCartController.getItemsByCategory';
4 import createPurchaseRecord from '@salesforce/apex/eCartController.createPurchaseRecord';
5 import fetchImageUrl from '@salesforce/apex/eCartController.fetchImageUrl';
6 export default class ECartItemSelector extends NavigationMixin(LightningElement) {
7     @api recordId;
8     lwcIf = false;
9     lwcElse = true;
10    @track itemCategories = ['Grocery', 'Accessories', 'Electronics'];
11    @track selectedCategory = '';
12    @track items = [];
13    @track ImageUrl = [];
14    @api.isTrue = false;
15    @track cartItems = [];
16    @track imageMapped = [];
17    @wire(getItemsByCategory, { category: '$selectedCategory' })
18    wiredItems({ data, error }) {
19        if (data) {
20            this.items = data;
21            console.log(data);
22            console.log(this.selectedCategory);
23            this.items = this.items.map(currentItem => {
24                console.log(JSON.stringify(currentItem));
25                const matchingItem = this.imageMapped.find(item => currentItem.Id === item.Id);
26                if (matchingItem) {
27                    return { ...currentItem, url: matchingItem.value }
28                } else {
29                    return { ...currentItem }
30                }
31            });
32            console.log(data);
33        } else if (error) {
34            console.error(error);
35            console.log('error on Electronic ' + JSON.stringify(error));
36        }
37    }
}
```

```
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 imageUrl 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));
}
}

HandleCart(event){
    this.lwcElse = false;
    this.lwclf = true;
}

@wire(fetchImageUrl)
fetchImageUrlwired({ data, error }) {
    if (data) {
        console.log('data fetchImageUrl' + 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('error 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;
    const itemIndex=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;
  const selectedItem = this.cartItems.find(item => item.id === itemId);
  if (selectedItem) {
    selectedItem.Quantity = quantity;
    // Update any other relevant
  } data
}

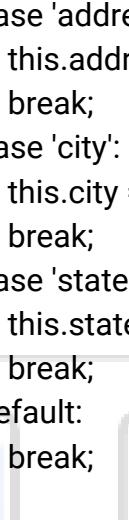
handleAddToCart(event) {
  const itemId = event.target.dataset.id;
```

```
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:

```
1 <template>
2
3     <lightning-card title="Customer Information Form" lwc:if={lwcElse}>
4         <!-- Customer Name -->
5         <lightning-input
6             data-id="customerName"
7             type="text"
8             label="Customer Name"
9             onchange={handleInputChange}>
10        </lightning-input>
11
12        <!-- Email -->
13        <lightning-input
14            data-id="email"
15            type="email"
16            label="Email"
17            onchange={handleInputChange}>
18        </lightning-input>
19
20        <!-- Phone -->
21        <lightning-input
22            data-id="phone"
23            type="tel"
24            label="Phone"
25            onchange={handleInputChange}>
26        </lightning-input>
27
28        <!-- Address -->
29        <lightning-input
30            data-id="address"
31            type="text"
32            label="Address"
33            onchange={handleInputChange}>
34        </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={HandleCart}></lightning-button>
```

```
</lightning-card>
```

```
<lightning-card lwc:if={lwclif}>
```

```
<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-item>

</lightning-layout>

<div lwc:if={istru}>

<c-e-cart-items-list cart-items={cartItems} istru={istru} recordid={recordId}
oncreatefire={handleCreatePurchase} onistru={lististru}
onremovefromcart={handleRemoveFromCart} onquantitychange={handleQuantityChange}
onaddtocar t={handleAddToCar t}>

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

</div>

</lightning-card>

```

**LEARN with Canisian**

### eCartItemSelector (XML) code:

```

eCartItemSelector > eCartItemSelector.js-meta.xml
1  <?xml version="1.0" encoding="UTF-8"?>
2  <LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
3      <apiVersion>58.0</apiVersion>
4      <isExposed>true</isExposed>
5
6          <targets>
7              <target>lightning__RecordPage</target>
8              <target>lightning__AppPage</target>
9              <target>lightning__HomePage</target>
10             <!--<target>lightning__Tab</target>-->
11             <!--<target>lightning__Inbox</target>-->
12             <!--<target>lightning__UtilityBar</target>-->
13             <!--<target>lightning__FlowScreen</target>-->
14             <!--<target>lightningSnipin__ChatMessage</target>-->
15             <!--<target>lightningSnipin__Minimized</target>-->
16             <!--<target>lightningSnipin__PreChat</target>-->
17             <!--<target>lightningSnipin__ChatHeader</target>-->
18             <!--<target>lightningCommunity__Page</target>-->
19             <!--<target>lightningCommunity__Default</target>-->
20             <!--<target>lightningCommunity__Page_Layout</target>-->
21             <!--<target>lightningCommunity__Theme_Layout</target>-->
22             <!--<target>lightningStatic__Email</target>-->
23             <!--<target>lightning_VoiceExtension</target>-->
24             <!--<target>analytics__Dashboard</target>-->
25             <target>lightning__RecordAction</target>
26         </targets>
27     </LightningComponentBundle>
28 
```

```
<?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:

```
CartItemSelector > eCartItemSelector.css
1  tr {
2    border-bottom: 1px solid #ddd;
3    object-fit: cover;
4  }
5  /* Example styling for buttons */
6  button {
7    background-color: #4CAF50;
8    border: none;
9    color: white;
10   padding: 10px 20px;
11   text-align: center;
12   text-decoration: none;
13   display: inline-block;
14   font-size: 16px;
15   margin: 4px 2px;
16   cursor: pointer;
17   border-radius: 5px;
18 }
19
20 button:hover {
21   background-color: #45a049;
22 }
23
24 button:active {
25   background-color: #3e8e41;
26 }
27
28 /* Example styling for quantity input fields */
29 input[type=number] {
30   width: 50px;
31   padding: 5px;
32   border: 1px solid #ccc;
33   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:

```
eCartItemList > eCartItemList.js  
1  import { LightningElement, api,track } from 'lwc';  
2  
3  export default class ECartItemList extends LightningElement {  
4      @api cartItems = [];  
5      @api istrue;  
6      @api recordid ;  
7      itemids =[];  
8      @track totalAmount = 0;  
9  
10     calculateTotalAmount() {  
11         this.totalAmount = 0;  
12         console.log("JSON.stringify(this.cartItems)" + JSON.stringify(this.cartItems))  
13         this.cartItems.forEach(item =>{  
14             this.itemids.push(item.Id);  
15             console.log(item.Price__c + item.Quantity)  
16             let i = item.Price__c * item.Quantity  
17             this.totalAmount = this.totalAmount + i;  
18         })  
19         console.log('total ' + this.totalAmount);  
20         console.log('this.items ' + this.itemids);  
21         this.dispatchEvent(new CustomEvent('createfire', { detail: this.itemids }));  
22     }  
23     handleRemoveFromCart(event) {  
24         const itemId = event.target.dataset.id;  
25         this.dispatchEvent(new CustomEvent('removefromcart', { detail: { itemId } }));  
26         if (cartItems.length < 1) {  
27             let i = false;  
28             this.dispatchEvent(new CustomEvent('onisttrue', { detail: { i } }));  
29         }  
30     }  
31 }
```

```
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('onisttrue', { detail: { i } }));  
  
}  
}
```



The logo features the word "LEARN" in large, light gray capital letters. The letter "N" has a red diagonal line through it. To the right of "LEARN" is the word "with" in a smaller, light gray sans-serif font. Below "with" is a yellow lightbulb icon with rays emanating from it. Underneath the "LEARN" and "with" text is the word "Sanjana" in large, stylized orange letters. A speech bubble shape is positioned to the left of the "Sanjana" text, containing the following code:

```
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:

```

eCartItemList > eCartItemList.html
1 <template>
2     <lightning-card>
3         <div class="cart-items-container">
4             <p>Cart Items:</p>
5             <table class="cart-items-table">
6                 <thead>
7                     <tr>
8                         <th>Item Name</th>
9                         <th>Quantity</th>
10                        <th>Price</th>
11                        <th>Action</th>
12                    </tr>
13                </thead>
14                <tbody>
15                    <template for:each={cartItems} for:item="item">
16                        <tr key={item.Id}>
17                            <td>{item.Name}</td>
18                            <td>{item.Quantity}</td>
19                            <td>
20                                ${item.Price__c}</td>
21                            <td><br>
22                                <lightning-button label="Remove" onclick={handleRemoveFromCart} data-id={item.Id}>
23                                    </lightning-button>
24                                </td>
25                            </tr>
26                        </template>
27                    </tbody>
28                </table>

```

<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><br>
<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 -->&nbsp;&nbsp;&nbsp;
    <lightning-button label="Buy Now" onclick={calculateTotalAmount}></lightning-button>
</div>
</lightning-card>
</template>
</lightning-card>
</template>

```

### eCartItemList (XML) code:

```

eCartItemList > eCartItemList.js-meta.xml
1  <?xml version="1.0" encoding="UTF-8"?>
2  <LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
3      <apiVersion>58.0</apiVersion>
4      <isExposed>true</isExposed>
5      <targets>
6          <target>lightning__AppPage</target>
7          <target>lightning__RecordPage</target>
8          <target>lightning__HomePage</target>
9      </targets>
10 </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:

```

eCartItemslist > cartutils.js
1 // Import necessary Apex methods
2 const applyReferralCodeDiscountApex = async (orderAmount, referralCode) => {
3   // Call the Apex class to apply referral code discount
4   const discountedOrderAmount = await applyReferralCodeDiscountApexMethod({ orderAmount, referralCode });
5   return discountedOrderAmount;
6 };
7
8 const isValidReferralCodeApex = async (referralCode) => {
9   // Call the Apex class to validate the referral code
10  const isValid = await isValidReferralCodeApexMethod({ referralCode });
11  return isValid;
12 };
13
14 // Your Apex class methods should have the @AuraEnabled annotation
15 const applyReferralCodeDiscountApexMethod = async (params) => {
16   // Your logic to apply referral code discount in Apex
17   // Example: Apply a 10% discount if referral code is valid
18   const { orderAmount, referralCode } = params;
19   if (await isValidReferralCodeApexMethod({ referralCode })) {
20     const discountPercentage = 0.10;
21     const discountAmount = orderAmount * discountPercentage;
22     const discountedOrderAmount = orderAmount - discountAmount;
23     return discountedOrderAmount;
24   }
25   // Return original order amount if referral code is invalid
26   return orderAmount;
27 };
28
29 const isValidReferralCodeApexMethod = async (params) => {
30   // Your logic to validate the referral code in Apex
31   // For simplicity, let's assume any non-empty code is valid
32   const { referralCode } = params;
33   return String.isNotBlank(referralCode);
34 };

```

```
// 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 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,

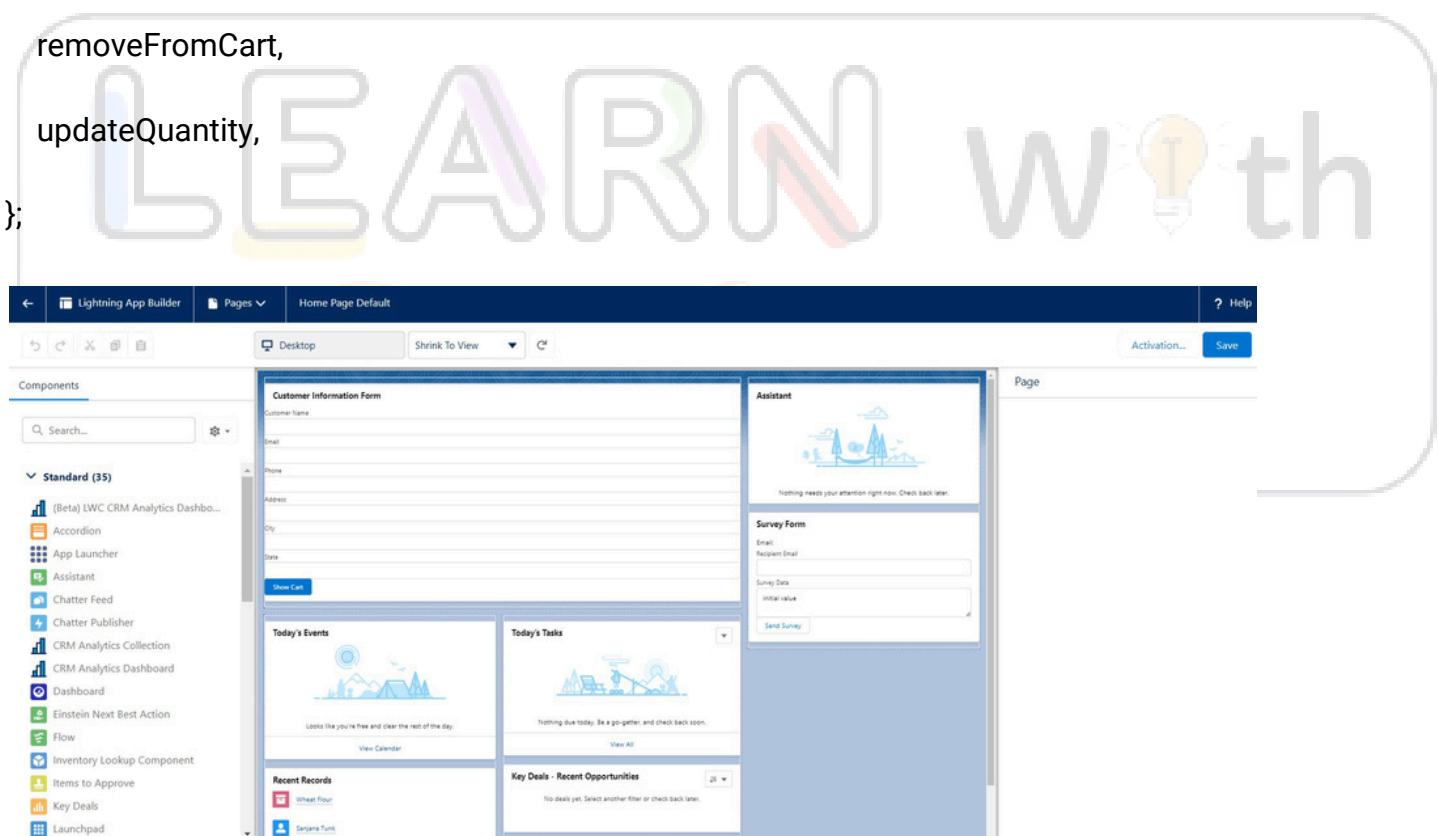
isValidReferralCode: isValidReferralCodeApex,

addToCart,

removeFromCart,

updateQuantity,

};



Deploy the source code.

Drag the custom component on the App page