

 dreamforce®

# Enhance Flows with Apex and Lightning Web Components

Flow + Apex + LWC = A Perfect Combination!

Dreamforce 2024



**Andrés Pérez**

Senior Manager, Solution Architect Lead



**Joseph Flowers**

Senior Mgr, Learning Solutions Architect

# Rahul Shah

Salesforce  
Manager, APAC Delivery



3

## Forward looking statements



This presentation contains forward-looking statements about, among other things, trend analyses and statements regarding future events, anticipated growth and industry prospects, and our strategies, expectation or plans regarding product releases and enhancements. The achievement or success of the matters covered by such forward-looking statements involves risks, uncertainties and assumptions. If any such risks or uncertainties materialize or if any of the assumptions prove incorrect, results or outcomes could differ materially from those expressed or implied by these forward-looking statements. The risks and uncertainties referred to above include those factors discussed in Salesforce's reports filed from time to time with the Securities and Exchange Commission, including, but not limited to: our ability to meet the expectations of our customers; uncertainties regarding AI technologies and its integration into our product offerings; the effect of evolving domestic and foreign government regulations; regulatory developments and regulatory investigations involving us or affecting our industry; our ability to successfully introduce new services and product features; our ability to execute our business plans; the pace of change and innovation in enterprise cloud computing services; and our ability to maintain and enhance our brands.

Last updated: April 25, 2024

4

# Copyright



© Copyright 2000-2024 salesforce.com, inc. All rights reserved. Various trademarks held by their respective owners.

Rights of ALBERT EINSTEIN are used with permission of The Hebrew University of Jerusalem. Represented exclusively by Greenlight.

This document contains proprietary information of salesforce.com, inc., it is provided under a license agreement containing restrictions on use, duplication and disclosure and is also protected by copyright law. Permission is granted to customers of salesforce.com, inc. to use and modify this document for their internal business purposes only. Resale of this document or its contents is prohibited.

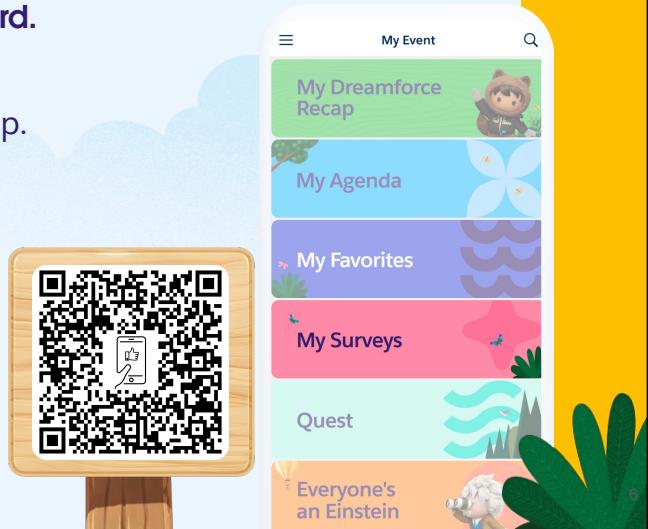
The information in this document is subject to change without notice. Should you find any problems or errors, please log a case from the Support link on the Salesforce home page. Salesforce.com, inc. does not warrant that this document is error-free.



## Coffee on us!

The first 4,000 attendees to provide feedback on this event will receive a \$5 Starbucks gift card.

1. Open the Salesforce Events mobile app.
2. Navigate to **My Event**.
3. Select **My Surveys**.
4. Complete four session surveys and present the completed Event Survey page at Badge Pickup to redeem.\*



\*Restrictions apply. See rules at [sforce.co/survey-terms](https://sforce.co/survey-terms)



# Thank you

7

## Agenda

- 01** Using record-triggered flows
- 02** Using Apex in flows
- 03** Combining lightning web components (LWCs) and flows
- 04** Wrap and Q&A

salesforce



8

## Setup: Set up your environment for this session



### Instructions Summary:

1. Log in to the online session site.
2. Log in to your training session Salesforce org.
3. Enable and launch Code Builder.

9

## Automating business processing by using flows

Code completion enhanced!



10

# Types of flows



In this session we'll look at these two types of flows

The screenshot shows the 'Flow Builder' interface with the title 'Select Type'. On the left, there are 'Tools' and 'Elements' panels. The main area is titled 'Recommended' and lists six flow types:

- Screen Flow**: Guides users through a business process that's launched from Lightning pages, Experience Cloud sites, quick actions, and more.
- Record-Triggered Flow**: Launches when a record is created, updated, or deleted. This autolaunched flow runs in the background.
- Schedule-Triggered Flow**: Launches at a specified time and frequency for each record in a batch. This autolaunched flow runs in the background.
- Platform Event—Triggered Flow**: Launches when a platform event message is received. This autolaunched flow runs in the background.
- Autolaunched Flow (No Trigger)**
- Record-Triggered Orchestration**

At the bottom right of the screen is the number '11'.

## Record-triggered flows run in the background (no UI)



Automatically executed on Data Manipulation Language (DML) events

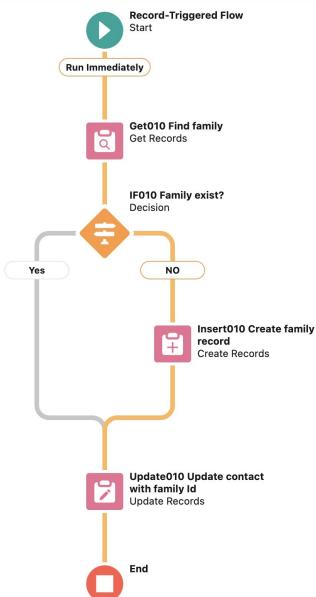
The diagram illustrates a Record-Triggered Flow for managing family records. The flow starts with a 'Record-Triggered Flow Start' (Run Immediately). It then performs a 'Get010 Find family Get Records' step. A decision diamond 'IF010 Family exist?' follows. If the condition is 'NO', it executes an 'Insert010 Create family record Create Records' step. If the condition is 'YES', it executes an 'Update010 Update contact with family Id Update Records' step. The flow concludes with an 'End' step.

Callouts provide additional context:

- Which sObject type will trigger the flow?
- Which DML operations can trigger the flow? (Created, Updated, Deleted)
- Are there any entry conditions to decide if the flow triggers?
- Need an async path for external system access post-commit?
- Flow optimization choice

The 'Flow optimization choice' callout points to the 'Optimize the Flow for' section in the configuration pane, which includes options for 'Fast Field Updates' and 'Actions and Related Records'.

# Deciding between flow optimization options



## Fast Field Updates



- Update fields on the record that **triggers the flow** to run
- This **higher-performance** flow runs *before* the record is saved to the database



## Actions and Related Records

- Update **any** record and perform actions, like sending an email
- This **more flexible** flow runs *after* the record is saved to the database

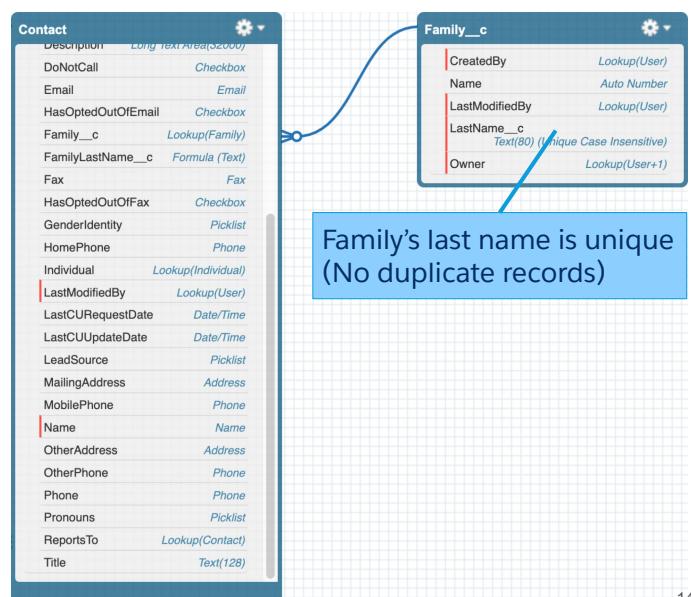
13

# Linking a contact with a family record



Business requirement for the exercise

- Contacts** should be related to their **family** records, based on their **last name**
- Whenever a **contact** is **created** or **updated**, find a related **family** record and **link** them together
- If a **family** record is not found, a new family record should be **created** and **linked**



14

## Consider this before we start the exercises:



~~Apex developers don't write code!~~

We invest more time in reading code than in writing it!

- Code reviews
- Troubleshooting / Debugging
- Einstein for Developers

In the exercises, you will read and analyze pre-built code examples

15

## Exercise 1: Analyze and test a Salesforce flow



### Goals:

Explore limits of using flows to automatically insert related family records when a collection of contact record change.

A cartoon illustration of a brown bear wearing a black hoodie. The bear is holding a white rectangular sign with the text "Join Me" written on it in a stylized font.

Join Me

### Instructions Summary:

1. Debug a pre-built Salesforce flow.
2. Analyze what the flow does.
3. Test behavior of an activated vs. deactivated flow.
4. Test the flow.
5. Identify issues with and limitations of the flow.

16

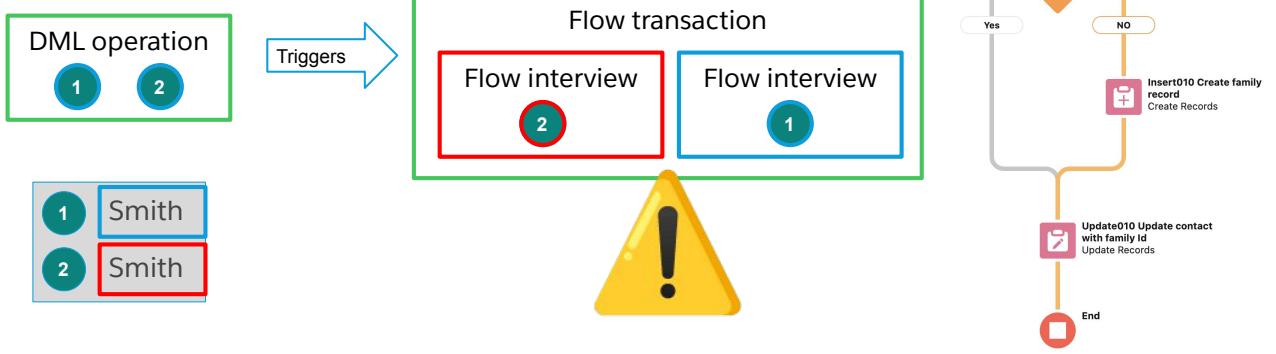
# Why did the test fail?

The error said that duplicate records cannot be inserted



# Working separate but together!

Flow interviews vs. Flow transactions



# It works!

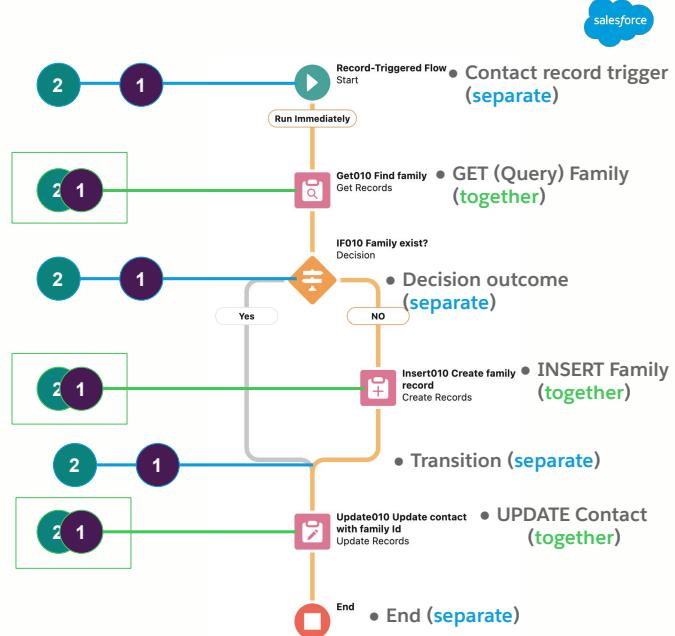
With different new last names

- Bulkification points
  - SOQL / DML operations
  - Screen elements
  - Pause actions
  - Apex calls (optional)

DML Operation



Contact records



19

# It fails!

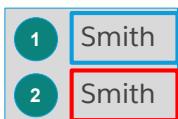
With the same new last names

- Inserts duplicate records in the flow transaction (bulkified operation)

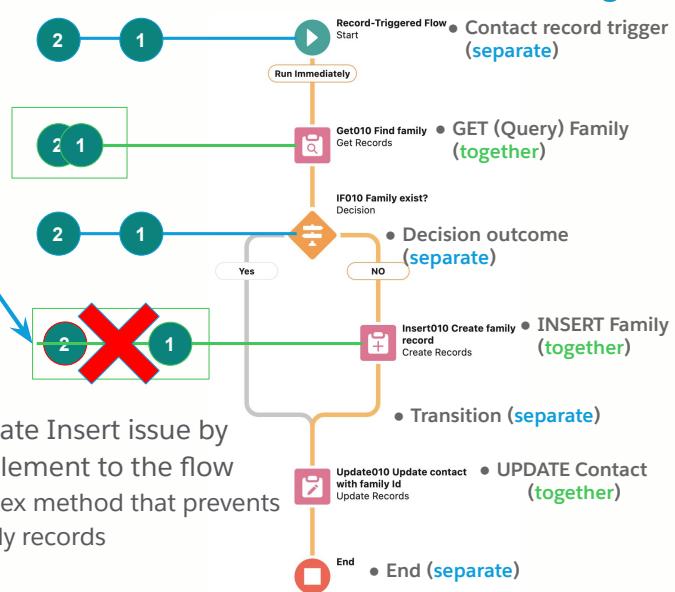
DML Operation



Contact records



- Next, we'll fix this duplicate Insert issue by adding an Apex action element to the flow
  - This element calls an Apex method that prevents inserting duplicate Family records

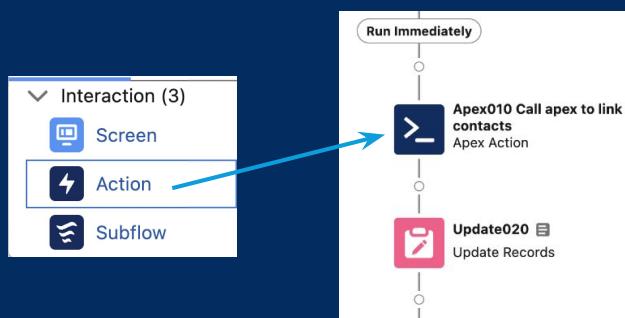


20



# Customize flows by adding Apex actions

Why and how to use Apex in flows

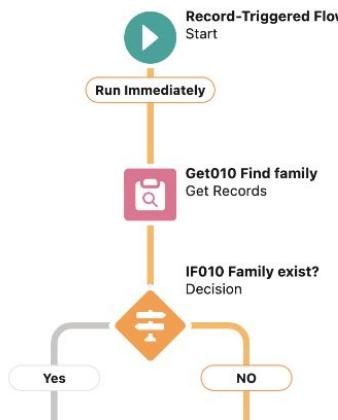


21

## Record-triggered flows vs. Apex triggers



- **Flows** let you create customized actions visually in Flow Builder **without writing code**
- **Apex** lets you create customized actions **with code**, usually in an IDE like Code Builder
  - Recommended to use **flows** as much as possible



```
trigger phoneFieldTrg on Account (before Insert)
{
    if(trigger.isBefore && trigger.isInsert)
    {
        if(!trigger.new.isEmpty())
        {
            for(Account acc : trigger.new)
            {
                if(acc.Phone == null || acc.Phone == '')
                {
                    acc.addError('You cannot insert account with phone field empty');
                }
            }
        }
    }
}
```

# Answers: Comparing record-triggered flows vs. Apex triggers

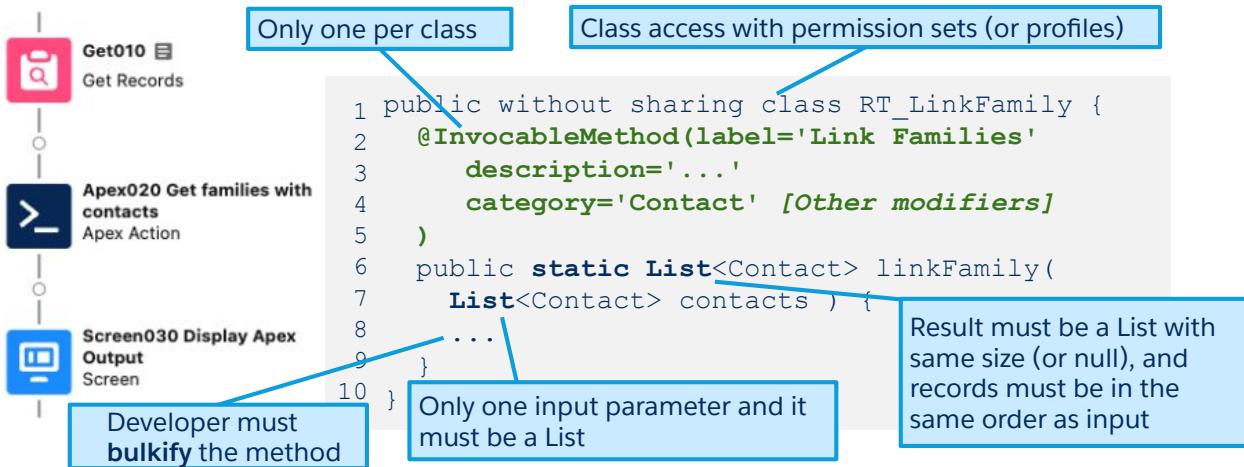


	Record-triggered flows	Apex triggers
Stops DML transaction (validations)	Yes	Yes
Entry condition	Automatic	Manual
Asynchronous path	Automatic	Manual
Undelete event	No	Yes
How many (per object/event)?	Many	One (Best Practice)
Queries	Simple	Relationships, Aggregates
Bulkification	Automatic	Manual
New record	\$Record (1 record)	Trigger.new (list)
Old record	\$Record__Prior (1 record)	Trigger.old (list)

24

## Combine flows and Apex

Invoke Apex methods from flows by using Action elements



# Solution: Can we use flows or do we need Apex? (or both)



No code

Pro code

	F.F.U. flow	A.R.R. flow + Apex Class	Apex Trigger / Classes
Same-Record Field Updates	Available (Best)	Available (Not Ideal)	Available (Not Ideal)
CRUD operations	Not Available	Available (Best)	Available (OK)
Asynchronous Processing	Not Available	Available (Best)	Available (OK)
Complex List Processing with Map/Set	Not Available	Not Available	Available (Best)
High-Performance Batch Processing	Available (Not Ideal)	Available (Not Ideal)	Available (Not Ideal)
Complex SOQL Queries	Not Available	Not Available	Not Available
Custom Validation Errors	Available (Best)	Available (OK)	Available (OK)

<https://architect.salesforce.com/decision-guides/trigger-automation>

27

## Exercise 2: Analyze how to fix issues and limitations in a flow by using an Apex action element



### Goals:

Explore how an APEX invocable method is used to customize a flow to link Contact and Family records.

### Instructions Summary:

1. Import the starter app from GitHub to Code Builder.
2. Authorize Code Builder with your training session Salesforce org.
3. Run Apex tests for a flow in Code Builder.
4. Analyze how Apex code is connected with the flow.
5. Analyze the Apex code used to customize the flow.
6. Analyze how the @invocableMethod annotation is used.
7. Analyze the permission set used with the flow.

Join Me



28



# Exposing Apex variables to flows

Customize the user experience and perform more complex actions



29

## Share Apex variables in custom data classes by using the @invocableVariable annotation



```
public with sharing class RT_Demo {  
    @InvocableMethod(label='Sample' description='.' category='Contact')  
    public static List<Result> execute(List<Request> requests) {  
        List<Result> responseWrapper = new List<Result>();  
        ...  
        //Inner classes - Custom data types  
        public class Request {  
            public String notAccessible;  
            @InvocableVariable(label='Input Contacts' required='true')  
            public List<Contact> inputContact;  
            @InvocableVariable(label='Family of all Contacts' required='false')  
            public Family theFamily;  
        }  
  
        public class Result {  
            @InvocableVariable(label='The Ultimate Answer' required='true')  
            public String theAnswer;  
            @InvocableVariable(label='Output Contact' required='true')  
            public RT_DemoAE outputContact;  
        }  
    }  
}
```

Each custom type can contain multiple invocable variables

The label appears in Flow Builder as info to help admins use the variable

Custom Apex-defined class

# Making custom data attributes available to flows by using the @AuraEnabled annotation



```

1 public with sharing class RT_Demo {
2     @InvocableMethod(label='Sample' description='...' category='Contact')
3     public static List<Result> execute(List<Request> requests) {
4         List<Result> responseWrapper = new List<Result>();
5         Result response = new Result();
6         response.outputContact = new RT_DemoAE();
7         response.outputContact.myContact = requests[0].inputContact[0];
8     }
9     public class Request {
10        @InvocableVariable(label='Input Contacts' required='true')
11        public List<Contact> inputContact;
12    }
13    public class Result {
14        @InvocableVariable(label='Output Contact' required='true')
15        public RT_DemoAE outputContact;
16    }
17 }
```

- Custom data type for variable
- Must be in a separate outer class file

```

public class RT_DemoAE {
    @AuraEnabled
    public Contact myContact { get; set; }
    @AuraEnabled
    public Integer theAnswer { get; set; }
}
```

- @AruaEnabled makes the property accessible in flows
- Serializes the property

31

## Example of Flow elements accessing @AuraEnabled attributes

1. External Apex data class is used in an invocable method response.



2. The next flow element can access the @AuraEnabled variable.

```

public class EX03AE_Response {
    @AuraEnabled
    public Integer theAnswer { get; set; }

    @AuraEnabled
    public List<EX03AE_FamilyWithContacts> familiesWithContacts { get; set; }
}
```

Edit Variable Apex020

* API Name	response
Description	
* Data Type	Apex-Defined
* Apex Class	EX03AE_Response
Availability Outside the Flow	
<input type="checkbox"/> Available for input	

Edit Screen Screen 030

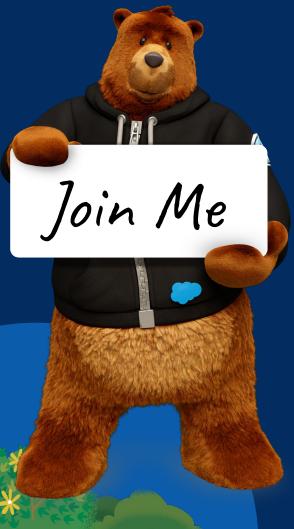
Screen Properties	
* Label	Screen030 Display Apex Output
* API Name	Screen030
Description	Displays "The Answer"

The Answer: {!response.theAnswer}

32

## Exercise 3: Analyze how to handle complex data in a screen flow by using Apex actions

salesforce



### Goals:

Explore how @invocableVariable and @AuraEnabled APEX annotations are used to expose custom data types to a flow.

### Instructions Summary:

1. Identify the flow elements used in the solution.
2. Identify how Apex code is connected to the flow.
3. Identify how @InvocableVariable and @AuraEnabled annotations are used.

33

## Combining Lightning web components (LWCs) and screen flows

Building reusable components of customized screens



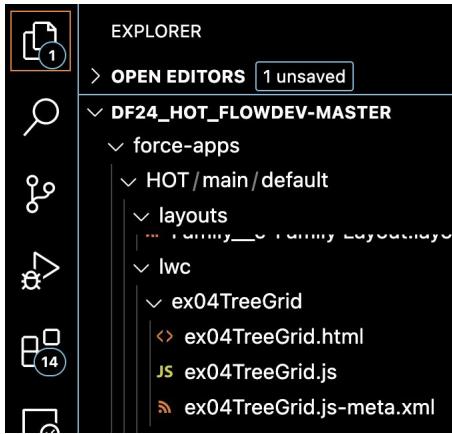
34

# Comparing how LWCs and screen flows are built



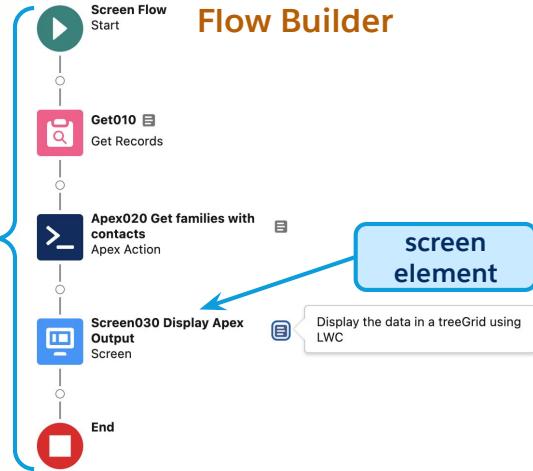
- **LWCs** are built in a standard web development tool (IDE) such as Salesforce Code Builder
  - Pro-code

## LWC in an IDE



## Screen flow

## Flow Builder



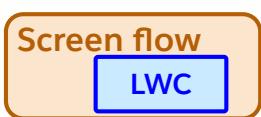
35

# Ways to use screen flows and LWCs together



Two opposite use cases to reuse screen flows and LWCs

## First use case



- **Screen flow** UI component embeds (display) **LWC** element(s) inside
  - To customize part of a **screen** element user experience (UX) by reusing an **LWC**

## Second use case

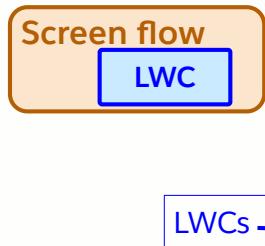


- The **LWC** embeds (displays) a **screen flow** inside
  - To reuse **screen flow** components that can be built with no-code
  - To restrict access to a screen flow to specific sections of the Lightning Experience (via the target property of the LWC)
- You can even combine both of these patterns in your UI
  - Leverage the best of no-code and pro-code options
  - Promote reusability

36

# Why embed an LWC in a screen flow?

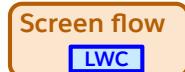
## First use case



- If OOTB **flow** components don't meet your needs
  - Add **LWC** screen components as part of a screen's layout
  - Build reusable advanced UI options such as pixel perfect, 4+ columns, tabs, colors, buttons, sliders, custom CSS, ...
- To create **interactive screens** (onblur, onfocus, onchange) with events not available to a **flow**
  - On the **client-side**, perform extra validation not available OOTB in the **flow**
  - Dynamically control requiredness, formatting, visibility, and/or read-only parts of the UI

37

## Example: How to embed an LWC in a screen flow



Make an **LWC** available to a **screen flow**

**LWC component.js-meta.xml**

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
3   <apiVersion>57.0</apiVersion>
4   <isExposed>true</isExposed>
5   <masterLabel>Best Component Ever</masterLabel>
6   <targets>
7     <target>lightning__FlowScreen</target>
8   </targets>
9   <targetConfigs>
10    <targetConfig targets="lightning__FlowScreen">
11      <property name="startDate" label="Start Date" type="Date" role="inputOnly"/>
12      <property name="name" label="Account Name" type="String" role="outputOnly"/>
13      <property name="account" label='Account' type="@salesforce/schema/Account[]"/>
14      <property name="data" label='Custom Data' type="apex://CustomClass"/>
15    </targetConfig>
16  </targetConfigs>
17 </LightningComponentBundle>
```

Specify the **LWC** is available for **screen flows**

Properties passed between **screen flow** and **LWC**

Complex data types (**Apex**)

38

# How an LWC dispatches events to the screen flow



Screen flow  
LWC

LWC code to dispatch screen flow events to the "parent" screen flow

```
1 <!-- LWC component ES Module JavaScript .js file -->
2 import { LightningElement, api, track} from 'lwc';
3 import { FlowAttributeChangeEvent, FlowNavigationNextEvent } from 'lightning/flowSupport';
4
5 export default class LwcInFlow extends LightningElement {
6     @track _someData;
7     @api availableActions = [];
8     @api get someData() { return this._someData; }
9     set someData(value) { this._someData = {...someData}; }
10 ...
11     demo() {
12         this.dispatchEvent(new FlowAttributeChangeEvent('someData', this._someData));
13         ...
14         if (this.availableActions.find((action) => action === 'NEXT'))
15             this.dispatchEvent(new FlowNavigationNextEvent());
16     }
}
```

LWC notifies screen flow when values changed  
("Reactive Screens")

Determine available screen flow actions

Example: If the LWC is the last screen component, don't dispatch Next event

Control screen flow navigation  
Back, Next, Pause, Finish

39

## Exercise 4: Analyze how to code and embed a Lightning Web Component (LWC) in a screen flow



### Goals:

Explore how LWC are used with flows in the training session scenario to display Contact records grouped by the Family field.

### Instructions Summary:

1. Analyze how the solution flow works.
2. Analyze how the solution was built in Flow Builder.
3. Analyze how the LWC was built in VS Code.

40

## Summary: Flows + Apex + LWCs



Leverage and combine the best of no-code and pro-code options

- **Flows** provide a graphical (no-code) way to improve the UI and add logical structure to the user experience based on business requirements
  - Helps you to build UI screens and automated business processes without needing to build out your own infrastructure
- **Apex** and **LWCs** are helpful to enhance and customize **flows** to meet specific requirements not supported by OOTB flow components
  - Example: Using Maps or Sets, avoid DML bulkification issues, custom input/output objects
- **Apex** code can be leveraged directly from a **flow** to customize the behavior
- **LWCs** are customized with HTML and JavaScript code, and can be used inside **flows**

41

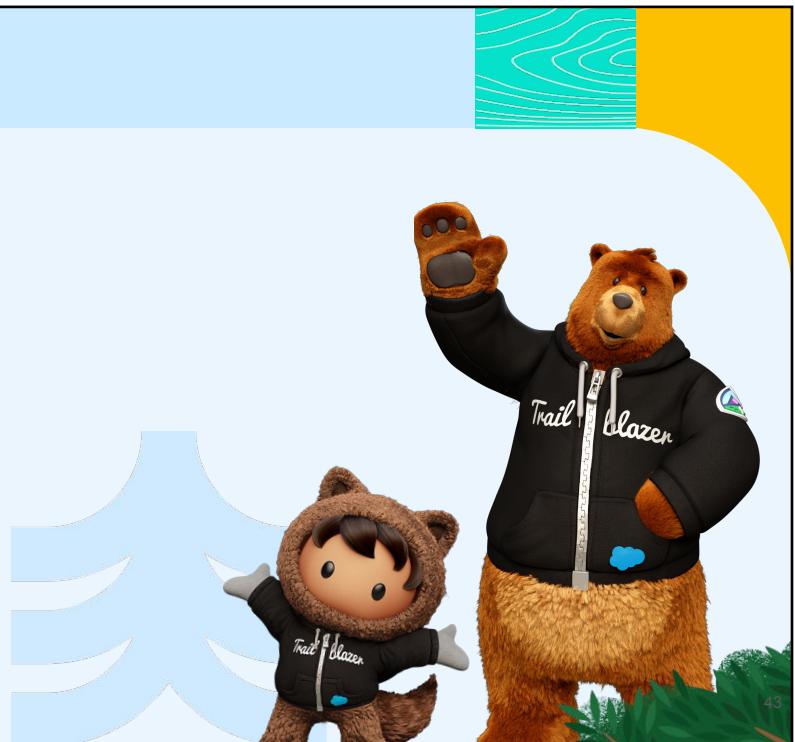


# Thank you



42

# Q&A

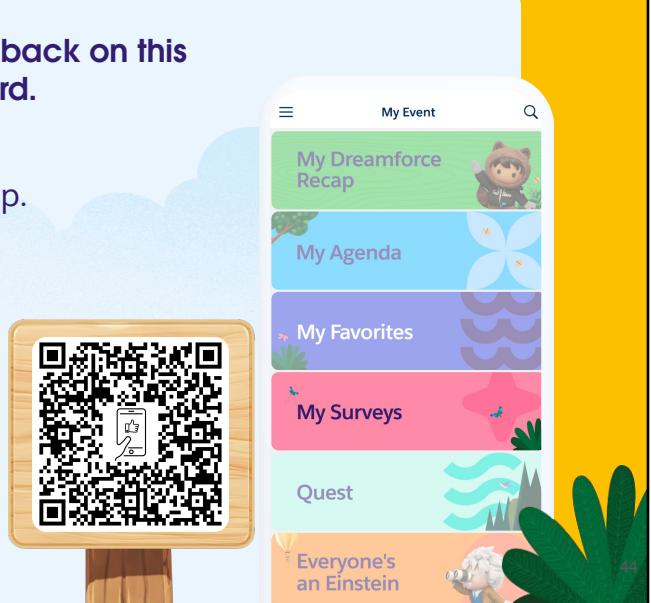


## Coffee on us

The first 4,000 attendees to provide feedback on this event will receive a \$5 Starbucks gift card.

1. Open the Salesforce Events mobile app.
2. Navigate to **My Event**.
3. Select **My Surveys**.
4. Complete four session surveys and present the completed Event Survey page at Badge Pickup to redeem.\*

\*Restrictions apply. See rules at [sforce.co/survey-terms](http://sforce.co/survey-terms)



salesforce

# Thank you



45