Aplicações Móveis

Memorando – Gravador de Chamadas

Coordenação de Engenharia Informática

Departamento de Engenharias e Tecnologias

Instituto Superior Politécnico de Tecnologias e Ciências

Nome : Marcelo Rocha - 20210032

Memorando Explicativo do Código: Aplicativo de Registro de Chamadas

1. **Objetivo do Código**

O código tem como objetivo criar um aplicativo Android que registra todas as chamadas recebidas e efetuadas, salvando os detalhes no armazenamento interno do dispositivo. O aplicativo solicita permissões para acessar o estado do telefone e o histórico de chamadas e armazena essas informações em um arquivo de texto chamado `call_logs.txt`. O usuário pode gerar o histórico das chamadas e visualizá-lo dentro do app.

2. **Estrutura Geral do Código**

O código está dividido em duas classes principais:

- **MainActivity**: Gerencia a interface do usuário e as permissões necessárias.

- **MyCallsReceiver**: Monitora o estado do telefone e registra as chamadas recebidas e efetuadas.

---

3. **Explicação Detalhada das Partes do Código**

MainActivity.java

A `MainActivity` é responsável por gerenciar a interface do usuário e as permissões do app. Vamos analisar suas partes principais:

- **Declaração de Constantes e Variáveis**:

  ```java
  private static final int PHONE_STATUS_REQUEST_CODE = 1;
  private TextView historico;
  ```

  - `PHONE_STATUS_REQUEST_CODE`: Um código que identifica a solicitação de permissão para acessar o estado do telefone.

  - `historico`: Um campo `TextView` que será usado para exibir o histórico de chamadas armazenado.

- **Método onCreate(Bundle savedInstanceState):**

  ```java
  protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      EdgeToEdge.enable(this);
      setContentView(R.layout.activity_main);
      historico = findViewById(R.id.historico);
  }
  ```

  - Esse método inicializa a atividade e define o layout para `activity_main.xml`. Aqui, a `TextView` `historico` é associada ao layout.

  - O `EdgeToEdge.enable(this)` é uma função opcional usada para modificar a aparência visual da interface, permitindo que a interface ocupe a tela toda, incluindo a área de status.

- **Método gerarHistorico(View v):**

  ```java
  public void gerarHistorico(View v) {
      FileInputStream fis = null;
      Scanner scanner = null;
      StringBuilder sb = new StringBuilder();
      try {
          fis = openFileInput("call_logs.txt");
          scanner = new Scanner(fis);
```

```
            while (scanner.hasNextLine()) {
                sb.append(scanner.nextLine());
            }
            Toast.makeText(this, "File read", Toast.LENGTH_SHORT).show();
        } catch (FileNotFoundException e) {
            Toast.makeText(MainActivity.this, "file not found", Toast.LENGTH_SHORT).show();
        } finally {
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                    Log.d("FileExplorer", "Close error.");
                }
            }
            if (scanner != null) {
                scanner.close();
            }
        }
        historico.setText(sb.toString());
    }
```

- Esse método é chamado quando o usuário clica no botão para gerar o histórico.

- Ele abre o arquivo `call_logs.txt` do armazenamento interno e lê linha por linha, usando um `Scanner`.

- O conteúdo do arquivo é exibido na `TextView` `historico`. Caso o arquivo não exista, um aviso de erro é mostrado ao usuário com um `Toast`.

- **Método onStart():**

```java
    protected void onStart() {
        super.onStart();
        askPhonePermission();
    }
```

```

- Esse método é chamado toda vez que a atividade é iniciada. Ele invoca o método `askPhonePermission()` para garantir que as permissões necessárias sejam concedidas.

**- **Método askPhonePermission()**:**

```java

**private void askPhonePermission() {**

**int hasPhonePermission = ContextCompat.checkSelfPermission(this, android.Manifest.permission.READ_PHONE_STATE);**

**if (hasPhonePermission != PackageManager.PERMISSION_GRANTED) {**

**ActivityCompat.requestPermissions(this, new String[]{android.Manifest.permission.READ_PHONE_STATE}, PHONE_STATUS_REQUEST_CODE);**

**}**

**hasPhonePermission = ContextCompat.checkSelfPermission(this, android.Manifest.permission.READ_CALL_LOG);**

**if (hasPhonePermission != PackageManager.PERMISSION_GRANTED) {**

**ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.READ_CALL_LOG}, PHONE_STATUS_REQUEST_CODE);**

**}**

**}**

```

- Esse método verifica se as permissões para ler o estado do telefone e o histórico de chamadas foram concedidas. Se não, ele solicita essas permissões ao usuário.

- **Método onRequestPermissionsResult():**

```java

**public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {**

**super.onRequestPermissionsResult(requestCode, permissions, grantResults);**

**if (requestCode == PHONE_STATUS_REQUEST_CODE) {**

**if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {**

**Toast.makeText(this, "permissão de estado do telefone concedida", Toast.LENGTH_SHORT).show();**

**} else {**

**Toast.makeText(this, "permissão de estado do telefone não concedida", Toast.LENGTH_SHORT).show();**

**}**

**}**

**}**

```

- Aqui, é tratado o resultado da solicitação de permissões. Se o usuário concedeu, uma mensagem de sucesso é exibida. Caso contrário, uma mensagem de erro é mostrada.

MyCallsReceiver.java

Essa classe é um `BroadcastReceiver` que recebe eventos do sistema relacionados ao estado do telefone. Sua função principal é interceptar chamadas e registrar suas informações.

- **Método onReceive(Context context, Intent intent):**

```java

public void onReceive(Context context, Intent intent) {

if (intent.getAction() == TelephonyManager.ACTION_PHONE_STATE_CHANGED) {

String state = intent.getStringExtra(TelephonyManager.EXTRA_STATE);

String number = intent.getStringExtra(TelephonyManager.EXTRA_INCOMING_NUMBER);

if (state.equals(TelephonyManager.EXTRA_STATE_RINGING)) {

saveCallDetails(context, "Recebida de: " + number);

} else if (state.equals(TelephonyManager.EXTRA_STATE_OFFHOOK)) {

saveCallDetails(context, "Atendida de: " + number);

}

}

}

```

- Esse método recebe atualizações sobre o estado do telefone.

- Quando o telefone toca (estado `RINGING`), o número da chamada recebida é registrado.

- Quando a chamada é atendida (`OFFHOOK`), a chamada é marcada como atendida.

**- **Método saveCallDetails(Context context, String logDetails)**:**

**```java**

**private void saveCallDetails(Context context, String logDetails) {**

**try {**

**FileOutputStream fos = context.openFileOutput("call_logs.txt", Context.MODE_APPEND);**

**String timeStamp = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());**

**fos.write((timeStamp + " - " + logDetails + "\n").getBytes());**

**fos.close();**

**} catch (IOException e) {**

**Log.e("Erro", "Erro ao salvar detalhes da chamada: " + e.getMessage());**

**}**

**}**

```

- Esse método grava os detalhes da chamada recebida ou feita no arquivo `call_logs.txt`, usando o armazenamento interno do dispositivo.

- Um `FileOutputStream` é utilizado para abrir o arquivo no modo de **append** (acrescentar), garantindo que novas chamadas sejam adicionadas ao arquivo sem sobrescrever as anteriores.

---

4. **Considerações Finais**

O aplicativo está bem estruturado, com responsabilidades claras entre as atividades e o receiver. Ele solicita as permissões necessárias e registra corretamente as chamadas recebidas e efetuadas. Há, contudo, melhorias que podem ser implementadas, como a verificação de permissões antes de executar certas ações e o tratamento de cenários em que o usuário nega as permissões permanentemente.

Este memorando resume o funcionamento e as partes principais do código, sendo um guia útil para futuras referências ou ajustes no aplicativo.

```

FIM DO MEMORANDO