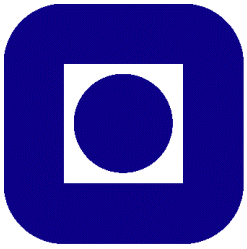


NTNU
Norwegian University of Science and Technology

Roomies - PROG2007 Project

Group 14:

Viktor Hans Didrik Chambe-Eng

Harald Andreas Løkkeberg Hansen

Omotayo Farouk Lawal

Mihai Alin Napirca

November 25, 2022

Project repo: https://git.gvk.idi.ntnu.no/hahansen/prog2007-prosjekt-roomies

1. Project summary

We wanted to create a social mobile application for roommates who want to manage their communal lives. The app would let roommates create a group where they can organize and communicate about important matters. This includes making and following schedules in a calendar, managing household expenditures and fees in an accountancy tab, alerting each other of issues such as a broken lightbulb, and chatting to be able to quickly communicate and keep each other up to date on anything that might be relevant. The data of each group should be sent to a database and synchronized in the app, so the information is always up to date for the end user. The application is made in Android Studio, written in Kotlin, and uses Firebase as its database.

2. Introduction

2.1 Project motivation, aim and objectives

As students, most of us have have been through the struggles of moving away from home to pursue our studies. Due to the lack of experience of living alone away from one's parents, it might be hard for some to create a structured daily routine with their new roommates. For this reason we wanted to create an application where people living together could create a group to communicate and structure household chores and issues. This scenario is not just limited to students, but to any group of people sharing a dormitory, apartment or house looking for a great working app to structure their lives together. We hope this can help people become more organized and gain a sense of community.

2.2 Scope

This is not an app for finding new roommates. Rather, it is for organizing and communicating between existing roommates to help improve the structure of their daily life. The target audience is people who are new roommates and may not know each other well, and want a tool to help get organized with communal chores and finance. This applies especially to students, who are often disorganized and without a source of income. The app is also limited to just android devices.

2.3 Application areas

The application requires the user to create an account, with their email address for contacting purposes, before they can join or create a group. Joining an existing group requires a code shared by an existing member of the group, similar to how Discord or Microsoft Teams operates. Once the user has joined a group, they have access to the group hub page. Here, each user is able to access the four main functionalities. These are the calendar, accounting, issues, and chat. The calendar has a month and week view, where the user can add events and schedules, as well as see entries made by other group members. Accounting allows the user to register when they pay for

something everyone uses, and see a total of how much they have contributed. In issues, the user can report important issues for the whole group to see, like a leaky tap, and mark them as resolved. In the chat, everyone in the group can chat with each other about anything they want, whether it be a serious discussion or planning a fun event.

2.4 Your contribution

Alin has worked on the implementation of:

- Calendar view and events with Omotayo
- The accounting tab for users to register and store items/services for a fee
- Report documentation work

Alin	Coding	Documentation	Research
Sprint 1		2	1.5
Sprint 2	10		3.5
Sprint 3	5	2	3
Sprint 4	7	4	
Sprint 5		7	
SUM:	45		

Omotayo has worked on the implementation of:

- Calendar view and events along with Alin
- Issue functionality
- Menu bar and design of the app
- Report documentation work

Omotayo	Coding	Documentation	Research
Sprint 1		1	1
Sprint 2	10		3.5
Sprint 3	5	2	
Sprint 4	4	5	
Sprint 5	3	6	
SUM:	40.5		

Harald has worked on the implementation of:

- User accounts like login, signup, edit account information, and authentication in general
- Groups: Create/join group, displaying groups the user is a member of, and displaying members with contact details for a selected group

- Report documentation work

Harald	Coding	Documentation	Research
Sprint 1	2		2
Sprint 2	5.5		
Sprint 3	3.5		
Sprint 4	10.75		
Sprint 5	10.25	7.25	
SUM:	41.25		

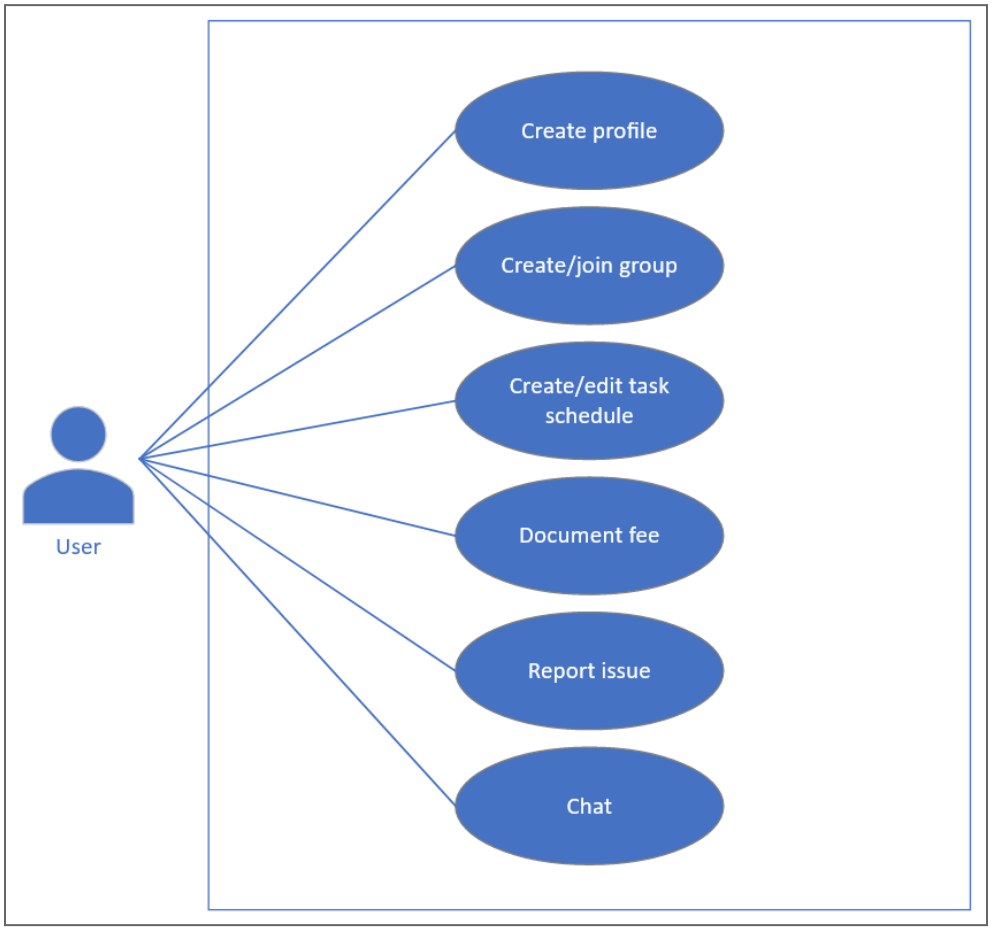
Viktor has worked on:

- Implementing the chat and instant updates in Firestore
- Documentation and diagrams

Viktor	Coding	Documentation	Research
Sprint 1		4	
Sprint 2	6		
Sprint 3	6		
Sprint 4	8		
Sprint 5	8	11	
SUM:	43		

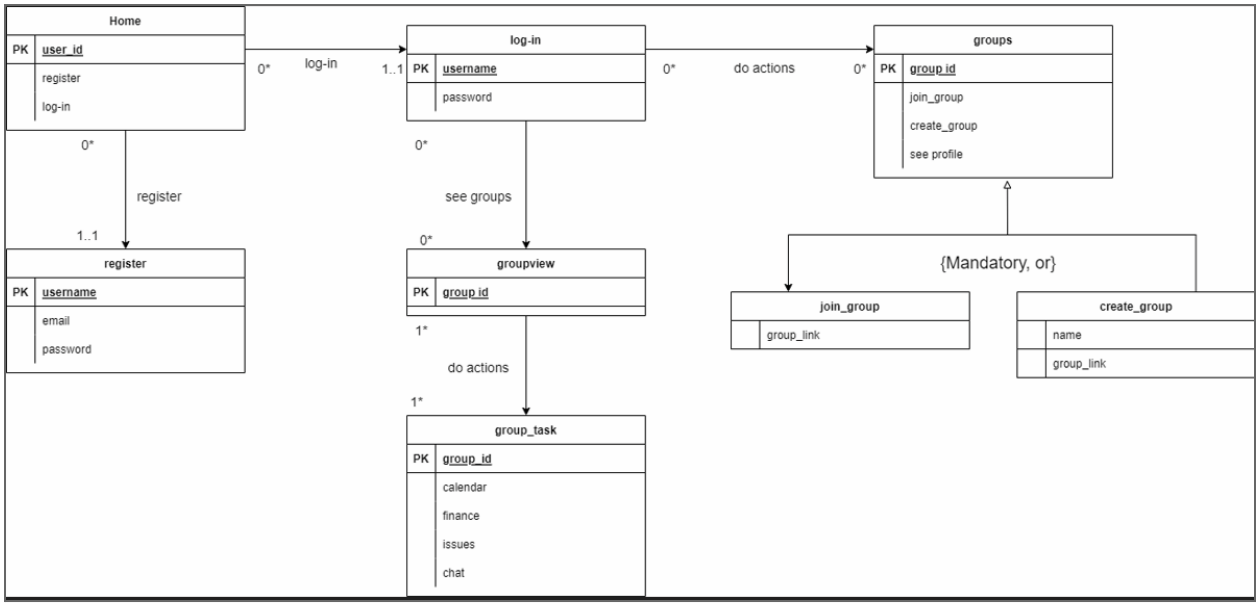
3. Implementation

3.1 Use case diagram



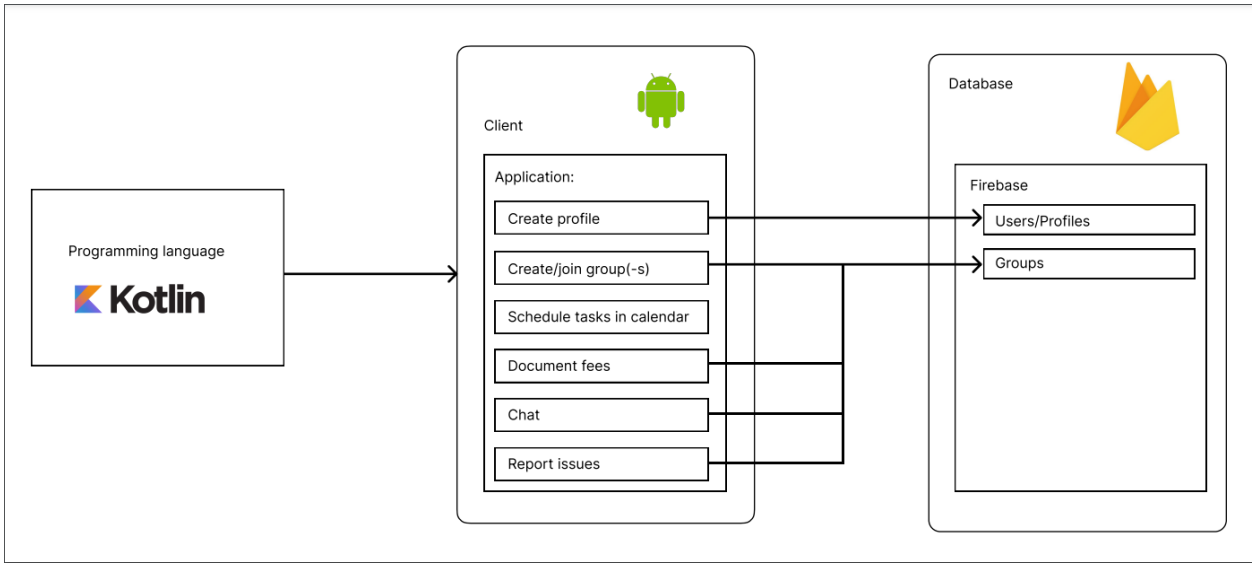
Since all group members have access to the same functionality, and a login is required to access all functionality, there is only one type of user. A user can create a profile and log in, create or join a group, and access the four basic functionalities of each group they are in.

3.2 ER diagram



This is our ER diagram. This diagram is meant to show us the relationship between the users, objects and events within our application. Most of the relation types are many to many except login or register. This is because in our application there can be multiple different users present and when a change is done by a user, people in the same group can see them too.

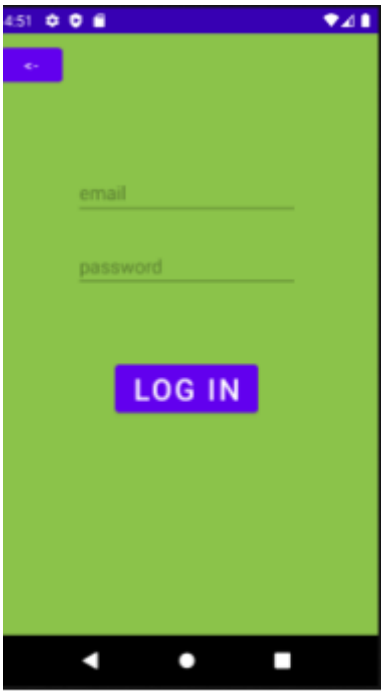
3.3 Overall system architecture



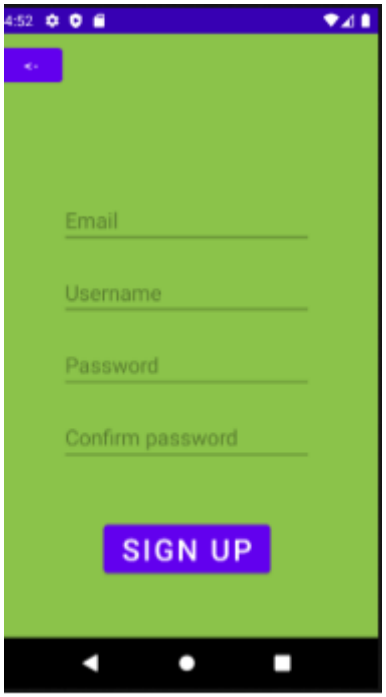
Our system architecture. Since the subject is being taught in Kotlin and Android Studio we chose that as our foundation. For the database, we used Firebase Authentication for users and Firebase's Firestore Cloud Database for our database system. Since Firebase is easy to implement into Android Studio and Kotlin, and we learnt about it in the labs, we could spend more time on our application instead of building a REST API for our database.

3.4 Explanation of the activities

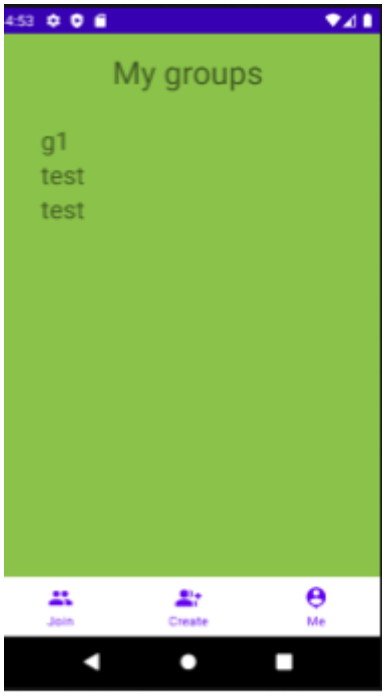
MainActivity



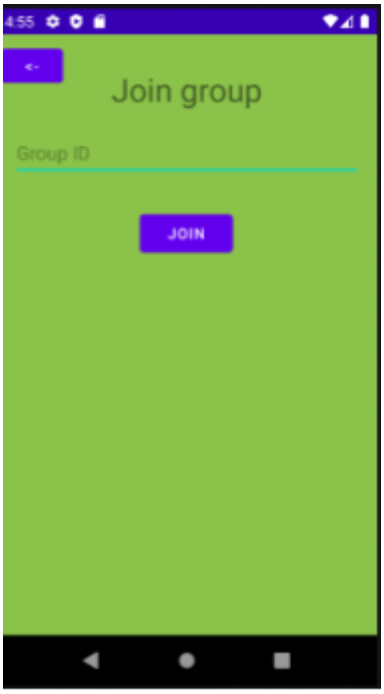
LogIn



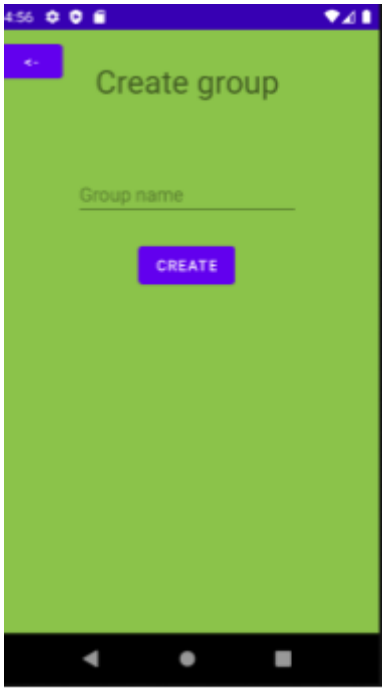
SignUp



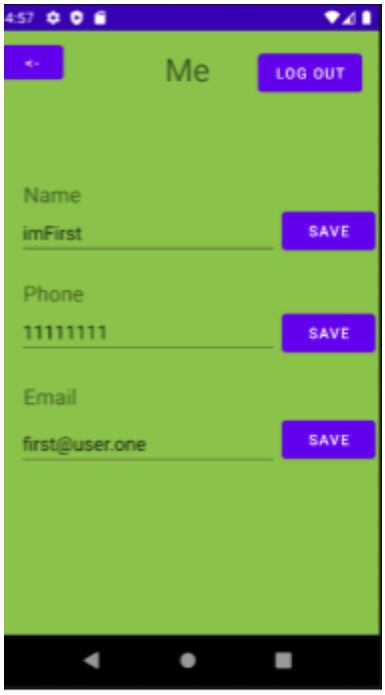
MyGroups



JoinGroup



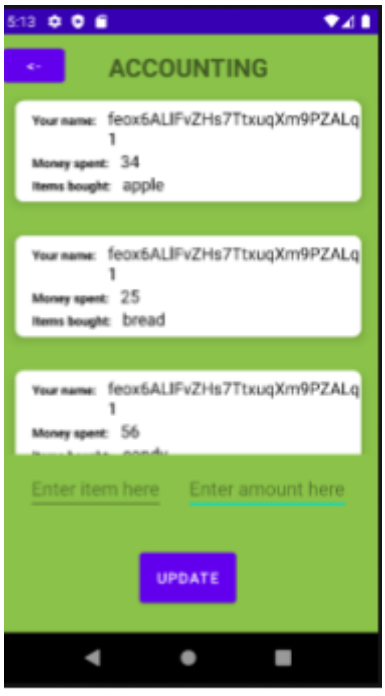
CreateGroup



MeInfo

GroupInfo

CalendarMain



AccountingMain

Issue

ChatMain

MainActivity: This is the activity the user will see when the app starts up. If the user is authenticated, they will immediately be redirected to MyGroups. If not, they can navigate to LogIn and SignUp with the respective buttons, where the user can log in or sign up. If a user is

not authenticated while being in an activity that requires them to be authenticated, they will be sent to this activity.

LogIn: This activity is used to authenticate unauthenticated users. Authentication is achieved by providing email and password, and then pressing the login button. The user will be sent to MyGroups on login success. The user can also navigate back to MainActivity with the back button in the top left corner.

SignUp: This activity is used to sign up users with no account. By entering email, username, password and password confirmation and clicking the signup button, an account will be created and the user will be authenticated and sent to MyGroups. The user can also navigate back to MainActivity with the back button in the top left corner.

MyGroups: This is in a way the main activity for authenticated users. This activity contains a list of the groups the current user is a member of, a button that takes the user to JoinGroup where the user can join a new group, a button that takes the user to CreateGroup where the user can create a new group, and a button that takes the user to MeInfo where the user can see and edit information about their account that other users can see. If the user clicks on a group in the group list, they will be taken to the GroupInfo activity for the group they clicked on.

JoinGroup: This activity lets the user join existing groups by entering a group's group link and clicking the join button. The user can also navigate back to MyGroups with the back button in the top left corner.

CreateGroup: This activity lets the user create a new group with the current user as the owner. This is done by entering a name for the group by the user's choice and clicking the create button. The user can also navigate back to MyGroups with the back button in the top left corner.

MeInfo: This activity lets the user view their user information which will be shown to the other members of the groups the current user is a member of. This information includes username, phone number and email. The user can also change this information by editing one of them and clicking the corresponding save button to the right. To navigate back to MyGroups, the user can use the back button in the top left corner. Finally, the user can log out by clicking the logout button in the top right corner.

GroupInfo: This is in a way the main activity for a specific group. The different functionalities for a group are accessed through this activity. First there is a list of all the members of the group. This list displays their username with their contact details, phone and email. At the bottom of the screen, there are four buttons that take the user to CalendarMain, AccountingMain, IssueMain,

and ChatMain. These activities are described below. The user can again navigate back to MyGroups with the back button in the top left corner.

CalendarMain: The calendar activity allows the user to view the current month / week and register events tied to a date. Additionally it allows the user to cycle forwards and backwards in the calendar, previewing previous and future months, or weeks depending on the current view. In the month view the user is unable to add events, so one must switch to the week view in order to do so. Also, it is only in the week view where the events are displayed. An event can be added by anyone within a group, however the calendar lacks database sync as of now so no data is stored and saved so that the rest of the group can see.

AccountingMain: Within the accounting tab a user is able to register expenditures, whether it be group related or personal use, that is up to the user. However, if a user is part of multiple groups he will have a separate accounting tab for each group he is a part of. An accounting issue can only be saved when both the “item” and the “price” is defined. That data is then stored within our Firestore database and fetched so the user can see past accounting events. Additionally, the data is updated in the recycler view everytime a new item is added to the accounting tab up to date with the database.

IssueMain: In this activity the user can write up different issues for all group members to see. The data is stored in Firestore with an id to identify the user who sent it, the issue that should be addressed and a Checkbox to check when an issue is completed. The user must write an issue to be able to push the data to Firestore. With the help of Firestore, each time a member of the same group goes to check the issues they can see all previous issues and see if they are completed or not. This activity also uses an adapter class and recycler view and are updated each time a new issue is added.

ChatMain: This activity lets the user send messages to all group members. When the group chat is accessed for the first time, a message is created letting the users know the group chat was created in the database. After that, any message sent is added to an array in the database. Anyone running the app and in the chat activity will have a listener that updates the chat display whenever a new message is sent, so all messages are always visible. Each message displays the name of the user who sent it, the time it was sent, and the content of the message. The id of the user is also sent to the database, in order to check which messages were sent by you. These are displayed differently, with a cyan color and shifted to the right, rather than the left.

3.5 Implementation details

Authentication:
The most secure way to do authentication would probably be to make our own REST API which the app has to send a request to every time the app wants to update the database. But, since this is

a mobile programming course, we decided to use Firebase's email/password authentication for users of our app. This allowed us to code everything in the app. We have implemented authentication checks in every activity which on activity start checks if a user is currently logged in or not. If the activity requires the user to be logged in while they aren't, they will be redirected to the main activity. Firebase.auth.currentUser is an object that includes data like email, username and UID. We used the UID to connect users to groups for example.

https://firebase.google.com/docs/auth/android/start
https://firebase.google.com/docs/auth/android/manage-users
https://firebase.google.com/docs/auth/android/password-auth

Database:
We used Firestore for our database. Like with authentication, database access is done directly from the app. Again, this is not the most secure way to do it, but it allowed us to focus more on the app itself. Firestore is a document database with collections of documents, but we partly ended up using it as a relational database. For example: When displaying the contact information of the members of a group in GroupInfo, the document in the "groups" collection has a list of member UIDs which is used to query for the users contact information in "userInfo". An example of using the database more like a document database is the chat. In the "chat" collection, each document represents a group, and the content of the document is an array of chat objects.

https://firebase.google.com/docs/firestore/manage-data/add-data
https://firebase.google.com/docs/firestore/query-data/get-data
https://firebase.google.com/docs/firestore/query-data/queries

Various adapters:
Multiple adapters were used in the creation of the view and functionality of our app, these are:

AccountingAdapter:
CalenderAdapter:
ChatAdapter:
EventAdapter:
GroupListAdapter:
MemberInfoListAdapter

As adapter objects are bridges for underlying data for a view, we could have made some of the code into fragments we could have reused, however the adapters are different enough for the majority of the data they display that we got away with creating one for each purpose.

4. Discussion

4.1 Learning outcome and achievements

This project proved to be a healthy experience to further improve our mobile programming. We mainly learned how to use Kotlin, Firebase, and Android Studios in conjunction. We learned how to efficiently pair collections, display, update and delete data, but also about the limitations of Firestore as a collection based database in regards to our project. We managed to create a working calendar, although it had no database sync. We also managed to create a live chat for each group between its group members which turned out well. However, the thing we are most proud of is our group system, which heavily resembles that of Discord or Microsoft Teams. We create a randomly generated string as an invite for each group, and whoever has that string can join said group. This group system glues together all other functionality. Overall we had a great learning experience and we worked hard to create a good application.

4.2 How you've worked as a team

We decided from the start we would use an agile development method. We quickly made a project on Jira to manage roadmaps, backlogs, and issue boards for the sprints. We decided on weekly sprints, as we only had around five weeks to make the project and wanted frequent meetings to manage the little time we had. We also made a discord server to communicate and hold meetings. We used a git repo, and almost always made new branches when making changes, as we often worked independently on our assigned tasks. We also sometimes worked together over discord to handle merges or help each other with tricky bugs.

Overall, the agile method worked well for our projects, as we implemented most of the functionality we wanted, and everyone contributed with both code and documentation. However, we did much more work in the last two sprints than the others. Though this is partly because we had to prioritize other subjects, and without weekly meetings we might have gotten even less work done in the early weeks.

4.3 Challenges and issues

Our project proposal was ambitious. We wanted to implement a lot of different functionalities, but we quickly understood that it would not all be possible with the time we had. Firstly, none of us had much experience working with Firebase as a database service and it proved time consuming to learn. Firestore is also a collection based database which makes it harder to update and display data across tables in contrast to a relationship based database such as SQL, which we also have more experience with. The biggest challenge we had was displaying all group members at all times, as whenever we made a change to one member within a group we made changes to every table that member was situated in. We decided to decrease the number of updates across collections by limiting what each user could see and change without sacrificing

too much of the app functionality. Nonetheless, despite these compromises the application still turned out well, and were happy with the result.

4.4 Incomplete features

During the development process we saw that our project proposal and our final product would not fully align. There were some issues along the way that made the final product a bit different from how we first envisioned the application, so we made some appropriate changes:

Incomplete:

- The calendar has all the features implemented, but it lacks database synchronization. At this point all the users are able to store events in the calendar, however, only the user that created the event can see the event in their calendar and it is not saved upon exiting and re-opening the application.
- Our original idea for the accounting tab was for all the users to see what the accounting sum of themselves and the other users, however, as we began working we saw that the since Firestore is a collection based database it was hard to make the sync across members and groups work due to how we set up our data. Because of this, we ended up restructuring the accounting tab to be limited to each user where he/she can keep an accounting tab on themselves.

Not implemented:

- The ability for a user to delete their account.
- The ability for a user to leave a group. We envisioned that a user would just be a member of one group at a time, but that they could “live” in two places when they are moving. Because of this, we did not set any limit to how many groups one can be a member of. When a member has moved out, it is natural for them to leave the group in the app, but we did not have enough time to implement this.
- In the “groups” collection in the database, there is a field called “owner”. This is the user that created the group. Owner has no meaning outside of that. We were planning to implement ways of the owner to administrate the group with the following features:
    - Kick members from the group if necessary.
    - Edit group info similarly to how it is done in MeInfo.
    - Change the group link if some unwanted people get a hold of it.
- Color coded users: We saw this as the method to differentiate between users, however due to the lack of time, we decided to not implement this as it was not crucial to the application overall.

5. Conclusion

In the end, we were able to implement most of the functionality we first envisioned in the project proposal. We became acquainted with Android Studio, Kotlin, and Firebase, and got them all to work the way we wanted. We created a functioning roommate app with synchronized data across users, a robust login and group system, and the four main utilities of the app. However, there were still many small things we didn't have time to implement, and small compromises we had to make. This is expected, as we worked with an agile framework, where the scope is always in flux. The fact that there are many small features we didn't implement that would improve the app, but ultimately were not necessary, shows that the project is highly scalable. If we were to continue development and keep adding features, we think it could become a highly polished and useful app. Overall, we are happy with the end result.