Mojaloop GitHub repository management proposal

Version: 0.2 draft

Authors: Pedro Sousa Barreto (pedrob@crosslaketech.com)

Revision Log

Date	Who	What	Version
12-Mar-2021	pedrob@crosslaketech.com	Released for comments	0.1
6-Aug-2021	Ref arch team	Updated repo naming schemas	0.2

Intro & problem statement

This document was created with the intention to propose a solution along with guidelines, to the following problems:

- Growing complexity of managing Mojaloop's GitHub Organization;
- Growing number of repositories, contributed and core;
- Increasing complexity of deploying Mojaloop platform, due to its multiple dependencies.

As stated in issue 1879:

> *"As a contributor/developer can we minimize the number of repos so it is easier to understand the core components of the hub*
>
> *I want to **develop a plan** to lessen the number of repos **so that** the mojaloop project is easier to maintain and deploy."*

Issue reference: https://github.com/mojaloop/project/issues/1879

Proposal

Top level hierarchy repository organisation

Today, Mojaloop's code is arranged in multiple repositories grouped inside a single GitHub Organization.

In respect to top level hierarchy, the options discussed were:

- A single GitHub Organization with all repositories;
- Multiple GitHub Organizations.

Regarding the option to implement multiple GitHub Organizations, one The recommendation is to use a single Git Organisation

Having a single org with multiple repos is preferable, it has the following advantages:

- Single mojaloop org means brand doesn't erode (not risks of finding the wrong organization);

- Easier to manage and secure, no need to sync permissions for admins, or other privileged members, like reviewers;

It is recommended to keep a single organization and simply reduce the number of active (not archived) repositories.

Classifying different types of repositories

We can continue using the same label logic, however most people ignore labels when forming opinions as to which repositories matter. The suggestion is that labels can be used to tag types of repositories for Mojaloop's internal purposes, not as a mechanism to facilitate repository identification by potential contributors.

This is a proposal for the types of repositories:

- Service - a deployable service;
- Library - a library that one or more Mojaloop repositories depend on;
- SDK - a library to be used by external components to connect to Mojaloop;
- Tool - a tool that provides independent functionality, can be IaC;
- UI - a user interface;
- Documentation - documentation project, a single one for

There should be an explicit way to mark official from non-official repositories. This could be a mandatory header in all repositories README.md files, clearly stating the status and the type of the repository:

- Official Mojaloop - Service
- Official Mojaloop - Library
- Official Mojaloop - Core Tool
- Official Mojaloop - Core UI

Repository names

It is important to have a naming scheme as consistent as possible, these are the recommendations:

- ~~All official repositories should have the prefix "mojaloop-", or the non-official have the "contrib-" prefix.~~ Only the "contrib" rule applies, everything not contrib should be official and not require a prefix.

- We should only include words like server, API or adapter when the component is actually what we're calling it.
- If it provides an external API it should include "-api" in the name.
- If it is a service suffix it in "-svc"
- If it is a library suffix it in "-lib"
- If it is a project repo for Zenhub, suffix it with "-project"
- If it is a documentation repo, suffix it with "-doc"
-
-

The reason to include only the word "api" in the repo name for services that provide an external API is to be explicit about the intent to serve external requests. External developers will connect to Mojaloop's external APIs, and some other internal services will provide some form of an API for inter-service consumption, we shouldn't add "api" to their names.

Another good example of misleading naming is the "api-adapter" suffix. Software components are either an Application Programming Interfaces (API) or adapters, but not both at the same time.
In our case we have *"ml-api-adapter"* and *"thirdparty-api-adapter"*, both are just APIs services or Web APIs, and should be called "ml-api-svc" and *"thirdparty-api-svc"*, respectively.

For reference, see here what an adapter is:
https://en.wikipedia.org/wiki/Adapter_pattern

The recommendation is not to go on a renaming spree, only to rename those few repositories that benefit from a rename, and start applying the naming scheme for future repositories.

Proposed clean-up

On the 12th of March of 2021, Mojaloop's Github organization repository had the following statistics:

- Total of 126 repositories, 107 are public, 19 are private;
- 23 repositories are archived;
- Only ~35% have been updated in the last month (~46 of 126);
- Have no updated since
- 2 are forks (with no activity)

Proposal:

- Make all experimental repositories private, if the repository cannot be made private, then maybe it is not experimental anymore;
- All non-pinned repositories with no activity in the last 3 months, should be archived;
- All repositories should have a clear status statement on the top of the readme.md (see above Classifying different types of repositories)

Team & User organisation

Mojaloop can take advantage of the **nested teams** feature that GitHub provides, this will facilitate access management for repositories while keeping a manageable number of top level teams.

Neste teams automatically inherit repository permissions but not members. This allows us to set top level teams with general access and have that access applied to all members that are neste below, while keeping the flexibility of arranging the nested teams and map them to specific repositories.

Proposed structure:

- Admins (rename ml-admins)
- ChangeControlBoard (rename board-ccb)
- DesignAuthority (rename board-da)
- TechnicalGovernanceBoard (rename board-tgb)
- Maintainers (rename ml-maintainers)
- Reviewers (new)
- Contributors (new)
    - Feature team 1
    - Feature team 2
    - etc...

The main benefit of the proposed structure is that, by reducing the number of top level teams, it is easier to manage top level access and membership, while maintaining the ability to manage team membership.

A fundamental change is the nesting of all current initiative, project or feature teams, below the *Contributors* team, all these teams will inherit the contributor permissions by default, and specific repository permissions can be managed at

feature team level. We can see that the structure of the "*feature-ml-dev*" team was set up like this, but that is not applied all around.

Maintainers - same as today

The purpose of the new top level team *"Reviewers"*, is to easily group all contributors that are willing, or have been appointed, to do code reviews for core Mojaloop code (on top of *"Maintainers"*). This will enable the usage of @mentions for the whole team in comments, and also the request of code reviews to the whole team.

Main github page:

- explaining repo structure/logic
- rules to create new repos (booted

Clear Text on top of readme showing if a repo is official or contributed
….

User organisation and Teams

Merge strategy

Core Repos

- Work is done in separate feature branches, PR is requested
- Repo owners are the only ones able to merge to main

Community Repos

Access management

Recurring maintenance

- Archive and remove contrib (non-official) repos with no activity in the last X months

- Promote to official the repos that might need to be promoted