# Executive Summary — What this app will do

A multi‑tenant SaaS for law firms with a **Super‑Admin** (you) above **Law Firms (tenants)** → **Workspaces/Teams** → **Users & Roles**. Core capabilities: case management, document storage with versioning and client‑sharing, calendars & court dates, automated reminders and post‑hearing tasks, task management, client portal (read‑only), and a public API to show active cases on the firm’s website. Security is enforced via **RBAC** and **PostgreSQL Row‑Level Security (RLS)** to isolate tenants. Billing is firm‑level (plans/limits, optional usage metering). Frontend: **Next.js + TypeScript**. Backend: **Postgres + Prisma**, object storage for files, **Stripe** for billing, **Resend** for email. No dates/timelines here—just tiny, actionable tickets.

Every ticket below states **Purpose → Steps → Acceptance Criteria → Deliverables**. You can paste these 1:1 into your PM tool.

# Phase 0 — Bootstrap & Tenant Security Foundation (prereq for all)

## 0.1 Initialize mono‑repo

**Purpose:** Create a clean scaffold to build on. **Steps:** Create repo; add apps/web, apps/api (or a single Next.js app); add packages/config for shared ESLint/TS. **Acceptance:** Repo installs with pnpm i/npm i and lints without errors. **Deliverables:** Repo URL; README with setup.

## 0.2 Configure TypeScript + ESLint + Prettier

**Purpose:** Consistent, strict code quality. **Steps:** TS strict on; ESLint rules for security/imports; Prettier config; pre‑commit hook. **Acceptance:** pnpm lint passes; pre‑commit runs on staged files. **Deliverables:** .eslintrc, .prettierrc, Husky hook.

## 0.3 Env & secrets management

**Purpose:** Safe configuration across envs. **Steps:** .env.example with required vars; add zod runtime validation; GitHub Actions secret placeholders. **Acceptance:** App fails fast on missing envs with helpful error. **Deliverables:** .env.example, config validator.

## 0.4 Database provisioning

**Purpose:** Stand up Postgres. **Steps:** Create Postgres instance; create app user; network/IP allowlist; connect string in secrets. **Acceptance:** CLI connects; SELECT 1 works from local and CI. **Deliverables:** DB host, user, database, readme notes.

## 0.5 Prisma init + baseline migration

**Purpose:** Establish schema & migrations. **Steps:** Add Prisma; model core tables (platform\_users, law\_firms, users, roles, user\_roles); run prisma migrate dev → baseline. **Acceptance:** DB has baseline tables; Prisma Client generates. **Deliverables:** schema.prisma, first migration.

## 0.6 Add tenant columns

**Purpose:** Prepare RLS. **Steps:** Add law\_firm\_id to all tenant‑scoped tables; FK to law\_firms. **Acceptance:** Migrations apply; integrity checks pass. **Deliverables:** Migration diff, ERD image.

## 0.7 Enable PostgreSQL RLS

**Purpose:** Enforce tenant isolation in DB. **Steps:** ALTER TABLE ... ENABLE ROW LEVEL SECURITY; add policies per table. **Acceptance:** Cross‑tenant selects/updates fail under tests; same‑tenant succeed. **Deliverables:** SQL policy migrations; test script.

## 0.8 Per‑request tenant context

**Purpose:** Set the current tenant for each DB call. **Steps:** In API middleware, set SET app.current\_law\_firm\_id = '<uuid>' at connection/request scope. **Acceptance:** Queries only return rows for the current tenant. **Deliverables:** DB client util with context setter; unit tests.

## 0.9 Auth (NextAuth) + session claims

**Purpose:** Log in users and carry roles/tenant. **Steps:** Configure providers (email/password or OAuth); JWT/session callbacks to embed user\_id, role, law\_firm\_id. **Acceptance:** Login works; session exposes claims; protected page redirects when unauth. **Deliverables:** /api/auth/\* routes; login page.

## 0.10 RBAC constants & guards

**Purpose:** Guard routes & actions by role. **Steps:** Define roles (super\_admin, owner, lawyer, assistant, secretary, client); add server‑side guards + UI route guards. **Acceptance:** Users see only authorized nav and endpoints. **Deliverables:** authz.ts helper; test matrix (role × action).

## 0.11 Seed script: initialize\_law\_firm

**Purpose:** One‑click firm setup. **Steps:** Script creates default workspace, roles, settings, categories. **Acceptance:** Running script yields a usable tenant with an Owner user. **Deliverables:** scripts/initialize\_law\_firm.ts.

## 0.12 Super‑Admin console (MVP)

**Purpose:** Manage tenants. **Steps:** List/create/suspend firms; impersonate with reason; audit log entries. **Acceptance:** Super‑Admin can enter any firm context; audit shows who/when/why. **Deliverables:** /admin/firms UI + API routes.

## 0.13 Email provider wiring (Resend)

**Purpose:** Send transactional emails. **Steps:** Configure domain, DKIM/SPF; simple send API; error logging. **Acceptance:** Test email delivers; events logged. **Deliverables:** /api/email/test endpoint; templates folder.

## 0.14 Object storage wiring (R2/S3‑compatible)

**Purpose:** Store documents cheaply. **Steps:** Bucket per env; server‑generated presigned URLs; private by default. **Acceptance:** Upload via presigned URL; object not publicly listable. **Deliverables:** Storage client; upload example route.

## 0.15 Global error & rate limiting

**Purpose:** Safety and stability. **Steps:** API error formatter; 429s for auth/public routes; per‑firm quotas skeleton. **Acceptance:** Errors are structured; abusive calls throttled. **Deliverables:** Middleware + tests.

**Exit of Phase 0:** Super‑Admin can create a firm; Owner can log in; RLS verified; email/storage wired.

# Phase 1 — Cases (core engine)

## 1.1 Cases schema

**Purpose:** Persist cases and relations. **Steps:** Create cases, case\_status\_history, case\_parties, case\_lawyers. **Acceptance:** FK constraints; cascade rules defined. **Deliverables:** Migration; ERD diff.

## 1.2 Case code generator

**Purpose:** Human‑friendly IDs per firm. **Steps:** Deterministic code (e.g., NF‑2025‑000123). **Acceptance:** Unique per firm; sortable by created\_at. **Deliverables:** Service util + tests.

## 1.3 Case CRUD API + RLS

**Purpose:** Create/update/archive with isolation. **Steps:** REST/Route handlers for CRUD; enforce RBAC; write status history on changes. **Acceptance:** CRUD works; unauthorized actions blocked. **Deliverables:** /api/cases/\* routes; tests.

## 1.4 Parties model & link to client users

**Purpose:** Track plaintiffs/defendants/others. **Steps:** Party types; optional link to users where role=client. **Acceptance:** A client user only links to their parties. **Deliverables:** Migrations; UI picker.

## 1.5 Lawyer assignment

**Purpose:** Who owns the case. **Steps:** Primary/secondary lawyer; ownership checks in API. **Acceptance:** Only Owner/assigned staff can reassign. **Deliverables:** API + UI control.

## 1.6 Statuses & notes

**Purpose:** Track lifecycle + context. **Steps:** Status enum (Open/Active/Stayed/Closed); rich notes with mentions. **Acceptance:** History entries created on change. **Deliverables:** Notes tab; history timeline.

## 1.7 Search & filters

**Purpose:** Find cases fast. **Steps:** Filter by status/lawyer/party/tags; text index on title/code. **Acceptance:** Query returns expected rows; perf < 200ms on typical dataset. **Deliverables:** List view with filters.

**Exit:** Firms can manage and find cases confidently.

# Phase 2 — Documents (uploads, versioning, sharing)

## 2.1 Document schema

**Purpose:** Attach files safely. **Steps:** documents, document\_versions, document\_shares (client\_visible). **Acceptance:** New version rows on re‑upload; back‑refs to case. **Deliverables:** Migration + ERD.

## 2.2 Presigned upload flow

**Purpose:** Efficient, secure uploads. **Steps:** Server returns presigned PUT; client uploads directly. **Acceptance:** Upload succeeds without exposing storage creds. **Deliverables:** /api/documents/presign route; demo page.

## 2.3 Basic virus‑scan hook (stub)

**Purpose:** Future safety. **Steps:** After upload, enqueue scan job (mock for now). **Acceptance:** Job row created; status visible. **Deliverables:** Queue table; worker stub.

## 2.4 Version history UI

**Purpose:** See file evolution. **Steps:** Show versions, uploader, checksum, created\_at; restore button. **Acceptance:** Restore creates a new head version. **Deliverables:** Case → Documents tab.

## 2.5 Client‑sharing flag + watermarks

**Purpose:** Share with clients safely. **Steps:** client\_visible toggle; expiring download URLs; watermark PDF if client. **Acceptance:** Client sees only shared docs; watermark applied. **Deliverables:** Download API; watermark util.

**Exit:** Versioned docs with safe client sharing.

# Phase 3 — Calendars & Court Dates

## 3.1 Calendar schema

**Purpose:** Events with attendees and reminders. **Steps:** Tables: calendars, events, event\_attendees, event\_reminders with optional case\_id. **Acceptance:** FK integrity; soft delete policy. **Deliverables:** Migration.

## 3.2 Create court‑date template

**Purpose:** Fast event capture. **Steps:** Form with court, room, judge, case, assigned lawyer(s). **Acceptance:** Saved event appears on owner & assignee calendars. **Deliverables:** UI + API.

## 3.3 Owner calendar view

**Purpose:** See all schedules. **Steps:** Month/week/day; filters by workspace/lawyer. **Acceptance:** Switching filters updates results; perf < 300ms typical. **Deliverables:** Calendar pages.

## 3.4 Availability warnings

**Purpose:** Avoid double‑booking. **Steps:** On save, detect overlaps for attendees; show warning. **Acceptance:** Warning shows when conflict exists. **Deliverables:** Conflict service + UI.

## 3.5 Reminder engine (before/after)

**Purpose:** N‑days prior + post‑event tasks. **Steps:** Config per firm; schedule emails; after event, auto‑create “Upload proceedings” task. **Acceptance:** Emails logged; task appears on assignee board. **Deliverables:** Cron/worker + templates.

**Exit:** Reliable scheduling with actionable follow‑ups.

# Phase 4 — Notifications System

## 4.1 Templates & localization

**Purpose:** Editable messages. **Steps:** DB templates with variables; locale column. **Acceptance:** Preview renders with sample data. **Deliverables:** Template editor page.

## 4.2 Delivery & retries

**Purpose:** Durable sending. **Steps:** Queue table; worker retries with backoff; provider webhooks to mark delivered/bounced. **Acceptance:** Failures retried; status visible in log. **Deliverables:** Worker + /notifications/log UI.

## 4.3 Per‑case comms log

**Purpose:** Trace who was told what. **Steps:** Link notifications to case; filter by recipient/type. **Acceptance:** Timeline shows events with status. **Deliverables:** Case → Activity tab.

**Exit:** Auditable notifications before/after events.

# Phase 5 — Tasks & Team Workflows

## 5.1 Tasks schema

**Purpose:** Track work. **Steps:** tasks, task\_assignees, task\_comments, optional case\_id & event\_id. **Acceptance:** Comments notify mentions; history preserved. **Deliverables:** Migration + API.

## 5.2 My Tasks & Board views

**Purpose:** Personal + team planning. **Steps:** Per‑user list with filters; Kanban by status. **Acceptance:** Drag‑drop updates status; ACL respected. **Deliverables:** UI pages.

## 5.3 Bulk actions & reminders

**Purpose:** Speed. **Steps:** Multi‑select reassign/close; due‑date reminders via notifications system. **Acceptance:** Bulk ops appear in audit log. **Deliverables:** UI actions + audit entries.

**Exit:** Teams coordinate and close loops.

# Phase 6 — Client Portal (read‑only)

## 6.1 Client user type & scope

**Purpose:** Limit to their own cases. **Steps:** role=client; link to party/cases; portal feature switch per firm. **Acceptance:** Client sees only their active cases. **Deliverables:** Portal auth route + guard.

## 6.2 Portal case list & detail

**Purpose:** Transparency. **Steps:** List with status & next date; detail with shared docs. **Acceptance:** No internal‑only fields leak; downloads are expiring links with watermark. **Deliverables:** Portal pages.

**Exit:** Clients can self‑serve updates and documents.

# Phase 7 — Public Website API (Active Cases)

## 7.1 Public endpoint + cache

**Purpose:** Embed active cases on firm site. **Steps:** Read‑only route returns allow‑listed fields only; cache 60–300s; rate limit. **Acceptance:** No PII; header indicates cache; 429 on abuse. **Deliverables:** /public/active-cases endpoint + docs + embed snippet.

**Exit:** Safe public integration for marketing/PR.

# Phase 8 — Billing & Usage (firm‑level)

## 8.1 Stripe customer & plans

**Purpose:** Attach billing to firms. **Steps:** Create customer on provisioning; map plans to limits. **Acceptance:** Webhooks update subscription state in DB. **Deliverables:** Billing service; webhook handler.

## 8.2 Usage meters (optional)

**Purpose:** Bill per emails/storage/etc. **Steps:** Emit usage events; send to Stripe; show usage UI. **Acceptance:** Stripe reflects reported usage; soft/hard limit gates enforced. **Deliverables:** Meter producer; usage dashboard.

**Exit:** Plans and (optional) usage limits enforced.

# Phase 9 — Security, Compliance & Admin Ops

## 9.1 Impersonation with audit

**Purpose:** Support without privacy loss. **Steps:** Only Super‑Admin; require reason/ticket; banner in UI; audit entries. **Acceptance:** Every impersonation has reason, start/end, actions logged. **Deliverables:** Admin tools + audit view.

## 9.2 Legal hold & retention

**Purpose:** Preserve evidence. **Steps:** legal\_hold on case/doc; disable delete; backup retention policy. **Acceptance:** Deletes blocked while on hold; logs show attempts. **Deliverables:** Flags + enforcement + tests.

## 9.3 Backups & restore drill

**Purpose:** Disaster recovery. **Steps:** Nightly DB dump; storage snapshots; quarterly restore test. **Acceptance:** Restore runbook validated end‑to‑end. **Deliverables:** Scripts + runbook.

**Exit:** Auditable, recoverable operations.

# Phase 10 — Observability & Performance

## 10.1 Structured logging & tracing

**Purpose:** See what’s happening. **Steps:** Add request id, user id, firm id; basic distributed traces across API, DB, storage. **Acceptance:** Trace shows a full request path for key flows. **Deliverables:** Logger util; tracing setup.

## 10.2 Indexes & query budgets

**Purpose:** Keep it fast. **Steps:** Add indexes (case status, lawyer id, event date); set p95 budgets. **Acceptance:** p95 within targets on sample data. **Deliverables:** Migration; perf report.

## 10.3 Job runner hardening

**Purpose:** Durable reminders. **Steps:** Visibility timeouts, dead‑letter; admin page to retry. **Acceptance:** Failed jobs visible and recoverable. **Deliverables:** Queue ops UI.

**Exit:** Healthy performance with basic SLOs.

# Phase 11 — UX Polish & Docs

## 11.1 Role‑based navigation & empty states

**Purpose:** Clarity for each role. **Steps:** Dynamic nav per role; helpful empty states & quick‑start tips. **Acceptance:** QA checklist per role passes. **Deliverables:** Nav config + copy.

## 11.2 Command palette & shortcuts

**Purpose:** Speed common actions. **Steps:** Palette to create case/event/task/doc; keyboard shortcuts. **Acceptance:** Palette actions create records with minimal fields. **Deliverables:** Palette component + docs.

## 11.3 Help center & change log

**Purpose:** Lower support cost. **Steps:** Admin/user/client guides; release notes page. **Acceptance:** Links from app footer; content reviewed. **Deliverables:** Docs site/pages.

**Exit:** Onboarding is smooth; support load reduced.

# Phase 12 — Launch & Post‑Launch

## 12.1 Pilot import tool

**Purpose:** Bring legacy data. **Steps:** CSV importer for cases/parties/docs metadata; mapping UI. **Acceptance:** Sample import succeeds with validation warnings where needed. **Deliverables:** Import CLI + UI.

## 12.2 SLA & support runbooks

**Purpose:** Be ready for customers. **Steps:** Support inbox; severity matrix; on‑call hours. **Acceptance:** Test ticket flows; response templates approved. **Deliverables:** Runbooks + templates.

## 12.3 Feedback & roadmap intake

**Purpose:** Guide iteration. **Steps:** In‑app feedback; upvote board by firm. **Acceptance:** Submissions visible to Super‑Admin with filters. **Deliverables:** Feedback page + admin board.

**Exit:** MVP live with feedback loop feeding the backlog.

## Notes

* Each task stands alone with clear acceptance criteria and deliverables.
* Keep security gates (RLS/RBAC/audit) tested in every phase.
* No dates here—your sprint planning can slice these as needed.