Google Summer of Code

GSoC'24 PROPOSAL - The Mifos Initiative

Functional Enhancements to Mobile Wallet for G2P Use Cases

Organization: Mifos Initiative

Project Name: Functional Enhancements to Mobile Wallet for G2P Use Cases

Candidate Name: Pratyush Singh

Expected Project Size: 350 hours

Mentors:

- Rajan Maurya
- Avinash Vijayvargiya

Contents

1. Project Idea

Abstract

- The project focuses on the enhancement and refinement of a feature-rich and secure Mobile Wallet application tailored for Government to Person (G2P) payments, demonstrating a potent tool for fintechs and financial institutions. Originating as a Google Summer of Code initiative from 2017-2023, the project delivered the initial mobile wallet framework, evolving into a sophisticated reference application with two primary apps: PixieCollect and MifosPay, later consolidating focus on MifosPay. The developmental trajectory of the project saw significant milestones: integration with Mojaloop transaction flows, user experience enhancements, support for Kotlin, and implementation of standing instructions and merchant transactions.

- Further advancements include integration with Fineract CN, multi-theme support, migration to Kotlin, and a transition towards a multiplatform approach using Kotlin multi-platform. The project aims, in 2024, to finalize G2P functionalities, ensuring production-readiness, improving security, and adopting modern development practices such as Jetpack Compose and leveraging Mifos' Android SDK. It underscores a commitment to evolving into a generic wallet management system through the Mifos X framework, incorporating cutting-edge features like the latest Payment Hub EE version, a new notifications framework, and a redesigned architecture to align with the Open Wallet Foundation's principles. This ambitious endeavor promises to redefine the landscape of mobile wallets, emphasizing security, extensibility, and a seamless user experience for G2P payments and beyond.

2. Implementation Details

2.1 Integrate latest version of Payment Hub EE

- Payment Hub EE is a gateway and integration layer enabling financial institutions (DFSPs) to seamlessly connect to external payment systems and operational control center to manage and monitor the transactions flowing through these systems. Payment Hub EE supports **Mobile Money & Merchant payments**, **Bulk Payments & G2P**, **Open Banking** among other digital payment use cases

- The Payment Hub EE abstracts out the complexity of connecting directly to the APIs of an external payment system and provides a scalable and sophisticated orchestration layer powered by Zeebe to elegantly route payments and transactions across various microservices and APIs facilitating the flow of transactions amongst the core banking system, channel applications, and external payment systems.

- To integrate this in our current setup we may have to replace the existing APIs with the payment hub apis that are listed here. Moreover, we also need to satisfy our G2P use cases and apis related to those can be found here. Currently the latest version of Payment Hub is **v1.13.0** with G2P Sandbox chart v1.5.0 which was released on 24th March 2024.

- By the time of implementation of this idea there may be changes in the version and I will adopt to those changes accordingly in consultation with my mentor. Since we will be dealing with real time data such as transfers and other transaction we may need to implement sockets in such scenario. This is only a hypothesis and I believe to gain more insight once I actually start working on this under the guidance of my mentor

- Again like last year if we have a session on payment hub then it will help me in understanding the use cases a lot better and thus improve the way in which I integrate this in my app

2.2 Integrate Mifos' notifications framework

- The current notification sub-system in Mifos provides a service to send notifications to a set of user. The service maps a notification message to recipient users and thus requires a list of user identifiers as parameter.At the point of notification generation, the list of users to be notified is evaluated in real time based on their organization, role and permissions

- The notification framework enables notification in fineract to work as follows: On event of a notification generation, the notification service is called with a notification message and a list of user identifiers that form the audience of the notification. For events that need sending notifications to a set of users based on the combination of their organization and role, the determination of the audience is done in real time. However, the approach is not optimum since there are multiple fetches from the datasource for the same data.

- A **topic-subscription model** is used to optimize the evaluation of audience. Topics are created for each organization and role combination. When users are added to an organization and assigned roles, they are added as subscribers to the relevant topics. Now, when a notification has to be sent to users having a particular role in an organization, the relevant topic is fetched and its subscribers are set as audience of the message

- From my research on Mifos' notification framework, I came across *Courage Angeh* who had contributed to this framework. She had documented all of her work in a gist that can be found here. Her report entails the **Use Case**, **Implementation Details** and the **Tables** that can be found in this Google Docs

- Based on the above research we may have to change the structure of **Notification Payload** to also accept the parameter for read/unread messages. It will look something like this :

```
@Parcelize
data class NotificationPayload(
    var title: String,
    var body: String,
    var timestamp: String,
    var isRead: Boolean: // if notification was read
) : Parcelable
```

- We can follow this tentative approach to fulfil merchant request-to-pay:

    - When a merchant initiates a request-to-pay from the mobile app, a new record is created in the *request_to_pay* table
    - The creation of a new request-to-pay fires an event (e.g., NEW_REQUEST_TO_PAY_CREATED)
    - An event listener listening to this event generates the notification content (e.g., "You have a new payment request") and retrieves the merchant's entity id
    - The system gets the topic mapped to the merchant's entity id and *member_type* (assuming merchant has a unique member type) from the *topic* table

    - The system retrieves the subscribers to the topic from the *topic_subscriber* table using the topic id
    - For each subscriber, a notification is created and the subscriber (user) id is mapped to the created notification id, stored in the *notification_mapper* table
    - The *notification_mapper* table is queried to get a user's notifications, which can then be sent out

- This is only an idea and will be revised in case my mentor says so. I feel once we have an api and a clear understanding of what to implement then it shouldn't take a lot of time in the front end

2.3 Incorporate Mifos' new design library

- Currently, our Android projects display a variety of user interfaces, with even individual applications experiencing inconsistencies in UI elements. To address this issue and ensure uniformity both within and across our apps, we need to develop a new design library. This will establish a consistent UI framework for all our projects.

- At the time of drafting this proposal, Mifos is utilizing an outdated version of the Mifos UI Library that relies on XML. It's important to acknowledge that this will become obsolete once we transition our project to Compose. Currently, we are defining our reusable components within the :core:ui module of the project.

- The integration of a new UI library will be contingent upon the nature and structure of this library. Should it contain components common across all our Android projects, we could seamlessly replace the existing definitions with those provided by the library. For instance, consider a **Login Screen** that appears in all our projects. By defining this screen within our new library and deploying it, we can ensure consistent UI across all projects.

2.4 Migrate XML to Jetpack compose

- Converting the project to jetpack compose is of utmost priority for this years GSoC. Currently we are using Fragment + Compose approach to migrate our project. We are designing compose UI and then setting the content in our fragments. This typically looks like the following

```
setContent {
    AppTheme {
        ComposeScreen()
    }
}
```

- This method though not wrong, leads you to a code that is bloated and outside of the Compose mentality. Hence we need to start introducing the **Navigation Compose**. The way it will work in our project is that each **NavController** will be associated with a single NavHost composable. The **NavHost** would link the NavController with a navigation graph that specifies the composable destinations that we should be able to navigate between. As we navigate between composables, the content of the NavHost is automatically recomposed. Each composable destination in our navigation graph is associated with a route which is basically a String that defines the path to your composable

- We will need to define a file lets say `AppNavigation.kt` to define screen names and routes for Navigation. This would typically look like:

```
enum class Screen {
    Screen1, // this will be HomeScreen in our project
    Screen2, // this will be PaymentsScreen in our project
    ... // other screens
}
sealed class NavigationItem(val route: String) {
    object Screen1 : NavigationItem(Screen.Screen1.name)
    object Screen2 : NavigationItem(Screen.Screen2.name)
    ... // other screens
}
```

- We then need to define **NavHost** with our screens in `AppNavHost.kt`. A demo implementation would like the following:

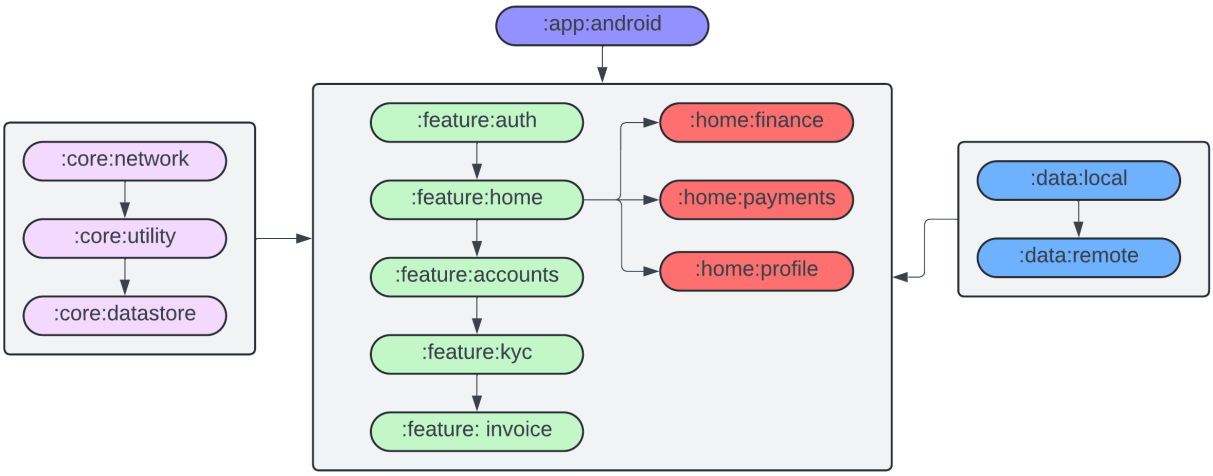
```
@Composable
fun AppNavHost(
    modifier: Modifier = Modifier,
    navController: NavHostController,
    startDestination: String = NavigationItem.Splash.route,
    ... // other parameters
) {
    NavHost(
        modifier = modifier,
        navController = navController,
        startDestination = startDestination
    ) {
        composable(NavigationItem.Splash.route) {
            SplashScreen(navController)
        }
        composable(NavigationItem.Screen1.route) {
            Screen1(navController)
        }
        ... // other composables
}
```

- After this is done we will call *AppNavHost* inside our `MainActivity.kt` file.This will lay the foundation for initial setup and will be crucial when we refactor the **Bottom Navigation** to compose.
- Since we already have a lot of resusable components in our project, migrating the project to compose while integrating NavigationCompose shouldn't take a lot of time. I will communicate any UI revamp ideas with my mentor and will implement the same if he gives me a nod

2.5 Basic implementation of multi-platform

- Implementing *Kotlin Multiplatform* (KMP) is another huge undertaking in this years' GSoC. Once implemented, it will upgrade our native android application to a cross platform application

- Our project is currently broken down into a few modules and I believe we can break it down further based on different features. Each feature module handles its own scope of responsibility, usually covering the functionality of a single screen or a closely related group of screens. Right now we only have `:feature:auth` module and we can extend it to different such features. We can create a *database module* inside of core to find all the necessary classes for our local database

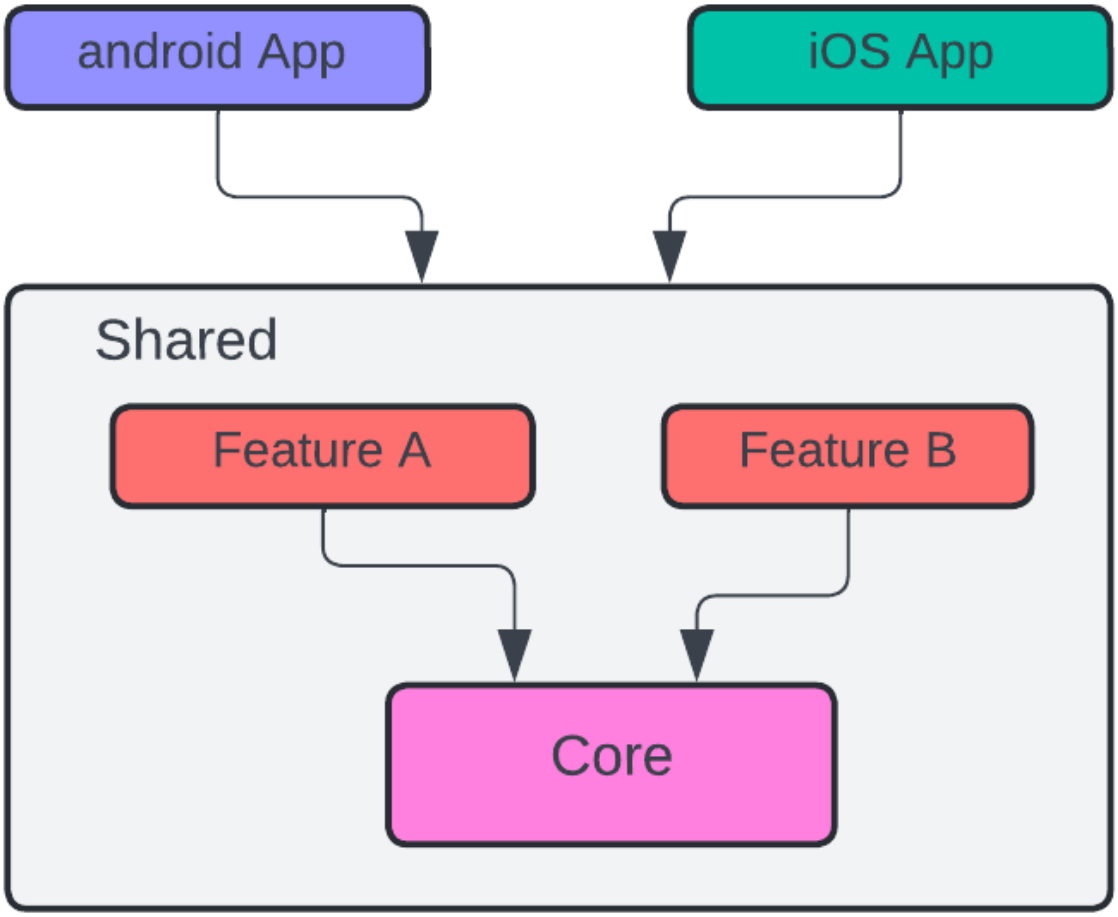


- To ensure cross platform application KMP shares the business logic between the platforms and it is of utmost importance that we migrate anything java related to kotlin. That includes migrating *SharedPreference* to **Data Store**, *Retrofit* to **Ktor**, *Hilt* to **Koin** and *DbFlow* to **SqlDelight**. We also have the option to choose between Koin & Kotlin-Injection and also between SqlDelight & Realm. The end purpose of their alternatives are the same and hence I will discuss the better alternative with my mentor before I start implementing any of them

- To make our code cross-platform we will have to consider the following points:

1. *Decide what code to make cross platform:*
   The idea is to share what we want to reuse as much as possible and hence the business logic is a great candidate for reuse because it is often the same in android & iOS. In our codebase login, mobile_verify, signup, social_signup, passcode, data, model, network among other packages & modules will share the same business logic in both android & iOS and as such we can make this cross-platform

2. *Create a shared module for cross-platform code:*
   The cross-platform code that is used for both iOS and Android is stored in the shared module. The Kotlin Multiplatform plugin provides a special wizard for creating such modules. Since we have multiple modules we should go ahead and create an **Umbrella** module for our multiplatform project. Both androidApp and iosApp depend only on the shared module using exactly the same mechanism as for the single KMM module



The project heirarchy should look something like this:

```
``` kotlin
:androidApp // native app
:iosApp // native app
:shared // KMM umbrella
:shared:featureA
:shared:featureB
:shared:core
```
```

Because the native app depends only on the shared Umbrella module we need to allow an app to access submodules hidden under this Umbrella. First of all we should decide which submodules can be accessed from the native app and which ones should not be accessible. We can have some modules which are used only internally in the KMM part and it doesn't make sense to make them available from the application's code. All the submodules which we want to expose to the app should be added to the **shared/build.gradle.kts** script as an api dependency. Similarly we add the modules which should not be accessible from the native app, but this time we use an implementation dependency.

Lastly we need one more step to make submodules accessible from the iOS application. In the cocoapods setup of the shared module we need to add an export for each module which was connected as an api to the Umbrella. This would look something like this:

```
kotlin {
    sourceSets {
        val commonMain by getting {
            dependencies {
                // Accessible by the native app
                api(project(":shared:featureA"))
                api(project(":shared:featureB"))
                api(project(":shared:core"))
                // Accessible only in the KMM part
                implementation(project(":shared:analytics"))
                implementation(project(":shared:configuration"))
            }
        }
    }
    cocoapods {
        framework {
        baseName = "shared"
        // Accesible by the iOS app
        export(project(":shared:featureA"))
        export(project(":shared:featureB"))
        export(project(":shared:core"))
        }
    }
}
```

3. *Make the business logic cross-platform:*
   Once we have decided on what logic to share between cross platform we can then move ahead and move it to the shared module we can call commonMain. This will ensure that our Android Application is now cross platform

4. *Make cross-platform application work on iOS:*
   To make our project run on iOS app we will need to perform the following steps:

   - Create an iOS project in Xcode
   - Connect the framework to our iOS project
   - Use the shared module from swift

Once we have made our application cross-platform, we can move ahead and: - Add dependencies on multiplatform libraries - Add Android dependencies - Add iOS dependencies

2.6 Implement Playstore release github action pipeline

- Right now we are pushing all of our code to development. I suggest we create another branch lets call it *release* whose purpose would be to depoy product to playstore and this would ensure a proper flow of CI/CD

- **Fastlane** is the easiest way to automate beta deployments and releases for your iOS and Android apps. It handles all tedious tasks, like generating screenshots, dealing with code signing, and releasing your application. Initially we will need a *Google Play Credential file (.json)* for the Fastlane to deploy our apps and assuming that Mifos would have their own google playstore account we will easily able to generate this file

- We can declare various lanes in Fastfile which can have different behaviours or simply we can call them tasks. Lets say we have to deploy our application for the **RELEASE** track. Then our lane would look like:

```
default_platform(:android)

platform :android do

  desc "Deploy a beta version to the Google Play"
  lane :beta do
    gradle(task: "clean bundleRelease")
    upload_to_play_store(track: 'beta')
  end

end
```

- We require a **Keystore file (.jks)** for signing APK/App Bundle before publishing app to the Google Play. Once all of these are done we will create a worflow file **release.yml** that will look like the following:

```
name: Deploy

on:
    ## add branches here for eg. release/beta
jobs:

    # add runner other jobs here

      - name: Configure Keystore
        run: |
        echo "$ANDROID_KEYSTORE_FILE" > keystore.jks.b64
        base64 -d -i keystore.jks.b64 > app/keystore.jks
        echo "storeFile=keystore.jks" >> keystore.properties
        echo "keyAlias=$KEYSTORE_KEY_ALIAS" >> keystore.properties
        echo "storePassword=$KEYSTORE_STORE_PASSWORD" >> keystore.properties
        echo "keyPassword=$KEYSTORE_KEY_PASSWORD" >> keystore.properties
        env:
        ANDROID_KEYSTORE_FILE: ${{ secrets.ANDROID_KEYSTORE_FILE }}
        KEYSTORE_KEY_ALIAS: ${{ secrets.KEYSTORE_KEY_ALIAS }}
        KEYSTORE_KEY_PASSWORD: ${{ secrets.KEYSTORE_KEY_PASSWORD }}
        KEYSTORE_STORE_PASSWORD: ${{ secrets.KEYSTORE_STORE_PASSWORD }}

      - name: Create Google Play Config file
        run : |
        echo "$PLAY_CONFIG_JSON" > play_config.json.b64
        base64 -d -i play_config.json.b64 > play_config.json
```

```
        env:
        PLAY_CONFIG_JSON: ${{ secrets.PLAY_CONFIG_JSON }}

      - name: Distribute app to Release track # Executing our release lane
        run: bundle exec fastlane release
```

- While deploying we need to keep in mind that we will have a KMP project by the time we finish it and hence we will have to modify our existing pipeline to add support for iOS part of the app as well whose runner will be *macos-latest*. Moreover since the idea is to implement **Multiplatform** and not just **Multimobile** jobs related to windows, linux will also have to be developed. A general implementation would look like this :

```
jobs:
  releaseLinuxAndroidWeb:
    name: Release - Android, Linux, Web
    runs-on: ubuntu-latest

    steps:

      # checkout repository and JDK setup here

      - name: Grant Permission to Execute Gradle and scripts
        run: |
          chmod +x gradlew
          chmod +x buildApps.sh

      - name: Build with Gradle
        uses: gradle/gradle-build-action@v2

      - name: Build App
        run: ./buildApps.sh

      - name: Attach Android and Linux App 🚀
        uses: softprops/action-gh-release@v1
        with:
          files: |
            distributions/chakt-android.apk
            distributions/chakt-linux-x64.jar

      - name: Publish Web app 🚀
        uses: JamesIves/github-pages-deploy-action@v4.3.3
        with:
          branch: gh-pages
          folder: distributions/chakt-web
```

2.7 Update wallet framework to be make use of Mifos' Android SDK

- Currently the mobile wallet is consuming the native code to make REST calls which leads to a significant amount of boiler plate code. This however can be reduced by consuming the Android SDK developed by mifos. This integration can offer considerable benefits in terms of development efficiency, as it eliminates much of the repetitive and time-consuming coding work associated with making REST calls.

- To integrate this we will have to add dependencies for these 2 SDKs:

```
    dependencies {
        implementation ("com.github.openMF:mifos-android-sdk-arch:$sdk_Version")
        implementation ("com.github.openMF:fineract-
client:$fineractClientVersion")
    }
```

- All of the retrofit calls are written in Mifos Android SDK and all the corresponding data classes are present in Fineract Client and as such complement each other. These two SDK have different models and hence we will have to create **Mappers** for converting one model to another

- However, Fineract Client is still in *java* and will be a hurdle in our plans of KMM implementation. So we will have to migrate this to Kotlin and in the process replace all of the payload class with data class. Moreover all the **REST** calls are returning an *Observable* which are subsequently being handle by **RxJava** and hence keeping in mind our plans for KMM we will have to use Kotlin **Coroutines** and **Flows** instead

- Another point to consider before we start implementing this is Ktor. To add support for KMM it will be better if we get rid of Retrofit on the way. The above mentioned points should be pondered upon before we decide to integrate it into our project. If the need arises then I will migrate the SDK to Java, this of course is a stretch goal and will be worked upon only after my discussion with the assigned mentor

2.8 Complete migration of Java code to Kotlin

- In the process of migrating to Kotlin, we significantly enhance our codebase by adopting Kotlin's more concise syntax, which notably reduces boilerplate code. Furthermore, Kotlin's built-in null safety and powerful features like coroutines and flows add substantial value to our development process. However, our ultimate objective is to transition our project to **Kotlin Multiplatform** (KMM). To achieve this, it's imperative that our entire codebase is converted to Kotlin. At the time of drafting this proposal, our entire project was migrated to kotlin and this entire migration process was really enriching in terms of kotlin's capability when it comes to android development

- I had migrated a huge chunk of codebase to kotlin and some of those PRs are mentioned below:

1. PR #1505: Client Usecase package from java to kotlin
2. PR #1503: KYC Usecase package from java to kotlin
3. PR #1501: Invoice Usecase package from java to kotlin
4. PR #1501: Notification Usecase package from java to kotlin

2.9 Improving the security framework

- **Mifos Pay** is a banking application that communicates with a lot of endpoints and third party libraries and as such we need to lay emphasis on its security framework. Some of the points to make our app secure are :
- *Securing API Keys using Android NDK*: There are different ways in which API keys can be kept hidden and one such way of storing keys in the native C/C++ class and accessing them in our Java classes. Doing this will add an extra layer of obfuscation to the keys. **Mifos Passcode** has already inculcated this feature and the same can be replicated by following this article
- *Don't process any payments on a rooted device*: Mifos Pay will have in app payment feature besides some other critical tasks. Automatically disable these features on a rooted device. Because the rooted device can change your code at runtime and alter the behavior of it. We can obviously check for such devices using the code here
- *Remove unecessary logs*: Debugging code is fine for a debug build but I will make sure to remove such logs that contains *service urls*, *response body*, *usernames*, *crashes* etc from the release apk otherwise everyone can see these logs on their computer and this logs could contain sensitive data that if it's misused, it would be a nightmare for us. Instead of removing logs manually from each part of the code we can use **Timber** which will make you enable or disable logs in a centralized place
- *Code Obfuscation*: We have already enabled Proguard/R8 in our project but it will be better to use it with DexGuard which is its specialized sibling that can additionally encrypt/obfuscate the strings and classes for us. This will save us from manually obfuscating our string with Base64 encoding

2.10 Exploring PoC Architecture for Open Wallet Foundation Alignment

- The OWF aims to set best practices for digital wallet technology through collaboration on standards-based OSS components that issuers, wallet providers and relying parties can use to bootstrap implementations that preserve user choice, security and privacy all of which aligns with Mifos' Initiative of advancing the **Sustainable Development Goal** of **No Poverty**
- Mifos already has a small project titled Alignment with Emerging Open Wallet Standards whose objective is to research the different open wallet standards emerging from the Open Wallet Foundation, GovStack wallet building block, and utilize other digital public goods for identity including Inji from MOSIP and the Gluu Project
- The above idea is no way different from what we have to do and hence this reseach will compliment mine in every possible way. During the GSoC I will explore the current status of the specifications, standards and design a POC architecture for mobile wallet that aligns with the standards and specifications in their current state

3. Contributions to Mifos

Ever since GSoC 2023 concluded, I have kept contributing to Mifos and this contribution is not just limited to **Mobile Wallet** but also extends to **Mifos Mobile** and **Mifos Passcode**

Below are links to some of my notable contributions at the time of submitting this proposal :

Merged Pull Requests for Mobile Wallet

1. PR #1591: Migrated Bank Account Detail Screen to compose
2. PR #1579: Migrated Finance Screen to compose
3. PR #1548: Migrated Payments Screen to compose
4. PR #1547: Migrated Profile Screen to compose
5. PR #1508: Migrated Settings Screen to compose
6. PR #1538: Migrated FAQ Screen to compose
7. PR #1426: Migrated Login Screen to compose
8. PR #1595: Migrated EditProfile Screen to compose
9. PR #1573: Introduced Detekt for static code analysis
10. PR #1524: Rectified Github Workflow of the project
11. PR #1526: Rectified Authentication Logic of the project
12. PR #1462: Fixed app crash when selecting FAQ
13. PR #1510: API package from java to kotlin
14. PR #1540: Checkstyle for all modules

- I have a total of **36 Merged Pull Requests** in mobile wallet. You can find all of my merged PRs here

Merged Pull Requests for Mifos Mobile

1. PR #2436: Migrated Login Screen to compose
2. PR #2456: Migrated User Profile Screen to compose
3. PR #2459: Migrated Registration Screen to compose
4. PR #2257: Migrated AboutUs Screen to compose
5. PR #2307: Fixed Unexpected Biometric Prompt
6. PR #2499: Updated endpoint for debug
7. PR #2430: Refactored Client Charges to stateflow
8. PR #2499: Update readme with current version of the project

- I have a total of **83 Merged Pull Requests** in mifos mobile alone. You can find all of my merged PRs here

Open Pull Requests

- At the time of drafting this proposal I dont have any PR across any of Mifos's project
- I intend to continue contributing to the codebase even after submitting my proposal and expect that there may be changes made to the Pull Requests that I have opened. Therefore, I am providing the links to those PRs here

Issues Reported

- I had opened a total of **15 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

Besides contributing to the various projects, I have also dedicated my time to reviewing Pull Requests raised by my fellow contributors and helping them onboard in our open source community. Ever since last years GSoC concluded, I have been actively connected to Mifos

4. Week Wise Breakdown

4.1 Community Bonding Period (1 May - 26 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 amendments; it would be better to go over these early

Week 2

- Go Through Mobile Wallet codebase
- Address issues that can be a hurdle during the GSoC
- Go through the Open Banking API and related documentations

Week 3

- Discuss the working of the existing application with the mentor
- Go over the new design in detail and ask for changes and suggestions
- Migrate the project to kotlin

4.2 Phase 1 (27 May - 12 July)

Week 4

- I will start off by picking up screens that are yet to be migrated to **compose**
- In the process, I shall remove fragments and handle navigation with compose itself
- I am already contributing to this task and I assume we can be done within a week

Week 5

- Before moving to KMP, I would like to integrate **Mifos's Android SDK**
- Right now the SDK uses retrofit so we wont be able to use it in our shared module
- If we go with this then we will have to create an abstraction over it, and provide the native/iOS implementation

Week 6

- I will start off with **KMP** implementation in this week
- Will make use of umbrella module so that iOS can also share the business logic
- Migrate from retrofit to ktor & also from Hilt to Koin

Week 7

- Migrate from DbFlow to SqlDelight
- Optionally migrate from SharedPreference to DataStore
- Start writing UI for iOS part

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 (12 July - 26 Aug)

Week 10

- Integrate Mifos's **Notification Framework** based on my research
- Last year we had sessions on PaymentHub and if something similar happens this year then it will be beneficial
- Connect with backend team to understand the requirement

Week 11

- Integrate latest version of **Payment Hub** in mobile wallet project
- I will consult with my mentor the API endpoints that needs to be hit for transactions, registrations, identification and so on before I start with the integration
- All of this will be done keeping in mind our use cases for G2P

Week 12

- I will research as to how I can improve the existing **security framework** of our project
- Being a banking application I know the risks it possess and I will discuss with my mentor before I start working on this project
- Discuss about **Open Wallet Foundation** with my mentor and how our existing application fits in this movement

Week 13

- Assuming that by the end of this year's GSoC we have published our **UI library** then we can start using them in the android section of our project
- Create a Playstore release github action pipeline with jobs to check the iOS part of our app

Week 14 and Rest of Phase 2

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

Priority Chart

Critical	XML to Compose	KMM implementation	Use Mifos Android SDK
High	Notification Framework	Payment Hub Integration	Pipeline to release app
Medium	Security Framework	Discussion around OWF	Mifos UI library integration

4.4 Post Phase 2 (After Aug 26)

- 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 two 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, jetpack compose, support libraries, version control, Networking Services, Firebase, Dependency Injection etc

Moreover, I had contributed to Mifos last year as well and successfully completed the project that I was assigned. Furthermore, I am now accustomed to the mobile wallet project since I have been contributing to it for the last 3 months

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
    - Collecting, Storing and Analyzing data using tools like SQL and Python
    - Learning about the fundamentals of ML that included supervised and unsupervised learning,neural networks and deep learning
    - Learning 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 of course by contributing to Open Source Projects.

9. My Projects

1. CardSwipeLibrary: *Source Code*

- Streamlined the development process for mobile applications by providing a ready-to-use library, reducing the time and effort required for implementing complex swipe gestures from scratch
- Empowered developers to create dynamic and interactive UI elements effortlessly, fostering innovation and creativity in mobile app design and development
- This Library was built entirely in compose and is also the very first library that I published

2. Cyclofit: *Source Code*

- Cyclofit is a safety and health monitoring system for cyclists
- It tracks the heart rate,calories burnt, distance covered 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

3. 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 Source 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
4. *Mifos Mobile*:
    - Contributed to this as a part of my GSoC last year and primarily worked on refactoring the project
    - Introduced *MVVM*, *Hilt*, *Coroutines*, *Flows* among other enhancements
    - My contributions for Mifos Mobile: check here

5. *Mifos Passcode*:

   - Passcode library being used across Mifos' android project
   - Migrated the library from java to kotlin & also migrated it to compose
   - My contributions for Mifos Passcode: 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 Mobile Wallet 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

15. Previous Participation in GSoC

Yes, I had participated in Google Summer of Code 2023 with this very organization and I would love to be a part of Mifos this year as well

16. Application to multiple orgs

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