УДК 004

Использование файла конфигурации для защиты API-ключа в Android-приложении

Андриенко Иван Сергеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье представлен подробный процесс разработки Android-приложения, фокусирующегося на безопасном управлении API-ключами. Используя язык Java и Android Studio, создано простое приложение для отображения погодных данных через OpenWeatherMap. Описан метод эффективного использования файла конфигурации для защиты API-ключа, обеспечивая безопасность и удобство управления конфиденциальными данными в процессе разработки Android-приложений. Рассмотрено как избежать случайных утечек ключей, разделять настройки для различных профилей сборки и поддерживать высокий уровень безопасности конфигураций в проектах.
Ключевые слова: Мобильное приложение, Android, API, OpenWeatherMap, Android Studio.

Using a configuration file to protect the API key in an Android application

Andrienko Ivan Sergeevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article provides a detailed development process for an Android application focusing on secure API key management. Using the Java language and Android Studio, a simple application has been created to display weather data via OpenWeatherMap. The method of effective use of the configuration file to protect the API key is described, ensuring the security and convenience of managing confidential data during the development of Android applications. It is considered how to avoid accidental key leaks, separate settings for different build profiles and maintain a high level of configuration security in projects.
Keywords: Mobile application, Android, API, OpenWeatherMap, Android Studio.

1 Введение

1.1 Актуальность

В мире постоянно растет зависимость от мобильных приложений, и с ней возрастает необходимость эффективного управления конфиденциальной информацией, такой как API-ключи. Безопасное хранение этих ключей

становится приоритетным вопросом для разработчиков Android-приложений, особенно учитывая увеличивающееся количество кибератак и несанкционированных попыток доступа к данным. Использование API-ключей в приложениях требует тщательного обращения, чтобы предотвратить возможные утечки и злоупотребления.

1.2 Обзор исследований

В своей работе С.В. Соленый, И.А. Воропаев, М.Н. Давиденко рассмотрели применение API и возможные уязвимости, которые связанны с данной технологией. Дается основное представление об устройстве и принципе работы с API. Подробно описывается, почему многие компании используют API в своем продукте, и какие методы защиты должны быть учтены при разработке [1]. Р.В. Еровлева, П.А. Еровлев рассмотрели возможности создания запросов REST API и защита их с использованием Spring Security. Работа будет происходить в среде разработки IntelijIdea [2]. В своей работе А. Календарев описал сложное монолитное приложение, деленное на несколько микросервисов и публичное API для клиентов. Публичное API и общение между открытыми микросервисами требуют авторизационной защиты [3]. В своей работе Д.Ю. Стенин рассмотрел применяемые способы защиты информации при интеграции различных программных приложений и систем в процессе возникающего взаимодействия. Практическая значимость публикации состоит в акцентировании внимания на ресурсоориентированном подходе, реализуемым различными стандартами. Предложен авторский алгоритм формирования маршрута взаимодействия всех обработчиков для API интерфейса, который может быть внедрен как документированная процедура [4]. В статье О.И. Ткаченко, В.А Шепель описали типизации алгоритмов разработки сервисов геопозиционирования путем выделения функций сервисов и классов задач. Определены пути дальнейшего развития и совершенствования разработанного сервиса. Описаны требования к разработке интерфейса, обеспечивающие on-line-функционирование разработанного программного продукта. Предложено использование удаленных виртуальных серверов, служб и функций, API Google [5].

1.3 Цель исследования

Цель исследования – Разработать и представить эффективный подход к безопасному управлению API-ключами в Android-приложениях, используя файл конфигурации.

2 Материалы и методы

Процесс создания Android-приложения, отображающего текущую погоду через OpenWeatherMap API с безопасным управлением API-ключом, проделан в Android Studio, язык программирования Java.

3 Результаты и обсуждения

Перед настройкой приложения необходимо получить API ключ. Для этого необходимо авторизоваться на сайте [6] и перейти в раздел API Keys. Там можно сгенерировать бесплатный ключ.

После получения ключа создаем проект в Android Studio. Важным этапом является выбор параметра Build configuration language, а именно Groovy DSL (build.gradle), так как дальнейшая работа будет со скриптами gradle (рис. 1).

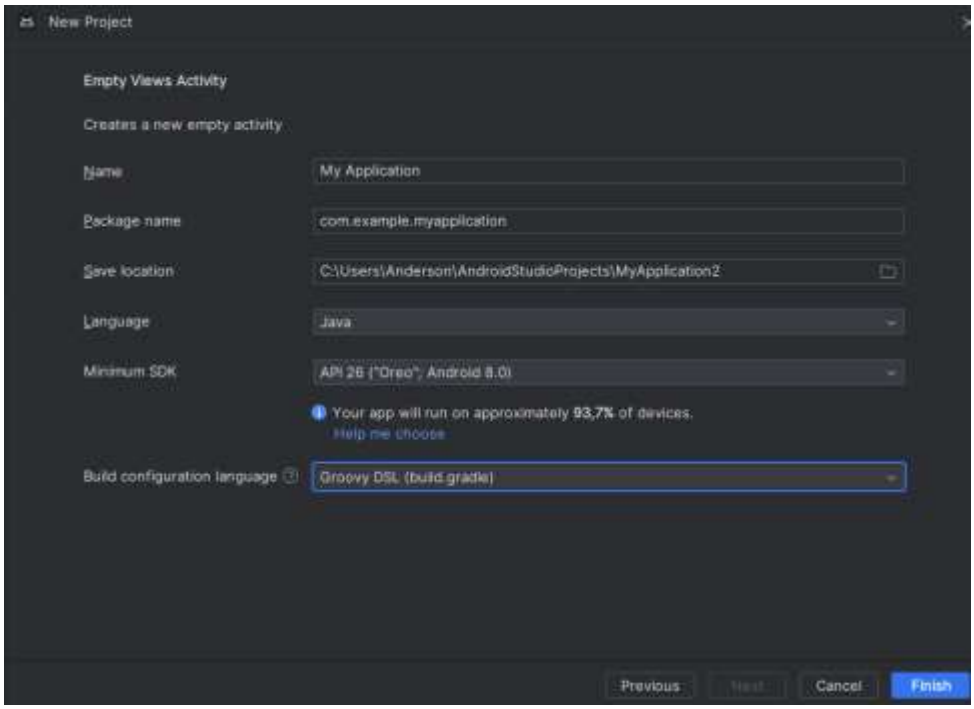


Рисунок 1 – Выбор параметра «Build configuration language»

После создания проекта необходимо добавить блоки с конфигурацией в build.gradle. Файл build.gradle представляет собой скриптовый файл конфигурации для проекта Android, написанный на языке Groovy или Kotlin. Этот файл определяет различные параметры сборки проекта, включая зависимости, настройки сборки и другие аспекты проекта. В Android Studio обычно есть два уровня build.gradle файлов: один для проекта в целом и другой для модуля приложения. В данном случае изменяем второй (рис. 2).

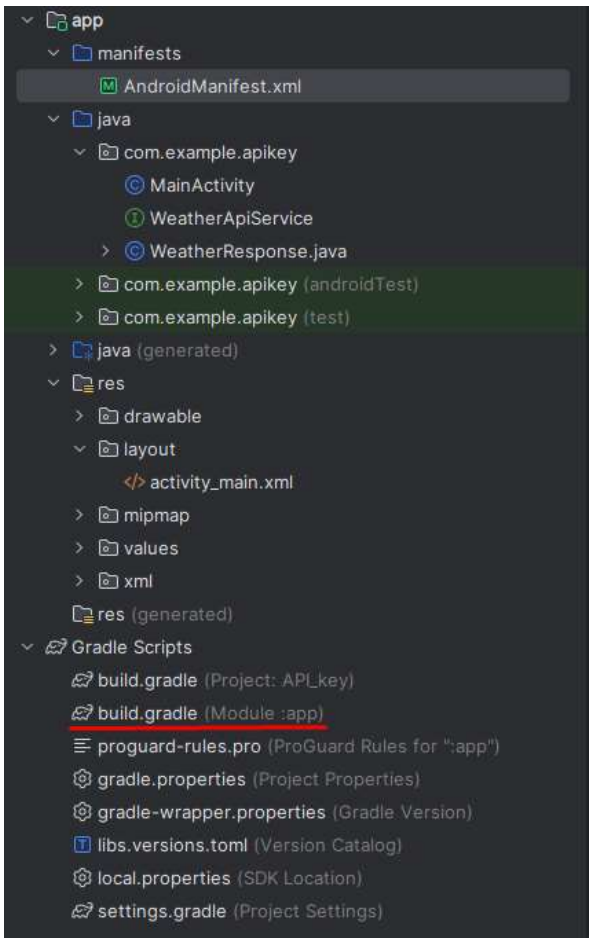


Рисунок 2 – Работа с файлом build.gradle (Module :app)

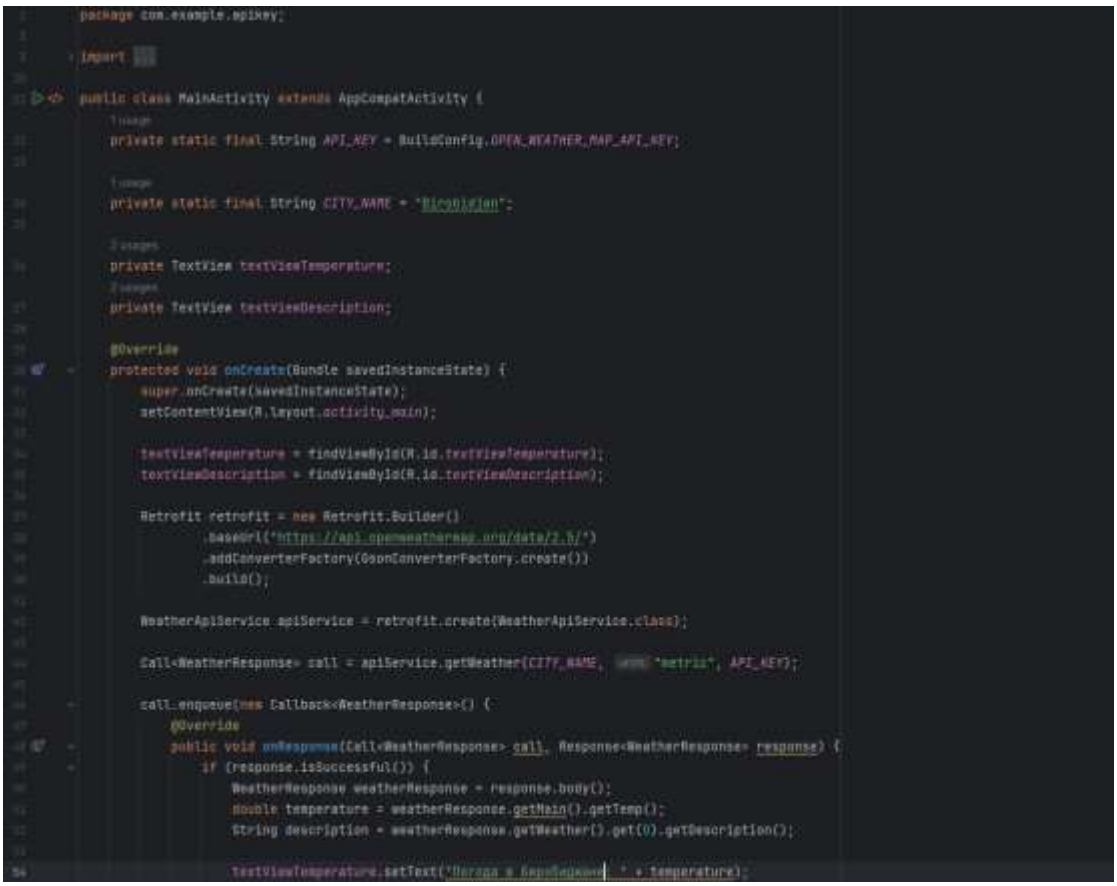
Далее пропишем код в файле build.gradle, который добавляет конфигурации для использования API-ключа. Всего нужно добавить две конфигурации buildFeatures и buildTypes. Конфигурация buildFeatures включает генерацию класса BuildConfig, который содержит различные переменные конфигурации сборки, включая собственные пользовательские поля. В конфигурации buildTypes определены различные типы сборок, такие как debug и release. Внутри каждого типа сборки используется метод buildConfigField, который добавляет поле OPEN_WEATHER_MAP_API_KEY в класс BuildConfig с соответствующим значением API-ключа. Пропишем их (рис. 3).

Рисунок 3 – Добавление конфигураций

После изменения файлов gradle проект необходимо синхронизировать с файлами, с помощью. Теперь API-ключ хранится внутри файла конфигурации build.gradle и не является частью исходного кода приложения, который может быть подвергнут инспекции или случайному раскрытию при публикации кода. В результате, API-ключ может быть легко обновлен, не затрагивая исходный код приложения, и при этом ключ не виден в исходном коде, поставляемом с приложением.

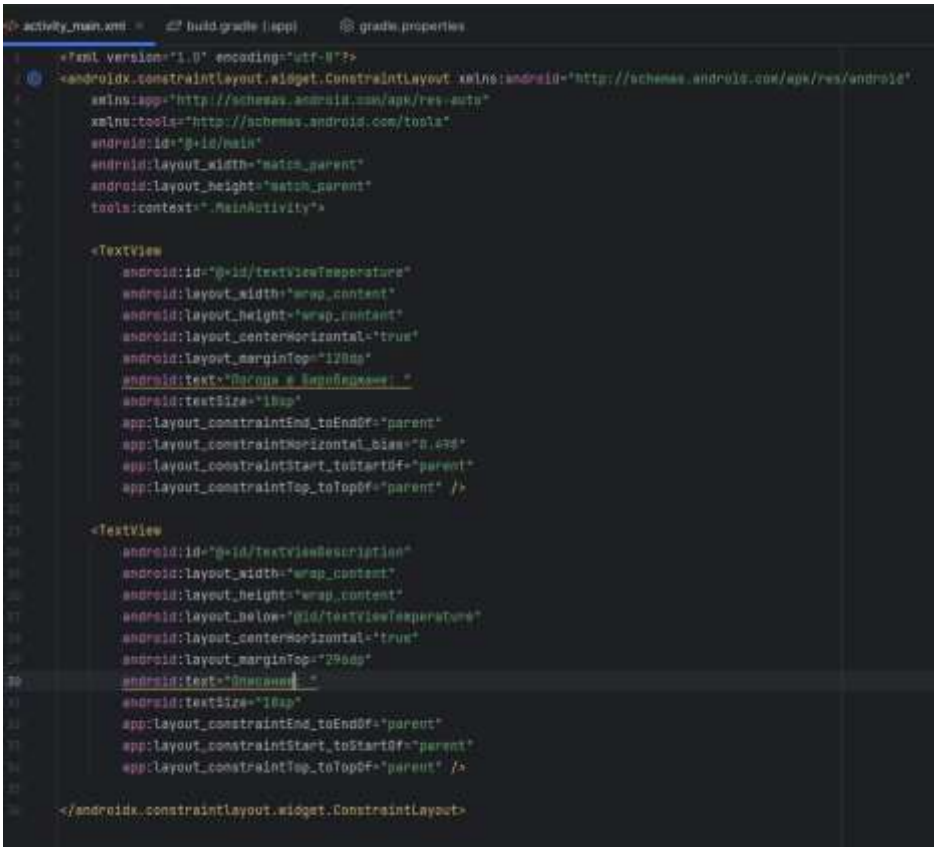
Для доступа к API-ключу в коде приложения, используем сгенерированный класс BuildConfig, который автоматически создается системой сборки Gradle. В данном случае, обратимся к ключу с помощью строки «String apiKey = BuildConfig.OPEN_WEATHER_MAP_API_KEY;». Где BuildConfig — это класс, сгенерированный в процессе сборки. Этот класс будет содержать все поля, определенные в блоке buildConfigField файла build.gradle.

Протестируем доступ к ключу. Создадим приложение, показывающее погоду в городе Биробиджан. Пропишем логику и создадим интерфейс (рис. 4, 5)



```
package com.example.apikey;

import ...

public class MainActivity extends AppCompatActivity {
    private static final String API_KEY = BuildConfig.OPEN_WEATHER_MAP_API_KEY;

    private static final String CITY_NAME = "Владивосток";

    private TextView textViewTemperature;
    private TextView textViewDescription;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        textViewTemperature = findViewById(R.id.textViewTemperature);
        textViewDescription = findViewById(R.id.textViewDescription);

        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl("https://api.openweathermap.org/data/2.5/")
                .addConverterFactory(GsonConverterFactory.create())
                .build();

        WeatherApiService apiService = retrofit.create(WeatherApiService.class);

        Call<WeatherResponse> call = apiService.getWeather(CITY_NAME, "metric", API_KEY);

        call.enqueue(new Callback<WeatherResponse>() {
            @Override
            public void onResponse(Call<WeatherResponse> call, Response<WeatherResponse> response) {
                if (response.isSuccessful()) {
                    WeatherResponse weatherResponse = response.body();
                    double temperature = weatherResponse.getMain().getTemp();
                    String description = weatherResponse.getWeather().get(0).getDescription();

                    textViewTemperature.setText("Погода в Барабашево: " + temperature);
```

Рисунок 4 – Код приложения



```
activity_main.xml   build.gradle (:app)   gradle.properties

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textViewTemperature"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="120dp"
        android:text="Погода в Барабашево: "
        android:textSize="18sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textViewDescription"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/textViewTemperature"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="298dp"
        android:text="Описание: "
        android:textSize="18sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Рисунок 5 – Интерфейс приложения

При запуске приложения происходит успешное получение погоды в городе, что показывает доступность к API ключу, при этом пользователь такого доступа не имеет (рис. 6).

Рисунок 6 – Тест приложения

Выводы

В данной статье был представлен пример использования файла конфигурации для защиты API-ключа в Android-приложении, на примере интеграции с OpenWeatherMap API. Процесс разработки включал в себя шаги создания проекта в Android Studio, добавление необходимых зависимостей, интеграцию с OpenWeatherMap API для тестирования работоспособности, и использование файла конфигурации для безопасного хранения API-ключа. В итоге, разработанное приложение успешно демонстрирует принципы защиты API-ключей в мобильных приложениях, обеспечивая безопасность и надежность работы с внешними сервисами.

Библиографический список

1. Соленый С.В., Воропаев И.А., Давиденко М.Н. Об API и архитектурных методах защиты при его разработке // Волновая электроника и инфокоммуникационные системы. Материалы XXV Международной научной конференции. Санкт-Петербург, 2022. С. 251-256.
2. Еровлева Р.В., Еровлев П.А. Защита REST API с помощью spring security // Постулат. 2021. № 8 (70).
3. Календарев А. Прозрачная защита микросервисов. Как защитить API и создать функциональность для REST API // Системный администратор. 2016. № 11 (168). С. 50-55.

4. Стенин Д.Ю. Способы защиты информации при работе с API интерфейсами // Инновационное развитие: ключевые проблемы и направления их решения. сборник статей по итогам Всероссийской научно-практической конференции с международным участием. Стерлитамак, 2023. С. 86-91.
5. Ткаченко О.И., Шепель В.А Некоторые аспекты разработки мобильного сервиса геопозиционирования на основе API Google // Водный транспорт. 2016. № 2 (25). С. 228-234.
6. URL: https://home.openweathermap.org/api_keys (дата обращения 12.01.2024)