

Long-form case study · 5 min read

Africonet

A professional + social super-app for the African diaspora — a jobs vertical, dual-origin identity, and a low-bandwidth-first design.

ROLE

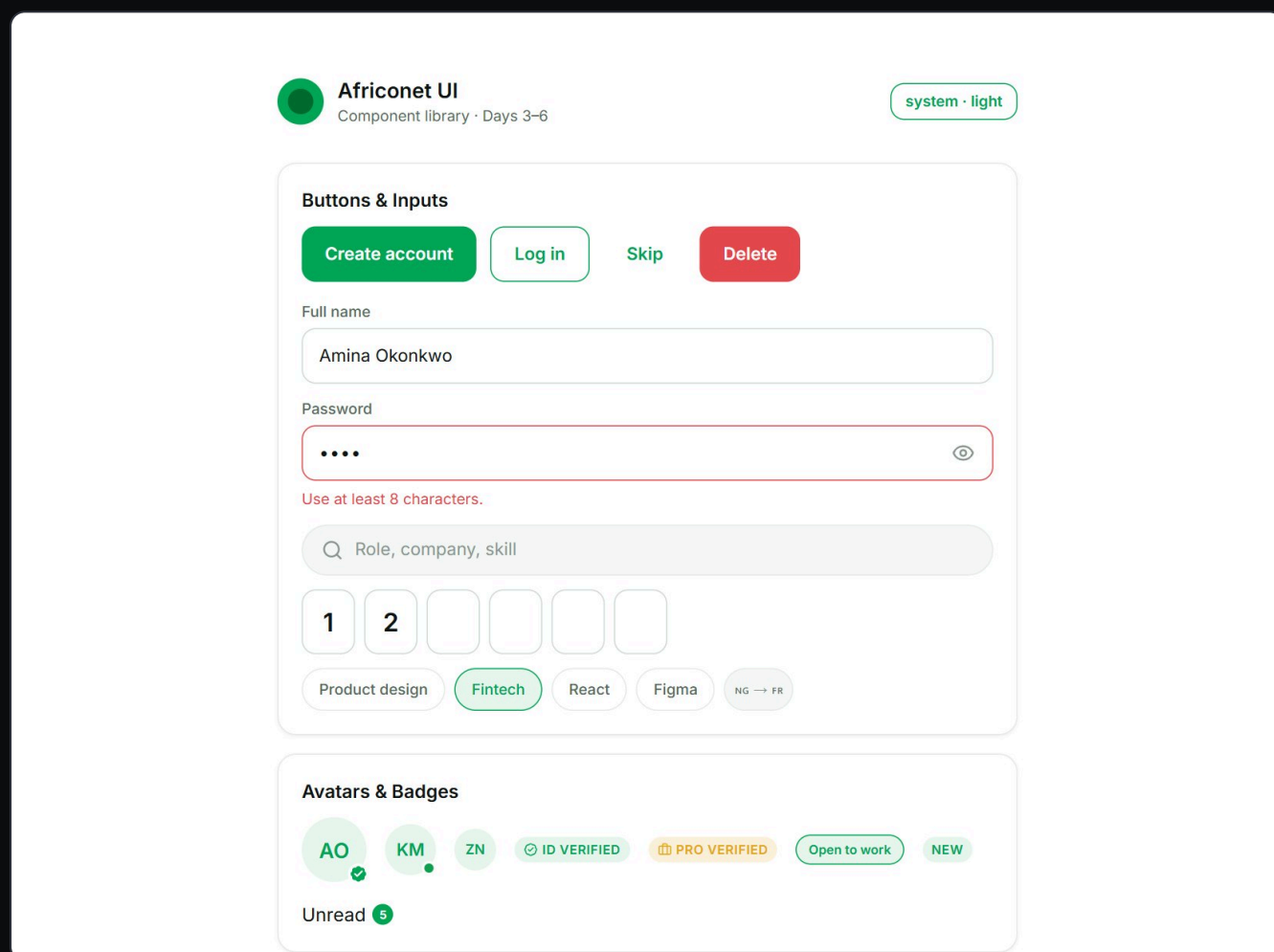
Design · architecture · build

PLATFORM

Web (Android planned)

SCOPE

In active development



Africonet — the UI component library (early stage)

Why Africonet

The African diaspora lives between two worlds — a country of origin and a country of residence — often on constrained bandwidth, with no single home for its professional and social life. Africonet is a professional

and social super-app for that community: connect with Africans nearby and across the world, build a profile and CV, follow people, chat, and find jobs.

The north star is a feeling: **LinkedIn's professional structure with WhatsApp's warmth and speed** — credible enough for recruiters, fast and friendly enough for daily conversation.

Scope, honestly

Africonet is at an early stage. What exists today is the **product design** for version 1 and the **design system and component library** it will be built from. The application features — accounts, profiles, the feed and the jobs vertical — are designed but not built yet.

Designing the product first

Version 1 is the **spine** — accounts, profile/CV, follow, feed, chat, search and notifications — plus the **jobs** vertical (online, on-site and freelance). Later verticals such as a marketplace, learning and payments each have their own briefs for later versions.

The design brief is the single contract that both the web and Android schedules reference by screen ID, so the two apps can't drift apart. Its principles:

- **Origin is everywhere.** Every profile, post and job carries where you are from and where you are — Africonet's signature, shown with flags and chips.
- **Fast and light.** Designed for low bandwidth: skeleton loaders, progressive images, a low-data mode and real empty states — never a blank white screen.
- **One system, two shells.** The same tokens and components: a bottom-navigation shell on Android, a left rail and top bar on the web that collapses to bottom navigation on phones.
- **Accessible by default.** WCAG AA contrast in both themes, 44 px minimum touch targets, full dark mode.

What's built: the design system

The web app runs on **Next.js 16** (App Router), **React 19**, TypeScript and **Tailwind CSS 4**. The first six days of the web schedule are done:

- Every colour token, light and dark, taken exactly from the approved prototype and exposed as CSS variables and a Tailwind `@theme`, with a theme provider that sets the theme before first paint so there's no flash of the wrong one.
- A type scale, radius tokens and layout primitives.
- A **25-component library**: buttons, text and password fields, a one-time-code input, search, chips including the origin chip, avatars with presence and verification badges, post, job, profile and company cards, list rows, badges, tabs, sheets, toasts, skeletons and empty states.
- A responsive app shell: sidebar, top bar and bottom navigation.

One accessibility finding is recorded rather than hidden: white text on the brand green reaches about **3.2:1** — enough for large text and UI components under WCAG AA, not for small text — so the darker pressed

green is noted for small text on green.

What's next

The next steps on the web schedule are the Firebase data model and security rules, authentication, and an onboarding flow that captures the dual origin — then the profile/CV, the feed and chat, and the jobs vertical. The Android app follows the same brief with Jetpack Compose.

What I learned

- A written design contract, referenced by screen ID, keeps two platforms honest.
- Build the system before the screens: every later feature reuses these components.
- Record accessibility trade-offs where the next person will see them.

BENOIT PRESLY NDONG OKE

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

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