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

I wish to contribute to Mifos Mobile 6.0 - Mobile Banking App

Abstract

- Replace API layer from self-service Fineract APIs to Open Banking APIs
- Complete Support for customer support/chat via RocketChat
- Integration with an external payment system (Mojaloop, mPesa) via our payment hub.
- Migrate Dagger to Hilt
- Migrate from MVP to MVVM-Clean architecture (Not the whole project, maybe 20-30%)
- The basic integration of the Navigation Graph in the project
- The basic integration of Coroutines
- Continue adding unit tests for Data Layer and UI Layer
- Cover all the screens with UI tests
- Improve Githhub workflows and add jobs to run Unit and UI tests

2. Implementation of the idea

2.1 Migration from MVP to MVVM

The MVVM architecture is designed to store and manage UI-related data in a lifecycle conscious way. Configuration changes such as screen rotations are handled properly by ViewModels.It separates the responsibilities of the user interface from the business logic, which makes the code easier to maintain and test.Because the ViewModel is decoupled from the View, it can be unit tested without the need for a graphical user interface (GUI). This makes testing faster and more reliable.MVVM leverages data binding to establish a connection between the View and ViewModel. This means that changes made to the ViewModel are automatically reflected in the View, and vice versa. This reduces the amount of boilerplate code needed to keep the UI up-to-date.Because the ViewModel contains the business logic, changes to the UI can be made without affecting the underlying data or business logic. This makes it easier to maintain and extend the code over time.

2.1.1 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. Below is the code for SplashViewModel.kt

```kotlin
import android.content.Intent
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.mifos.mobile.passcode.utils.PasscodePreferencesHelper
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

class SplashViewModel(private val passcodePreferencesHelper:
PasscodePreferencesHelper) : ViewModel() {
    var intent: Intent?=null
    val liveData : MutableLiveData<String> = MutableLiveData()
    fun getDestinationIntent() {
        CoroutineScope(Dispatchers.IO).launch{
            if (passcodePreferencesHelper.passCode.isNotEmpty()) {
                liveData.postValue("Yes")
            } else {
                liveData.postValue("No")
            }
        }
    }
}
```

Below is the code for SplashViewModelFactory.kt:

```kotlin
import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import com.mifos.mobile.passcode.utils.PasscodePreferencesHelper

class SplashViewModelFactory(private val passcodePreferencesHelper:
PasscodePreferencesHelper) : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        return SplashViewModel(passcodePreferencesHelper) as T
    }
}
```

Refactored code for SplashActivity.kt :

```kotlin
class SplashActivity : BaseActivity() {

    private lateinit var viewModel: SplashViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        activityComponent?.inject(this)
        val passcodePreferencesHelper = PasscodePreferencesHelper(this)
        viewModel = ViewModelProvider(this,
SplashViewModelFactory(passcodePreferencesHelper)).get(
            SplashViewModel::class.java)

        viewModel.getDestinationIntent()
        viewModel.liveData.observe(this){
            if (it.equals("Yes")){
                val i=(Intent(this,PassCodeActivity::class.java))
                i.putExtra(Constants.INTIAL_LOGIN,true)
                startActivity(i)
                finish()
            }
            else {
                startActivity(Intent(this,LoginActivity::class.java))
                finish()
            }
        }
    }
}
```

2.1.2 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.

Below is the LocationsViewModel.kt :

```kotlin
import androidx.lifecycle.ViewModel
import com.google.android.gms.maps.CameraUpdateFactory
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.model.LatLng
import com.google.android.gms.maps.model.MarkerOptions
import org.mifos.mobile.MifosSelfServiceApp.Companion.context
import org.mifos.mobile.R

class LocationsViewModel : ViewModel() {

    private val headquarterLatLng = LatLng(47.61115, -122.34481)

    fun addMarker(googleMap: GoogleMap) {
        googleMap.addMarker(
            MarkerOptions().position(headquarterLatLng)
            .title(context?.getString(R.string.mifos_initiative)))
    }

    fun addAnimationToHeadquarter(googleMap: GoogleMap) {

googleMap.animateCamera(CameraUpdateFactory.newLatLngZoom(headquarterLatLng,
16.0f))
    }
}
```

Below is the modified LocationsFragment.kt :

```kotlin
import android.os.Bundle
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.lifecycle.ViewModelProvider
import butterknife.BindView
import butterknife.ButterKnife
import com.google.android.gms.maps.*
import org.mifos.mobile.R
import org.mifos.mobile.ui.fragments.base.BaseFragment
import org.mifos.mobile.viewModels.LocationsViewModel

class LocationsFragment : BaseFragment(), OnMapReadyCallback {

    private lateinit var viewModel: LocationsViewModel

    @BindView(R.id.map)
    lateinit var mapView: MapView
```

```kotlin
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {
        val rootView = inflater.inflate(R.layout.fragment_locations, container,
false)
        ButterKnife.bind(this, rootView)

        mapView.onCreate(savedInstanceState)
        mapView.onResume()

        try {
            MapsInitializer.initialize(requireContext())
        } catch (e: Exception) {
            Log.d(LocationsFragment::class.java.simpleName, e.toString())
        }

        mapView.getMapAsync(this)

        viewModel = ViewModelProvider(this).get(LocationsViewModel::class.java)

        return rootView
    }

    override fun onMapReady(googleMap: GoogleMap) {
        viewModel.addMarker(googleMap)
        viewModel.addAnimationToHeadquarter(googleMap)
    }
    companion object {
        fun newInstance(): LocationsFragment {
            return LocationsFragment()
        }
    }
}
```

2.2 Basic Integration of Navigation Graph

- 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 `replaceFragment(SettingsFragment.newInstance(), false,R.id.fragmentContainerView)` in the SettingsActivity.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 **RegisterationActivity** .But the role of the **RegisterationActivity** is just to replace the fragment which is **RegisterationFragment**. We can get rid of the two activities and replace them by just one activity let's say **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.

2.3 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

2.4 My Ideas for the app

There are certain features that are not currently present in the ideas list but in my opinion would look good and also be in accordance with the new android standards.

2.4.1 Migrating from Butter Knife to View 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 your application. However, Butterknife still had its own issues such as null safety and speed. Now introduce View Binding: a low code, null safe, and fast binding tool! View Binding is a 1st party tool built by the Google Android Development team which helps reference and easily manage views within an app

- Butterknife is a 3rd party library made to address the issue of using findViewById(...) functions to reference and interact with views. It helped reduce boilerplate code but also had to set up a @Bind annotation every time you wanted to interact with a view. Then View Binding was introduced starting at Gradle version 3.6.This single binding variable allows us to access every view, set up event listeners, and any other functionality we'd do with Butterknife.

- On top of Butterknife being deprecated, another reason to switch is because View Binding is compile time safe and builds fast. As for findViewById(...), this way of referencing views will lead to so much unnecessary code that could be replaced with a View Binding variable.

- First step would be to upgrade the gradle version to 7.3.1 from 7.2.0 and then adding the dependency as:

> android { ... buildFeatures { viewBinding = true } }
> Below is LoginActivity.kt after its migration from Butter Knife to View Binding:

```
package org.mifos.mobile.ui.activities

import android.content.Intent
import android.os.Bundle
import android.view.View
import android.widget.Toast
import org.mifos.mobile.R
import org.mifos.mobile.databinding.ActivityLoginBinding
import org.mifos.mobile.models.payload.LoginPayload
```

```kotlin
import org.mifos.mobile.presenters.LoginPresenter
import org.mifos.mobile.ui.activities.base.BaseActivity
import org.mifos.mobile.ui.views.LoginView
import org.mifos.mobile.utils.Constants
import org.mifos.mobile.utils.Network
import org.mifos.mobile.utils.Toaster

import javax.inject.Inject

/**
 * @author Vishwajeet
 * @since 05/06/16
 */
class LoginActivity : BaseActivity(), LoginView {

    @JvmField
    @Inject
    var loginPresenter: LoginPresenter? = null

    private lateinit var binding:ActivityLoginBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        activityComponent?.inject(this)
        setContentView(R.layout.activity_login)
        loginPresenter?.attachView(this)
        binding=ActivityLoginBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)
        binding.btnLogin.setOnClickListener {
            onLoginClicked()
        }
        binding.btnRegister.setOnClickListener {
            onRegisterClicked()
        }

    }

    /**
     * Called when Login is user has successfully logged in
     */
    override fun onLoginSuccess() {
        loginPresenter?.loadClient()
    }

    /**
     * Shows ProgressDialog when called
     */
    override fun showProgress() {
        showProgressDialog(getString(R.string.progress_message_login))
    }

    /**
     * Hides the progressDialog which is being shown
```

```kotlin
    */
    override fun hideProgress() {
        hideProgressDialog()
    }

    /**
     * Starts [PassCodeActivity]
     */
    override fun showPassCodeActivity(clientName: String?) {
        showToast(getString(R.string.toast_welcome, clientName))
        startPassCodeActivity()
    }

    /**
     * It is called whenever any error occurs while executing a request
     *
     * @param errorMessage Error message that tells the user about the problem.
     */
    override fun showMessage(errorMessage: String?) {
        showToast(errorMessage!!, Toast.LENGTH_LONG)
        binding.llLogin?.visibility = View.VISIBLE
    }

    override fun showUsernameError(error: String?) {
        binding.tilUsername?.error = error
    }

    override fun showPasswordError(error: String?) {
        binding.tilPassword?.error = error
    }

    override fun clearUsernameError() {
        binding.tilUsername?.isErrorEnabled = false
    }

    override fun clearPasswordError() {
        binding.tilPassword?.isErrorEnabled = false
    }

    /**
     * Called when Login Button is clicked, used for logging in the user
     */
    fun onLoginClicked() {
        val username = binding.tilUsername?.editText?.editableText.toString()
        val password = binding.tilPassword?.editText?.editableText.toString()
        if (Network.isConnected(this)) {
            val payload = LoginPayload()
            payload.username = username
            payload.password = password
            loginPresenter?.login(payload)
        } else {
            Toaster.show(binding.llLogin,
 getString(R.string.no_internet_connection))
        }
```

```
    }

    fun onRegisterClicked() {
        startActivity(Intent(this@LoginActivity,
RegistrationActivity::class.java))
    }

    override fun onDestroy() {
        super.onDestroy()
        loginPresenter?.detachView()
    }

    /**
     * Starts [PassCodeActivity] with `Constans.INTIAL_LOGIN` as true
     */

    private fun startPassCodeActivity() {
        val intent = Intent(this@LoginActivity, PassCodeActivity::class.java)
        intent.putExtra(Constants.INTIAL_LOGIN, true)
        startActivity(intent)
        finish()
    }
}
```

Similar approach could be applied on the fragments as well .

2.4.2 Adding Option to Change Connection setting from login page

- Currently the 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

3. Week Wise Breakdown

3.1 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

Phase 1

Phase 2

4. 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.

5. Current area of study

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

6. 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)

7. 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.

8. Project (Source Code I have written)

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.

9. Gitter Channel

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

10. Other Open Source Contributions

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

1. Catroid

2. Paintroid

3. Anki-Droid

4. Dare2Change

11. 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 haven't particularly worked with AngularJS but I am familiar with JavaScript and I have also used React.js framework to build a Path finding visualizer,So If needed I can learn AngularJS at a quick pace.

12. Other Commitments

1. College Exams
2. Course projects

These are going to be an integral part of my 6th semester activities. They are not going to affect my GSOC efforts.
I will be able to deliver my commited number of hours per week without much hindrance.

13. Contributions to Mifos

Below are the links to my contributions :

1. PR #1901: Finger Print Authentication in the Passcode Activity
2. PR #1899: Fixes App Crash if you click on the submit button without selecting product ID
3. PR #1930: Fixes 'Change Passcode' in Settings
4. PR #1929: Fixes background color of Passcode with the theme
5. PR #1927: Fixes the behaviour of filters throught the app
6. PR #1926: Modified Error message in case of wrong endpoint
7. PR #1912: Fixes Scrolling when in landscope mode
8. PR #1910: Fixes App Crash when orientation is changed to landscape mode
9. PR #1908: Fixes duplication in Material Auto Complete Text View
10. PR #1905: Allows User to enter the correct passcode even after three unsuccessful tries
11. PR #1904: Added Country Code Picker in the Signup form

14. What motivates 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