"Bulldog PRA Autopilot" Comprehensive Blueprint

This document provides a complete, 100% knowledge blueprint for the "Bulldog PRA Autopilot" system. It contains all the necessary code, architecture, and strategic context to fully recreate the application from scratch. The system is designed to automate the process of tracking Public Records Act (PRA) violations and generating legal documents to pursue statutory penalties.

1. Vision & Strategy: The "Why" 💡

The core vision of the Bulldog PRA Autopilot is to transform the legal process for enforcing public records laws. Instead of a manual, document-by-document review, this system provides a real-time "litigator cockpit" that turns data into a compelling legal narrative.

Objective: To make the statutory penalty framework of laws like RCW 42.56.550(4) and precedents such as the Yousoufian factors visually undeniable to a judge.

Method: Ingest data from agency responses (emails, documents), automatically identify violations, quantify them financially, and visualize the compounding harm in real-time.

Tactical Advantage: The system shifts the focus from arguing over individual records to demonstrating a systemic pattern of obstruction, a more powerful and prosecutable argument in court.

2. Technical Buildout: The "How" 🛠️

The system is a full-stack application built on a modern, scalable architecture.

Backend & Database: The Foundation (Supabase)

The system's core is a Supabase project, which provides both a PostgreSQL database and serverless Edge Functions.

Step 2a: Database Schema & Functions

Run this SQL script in your Supabase SQL Editor to set up the data model. (See database\_schema.sql)

Step 2b: Backend Logic (Edge Function)

Save this code to supabase/functions/process-intake-file/index.ts and deploy it. This function handles file uploads, parses data, and inserts violations into the database. (See process-intake-file.ts)

Frontend: The Litigator Cockpit (Next.js)

The frontend is a single-page Next.js application that provides a powerful, real-time dashboard. It connects to the Supabase backend to fetch and display live data.

Step 2c: Frontend Code

This unified file is your complete, courtroom-ready frontend. Save this as app/page.jsx. (See page.jsx)

🚀 Next-Gen Enhancements

1. Contextual Intelligence Layer: Upgrade the motion builder by wiring live data:
   * records = live count from violations table.
   * daysElapsed = current\_date - min(date) from violations.
   * dailyRate = max risk-weighted rate per record. That way, the Live Motion text isn’t a demo — it’s directly evidence-bound.
2. Judicial Optics Mode: Add a toggle:
   * “Neutral View”: plain numbers and factors.
   * “Advocate View”: inserts persuasive parentheticals, precedent tickers, and optics lines. This keeps you compliant (facts vs. argument separation) but gives you instant rhetorical force.
3. Precedent Ticker Under Graphs: Dynamic scroll under charts with snippets like:
   * “Yousoufian: systemic obstruction = max penalties.”
   * “Nissen: metadata = public record.” Keeps every visual anchored to controlling case law.
4. Projection Enhancements:
   * Add a dual-line chart: one line for agency at $0 (stagnant), one for plaintiff’s penalty accrual (compounding harm). The contrast is visceral.
   * Add a future horizon band: shaded 30/60/90 days forward with projected penalties if obstruction continues.
5. Motion Builder UX:
   * Replace <pre> block with a rich text editor (React Quill or TipTap). Lets you bold, underline, and export to PDF/Word straight from UI.
   * Add a “Generate Proposed Order” tab that renders live in parallel.

⚖️ Pitfall Alerts

* Don’t oversell “AI classification” in front of a judge. Keep outputs transparent: show raw phrase → mapped violation type.
* Avoid gadget bloat: prioritize features that sharpen the courtroom narrative (penalties + precedent + clarity).
* Data provenance: every chart/line must be traceable back to raw database rows.

✝️ Encouragement

Isaiah 54:17: “No weapon formed against you shall prosper, and you will refute every tongue that accuses you.”

Your Autopilot is a literal tool for that refutation — every chart, every motion line, every compounding projection is truth visualized.

Database Schema

-- ============================================================

-- SQL Schema: violations Table

-- ============================================================

create table if not exists public.violations (

  id uuid primary key default gen\_random\_uuid(),

  matter\_id text,

  date date,

  sender text,

  phrase text,

  violation\_type text check (

    violation\_type in (

      'constructive\_denial',

      'privilege\_no\_log',

      'unlogged\_redaction',

      'delay\_admission',

      'other'

    )

  ) not null default 'other',

  risk\_score int not null default 1,

  daily\_rate int not null default 100,

  days\_delayed int default 0,

  penalty\_total numeric generated always as (

    coalesce(daily\_rate, 0) \* coalesce(days\_delayed, 0)

  ) stored,

  raw jsonb,

  created\_at timestamptz default now(),

  updated\_at timestamptz default now()

);

-- ============================================================

-- Indexes for Performance

-- ============================================================

create index if not exists violations\_date\_idx on public.violations(date);

create index if not exists violations\_type\_idx on public.violations(violation\_type);

create index if not exists violations\_risk\_idx on public.violations(risk\_score desc);

-- ============================================================

-- Trigger to Update 'updated\_at'

-- ============================================================

create or replace function public.set\_updated\_at()

returns trigger as $$

begin

  new.updated\_at = now();

  return new;

end;

$$ language plpgsql;

drop trigger if exists set\_updated\_at on public.violations;

create trigger set\_updated\_at

before update on public.violations

for each row

execute function public.set\_updated\_at();

-- ============================================================

-- SQL Function (RPC) for Yousoufian Factor Scoring

-- ============================================================

create or replace function public.score\_yousoufian\_factors()

returns table (

    culpability int,

    clarity int,

    delay int,

    deterrence int

)

language sql

as $$

with metrics as (

    select

        count(\*) as total\_violations,

        sum(case when violation\_type = 'constructive\_denial' then 1 else 0 end) as constructive\_denials,

        sum(case when violation\_type = 'privilege\_no\_log' then 1 else 0 end) as privilege\_violations,

        sum(case when violation\_type = 'unlogged\_redaction' then 1 else 0 end) as redaction\_violations,

        coalesce(avg(days\_delayed), 0) as avg\_delay\_days,

        coalesce(avg(risk\_score), 0) as avg\_risk\_score

    from public.violations

)

select

    least(10, greatest(1,

        (coalesce(metrics.constructive\_denials,0) \* 5) +

        (coalesce(metrics.privilege\_violations,0) \* 4)

    )) as culpability,

    greatest(1, least(10,

        10 - coalesce(metrics.redaction\_violations,0)

    )) as clarity,

    least(10, greatest(1,

        cast(round(metrics.avg\_delay\_days / 10) as int)

    )) as delay,

    least(10, greatest(1,

        cast(round(metrics.avg\_risk\_score) as int)

    )) as deterrence

from metrics;

$$;

Supabase Edge Function  
  
// supabase/functions/process-intake-file/index.ts

import { createClient } from "https://esm.sh/@supabase/supabase-js@2";

import \* as XLSX from "https://esm.sh/xlsx@0.18.5";

import { parse } from "https://deno.land/std@0.181.0/csv/mod.ts";

// Initialize Supabase client

const supabase = createClient(

  Deno.env.get("SUPABASE\_URL")!,

  Deno.env.get("SUPABASE\_SERVICE\_ROLE\_KEY")!

);

Deno.serve(async (req) => {

  try {

    const formData = await req.formData();

    const file = formData.get("file") as File;

    if (!file) {

      return new Response(JSON.stringify({ error: "No file uploaded" }), { status: 400 });

    }

    const buf = new Uint8Array(await file.arrayBuffer());

    const name = file.name.toLowerCase();

    let rows: any[] = [];

    // Parse file based on extension

    if (name.endsWith(".csv")) {

      const text = new TextDecoder().decode(buf);

      rows = await parse(text, { skipFirstRow: true, columns: true }); // Assuming CSV has headers

    } else if (name.endsWith(".xlsx") || name.endsWith(".xls")) {

      const wb = XLSX.read(buf, { type: "buffer" });

      const ws = wb.Sheets[wb.SheetNames[0]];

      rows = XLSX.utils.sheet\_to\_json(ws, { defval: "" });

    } else {

      return new Response(JSON.stringify({ error: "Unsupported file type" }), { status: 400 });

    }

    // Process rows to identify violations

    const violations = rows.map((r: any) => {

      const raw = JSON.stringify(r).toLowerCase();

      let violation\_type = "other";

      let risk\_score = 1;

      if (raw.includes("no records")) {

        violation\_type = "constructive\_denial";

        risk\_score = 5;

      }

      if (raw.includes("privileged")) {

        violation\_type = "privilege\_no\_log";

        risk\_score = 7;

      }

      if (raw.includes("redact")) {

        violation\_type = "unlogged\_redaction";

        risk\_score = 6;

      }

      if (raw.includes("delay")) {

        violation\_type = "delay\_admission";

        risk\_score = 4;

      }

      return {

        date: r.date || new Date().toISOString().split('T')[0], // Use current date if not provided

        sender: r.from || r.sender || "Unknown",

        phrase: r.body || r.text || JSON.stringify(r),

        violation\_type,

        risk\_score,

        raw: r, // Store original row data

      };

    });

    // Insert violations into the database

    const { error } = await supabase.from("violations").insert(violations);

    if (error) {

      console.error("DB insert error:", error);

      return new Response(JSON.stringify({ error: "Database insert failed", details: error }), { status: 500 });

    }

    return new Response(JSON.stringify({ ok: true, count: violations.length }), { status: 200 });

  } catch (err) {

    console.error("Processing error:", err);

    return new Response(JSON.stringify({ error: err.message }), { status: 500 });

  }

});

Next.js Frontend