

Long-form case study · 8 min read

Spotitube

A social app for music, movies, and series on Android (YouTube-Music-style), with a tested web admin console and a hardened backend.

ROLE

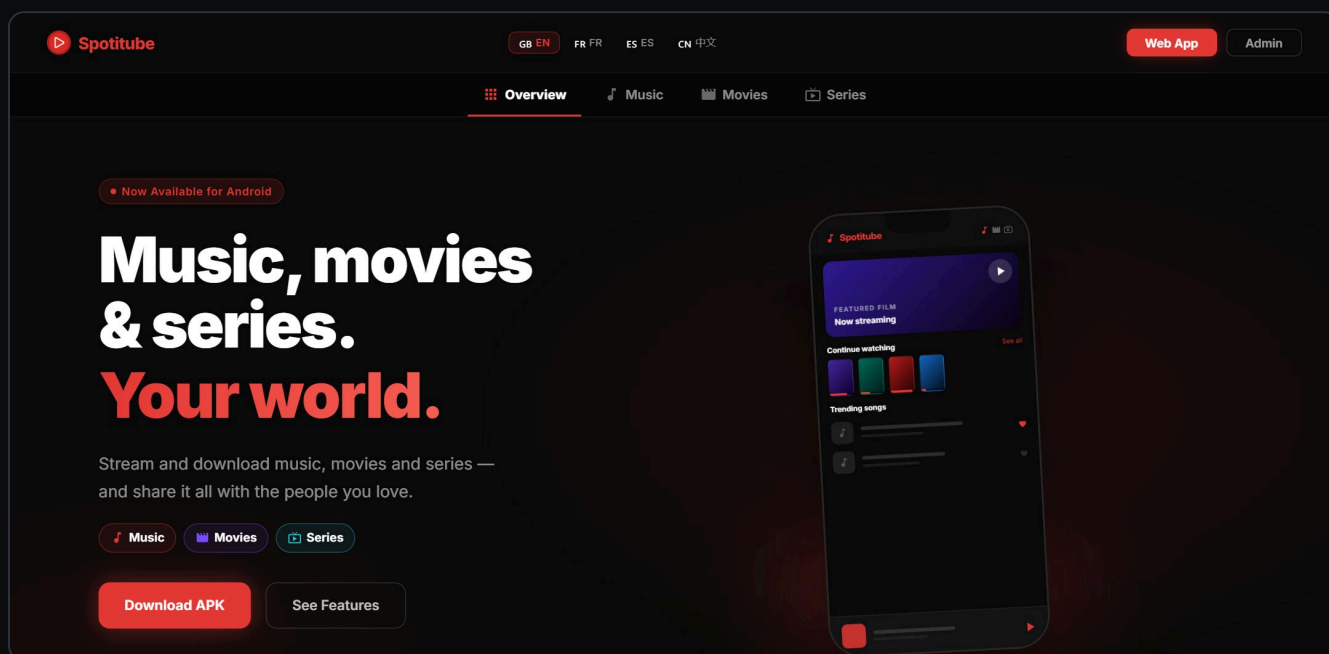
Design · architecture · build

PLATFORM

Android · Web admin

SCOPE

v1.4.3 · ~136 build-days · tested



Spotitube web — landing page

The product

Spotitube is a social app for **music, movies and series**: stream and download, build playlists, follow friends, see what they're playing, and watch together. It ships as an Android app and a React web app, with a web **admin console** to run the catalogue, and a Firebase backend with Cloud Functions.

It is my most mature product — **v1.4.3**, about **920 commits**, and a real test suite — and the one that taught me the most about running something, not just building it.

Architecture

- **Android**: a layered data / domain / UI app — Kotlin, Jetpack Compose, Hilt for dependency injection, Room, Retrofit and a Media3/ExoPlayer player.

- **Web:** React 18, TypeScript and Vite, with MUI and TanStack Query; the admin console adds Recharts, PWA support and i18n.
- **Backend:** Firebase Auth, Firestore, Storage and FCM, plus Cloud Functions for everything a client must not be trusted to do.
- **Media:** audio and video are served from Cloudflare R2 and a CDN rather than Firebase Storage.

One player UI, two engines

Films come from three sources — the CDN, a local download, or YouTube — and each platform plays them through **two engines**: a native player for CDN and local files, and an IFrame/WebView player for YouTube. The rule the whole player is built on is written into its contract: *the UI never asks which engine it has*. Each engine adapts itself to one shared state and control interface, and the shell consumes only that.

Before extending the player, I audited both engines **against the code** rather than from memory, and corrected the plan where it was wrong:

Capability	CDN / local	YouTube
Precise seek	Yes	Coarse — snaps to keyframes
Time updates	Events	Polled — the IFrame API has none
Picture-in-picture	Yes	No — PiP is a <video> API
Background playback (Android)	Yes	No — it runs in a WebView
Works offline	Local files only	No

One subscription, many devices

A subscription can be used on several devices, so something has to decide who keeps a seat and who gets locked out. That policy decides who pays, so I **extracted it from the Cloud Functions entry point into its own module** to test it without Firebase.

One detail shows the kind of thinking it needed: the app syncs on every launch and resume, and every write to a subscription re-fires the listener on every other device. So a session's heartbeat is persisted **at most once an hour** — far finer than the multi-day idle window it feeds, and without a write storm across devices.

Guarding a real budget

Some admin tools call the YouTube Data API, which allows **10,000 units a day** and charges **100 units for a single search**. An afternoon of enthusiastic searching can exhaust it, and when it runs out the error looks like a broken API key. The rate limits on those functions exist to make that impossible by accident — and to fail with a sentence that says what happened instead of sending someone off to debug a key that is fine.

Quality and security

- **139 test files** across the web app (Vitest), Cloud Functions (Node's test runner) and Android (JUnit).
- `npm run ci` runs the type-check, the tests and a production build; Firestore **security-rule tests** run on the emulator.
- A **security review** (June 2026) covered Firestore rules, callable-function auth and Storage rules. It found that Storage rules existed only in the console — not in source control — and fixed it with owner-only, image-only, 5 MB-capped rules. It also flagged a one-off migration endpoint guarded by a token in its URL, with concrete options to remove it.
- **Server-enforced user blocking** replaced a client-side filter: Cloud Functions mirror each block, unfollow both ways and drop collaborations; Firestore rules enforce it for comments, rooms and mentions; rule tests cover it. Planned and shipped as a 7-day plan.

What's next

Today the paywall protects the **apps**, not the audio files themselves. The fix is fully designed and scheduled before public launch: a Cloud Function mints short-lived HMAC-signed URLs after checking the subscription and the device seat, and a Cloudflare Worker in front of a private bucket verifies them — with the cache key set to the path only, so every user still shares the same edge cache. It is not built yet, and I don't claim it is.

What I learned

- Extract the policies that cost money or lock people out, and test them in isolation.
- Audit what the code actually does before planning on top of it.
- Write errors for the person who will read them.

BENOIT PRESLY NDONG OKE

✉ oke.benoit@gmail.com 🌐 okebenoithub.web.app <> github.com/OkeBenoitHub

🌐 linkedin.com/in/presly-benoit-ndong-oke-27987b117