# MASTER BUILD PROMPT (Paste into Replit AI)

**Role**: You are a senior full-stack architect and staff engineer. Build a production-ready SaaS (web + mobile) that analyzes legal documents from photos or file uploads, flags risky clauses with reasons and negotiation tactics, and produces a shareable report. The system must support both **organization/team accounts** and **multiple independent single-user (Solo) accounts** in the same deployment. Follow the PRD below exactly. Where details are ambiguous, make the smallest sensible, well-documented assumption and proceed.

**High-level stack (preferred, adjust only if needed and document changes):**

- **Frontend (Web):** Next.js 14 (App Router) + TypeScript + Tailwind + shadcn/ui + React Query.
- **Mobile:** React Native via Expo (iOS/Android) with the same design system and API types.
- **Backend:** Node.js (TypeScript) + NestJS (or Express if simpler) + OpenAPI spec.
- **Database:** PostgreSQL + Prisma with **Row Level Security** (multi-tenant).
- **Storage:** S3-compatible (e.g., MinIO in dev) with signed URL uploads.
- **Queue/Jobs:** BullMQ (Redis) for OCR + AI analysis pipelines.
- **OCR:** Pluggable provider interface with Tesseract-based local fallback; provider adapters (Google Vision/AWS Textract/Azure Read) behind the same interface.
- **LLM/AI:** Provider-agnostic service (OpenAI-compatible by default) with safety rails and deterministic prompts; support tool-use for retrieval.
- **Infra (dev):** Docker Compose for Postgres, Redis, MinIO. Turborepo monorepo. CI with GitHub Actions.

---

## PRODUCT REQUIREMENTS DOCUMENT (PRD)

### 1) Problem & Outcome

Users need fast, plain-English insight into contracts. The app ingests photos or files, normalizes text, identifies jurisdiction, highlights risky/important clauses, explains *why* they matter, proposes safer alternatives, and outputs a report. The system must be multi-user, multi-tenant, secure, and mobile-friendly. It must also support a **Solo** mode where a single user has a personal workspace (no team features) and can later upgrade to a team; multiple Solo users co-exist as isolated tenants.

### 2) Key Use Cases (User Stories)

1. **Upload/Scan**: As a user, I can upload PDFs/Docs or capture photos of each page on mobile; the app auto-crops, de-skews, and enhances readability.
2. **Auto-Jurisdiction**: As a user, I can set jurisdiction (country/region/state/province) or let the system infer from the document (e.g., Governing Law clauses, addresses) and my profile.
3. **Analyze**: As a user, I trigger analysis. I see a timeline of jobs (OCR → normalization → clause extraction → risk scoring → suggestions).
4. **Highlights**: As a user, I see highlighted text spans mapped to clause categories and risk levels with plain-English explanations and links to sources.
5. **Negotiation Tips**: For each risky clause, I receive rewrite options and negotiation tactics.

6. **Report**: I can export a branded report (PDF/HTML) with executive summary, risk table, redlines, and next steps.
7. **Teams & Sharing**: I can invite teammates, control access (Owner/Admin/Member/Viewer), and share read-only reports via expiring links.
8. **Compliance & Privacy**: I can choose retention (e.g., 7/30/365 days) or immediate purge after report generation.
9. **Auditability**: I can view an audit trail of uploads, analyses, and changes.
10. **Billing**: Plans: **Solo** (single user), **Pro** (teams), **Business** (SSO, higher caps), with metered analysis credits.
11. **Solo (Single-User) Flow**: Onboarding selects **Solo**; a personal workspace is auto-created; invites/roles/SSO are hidden; expiring link sharing remains; upgrading to a team later does not move or rename resources.

## 3) Functional Requirements

### 3.1 Ingestion

- Accept: PDF, DOCX, PNG/JPG, HEIC. Max 50 pages per document (configurable).
- Mobile camera flow: multi-page capture, auto-crop, de-skew, glare/shadow reduction, reorder pages.
- Large file uploads via signed URLs directly to S3; virus scan (clamav container) async before processing.

### 3.2 OCR & Normalization

- Detect language; OCR per page; output per-token bounding boxes.
- De-duplicate headers/footers; merge hyphenated line breaks; normalize whitespace; preserve page numbers.
- Confidence scores per token; low-confidence spans flagged for user review.

### 3.3 Jurisdiction Determination

- Inputs: user-selected jurisdiction override; inferred from document (Governing Law/venue), addresses, phone formats, currency symbols.
- Output: ISO country + region code(s), confidence, and rationale.

### 3.4 Clause Extraction & Taxonomy

- Extract clauses with start/end character offsets and page indices.
- Minimum taxonomy (extendable): Governing Law, Venue/Jurisdiction, Term, Termination, Auto-Renewal, Scope/Deliverables, Payment/Fees, Interest/Late Fees, Confidentiality, IP Ownership, Work-Made-For-Hire, Non-Compete, Non-Solicit, Assignment, Warranties, Indemnification, Limitation of Liability, Insurance, Force Majeure, Dispute Resolution/Arbitration, Notices, Privacy/Data Protection, Audit/Inspection, Publicity, Change Orders.

### 3.5 Risk Scoring & Rationale

- Score each clause (Low/Medium/High/Critical) using: (a) rule heuristics per jurisdiction ("Law Packs"), (b) learned patterns via LLM rubric prompts, (c) user profile (e.g., SMB buyer vs contractor).
- Provide plain-English rationale and cite which heuristic/policy triggered the risk.

### 3.6 Suggestions & Redlines

- For each risky clause, provide: (1) what to ask, (2) why it helps, (3) a redlined rewrite option, (4) fallback compromise. Render redlines clearly.

### 3.7 Sources & Disclaimers

- When referencing laws/guidelines, cite the source text snippet **and date accessed** (from the Law Pack KB). Include a prominent **NOT LEGAL ADVICE** disclaimer and suggest consulting counsel for critical items.

### 3.8 Reporting

- Export PDF/HTML report with: cover, summary metrics, risk table, clause highlights, suggestions, appendix (jurisdiction assumptions, OCR confidence heatmap, changelog), and shareable link (expiring).

### 3.9 Teams, Roles, Tenancy

- Workspaces (orgs) with users. Roles: Owner, Admin, Member, Viewer. Row-level data isolation enforced via Postgres RLS.
- **Solo mode:** Every new user can choose Solo; we auto-provision a personal workspace and a membership as Owner. Team features (invites, role management, SSO) are disabled in Solo.
- **Upgrade path:** Provide `POST /orgs/{id}/upgrade` to convert Solo → Team (plan change + enable features) without changing IDs; all data remains.
- **Downgrade guardrails:** Only allow Team → Solo if one member remains and no pending invites.

### 3.10 Billing & Limits

- Plans: **Solo** (single user), **Pro** (teams), **Business** (SSO, higher caps). All plans use metered analysis credits.
- Per-plan caps: pages per analysis, monthly analyses, storage retention.
- Solo plan hides team features; upgrade to Pro/Business unlocks them seamlessly.
- Soft warnings at 80% usage; hard stops at 100% with upgrade CTA.

## 4) Non-Functional Requirements (NFRs)

- **Security**: RLS + least privilege; encrypt at rest and in transit; signed URLs; delete on user request; tamper-evident audit log.
- **Privacy**: Region-aware storage; data residency setting per org; retention policies (7/30/365 days); PII minimization.
- **Compliance**: Support PIPEDA, GDPR, CCPA basics (DPO contact, data export, right to be forgotten). Cookie banner on web.
- **Observability**: Structured logs, traces, metrics; job dashboards; dead-letter queue.
- **Performance**: First analysis result within 90s for 10-page PDF on Pro plan; page OCR < 3s median on dev hardware.
- **Accessibility**: WCAG 2.1 AA. Keyboard navigable, proper semantics.
- **i18n**: English initially; architecture supports additional locales.

## 5) Data Model (ERD Outline)

**Solo mode model note:** Each newly registered Solo user is auto-provisioned a personal `org` (workspace) and a `membership` with `role='Owner'`. Team features are disabled while `orgs.tier='Solo'`. Upgrading flips `tier` and enables invites without migrating IDs.

- `orgs(id, name, tier, data_residency, retention_days, created_at)`
- `users(id, email, name, auth_provider_id, created_at)`
- `memberships(id, org_id, user_id, role)` (unique org_id+user_id)
- `documents(id, org_id, title, status, page_count, upload_src, storage_key, sha256, created_by, created_at)`
- `pages(id, document_id, index, storage_key, ocr_json, text, confidence_avg)`
- `analyses(id, document_id, jurisdiction, jurisdiction_confidence, started_at, completed_at, status, metrics_json)`
- `clauses(id, analysis_id, type, risk, page_index, start_char, end_char, text, rationale, sources_json)`
- `suggestions(id, clause_id, summary, why_it_matters, ask, rewrite_option, fallback_option)`
- `audit_logs(id, org_id, actor_user_id, action, target_type, target_id, meta_json, created_at)`
- `invites(id, org_id, email, role, token, expires_at, accepted_at)`
- `billing_customers(id, org_id, stripe_customer_id, plan, period_start, period_end, usage_json)`
- `jobs(id, type, payload_json, state, attempts, last_error, created_at, updated_at)`

## 6) API (sample; generate full OpenAPI)

`POST /auth/*` (provider-backed) `POST /orgs` Create org `POST /orgs/{id}/invites` Invite user `GET /documents` List `POST /documents` Create (returns signed URLs per page) `POST /documents/{id}/finalize` Begin virus scan → OCR → analysis `GET /documents/{id}` Detail `GET /documents/{id}/analyses/{analysisId}` Status + results `GET /clauses/{id}` Clause detail (with spans) `POST /reports` Generate + store PDF/HTML, return share link `POST /billing/checkout` Begin checkout `GET /me` Current user profile + default workspace & tier `POST /orgs/{id}/upgrade` Convert Solo → Team (enable invites/SSO) `POST /orgs/{id}/downgrade` Team → Solo (if eligible)

## 7) Analysis Pipeline (Jobs)

1. **Virus Scan** → fail fast on threat, notify user.
2. **OCR** (per page) → text + tokens + boxes + confidence.
3. **Normalization** → clean text, reconstruct paragraphs, map spans.
4. **Jurisdiction** → detect vs user override; write to `analyses`.
5. **Clause Extraction** → LLM w/ function calling returns clause spans + types.
6. **Risk Scoring** → rules (Law Pack KB) + rubric prompt; produce risk + rationale + sources.
7. **Suggestions** → generate ask/why/rewrite/fallback; attach to clauses.
8. **Report Build** → HTML → PDF; store artifact; issue share token.

### 8) Law Pack Knowledge Base (KB)

- File-based KB per jurisdiction with versioned YAML/Markdown entries describing heuristics, thresholds, and canonical language examples. Each rule has: `id`, `jurisdiction`, `clause_type`, `pattern_examples`, `risk_logic`, `citations`, `last_reviewed`.
- Retrieval layer: select KB by inferred/selected jurisdiction; pass relevant snippets to LLM; include KB `id`s in outputs for traceability.

### 9) UX Requirements

- Clear progress UI for each job stage with retry on failed stages where possible.
- Document viewer: side-by-side (text + page image) with highlighted spans; click to see explanation & tactics.
- Mobile flows: capture wizard; offline queue + background upload; report preview; share via link.
- **Solo UX**: Hide team management/invites; replace org switcher with fixed personal workspace and an "Upgrade to Team" CTA.

### 10) Security & Threat Modeling (minimum)

- Multi-tenant isolation via Postgres RLS and org_id scoping at every query.
- Signed URL least privilege (PUT only on upload, GET only on view; short expiries).
- Secrets via environment; rotate keys. Rate limit auth + uploads. Content Security Policy on web.
- Prompt injection defense: never blindly execute text from docs; prompts must quote and constrain inputs; strip/escape control tokens.
- Store LLM prompts/responses with redaction; attach job/run IDs to audit trail.

### 11) Acceptance Criteria (AC)

- Can upload a 10-page PDF *and* 10 photos; full pipeline completes; report generated with at least 5 clause types detected.
- Jurisdiction inference works and can be overridden; the chosen jurisdiction is referenced in rationales.
- Each risky clause shows: highlighted span, risk level, rationale, 2+ negotiation tactics, and at least one redline.
- Report exports as PDF and shareable HTML with expiring link; viewer requires token.
- RLS verified by tests: a user from Org A cannot access Org B resources.
- Accessibility checks pass (axe) on key pages.
- Solo user can sign up → upload → analyze → export without seeing team UI.
- Upgrading a Solo account to Team preserves document history and enables invites/roles without changing resource IDs.

---

# ARCHITECTURE & IMPLEMENTATION PLAN

### Monorepo Structure (Turborepo)

```
apps/
  web/ (Next.js)
  mobile/ (Expo)
  api/ (NestJS)
packages/
  ui/ (shared shadcn components)
  types/ (zod/OpenAPI types)
  sdk/ (typed client)
  prompts/ (LLM prompts + evaluators)
  lawpacks/ (KB)
infra/
  docker/compose.yml (postgres, redis, minio, clamav)
  migrations/
```

### Step-by-Step

1. **Scaffold** monorepo, Tailwind, shadcn, ESLint/Prettier, Husky pre-commit.
2. **DB + Prisma**: schemas + migrations; enable RLS; seed dev users/orgs.
3. **Auth**: provider integration; org selector; invite flows.
4. **Storage**: S3 signed PUT/GET; clamav scan job; metadata saved.
5. **Queues**: BullMQ; job definitions; worker processes.
6. **OCR service** with provider interface + Tesseract fallback; store tokens/boxes/confidence.
7. **LLM service**: provider abstraction; rate limiting; cost logging; deterministic prompts.
8. **Law Pack KB**: file format + loader; retrieval by jurisdiction and clause; evaluator harness.
9. **Clause extraction & risk** pipeline; suggestion generator.
10. **Web UI**: dashboard → upload → analysis status → viewer → report export → billing.
11. **Mobile (Expo)**: capture wizard, upload queue, analysis status, viewer, share.
12. **Reports**: HTML templates + Puppeteer to PDF; share tokens.
13. **Billing**: Stripe test mode; plan gates.
14. **Observability**: request logging, job metrics, health checks.
15. **Docs**: README, runbook, API reference, admin notes.

### Testing

- **Unit**: Prisma models, RLS policies, KB loader, prompt builders.
- **Integration**: upload → OCR → analysis pipeline (mock providers), report generation.
- **E2E**: Playwright for web critical paths; Detox for mobile basics.
- **Security**: RLS regression tests; rate-limit tests; token leakage tests.
- **Accessibility**: axe checks in CI.
- **Solo mode**: E2E for Solo sign-up, hidden team UI, and upgrade flow preserves resource IDs.

---

## AI PROMPTS (Deterministic Rubrics)

### Clause Extraction (function calling)

- System: *You extract clauses from normalized contract text. Return JSON with clause_type, start_char, end_char, page_index, and short evidence.*
- Rules: use the taxonomy; prefer exact spans; if uncertain, mark `confidence` and `notes`.

### Risk Scoring

- System: *You are a compliance/risk engine. Using supplied Law Pack rules + org profile, assign a risk and explain which rule(s) fired. Always reference KB rule IDs.*

### Suggestions/Redlines

- System: *Propose 2–3 negotiation tactics per risky clause. Provide a clear rewrite option and a fallback, both jurisdiction-appropriate.*

*(Store these prompt templates in `` with unit tests that validate output schemas.)*

---

## REPORT TEMPLATE (HTML → PDF)

- Cover: document title, org, date, jurisdiction, disclaimer.
- Executive Summary: risk score distribution, top 5 issues.
- Detailed Findings: for each clause → snippet highlight, risk, rationale (with KB citations), tactics, redlines.
- Appendix: OCR confidence map, changelog, methodology, data retention choice.

---

## COMPLIANCE & ETHICS

- Prominent **NOT LEGAL ADVICE** banner.
- Data export & deletion endpoints; retention settings respected by a daily cleanup job.
- Track law-pack version used for each analysis for reproducibility.

---

## DELIVERABLES

1. Running dev stack (`docker compose up`).
2. Seed script that creates: (a) one Solo user with a personal workspace; (b) one Team org with 2 users; plus a sample 6-page contract and a completed analysis.
3. OpenAPI JSON + Swagger UI for the API.
4. Playwright E2E covering the ACs.
5. README with setup, env vars, and deployment notes.

---

## FINAL SELF-REVIEW (REQUIRED)

At the end of implementation, run an automated PRD traceability check and output it to the console and `/artifacts/build_report.md`:

- Generate a matrix mapping each **PRD requirement** (section 3 & 11 items) to **code artifacts** (routes, services, components, tests).
- Confirm every API in the PRD exists in the OpenAPI spec and has tests.
- Confirm RLS policies exist and tests prove cross-org access is denied.
- Confirm report export contains all required sections.
- Confirm prompts reference Law Pack rule IDs and that outputs store these IDs.
- List any gaps or deviations with rationale and TODOs.

**If any check fails, fix it now and re-run the matrix until all items pass. Output the passing matrix in ``.**

---

## RUN COMMANDS (Dev)

- `docker compose up -d` (postgres, redis, minio, clamav)
- `pnpm i && pnpm dev` (starts web, api, mobile)
- `pnpm test` (unit/integration)
- `pnpm e2e:web` (Playwright)

---

## NOTES FOR YOU (Replit AI)

- Minimize third-party lock-in; abstract providers (OCR/LLM/auth/storage) behind interfaces.
- Keep secrets out of repo; use `.env.example`.
- Prefer small, well-tested modules; add types and zod validators at boundaries.
- Keep UX clean and accessible; use the shared UI package across web and mobile.

**Build now. When finished, run the Final Self-Review and include the passing traceability matrix.**