

Étude de cas détaillée · 8 min de lecture

Pictotube

Une app sociale sombre, privée et multiplateforme dont l'unité est l'album — un moment entier de photos et de vidéos lié à un événement, un lieu et les personnes présentes.

RÔLE

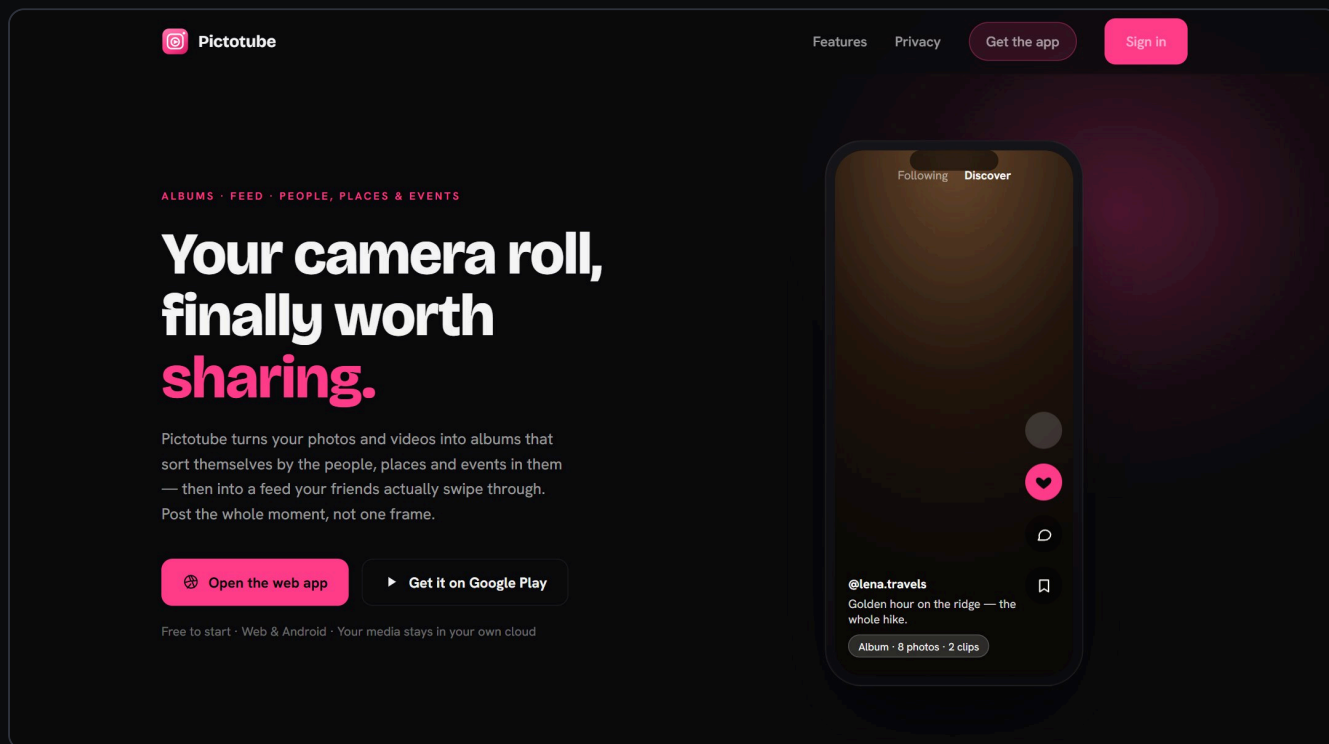
Conception · architecture · développement

PLATEFORME

Android · Web · Cloud

ÉTAT

En développement actif



Pictotube web — page d'accueil

Pourquoi Pictotube

Les photos et vidéos des moments importants finissent éparpillées entre pellicules et fils d'actualité. Pictotube est une application sociale privée, sombre et multiplateforme, construite autour d'une idée : l'unité, c'est l'**album** — un moment entier de photos et de vidéos rattaché à un **événement**, un **lieu** et les **personnes** présentes.

Elle a volontairement deux modes. On **parcourt** en mosaïque façon Pinterest (albums, recherche, favoris, profils) et on **s'immerge** dans un fil plein écran façon TikTok (fil d'accueil, visionneuse, diaporamas). Un

simple toucher fait passer de l'un à l'autre. Tout est sombre pour que le média soit le sujet ; l'accent rose est réservé aux actions.

Mon rôle

Je porte tout le produit : la conception, l'application Android (Kotlin, Jetpack Compose, Hilt, Room, WorkManager, Media3, CameraX, ML Kit sur l'appareil), l'application web (React, TypeScript, Vite), le Worker Cloudflare et les règles de sécurité Firebase. Le dépôt compte à ce jour environ **284 commits** et **63 fichiers de tests**.

Décision : un pipeline média hors de Firebase

Pictotube fonctionne volontairement **sans Cloud Functions**, afin de ne jamais dépendre de l'offre payante Blaze de Firebase. Les médias vivent dans deux buckets Cloudflare R2 (originaux et miniatures), et un petit Worker est le seul à pouvoir y donner accès.

Le client envoie un jeton d'identité Firebase ; le Worker le vérifie lui-même avec les clés publiques de Google — signature RS256, `aud`, `iss`, `exp`, `iat`, `auth_time` et un `sub` non vide — sans SDK Firebase ni compte de service. Il signe ensuite des URL de courte durée : **15 minutes** pour l'envoi, **1 heure** pour la lecture. Les clés d'objets sont générées côté serveur au format `<uid>/<uuid>` : un utilisateur ne peut écrire que sous son propre préfixe, et les identifiants R2 ne quittent jamais Cloudflare.

Décision : comment se construit le fil d'accueil

J'ai rédigé la fiche de décision du fil avant de l'écrire, pour que le choix soit délibéré et non dicté par la première requête venue. Trois architectures étaient possibles :

Option	Verdict	Pourquoi
Distribution à la lecture	Écartée	Firestore limite <code>in</code> à 30 valeurs : suivre 200 comptes impose 7 requêtes fusionnées, sans curseur commun.
Distribution à l'écriture, côté client	Écartée	Le fil de chaque abonné devrait être modifiable par le client — la permission rêvée d'un spammeur — et une app interrompue laisse une distribution partielle.
Distribution à l'écriture, dans le Worker	Retenue	Le Worker vérifie déjà les jetons : il relit la publication, confirme l'auteur et écrit une entrée par abonné avec des droits administrateur.

Les entrées du fil sont **lisibles par leur seul propriétaire, sans écriture client** : il n'existe aucun chemin modifiable par un client. Lire le fil devient une seule requête ordonnée avec un vrai curseur. J'ai volontairement reporté deux sujets qui ne paient qu'à une échelle que l'app n'a pas encore : un chemin hybride pour les comptes très suivis, et le classement.

Mesurer, puis corriger : coût du fil et fluidité

Une page du fil (20 entrées) coûte **environ 61 lectures de documents**, surtout deux lectures par publication. L'optimisation évidente — copier chaque publication dans l'entrée du fil — économiserait une quarantaine de

lectures par page. J'ai choisi de **ne pas** la faire : la copie est écrite par un compte de service et échappe à la règle de lecture de la publication ; une publication passée en « abonnés seulement », supprimée, ou dont l'auteur a bloqué le lecteur s'afficherait quand même. Relire la vraie publication est ce qui applique les règles de sécurité au fil.

Ce que j'ai corrigé : un démarrage à froid qui chargeait deux fois la première page, des cartes qui téléchargeaient les originaux en pleine résolution au lieu des miniatures, et une requête de suggestions lancée même quand le fil n'était pas vide.

Défilement d'un fil vidéo (dumpsys gfxinfo)	Images saccadées	Commandes de dessin lentes
Avant	61,4 %	735
Après : un seul ExoPlayer mutualisé entre les cartes	36,5 %	130

La correction : arrêter de construire un nouveau lecteur — initialisation du codec comprise — à chaque carte activée. Ce n'est pas terminé : l'image médiane reste à 23 ms pour un budget de 16,7 ms pendant la lecture vidéo, car une `TextureView` (qui respecte les coins arrondis) coûte plus cher à composer qu'une `SurfaceView`. Ce compromis est documenté, à réévaluer avec un contenu plus représentatif.

La confidentialité dès la conception

- La détection de visages s'exécute **sur l'appareil** avec ML Kit — les données faciales ne quittent jamais le téléphone.
- L'accès aux médias passe toujours par des URL signées de courte durée ; rien n'est public dans les buckets.
- Le fil est conçu pour que la visibilité soit décidée par les règles de sécurité, jamais par le client.

Où en est le projet

Pictotube est en développement actif (v0.1.0), avec des variantes de build dev et prod et une vraie connexion Firebase. Après le lancement, le projet prévoit des **espaces thématiques** — Anniversaire, Mariage et Mode — une seule app, un seul compte et un seul moteur de fil, chaque espace adaptant le parcours de création sur le même graphe albums / événements / personnes / lieux.

Ce que j'en retiens

- Écrire la fiche de décision avant le code : la forme du fil était figée avant qu'un seul écran en dépende.
- Mesurer avant d'optimiser, et garder les chiffres — la page à 61 lectures et les mesures d'images ont rendu les compromis concrets.
- Une propriété de sécurité peut valoir son coût : j'ai gardé les lectures supplémentaires parce que ce sont elles qui appliquent les règles.

BENOIT PRESLY NDONG OKE

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

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