Instagram clone clean architecture

Flow of the app

Activity--->ViewModel--->Use Case--->Repository--->Datasource

SignUpActivity--->SignUpViewModel--->SignUpUseCase--->SignUpRepository--->SignUpDataSource(implementing interface ISignUpDataSource)

SignInActivity--->SignInViewModel--->SignInUseCase--->SignInRepository--->SignInDataSource(implementing interface ISignInDataSource)

Flow for SignUp and Signin→

1.From the **SignUpActivity** call the **SignUpViewModel** method *firebaseSignUp()* and provide a *user class object* as an argument.

Note-: We provide a user class object as our parameter and not the username and password directly because the method signature of our SignUpUseCase perform() function accepts the single argument ExecutableParam.

2.In our **SignUpViewModel** we will inject the **SignUpUseCase** through the constructor and call the *perform()* function passing the user object to it.

3.The **UseCaseSignUp** will contain the Firebase repository **FirebaseRepo** as a dependency and we will call function *firebaseSignUp()* and pass the arguments username and password, which we got from our **SignUpUseCase** user object and return the result FIrebaseUser.

4.The FirebaseRepo will get the final result/error(filtered using **SafeResult**) from our DataSource namely **FirebaseSignUpDataSource**.
5. Inside the **FirebaseSignUpDataSource**, we will return **SafeResult(firebaseUser)** if the sign up is successful otherwise we will return **SafeResult(null).**

6.Using this “saferesult” type response we get from the *FirebaseSignUpDataSource-->FirebaseRepo---> FirebaseUseCase*, we will make make a class to store the state of response(**SignUpViewState**) inside **SignUpVM** class.

6.Based on the state of response we store inside the viewmodel class, now we will perform appropriate actions and changes to our UI.

Same flow for LoginActivity

Dependencies

Our AppComponent is the interface will which contain all the modules-:

1. **AppModule** module will contain dependencies which will be used throughout the application. For example notificationmanager,activity context, application context.We can differentiate between 2 methods returning the same type by using qualifiers.For example activity scope and activity scope can be differentiated using
2. **ActivityModule** module contains the dependencies which are used to provide dependencies to activities. Part of dagger android
3. **ViewModelFactoryModule** provides a collection of viewmodels as a map(key value pair) and each viewmodel is injected in the viewmodelFactory using this using key. @Intomap annotation is used to inject each viewmodel. We use @Mapkey in viewmodel scope annotation to create this collection of viewmodels as a key value pair.
4. **NetworkModule** module contains the dependencies required by our Sign up and Login Data Source. It will be injected using a constructor.
5. **UsecaseModule** module provides usecase to our viewmodel
6. **RepositoryModule** module provides the repositories to our UseCases.
7. **DataSourceModule** module provides the datasources to repositories

Flowchart of dependencies