Principles of Software Development
Chat App Report

Vinicius Amaro Cechin

30 of May, 2021

1 - Project Idea

The objective of this project is to implement a chat app without the use of a side server (p2p) which should also search for other near users to chat with.

The implementation was done using WiFi Direct, which is a tool used to share content with near androids via WiFi signal.

2 - Implementation Details

2.1 - Main Activity:

Initial screen of the application, where the user choose its username before connecting to peers. It has 2 main components.

The Username EditText view enables the Enter button when something is typed (or disable if string has length 0). It uses local storage to keep the chosen username and automatically set it on start up. When enter is typed on the keyboard, it is the same as clicking the Enter button.

The Enter Button view creates an Intent to go to Lobby Activity and pass username as extra data. It also saves the current username on local storage.

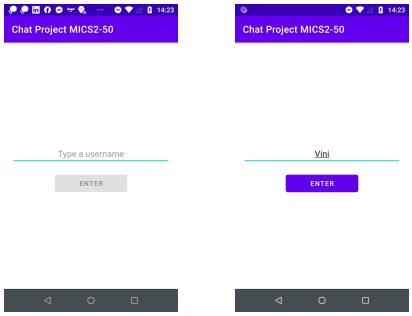

Figure 1: Main Activity

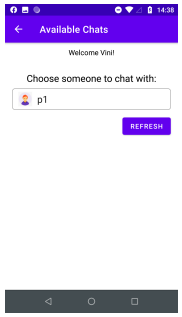

Figure 2: Lobby Activity

2.2 - Lobby Activity:

Screen where the user can see and choose a peer to connect. It shows the user the current chosen username by using the passed extra from Main Activity. It registers the broadcast receiver on resume and unregisters on pause. It has 2 main components.

Peers List ListView view updates its content based on the received peers list from the discovery process of WiFi Direct. It uses a drawable for the external border and for the items divisor. Uses layout fragment_peer as its ListItem.

Peer ListItem is a LinearLayout that contains a TextView with an avatar icon on the left of the peer name. When it is clicked it will try to open a connection between the two users. If the connection can't be done, a toast pops on the bottom of the user screen with an error message. If the connection is closed between the two parties, then the application is redirected to the Chat Activity.

2.3 - Chat Activity:

Chat ListView view shows all the messages sent by user and peer (type displayed is controlled by MessageAdapter). Message ListItem represent each message and uses the model Message to pass data into the MessageAdapter. This Message model contains 4 fields: text (the message text), username (the name of who sent the message), fromUser (True if message was sent by the user, False if it was sent by peer) and the avatar of the peer being chatted with.

MessageAdapter adapter contains the logic to add the message ListItem to the ListView. If is a message from the user, it uses the layout chat_message_user (has only messageBody field), else it uses chat_message_received (has messageBody field and name field).

2.4 - WiFi Direct:

WifiDirectController controls all the necessary logic of WiFi Direct implementation. It implements ConnectionInfoListener to receive the result of setting a connection with a peer (not yet done). It sets the intent filters to be

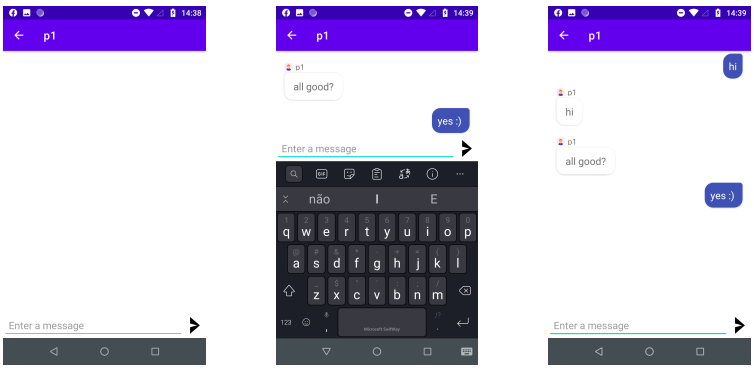

Figure 3: Chat Activity

listened to by the application. Also is responsible for discovering peers and updating the list of peers used by Lobby activity. Control the un/register of the BroadcastReceiver.

WifiDirectBroadcastReceiver listen to actions defined on intent filters by WifiDirectController. It Will request peers to WiFi Direct when something is changed in WiFi Direct list or device p2p state (which will be answered on WifiDirectPeersListListener)

WifiDirectPeersListListener just listens to the result of the call for peers list from WiFi Direct made by WifiDirectBroadcastReceiver.

2.5 - Socket Service:

When the chat activity is created, the extra data is retrieve which contains the info stating if the user is the group owner or not. If it is, then it will create a server socket, else it creates a socket and tries to connect at the server (group owner). The door used for both cases is the same, 8000.

The server starts accepting client connections and start listening to any future message the client sends. While the client starts listening to the group owner (server).

When a message is sent, if it is the owner, it sends to the clients (in our case there is only one). While if it is from the client, it will send a message to the server (which in our case is the other user).

3 - Difficulties

The most troublesome part was that i didn't already know android, so i had a long curve of knowledge to be able to do what is already done in the application (it was not easy at all). I believe that if instead of this big application we had smaller simpler ones would be best to learn the content of the lectures.

After That the most troublesome problem was that p2p and WiFi connections are not trivial to test using the emulator of android studio, so it is simpler to test with physical androids if one doesn't know how to set up the environment to use the emulators with WiFi communication. The problem is that in my house there is only 2 androids (an old cellphone and a portable emulator), so i had to set both of them up to test the discovery of peers by the application.

Another problem was exactly implementing the communication process between the two applications and manage it without a separated server like for example fire-base (which would help with understanding of API calls, as well as all the communication and connection between clients).

The library used for discovery and first connection between users does not support extra data being passed, so it was not the best option for this kind of app where we would need extra data previous to the connection (like use geolocation to pass the current position which each peer is, or a chosen image to use as avatar on other peers applications).

This kind of information can be given only after the socket connection is closed, because then we only have to care about the stream of data received.

4 - Learned Concepts Used

1. Github was used as the repository to keep track of the application changes during development time.
2. Diagrams and core functionalities pseudo-codes were generated previously to being programmed.
3. The structure used was MVA, where views are the xml files and adapters are the activity files.
4. Obviously Views, ListViews, Buttons, etc, were used to implement the front-end of the application.

5. handlers and event listeners were added for the peer discovery and front-end events.
6. Activities transitions with extra data were used to transition between the different activities of the app.
7. Constraints were used to try to allow all the mobile devices to have a good experience while using the app. Only dp measures were used.
8. Fixed strings were defined in a file to future support multi-language on the app.
9. Services using external threads were added to handle the socket connection between two peers and also to listen to changes in the peer list while in the lobby activity.
10. Persistent data of chosen user name was added to the main activity after the first login. It will be used on each time the user enter the application again to fill the username field.

5 - Conclusion

The application was really interesting, but i think that the definition of the project was too limiting, because we could only do p2p and not use a server to control the data. With a server we could do so much more while learning how to deal with a server api like Firebase for example. It would be easier to pass extra data and store old chat content to keep the data stored, or even create a login feature to protect each users data.

But the experience was also very interesting. Since i had the limitations stated in the report, i had to deal with them in a different way as i would in a usual case, like for example that i used one of the users as the "server" of the socket connection between the peers.

Since it was my first time programming with android i have the feeling that it could have being much better than the result obtained, but i think it will always be like that for the first app someone implements on any language.

6 - Code Repository

All the content referenced in this report can be found at **my GitHub page**.