Name : Tushar Srivastav
College Name : Graphic Era Deemed To Be University

In this document I have explained all the **Modules/Badges** along with the **Super-Badges**.

Module Name- Trailhead and Trailblazer Community

Here we have four unit inside this module so let's understand in short .

Unit-1 Get Started with Trailhead and Trailblazer Community

here we understand what is Trailhead and for whom it is for and also what is Trailblazer Community and for whom it is for and also create identity and answer some quiz question.

Unit-2 Find your way around

In this unit we understand the difference between a module and project, the difference between a topic and a group, how you earn points, badges, and ranks. Navigate Trailhead and the Trailblazer Community. Then we give Quiz

Unit-3 Build Your Network with Trailblazer Community

Here in this unit we Join groups that interest us and Find events near us or online Follow other Trailblazers and Find inspiration with the #MyTrailblazerStory topic and in finally attend the Quiz

Unit-4 Troubleshoot and Solve Problems Together

Here we understand where to go if we need help with Trailhead and Trailblazer Community.

Have overlook over Troubleshoot common issues along with quiz.

Module-Salesforce Platform Basics

Here we 5 unit lets see each of them.

Unit-1 Get Started with the Salesforce Platform

Here we understand how to define salesforce platform and have a look over Dream House scenario and create a trailhead account and understand the difference between no code (declarative) and programmatic development.

Here we a challenge lab we need to create object with given details.

Unit-2 Discover Use Cases for the Platform

Here we study the sample use case for the platform and have a look over the reasons for using the

platform across the multiple departments and had simple quiz.

Unit-3 Understand the Salesforce Architecture

Here we have the salesforce architecture and key terms related to the salesforce architecture, then we find the information related to the trust in cloud and have look over the salesforce API and one use case for the Salesforce API and finally one quiz.

Unit-4 Navigate Setup

Here we learn about setup i.e. how to locate setup and identify the key elements , also identify the important menus for customizing the org and how to use Quick find to access the menu items And then a quiz to the test the knowledge.

Unit-5 Power Up with AppExchange

Here we understand what is AppExchange and how to install the App and develop the app exchange with different strategy and then quiz.

Module- Platform Development Basics

here we have tools and technologies that power development on the salesforce platform. Here we explore 4 units that are-

Unit-1 Get Started with Platform Development

Here we study the salesforce platform along the with platform building blocks and saw a scenario i.e. the dream house and install dream house app package. here we perform the challenge lab in which we install the Dream House app for all users, and grant access to third party applications then from the App Launcher, open the Dream House app, On the Data Import tab, we import the data and then we click Initialize Sample Data and complete the task.

Unit-2 Develop Without Code

Here see the metadata, the power of the meta data and also the benefits of the metadata-driven development model. See what is inside the dream house app , then we see what is no-code and low code development approaches and examples of no-code and low-code development approaches . Then we have quiz at last.

Unit-3 Code with Salesforce Languages

Here we see three core programmatic technologies i.e. Lightning Component Framework, Apex, Visualforce, then we identify the benefits of lightning components and how visual force is used in Lightning experience and see how the apex used to support lightning components and visual force. Then we quiz in last.

Unit-4 Extend the Salesforce Platform

Here we know the lightning platform API’s and we can do with those API’s then we how the heroku and salesforce are related and study some topics like IoT, Bots and many more and identify ways the how

salesforce interacts with Iot and Bots and then we have quiz at last.

Module - Customize a Salesforce Object

Here we earn how to use picklists, filters, formula and other tools to customize an object in your org. Here we explore total 8 units.

Unit-1 Work with Standard and Custom Fields

Here we Create new custom fields to meet business requirements and facilitate accurate data entry with formulas, picklists, and lookups. We also Ensure that users have access to the right fields with page layouts. Manage multiple business scenarios with record types and business processes. Maintain data quality with history tracking and data validation. In this we have a challenge lab where we customize profiles and objects.

Unit-2 Create Picklists and Field Dependencies

Here in this unit we create a picklist datatype for region and then we create a dependency between the between the picklists and it was the challenge lab for this unit and we were guide by Meghan Butler.

Unit-3 Create Lookup Filters

Here in this unit we create lookup filters. Lookup filters limit the records available in the lookup. A lookup filter can reference other fields on the same record (source); fields on the records of the lookup object (target); fields on the user's record, profile, and role; and fields on records directly related to the target object. it was the challenge lab for this unit and we were guide by Meghan Butler.

Unit-4 Create Formula Fields

Here in this unit we create a formula to calculate value and that formula was written in formula field data type here we create commission field with formula and it was the challenge lab for this unit and we were guided by Meghan Butler.

Unit-5 Create Record Types

Here we create simple records according to the given details in the unit and it was the challenge lab of this unit and we were guided by Meghan Butler.

Unit-6 Create Account Page Layout

Here in this unit we learn about the page layout. Page layouts control the layout and organization of buttons, fields, s-controls, Visualforce, custom links, and related lists on object record pages. Here we a create the page layout on account object according the details given in unit and it was the challenge lab and we were guided by Meghan Butler.

Unit-7 Enable Account Field History Tracking

Here in this unit we will enable account field history tracking to track the history of the particular field in

account object and to track the history.it was challenge in this lab and we were guided by Meghan Butler.

Unit-8 Create Validation Rules

Here we learn how to create to rule i.e. validation rule which validate the whether the value enter is validating the rules applied or not. Here in validation rule we create a formula in formula filed and it was the challenge in this lab and we were guided by Meghan Butler.

Module-Data Modeling

Here we learn about data structure along with objects, fields, relationships. Here we have 3 units so lets see each of them.

Unit-1 Understand Custom & standard object

Here we learn about the custom and standard objects, difference between standard and custom objects, perk using objects on the salesforce platform and types of custom field an object can have. Here we a challenge lab in which we a different custom object according the given details.

Unit-2 Create Object Relationships

Here we learn the different types of object relationship and their use case. we learn how to create and modify the lookup relationship and also we learn how to crate and modify the master-details relationship. A lookup relationship essentially links two objects together so that you can “look up” one object from the related items on another object. In master-detail relationships, one object is the master and another is the detail. The master object controls certain behaviors of the detail object, like who can view the detail’s data.in this we have a challenge lab were we create master-detail and look between objects.

Unit-3 Work with Schema Builder

Here we learn about the Schema Builder, Schema Builder is a dynamic environment provided by Salesforce for viewing and modifying all the objects and relationships in your organization. The tool simplifies the task of designing, implementing, and modifying your data model, or schema. It's also a helpful way for understanding complex Salesforce data models. The we see advantages of using schema builder for data modeling, Using Schema Builder to create a schema for a given object model, Using Schema Builder to add a custom object to your schema, using Schema Builder to add a custom field to your schema. Then we have challenge lab in which using schema to create a custom field for object.

Module- Picklist Administration

Here we learn how to choose the right picklist field for the job and how to manage picklist and how to share picklist values, here we have 3 units.

Unit-1 Get Started with Picklists

Here we learn how to when to add a picklist field and which type of picklist we need to use and how to create a custom picklist. Here in challenge lab we perform this tasks.

Unit-2 Manage Your Picklist Values

here in this unit we learn how to manage the values for the picklists. How to use the formulas to define the picklist values dynamically and understand what happen to existing data as you change picklist values. Here in challenge lab we use a formula to define a default picklist value.

Unit-3 Share Values with Global Value Sets

Here in this unit we learn how to create global value sets and how to manage the global values sets and also learn how to promote an existing filed value to a global value set. Here we have challenge lab were we Promote an existing picklist's values to a global value set, and assign it to another picklist.

Module- Duplicate Management

Here we learn how to resolve and prevent duplicate record to increase user confidence in data. Here we have to 2 units.

Unit-1 Improve Data Quality in Salesforce

Here we learn how the duplicate data is wasteful and how it affects the progress. here we learn how to make our company win my preventing the duplicate data. Here we learn Techniques and solution for resolving and preventing duplicate data. Then we a quiz in last.

Unit-2 Resolve and Prevent Duplicate Data in Salesforce

Here we learn about the rules i.e. matching rules and duplicates rules, difference matching rules and duplicate rules. We learn how to identify the features of the standard matching rules. Discover the options for customizing matching rules. How to create matching rules.

Module - Formulas and Validations

Here in this module we learn about the formulas and validation. Here we have 3 units.

Unit-1 Use Formula Fields

Here we learn how to create a simple formula and then how to create a custom formula field and how to use the formula editor. Also we understand why formula field important and useful and we outline one use case for formula field. In challenge lab we create a formula field by creating a custom filed.

Unit-2 Implement Roll-Up Summary Fields

Here we see what is Roll-up summary fields, a roll-up summary field calculates values from related records, such as those in a related list. Then we create a roll-up summary field and also learn how to apply filed-level security to roll-up summary field. In challenge lab we create a roll –up summary field.

Unit-3 Create Validation Rules

Here we see what is validation Rules, Validation rules verify that data entered by users in records meets the standards you specify before they can save it. A validation rule can contain a formula or expression that evaluates the data in one or more fields and returns a value of “True” or “False.” Then we the elements of the validation rules and in last we create a validation rule. Then we a challenge lab in ehcuh we create a validation rule.

Module- Build a Data Model for a Travel Approval App

Here we are going to build a data model for travel app. Here we are going build in 5 units.

Unit-1 Create a Travel Approval Lightning App

Here we have basic introduction and also create a new playground and launch a travel app from app launcher.

Unit-2 Create a Department Object

Here we perform very task that is we create department object according to the details provided.

Unit-3 Create a Travel Approval Object

Here we create a object for the travel approval object and perform the task according to the given details in unit.

Unit-4 Create an Expense ItemObject

Here we a final object for that is expense item object. this will help the app to provide estimated for the trip cost such airfare, hotel, rental car etc.

Unit-5 Import Data and Test the App

Here it is the last step were we import data for testing purpose. Here we import the data and after importing the data we test the app.

Module- Improve Data Quality for a Recruiting App

Here we have app name recruiting app where we need to work to improve the data quality by using the

validation rule and formulas. Here we 3 units.

Unit-1 Create Cross-Object Formulas

Here we will set-up formulas and validation rules in an existing Recruiting app so users keep their data consistent and also we Create roll-up summary fields to calculate the information users enter in the app.

Unit-2 Create Validation Rules

Here we create validation rules for the Recruiting app. Validation rules verify that the data a user enters in a record meets the standards you specify before the user can save the record. A validation rule can contain a formula or expression that evaluates the data in one or more fields and returns a value of “True” or “False.”

Unit-3 Create Formula and Roll-Up Summary Fields

Here we create a formula field that calculates the overall score from the Review object. And also we create a roll-up summary fields.

Module- Customize the User Interface for a Recruiting App

Here we are going to customize the user interface for a recruiting app by creating tabs, quick action, record types etc. here we have 7 units so let’s see each of them.

Unit-1 Create a Tab for the Review Object

Here we Create tabs, quick actions, record types, and customized page layouts on an existing app for a more streamlined user experience and also Enable Chatter for the objects in the app to allow recruiters to more easily share information about candidates.

Unit-2 Create an Object-Specific Quick Action

Here we in this unit we Create records that have automatic relationships to other records then Make updates to specific records. Also we Interact with records in different ways.

Unit-3 Customize the Review Page Layout

Here we customize the review page layout according to the details provided in the unit. A page layout determines the fields, sections, related lists, and buttons that appear when users view or edit a record. You can modify an object’s default page layout or create a new one**.**

Unit-4 Create a Custom Candidate Record Page

Here we create a candidate record page that is Custom candidate record page to store record of the candidate. here we create custom candidate record page according to the details provided.

Unit-5 Create Record Types on the Position Object

Here first we create a profile and then we create record types according to the details given in the unit.

Unit-6 Customize the Position Page Layout

Here we customize the position page layout. here we first we create field on the position object and work on that field and customize the position page layout according to the details provided in unit.

Unit-7 Enable Chatter on the Review Object

Here it is last step were we will enable chatter on the review object. Chatter is a powerful tool. It enables information sharing, collaboration, visibility, general and record-specific conversations, feed tracking for specific records or individuals, and the option to create public or private groups.

Module-Lightning App Builder

Here we are going to build custom pages for lightning experience and here we have 5 units, let's see each of them.

Unit-1 Get Started with the Lightning App Builder

Here we Understand how the Lightning App Builder can help to build responsive apps and custom pages for Lightning Experience and the mobile app and also Understand the layout of the Lightning App Builder user interface and also difference between a Lightning page and a Lightning component

Unit-2 Build a Custom Home Page for Lightning Experience

Here we create a custom home pages for lightning experience users and also assign different home page to different profiles and set the default home page.

Unit-3 Build a Custom Record Page for Lightning Experience and the Salesforce Mobile App

Here we create a customized object record page for Lightning Experience and the Salesforce mobile app. Then we Add visibility rules to a record page component. And Activate the custom record page for your users.

Unit-4 Build an App Home Lightning Page

Here we Add components to a Lightning page and Configure the properties of a Lightning page and a Lightning Component Then We Add actions to a Lightning page and also we Add a Lightning page to Lightning Experience and the mobile app.

Unit-5 Work with Custom Lightning Components

Here we have a last unit where we Install a custom Lightning component and Use mobile device to preview the app in the Salesforce mobile app.

Module-Data Management

Here we how to import and export the data. Here we have 2 units let's see them.

Unit-1 Import Data

Here we see and compare the different options for importing data into salesforce. And also list the steps involved in preparing and importing data from a sample .csv file using the Data Import Wizard.

Unit-2 Export Data

Here we see and compare the two methods of exporting data from Salesforce. And how to Export data manually using the Data Export Service. How to Set up automatic export of data on a weekly or monthly schedule**.**

Module- Leads & Opportunities for Lightning Experience

Here we learn how to power the sales process with leads and opportunities in salesforce. Here we have 4 units let's see them.

Unit-1 Create and Convert Leads as Potential Customers

Here we Understand how leads fit into the sales process. Then we Understand what happens when you convert a lead. Also learn how to Create and convert leads.

Unit-2 Work Your Opportunities

Here we learn how to use opportunities and how to Create an opportunity. Then we learn how to Add contact roles to an opportunity.

Unit-3 Sell as a Team and Split the Credit

Here we learn how to use opportunity teams and account teams. Then we learn how to Apply revenue splits and overlay splits to an opportunity.

Unit-4 Visualize Success with Path and Kanban

This is the last unit where we Update a record's stage or status using Path. Also Update records in the Kanban view. And Use filters and view charts in the Kanban view.

Module- Quick Start: Process Builder

Here we create a process and add process criteria and also add process action and at last we test the process and here we do all task without code. Here have 4 unit.

Unit-1 Create a New Process on the Account Object

Here we see what is process builder and also create new process on the account object. process Builder is a workflow tool that helps automate business processes without writing a single line of code.

Unit-2 Add Process Criteria

Here we learn how to create and add process criteria. Here we add process criteria according to the details

provided in unit.

Unit-3 Add Your Process Action

Here we learn how to create and add process action. Here we perform the task according to details provide in the unit.

Unit-4 Test Your Process

Here we test the process that we have create and check whether it process is correct or not. And it is the last task of this module.

Module- Quick Start: Lightning App Builder

Here we build an app for sales reps in the field without code. Here we 3 units let see them.

Unit-1 Create Your First Page

Here we introduction about the lightning App builder, then we create first page. Lightning App Builder lets developers and business users build beautiful custom user interfaces that are designed to work perfectly on your desktop and mobile devices, all without writing a single line of code.

Unit-2 Add More Components

Here we simply add more components to the page according to details provided in unit respectively.

Unit-3 Add Quick Actions and Activate the App

Here it is the last unit where we add Quicks action and configure the app . after configuration of the app we activate the app.

Module- Automate Business Processes for a Recruiting App

Here Automate business processes for recruiting app by creating records, process and flows with process builder and cloud flow designer. Here we 5 units let’s see them.

Unit-1 Build a Process for Creating Interviewer Records

Here we Build processes in an HR recruiting app that streamline workflow for creating interviewer records and approving new positions. Then Create a flow for the app to simplify the process of rating candidates and make the rating process consistent.

Unit2 Lay the Groundwork for an Approval Process

Here we create folder and learn about the email template and how to create email template , then after

creating template we create fields that will we use in approval process.

Unit-3 Create an Approval Process

Here we simply create an approval process with multistep approval process.

Unit-4 Create a Process for Submitting Positions for Approval

Here we first use process builder to create a new process for submitting positions for the approval.

Unit-5 Create a Candidate Rating Flow

Here it the last step were we will be creating candidate rating flow to rate the candidate.

Module-Build a Discount Approval Process

Here we build a discount approval process in order to make sales reps to get the approval if they need for discount offer. Here we have 4 units.

Unit-1 Prepare Your Org

Here we use prepare org were we create new user according given details then we create a role and add custom fields. Then we create a folder and email templates.

Unit-2 Create an Approval Process

Here we simple create approval process according to the details provided in the unit.

Unit-3 Create Initial Submission Actions

Here we create initial submission actions. An initial submission action occurs when a user first submits a record for approval.

Unit-4 Specify Final Approval and Rejection Actions

Here it is last step were we will mention condition for the final approval and final rejection and what action we need to performed in final approval and final rejection.

Module-Salesforce Flow

Here we create flow and automated flows with tools. Here we total 4 units.

Unit-1 Choose the Right Automation Tool

Here we see the List the tools included in Salesforce Flow also see the tools available for automating guided visual experiences and compare the tools available for behind-the-scenes automation. Also see the tools available for approval automation.

Unit-2 Automate Simple Business Processes with Process Builder

Here we see the types of processes that you can build in Process Builder. Then we the key components

used to create a process and Build a process that updates the addresses for an account's contacts when the account’s address is updated.

Unit-3 Guide Users Through Your Business Processes with Flow Builder

Here we see a flow and list its key components then we the types of flow elements. Then we Build a flow that creates a record and uploads files.

Unit-4 Customize How Records Get Approved with Approvals

It is the last unit. Here we Define an approval process, and list its key components. Then we define a business process that can be automated using an approval process. Then we Set up an approval process to automatically manage when an account changes from a prospect to a new customer.

Module- Flow Builder

Here we see what is flow builder and learn how to automate the process with flow. Here we have 5 units.

Unit-1 Learn About Flow Resources and Variables.

Here we see the list of resources available in flow builder and see what a flow variable is.

Unit-2 Create a Variable

Here we see how to create a flow a variable and then we see how to define input and output variable

Unit-3 Add Screens to Your Flow

Here we the list of types of components that we can add to screen and also how add a confirmation screen to the flow.

Unit-4 Add Logic to Your Flow

Here we are to add logic to the flow, then we see the List the logic elements available in Flow Builder. How to Add branching logic to a flow and how to Change a variable value in a flow.

Unit-5 Add Actions to Your Flow

Here we see the List of actions available in Flow Builder. Understand when to use a record variable in a data element and how to Configure a Post to Chatter core action.

Module- Data Security

Here we how to control and access the data using point and click security tools. Here we have 7 units.

Unit-1 Overview of Data Security

Here we the importance of giving the right people access to the right data. Then we see the List of four levels at which we can control data access. And see a scenario for limiting data access at each of the four

levels.

Unit-2 Control Access to the Org

Here we Create, view, and manage users then we Set password policies. Then we Limit the IP addresses from which users can log in and Limit the times at which users can log in.

Unit-3 Control Access to Objects

Here we learn how to View existing profiles and create new ones along how to Modify access to objects using profiles. How to View all assigned users in a profile and Create new permission sets. How to Assign permission sets to single and multiple users.

Unit-4 Control Access to Fields

Here we see how to view and edit field-level security setting and how list the reason to limit access to specific field.

Unit-5 Control Access to Records

Here we see the List of four ways to control access to records. And Describe situations in which to use each of the four record-level security controls. Then we how the different record controls interact with each other. And Set org-wide sharing defaults to control access to records.

Unit-6 Create a Role Hierarchy

Here we see how a role hierarchy is different from an org chart. And how to View and modify a role hierarchy. Then how to Create and assign roles to simplify access to records.

Unit-7 Define Sharing Rules

Here we how to Create a public group that includes users with different profiles and roles.

And Use sharing rules to extend access beyond the role hierarchy structure.3

Module- Keep Data Secure in a Recruiting App

Here we Use field-level security and permission sets to control what data users can see in an app.

Here we see 2 units.

Unit-1 Create Custom Profiles

Here in this unit we Update field-level security and create permission sets in an HR recruiting app so that sensitive data can be viewed only by those who need it. Further restrict data access in the app by changing sharing settings.

Unit-2 Restrict Data Access with Field-Level Security, Permission Sets, and Sharing Settings

Here in this unit we first we create permission sets then we modify field-level security. Now we learn how to create sharing setting.

Module - Apex Triggers

Here we learn how to Write Apex triggers to perform custom database actions. Here we have two units.

Unit-1 Get Started with Apex Triggers

Here we learn how to Write a trigger for a Salesforce object and Use trigger context variables. How to Call a class method from a trigger and Use the sObject addError() method in a trigger to restrict save operations.

In challenge lab we write the following code for apex trigger (AccountAddressTrigger)

```
trigger AccountAddressTrigger on Account (before insert, before update) {
    for(Account account: Trigger.New)
    {
        if(account.Match_Billing_Address__c == True)
        {
            account.ShippingPostalCode = account.BillingPostalCode;
        }
    }
}
```

Unit-2 Bulk Apex Triggers.

Here we Write triggers that operate on collections of sObjects and also we Write triggers that perform efficient SOQL and DML operations.

Here in challenge lab we write the following code for apex trigger (ClosedOpportunityTrigger)

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update)
{
    List<Task> tasklist = new List<Task>();
    for(Opportunity opp: Trigger.New)
    {
        if(opp.StageName == 'Closed Won')
        {
```

```
            tasklist.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));
        }


    }
    if(tasklist.size()>0)
        {
          insert tasklist;
        }


}
```

Module- Apex Testing

Here we deal with apex testing that is we Write robust code by executing Apex unit tests. Here we going to see 3 units.

Unit-1 Get Started with Apex Unit Tests

Here we Describe the key benefits of Apex unit tests, define a class with test methods, execute all test methods in a class and inspect failures. Then Create and execute a suite of test classes.

Here in challenge lab we create a apex class (VerifyDate) the code is

```
public class VerifyDate {


        //method to handle potential checks against two dates
        public static Date CheckDates(Date date1, Date date2) {
                //if date2 is within the next 30 days of date1, use date2.  Otherwise use the end of the
month
                if(DateWithin30Days(date1,date2)) {
                        return date2;
                } else {
                        return SetEndOfMonthDate(date1);
                }
```

```
        }

        //method to check if date2 is within the next 30 days of date1
        @TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {
                //check for date2 being in the past
        if( date2 < date1) { return false; }

        //check that date2 is within (>=) 30 days of date1
        Date date30Days = date1.addDays(30); //create a date 30 days away from date1
                if( date2 >= date30Days ) { return false; }
                else { return true; }
        }

        //method to return the end of the month of a given date
        @TestVisible private static Date SetEndOfMonthDate(Date date1) {
                Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
                Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
                return lastDay;
        }

}
```

Code for unit test-

```
@isTest
private class TestVerifyDate {
   @isTest static void Test_CheckDates_case1()
      {
        Date D =VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('01/05/2020'));
        System.assertEquals(date.parse('01/05/2020'), D);
      }
```

```
    @isTest static void Test_CheckDates_case2()
        {
            Date D =VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('05/05/2020'));
            System.assertEquals(date.parse('01/31/2020'), D);
        }
    @isTest static void Test_DateWithin30Days_case1()
    {
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'), date.parse('12/30/2019'));
        System.assertEquals(false, flag);
    }
     @isTest static void Test_DateWithin30Days_case2()
    {
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'), date.parse('02/02/2019'));
        System.assertEquals(false, flag);
    }
     @isTest static void Test_DateWithin30Days_case3()
    {
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'), date.parse('01/15/2019'));
        System.assertEquals(true, flag);
    }
    @isTest static void Test_SetEndOfMonthDate()
    {
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }



}
```

Unit-2 Test Apex Triggers

Here we Write a test for a trigger that fires on a single record operation and Execute all test methods in a

class.

Here in challenge lab we write the following code for apex trigger (RestrictContactByName)

```
trigger RestrictContactByName on Contact (before insert, before update) {

        //check contacts prior to insert or update for invalid data
        For (Contact c : Trigger.New) {
                if(c.LastName == 'INVALIDNAME') {  //invalidname is invalid
                        c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
                }

        }



}
```

Code for unit test-

```
@isTest
public class TestRestrictContactByName {
   @isTest static void Test_insertupdateContact()
   {
      Contact cnt= new Contact();
      cnt.LastName='INVALIDNAME';

      Test.startTest();
      Database.SaveResult result = Database.insert(cnt, false);
      Test.stopTest();
      System.assert(!result.isSuccess());
      System.assert(result.getErrors().size()>0);
```

```
        System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());
    }


}
```

Unit-3 Create Test Data for Apex Tests

Here we Create a test utility class and Use a test utility method to set up test data for various test cases. At last we Execute all test methods in a class.

Here in challenge lab we create apex class (RandomContactFactory (without the @isTest annotation)) with following code-

```
public class RandomContactFactory {


    public static List<Contact> generateRandomContacts(Integer numcnt, string lastname)
    {
        List<Contact> Contacts = new List<Contact>();
        for (Integer i=0;i<numcnt;i++)
        {
            Contact cnt = new Contact(FirstName = 'Test '+i, LastName = lastname);
            contacts.add(cnt);
        }
        return contacts;
    }


}
```

Module- Asynchronous Apex

Here are going to learn how to write more efficient apex coded with asynchronous processing.

Here we are going to see 6 units so let’s see them.

Unit-1 Asynchronous Processing Basics

Here we the difference between synchronous and asynchronous processing. Understand how to Choose which kind of asynchronous Apex to use in various scenarios.

Unit-2 Use Future Methods

Here we see When to use future methods, the limitations of using future methods. Then How to use future methods for callouts and Future method best practices.

Here in challenge lab we create apex class (AccountProcessor) with following code-

```
public class AccountProcessor {
   @future
   public static void countContacts(List<Id> accountIds){
      List<Account> accounts = [Select Id, Name from Account Where Id IN : accountIds];
      List<Account> updatedAccounts = new List<Account>();
      for(Account account : accounts){
         account.Number_of_Contacts__c = [Select count() from Contact Where AccountId =: account.Id];
          System.debug('No Of Contacts = ' + account.Number_of_Contacts__c);
          updatedAccounts.add(account);
      }
      update updatedAccounts;
   }

}
```

Then apex test class (AccountProcessorTest) with following code-

```
@isTest
public class AccountProcessorTest {
   @isTest
   public static void testNoOfContacts(){
      Account a = new Account();
      a.Name = 'Test Account';
```

```
        Insert a;

        Contact c = new Contact();
        c.FirstName = 'Bob';
        c.LastName =  'Willie';
        c.AccountId = a.Id;

        Contact c2 = new Contact();
        c2.FirstName = 'Tom';
        c2.LastName = 'Cruise';
        c2.AccountId = a.Id;

        List<Id> acctIds = new List<Id>();
        acctIds.add(a.Id);

        Test.startTest();
        AccountProcessor.countContacts(acctIds);
        Test.stopTest();
    }

}
```

Unit-3 Use Batch Apex

Here we see Where to use Batch Apex, the higher Apex limits when using batch, then Batch Apex syntax and Batch Apex best practices.

Here in challenge lab we create a apex class (LeadProcessor) with following code-

```
public class LeadProcessor implements Database.Batchable<sObject> {

    public Database.QueryLocator start(Database.BatchableContext bc) {
      // collect the batches of records or objects to be passed to execute
```

```
        return Database.getQueryLocator([Select LeadSource From Lead ]);
    }
    public void execute(Database.BatchableContext bc, List<Lead> leads){
        // process each batch of records
          for (Lead Lead : leads) {
             lead.LeadSource = 'Dreamforce';
          }
       update leads;
    }
    public void finish(Database.BatchableContext bc){
      }

}
```

Code for apex test class(LeadProcessorTest)-

```
@isTest
public class LeadProcessorTest {

       @testSetup
    static void setup() {
       List<Lead> leads = new List<Lead>();
       for(Integer counter=0 ;counter <200;counter++){
          Lead lead = new Lead();
          lead.FirstName ='FirstName';
          lead.LastName ='LastName'+counter;
          lead.Company ='demo'+counter;
          leads.add(lead);
       }
       insert leads;
    }
```

```
    @isTest static void test() {
        Test.startTest();
        LeadProcessor leadProcessor = new LeadProcessor();
        Id batchId = Database.executeBatch(leadProcessor);
        Test.stopTest();
    }


}
```

Unit-4 Control Processes with Queueable Apex

Here we see When to use the Queueable interface, The differences between queueable and future methods. Then we see Queueable Apex syntax and Queueable method best practices.

Here in challenge we create a apex class (AddPrimaryContact) with following code-

```
public class AddPrimaryContact implements Queueable
{
    private Contact c;
    private String state;
    public  AddPrimaryContact(Contact c, String state)
    {
        this.c = c;
        this.state = state;
    }
    public void execute(QueueableContext context)
    {
         List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName from contacts ) FROM ACCOUNT WHERE BillingState = :state LIMIT 200];
         List<Contact> lstContact = new List<Contact>();
         for (Account acc:ListAccount)
         {
```

```
            Contact cont = c.clone(false,false,false,false);
            cont.AccountId =  acc.id;
            lstContact.add( cont );
        }

        if(lstContact.size() >0 )
        {
           insert lstContact;
        }

    }

}
```

Code for apex test class (AddPrimaryContactTest)-

```
@isTest
public class AddPrimaryContactTest
{
    @isTest static void TestList()
    {
        List<Account> Teste = new List <Account>();
        for(Integer i=0;i<50;i++)
        {
           Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
        }
        for(Integer j=0;j<50;j++)
        {
           Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
        }
        insert Teste;
```

```
        Contact co = new Contact();
        co.FirstName='demo';
        co.LastName ='demo';
        insert co;
        String state = 'CA';

         AddPrimaryContact apc = new AddPrimaryContact(co, state);
         Test.startTest();
           System.enqueueJob(apc);
         Test.stopTest();
     }
 }
```

Unit-5 Schedule Jobs Using the Apex Scheduler

Here we see When to use scheduled Apex, how to monitor scheduled jobs, Scheduled Apex syntax, Scheduled method best practices.

Here in challenge lab we create apex class (DailyLeadProcessor) with following code-

```
public class DailyLeadProcessor implements Schedulable  {
   Public void execute(SchedulableContext SC){
      List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit 200];
       for(Lead l:LeadObj){
          l.LeadSource='Dreamforce';
          update l;
       }
   }
}
```

Code for apex test class (DailyLeadProcessorTest)-

```
@isTest
private class DailyLeadProcessorTest {
```

```
    static testMethod void testDailyLeadProcessor() {
        String CRON_EXP = '0 0 1 * * ?';
        List<Lead> lList = new List<Lead>();
      for (Integer i = 0; i < 200; i++) {
            lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1 Inc.',
Status='Open - Not Contacted'));
        }
        insert lList;

        Test.startTest();
        String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
DailyLeadProcessor());
    }
}
```

Unit-6 Monitor Asynchronous Apex

Here it is last unit of the module where we see How to monitor the different types of jobs and How to use the flex queue.

Module- Lightning Web Components Basics

Here we build the reusable, performant components that follow modern web standards. Here we have 5 units let see them.

Unit-1 Discover Lightning Web Components

Here we see the Lightning Web Components programming model and List the benefits of using Lightning web components. Then we Find why we need to get started developing Lightning web components.

Unit-2 Create Lightning Web Components

Here we Describe the contents of each component file and Create JavaScript methods for a Lightning web component. And how to Use Lifecycle Hooks in component JavaScript.

Unit-3 Deploy Lightning Web Component Files

Here we Configure Lightning web component files for display in an org and Deploy your files to an org then Verify component behavior in an org environment.

Here in challenge we done the following coding-

bikeCard.html code-

```
<template>
    <div>
        <div>Name: {name}</div>
        <div>Description: {description}</div>
        <lightning-badge label={material}></lightning-badge>
        <lightning-badge label={category}></lightning-badge>
        <div>Price: {price}</div>
        <div><img src={pictureUrl}/></div>
    </div>
</template>
```

bikeCard.js code-

```
import { LightningElement } from 'lwc';
export default class BikeCard extends LightningElement {
    name = 'Electra X4';
    description = 'A sweet bike built for comfort.';
    category = 'Mountain';
    material = 'Steel';
    price = '$2,700';
    pictureUrl = 'https://s3-us-west-1.amazonaws.com/sfdc-demo/ebikes/electrax4.jpg';
 }
```

bikeCard.js-meta.xml code-

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <!-- The apiVersion may need to be increased for the current release -->
    <apiVersion>52.0</apiVersion>
    <isExposed>true</isExposed>
    <masterLabel>Product Card</masterLabel>
    <targets>
```

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

Unit-4 Handle Events in Lightning Web Components

Here we Create an app that includes multiple components. And Describe the file structure of a complex component. And Handle events.

Unit-5 Add Styles and Data to a Lightning Web Component

Here it is the last unit were we Use CSS and Lightning Design System with a component, Get data from a Salesforce org. And Deploy your app to an org and test it.

Module-API Basics

Here we Learn the fundamentals and benefits of developing with APIs. Here we have total 3 units.

Unit-1 Make APIs for You and Me

Here we define and describe an API. Also we see the name common uses of API's.

Then we simple quiz about the this unit.

Unit-2 Learn the Benefits of APIs

Here we Define abstraction. Then we Name the benefits of using an API. And see what is HTTP verbs and their usage. In last a simple quiz.

Unit-3 Put the Web in Web API

Here we State common uses of web APIs and Understand the API economy and why API growth has been significant. Also we see two reasons why web applications are becoming more popular.

Module-Event Monitoring

Here we Discover insights of Salesforce org with powerful monitoring feature. Here we total 3 units.

Unit-1 Get Started with Event Monitoring

Here we see the event types supported by Event Monitoring. Then we Define event log files.Then we see three use cases for Event Monitoring and Describe the application programming interface (API)-first approach to development.

Unit-2 Query Event Log Files

Here we Query an EventLogFile object using Developer Console and View events in Salesforce Event Log File (ELF) Browser. Then we Learn about EventLogFile event types.

Unit-3Download and Visualize Event Log Files

Here we Download an event log file.And see the structure of event log files. Then we Identify an application for downloading event log files without writing code using the Event Log File (ELF) Browser, see how to use a cURL or Python script for downloading data and also Identify options for visualizing event log file data

Module-Shield Platform Encryption

Here we Encrypt data at-rest in the cloud and manage the life cycle of encryption keys.

Here we have total 3 units.

Unit-1 Get Started with Shield Platform Encryption

Here we see encryption and understand how to protects data then we see the difference between Classic Encryption and Shield Platform Encryption. Then we understand the relationship between tenant secrets, keys, and master secrets and Identify the permissions needed to set up Shield Platform Encryption in an org**.**

Unit-2 Set Up and Manage Shield Platform Encryption

Here we Create a tenant secret then we Enable encryption for files, fields, and attachments. And

Assign permission to generate, rotate, and archive your org's keys.

Unit-3 Deploy Shield Platform Encryption the Smart Way

Here we Identify best practices when setting up Shield Platform Encryption and see how Shield Platform Encryption affects apps and sandboxes then we Understand how Shield Platform Encryption affects the way users access information in org.

Module- Apex Integration Services

Here we use Apex Integration Services to integrate with external apps using Apex REST and SOAP services. Here we have 4 units.

Unit-1 Apex Integration Overview

Here we see the differences between web service and HTTP callouts. And Authorize an external site with remote site settings.

Unit-2 Apex REST Callouts

Here we Perform a callout to receive data from an external service, then we Perform a callout to send data to an external service. At last we Test callouts by using mock callouts.

Here in challenge lab we do the following code-

Code for apex class (AnimalLocator)-

```
public class AnimalLocator{
    public static String getAnimalNameById(Integer x){
        Http http = new Http();
        HttpRequest req = new HttpRequest();
        req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);
        req.setMethod('GET');
        Map<String, Object> animal= new Map<String, Object>();
        HttpResponse res = http.send(req);
            if (res.getStatusCode() == 200) {
        Map<String, Object> results = (Map<String, Object>)JSON.deserializeUntyped(res.getBody());
      animal = (Map<String, Object>) results.get('animal');
        }
return (String)animal.get('name');
    }
}
```

Code for test class (AnimalLocatorTest)

```
@isTest
private class AnimalLocatorTest{
    @isTest static void AnimalLocatorMock1() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        string result = AnimalLocator.getAnimalNameById(3);
        String expectedResult = 'chicken';
```

```
        System.assertEquals(result,expectedResult );
    }
}
```

Code for (AnimalLocatorMock)

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    // Implement this interface method
    global HTTPResponse respond(HTTPRequest request) {
        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken", "mighty moose"]}');
        response.setStatusCode(200);
        return response;
    }
}
```

Unit-3 Apex SOAP Callouts

Here we Generate Apex classes using WSDL2Apex then we Perform a callout to send data to an external service using SOAP. At last Test callouts by using mock callouts.

Here in challenge we do the following code –

Code for class (parkLocator)-

```
public class ParkLocator {
    public static string[] country(string theCountry) {
        ParkService.ParksImplPort  parkSvc = new  ParkService.ParksImplPort(); // remove space
        return parkSvc.byCountry(theCountry);
    }
```

```
}
```

Code for class test (ParkLocatorTest)-

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock ());
        String country = 'United States';
        List<String> result = ParkLocator.country(country);
        List<String> parks = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};
         System.assertEquals(parks, result);
    }
}
```

Code for unit test (ParkServiceMock)-

```
@isTest
global class ParkServiceMock implements WebServiceMock {
   global void doInvoke(
            Object stub,
            Object request,
            Map<String, Object> response,
            String endpoint,
            String soapAction,
            String requestName,
            String responseNS,
            String responseName,
            String responseType) {
        // start - specify the response you want to send
        ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
        response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};
```

```
        // end
        response.put('response_x', response_x);
    }
}
```

Unit-4 Apex Web Services

Here we see the two types of Apex web services and provide a high-level overview of these services. Then we Create an Apex REST class that contains methods for each HTTP method and Invoke a custom Apex REST method with an endpoint. Then we learn how to Pass data to a custom Apex REST method by sending a request body in JSON format. How to Write a test method for an Apex REST method and set properties in a test REST request. And how to Write a test method for an Apex REST method by calling the method with parameter values.

Here in challenge lab we do the following code-

Code for apex class (Account Manager)-

```
@RestResource(urlMapping='/Accounts/*/contacts')
global class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest req = RestContext.request;
        String accId = req.requestURI.substringBetween('Accounts/', '/contacts');
        Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                       FROM Account WHERE Id = :accId];
        return acc;
    }
}
```

Code for unit test (Account Manager Test)-

```
@isTest
private class AccountManagerTest {

    private static testMethod void getAccountTest1() {
```

```
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts' ;
        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Account thisAccount = AccountManager.getAccount();
        // Verify results
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);

    }

    // Helper method
        static Id createTestRecord() {
        // Create test record
        Account TestAcc = new Account(
          Name='Test record');
        insert TestAcc;
        Contact TestCon= new Contact(
        LastName='Test',
        AccountId = TestAcc.id);
        return TestAcc.Id;
    }
}
```

Super Badge-Apex Specialist

Here we Use integration and business logic to push your Apex coding skills to the limit.

Before going to it we need to complete apex trigger, apex testing, asynchronous apex, apex integration service, then apex specialist

Here in apex specialist we will do the following tasks-.

1. Automate record creation using Apex triggers
2. Synchronize Salesforce data with an external system using asynchronous REST callouts
3. Schedule synchronization using Apex code
4. Test automation logic to confirm Apex trigger side effects
5. Test integration logic using callout mocks
6. Test scheduling logic to confirm action gets queued

Code work-

Step-1

Code for Maintenance Request helper class-

```
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();
        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                    validIds.add(c.Id);
                }
            }
        }

        //When an existing maintenance request of type Repair or Routine Maintenance is closed,
        //create a new maintenance request for a future routine checkup.
        if (!validIds.isEmpty()){
            Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,
```

```
Equipment__r.Maintenance_Cycle__c,
                                    (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
                                    FROM Case WHERE Id IN :validIds]);
    Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

    //calculate the maintenance request due dates by using the maintenance cycle defined on the
related equipment records.
    AggregateResult[] results = [SELECT Maintenance_Request__c,
                        MIN(Equipment__r.Maintenance_Cycle__c)cycle
                        FROM Equipment_Maintenance_Item__c
                        WHERE Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];

    for (AggregateResult ar : results){
      maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
    }

    List<Case> newCases = new List<Case>();
    for(Case cc : closedCases.values()){
      Case nc = new Case (
        ParentId = cc.Id,
        Status = 'New',
        Subject = 'Routine Maintenance',
        Type = 'Routine Maintenance',
        Vehicle__c = cc.Vehicle__c,
        Equipment__c =cc.Equipment__c,
        Origin = 'Web',
        Date_Reported__c = Date.Today()
      );
```

```
            //If multiple pieces of equipment are used in the maintenance request,
            //define the due date by applying the shortest maintenance cycle to today’s date.
            If (maintenanceCycles.containskey(cc.Id)){
                nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
            } else {
                nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
            }

            newCases.add(nc);
        }

        insert newCases;

        List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
        }
        insert clonedList;
    }
  }
}
```

Code for Maintenance Request Trigger

```
trigger MaintenanceRequest on Case (before update, after update) {
    if(Trigger.isUpdate && Trigger.isAfter){
        MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
    }
}
```

Step-2

Code for ware houser service

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

    //Write a class that makes a REST callout to an external warehouse system to get a list of equipment
that needs to be updated.
    //The callout’s JSON response returns the equipment records that you upsert in Salesforce.

    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        System.debug('go into runWarehouseEquipmentSync');
        Http http = new Http();
        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

        List<Product2> product2List = new List<Product2>();
```

```
        System.debug(response.getStatusCode());
        if (response.getStatusCode() == 200){
            List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());


            //class maps the following fields:
            //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce
            for (Object jR : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)jR;
                Product2 product2 = new Product2();
                //replacement part (always true),
                product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                //cost
                product2.Cost__c = (Integer) mapJson.get('cost');
                //current inventory
                product2.Current_Inventory__c = (Double) mapJson.get('quantity');
                //lifespan
                product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                //maintenance cycle
                product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                //warehouse SKU
                product2.Warehouse_SKU__c = (String) mapJson.get('sku');


                product2.Name = (String) mapJson.get('name');
                product2.ProductCode = (String) mapJson.get('_id');
                product2List.add(product2);
            }
```

```
            if (product2List.size() > 0){
                upsert product2List;
                System.debug('Your equipment was synced with the warehouse one');
            }
        }
    }

    public static void execute (QueueableContext context){
        System.debug('start runWarehouseEquipmentSync');
        runWarehouseEquipmentSync();
        System.debug('end runWarehouseEquipmentSync');
    }

}
```

For execute

```
System.enqueueJob(new WarehouseCalloutService());
```

STEP-3

Code for ware house

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
    //implement scheduled code here
    global void execute(SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

Step-4

Code for Maintenace request help

```
public with sharing class MaintenanceRequestHelper {
```

```
public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {
    Set<Id> validIds = new Set<Id>();
    For (Case c : updWorkOrders){
        if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
            if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                validIds.add(c.Id);
            }
        }
    }

    //When an existing maintenance request of type Repair or Routine Maintenance is closed,
    //create a new maintenance request for a future routine checkup.
    if (!validIds.isEmpty()){
        Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,
Equipment__r.Maintenance_Cycle__c,
                                    (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
                                    FROM Case WHERE Id IN :validIds]);
        Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

        //calculate the maintenance request due dates by using the maintenance cycle defined on the
related equipment records.
        AggregateResult[] results = [SELECT Maintenance_Request__c,
                            MIN(Equipment__r.Maintenance_Cycle__c)cycle
                            FROM Equipment_Maintenance_Item__c
                            WHERE Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];

        for (AggregateResult ar : results){
            maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
```

```
        }

        List<Case> newCases = new List<Case>();
        for(Case cc : closedCases.values()){
            Case nc = new Case (
                ParentId = cc.Id,
                Status = 'New',
                Subject = 'Routine Maintenance',
                Type = 'Routine Maintenance',
                Vehicle__c = cc.Vehicle__c,
                Equipment__c =cc.Equipment__c,
                Origin = 'Web',
                Date_Reported__c = Date.Today()
            );

            //If multiple pieces of equipment are used in the maintenance request,
            //define the due date by applying the shortest maintenance cycle to today’s date.
            If (maintenanceCycles.containskey(cc.Id)){
                nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
            }

            newCases.add(nc);
        }

        insert newCases;

        List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
```

```
            for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
        }
        insert clonedList;
    }
  }
}
```

Code for tesr maintenance test helper

```
@isTest
public with sharing class MaintenanceRequestHelperTest {

  // createVehicle
  private static Vehicle__c createVehicle(){
    Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
    return vehicle;
  }

  // createEquipment
  private static Product2 createEquipment(){
    product2 equipment = new product2(name = 'Testing equipment',
                                      lifespan_months__c = 10,
                                      maintenance_cycle__c = 10,
                                      replacement_part__c = true);
    return equipment;
  }
```

```
    // createMaintenanceRequest
    private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cse = new case(Type='Repair',
                            Status='New',
                            Origin='Web',
                            Subject='Testing subject',
                            Equipment__c=equipmentId,
                            Vehicle__c=vehicleId);
        return cse;
    }

    // createEquipmentMaintenanceItem
    private static Equipment_Maintenance_Item__c createEquipmentMaintenanceItem(id equipmentId,id
requestId){
        Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
            Equipment__c = equipmentId,
            Maintenance_Request__c = requestId);
        return equipmentMaintenanceItem;
    }

    @isTest
    private static void testPositive(){
        Vehicle__c vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        Product2 equipment = createEquipment();
```

```
        insert equipment;
        id equipmentId = equipment.Id;

        case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
        insert createdCase;

        Equipment_Maintenance_Item__c equipmentMaintenanceItem =
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
        insert equipmentMaintenanceItem;

        test.startTest();
        createdCase.status = 'Closed';
        update createdCase;
        test.stopTest();

        Case newCase = [Select id,
                        subject,
                        type,
                        Equipment__c,
                        Date_Reported__c,
                        Vehicle__c,
                        Date_Due__c
                       from case
                       where status ='New'];

        Equipment_Maintenance_Item__c workPart = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c =:newCase.Id];
        list<case> allCase = [select id from case];
```

```
        system.assert(allCase.size() == 2);

        system.assert(newCase != null);
        system.assert(newCase.Subject != null);
        system.assertEquals(newCase.Type, 'Routine Maintenance');
        SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
        SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
    }

    @isTest
    private static void testNegative(){
        Vehicle__C vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        product2 equipment = createEquipment();
        insert equipment;
        id equipmentId = equipment.Id;

        case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
        insert createdCase;

        Equipment_Maintenance_Item__c workP = createEquipmentMaintenanceItem(equipmentId,
createdCase.Id);
        insert workP;

        test.startTest();
        createdCase.Status = 'Working';
```

```
        update createdCase;
        test.stopTest();

        list<case> allCase = [select id from case];

        Equipment_Maintenance_Item__c equipmentMaintenanceItem = [select id
                                        from Equipment_Maintenance_Item__c
                                        where Maintenance_Request__c = :createdCase.Id];

        system.assert(equipmentMaintenanceItem != null);
        system.assert(allCase.size() == 1);
    }

    @isTest
    private static void testBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> equipmentMaintenanceItemList = new
list<Equipment_Maintenance_Item__c>();
        list<case> caseList = new list<case>();
        list<id> oldCaseIds = new list<id>();

        for(integer i = 0; i < 300; i++){
            vehicleList.add(createVehicle());
            equipmentList.add(createEquipment());
        }
        insert vehicleList;
        insert equipmentList;
```

```
for(integer i = 0; i < 300; i++){
    caseList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));
}
insert caseList;

for(integer i = 0; i < 300; i++){
    equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipmentList.get(i).id,
caseList.get(i).id));
}
insert equipmentMaintenanceItemList;

test.startTest();
for(case cs : caseList){
    cs.Status = 'Closed';
    oldCaseIds.add(cs.Id);
}
update caseList;
test.stopTest();

list<case> newCase = [select id
                      from case
                      where status ='New'];



list<Equipment_Maintenance_Item__c> workParts = [select id
                                                 from Equipment_Maintenance_Item__c
                                                 where Maintenance_Request__c in: oldCaseIds];
```

```
        system.assert(newCase.size() == 300);


        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 600);
    }
}
```

Code for maintenance help

```
trigger MaintenanceRequest on Case (before update, after update) {
    if(Trigger.isUpdate && Trigger.isAfter){
        MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
    }
}
```

Step-5

Code for WarehouseCallout Service

```
public with sharing class WarehouseCalloutService implements Queueable {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';


    //Write a class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.
    //The callout's JSON response returns the equipment records that you upsert in Salesforce.


    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        System.debug('go into runWarehouseEquipmentSync');
        Http http = new Http();
        HttpRequest request = new HttpRequest();
```

```
        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

        List<Product2> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() == 200){
            List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            //class maps the following fields:
            //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce
            for (Object jR : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)jR;
                Product2 product2 = new Product2();
                //replacement part (always true),
                product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                //cost
                product2.Cost__c = (Integer) mapJson.get('cost');
                //current inventory
                product2.Current_Inventory__c = (Double) mapJson.get('quantity');
                //lifespan
                product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                //maintenance cycle
                product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                //warehouse SKU
                product2.Warehouse_SKU__c = (String) mapJson.get('sku');
```

```
            product2.Name = (String) mapJson.get('name');
            product2.ProductCode = (String) mapJson.get('_id');
            product2List.add(product2);
        }

        if (product2List.size() > 0){
            upsert product2List;
            System.debug('Your equipment was synced with the warehouse one');
        }
    }
}

public static void execute (QueueableContext context){
    System.debug('start runWarehouseEquipmentSync');
    runWarehouseEquipmentSync();
    System.debug('end runWarehouseEquipmentSync');
}

}
```

Code for warehousecalloutserive mock

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request) {

        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
```

```
response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

Code for warehouseservice test

```
@IsTest
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
        @isTest
    static void testWarehouseCallout() {
        test.startTest();
        test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.execute(null);
        test.stopTest();

        List<Product2> product2List = new List<Product2>();
        product2List = [SELECT ProductCode FROM Product2];

        System.assertEquals(3, product2List.size());
        System.assertEquals('55d66226726b611100aaf741', product2List.get(0).ProductCode);
        System.assertEquals('55d66226726b611100aaf742', product2List.get(1).ProductCode);
```

```
        System.assertEquals('55d66226726b611100aaf743', product2List.get(2).ProductCode);
    }
}
```

Step-6

Code for ware house Sync Schedule test.apxc

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    // implement scheduled code here
    //
    @isTest static void test() {
        String scheduleTime = '00 00 00 * * ? *';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobId = System.schedule('Warehouse Time to Schedule to test', scheduleTime, new
WarehouseSyncSchedule());
        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
        System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');

        Test.stopTest();
    }
}
```

Code for Ware house Callout Service Mock

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request) {
```

```
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Gene
rator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226726b61110
0aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611100aaf74
3","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

Code for warehousesync schedule

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
    //implement scheduled code here
    global void execute(SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

Super badge- **Process Automation Specialist**

Here we see mastery of business process automation without writing a line of code.
Before doing this we formulas and validations, sales flow, leads & opportunities for lightning experience then super badge i.e. Process Automation Specialist.

Here we will be doing following tasks-

1. Automate lead ownership using assignment rules
2. Enforce data integrity with formula fields and validation rules
3. Create a custom object in a master-detail relationship to a standard object
4. Define an opportunity sales process using stages, record types, and validation rules
5. Automate business processes to send emails, create related records, and submit opportunities for approval
6. Create a flow to display dynamic information on a Lightning record page
7. Create a process to evaluate and update records.

SUPERBADGE COMPLETE!

+10000 Points

Discover more trailmixes

Completed 6/23/22