Temporal

Jumpstart Reference Application

Onboarding Use Case

Overview

This Use Case generalizes a process commonly found in Applications that need to introduce some “Entity” that has a distinct identity in your Domain. This “Entity” which results from the Onboarding process can be a User, an Employee, a Customer, a Company, etc. Temporal’s Durable Execution is vital to maintain the integrity of this transaction to avoid partial configuration of such Entities.

An “Onboarding” commonly describes a multi-step process which happens over a period of time due to integration with other services and/or human interaction. While the process is performing such tasks, it typically needs to coordinate values across these boundaries but this state is considered transient. The final Entity produced at the end is the “goal” of all this work. It is most often expected to be completed within a certain amount of time before some form of compensation (eg “rollback”) is necessary to prevent incomplete, partial Onboarding processes.

The outcome of an Onboarding can be a simple Database record with accumulated state, or as complex as multiple APIs being made aware of the Entity’s identity.

Problems Commonly Encountered:

Partial Configuration

If you have had a User signup for your Application but some of the services or data stores failed to register them, you know the problems that caused went on to use your product.
For example, when the User submitted their information while your Customer Relationship Management software was down - but your application did not remove all the data it had stored up to that point. The User was left “partially configured”.

Duplicated Entities

It is easy to miss lookups in some services or data stores while storing submitted information, leading to duplication of the User in your services. Your Application is now corrupted and it is even harder to know

which is the legitimate referenced User across all services so you have to look at Timestamps to help decide which to remove.

Accumulated Configuration

It's easy to send a giant blob of data into a service that represents the entire configuration of a User across all your services. It's very difficult to pull the missing configuration from contributing APIs, manual intervention, etc. All of this could take some time so it is easy to end up in the partial or duplicated Entity states mentioned above when pulling data from various sources to produce a cohesive User, Customer or other representation.

Product Requirements, v1

Personas

There are two personas that interact with our Use Case:

- **User**: Submits the Entity Onboarding request with details. The Entity is subject to the Application **Owner** approval.
- **Owner**: Approves or Rejects the Entity Request submitted by the **User**

Onboarding Journey

The User should be able to introduce a generic "Entity" into our Application based on an input "value". This Entity MUST have its identity provided by the User and the "value" becomes the basis for other operations we perform while onboarding.
Each Entity MUST be approved by the Application Owner. Application Owners can visit the Entity Onboarding page to Approve or Reject Entity submissions. Right now, there will not be any other affordance for Owners to know which Entities are awaiting approval. When an Entity is approved or rejected, the "value" cannot be changed.
Our agents use the Entity Relationship Management System (ERM) dashboards to view approved Entities, so we need to update that system during this Onboarding process.

Therefore, Onboarded Entities MUST:

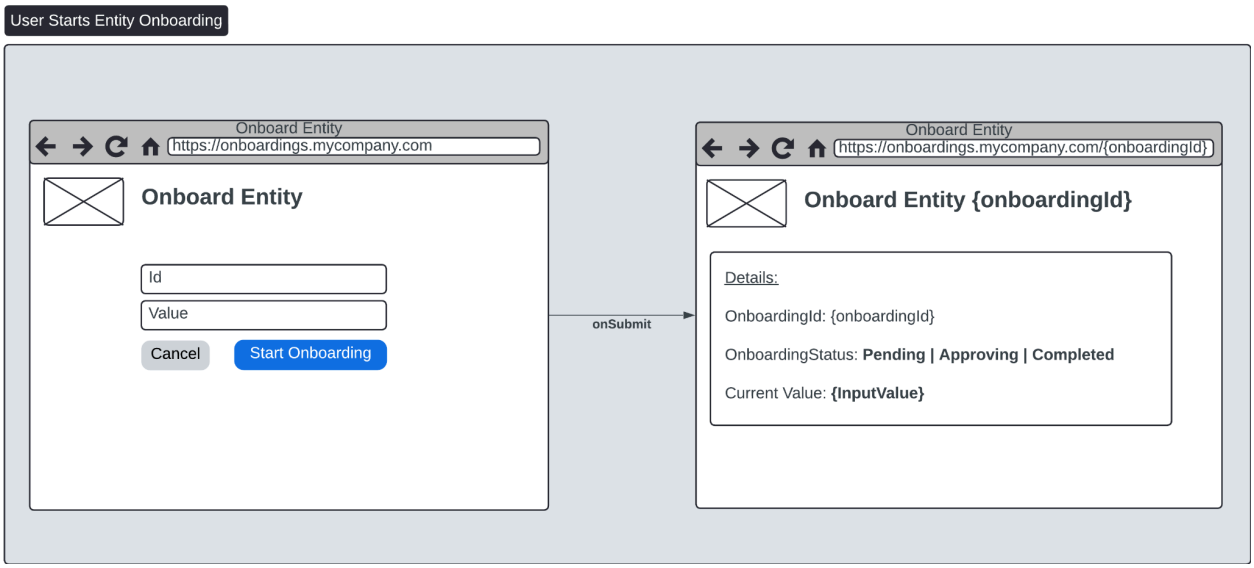
1. Be uniquely identified. The provided Id must not already be used in our Application.
2. Be discoverable via a search interface for their originating Id and Value attributes
3. Be approved by an Owner for inclusion in our Entity Management System

   a. If the Owner has not approved within ***three*** days, we should notify a Deputy Owner that the Entity is awaiting approval
4. Be onboarded successfully within ***seven*** days of submission. If this condition cannot be met:
   a. We cannot tolerate partial representation of the Entity in other systems so undo as necessary
   b. Allow the same identifier to be used later
   c. We want the Entity to be considered **Failed**
5. Allow cancellation prior to their approval, after which it cannot be canceled.
6. Allow mutations to the “value” until approval or rejection.
7. Expose their current state to external users as they are being onboarded.

Onboarding CAN take some time, subject to Approval submission and/or API response times.
The Entity WILL be represented by these various states while onboarding:

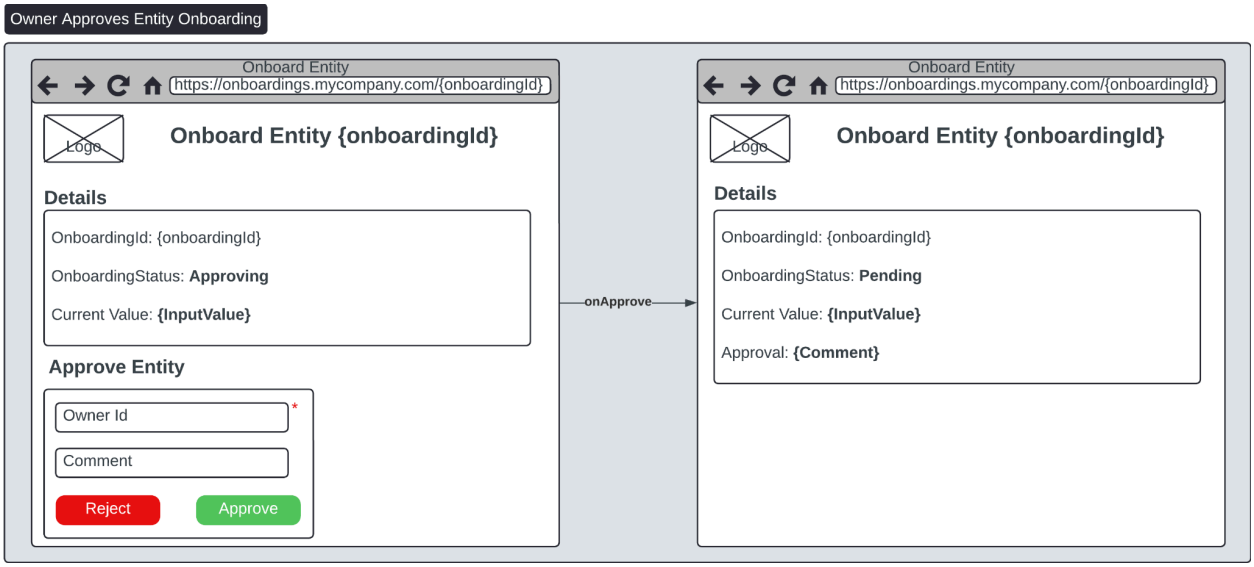
1. **Pending**: The onboarding is awaiting Approval for integrating the Entity into various Application systems
2. **Completed**: The Entity has successfully been onboarded
3. **Canceled**: The onboarding was Canceled and the Entity has not been onboarded
   a. // TODO we may not address this in the Curriculum for v1
4. **Rejected**: The Entity was rejected and cannot be onboarded RIGHT NOW.
   a. // TODO this will be v2 . Use **Canceled** for now.
5. **Failed**: The Entity could not be onboarded

Start Entity Onboarding



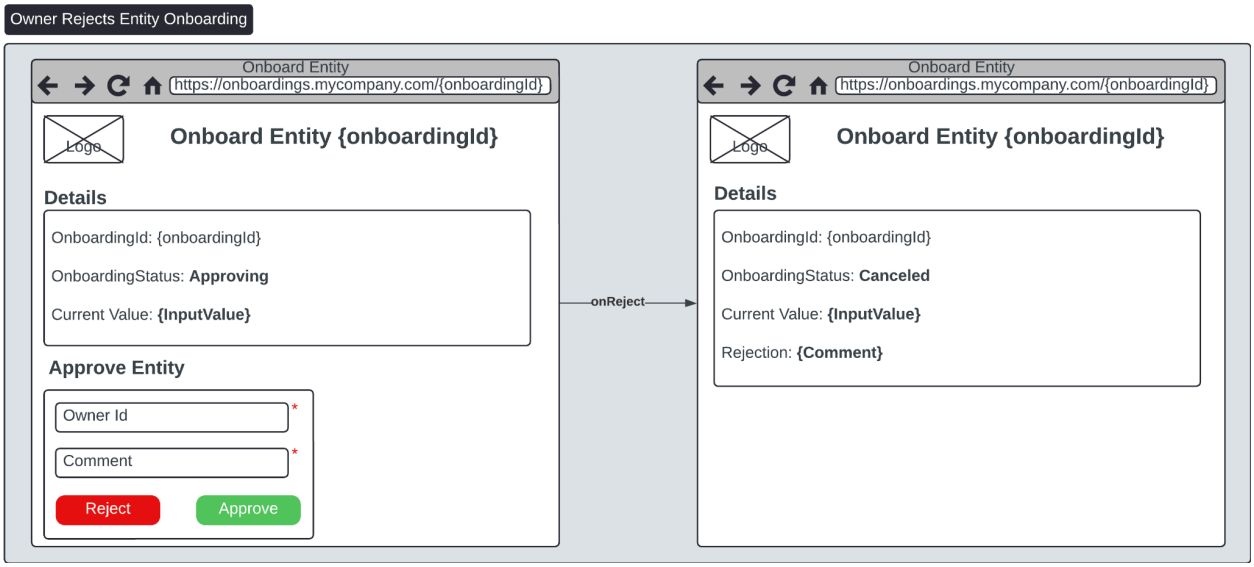
1. User navigates to our “Onboard Entity” application page.
2. User provides the following properties to be used as properties for the Entity being onboarded
    a. `Id`: The Entity Id being created in our application
    b. `Value`: The string value our application should use as the basis for other values in the Entity
3. Upon clicking “Start Onboarding”, user is directed to a “Details” page that shows
    a. The identifier the user provided
    b. The current value, based upon the user’s original Value
    c. The onboarding status of the Entity (eg **Pending**)

Owner Approves Entity



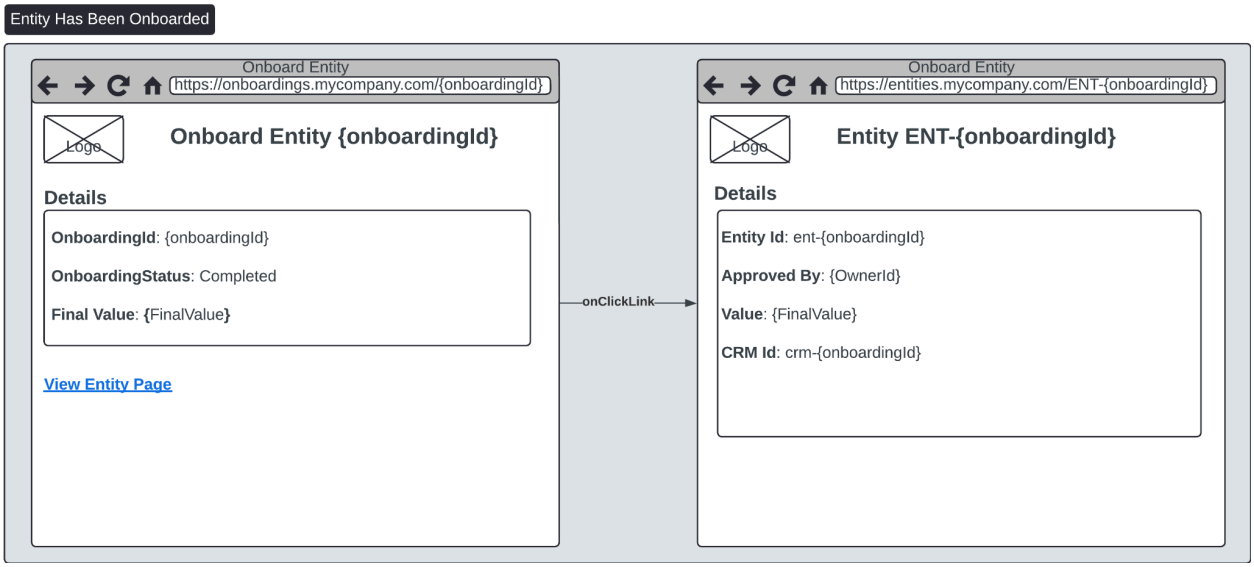
1. User who is signed in as an “Owner” visits the Entity Onboarding status page
2. Owner enters the “Owner Id”, an [Optional] comment, and clicks “Approve”
3. The Owner is directed to the Entity Onboarding page and sees that the Onboarding status is now at “Pending”

Owner Rejects Entity



1. User who is signed in as an “Owner” visits the Entity Onboarding status page
2. Owner enters the “Owner Id”, a *Required* comment, and clicks “Reject”
3. The Owner is directed to the Entity Onboarding page and sees that the Onboarding status is now at “Canceled” with the Comment on why it was rejected.

Entity Onboarding Completed



When an Entity has successfully completed Onboarding, Users or Owners may still visit the Onboarding page, but should also see an affordance for easily navigating to the details of the created Entity.

Technical Requirements, v1

These requirements are decisions based on the Product Requirements (PRD) above. They seek to map these product expectations to implementation details in our Application.

Entity Failure Due to Rejection or Timeout

We elected to explicitly **fail** the Workflow Execution when the Entity was not Approved in time.
This means:

1. An Temporal **ApplicationError** will be raised
2. The query **Approval** will show as **Pending** or **Rejected**
3. The `workflow_failed` metric will be incremented by **one**
    a. This could skew our analytics but we elect to have the UX of seeing **Failed** workflows clearly represented in our Temporal UI without using Search Attributes

Two items were taken into consideration:

1. The **Failed** status will only be impactful on **Starters** while the Failed Workflow falls within the Namespace Retention Policy. In other words, failures will not prevent Entities from being onboarded in the future with the same Id beyond this Namespace Retention Policy.
    a. A future implementation could do a look up in this Workflow to fast-fail if the Entity already exists
2. The `workflow_failed` metric admittedly mixes a failure of business concerns with execution concerns but this is okay for now. In the future we might expose the timeouts or rejections as explicit metrics.

Notification Service Protection

We aren't sure if the Service we use for our notifications dedupes attempted emails and are concerned about failures flooding a user's inbox. We decided to try only *two times* to send an email before moving on using Retry Options.

Recursive Execution Using Continue-As-New

The logic for approval is very similar regardless of the presence of a **DeputyOwnerEmail**, so we have chosen to prune input arguments and use the *Continue As New* API to perform another Workflow Run. This should reduce conditional branches in the definition body and make it easier to follow.

Product Requirements, v2

Scenario:

We shipped to production *v1* of **Onboard Entity** four weeks ago.
It has been successful but Users have asked to receive a Notification when their Onboarding is completed instead of being required to check the `/onboardings/{id}` page for status. Product has submitted the following requirement changes and new user experience views.

Change Order

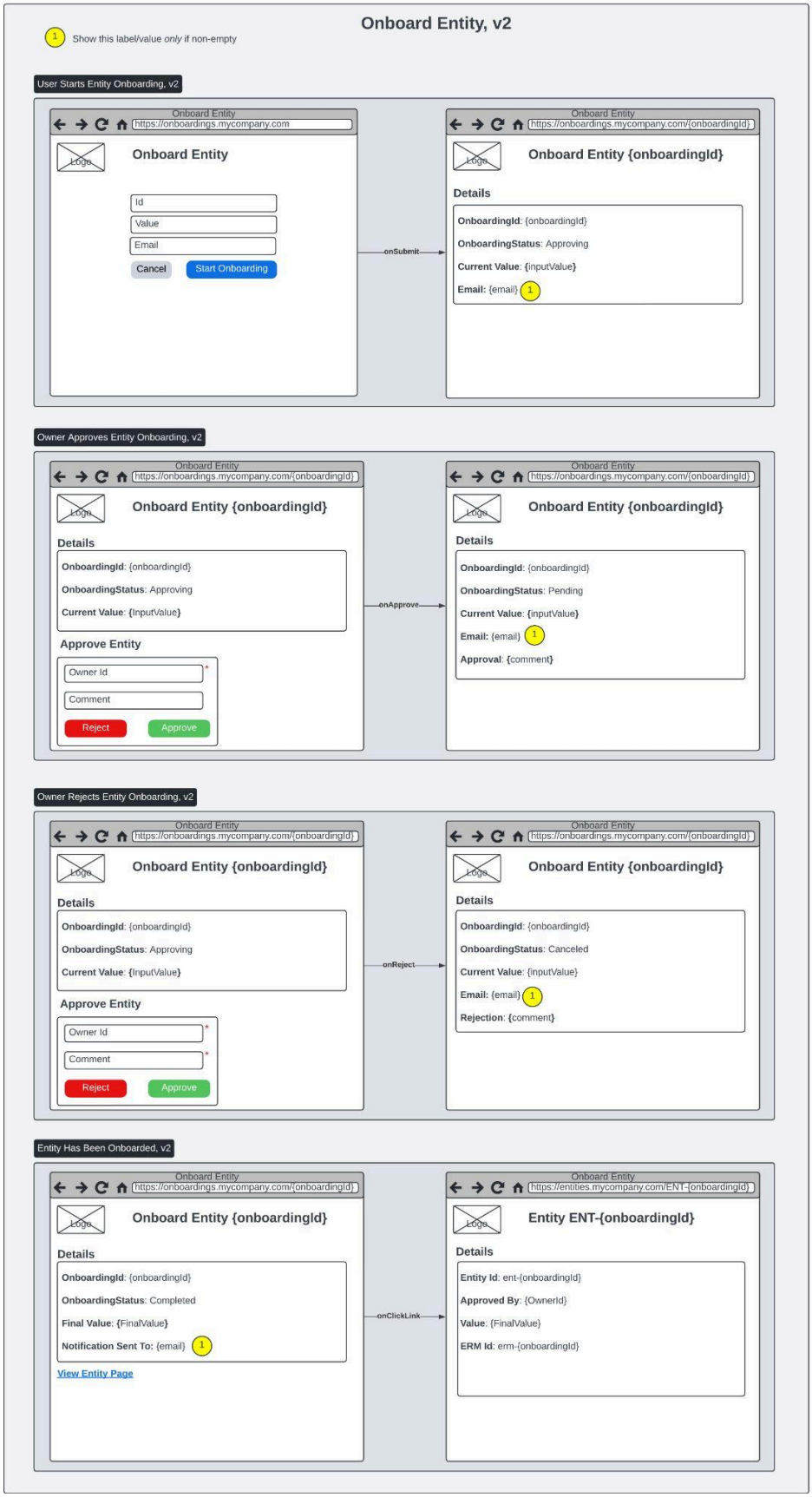
1. Accept an [optional] **email** field when submitting an Onboard Entity request from the User
2. If **email** has been provided, send the User an email notification when:
    a. they have been **Approved** : include instructions on how to access the new **Entity** page.
    b. they are **Rejected** : do not include the Comment about their rejection.
    c. *Approval* was not completed in time: include instructions on how to retry the Onboarding

Only new Onboarding requests will support this notification feature. **Onboardings** that are not in terminal state will not be retrofitted to accept a notification email.

Designs are shared on the next page which should be straightforward enough to turn into technical requirements.

Temporal

Updated User Interfaces



Technical Requirements, v2

Packaged Versions

We have opted for the **patched** version strategy but do not have the ability to store history JSON externally to perform Replay tests.
This means we need to keep previous implementations in our source control so we can produce Event Histories by the previous Workflow implementations to pass into the proposed `latest` implementation.
We see the directory structure looking something like this:

```
Unset
- latest
    - myworkflow.go
- v1
    - myworkflow.go
- v2
    - myworkflow.go # the version just before "latest"
...
```

This gives us a "normal" git diff experience if we just update the same file (**myworkflow.go**) but we still get to do Replay tests without dealing with Data Converter.

API Support

// TODO