Here is the sequence we propose when we provide access to client rest api requests of our data either internally or externally. As we ingest restapi based data of NCR/Aloha, we will abide by the same sequence for authorization.

We will need to be provided with API access keys, user accounts, developer information necessary to document each restapi internally and convert that dataset into a subject area to be database driven. We forsee many definition documents, coming from many individuals from many companies for many initiatives. This will quickly get out of hand if we do not have a centralized authority to manage a MISO API CATALOG. The goal is to establish a single process that can track a specific REST API’s definitions across time. This will allow us to have a single place where change can be introduced and tracked for all of their dependencies, across Miso. Since each definition has data dependencies this lends itself wonderfully to use our nocode, lowcode frameworks that will automatically take the restapi definition and write the code to perform the consumption of that data, the serialization of that data back to backend tables and the placement of the next data to source into one or many queues so that the latest greatest data that needs to be consumed downstream is ready to go.

API Flowchart (Click on image to modify online)

## What is an API?

An API is a set of routines, protocols, and tools developers use to build software applications and to specify how software components should interact. In simple terms, an API is a messenger that sends a request to a provider you want data from and then delivers the response back to you. Think of an API as an intermediary that allows two programs to talk to each other.

## Why create an API diagram?

Well need to determine the systems interactions, what it’s function is for => get, put, post, batch and delete rest api data and what data it will need to process. A API is very much like a jig saw puzzle, there will be many fragments of data we will have to glue together

* What calls are you using for each API?
* What order do they 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?

Answering these questions as thoroughly as possible will give us a better idea of what you/we want to accomplish with the API and how you want to develop it.

I propose that the restapi definitions be converted into database schemas so that we can synchronize a scheduled rest api call with. That the database be used for most tasks to include dashboards for the API’s being calling in progress, business process creation and activity tracking, api job status, analytics , storage and downstream integrations.

Diagram

Description automatically generated

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 that is used to synchronize itself with database in near real time, via a pre-created scheduler job. 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 want our API are doing in real time or waiting on.

A database will make clear all data involved by creating a visual model of your API so all involved can understand what happens when successful calls and responses are made, and when errors are returned.

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. A individual IPA my be just a small fragment of the whole system but if its missing or misinterpreted, the system breaks. 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 is a job fails , it can be rerun accordingly.

From the below pics and working POC metadata system, the proposed relationships between objects, and their attributes can be discussed.

In addition to the above referenced diagrams, an additional 14 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 not provided for internal use only.**

Table

Description automatically generated

Table

Description automatically generated

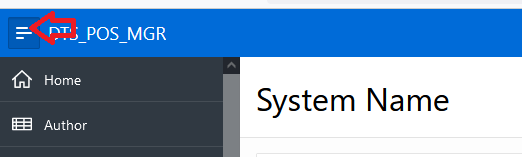
A picture containing table

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.

I am proposing the below as an example of this rest integration by nocode by performing exactly that.

The specific instructions are not attached to this document, they are included accordingly.