Fineract CN Project Structure

Fineract CN is spread into roughly 30 projects. This is an overview of the various categories of projects, their generic structure, and what role each plays in the Fineract CN ecosystem.

Integration Tests

Integration tests run multiple services together to check that their inter-dependencies work as intended. Integration tests are named for the services they include.

https://github.com/apache/fineract-cn-integration-tests

demo-server

demo-server is a project which uses the same methodology as the integration tests to deploy **all** the services together except the UI. Demo-server provisions the services in a generic way. This is a reasonable place to start in playing with deployment. You will need to understand this provisioning code to be able to provision in your production environment.

It is unlikely that you will be able to build a production environment based solely on demo-server. The services are designed for a cloud deployment in which services run in containers and scaling up is achieved by spinning up new instances. Demo-server runs one instance of each service, plus an embedded version of Eureka, PostgreSQL, Cassandra, and ActiveMQ locally.

Because of the number of processes involved and the resources required, startup of all the services in the demo-server can take up to 20 minutes. Older computers may "choke" on all those processes. I strongly recommend a computer with plenty of memory and with an SSD drive.

https://github.com/apache/fineract-cn-demo-server

test-accounting-portfolio

Tests the interaction between accounting and portfolio. Because token validation requires a valid public key from identity, the test harness for this test is more sophisticated than for integration tests which contain provisioner and identity.

test-provisioner-identity-organization

Mainly tests the interaction between provisioner and identity. The "organization" in the name stands for "office", which is in there too, but it's secondary.

test-provisioner-identity-rhythm-portfolio-accounting

Tests the interaction between the named services. This test is currently portfolio-centric, since it is portfolio which requires all the other services.

service-starter

A library which starts services, Eureka, and Apache ActiveMQ for use in the integration tests and in demo-server.

https://github.com/apache/fineract-cn-service-starter

default-setup

A library which contains example setup information. Currently only contains a standard chart of accounts for the accounting service. Should in the future contain roles for identity, and other useful standard data for the rest of the services for a generic initial setup.

https://github.com/apache/fineract-cn-default-setup (credit union basics)

Deployment

Docker scripts

https://github.com/apache/fineract-cn-docker-compose

This library contains also collection of example Postman scripts (REST requests) to provision a tenant.

UI

fims-web-app

Based on AngularJS. Depends on all the services (except group) to provide back office functionality.

https://github.com/apache/fineract-cn-fims-web-app

fims-e2e

Protractor e2e testing for fims-web-app.

https://github.com/apache/fineract-cn-fims-e2e

fineract-cn mobile

https://github.com/apache/fineract-cn-mobile

Services

Services rely on libraries and sometimes each other. They are built as spring boot deployable jars. Some service projects include libraries intended to support other services in functionality closely related to their own. We started off giving the services the names of egyption gods. These names are still visible in the data structures for the services. So I've included their "code" names here.

template

Though it is deployable, this service is not meant for deployment, but rather as an example of best practices in creating a service. When we create a new service we copy this project and then follow its instructions to specialize it. The template, like most services produces the following three build artifacts: Sometimes new best practices discovered while working on other services may take a while to migrate back to template.

https://github.com/apache/fineract-cn-template

service

This is the spring boot deployable. When running it registers with a Eureka server, connects with a PostgreSQL and a Cassandra database, and provides a REST interface, with endpoints grouped into permittable groups. When initialize is called for a tenant within the service, the service allocates tables and other resources specifically for that tenant. GET calls are implemented against the PostgreSQL using spring data jpa, and PUT, POST, and DELETE calls cause cassandra-persisted commands to be placed on command bus and processed asynchronously into the PostgreSQL DB.

api

This is a Feign client to make it easier for java clients to communicate with the service. The api artifact contains constants defined for listening to ActiveMQ events emitted by the service. The api artifact also contains constants naming the permittable groups which define the permissions which the service offers. The api artifact contains any domain object validation code, and the unit tests for that validation code.

component-test

Tests in component-test start this service and **only** this service to test the functionality offered by it. Tests make their calls into the service via the Feign API. Tests are based on a combination of Mockito, JUnit, and Spring. Dependencies to other services are simulated via mocking, but no other mocking is done in the component-tests.

provisioner

Code name: seshat

Use the provisioner to create tenants and assign services to those tenants. The provisioner is responsible for allocating databases, tracking which tenant has access to which applications, and taking care of the "magic" which allows applications to provision their own resources and declare which permissions are available for the application.

https://github.com/apache/fineract-cn-provisioner

identity

Code name: isis

Domain: users, permissions, roles

Identity authenticates users with passwords, and provides bearer JWT tokens via which other services can check a request's authentication and authorization. Identity can also authenticate services and check if a service has the "right" to act in a user's name.

https://github.com/apache/fineract-cn-identity

rhythm

Code name: khepri

Domain: timing of tasks

Provides services with the ability to request notification at a certain time. Useful for daily interest calculation and other offline jobs that should be done once and only once.

https://github.com/apache/fineract-cn-rhythm

accounting

Code name: thoth

Domain: general ledger, journal entries, trial balance, bank reconciliation

Keeps track of accounts, ledgers, and journal entries.

https://github.com/apache/fineract-cn-accounting

office

Code name: horus

Domain: offices, employees

Employees should have a 1:1 relationship with users in the identity service.

https://github.com/apache/fineract-cn-office

customer

Code name: maat

Domain: customers, identities, user defined fields

https://github.com/apache/fineract-cn-customer

group

Code name: ptah

Domain: groups, meetings

Intended for use in group loans, but has not yet been used from any other service or from the UI. Incomplete.

https://github.com/apache/fineract-cn-group

https://github.com/apache/fineract-cn-group-finance

deposit-account-management

Code name: shed

Domain: deposit accounts, fees, interest calculations, line of credit, payment strategies

Tracks deposit products and deposit accounts for customers.

https://github.com/apache/fineract-cn-deposit-account-management

portfolio

Code name: bastet

Domain: loans, fees, interest calculations, repayment strategies

https://github.com/apache/fineract-cn-portfolio

teller

Code name: tajet

Provides management of teller cash. Supports a customer-service-centric view of payment and repayment for deposit, and portfolio.

https://github.com/apache/fineract-cn-teller

reporting

Code name: shu

Domain: Generates partial and complete balance Sheets and Income Statement

Provides the ability to generate simple reports based on predefined report definitions.

https://github.com/apache/fineract-cn-reporting

notification

Provides the ability to send SMS and Email notification/messages to customers.

https://github.com/apache/fineract-cn-notifications

cheques

https://github.com/apache/fineract-cn-cheques

Stellar bridge

https://github.com/apache/fineract-cn-stellar-bridge

Libraries

Libraries are not deployable pieces of code. They are intended to support the common functionality of the services. Most library projects produce one build artifact and contain unit tests for their classes.

anubis

Services which consume JWT tokens created by identity use anubis to check them. Anubis uses spring security to manage authentication information, and publishes the permissions defined by the programmer for the service. Anubis also contains library support for testing services which use anubis for their security.

https://github.com/apache/fineract-cn-anubis

permitted-feign-client

Services which make requests to other services can use permitted-feign-client to transparently specify which requests and to request impersonation rights to specific endpoints for a user.

https://github.com/apache/fineract-cn-permitted-feign-client

test

Support for unit tests and component tests in the services

https://github.com/apache/fineract-cn-test

api

Each of the services has a Feign application programming interface (api). This project contains code intended to support the creation of a Feign API.

https://github.com/apache/fineract-cn-api

command

Services are all implemented using CQRS. Command provides library support for asynchronous processing and audit-able NoSQL persistence of POST, PUT, and DELETE requests. The non-central, eventually-consistent persistence approach is central to our disaster recovery strategy.

https://github.com/apache/fineract-cn-command

postgresqlDB and data persistence

Most (but not all) services use an SQL persistence for querying of data via GET requests. This library provides tenant-separation based on header contents.

https://github.com/apache/fineract-cn-postgresql

https://github.com/apache/fineract-cn-data-jpa

cassandra

All services save commands to the NoSQL DB Apache Cassandra. Some save other data there as well. This library provides tenant-separation based on header contents.

https://github.com/apache/fineract-cn-cassandra

async

Handles correct multi-threading for command handling.

https://github.com/apache/fineract-cn-async

crypto

Hashes passwords.

https://github.com/apache/fineract-cn-crypto

lang

Miscellaneous. If you can't figure out where to put it, put it here. But first consider creating a new library project.

https://github.com/apache/fineract-cn-lang