Google Summer of Code

GSOC'23 PROPOSAL - Mifos Initiative

Mifos Mobile 6.0 - Mobile Banking App

Organization: Mifos Initiative

Project Name: Mifos Mobile 6.0 - Mobile Banking App

Candidate Name: Pratyush Singh

Expected Project Size: 350 hours

Mentors:

- Ahmad Jawid Muhammadi
- Garvit Agarwal
- Saksham Handu

Contents

1. Project Idea

Abstract

- The primary objective of this year's project is to modernize and enhance the project's functionality and performance by replacing the existing self-service Fineract APIs with Open Banking APIs. This will ensure that the project is updated and compatible with the latest technology. Furthermore, the project requires integration with an external payment system, such as Mojaloop or mPesa, through the payment hub.

- A critical task in achieving these objectives is migrating from Dagger to Hilt, a more efficient dependency injection library. Additionally, the aim is to migrate from MVP to MVVM-Clean architecture to improve maintainability and scalability. Integration of the Navigation Graph will enhance the user experience. Moreover, integrating Coroutines into the project will enable asynchronous programming and simplify concurrency management. Additionally, the plan is to add unit tests for the Data Layer and UI Layer, along with UI tests covering all screens, ultimately integrating Jetpack Compose.

- Finally, improving Github workflows and adding jobs to run unit and UI tests will help us ensure continuous integration and delivery. By completing these tasks, the project will become updated, efficient, and maintainable, ultimately improving its functionality and performance. I believe that achieving these objectives will enhance the project's potential for scalability and maintainability, contributing to its long-term success.

2. Implementation Details

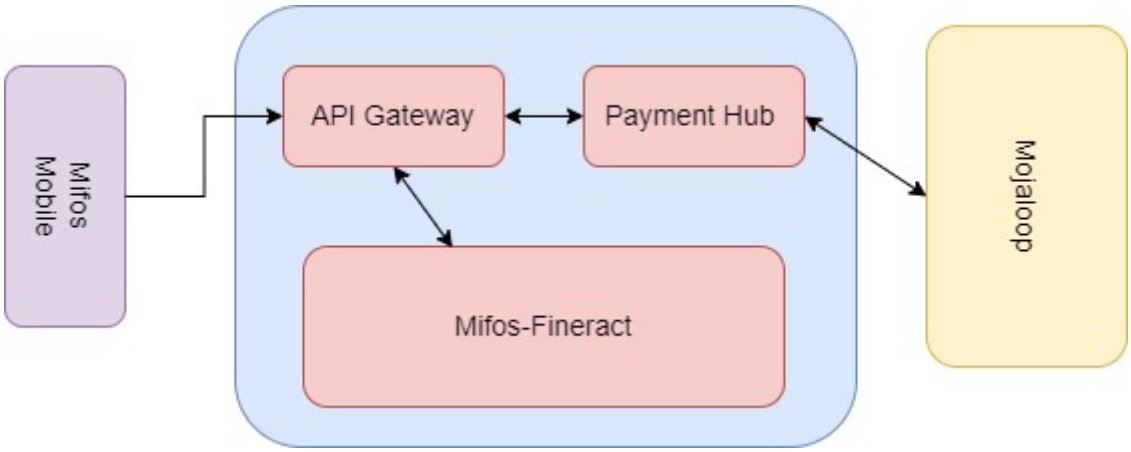
2.1 Replace API layer from self-service Fineract to Open Banking

- Replacing the API layer from self-service Fineract API to Open Banking API can provide significant benefits to financial institutions, including increased security, access to more financial data, improved customer experience, enhanced innovation, and compliance with regulatory requirements such as PSD2 and GDPR
- I had gone through the Open Banking App and found another project by the name of common that was imported in this project. It contains Api.kt file that has all the functions and constants related to network data sources for the Mifos Open Banking project. That file can be found here
- Using the functions defined in the above file, I can work my around the **Client APi** and the **Transaction API**. ClientApi would provide a simple interface for making requests to the banking API, handling errors, and returning the parsed response data and the role of TransactionApi would be providing methods for creating, fetching, and managing transactions. These two files can be found here
- I will also import common in the mifos mobile project and use the App object to use the functionalities of the imported project in the mifos mobile project.The App object can be found here

2.2 Integration with Mojaloop via payment hub

- The Mifos Payment Hub can be used to connect to Mojaloop (payments switch) for inbound and outbound transactions by implementing Mojaloop API and mapping them to the Fineract API

- To my understanding we will Install Mifos's Payment Hub on a server or cloud platform & then configure the Payment Hub to use the Mojaloop API as the payment gateway. We can then integrate the payment hub with our mobile using the available APIs that can be found here
- We can create a Fineract API manager that can be used to create and manage instances of different Fineract API services and would hold constants for different API service URLs and creates a Retrofit instance to communicate with the API. Mobile wallet currently has this feature and the file would look something like this
- To facilitate transaciton when initiating a payment, the client can create a new instance of the **Transaction class**, providing the necessary information such as the transacting entities and the amount to be transferred. This Transaction object can then be serialized and sent to the server to initiate the payment.Once the payment has been initiated, the server will respond with a **TransactionResponse object**, which contains details about the transaction such as the completed timestamp and transfer state.
- The client can then use the **TransactionInfo class** to query the server for additional information about the transaction, such as its status or any errors that may have occurred.Finally, **TransactingEntity class** is used to represent the parties involved in the payment transaction, such as the payee and payer. These objects can be created and attached to the Transaction object to provide the necessary information for the payment
- Besides the above mentioned entites a few more entites such as *Amount* , *Identifier*, *IdentifierType*, *PartyIdentifiers* and *RegistrationEntity* will need to be implemented
- When a transaction is initiated, the parties involved will be identified using their **Identifier entity**, which is then added to the **PartyIdentifiers entity**. The initiating party's details will then added to the **Registration Entity**. Finally, the **Amount** entity will represent the amount of money being transferred
- I have taken this PR as reference and the same can be used to implement this feature into the app
- For testing purposes,the integration can be tested in a sandbox environment to ensure it is working as expected



2.3 Migration from MVP to MVVM

- Currently in the MVP architecture the View and Presenter are tightly coupled, which can make it difficult to modify the code or reuse components. However with MVVM, the ViewModel serves as an intermediary between the View and Model, allowing for a **looser coupling** and greater **flexibility**
- MVP can lead to code that is complex and difficult to maintain, especially when dealing with complex user interfaces. MVVM can simplify the code by separating the business logic from the UI components, making it easier to modify and maintain the codebase
- In MVP, managing the state of the View can be challenging, especially when dealing with configuration changes such as screen rotation. With MVVM, the ViewModel can maintain the state of the View, making it easier to manage and **avoid data loss**
- Furthermore,MVVM architecture can be combined with reactive programming libraries like RxJava, which can make it easier to handle **asynchronous data flows** and improve the application's **performance**

2.3.1 Migrating an Activity

- In the original code, all the business logic was in the SplashActivity class. This violates the separation of concerns principle, which states that each class should have a **single responsibility**. In the MVVM architecture, the ViewModel is responsible for holding the data and the business logic, while the View (in this case, the SplashActivity) is responsible for displaying the data and handling user interactions.

- So, to follow this principle, we create a separate SplashViewModel class that holds the business logic, and move the logic from the SplashActivity to the SplashViewModel. This way, the SplashActivity is only responsible for displaying the data returned by the SplashViewModel, and the SplashViewModel is responsible for deciding what data to return and how to generate it.

- The SplashViewModelFactory is used to create an instance of the SplashViewModel. We use a Factory instead of directly creating a new instance of the ViewModel class because the ViewModel class has a lifecycle that is tied to the lifecycle of the View (i.e., the SplashActivity), and the Factory ensures that we always get the same instance of the ViewModel even if the View is recreated due to configuration changes like screen rotation. The Factory creates the ViewModel using the PasscodePreferencesHelper object as a constructor parameter, so that the ViewModel has access to the data it needs to generate the appropriate Intent.

- I have migrated the SplashViewModel to MVVM and the source code can be found here

2.3.2 Migrating a fragment

- I have converted the LocationsFragment from MVP to MVVM because the MVVM pattern separates the business logic from the view, making the code more modular and easier to test.In the MVP pattern, the presenter is responsible for handling the business logic and updating the view, which can lead to tight coupling between the view and the presenter. This can make the code harder to maintain and test.

- In the MVVM pattern, the view model is responsible for handling the business logic, while the view is responsible for displaying the data. This separation of concerns makes the code more modular and easier to test.To convert the LocationsFragment from MVP to MVVM, I removed any reference to the presenter and created a view model to handle the business logic.

- In the LocationsViewModel class, I created two methods to handle the logic previously handled by the presenter. These methods receive the GoogleMap instance as a parameter and use it to add a marker and animate the camera to the headquarter location. In the LocationsFragment, I removed the onCreateView() method from the MVP implementation and created a viewModel property that is initialized in the onCreateView() method. I also removed the addMarker() and addAnimationToHeadquarter() methods and called the corresponding methods in the viewModel instance from the onMapReady() method.

- Overall, the MVVM implementation is more modular and easier to test, as the business logic is separated from the view.

- I have migrated LocationFragment to MVVM and the source code can be found here

2.4 Basic Integration of Navigation Graph

- There aren't any navigation graphs in the project yet hence all navigation logic between different screens and activities are handled manually making the project more complex and error-prone code, which can slow down the app and make it more difficult to maintain.As the app grows in complexity, it can become increasingly difficult to manage the navigation logic manually. This can make it challenging to add new features or modify existing ones, which can limit the app's ability to scale and evolve over time.
- With navigation graph in place can improve user experience by allowing users to easily navigate between screens and tasks .Besides it provides a simplified and standardized approach to app navigation, making it easier for developers to build and maintain complex apps. It also helps to reduce the amount of code required to handle navigation, which can save development time and effort.
- The **Navigation Component** in Android provides a framework for implementing navigation between screens in an application. It uses a Navigation Graph, which is an XML file that defines the navigation paths and destinations in an app.The first step in integrating the Navigation Graph is to create the graph file and define the navigation paths and destinations. This can be done using Android Studio's Navigation Editor, which provides a visual interface for designing the graph.
- Once the Navigation Graph is created, the next step is to integrate it into the app's code. This involves using the Navigation Component APIs to load and navigate to the appropriate destination based on user input.
- As an example , I have taken up **SettingsFragment** and **UpdatePasswordFragment** . To implement a navigation graph I have added these two files in the xml and defined an action from SettingsFragment to the UpdatePasswordFragment.Having done this, I added a **FragmentContainerView** in the SettingsActivity that will act as my NavHost.
- The appropriate changes in activity_settings.xml is :

```
<androidx.fragment.app.FragmentContainerView
            android:id="@+id/fragmentContainerView"
            android:name="androidx.navigation.fragment.NavHostFragment"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:defaultNavHost="true"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
```

```
            app:layout_constraintHorizontal_bias="0.0"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintVertical_bias="0.0"
            app:navGraph="@navigation/nav_frags" />
```

- Now we do not need the below code in **SettingsActivity**

```
replaceFragment(SettingsFragment.newInstance(), false,R.id.fragmentContainerView)
```

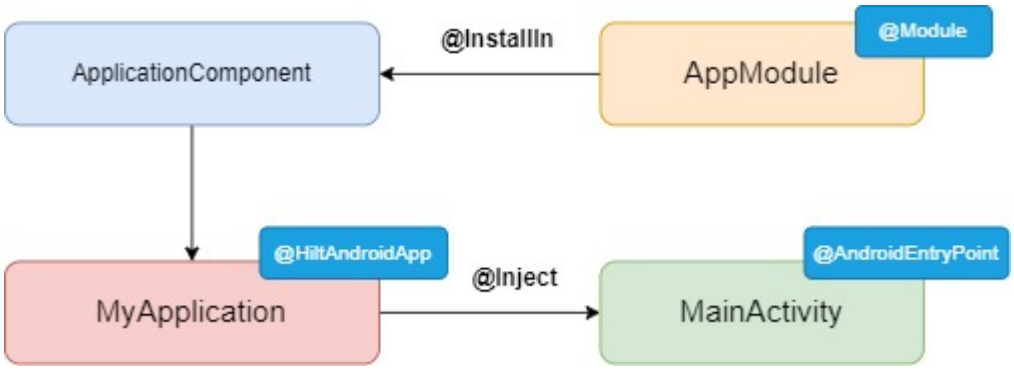
- We now just need to do one last change in the SettingsFragment i.e. we need to add findNavController.
- The change in the SettingsFragment is :

```
override fun onPreferenceTreeClick(preference: Preference): Boolean {
        when (preference.key) {
            Constants.PASSWORD ->
findNavController().navigate(R.id.action_settingsFragment_to_updatePasswordFragmen
t)
        }
        return super.onPreferenceTreeClick(preference)
    }
```

- I have taken this just to show it as demo and I can add a few more navigation graphs that would make the app further scalable. For instance , currently we have a **LoginActivity** and a **RegistrationActivity** .But the role of the **RegistrationActivity** is just to replace the fragment which is **RegistrationFragment**. We can get rid of the two activities and replace them by just one activity let's call it **AuthenticationActivity** and convert the **LoginActivity** to a fragment.
- So now you will have just one Activity and on top of that you will have two fragments that can be added in another navigation graph.

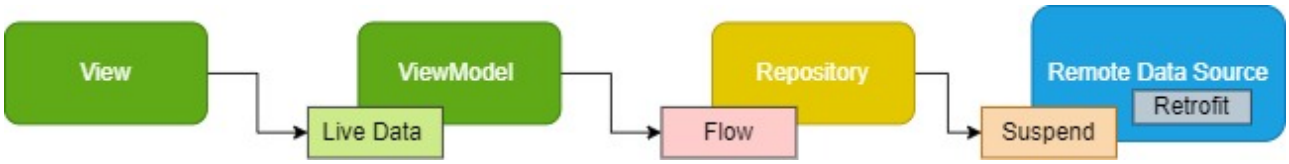
2.5 Migration from Dagger to Hilt

- We will start off with a @Singleton component and then later migrate activities and fragments
- I propose we migrate the Login and Registeration first and then move towards inner fragments and during the migration I'll remove all @Component and @Subcomponent interfaces and annotate all modules with @InstallIn.
- After the migration, all Application/Activity/Fragment/View classes will be annotated with @AndroidEntryPoint and any code instantiating or propagating components shall be removed.



2.6 Basic integration of coroutines

- Keeping in mind that we will be migrating to MVVM ,our **Repository layer** would use coroutines to run long running code off the main thread
- We can use Hilt to inject dispatchers thus making it easier to test our code and also we can ensure that our code is thread-safe and that we are using the appropriate dispatcher for each task.Therefore, I believe that using Hilt to inject dispatchers in repositories is an important aspect of our project's architecture, and I plan to incorporate it into our MVVM architecture.
- Moreover, we could use **Flow** to compose and coordinate multiple related operations; each of which in-turn call multiple coroutines **ViewModel** calls this code confining the scope to its *viewModelScope* so that when this ViewModel is cleared; all running operations in this scope are cancelled. We then use **LiveData** to communicate the result to the View layer.Finally View's will call these method's from ViewModel
- Thus, using coroutines and viewModelScope allows for asynchronous operations to be run off the main thread, thus preventing the UI from freezing or becoming unresponsive during long-running tasks and use of viewModelScope ensures that all running operations in the ViewModel are cancelled when the ViewModel is cleared, preventing potential memory leaks and ensuring efficient use of system resources.



2.7 Customer support via rocket chat

- A customer support will be a viable step which will provide a better experience to our users and rocket chat can support large volumes of customer inquiries and interactions
- Rocket.Chat.Android SDK can be used to implement this feature . Moreover, Shivangi Singh had forked the same and integrated code such that it connects to mifos mobile integration as a functionality .The forked repository can be found here.
- After doing the above modifications the flow would be :
  - Saving client id and client secret in the app

- Request access to user data from the app
- Use the authorization code to get the access token.
- Use the access token to interact with the API
- Send back the response using the same token and storing the token to reach out to the daily customers and store the progress of the queries addressed and solved

2.8 Unit tests for data and UI layer

- **UI testing**: Using the latest frameworks and techniques, I will develop a robust and scalable suite of tests that can simulate user interactions and detect potential bugs, errors, or inconsistencies in the app's visual interface. By leveraging tools such as Espresso, Robolectric, and UI Automator, I will create a comprehensive set of test cases that cover various scenarios, from basic navigation and input validation to more complex workflows and edge cases

- **Non-UI testing**: In addition to the UI tests, I will also design and implement a set of unit tests that focus on the underlying logic and functionality of the app, without relying on the graphical interface. These tests will use frameworks such as JUnit and Mockito to verify the correct behavior of the code, including data storage, network communication, and business logic. By separating the UI and non-UI tests, I will ensure that the app can be tested thoroughly and efficiently, without sacrificing speed or accuracy.

2.9 Improving Github Workflows

- Currently the workflows do not have any jobs that runs the Unit tests or the UI tests. Hence modifications in the workflow to allow both tests to run will be a good feature to have.By running unit tests and UI tests as part of our Github workflow, we can get early feedback on code changes and catch issues before they are merged into the main branch. This can help prevent bugs from being introduced into the production code.
- It can provide a faster feedback loop for developers and can help them iterate and develop features more quickly.By automating testing as part of your Github workflow, you can reduce the need for manual testing. This can save time and resources and allow developers to focus on other tasks.
- Job for Unit test would be :

```
unitTest:
    name: Unit Tests
    runs-on: ubuntu-latest
    steps:
      - name: Checking out repository
        uses: actions/checkout@v2

      - name: Run Unit Tests
        run: ./gradlew testDebugUnitTest

      - name: Upload Test Report
        uses: actions/upload-artifact@v2.2.2
        if: always()
        with:
```

```
          name: Unit Test Report
          path: app/build/reports/tests/testDebugUnitTest/
```

Job for UI tests would be :

```
  uiTest:
    name: UI Tests
    runs-on: ubuntu-latest
    needs: build
    steps:
      - name: Checking out repository
        uses: actions/checkout@v2

      - name: Start Emulator
        uses: reactivecircus/android-emulator-runner@v2
        with:
          api-level: 29
          arch: x86
          profile: Nexus 5X

      - name: Wait for Emulator to Start
        run: android-wait-for-emulator

      - name: Unlock Screen
        run: adb shell input keyevent 82

      - name: Run UI Tests
        run: ./gradlew connectedDebugAndroidTest

      - name: Upload Test Report
        uses: actions/upload-artifact@v2.2
```

- **unitTest job** : This job is responsible for running unit tests on the application's codebase. Unit tests are automated tests that check individual units or modules of an application in isolation to ensure that they are functioning correctly. The unitTest job runs the unit tests using the Gradle wrapper and a testing framework such as JUnit or Mockito.
- **uiTest job** : This job is responsible for running UI tests on the application's user interface. UI tests are automated tests that simulate user interactions with the application to ensure that its UI is functioning correctly. The uiTest job runs the UI tests using a testing tool such as Espresso
- I have modified the master_dev_ci.yml and the modified yaml file can be found here

2.10 Integration of Jetpack Compose

- Currently the app is using XML layouts that can lead to overhead in terms of memory usage and application performance, as they need to be inflated at runtime, which can slow down the

application.Even the errors are manifested at runtime making it cumbersome to track down the root cause

- With jetpack compose we can have a a more modular approach to UI development, which makes the code easier to maintain as it will allow us to break down UI into smaller, reusable components that can be tested and maintained independently. This can help reduce code complexity and improve code readability, making it easier to make changes to the UI without affecting the rest of the codebase
- We will also be migrating our project to MVVM and compose integrates well with **ViewModel** and **Live Data** which will make our app robust and scalable. Furthermore, we are also adding navigation graphs in our project and compose provides its own version of the Navigation component called **Navigation Compose** that allows to navigate between composables and screens.This can be implemented at a basic level and scaled further
- Since we are aiming for a basic integration of jetpack we can start off with **LoginActivity** followed by **RegistrationFragment** .This should then be followed by those activties/fragments that handles lesser data such as **About Us** page in the settings
- We need to add dependency in build.gradle to start using jetpack compose

```
implementation("androidx.activity:activity-compose:1.3.0-alpha07")
```

- My kotlin version is 1.7.20 and thus I will need to set the compose compiler version accordingly and in my case it was set to 1.4.0-alpha01.Again in build.gradle we need to add composeOptions as:

```
composeOptions {
        kotlinCompilerExtensionVersion = "1.4.0-alpha01"
    }
```

- After setting up the pre-requisities we can start off by adding @Composable functions in the kotlin file and adding the neccessary views

2.11 My Ideas for the app

These are the stretch goals for the project and would be considered only when we will be done with the main idea list

2.11.1 Use Fineract SDK

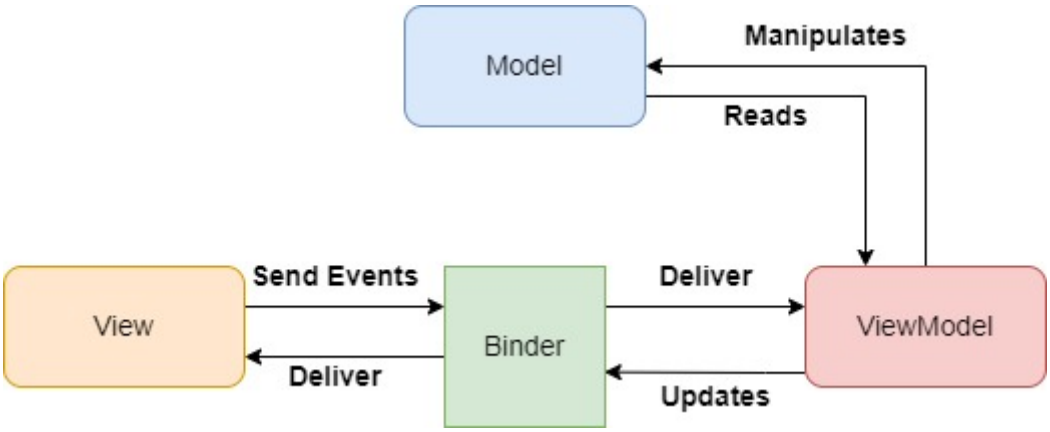
- We have a Fineract SDK which can be imported in our project as well . This will allow us to reduce the boiler plate code by reducing the number of server calls from within our project
- This will allow our app to be light weight and will provide a better experience to our users

2.11.2 Migrating from Butter Knife to Data Binding

- As the Android Development scene continues to grow, new build tools are created in order to address issues developers had seen using previous solutions. A commonly used tool was Butterknife. At the time, Butterknife was used to reduce the the amount of times the findViewById(...) function had been

used in order to reference a view in our application. However, Butterknife still had its own issues such as null safety and speed.

- Data Binding is a powerful feature of Android that allows us to bind UI components in our layout files directly to the data model, eliminating the need for boilerplate code and reducing the likelihood of errors. Here are some benefits of using Data Binding with the MVVM pattern, and its advantages over Butter Knife are :
    - Data Binding allows us to eliminate a lot of boilerplate code that is required when using Butter Knife. With Data Binding, we can bind our UI components directly to the data model, reducing the need for manual view injection.
    - Data Binding is type-safe, meaning that we can detect errors at compile time rather than runtime. This can help reduce the likelihood of bugs and improve the stability of our code.
    - Considering the fact that we are migrating to MVVM , Data Binding is a natural fit with the MVVM pattern. By binding our UI components directly to our ViewModel, we can create a clean and modular code structure that is easy to understand and maintain.



2.11.3 Adding Option to Change Connection setting from login page

- Currently when you go to **Settings** and then click on **Update Endpoint** and try to change the endpoint they user is sent back to the LoginActivity. And now the user cannot log in back because of the updated endpoint which might be incorrect.The user would then have to uninstall the app and then install it again just to gain access of the system
- To overcome this there should be a support in the LoginActivity to change the **Connection Settings** that would allow the user to change the endpoint without having to uninstall the app. This would save the users time and would be a good feature to have
- The implementation idea can be derived from **Android Client** that presently has this feature and the same can be replicated over in mifos mobile. Here is a video for your reference

3. Contributions to Mifos

Below are the links to my contributions :

Merged Pull Requests

1. PR #1926: Modified Error message in case of wrong endpoint
2. PR #1927: Fixes the behaviour of filters in savings section
3. PR #1930: Fixes unresponsive nature of 'Change Passcode' in Settings
4. PR #1981: Fixes the white background in dark mode
5. PR #1984: Fixes the scrolling feature in landscape mode
6. PR #2017: Fixed the crashing of app when changing main mobile theme
7. PR #1910: Fixes App Crash when orientation is changed to landscape mode
8. PR #1904: Added Country Code Picker in the Signup form
9. PR #1899: Fixes crashing of app when selecting an empty Product ID

Open Pull Requests

1. PR #2034: Fingerprint Authentication in the Passcode Activity
2. PR #1929: Fixes background color of Passcode with the theme
3. PR #1912: Fixes Scrolling in landscape mode
4. PR #1908: Fixes duplication in Material Auto Complete Text View
5. PR #1905: Allows User to enter the correct passcode even after three unsuccessful tries
6. PR #1938: Offline Support in the home fragment
7. PR #2002: Fixed the app to remember language set by the user before logging out

Issues Reported

- I had opened quite a few issues that mainly focused on bugs that were present in the codebase and also on the features that were absent from the mifos mobile at the time . Some of them are still open and have PRs either by me or from my fellow contributors
- All of my open and closed issues can be accessed from here

4. Week Wise Breakdown

4.1 Community Bonding Period (4 May - 28 May)

Week 1

- Get in touch with the developers and the mentor
- Introduction to the community, to the mentor and fix timings to communicate
- Discuss any suggestions and changes to the project. There could modifications, new additions or ammendments; it would be better to go over these early

Week 2

- Go Thorugh Mifos-Mobile codebase

- Take reference from android client and try to implement features that are currently absent in mifos-mobile
- Go through the Open Banking API and related documentations

Week 3

- Discuss the working of the existing application with the mentor. Discuss the implementations of the new features
- Go over the new design in detail and ask for changes and suggestions
- Discuss the changes that need to take place in the frontend to accomodate for the breaking changes

4.2 Phase 1 (29 May - 9 July)

Week 4

- I will start off gradually with the migration from MVP to MVVM architectural style
- The order that I will follow is :
    - Splash Activity
    - Login Activity
    - Registration Fragment
    - PassCode Activity
    - HomeOld Fragment
    - Home Fragment and so on
- This order is somewhat sequential and will be one of the first steps to be undertaken
- I would also take inputs from my mentors and if the need arises, change the order of the migration

Week 5

- I would dedicate this week to the implementation of the Navigation graph
- I would start off by grouping the fragments that are connected in a sequential flow and map out a graph for the same in the xml
- This would require the addition of FragmentContainerView in certain activities which will allow further FragmentTransaction operations on the FragmentContainerView and providing a consistent timing for lifecycle events.

Week 6

- This week would be reserved for the migration from dagger to hilt & also for the integration of coroutines in the project
- Considering the fact that we need to write unit tests as well, Hilt will make it easier to write unit tests for the code by allowing us to easily swap out dependencies with mock objects
- Hence it will be better to get on with this at the earliest so that we can lay foundation for a better unit test phase

Week 7

- Replace existing api layer with the guidance of the mentors

Week 8 and Rest of Phase 1

- *Buffer* for any pending tasks
- Prepare a report for evaluation
- Discuss brief plan for Phase 2 with the mentors

4.3 Phase 2 (14 July - 21 Aug)

Week 10

- Rocket chat will provide users with the support and facility of fast solutions to their queries. Rocket chat will provide users to communicate securely in real-time.

- Shivangi singh had already updated the callbacks in the library and had further added lifecycle to decline requests . The library can be found here
- Now all is left to be done is the integration into the project and I will integrate this once I get the updated *APIs* during the GSoC period

Week 11

- I will dedicate this week for the integratation of mojaloop via the payment hub. Currently Mobile wallet already has this feature and the same can be taken as a reference while implementing it in mifos mobile
- I will consult with my mentor the API endpoints that needs to be hit for transactions, registrations, identification and so on

Week 12

- I shall be writing unit tests for the data layer in this week of GSoC
- Uptill here we would have migrated a certain portion of our project to MVVM and hence the tests can now be written keeping the new architecture in mind
- I shall be using Mockito framework to write the tests

Week 13

- This week will be dedicated to performing the UI tests using espresso whose dependencies are already present in the gradle
- We will have decent amount of changes to the UI uptill here and hence tests that cover all the screens must be put in place
- This will ensure that the views are displayed correctly and look the way they're supposed to.Besides it will ensure that the interactions are handled correctly.

Week 14

- After writing all the unit tests I will add appropriate jobs in the existing workflow to run these tests
- Running tests regularly and automating the testing process can save a lot of time and prevent issues from arising in the codebase

Week 15 and Rest of Phase 2

- If the mentors permit then I would like to implement my own ideas. Considering the ideas aren't that cumbersome it will work out even if its accomplished at the end of the project

- *Buffer* to complete any remaining tasks
- Prepare report for evaluation

Priority Chart

High	Replace Api Layer	Integrate Moojaloop via Payment Hub	MVP to MVVM	Dagger to Hilt	Customer Support via Rocket Chat
Medium	Integration of Navigation Graph	Integration of Coroutines	Tests for Data & UI layer	Improving Github Workflows	Jetpack Compose
Optional	Use of Fineract SDK	Data Binding	Features of Android Client		

4.4 Post Phase 2 (After Aug 28)

- Discuss the project's outcome with my mentors and devise a plan of action for future contributions
- Engage with members of the community to solicit feedback on project implementation and to identify possible add-on features

5. Why am I the right person ?

I have been doing Android Application Development for more than one year now. I have gained proficiency in it by doing multiple Internships, several Open Source Contributions as well as Hackathons.I am quite conversant with Android Architectural Components, MVVM, activities, fragments, support libraries, version control, Networking Services, Firebase, UI Development, Jetpack Compose etc.

6. Current area of study

I am pre final year student pursuing **Information Science and Engineering** at Dayananda Sagar College of Engineering.

- Over the course my area of study have included :
    - Learning the fundamentals of Java, Python and C++ along with Data Structures and Computer Architecture
    - I have learnt to collect,store and analyze data using tools like SQL and Python
    - My studies have focused on Information Retrieval systems such as Search Engines,Cross Language Information Retrieval and digital libraries
    - Learning about the fundamentals of ML that included supervised and unsupervised learning,neural networks and deep learning
    - I have also had the privilege to learn about distributed systems, cloud computing, and virtualization technologies.

7. Contact Information

Name: Pratyush Singh

Email: aries.pratyush@gmail.com

LinkedIn: https://www.linkedin.com/in/Pratyush-Singh/

GitHub: https://github.com/PratyushSingh07

Gitter Id: Pratyush Singh

Mobile Number: +91 9693565684

Time Zone: Indian Standard Time (GMT+5:30)

8. Career Goals

As an android developer my first and foremost goal is to master the Android SDK,Multithreading and other related technologies and create a portfolio of projects that would include personal projects, open source contributions, or projects done as part of my studies or work. As I gain more experience , I would want to specialize in UI/UX design along with enterprise app development and seek out for leadership roles. I would sooner or later venture into AOSP and get a grasp of low level android as well and transition into a full stack mobile developer. I am self taught like many other developers out there and Open Source Projects have played an integral part in my growth and thus I would give back to this android community by sharing my knowledge and expertise through blog posts, tutorials,speaking engagements and ofcourse by contributing to Open Source Projects.

9. My Projects

1. Cyclofit : *Source Code*

- Cyclofit is a safety and health monitoring system for cyclists
- It tracks the heart rate,calories burnt, distance coverered and much more using the sensors integrated in a single device.Our device has a proximity sensor that can track any incoming vehicle by monitoring it’s rate of change of speed. The data such as the heart rate are displayed into the app through an api
- Worked with firestore , firebase , Coroutines created a community section in this app that implements a real time database
- Visualized data in form of graphs using MPAndroidChart

2. NewsLive : *Source Code*

- Developed an Android application using the NewsAPI.org API, Retrofit, MVVM architecture, Room database, and Coroutines.
- Implemented background tasks and asynchronous operations using Coroutines, ensuring smooth app performance and user engagement.
- Integrated Room database to provide offline caching and storage of user preferences and data, reducing server requests and improving app speed and reliability

- Utilized MVVM architecture to separate concerns and increase code maintainability, as well as Retrofit to easily connect to the NewsAPI.org API and retrieve real-time news data.

10. Gitter Channel

Yes, I have visited all of mifos's gitter channel and my id was PratyushSingh07

11. Other Open Source Contributions

I have been contributing to Open Souce for quite some time and here are some of my contributions:

1. Dare2Change :
    - It's an all in one android application to enhance your productivity and I got to work on this as a part of SLoP 2022
    - Revamped the complete UI of the application and introduced a dark mode theme
    - My contributions for Dare2Change: check here
2. Anki-Android :
    - There were certain @KotlinCleanup annotations in the codebase with messages such as *make data not null* and *simplify through scope functions*
    - I opened a few pull requests adhering to above mentioned messages
    - My contributions to Anki-Android : check here
3. Catroid :
    - It is written predominantly in JAVA and hence refactoring the existing codebase to Kotlin is a vital task
    - I refactored certain files to kotlin
    - My contributions for Catroid: check here

12. Experience with Angular/Java/Spring/Hibernate/MySQL/Android

Yes, I do have experience with Android and Java and have built projects centered around them. I have decent knowledge of MySQL and SQLite Databases.I have built a full stack application during the course of my internship by using Angular as the frontend and Spring Boot for developing RESTful APIs.

13. Other Commitments

I am fully committed to enhancing the Mifos mobile platform during the upcoming summer as I do not have any other conflicting commitments.

14. What motivates me to work with Mifos for GSoC

Mifos Initiative is making a significant difference in the world by providing financial inclusion to people who would otherwise be excluded from the formal financial system. This mission is truly inspiring, and being a part of it through the Google Summer of Code program is a privilege. From a professional point of view mifos has a vibrant community of developers, volunteers, and supporters who are passionate about the mission of the organization. It's motivating to be a part of a community that is so dedicated to making a positive impact on the world.Besides Google Summer of Code is a fantastic opportunity for students to gain real-world experience working on open-source projects. Mifos Initiative's participation in this program shows that they are committed to helping the next generation of developers like me succeed and contribute to something that would impact lives of billions around the globe.The work that we as a students do during the Google Summer of Code program can have a lasting impact on the Mifos Initiative and the people it serves. Knowing that the work you do can make a real difference in people's lives is incredibly rewarding.

Mifos Initiative's commitment to open-source software is essential for the sustainability of the financial inclusion ecosystem. By making their software freely available, they are enabling other organizations to provide financial services to underserved communities.

15. Previous Participation in GSoC

No,this is my first participation in Google Summer of Code

16. Application to multiple orgs

I will be applying only to Mifos-Mobile for this years' GSoC