Google Summer of Code

GSoC'25 PROPOSAL - The Mifos Initiative

Mifos Mobile 7.0 - Mobile Banking App

Organization: Mifos Initiative

Project Name: Mifos Mobile 7.0 – Mobile Banking App

Candidate Name: Nagarjuna Banda

Expected Project Size: 350 hours

Mentors:

- Rajan Maurya
- Ahmad Jawid Muhammad
- Garvit Agarwa
- Saksham Handu
- Paras Dhama

Table of Contents

1. Project Idea 3
2. Implementation Details 4
   2.1 Replace API layer from self-service Fineract APIs to self-service middleware layer 4
   2.2 Implement new look and feel based on modular UI components 6
   2.3 Integration with an external payment system (Mojaloop, mPesa) via our PH-EE 18
   2.4 Complete migrate Kotlin multi-module to Kotlin multiplatform 26
   2.5 Make sure fineract-client-sdk implemented properly and have no bugs 32
   2.6 Continue adding unit tests for Data Layer and UI Layer 35
   2.7 Cover all the screens with UI test 43
   2.8 Implemented Playstore release github action pipeline 45
   2.9 Improve Github workflows and add jobs to run Unit and UI test 51
   2.10 Add CI to build APK and code analysis. 56
3. Work Wise Breakdown 62
   3.1 Community Bonding Period (8 May - 1 June) 62
      Week 1 62
      Week 2 63
      Week 3 63
   3.2 Phase 1 (2 June - 18 July) 63
      Week 4 63
      Week 5 63
      Week 6 64
      Week 7 64
      Week 8 and Rest of Phase 1 64
   3.3 Phase 2 (19 July - 25 Aug) 64
      Week 10 64
      Week 11 64
      Week 12 65
      Week 13 65
      Week 14 and Rest of Phase 2 65
   3.4 Post Phase 2 (After Aug 26) 66
4. Why am I the right person ? 66
5. Current area of study 67
6. Contact Information 67

7. Career Goals 68
8. My Projects 68
9. Slack Channel 69
10. Interaction with mentor 69
11. Experience with Angular/Java/Spring/Hibernate/MySQL/Android 70
12. Other Commitments 70
13. Deployed and tested the mobile Applications 70
14. Contributions to Mifos 70
15. What motivates me to work with Mifos for GSoC 71
16. Previous Participation in GSoC 72
17. Application to multiple orgs 72

1. Project Idea

Abstract

- The Mifos Mobile 7.0 - Mobile Banking App project aims to modernize and extend the self-service mobile banking experience for financial institutions using Mifos X. A major part of this effort is migrating the app from a multi-module Kotlin structure to Kotlin Multiplatform (KMP).
  This will allow the app to share business logic across Android and iOS while maintaining a native UI for each platform, increasing code maintainability and reducing duplication. Additionally, the app's API layer will be replaced, shifting from the existing Fineract self-service APIs to a self-service middleware layer that leverages the Open Banking API standard. This transition enhances security, scalability, and flexibility, ensuring compliance with global financial regulations and enabling seamless third-party integrations.

- Another critical focus area is the integration of external payment systems, particularly Mojaloop and mPesa, through the Payment Hub Enterprise Edition (PH-EE). This will allow users to make seamless transactions across different financial institutions and mobile money providers, expanding accessibility for underbanked and unbanked populations. Alongside this, the app's UI will be modularized, making it more adaptable for financial institutions that wish to customize their branding and features. A proper implementation of the fineract-client-kmp-sdk will also be ensured, eliminating bugs and enhancing stability for API interactions.

- To maintain reliability and ensure smooth development, a significant focus will be on expanding unit and UI test coverage for the Data Layer and UI Layer. Automated GitHub workflows will be improved by adding jobs to run unit tests, UI tests, and static code analysis. A CI/CD pipeline will be set up for building APKs, running tests, and preparing Play Store releases through GitHub Actions. By implementing these enhancements, the project will provide an extensible, secure, and feature-rich mobile banking solution, empowering financial institutions to offer their customers a seamless self-service banking experience

2. Implementation Details

2.1 Replace API layer from self-service Fineract APIs to self-service middleware layer

- **selfservice-plugin** This is the plugin we are going to replace with.
- Current we have the following API. We are making request for this (Registration, account, clients, transfers) and for remaining operations.

```
class BaseURL {
    val url: String
        get() = PROTOCOL_HTTPS + API_ENDPOINT + API_PATH


    val defaultBaseUrl: String
        get() = PROTOCOL_HTTPS + API_ENDPOINT

    fun getUrl(endpoint: String): String {
        return endpoint + API_PATH
    }

    companion object {
        const val API_ENDPOINT = "gsoc.mifos.community"
        const val API_PATH = "/fineract-provider/api/v1/"
        const val PROTOCOL_HTTPS = "https://"
    }
}
```

- For Registration We are calling this path registration

```
interface RegistrationService {
    @POST(ApiEndPoints.REGISTRATION)
    suspend fun registerUser(@Body registerPayload: RegisterPayload?): HttpResponse

    @POST(ApiEndPoints.REGISTRATION + "/user")
    suspend fun verifyUser(@Body userVerify: UserVerify?): HttpResponse
}
```

- Now we have to replace this with self service API.

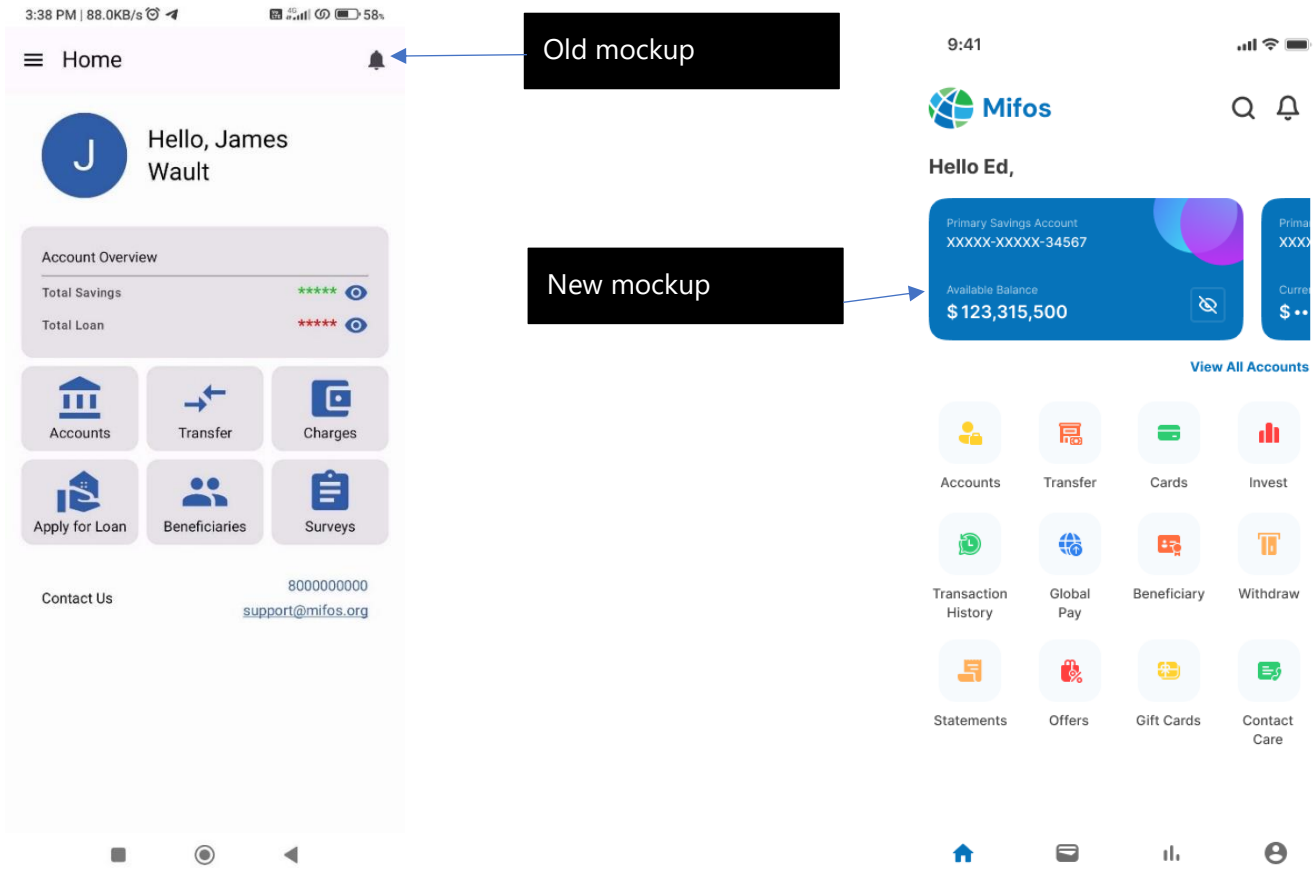
```
class BaseURL {
    val url: String
        get() = PROTOCOL_HTTPS + API_ENDPOINT + API_PATH

    val defaultBaseUrl: String
        get() = PROTOCOL_HTTPS + API_ENDPOINT

    fun getUrl(endpoint: String): String {
        return endpoint + API_PATH
    }

    companion object {
        const val API_ENDPOINT = "tt.mifos.community"
        const val API_PATH = "/fineract-provider/api/v1/self"
        const val PROTOCOL_HTTPS = "https://"
    }
}
```

- As we are using Koin for dependency injection We can change here once.
- Currently, the Mifos Mobile 7.0 app interacts directly with Apache Fineract's self-service APIs to fetch user data, process transactions, and manage banking operations. These APIs allow clients to perform actions like checking account balances, transferring funds, and making repayments.
- However, instead of directly using Fineract's self-service APIs, we can introduce a self-service middleware layer between the mobile app and the core banking system. This middleware layer will act as an intermediary.

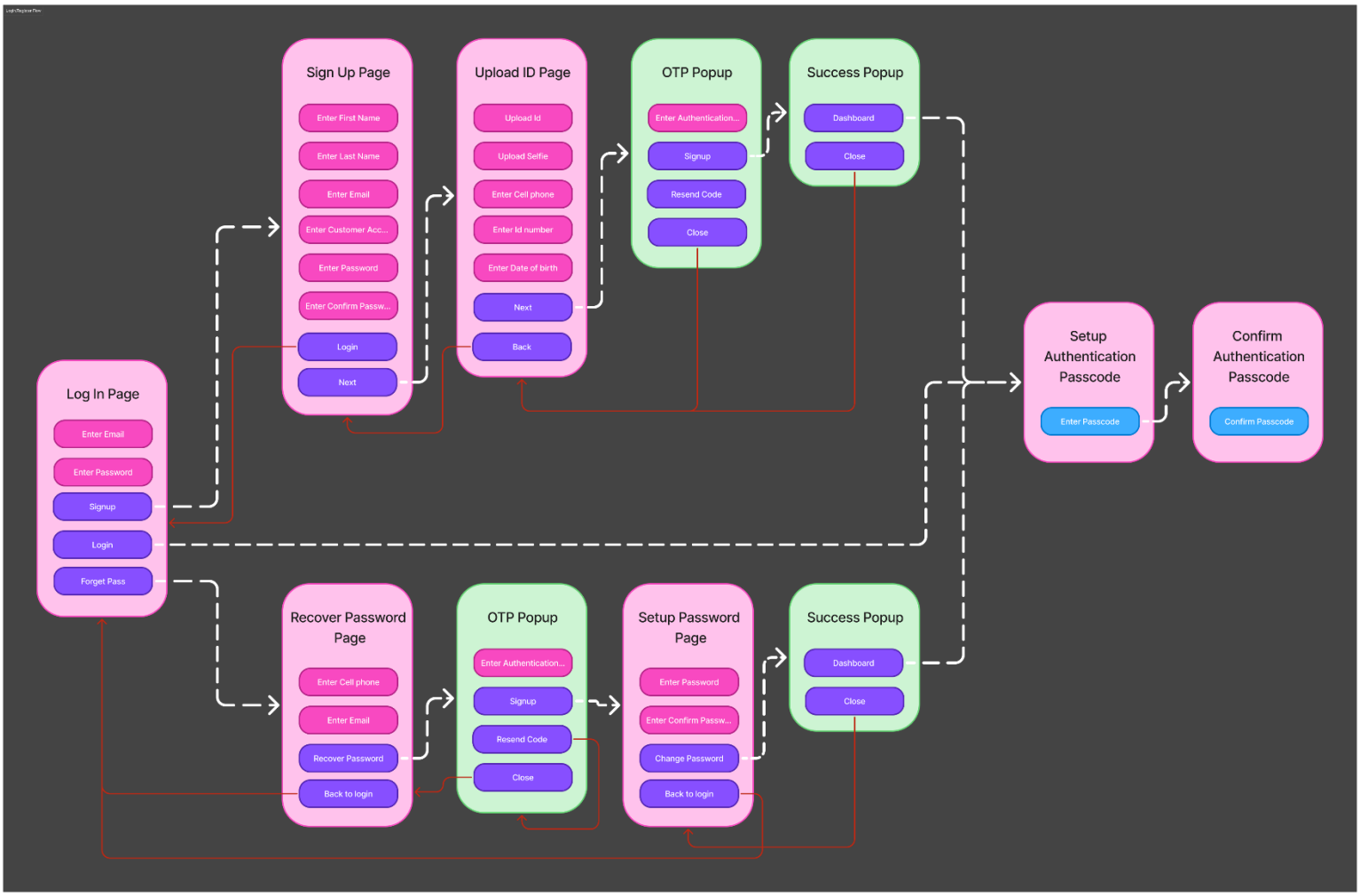
2.2 Implement new look and feel based on modular UI components

- The goal of this task is to redesign the Mifos Mobile 7.0 app's UI using modular UI components, making it more flexible, reusable, and maintainable. This approach will improve code reusability, consistency, scalability, and theming across the app while keeping the UI modern and user-friendly.
- We have this modular UI components in core/UI extending core/designSystem refactoring those Components and enhancing the UI for user experience.
- Adding motion effects to make the UI more dynamic and intuitive. Using Jetpack Compose Animations for smooth screen transitions and button effects.
- **We initially worked with outdated mockups that lacked consistency, modern design principles, and intuitive user flow. Our design team has now created new mockups that follow modern UI/UX standards, offering a more user-friendly, visually appealing, and accessible interface. These updated designs improve navigation, enhance clarity, and align better with the goals of Mifos Mobile 7.0, ensuring a more seamless experience for all users.**

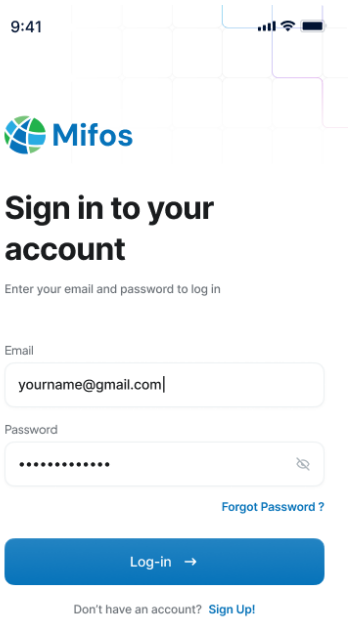
- These are the some of the screens that I am attaching here that are presently using.
- **To enhance the user experience and streamline navigation, we propose simplifying the UI by reducing redundant options. Currently, all four options lead to the same screen, which may cause confusion for users. Instead of multiple redundant entries, a single, clearly labelled option will be provided to ensure intuitive navigation.**
- **Additionally, the home screen and navigation drawer contain duplicate options, leading to unnecessary clutter. To improve usability and maintain a clean interface, these redundant options will be consolidated into a single, well-placed entry. This approach will enhance clarity, minimize cognitive load for users, and create a more efficient and user-friendly experience.**
- **Our designing team is working on creating new UI for the mifos mobile.** https://www.figma.com/design/7HY4EemmGwm0T5MTYfhCb7/Mifos-Mobile?node-id=0-1&t=JyAwbiqDPWG5e4pV-1



- **This is the mock up link for the design. We are going to migrate the whole mifos mobile with these new mock ups that will be finalized and created by UI/UX team.**
- **I am attaching some Screens that are designed by our design team. Like this we are going to migrate full application. You can explore total screens by the above link as there are so many screens I can't attach them here because of space constraint.**
- **These are images of first draft and there is second draft which you can see in the above link.**
- **This is the flow of login page that is designed by our design team and we are going to implement this flow in out mifos mobile 7.0**
- **The above Figma user flow diagram covers the complete user onboarding, authentication, identity verification, password recovery, and Mojaloop passcode setup journey. It's carefully segmented into several critical flows, each interacting cleanly with one another. Let's break each section down with detailed insights.**

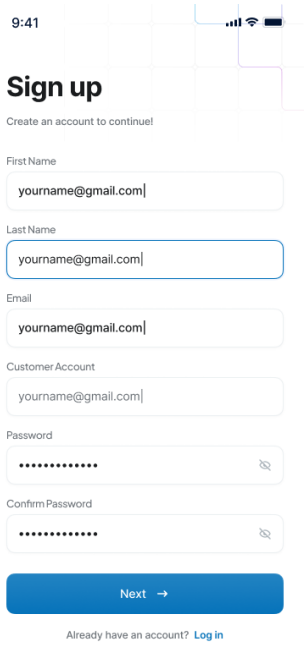


- **Login & Entry Point**



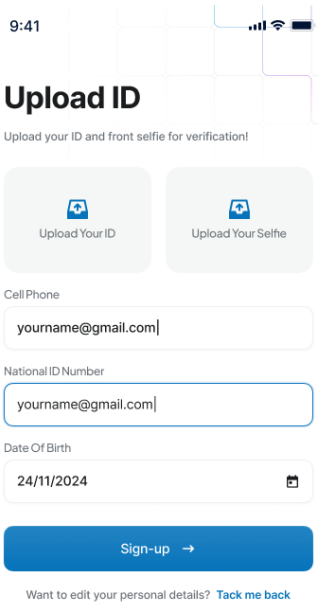
  - **Screen: Log In Page**
  - **Actions:**
    - **Enter Email**
    - **Enter Password**
    - **Navigate to SignUp, Forget Pass, or Login**
  - **Key Logic:**
    - **This is the main entry point for existing users.**
    - **Based on login success, the user is redirected to the dashboard or to passcode setup (if first time).**
    - **Failed attempts can trigger UI feedback (snackbar, retry logic).**

- **User Registration**



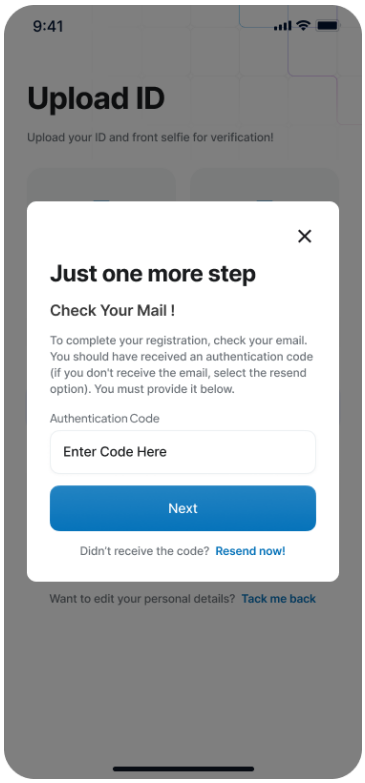
- **Screen: Sign Up Page**
  Inputs:
  - **First Name, Last Name**
  - **Email**
  - **Customer Account Number**
  - **Password & Confirm Password**
- **Actions:**
  - **Next → Upload ID Page**
  - **Login → Back to Login Page**
- **Key Logic:**
  - **Field validation (email format, password match)**
  - **If all fields are valid, continue onboarding with ID verification.**

- **Identity Verification**



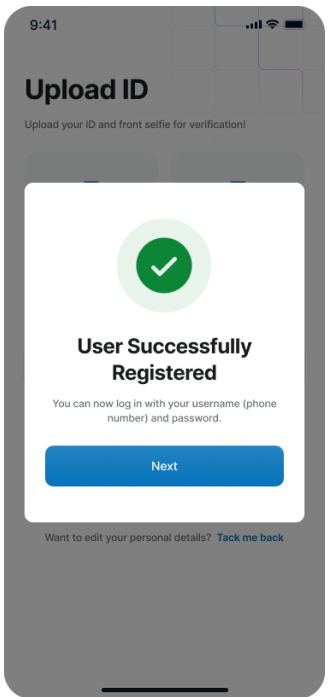
- **Screen: Upload ID Page**
  Inputs:
  - **Upload ID Document**
  - **Upload Selfie**
  - **Cell Phone**
  - **ID Number**
  - **Date of Birth**
- **Actions**
  - **Next → OTP Popup**
  - **Back → SignUp Page**
- **Key Logic:**
  - **Basic KYC step, aligning with Mojaloop's financial identity onboarding.**
  - **File/image validation (format, size).**
  - **Transition to OTP screen for mobile verification.**

- **OTP Verification Popup**



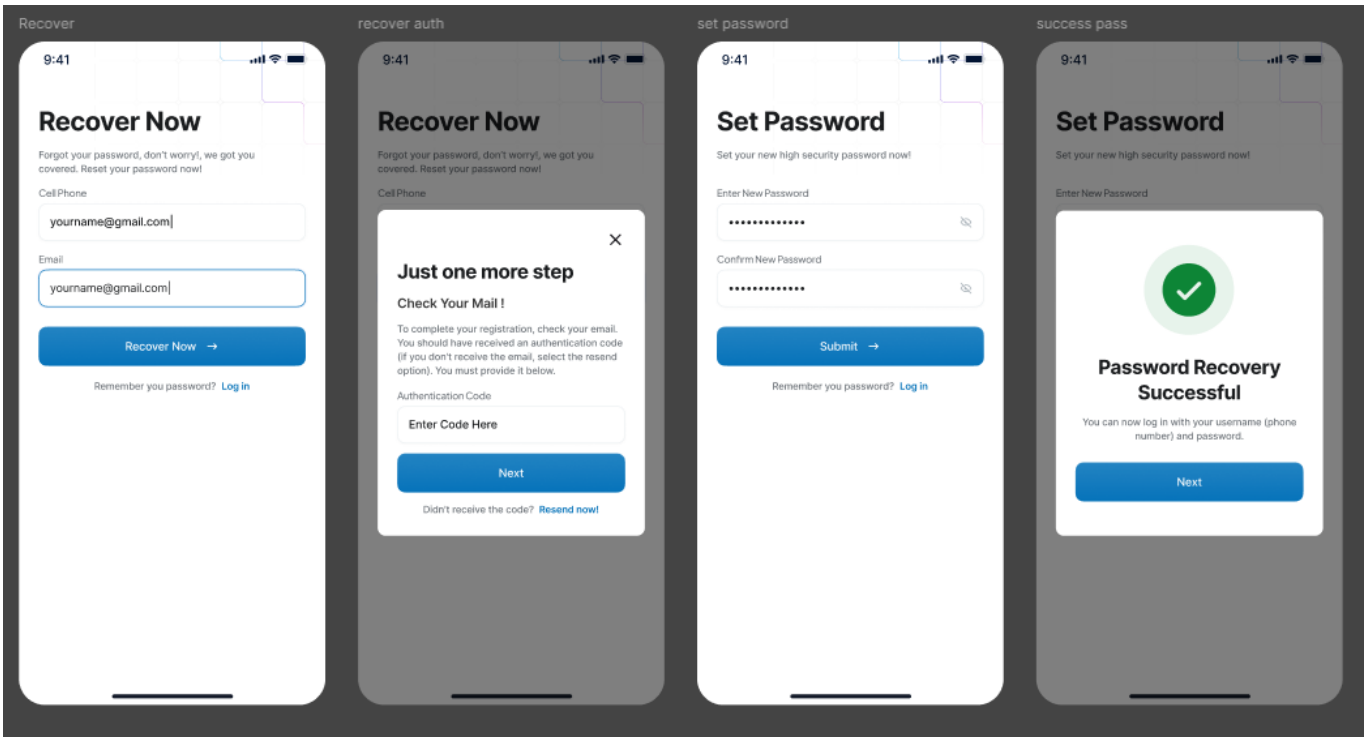
  - **Screen: OTP Popup**
    Inputs:
    - **Authentication Code (OTP)**
    - **Actions:**
      - **SignUp → Success Popup**
      - **Resend Code**
      - **Close → Back to previous screen**
  - **Key Logic:**
    - **Ensures the mobile number provided is active.**
    - **On successful OTP, completes registration and shows success.**

- **Registration Completion**



- **Screen: Success Popup**
  Actions:
  - **Dashboard → Proceed to home**
  - **Close → Ends popup**
- **Key Logic:**
  - **Confirmation message**
  - **Used after sign-up or password reset to finalize flow**

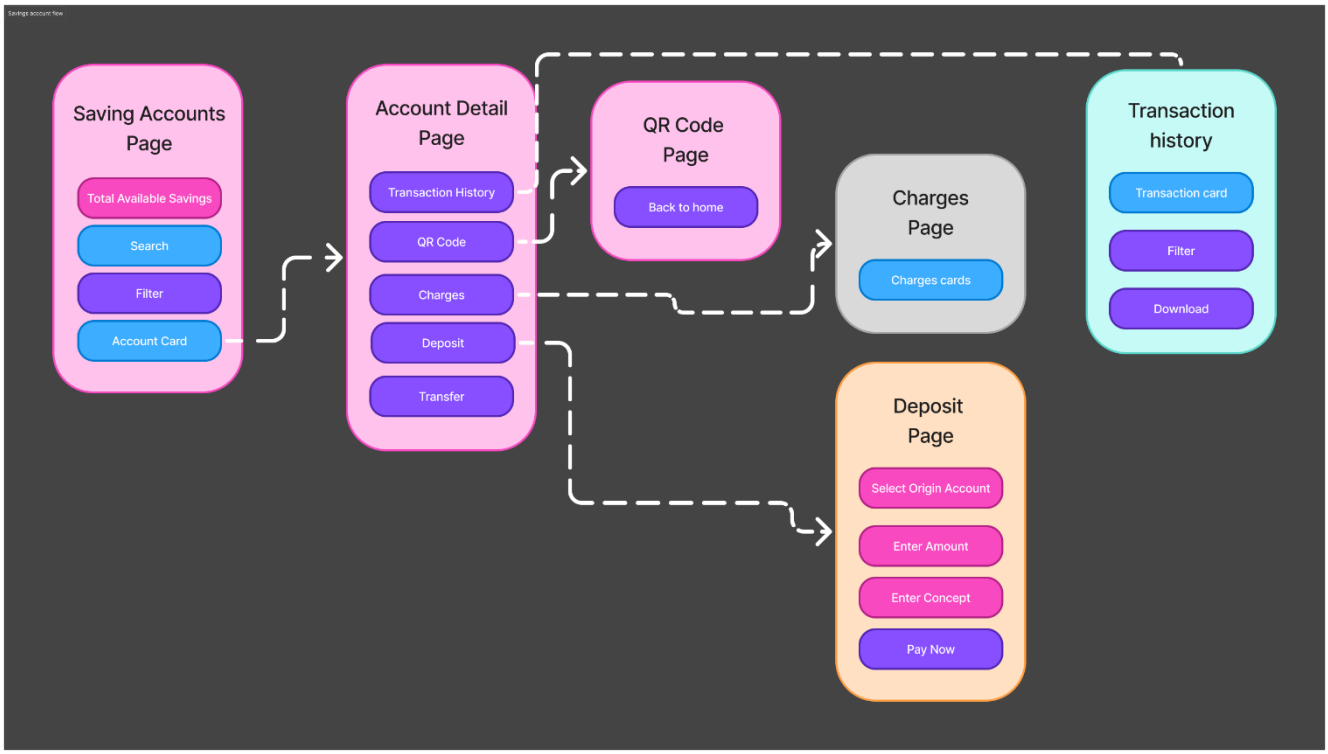
- **Forgotten Password Flow**



- **Screen: Recover Password Page**
  Inputs:
  - **Cell Phone**
  - **Email**
- **Actions:**
  - **Recover Password → OTP Popup**
  - **Back to Login**
- **Key Logic:**
  - **Collects user identification for password reset**
  - **OTP is triggered for verification before allowing password change**

- **Password Reset Flow**
  - **Screen 1: OTP Popup**
    Screen 2: Setup Password Page
    Inputs:
    - **New Password & Confirm Password**
  - **Actions:**
    - **Change Password → Success Popup**

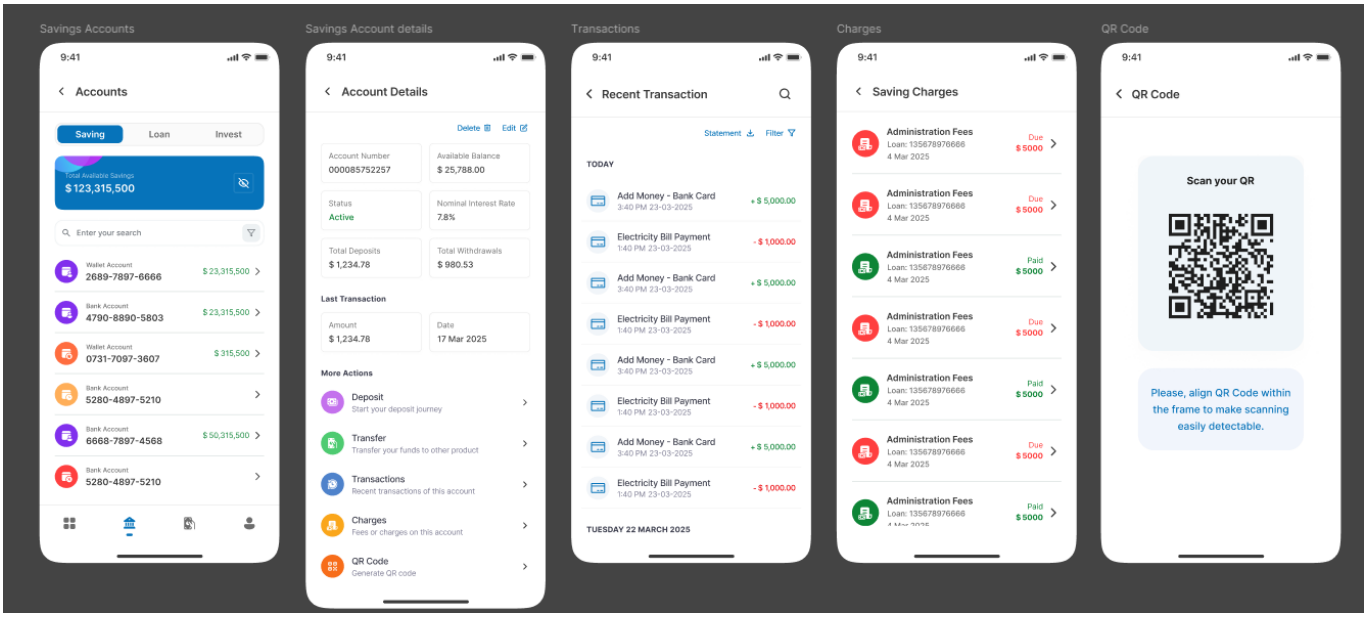
        - **Back to Login**
    - **Key Logic:**
        - **Secures the password change with OTP authentication**
        - **Successful reset transitions to Success Popup again**



- **Authentication Passcode Setup**
    - **Screen: Setup Authentication Passcode**
      Input:
        - **Enter 4-6 digit passcode (like a pin code)**
    - **Screen: Confirm Authentication Passcode**
      Input:
        - **Re-enter passcode for confirmation**
    - **Key Logic:**
        - **This is a Mojaloop-related security layer**
        - **After login/signup, the user must create a passcode (like UPI PIN) to use for transactions**
        - **After confirmation, users are directed to the dashboard**

- **Navigation and Flow Logic**
    - **Dashed Arrows: Represent navigation between screens**
    - **Solid Arrows: Forward and backward actions within a module**
    - **Red Arrows: Special cases like going back to login, or closing modals**

- **Savings Account Flow**



- **This module includes:**
  - **Savings account overview**
  - **QR/charge utilities**
  - **Deposit functionality**
  - **Transaction and charges history**
- **Saving Accounts Page**
  - **Main Entry Point for Savings Module**
  - **Actions:**
    - **Total Available Savings: Displays total savings balance (static or animated counter)**
    - **Search: Opens input to find accounts**
    - **Filter: Helps narrow down accounts by criteria (status, balance, etc.)**
    - **Account Card: Tapping a card navigates to the Account Detail Page**
  - **Logic Highlights:**
    - **Paginated or lazy column for account list**
    - **Trigger to load data using ViewModel.onEvent(LoadAccounts)**
    - **Navigation to details screen with selected account ID or encoded AccountArgs**

- **Account Detail Page**
  - **Central Hub for Account-Specific Actions**
  - **Actions:**
    - **Transaction History: View detailed transactions → leads to Transaction History Page**
    - **QR Code: Generates QR linked to this account → leads to QR Code Page**
    - **Charges: Review charges associated with account → leads to Charges Page**
    - **Deposit: Opens deposit screen → leads to Deposit Page**
    - **Transfer: (Not covered in UI here but space is designed for future enhancement)**
  - **Logic Highlights:**
    - **Argument passed is accountId**
    - **Allows deep linking into savings features for modular navigation**

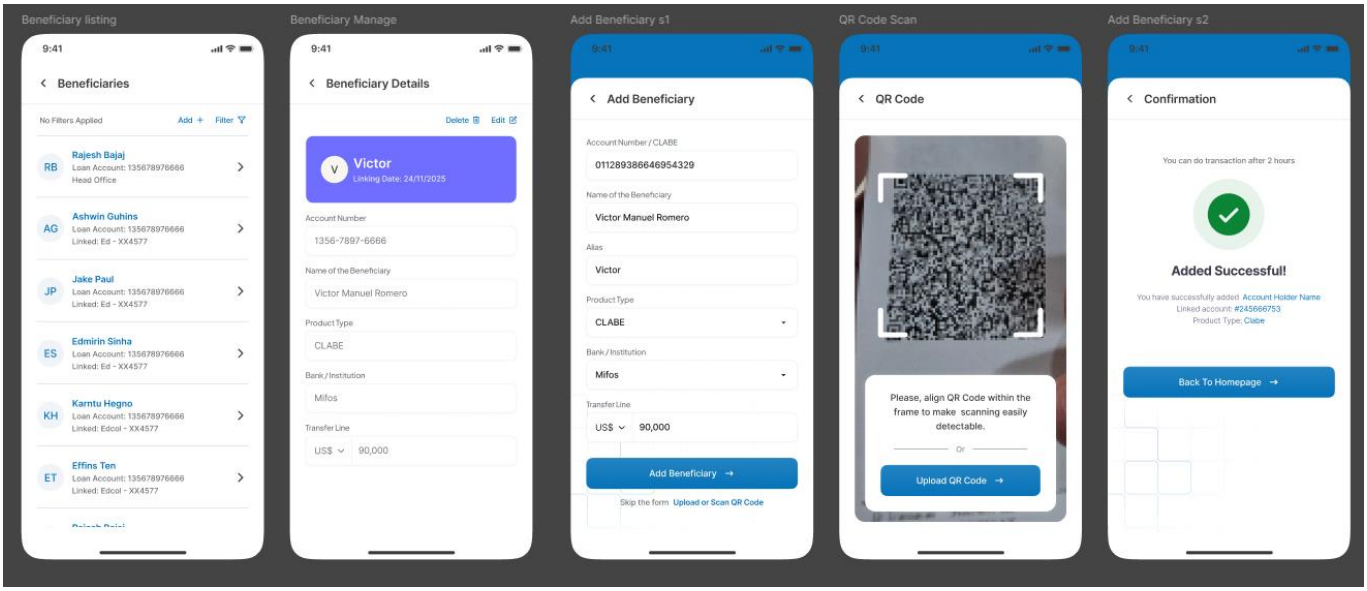
- **QR Code Page**
  - **Purpose: To generate or view the QR associated with this account.**
  - **Action:**
    - **Back to Home: Takes user back to Account Detail Page**
  - **Logic:**
  - **Dynamic QR generation (fetched via API or generated locally)**
  - **Visual elements like share or copy button may be added later**

- **Charges Page**
  - **Purpose: To review all charges related to the selected savings account.**
  - **Component:**
    - **Charges cards: Each card displays a charge (type, amount, date)**
  - **Logic:**
    - **Loaded from server or local DB using accountId**
    - **UI interaction includes viewing charge breakdown**

- **Deposit Page**
  - **Core transactional screen to deposit into the savings account**
  - **Inputs:**
    - **Select Origin Account: Dropdown or selection from user's accounts**
    - **Enter Amount: Input field with validation**
    - **Enter Concept: Reason or purpose of deposit**
    - **Pay Now: Trigger action to submit deposit**
  - **Logic:**
    - **Form validation (amount > 0, origin account != target account)**
    - **On success, show confirmation/snackbar and navigate back**

- **Transaction History Page**
  - **Purpose: To review transaction logs for the selected savings account**
  - **Components:**
    - **Transaction card: Lists individual transactions**
    - **Filter: Filter by date range, type (credit/debit), etc.**
    - **Download: Download statement as PDF or CSV**
  - **Logic:**
    - **Supports pull-to-refresh and lazy loading**
    - **Long press or swipe on transaction card can allow extra actions**

- **Beneficiary Flow**



- **This module includes:**
  - **Viewing existing beneficiaries**
  - **Editing or adding new ones**
  - **Transferring funds using beneficiary information**
  - **Uploading QR for payment**

- **Beneficiary Card Page**
  - **Displays a list of existing beneficiaries**
  - **Actions:**
    - **New Beneficiary: Navigate to add a new beneficiary → leads to Edit/Add Beneficiary Page**
    - **QR: Upload QR code for transfer → leads to Upload QR Page**
    - **List: Opens detailed info of a specific beneficiary → leads to Beneficiary Details Page**
  - **Logic:**
    - **Uses a LazyColumn to display BeneficiaryCard**
    - **Button click navigates using NavController and encoded arguments if needed**

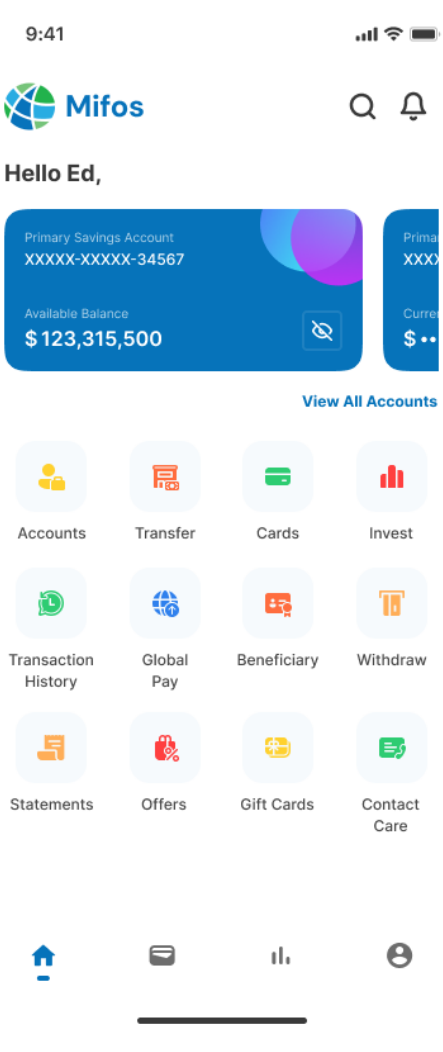
- **Beneficiary Details Page**
  - **Shows full details of a selected beneficiary**
  - **Actions:**
    - **Edit: Opens form to update beneficiary → leads to Edit/Add Beneficiary Page**
  - **Logic:**

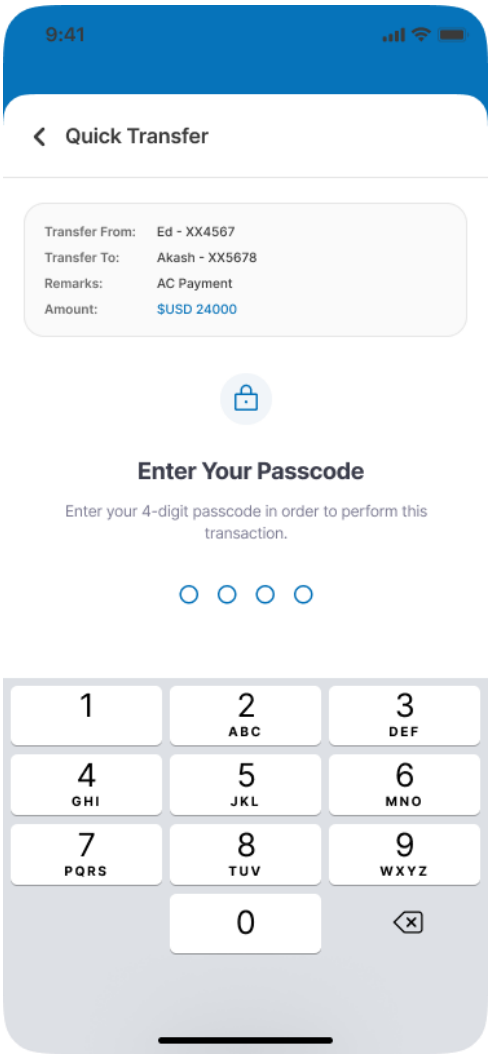
        - **Data fetched via ViewModel.loadBeneficiaryDetails(id)**
        - **Can include delete confirmation modal in future**

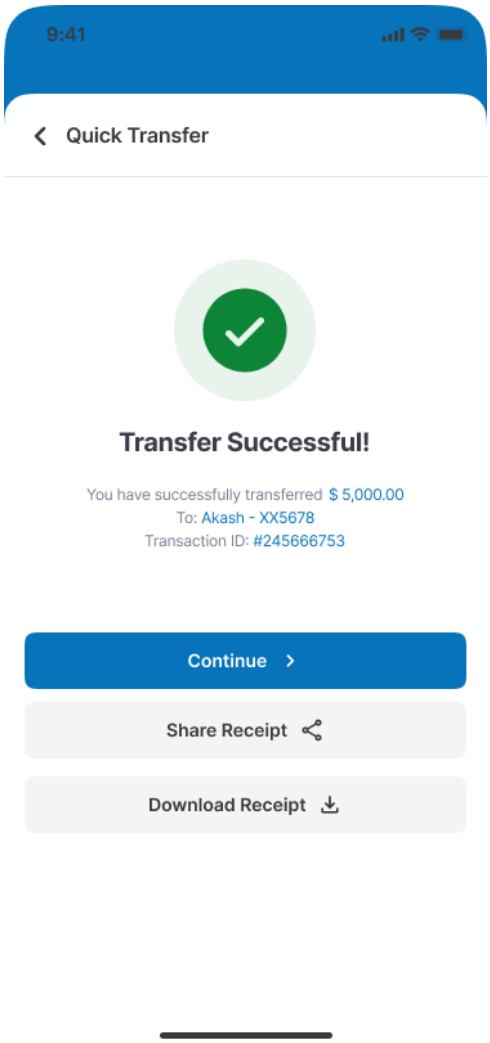
- **Edit/Add Beneficiary Page**
    - **Core form screen for adding or editing a beneficiary**
    - **Inputs/Sections:**
        - **Account Number**
        - **IFSC**
        - **Branch Name**
        - **Email/Phone**
        - **Save button to submit data**
    - **Logic:**
        - **Handles both Add and Edit modes (based on passed beneficiary ID or null)**
        - **Includes field validations (account number format, IFSC regex)**
        - **Displays error states or success snackbar after action**

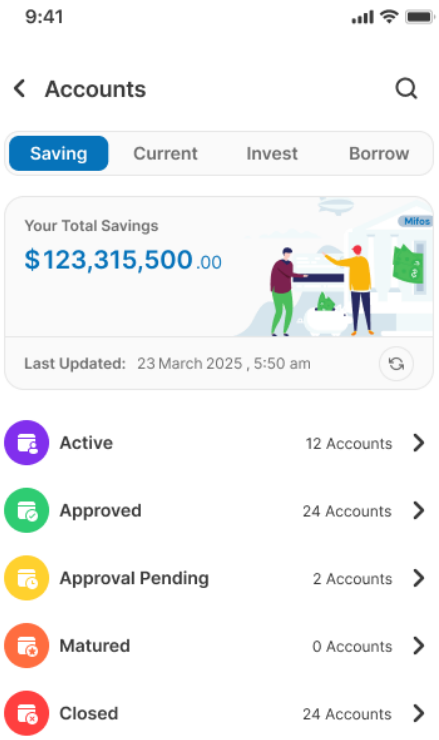
- **Upload QR Page**
    - **Purpose: To upload a QR code for scanning beneficiary payment details**
    - **Action:**
        - **Upload: Triggers file/image picker (camera/gallery) and reads QR**
    - **Logic:**
        - **Integrates with image picker and barcode scanner (ML Kit or BarcodeScanning from kmp-vision)**
        - **On success, redirects back to Edit/Add form with parsed data pre-filled**

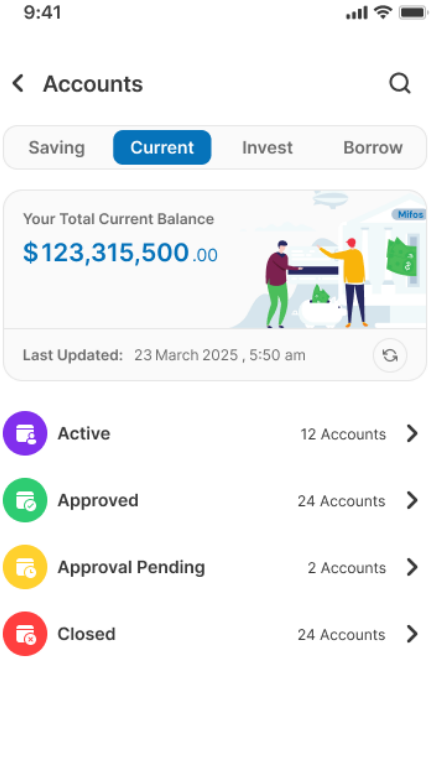
- **As part of the Mifos Mobile 7.0 project, our design team has prepared complete screen flows for all major modules — including Payment, Login, Registration, Savings accounts, Loan Accounts, Transactions, Settings, Profile, Transfer, Charges, Applying Loan, Updating Loan, Beneficiary, and more.**
- **These designs are created to ensure a smooth and intuitive user experience across all platforms (Android, iOS, Web, Desktop). My role involves converting these design flows into functional UI using Jetpack Compose and Kotlin Multiplatform, while maintaining consistency and performance.**
- **By following a architecture, I'm ensuring each screen has a well-structured ViewModel with clear state, actions, and event handling. This not only improves maintainability but also helps us deliver a more stable and responsive app.**
- **With this approach, we're building a user-friendly, scalable, and consistent mobile banking experience — perfectly aligned with Mifos' mission of financial inclusion.**

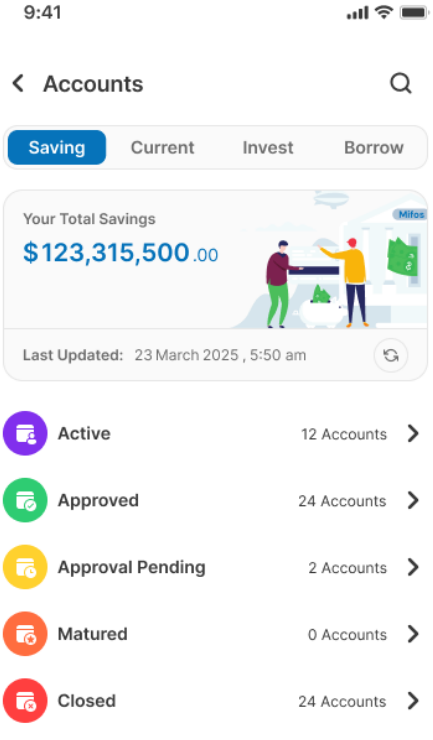

9:41
Mifos
Hello Ed,
Primary Savings Account
XXXXX-XXXXX-34567
Available Balance
$123,315,500
View All Accounts
Accounts
Transfer
Cards
Invest
Transaction History
Global Pay
Beneficiary
Withdraw
Statements
Offers
Gift Cards
Contact Care


9:41
Quick Transfer
Transfer From: Ed - XX4567
Transfer To: Akash - XX5678
Remarks: AC Payment
Amount: $USD 24000
Enter Your Passcode
Enter your 4-digit passcode in order to perform this transaction.
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
0


9:41
Quick Transfer
Transfer Successful!
You have successfully transferred $ 5,000.00
To: Akash - XX5678
Transaction ID: #245666753
Continue
Share Receipt
Download Receipt


9:41
Accounts
Saving
Current
Invest
Borrow
Your Total Savings
$123,315,500.00
Last Updated: 23 March 2025 , 5:50 am
Active 12 Accounts
Approved 24 Accounts
Approval Pending 2 Accounts
Matured 0 Accounts
Closed 24 Accounts

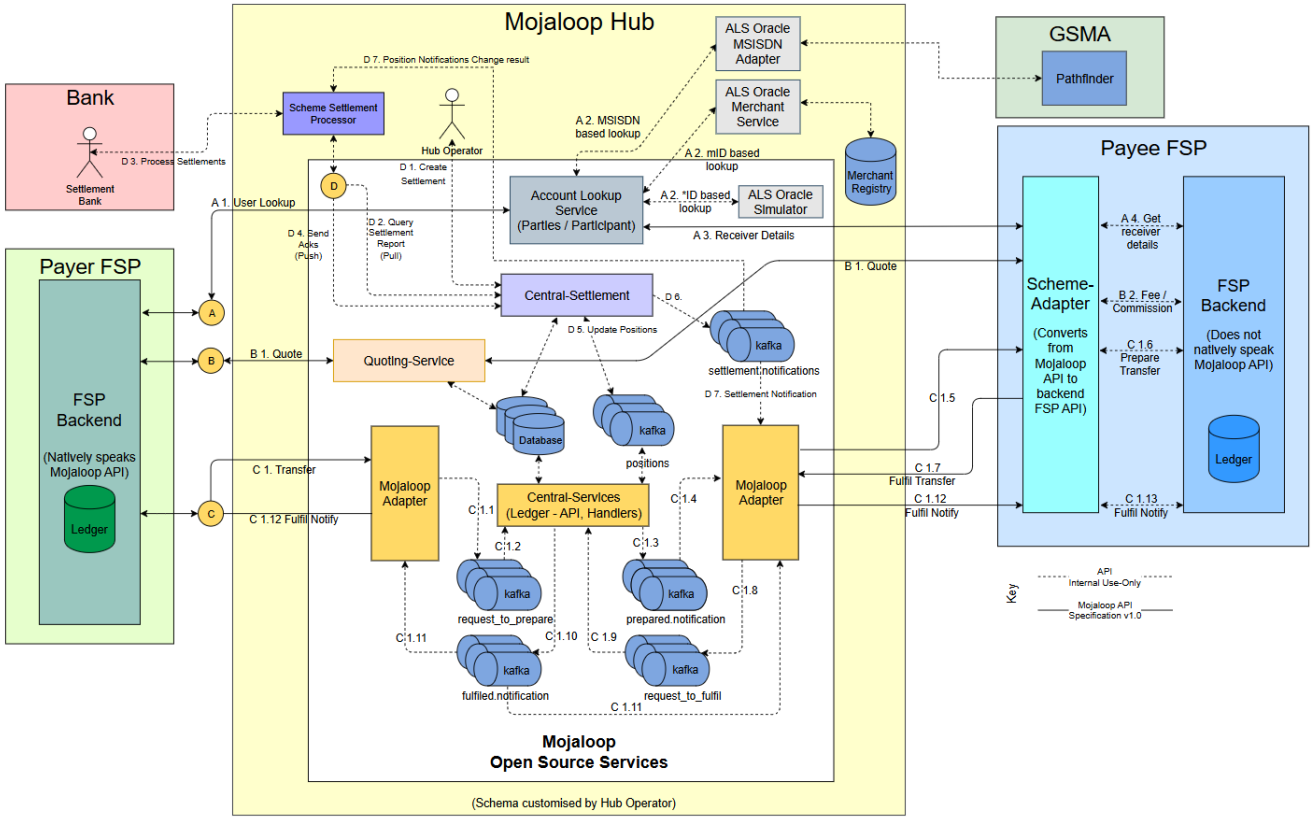

9:41
Accounts
Saving
Current
Invest
Borrow
Your Total Current Balance
$123,315,500.00
Last Updated: 23 March 2025 , 5:50 am
Active 12 Accounts
Approved 24 Accounts
Approval Pending 2 Accounts
Closed 24 Accounts


9:41
Accounts
Saving
Current
Invest
Borrow
Your Total Savings
$123,315,500.00
Last Updated: 23 March 2025 , 5:50 am
Active 12 Accounts
Approved 24 Accounts
Approval Pending 2 Accounts
Matured 0 Accounts
Closed 24 Accounts

2.3 Integration with an external payment system (Mojaloop, mPesa) via our PH-EE

- **Mojaloop**
    - Mojaloop is an open-source payment platform designed to enable interoperable digital financial services for unbanked and underbanked populations. It allows financial institutions, mobile money operators, and service providers to connect and facilitate real-time, low-cost, and secure transactions. Mojaloop was developed by the **Mojaloop Foundation**, with support from the Bill & Melinda Gates Foundation, to promote financial inclusion globally.
    - The basic idea behind Mojaloop is that we need to connect multiple Digital Financial Services Providers (DFSPs) together into a competitive and interoperable network in order to maximize opportunities for poor people to get access to financial services with low or no fees. We don't want a single monopoly power in control of all payments in a country, or a system that shuts out new players. It also doesn't help if there are too many isolated subnetworks. The following diagrams shows the Mojaloop interconnects between DFSPs and the Mojaloop Hub (schema implementation example) for a Peer-to-Peer (P2P) Transfer:
- Mojaloop addresses these issues in several key ways:

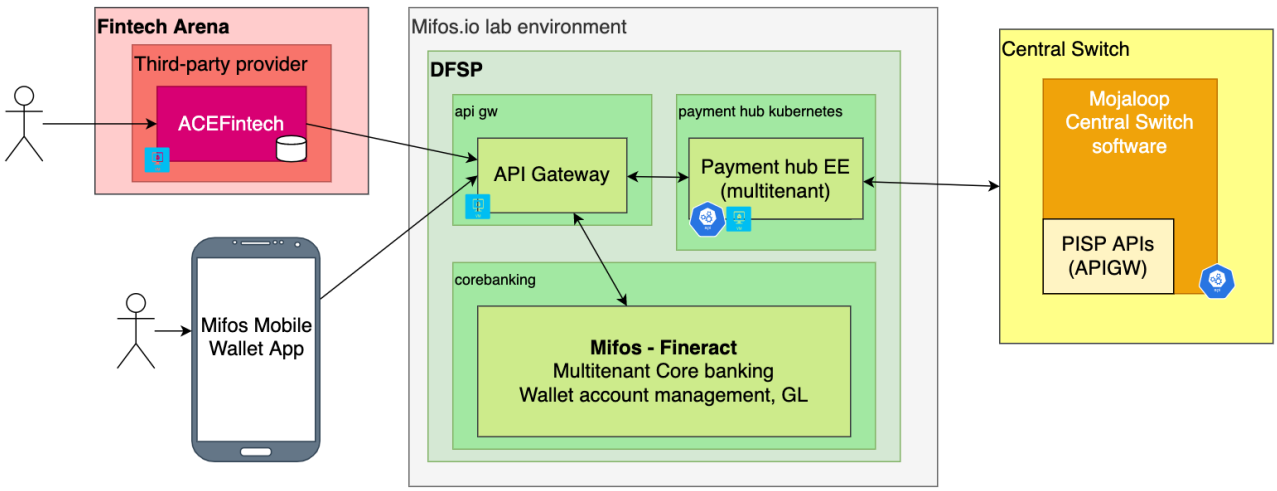
    - A set of central services provides a hub through which money can flow from one DFSP to another. This is similar to how money moves through a central bank or clearing house in developed countries. Besides a central ledger, central services can provide identity lookup, fraud management, and enforce scheme rules.
    - A standard set of interfaces a DFSP can implement to connect to the system, and example code that shows how to use the system. A DFSP that wants to connect up can adapt our example code or implement the standard interfaces into their own software. The goal is for it to be as straightforward as possible for a DFSP to connect to the interoperable network.
    - Complete working open-source implementations of both sides of the interfaces - an example DFSP that can send and receive payments and the client that an existing DFSP could host to connect to the network.

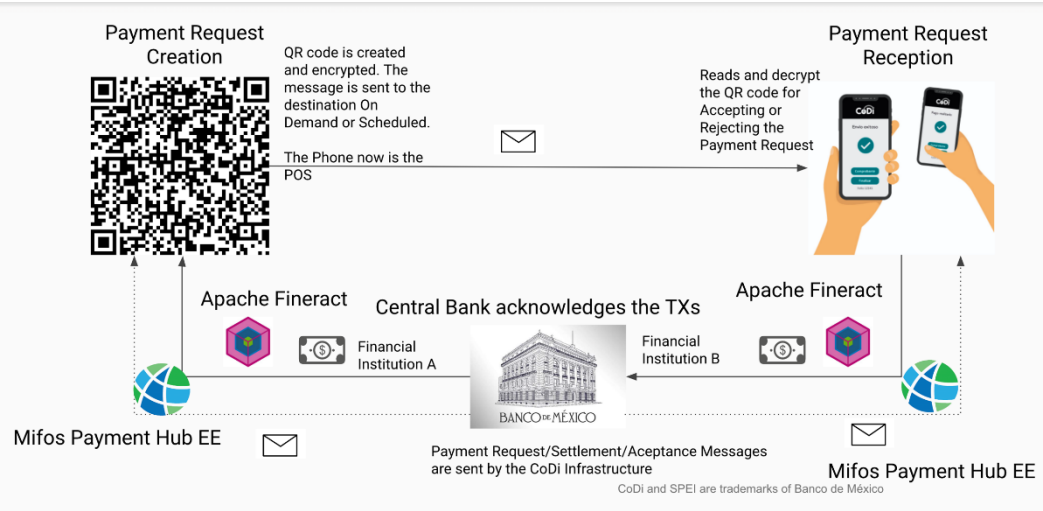


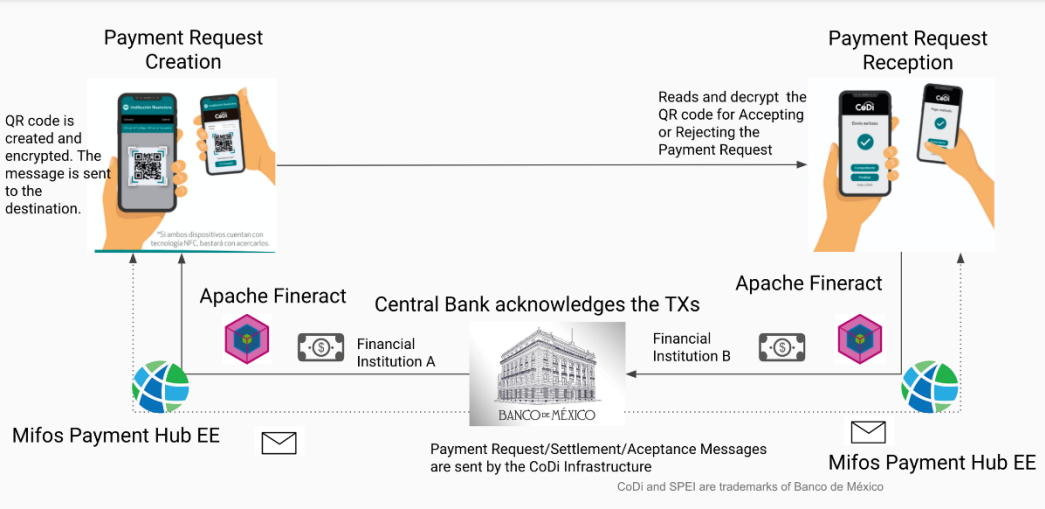
- This diagram represents the Mojaloop Hub architecture and the flow of transactions between financial service providers (FSPs) using the Mojaloop platform. It details how payer and payee FSPs interact through Mojaloop's core components, including the Account Lookup Service (ALS), Quoting-Service, Mojaloop Adapter, Central Settlement, and other Open Source Services. Below is a detailed breakdown of this architecture:
- Key Participants in the Architecture
    - **Payer FSP (Green Box)**: The financial institution or digital wallet provider initiating the transaction.
    - **Payee FSP (Blue Box)**: The receiving financial institution or mobile money provider.
    - **Mojaloop Hub (Yellow Box)**: The core of Mojaloop that facilitates interoperability, transaction processing, and settlement.
    - **Bank (Pink Box)**: Handles final settlement processing.
    - **GSMA Pathfinder (Light Green Box)**: Assists in phone number lookup and routing.
- Core Components and Data Flow in Mojaloop Hub
    - **Account Lookup Service (ALS):**
        - **Function**: Determines which financial institution (FSP) holds the recipient's account.
        - **Process:**
            1. **A.1 User Lookup**: The Payer FSP queries Mojaloop to find the recipient's bank/mobile money provider.
            2. **A.2, A.3, A.4**: The ALS Oracle checks phone number-based registries to identify the Payee FSP.
    - **Quoting Service:**
        - **Function:** Provides transaction cost estimates before the transfer.
        - **Process:**
            1. **B.1 Quote:** The Payer FSP requests a quote from Mojaloop.
            2. **B.2 Fee / Commission:** The Payee FSP responds with transfer fees
    - **Transaction Processing (Transfers):**

        - **Function:** Handles the actual money transfer between FSPs.
        - **Process:**
            1. **C.1 Transfer:** The Payer FSP initiates a fund transfer request.
            2. **C.1.1 – C.1.4:** The Mojaloop Hub processes the request using the Mojaloop Adapter and Central Services.
            3. **C.1.5 – C.1.8:** The funds are moved to the Payee FSP, with notifications sent throughout the process.
            4. **C.1.12 – C.1.13 Fulfill Notify:** Both FSPs receive transaction completion confirmations.
    - **Settlement Process:**
        - **Function:** Ensures funds are reconciled between FSPs and the banking system.
        - **Process:**
            1. **D.1 Create Settlement:** The Scheme Settlement Processor generates a settlement request.
            2. **D.2 – D.4:** Queries and updates are performed to finalize the transaction.
            3. **D.5 Update Positions:** The Central Settlement component updates balances.
            4. **D.6 – D.7 Settlement Notifications:** Notifications are sent to involved parties.
- **Open-Source Services in Mojaloop**
    - **Mojaloop Adapter**
        - Converts Mojaloop API requests into a format compatible with the FSP's backend systems.
    - **Central Services**
        - Manages the ledger, transaction routing, and messaging between components.
    - **Kafka Message Queue**
        - Handles event-driven notifications for requests, approvals, and transaction settlements.
- **Proposed Architecture Overview:**

```
[Mobile App (KMP)]
      |
      | ---> [Middleware Layer - PH-EE API]
                  |
                  | ---> [Mojaloop Adapter (Kafka-based)]
                              |
                              | ---> [Mojaloop Core Services (ALS,
Quoting, Transfer)]
```







- **Implementation in Mifos Mobile 7.0**
  - Add required dependencies in gralde like **ktorFit** for network calls and **Kotlin serialization** and other modules.

```
kotlin {
    sourceSets {
        val commonMain by getting {
            dependencies {
                // HTTP Client
                implementation("io.ktor:ktor-client-core:$ktorVersion")
                implementation("io.ktor:ktor-client-content-negotiation:$ktorVersion")
                implementation("io.ktor:ktor-serialization-kotlinx-json:$ktorVersion")

                // Serialization
                implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.6.0")

                // Retrofit-like client for KMP
                implementation("de.jensklingenberg.ktorfit:ktorfit-lib:$ktorfitVersion")
            }
        }
    }
}
```

- Create required models for sending payload and receiving response from network call

```kotlin
@Serializable
data class PartyIdInfo(
    val partyIdType: String, // e.g., "MSISDN"
    val partyIdentifier: String // e.g., "27710101999"
)

@Serializable
data class Payer(
    val partyIdInfo: PartyIdInfo
)

@Serializable
data class Payee(
    val partyIdInfo: PartyIdInfo
)

@Serializable
data class Amount(
    val amount: Int,
    val currency: String
)

@Serializable
data class MojaloopTransferRequest(
    val payer: Payer,
    val payee: Payee,
    val amount: Amount
)

@Serializable
data class MojaloopTransferResponse(
    val transactionId: String,
    val status: String,
    val timestamp: String
)
```

- Creating network interface using ktorfit annotations

```
interface MojaloopApi {
    @POST("payments/mojaloop/transfer")
    suspend fun initiateTransfer(
        @Body request: MojaloopTransferRequest
    ): MojaloopTransferResponse
}
```

- Creating repository interface with methods

```
interface MojaloopRepository {
    suspend fun initiateTransfer(
        request: MojaloopTransferRequest
    ): Result<MojaloopTransferResponse>
}
```

- Creating repository implementation extending the above repository interface

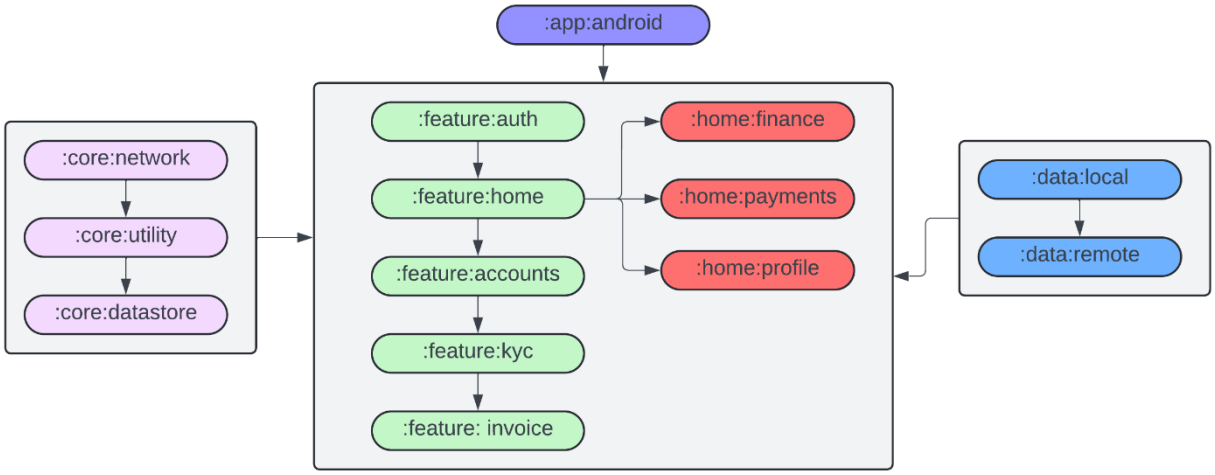
```
class MojaloopRepositoryImpl(
    private val api: MojaloopApi
) : MojaloopRepository {
    override suspend fun initiateTransfer(
        request: MojaloopTransferRequest
    ): Result<MojaloopTransferResponse> {
        return try {
            val response = api.initiateTransfer(request)
            Result.success(response)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }
}
```

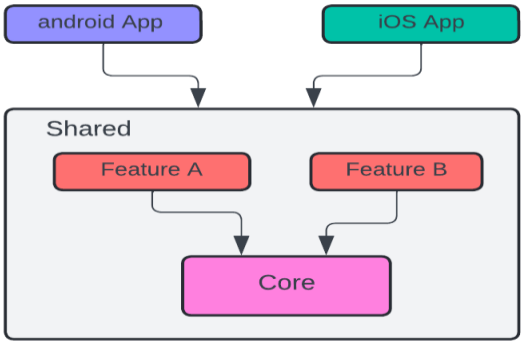
- Calling this method from viewModel by injecting **MojaloopRepository interface using koin dependency injection**

```
val request = MojaloopTransferRequest(
    payer = Payer(
        partyIdInfo = PartyIdInfo(
            partyIdType = "MSISDN",
            partyIdentifier = "27710101999"
        )
    ),
    payee = Payee(
        partyIdInfo = PartyIdInfo(
            partyIdType = "MSISDN",
            partyIdentifier = "27710102999"
        )
    ),
    amount = Amount(
        amount = 230,
        currency = "TZS"
    )
)

// In ViewModel call:
viewModel.sendMoney(request)
```

2.4 Complete migrate Kotlin multi-module to Kotlin multiplatform

1. The migration of Mifos Mobile 7.0 from a Kotlin multi-module structure to Kotlin Multiplatform (KMP) is a significant architectural upgrade that enhances code sharing, maintainability, and platform consistency. The transition allows business logic, networking, and data layer to be shared across Android, iOS, and other platforms while keeping platform-specific UI implementations.


:app:android
:core:network
:core:utility
:core:datastore
:feature:auth
:feature:home
:feature:accounts
:feature:kyc
:feature: invoice
:home:finance
:home:payments
:home:profile
:data:local
:data:remote


android App
iOS App
Shared
Feature A
Feature B
Core

```
include(":androidApp")

// Core Modules
include(":core:ui")
include(":core:designsystem")
include(":core:logs")
include(":core:model")
include(":core:common")
include(":core:data")
include(":core:network")
include(":core:database")
include(":core:datastore")
include(":core:qrcode")
include(":core:testing")

// Feature Modules
include(":feature:loan")
include(":feature:beneficiary")
include(":feature:savings")

// Other modules

// Lint Modules
//include(":lint")

// Library Modules
include(":libs:country-code-picker")
include(":libs:pullrefresh")
include(":libs:material3-navigation")
include(":libs:mifos-passcode")

// Kotlin Multiplatform Modules
include(":shared")
```

- This is the **settings.gradle.kt before migrating to KMP**. That is purely in android and it will only support and runs in android devices. After migrating we are expecting the structure will looks like below one.

- This is the **settings.gradle.kt after migrating to Kotlin Multiplatform**. In the below code you can see **shared, android, desktop, web** modules which are responsible for running in their respective platforms. **cmp-android** targets **android**, **cmp-desktop** targets **desktop**, **cmp-web** targets **browser**, **cmp-ios** targets **IOS.** After this I will explain how we have migrated and what steps we have taken before migrating to KMP and CMP.

```
include(":cmp-shared")
include(":cmp-android")
include(":cmp-desktop")
include(":cmp-web")
include(":cmp-navigation")

// Core Modules
include(":core:ui")
include(":core:designsystem")
include(":core:logs")
include(":core:model")
include(":core:common")
include(":core:data")
include(":core:network")
include(":core:database")
include(":core:datastore")
include(":core:qrcode")
//include(":core:testing")

// Feature Modules
include(":feature:loan")
include(":feature:savings")
include(":feature:beneficiary")
include(":feature:savings")
// Other modules

// Lint Modules
//include(":lint")

// Library Modules
include(":libs:country-code-picker")
include(":libs:pullrefresh")
include(":libs:material3-navigation")
include(":libs:mifos-passcode")
```

- **Steps Taken for Migration**
  - **Identified and migrated the libraries that are not compatible with KMP**
    - In android we are using DBFlow database, Hilt which is not compatible with KMP. So firstly, we have migrated those from DBFlow to room database and we make sure that all libraries are compatible with KMP. This makes the process easy while migrating to project to KMP.
  - **Setting up plugins**
    - To use the koin and some other libraries we have setup plugins. So that it can be used any where in the project without adding in dependencies block every time in the. Just we can add one plugin in the Gradle file.
  - **Converting Modules to KMP**
    - Identified core modules such as core, network, and database that could be shared.
    - Refactored these modules to expect/actual declarations to support multiple platforms
    - Introduced Kotlin Multiplatform Gradle Plugin and updated Gradle dependencies.
  - **Refactoring the API Layer**
    - The network layer was updated to use Ktor Client instead of platform-specific Retrofit.
    - Serialization was migrated to Kotlinx.serialization for better cross-platform support.
  - **Handling Dependency Injection**
    - Migrated to Koin DI to work within KMP-compatible architecture from hilt.
    - Ensured dependency resolution works correctly for Android and iOS.
  - **Updating UI & Platform-Specific Code**
    - Separated Jetpack Compose UI (Android-specific) from shared logic.
    - Ensured platform-specific implementations remained modular

- **. Pull Requests (PRs) Contributed**

1. PR #2728: Migrated DB Flow to Room
2. PR #2735: Migrated core/logs module to KMP
3. PR #2739: Migrated core/network module to KMP
4. PR #2747: Migrated core/data module to KMP
5. PR #2751: Migrated core/QrCode module to KMP
6. PR #2760: Setting up modules for different platforms (Android, Desktop, Web)
7. PR #2772: Migrated feature/auth to CMP
8. PR #2785: Migrated feature/home to CMP
9. PR #2499: Update readme with current version of the project

You can find my PRs in the Mifos GitHub repository and list them here

- **Current Status & Completion Timeline**
  - Most of the migration is completed.
  - The remaining tasks include final testing and integration.
  - The migration is expected to be fully completed by the end of this month.
  - This will ensure Mifos Mobile 7.0 is future-proof, allowing cross-platform support and easy expansion to Android, Desktop, Web, IOS.

2.5 Make sure fineract-client-sdk implemented properly and have no bugs

- **Verify API Functionality**: Test and ensure that all API calls (authentication, account details, transactions, loan applications, etc.) work correctly.
- **Fix Existing Bugs**: Identify and resolve any inconsistencies or failures in API request handling.
- **Optimize Network Calls**: Improve request handling to minimize redundant network calls and enhance efficiency.
- **Improve Error Handling**: Implement proper retry logic for unstable network conditions, enhance exception handling, and improve logging for debugging.
- **Ensure Middleware Compatibility**: Validate that the SDK correctly interacts with the new self-service middleware layer.
- **Write Unit Tests**: Cover all SDK functions with unit tests to handle edge cases like authentication failures, empty responses, and slow network conditions.
- **Ensure Stability & Performance**: Conduct thorough testing to confirm that the SDK is stable, secure, and performs efficiently across platforms.

- Steps to integrating fineract-client-sdk
    - Add dependency in **build.gradle**

```
dependencies {
    implementation("com.github.openMF:fineract-client-kmp-sdk:$sdk_Version")
}
```

    - Add the below code snippet in your root settings.gradle.

```
dependencyResolutionManagement {
    repositories {
        maven {
            setUrl("https://jitpack.io")
        }
    }
}
```

- Initialize the BaseApiManager and create a service to initialize the KtorFit client.

```
val baseUrl = PROTOCOL_HTTPS + API_ENDPOINT + API_PATH
val tenant = "default"
val username = "mifos"
val password = "password"
val baseApiManager = BaseApiManager.getInstance()
baseApiManager.createService(username, password, baseUrl, tenant, false)
```

- **Authentication**
    - Example code to use the Authentication API:

```
val req = PostAuthenticationRequest(username = username, password = password)

lifecycleScope.launch(Dispatchers.IO) {
    val response = baseApiManager.getAuthApi().authenticate(req, true)
    Log.d("Auth Response",response.toString())
}
```

- To get Client Details with clientId 1:

```
private fun getClient() {
    lifecycleScope.launch(Dispatchers.IO) {
        val response = baseApiManager.getClientsApi().retrieveOne11(1, false)
        Log.d("Client Details Response",response.toString())
    }
}
```

- **Improve Error Handling:**
  - I will implement standardized error-handling patterns that support retry logic for unstable networks, distinguish between HTTP and IO exceptions, and improve error messages for both developers and users. These improvements can be done either in the **SDK layer** or wrapped in the **core/data** layer of the application.
  - Example Error Handling in **core/data Layer:**

```kotlin
suspend fun safeApiCall(
    apiCall: suspend () -> HttpResponse
): ResultWrapper<HttpResponse> {
    return try {
        val response = apiCall()
        if (response.status.isSuccess()) {
            ResultWrapper.Success(response)
        } else {
            ResultWrapper.Error("API error: ${response.status}")
        }
    } catch (e: IOException) {
        ResultWrapper.Error("Network error: ${e.message}")
    } catch (e: Exception) {
        ResultWrapper.Error("Unknown error: ${e.message}")
    }
}
```

- This approach can be extended within the BaseApiManager or in service classes to centralize error handling for all endpoints.

2.6 Continue adding unit tests for Data Layer and UI Layer

Unit testing is a crucial part of ensuring the **stability, correctness, and reliability** of the Mifos Mobile 7.0 app. Since this project is migrating to **Kotlin Multiplatform (KMP)**, we need to structure tests in a way that works across platforms (Android, iOS, and possibly desktop/web).

Unlike traditional Android projects where unit tests are run with **JUnit** or UI tests with **Espresso**, KMP requires a **cross-platform testing approach**.

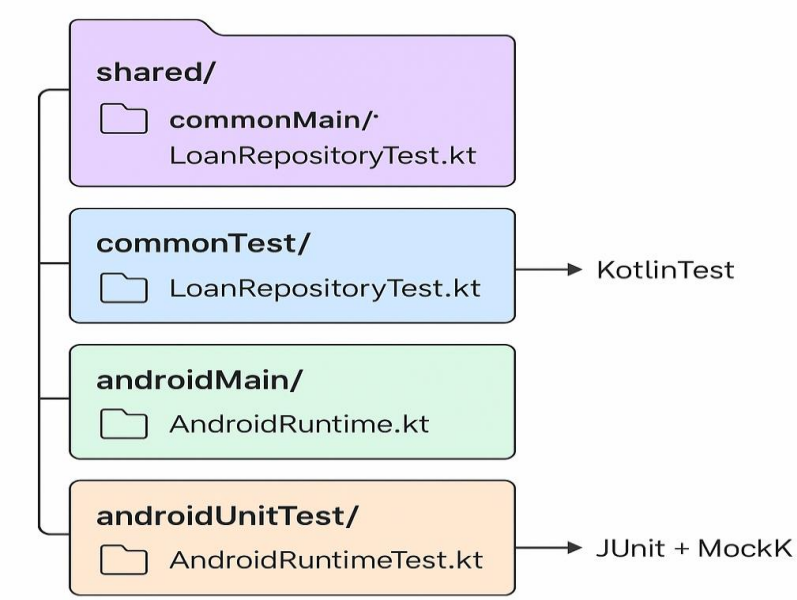
- **Common code testing** (Shared business logic, API calls, database, use cases) → Uses **Kotlin Test Framework** (kotlin.test)
- **Platform-specific testing** (Android UI, database, networking) → Uses platform-specific test libraries like **JUnit**, **MockK**, **Compose Testing**, etc.

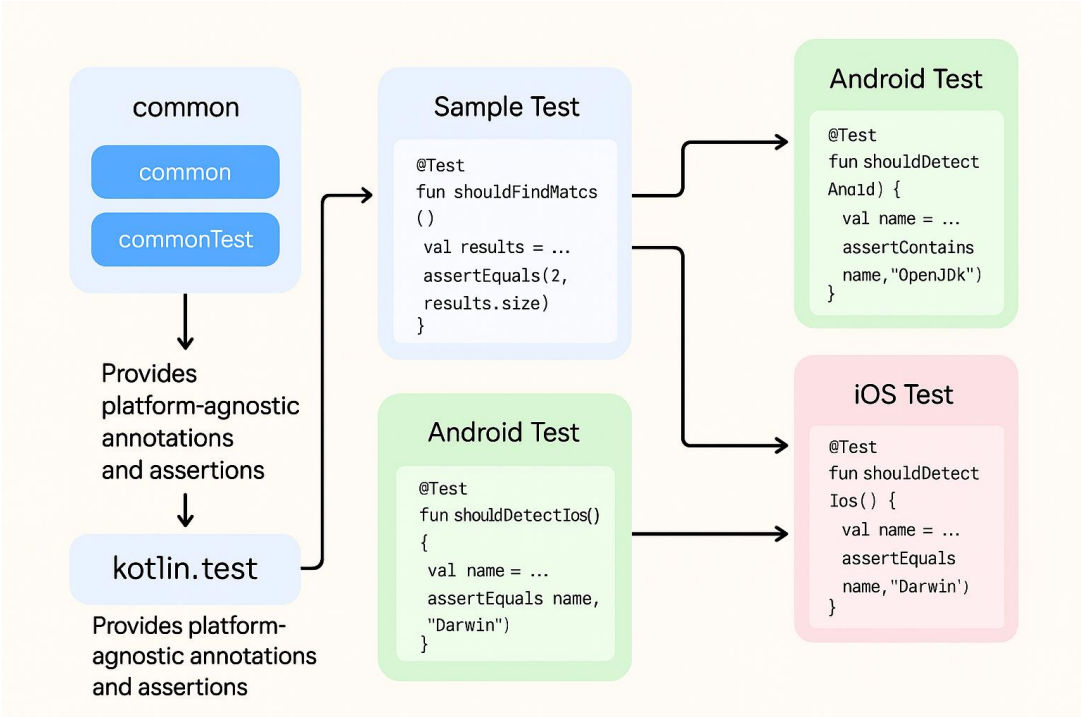
Layer	Test Type	Library	Usage
Common Code	Unit Tests	kotlin.test (Default)	General business logic, API interactions, data processing

Android-Specific	UI Tests	Jetpack Compose UI Testing	Verifying UI elements and interactions

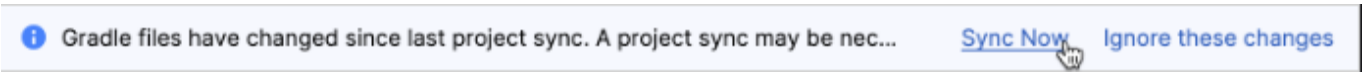
Steps to implement testing:

- An essential part will be a source set for common tests, which has the kotlin.test API library as a dependency.
- In the `shared` directory, open the `build.gradle.kts` file. Add a source set for testing the common code with a dependency on the `kotlin.test` library:

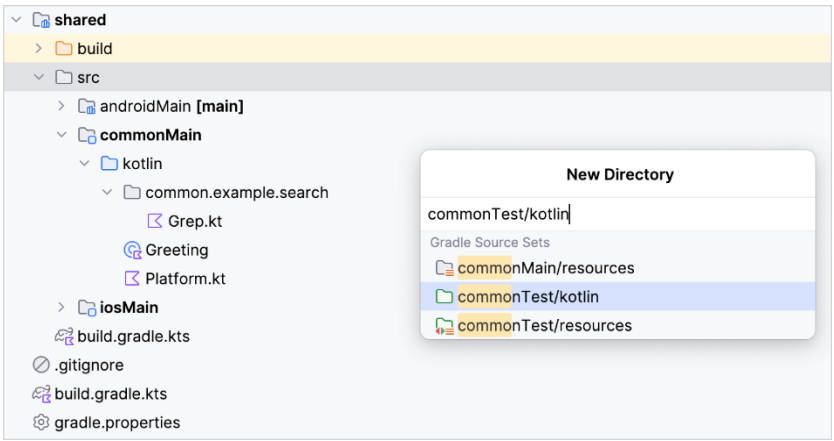




- Once the dependency is added, you're prompted to resync the project. Click Sync Now to synchronize Gradle files:



- The commonTest source set stores all common tests. Now you also need to create a directory with the same name in your project:
    - Right-click the `shared/src` directory and select New | Directory. The IDE will present a list of options.
    - Start typing the commonTest/kotlin path to narrow down the selection, then choose it from the list:



- In the `commonTest/kotlin` directory, create a new `common.example.search` package.
- In that package we can start writing unit tests for the logic

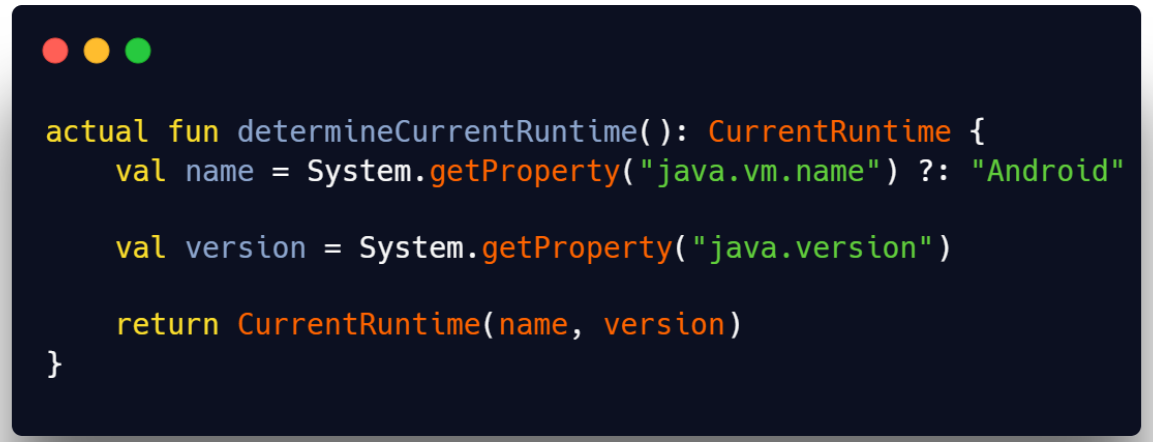
```kotlin
import kotlin.test.Test
import kotlin.test.assertContains
import kotlin.test.assertEquals

class GrepTest {
    companion object {
        val sampleData = listOf(
            "123 abc",
            "abc 123",
            "123 ABC",
            "ABC 123"
        )
    }

    @Test
    fun shouldFindMatches() {
        val results = mutableListOf<String>()
        grep(sampleData, "[a-z]+") {
            results.add(it)
        }

        assertEquals(2, results.size)
        for (result in results) {
            assertContains(result, "abc")
        }
    }
}
```

- The kotlin.test library provides platform-agnostic annotations and assertions for you to use in your tests. Annotations, such as Test, map to those provided by the selected framework or their nearest equivalent.

- Assertions are executed through an implementation of the Asserter interface. This interface defines the different checks commonly performed in testing. The API has a default implementation, but typically you will use a framework-specific implementation.
- For example, the JUnit 4, JUnit 5, and TestNG frameworks are all supported on JVM. On Android, a call to assertEquals() might result in a call to asserter.assertEquals(), where the asserter object is an instance of JUnit4Asserter. On iOS, the default implementation of the Asserter type is used in conjunction with the Kotlin/Native test runner.
- We can written tests for common code as well as for platform specific
- **Platform specific tests:**
- We can write test for platform specific using expect/actual classes

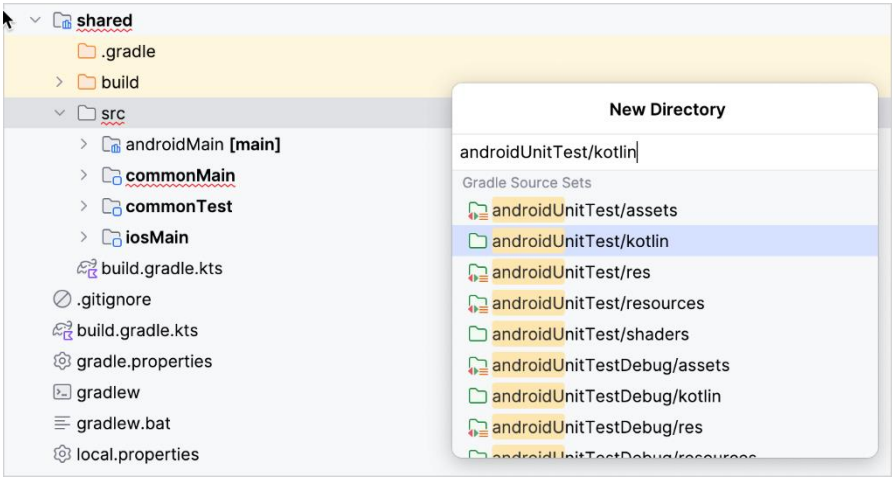
```
expect fun determineCurrentRuntime(): CurrentRuntime
```

- **For Android**
  - In the `androidMain/kotlin` directory, create a new `org.kmp.testing` package.
  - In this package, create the AndroidRuntime.kt file and update it with the actual implementation of the expected determineCurrentRuntime() function:



```
actual fun determineCurrentRuntime(): CurrentRuntime {
    val name = System.getProperty("java.vm.name") ?: "Android"

    val version = System.getProperty("java.version")

    return CurrentRuntime(name, version)
}
```

- Create a directory for tests inside the shared/src directory:
  - Right-click the `shared/src` directory and select New | Directory. The IDE will present a list of options.
  - Start typing the `androidUnitTest/kotlin` path to narrow down the selection, then choose it from the list:



- In the kotlin directory, create a new org.kmp.testing package.
- In this package, create the AndroidRuntimeTest.kt file and update it with the following Android test:

```kotlin
import kotlin.test.Test
import kotlin.test.assertContains
import kotlin.test.assertEquals

class AndroidRuntimeTest {
    @Test
    fun shouldDetectAndroid() {
        val runtime = determineCurrentRuntime()
        assertContains(runtime.name, "OpenJDK")
        assertEquals(runtime.version, "17.0")
    }
}
```

- Here's a **concise example** of how you might write unit tests for a **data layer** in a Kotlin Multiplatform (KMP) project, involving:
    - A LoanRepository interface (in commonMain)
    - A LoanRepositoryImpl implementation (in commonMain)
    - A unit test (in commonTest)

- **LoanRepository Interface:**

```
interface LoanRepository {
    suspend fun getLoanStatus(loanId: Int): String
}
```

- **LoanRepository Implementation:**

```
class LoanRepositoryImpl(
    private val loanApi: LoanApi
) : LoanRepository {
    override suspend fun getLoanStatus(loanId: Int): String {
        val response = loanApi.fetchLoanStatus(loanId)
        return response.status ?: "Unknown"
    }
}
```

- **Assume LoanApi is a KtorFit-generated interface:**

```
interface LoanApi {
    suspend fun fetchLoanStatus(loanId: Int): LoanResponse
}

data class LoanResponse(val status: String?)
```

- **Unit Test**

```
import kotlin.test.*
import kotlinx.coroutines.test.runTest

class LoanRepositoryTest {

    private class FakeLoanApi : LoanApi {
        override suspend fun fetchLoanStatus(loanId: Int): LoanResponse {
            return LoanResponse(status = if (loanId == 1) "Approved" else null)
        }
    }

    @Test
    fun testGetLoanStatusApproved() = runTest {
        val repo = LoanRepositoryImpl(FakeLoanApi())
        val result = repo.getLoanStatus(1)
        assertEquals("Approved", result)
    }

    @Test
    fun testGetLoanStatusUnknown() = runTest {
        val repo = LoanRepositoryImpl(FakeLoanApi())
        val result = repo.getLoanStatus(999)
        assertEquals("Unknown", result)
    }
}
```

- **Similarly, we can write tests for different platforms**

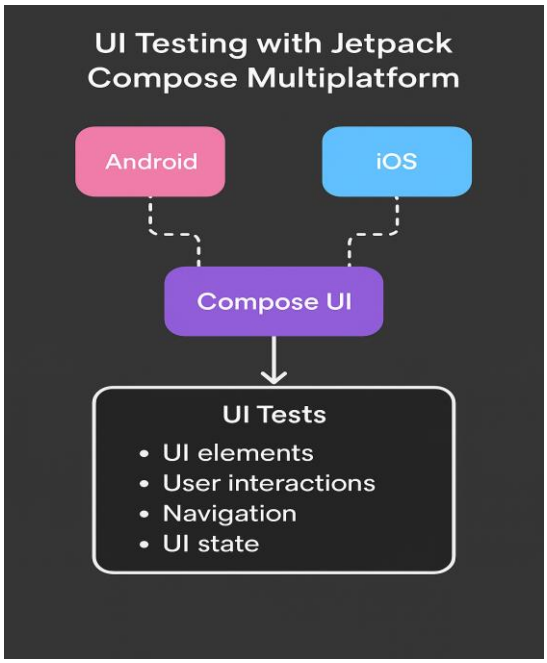
2.7 Cover all the screens with UI test

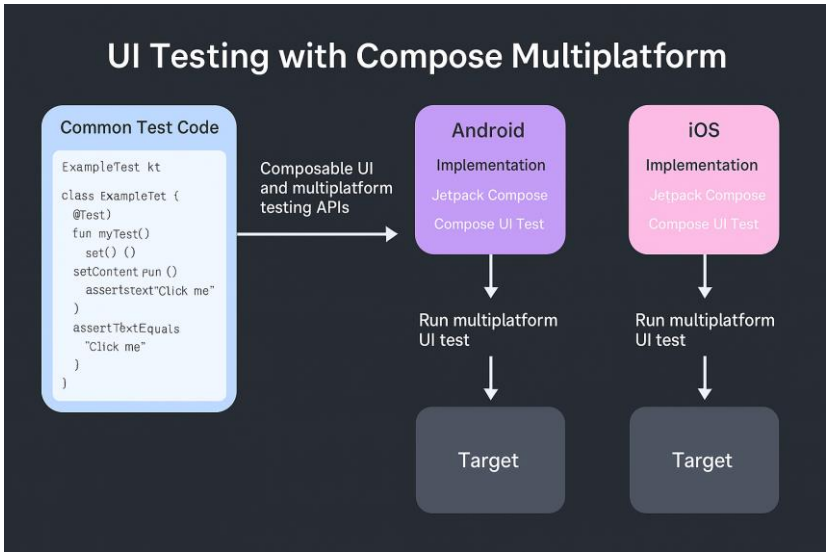
UI (User Interface) Testing is a process of verifying that an application's **user interface behaves as expected**. It ensures that all UI components—buttons, text fields, images, and navigation—work correctly and provide a smooth user experience.

In Jetpack Compose Multiplatform (CMP), UI testing focuses on both Android and iOS, making sure that:

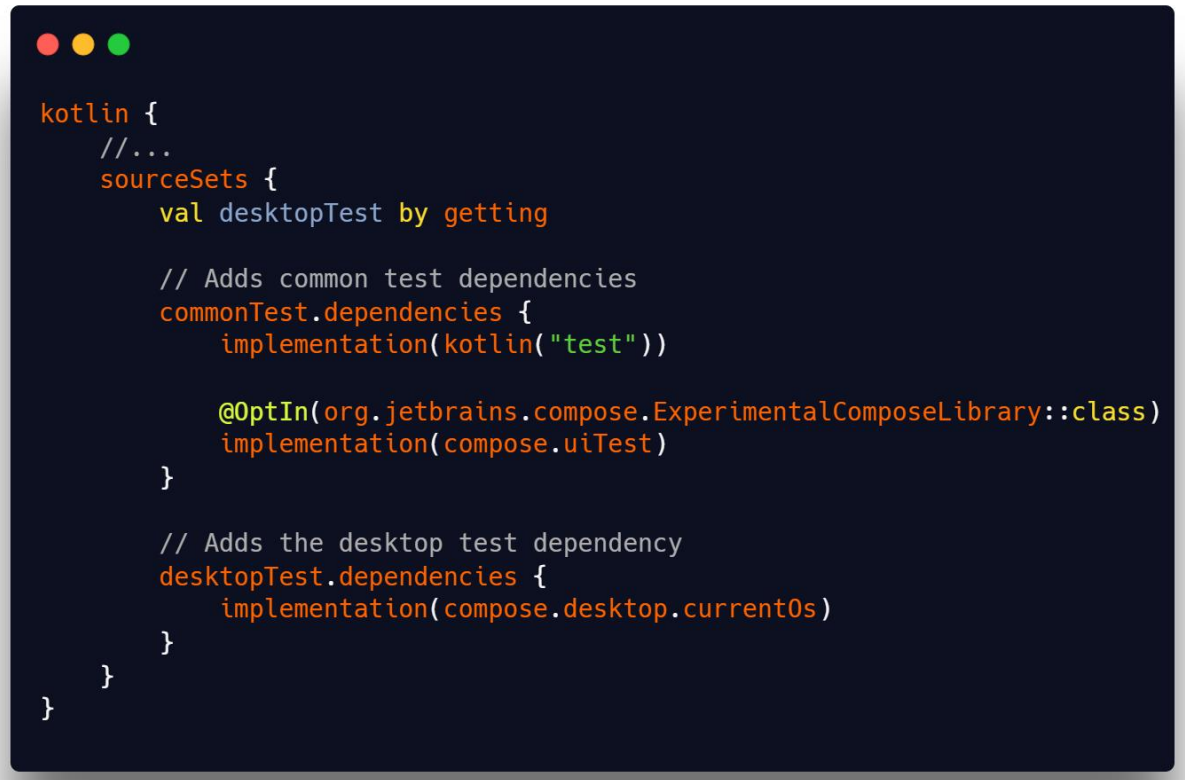
- The UI elements are rendered correctly.
- Users can interact with the UI without issues.
- Navigation and animations work smoothly.
- The UI updates correctly based on user inputs.

Since Mifos Mobile 7.0 is migrating to Jetpack Compose Multiplatform (CMP), UI testing should be properly set up to support both Android and iOS.



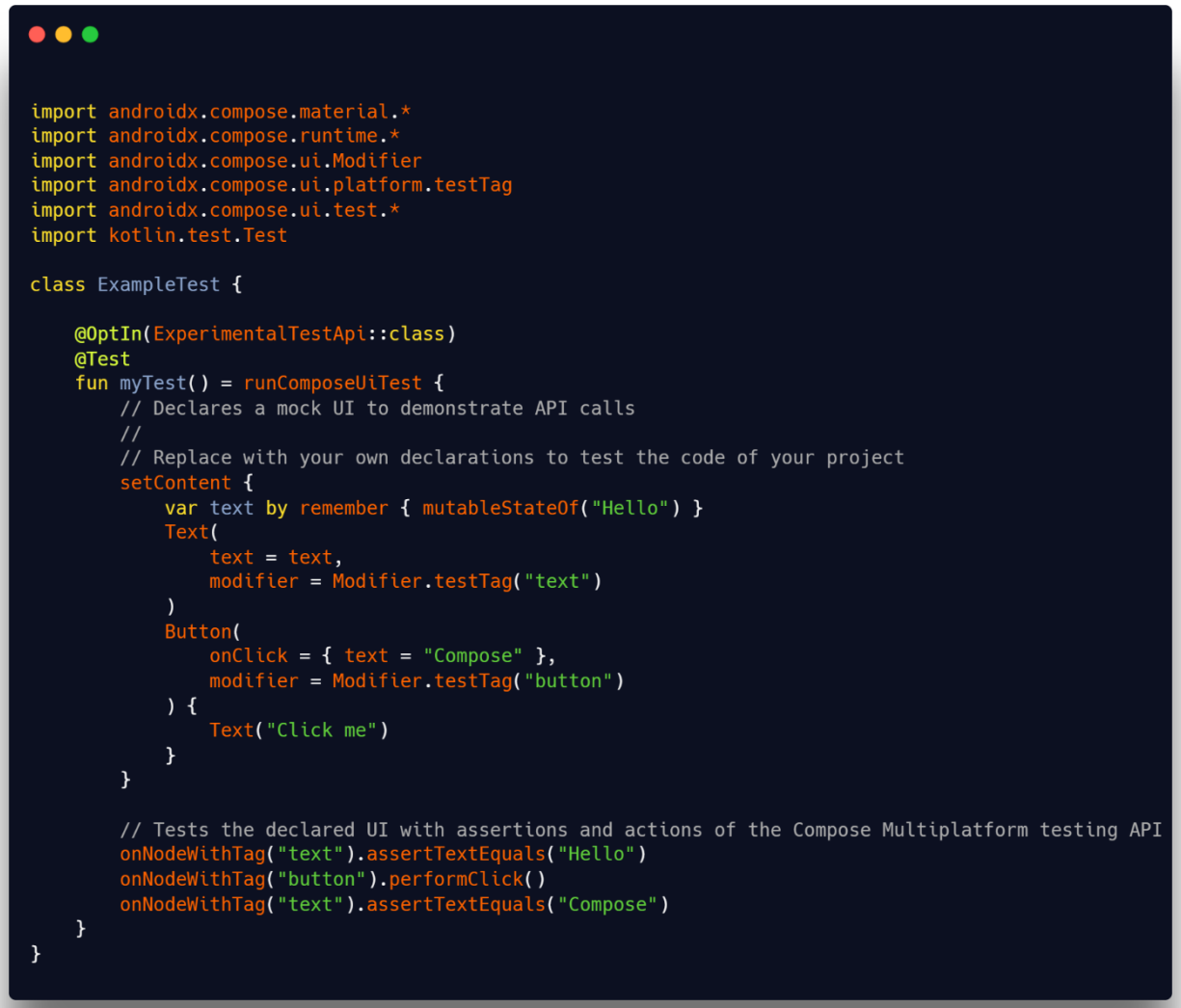


- **Add Dependencies for UI Testing**



```
kotlin {
    //...
    sourceSets {
        val desktopTest by getting

        // Adds common test dependencies
        commonTest.dependencies {
            implementation(kotlin("test"))

            @OptIn(org.jetbrains.compose.ExperimentalComposeLibrary::class)
            implementation(compose.uiTest)
        }

        // Adds the desktop test dependency
        desktopTest.dependencies {
            implementation(compose.desktop.currentOs)
        }
    }
}
```

- **Write and run common tests**

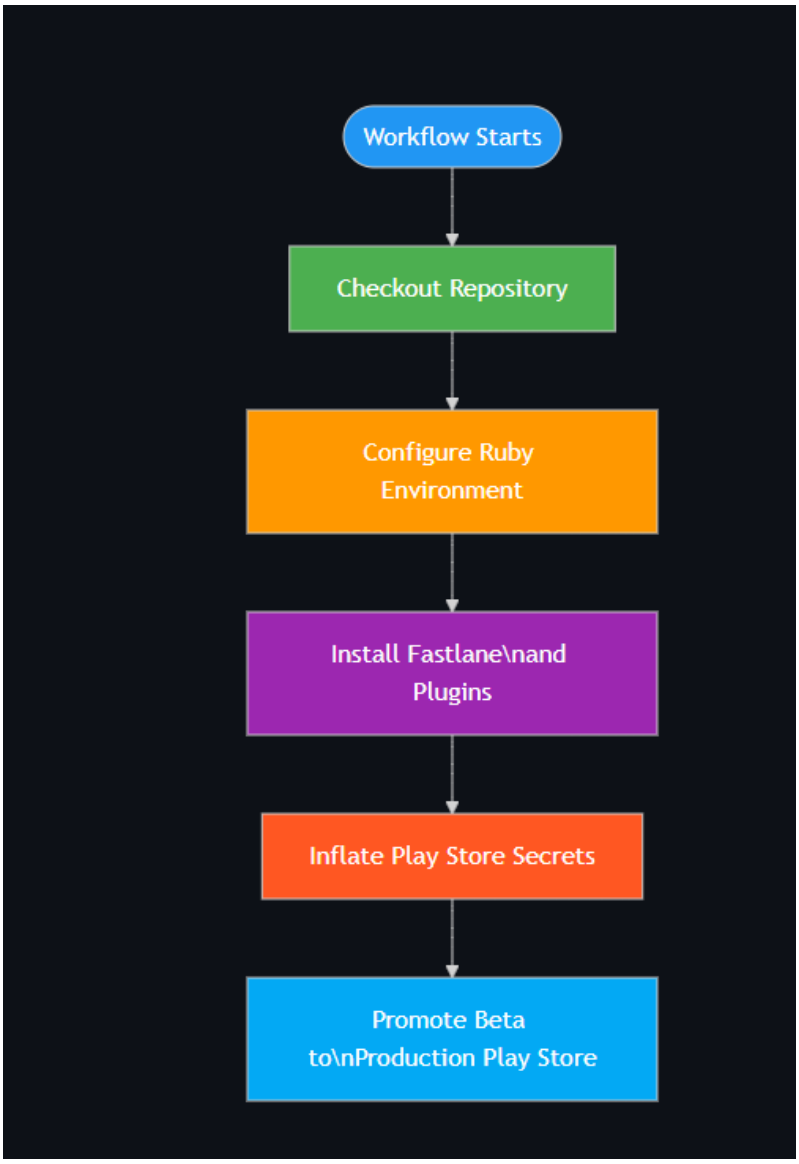


```kotlin
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.testTag
import androidx.compose.ui.test.*
import kotlin.test.Test

class ExampleTest {

    @OptIn(ExperimentalTestApi::class)
    @Test
    fun myTest() = runComposeUiTest {
        // Declares a mock UI to demonstrate API calls
        //
        // Replace with your own declarations to test the code of your project
        setContent {
            var text by remember { mutableStateOf("Hello") }
            Text(
                text = text,
                modifier = Modifier.testTag("text")
            )
            Button(
                onClick = { text = "Compose" },
                modifier = Modifier.testTag("button")
            ) {
                Text("Click me")
            }
        }

        // Tests the declared UI with assertions and actions of the Compose Multiplatform testing API
        onNodeWithTag("text").assertTextEquals("Hello")
        onNodeWithTag("button").performClick()
        onNodeWithTag("text").assertTextEquals("Compose")
    }
}
```

- In this way, we have to cover all the screens with UI testing for better test coverage, ensuring that every user interaction, navigation flow, and UI state is validated across both Android and iOS platforms in Jetpack Compose Multiplatform.

2.8 Implemented Playstore release github action pipeline

- **Branching Strategy:** Currently, all code is pushed to the development branch. To ensure a proper CI/CD flow, we should introduce a **release** branch dedicated to deploying the product to the Play Store. This would streamline version control and deployment processes.
- Fastlane for Deployment: Fastlane is the most efficient way to automate beta deployments and releases for both iOS and Android. It simplifies tasks such as generating screenshots, managing code signing, and handling app releases.
- Google Play Credential File: To deploy our app using Fastlane, we need a Google Play Credential file (.json). Assuming that Mifos has its own Google Play Developer account, we can generate this file and integrate it into our CI/CD pipeline.



- Current Fastlane Setup: We already have Fastlane configurations for both Android and iOS, which are used for deploying applications to Firebase Distribution. Moving forward, we can extend this setup to support Play Store and App Store deployments.
- This workflow automates the promotion of a beta release to the production environment on the Google Play Store.
- **Workflow Trigger**
  - Callable workflow (can be invoked from other workflows)
- **Prerequisites**
  - Ruby Environment
    - Requires Ruby setup (uses ruby/setup-ruby action)
    - Bundler version 2.2.27
    - Fastlane installed with specific plugins
  - Required Plugins
    - firebase_app_distribution
    - increment_build_number
- **Configuration Steps**
  - Ensure your repository is properly structured for Android app deployment
  - Have a Fastfile configured with promote_to_production lane

- **Fastlane Configuration Create a Fastfile in your fastlane directory with a promote_to_production lane:**

```
default(:android)
lane :promote_to_production do
  # Your specific Play Store promotion logic
  supply(
    track: 'beta',
    track_promote_to: 'production'
  )
end
```

- **Example Workflow**

```yaml
# Workflow triggers:
# 1. Manual trigger with option to publish to Play Store
# 2. Automatic trigger when a GitHub release is published
on:
  workflow_dispatch:
    inputs:
      publish_to_play_store:
        required: false
        default: false
        description: Publish to Play Store?
        type: boolean
  release:
    types: [ released ]

concurrency:
  group: "production-deploy"
  cancel-in-progress: false

permissions:
  contents: write

jobs:
  # Job to promote app from beta to production in Play Store
  play_promote_production:
    name: Promote Beta to Production Play Store
    uses: openMF/mifos-mobile-github-actions/.github/workflows/promote-to-production.yaml@main
    if: ${{ inputs.publish_to_play_store == true }}
    secrets: inherit
    with:
      android_package_name: 'mifospay-android'
```

- **This is sample script to deploy application to playstore**
- **End-to-End CI/CD Automation Strategy**
- **We'll use the prebuilt workflow:**

```yaml
uses: openMF/mifos-mobile-github-actions/.github/workflows/multi-platform-build-and-publish.yaml@main
```

- **This supports:**
  - **release_type: internal / beta**
  - **publish_android, publish_ios: control releases to Play Store / App Store**
  - **build_ios: optionally build iOS even if not releasing**
  - **tester_groups: Firebase distribution tester group**
  - **target_branch: e.g., release, dev**
- **Release Workflow for All Platforms**
  - **Android**
  - **Uses Fastlane via:**

```
openMF/mifos-mobile-github-actions/.github/workflows/promote-to-production.yaml@main
```

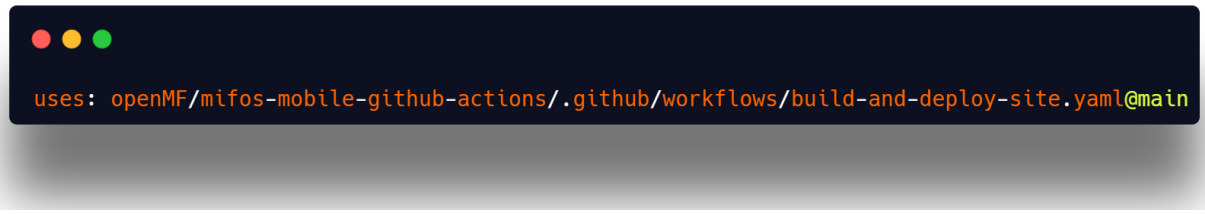
- **Uses Google Play credentials (.json) from secrets.**
- **Deploys directly to Play Store using supply.**
- **Fastfile Snippet:**

```
lane :promote_to_production do
  supply(
    track: 'beta',
    track_promote_to: 'production'
  )
end
```

- **Play Store Promotion Trigger Example:**

```yaml
on:
  workflow_dispatch:
    inputs:
      publish_to_play_store:
        type: boolean
  release:
    types: [ released ]
```

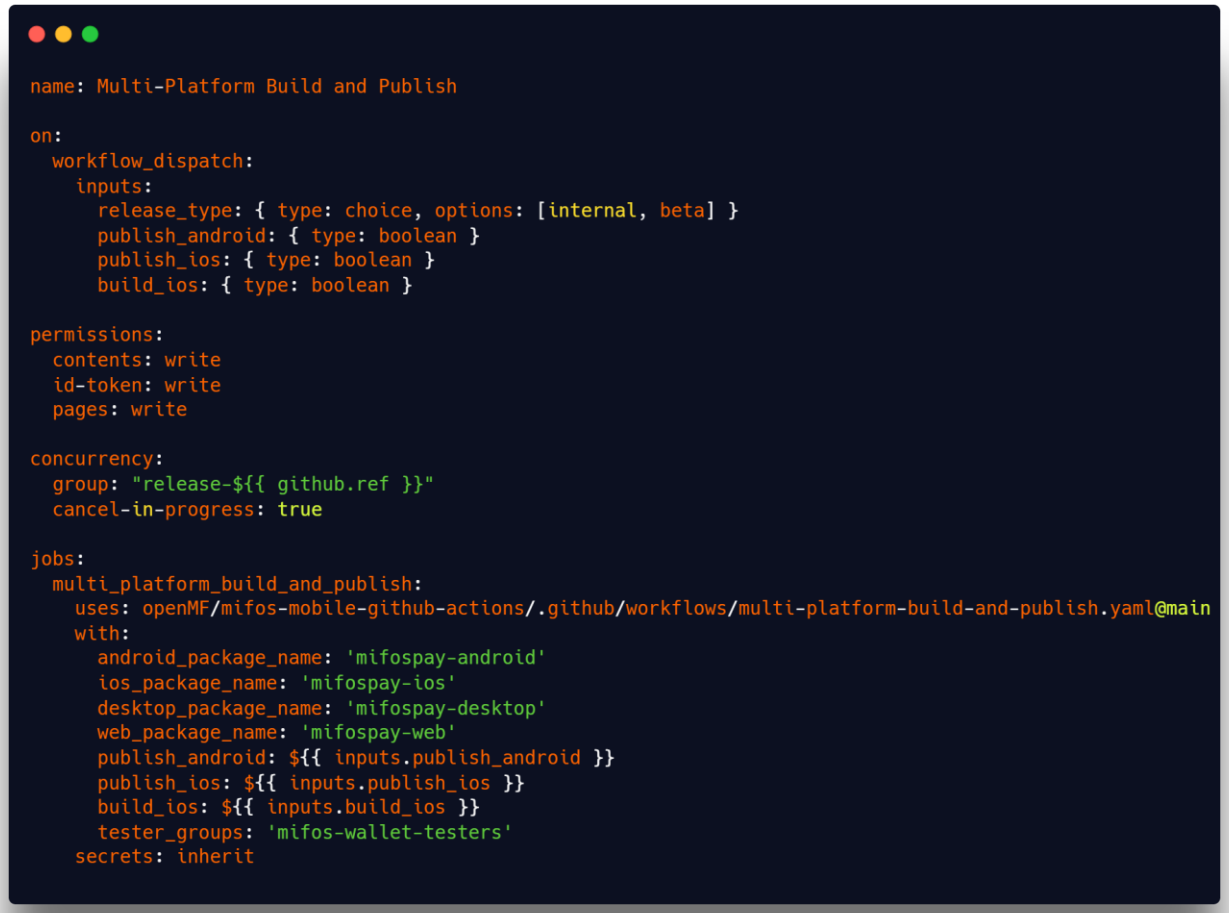
- **iOS**
  - **Utilizes Fastlane for:**
    - **Certificate handling**
    - **Firebase Distribution**
    - **TestFlight / App Store upload (future enhancement)**
  - **We'll extend the existing build_ios: true and publish_ios: true setup using:**
- **Web (Kotlin JS)**
  - **GitHub Pages Deployment via:**



```yaml
uses: openMF/mifos-mobile-github-actions/.github/workflows/build-and-deploy-site.yaml@main
```

  - **Requires web_package_name and GitHub Pages enabled in repo settings:**
    - **Source: GitHub Actions**
    - **Branch: auto-managed by action**
- **Desktop (Windows, macOS, Linux)**
  - **Builds executables**

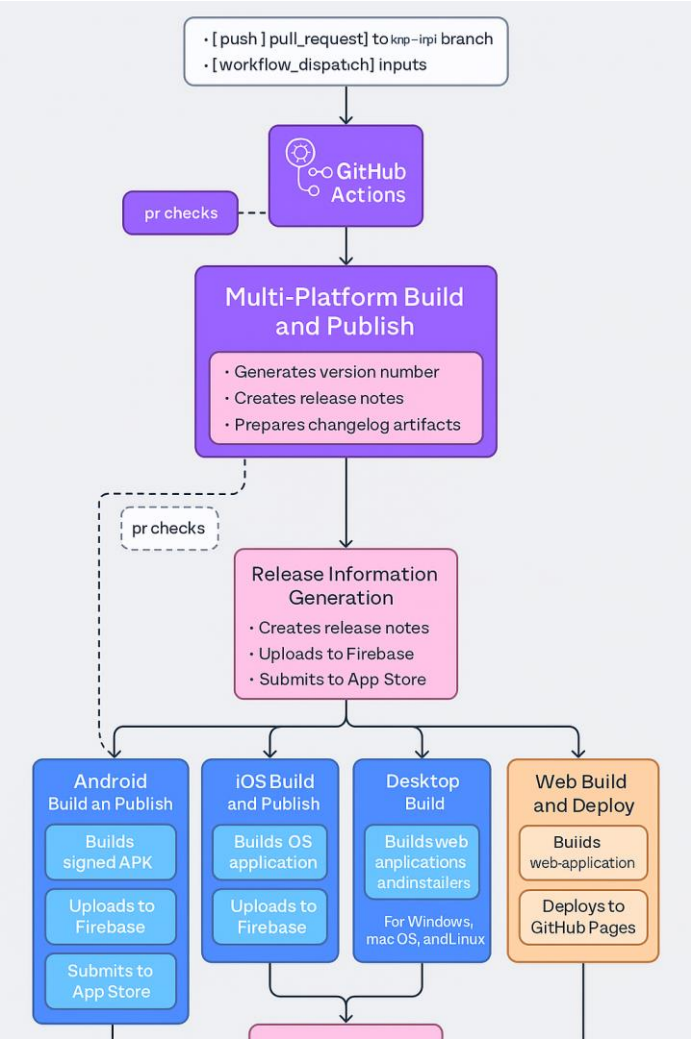
    - **Uploads to GitHub Releases as .exe, .dmg, .AppImage, etc.**
    - **Uses desktop_package_name to distinguish builds**
- **Final Workflow Sample**



```yaml
name: Multi-Platform Build and Publish

on:
  workflow_dispatch:
    inputs:
      release_type: { type: choice, options: [internal, beta] }
      publish_android: { type: boolean }
      publish_ios: { type: boolean }
      build_ios: { type: boolean }

permissions:
  contents: write
  id-token: write
  pages: write

concurrency:
  group: "release-${{ github.ref }}"
  cancel-in-progress: true

jobs:
  multi_platform_build_and_publish:
    uses: openMF/mifos-mobile-github-actions/.github/workflows/multi-platform-build-and-publish.yaml@main
    with:
      android_package_name: 'mifospay-android'
      ios_package_name: 'mifospay-ios'
      desktop_package_name: 'mifospay-desktop'
      web_package_name: 'mifospay-web'
      publish_android: ${{ inputs.publish_android }}
      publish_ios: ${{ inputs.publish_ios }}
      build_ios: ${{ inputs.build_ios }}
      tester_groups: 'mifos-wallet-testers'
    secrets: inherit
```

2.9 Improve Github workflows and add jobs to run Unit and UI test

- As part of the Mifos Mobile 7.0 project, I propose enhancing our GitHub Actions workflow to ensure a seamless Continuous Integration (CI) pipeline for the Kotlin Multiplatform (KMP) implementation. Currently, we have a well-structured workflow that automates essential tasks such as code quality checks, dependency verification, and multi-platform builds for Android, Desktop (Windows, Linux, MacOS), Web, and iOS. This setup allows us to maintain code consistency, detect issues early, and ensure a smooth development process across all platforms.
- The workflow is automatically triggered on every push and pull request to the kmp-impl branch, ensuring that all new code meets project standards before merging. To improve efficiency, I suggest optimizing concurrency settings to prevent unnecessary duplicate runs and speed up execution. The main job, pr checks, is structured using a reusable GitHub Actions workflow from the Mifos Mobile repository, making it easier to manage and update across multiple projects. Additionally, we inherit necessary secrets to securely handle API integrations, build processes, and deployment configurations.
- One of the key areas I plan to improve is test automation within this pipeline. By integrating unit tests and UI tests execution, we can ensure that all components function correctly before deployment. Furthermore, generating detailed test reports will improve debugging and provide better visibility into



test coverage. Another critical improvement is parallel execution of test jobs, which will significantly reduce build times and speed up feedback loops for developers. These enhancements will strengthen our quality assurance process and make the development workflow more reliable.

- To streamline deployment, I propose automating the Play Store release process using Fastlane. This will handle tedious tasks such as signing, versioning, and uploading APKs directly from GitHub Actions, reducing manual effort and ensuring a structured release cycle. Additionally, we will improve GitHub workflows by adding dedicated jobs for running unit tests and UI tests, ensuring comprehensive validation at every stage. This will align with our goal of fully covering all screens with UI tests, reinforcing stability and preventing regressions in the mobile banking app.
- With these improvements, our CI/CD pipeline will be more efficient, scalable, and robust, allowing us to deliver a higher-quality mobile banking experience. By automating testing, optimizing workflows, and streamlining deployment, we will enhance developer productivity, improve stability, and accelerate releases. This will be a crucial step in ensuring the long-term success of Mifos Mobile 7.0.
- **Workflow Jobs**
    - **Release Information Generation**
        - **Generates version number**
        - **Creates release notes**
        - **Prepares changelog artifacts**
    - **Platform-Specific Build Jobs**
        - **Android**
            - **Builds signed APK**
            - **Uploads to Firebase App Distribution**
            - **Optionally publishes to Play Store**
        - **iOS**
            - **Builds iOS application**

        - **Uploads to Firebase App Distribution**
        - **Prepares for App Store submission**
    - **Desktop**
        - **Builds for Windows, macOS, and Linux**
        - **Packages executables and installers**
    - **Web**
        - **Builds web application**
        - **Deploys to GitHub Pages**
    - **GitHub Release**
        - **Creates a pre-release with all built artifacts**
        - **Includes detailed changelog**
- **Workflow usage example**

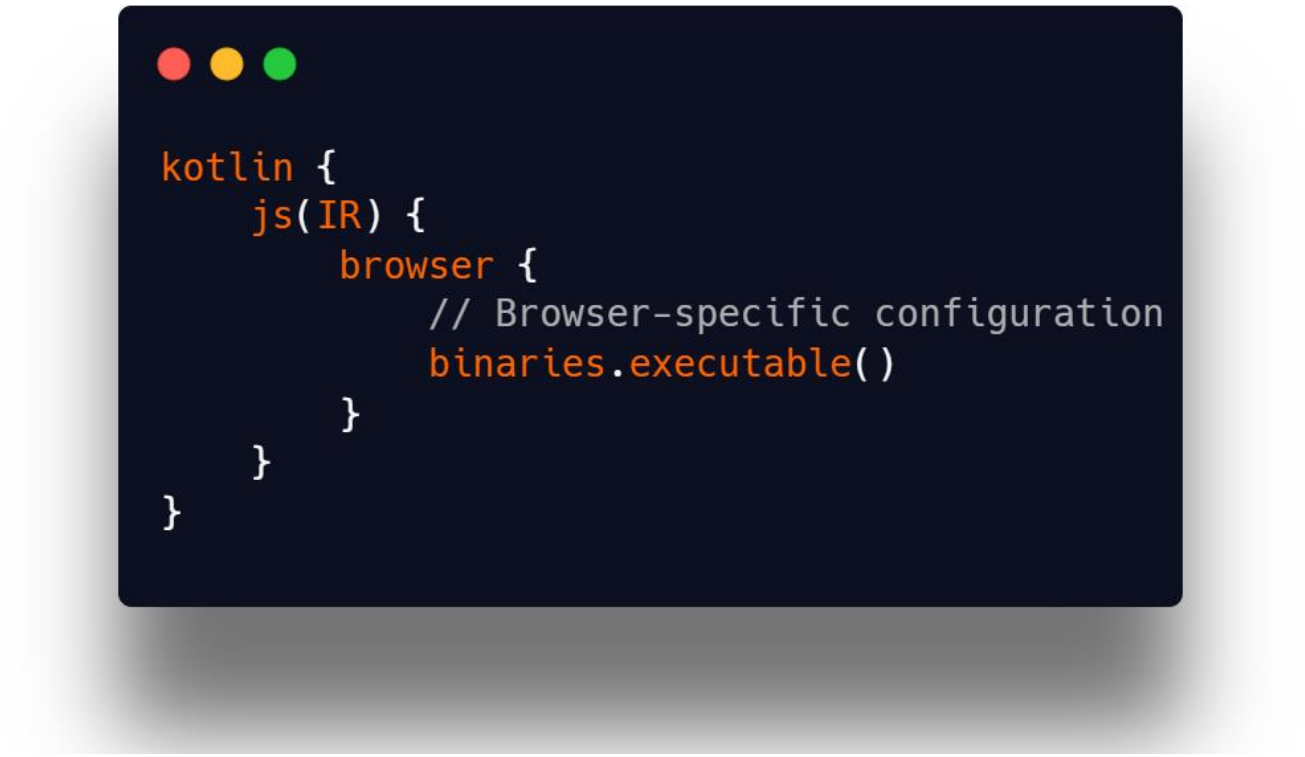
```yaml
name: Multi-Platform Build and Publish

on:
  workflow_dispatch:
    inputs:
      release_type:
        type: choice
        options:
          - internal
          - beta
        default: internal
        description: Release Type

      target_branch:
        type: string
        default: 'dev'
        description: 'Target branch for release'

      publish_android:
        type: boolean
        default: false
        description: Publish Android App On Play Store

      build_ios:
        type: boolean
        default: false
        description: Build iOS App

      publish_ios:
        type: boolean
        default: false
        description: Publish iOS App On App Store

permissions:
  contents: write
  id-token: write
  pages: write

concurrency:
  group: "reusable"
  cancel-in-progress: false

jobs:
  multi_platform_build_and_publish:
    name: Multi-Platform Build and Publish
    uses: openMF/mifos-mobile-github-actions/.github/workflows/multi-platform-build-and-publish.yaml@main
    with:
      release_type: ${{ inputs.release_type }}
      target_branch: ${{ inputs.target_branch }}
      android_package_name: 'mifospay-android' # <-- Change this to your android package name
      ios_package_name: 'mifospay-ios' # <-- Change this to your ios package name
      desktop_package_name: 'mifospay-desktop' # <-- Change this to your desktop package name
      web_package_name: 'mifospay-web'   # <-- Change this to your web package name
      publish_android: ${{ inputs.publish_android }}
      build_ios: ${{ inputs.build_ios }}
      publish_ios: ${{ inputs.publish_ios }}
      tester_groups: 'mifos-wallet-testers'
    secrets:
      original_keystore_file: ${{ secrets.ORIGINAL_KEYSTORE_FILE }}
      original_keystore_file_password: ${{ secrets.ORIGINAL_KEYSTORE_FILE_PASSWORD }}
      original_keystore_alias: ${{ secrets.ORIGINAL_KEYSTORE_ALIAS }}
      original_keystore_alias_password: ${{ secrets.ORIGINAL_KEYSTORE_ALIAS_PASSWORD }}

// Remaining Secrets you can provide
```

- **How to Setup the Workflow (Eg: Web)**

- This workflow is designed as a reusable workflow. We'll need to call it from another workflow file. Create a workflow file (e.g., .github/workflows/deploy.yml) that looks like this:

```yaml
name: Build And Deploy Web App

# Trigger conditions for the workflow
on:
  pull_request:
    branches: [ "dev" ]
    types: [ closed ]
  workflow_dispatch:

# Concurrency settings to manage multiple workflow runs
# This ensures orderly deployment to production environment
concurrency:
  group: "web-pages"
  cancel-in-progress: false

permissions:
  contents: read  # Read repository contents
  pages: write    # Write to GitHub Pages
  id-token: write # Write authentication tokens
  pull-requests: write # Write to pull requests

jobs:
  build_and_deploy_web:
    name: Build And Deploy Web App
    uses: openMF/mifos-mobile-github-actions/.github/workflows/build-and-deploy-site.yaml@main
    secrets: inherit
    with:
      web_package_name: mifos-web'
```

- **GitHub Pages Configuration**
- **Repository Settings**
  - **Go to repository's "Settings" tab**
  - **Navigate to "Pages" section**
  - **Under "Source", select "GitHub Actions" as the deployment method**
- **Gradle Configuration**



- This is the **Kotlin/JS Web Application GitHub Pages Deployment Workflow**

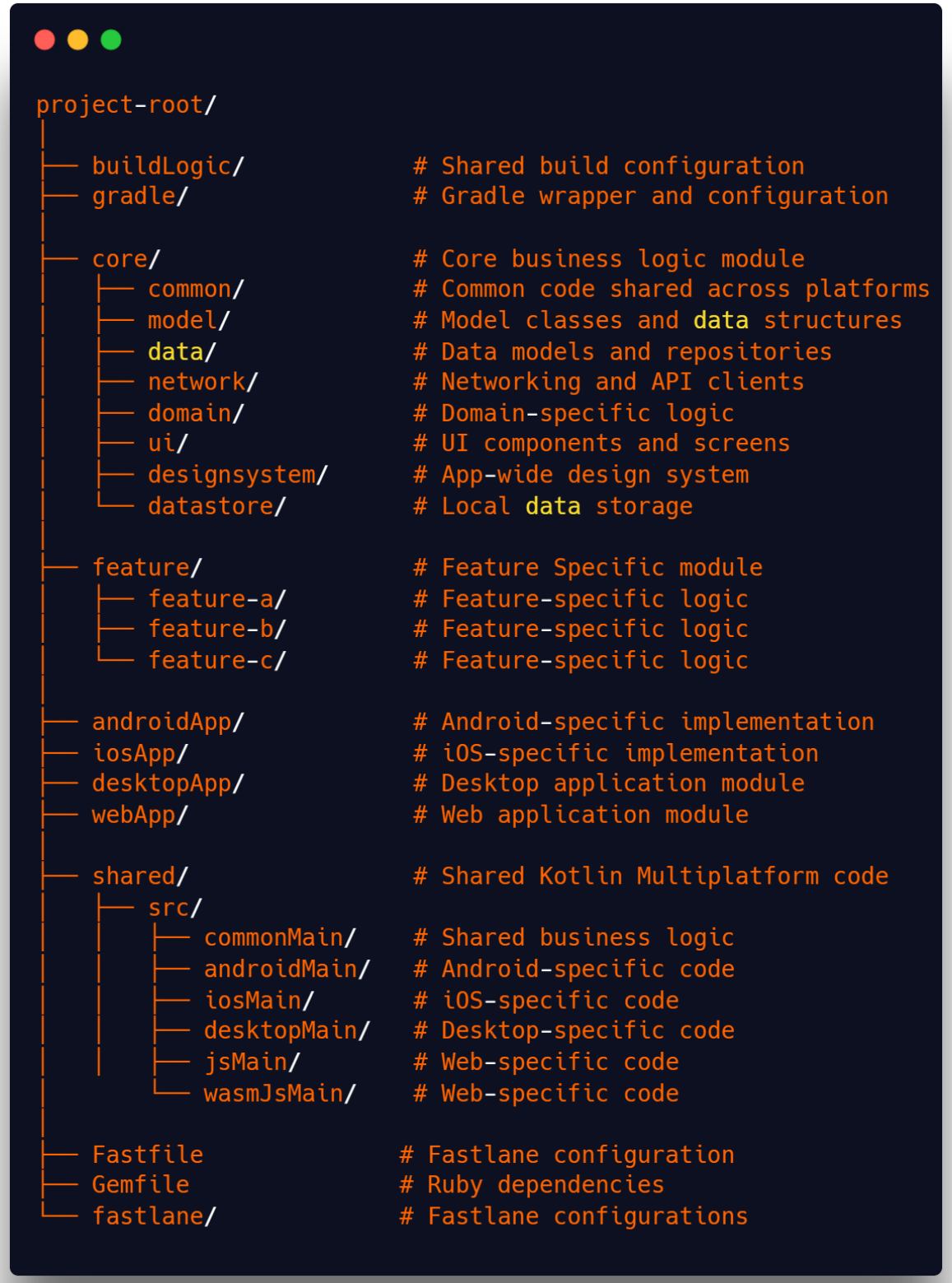
2.10 Add CI to build APK and code analysis.

- To ensure a robust and automated development workflow, I propose integrating Continuous Integration (CI) for building APKs and performing comprehensive code analysis within our GitHub Actions workflow. This will streamline the development process by automatically building APKs, analysing code quality, and detecting potential issues early, reducing manual effort and ensuring a high standard of code quality. By automating these tasks, we can significantly improve developer productivity, maintainability, and the stability of the Mifos Mobile 7.0 application.
- The CI pipeline will automatically build APKs for multiple flavors (debug, release, and test variants) whenever a new change is pushed to the repository or a pull request is created. This ensures that we always have the latest application builds available for testing and deployment. The generated APKs can be stored as workflow artifacts, allowing developers and testers to download and verify the latest version without needing to build the app locally. This step will be crucial in improving our testing and release processes, ensuring that each commit results in a fully functional and installable application package.
- The existing CI pipeline already integrates essential code analysis tools such as Detekt for Kotlin static analysis, Spotless for code formatting enforcement, and Dependency Guard for dependency tracking and verification. These tools play a crucial role in ensuring high code quality, identifying security vulnerabilities, and enforcing best practices. Moving forward, I propose optimizing and extending their configurations to ensure maximum effectiveness. This includes fine-tuning rules, reducing false positives, and enhancing reporting within GitHub Actions. Additionally, ensuring that these tools run efficiently within the CI workflow by leveraging Gradle caching and parallel execution will further

streamline the process. By maintaining and refining these automated checks, we can continue to uphold coding standards, prevent regressions, and provide developers with instant feedback on their contributions.

- To optimize efficiency, I propose leveraging Gradle's build caching and parallel execution to speed up the CI pipeline. This will reduce redundant builds and improve execution times, ensuring that code analysis and APK generation complete faster. Additionally, the pipeline will be structured to only trigger full builds when necessary, avoiding unnecessary computation on minor documentation or configuration changes. By making these optimizations, we can significantly reduce CI run times and improve the overall development workflow.

- We already have github action library so now we have to integrate in to mifos mobile project.
- Supported platforms
    - Android
    - IOS
    - Desktop (Windows, linux, macOS)
    - Web (Kotlin/JS, wasmJS)
- This is the recommended project structure to integrate the github repository



- **Fastlane** is an open-source automation tool primarily used for **continuous deployment (CD)** in mobile development. It simplifies and automates tasks like building, testing, signing, and deploying mobile apps to app stores (Google Play Store, Apple App Store).

- **Fastfile Configuration**
- This is for android Assemble and Releasing APK

```
default_platform(:android)

platform :android do
  desc "Assemble debug APKs."
  lane :assembleDebugApks do |options|

  gradle(
      tasks: ["assembleDebug"],
    )
  end

  desc "Assemble Release APK"
  lane :assembleReleaseApks do |options|
    options[:storeFile] ||= "release_keystore.keystore"
    options[:storePassword] ||= "Mifospay"
    options[:keyAlias] ||= "key0"
    options[:keyPassword] ||= "Mifos@123"

    # Generate version
    generateVersion = generateVersion()

    buildAndSignApp(
      taskName: "assemble",
      buildType: "Release",
      storeFile: options[:storeFile],
      storePassword: options[:storePassword],
      keyAlias: options[:keyAlias],
      keyPassword: options[:keyPassword],
    )
  end
```

Code Quality Checks

- **Static code analysis (Detekt)**
- **Code formatting (Spotless)**
- **Dependency guard**

- **Spotless Custom Plugin**

```kotlin
class MifosSpotlessConventionPlugin : Plugin<Project> {
    override fun apply(target: Project) {
        with(target) {
            applyPlugins()

            spotlessGradle {
                configureSpotless(this)
            }
        }
    }

    private fun Project.applyPlugins() {
        pluginManager.apply {
            apply("com.diffplug.spotless")
        }
    }
}
```

- Command to apply spotless

- **Spotless Custom Plugin**

```kotlin
class MifosDetektConventionPlugin : Plugin<Project> {
    override fun apply(target: Project) {
        with(target) {
            applyPlugins()

            detektGradle {
                configureDetekt(this)
            }
        }
    }

    private fun Project.applyPlugins() {
        pluginManager.apply {
            apply("io.gitlab.arturbosch.detekt")
        }
    }
}
```

- Command to apply detekt

- Register these plugins in plugin block in root gradle file (These plugins are defined in build-logic module).

3. Work Wise Breakdown

3.1 Community Bonding Period (8 May - 1 June)

Week 1

- **Get in touch with the developers and the mentor.**
  I will initiate communication with my mentor and other contributors. I'll introduce myself, understand their expectations, and establish preferred channels (Slack, GitHub, or email). I'll also finalize a schedule for weekly check-ins and updates.
- **Introduction to the community, to the mentor and fix timings to communicate**
  I'll actively participate in community discussions, introduce my project, and engage with contributors. I'll also confirm a consistent time to connect with my mentor and discuss weekly progress or blockers.
- **Discuss any suggestions and changes to the project. There could modifications, new additions or amendments; it would be better to go over these early**

I'll present my project plan and implementation approach to the mentor. I'll be open to feedback, modifications, or new ideas that can enhance the scope or flow. This will help align the project early on and prevent surprises during implementation.

Week 2

- **Go Through Mifos Mobile codebase**
  I will thoroughly explore the Mifos Mobile codebase, understand its architecture, modules, and design patterns. I'll identify areas that need migration to Kotlin Multiplatform and list down important dependencies.
- **Address issues that can be a hurdle during the GSoC**
  During this time, I'll flag build issues, outdated dependencies, or any blockers that could affect progress. I'll try to resolve what I can and bring up the rest for community input.
- **Go through Open Banking API and Mojaloop documentation for understanding flows**
  I'll read through the Mojaloop and Open Banking specifications to understand the transaction flows, security practices, and how the APIs are expected to work. This will guide the development of new features and UI screens.

Week 3

- **Discuss the working of the existing application with the mentor**
  I'll walk through the current Mifos Mobile app with my mentor to understand how different modules and features work. This discussion will help me identify what to improve and what can be retained.
- **Finalize the new UI design and discuss flow enhancements**
  I'll review the finalized mockups with my mentor and design team, confirm user flows, and note any changes that improve usability. I'll make sure all design assets are ready for implementation.
- **Start creating reusable UI components for the new design (buttons, input fields, layouts, etc.)**
  I'll begin writing reusable components like buttons, cards, input fields, etc., using Jetpack Compose. This will make it easier to build screens consistently across the app.

3.2 Phase 1 (2 June - 18 July)

Week 4

- **Start implementing new UI screens using Jetpack Compose**
  I'll begin transforming the designs into actual screens using Jetpack Compose, starting with high-priority modules like Login, Dashboard, and Accounts.
- **Integrate design system, colors, typography, and layout structure**
  I'll set up the theme files, define typography styles, and maintain visual consistency throughout the app based on the new design system.
- **Ensure proper navigation setup for all new screens**
  I'll configure navigation using Compose Navigation, define routes, pass arguments, and ensure smooth transitions between screens.

Week 5

- **Complete the UI design implementation.**
  I'll wrap up the remaining UI screens and ensure all states (loading, empty, error) are visually handled.
- **Begin replacing the API layer from self-service Fineract APIs to the self-service middleware layer.**

I'll refactor the data layer to use the new middleware APIs, rewrite models and mappers accordingly, and make sure the new layer is testable and clean.

- **Discuss API replacements with the mentor and collaborate with the backend team for better understanding.**
  I'll work closely with my mentor and the backend team to understand how to consume the new APIs, clarify any discrepancies, and ensure correct data flow.

Week 6

- **Implement Mojaloop-related UI flows such as transfer form, lookup, and confirmation**
  I'll begin implementing Mojaloop modules such as phone number lookup, transfer form, and confirmation screens.
- **Use mock data for developing and testing the UI**
  I'll simulate API responses using mock repositories so that UI development doesn't get blocked due to incomplete backend APIs.
- **Validate all form field states (valid, error, loading, disabled, etc.)**
  I'll handle all form scenarios like validation errors, loading spinners, disabled states, and success messages for a seamless user experience.

Week 7

- **Finalize Mojaloop-related flows in UI**
  I'll polish the Mojaloop UI implementation, test it thoroughly, and ensure it works well with all required input and output cases.
- **Add support for UI states (success, failure, loading) using Snackbars, animations, etc.**
  I'll improve user feedback by adding snackbars, loading animations, and visual cues for different UI states.
- **Ensure edge-case handling like invalid phone numbers, timeouts, and retries**
  I'll test for and handle edge cases like incorrect input, slow internet, and server errors to ensure the app remains robust.

Week 8 and Rest of Phase 1

- **Allocate buffer time for any pending tasks.**
  I'll use this time to revisit incomplete items, fix bugs, and refine the existing implementation.
- **Begin writing unit tests, focusing on the data layer to ensure reliability.**
  I'll write unit tests for repositories and view models to ensure that core business logic is reliable and testable.
- **Prepare a report for the mid-term evaluation and discuss the next steps with mentors.**
  I'll document everything I've achieved so far, the issues I've tackled, and what I plan to do in Phase 2. I'll review this with my mentor.

3.3 Phase 2 (19 July - 25 Aug)

Week 10

- **Start writing UI tests for all screens to ensure complete test coverage.**
  I'll use Jetpack Compose testing libraries to write UI tests that validate the user interactions and screen behavior.
- **Test flows end-to-end: login → dashboard → Mojaloop transfer → confirmation**

I'll test full user flows (e.g., login → dashboard → transfer → confirmation) to ensure the app works as intended.

Week 11

- **Discuss Play Store release automation with the mentor and explore improvements in GitHub Action workflows.**
  I'll discuss automating beta and production releases using GitHub Actions and Fastlane to speed up deployment.
- **Optimize GitHub workflows to improve CI/CD efficiency.**
  I'll make changes to the CI pipeline to ensure faster builds and efficient dependency caching.
- **Validate that unit tests and UI tests run successfully without failures.**
  I'll ensure all tests pass consistently in the CI pipeline and that the app is stable under test coverage.

Week 12

- **Integrate Continuous Integration (CI) to automate APK builds.**
  I'll set up CI to automatically generate APKs and run tests for each pull request or code push.
- **Improve code analysis using tools like Spotless and Detekt.**
  I'll add static code analysis tools to enforce code quality, detect smells, and maintain consistency.
- **Ensure a smooth deployment process to the Play Store by resolving any CI/CD issues.**
  I'll make sure there are no issues in generating signed builds and that the app is ready for deployment.

Week 13

- **Conduct extensive manual and automated testing to detect and fix any remaining issues.**
  I'll perform in-depth manual testing and use automated tools to catch bugs that UI tests might miss.
- **Optimize API calls, UI rendering, and payment integration performance.**
  I'll look for slow-loading screens or excessive API calls and optimize performance and UI responsiveness.
- **Perform a final security audit, ensuring all authentication and API calls follow best practices.**
  I'll ensure secure handling of user data, encrypted storage, and correct use of authentication tokens.

Week 14 and Rest of Phase 2

- **Write detailed documentation for future contributors, covering project setup, API integrations, CI/CD workflows, and testing strategies.**
  I'll create detailed documentation for future contributors, including setup guides, architecture, and testing strategies.
- **Gather feedback from mentors and the community, making final refinements if needed.**
  I'll demo my work, gather feedback, and make any last-minute refinements to improve the app further.
- **Ensure all necessary approvals and configurations are in place for deploying the app to production.**
  I'll verify signing keys, build configs, and release notes so everything is ready for production deployment.
- **Assist in onboarding future contributors by writing blog posts or conducting a demo session.**

I'll help future contributors by sharing guides, writing blog posts, and possibly recording a walkthrough session.

3.4 Post Phase 2 (After Aug 26)

- **Discuss the project's outcome with mentors**
  I'll review the overall impact of my work, note what can be improved, and plan how to continue contributing after GSoC.
- **Engage with the community**
  I'll continue being active in the Mifos community, reviewing pull requests, and helping other contributors.
- **Identify possible add-on features**
  I'll propose new ideas or additional features that can be taken up as follow-up tasks or community projects.

4. Why am I the right person ?

When I first started contributing to **Mifos**, I was primarily focused on understanding the existing codebase, fixing smaller issues, and learning how different components interacted. Over time, I evolved from making minor fixes to handling **end-to-end feature development**, improving architecture, and working on **core functionalities** that shape the project's future.

When I first started contributing to **Mifos**, I was focused on understanding the project structure, fixing minor issues, and getting familiar with how the different modules worked together. Initially, my role was limited to **small bug fixes, UI improvements, and minor enhancements**.

However, as I kept contributing, I started gaining a **deeper understanding of the project architecture**, backend interactions, and the **core functionalities** of the app. This allowed me to move beyond just fixing issues to actively **developing features, optimizing performance, and improving the overall code quality**.

Over the past **six months**, my contributions to **Mifos** have been significant. Initially, I focused on **bug fixes and minor enhancements**, but as I became more comfortable, I started working on **feature development, architectural improvements, and migrating the app to Kotlin Multiplatform (KMP)**.

I started with **Android development** about a year ago, gaining experience through **personal projects, open-source contributions, and hackathons**. My technical expertise spans across **Jetpack Compose, MVVM, Dependency Injection (Koin/Dagger-Hilt), Kotlin Multiplatform, Firebase, Networking (Ktor, Retrofit), and CI/CD workflows**.

- **Deep Understanding of Mifos Mobile:**
  - I have been actively contributing for over **six months**, completing key assignments, fixing issues, and developing new features. I now have **a strong grasp of the project architecture** and its integration with Mifos's backend services.
- **Expertise in Android & Kotlin Multiplatform:**
  - My expertise in **Jetpack Compose, KMP, Networking, Dependency Injection, and Android best practices** allows me to handle both frontend and backend interactions efficiently.

- **End-to-End Development & Leadership:**
    - I have moved from contributing small features to **leading migration efforts, improving architecture, and integrating complex external services**. I now take a **solution-oriented approach**, ensuring that changes are scalable, maintainable, and improve the overall developer experience.
- **Mentorship & Community Engagement:**
    - Beyond development, I actively mentor new contributors, propose and implement architectural improvements, and collaborate with the community to plan solutions for upcoming challenges.
- **Problem-Solving & Innovation:**
    - I always strive to identify inefficiencies and improve code quality. Whether it's optimizing API calls, improving caching mechanisms, or reducing unnecessary dependencies, my focus is on performance, maintainability, and user experience.
- **With my experience, technical skills, and commitment to Mifos Mobile 7.0, I am confident that I can play a key role in taking the project to the next level.**

5. Current area of study

I am a pre-final year Computer Science and Engineering student at Rajiv Gandhi University of Knowledge Technologies. My expertise lies in Kotlin Multiplatform (KMP) and modern Android development, including Jetpack Compose, Dependency Injection (Koin/Dagger-Hilt), and Networking (Ktor, Retrofit).

Over the few months, I have transitioned from developing Android applications to working extensively with KMP, enabling seamless code sharing across platforms. My contributions to Mifos Mobile include migrating the app from a multi-module Kotlin structure to Kotlin Multiplatform, ensuring compatibility with iOS, and optimizing networking and data layers.

I have an in-depth understanding of:

- **Kotlin Multiplatform (KMP)**: Structuring shared modules, handling platform-specific implementations, and integrating Ktor for networking.
- **Jetpack Compose & KMM UI:** Implementing modern UI patterns with Jetpack Compose
- **Performance Optimization**: Reducing redundant API calls, improving caching mechanisms, and enhancing cross-platform consistency.

With my hands-on experience in end-to-end development, architectural improvements, and API integrations, I am well-equipped to take Mifos Mobile 7.0 forward by leveraging the full potential of Kotlin Multiplatform.

6. Contact Information

Name: Nagarjuna Banda

Email: nagarjunabanda4@gmail.com

Slack Account: Arjun

LinkedIn: https://www.linkedin.com/in/nagarjuna3/

GitHub: https://github.com/Nagarjuna0033

Mobile Number: +91 9885930886

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

7. Career Goals

As an Android developer, my journey began with mastering Android SDK, Jetpack Compose, multithreading, and modern architectural patterns to build scalable and high-performance applications. Over time, I gained expertise in networking, dependency injection, and Jetpack libraries, enabling me to create robust Android apps that adhere to best practices.

As mobile development evolved, I recognized the need for **cross-platform** solutions that maintain high performance while reducing code duplication. This led me to **Kotlin Multiplatform (KMP),** a powerful technology that allows sharing business logic across platforms while ensuring a native experience for each. By leveraging **KMP**, I can create efficient, maintainable, and scalable applications that seamlessly run on Android, iOS, and beyond.

My focus is on writing platform-agnostic business logic while ensuring smooth native UI and performance optimizations tailored to each platform's requirements. By continuously exploring KMP's evolving ecosystem, I aim to stay at the forefront of modern architectures, concurrency models, and tooling improvements to build next-generation mobile applications.

I strongly believe in knowledge-sharing and mentorship. Through open-source contributions, blog posts, and developer talks, I aspire to help more developers transition into KMP, making cross-platform development more accessible and efficient. With my deep passion for Android, KMP, and open-source, I am committed to becoming a leader in cross-platform development and driving innovation in the mobile ecosystem.

8. My Projects

While I haven't worked on standalone Android projects independently, my journey in Android development has been driven through open-source contributions. Instead of building small projects, I directly started contributing to large-scale production applications like **Mifos Mobile**. Through this, I have gained **practical experience in Kotlin, Jetpack Compose, Android Architecture Components, and Kotlin Multiplatform (KMP) and Compose Multiplatform (CMP)**.
So far, I have successfully **merged 30+ PRs** in open-source repositories, where I have worked on **feature implementations, bug fixes, API integrations, modularizing the codebase, optimizing performance**. This experience has given me exposure to real-world development challenges, team collaboration, and best coding practices used in professional Android applications.

Additionally, I have a background in **web development**, where I have worked on various projects. This experience has helped me understand software architecture, backend interactions, and scalable design, which I have effectively translated into Android development. My ability to **adapt, learn, and contribute to large open-source projects** without prior standalone projects demonstrates my strong problem-solving skills and ability to work in real-world scenarios.
In summary, while I may not have built an independent Android project, my experience in **open-source development has been equivalent (or even more challenging)** than working on personal projects. I am confident that my experience working on production-level Android applications has prepared me well for this role.

Here are some of my notable contributions at the time of submitting this proposal:

- **Merged Pull Requests** for Mobile Wallet
    1. PR #1764: Migrated Dependency injection from Hilt to Koin
    2. PR #1773: Refactoring Payment screen
    3. PR #1777: Refactoring Finance Screen
    4. PR #1789: Refactoring Both light Theme and Dark Theme
    5. PR #1797: Fixing Invoice API and refactoring Invoice Screen

- **Merged Pull Requests** for Mifos Mobile
    1. PR #2728: Migrated DB Flow to Room
    2. PR #2735: Migrated core/logs module to KMP
    3. PR #2739: Migrated core/network module to KMP
    4. PR #2747: Migrated core/data module to KMP
    5. PR #2751: Migrated core/QrCode module to KMP
    6. PR #2760: Setting up modules for different platforms (Android, Desktop, Web)
    7. PR #2772: Migrated feature/auth to CMP
    8. PR #2785: Migrated feature/home to CMP
    9. PR #2499: Update readme with current version of the project

- **At the time of writing this proposal I have 30+ merged PRs and I am still contributing. You can find all those PRs here. While I am writing this proposal and I am working on fixing bugs and issues. Created some tickets in jira to work on issues, bugs, and refactoring the code.**

9. Slack Channel

Yes, I have visited all of mifos's Slack channels and my id is Arjun

10. Interaction with mentor

Yes, I had an interaction with mentor Mr. Rajan Maurya sir

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

Yes, I do have experience with Android and kotlin and have built projects centered around them. I have decent knowledge of MongoDb and SQLite Databases.I have built a full stack application during the course for my university club by using React as the frontend and Node js for developing RESTful APIs

12. Other Commitments

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

13. Deployed and tested the mobile Applications

Yes, as a part of mobile application developer I had a chance to deploy to firebase distributions and testing the mifos mobile, mifos pay, and mifos x droid applications.

14. Contributions to Mifos

Since the conclusion of GSoC 2024 (September), I have continued contributing to Mifos, expanding my efforts beyond the **Mifos Mobile** to include **Mobile Wallet**

- **Project Contributions:**
  - **Merged Pull Requests** for Mobile Wallet
    1. PR #1764: Migrated Dependency injection from Hilt to Koin
    2. PR #1773: Refactoring Payment screen
    3. PR #1777: Refactoring Finance Screen
    4. PR #1789: Refactoring Both light Theme and Dark Theme
    5. PR #1797: Fixing Invoice API and refactoring Invoice Screen

  - **Merged Pull Requests** for Mifos Mobile

    1. PR #2718: Moved .kotlin folder to gitignore
    2. PR #2720: fix: notifications rendering issue

3 PR #2728: feat(notifications, clientCharges): DB flow to room database
4 PR #2733: fix: API endpoint
5 PR #2735: feat(:core:logs): migrated to kmp
6 PR #2739: feat(:core:network): migrated to kmp
7 PR #2747: feat(:core:data): migrated to kmp
8 PR #2751: feat(:coreQrCode): migrated to kmp
9 PR #2760: feat(application modules): Setting up modules for different platforms (Android, Desktop, Web)
10 PR #2772: feat(:feature:auth): migrated to cmp
11 PR #2785: feat(:feature:home): mgrated to cmp
12 PR #2786: feat(:feature:loan): migrated to cmp
13 PR #2787: fix: loan charge display
14 PR #2797: fix: BasApi URL
15 PR #2796: refactor: username length validation from 5 to 4
16 PR #2799: feat(:feature:client-charge): migrated to cmp
17 PR #2801: feat(;feature:tpt): migrated to cmp
18 PR #2803: feat(:feature:transfer-process): migrated to cmp
19 PR #2807: feat(:feature:beneficiary): migrated to cmp
20 PR #2809: feat(:feature:qr): migrated to cmp
21 PR #2812: feat: qrcode generation in all platforms
22 PR #2815: feat(:feature:user-profile): migrated to cmp
23 PR #2819: refactor: removed explicitly specified colors
24 PR #2821: fix: notification rendering issue
25 PR #2822: fix: QR code displaying and savings charges navigation
26 PR #2824: refactor: moved intents into core module
27 PR #2828: fix: user registration

- **At the time of writing this proposal I have 30+ merged PRs and I am still contributing. You can find all those PRs here**

  - **Open Pull Requests**
    - At the time of drafting this proposal I am working on Mifos Mobile 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

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 year GSoC concluded, I have been actively connected to Mifos

15. 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 (GSoC)** program is a privilege. Beyond the impact, Mifos has a **vibrant community of developers, volunteers, and contributors** who are passionate about building open-source financial solutions. Being surrounded by such a dedicated community is a strong motivator, pushing me to continuously improve and contribute in meaningful ways.

From a professional perspective, **GSoC is an incredible opportunity** to work on production-level open-source projects, collaborate with experienced mentors, and gain hands-on experience in real-world software development. Mifos's participation in this program reflects its commitment to nurturing the next generation of developers like me, empowering us to work on projects that **impact millions globally**. The knowledge that my contributions can enhance financial accessibility and inclusion is incredibly rewarding.

Over the past year, my journey in **Kotlin Multiplatform (KMP) and Compose Multiplatform (CMP)** has been transformative. Initially, I started with native **Android development**, focusing on Jetpack Compose, MVVM, and Dependency Injection. However, as I explored the world of **cross-platform development**, I was drawn to **KMP's ability to share business logic across platforms** while still maintaining native experiences. This shift was a game-changer, allowing me to contribute to large-scale open-source projects like Mifos Mobile while learning to build scalable, modular architectures.

Working with **KMP and CMP** has not only deepened my understanding of **multi-platform development** but also made me more efficient in writing reusable, maintainable code. The ability to **bridge the gap between Android, iOS, and even web applications** excites me, and I am eager to continue learning, experimenting, and mastering these technologies. Through my journey, I have realized that technology is constantly evolving, and my love for learning keeps me motivated to stay ahead.

Being part of **Mifos Mobile 7.0**, helping migrate key modules to **KMP**, and working alongside experienced developers has solidified my passion for open source and Kotlin Multiplatform. I look forward to not only enhancing my skills further but also **mentoring new contributors**, refining architectures, and continuing my journey towards **becoming a leader in KMP development**.

16. Previous Participation in GSoC

No, I didn't participate in Google Summer of Code.

17. Application to multiple orgs

I will be applying only to Mifos Mobile for this year's GSoC