**Министерство науки и высшего образования Российской Федерации**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Факультет безопасности информационных технологий**

**Дисциплина:**

«Безопасность мобильных устройств»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3**

«Создание приложения и поиск URL, конечных точек и секретов в APK-файлах (Kotlin)»

**Выполнили:**

Ахраров Али Рустамович, студент группы N3350

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

**Проверил:**

Федоров Иван Романович

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(отметка о выполнении)

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

Содержание

[Введение 4](#_Toc194100742)

[1 РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ 5](#_Toc194100743)

[1.1 Создание проекта в Android Studio 5](#_Toc194100744)

[1.2 Реализация экранов приложения 6](#_Toc194100745)

[1.2.1 Экран вывода ФИО студента 7](#_Toc194100746)

[1.2.2 Экран отображения данных из REST API 8](#_Toc194100747)

[1.2.3 Экран формы обратной связи 10](#_Toc194100748)

[2 СБОРКА И АНАЛИЗ APK-ФАЙЛОВ 12](#_Toc194100749)

[2.1 Получение APK-файла без обфускации (Debug) 12](#_Toc194100750)

[2.2 Получение APK-файла с обфускацией (Release) 12](#_Toc194100751)

[2.3 Декомпиляция APK-файлов с помощью ApkTool 13](#_Toc194100752)

[2.4 Сравнение результатов декомпиляции 13](#_Toc194100753)

[3 АНАЛИЗ БЕЗОПАСНОСТИ ПРИЛОЖЕНИЯ 16](#_Toc194100754)

[3.1 Сканирование разработанного приложения утилитой apkleaks 16](#_Toc194100755)

[3.2 Сканирование стороннего приложения утилитой apkleaks 16](#_Toc194100756)

[3.3 Сравнение результатов сканирования apkleaks 16](#_Toc194100757)

[3.4 Проверка разработанного приложения через VirusTotal 17](#_Toc194100758)

[4 ЗАКЛЮЧЕНИЕ 19](#_Toc194100759)

[5 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20](#_Toc194100760)

Введение

**Целью** данной лабораторной работы является получение практических навыков разработки мобильных приложений под ОС Android на языке Kotlin, а также освоение базовых методов анализа безопасности скомпилированных приложений.

**Задачи** лабораторной работы:

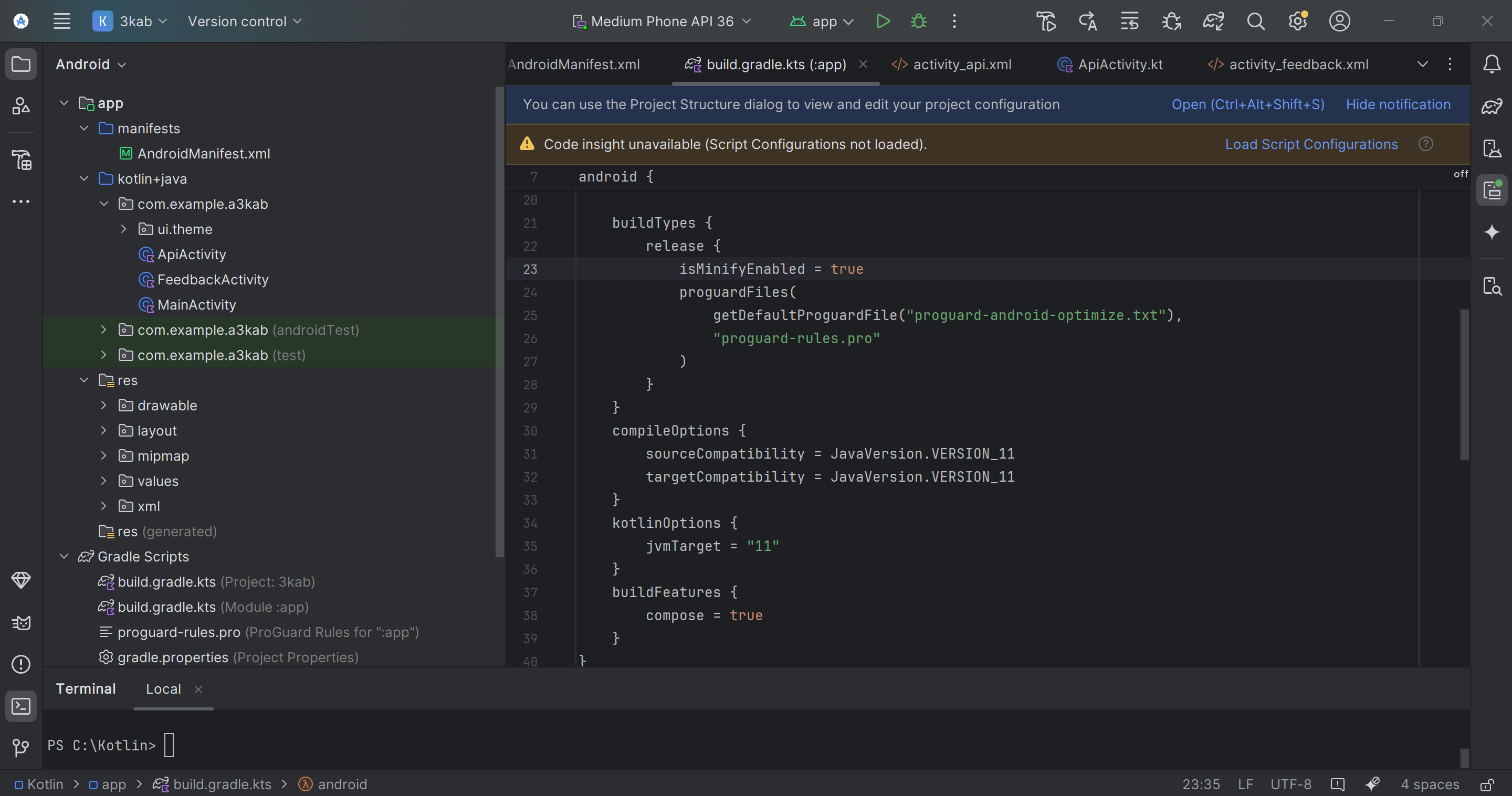
1. Создать проект мобильного приложения в среде Android Studio.
2. Реализовать три функциональных экрана: вывод ФИО, отображение данных из REST API, форма обратной связи.
3. Выполнить сборку приложения в двух вариантах: без обфускации (debug) и с обфускацией кода (release).
4. Декомпилировать полученные APK-файлы с использованием утилиты ApkTool.
5. Сравнить результаты декомпиляции для выявления эффекта обфускации.
6. Просканировать разработанное и стороннее приложение утилитой apkleaks для поиска потенциальных утечек данных.
7. Проверить разработанное приложение с помощью сервиса VirusTotal на наличие вредоносного кода.

В ходе работы были использованы среда разработки Android Studio, язык программирования Kotlin, библиотеки Android SDK, Volley, а также инструменты анализа ApkTool, apkleaks и сервис VirusTotal.

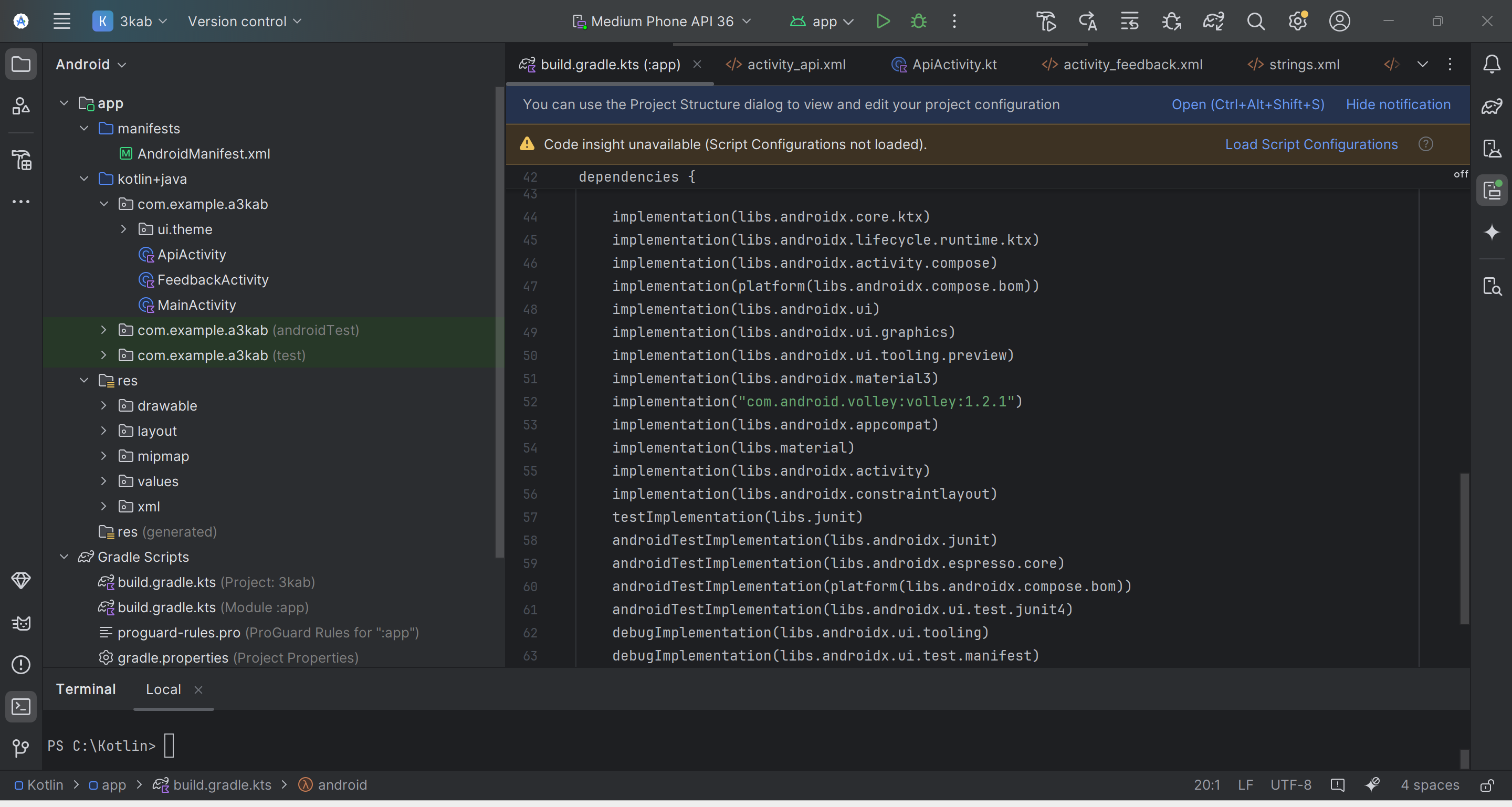
# **РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ**

## Создание проекта в Android Studio

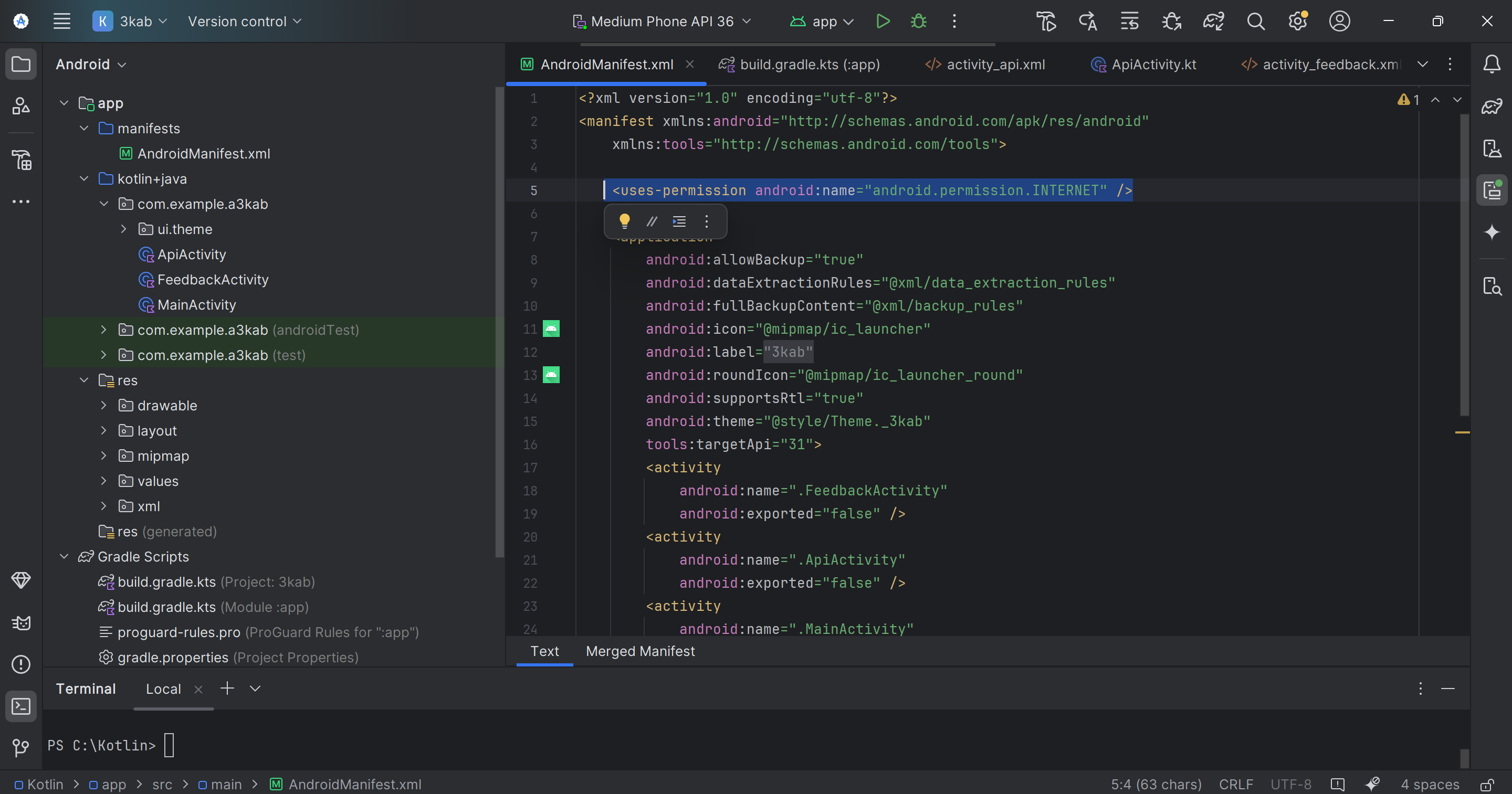
Работа началась с создания нового проекта в среде Android Studio. Был выбран шаблон "Empty Activity". В качестве языка программирования был указан Kotlin. Минимальная версия SDK была установлена как API 24 (Android 7.0 Nougat), целевая версия SDK - API 35. Имя пакета приложения было задано как com.example.a3kab.

  
Рисунок 1 – Структура проекта в Android Studio

Были настроены необходимые зависимости в файле app/build.gradle.kts, в частности, добавлена библиотека Volley для выполнения сетевых запросов. Также была добавлена зависимость для использования View Binding / findViewById (в вашем случае findViewById).

  
Рисунок 2 – Фрагмент файла build.gradle.kts с зависимостями

В файл AndroidManifest.xml было добавлено разрешение на использование интернета, необходимое для работы с REST API.

  
Рисунок 3 – Запрос разрешения INTERNET в AndroidManifest.xml

## Реализация экранов приложения

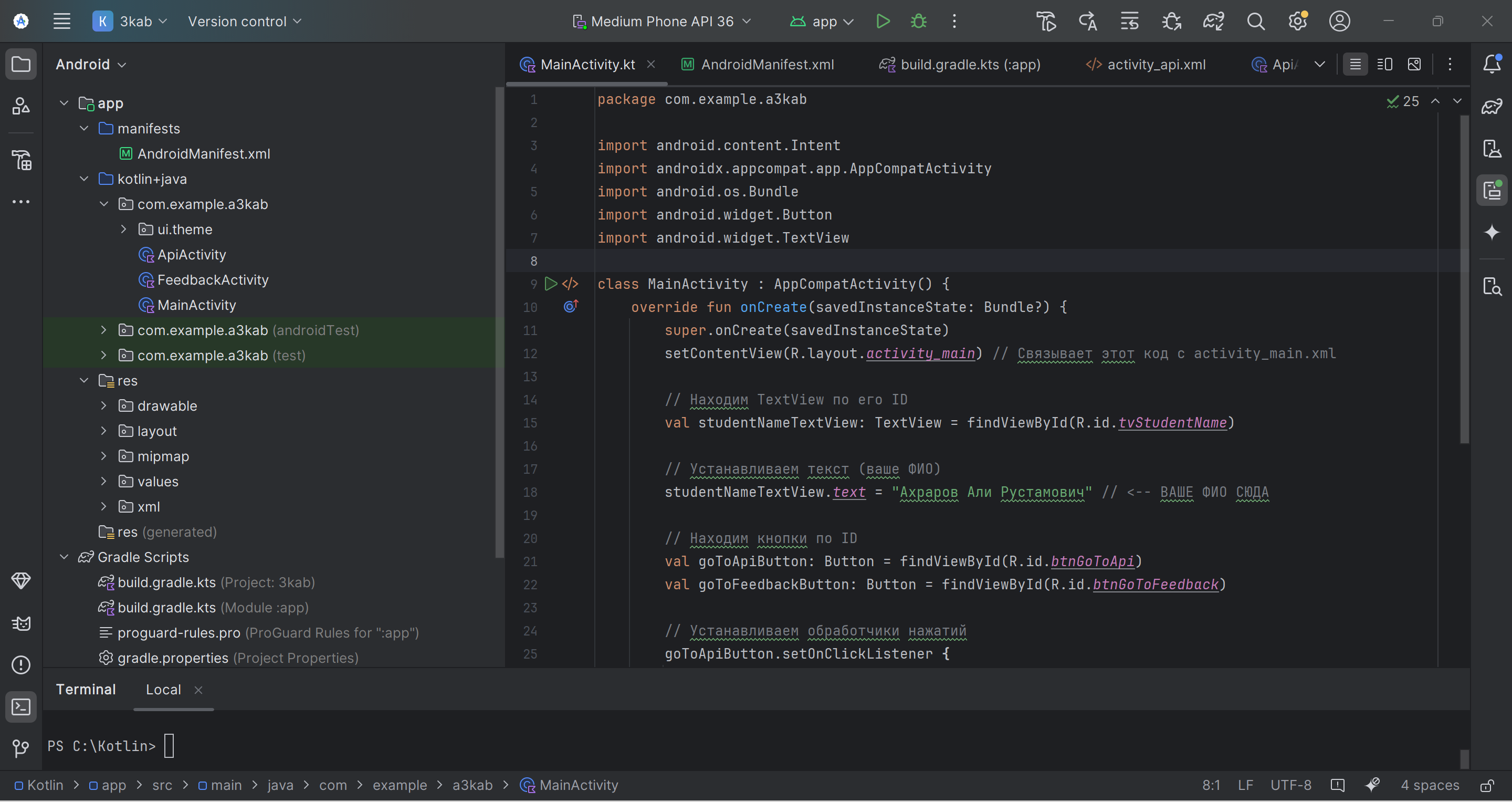
В соответствии с заданием было реализовано три экрана (Activity).

### **Экран вывода ФИО студента**

Данный экран (MainActivity) является стартовым экраном приложения. Его разметка (activity\_main.xml) содержит элемент TextView для отображения ФИО студента и две кнопки (Button) для перехода на другие экраны.

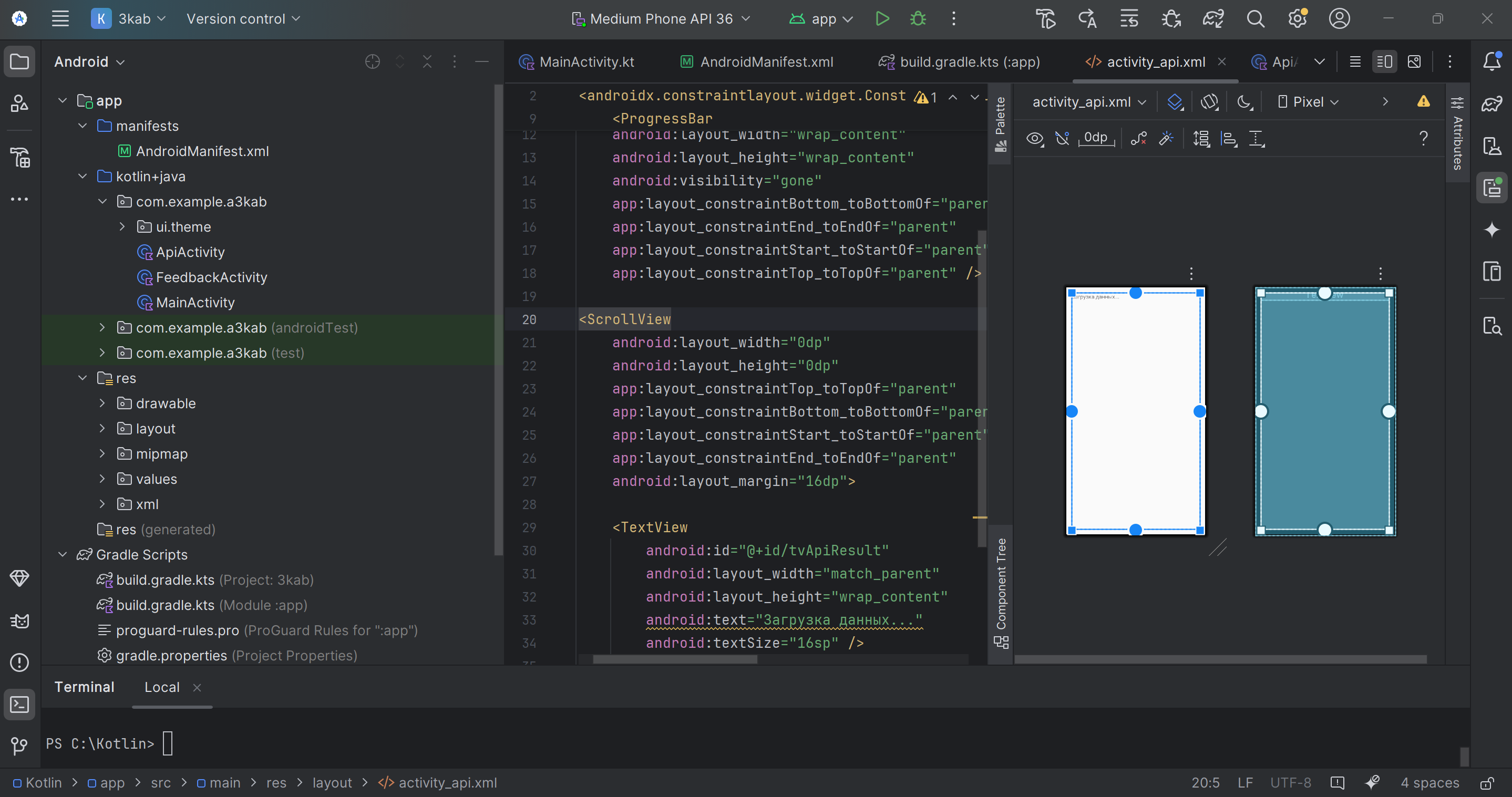
  
**Рисунок 4 – Разметка экрана activity\_main.xml**

В коде MainActivity.kt с помощью метода findViewById находятся необходимые View-элементы. В TextView программно устанавливается текст с ФИО студента. Для кнопок устанавливаются обработчики нажатий (setOnClickListener), которые создают Intent для запуска ApiActivity и FeedbackActivity соответственно.

  
Рисунок 5 – Фрагмент кода MainActivity.kt

### Экран отображения данных из REST API

Этот экран (ApiActivity) предназначен для демонстрации получения данных из сети. Был выбран публичный REST API jsonplaceholder (эндпоинт https://jsonplaceholder.typicode.com/posts/1). Разметка экрана (activity\_api.xml) содержит TextView для вывода полученных данных и ProgressBar для индикации процесса загрузки.

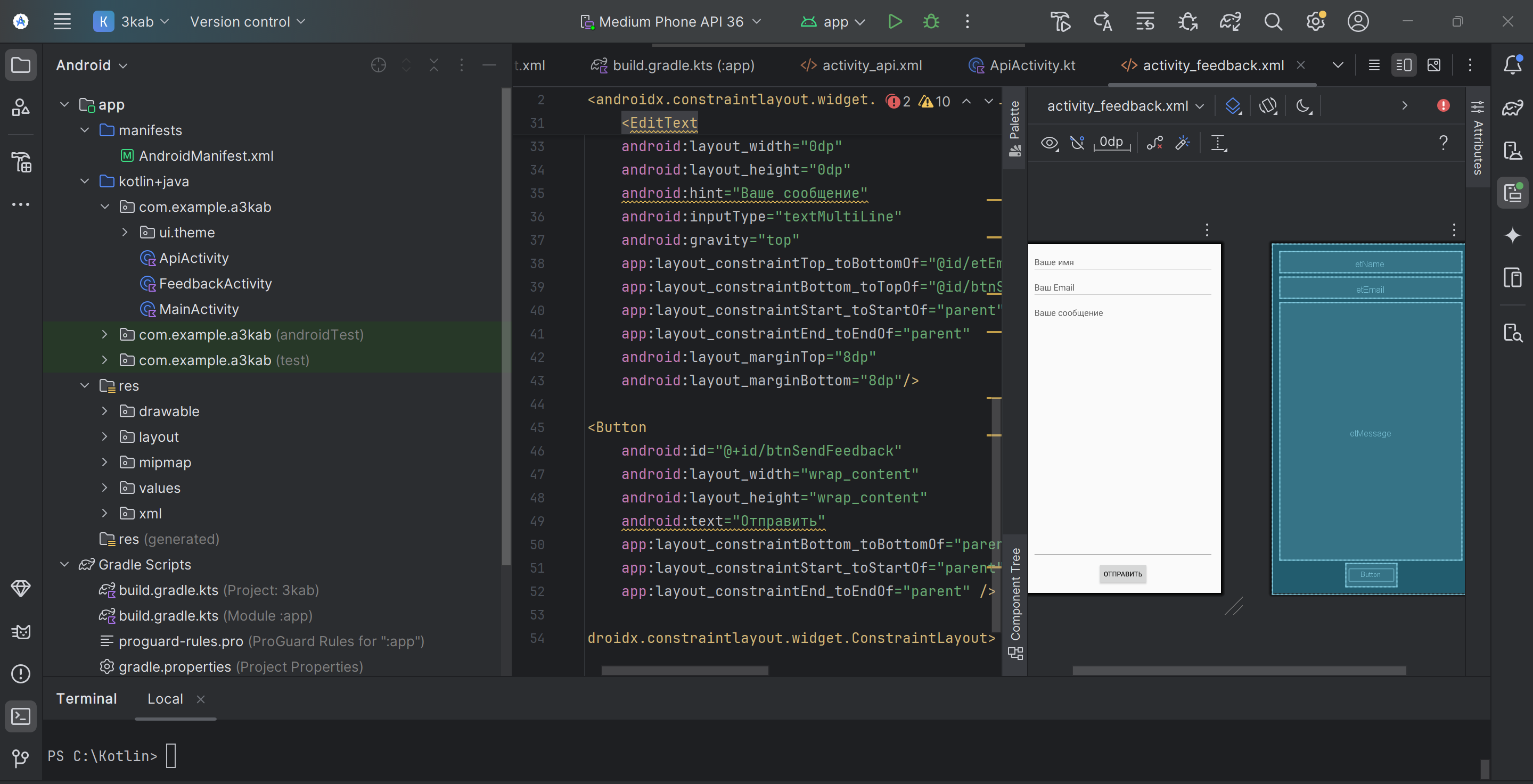
  
Рисунок 7 – Разметка экрана activity\_api.xml

В коде ApiActivity.kt используется библиотека Volley. В методе onCreate инициализируется очередь запросов RequestQueue. Вызывается метод fetchDataFromApi, который создает StringRequest для GET-запроса по указанному URL. Перед началом запроса ProgressBar делается видимым. В слушателе успешного ответа (Response.Listener) ProgressBar скрывается, и полученный текстовый ответ отображается в TextView. В слушателе ошибки (Response.ErrorListener) ProgressBar также скрывается, и выводится сообщение об ошибке. Запрос добавляется в очередь для выполнения.

  
Рисунок 8 – Фрагмент кода ApiActivity.kt с использованием Volley

### Экран формы обратной связи

Третий экран (FeedbackActivity) представляет собой простую форму для отправки отзыва. Разметка (activity\_feedback.xml) включает поля ввода (EditText) для имени, email и сообщения, а также кнопку (Button) "Отправить".

  
Рисунок 10 – Разметка экрана activity\_feedback.xml

В коде FeedbackActivity.kt в методе onCreate находятся все View-элементы через findViewById. На кнопку "Отправить" устанавливается OnClickListener. В обработчике нажатия считываются тексты из полей EditText. Выполняется простая проверка на то, что поля не пустые. Если поля заполнены, имитируется отправка данных: текст отзыва выводится в Logcat Android Studio и отображается Toast-сообщение об успехе. Поля ввода очищаются. Если поля пустые, выводится Toast-сообщение с просьбой заполнить все поля.

# СБОРКА И АНАЛИЗ APK-ФАЙЛОВ

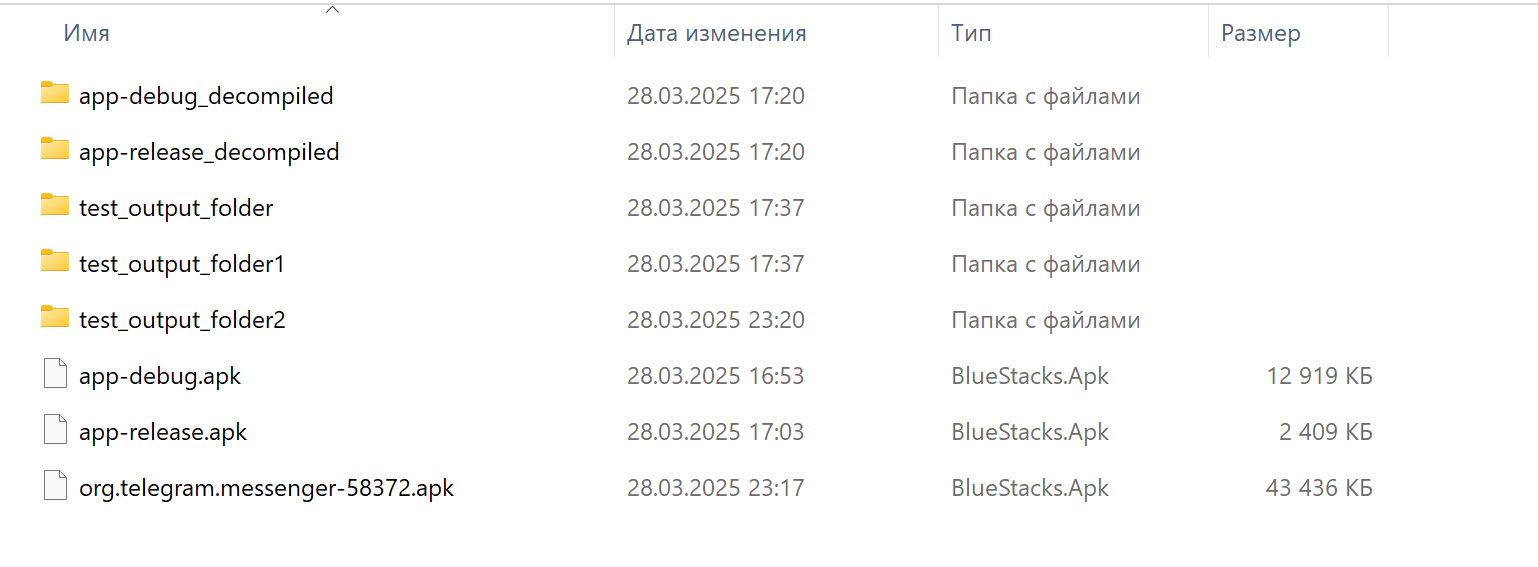
## Получение APK-файла без обфускации (Debug)

Для получения отладочной версии APK-файла была использована стандартная функция сборки в Android Studio. Через меню Build -> Build Bundle(s) / APK(s) -> Build APK(s) была инициирована сборка debug-варианта.

## Получение APK-файла с обфускацией (Release)

Для создания release-версии с обфускацией кода были выполнены следующие шаги:

1. В файле app/build.gradle.kts в секции buildTypes -> release параметр isMinifyEnabled был установлен в значение true. Это включает R8/ProGuard для сжатия, оптимизации и обфускации кода.
2. Был сгенерирован ключ подписи приложения с использованием инструмента Android Studio (Build -> Generate Signed Bundle / APK...). Создан файл хранилища ключей (.jks) и заданы пароли.
3. С помощью того же инструмента (Generate Signed Bundle / APK...) была выполнена сборка подписанного release APK. Был выбран созданный ключ, указаны пароли, выбран вариант сборки release.

  
Рисунок 13 – Полученные файлы после всех манипуляций

## Декомпиляция APK-файлов с помощью ApkTool

Для анализа содержимого собранных APK-файлов был использован инструмент ApkTool версии. Инструмент был предварительно установлен и настроен для работы из командной строки Windows.

Декомпиляция выполнялась с помощью следующих команд в терминале, находясь в директории с APK-файлами:

 apktool d app-debug.apk -o app-debug\_decompiled

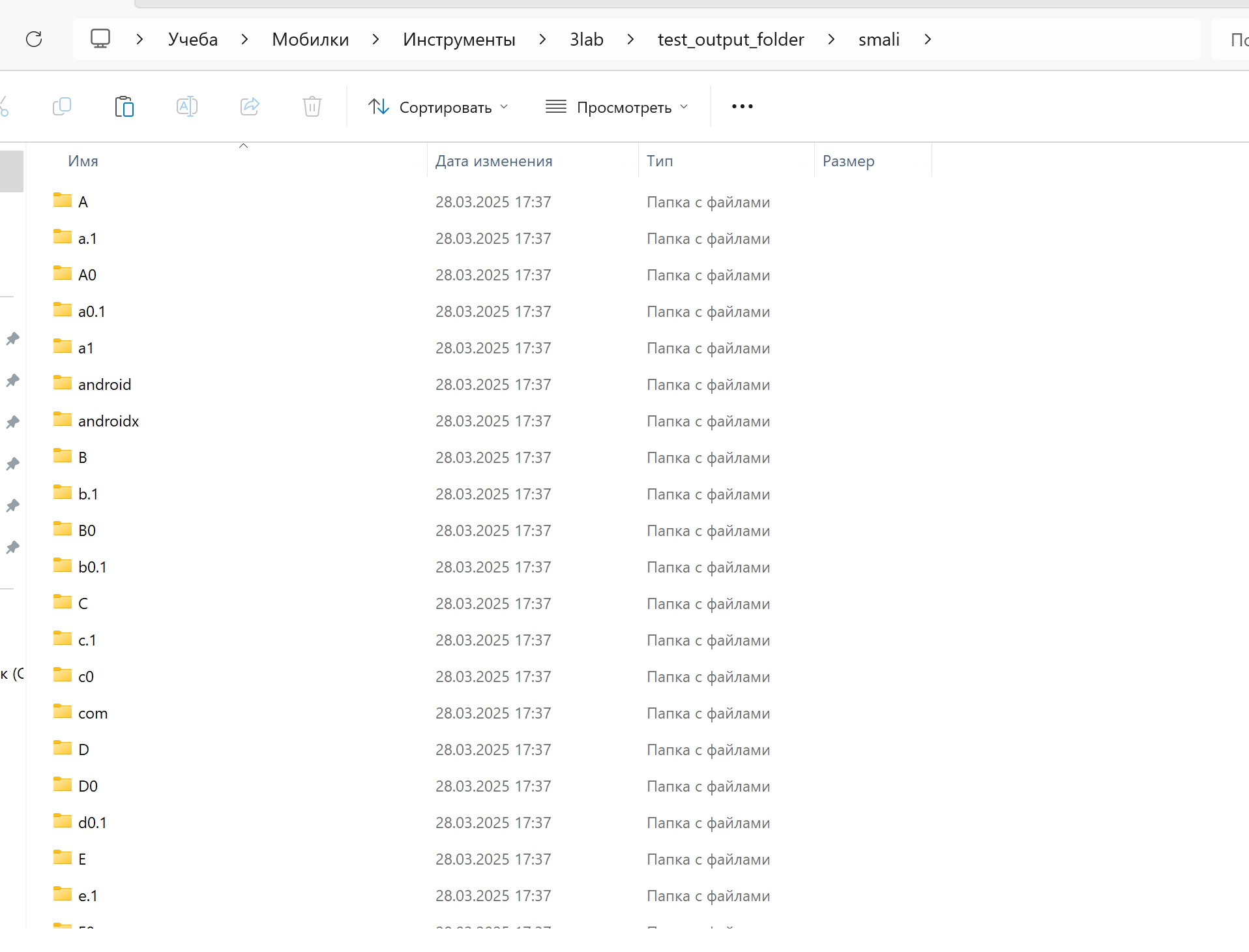
apktool d app-release.apk -o app-release\_decompiled

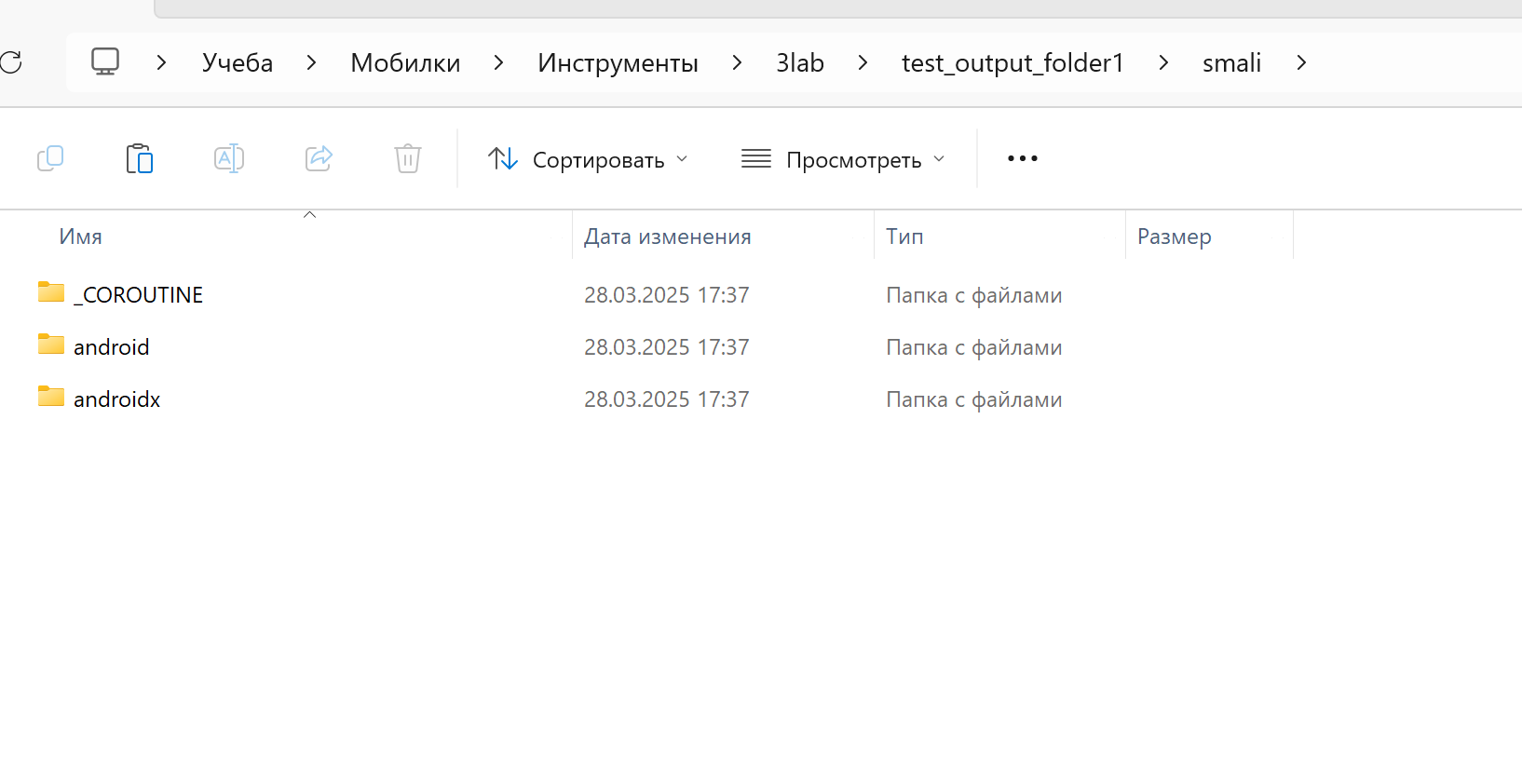
В результате выполнения команд были созданы две папки (app-debug\_decompiled и app-release\_decompiled), содержащие декомпилированные ресурсы и smali-код приложения.

## Сравнение результатов декомпиляции

Было проведено сравнение содержимого папок app-debug\_decompiled и app-release\_decompiled. Основное внимание было уделено папкам smali (содержащим байт-код Dalvik в текстовом представлении) и файлу AndroidManifest.xml.

* **AndroidManifest.xml:** Существенных различий в манифесте между debug и release версиями обнаружено не было. Структура и основные компоненты остались прежними.
* **Ресурсы (папка res):** Имена файлов ресурсов и их структура также не изменились.
* **Smali-код (папка smali):** Здесь наблюдались значительные различия, демонстрирующие эффект обфускации:
  + **В app-debug\_decompiled:** Имена классов (например, MainActivity, ApiActivity), методов (onCreate, fetchDataFromApi) и полей соответствовали именам в исходном Kotlin-коде. Код был относительно читаем (насколько может быть читаем smali).
  + **В app-release\_decompiled:** Имена пользовательских классов, методов и полей были заменены короткими, ничего не значащими идентификаторами (например, a, b, c, a.a, a.b и т.д.). Это существенно затрудняет понимание логики работы приложения при анализе декомпилированного кода. Системные классы и методы Android SDK остались с оригинальными именами.



  
Рисунок 17 – Сравнение smali-кода до и после обфускации

**Вывод:** Обфускация, включенная параметром isMinifyEnabled = true, эффективно скрывает оригинальные имена классов, методов и полей в скомпилированном коде, что затрудняет реверс-инжиниринг приложения. Однако обфускация не затрагивает строковые литералы и ресурсы.

# АНАЛИЗ БЕЗОПАСНОСТИ ПРИЛОЖЕНИЯ

## Сканирование разработанного приложения утилитой apkleaks

Для поиска потенциальных утечек чувствительной информации (URL, ключи API, email и т.д.) была использована утилита apkleaks. Утилита была установлена с помощью менеджера пакетов pip. Сканирование проводилось для обеих версий APK (debug и release).

Сканирование выполнялось из командной строки с помощью следующих команд (предварительно был выполнен переход в директорию с APK-файлами, не содержащую кириллических символов, например C:\TempTest, из-за особенностей работы apkleaks в Windows):

apkleaks -f app-debug.apk -o apkleaks\_debug\_report.txt

      apkleaks -f app-release.apk -o apkleaks\_release\_report.txt

## Сканирование стороннего приложения утилитой apkleaks

Для сравнения было выбрано стороннее приложение [Укажите название приложения, например, Telegram Messenger]. APK-файл приложения (org.telegram.messenger.apk) был получен с помощью

Сканирование стороннего приложения выполнялось аналогично командой:

apkleaks -f org.telegram.messenger.apk -o

      apkleaks\_telegram\_report.txt

## Сравнение результатов сканирования apkleaks

При сравнении отчетов apkleaks для разработанного приложения и для стороннего приложения (Telegram) были выявлены следующие различия:

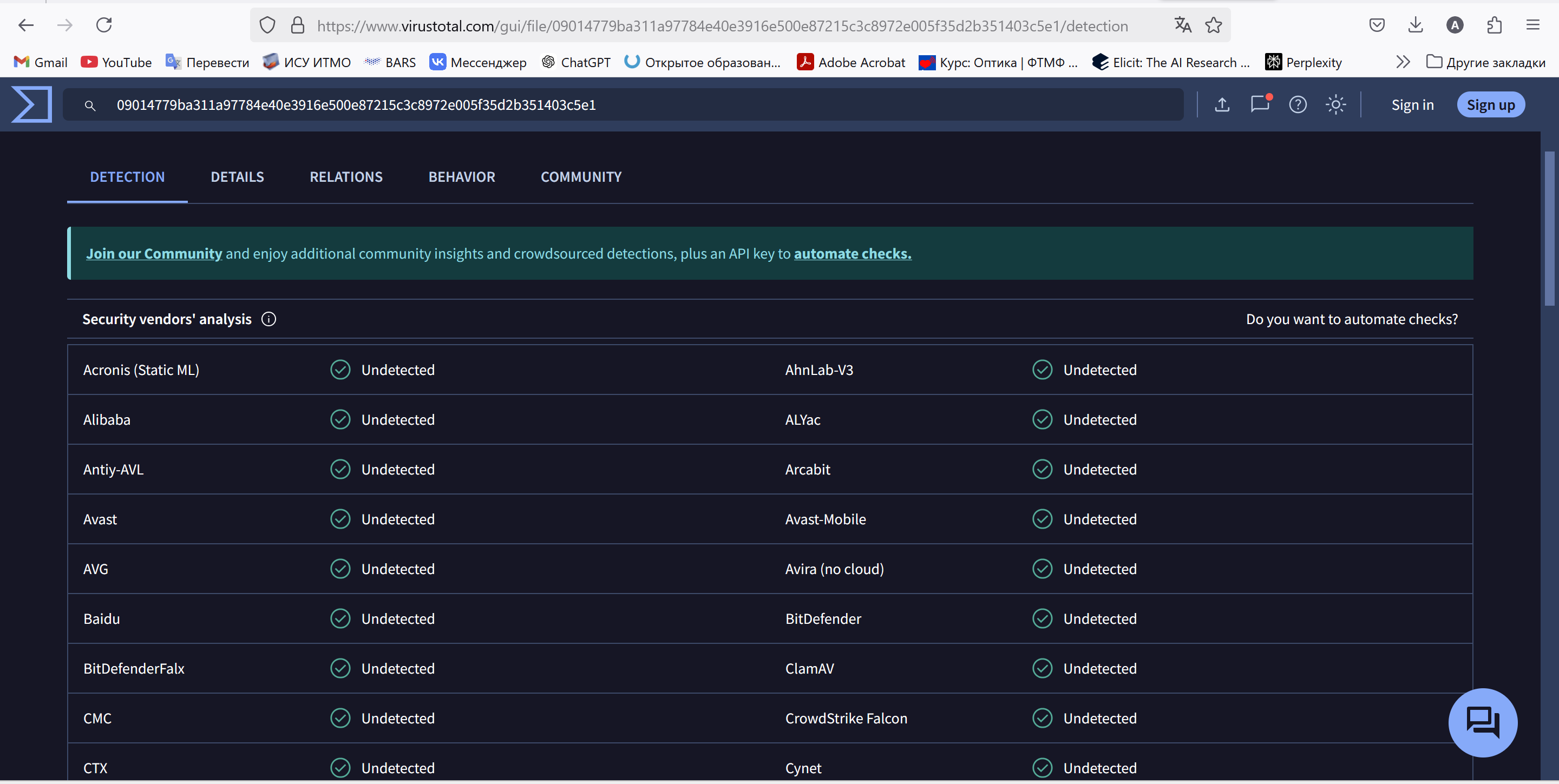
* **Объем найденной информации:** Отчет для Telegram содержит значительно большее количество найденных строк (URL, потенциальных эндпоинтов), чем отчеты для разработанного приложения. Это ожидаемо, учитывая сложность коммерческого мессенджера.
* **Типы URL:** В отчете Telegram присутствуют URL, относящиеся к собственным серверам Telegram, CDN, возможно, аналитическим и рекламным сервисам, в то время как в разработанном приложении был найден только URL тестового API и стандартные схемы.
* **Потенциальные "секреты":** В отчете Telegram apkleaks мог обнаружить строки, соответствующие шаблонам ключей API или других идентификаторов, что требует дополнительного анализа для определения их реальной чувствительности (в учебных целях детальный анализ не проводился). В разработанном приложении таких строк не было.

**Вывод:** Сканирование показало, что даже простые приложения могут содержать строки (например, URL API), обнаруживаемые инструментами анализа. Сложные коммерческие приложения содержат значительно больше таких строк, что подчеркивает важность контроля за отсутствием утечек реальных секретных данных в коде и ресурсах.

## Проверка разработанного приложения через VirusTotal

Заключительным этапом анализа стала проверка разработанного приложения (app-release.apk) с помощью онлайн-сервиса VirusTotal. Файл был загружен на сайт [https://www.virustotal.com/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fwww.virustotal.com%2F) для сканирования множеством антивирусных движков.

Анализ показал, что файл был признан безопасным всеми антивирусных сканеров.

  
Рисунок 23 – Результат проверки приложения на VirusTotal

**Вывод:** Проверка на VirusTotal подтвердила, что разработанное в рамках лабораторной работы приложение не содержит известных вредоносных программ и не определяется как угроза безопасности стандартными антивирусными средствами.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были успешно решены все поставленные задачи. Было разработано мобильное приложение для ОС Android с тремя экранами, реализующими заявленный функционал: вывод статической информации (ФИО), получение и отображение данных из внешнего REST API с использованием библиотеки Volley, а также реализация формы обратной связи с имитацией отправки данных.

Были освоены процессы сборки приложения в debug и release конфигурациях, включая настройку обфускации кода с помощью R8/ProGuard. Декомпиляция полученных APK-файлов с использованием ApkTool позволила наглядно продемонстрировать эффект обфускации, заключающийся в замене имен классов, методов и полей, что затрудняет реверс-инжиниринг.

Проведено сканирование разработанного и стороннего приложений утилитой apkleaks. Анализ показал наличие URL API в разработанном приложении и значительно большее количество потенциально интересных строк в стороннем приложении, подтвердив, что обфускация не скрывает строковые литералы.

Проверка разработанного приложения сервисом VirusTotal показала отсутствие детекций со стороны антивирусных сканеров, подтвердив его безопасность.

Таким образом, в результате выполнения работы были получены практические навыки разработки Android-приложений и использования инструментов для их базового анализа на предмет обфускации кода и наличия потенциальных утечек информации.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Основы разработки под Android. – URL: [https://polis-mail-ru.github.io/2019-android/01\_intro/011\_hello\_world/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fpolis-mail-ru.github.io%2F2019-android%2F01_intro%2F011_hello_world%2F)
2. Android Studio documentation | Android Developers. – URL: [https://developer.android.com/studio](https://www.google.com/url?sa=E&q=https%3A%2F%2Fdeveloper.android.com%2Fstudio)
3. Apktool - A tool for reverse engineering Android apk files. – URL: [https://ibotpeaches.github.io/Apktool/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fibotpeaches.github.io%2FApktool%2F)
4. Как работать с Apktool. – URL: [https://computer76.ru/2018/04/17/how-to-work-with-apktool/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fcomputer76.ru%2F2018%2F04%2F17%2Fhow-to-work-with-apktool%2F)
5. Вскрытие покажет: потрошим и модифицируем Android-приложения / Компьютерра. – URL: [https://xakep.ru/2013/10/22/modify-and-packing-android-apps/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fxakep.ru%2F2013%2F10%2F22%2Fmodify-and-packing-android-apps%2F)
6. GitHub - dwisiswant0/apkleaks: Scanning APK file for URIs, endpoints & secrets.. – URL: [https://github.com/dwisiswant0/apkleaks](https://www.google.com/url?sa=E&q=https%3A%2F%2Fgithub.com%2Fdwisiswant0%2Fapkleaks)
7. VirusTotal. – URL: [https://www.virustotal.com/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fwww.virustotal.com%2F)