KYX PLATFORM TECH STACK

AT A GLANCE

â¢ Dual surface: ‘landing-page/‘ (Next.js marketing + community UI) and ‘demo-game/‘ (Python/pygame platformer shipped through pygbag)
â¢ Shared config JSON generated via the Madlib lab UI or ‘/api/generate-config‘, consumed by both the Python runtime and Next.js APIs
â¢ Supabase handles PostgreSQL data, authentication, and storage for published WebAssembly bundles

FRONTEND (LANDING-PAGE/)

â¢ **Next.js 14 (App Router) & React 18** for server components, streaming routes, and client-side interactivity
â¢ **TypeScript** across app routes, API handlers, and shared schema utilities
â¢ **Tailwind CSS** plus **shadcn/ui** primitives for layout, theming, and accessible components
â¢ **Lucide React** icons and **next-themes** for light/dark mode support
â¢ Madlib lab UI on ‘/lab‘ that emits config JSON and posts to the build pipeline
â¢ Community surfaces: ‘/community‘, ‘/dashboard‘, ‘/auth/*‘ built atop Supabase auth helpers and route middleware

BACKEND & APIS

â¢ Next.js API routes (‘app/api/*‘) for game CRUD, build queue orchestration, prompt-to-config generation, comments, reports, and moderation
â¢ Supabase client for server-side actions plus RLS-protected access from the browser
â¢ Rate limiting + content filtering utilities (in-memory guards plus heuristics on user-generated text)
â¢ Build processor endpoints (‘/api/games/build‘, ‘/api/games/process-build‘) shell out to Python, monitor status rows, and upload bundles to storage

GAME ENGINE (DEMO-GAME/)

â¢ **Python 3.12** runtime with ‘pygame-ce‘ for rendering, physics, and input
â¢ **pygbag** compiles the platformer to WebAssembly for iframe embedding
â¢ ‘tools/build_game.py‘ CLI wraps config writing, pygbag execution, and artifact copying
â¢ ‘game_config.json‘ merges Madlib output with engine defaults at launch

DATA & INFRASTRUCTURE

â¢ **Supabase PostgreSQL** tables: ‘profiles‘, ‘games‘, ‘build_queue‘, ‘likes‘, ‘comments‘, ‘reports‘
â¢ **Supabase Auth** (email/password + GitHub OAuth) with RLS policies enforced per table
â¢ **Supabase Storage** ‘game-bundles‘ bucket holds public WebAssembly builds referenced by ‘/community/[username]/[slug]‘
â¢ Recommended deployment: Next.js on Vercel + worker/queue (Railway/Fly/etc.) for long-running pygbag builds

DEVELOPER TOOLING & WORKFLOW

â¢ Node 18+ toolchain: ‘npm run dev‘, ‘npm run lint‘, ‘npm run build‘
â¢ Python requirements pinned via ‘requirements.txt‘; local loop is ‘python main.py‘ for the game and ‘pygbag main.py‘ for web builds
â¢ Automated workflow summary:

1. Author lore in Madlib lab or via ‘/api/generate-config‘
2. Persist config, queue build with ‘/api/games/build‘, and monitor ‘build_queue‘
3. Python worker runs pygbag, uploads artifacts, and marks the game ‘built‘
4. Publish through ‘/api/games/publish‘ to surface in community feeds