Functional Enhancements to Fineract CN Mobile

About me

Name: Snehil

University: Siddaganga Institute of technology, Tumkuru

Bachelor: Computer Science and Engineering, BE, Third Year

Email: snehil101@gmail.com (connect with me on linkedin)

Github : snehilrx

Telephone: +91 7022616211

Country of Residence: India

Timezone: IST (GMT + 0530)

Primary Language: English

Programming Languages: Java, Kotlin, CPP, JavaScript

Framework / API worked on:
Android, Spring, React, JUnit, Mokito, Dagger, Retrofit, RxJava, Hilt

Background

Apache Fineract is open-source software that aims to innovative mobile and cloud-based solutions and enables digital transaction accounts for all. Fineract 1.x is a banking platform with open APIs. Fineract CN takes a different approach. It looked at the forks that were occurring on the active Fineract1.x project and sought to reduce that behavior by creating a composable release. The architecture is microservices and builds on cloud-native technologies like cloud spring, Apache Kafka. The Fineract CN Mobile is the official mobile client for the new Fineract CN. It aims to provide customers with MFIs, credit unions, cooperatives, savings groups, agent banking, etc. This app will have different versions to fulfill the specific needs of its customers.

The tasks that are to perform during the GSoC project are:-

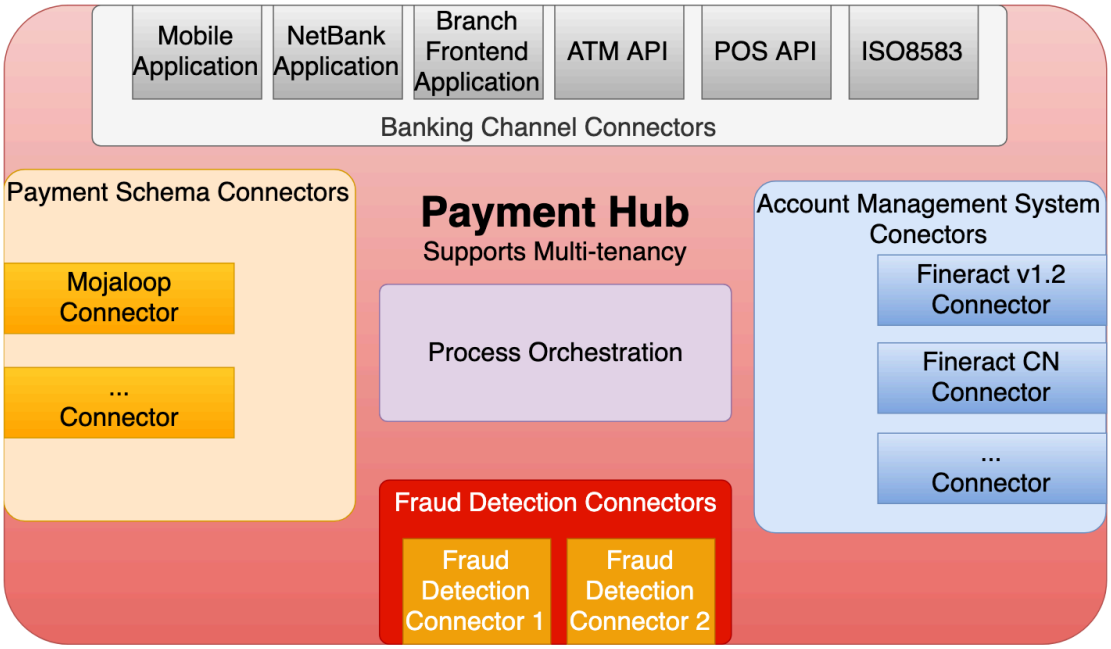
1. Integrating the app with Payment Hub to enable disbursement via Mobile Money API.
2. Improving Task management features in the app.
3. Creating UI for creating a new account and displaying account details.

4. Creating UI for creating tellers and displaying tellers details.
5. Improving GIS features like location tracking, dropping of the pin into the app
6. Improving offline mode via Couchbase.
7. Support Writing Unit Test, Integration Test, and UI tests

Design / description of work

1. **Integration with payment hub.**

   The Payment Hub is a self-contained application to enable a financial institution (DFSP - digital financial service provider) to integrate its various core banking systems (AMS - account management system).
   Technically it is a spring microservice running on Kubernetes. It orchestras the communications between the payment schema connector and account management system.



Payment Hub Architecture

The payment hub APIs can be used to initiate a payment, query payment details, register MSISDN, initiate a request to pay.
The description of the payment hub APIs to initiate a payment, query payment details, register MSISDN, initiate request to pay.

Configured domain for channel-connectors:

- barebone: -

- medium: med-connector-channel.mifos.io
- large: large-connector-channel.mifos.io

Configured tenants in lab environments:

- barebone: tn03, tn04
- medium: tn05, tn06
- large: tn01

a. **Initiate Payment** Payer sends the amount of money to the payee.

```
Url: http://{environment-channel-connector-domain}/channel/transfer
Method: POST
Headers:
 Platform-TenantId: {configured-tenantId-in-channel-connector}
 Content-Type: application/json
Body:
{
 "payer": {
       "partyIdInfo": {
            "partyIdType": "MSISDN",
            "partyIdentifier": "27710101999"
       }
 },
 "payee": {
       "partyIdInfo": {
             "partyIdType": "MSISDN",
             "partyIdentifier": "27710102999"
       }
 },
 "amount": {
 "amount": 230,
 "currency": "TZS"
 }
}
Response:
{
 "transactionId":"84b34cfc-15ee-4646-902b-d92152028200"
}
```

b. **Query Payment Details** Check the details of an ongoing transfer, not the details of a transaction request.

```
Url: http://{environment-channel-connector-domain}/channel/transfer/{transactionId}
Method: GET
Headers:
```

```
Platform-TenantId: {configured-tenantId-in-channel-connector}
Response:
{
"clientRefId": "000000", -- possibly unknown client started it because it is not a
required field in the request
"completedTimestamp": "2020-07-06T18:58:46.883", -- possibly null field in case of
not yet completed transfer
"transactionId" : "84b34cfc-15ee-4646-902b-d92152028200",
"transferState" : "RECEIVED",
"transferId" : "b155e298-dd7f-4199-9636-fa5ebec2bc58" -- possibly null field,
transfer code only captured after transfer sent
}
```

c. **Initiate Request To Pay** Payee asks the payer to send an amount of money. (currently without confirmation and authorization) The receiving DFSP(payee) of this request will be the target of the transfer from the payer.

```
Url:
http://{environment-channel-connector-domain}/channel/transactionRequest
Method: POST
Headers:
Platform-TenantId: {configured-tenantId-in-channel-connector}
Content-Type: application/json
Body:
{
"payer": {
     "partyIdInfo": {
          "partyIdType": "MSISDN",
          "partyIdentifier": "27710203999"
     }
},
"payee": {
     "partyIdInfo": {
          "partyIdType": "MSISDN",
          "partyIdentifier": "27710305999"
     }
},
"amount": {
     "amount": 77,
     "currency": "TZS"
}
}
Response:
{
"transactionId":"84b34cfc-15ee-4646-902b-d92152028200"}
```

The Fineract CN Mobile uses retrofit and rxjava. Retrofit is a type-safe HTTP client for Android and Java. It turns Rest API into Java Interface. As IO call cannot be executed on the main thread of android, The IO operations must be called within a background thread. RxJava simplifies the Issue Thread by providing an Observable architecture. RxJava resolves the memory leak issue in android by providing Android Lifecycle Aware Observable.

Retrofit can be used for the integration as follows

```
interface PaymentHubService {
  @GET("/*API_PATH*/")
  fun initiatePayment(/*Request body, Path etc*/): Call<TransactionDTO>
  ...
}
// TransactionDTO is a POJO Object
```

The Retrofit class generates an implementation of the GitHubService interface.

```
val paymentHubClient : Retrofit = Retrofit.Builder()
    .baseUrl("$PAYMENT_HUB_API_URL")
    .build()


val mPaymentHubService:  =
paymentHubClient.create(PaymentHubService::class.java)
```

As Fineract CN Mobile uses dagger the mPaymentHubService will be the part of the PaymentHub dagger component.

This is the same feature as Implemented in openMF mobile wallet, https://github.com/openMF/mobile-wallet/pull/549

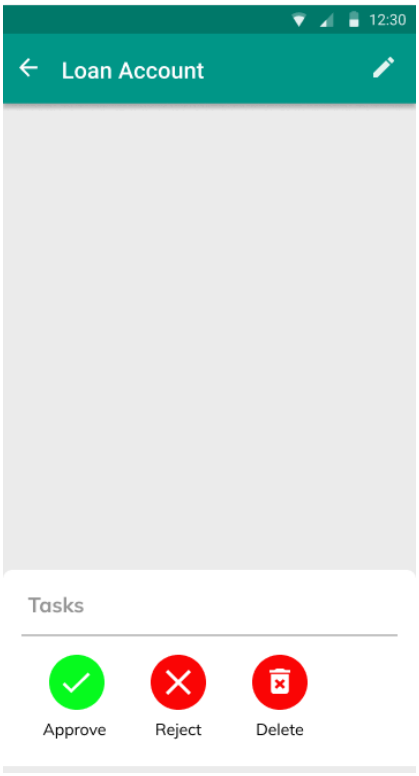
2. **Improving Task management features**
    a. Task management feature for Teller.

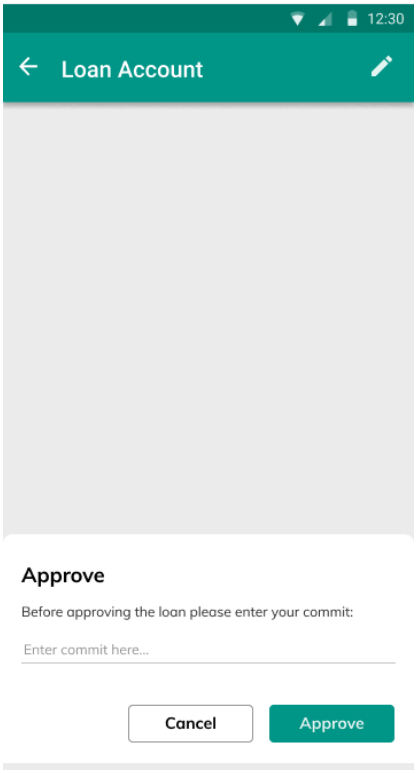
        In Apache Fineract-CN, documentation for teller-management has stated the rest of the API endpoints for a teller. The document has also mentioned the operations or tasks performed by the teller. These tasks are as follows -
        1. Unlock A Drawer
        2. Open An Account
        3. Close An Account
        4. Confirm A Transaction
        5. Save A Denomination
        6. Cancel A Transaction

The process of creating the account is described here. Rest of the task can be worked in a similar manner.

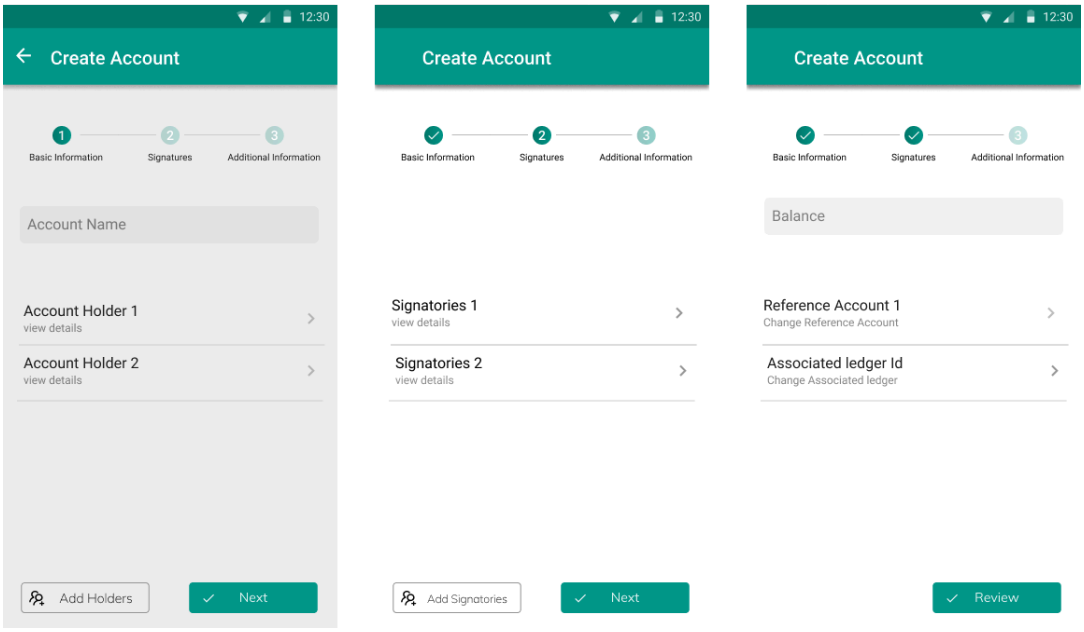
b. Task management feature for Loan Account.

Previously @jawidMuhammadi had worked on this feature. He had Implemented the UI and created a draft on github. He had stated that the Task management feature APIs for customer's loan accounts is Missing. He has also raised an issue (FINCN-234) on JIRA regarding this. Till now Issue is still open. Below are the screens for Loan Account Task.





3. **Creating UI for creating a new account and displaying account details.**
As per documentation, to create a new account Account name, Account Holders, Account signatories, Account balance, ledger are required. Appropriate message is to be shown in case of failure or success using infographics. Creating a new account can be divided into three fragments as shown in the mockup.



Android Jetpack's Navigation component is to be used to orchestrate the navigation of fragments. Displaying the account could simply reuse the review UI.

4. **Creating UI for creating tellers and displaying tellers details.**

   As per the documentation, http-request for creating the teller is;

```
POST /offices/office1234/teller/ HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: localhost:8080
Content-Length: 317

{
  "code" : "1234",
  "password" : "password",
  "cashdrawLimit" : 5000000,
  "tellerAccountIdentifier" : "TEL123BA",
  "vaultAccountIdentifier" : "TEL123BA",
  "chequesReceivableAccount" : "CHA2018XYZ",
  "cashOverShortAccount" : "CHA2018XYZ",
  "denominationRequired" : false,
  "assignedEmployee" : "Nakuve Lah"
}
```

5. **Improving GIS features like location tracking, dropping of the pin into the app.**

   There are two ways to get a user's location in any android:

   - android.location.LocationListener
   - com.google.android.gms.location.LocationListener

   Google officially recommends using Google Play Location Service APIs. **GoogleAPIClient** allows us to call multiple Google APIs using a single call.Following is an example snippet to invoke the GoogleAPIClient with two APIs : Location Services and Drive API.

```
val mGoogleApiClient = GoogleApiClient.Builder(this)
                      .addApi(LocationServices.API)
                      .addApi(Drive.API)
                      .addScope(Drive.SCOPE_FILE)
                      .addConnectionCallbacks(this)
                      .addOnConnectionFailedListener(this).build()

mGoogleApiClient.connect()
```

   Location data can be tracked using the LocationListener interface. The received location can be shared through a proper communication channel. Then the received location data can be shown using google maps API.

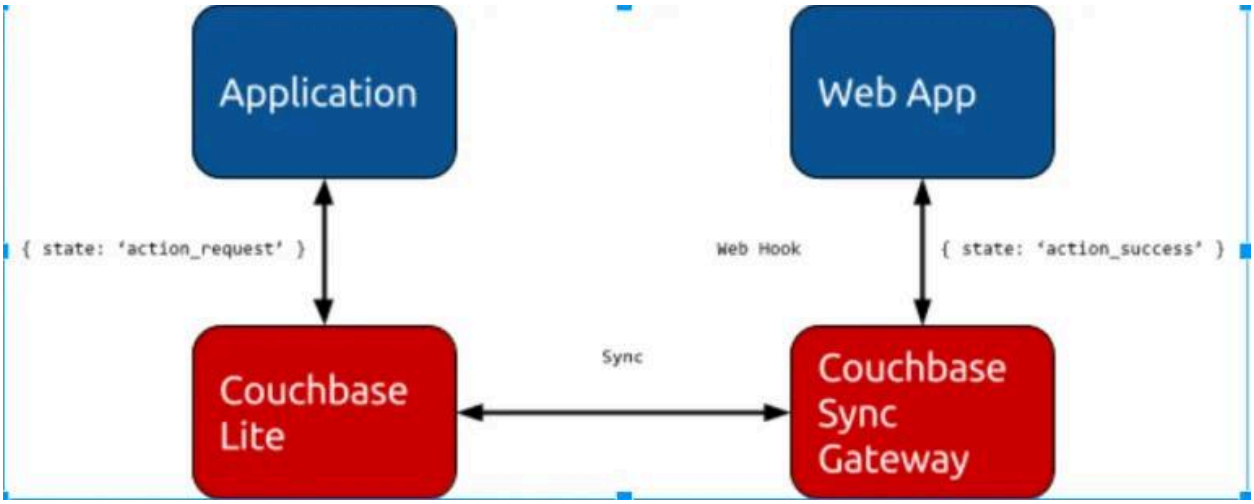
   Pin dropping can also be implemented using Google maps API. Google maps API provides markers for this purpose. Markers indicate single locations on the map. Markers can customize your markers by changing the default color, or replacing the marker icon with a custom image. For example,

```
override fun onMapReady(googleMap: GoogleMap) {
    // Add a marker in Sydney, Australia,
    // and move the map's camera to the same location.
    val sydney = LatLng(-33.852, 151.211)
    googleMap.addMarker(
        MarkerOptions()
            .position(sydney)
            .title("Marker in Sydney")
    )
    googleMap.moveCamera(CameraUpdateFactory.newLatLng(sydney))
}
```

   Previously @jawidMuhammadi had already worked on this feature (Pull Request Link).

6. **Improving offline mode via Couchbase.**

   Couchbase is a distributed NoSQL cloud database. It delivers unmatched versatility, performance, scalability, and financial value across cloud, on-premises, hybrid, distributed cloud, and edge computing deployments. Couchbase Lite is an embedded, NoSQL JSON Document Style database for your mobile apps. After understanding a bit more about Couchbase and Couchbase Sync Gateway, it seems possible to apply this kind of pattern and let Couchbase deal with all the communication of the App and having a fully functional offline app.This pattern provides a very good user experience because you actually render the page based on the current status of the app.



   The application itself never does any call to the backend, the only responsibility it has is to save states and render them, this way it's possible to have a fully functional app working offline. Couchbase Lite will be responsible to sync the current state of the App to Couchbase Sync Gateway and retrieve new information once the document is updated in the backend.

   Previously @jawidMuhammadi had already worked on this feature.

7. **Support Writing Unit Test, Integration Test, and UI tests**

   a. Unit Test -

   Unit testing is one of the software testing types which includes the initial testing phase where the smallest components or the modules of a software are tested individually. Fineract CN Mobile uses JUnit, Mockito and PowerMock for unit testing purposes. Mockito is a mocking framework written for java. Using it with kotlin has a lot of issues. For example - kotlin by default makes every class final and Mockito doesn't work with final classes. Due to these shortcomings of mockito with kotlin, PowerMock must be used. PowerMock is a Java framework that allows you to unit test code normally regarded as untestable.

b. Integration Test -

Integration tests verify how different units collaborate with one another. With only single-class tests, the test suite may pass but the feature may be broken, if a failure occurs in the interface between modules. Integration tests will verify end-to-end feature behaviour and catch these bugs.

c. UI Test -

User interface (UI) testing lets you ensure that your app meets its functional requirements and achieves a high standard of quality such that it is more likely to be successfully adopted by users. Android UI tests are executed using Espresso Framework. Espresso test takes up a lot of code to test. However espresso tests can be easily created with an espresso test recorder.

Results for the Apache community

After the completion of this project the Fineract CN mobile app will have Improved and new features as mentioned. The code will have improved test coverage. Major and minor bugs will be fixed in parallel with the completion of this project. All the changes of the app will be well documented.

Derivables

- Integration of Mifos Payment hub.
- Improved Task management features.
- Working Android fragment for creating new account and displaying account details.
- Working Android fragment for creating tellers and displaying tellers details.
- Enhanced GIS features like location tracking, dropping of the pin into the app
- Enhanced offline support.

Scheduling

Brief Timeline

- (Phase 0) Till May 16 : Pre-GSoC Period
- (Phase 1) May 17 - June 7 : Community Bonding Period
- (Phase 2) June 7 - July 11 : Coding
- (Phase 3) July 12 - July 16 : Evaluations
- (Phase 4) July 19 - August 15 : Coding
- (Phase 5) August 16 - August 23 : Students Submit Code and Final Evaluations

Detailed Project Timeline

Phase 0 - Pre-GSoC Period -

Week no	Date	Task
1	April 05	Configuring Environment, Creating First Build
2	April 12	Going through the code base.
3	April 19	Going through the code base.
5	April 26	Going through Power Mock Docs.
6	May 01	Going through Couchbase Docs.
7	May 10	Configuring couchbase.

Phase 1 - Community Bonding Period -

Weeks	Date	Task
7 - 10	May 17 - June 6	Discuss project implementation in detail with mentor and knowledgeable community members. The main focus will be to frame a roadmap for the project with the guidance of the mentor. This period will also be used to study the already present work on Fineract CN mobile app and to figure a concrete plan to integrate the payment hub into the app.

Phase 2 - Coding -

Weeks no	Date	Task
11	June 07	Creating Payment hub retrofit module in a test driven development environment.
12	June 14	Creating Mockup screens for task management features And Implementing the mockups with the guidance of the mentor.
13	June 21	Creating mockup UI for creating new accounts and displaying account details and Implementing the mockups with the guidance of the mentor. Writing unit tests for the ui, in parallel with implementation.
14	June 28	Creating mockup UI for creating tellers and displaying tellers details and Implementing the mockups with the guidance of the mentor. Writing unit tests for the ui, in parallel with implementation.
15	July 05	Writing Integration test and Espresso test for accounts and teller ui.

Phase 3 - Evaluations -

Weeks	Date	Task
16 - 16	July 12 - July 16	This period will be used to write a detailed report on the work done in phase 2. All the work done will be uploaded and documentation will be created/uploaded to **fineract-cn-mobile's** confluence page.

Phase 4 - Coding -

Weeks no	Date	Task
17	July 19	Implementing location tracking GIS feature using google maps api.
18	July 26	Adding location pinning feature using google maps api**.**
19	August 02	Writing junit test, integration test and UI test for new GIS features.
14	August 09	Improving offline capabilities of the app using couchbase.
15	July 05	Writing junit test, integration test and UI test for testing offline capabilities of the app.

Phase 5 - Students Submit Code and Final Evaluations -

Weeks	Date	Task
7 - 10	May 17 - June 6	All documentation, modules, and tests will be uploaded and CI will be integrated into the project Github page. ***All the deliverables promised for GSoC will be provided by this stage.***

Other commitments

I have Internal Exam 1 from May 17 to May 20, Internal Exam 2 from July 8 to July 10 and End Semester Exams from August 1 to August 9. Apart from this I have no other commitments.

Community engagement

1. Repo on Github:
   https://github.com/apache/fineract-cn-mobile
2. Documentation:
   https://cwiki.apache.org/confluence/display/FINERACT/Fineract+CN