Leo Xu, Kaustubh Khulbe, Niranjan Mathirajan

Note* Very Similar To Initial Specification

Fulkscord's Functional Specifications

1. Structural Design

Our Structural Design decisions are briefly summarized in the table below.

Data	Interface => Class
Users	Map => HashMap<String, User>
Friends	List => ArrayList<User>
Notifications	Queue => LinkedList<String>
Messages	LinkedList<Message>
Phone Number	String

Structural Design Choices For *Fulkscord*

At Fulkscord, our users are the key influencer of all our Design Choices, thus we make no compromises on efficiency and the speed of Fulkscord. Thus, we've chosen to store all of our users within a HashMap which holds the username of a user as a key, and the actual user object as the value. Moreover, given that Firebase, the database of our choice, holds data as a large JSON, storing our users inside a HashMap will be convenient and seamless for integration with Firebase. We could have potentially used a TreeMap, but since we don't necessarily need to grab our users in a particular order, grabbing a user could potentially take O(n) time, which we don't want. Our implementation with a HashMap will allow us to grab users in best case O(1) time.

Since each user will likely want to see who they are friends with we've decided to hold this data as a simple ArrayList of Users because it isn't necessary to hold this information in a particular order. Furthermore, it's better to use an ArrayList rather than a typical array for this task because an ArrayList can grow in size while a traditional array has a fixed size.

As far as overall notifications for a user go, we will utilize a queue. The reason for using a queue is simply because we want the latest message to show up first, rather than last. This could be counterintuitive since Queues follow the principle of First in Last out, but the key takeaway is that the older messages will display first, which will leave the first message that arrived on the top of the notifications for the user to see.

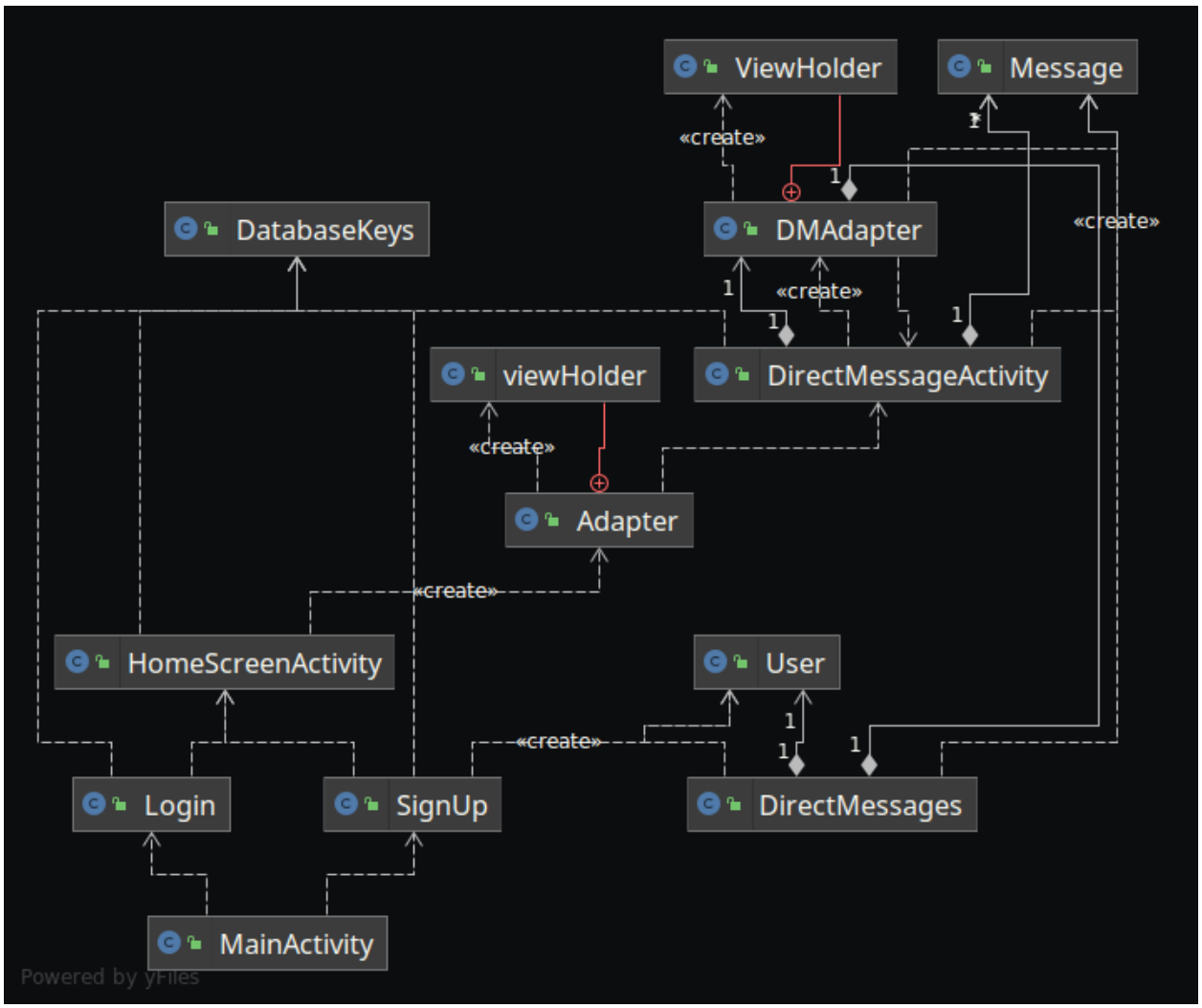
We've chosen to implement the messages between two users as a LinkedList so that we can easily keep track of the messages that occurred between two users and both add/delete messages in best case O(1) time. Since Fulkscord is looking to hold dynamic messages that are

stored in Firebase rather than the users' computer, we can simply just load the data into our LinkedList. Given that we're aiming to create a real-time chatting application, we can't jeopardize any bit of faster run-time, which is also why we decided to go for a LinkedList.

Finally, we've decided to hold our users' phone numbers as a simple string since we didn't want to overcomplicate our code. Furthermore, getting the phone number from Firebase will be easily convertible into a string, so it would be suitable to hold it as a string.

---

2. Object-Oriented Design



Fulkscord utilizes 13 classes along with multiple activity implementations and XML elements for Front-End with the Android Platform.

The User class is a data class, which is used to store the user information. This class is mainly used to hold information and doesn't do any logic on its own. An instance of the User class allows Fulkscord to grab a user's friends and other information associated with him/her.

The Message class is also a data class, which is used to store the contents of each message. This class holds both the usernames of the receiver and sender of the message, and it also holds the actual text of the message.

The Direct Message class is another data class, which is used to store the contents of the Direct Message with each person. This class utilizes the Message class in a Linked List to store all of the messages. It has getters and setters for the users involved in a direct message exchange.

The Main Activity is the opening screen of Fulkscord and allows the users to either Login or Sign Up.

They are taken to the corresponding page and either log in with a previous account or register for a new account.

Both activities lead to the HomeScreenActivity, which is the activity where users can see all the people they are talking with and have an option to make new friends. From there, a user can click on any of their friends and be taken to a DirectMessageActivity, where they will see all their messages with the users.

All the sign-ups, logins, messages, and friends are linked with Firebase to make sure the users can access the app and its information from any device and it is not a local app. In Firebase, the app will use the RealTime Database, which is structured as an ORDBMS (Object-Relational Database Management System). This allows us to easily store the information while retaining fields in objects and be able to retrieve information that is still structured. This prevents the need for parsing JSON data and allows for a much more concise application.

After sending and receiving messages from Firebase, we need to display them. To do this, we have decided to use a Recycler View for a few reasons. Firstly, it can use a Linear Layout and display messages stacked on top of each other. Second, it can allow us to "hide" some messages and be able to enhance the efficiency of our application. The recycler view uses an Adapter class so that we can customize all the message UI and cure it to our needs. The Adapter has an inner class called ViewHolder and the purpose of this class is to make objects of our XML of an individual message.

In a similar fashion, we implement the list of friends using Recycler Views, Adapters, and ViewHolders.

Lastly, we have developed a static class called DatabaseKeys. The purpose of this class is to simplify the access to our database and make sure we do not make errors by accessing the

wrong children. The class uses a static block since we only need one class accessible throughout Fulkscord.