Assignment 4 – Mobile Mess

Introduction

Your organization has decided that they want to make an Android application available for students who want to purchase NYU GiftCards. They took the liberty of hiring a contractor to create the application, but the code came back less useful than desired. Though your boss never told you which contracting company was hired, you're pretty sure it as Shoddy Corp's Cut-Rate Contracting. They also created a back-end for the application to interact with, but that was given to another member of your team at work to fix.

Part 1 – Setting up environment

Followed directions given in assignment

1. Downloaded Android Studio
2. Created Virtual Device Pixel 3a with Build R
3. Imported Project GiftcardSite/ into Android Studio
4. Run Virtual Device and Launched AVD emulator and able to open up GiftcardSite

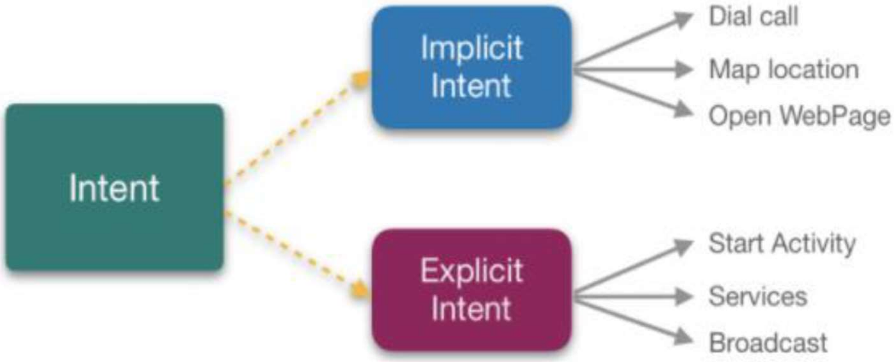
Part 2 – It's all about intent

Android Intent is the message that is passed between components such as activities, content providers, broadcast receiver, services etc. Suppose you want to go from one screen to another screen or communication between applications, this activity in called Intent. For an application to work and navigate thru one screen to another screen within application or from one application to another application intents are used in Android.

Part 2.1 - Intents

Q1. What are two types of Intent?

There are 2 types of Intent



Explicit Intent – An intent is explicit when an action uses or calls class/package or component within its own app to satisfy the need. For example, perform an activity to download a file but action is called within app

Implicit Intent – An intent is implicit when an action calls a component from another app or source to handle it. For example, a user wants to get location and action call google maps to provide need info. This is normally done using (Intent.ACTION_VIEW) as in line # 69 in SecondFragment.kt:

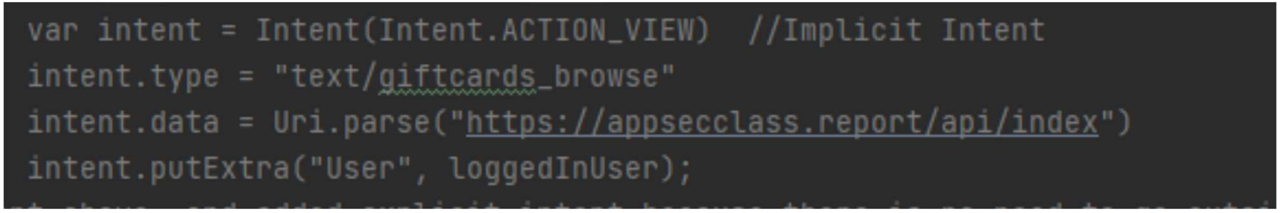
var intent = Intent(Intent.ACTION_VIEW)

Q2. Which of the two types of Intents are more secure?

Explicit Intent is more secure as the actions are handled within an app. Since an implicit intent does not specify a particular application to receive the data, any application can process the intent by using an Intent Filter for that intent. This can allow untrusted applications to obtain sensitive data, cause privilege escalation or denial-of-service

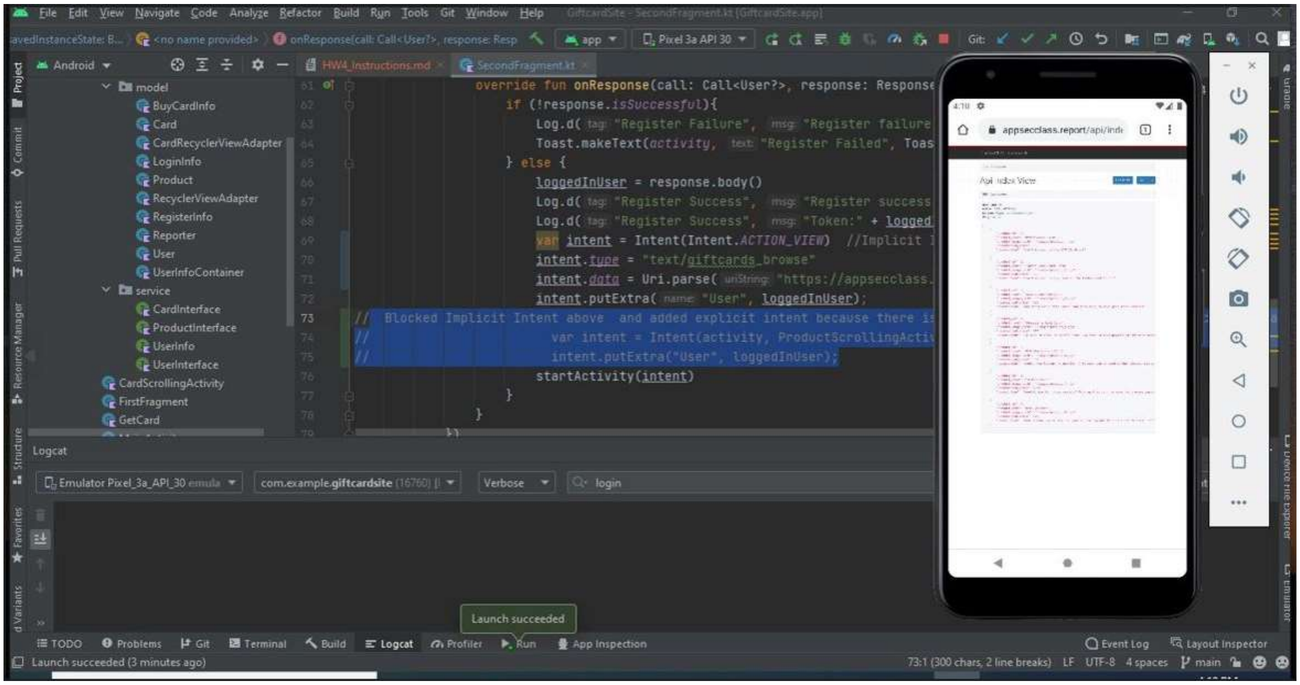
Q3. What type of Intent is shown on lines 69 to 73 of SecondFragment.kt

The intent in lines 69 to 73 is an Implicit Intent. A URL is used to go thru a https service to get the index of products. See below



```
var intent = Intent(Intent.ACTION_VIEW)  //Implicit Intent
intent.type = "text/giftcards_browse"
intent.data = Uri.parse("https://appsecclass.report/api/index")
intent.putExtra("User", loggedInUser);
```

Secondfragment is called by register new user (action/intent) and shows this implicit intent when the user is registered successfully. See below

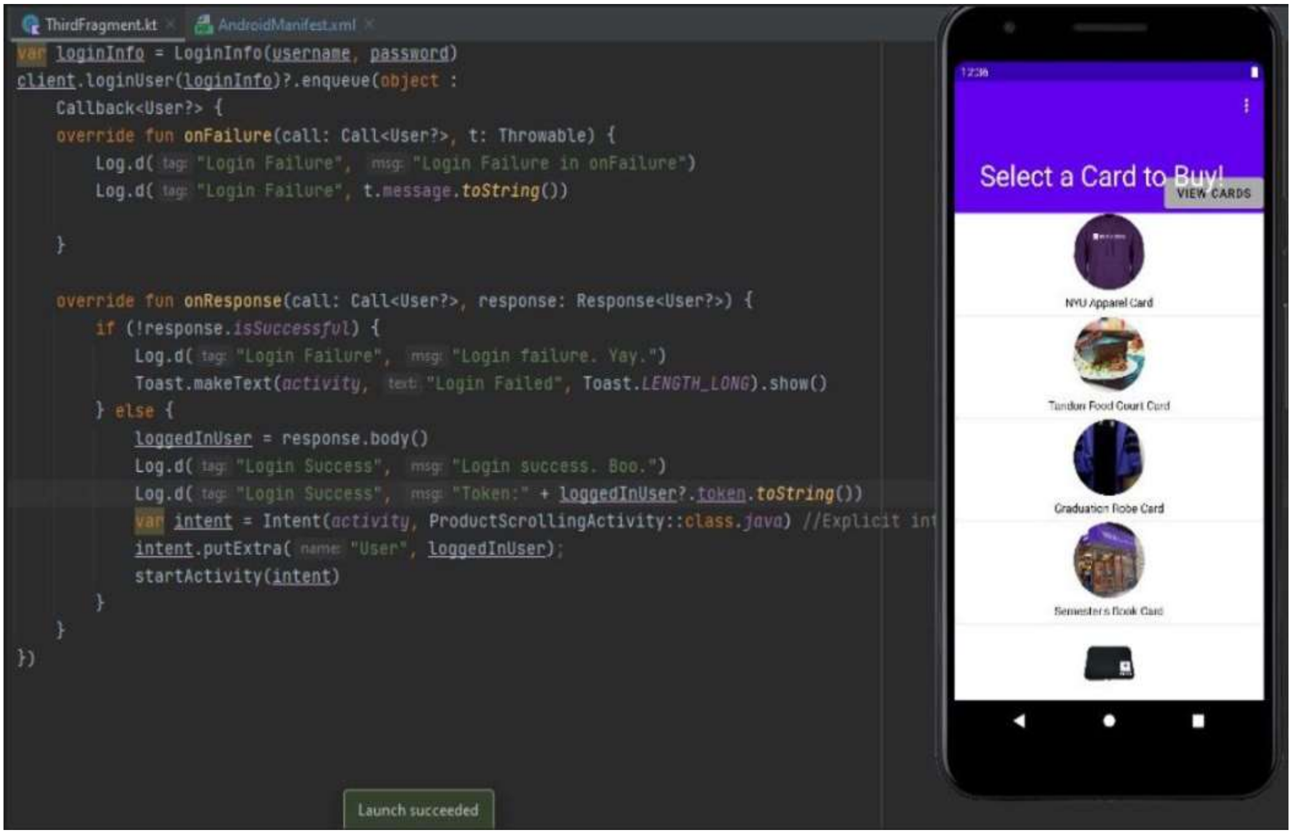


```
61              override fun onResponse(call: Call<User?>, response: Response
62                  if (!response.isSuccessful){
63                      Log.d( tag: "Register Failure",  msg: "Register failure
64                      Toast.makeText(activity,  text: "Register Failed", Toas
65                  } else {
66                      loggedInUser = response.body()
67                      Log.d( tag: "Register Success",  msg: "Register success
68                      Log.d( tag: "Register Success",  msg: "Token:" + logged
69                      var intent = Intent(Intent.ACTION_VIEW)  //Implicit I
70                      intent.type = "text/giftcards_browse"
71                      intent.data = Uri.parse( uriString: "https://appsecclass.
72                      intent.putExtra( name: "User", loggedInUser);
73  //  Blocked Implicit Intent above  and added explicit intent because there is
74  //                  var intent = Intent(activity, ProductScrollingActiv
75  //                  intent.putExtra("User", loggedInUser);
76                      startActivity(intent)
77                  }
78              }
```

Figure stored in images/2ndFragment_Implicit_intent.JPG

Q4. What type of Intent is shown on line 68 to 70 of ThirdFragment.kt

In file ThirdFragment.kt lines 68 to 70 an Explicit intent is used by when ProductScrollingActivity class makes a call within the app to get the list of products. ThirdFragment is called when registered user uses the login screen. The application shows the products list. See below



```kotlin
var loginInfo = LoginInfo(username, password)
client.loginUser(loginInfo)?.enqueue(object :
    Callback<User?> {
    override fun onFailure(call: Call<User?>, t: Throwable) {
        Log.d( tag: "Login Failure", msg: "Login Failure in onFailure")
        Log.d( tag: "Login Failure", t.message.toString())

    }

    override fun onResponse(call: Call<User?>, response: Response<User?>) {
        if (!response.isSuccessful) {
            Log.d( tag: "Login Failure", msg: "Login failure. Yay.")
            Toast.makeText(activity, text: "Login Failed", Toast.LENGTH_LONG).show()
        } else {
            loggedInUser = response.body()
            Log.d( tag: "Login Success", msg: "Login success. Boo.")
            Log.d( tag: "Login Success", msg: "Token:" + loggedInUser?.token.toString())
            var intent = Intent(activity, ProductScrollingActivity::class.java) //Explicit int
            intent.putExtra( name: "User", loggedInUser);
            startActivity(intent)
        }
    }
})
```

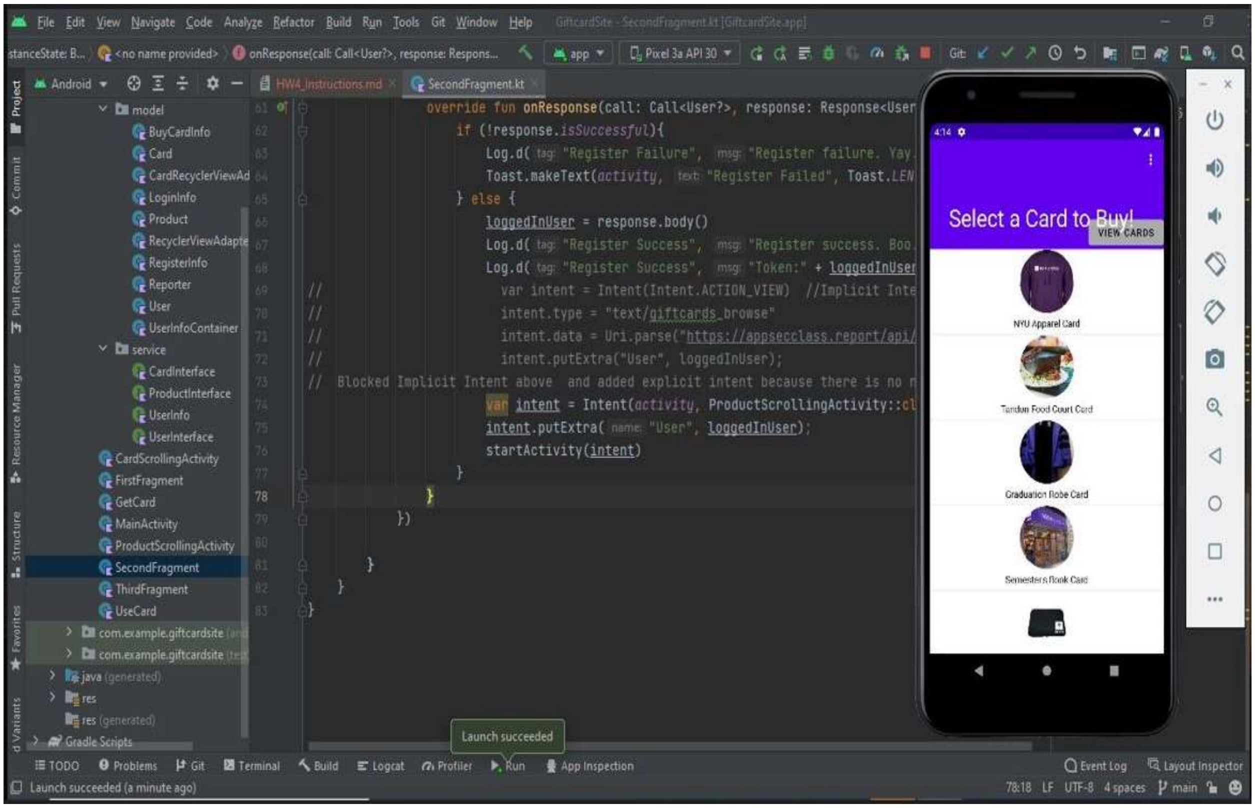
Figure stored in images/3rdFragment_explicit_intent.JPG

Q5. Which of these two Intents is the proper way to do an Intent?

Explicit intent is the proper way to do an Intent as it is more secure. However, sometimes there may be the need to use implicit intents. In those cases, secure practices must be implemented.

As stated above explicit intent are more secure to use. SecondFragment uses implicit intent to get product list.

Using Right Intent -- By looking at the output and clicking on the link used in SecondFragment, I compared the Products View (Same screen for Login workflow called in ThirdFragment), I was able to replace the implicit intent with an explicit intent by calling ProductScrollingActivity class to get the information within the application. The application worked without any issue. The change was also confirmed with registering new user which landed on Products view, similar to login screen using explicit intent as shown below



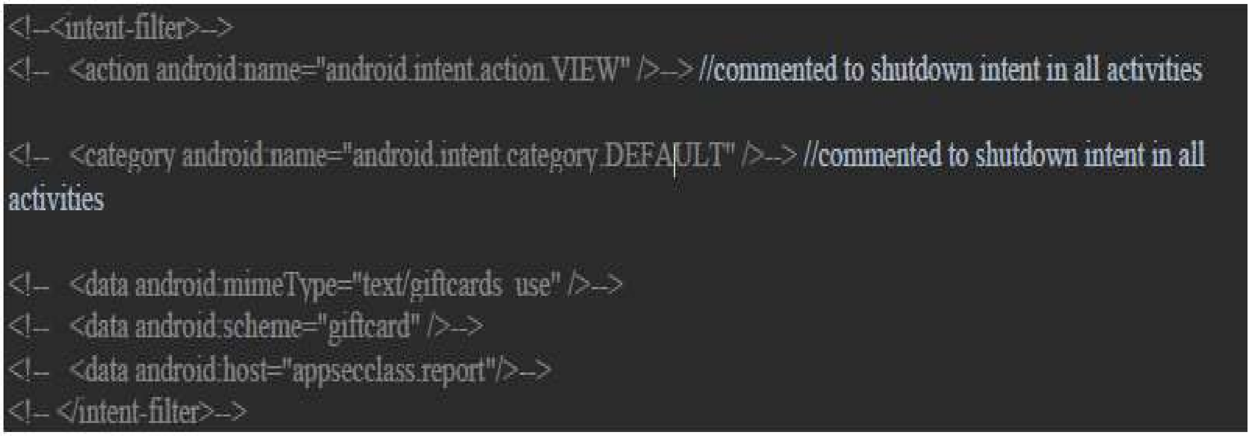
```
            override fun onResponse(call: Call<User?>, response: Response<User
                if (!response.isSuccessful){
                    Log.d( tag: "Register Failure",  msg: "Register failure. Yay.
                    Toast.makeText(activity,  text: "Register Failed", Toast.LEN
                } else {
                    loggedInUser = response.body()
                    Log.d( tag: "Register Success",  msg: "Register success. Boo.
                    Log.d( tag: "Register Success",  msg: "Token:" + loggedInUser
//                    var intent = Intent(Intent.ACTION_VIEW)  //Implicit Inte
//                    intent.type = "text/giftcards_browse"
//                    intent.data = Uri.parse("https://appsecclass.report/api/
//                    intent.putExtra("User", loggedInUser);
//  Blocked Implicit Intent above  and added explicit intent because there is no n
                    var intent = Intent(activity, ProductScrollingActivity::cl
                    intent.putExtra( name: "User", loggedInUser);
                    startActivity(intent)
                }
            }
        })

    }
}
}
```

Figured stored in images/2ndFragment_explicit_intent_chage.JPG

Part 2.2 – Shutting out the world

In order to remove the possibility of other applications using Intents to launch activities of the application, I identified the following lines in the activity-filter section to stop not needed intents in the AndroidManifes.xml. I removed all activities (4) except main activity. The following screen shot is from UseCard activity. Similar lines removed from GetCard, ProductScrolling activity and CardScrollingActivity



```
<!--<intent-filter>-->
<!--   <action android:name="android.intent.action.VIEW" />--> //commented to shutdown intent in all activities

<!--   <category android:name="android.intent.category.DEFAULT" />--> //commented to shutdown intent in all activities

<!--   <data android:mimeType="text/giftcards_use" />-->
<!--   <data android:scheme="giftcard" />-->
<!--   <data android:host="appsecclass.report"/>-->
<!-- </intent-filter>-->
```

Part 3 -- Can you read me out there?

Communication non secure in transit is always vulnerable to attacks and the communication can be exposed. In order to make sure the communication is secure; we can use HTTPS. HTTPS uses TLS (SSL) to encrypt normal HTTP requests and responses, and to digitally sign those requests and responses. The following lines were changed

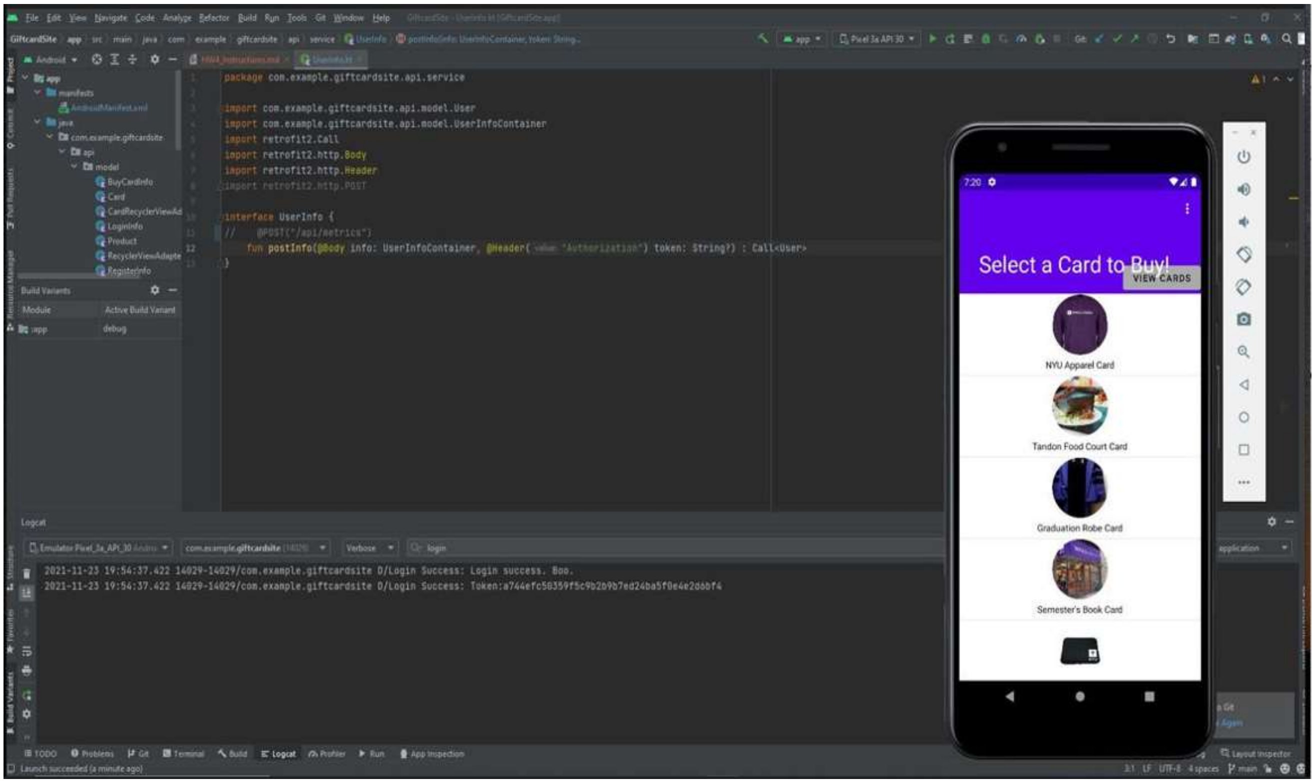
1. SecondFragment.kt – Line 48
2. ThirdFragment.kt –Line 45
3. CardScrollingActivity.kt – Lines 59, 98, and 123
4. ProductScrollingActivity.kt – Lines 61, 101, and 127
5. UseCard.kt – Lines 35 and 43
6. GetCard.kt – Lines 31 and 40
7. CardRecyclerViewAdapter.kt – Line 21
8. RecyclerViewAdapter.kt – Line 23

Application Launched successfully after http to https changes.

Part 4: Oops, was that card yours?

While exploring the application for vulnerabilities, I used Android studio Logcat utility with filter "login" and verbose on option. The following lines were received after a successful login showing token issued to user (also in screenshot)

2022-12-08 19:54:37.422 14029-14029/com.example.giftcardsite D/Login Success: Login success. Boo.
2022-12-08 19:54:37.422 14029-14029/com.example.giftcardsite D/Login Success: Token:a744efc50359f5c9b2b9b7ed24ba5f0e4e2d6bf4



```
package com.example.giftcardsite.api.service

import com.example.giftcardsite.api.model.User
import com.example.giftcardsite.api.model.UserInfoContainer
import retrofit2.Call
import retrofit2.http.Body
import retrofit2.http.Header
import retrofit2.http.POST

interface UserInfo {
//    @POST("/api/metrics")
    fun postInfo(@Body info: UserInfoContainer, @Header( value: "Authorization") token: String?) : Call<User>
}
```

Logcat: Emulator Pixel_3a_API_30 | com.example.giftcardsite | Verbose | login

```
2021-11-23 19:54:37.422 14029-14029/com.example.giftcardsite D/Login Success: Login success. Boo.
2021-11-23 19:54:37.422 14029-14029/com.example.giftcardsite D/Login Success: Token:a744efc50359f5c9b2b9b7ed24ba5f0e4e2d6bf4
```

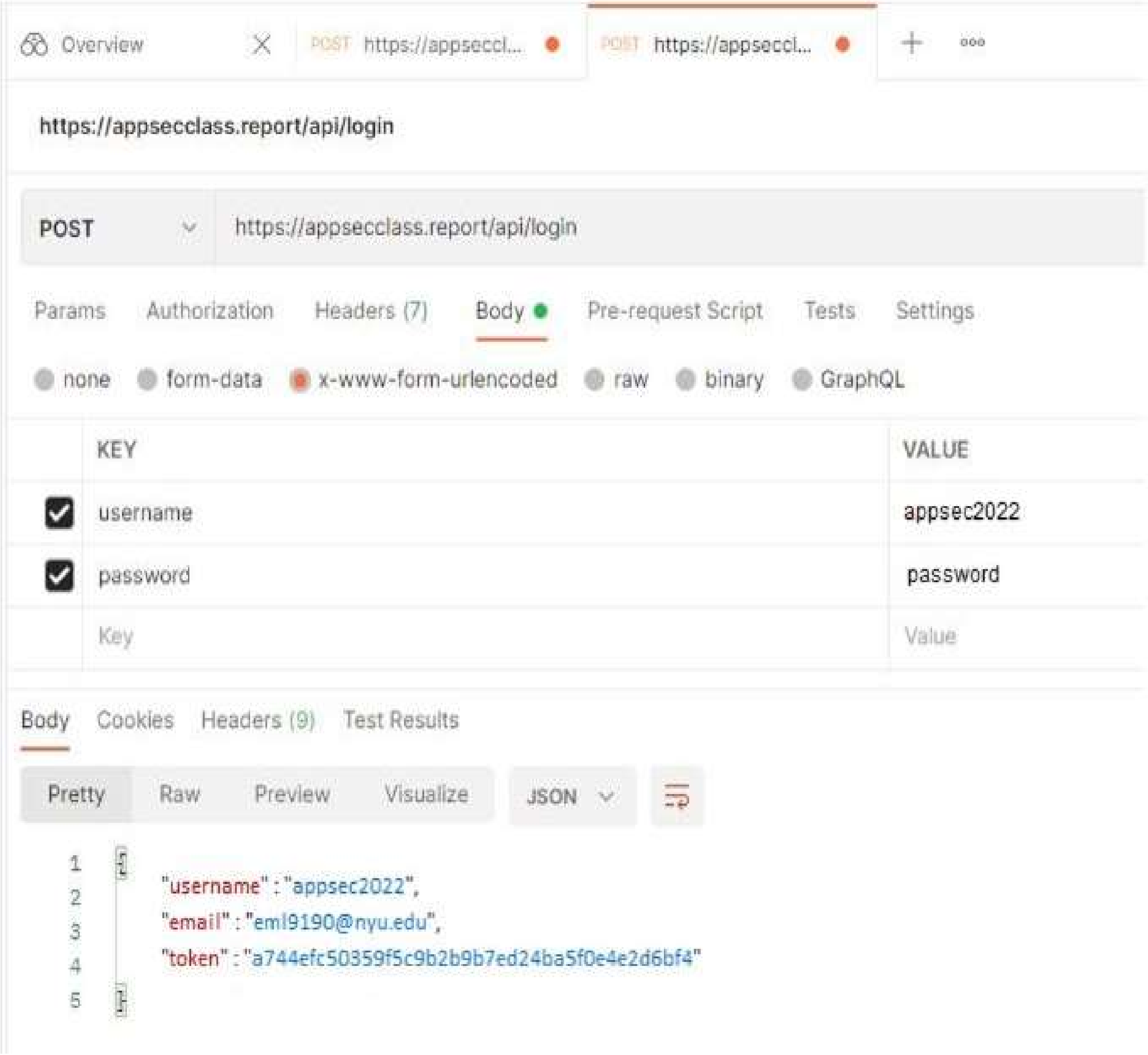
Figured stored in images/logcat_login.JPG

Solution: Logs proved my assumption that the sensitive data is easily accessible. Hence, code which writes to the log must be removed.

By further looking into files UseCard.kt and CarInterface.kt I got the api calls used for card list and card use

Using Postman utility, I was able to get user token by issuing

1. Sending POST command on https://appsecclass.report/api/login -- got token issued to user using body x.www-form-urlencoded (shown below)



https://appsecclass.report/api/login

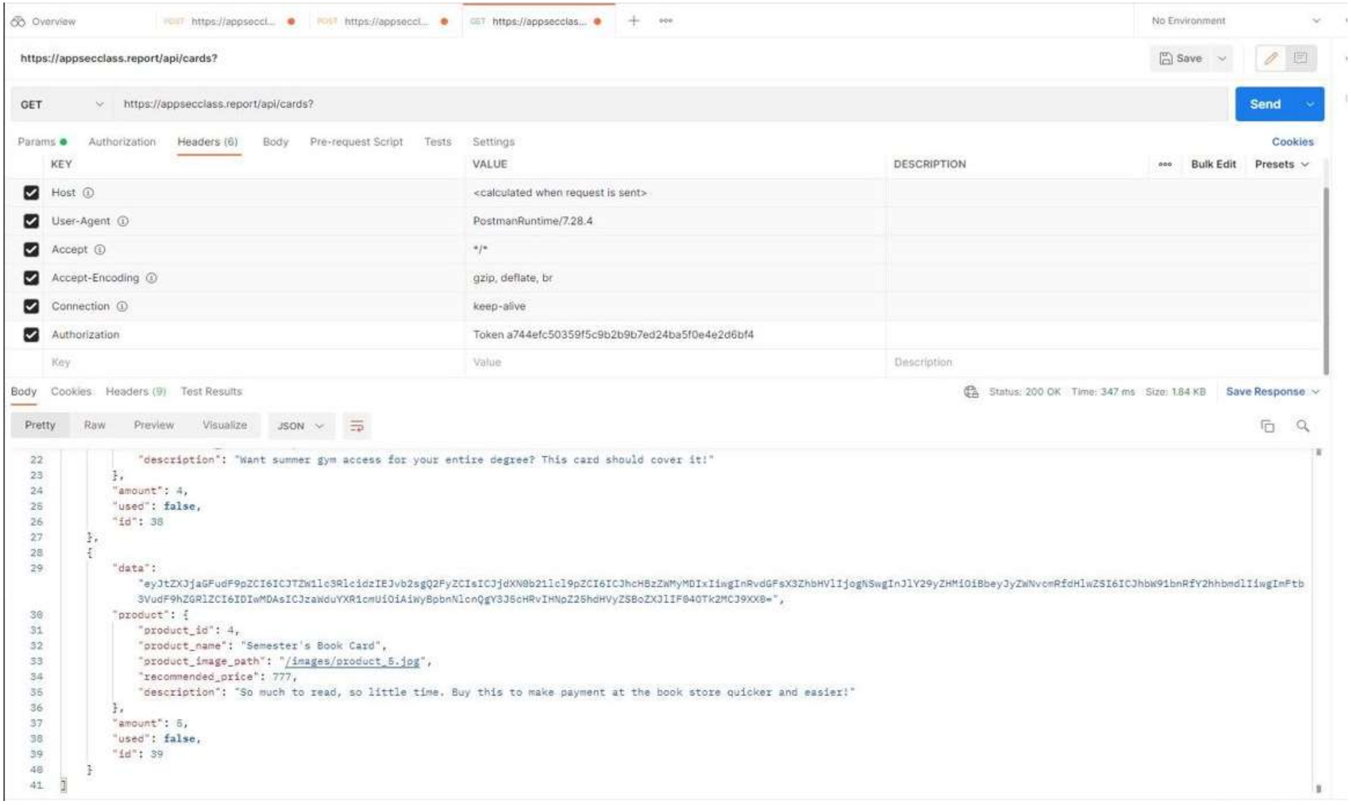
POST https://appsecclass.report/api/login

	KEY	VALUE
☑	username	appsec2022
☑	password	password
	Key	Value

```
{
    "username" : "appsec2022",
    "email" : "eml9190@nyu.edu",
    "token" : "a744efc50359f5c9b2b9b7ed24ba5f0e4e2d6bf4"
}
```

Figure stored in images/usertoken1.JPG

2. Sending GET command on https/appsecclass.report/api/cards? -- got the list of users unused cards using token in header authorization field (as show below)



https://appsecclass.report/api/cards?

GET https://appsecclass.report/api/cards?

Params ● Authorization **Headers (6)** Body Pre-request Script Tests Settings

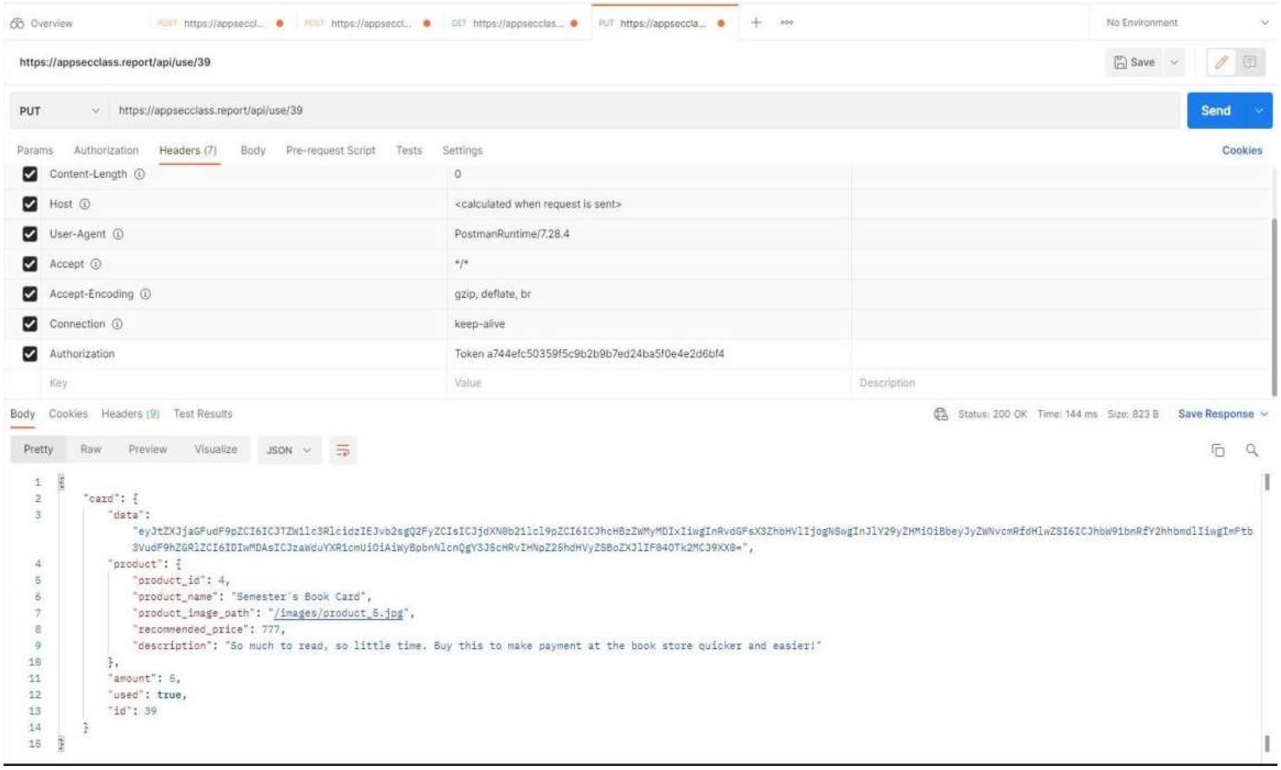
	KEY	VALUE	DESCRIPTION
☑	Host	<calculated when request is sent>	
☑	User-Agent	PostmanRuntime/7.28.4	
☑	Accept	*/*	
☑	Accept-Encoding	gzip, deflate, br	
☑	Connection	keep-alive	
☑	Authorization	Token a744efc50359f5c9b2b9b7ed24ba5f0e4e2d6bf4	
	Key	Value	Description

Body Cookies Headers (9) Test Results — Status: 200 OK Time: 347 ms Size: 1.84 KB

```
            "description": "Want summer gym access for your entire degree? This card should cover it!"
        },
        "amount": 4,
        "used": false,
        "id": 38
    },
    {
        "data":
            "eyJtZXJjaGFudF9pZCI6ICJTZW1lc3RlcidzIEJvb2sgQ2FyZCIsICJjdXN0b21lcl9pZCI6ICJhcHBzZWMyMDIxIiwgInRvdGFsX3ZhbHVlIjogNSwgInJlY29yZHMiOiBbeyJyZWNvcmRfdHlwZSI6ICJhbW91bnRfY2hhbmdlIiwgImFtb
            3VudF9hZGRlZCI6IDIwMDAsICJzaWduYXR1cmUiOiAiWyBpbnNlcnQgY3J5cHRvIHNpZ25hdHVyZSBoZXJlIF040Tk2MCJ9XX0=",
        "product": {
            "product_id": 4,
            "product_name": "Semester's Book Card",
            "product_image_path": "/images/product_5.jpg",
            "recommended_price": 777,
            "description": "So much to read, so little time. Buy this to make payment at the book store quicker and easier!"
        },
        "amount": 5,
        "used": false,
        "id": 39
    }
]
```

Figure stored in images/usercard_existing_cards.JPG

Sending PUT command on https://appsecclass.report/api/use/39 -- able to use card 39 using user's token in header authorization field (shown below)



https://appsecclass.report/api/use/39

PUT https://appsecclass.report/api/use/39

Params Authorization **Headers (7)** Body Pre-request Script Tests Settings

☑	Content-Length	0	
☑	Host	<calculated when request is sent>	
☑	User-Agent	PostmanRuntime/7.28.4	
☑	Accept	*/*	
☑	Accept-Encoding	gzip, deflate, br	
☑	Connection	keep-alive	
☑	Authorization	Token a744efc50359f5c9b2b9b7ed24ba5f0e4e2d6bf4	
	Key	Value	Description

Body Cookies Headers (9) Test Results — Status: 200 OK Time: 144 ms Size: 823 B

```
{
    "card": {
        "data":
            "eyJtZXJjaGFudF9pZCI6ICJTZW1lc3RlcidzIEJvb2sgQ2FyZCIsICJjdXN0b21lcl9pZCI6ICJhcHBzZWMyMDIxIiwgInRvdGFsX3ZhbHVlIjogNSwgInJlY29yZHMiOiBbeyJyZWNvcmRfdHlwZSI6ICJhbW91bnRfY2hhbmdlIiwgImFtb
            3VudF9hZGRlZCI6IDIwMDAsICJzaWduYXR1cmUiOiAiWyBpbnNlcnQgY3J5cHRvIHNpZ25hdHVyZSBoZXJlIF040Tk2MCJ9XX0=",
        "product": {
            "product_id": 4,
            "product_name": "Semester's Book Card",
            "product_image_path": "/images/product_5.jpg",
            "recommended_price": 777,
            "description": "So much to read, so little time. Buy this to make payment at the book store quicker and easier!"
        },
        "amount": 5,
        "used": true,
        "id": 39
    }
}
```

Figure stored in images/usercard_stolen_used.JPG

Reason – When a permanent token is issued, it is stored in db. this token could be used at any time after. Permanent tokens are a risk. Another reason is that unnecessary data exposure can also cause a token exposure

Fix -- Permanent token should not be used; instead, a temporary session-based token should be issued, one per user per session for authorization. Once the session is complete temporary session token should expire and be deleted from system. This will avoid any attacks even if the token is stolen as it will not be useful for another session. Commenting Log.d will not be of much help. The better approach is not to issue and store permanent tokens.

- Proper authentication and authorization access controls on server side api, validating card belongs to user and related token will enforce card use by owner,
  and not to another user.
- Implementing a function to generate A long random card number will make it difficult to guess and prevent stealing
- Token used here is permanent and stored in DB. If this changed to limited time / session cookie will enforce use of username and password. to take it further,
  MFA implementation will enforce strong authentication

Part 5: Privacy is Important

In this section the goal is to remove all privacy invasive code. This is done by removing all metric collecting code, all areas that needlessly interact with sensors, and all permissions that are not needed for the basic functionality of the application (buying, browsing, and using gift cards). I noticed many invasive (privacy) metrics in the files mentioned. I have removed all the code related to the location and the sensors in files mentioned below.

1. **AndroidManifest.xml -** The following lines were removed to remove permissions:

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

2. **UserInfo.kt -** Removed metrics API call

```
// @POST("/api/metrics")
```

3. **CardScrollingActivity.kt** –Removed interaction with SensorEventListener LocationListener class. The following lines were commented out

Line 32

```
class CardScrollingActivity : AppCompatActivity() { //, SensorEventListener, LocationListener {
```

Lines 40 to 47

```
// val locationPermissionCode = 2
// var locationManager = getSystemService(Context.LOCATION_SERVICE) as LocationManager
// if ((ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED)) {
// ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
locationPermissionCode)
// }
// locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 5000, 5f, this)
// sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
// mAccel = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
```

Lines 96 to 160

```
// override fun onLocationChanged(location: Location) {
// var userInfoContainer = UserInfoContainer(location, null, loggedInUser?.token)
// var builder: Retrofit.Builder =
Retrofit.Builder().baseUrl("https://appsecclass.report").addConverterFactory(
// GsonConverterFactory.create())
// var retrofit: Retrofit = builder.build()
// var client: UserInfo = retrofit.create(UserInfo::class.java)
// client.postInfo(userInfoContainer, loggedInUser?.token)?.enqueue(object: Callback<User?> {
// override fun onFailure(call: Call<User?>, t: Throwable) {
// Log.d("Metric Failure", "Metric Failure in onFailure")
// Log.d("Metric Failure", t.message.toString())
//
// }
//
// override fun onResponse(call: Call<User?>, response: Response<User?>) {
// if (!response.isSuccessful) {
// Log.d("Metric Failure", "Metric failure. Yay.")
// } else {
// Log.d("Metric Success", "Metric success. Boo.")
// Log.d("Metric Success", "Token:${userInfoContainer.token}")
// }
// }
// })
// }
//
// override fun onSensorChanged(event: SensorEvent?) {
// if (event != null) {
// var userInfoContainer = UserInfoContainer(null, event.values[0].toString(), loggedInUser?.token)
// var builder: Retrofit.Builder =
Retrofit.Builder().baseUrl("https://appsecclass.report").addConverterFactory(
// GsonConverterFactory.create())
// var retrofit: Retrofit = builder.build()
// var client: UserInfo = retrofit.create(UserInfo::class.java)
// client.postInfo(userInfoContainer, loggedInUser?.token)?.enqueue(object: Callback<User?> {
// override fun onFailure(call: Call<User?>, t: Throwable) {
// Log.d("Metric Failure", "Metric Failure in onFailure")
// Log.d("Metric Failure", t.message.toString())
//
// }
//
// override fun onResponse(call: Call<User?>, response: Response<User?>) {
// if (!response.isSuccessful) {
// Log.d("Metric Failure", "Metric failure. Yay.")
// } else {
```

```
// Log.d("Metric Success", "Metric success. Boo.")
// Log.d("Metric Success", "Token:${userInfoContainer.token}")
// }
// }
// })
// }
// }
//
// override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {
// return
// }
//
// override fun onResume() {
// super.onResume()
// mAccel?.also { accel ->
// sensorManager.registerListener(this, accel, SensorManager.SENSOR_DELAY_NORMAL)
// }
// }
//
// override fun onPause() {
// super.onPause()
// sensorManager.unregisterListener(this)
// }
```

4. **ProductScrollingActivity.kt - **Removed interaction with SensorEventListener LocationListener class

Line 33

```
class CardScrollingActivity : AppCompatActivity() { //, SensorEventListener, LocationListener {
```

Also Removed Sensor and location related code similar to code in **CardScrollingActivity.kt** file