Implementing Push Notification-Based Two-Factor Authentication (2FA)

Prakash Ranjan Follow 3 min read · Mar 8, 2025

Two-factor authentication (2FA) enhances security by requiring a second verification step beyond just a username and password. In this guide, we will implement a push notification-based 2FA system where logging in on the web triggers a push notification on the user's mobile device. The user can approve or deny the request, and upon approval, they are logged in.

Architecture Overview

1. **User logs in on the web** with their username and password.
2. **Backend validates credentials** and triggers a push notification to the user's Android device.
3. **If the app is in the foreground**, a full-screen approval UI is shown.
4. **If the app is in the background**, a push notification is displayed.

5. **User taps "Approve"** to authenticate.
6. **Backend validates the response** and logs the user in.
7. **Web client completes authentication** and grants access.

Step 1: Implementing 2FA on the Web

1.1 Creating the Backend API

The backend should expose an API to trigger 2FA authentication requests and verify approvals.

API Endpoints:

- **POST** — Triggers a push notification to the user's device.
- **POST** — Verifies the user's approval and completes login.

Request 2FA (Trigger Push Notification)

```
{
    "userId": "12345",
    "deviceToken": "abcdef123456"
}
```

Verify Approval (Upon User Action)

```
{
    "requestId": "req-67890",
```

```
    "approved": true
}
```

1.2 Implementing the Web Login Flow

When the user logs in, the backend should:

1. Validate username and password.
2. Check if 2FA is enabled.
3. Send a push notification request to the mobile device.
4. Wait for user approval before completing authentication.

Example frontend logic using JavaScript (React-based authentication flow):

```
async function login(username, password) {
    const response = await fetch('/api/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, password })
    });

    const data = await response.json();
    if (data.requires2FA) {
        await fetch('/api/2fa/request', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ userId: data.userId, deviceToken: data.device
        });
```

```
        }
    }
```

Step 2: Setting Up Firebase for Push Notifications

Firebase Cloud Messaging (FCM) is used to send push notifications to Android devices. Below are detailed steps to set up FCM.

2.1 Setting Up Firebase Project

1. **Create a Firebase Project**

- Go to the Firebase Console.
- Click “Add Project” and follow the setup instructions.

1. **Register Your App**

- In the Firebase Console, navigate to **Project Settings > General**.
- Click “Add App” and select “Android”.
- Enter your **package name** (should match your AndroidManifest.xml).

1. **Download and Add**

- Download the `google-services.json` file.
- Place it in the `app/` directory of your Android project.

1. **Enable Cloud Messaging**

- In Firebase Console, go to **Cloud Messaging**.
- Enable **Cloud Messaging API (Legacy)**.
- Copy the **Server Key** (used for backend authentication).

2.2 Sending Push Notifications from Backend

The backend should send push notifications using Firebase Cloud Messaging (FCM). This requires making a request to the Firebase API.

Example Backend Code (Node.js & Express)

```
const admin = require('firebase-admin');
const serviceAccount = require('./firebase-admin-sdk.json');
admin.initializeApp({
    credential: admin.credential.cert(serviceAccount)
});

async function sendPushNotification(deviceToken, requestId) {
    const message = {
        data: { requestId },
        token: deviceToken
    };
    await admin.messaging().send(message);
}
```

Example API Request Using

```
curl -X POST "https://fcm.googleapis.com/v1/projects/YOUR_PROJECT_ID/messages:se
-H "Authorization: Bearer YOUR_SERVER_KEY" \
-H "Content-Type: application/json" \
-d '{
```

```
    "message": {
      "token": "USER_DEVICE_TOKEN",
      "data": {
        "requestId": "req-12345"
      }
    }
  }'
```

Step 3: Handling Push Notifications on Android

(Continues with Android implementation...)

2fa Fcm Android

Written by Prakash Ranjan

8 followers · 75 following

Android Developer

Follow

No responses yet

Write a response

What are your thoughts?

More from Prakash Ranjan

Prakash Ranjan

Handling JWT Token Expiration and Re-authentication in Android...

Introduction

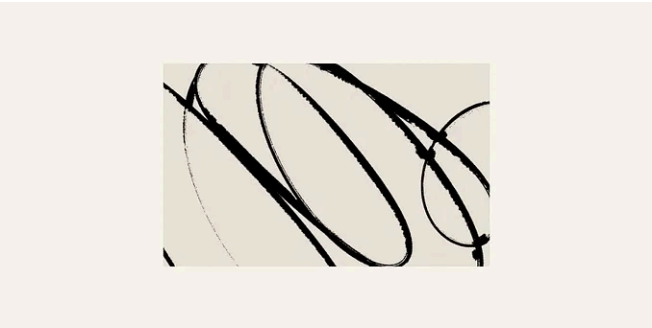
Feb 16 1 1

Prakash Ranjan

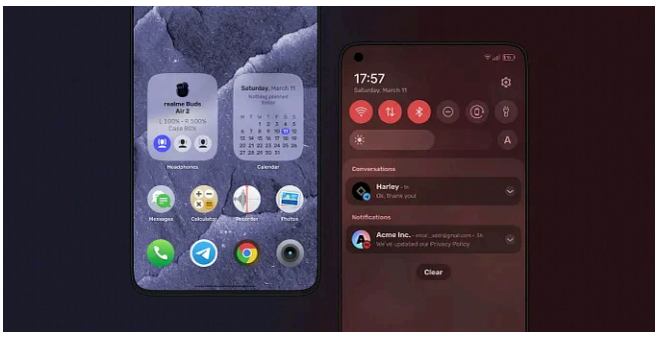
Best Practices for Coroutine Cancellation and Non-Cancellatio...

Kotlin Coroutines offer a powerful mechanism for handling concurrency while maintaining...

Mar 19



Prakash Ranjan



Prakash Ranjan

Dynamic UI in Android Using Kotlin

Building Multiple Applications in a Multi-Module Android Project

If you are planning to build two or more applications that shares some common logi...

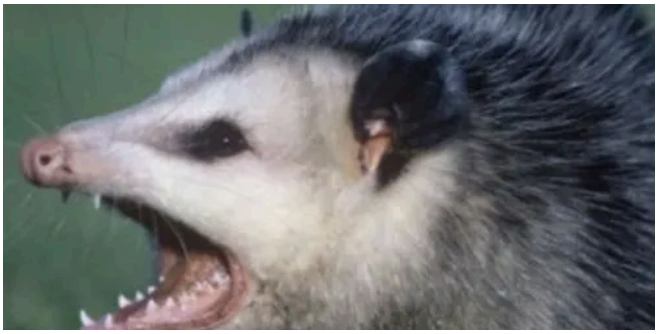
Apr 19 1

Building a dynamic UI in Android is crucial when working with content that changes...

Mar 11

See all from Prakash Ranjan

Recommended from Medium

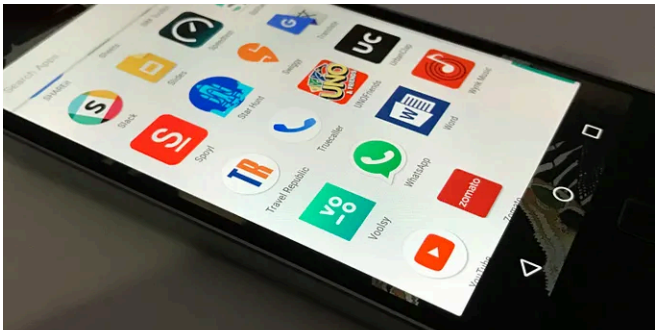


Louis Trinh

Understanding OAuth

OAuth (Open Authorization) is a standard authorization framework that allows...

May 26 1

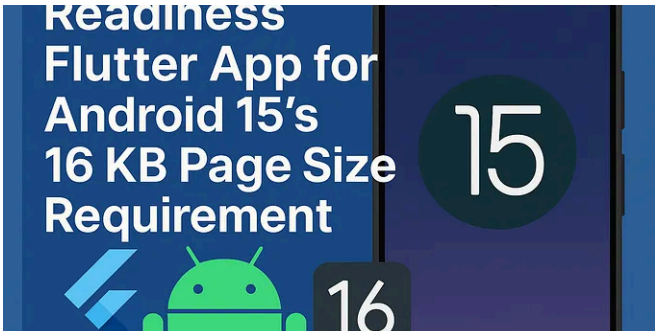


Nine Pages Of My Life

🌊 🔝 12 Firebase Libraries 💥

👨🏻‍💻 🧑‍🏫 Who this article is for?

Jul 22 13

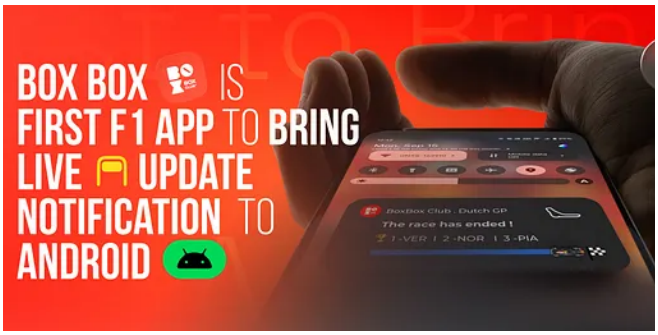


Santosh Botre

Ensuring Flutter App Readiness for Android 15's 16 KB Page Size

Google is introducing 16 KB memory page support for Android 15 (API level 35) on 64-b...

Sep 3 3



In Box Box Club® by Bhoomi Vaghasiya Gadhiya

Live F1 Update Notification in Android

Box Box makes real-time F1 race tracking possible on Android 16

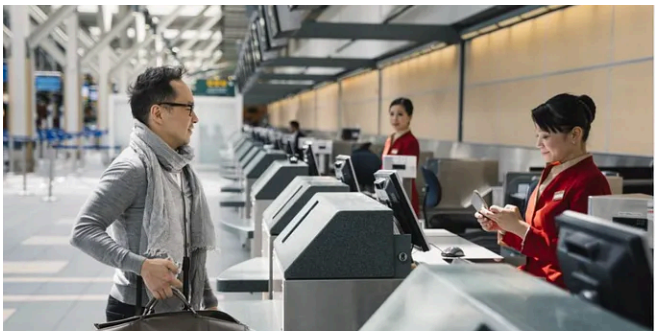
4d ago 51 1

In Stackademic by Sivavishnu

🔔 Understanding the Android Push Notification Flow Using FCM...

A Step-by-Step Walkthrough of Firebase Cloud Messaging, Token Management, and...

Sep 12



Sagara Gunathunga

mTLS and OAuth2—Client Authentication

Whenever we hit a website or web application, we simply trust the browser to verify its...

Apr 24 10

See more recommendations