CONTROLE SIM CARD À NÍVEL AOSP

André Rondi, Andreina Oliveira, Sthefanye Guimarães, Wesllen Costa

Universidade do Estado do Amazonas (UEA)
Escola Superior de Tecnologia (EST)
Manaus – AM – Brazil

Projeto Hefesto - Hands On

{Rondi,André}adcr.hef4@uea.edu.br,{Oliveira,Andreina}adso.hef4@uea.edu.br, {Guimarâes,Sthefanye}sgo.hef4@uea.edu.br, {Costa,Wesllen}wvc.hef4@uea.edu.br

Abstract. *Aiming to include a security mechanism for the SIM Card natively, the project in question uses AOSP as the basis for creating a custom ROM for Android 9.0 (Pie). The ROM in question has modifications in the telephony layer to identify the insertion and removal of SIM Cards, in addition to storing SIM data. Through the security mechanism implemented in the AOSP, the APP created in the "SIM Card" project, monitors the input and output of SIM's, presenting the user with the card's history, as well as identifying data such as ICCID, Number, Operator, among others.*

Resumo. *Visando incluir um mecanismo de segurança para o SIM Card de forma nativa, o projeto em questão utiliza o AOSP como base para a criação de uma ROM personalizada para o Android 9.0 (Pie). A ROM em questão conta com modificações na camada de telefonia para identificação de inserção e remoção de SIM Cards, além de armazenar os dados SIM Card. Através do mecanismo de segurança implementado no AOSP, o APP criado no projeto "SIM Card", monitora a entrada e saída de SIM 's, apresenta ao usuário o histórico dos cartões, além de identificar dados como ICCID, Número, Operadora, entre outros.*

1. Android Open Source Project (AOSP)

Buscando identificar uma API responsável por tratar os dados do SIM Card, na camada de telefonia do AOSP foi identificado a classe “Telephony Manager”. Analisando a classe em questão, foi identificado os seguintes métodos:

- getSimSerialNumber() que retorna o ICCID;
- getLine1Number() que retorna o número de telefone;
- getSimCarrierIdName() que retorna a operadora;
- “getSimState() que retorna o estado do SIM atual.

```
@RequiresPermission(android.Manifest.permission.READ_PHONE_STATE)
public String getSimSerialNumber(int subId) {
    try {
        IPhoneSubInfo info = getSubscriberInfo();
        if (info == null)
            return null;
        return info.getIccSerialNumberForSubscriber(subId, mContext.getOpPackageName());
    } catch (RemoteException ex) {
        return null;
    } catch (NullPointerException ex) {
        // This could happen before phone restarts due to crashing
        return null;
    }
}
```

Figure 1. AOSP - Telephony Manager - getSimSerialNumber()

```
public String getLine1Number(int subId) {
    String number = null;
    try {
        ITelephony telephony = getITelephony();
        if (telephony != null)
            number = telephony.getLine1NumberForDisplay(subId, mContext.getOpPackageName());
    } catch (RemoteException ex) {
    } catch (NullPointerException ex) {
    }
    if (number != null) {
        return number;
    }
    try {
        IPhoneSubInfo info = getSubscriberInfo();
        if (info == null)
            return null;
        return info.getLine1NumberForSubscriber(subId, mContext.getOpPackageName());
    } catch (RemoteException ex) {
        return null;
    } catch (NullPointerException ex) {
        // This could happen before phone restarts due to crashing
        return null;
    }
}
```

Figure 2. AOSP - Telephony Manager - getLine1Number()

```
public CharSequence getSimCarrierIdName() {
    try {
        ITelephony service = getITelephony();
        if (service != null) {
            return service.getSubscriptionCarrierName(getSubId());
        }
    } catch (RemoteException ex) {
        // This could happen if binder process crashes.
    }
    return null;
}
```

Figure 3. AOSP - Telephony Manager - getSimCarrierIdName()

```
@SystemApi
public int getSimCardState() {
    int simCardState = getSimState();
    switch (simCardState) {
        case SIM_STATE_UNKNOWN:
        case SIM_STATE_ABSENT:
        case SIM_STATE_CARD_IO_ERROR:
        case SIM_STATE_CARD_RESTRICTED:
            return simCardState;
        default:
            return SIM_STATE_PRESENT;
    }
}
```

Figure 4. AOSP - Telephony Manager - getSimCardState()

2. Monitoramento das Entrada e Saídas de SIM Card's AOSP

Objetivando monitorar a entrada e saída de SIM Cards, o método observerData(), criado na classe Telephony Manager, utiliza o método getSimState() para auxiliar nesse monitoramento.

Primeiramente, é associado a variável oldSIM o estado inicial do SIM Card. Posteriormente, com auxílio de um loop, é novamente feita a leitura do estado do SIM, desta vez, para a variável newSIM. Enquanto o valor de newSIM for igual a oldSIM significa que os SIM's são os mesmos, portanto nada é feito e novamente o loop volta a buscar o estado do newSIM até que haja diferença. Havendo diferença entre o newSIM e o oldSIM é afirmado que há uma troca de SIM's.

Utilizando a estrutura switch case, é passado por parâmetro o newSIM e analisado se o newSIM possui estado "Absent" (ausente) ou "Ready" (pronto). Sendo Absent, é enviado o log "SIM CARD REMOVIDO" e o método retorna a string "Removed". Sendo Ready é enviado o log "SIM CARD INSERIDO", chamado o método WriteTxt() e o método observerData() retorna a string "Inserted".

```
public String observerData() {
    int oldSIM = getSimState();

    while(true){
        int newSIM = getSimState();
        if (oldSIM != newSIM){
            oldSIM = newSIM;
            switch(newSIM){
                case TelephonyManager.SIM_STATE_ABSENT:
                    Rlog.v(TAGT06, "SIM CARD REMOVIDO");
                    return "Removed";
                case TelephonyManager.SIM_STATE_READY:
                    Rlog.v(TAGT06, "SIM CARD INSERIDO");
                    WiriteTxt();
                    return "Inserted";
            }
        }
    }
}
```

Figure 5. AOSP - Telephony Manager - observerData()

3. Armazenamento de dados AOSP

Além do monitoramento dos SIM Card, outro ponto que o nosso app buscou atender foi a persistência dos dados analisados. Para fazer isso de forma segura buscamos dar a responsabilidade do registro dos dados para a camada do AOSP. Utilizamos a classe Telephony Manager. Os métodos criados foram o checkPathDirectory, WriteTxt e o readFile. Utilizando as propriedades da classe File, criamos um diretório no qual geramos um txt para acumular os logs do aplicativo. Como o caminho do diretório não por conta do Context podemos acessar esse txt com as informação dentro da aplicação

```java
public File checkPathDirectory() {
        File path = new File(mContext.getFilesDir(), "LogTxt");
        if (!path.exists()){
            path.mkdirs();
        }
        Rlog.v(TAGT06, "Arquivo criado " + path);
        return path;
}

public void WiriteTxt(){
    try{
        File txtFile = new File(checkPathDirectory(), "LogSincard");
        FileWriter writer = new FileWriter(txtFile, true);
        PrintWriter printWriter = new PrintWriter(writer);

        String number_txt = getLine1Number();
        String iccid_txt = getSimSerialNumber();
        CharSequence operator_txt = getSimCarrierIdName();
        printWriter.append("Operadora: " + operator_txt + "\nNúmero: " + number_txt + "\nICCID: " + iccid_txt);
        printWriter.flush();
        printWriter.close();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}

public String readFile() {
    File fileEvents = new File(checkPathDirectory()+"/LogSincard");
    StringBuilder text = new StringBuilder();
    try {
        BufferedReader br = new BufferedReader(new FileReader(fileEvents));
        String line = "";
        while ( (line = br.readLine()) != null){
            text.append(line);
            text.append('\n');
        }
        br.close();
    } catch (IOException e) {
        throw new RuntimeException(e);
     }
    String result = text.toString();
    return result;
}
```

Figure 6. AOSP - Telephony Manager - Armazenamento

4. Permissões do App

A maioria dos desenvolvedores Android já desenvolveu algum aplicativo que precisou utilizar recursos como o serviço de telefonia ou acessar os arquivos do dispositivo do usuário. Para ter acesso a esses recursos, o SDK do Android exige que o usuário libere o acesso a eles antes que o app possa utilizá-los. Geralmente, esse pedido de permissão é exibido ao usuário no formato de um Dialog, para que ele selecione se permite ou não o uso.

No nosso app surgiu a necessidade de incluir uma solicitação de permissão ao usuário para que ele tenha acesso para gerenciar os serviços de telefonia do aparelho, pois sem esta solicitação o app funcionava da seguinte forma:

1º Ao clicar no ícone do aplicativo, o usuário era interceptado por essa tela, na qual diz que o app apresenta falhas e ele teria que ir para a tela de informações.

2º Na tela de informações nas configurações do aparelho, ele clicaria em permissões.

3º E por fim habilitaria manualmente a funcionalidade.

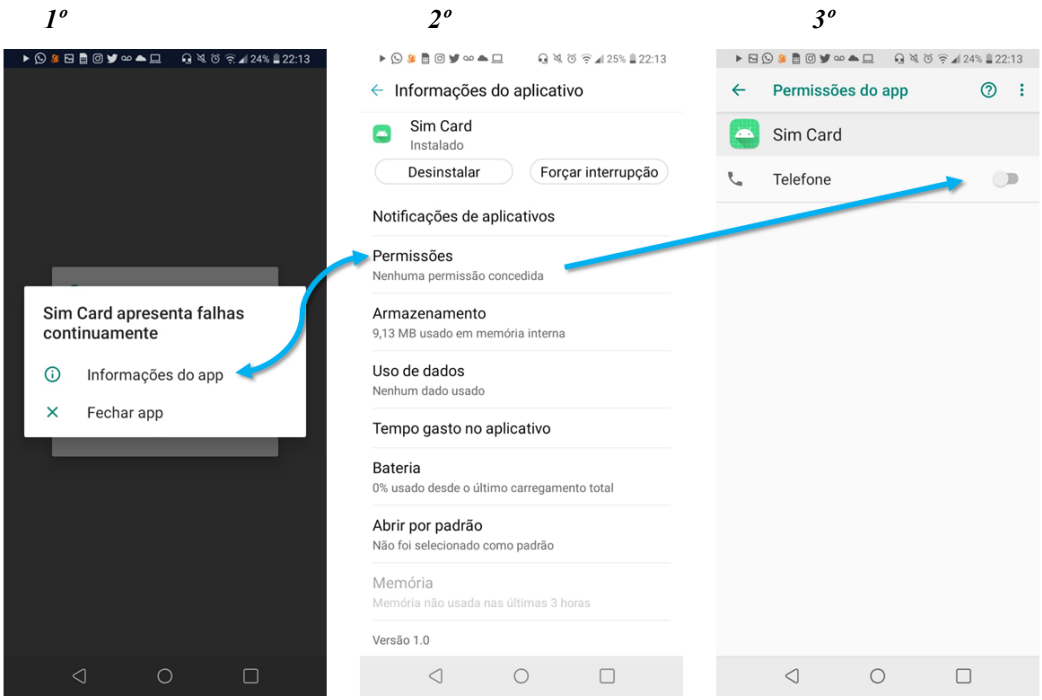
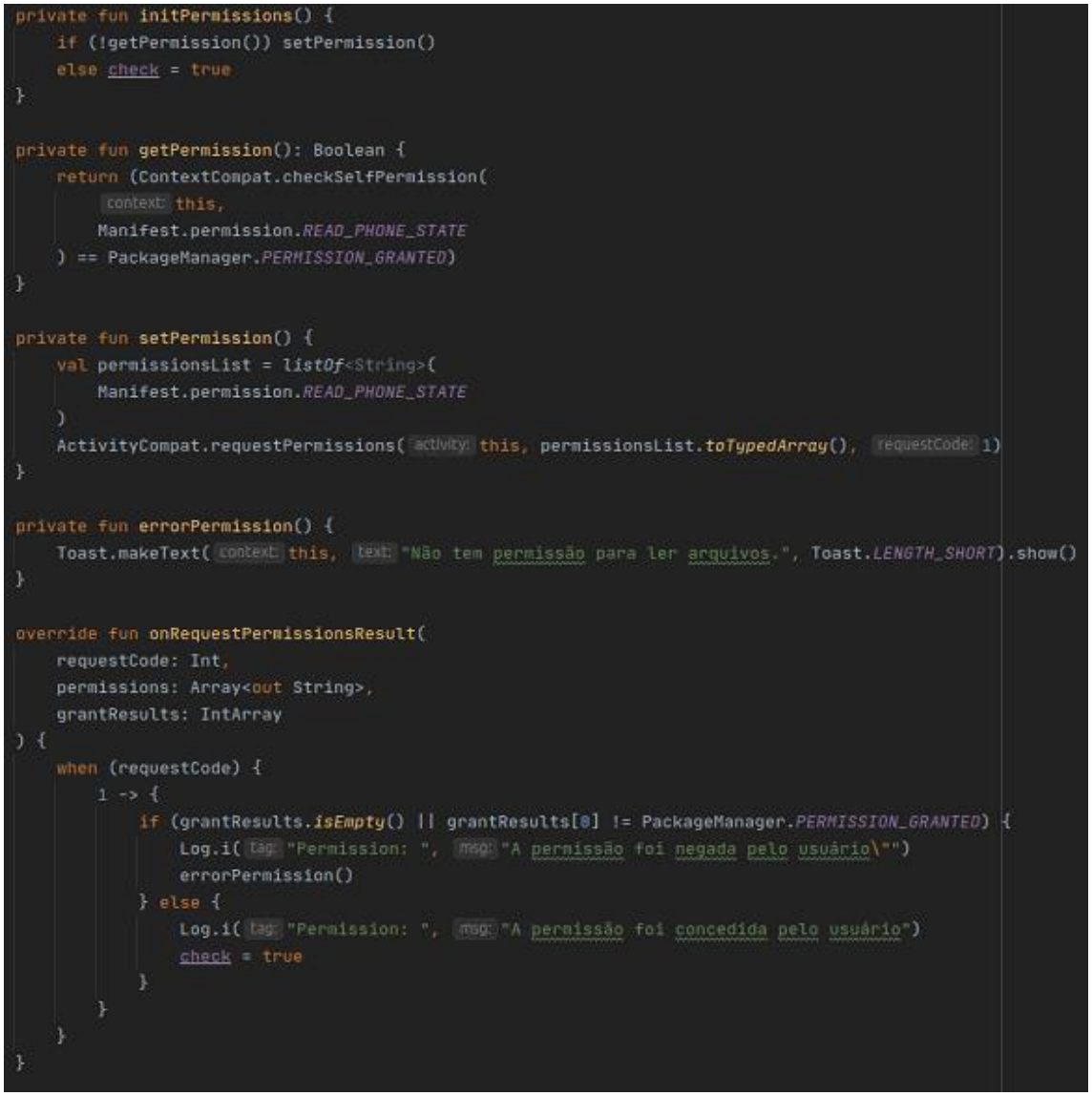


Figure 7. APP - Permissões do APP

Devido a isto, decidimos incluir a solicitação da tela de permissões. No Android Studio, criamos 3 métodos iniciais para setar a solicitação:

- initPermissions() - que somente se inicia se caso o usuário clique em “Permitir”, se não ele faz um check e insere logs e um toast de que não foi concedida a permissão.
- getPermission() - retorna a checagem da permissão no AndroidManifest.xml
- setPermisson() - seta a solicitação.
- errorPermisson() - envia um toast na tela com uma mensagem.
- onRequestPermissonResults() - são os resultados das permissões, se caso o usuário permitir ou não, e chama o errorPermisson se caso ele não permitir.

Na imagem abaixo é possível ver como ficou construído todo o código para solicitar a permissão.



```
private fun initPermissions() {
    if (!getPermission()) setPermission()
    else check = true
}

private fun getPermission(): Boolean {
    return (ContextCompat.checkSelfPermission(
        context: this,
        Manifest.permission.READ_PHONE_STATE
    ) == PackageManager.PERMISSION_GRANTED)
}

private fun setPermission() {
    val permissionsList = listOf<String>(
        Manifest.permission.READ_PHONE_STATE
    )
    ActivityCompat.requestPermissions( activity: this, permissionsList.toTypedArray(), requestCode: 1)
}

private fun errorPermission() {
    Toast.makeText( context: this, text: "Não tem permissão para ler arquivos.", Toast.LENGTH_SHORT).show()
}

override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
) {
    when (requestCode) {
        1 -> {
            if (grantResults.isEmpty() || grantResults[0] != PackageManager.PERMISSION_GRANTED) {
                Log.i( tag: "Permission: ", msg: "A permissão foi negada pelo usuário\"")
                errorPermission()
            } else {
                Log.i( tag: "Permission: ", msg: "A permissão foi concedida pelo usuário")
                check = true
            }
        }
    }
}
```

Figure 8. Código no Android Studio - Permissões do APP

5. Detalhes do SIM Card

A página inicial do aplicativo conta com os detalhes do SIM Card, sendo o ICCID, Número, Operadora, x, y e z. Os dados em questão são consultados através de métodos à nível de APP que consulta os métodos nativos da classe Telephony Manager.

```kotlin
@SuppressLint( ...value: "MissingPermission")
fun getDetails(): String {
    val telephonyManager = this.getSystemService(Context.TELEPHONY_SERVICE) as TelephonyManager
    val d1 = telephonyManager.simCarrierId
    val d3 = telephonyManager.simCountryIso
    val d4 = telephonyManager.simState
    return "ID Operadora: $d1 \nPais: $d3 \nStatus: $d4"
}

@SuppressLint( ...value: "MissingPermission")
fun getNumber(): String {
    val telephonyManager = this.getSystemService(Context.TELEPHONY_SERVICE) as TelephonyManager
    val number = telephonyManager.lineINumber
    Log.i(TAG,  msg: "Numero: $number")
    return "$number"
}

@SuppressLint( ...value: "MissingPermission")
fun getIccId(): String {
    val telephonyManager = this.getSystemService(Context.TELEPHONY_SERVICE) as TelephonyManager
    val iccid = telephonyManager.simSerialNumber
    Log.i(TAG,  msg: "ICCID: $iccid")

    return "$iccid"
}

@SuppressLint( ...value: "MissingPermission")
fun getOperator(): CharSequence? {
    val telephonyManager = this.getSystemService(Context.TELEPHONY_SERVICE) as TelephonyManager
    val operator = telephonyManager.simCarrierIdName
    Log.i(TAG,  msg: "Operadora: $operator")

    return "$operator"
}
```

Figure 9. APP - Android Studio - Detalhes do SIM Card

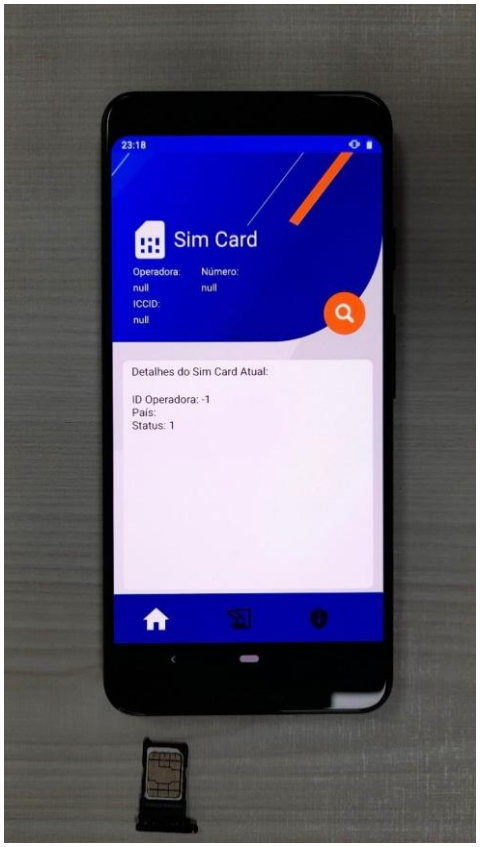


Figure 10. APP - Pixel 3 - Detalhes do SIM Card

6. Histórico dos SIM's

Através do método readFile() criado na telephony manager, é possível retornar para a activity histórico do aplicativo os dados dos SIM Cards que passaram pelo dispositivo.

Para isso, foi criado um TextView para receber esses dados. O TextView está associado a variável "historico" que tentará receber o string retornado em readFile(), contudo, caso o retorno seja vazio, receberá "Sem dados históricos".

```
try {
    historico.text = txtRead()
} catch (e: Exception){
    historico.text = ("Sem dados históricos")
}
```

Figure 11. APP - Android Studio - Histórico dos SIM's

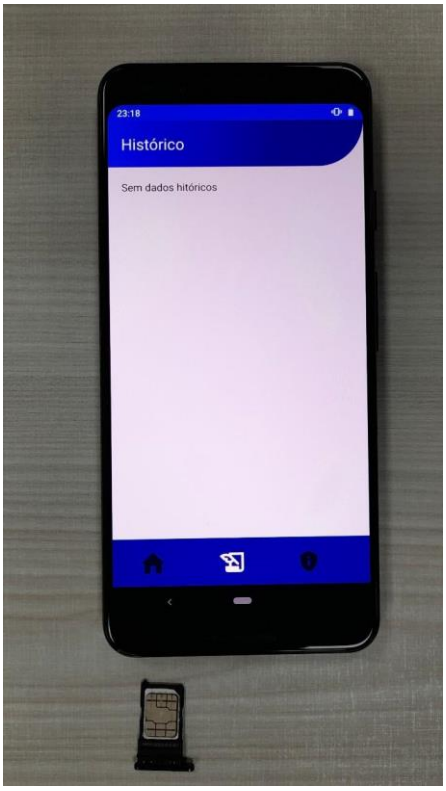


Figure 12. APP - Pixel 3 - Histórico dos SIM's

7. Dicas de Segurança para o Usuário

Há inúmeras armadilhas que o SIM Card está exposto. É necessário expô-las ao usuário para prevenir danos. Portanto, a activity Dicas de Segurança visa orientar o usuário quanto a esses possíveis riscos.

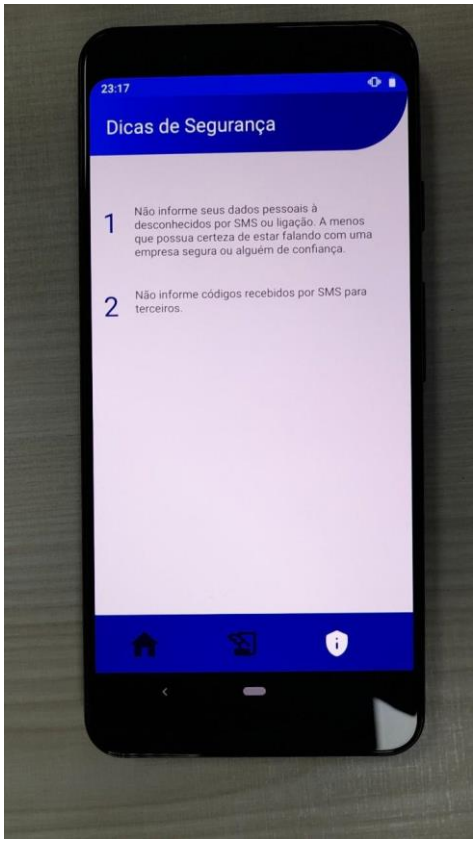


Figure 13. APP - Pixel 3 - Dicas de segurança para o usuário.

8. APP Serviço

Para garantir o monitoramento dos SIM's, é necessário que os principais métodos do aplicativo sejam executados independentemente do APP estar aberto ou não. Para isso, foi criado um serviço para execução constante do método observerData().

De acordo com o retorno trazido em observerData(), inserted ou removed, é chamado o método correspondente de notificação.

```
override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
    showLog("OnStartCommand")
    val telephonyManager = this.getSystemService(Context.TELEPHONY_SERVICE) as TelephonyManager
    val loop = true

    val runnable = Runnable {
        while (loop){
            var simState = telephonyManager.observerData()
            createNotificationChannel()
            if (simState == "Inserted"){
                sendNotificationInsert()
            }
            if (simState == "Removed"){
                sendNotificationRemove()
            }
        }
    }
    val thread =  Thread(runnable)
    thread.start()
    return super.onStartCommand(intent, flags, startId)
}
```

Figure 14. APP - Android Studio - APP Serviço

9. Notificação

Para o ciência do usuário quanto ao estado do SIM Card, é necessário que a aplicação envie notificações. Há duas notificações possíveis, sendNotificantionInserted() que é chamada quando um cartão é identificado e sendNotificantionRemoved() , chamada quando um cartão é removido.

Ambas as notificações, quando clicadas, direcionam para a activity principal do projeto. A tela de detalhes.

```
private fun sendNotificationInsert() {
    val intent = Intent( packageContext: this, MainActivity::class.java).apply {  this: Intent
        flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
    }
    val pendingIntent: PendingIntent = PendingIntent.getActivity( context: this,  requestCode: 0, intent,  flags: 0)

    val builder = NotificationCompat.Builder( context: this, CHANNEL_ID)
        .setSmallIcon(R.drawable.ic_notification)
        .setContentTitle("SIM Card Identificado")
        .setContentText("Veja detalhes do seu novo SIM")
        .setContentIntent(pendingIntent)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)

    with(NotificationManagerCompat.from( context: this)) {  this: NotificationManagerCompat
        notify(notificationId, builder.build())
    }
}
```

Figure 15. APP - Android Studio - Notificação

```
private fun sendNotificationRemove() {
    val intent = Intent( packageContext: this, MainActivity::class.java).apply { this: Intent
        flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
    }
    val pendingIntent: PendingIntent = PendingIntent.getActivity( context: this,  requestCode: 0, intent,  flags: 0)

    val builder = NotificationCompat.Builder( context: this, CHANNEL_ID)
        .setSmallIcon(R.drawable.ic_notification)
        .setContentTitle("SIM Card Removido")
        .setContentText("Seu SIM Card foi ejetado.")
        .setContentIntent(pendingIntent)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)

    with(NotificationManagerCompat.from( context: this)) { this: NotificationManagerCompat
        notify(notificationId, builder.build())
    }
}
```

Figure 16. APP - Android Studio - Notificação

References

Android Tutorial (Kotlin)-38-Service. CodeAndroid, 2018. Disponível em: https://www.youtube.com/watch?v=whceyE2ea2c. Acesso em: 24 jun. 2021.

Compilar e “flashar” o AOSP no Pixel 3. Ava UEA, 2021. Disponível em: https://avauea.uea.edu.br/mod/forum/discuss.php?d=16313. Acesso em: 24 jun. 2021.

Creating Notifications in Android Studio 2020 (Kotlin). Code Palace, 2020. Disponível em: https://www.youtube.com/watch?v=B5dgmvbrHgs&t=325s. Acesso em: 24 jun. 2021.

Cross Reference: TelephonyManager.java. Android XRef, 2018. Disponível em: http://androidxref.com/9.0.0_r3/xref/frameworks/base/telephony/java/android/telephony/TelephonyManager.java. Acesso em: 24 jun. 2021.

Context. Android Developers, 2021. Disponível em: https://developer.android.com/reference/kotlin/android/content/Context. Acesso em: 24 jun. 2021.

Visão geral do armazenamento de dados e arquivos. Android Developers, 2021. Disponível em: https://developer.android.com/training/data-storage. Acesso em: 24 jun. 2021.

TelephonyManager. Android Developers, 2021. Disponível em: https://developer.android.com/reference/android/telephony/TelephonyManager. Acesso em: 24 jun. 2021.

Permissões do Android. Android Developers, 2021. Disponível em: https://developer.android.com/guide/topics/permissions/overview?hl=pt-br. Acesso em: 24 jun. 2021.