Lab #2: QR Code

EE447 Mobile Network, Luoyi Fu, Spring 2021

Due: *Sunday, May 2nd*

Name: Hongjie Fang Student ID:518030910150 Email: galaxies@sjtu.edu.cn

1 Purpose and Objective

A barcode is an optical machine-readable representation of data relating to the object to which it is attached. Originally barcodes systematically represented data by varying the widths and spacings of parallel lines, which can be viewed as one-dimensional representation.

Later, two-dimensional representation is developed, and among them, most important and widely used one is QR code, *i.e.*, Quick Response code. QR code has become the most popular encoding methods on mobile devices nowadays, and the QR Code system became popular outside the automotive industry due to its fast readability and greater storage capacity compared to standard UPC barcodes. The applications of QR code include product tracking, item identification, time tracking, document management, and general marketing.

In this lab, we are given a well-written sample application for QR code encoding and decoding. Basically, what we are required to do is to successfully compile and execute the application in a real-world mobile phone and test for its performance. What's more, I make some critical modifications to make the application handier to users.

2 Compilation and Execution of the Application

We use Android Studio to open the project. While automatically configuring the project, the Android Studio found out that the SDK path is not correctly set, as shown in Fig. 1. Click "OK", and it will automatically set the correct SDK path for you.

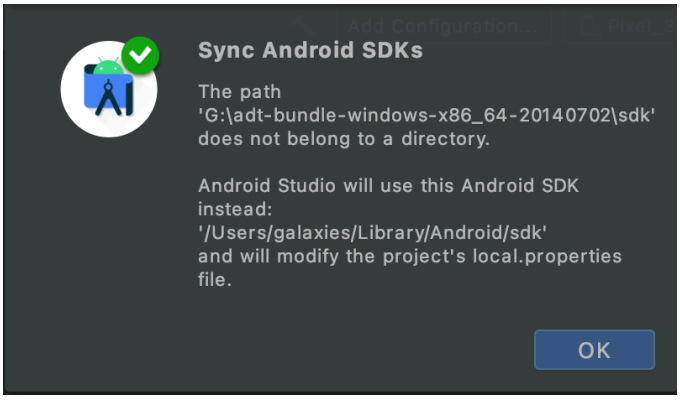

Fig. 1. Android Studio automatically set SDK path

Then, from the following lines of the `build.gradle` for the application, we know that the version of SDK compiler is 21, and the version of the building tools is 23.0.1.

```
compileSdkVersion 21
buildToolsVersion "23.0.1"
```

Open SDK Manager, and install SDK Platform for Android 5.0 (Lollipop) along with Android SDK build tools of version 23.0.1. Thus, we successfully complete the SDK settings of the project.

Then, we need to fix the gradle version of the project. During this process, many unexpected errors may occur due to different environment of the computer. I compile the application under MacOS Big Sur 11.3, and have encountered the folowing exception as Fig. 2 shown.

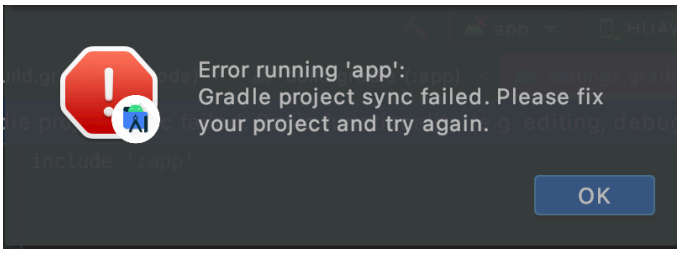

Fig. 2. The exception that I have encountered due to incorrect settings of gradle

To solve this, we need first open "project structure" and set the Android Gradle Plugin Version to 4.1.3, and set the Gradle Version to 6.5, as Fig. 3 shown.

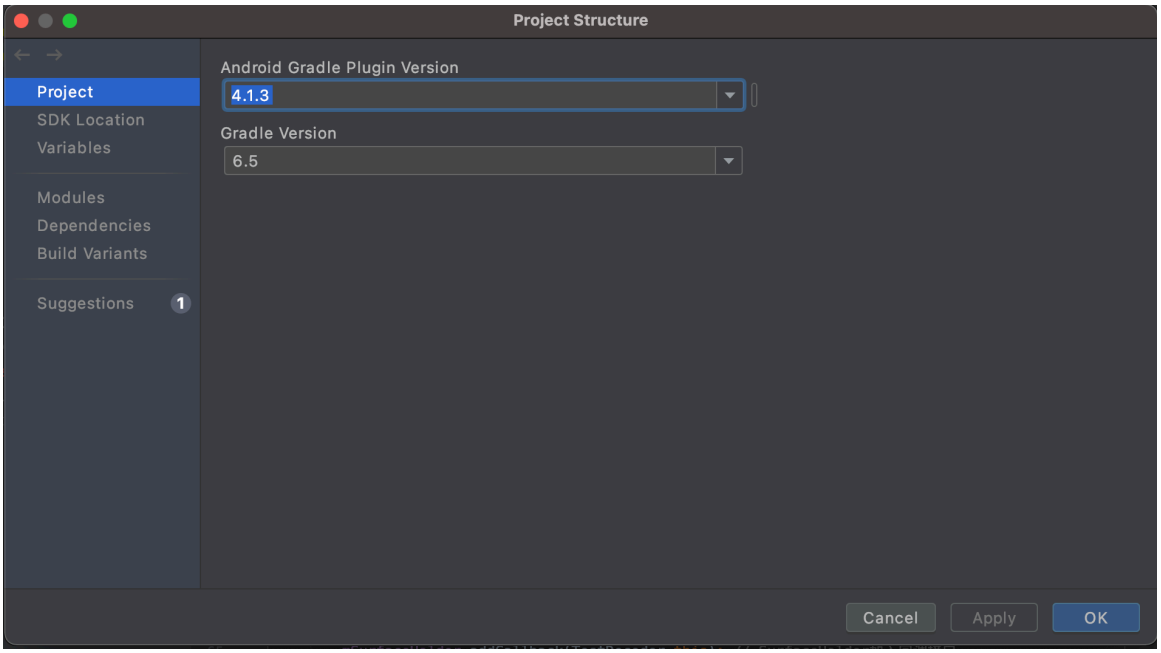

Fig. 3. The correct gradle configuration

Then, the Android Studio will automatically synchronize the project. But unfortunately, we may meet another problem shown as follows.

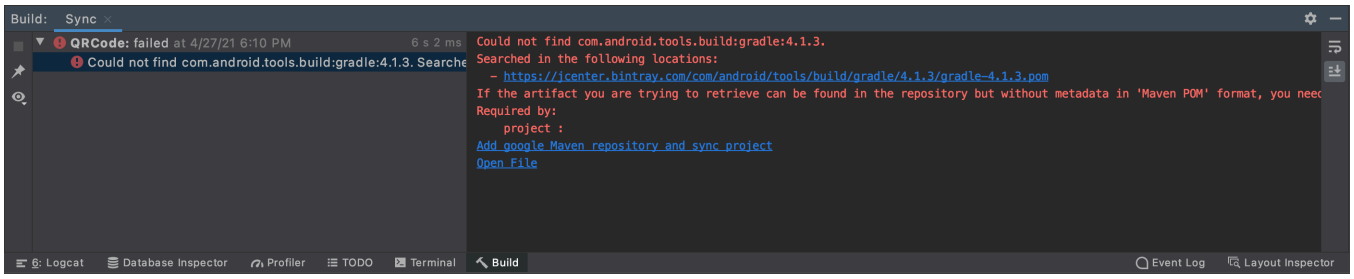

Fig. 4. The correct gradle configuration

The error message indicates that we should add Google Maven repository and synchronize the project again. Therefore, we can click the "Add google Maven repository and sync project" and then click "Do refactor" to add Maven repository to the project.

After adding Maven repository to the project, the project synchronized successfully. Therefore, we can write the application into a real-world mobile to test the performance.

We use a HUAWEI P20 mobile for real-world testing, and the main function of the initial application is shown in Fig. 5.

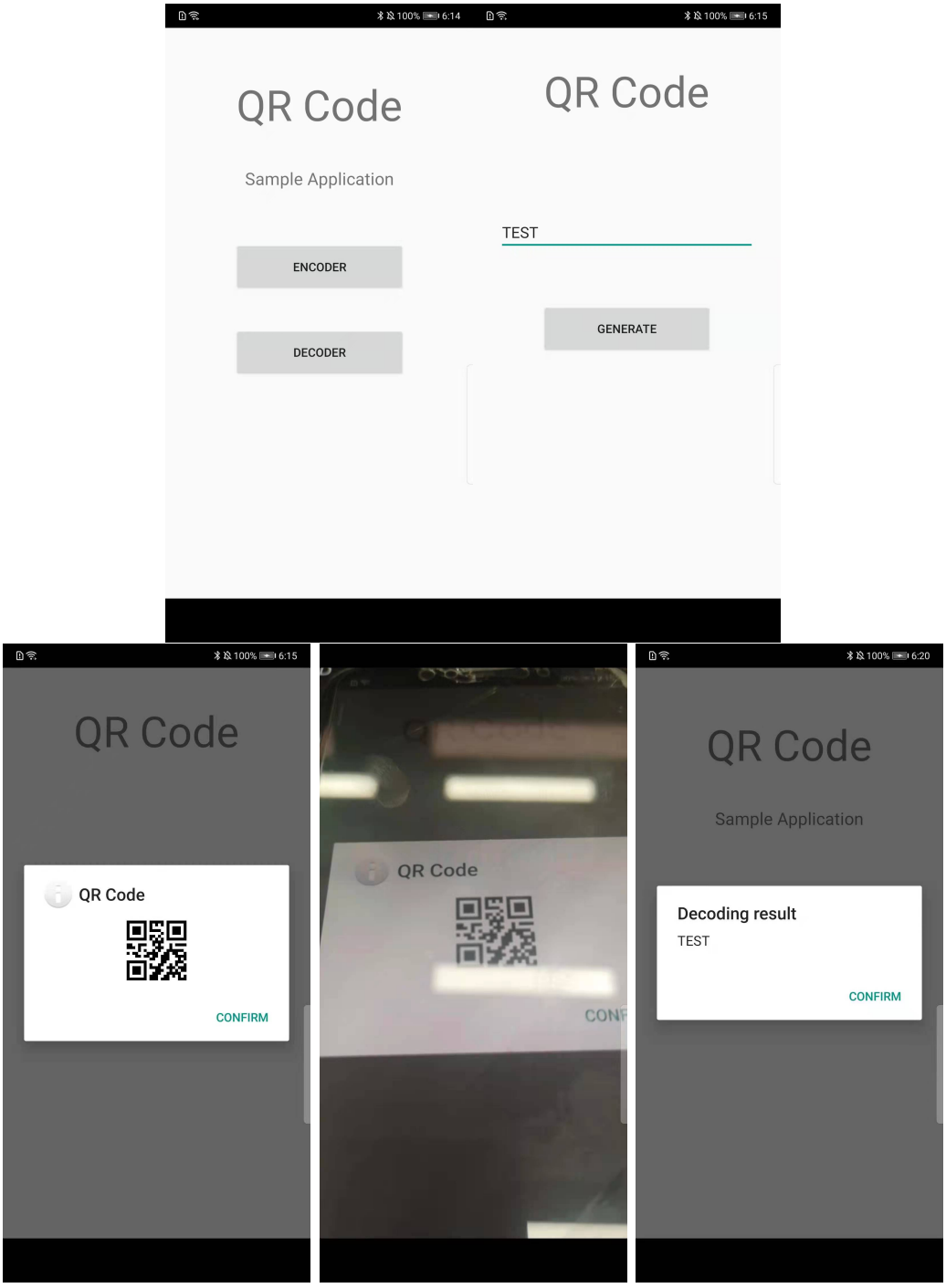

Fig. 5. The main function of the initial application

From a user's view, the application is not so convenient because:

- It does not support saving the generated QR code into the mobile storage.
- It does not support detecting QR code from the pictures stored in the mobile.

Therefore, we propose our improved QR code application in the following sections.

3 Code Explanation

To improve the application, we first need to understand the logic of the initial application.

3.1 MainActivity

Notice that the function `onActivityResult()` in `MainActivity` appears for the first time, which aims to get the response from some other triggers or message passing. In the application, the function receive responces from `TestDecoder`. When `TestDecoder` meets a valid QR code, it send the decoding result back and `MainActivity` make a `AlterDialog` to show the result.

Therefore, for `TestEncoder`, we just need to start the activity, and we call `StartActivity`; for `TestDecoder`, we need to start the activity and receive the responses, and according to the previous introduction, we should call `StartActivityForResult`.

3.2 TestEncoder

In `TestEncoder`, we just need to call the APIs provided by the zxing* library. Basically, what we need to do is:

1. Get the text from the `EditText` module;
2. Call `MultiFormatWriter()` API provided by zxing library with the parameter of the text reading fro m the previous step.
3. Convert the returning `bitMatrix` into RGB colors and draw the pixels on a bitmap.
4. Put the bitmap on an `ImageView` object, and display the object on an `AlterDialog` to show the result.

3.3 TestDecoder

In `TestDecoder`, we need to read the image stream from the camera and call the APIs provided by the zxing library to decode the QR code. The first function is implemented using `SurfaceView`, and the second function is implemented using the `QRCodeReader` object in the zxing lib.

For `SurfaceView`, the project implements three methods: `surfaceCreated`, `surfaceChanged` and `surfaceDestroyed`. The first one and the last one is similar to the constructor and destructor of an object, and the second method is called only when the format or size of the preview interface is changed, which allows us to dynamically detect the QR code in the camera image stream.

4 Improvements

4.1 Save QR Code to Mobile Storage

To enable saving QR code to mobile storage, we first modify the `AlertDialog` in `TestEncoder` to add a `save` button. When the user press the `save` button, the application will call our `save_pics` method, which is implemented as follows.

```
private void save_pics(Bitmap bitmap) {
    String path = getApplicationContext().getFilesDir().getPath()
                  + File.separator + "QRcode";
    File appDir = new File(path);
```

*https://github.com/zxing/zxing

```
    if(!appDir.exists()) appDir.mkdirs();
    String filename = "QRCode" + System.currentTimeMillis() + ".jpg";
    File file = new File(appDir, filename);
    try {
        FileOutputStream fo = new FileOutputStream(file);
        bitmap.compress(Bitmap.CompressFormat.JPEG, 60, fo);
        fo.flush();
        fo.close();
        MediaStore.Images.Media.insertImage(
            getBaseContext().getContentResolver(),
            file.getAbsolutePath(), filename, null);
        Uri uri = Uri.fromFile(file);
        getBaseContext().sendBroadcast(
            new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE, uri));
    } catch (IOException e) {
            e.printStackTrace();
    }
}
```

Notice that saving the picture to the mobile storage takes three steps[†]:

1. Save the picture first. In this step, we should add the permissions in the `AndroidManifest.xml` file of the project since we are accessing external storage. Therefore, we can add the following lines in the `AndroidManifest.xml` file:

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

2. Insert the picture to the system picture gallery. This step is vital since we need to let the Android system to find the pictures we have saved in the system. To achieve this goal, we just need to call the `MediaStore.Images.Media.insertImage` method.

3. Notify the picture gallery to update. We sent a broadcast to notify the system picture gallery to update using `Uri` and `sendBroadcast` method.

Therefore, we have finished adding "save QR code" button in the encoding process.

4.2 Detect QR Code from Pictures in the Gallery

To enable detecting QR code from pictures in the gallery, we first add a button and make minor modifications to the layout of the `TestDecoder` interface. Then, we use `Intent.ACTION_PICK` and `intent.setType("image/*")` method to get access to all the images in the mobile storage. Therefore, we just need to fetch the user-selected image and send it back to the main process. Here we reference some implementations of the `surfaceCreated` and use the `onActivityResult()` function we have introduced before. For detail of the implementation, refer to the source codes.

[†]https://www.cnblogs.com/monkey0928/p/10716923.html

5 Final Performance

We use the same HUAWEI P20 mobile for real-world testing, and the main function of the our modified application is shown in Fig. 6.

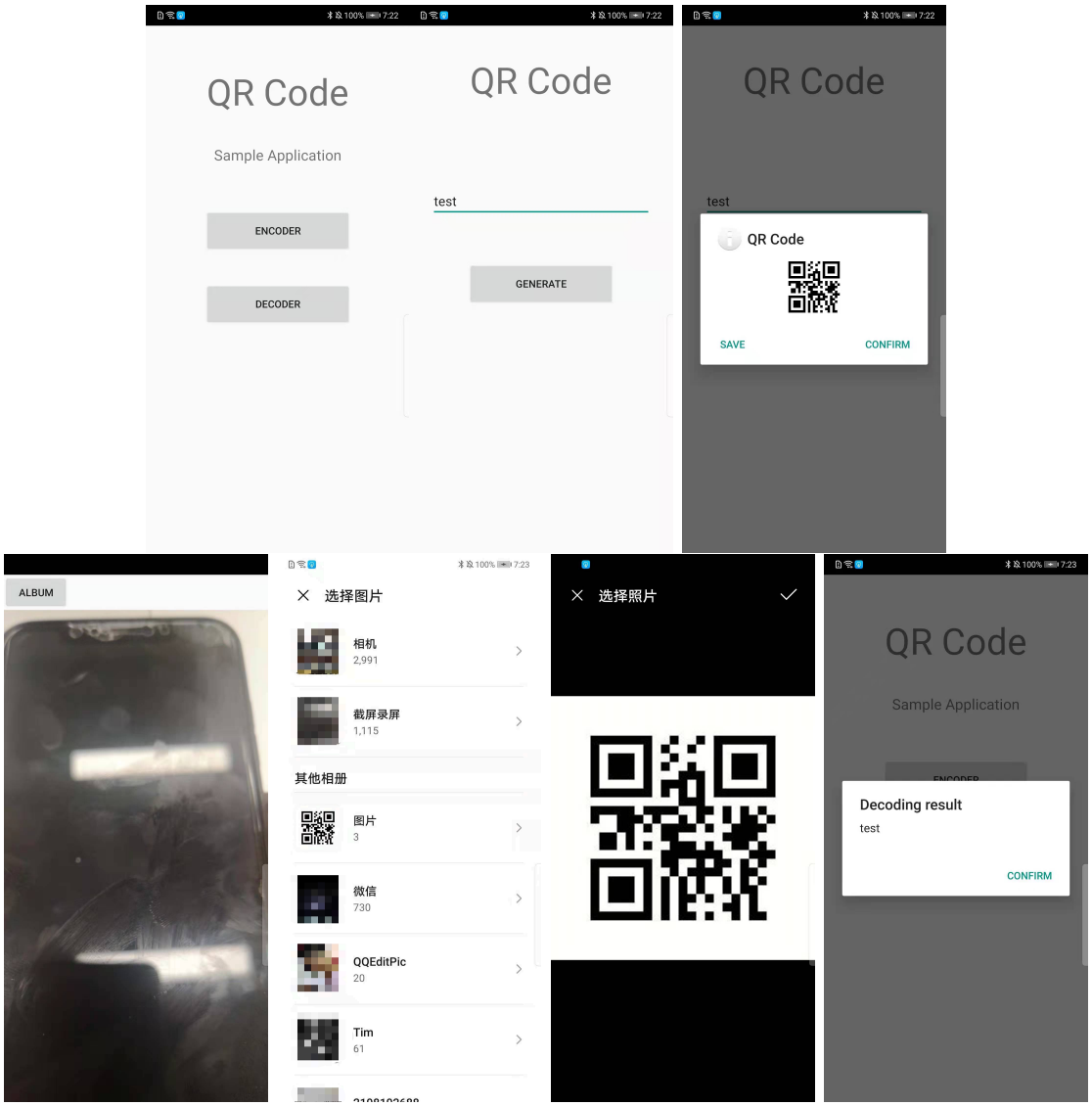

Fig. 6. The main function of our modified application

Our modified application has all the functions of the initial application. What's more, the two additional functions (saving the QR code to the mobile storage, and detecting QR code from pictures in the gallery) make the application handier, and more user-friendly.

6 Conclusion

To complete this lab, I have read some materials about how to use zxing library. Moreover, I'm more familiar with the Android Studio to develop Android applications. After finishing the basic requirements of compiling and executing the application, I make some important modifications to make the application more user-friendly, such as saving QR code to the mobile storage and detecting QR code from the mobile gallery. Future work may include developing a prettier UI, supporting different types of QR code and redirect to the corresponding applications, such as Alipay and Wechat. I am looking forward to implement them in the future.

The full implementation codes of the lab is available in my github repository.