# Product Requirements Document (PRD)

**Project Name:** *maicle.co.uk*\ **Owner:** You\ **Purpose:** Build a secure, scalable web app on **Next.js**, powered by **Sanity** for content management, **Mux** for video streaming, **Supabase** for authentication and core data, **Stripe** for payments, **MailerLite** for marketing automation, and **PostHog** for analytics.

---

## 1. Tech Stack Overview

### Frontend

- **Framework:** Next.js (App Router, React Server Components)
- **Styling:** Tailwind CSS with CSS variables (design tokens)
- **UI Primitives:** Radix UI (accessibility) + shadcn/ui patterns
- **Motion:** Framer Motion (only where needed)
- **Icons:** lucide-react
- **Content Rendering:** Sanity for CMS content
- **Video Player:** integration throught Sanity
- **Forms:** React Hook Form + Zod (validation)
- **Analytics:** PostHog JS client (with consent gating)

---

### Backend

- **Auth & Database:** Supabase (Postgres + Row Level Security)
- **CMS:** Sanity (hosted)
- **Video Hosting:** Mux (Direct Uploads + signed playback)
- **Payments:** Stripe Checkout + Billing Portal
- **Transactional Email:** Resend
- **Marketing Automation:** MailerLite (via API & webhooks)
- **Analytics:** PostHog Node client for server-side events
- **Hosting:** Vercel (serverless functions, ISR caching)

---

### Design System

- **Tokens:** CSS variables for colors, spacing, radii, typography
- **Tailwind Config:** maps directly to tokens
- **Component Tiers:**
- **Primitives:** Button, Input, Select, Checkbox, Dialog, Tooltip, Toast, Badge
- **Patterns:** PageHeader, FormRow, EmptyState, PaywallCTA, Prose
- **Layouts:** MarketingLayout, AppShellLayout, SettingsLayout
- **Documentation:** Storybook (with a11y testing and component playground)
- **Testing:**
- Unit: React Testing Library + Vitest

- Visual Regression: Playwright + Percy
- Accessibility: axe-core in Storybook

---

# 2. Phased Development Plan

---

### Phase 0: Project Setup & Infrastructure

**Objective:** Establish a production-ready development environment and CI/CD pipeline.

**Tasks:**

- Create Next.js app with TypeScript.
- Configure Vercel environments: `dev`, `staging`, `prod`.
- Install and configure Tailwind CSS with design tokens.
- Initialize Supabase project (Postgres + Auth).
- Initialize Sanity project (dataset: `production`).
- Set up GitHub Actions for build, lint, and test.
- Integrate Prettier, ESLint, Husky (pre-commit hooks).

**Security Requirements:**

- All secrets in Vercel environment variables (never in code).
- Enable HTTPS everywhere (Vercel default).
- Lock Supabase RLS to deny-all by default.

**Optimization Requirements:**

- Cold start < 500ms for serverless routes in staging.
- Tailwind purge config to eliminate unused CSS.

---

### Phase 1: Authentication & Core Database Schema

**Objective:** Implement secure login and core tables for profiles, entitlements, subscriptions.

**Tasks:**

- Enable Supabase Auth providers: Email, Google, Facebook.
- Create tables:
- `profiles`
- `entitlements`
- `subscriptions`
- Add `getSessionUser()` server helper.
- Implement RLS policies for each table.

- Build signup/login/logout UI using design system.

**Security Requirements:**

- Row Level Security enabled for all tables.
- JWT expiry ≤ 1 hour; refresh token flow in place.
- Email confirmation required for new accounts.

**Optimization Requirements:**

- Use Supabase server-side helpers to avoid client-heavy auth state.
- Cache user profile in React Query or SWR (stale-while-revalidate).

---

## Phase 2: Sanity Content Integration

**Objective:** Pull CMS content into the app with live preview and ISR.

**Tasks:**

- Define Sanity schemas: `post`, `videoPost`, `page`.
- Integrate `next-sanity` and GROQ queries.
- Add Sanity webhook → `/api/webhooks/sanity` → `revalidatePath()` for updated slugs.
- Implement PortableText renderer mapped to design system typography.
- Build Marketing pages from Sanity content.

**Security Requirements:**

- Use Sanity tokens only in server-side code.
- Verify webhook signatures from Sanity.
- Prevent draft content from leaking without preview token.

**Optimization Requirements:**

- Use ISR for all CMS-driven pages (`revalidateTag` per document type).
- Use Sanity `@sanity/image-url` for responsive images with `next/image`.

---

## Phase 3: Mux Video Integration

**Objective:** Add secure video streaming with Mux uploads and playback.

**Tasks:**

- Create `/api/mux/create-direct-upload` → returns signed upload URL.
- Handle `video.asset.ready` webhook → store `mux_asset_id` in Supabase.
- Add `<mux-player>` with signed playback IDs for gated videos.

• Integrate with paywall logic (check entitlements before returning playback ID).

**Security Requirements:**

• Store Mux API keys server-side only.
• Verify Mux webhook signatures.
• Generate signed playback URLs with expiry for members-only content.

**Optimization Requirements:**

• Use Mux's poster images for video thumbnails.
• Lazy-load player JS; use native `<video>` fallback for low-bandwidth.

---

## Phase 4: Payments & Entitlement Automation

**Objective:** Accept payments and auto-grant access to gated content.

**Tasks:**

• Implement `/api/checkout` (Stripe Checkout).
• Implement `/api/stripe/webhook`:
• `checkout.session.completed` → create subscription + entitlement.
• `subscription.updated/deleted` → update/revoke entitlement.
• Add `/api/billing/portal` for Customer Portal.
• Add Pricing page and PaywallCTA.

**Security Requirements:**

• Webhook raw-body parsing for Stripe signature verification.
• Test webhook replay protection.
• Entitlement granting only via trusted webhook events.

**Optimization Requirements:**

• Cache pricing data from Stripe in Supabase (reduces API calls).
• Show cached entitlement state in UI instantly; reconcile async via webhook.

---

## Phase 5: Email & Marketing Automation

**Objective:** Send transactional emails and sync marketing consent.

**Tasks:**

• Integrate Resend for:
• Welcome emails

- Payment receipts
- Passwordless login links
- Integrate MailerLite:
- On signup (with consent) → add to audience.
- Webhook to `/api/mailerlite/webhook` → sync unsubscribes.

**Security Requirements:**

- Validate webhook signatures from MailerLite.
- Don't send transactional emails to unsubscribed marketing users.

**Optimization Requirements:**

- Batch MailerLite updates (avoid API rate limits).
- Use transactional email templates for speed.

---

## Phase 6: Analytics & Consent Management

**Objective:** Track product metrics and comply with privacy laws.

**Tasks:**

- Integrate PostHog (client + server).
- Track:
- `signup_completed`
- `checkout_started`
- `checkout_succeeded`
- `video_played`
- Implement cookie consent banner (gates analytics until accepted).

**Security Requirements:**

- No PII in event properties.
- Respect `Do Not Track` browser setting.

**Optimization Requirements:**

- Use PostHog EU data residency.
- Sample replays at ≤10% of sessions.

---

## Phase 7: Design System Completion

**Objective:** Finalize design system for consistent UX.

**Tasks:**

- Finalize tokens: colors, typography, spacing.
- Complete primitives and patterns.
- Build Storybook with a11y testing.
- Write usage documentation.

**Security Requirements:**

- Audit external component dependencies for vulnerabilities.
- Ensure all components pass a11y tests.

**Optimization Requirements:**

- Tree-shakeable components.
- Avoid unnecessary `use client` where possible.

---

# 3. Completion Criteria by Phase

For **each phase**:

1. All tasks marked as complete in GitHub project board.
2. All **security requirements** tested and passed.
3. All **optimization requirements** tested and passed.
4. Code merged to `main` with CI build + deploy green.
5. Staging smoke test performed.

---

# 4. Development Process & Environments

## 4.1 Delivery strategy

- **Branching:** Trunk-based development with short-lived feature branches (`feat/*`, `fix/*`).
- **Reviews:** Every PR requires code review + automated checks (build, typecheck, lint, tests, preview deploy link).
- **Feature flags:** Ship behind flags (PostHog) to decouple deploy from release.
- **Versioning:** App is continuously deployed; the **design system** package uses Changesets + semver (pre-releases on `next`).

### 4.2 Environments

| Env | Purpose | Domains | Data | Stripe/ Mux/ MailerLite/ PostHog | Sanity dataset | Supabase | No |
|---|---|---|---|---|---|---|---|
| **Local** | Dev on laptop | `localhost:3000` | Fake/seed | Test/ sandbox keys | `development` | Local (Docker via `supabase start`) | Us CL as |
| **Preview** | Per-PR QA | `vercel.app` preview URL | Ephemeral | Test/ sandbox keys | `development` (read-only) | Shared staging DB schema or ephemeral | Au or |
| **Staging** | Pre-prod UAT | `staging.maicle.co.uk` | Staging subset | Test/ sandbox keys | `staging` | Managed (RLS on) | Lo te co |
| **Production** | Live | `maicle.co.uk` | Real | Live keys | `production` | Managed (RLS on) | Ol + |

**Rule:** Never mix test/live credentials across environments. Each provider gets its own project/workspace per environment.

### 4.3 Containerization (Docker or not?)

- **Next.js app:** Run **natively** with `pnpm dev` for fastest HMR. Provide a Dockerfile for CI/build parity and for contributors who prefer containers.
- **Supabase:** Use **Docker** locally via `supabase start` (Postgres, Auth, Storage, Studio). Commit migrations to version control.
- **Sanity Studio:** Run locally (`sanity dev`) or embed as a route in the app; no docker needed.
- **Ancillaries:** No docker for Stripe/Mux/MailerLite—use their CLIs/Webhooks.

**Dockerfiles provided:**

- `apps/web/Dockerfile` (prod image with `next build` + standalone output).
- `docker-compose.yml` (optional) to start **web + supabase local** together.

### 4.4 Local development setup

1. **Tooling:** Node LTS, `pnpm`, Git, Stripe CLI, Supabase CLI, Sanity CLI, `direnv` (or Doppler) for envs.
2. **Bootstrap:** `pnpm i && supabase start && pnpm dev`.
3. **Env management:**
4. `.env.development.local` for local secrets.
5. Vercel envs for preview/staging/prod; pull with `vercel env pull`.

6. **Seed data:** `pnpm db:seed` loads minimal users, content, and example video references (Mux test asset IDs).
7. **Webhooks:**
8. Stripe: `stripe listen --forward-to localhost:3000/api/stripe/webhook`.
9. Sanity: point staging webhook to preview domain; local via `ngrok` if needed.
10. Mux: use test webhooks or poll asset status in dev.

### 4.5 Secrets & configuration

- All secrets live in **Vercel Environment Variables** or a secret manager (Doppler/1Password) in dev.
- Prefix only safe client values with `NEXT_PUBLIC_`.
- Rotate keys on role changes; restrict vendor API keys by domain/callback URLs.

### 4.6 Database migrations & data policy

- Use Supabase migrations (`supabase db diff` / `migrate`).
- One-way schema changes only on `main`; rollbacks via forward fix.
- Red-gate prod access: migrations run via CI, not from laptops.
- **Data separation:**
- Prod data never copied to lower envs.
- Synthetic seeds for staging.
- **Backups:** Automatic daily backups; test restore quarterly in staging.

### 4.7 CI/CD pipeline (GitHub Actions + Vercel)

**PR pipeline:**

1. Install deps + cache
2. Typecheck (TS), ESLint
3. Unit tests (Vitest)
4. Build Next.js (no emit on types)
5. Storybook build (design system)
6. Playwright component smoke (headless)
7. Upload artifacts + comment preview URLs

**Merge to main:**

1. Run migrations against **staging**
2. Deploy to **staging** (Vercel)
3. Run e2e smoke on staging (Playwright): auth → checkout (Stripe test) → gated content
4. Manual approval (release manager)
5. Promote to **production** (Vercel)
6. Post-deploy checks: health, error rate, Core Web Vitals

### 4.8 Testing strategy

- **Unit tests:** utilities, reducers, pricing logic, entitlement guards.
- **Component tests:** UI primitives in Storybook with axe a11y.

- **E2E tests (Playwright):**
- Auth flows (email, Google)
- Checkout (Stripe) with webhook simulation
- Sanity content publish → page revalidation
- Mux video playback gating
- **Performance tests:** Lighthouse CI on preview; K6 (optional) on staging critical routes.
- **Security tests:**
- RLS policy tests (PostgREST requests)
- SSRF/redirect tests on OAuth callbacks
- Webhook signature verification tests

### 4.9 Observability & operations

- **Errors:** Sentry (server + client) with source maps.
- **Logs:** Vercel + vendor dashboards; optional Logtail.
- **Metrics:** PostHog dashboards for activation and conversion.
- **Uptime:** Healthcheck route `/api/health` + Statuspage (optional).
- **Alerts:** Slack/Email for webhook failures, Stripe invoice failures, error rate spikes.

### 4.10 Access control & least privilege

- Separate vendor projects/keys per environment.
- Principle of least privilege on team roles (Vercel, Supabase, Sanity, Stripe, Mux).
- Protected GitHub environments (required reviews, secret access limited).

### 4.11 Release & rollback

- **Release:** Merge to `main` → staged deploy → promote to prod after smoke passes.
- **Rollback:** Vercel instant rollback to previous build + revert of the last migration (apply compensating migration script).
- **Feature kill-switch:** Use PostHog flags to disable features without redeploying.

### 4.12 Performance budgets & checks

- **Budgets:** TTFB < 300ms (staging), LCP < 2.5s, CLS < 0.1 on core pages.
- **Build size:** First-load JS < 180KB for marketing, < 250KB for app shell.
- **Images:** next/image + Sanity CDN with smart crops.
- **Caching:** ISR + `revalidateTag()`; CDN headers on API responses where safe.

### 4.13 Phase exit checklists (additive to PRD phase criteria)

For each PRD phase, before marking **Done**:

- **Security:**
- Secrets only in env store; no plaintext in repo.
- Webhooks verify signatures; replay guarded.
- RLS policies tested with negative cases.
- OAuth redirect URIs locked to env domains.
- **Optimization:**

• Lighthouse ≥ 90 (perf & a11y) on target pages.
• API p95 < 400ms in staging under nominal load.
• No client bundle regressions beyond budgets.
• **Observability:**
• Sentry DSNs configured; sample errors visible.
• Alerts configured for Stripe/Mux/Sanity webhook failures.
• **Docs:**
• README updated (run, test, deploy steps).
• Changelog (Changesets) updated for UI package.