PROJECT DOCUMENT

APEX TRIGGERS:

Create an Apex trigger that sets an account's Shipping Postal Code to match the Billing Postal Code if the Match Billing Address option is selected. Fire the trigger before inserting an account or updating an account.

Pre-Work:

Add a checkbox field to the Account object:

- Field Label: `Match Billing Address`
- Field Name: `Match_Billing_Address`

  Note: The resulting API Name should be `Match_Billing_Address__c`.
- Create an Apex trigger:
  - Name: `AccountAddressTrigger`
  - Object: **Account**
  - Events: before insert and before update
  - Condition: Match Billing Address is true
  - Operation: set the Shipping Postal Code to match the Billing Postal Code

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;

      }
   }
```

2.BULK APEX TRIGGERS:

Create a bulkified Apex trigger that adds a follow-up task to an opportunity if its stage is

Closed Won. Fire the Apex trigger after inserting or updating an opportunity.

- Create an Apex trigger:
  - Name: `ClosedOpportunityTrigger`
  - Object: **Opportunity**
  - Events: after insert and after update
  - Condition: Stage is `Closed Won`
  - Operation: Create a task:
    - `Subject`: `Follow Up Test Task`
    - `WhatId`: the opportunity ID (associates the task with the opportunity)
  - Bulkify the Apex trigger so that it can insert or update 200 or more opportunities

`ClosedOpportunityTrigger:`

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
   List<Task> taskList = new List<Task>();
   //first way
   for(Opportunity opp : [SELECT Id, StageName FROM Opportunity WHERE StageName='Closed
Won' AND Id IN : Trigger.New]){
      taskList.add(new Task(Subject='Follow Up Test Task', WhatId = opp.Id));
   }

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

}
```

APEX TESTING

Apex Unit Tests:

The Apex testing framework enables you to write and execute tests for your Apex classes and triggers on the Lightning Platform. Apex unit tests ensure high quality for your Apex code and let you meet requirements for deploying Apex.

Testing is the key to successful long-term development and is a critical component of the development process. The Apex testing framework makes it easy to test your Apex code. Apex code can only be written in a sandbox environment or a Developer org, not in production. Apex code can be deployed to a production org from a sandbox. Also, app developers can distribute

Apex code to customers from their Developer orgs by uploading packages to the Lightning Platform AppExchange. In addition to being critical for quality assurance, Apex unit tests are also requirements for deploying and distributing Apex.

GET STARTED WITH APEX UNIT TEST:

Create and install a simple Apex class to test if a date is within a proper range, and if not, returns a date that occurs at the end of the month within the range. You'll copy the code for the class from GitHub. Then write unit tests that achieve 100% code coverage.

- Create an Apex class:
  - Name: `VerifyDate`
  - Code: **Copy from GitHub**
- Place the unit tests in a separate test class:
  - Name: `TestVerifyDate`
  - Goal: 100% code coverage
- Run your test class at least once

1.VerifyDate class :

```
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
 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
  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;
  }

}
```

2.TestVerifyDate :

```
@isTest
public class TestVerifyDate
{
   static testMethod void testMethod1()
   {
      Date d = VerifyDate.CheckDates(System.today(),System.today()+1);
      Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);
   }
}
```

TEST APEX TRIGGERS:

Create and install a simple Apex trigger which blocks inserts and updates to any contact with a last name of 'INVALIDNAME'. You'll copy the code for the class from GitHub. Then write unit tests that achieve 100% code coverage.

- Create an Apex trigger on the Contact object
  - Name: `RestrictContactByName`
  - Code: **Copy from GitHub**
- Place the unit tests in a separate test class
  - Name: `TestRestrictContactByName`
  - Goal: 100% test coverage
- Run your test class at least once

=>To run this test, click Test | New Run. Under Test Classes, click TestAccountDeletion. To add all the methods in the TestAccountDeletion class to the test run, click Add Selected. Click Run.

1.RestrictContactByName :

```
trigger RestrictContactByName on Contact (before insert, before update) {
  For (Contact c : Trigger.New) {
    if(c.LastName == 'INVALIDNAME') {
      c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
    }
  }
}
```

CREATE TEST DATA FOR APEX TESTS:

Create an Apex class that returns a list of contacts based on two incoming parameters: the number of contacts to generate and the last name. Do not insert the generated contact records into the database.

NOTE: For the purposes of verifying this hands-on challenge, don't specify the @isTest annotation for either the class or the method, even though it's usually required.

- Create an Apex class in the `public` scope
  - Name: `RandomContactFactory` (without the @isTest annotation)
- Use a Public Static Method to consistently generate contacts with unique first names based on the iterated number in the format Test 1, Test 2 and so on.
  - Method Name: `generateRandomContacts` (without the @isTest annotation)
  - Parameter 1: An integer that controls the number of contacts being generated with unique first names
  - Parameter 2: A string containing the last name of the contacts
  - Return Type: `List < Contact >`

TestRestrictContactByName :

```
@isTest
private class TestRestrictContactByName {
   static testMethod void  metodoTest()
   {
      List<Contact> listContact= new List<Contact>();
      Contact c1 = new Contact(FirstName='Francesco', LastName='Riggio' ,
email='Test@test.com');
      Contact c2 = new Contact(FirstName='Francesco1', LastName =
'INVALIDNAME',email='Test@test.com');
      listContact.add(c1);
      listContact.add(c2);
      Test.startTest();
         try
         {
            insert listContact;
         }
         catch(Exception ee)
         {
         }
      Test.stopTest();
   }
}
```

1.RandomContactFactory.apxc

```
public class RandomContactFactory {
   public static List<Contact> generateRandomContacts(Integer
numContactsToGenerate, String FName) {
      List<Contact> contactList = new List<Contact>();
      for(Integer i=0;i<numContactsToGenerate;i++) {
         Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
         contactList.add(c);
         System.debug(c);
      }
      //insert contactList;
      System.debug(contactList.size());
      return contactList;
   }
}
```

ASYNCHRONOUS APEX:

Asynchronous Apex is used to run processes in a separate thread, at a later time.
An asynchronous process is a process or function that executes a task "in the background" without the user having to wait for the task to finish.

=>USE FUTURE METHODS:

Create an Apex class with a future method that accepts a List of Account IDs and updates a custom field on the Account object with the number of contacts associated to the Account. Write unit tests that achieve 100% code coverage for the class. Every hands-on challenge in this module asks you to create a test class.

● Create a field on the Account object:
○ Label: Number Of Contacts
○ Name: Number_Of_Contacts
○ Type: Number
○ This field will hold the total number of Contacts for the Account
● Create an Apex class:
○ Name: AccountProcessor
○ Method name: countContacts
○ The method must accept a List of Account IDs
○ The method must use the @future annotation
○ The method counts the number of Contact records associated to each Account ID passed to the method and updates the 'Number_Of_Contacts__c' field with this value
● Create an Apex test class:
○ Name: AccountProcessorTest
○ The unit tests must cover all lines of code included in the AccountProcessor class, resulting in 100% code coverage.
● Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

1.AccountProcessor.apxc

```
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;
   }

}
```

2.AccountProcessorTest.apxc

```
@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();
   }
}
```

USE BATCH APEX:

Create an Apex class that implements the Database.Batchable interface to update all Lead records in the org with a specific LeadSource.

- Create an Apex class:
  - Name: `LeadProcessor`
  - Interface: `Database.Batchable`
  - Use a QueryLocator in the start method to collect all Lead records in the org
  - The execute method must update all Lead records in the org with the LeadSource value of `Dreamforce`
- Create an Apex test class:
  - Name: `LeadProcessorTest`
  - In the test class, insert 200 Lead records, execute the LeadProcessor Batch class and test that all Lead records were updated correctly
  - The unit tests must cover all lines of code included in the **LeadProcessor**

class, resulting in 100% code coverage

- Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

1.LeadProcessor.apxc

```
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){
    }

}
```

2.LeadProcessorTest.apxc

```
@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();
    }

}
```

=>CONTROL PROCESSES WITH QUEUEABLE APEX:

Create a Queueable Apex class that inserts the same Contact for each Account for a specific state.

- Create an Apex class:
  - Name: `AddPrimaryContact`
  - Interface: `Queueable`
  - Create a constructor for the class that accepts as its first argument a Contact sObject and a second argument as a string for the State abbreviation
  - The `execute` method must query for a maximum of 200 Accounts with the `BillingState` specified by the State abbreviation passed into the constructor and insert the Contact sObject record associated to each Account. Look at the sObject `clone()` method.
- Create an Apex test class:
  - Name: `AddPrimaryContactTest`
  - In the test class, insert 50 Account records for `BillingState NY` and 50 Account records for `BillingState CA`
  - Create an instance of the `AddPrimaryContact` class, enqueue the job, and assert that a Contact record was inserted for each of the 50 Accounts with the `BillingState` of `CA`
  - The unit tests must cover all lines of code included in the **AddPrimaryContact** class, resulting in 100% code coverage
- Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

1.AddPrimaryContact.apxc

```
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;
       }

   }

}
```

2.AddPrimaryContactTest.apxc

```
@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();
    }
}
```

SCHEDULE JOBS USING APEX SCHEDULER:

Create an Apex class that implements the Schedulable interface to update Lead records with a specific LeadSource. (This is very similar to what you did for Batch Apex.)

- Create an Apex class:
    - Name: `DailyLeadProcessor`
    - Interface: `Schedulable`
    - The execute method must find the first 200 Lead records with a blank LeadSource field and update them with the LeadSource value of `Dreamforce`
- Create an Apex test class:
    - Name: `DailyLeadProcessorTest`
    - In the test class, insert 200 Lead records, schedule the DailyLeadProcessor class to run and test that all Lead records were updated correctly
    - The unit tests must cover all lines of code included in the **DailyLeadProcessor** class, resulting in 100% code coverage.
- Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

1.DailyLeadProcessor.apxc:

```
global class DailyLeadProcessor implements Schedulable {
 global void execute(SchedulableContext ctx) {
      List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null];

      if(!lList.isEmpty()) {
  for(Lead l: lList) {
   l.LeadSource = 'Dreamforce';
  }
  update lList;
 }
  }

}
```

2.DailyLeadProcessorTest.apxc

```
@isTest
public class DailyLeadProcessorTest {
//Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
   public static String CRON_EXP = '0 0 0 2 6 ? 2022';

   static testmethod void testScheduledJob(){
      List<Lead> leads = new List<Lead>();

      for(Integer i = 0; i < 200; i++){
         Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company ' + i,
Status = 'Open - Not Contacted');
         leads.add(lead);
      }

      insert leads;

      Test.startTest();
      // Schedule the test job
      String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP, new
DailyLeadProcessor());

      // Stopping the test will run the job synchronously
      Test.stopTest();
   }

}
```

APEX INTEGRATION SERVICES:

APEX REST CALLOUTS:

Create an Apex class that calls a REST endpoint to return the name of an animal, write unit tests that achieve 100% code coverage for the class using a mock response, and run your Apex tests.

Prework: Be sure the Remote Sites from the first unit are set up.

- Create an Apex class:
  - Name: `AnimalLocator`
  - Method name: `getAnimalNameById`
  - The method must accept an Integer and return a String.
  - The method must call https://th-apex-http-callout.herokuapp.com/animals/<id>, replacing <id> with the ID passed into the method
  - The method returns the value of the **name** property (i.e., the animal name)
- Create a test class:
  - Name: `AnimalLocatorTest`
  - The test class uses a mock class called `AnimalLocatorMock` to mock the callout response
- Create unit tests:
  - Unit tests must cover all lines of code included in the **AnimalLocator** class, resulting in 100% code coverage
- Run your test class at least once (via **Run All** tests the Developer Console) before attempting to verify this challenge

sol:

From Setup, enter Remote Site Settings in the Quick Find box, then click Remote Site Settings.
Click New Remote Site ,
For the remote site name, enter = animals_http
For the remote site URL, enter = https://th-apex-http-callout.herokuapp.com
For the description, enter = Trailhead animal service: HTTP.
Click Save & New.
For the second remote site name, enter = animals_soap
For the remote site URL, enter = https://th-apex-soap-service.herokuapp.com
For the description, enter = Trailhead animal service: SOAP.
Click Save.

1.AnimalLocator.apxc:

```
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');
    }
}
```

2.AnimalLocatorMock.apxc:

```
@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;
    }
}
```

3.AnimalLocatorTest.apxc:

```
@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 );
    }
}
```

APEX SOAP CALLOUTS:

Generate an Apex class using WSDL2Apex for a SOAP web service, write unit tests that achieve 100% code coverage for the class using a mock response, and run your Apex tests.

Prework: Be sure the Remote Sites from the first unit are set up.

- Generate a class using this using this WSDL file:
  - Name: `ParkService` (Tip: After you click the **Parse WSDL** button, change the Apex class name from **parksServices** to `ParkService`)
  - Class must be in public scope
- Create a class:
  - Name: `ParkLocator`
  - Class must have a **country** method that uses the **ParkService** class
  - Method must return an array of available park names for a particular country passed to the web service (such as Germany, India, Japan, and United States)
- Create a test class:
  - Name: `ParkLocatorTest`
  - Test class uses a mock class called `ParkServiceMock` to mock the callout response
- Create unit tests:
  - Unit tests must cover all lines of code included in the **ParkLocator** class, resulting in 100% code coverage.
- Run your test class at least once (via **Run All** tests the Developer Console) before attempting to verify this challenge.

sol:

From Setup, enter Remote Site Settings in the Quick Find box, then click Remote Site Settings.
Click New Remote Site ,
For the remote site name, enter = animals_http
For the remote site URL, enter = https://th-apex-http-callout.herokuapp.com

For the description, enter = Trailhead animal service: HTTP.
Click Save & New.
For the second remote site name, enter = animals_soap
For the remote site URL, enter = https://th-apex-soap-service.herokuapp.com
For the description, enter = Trailhead animal service: SOAP.
Click Save.

1.ParkService.apxc:

```
public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new
String[]{'return','http://parks.services/',null,'0','-1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'return_x'};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new
String[]{'arg0','http://parks.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'arg0'};
    }
    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/',
'ParkService'};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x = new
Map<String, ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
```

```
                '',
                'http://parks.services/',
                'byCountry',
                'http://parks.services/',
                'byCountryResponse',
                'ParkService.byCountryResponse'}
              );
              response_x = response_map_x.get('response_x');
              return response_x.return_x;
            }
          }
        }
```

2.ParkLocator.apxc:

```
public class ParkLocator {
    public static String[] country(String country){
        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
        String[] parksname = parks.byCountry(country);
        return parksname;
    }
}
```

3.ParkLocatorTest.apxc:

```
@isTest
private class ParkLocatorTest{
    @isTest
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String[] arrayOfParks = ParkLocator.country('India');

        System.assertEquals('Park1', arrayOfParks[0]);
    }
}
```

4.ParkServiceMock.apxc:

```
@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) {
      ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
      List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
      response_x.return_x = lstOfDummyParks;

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

APEX WEB SERVICES:

Create an Apex REST class that is accessible at /Accounts/<Account_ID>/contacts. The service will return the account's ID and name plus the ID and name of all contacts associated with the account. Write unit tests that achieve 100% code coverage for the class and run your Apex tests.

Prework: Be sure the Remote Sites from the first unit are set up.

- Create an Apex class
    - Name: `AccountManager`
    - Class must have a method called `getAccount`
    - Method must be annotated with **@HttpGet** and return an **Account** object
    - Method must return the **ID** and **Name** for the requested record and all associated contacts with their **ID** and **Name**
- Create unit tests
    - Unit tests must be in a separate Apex class called `AccountManagerTest`
    - Unit tests must cover all lines of code included in the **AccountManager** class, resulting in 100% code coverage
- Run your test class at least once (via **Run All** tests the Developer Console) before

attempting to verify this challenge

1.AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing 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;
   }
}
```

2.AccountManagerTest.apxc

```
@IsTest
private class AccountManagerTest{
   @isTest static void testAccountManager(){
      Id recordId = getTestAccountId();
      // Set up a test request
      RestRequest request = new RestRequest();
      request.requestUri =
         'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;

      // Call the method to test
      Account  acc = AccountManager.getAccount();

      // Verify results
      System.assert(acc != null);
   }

   private static Id getTestAccountId(){
      Account acc = new Account(Name = 'TestAcc2');
      Insert acc;
```

```
        Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
        Insert con;

        return acc.Id;
    }
}
```

APEX SPECIALIST SUPERBADGE:

AUTOMATE RECORD CREATION:

1.MaintenanceRequest.apxc :-

```
trigger MaintenanceRequest on Case (before update, after update) {

    Map<Id,Case> validCaseMap = new Map<Id,Case>();

    if(Trigger.isUpdate  && Trigger.isAfter){
        for(Case caseHere: Trigger.new){
            if (caseHere.IsClosed && (caseHere.Type.equals('Repair') ||
caseHere.Type.equals('Routine Maintenance'))){
                validCaseMap.put(caseHere.Id, caseHere);
            }
        }

        if(!validCaseMap.values().isEmpty()){
            MaintenanceRequestHelper.createNewRequest(validCaseMap);
        }
    }

}
```

2)MaintenanceRequestHelper.apxc:

```
public class MaintenanceRequestHelper {

    public static void createNewRequest(Map<Id, Case> validCaseMap){
        List<Case> newCases = new List<Case>();
        Map<Id, Integer> productMaintenanceCycleMap = new Map<Id, Integer>();
        Map<Id, Integer> workPartMaintenanceCycleMap = new Map<Id, Integer>();

                    for (Product2 productHere : [select Id, Maintenance_Cycle__c from Product2]) {
            if (productHere.Maintenance_Cycle__c != null) {
                productMaintenanceCycleMap.put(productHere.Id,
```

```
Integer.valueOf(productHere.Maintenance_Cycle__c));
        }
      }

      for (Work_Part__c workPart : [select Id, Equipment__c, Maintenance_Request__c from
Work_Part__c where Maintenance_Request__c in :validCaseMap.keySet()]) {
        if (workPart.Equipment__c != null) {
          if(!workPartMaintenanceCycleMap.containsKey(workPart.Maintenance_Request__c)){
            workPartMaintenanceCycleMap.put(workPart.Maintenance_Request__c,
productMaintenanceCycleMap.get(workPart.Equipment__c));
          }
          else if(productMaintenanceCycleMap.get(workPart.Equipment__c) <
workPartMaintenanceCycleMap.get(workPart.Maintenance_Request__c)){
            workPartMaintenanceCycleMap.put(workPart.Maintenance_Request__c,
productMaintenanceCycleMap.get(workPart.Equipment__c));
          }
        }
      }

      for(Case caseHere: validCaseMap.values()){
        Case newCase = new Case();
        newCase.Vehicle__c = caseHere.Vehicle__c;
        newCase.Equipment__c = caseHere.Equipment__c;
        newCase.Type = 'Routine Maintenance';
        newCase.Subject = String.isBlank(caseHere.Subject) ? 'Routine Maintenance Request' :
caseHere.Subject + ' New';
        newCase.Date_Reported__c = Date.today();
        newCase.Date_Due__c =
workPartMaintenanceCycleMap.containsKey(caseHere.Product__c) ?
Date.today().addDays(workPartMaintenanceCycleMap.get(caseHere.Product__c)) :
Date.today();
        newCase.Status = 'New';
        newCase.Product__c = caseHere.Product__c;
        newCase.AccountId = caseHere.AccountId;
        newCase.ContactId = caseHere.ContactId;
        newCase.AssetId = caseHere.AssetId;
        newCase.Origin = caseHere.Origin;
        newCase.Reason = caseHere.Reason;

        newCases.add(newCase);
      }
```

```
        if(newCases.size() > 0){
            insert newCases;
        }
    }

}
```

SYNCHRONIZATION SALESFORCE DATA WITH AN EXTERNAL SYSTEM:

1)WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService {

    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

    // complete this method to make the callout (using @future) to the
    // REST endpoint and update equipment on hand.
    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

        if (response.getStatusCode() == 200) {
            List<Object> results = (List<Object>) JSON.deserializeUntyped(response.getBody());
            List<Product2> equipmentList = new List<Product2>();

            for (Object record: results) {
                Map<String, Object> recordMap = (Map<String, Object>)record;
                Product2 equipment = new Product2();

                equipment.Name = (String)recordMap.get('name');
                equipment.Cost__c = (Decimal)recordMap.get('cost');
                equipment.ProductCode = (String)recordMap.get('_id');
                equipment.Current_Inventory__c = (Integer)recordMap.get('quantity');
                equipment.Maintenance_Cycle__c = (Integer)recordMap.get('maintenanceperiod');
                equipment.Replacement_Part__c = (Boolean)recordMap.get('replacement');
                equipment.Lifespan_Months__c = (Integer)recordMap.get('lifespan');
                equipment.Warehouse_SKU__c = (String)recordMap.get('sku');
```

```
            equipmentList.add(equipment);
        }

        if(equipmentList.size() > 0){
            upsert equipmentList;
        }
    }

  }



}
```

Steps to follow to complete the challenge:

- from Developer Console Menu : Debug => Open Execute Anonymous Window
- delete all previous codes
- execute this line of code : WarehouseCalloutService.runWarehouseEquipmentSync();
- Now check to pass the challenge (click 'Check Challenge')

SCHEDULE SYNCHRONIZATION USING APEX CODE:

1)WarehouseSyncSchedule.apxc

```
public class WarehouseSyncSchedule implements Schedulable{
  // implement scheduled code here
    public void execute(System.SchedulableContext context){
        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

Steps to follow to complete the challenge:

- from Developer Console Menu : Debug => Open Execute Anonymous Window
- delete all previous codes
- execute this line of code : System.schedule('WarehouseSyncScheduleTest', '0 0 1 * * ?', new WarehouseSyncSchedule());
- Now check to pass the challenge (click 'Check Challenge')

TEST AUTOMATION LOGIC:

1)MaintenanceRequestHelper.apxc

```
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);



            }
         }
      }

      if (!validIds.isEmpty()){
         List<Case> newCases = new List<Case>();
         Map<Id,Case> closedCasesM = 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>();
         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'));
      }

         for(Case cc : closedCasesM.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 (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> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c wpClone = wp.clone();
                wpClone.Maintenance_Request__c = nc.Id;
                ClonedWPs.add(wpClone);

            }
        }
        insert ClonedWPs;
    }
  }
}
```

2.MaintenanceRequestHelperTest.apxc :-

```
@istest
public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS_NEW = 'New';
    private static final string WORKING = 'Working';
    private static final string CLOSED = 'Closed';
    private static final string REPAIR = 'Repair';
    private static final string REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_TYPE = 'Routine Maintenance';
    private static final string REQUEST_SUBJECT = 'Testing subject';
```

```
    PRIVATE STATIC Vehicle__c createVehicle(){
        Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
        return Vehicle;
    }

    PRIVATE STATIC Product2 createEq(){
        product2 equipment = new product2(name = 'SuperEquipment',
                                lifespan_months__C = 10,
                                maintenance_cycle__C = 10,
                                replacement_part__c = true);
        return equipment;
    }

    PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cs = new case(Type=REPAIR,
                    Status=STATUS_NEW,
                    Origin=REQUEST_ORIGIN,
                    Subject=REQUEST_SUBJECT,
                    Equipment__c=equipmentId,
                    Vehicle__c=vehicleId);
        return cs;
    }

    PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id
requestId){
        Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
                                                Maintenance_Request__c = requestId);
        return wp;
    }


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

        Product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;
```

```
        case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
        insert somethingToUpdate;

        Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
        insert workP;

        test.startTest();
        somethingToUpdate.status = CLOSED;
        update somethingToUpdate;
        test.stopTest();

        Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,
Date_Due__c
                from case
                where status =:STATUS_NEW];

        Equipment_Maintenance_Item__c workPart = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c =:newReq.Id];

        system.assert(workPart != null);
        system.assert(newReq.Subject != null);
        system.assertEquals(newReq.Type, REQUEST_TYPE);
        SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
        SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
    }

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

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

        case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
```

```
        insert emptyReq;

        Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
        insert workP;

        test.startTest();
        emptyReq.Status = WORKING;
        update emptyReq;
        test.stopTest();

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

        Equipment_Maintenance_Item__c workPart = [select id
                                                  from Equipment_Maintenance_Item__c
                                                  where Maintenance_Request__c = :emptyReq.Id];

        system.assert(workPart != null);
        system.assert(allRequest.size() == 1);
    }

    @istest
    private static void testMaintenanceRequestBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
        list<case> requestList = new list<case>();
        list<id> oldRequestIds = new list<id>();

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

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

```
        for(integer i = 0; i < 300; i++){
            workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
        }
        insert workPartList;

        test.startTest();
        for(case req : requestList){
            req.Status = CLOSED;
            oldRequestIds.add(req.Id);
        }
        update requestList;
        test.stopTest();

        list<case> allRequests = [select id
                                  from case
                                  where status =: STATUS_NEW];

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

        system.assert(allRequests.size() == 300);
    }
}
```

3)MaintenanceRequest.apxt

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

TEST CALLOUT LOGIC:

1.WarehouseCalloutService.apxc :-

```
public with sharing class WarehouseCalloutService {

    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

    //@future(callout=true)
    public static void runWarehouseEquipmentSync(){
```

```
        Http http = new Http();
        HttpRequest request = new HttpRequest();

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


        List<Product2> warehouseEq = new List<Product2>();

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

            for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Decimal) mapJson.get('lifespan');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                warehouseEq.add(myEq);
            }

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

        }
    }
}
```

2)WarehouseCalloutServiceTest.apxc

```
@isTest
```

```
private class WarehouseCalloutServiceTest {
    @isTest
    static void testWareHouseCallout(){
        Test.startTest();
        // implement mock callout test here
        Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.runWarehouseEquipmentSync();
        Test.stopTest();
        System.assertEquals(1, [SELECT count() FROM Product2]);
    }
}
```

3.WarehouseCalloutServiceMock.apxc

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

        System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
        System.assertEquals('GET', request.getMethod());

        // Create a fake response
        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"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

TEST SCHEDULING LOGIC:

Go to the developer console use below code

1)WarehouseSyncSchedule.apxc

```
global class WarehouseSyncSchedule implements Schedulable {
    global void execute(SchedulableContext ctx) {

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
```

```
}
```

2)WarehouseSyncScheduleTest.apxc

```
@isTest
public class WarehouseSyncScheduleTest {

    @isTest static void WarehousescheduleTest(){
        String scheduleTime = '00 00 01 * * ?';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new
WarehouseSyncSchedule());
        Test.stopTest();
        //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on
UNIX systems.
        // This object is available in API version 17.0 and later.
        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');


    }
}
```

Formulas and Validations

1.Implement Roll-Up Summary Fields

Add a custom field to the standard account object that provides a rollup summary of the total expected revenue from all related opportunities.

- The rollup summary field should be labeled **Potential Value** and have the Field Name of **Potential_Value**. The resulting API name should be **Potential_Value__c**
- The rollup summary should calculate the total expected revenue of all the opportunities related to the account.

2.Create Validation Rules

To complete this challenge, add a validation rule which will block the saving of a new or updated contact if the contact is related to an account and has a mailing postal code (which has the API Name **MailingPostalCode**) different from the account's shipping

postal code (which has the API Name **ShippingPostalCode**).

- Name the validation rule **Contact must be in Account ZIP Code**
- A contact with a **MailingPostalCode** that has an account and does not match the associated Account **ShippingPostalCode** should return with a validation error and not be saved
- The validation rule should **ONLY** apply to contact records with an associated account. Contact records with no associated parent account can be added with any MailingPostalCode value.

Process Automation Specialist Superbadge:

1.Automate Leads

1.Search for Validation rule and create a new under Leads

2.Rule Name: Anything

3.Error Condition Formula :

OR(AND(LEN(State) > 2,

NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:MA:MI:MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:WI:WY", State))), NOT(OR(Country ="US",Country ="USA",Country ="United States", ISBLANK(Country))))

4.Create two Queues:

Search in quick box and select lead as object and create the below queues.

Queue Name: Rainbow Sales AND Assembly System Sales

5.Assignment Rule: Search from quick box and create a new.

Rule Name: Anything

Sort Order:2

Run this rule if the criteria are met

Lead:Lead Source not equal to Web

Queue-->Assembly System sales-->save

2.Automate Accounts

Create 4 Roll Up Summary fields as below:

Field 1: Label: Number of deals

Summary Type: COUNT

Summarized Object: Opportunity

Filter Criteria: None

Field 2: Label: Number of won deals

Summary Type: COUNT

Summarized Object: Opportunity

Filter Criteria: Stage EQUALS Closed Won

Field 3: Label: Last won deal date

Summary Type: MAX

Field to Aggregate: Opportunity: Close Date

Summarized Object: Opportunity

Filter Criteria: Stage EQUALS Closed Won

Field 4: Label: Amount of won deals

Summary Type: SUM

Field to Aggregate: Opportunity: Amount

Summarized Object: Opportunity

Filter Criteria: Stage EQUALS Closed Won

And 2 Formula Fields with below:

Field 5:Label: Deal win percent

Return Type: Percent

Decimal Places: 2

Formula: (Number_of_won_deals__c / Number_of_deals__c)

Field 6:Label: Call for Service

Return Type: Text

Formula: IF(DATE(YEAR(Last_won_deal_date__c)+2 ,
MONTH(Last_won_deal_date__c),DAY(Last_won_deal_date__c)) <= TODAY(), “Yes”,
“No”)

Create 2 validation rules as below

Validation Rule 1 : Rule Name : US_Address (Anything)

Error Condition Formula :

OR(AND(LEN(BillingState) > 2,
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:M
A:MI:MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:
VA:WA:WV:WI:WY", BillingState))
),AND(LEN(ShippingState) > 2,
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:M
A:MI:MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:
VA:WA:WV:WI:WY", ShippingState))
),NOT(OR(BillingCountry ="US",BillingCountry ="USA",BillingCountry ="United States",

ISBLANK(BillingCountry))),
NOT(OR(ShippingCountry ="US",ShippingCountry ="USA",ShippingCountry ="United States", ISBLANK(ShippingCountry))))

Error Message : You can not save a new account unless the shipping and billing state fields are valid US state abbreviations, and the country field is either blank or US, USA, or United States.

Error Location : Top Of Page

VALIDATION RULE 2 : Rule Name : Name Change

Error Condition Formula :

ISCHANGED(Name) && (OR(ISPICKVAL(Type ,'Customer - Direct') ,ISPICKVAL(Type ,'Customer - Channel')))
Copy
Error Message : You can't change the Account name for "Customer – Direct" or "Customer – Channel"

Error Location : Account Name

3.Create Robot Setup Object

Create a custom object Robot Setup with a Master-Detail relationship to the opportunity include Autonumber the record name, starting with 0 using name format: ROBOT SETUP-{0000}.

Use the following field names.

Date, Date__c : Date type

Notes, Notes__c : Text type

Day of the Week, Day_of_the_Week__c : Number

4.Create Sales Process and Validate Opportunities

Start by adding a field to Opportunity

Approval: Checkbox type

Ideally, the sales reps shouldn't be able to check that box and only system administrators like and sales managers should be able to check it. Though it doesn't throw an error for that condition.

Also, Click on the Opportunity field STAGE and add a picklist value as "Awaiting Approval"

Next, create a sales process under opportunities by searching the sales process in the Search box. Add the desired fields

Next add the Opportunity Validation Rule with error formula as below

IF((Amount > 100000 && Approved__c <> True && ISPICKVAL(StageName,'Closed Won')),True,False)

5.Automate Opportunities

Create Three Email Templates:

Finance: Account Creation,

SALES: Opportunity Needs Approval,

Sales: Opportunity Approval Status

Create related Email Alert from search box for the templates above.

Create an approval process:

Search for the approval process and select an opportunity object.

Criteria :

(Opportunity: Stage EQUALS Negotiation/Review) AND (Opportunity: Amount GREATER

THAN 100000)

SALES: Opportunity Needs Approval——->Template. Make sure to populate your manager as Nushi Davoud in Manage Users.

Create a process with the process builder

Opportunity object with option created and updated.

Node 1 Criteria.: Opportunity.Account Type = customer and Opportunity.account id not equal to null

Node 2 Criteria.: Opportunity.Account Type = Prospect, Opportunity stage = prospecting and Opportunity.account id not equal to null

Node 3 Criteria.: Opportunity Stage = Negotiation/Review and Opportunity Amount > 100,000

Node 4 Criteria.: Opportunity Stage = Closed Won

Creation.

Action for Node 2 :

Email Alert to mail notifies account creation : Finance: Account Creation.

Create a Record: Task with any name but mandatory subject line 'Send Marketing Materials'.

Make sure the string has no full stop or comma to it.

Assigned to the Account owner

Action for Node 3: Approvals

Choose the one we created for the opportunity here. And it takes care of the process thereby.

Action for Node 4: Record for Robot Setup

Set fields as below and Date formula being (closed date +180)

Make sure to inactive other processes in Processes to avoid interference and activate your process builder.

6.Create Flow for Opportunities

1.Create Flow named Product Quick Search

2.Element 1: Screen component from the palette

Name: Product Quick Search

Add Record Button from the Input as below:

Label: Product Type

Data Type: Text

Required: Check

Under Choices: Add new resource

Type: Choice

3.Create three choices as below for RainbowBot, CloudyBot, and Assemble Systems.

Element 2: Get Record as below

Label: Search Prod select object as Product.

Under Record Collection: Add New Resource Filterresult: Variable Type as below.

Add New Resource Loop : Variable Type as below.

Element 4: Assignment as below

Add New Resource Looptxt1 : Variable Type as below.

Element 5: Screen as below.

Save and Activate the flow.

4.Now search Lightning App Builder

5.Add New page: Select Record Type

Label: Product_Quick_Search

Object: Opportunity

6.Pick any template

7.And Drag and drop Flows from Left palette, select the flow we made and Save!

7.Automate Setups

1.Search for the field “Day of the Week” on robot object and change the field type from

Number to formula field of return type: text and use the below formula.
If you don’t find the formula field in the edit option of the field, you can delete and recreate the field with the same name as well.

2.Formula is :

```
Case ( WEEKDAY( Date__c ),
1,"Sunday",
2,"Monday",
3,"Tuesday",
4,"Wednesday",
5,"Thursday",
6,"Friday",
7,"Saturday",
Text(WEEKDay(Date__c)))
```

3.Go to the Process we created in step 5. Clone this Process. Go to action on the last node where we set up robo record. Change formula of date field from [Opportunity].CloseDate + 180..to.. below formula.