APEX AND VISUALFORCE ARCHITECTURE RESOURCE GUIDE

Table of Contents

Copyright

Introduction

Who is the Salesforce Certified Platform Developer I?

The candidate looking to obtain the Salesforce Certified Platform Developer I Certification can assess the architecture environment and requirements and designs solutions on the Force.com platform that meet sharing and visibility requirements. The candidate has experience communicating solutions and design trade-offs to business stakeholders. The Salesforce Platform Developer I Certification leverages the knowledge and content from the Apex and Visualforce Architecture eBook.

The experience and skills that the candidate should possess are outlined below:

- Has 5+ years of delivery experience.
- Provides experienced guidance on the appropriate choice of platform technology.
- Understands architecture options, design trade-offs, and has the ability to communicate design choices.
- Aware of globalization (multi-language, multi-currency) application design considerations on a project.
- Able to identify development-related risks, considerations, and limits for the platform.
- Experience with different types of development patterns/principles.
- Aware of platform-specific design patterns and key limits.
- Has held a technical architect role on multiple complex deployments **or** has gained equivalent knowledge through participation and exposure to these types of projects [either with single or multiple projects].
- Familiarity with code development on the Force.com platform.
- Understands object-oriented design patterns.
- Experience with project and development lifecycle methodologies.
- Understands strategies to build an optimized and performant solution.

Learn Materials

General Resources

Here are some comprehensive general resources that are a good starting place for your self-paced study.

Force.com Apex Code Developer's Guide

To revolutionize the way that developers create on-demand applications, Salesforce introduces Force.com Apex code, the first multi-tenant, on-demand programming language for developers interested in building the next generation of business applications.

Apex Workbook

This workbook provides an introduction to the Apex programming language and the contexts in which you can use Apex, such as triggers. This workbook does assume you know a little about programming.

Apex Code Best Practices

As with any programming language, there are key coding principles and best practices that will help you write efficient, scalable code. This article illustrates many of the key best practices for writing and designing Apex Code solutions on the Force.com platform.

Introduction to Force.com

This training introduces you to the Force.com platform features and functionality.

Introduction to Apex

Get an introduction to Apex and discover what this on-demand programming language is capable of, what makes it different from other programming languages, how it works with the Force.com IDE, and where it can be authored and executed.

Working with Web Services

Learn all about creating your own custom Web Services for handling inbound requests as well as Apex callouts for making outbound calls to external Web Services.

Data Types and Logic

Discover the different data types available in Apex, including the very useful collection types. Learn the syntax for basic logic structures, as well as how they should be used to achieve best performance when programming in the cloud.

Object-Oriented Programming in Apex

Learn the basics around creating classes in the cloud. Understand how the class and method access modifiers differ from other programming languages. Ensure that you are providing the appropriate level of record-access by leveraging on the existing Force.com sharing model.

Records in the Database

Find out how the Force.com platform handles sObject relationships and how to use those within Apex. Learn how to retrieve and submit data to the Force.com database and discover the basics of working with data sets in the cloud.

Deploying Apex Code

Find out all about migrating your Apex code from the development environment into a production environment. Learn about the tools available for deployment and the various sandboxes available for development. Details include specific deployment steps, and tips for training end users in the new production environment.

[Receiving and Sending Emails Through Apex]

Create a Salesforce email service for handling inbound messages. Learn to send outbound messages directly from Force.com. This module includes thorough setup instructions, as well as information about creating debugging and email logs.

[Advanced Topics]

Continue to hone your Apex skills with these advanced topics. Subjects include Dynamic Apex, Visualforce controllers, Apex schedulers, and custom settings objects that can be used to store configuration data.

[Visualforce Developer's Guide]

Salesforce introduces Visualforce, the next-generation solution for building sophisticated custom user interfaces on the Force.com platform.

[Overview of Visualforce]

Refresh your memory on the basics of programming Visualforce components. Review syntax for tagging, binding, and expressions.

Please register in the Salesforce Success Community and join our Architect Success group here.

General Overview

The following pages will introduce you to various focuses within the Apex and Visualforce. You will be introduced to relevant objectives that require a very specific set of skills and the curated learning materials that will help you to achieve them.

- Design and Functionality
- Performance and Scalability
- Maintainability and Reuse

Each learning resource has a related skill level: **Beginner**, **Intermediate**, or **Advanced**. Resources marked **Core** cover essential concepts, while those marked **Recommended** provide additional materials for further edification.

1. Design and Functionality

In this section, you will find the resources that will allow you to understand the capabilities of Apex and Visualforce code and when to apply coded solutions to meet business requirements.

Intro to Apex Code for Programmers Webinar

If you are familiar with object-oriented languages like Java or C#, Apex may be the language you already almost know. Apex is the cloud-based programming language used on the Salesforce1 Platform to take your enterprise applications to the next level. In this webinar, get an introduction to how Apex is similar to other languages, how you can start coding in Apex with just a web browser, and an overview of the many functions Apex can perform for your applications and users.
Tags: Beginner, Core

Visual Development – When to Click Instead of Write Code

Force.com started out as a declarative development platform in its early days and continues to provide a rich set of declarative development features to this day, so let's dig in and find out what we can (and should) do with clicks instead of writing code.
Tags: Beginner, Core

Implementing Triggers

Triggers are unique to Force.com and can execute when any data is saved. Learn about the different types of Apex triggers and the context for which to use each of them. Specifics include creating and executing triggers, and the proper syntax and variables to use with them.
Tags: Beginner, Core

1.2 Describe the continuum of UI options available on the platform and summarize capabilities and trade-offs of using each technique.

Less is More: Visualforce Page Usability and Performance

When you group and present smaller sets of data, you can build smaller pages that load faster, or update only a portion of a page instead of reloading the entire page repeatedly.
Tags: Beginner, Core

Apex UI Buttons

In the following example, we'll create a button on the Account object, and the logic behind this button will set the industry on the account to "Spice Trading" and reload the page. This pattern will work for any object, and can contain any Apex logic you need for your application.
Tags: Beginner, Recommended

Mashups: The What and Why

This first article provides an introduction to the core concepts found in mashups, examining the background, techniques, and consequences of using, creating, and enabling mashups in your IT infrastructure.
Tags: Beginner, Recommended

Building Content-Rich Visualforce Pages

In this blog post, you can learn about the benefits of storing and managing images in Salesforce CRM Content, and about how to use the power of Salesforce CRM Content and Visualforce to build dynamic pages.
Tags: Intermediate, Recommended

Ask the PM – What is Force.com Canvas?

Force.com Canvas is an exciting new feature for integrating new or existing Web apps into Salesforce.
Tags: Intermediate, Recommended

1.3 Given a scenario, describe how you would leverage the sharing and visibility model in code solution.

Enforcing CRUD and FLS

After an introduction to object- and field-level security, this article looks at the different techniques that Force.com applications can use to enforce a customer's security settings.
Tags: Beginner, Core

Understanding Apex Managed Sharing

Sharing is the act of granting a user or group of users permission to perform a set of actions on a record or set of records. Sharing access can be granted using the Salesforce user interface and Force.com, or programmatically using Apex.
Tags: Intermediate, Recommended

1.4 Describe security best practices to ensure Visualforce and Apex code is developed per OWASP standards.

Secure Coding Guidelines

The following documentation walks you through the most common security issues salesforce.com has identified while auditing applications built on or integrated with Force.com.
Tags: Intermediate, Core

1.5 Describe the object-oriented design principles and design patterns that were taken into consideration when determining the appropriate architecture for a solution and articulate typical design patterns available for use with Apex and Visualforce.

Apex Design Patterns - Singleton

This article describes how to implement a Singleton design pattern in Apex. The Singleton pattern attempts to solve the issue where you are repeatedly using an object instance but only wish to instantiate it once within a single transaction context.
Tags: Intermediate, Core

Implementing Idempotent Operations

This post introduces you to idempotence, explains idempotence in the context of Salesforce system integrations, and gives you links to some additional resources that you can review for more information about idempotence.
Tags: Intermediate, Core

Apex Design Patterns and Other Tips

Here are some words of wisdom which can help an Apex developer navigate some of the issues when creating more complex applications.
Tags: Advanced, Recommended

1.6 Recommend and justify the appropriate use and applicability of the Model View Controller pattern relative to business requirements.

An Introduction to Visualforce

This article introduces Visualforce. It illustrates the major areas of functionality, provides an example of the MVC paradigm in action, shows how to include database integration, and demonstrates how to create your own components.
Tags: Beginner, Core

1.7 Describe types, considerations, and trade-offs when designing Apex controllers.

Custom Controllers and Controller Extensions

Use custom controllers when you want your Visualforce page to run entirely in system mode, which does not enforce the permissions and field-level security of the current user.
Tags: Beginner, Core

Visualforce Standard Controllers

Learn how to use standard controllers to define what data and behaviors are available to users when they interact with a Visualforce component.
Tags: Beginner, Core

Visualforce Controller Extensions and Custom Controllers

Learn when and how to use the extra functionality provided by two additional Visualforce constructs: extensions to standard controllers and custom controllers.
Tags: Beginner, Core

Further Visualforce Controller Topics

Explore additional topics related to Visualforce controllers, including: creating wizards to simplify lengthy tasks; using the keyword "transient" to easily reference non-persisted data; and using dynamic Visualforce to create permutations of Visualforce pages; and testing Visualforce pages.
Tags: Intermediate, Core

1.8 Describe when to apply the appropriate Apex functionality, such as custom settings, when to use synchronous vs. asynchronous patterns, and all the available execution contexts (e.g., batch, trigger, callout, etc.) when Apex should be used on the platform and articulate appropriate scenarios for each of them.

Asynchronous Processing in Force.com

This paper is for experienced technical architects who work with Salesforce deployments who want to have a better understanding of Force.com asynchronous processing. This will help an architect build effective design patterns around asynchronous processing.
Tags: Intermediate, Core

Force.com Apex Code Developer's Guide: Future Annotation

Use the future annotation to identify methods that are executed asynchronously. When you specify future, the method executes when Salesforce has available resources.
Tags: Intermediate, Core

Named Credentials Overview

A named credential specifies the URL of a callout endpoint and its required authentication parameters in one definition. You can simplify the setup of authenticated Apex callouts by specifying a named credential as the callout endpoint.
Tags: Intermediate, Core

---

4 Steps to Successful Asynchronous Processing with Force.com

This post is a quick survival guide that can help you to avoid thread starvation and resource load competition, maintain transaction response times, and preserve the productivity and happiness of your org's users.
Tags: Intermediate, Recommended

1.9 Given a scenario, describe how batch apex and the flex queue could be applied.

Force.com Apex Code Developer's Guide: Using Batch Apex

To use batch Apex, write an Apex class that implements the Salesforce-provided interface Database.Batchable, and then invoke the class programmatically.
Tags: Intermediate, Recommended

Apex Flex Queue: Batch Apex Liberated

The Apex Flex Queue brings an eagerly awaited update to batch Apex processing, allowing more jobs to be submitted at once and bringing greater control over queued jobs.
Tags: Advanced, Core

Monitoring the Apex Flex Queue

Use the Apex Flex Queue page to view and reorder all batch jobs that have a status of Holding.
Tags: Advanced, Recommended

1.10 Demonstrate an understanding of Apex / Visualforce governor limits on coding patterns and methods to avoid hitting those limits, as well as anti-patterns for development on the platform.

Salesforce Limits Quick Reference Guide

This guide provides commonly referenced limits for Salesforce. This guide may not cover all limits or may contain limits that don't apply to your organization.
Tags: Beginner, Core

Designing Force.com Applications That Avoid Hitting Concurrent Request Limits

In this blog post, we'll discuss best practices that you can follow to improve the performance of your application and help you avoid this limit.
Tags: Advanced, Recommended

1.11 Describe the implications of the order of execution of transactions within the platform.

Force.com Apex Code Developer's Guide: Triggers and Order of Execution

When you save a record with an insert, update, or upsert statement, Salesforce performs the following events in order.
Tags: Intermediate, Core

1.12 Articulate the importance of unit testing and the role of test coverage when developing with the platform

10 Principles of Apex Testing

Like it or not, testing your Apex code is a requirement for deployment. Often it seems developers are frustrated with testing because it seems like a platform imposed chore, rather than a useful software development tool. Apex developers can harness testing as a way of improving software quality.
Tags: Beginner, Core

Unit Test Scoping

There are several pitfalls that developers can fall into when writing unit tests, and traditionally speaking one of the easiest to fall into can be coding with a reliance on existing data.
Tags: Intermediate, Core

Testing Apex Callouts using HttpCalloutMock

The release added a couple of new features for testing Apex callouts.
Tags: Intermediate, Core

Testing HTTP Callouts with Static Data

One of the most eagerly awaited features, at least for developers, has been the ability to test Apex callouts.
Tags: Intermediate, Core

1.13 Describe an approach to error/exception handling.

Force.com Apex Code Developer's Guide: Exceptions in Apex

Exceptions note errors and other events that disrupt the normal flow of code execution. Throw statements are used to generate exceptions, while try, catch, and finally statements are used to gracefully recover from exceptions.
Tags: Intermediate, Core

See “Suggested Activities,” later in this document.

2. Performance and Scalability

In this section, you will find the resources that cover architectural considerations of implementing programmatic solutions in an enterprise environment.

2.1 Describe how to troubleshoot performance and scalability problems with Visualforce pages and Apex.

Developer Console

The Developer Console is an integrated development environment with a collection of tools you can use to create, debug, and test applications in your Salesforce organization.
Tags: Beginner, Core

Exceptions, Debugging, and Testing

Ensure that your code runs smoothly by learning about the Apex error handling framework, and learn how to expand that framework with custom exceptions. Learn about the various tools available to you for debugging code as well as creating the required, environment-independent unit tests that must be created in order to deploy your code.
Tags: Beginner, Core

Visualforce Performance: Best Practices

This document contains best practices for optimizing the performance of Visualforce pages. Its contents are a subset of the best practices in the Visualforce Developer's Guide.
Tags: Beginner, Recommended

Force.com Performance Profiling Using the Developer Console

Using the Developer Console, you can use “performance profiling” to identify and fix performance hotspots, and ensure that your applications are both fast and scalable.
Tags: Intermediate, Core

What is a Static Resource?

Static resources allow you to upload content that you can reference in a Visualforce page, including archives (such as .zip and .jar files), images, stylesheets, JavaScript, and other files.
Tags: Intermediate, Recommended

A Guide to Application Performance Profiling in Force.com

In this guide, you can learn about both the Force.com performance profiling tools and their associated methodologies, which have been informed by recommendations from salesforce.com Customer Support and our own engineering teams.
Tags: Intermediate, Recommended

Apex Performance Troubleshooting

This article discusses how to work with Apex debug logs, including understanding log time gaps and how overuse of System.Debug can affect logs. Additionally, the article talks about an Apex Timeline tool that uses the Tooling API to programmatically access debug logs.
Tags: Intermediate, Recommended

2.2 Given a scenario, describe how to optimize Apex and Visualforce performance to include LDV scenarios.

Quick Look into JavaScript Remoting

So what is JavaScript Remoting? Essentially, the feature allows for quick, tight integration between Apex and JavaScript. If you had used the old AJAX toolkit in the past, it's similar in some ways to how accessing an Apex Web Service worked – but is far more lightweight and simpler to get up and running.
Tags: Beginner, Core

Paginating Data for Force.com Applications

This article explores several pagination strategies to demystify when you should use each tool to achieve peak application performance.
Tags: Intermediate, Core

Developing Selective Force.com Queries through the Query Resource Feedback Parameter Pilot

After reading the article, you should be able to understand what a selective filter is, test your query's selectivity without running that query, and determine whether your query contains a selective filter.
Tags: Intermediate, Core

Force.com SOQL Best Practice: Sort Optimization

Read on to learn more about sort optimization, a simple technique that many developers and architects overlook when applying SOQL performance tuning best practices.
Tags: Intermediate, Core

Dealing with Exception Filters in Force.com

You might be able to overcome your performance issue with indexed formula fields, which this blog post explains in detail.
Tags: Intermediate, Core

Overview of Single View State

As HTTP is a stateless format, the viewstate is a section of data, represented as a long hashed string, on your Visualforce pages, which allows Visualforce to maintain the state between the client and the server.
Tags: Intermediate, Recommended

Force.com SOQL Performance Tips: LastModifiedDate vs SystemModStamp

To help your application scale and perform with large data volumes, it is important to understand the performance implications when choosing one field over the other as you use them in your SOQL filters.
Tags: Intermediate, Recommended

2.3 Describe how Apex and Visualforce can work with web technologies and complementary UI technologies.

[Hands On with the New Native JSON Parser]

One of the most eagerly awaited features is a native JSON implementation.
Tags: Intermediate, Recommended

[Visualforce Developer's Guide: Using JavaScript Libraries with Visualforce]

You can include JavaScript libraries in your Visualforce pages to take advantage of functionality provided by these libraries.
Tags: Intermediate, Recommended

See "Suggested Activities," later in this document.

3. Maintainability and Reuse

In this section, you will find resources that will help you create solutions that are easy to maintain and can be reused for optimal performance.

3.1 Given a set of business requirements that include multi-language support, determine the appropriate solution to use to support the globalization.

Deploying Salesforce Globally

This slideshow provides an introduction to some of the key areas to consider when going global, and provides guidance around some key “gotchas.”
Tags: Beginner, Core

Custom Labels

Custom labels enable developers to create multilingual applications by automatically presenting information (for example, help text or error messages) in a user's native language.
Tags: Beginner, Core

3.2 Describe the options and techniques to make programmatic components maintainable and reusable on the platform.

Force.com Apex Code Developer's Guide: Custom Settings Methods

Custom settings are similar to custom objects and enable application developers to create custom sets of data, as well as create and associate custom data for an organization, profile, or specific user.
Tags: Beginner, Core

Visualforce Developer's Guide: Defining Templates with <apex:composition>

The following example shows how you can use <apex:composition>, <apex:insert>, and <apex:define> to implement a skeleton template.
Tags: Beginner, Recommended

Introducing custom metadata types: the app configuration engine for Force.com

Developing app configurations today can take one or two engineering-years and put a 20% tax on ongoing development. With custom metadata types, the upfront work can be reduced to weeks and the updates to configuration tools to hours.
Tags: Intermediate, Recommended

Alert! Salesforce Event Notification Designs for Force.com Apps

The requirements for different event notification use cases can vary in many ways and according to many factors, so you should carefully consider your event notification requirements during the design phase of your application.
Tags: Advanced, Recommended

See "Suggested Activities," later in this document.

Suggested Activities

To practice these activities, you may do one of the following:

- Request a free Practice Org by creating a case here.
  - Question Type: Architect Support
  - Question Detail: Request Practice Org

  You should receive login information in about two business days.
- Use your existing Developer org.
- Sign up for a free Developer Edition account here.

1. Using the Developer Console to Execute Apex Code

The Developer Console can look overwhelming, but it's just a collection of tools that help you work with code. In this lesson, you'll execute Apex code and view the results in the Log Inspector. The Log Inspector is a useful tool you'll use often.

3. Click **Debug** > **Open Execute Anonymous Window** or CTRL+E.
4. In the Enter Apex Code window, enter the following text:

   System.debug('Hello World');

5. Deselect **Open Log** and then click **Execute**.

   Every time you execute code, a log is created and listed in the *Logs* panel.

   Double-click a log to open it in the Log Inspector. You can open multiple logs at a time to compare results.

   Log Inspector is a context-sensitive execution viewer that shows the source of an operation, what triggered the operation, and what occurred afterward. Use this tool to inspect debug logs that include database events, Apex processing, workflow, and validation logic.

   The Log Inspector includes predefined perspectives for specific uses. Click **Debug** > **Switch Perspective** to select a different view, or click CTRL+P to select individual panels. You'll probably use the Execution Log panel the most. It displays the stream of events that occur when code executes. Even a single statement generates a lot of events. The Log Inspector captures many event types: method entry and exit, database and web service interactions, and resource limits. The event type USER_DEBUG indicates the execution of a System.debug() statement.

6. Click **Debug** > **Open Execute Anonymous Window** or CTRL+E and enter the following code:

   ```
   System.debug( 'Hello World' );System.debug( System.now() );System.debug( System.now() + 10 );
   ```

7. Select **Open Log** and click **Execute**.
8. In the Execution Log panel, select **Executable**. This limits the display to only those items that represent executed statements. For example, it filters out the cumulative limits.
9. To filter the list to show only USER_DEBUG events, select Debug Only or enter *USER* in the Filter field.

NOTE: The filter text is case sensitive.

Congratulations–you have successfully executed code on the Force.com platform and viewed the results!

2. Creating and listing Visualforce Pages

In this tutorial, you'll learn how to create and edit your first Visualforce page. The page will be really simple, but this is the start, and we'll soon expand on it. Along the way you'll familiarize yourself with the editor and automatic page creation.

Enable Visualforce Development Mode

Development mode embeds a Visualforce page editor in your browser that allows you to see code and preview the page at the same time. Development mode also adds an Apex editor for editing controllers and extensions.

1 At the top of any Salesforce page, click the down arrow next to your name. From the menu under your name, select Setup or My Settings–whichever one appears.
2 From the left panel, select one of the following:
   - If you clicked Setup, select **My personal information** > **Personal Information**.
   - If you clicked **My Settings**, select **Personal** > **Advanced User Details**.
3 Click **Edit**.
4 Select the Development Mode checkbox, then click **Save**.

Create a Visualforce Page

1. In your browser, add */apex/hello* to the URL for your Salesforce instance. For example, if your Salesforce instance is:

   https://na1.salesforce.com, the new URL is *https://na1.salesforce.com/apex/hello*. You will see the following error:

2. Click the **Create Page hello** link to create the new page. You will see your new page with some default markup.

NOTE: If you don't see the Page Editor below the page, just click the **hello** tab in the status bar.

That's it! The page includes some default text, as well as an embedded page editor displaying the source code. This is the primary way you'll be creating pages in this section.

Build Materials

1. @future and Apex

1.8 Describe when to apply the appropriate Apex functionality, such as custom settings, when to use synchronous vs. asynchronous patterns, and all the available execution contexts (e.g., batch, trigger, callout, etc.) when Apex should be used on the platform and articulate appropriate scenarios for each item.

Use Case

Universal Containers has a group of users that manage cases regarding printer hardware issues. There is a lot of employee turnover in this group of users, so the administrators would like to automate the ability to add and remove users from this queue when users are created or deactivated.

Detailed Requirements

1. When a new user is created with a Department = 'Printers,' add the user to the 'Printers Queue.'
2. When a user is de-activated and is part of the 'Printers Queue,' remove them from the queue.
3. When a user is updated with a Department = 'Printers,' add them to the 'Printers Queue.'
4. When a user is re-activated with their Department = 'Printers,' add them to the 'Printers Queue.'
5. When a user's department changes from Printers to something else and they are part of the 'Printers Queue,' remove them from the queue.

Prerequisite Setup Steps

1. Case Queue named **Printers Queue**.
2. Assign one to two users to the **Printers Queue** with Department field = **Printers**.

Considerations

- How many triggers are needed?
- How much test coverage is required?
- What kinds of test methods should be created?
- How do you properly structure test classes to test @future methods?
- Determine whether @future is needed or not?
- Governor limits: What should you be concerned about?
- How do you want to handle any errors with executing the code?
- Should before or after update/insert triggers be used?

- What values are available with the Trigger.new and Trigger.old objects? When do you use each?
- When should you use with sharing versus without sharing with classes?

Solution

Best Solution Overview

Considerations Solutions:

- How many triggers are needed?
  - Ensure only one trigger per object to avoid sequencing issues.
- How much test coverage is required?
  - Triggers need a minimum of one line of test coverage. Recommend 100%.
  - Ensure 75% overall code coverage for the org. Recommend 100%.
- What kinds of test methods should be created?
  - Ensure positive, negative, single, and bulk record test methods are created.
- How do you properly structure test classes to test @future methods?
  Ensure test.startTest() & test.stopTest() are used to force future methods to fire before doing asserts on the results in the test class methods.
- Determine whether @future is needed or not?
  - Should the user wait for the code to finish? Or should it be run behind the scenes?
  - Does the user need to know what happened with this code?
- Governor limits: What should you be concerned about?
  - Ensure code is bulkified, especially important for triggers.
  - Number of @future calls in a single transaction.
- How do you want to handle any errors with executing the code?
  - Should the user receive notifications?
  - Should all records fail if one record in the batch fails?
- Should before or after update/insert triggers be used?
  - Before triggers to add to changes on the record before it is saved to the database.
  - After triggers to perform actions after a linkage or record creation/update, or access system-set fields.
  - Records in the after triggers are read-only.
- What values are available with the Trigger.new and Trigger.old objects? When do you use each?
  - Trigger.new and newMap has the new versions of the objects – available in insert and update and can only be modified in before triggers.
  - Trigger.old and oldMap has the previous versions of the objects – available in update and delete and cannot be modified.
  - Other context variables: Trigger.isExecuting, isInsert, isUpdate, isDelete, isBefore, isAfter, isUndelete, size - used to determine context of the trigger code that is being executed.
- When should you use with sharing versus without sharing with classes?
  - Always use sharing; very few instances require without sharing (and with sharing is the default).
  - Without sharing is only used when the code must access/work with data that is not normally visible for the user executing the code.

Solution Description:

1. Create an Apex trigger for after update and after insert on the user object.
   - Determine if user record matches scenario from detailed requirements or not and add to an update or remove list.
   - Pass the list to @future methods to perform the update or removal in bulk, if not empty.

2. Create an Apex class with two @future methods.
   - One method to add the users to the ‘Printers Queue’ queue.
   - One method to remove the users from the ‘Printers Queue’ queue.

Apex Class

```
1   public class CaseSync {
2
3       @future
4       public static void addMembers(List<Id> memberIds) {
5
6           List<GroupMember> ToInsertUserList = new List<GroupMember>();
7           Group groupId = [SELECT Id FROM Group WHERE Name = 'Printers Queue' LIMIT 1];
8
9           // for each user
10          for (Id u : memberIds) {
11              // user is new, add to queue
12              GroupMember gma = new GroupMember( GroupId = groupId.Id, UserOrGroupId = u );
13              ToInsertUserList.add(gma);
14          }
15
16          insert ToInsertUserList;
17      }
18
19      @future
20      public static void removeMembers(List<Id> memberIds) {
21
22          List<GroupMember> ToDeleteUserList = new List<GroupMember>();
23          Group groupId = [SELECT Id FROM Group WHERE Name = 'Printers Queue' LIMIT 1];
24
25          ToDeleteUserList = [SELECT GroupId, UserOrGroupId FROM GroupMember WHERE Group.Type =
    'Queue' AND GroupId = :groupId.Id AND UserOrGroupId in :memberIds];
26
27          delete ToDeleteUserList;
28      }
29
30  }
31
32  trigger CaseQueueSync on User (after update, after insert) {
33      // Create an empty list of IDs
34      List<Id> addUserIds = new List<Id>();
35      List<Id> removeUserIds = new List<Id>();
36
37      if (Trigger.isInsert) {
38          for (User u : Trigger.new) {
39              if (u.isActive == true && u.department == ‘Printers') {
40                  // will be an add, not a delete if either field was updated
41                  addUserIds.add(u.Id);
42              }
43          }
```

```
    } else {

        // Create a map of IDs to all of the *old* versions of Users
        // updated by the call that fires the trigger.
        Map<ID, User> oldMap = new Map<ID, User>(Trigger.old);

        // Iterate through all of the *new* versions of User
        // sObjects updated by the call that fires the trigger, adding
        // corresponding IDs to the two userIds lists, but *only* when an
        // sObject's status changed from active to not active, vice-versa, or profile changes
to/from Standard User.
        for (User u : Trigger.new) {
            if (u.isActive == true && u.department == 'Printers') {
                // will be an add, not a delete if either field was updated
                if ((oldMap.get(u.Id).isActive != true) || (oldMap.get(u.Id).department !=
'Printers')) {
                    addUserIds.add(u.Id);
                } // else these two fields didn't change, so don't do anything
            } else if (u.isActive == false || u.department != 'Printers') {
                // will be a delete, not an add if either field was updated
                if ((oldMap.get(u.Id).isActive != false) || (oldMap.get(u.Id).department ==
'Printers')) {
                    removeUserIds.add(u.Id);
                } // else these two fields didn't change, so don't do anything
            }

        }
    }
    // If the list of IDs is not empty, call CaseSync.addMembers & CaseSync.removeMembers
    // supplying the list of IDs for processing.
    if (addUserIds.size() > 0) {
        CaseSync.addMembers(addUserIds);
    }
    if (removeUserIds.size() > 0) {
        CaseSync.removeMembers(removeUserIds);
    }
}
```

2. Batch Apex and Apex Callouts

2. Batch Apex and Apex Callouts

1.9 Given a scenario, describe how batch apex and the flex queue could be applied.

Use Case

Universal Containers has a Sales Cloud implementation. The sales team executes a routine process to close out Opportunities during the course of the day. At the end of the day, all closed-out Opportunities are required to be batched and sent to a backend service application for calculating the commission on the Opportunity for each of the Opportunity records. The commission amount is updated on each of the Opportunity records in response.

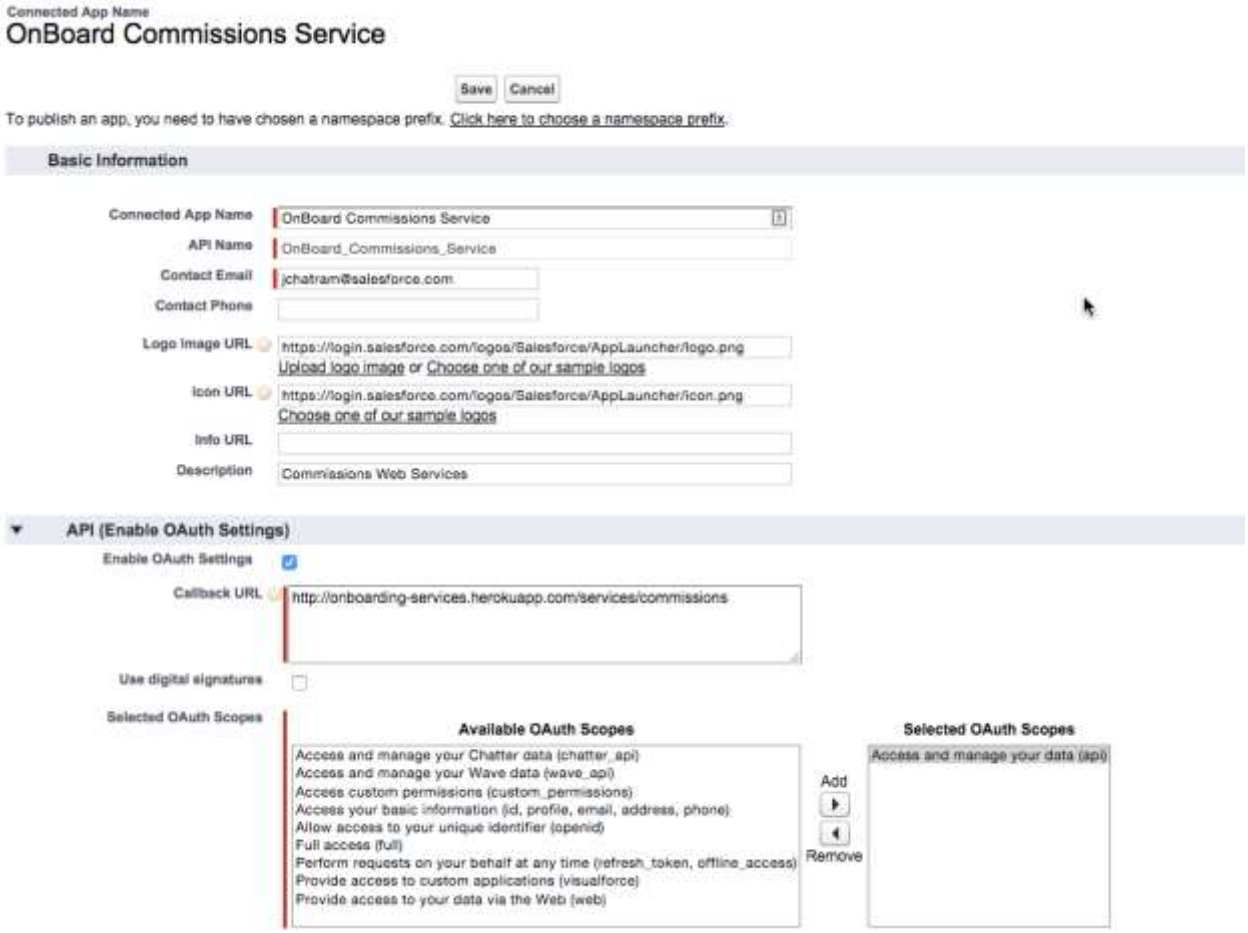
Detailed Requirements

The Universal Containers sales team has given a set of requirements for their development team to set up a recurring job to process all closed Opportunities for the day. The requirements:

1. To query all closed Opportunities records.
2. For Opportunities in "closed-won" status, group records in a batch and send the batch of Opportunities to a Commission Web Service:
   - Endpoint: http://onboarding-services.herokuapp.com/services/commissions
3. Responses received via web services call should update the commission amount for each of the opportunity records in scope.

Prerequisite Setup Steps

1. Create a new custom field of type "Currency" called "Commissions Amount" on the Opportunity object. Add this field to an appropriate Page layout.
2. Create a new custom field of type "Text" called "Opportunity Number" on the Opportunity object. Add this field to an appropriate Page layout.
3. Verify http://onboarding-services.herokuapp.com/services/ - services URL is available and active in a browser.



Considerations

1. Test class to have 75% or greater code coverage for this scenario.
2. Verify Opportunities in the processed batch were successfully updated with Commissions Amount.
3. How do you address limits considerations? For schedulable batch apex?
4. How do you chunk the response size?
5. Implications of writing a test class?
6. Do you need to use a database that allows callouts?
7. How are you grouping the records for callouts?

Solution

Best Solution Overview

Log in to your Developer Org where this service setup is required.

1. Download and save the WSDL from this URL below:

http://onboarding-services.herokuapp.com/services/commissions?wsdl

2. Create a "Connected App" in the Org for the endpoint web services URL.

- Log in to your Developer Org where this service is required to be setup.
- Navigate to **"Your Name" | Setup | App Setup | Create | Apps.**
  - Under Connected Apps, click **New**.
  - Create new Connected App as shown in the example screen shot below by providing following information:
  - Click **Save**, to complete to create the Connected App:
    - Basic Information
    - API (Enable OAuth Settings)

3. Generate an Apex class from the WSDL.

- Navigate to **"Your Name" | Setup | App Setup | Develop | Apex Classes**.
- Click **Generate from WSDL**. Choose the downloaded WSDL file.
- Follow along with steps to generate a web service stub class. Provide a meaningful name for the service Apex class, for example, "Commissions Service."

4. Create a Batch Apex class.

- Navigate to **"Your Name" | Setup | App Setup | Develop | Apex Classes**.
- Give the Apex Class a name, "BatchUpdateCommissions."

5. Create an Apex class that implements schedulable.

- Navigate to **"Your Name" | Setup | App Setup | Develop | Apex Classes**.
- Give the Apex Class a name, "ExecuteCommissionsBatch."

6. Create test Apex class to:

- Execute the batch and set a mock web service call.
- Execute the commissions update with the mock service callout.

Sample Code

1. Generated Web Services Stub Class

```
1 //Generated by wsdl2apex
2
3 public class CommissionsService {
4     public class getCommissionsResponse {
5         public Double return_x;
6         private String[] return_x_type_info = new
    String[]{'return','http://onboarding.salesforce.com/',null,'0','1','false'};
7         private String[] apex_schema_type_info = new
    String[]{'http://onboarding.salesforce.com/','false','false'};
8         private String[] field_order_type_info = new String[]{'return_x'};
9     }
10    public class getCommissions {
11        public String arg0;
12        public String arg1;
13        private String[] arg0_type_info = new
    String[]{'arg0','http://onboarding.salesforce.com/',null,'0','1','false'};
14        private String[] arg1_type_info = new
    String[]{'arg1','http://onboarding.salesforce.com/',null,'0','1','false'};
15        private String[] apex_schema_type_info = new
    String[]{'http://onboarding.salesforce.com/','false','false'};
16        private String[] field_order_type_info = new String[]{'arg0','arg1'};
17    }
18    public class GetCommisionsPort {
19        public String endpoint_x = 'http://onboarding-
    services.herokuapp.com/services/commissions';
20        public Map<String,String> inputHttpHeaders_x;
21        public Map<String,String> outputHttpHeaders_x;
22        public String clientCertName_x;
23        public String clientCert_x;
24        public String clientCertPasswd_x;
25        public Integer timeout_x;
26        private String[] ns_map_type_info = new String[]{'http://onboarding.salesforce.com/',
    'Onboard_OppsCommission'};
27        public Double getCommissions(String arg0,String arg1) {
28            CommissionsService.getCommissions request_x = new
    CommissionsService.getCommissions();
29            request_x.arg0 = arg0;
30            request_x.arg1 = arg1;
31            CommissionsService.getCommissionsResponse response_x;
32            Map<String, CommissionsService.getCommissionsResponse> response_map_x = new
    Map<String, CommissionsService.getCommissionsResponse>();
33            response_map_x.put('response_x', response_x);
34            WebServiceCallout.invoke(
35              this,
36              request_x,
37              response_map_x,
38              new String[]{endpoint_x,
39              '',
40              'http://onboarding.salesforce.com/',
41              'getCommissions',
42              'http://onboarding.salesforce.com/',
43              'getCommissionsResponse',
44              CommissionsService.getCommissionsResponse'}
45            );
46            response_x = response_map_x.get('response_x');
47            return response_x.return_x;
48        }
49    }
50}
```

2. Batch Class

```
1 global class BatchUpdateCommissions implements Database.Batchable<sObject>,
    Database.AllowsCallouts {
2
3 global final String AsOfTime;
4
5 global BatchUpdateCommissions(String AsOf) {
6 // AsOf should be something like:
7 // LAST_WEEK, THIS_WEEK, NEXT_WEEK, LAST_MONTH, TODAY, LAST_N_DAYS:N, etc
8 this.AsOfTime = AsOf;
9 }
10
11    global BatchUpdateCommissions() {
12        this.AsOfTime = null;
13    }
14
15// start method should find all Closed Opportunities that are Closed-Won
16global Database.QueryLocator start(Database.BatchableContext BC){
17//String lastfewdays = this.AsOfDate.format('YYYY-MM-DDThh:mm:ssZ');
18        //System.debug(lastfewdays);
19String query = 'SELECT Id, Account.AccountNumber, Opportunity_Number__c, StageName,
    Commissions_Amount__c FROM Opportunity WHERE IsClosed = true';
20        if (AsOfTime != null) {
21            query += ' AND CreatedDate '+AsOfTime;
22        }
23        System.debug(query);
24return Database.getQueryLocator(query);
25    }
26
27// execute method should call the CommissionsService to get & update the commission amount
    field
28    global void execute(Database.BatchableContext BC, List<sObject> scope){
29        System.debug(scope);
30    List<Opportunity> opplist = new List<Opportunity>();
31 // create stub
32CommissionsService.GetCommisionsPort cs = new CommissionsService.GetCommisionsPort();
33
34        for(Sobject s : scope){
35        Opportunity o = (Opportunity)s;
36        if (o.StageName == 'Closed Won') {
37        Double value = cs.getCommissions(o.Opportunity_Number__c, o.Account.AccountNumber);
38        o.Commissions_Amount__c = value;
39        opplist.add(o);
40        } // else do nothing!  or should I remove it from scope?
41        }
42        update opplist;
43    }
44
45    global void finish(Database.BatchableContext BC){
46// do nothing
47System.debug(BC);
48    }
49
50}
```

3. A Scheduable Apex Class

```
a.    global class ExecuteCommissionsBatch implements Schedulable {
b.
c.        public void execute(SchedulableContext sc) {
d.            Database.executeBatch(new BatchUpdateCommissions(), 1);
e.        }
f.
g.    }
```

4. Apex Test Class

```
1   @isTest public with sharing class BatchUpdateCommissions_Test {
2   // Execute the commissions update with mock service callout
3   @isTest public static void testbatch() {
4
5           BatchUpdateCommissions buc = new BatchUpdateCommissions();
6           createOpty();
7           Test.startTest();
8           CommissionsService.mock = new MockCommissionsCall();
9
10          Database.executeBatch(buc, 1);
11
12          // verify not set yet
13          List<Opportunity> olist = [SELECT Id, Name, Commissions_Amount__c FROM
    Opportunity];
14          System.assertEquals(olist[0].Commissions_Amount__c, null);
15          Test.stopTest();
16
17          olist = [SELECT Id, Name, Commissions_Amount__c FROM Opportunity];
18
19          // should be only one opportunity that we can see, so check that the commissions
    amount matches our mock call
20          System.assertEquals(olist[0].Commissions_Amount__c, 3245.4455);
21  }
22
23  // CRON expression: sample set to midnight on March 15. Set the time as appropriate for
    test run
24     // Because this is a test, job executes
25     // immediately after Test.stopTest().
26     public static String CRON_EXP = '0 0 0 15 3 ? 2022';
27
28  @isTest public static void createOpty() {
29  Account a = new Account(Name='Test Acct', AccountNumber='3456');
30          insert a;
31          Opportunity o = new Opportunity(Name='Test Opty 1', StageName='Closed Won',
    AccountId = a.Id, CloseDate=Date.today());
32          insert o;
33  }
34
35  // Execute batch and set up a mock web service call
36  @isTest public static void test() {
37       createOpty();
38
39       ExecuteCommissionsBatch ex = new ExecuteCommissionsBatch();
40        Test.startTest();
41    Test.setMock(WebServiceMock.class, new MockCommissionsCall());
42       // Schedule the test job
43       String jobId = System.schedule('ScheduleApexClassTest',
44                          CRON_EXP,
45                          ex);
46
47       // Get the information from the CronTrigger API object
48       CronTrigger ct = [SELECT Id, CronExpression, TimesTriggered,
49          NextFireTime
50          FROM CronTrigger WHERE id = :jobId];
51
52       // Verify the expressions are the same
53       System.assertEquals(CRON_EXP,
54          ct.CronExpression);
55
56       // Verify the job has not run
57       System.assertEquals(0, ct.TimesTriggered);
58
59       // Verify the next time the job will run
```

```
60        System.assertEquals('2022-03-15 00:00:00',
61           String.valueOf(ct.NextFireTime));
62        // Verify the scheduled job hasn't run yet.
63        Opportunity[] olist = [SELECT Id, Commissions_Amount__c FROM Opportunity WHERE
    Commissions_Amount__c = NULL];
64        System.assertEquals(olist.size(),1);
65        Test.stopTest();
66
67        // Now that the scheduled job has executed after Test.stopTest(),
68        //   fetch the new updates (which won't happen since the schedule won't run the
    batch process).
69        olist = [SELECT Id, Commissions_Amount__c FROM Opportunity WHERE
    Commissions_Amount__c = NULL];
70        System.assertEquals(olist.size(), 1);
71
72    }
73 }
```

Mock Service Class

```
1   @isTest global class MockCommissionsCall implements WebServiceMock {
2       global void doInvoke(
3               Object stub,
4               Object request,
5               Map<String, Object> response,
6               String endpoint,
7               String soapAction,
8               String requestName,
9               String responseNS,
10              String responseName,
11              String responseType) {
12
13          // Create response element from the autogenerated class.
14          CommissionsService.getCommissionsResponse responseElement = new
    CommissionsService.getCommissionsResponse();
15          // Populate response element.
16          responseElement.return_x = 3245.4455;
17          // Add response element to the response parameter, as follows:
18          response.put('response_x', responseElement);
19      }
20  }
```

3. New Invoice Button

3. New Invoice Button

1.7 Describe types, considerations, and tradeoffs when designing Apex controllers.

Use Case

Universal Containers uses Sales Cloud. They use Opportunities to track their sales orders. Once an Opportunity is closed, they can create multiple invoices from the Sales Cloud application from the Opportunity. The typical process for invoice creation is:

1. Search for the Opportunity.
2. Click **Add new Invoice** from the Invoice related list under Opportunity.
3. When they are in the new Invoice screen, they would like to default the following fields on the Invoice from the Account and Opportunity:
    a. Account Billing Address (Street 1, 2, City, State and Zip code).
    b. A comma-separated list of Product codes that are associated with the Opportunity.
    c. Default Discount code that is a field in the account.
4. The sales rep should have the option to change the values in the invoice once they are defaulted. If the rep clicks **Cancel**, the rep should be taken back to the Opportunity that they navigated from.

Detailed Requirements:

- Search for the Opportunity: either global search or select an **Opportunity** from Opportunity List views.
- User clicks on the **New Invoice** button from a related list on the Opportunity page.
- Populate details from the Opportunity into the Invoices.
- Let users edit the values in the invoice edit page.
- User clicks **Save**, invoice is inserted into Salesforce and returned back to the Opportunity read page.
- When users click on the **Cancel** button, they are redirected to the Opportunity read page.

Considerations:

- How many controllers and controller extensions are required?
- How much test coverage is required?
- What kinds of test methods should be written?
- When should you use with sharing and without sharing classes?

- Governor limits: what should you be concerned about?
- How do you want to handle errors with executing the code?

Solution

Best Solution Overview

1. How many controllers and controller extensions are needed?
   a. One controller extension is required. If needed, Helper classes can be created.

2. How much test coverage is required?
   a. A minimum of 75% test coverage is required.

3. What kind of test methods should be written?
   a. Ensure that positive, negative, single, and bulk record test methods are created.

4. When should you use with sharing and without sharing classes?
   a. Always use sharing; very few instances require without sharing (and with sharing is the default).
   b. Without sharing is only applicable when the code must access work with data that is not normally visible for the user executing the code.

5. Governor limits: what should you be concerned about?
   a. Ensure code is bulkified.
   b. No SOQL queries within loops.
   c. Keep viewstate to minimal.

6. How do you handle errors with executing the code?
   a. Should the user receive notifications?
   b. Should all records fail if one record in the batch fails?

Solution Description:

Create a custom invoice button for the invoice related list under Opportunity.
Create a VF page for the invoice and custom buttons for the Save and Cancel operation.
Navigate to **Setup | Develop | Visualforce Pages**.

```
<apex:page controller=”Invoice” extensions=”InvoiceExtension” showHeader="false" >
<apex:pageblock>
    <apex:commandButton value="Next" action="{!save}" id="savebtn" />
    <apex:commandButton value="Next" action="{!cancel}" id="savebtn" />
    <apex:pageblock>
            <apex:inputfield name=”Opportunity” value=”Opportunity” />
            <apex:inputfield name=”Street1” value=”Street1” />
            <apex:inputfield name=”Street2” value=”Street2” />
            <apex:inputfield name=”City” value=”City” />
            <apex:inputfield name=”State” value=”State” />
            <apex:inputfield name=”Zipcode” value=”Zipcode” />
            <apex:inputfield name=”productcodes” value=”ProductCodes” />
            <apex:inputfield name=”DiscountCode” value=”DiscountCode” />

    </apex:pageblock>
</apex:pageblock>
</apex:page>
```

1. Create a controller extension for the Invoice controller. Navigate to **Setup | Develop | Apex Classes.**

```
public class with Sharing InvoiceControllerExtension{
            private final Invoice inv;
            public String StreetName1{get;set;}
            public String StreetName2{get;set;}
            public String City{get;set;}
            public String State{get;set;}
public String Zipcode{get;set;}
public string opportunityProductCodes{get;set;}
            public InvoiceControllerExtension(ApexPages.StandardController
stdController){
this.inv = (Invoice)stdController.getRecord();
                        Opportunity opp = [select id, name, Account,
Account.BillingAddress from Opportunity where Id =: inv.Opportunity];
                        OpportunityProduct OppProducts = [select id, productcode from
OpportunityProduct where Opportunity =: inv.Opportunity];
                        inv.Street1 = opp.BillingAddress.Street1;
                        inv.Street2 = opp.BillingAddress.Street2;
                        inv.City = opp.BillingAddress.City;
                        inv.State = opp.BillingAddress.state;
                        inv.Zipcode = opp.BillingAddress.Zipcode;

                        for(OpportunityProduct oppproduct : OppProducts){
                                if (OpportunityProductCodes == null ||
OpportunityProductCodes.length == 0){
                                        OpportunityProductCodes =
OpportunityProduct.ProductCode;
                                }else{
                                        OpportunityProductCodes += “,” +
OpportunityProduct.ProductCode;
                                }
                        }
                        inv.OpportunityProductCodes = OpportunityProductCodes;
            }

            public PageReference Save(){
```

```
30                          //Insert code to customize the Save logic as required
31                          return Pagereference;
32                  }
33
34                  public PageReference Cancel(){
35                          //Insert code to customize the Cancel logic as required
36                          return Pagereference;
37                  }
38      }
```

1. Get the Opportunity related information and assign it to the variables in the controller extension. Please refer to the above in which Opportunity, Account and OpportunityProduct information are retrieved from the database and stored in the variables via a constructor.
2. Keep the fields in the Invoice VF page editable and map the values to the variables in the controller extension. Please refer to the VF page above where all fields are listed as input fields that will let the user edit the values populated through the controller.
3. Override the Save() and Cancel() methods for the custom logic and URL redirection.

Request a Practice Org

To request a Practice Org that contains information from some of the Build Materials, please click here to open a case.

Select Question Type: Architect Support

Question Detail: Request Practice Org

ALERT: If you are not active within your practice org for 6 months, it may be deactivated.

Join the Salesforce Architect Success Group

- Want to make sure you don't miss any content release updates or news regarding the Salesforce Architect Journey?
- Looking to connect with others that have the same interest?

Click here and request to join the Salesforce Architect Success Group.