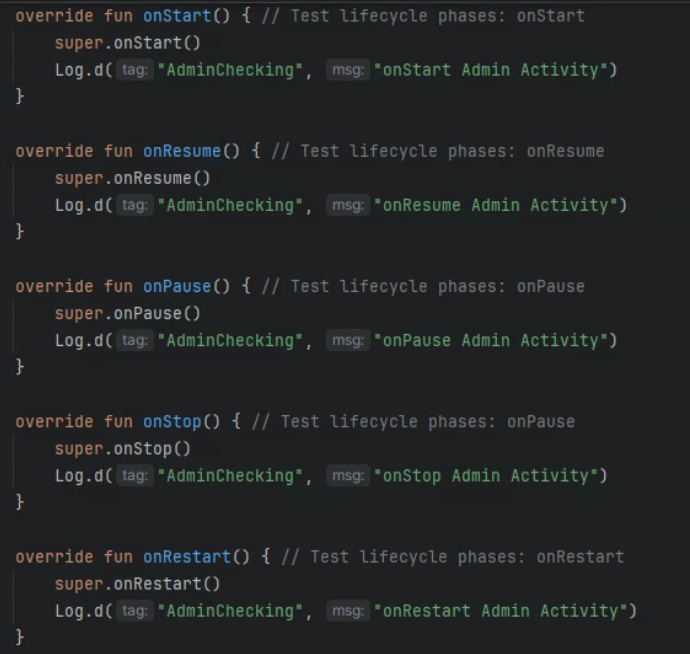
Name: Anh-Quan Nguyen

ID: 104850254

**Activity Lifecycle Report**

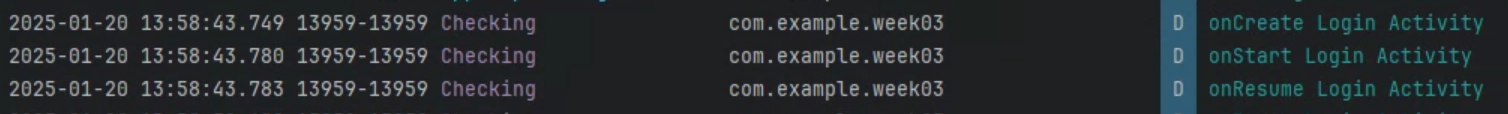
According to Google, each Android activity has their own lifecycle, which includes *onCreate(), onStart(), onPause(), onResume(), onRestart(), onStop() and onDestroy()*.

When there is any changes to the state, the app will invoke these callback functions. In order to observe the behaviour of the activity, I modified the callback functions by adding a log. When these functions are invoked, this activity will be logged and we can see how the app work in its own lifecycle. **Figure 1** shows how we can do this in Android Studio.



*Figure 1. Logging the invocation of functions*

After that, I run the app and observe the invocation of callback functions in the *Logcat* window. **Figure 2** shows the result after running the app.



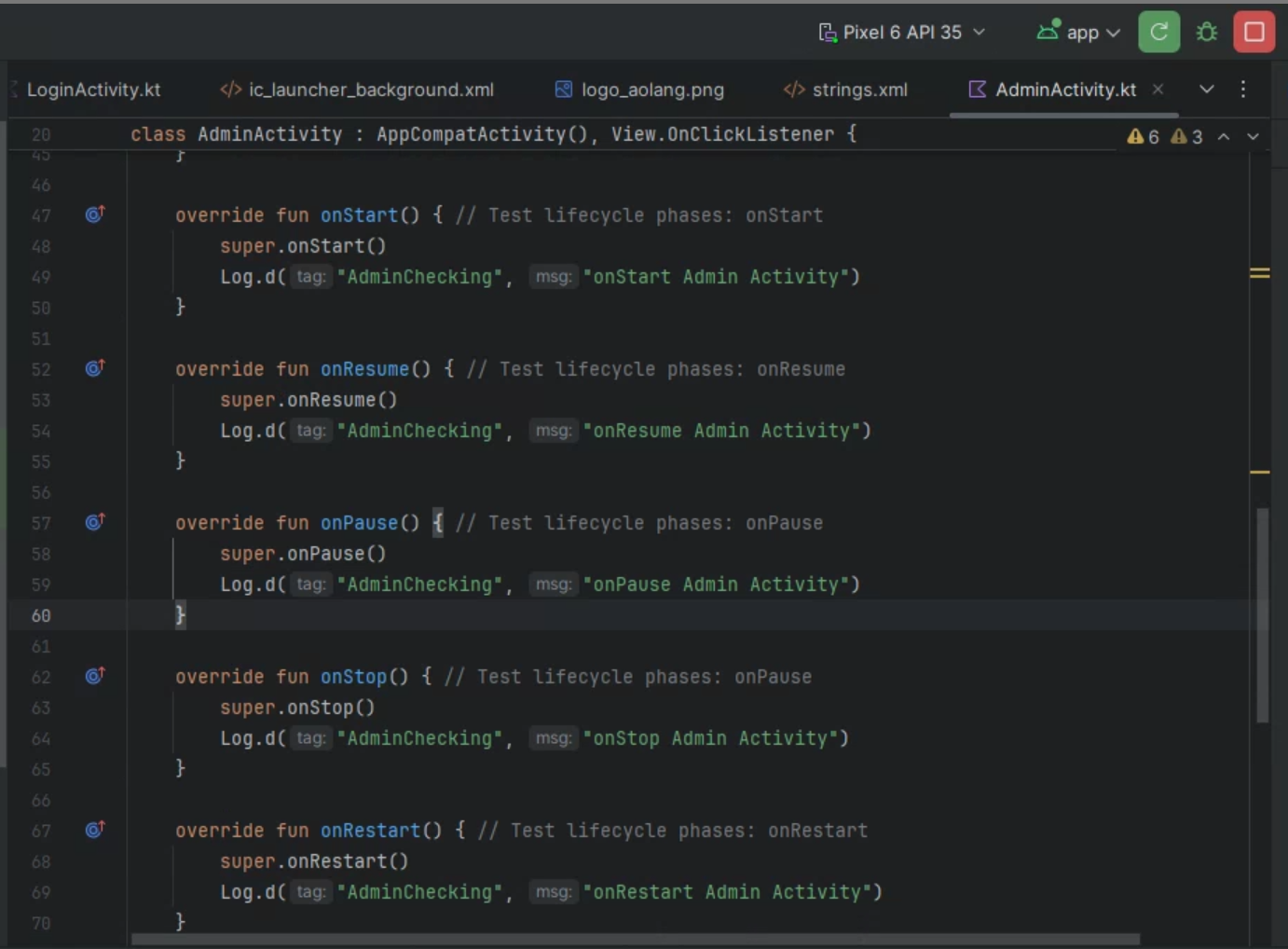
*Figure 2. Observe the logs in the LoginActivity*

When the app runs for the first time, the *onCreate()* function of the entrance activity will be fired first. The *onStart()* and *onResume()* functions are also invoked.

What happens if we change to another activity? The Android documentation said that “The activity state can also change in response to device-configuration changes, for example when the user rotates the device from portrait to landscape. When these configuration changes happen, the activity is destroyed and recreated in its default state, and the user might lose information that they've entered in the activity”. Let’s observe it by adding the logs for the other activity.

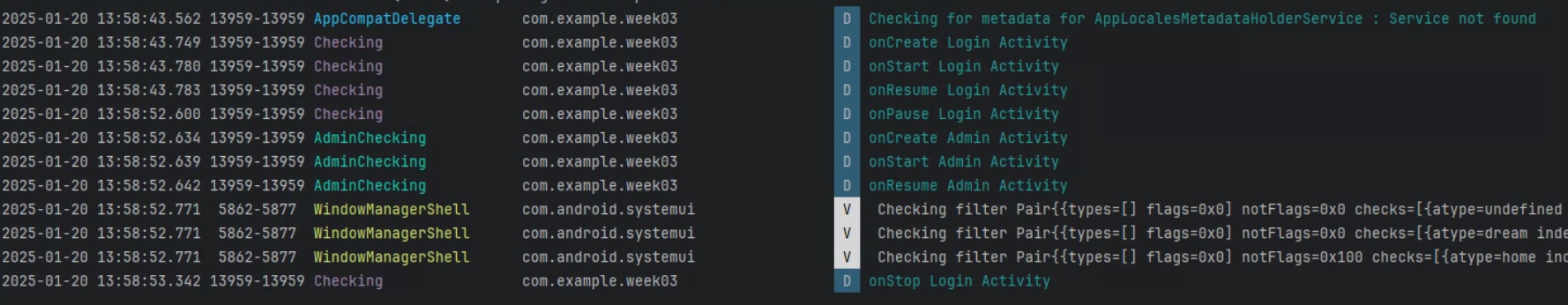
We will create another activity called *AdminActivity* and adding some code to the *LoginActivity*, to ensure that when users click on the “Login” button, the app will switch to the *AdminActivity* screen. It is essential to add logging for each callback functions in the *AdminActivity*. To separate the logs of each activity, we use different tagnames for them.

The following **Figure 3** shows how we can do this.



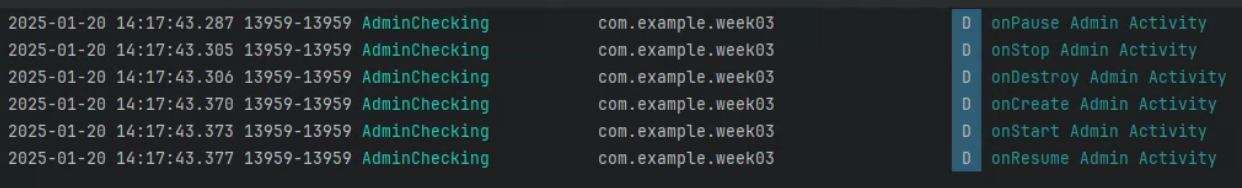
*Figure 3. Adding the log for AdminActivity.*

The **Figure 4** shows the log of *AdminActivity.* When the user clicks on the “Login” button, we can see that the *LoginActivity* has fired the *onPause()* function. The same procedure also applied with the *AdminActivity* when it started the lifecycle. After the *AdminActivity* has completed this process, the *LoginActivity* will stopped working.



*Figure 4. Observe the logs in the AdminActivity*

Another case to consider is that, when we change the configuration of a device, does the activity stop its lifecycle? The answer is yes, this activity will be destroyed and recreated as in the initial state. To test this scenario, we will rotate the device horizontally. From the **Figure 5**, we can see that the *AdminActivity* triggered many functions: *onPause(), onStop(), onDestroy()*, then it fired the functions just like the same when we opened that activity.



*Figure 5. Observe the logs in the AdminActivity after rotating the device*

In other words, as the activity will be destroyed and recreated as if it was newly created, all the data saved in that activity will be lost. To save the data that the user has interacted with that activity, Kotlin provide us with a method called ***onSaveInstanceState().*** This method will be called between *onPause()* and *onStop().* However, this method only works with the data we made in this specific activity and in the current app session. This method will **not** store the data permanently, which means, when we close and open the app again, the data will still be lost.