

:years: 2015-2022

# Principles

## RESTful API

The platform exposes all its functionality via a practically-RESTful API, that communicates using JSON.

We use the term practically-RESTful in order to make it clear we are not trying to be fully REST compliant but still maintain important RESTful attributes like:

- ¥ Stateless: platform maintains no conversational or session-based state. The result of this is ability to scale horizontally with ease.
- ¥ Resource-oriented: API is focussed around set of resources using HTTP vocabulary and conventions e.g GET, PUT, POST, DELETE, HTTP status codes. This results in a simple and consistent API for clients.

See online API Documentation for more detail.

## Multi-tenanted

The Fineract platform has been developed with support for multi-tenancy at the core of its design. This means that it is just as easy to use the platform for Software-as-a-Service (SaaS) type offerings as it is for local installations.

The platform uses an approach that isolates an FI's data per database/schema (See Separate Databases and Shared Database, Separate Schemas).

## Extensible

Whilst each tenant will have a set of core tables, the platform tables can be extended in different ways for each tenant through the use of Data tables functionality.

## Command Query Separation

We separate commands (that change data) from queries (that read data).

Why? There are numerous reasons for choosing this approach which at present is not an attempt at full blown CQRS. The main advantages at present are:

- ¥ State changing commands are persisted providing an audit of all state changes.
- ¥ Used to support a general approach to maker-checker.

¥ State changing commands use the Object-Oriented paradigm (and hence ORM) whilst queries can stay in the data paradigm.

## Maker-Checker

Also known as four-eyes principal. Enables apps to support a maker-checker style workflow process. Commands that pass validation will be persisted. Maker-checker can be enabled/disabled at fine-grained level for any state changing API. Fine grained access control

A fine grained permission is associated with each API. Administrators have fine grained control over what roles or users have access to.

## Package Structure

The intention is for platform code to be packaged in a vertical slice way (as opposed to layers). Source code starts from <https://github.com/apache/fineract/tree/develop/fineract-provider/src/main/java/org/apache/fineract>

¥ accounting

¥ useradministration

¥ infrastructure

¥ portfolio

! charge

! client

! fund

! loanaccount

¥ accounting

Within each vertical slice is some common packaging structure:

¥ api - XXXApiResource.java - REST api implementation files

¥ handler - XXXCommandHandler.java - specific handlers invoked

¥ service - contains read + write services for functional area

¥ domain - OO concepts for the functional area

¥ data - Data concepts for the area

¥ serialization - ability to convert from/to API JSON for functional area