Individual Report

Penghui Xiao c1020412

1. What did you contribute to the group? List skills as well as concrete tasks (e.g., presentation writing skills, design, coding, testing, etc.). [6 marks] (1 page including figures)

- **Back-end coding:(my code is mainly for user information and local cache)**
  ForgetPasswordFragment.java, LoginMainFragment.java, NewAccountFragment.java, UserLogInFragment.java, AccountEditActivity.java, AppsettingsActivity.java, ImageUtil.java, PasswordUtil.java, WiFiUtil.java, AboutThisApp.java, Avatar.java, LogInActivit.java, MainActivity.java.
- **Front-end coding:(I modify some layout files that are provided by JiaCheng and Mai who oversee Front-end, when their designs do not meet the function I want)**
  activity_about_this_app.xml, activity_account_edit.xml, activity_app_settings.xml, activity_avatar.xml, activity_log_in.xml, activity_main.xml, bottom_sheet_account_edit.xml, bottom_sheet_avatar.xml, fragment_forget_password.xml, fragement_log_in_main.xml, fragment_new_account.xml, fragment_user__log__in_.xml.
- **Testing and debugging:**
  Testing: Every time I finish a function, I run the App in 3 devices to test it -- Android 12 phone, Android 8 phone, and Android emulator. Because these 3 devices have different sizes and android versions, some functions may not be suitable for them all. In the final test, I acted as the user to test it from start to finish. Then I gave my devices to Aysha, Mai and Alex to ask them to test it again.
  Debugging: Sometimes the data could not be transmitted correctly. I used debugger function in Android Studio to retrieve the data. The data would be shown in the logcat. Then I would find which step was wrong.
- **Remote coding and version control:**
  Remote coding: I created a GitHub repository for our team to code remotely because during the Easter Holiday, some team members were not in Newcastle. With the repository, we would not need to send code each other anymore. I also told my team members how to use the repository and I created a branch for everyone.
  Version control: Some team members were not so familiar with the GitHub repository. Therefore, they sometimes made mistakes when committing, push or pull the code. When my team members misoperated, they did not know how to return to the previous version. Then I helped them to return the version. Sometimes there were conflicts in code when merging into the master branch. I also deal with the conflicts.

2. Describe three things you feel you personally did well. [6 marks] (1 page including figures)

- **Implementing the function of local cache to optimize the application loading speed**

  Reason: Without local cache, every time we use the App, it will connect to database and get data, decreasing the speed of loading.

  How to implement:

  Example: save, get, and show user's avatar locally.

  Save:

  Step 1. Convert the BMP format photo to Base64

  ```
  imageBase64 = ImageUtil.imageToBase64(bitmap);
  ```

  Step 2. Create a unique SharedPreferences for the user and put the Base64 format photo to it.

  ```
  SharedPreferences spfRecord = getSharedPreferences( name: "spfRecord"+username, MODE_PRIVATE);
  SharedPreferences.Editor edit = spfRecord.edit();
  edit.putString( s: "image_64", imageBase64);
  edit.apply();
  ```

  Get:

  ```
  SharedPreferences spfRecord = getSharedPreferences( name: "spfRecord"+username, MODE_PRIVATE);
  String image64 = spfRecord.getString( s: "image_64", s1: "");
  ```

  Show:

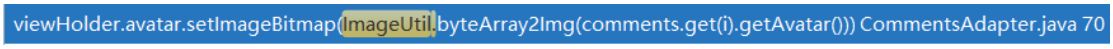
  ```
  imageView.setImageBitmap(ImageUtil.base64ToImage(image64));
  ```

- **Writing some Util classes in which are static methods that my peer Jing can also use.**

  Reason: public static methods can be called in the whole project, reducing the redundancy of the code.

  Example: ImageUtil.java contains methods to convert format of avatar. I had finished the function of uploading Avatars by using these methods and I guessed Jing may also need to use them when he set the user avatar on the comment item. With the ImageUtil.java, Jing does not need to write methods to deal with avatar.

  The screenshot below shows that Jing called byteArray2Img method in ImageUtil.java in line 70 in CommentsAdapter.java

- **Communication with team members.**

  Reason: My main role is back-end. Sometimes when I was writing a function, I might need a front-end layout and a table in database. Then I asked JiaCheng who oversees front-end to design a new page for me. Meanwhile, I also told him what widgets I need, describing my requirements clearly to him. When I asked Alex who oversees database to create a new table for me, I also told him why I needed the table and what columns and key constraints I need in this table.

  Example 1: I needed to provide a function for user to find his password back. So, I asked JiaCheng to create a fragment named fragment_forget_password.xml based on activity_log_in.xml in log_in_nav_graph.xml. Meanwhile, I told him the fragment should contain at least 2 EditText and 2 Button widgets…

  Example 2: I needed an Avatar table in database to save user's avatar. Then I told Alex to add the table, using username as primary key and foreign key…

3. Identify three issues you encountered while doing the group project, and what you personally did to resolve these issues [6 marks] (1 page)

- **The application consumes too much CPU and network during running.**

  Bug: When the user logs in successfully, the application continuously retrieves the avatar from the database and caches it locally.

  Reason: I wrote an endless loop carelessly in a Thread in MainActivity.java to retrieve avatar from database.

  How did I fix: Stop calling the getAvatarFromDB method in this thread. Instead, set the avatar(is got from local cache first and then from database) to the imageview directly when user clicks the avatar to open Avatar-setting page.

- **The application could not get Wi-Fi information on Android 9+**

  Reason: On android 9+, we need location permission to get Wi-Fi information.

  How did I fix: Add location permission in AndroidManifest.xml.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

  Check android version. If it is 9.0+, apply for runtime permission.

```
//wifi connected, check wifi ssid
if(Build.VERSION.SDK_INT < Build.VERSION_CODES.P) {//if lower than android 9
    ssid = WiFiUtil.getConnectWifiSsid(wifiManager);
}else {//higher than Android 9, apply for runtime permmison
    if(ContextCompat.checkSelfPermission( context: LogInActivity.this,
            Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: LogInActivity.this,
                new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, requestCode: 200);
    }
    wifiManager = (WifiManager) getApplicationContext().getSystemService(WIFI_SERVICE);
    WifiInfo wifiInfo = wifiManager.getConnectionInfo();
    ssid = wifiInfo.getSSID();
}
```

- **After the user password is encrypted and saved to database using the DES encryption algorithm, garbled characters are displayed. Therefore, no way to decrypt the password.**

  Reason: MySQL database encoding format does not match android. This is one of the reasons why SQLite is recommended for Android apps rather than MySQL.

  How did I fix: Instead of using DES encryption and decryption algorithms, wrote algorithms that convert strings to hexadecimal numbers, and convert hexadecimal numbers back to strings.

```
public static String str2Hex(String string) {
    char[] chars = "0123456789ABCDEF".toCharArray();
    StringBuilder sBuilder = new StringBuilder("");
    byte[] bs = string.getBytes();
    int bit;
    for (byte b : bs) {
        bit = (b & 0x0f0) >> 4;
        sBuilder.append(chars[bit]);
        bit = b & 0x0f;
        sBuilder.append(chars[bit]);
    }
    return sBuilder.toString().trim();
}
```

```
public static String hex2Str(String hex) {
    String string = "0123456789ABCDEF";
    char[] hexs = hex.toCharArray();
    byte[] bytes = new byte[hex.length()/2];
    int n;
    for(int i =0;i<bytes.length;i++) {
        n=string.indexOf(hexs[2*i])*16;
        n+=string.indexOf(hexs[2*i+1]);
        bytes[i] = (byte) (n & 0xff);
    }
    return new String(bytes);
}
```

  The algorithms are referenced from a blog.

4. Think about the things you personally did not do so well. Identify three areas in which you would seek training as part of your continuing professional development. [6 marks] (1 page including figures)

Things I did not do well:

- Commonly, when we register or find password back, we will receive an email of verify code. But I did not add this function to the App. Because I need to deploy some APIs which are not free of charge. And it is also complicated to deploy APIs.
- After registering successfully, it should navigate to the login page directly. But I let it return to the initial page and ask user "Registering successfully. Would you like to login with the account?" Because the login page and registration page are both Fragments, only the initial page is an Activity. It is easy to jump from a fragment to an activity. But it is hard to jump from a fragment to another fragment.
- User can login with the same account in 2 or more different devices. But commonly, if we login an account in phone A and try to login the same account in phone B, it will tell you 'login failed' and show the warning message "This account has already been logged in on another device." I have an idea to deal with this problem. Create a column called 'state' in Users table to record login state(0 for log out and 1 for login). After logging in successfully, the state become 1 from 0. After logging out, it becomes 0 from 1. Every time the user logs in, check the state first. If it is 0, login successfully and let it become 1. Else if it is 1, show the warning message, login failed… I did not add this function because if I did, I would have to modify User.java. And because User object is used in almost everywhere in the whole project, we would also need to modify many other java classes.

Three areas in which I would seek training

- **API deployment**
  For example, email sending API, Bus departure information API and Google Map API. Because API is always useful and popular in many Applications.
- **Android Fragment Life Circle and data transmission**
  It is easy to transmit data from activity to activity and from fragment to activity. But if we want to transmit from activity to fragment, from fragment to activity or from fragment to fragment, we will need many lines of code. In future coding, these transmissions are inevitable. It is necessary to understand Android Fragment life circle and how to deal with these transmissions.
- **Understand mechanism of accessing the database by java code.**
  Most open source development frameworks(I prefer using them) provide developers with tools to connect to databases. For example, Springboot offers us Mybatis in which developers can write sql code directly without creating Dao java class to modify database. Therefore, I did not understand the mechanism of accessing database by java code. In our team project, Jing created Dao classes for me. I think it is necessary to understand them. Because in future working, it is inevitable to develop applications without frameworks.

5. Identify the main thing the group could improve on if you were able to redo the project [3 marks] (0.5 pages).

If we were able to redo the project, the group would strictly follow the agile development process.

During the beginning of development process, we sometimes came up with a requirement and just started implementing it. We did not have meetings regularly. Sometimes 7 days per meeting. Sometimes 3 days. Therefore, we did not know what other team members had done and what was the plan in the next week··· Sometimes we had conflicts. For example, Alex changed some structures of the database (for example, he changed the column name from 'led' to 'leg') without telling me and Jing. Then when Jing and I were testing the App, it crashed and showed no column called 'led' in table 'Journey'.

If we followed agile process, the development would be more efficient with less conflicts.