Recipe App Development Plan (Vertical Slices with Clerk)

Phase A — Auth & Accounts

- Backend: Integrate Clerk SDK in Next.js, set up Clerk-protected API routes, link Prisma User to Clerk userId.
- UI: Build sign-up/login/logout pages with Clerk’s React components, add nav bar with conditional 'Sign in / Sign out'.
- DoD: You can register/login in the web UI and see your identity reflected in the app.

Phase B — Recipes Data Model & Global Feed

- Backend: Prisma Recipe model with authorId; Public API routes: GET /api/recipes (search), GET /api/recipes/random.
- UI: Home page with search bar (debounced input), randomized recipe grid/cards that load on first visit and refresh on 'Shuffle'.
- UI: Empty state and loading skeletons.
- DoD: Logged-out users see global feed; search filters results in real time.

Phase C — User-Owned Recipes

- Backend: Authenticated API routes for /api/me/recipes CRUD (Clerk protected), ownership checks via authorId.
- UI: 'My Recipes' page (requires login), Recipe form (create/edit), delete buttons, owner-only controls.
- DoD: Logged-in user can create recipes, see them in 'My Recipes', edit, and delete them.

Phase D — Randomization & Search Enhancements

- Backend: Optimize random queries for larger datasets, improve search with Postgres full-text search if needed.
- UI: Better search UX (highlight matches, debounce indicator), 'Shuffle again' button with animation.
- DoD: Fast, varied random results; polished search UX.

Phase E — Quality, Security & Analytics

- Backend: Add input validation, rate-limiting, error logging, track usage (search queries, recipe creates).

- UI: Show error states (toast/snackbar), add analytics events on search and recipe clicks.
- DoD: Errors handled gracefully; usage signals visible.

Phase F — Deployment & Mobile Foundation

- Backend: Deploy to Vercel with Clerk + Neon secrets configured, staging → prod workflow.
- UI: Verify all flows on deployed site, start Expo app with Clerk auth and global feed screen.
- DoD: Web app live; Expo app can log in and display random feed.

Why Vertical Slices?

- Each feature is tested end-to-end (DB ↔ API ↔ UI).
- Avoids half-built backend waiting for UI, or vice versa.
- Always have a working slice of the product to demo or test.