CSCI 5448 Object Oriented Analysis and Design
Project 7 report

Name of project and names of all team members

SimplifyDebt

1. Rakshith Jayanna
2. Rakesh Chowdary Yarlagadda

Final State of System Statement

A paragraph on the final state of your system: what features were implemented, what features were not and why, what changed from Project 5 and 6

We have implemented the following features

- Login and Signup feature (Users can use email or google sign in)
- Dashboard feature (Users can view their friends , groups and activities)
- Creating Groups ,Viewing group expenses and status
- Viewing friend expenses and status
- Adding expenses
- Settling the expenses

We were not able to implement the following features/functionalities

- We were not able to implement invite a friend (qr code or deep link) due to increased complexity
- We were not able to add bill images (Were unable to setup a server to map, store and fetch images)
- We did not implement scan bill and fill an expense feature

While our end goal was high during Project 5 and 6, we underestimated the efforts required to learn and experiment unfamiliar technologies (firebase, kotlin, android, S3 bucket server[could not integrate]).

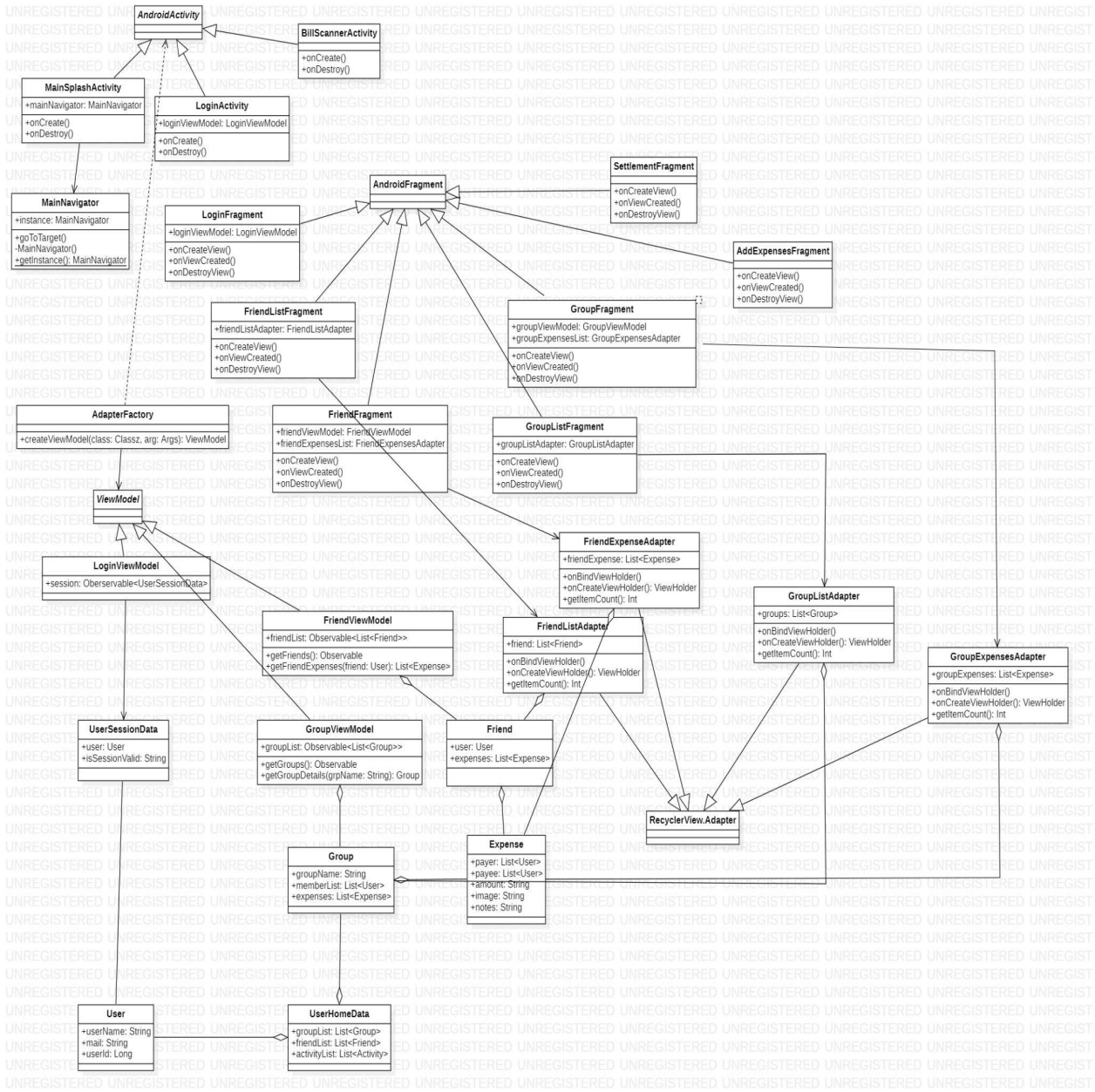
Final Class Diagram and Comparison Statement

• A thorough UML class diagram representing your final set of classes and key relationships of the system

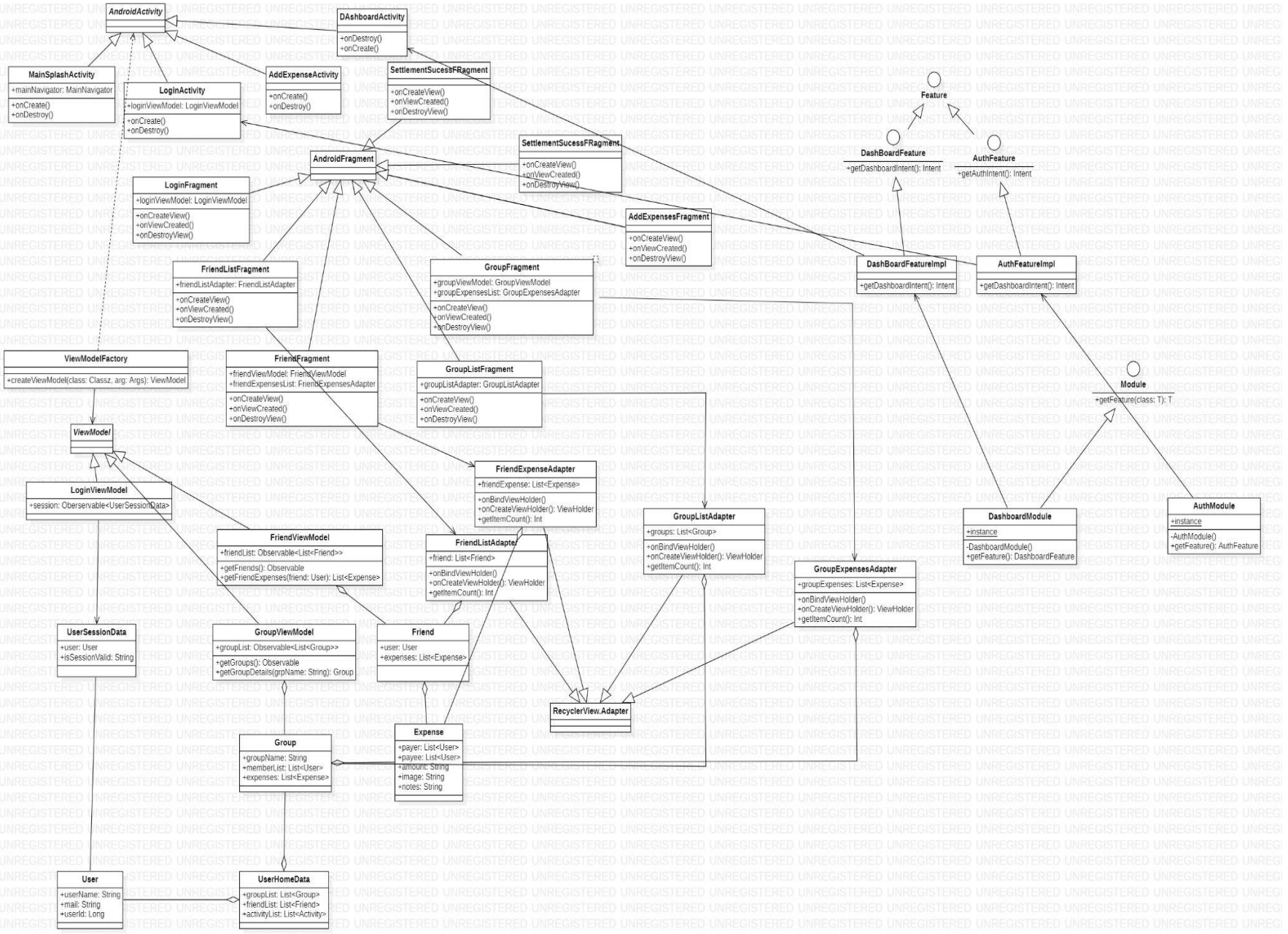
• Highlight and document in that diagram any patterns that were included (in whole or part) in your design

• Include the class diagram submitted in Project 5, and use it to show what changed in your system from that point into the final submission

• Support the diagrams with a written paragraph identifying key changes in your system since your design/work was submitted in Projects 5 and 6



Our UML for Project 5, Please find our update uml for project 7 in the following page


AndroidActivity
DAshboardActivity
+onDestroy()
+onCreate()
MainSplashActivity
+mainNavigator: MainNavigator
+onCreate()
+onDestroy()
LoginActivity
-loginViewModel: LoginViewModel
+onCreate()
+onDestroy()
AddExpenseActivity
+onCreate()
+onDestroy()
SettlementSucessFRagment
+onCreateView()
+onViewCreated()
+onDestroyView()
Feature
DashBoardFeature
+getDashboardIntent(): Intent
AuthFeature
+getAuthIntent(): Intent
AndroidFragment
SettlementSucessFRagment
+onCreateView()
+onViewCreated()
+onDestroyView()
LoginFragment
-loginViewModel: LoginViewModel
+onCreateView()
+onViewCreated()
+onDestroyView()
AddExpensesFragment
+onCreateView()
+onViewCreated()
+onDestroyView()
FriendListFragment
+friendListAdapter: FriendListAdapter
+onCreateView()
+onViewCreated()
+onDestroyView()
GroupFragment
+groupViewModel: GroupViewModel
+groupExpensesList: GroupExpensesAdapter
+onCreateView()
+onViewCreated()
+onDestroyView()
DashBoardFeatureImpl
+getDashboardIntent(): Intent
AuthFeatureImpl
+getDashboardIntent(): Intent
ViewModelFactory
+createViewModel(class: Classz, arg: Args): ViewModel
FriendFragment
+friendViewModel: FriendViewModel
+friendExpensesList: FriendExpensesAdapter
+onCreateView()
+onViewCreated()
+onDestroyView()
GroupListFragment
+groupListAdapter: GroupListAdapter
+onCreateView()
+onViewCreated()
+onDestroyView()
Module
+getFeature(class: T): T
ViewModel
FriendExpenseAdapter
+friendExpense: List<Expense>
+onBindViewHolder()
+onCreateViewHolder(): ViewHolder
+getItemCount(): Int
LoginViewModel
+session: Oberservable<UserSessionData>
FriendViewModel
+friendList: Observable<List<Friend>>
+getFriends(): Observable
+getFriendExpenses(friend: User): List<Expense>
FriendListAdapter
+friend: List<Friend>
+onBindViewHolder()
+onCreateViewHolder(): ViewHolder
+getItemCount(): Int
GroupListAdapter
+groups: List<Group>
+onBindViewHolder()
+onCreateViewHolder(): ViewHolder
+getItemCount(): Int
DashboardModule
+instance
-DashboardModule()
+getFeature(): DashboardFeature
AuthModule
+instance
-AuthModule()
+getFeature(): AuthFeature
GroupExpensesAdapter
+groupExpenses: List<Expense>
+onBindViewHolder()
+onCreateViewHolder(): ViewHolder
+getItemCount(): Int
UserSessionData
+user: User
+isSessionValid: String
GroupViewModel
+groupList: Observable<List<Group>>
+getGroups(): Observable
+getGroupDetails(grpName: String): Group
Friend
+user: User
+expenses: List<Expense>
RecyclerView.Adapter
Group
+groupName: String
+memberList: List<User>
+expenses: List<Expense>
Expense
+payer: List<User>
+payee: List<User>
+amount: String
+image: String
+notes: String
User
+userName: String
+mail: String
+userId: Long
UserHomeData
+groupList: List<Group>
+friendList: List<Friend>
+activityList: List<Activity>

Updates since project 5,

- For project 6 we had completed login feature and dashboard feature.
- Due to our detailed planning and well designed UI mockup, we were able to finalize our data models and component classes. Hence there are not many updates in these classes.
- Due to increasing complexity we dropped BillScanner feature. Hence removed that activity class
- We added more activities to make our in app navigation easier and group our feature flows. We added AppExpense and Dashboard as separate activities
- After project 6 our biggest change was changing our project and dependency structure to be more modular This did not change our classes but significantly changed the project structure and introduced feature and module interfaces.
- After project 6 we added AddExpense classes, Settle expense classes to complete our minimum viable features

Patterns Used,

- **Singleton pattern:** All our feature modules are singleton classes. They are eagerly initialised once when the app starts and their purpose is to initialise and provide feature implementations for each feature
- **Factory pattern:** Used to initialise a viewmodel instance. Android activities have Default viewmodel providers. View Model providers have a default factory implementation. We are overriding that to create a new viewmodel instance with parameters
- **Observable pattern:** Livedata in android are observables that respect component lifecycle. Viewmodels hold their reference and we use them for navigation events and network calls. They are observed in activities/fragments
- **Adapter pattern:** To display a list of row objects, A List View requires view bindings and view holders. Adapter wraps/converts these objects in/to view holders that is required by list view

Third-Party code vs. Original code Statement
A clear statement of what code in the project is original vs. what code you used from other sources – whether tools, frameworks, tutorials, or examples – this section must be present even if you used NO third-party code - include the sources (URLs) for your third-party elements

Our codebase is developed from scratch on Android studio(We haven't used a starter or code solutions online)
We have used the following resources as our reading/tutorial materials.

- Android and Kotlin documentation - https://developer.android.com/
- Sign in service for Android - https://developers.google.com/identity/sign-in/android/sign-in

- Firebase FireStore integration for Android - https://cloud.google.com/firestore/docs/manage-data/add-data#kotlin+ktxandroid
- https://developers.google.com/identity/one-tap/android/get-saved-credentials#kotlin
- https://sudishkumar-edugaon.medium.com/how-to-retrieve-data-from-firestore-with-kotlin-3e00bb2f9900
- App modularisation strategies, all 3 parts - https://proandroiddev.com/modularization-of-android-applications-in-2021-a79a590d5e5b

Statement on the OOAD process for your overall Semester Project
List three key design process elements or issues (positive or negative) that your team experienced in your analysis and design of the OO semester project

- One thing that helped us a lot was the design discussions while drawing the uml, All the possibilities were discussed, which made the code development faster
- Given that our project was done from scratch, we needed more time and effort to setup base UI components. With our limited experience in the technologies, We did a bad job in estimating timelines and committed to complex deliverables
- For project 5, we created a good UX mockup with figma. This significantly made our development faster as we considered this mockup as final UI requirement.
- We have attempted to employ a good modular design. This helped us to achieve good feature abstraction, easier communication between modules, minimal coupling within modules and avoid circular dependencies
- We have designed each activity flow in an MVVM fashion. This pattern provides a good separation of logical code in viewmodels from UI code in views.