# Account 360: Salesforce Development (v1)

By: Nicholas Zozaya

## Background

This document details development comprising the Salesforce component of the Account 360 project, which spans several systems. The high-level objective of this first version of development is to enforce a governance process around the creation of Salesforce Accounts, with the ultimate goal of enabling Growth Ops awareness of (and ability to merge) duplicate Accounts. We will achieve this objective by (1) validating attempted Account inserts against existing Accounts via various matching parameters, and (2) creation of a process by which Growth Ops can merge any duplicate Accounts that weren't rejected. We'll begin by clarifying our matching logic before delving into specification of the Salesforce development, the latter being segmented by (1) Domain Layer Governance and (2) flow-level governance.

## Success Metrics

We'll evaluate the success of this v1 Account 360 implementation with Salesforce reporting on the following metrics:

1. Cases Resulting in Merged Accounts / Cases Resolved without a merge
    a. Every Case with a Status equal to 'Merged' is one less duplicate Account in the system; as such it is a metric indicative of the effectiveness of our matching algorithm
    b. We'll also want to know the proportion of Cases resulting in a merge against those that are not resulting in a merge to understand how effectively our matching algorithm detects duplicates.
2. Number of Leads converted into an existing Account
    a. Each incoming Lead that is converted into an existing Account is one less potential duplicate; as such it is a metric indicative of the effectiveness of our matching algorithm.
3. Case Resolution Time (see bottom right for avg resolution time)
    a. Currently, duplicate Accounts are merged on an ad-hoc basis in a manually-intensive process that involves searching for potential duplicates and a lot of wasted time. We'll track how long each Duplicate Management case takes to resolve and compare it to the time it currently takes (2 hrs per Case will be our conservative benchmark) to resolve duplicate Accounts.
4. Average number of Potential Duplicate Accounts per Duplicate Management Case
    a. An efficient matching algorithm should return a manageable number of potential duplicate Accounts for each Duplicate Management Case - we'll keep a measure of this count per Case to understand the efficacy of our matching algorithm.
5. User Behavior Tracking (Account Created on New Account Form open?)

## Matching Logic

In this v1 implementation, we will consider any **Unomy_Company_ID__c** matches to be *high-confidence* matches. Matches against any other criteria will be considered *low-confidence* matches.

1. **Unomy_Company_ID__c**
2. (**Website** OR **Phone**) AND (**Cleansed_Unomy_Company_Name__c** OR **Cleansed_Account_Name__c**)
3. **Website** AND (**Cleansed_Unomy_Company_Name__c** OR **Cleansed_Account_Name__c**)
4. (**Cleansed_Unomy_Company_Name__c** OR **Cleansed_Account_Name__c**) AND **Address**
5. (**Cleansed_Unomy_Company_Name__c** OR **Cleansed_Account_Name__c**) AND **Phone**
6. (**Website** OR **Phone**) AND **Name**
7. **Name**

Please see this doc by Enzigma for the current state of our matching logic (10/25/2019).

### Legacy Matching Metrics

### New Matching Metrics

## Matching Mechanism (P1)

We'll use LeanData's Matching API to find potential duplicates of incoming Accounts Contacts. This api takes an Object as an input, and returns an Array of potential matches in JSON which will be deserialized into Salesforce objects for further processing. We'll enhance an existing internal matching service to detect incoming duplicate Accounts previously built by Enzigma.

### Leandata Matching API Questions

1. The documentation states "Improve org-wide efficiency by utilizing LeanData's industry-leading **matching algorithm** to ensure the highest fidelity match..." in the **Matching API Overview** section. It also states, under the description for the **All Related Accounts** matching functionality "Returns all related Accounts (all accounts that matched via LeanData's Lead to Account Algorithm)". Is this algorithm configurable to an extent? We have specific parameters with which we'd like to distinguish "matching" Accounts.
2. In the only meeting I was a part of I saw a GUI matching rule configurator (not sure what the name of this application is), how does this configurator tie into the matching API if at all?
3. Can the matching API match more than one lead per request? The documentation contains examples of matching against a single *inputData* parameter - can the API handle more than one *inputData* parameter per invocation?
4. Can we configure confidence levels on the returned matches, or somehow understand at what tier the returned Accounts are considered duplicates so that we can consider this confidence-level when processing the API results?

### Leandata Matching API Assumptions

There will be a way for us to match against multiple Accounts with one api call, and to get back results that are segmented by the provided Accounts (i.e. if we feed the api 3 accounts and receive 9 matches, those 9 matches will be returned in 3 different arrays or maps, as opposed to simply getting back 9 matches all in the same array). We have since learned that we need to make one api call per Account/Contact we'd like to match.

# Domain Layer Governance (P1/P2)

We'll want to implement certain governance on every instance of some transactional contexts, like Lead Update, Account Insert and Account Update. The following sections specify the governance logic for each of these contexts.

## Before Lead Update (P1)

Any converted Lead should be matched against existing Accounts as a potential duplicate. If a high-confidence match is found, the Lead will be converted into the matched Account. If a low-confidence match is found, the Lead will be converted into a pending Account and a Duplicate Management case will be created for later review by Growth Ops.

1. Lead conversion attempt invokes LeadConverterHelper.cls.
    1. Invoke OrganizationHelper.cls matching, providing list of updated Leads as parameter.
    2. Process Results:
        1. For high-confidence matches:
            1. Convert Lead into the existing matched Account.
            2. Converted Contact and Opportunity will automatically be parented to this existing Account.
            3. Increment the matched Account's **Converted_Lead_Count__c** field +1.
            4. Self-Booked Tour governance processing ends for this particular Lead.
        2. For low-confidence:
            1. Set **Lead.Account_Duplicate_Status__c** == "Pending - Main".
            2. Set **Lead.Potential_Match_Ids__c** == comma-separated account Ids returned by matching service.
            3. Convert into new Account.
                1. Case will be created via Account Insert governance.
        3. For no-matches:
            1. If **Number_of_Full_Time_Employees__c** < FTE threshold for new "Main" Accounts:
                1. Allow conversion into new non-pending Main Account.
            2. If **Number_of_Full_Time_Employees__c >** FTE threshold for new "Main" Accounts:
                1. Set **Lead.Account_Duplicate_Status__c ==** "Pending - Main".
                2. Allow conversion into new pending Account.

## After Account insert (P1)

The objective of Account Insert governance is the creation of a Duplicate Management Case for incoming Accounts with **Duplicate_Status__c** = 'Pending Sub/Main'. An incoming Account's **Duplicate_Status__c** field will be mapped from the converted Lead's **Duplicate_Status__c** field (see Lead Governance).

1. Account Insert invokes *After Insert* Account trigger.
2. **TrAccountInsertPendingDupeActions** handler class fires.
    1. For each inserted Account:
        1. If **Account.Duplicate_Status__c.contains(**"Pending"**)**:
            1. Add Account to List<Account> variable.

2. Fire Queueable (async) class **Async_SetAcctInsertDupeMgtCase** with List<Account> constructor:
    1. Initialize Map<Case, List<Account>>.
    2. Initialize **Case** record:
        1. RecordType = 'Duplicate Management'
        2. Owner = Duplicate Management Queue
        3. Pending_Account__c = New Account Id
    3. For each Account in List<Account> constructor.
        1. Create List<String> out of **Account.Potential_Match_Ids__c**.
        2. For each Id in new List<String>:
            1. Initialize Account with **Id** as parameter.
            2. Add initialized Account to a new List<Account>.
        3. Add in-loop Account to List<Account>.
        4. Put initialized Case, List<Account> in Map<Case, List<Account>>.
    4. Insert new List<Case>{Map<Case, List<Account>>.keyset()}.
    5. For each Case in Map<Case, List<Account>>.keyset():
        1. Initialize new List<Account>.
        2. For each Account in Map<Case, List<Account>>.get(key):
            1. Set **Account.Duplicate_Management_Case__c** = key.Id;
            2. Add updated Account to new List<Account>.
        3. Put(key, new List<Account>)
    6. Update Map<Case, List<Account>>.values().

## After Account Update

Governance logic around Account updates will regulate (1) updates to the **Account.Unomy_Company_ID__c** by the Unomy API, and (2) updates to the **Account.Account_Type__c** field. The objective of this governance is to ensure creation of Growth Ops Cases for the former situation, and proper hierarchical placement of Accounts in the latter situation (i.e. an "Org" Account changing to a "Sales" Account has the correct parent).

1. Account update invokes *After Update* Account trigger.
2. **TrAccountUpdatePendingDupeActions** handler class fires.
    1. For each updated Account:
        1. If (**Account.Unomy_Company_ID__c** has been updated AND **Account.Unomy_Company_ID__c** != NULL) or **Account.Account_Type__c** has been updated:
            1. Initialize empty Map<Id, Account>.
            2. Add Account to Map<Id, Account>.
    2. For each entry in Map<Id, Account>
        1. If **Account.Unomy_Company_ID__c** updated:
            1. Construct Map<Account.Unomy_Company_ID__c, List<Account>>
            2. Construct Map<Account.Unomy_Company_ID__c, Account.Id>.
            3. For **Account.Unomy_Company_ID__c** change:
                1. Add entry to Map<Account.Unomy_Company_ID__c, Account.Id>.

4. Loop through each Account **WHERE** Account.Unomy_Company_ID__c **IN** Map<Unomy_Company_ID__c, Account>.keyset() **AND** Id **NOT IN** Map<Unomy_Company_ID__c, Account.Id>.values().
    1. Create Map<Account.Unomy_Company_ID__c, List<Account>> entry.
    2. Add TriggerNew.get(Map<Account.Unomy_Company_ID__c, Account.Id>.get(this.Account.Unomy_Company_ID__c)) entry to Map<Account.Unomy_Company_ID__c, List<Account>>.
5. Fire Queueable (async) class **Async_SetAcctUpdateDupeMgtCase**:
    1. Construct empty List<Case>, List<Account>, Map<Case, List<Account>>.
    2. For each entry in Map<Account.Unomy_Company_ID__c, List<Account>> parameter:
        1. Initialize Case:
            1. RecordType = ‘Duplicate Management’
            2. Owner = Duplicate Management Queue
            3. Pending_Account__c = Map<Account.Unomy_Company_ID__c, List<Account>> entry.value() with most recent LastModifiedDate.
        2. Add entry to Map<Case, List<Account>, using new Case as key and in-loop Accounts as key value.
    3. Insert new List<Case>{Map<Case, List<Account>.keyset()}.
    4. For each Case in Map<Case, List<Account>.keyset():
        1. Construct new List<Account>.
        2. For each Account in Map<Case, List<Account>>.get(this.Case):
            1. Set **Account**.**Duplicate_Management_Case__c** == this.Case.Id.
            2. Add updated Account to new List<Account>.
    5. Put this.Case, new List<Account> in Map<Case, List<Account>>.
    6. Update Map<Case, List<Account>>.values().

2. If **Account_Type__c** updated:
    1. Initialize Map<Id, Account>, Map<Case, List<Account>>.
    2. If **Account_Type__c** == “Org” AND PRIORVALUE(**Account_Type__c**) == “Sales”:
        1. If Account is in same hierarchy:
            1. Allow update.
        2. If Account is in different hierarchy:
            1. Add entry to Map<Id, Account>.
            2. Initialize Case
                1. RecordType = ‘Duplicate Management’
                2. Owner = Duplicate Management Queue
                3. Pending_Account__c = this.Account.Id
            3. Put Case, new List<Account> in Map<Case, List<Account>>.
        3. Invoke matching API, providing Map<Id, Account>.values() as a parameter.
        4. Process results:
            1. For returned matches:
                1. For each returned Account:

1. Add entry to List<Account> key value of Map<Case, List<Account>>.
2. Add Map<Case, List<Account>> entry.
3. Fire Queueable (async) class **Async_SetAcctInsertDupeMgtCase**:
    1. Insert new List{Map<Case, List<Account>>.keyset()}.
    2. For each Case in Map<Case, List<Account>>.keyset():
        1. Instantiate new List<Account>.
        2. For each Account in Map<Case, List<Account>>.get(this.Case):
        3. Set **Account.Duplicate_Management_Case__c** = this.Case.Id.
        4. Add Account to new List<Account>.
        5. Put this.Case, List<Account>.
    3. Update Map<Case, List<Account>>.values().

# Flow Governance (P1/P2)

## Self-Booked Tour

This flow begins when a User books a Tour through WeWork.com. After the tour is booked, Salesforce (1) creates a Lead/Journey for the User, (2) requests Lead enrichment from Unomy via the Unomy API, then (3) converts the Lead into an Account, Contact and Opportunity.

1. Before Lead Update governance.
    1. This will result in the Lead either (1) being converted into an existing Account, or (2) being converted, thus invoking Account Insert trigger.
2. After Account Insert governance.
    1. Will only occur if the incoming Lead was (1) matched at a low-confidence level or (2) was not matched with any existing Account.
    2. System will create a Case linking the new “Pending” Account to its potential duplicates.
    3. Growth Ops will review the Case to analyze if this pending Account needs to be merged.

## Journey Conversion (P1)

This flow starts after a user clicks the **Manage/Book Tours**, **Add Opportunity** or **Convert/Handoff** button on the Journey record page.

1. If **Journey__c.Primary_Lead__c** != NULL:
    1. ManageTours.cmp **init** class will do the following:
        1. Invoke LeanData matching API on initialization to get potential matching Contacts, providing Journey__c.Primary_Lead__c data as the *inputData* parameter. (*AllDuplicateContacts*).
            1. Parse List<Contact> results and store them in aura attributes.
        2. When User clicks **Schedule Tour** button, System will invoke **OrganizationHelper.cls** matching service:

1. For high-confidence matches:
    1. Account Selector returns queried Account as the only option to continue the process with.
    2. After confirming Account selection:
        1. If Contacts returned via LeanData API:
            1. Surface returned Contacts in a table for User as merge Lead options.
        2. If no Contacts returned via LeanData API:
            1. Proceed with Lead conversion into new Contact.
    3. Convert Lead into high-confidence Account and new/selected Contact.
2. For low-confidence Account matches:
    1. Account Selector returns the low-confidence Account matches as options to continue the process with.
    2. User can:
        1. Continue with a low-confidence Account match by:
            1. clicking **Create Account under selected**.
                1. System will show a form pre-populated with the prospective Account's details:
                    1. **Journey__c.Primary_Lead__c** data.
                    2. **ParentId** = selected low-confidence Account.Id
                    3. Main Account = ParentId.UltimateParent
                2. After confirming Account details, User should click **Continue**:
                    1. Present on screen message to user : " When you click 'Continue' a new temporary Sub Account will be created.
                3. If Contacts were returned via LeanData API:
                    1. Surface returned Contacts in a table for User as merge Lead options.
                    2. User should click **Continue.**
                4. If no Contacts returned via LeanData API:
                    1. Proceed with Lead conversion into new Contact.
                5. If **FTE <** threshold: (Threshold = 50)
                    1. Proceeed to (7).
                6. If **FTE** > threshold:
                    1. Set **Journey__c.Primary_Lead__r.Account_Duplicate_Status__c** == "Pending - Sub-Account".
                    2. Set **Journey__c.Primary_Lead__r.Potential_Match_Ids__c** == comma-separated account Ids returned by **OrganizationHelper.cls**.
                    3. Update **Journey__c.Primary_Lead__c**.
                7. When user clicks **Continue** after choosing Contact, or if matching API didn't return any Contacts:

1. Convert the Lead into a new/selected Contact and new Account.
2. Find Account Where ParentId = selected low-confidence account's Id.
3. Update returned Account's ParentId = Newly Created Account's Id.
4. After Account Insert governance processing begins.

8. Return user to regular Manage/Book Tours flow with new/matched Account selected.

2. click **Create Account above selected**
    1. System will show a form pre-populated with the prospective Account's details:
        1. **Journey__c.Primary_Lead__c** data.
        2. **ParentId** = selected low-confidence Account.Id
        3. Main Account = ParentId.UltimateParent
    2. After confirming Account details, User should click **Continue**:
        1. Present on screen message to user : " When you click 'Continue' a new temporary Main Account will be created."
    3. If Contacts were returned via LeanData API:
        1. Surface returned Contacts in a table for User as merge Lead options.
        2. User should click **Continue.**
    4. If no Contacts returned via LeanData API:
        1. Proceed with Lead conversion into new Contact.
    5. If **FTE <** threshold:
        1. Proceeed to (7).
    6. If **FTE** > threshold:
        1. Set **Journey__c.Primary_Lead__r.Account_Duplicate_Status__c** == "Pending - Sub-Account".
        2. Set **Journey__c.Primary_Lead__r.Potential_Match_Ids__c** == comma-separated account Ids returned by **OrganizationHelper.cls**.
        3. Update **Journey__c.Primary_Lead__c**.
    7. When user clicks **Continue** after choosing Contact, or if matching API didn't return any Contacts:
        1. Convert the Lead into a new/selected Contact and new Account.
        2. Update the selected low-confidence Account's ParentId to the newly created Account's Id.

3. After Account Insert governance processing begins.
8. Return user to Manage/Book Tours flow with new Account selected.

2. Ignore the returned matches in lieu of searching an existing Account or creating a new Account.

3. If no exact match, allow user to search/create account.
    1. Return low-confidence matches in the Account Selector table.
    2. User can:
        1. Search new Account - no governance needed.
        2. Create new Account:
            1. User will be directed to existing Org Creation form pre-populated with Journey__c.Primary_Lead__c data.
            2. After giving information, user will click **Continue**.
            3. If FTE < threshold:
                1. Convert Lead into created Account.
                2. Return user to Manage/Book Tours flow with new Account selected.
            4. If FTE > threshold:
                1. Present on screen message to user : " When you click 'Continue' a new temporary Main Account will be created."
                2. User clicks **Continue**.
                3. System will:
                    1. Set **Journey__c.Primary_Lead__r.Account_Duplicate_Status__c** == "Pending - Main".
                    2. Set **Journey__c.Primary_Lead__r.Potential_Match_Ids__c** == comma-separated account Ids returned by matching API.
                    3. Update **Journey__c.Primary_Lead__c**.
                    4. Convert Lead into new Account with Type__c = "Org".
                    5. After Account Insert governance processing begins.
                    6. Return user to Manage/Book Tours flow with new Account selected.

## Automated Account Bulk Upload

## Manual Account Bulk Upload

This flow begins when a User bulk-uploads Account records via the Journey Importer tool, Dataloader or any other client allowing the insert of bulk accounts. Manual Account Bulk Upload governance will aim to (1) reject incoming Accounts with same Unomy Id as existing Accounts before insert, or (b) flag potential matches with Cases for later

Growth Ops review. Governance processing will allow partial success of the insert, which means that one Account rejection will not reject the entire import.

**Before Account Insert**

1. For each Account in Trigger.New:
    1. If **Account.Unomy_Company_Id__c** != NULL:
        1. Put Account in Map<Unomy_Company_Id__c, Account>.
    2. Query Account table WHERE Unomy_Company_Id__c IN: Map<Unomy_Company_Id__c, Account>.keyset():
        1. For each returned Account:
            1. If Map<Unomy_Company_Id__c, Account>.containsKey(this.Account.Unomy_Company_Id__c):
                1. Add Map<Unomy_Company_Id__c, Account>.get(this.Account.Unomy_Company_Id__c) to List<Account> for rejection.
2. Invoke matching API, providing non-rejected List<Account> as a parameter.
3. If low-confidence match found:
    1. If Account.Account_Type__c = “Sales”:
        1. Set incoming **Account.Duplicate_Status__c** = “Pending - Sub-Account”.
    2. If Account.Account_Type__c = “Org”:
        1. Set incoming **Account.Duplicate_Status__c** = “Pending - Main”.
4. If no match found:
    1. If FTE > threshold:
        1. If Account.Account_Type__c = “Sales”:
            1. Set incoming **Account.Duplicate_Status__c** = “Pending - Sub-Account”.
        2. If Account.Account_Type__c = “Org”:
            1. Set incoming **Account.Duplicate_Status__c** = “Pending - Main”.
5. See Account Insert Governance for steps after Account Insert.

## Manual Single Account Creation (P1)

This flow will start whenever a User attempts to create an Account from the **Account** tab, or by using the **Create/Request New Org** global action.

1. System will surface a form with 4 required input fields:
    1. Name
    2. Website
    3. Phone
    4. FTEs
2. After User clicks **Continue**:
    ~~1. Invoke Unomy API to return a single match with Unomy Id.~~
    ~~2. If Unomy API returns single match:~~
        ~~1. Search Account table for Account where Unomy_Company_Id__c = returned Unomy Account Id.~~
        ~~2. Suggest this as Account for user instead of creation of new Account.~~

~~3. If no Unomy match:~~

1. Invoke Matching Service with form fields as input parameter.
2. If returned matches:
    1. Present matches in radio-select table for user to choose instead of creating a new Account.
    2. Each row should have buttons:
        1. **Create Account Above**
        2. **Create Account Below**
3. User can:
    1. Choose a matched Account.
        1. User will be navigated to the chosen Account.
    2. Create an Account above a matched Account.
        1. User will be directed to existing Org Creation page.
        2. If FTE > threshold:
            1. Set incoming **Account.Duplicate_Status__c** = “Pending - Main”.
            2. Present on screen message to user : " When you click 'Continue' a new temporary Main Account will be created.”
            3. User clicks **Continue**.
            4. See Account Insert Governance for next steps.
    3. Create Account below a matched Account.
        1. User will be directed to existing Org Creation page.
        2. If FTE > threshold:
            1. Set incoming **Account.Duplicate_Status__c** = “Pending - Sub-Account”.
            2. Present on screen message to user : " When you click 'Continue' a new temporary Sub Account will be created.”
            3. User clicks **Continue**.
            4. See Account Insert Governance for next steps.
    4. Continue to Search Accounts.
        1. User will be taken to current Account Selector page.
    5. Continue to Create a new Account.
        1. User will be directed to existing Org Creation page.
        2. If FTE > threshold:
            1. Set incoming **Account.Duplicate_Status__c** = “Pending - Sub-Account”.

## Growth Ops Account Merge Flow (P1)

Growth Ops will use Cases to analyze potential Account duplicates. See Account Insert Governance for how these Cases will be created. Each Case will have a related list of potential duplicates of the “Pending” Account, which will be found on the **Case.Pending_Account__c** field. Growth Ops will able to find these Cases via a **Duplicate Management Cases** list view. When a Case is set to “Resolved”, Salesforce will automatically clear the Account’s **Duplicate_Status__c** field, which will remove the alert at the top of the Account record page.

1. When Case.Status = “Resolved”

1. Workflow rule will set Pending_Account__r.Duplicate_Status__c == “”.

# Related Resources

PRD

# Development Tasks

## Apex

1. **TrLeadConversionDuplicateCheck** (Lead Update Trigger Handler) **(P1)**
2. **TrAccountInsertPendingDupeActions** (Account Insert Trigger Handler) **(P1)**
3. **TrAccountUpdatePendingDupeActions** (Account Update Trigger Handler)
4. **Async_SetAcctInsertDupeMgtCase** (Queueable Apex) **(P1)**
5. **Async_SetAcctUpdateDupeMgtCase** (Queueable Apex)

## Objects

Account **(P1)**

1. Fields
    1. **Duplicate Status** (Duplicate_Status__c) (Picklist)
        1. “Pending - Main”
        2. “Pending - Sub”
    2. **Duplicate Management Case** (Duplicate_Management_Case__c) (Lookup - Case)
    3. **Potential Match Ids** (Potential_Match_Ids__c) (Text Area (Long))
    4. **Converted Lead Count** (Converted_Lead_Count__c) (Integer)
    5. Change **Unomy_Company_ID__c** to external Id

Lead **(P1)**

1. Fields
    1. **Account Duplicate Status** (Account_Duplicate_Status__c) (Picklist)
        1. “Pending - Main”
        2. “Pending - Sub”
    2. **Potential Match Ids** (Potential_Match_Ids) (Text Area (Long))
    3. Change **Unomy_Company_ID__c** to external Id
2. Field Mappings
    1. Lead.Account_Duplicate_Status__c **=>** Account.Duplicate_Status__c

Case **(P1)**

1. Fields
    1. **Pending Account** (Pending_Account__c) (Lookup - Account)
2. Record Type
    1. Name = “Duplicate Management”

2. Status Values
    1. Open
    2. Merged
    3. Resolved
3. Assignment
    1. **Growth Ops Admin**

Custom Metadata **(P1)**

1. Create new MetadataSetting__mdt record:
    1. Name = Account_FTE_Threshold
    2. Data = {“Limit” : x}

### Data (P1)

1. Create new Queue.
    1. Name = “Duplicate Management”
    2. Members = TBD

## Dev. Questions

1. What is our best option for duplicate detection? The document proposes using LeanData's matching API within our current Account creation flows; we should come to agreement on this being the best solution.
    1. We need to validate what matches LeanData is returning in a sandbox, do at least some comparison to other tools.
    2. Talk to Mike about DemandTools and its matching capabilities.
2. Our current matching logic isn't comprehensive, but I would argue that this is okay for a v1 implementation. As long as we make the logic itself modular, we can always adjust later - we can discuss if this is correct conclusion.
    1. We need to back this up with some sandbox testing.
3. What is the difference between (1) pinging Unomy's API for potential Account matches and (2) querying Salesforce C.I. Company records? What is the C.I. Company object for?
    1. Unomy only gives back one match.
    2. Don’t use CI companies.
4. We'll be revising the Journey Conversion flows to do governance processing (**Manage/Book Tour**, **Add Opportunity** and **Convert/Handoff** flows). I have the following questions on this flow:
    1. Which Apex Class facilitates Lead conversion in these flows? We'll need to convert Journey__c.Primary_Lead__c into specific Accounts in some situations, or we'll have to set certain fields on this Lead before allowing it to convert.
    2. In the current flow a User can choose to create a new Account out of the converted Journey - what happens to Journey__c.Primary_Lead__c in this situation? Do we create the new Account then convert the Lead into the new Account?
        1. Leads are always converted.
        2. Account is created first.
        3. Lead is converted into New Account.

5. Regarding the governance of Bulk Account Insert, is it possible for us to restrict bulk import of Accounts to some custom tool like the Journey Importer, so that we can implement some governance processing within that tool before Account Insert? Does Journey Importer allow the import of Accounts, and can we add code to the Journey Importer that will allow us to run governance on the .csv that is being imported?
    1. Will have to use dataloader / data import wizard and deal with accounts during the insert transaction.
6. How many ways can users manually create single Accounts? We'll need to override every method of manual Account creation with a custom flow.
    1. Global Action
    2. Account tab insert
    3. Manage/Book Tour && Add Opportunity Journey Conversion