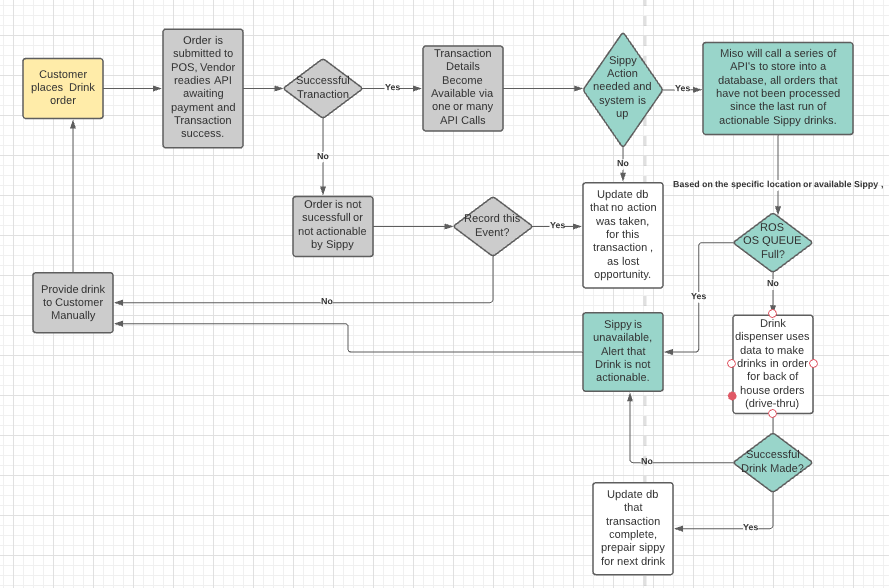
In General, here is the Journey Map as we imagine the high level steps.



[https://lucid.app/lucidchart/c973b67d-0e3e-464b-a3d9-8b4923567beb/view?page=0\_0&invitationId=inv\_35ff1205-bc61-4f6f-b594-23803b8c9a4d#](https://lucid.app/lucidchart/c973b67d-0e3e-464b-a3d9-8b4923567beb/view?page=0_0&invitationId=inv_35ff1205-bc61-4f6f-b594-23803b8c9a4d)

Here is the sequence we believe we will encounter as we ingest rest api based data of NCR/Aloha.

We will need to be provided with API access keys, user accounts, developer information necessary to document each rest api internally and convert that dataset into a subject area to be database driven. We forsee many api definitions, so each API call will have it’s data attributes stored in a database. We can anticipate a MISO API CATALOG where we will have metadata available, each API maintaining both version , history and dependency information. Each Api will have an order of execution and apis will consume data that are immediately synched with a database tables. Joins of these tables will serve to maintain a dashboard for execution summary. Individual asynchronous queues will serve as the first in first out for drink requests and order fulfillment. We anticipate the below as the flow for authorization to obtain access to the API’s.

Well need to determine the systems interactions, what each API’s function is => get, put, post, batch and delete rest api, questions to follow are :

* What order do API’s go in?
* What will other developers need from your API to integrate?
* How can you make it as simple as possible for them to access?
* How will errors be handled?
* What reporting is needed and how do we determine the life of an individual message , from creation to fully executed dispensed drink.

This envision a process that will be database and data driven where the RESTAPI’s can be created , have the data normalized , cleansed and made available to internal systems via the same nocode auto created frameworks. We will work together to identify how to translate the API’s into database schemas.

Each API will have a table or tables where normalize that data that is used to synchronize itself with database in near real time, via pre-created scheduled jobs. Foreign keys will be created to synchronize the dependent data in the web api. A formal job queue will keep critical processes on schedule and alter if any issues arrise. The benefits this offers is having a realtime visual flow of what our API are doing in real time or waiting on.

**The below is outside the scope of the Journey Map but is made available for here reference purposes.**

Using traditional database modeling tools is a common and standard way to visualize complex software structures. I have model the metadata of proposed API. Think of the model as a blueprint of how the data is inter-related. The goal it to automate the registration of new API’s, keep certain critical subject areas in sync in near realtime as possible and maintain a status of each job invocation so that if a job fails , it can be rerun accordingly.

In addition to the above referenced diagrams, an additional set of diagram types organized into two groups: structural diagrams and behavioral (or interaction) diagrams are open for introduction if the need arrises.

### Structural diagrams

* **Class diagram:** This diagram represents the static structure of a system. It shows relationships between classes, objects, attributes, and operations.
* **Component diagram:** A component diagram displays the relationships of software components, including source code, run-time code, and executables.
* **Deployment diagram:** These diagrams represent the hardware and software resources in your system. A deployment diagram is useful for mapping how a software solution will be deployed across multiple machines and platforms.
* **Object diagram:** These show the relationship between objects using real-world examples. They display a representation of what a system looks like at any given time. These diagrams can be used to test class diagrams for accuracy.
* **Package diagram:** A package organizes elements into groups. A package diagram shows the dependencies among different packages.
* **Profile diagram:** These diagrams describe profiles that allow adaptation of the UML metamodel for different platforms and domains.
* **Composite structure diagram:** This diagram displays the internal structure of a class.

### Behavioral diagrams

* **Use case diagram:** These diagrams model the functionality of a system and the actors who are involved in the system and its functionality. Actors are real-world examples such as customers, agents, and companies.
* **Activity diagram:** Activity diagrams are used to diagram the workflow from one activity to the next.
* **State machine diagram:** Similar to activity diagrams, state machine diagrams describe the behavior of objects that act differently according to their current state.
* **Sequence diagram:** Like the name implies, sequence diagrams map the sequence of when and how objects interact with each other.
* **Communication diagram:** These diagrams map the flow of messages passed between objects.
* **Interaction overview diagram:** This type of diagram shows the sequence of a collection of interaction diagrams.
* **Timing diagram:** These diagrams represent the behavior of objects within a specified time frame.

**Access to the restapi application poc that I am advocating for is for internal use only.**

Table

Description automatically generated

Table

Description automatically generated

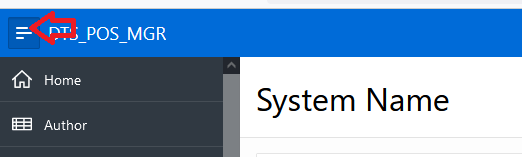
A picture containing table

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

**For example system name table can be accessed by clicking on top blue bar**



From there the entities are exposed and can be access for select, insert, update or delete. Views will help join that data accordingly.

Since the code for the rest api’s will be created from this metadata, the system will be able to support multiple parallel development efforts and the

Company will have full visibility to every concern or dependency rest api based.