Product: Tadashi

Team: Detroit
Student: Yuchen Niu

Abstract

Tadashi is a delivery robot for assisted living and quarantine situations. In the team, my main responsibilities include developing an Android application for carers to operate the robot and implementing network communication between the app and the robot. I learned user interface design and Android development by developing the Android application. I also learned team development and project management by working with other team members.

1. Contribution

I carried out majority of the features of the app and helped other teammates study Android and debug their code.

At start, I set up the skeleton of the app and helped my teammates study Android quickly by giving them some tutorials and videos. My teammates and I discussed the main features of the app and I mentioned a very useful website, Dribbble, about UI design. We checked several app templates for clinics and considered our use cases carefully, to determine what features the app should include. Since the app was mainly used for robot control, I thought our app must have the ability to send commands through the network to the robot, also caregivers could manage information of residents and monitor the residents' data. Eventually, I decided to use four fragments, different fragments for different purposes, and a bottom navigation bar as navigation logic on the main page, like other popular apps on the market. This was the best way to fit user familiarity. Moreover, data visualisation is also an important factor for pursuing a better user-friendly design. So, we replaced a suitable amount of texts with graphs and icons in our design to keep pages clear and simple.

After I finished the skeleton of the app, I asked my teammates about their skillsets and decided which task should go to which team member. It could play to team members' strengths. A team member was proficient in coding, so she helped with designing database format and data visualisation. Another team member is relatively weaker in coding and Android, so I delegated UI design to him at first and transferred him over to user testing later. I also suggested using GitHub's project management tools to keep tracking the progress of the project and let my teammates concentrate on their work.

I suggested using Firebase to store data and sync data across all clients in real time. Firebase provides platforms of NoSQL databases and easy-to-use SDKs for Android. Some features of the Firebase helped us build and manage a database easily for multiple clients instead of creating a complex web server to achieve the same goal. More importantly, nobody in the app team has a background in databases and web development. We also communicated the choice of platforms of Firebase databases and which data kept in the database during the implementation. I chose Firebase Auth, Realtime Database and Cloud Firestore to manage users' accounts, robot's state, residents' data respectively. We tried to avoid any sensitive information about users and residents when we developed the app because of the requirements of GDPR.

I implemented a login page and a sign-up page by using Firebase Auth. Firebase Auth provides encrypted backend services to authenticate users to the app. Firebase Auth allows users to login with their emails and passwords, and they can stay signed in afterward. Our app was used by authorized staff in care houses. Therefore, an authentication step and limited accesses to the app were necessary. It is important that all users can monitor the state of the robot and residents. Both Realtime Database and Cloud Firestore are NoSQL databases that can keep our data in sync across client apps through real time listeners and offer offline support for mobile. However, the Cloud Firestore features richer, faster queries and scales further than the Realtime Database. I used the Cloud Firebase to store residents' profiles and data of deliveries since they are unlikely to be modified frequently and the collections of documents model is more suitable. The Realtime Database is an efficient, low-latency solution for mobile apps that require synced light-weighted data, such as robot state, across clients in real time

I implemented a UDP client service on the app for bidirectional communication between the app and the robot. The service was a foreground service receiving messages from the robot, which kept running even after the app is destroyed theoretically. Any message received by the service was shown on notification, so users could track the state of the robot on any page. I also implemented a thread pool to control all egress packets from the app.

I solved a personal device discovery problem on the network while testing the communication between the app and the robot. My phone was not discoverable on SDPRobot, but my laptop could be discovered by shutting down all firewalls. I must use the emulator on my laptop to test the communication. Since the emulator ran behind a virtual router/firewall service, I redirected incoming UDP packets from the localhost to the emulator. However, telnet occupies a port of localhost while it redirects the packets. I opened two UDP ports, one received and re-sent packets to another port, another port occupied by telnet redirected the packet to the emulator, on my laptop. The emulator popped up the received message on the notification.

After the basic features were finished, I improved the user interface to make it more user-friendly. I picked three base colors to generate a set of colors by using Leonardo. Two of these colors were chosen from the team logo to keep the app consistent with the project. A good user interface requires not only clear, contrast colors, but also responses for every event from users. I indicated whether a button/text is clickable, by different colors. I also used either animation or static responses as the feedbacks of users' operations.

I wrote the app part of the user guide. The user guide was still followed similar principles as user interface design. I inserted as many screenshots or pictures as I could to make the user guide easier to read and more understandable.

2. Lessons learned

The project is my first fully functional Android program. It reinforced my knowledge about Android, especially on four app components and fragments. The reinforcement and previous experiences helped me develop the app, spot and solve problems quickly. My solid comprehension of Android was also proved by helping my teammates and classmates developing Android apps.

In my implementation of the app, both network communication and database access were background tasks. I learned several methods to keep a service alive. Because the network communication is supposed to run indefinitely to receive packets from the robot, I had trouble finding a method that keeps the network service running even if the app is closed. I tried AlarmManager, guard service, promoting priority of the process and other methods. However, these methods had been forbidden by Google on a higher version of Android for battery efficiency. All processes were terminated eventually on our targeted Android version as the app was closed. I was also looking for a place on UI to show the robot state. I decided on using a foreground service for the ingress network and the state of robot received by the service showed on the notification. Because of the stability of foreground service, it also kept another background service running for a long time. Therefore, the foreground service solved three problems simultaneously. I learned the basics of multi-threads and thread pools. For egress networks, A new background thread was created whenever a user sent a packet to the robot. So, I implemented a thread pool to manage all background threads for better performance and stability.

I read through the guides on the Firebase website and learned basic operations of these NoSQL databases. The first time I used the APIs, I did not know that all Firebase operations did not run on the UI thread. But the result of reading operations was required to be shown on the page. It resulted in an app crash if the user moved to another page before the read finished because the UI element for showing the results could not be found on the new page. Therefore, I added a flag when the focus of the current page was lost. If the flag was set, it stopped inflating the related UI element with the result of reading operation. I also prefetched commonly used data to improve the runtime performance of the app. It saved the database access time and left only UI loading time. Data was fetched into a list if and only if the app was started by authorized users. When data was changed on the database by other users, a real-time listener could find and update the data in the list. Therefore, heavy data loading did not affect user experience anymore.

I acquired the principles of UI design. A user interface that is invisible and that provides seamless interaction possibilities will help the user focus on their goals and direct them to what they need. I used common page layout and UI elements to make users feel more comfortable and can get things done more quickly. The pages with similar features had the same layout and pattern. Once a user learned how to do something, they were able to transfer that skill to other parts of the site. I directed the user's attention toward interactive items, mostly buttons, using colors, fonts, and arrangements of the text to help increase scannability, legibility, and readability. A good user interface also makes sure that the system communicates what's happening. The app always informed users of the location, actions, changes in state, or errors by using various UI elements.

Coding in a team is different from coding solely. The project taught me how to work as a team player. Staying shadow will not solve any problems. I learned how to share my idea with teammates and resolve conflicts between the team and myself.

I still need to improve some defects revealed from the project. From a developer point of view, I failed to design a clear and useable skeleton. My skeleton only achieved high cohesion low coupling by creating essential classes in an object-oriented way. However, there was no interface and comments for every class, even utility class, which made other team members harder to utilize my code. I implemented a few features without comments again as my teammates studied Android. It caused my teammates to have trouble understanding my code and taking over tasks. From a team leader's point of view, I need to enhance oral expression, I realize that the only way to let others understand my idea and jobs is to explain them. I also should know what the audiences want to hear and be flexible to answer any questions they ask. Another problem is that the delegation of the app team was a failure since nobody knew each other before the project and the only assessment of their skills was through communication. I delegated an important task to an incompetent teammate who volunteered. However, the teammate did not pay enough attention to it and failed the work multiple times. I had to carry out the job for the teammate. Otherwise, we missed the milestone. It made me disappointed and started ignoring the teammate. I reported the situation to our group leader and our group leader gave the teammate another job to do fortunately. I still do not know how to handle this kind of situation. Probably I should talk to the teammate first. But I think the teammate will still act as usual even if the teammate takes my suggestion.

References

Documentation For App Developers. 2020. [online] Available at: <https://developer.android.com/docs>.

Dribbble - Discover The World'S Top Designers & Creative Professionals. 2020. [online] Available at: <https://dribbble.com/>.

Firebase Android Guides. 2020. [online] Available at: <https://firebase.google.com/docs/android/>.

Leonardo. 2020. [online] Available at: <https://leonardocolor.io/>.