IMPLEMENTATION_PLAN.md — VERDICT

AI-Powered Deposition Coaching & Trial Preparation Platform
Version: 1.0.0 — Hackathon Edition | February 21, 2026
Team: VoiceFlow Intelligence | Track: AI Automation — August.law Sponsor Track
Build Window: **48 hours** | Feb 20 (6 PM) → Feb 22 (6 PM)

TABLE OF CONTENTS

---

1. OVERVIEW

Project Identity

Field	Value
App Name	VERDICT
Tagline	AI-powered deposition coaching. From 16 hours of prep to 6.
Build Type	48-hour hackathon MVP
Demo Deadline	Sunday, February 22, 2026 — 6:00 PM ET
Track	AI Automation — August.law Sponsor Track
Secondary Prizes	ElevenLabs (primary target), Databricks, Anthropic Claude

Team Roles (4 Members)

Member	Primary Role	Owns
Aman	AI/ML + Orchestration	Claude SDK agent loop, Nemotron integration, Behavioral Sentinel, Nia RAG
Nikhil	Backend + Data Pipelines	Fastify API, PostgreSQL, Redis, Databricks Delta Lake schema
Dhanush	Frontend + UI/UX	Next.js pages, shadcn/ui components, Framer Motion, Recharts radar

[Member 4]	Full-Stack / Integration	Auth (SAML/JWT), WebSocket plumbing, ElevenLabs Conversational AI, deployment

Build Philosophy

Documentation-first → code fast. Every decision was made in PRD, APP_FLOW, TECH_STACK, and BACKEND_STRUCTURE before the hackathon started. During the build window we execute against the spec, not debate architecture.

Non-negotiables (P0):

1. P0.1 Interrogator Agent — ElevenLabs voice questions
2. P0.2 Objection Copilot — FRE classification ≤1.5s
3. P0.3 Inconsistency Detector — Nemotron scoring ≤4s
4. P0.4 Case File Ingestion — PDF → Nia → extracted facts
5. P0.5 Coaching Brief — Review Orchestrator + ElevenLabs narration

Cut if behind schedule (in order): P1.4 Behavioral Sentinel → P1.2 Weakness Map → P1.1 Multi-session profile → brief PDF export → SAML SSO (keep email/password)

Reference Documents

Doc	Use During Build
`PRD`	Feature scope, acceptance criteria, sponsor prize requirements
`APP_FLOW.md`	Screen inventory, user flows, error states, decision trees
`TECH_STACK.md`	Exact library versions, configuration snippets
`BACKEND_STRUCTURE.md`	API endpoints, schema, validation rules, error codes

---

2. PRE-HACKATHON CHECKLIST (T-24 HRS)

> Complete EVERYTHING on this list before 6 PM Friday. Not completing pre-work is the most common cause of hackathon failure.

API Keys & Credentials (all 4 members verify access)

```
# Test each API key before the clock starts: # Anthropic Claude curl -X POST
https://api.anthropic.com/v1/messages \ -H "x-api-key: $ANTHROPIC_API_KEY" \ -H "content-type:
application/json" \ -d '{"model":"claude-sonnet-4-20250514","max_tokens":10,"messages":
[{"role":"user","content":"ping"}]}' # ElevenLabs curl -H "xi-api-key: $ELEVENLABS_API_KEY" \
https://api.elevenlabs.io/v1/voices | jq '.voices | length' # NVIDIA Nemotron curl -X POST
https://integrate.api.nvidia.com/v1/chat/completions \ -H "Authorization: Bearer $NEMOTRON_API_KEY" \ -H
"Content-Type: application/json" \ -d '{"model":"nvidia/llama-3.1-nemotron-ultra-253b-v1","messages":
[{"role":"user","content":"ping"}],"max_tokens":10}' # Nia API (confirm index endpoint) curl -H "Authorization:
Bearer $NIA_API_KEY" \ $NIA_BASE_URL/health # Databricks SQL warehouse npx tsx scripts/test-
databricks.ts # Should return: { status: 'connected', warehouseId: '...' }
```

Infrastructure Pre-Provisioned

- ☐ GitHub private repo created, all 4 members added
- ☐ Vercel project linked to GitHub repo → auto-deploy on push to `main`

- [ ] Railway project created, PostgreSQL + Redis plugins added
- [ ] Supabase project created as PostgreSQL backup
- [ ] Upstash Redis created as Redis backup
- [ ] AWS S3 bucket `verdict-documents-hackathon` created (us-east-1)
- [ ] AWS IAM user with S3 PutObject/GetObject/DeleteObject policy
- [ ] Resend account verified, domain DNS records set
- [ ] All env vars loaded in `.env.local` (frontend) and `.env` (backend)
- [ ] All env vars added to Vercel dashboard and Railway dashboard

Pre-Built Assets (prepared in the 24 hrs before)

- [ ] Prisma schema written and validated (`npx prisma validate`)
- [ ] FRE corpus text file prepared for Nia indexing (`scripts/nia/fre-corpus.txt`)
- [ ] Sample case document ready for demo: `demo/chen_v_metropolitan.pdf` (real-looking, 50 pages)
- [ ] ElevenLabs voice IDs confirmed: Interrogator (`Adam`), Coach (`Rachel`)
- [ ] MediaPipe face_landmarker.task model file downloaded to `public/models/`
- [ ] Figma component library exported or shadcn/ui components installed
- [ ] Monorepo workspace structure initialized (see Step 1.1)

---

3. PHASE 1 — FOUNDATION (Hour 0–4, Fri 6–10 PM)

Goal: All 4 team members have a running repo, working database, passing health check, and can `npm run dev` locally.

All 4 members work in parallel on separate concerns during this phase.

Step 1.1 — Initialize Monorepo Structure

Owner: Aman (sets up, others pull)
Duration: 30 minutes
Goal: Establish the folder structure that all 4 members will code into for the next 47.5 hours.

```
# Create monorepo mkdir verdict && cd verdict git init echo "packages:\n - apps/*\n - packages/*" > pnpm-
workspace.yaml # Initialize workspaces mkdir -p apps/frontend apps/backend packages/shared # Root package.json
cat > package.json << 'EOF' { "name": "verdict", "private": true, "workspaces": ["apps/*", "packages/*"], "scripts":
{ "dev": "concurrently \"npm run dev --workspace=apps/frontend\" \"npm run dev --workspace=apps/backend\"",
"build": "npm run build --workspace=apps/frontend && npm run build --workspace=apps/backend", "lint": "eslint
.", "typecheck": "tsc --noEmit -p apps/frontend/tsconfig.json && tsc --noEmit -p apps/backend/tsconfig.json",
"test": "vitest run", "db:migrate": "prisma migrate dev", "db:migrate:deploy": "prisma migrate deploy", "db:seed":
"tsx prisma/seed.ts", "db:studio": "prisma studio", "prepare": "husky" }, "devDependencies": { "concurrently":
"9.1.2", "husky": "9.1.7", "lint-staged": "15.4.3", "@commitlint/cli": "19.7.1", "@commitlint/config-conventional":
"19.7.0", "eslint": "9.20.0", "typescript-eslint": "8.24.1", "prettier": "3.4.2", "prettier-plugin-tailwindcss": "0.6.11",
"vitest": "2.1.8" } } EOF # Initialize frontend (Next.js) cd apps/frontend npx create-next-app@15.1.6 . \ --typescript
\ --tailwind \ --eslint \ --app \ --src-dir \ --import-alias "@/*" \ --no-git # Initialize backend (Fastify) cd ../backend
npm init -y npm install fastify@5.2.1 typescript@5.7.3 tsx@4.19.2 # Shared types package cd ../../packages/shared
npm init -y mkdir src touch src/index.ts # API types, Zod schemas shared between frontend + backend # Push
initial commit cd ../.. git add -A git commit -m "chore: initialize monorepo structure" git remote add origin
git@github.com:voiceflow-intelligence/verdict.git git push -u origin main
```

✅ **Success Criteria:**

- [ ] `npm run dev` from root starts both Next.js (port 3000) and Fastify (port 4000)
- [ ] All 4 team members can `git pull` and run locally

- ☐ GitHub repo visible at correct URL
- ☐ Vercel auto-deploys frontend on push (verify in Vercel dashboard)

Step 1.2 — Backend Project Setup

Owner: Nikhil
Duration: 45 minutes
Goal: Fastify server running with health check, CORS, logging, and error handler.

cd apps/backend # Install all backend dependencies (exact versions from TECH_STACK.md §10) npm install \ fastify@5.2.1 \ @fastify/cors@10.0.2 \ @fastify/jwt@9.0.1 \ @fastify/multipart@9.0.3 \ @fastify/rate-limit@10.2.1 \ @fastify/socket.io@3.0.0 \ socket.io@4.8.1 \ @prisma/client@6.3.1 \ ioredis@5.4.1 \ @anthropic-ai/sdk@0.36.3 \ elevenlabs@1.14.0 \ axios@1.7.9 \ @databricks/sql@1.10.0 \ @aws-sdk/client-s3@3.726.1 \ @aws-sdk/s3-request-presigner@3.726.1 \ @aws-sdk/lib-storage@3.726.1 \ jsonwebtoken@9.0.2 \ bcrypt@5.1.1 \ nanoid@5.0.9 \ resend@4.1.2 \ pdf-parse@1.1.1 \ mammoth@1.8.0 \ pino@9.6.0 \ zod@3.24.1 \ dotenv@16.4.7 \ uuid@11.0.5 \ @sentry/node@8.51.0 npm install -D \ prisma@6.3.1 \ @types/node@22.13.4 \ @types/bcrypt@5.0.2 \ @types/jsonwebtoken@9.0.9 \ typescript@5.7.3 # Create entry point mkdir -p src/{modules,plugins,middleware,lib,types}

```
// apps/backend/src/index.ts
import Fastify from 'fastify';
import { config } from 'dotenv';
config();

const app = Fastify({
  logger: {
    level: process.env.LOG_LEVEL ?? 'info',
    transport: process.env.NODE_ENV === 'development'
      ? { target: 'pino-pretty' }
      : undefined,
  },
  requestIdHeader: 'x-request-id',
  genReqId: () => `req_${Date.now()}_${Math.random().toString(36).slice(2, 9)}`,
});

// Health check — FIRST endpoint, no auth
app.get('/api/v1/health', async () => ({
  status: 'ok',
  service: 'verdict-api',
  version: '1.0.0',
  timestamp: new Date().toISOString(),
}));

const start = async () => {
  try {
    await app.listen({ port: parseInt(process.env.PORT ?? '4000'), host: '0.0.0.0' });
  } catch (err) {
    app.log.error(err);
    process.exit(1);
  }
};

start();
```

# Add dev script to package.json # "dev": "tsx watch src/index.ts" # Test it works npm run dev curl http://localhost:4000/api/v1/health # Expected: {"status":"ok","service":"verdict-api",...}

✅ **Success Criteria:**

- ☐ `curl http://localhost:4000/api/v1/health` returns `{ "status": "ok" }`
- ☐ `curl http://localhost:4000/api/v1/health` from frontend origin returns CORS headers

- ☐ Pino logs show request in terminal
- ☐ TypeScript compiles with zero errors (`npx tsc --noEmit`)

Step 1.3 — Database Setup

Owner: Nikhil
Duration: 45 minutes
Goal: PostgreSQL connected, all 11 tables created, Prisma client generated, seed data ready.

# Install Prisma CLI npm install -D prisma@6.3.1 # Copy schema from BACKEND_STRUCTURE.md §2 # (all 11 tables: firms, users, refresh_tokens, cases, documents, witnesses, # sessions, session_events, alerts, briefs, attorney_annotations) cp ../../docs/prisma-schema.prisma prisma/schema.prisma # Set DATABASE_URL in .env echo 'DATABASE_URL="postgresql://postgres:password@localhost:5432/verdict_dev"' >> .env # Create database and run first migration npx prisma migrate dev --name "create_initial_schema" # Verify tables created npx prisma studio # Opens at http://localhost:5555 — confirm all 11 tables visible # Generate Prisma client npx prisma generate

```
// apps/backend/src/lib/prisma.ts — singleton client
import { PrismaClient } from '@prisma/client';

const globalForPrisma = globalThis as unknown as { prisma: PrismaClient };

export const db = globalForPrisma.prisma ?? new PrismaClient({
  log: process.env.NODE_ENV === 'development' ? ['query', 'error'] : ['error'],
});

if (process.env.NODE_ENV !== 'production') globalForPrisma.prisma = db;
```

```
// prisma/seed.ts — demo data for development + hackathon demo
import { db } from '../apps/backend/src/lib/prisma';
import bcrypt from 'bcrypt';

async function main() {
  const firm = await db.firm.create({
    data: {
      name: 'Demo Law Group LLP',
      slug: 'demo-law-group',
      seats: 10,
      setupComplete: true,
      retentionDays: 90,
    },
  });

  const attorney = await db.user.create({
    data: {
      firmId: firm.id,
      email: 'sarah.chen@demo.com',
      name: 'Sarah Chen',
      role: 'PARTNER',
      passwordHash: await bcrypt.hash('Demo!Pass123', 12),
      emailVerified: true,
    },
  });

  const demoCase = await db.case.create({
    data: {
      firmId: firm.id,
      ownerId: attorney.id,
      name: 'Chen v. Metropolitan Hospital',
      caseType: 'MEDICAL_MALPRACTICE',
      opposingFirm: 'Defense Partners LLP',
      depositionDate: new Date('2026-03-15'),
    },
  });

  console.log('✅ Seed complete:', { firmId: firm.id, caseId: demoCase.id });
}

main().finally(() => db.$disconnect());
```

npx prisma db seed # Expected: ✅ Seed complete: { firmId: 'cl...', caseId: 'cl...' }

✅ Success Criteria:

- ☐ `npx prisma studio` shows 11 tables with seed data in `firms`, `users`, `cases`
- ☐ `db.firm.findMany()` returns the demo firm
- ☐ No TypeScript errors in Prisma client usage
- ☐ Migration file committed to Git

Step 1.4 — Redis + AWS S3 Setup

Owner: Nikhil
Duration: 30 minutes
Goal: Redis connected, S3 bucket accessible, both tested.

```
// apps/backend/src/lib/redis.ts
import { Redis } from 'ioredis';

export const redis = new Redis(process.env.REDIS_URL!, {
  maxRetriesPerRequest: 3,
  enableReadyCheck: true,
  reconnectOnError: (err) => {
    console.error('Redis reconnect on error:', err.message);
    return true;
  },
});

redis.on('connect', () => console.log('✅ Redis connected'));
redis.on('error', (err) => console.error('Redis error:', err));
```

```
// apps/backend/src/lib/s3.ts
import { S3Client } from '@aws-sdk/client-s3';

export const s3 = new S3Client({
  region: process.env.AWS_REGION!,
  credentials: {
    accessKeyId: process.env.AWS_ACCESS_KEY_ID!,
    secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY!,
  },
});

export const S3_BUCKET = process.env.S3_BUCKET_NAME!;
```

# Test Redis node -e " const { Redis } = require('ioredis'); const r = new Redis(process.env.REDIS_URL); r.set('test', 'verdict').then(res => console.log('Redis SET:', res)); r.get('test').then(res => { console.log('Redis GET:', res); r.quit(); }); " # Test S3 (upload a test file) node -e " const { S3Client, PutObjectCommand } = require('@aws-sdk/client-s3'); const s3 = new S3Client({ region: 'us-east-1' }); s3.send(new PutObjectCommand({ Bucket: process.env.S3_BUCKET_NAME, Key: 'test/connection-test.txt', Body: 'verdict s3 test' })).then(() => console.log('✅ S3 write success')); "

✅ Success Criteria:

- ☐ `Redis GET test` returns `"verdict"`

- [ ] S3 `verdict-documents-hackathon/test/connection-test.txt` visible in AWS console
- [ ] No connection errors in Fastify startup logs

Step 1.5 — Authentication Foundation

Owner: [Member 4]
Duration: 60 minutes
Goal: JWT middleware + login/refresh endpoints working. This unblocks all other authenticated routes.

```
// apps/backend/src/modules/auth/auth.service.ts
import jwt from 'jsonwebtoken';
import bcrypt from 'bcrypt';
import { nanoid } from 'nanoid';
import { db } from '../../lib/prisma';
import { redis } from '../../lib/redis';

const ACCESS_SECRET = process.env.JWT_SECRET!;
const REFRESH_SECRET = process.env.JWT_REFRESH_SECRET!;

export function signAccessToken(payload: { sub: string; firmId: string; role: string; email: s
  return jwt.sign(payload, ACCESS_SECRET, { expiresIn: '8h' });
}

export function signRefreshToken(payload: { sub: string; firmId: string; jti: string }) {
  return jwt.sign(payload, REFRESH_SECRET, { expiresIn: '30d' });
}

export async function loginWithPassword(email: string, password: string) {
  const user = await db.user.findUnique({
    where: { email: email.toLowerCase() },
    include: { firm: { select: { id: true, sentinelEnabled: true } } },
  });
  if (!user || !user.passwordHash) throw new Error('INVALID_CREDENTIALS');
  if (!user.isActive) throw new Error('ACCOUNT_INACTIVE');

  const valid = await bcrypt.compare(password, user.passwordHash);
  if (!valid) {
    await db.user.update({
      where: { id: user.id },
      data: { loginAttempts: { increment: 1 } },
    });
    throw new Error('INVALID_CREDENTIALS');
  }

  const jti = nanoid(16);
  const accessToken = signAccessToken({
    sub: user.id, firmId: user.firmId, role: user.role, email: user.email,
  });
  const refreshToken = signRefreshToken({ sub: user.id, firmId: user.firmId, jti });

  // Store refresh token hash in DB
  const crypto = await import('crypto');
  const tokenHash = crypto.createHash('sha256').update(refreshToken).digest('hex');
  await db.refreshToken.create({
    data: { userId: user.id, firmId: user.firmId, tokenHash, expiresAt: new Date(Date.now() +
  });

  await db.user.update({ where: { id: user.id }, data: { lastLoginAt: new Date(), loginAttempt

  return { user, accessToken, refreshToken };
}
```

```
// apps/backend/src/middleware/require-auth.ts
import { FastifyRequest, FastifyReply } from 'fastify';
import jwt from 'jsonwebtoken';
import { db } from '../lib/prisma';

export async function requireAuth(request: FastifyRequest, reply: FastifyReply) {
  const token = request.cookies?.access_token?.replace('Bearer ', '');
  // Hackathon shortcut: also accept Authorization header
  const headerToken = request.headers.authorization?.replace('Bearer ', '');
  const raw = token ?? headerToken;

  if (!raw) return reply.code(401).send({ success: false, error: { code: 'TOKEN_MISSING' } });

  try {
    const payload = jwt.verify(raw, process.env.JWT_SECRET!) as {
      sub: string; firmId: string; role: string; email: string;
    };
    const user = await db.user.findUnique({
      where: { id: payload.sub },
      select: { id: true, firmId: true, role: true, isActive: true },
    });
    if (!user || !user.isActive) return reply.code(403).send({ success: false, error: { code:
    (request as any).user = user;
  } catch {
    return reply.code(401).send({ success: false, error: { code: 'TOKEN_INVALID' } });
  }
}
```
# Test login endpoint (after registering auth routes) curl -X POST http://localhost:4000/api/v1/auth/login \ -H "Content-Type: application/json" \ -d '{"email":"sarah.chen@demo.com","password":"Demo!Pass123"}' # Expected: { "success": true, "data": { "user": {...} } } # + Set-Cookie headers with access_token and refresh_token

✅ **Success Criteria:**

- ☐ `POST /auth/login` with seed user credentials returns 200 + JWT
- ☐ Passing JWT to a protected route returns 200
- ☐ Passing expired/missing JWT returns 401
- ☐ `POST /auth/logout` clears tokens

Phase 1 Gate Check ✅ (Hour 4 — Fri 10 PM)

Before moving to Phase 2, all 4 members verify:

- ☐ `npm run dev` — both services start, no errors
- ☐ `curl /api/v1/health` → `{ "status": "ok" }`
- ☐ `curl /api/v1/auth/login` with seed data → JWT returned
- ☐ Prisma Studio shows all 11 tables with seed rows
- ☐ Redis SET/GET working
- ☐ S3 test file exists
- ☐ Vercel preview URL deploying (check Vercel dashboard)

---

4. PHASE 2 — CORE AI AGENTS (Hour 4–12, Fri 10 PM–Sat 6 AM)

Goal: Interrogator Agent asking questions via ElevenLabs, Objection Copilot firing FRE alerts, and Nia FRE corpus indexed. All three testable via manual curl before Phase 3.

Parallel workstreams — split the team:

Hours 4–8	Hours 8–12

Aman: Interrogator Agent + Claude SDK setup	**Aman:** Objection Copilot + Nia FRE query
[M4]: ElevenLabs TTS/STT integration	**[M4]:** WebSocket session room setup
Nikhil: Session + Cases API routes	**Nikhil:** Document upload pipeline (S3 presign)
Dhanush: Design system + layout shell	**Dhanush:** Login page + Dashboard shell

---

Step 2.1 — Claude SDK Agent Framework

Owner: Aman
Duration: 90 minutes
Goal: Reusable Claude streaming function that all 4 agents will call.

```
// apps/backend/src/lib/claude.ts
import Anthropic from '@anthropic-ai/sdk';

export const anthropic = new Anthropic({ apiKey: process.env.ANTHROPIC_API_KEY! });

export const CLAUDE_MODEL = 'claude-sonnet-4-20250514';

/**
 * Non-streaming Claude call — for Objection Copilot, Inconsistency Detector
 */
export async function claudeChat(
  systemPrompt: string,
  userMessage: string,
  maxTokens = 1024
): Promise<string> {
  const response = await anthropic.messages.create({
    model: CLAUDE_MODEL,
    max_tokens: maxTokens,
    system: systemPrompt,
    messages: [{ role: 'user', content: userMessage }],
  });
  const block = response.content[0];
  if (block.type !== 'text') throw new Error('Unexpected Claude response type');
  return block.text;
}

/**
 * Streaming Claude call — for Interrogator question generation
 * Returns an async generator of text chunks
 */
export async function* claudeStream(
  systemPrompt: string,
  userMessage: string,
  maxTokens = 512
): AsyncGenerator<string> {
  const stream = await anthropic.messages.create({
    model: CLAUDE_MODEL,
    max_tokens: maxTokens,
    stream: true,
    system: systemPrompt,
    messages: [{ role: 'user', content: userMessage }],
  });

  for await (const event of stream) {
    if (
      event.type === 'content_block_delta' &&
      event.delta.type === 'text_delta'
    ) {
      yield event.delta.text;
    }
  }
}
```

# Quick test node -e " const { claudeChat } = require('./src/lib/claude'); claudeChat('You are a test.', 'Say pong').then(r => console.log('Claude:', r)); "

Step 2.2 — Nia API Integration + FRE Corpus Indexing

Owner: Aman
Duration: 60 minutes
Goal: Nia client working, FRE rules corpus indexed and queryable.

```
// apps/backend/src/lib/nia.ts
import axios from 'axios';

const nia = axios.create({
  baseURL: process.env.NIA_BASE_URL,
  headers: { Authorization: `Bearer ${process.env.NIA_API_KEY}` },
  timeout: 10_000,
});

export async function indexDocument(params: {
  indexId: string;
  documentId: string;
  content: string;
  metadata?: Record<string, unknown>;
}) {
  const { data } = await nia.post('/index', params);
  return data;
}

export async function searchIndex(params: {
  indexId: string;
  query: string;
  topK?: number;
  filters?: Record<string, unknown>;
}) {
  const { data } = await nia.post('/search', {
    ...params,
    topK: params.topK ?? 5,
  });
  return data.results as Array<{
    id: string;
    content: string;
    score: number;
    metadata: Record<string, unknown>;
  }>;
}
```

```
// scripts/nia/index-fre-corpus.ts — run ONCE before hackathon
import { indexDocument } from '../../apps/backend/src/lib/nia';

const FRE_RULES = [
  {
    id: 'fre-611c',
    rule: 'FRE 611(c)',
    category: 'LEADING',
    description: 'Leading questions on direct examination are generally not permitted...',
    examples: ['Isn\'t it true that...', 'You would agree that...'],
  },
  {
    id: 'fre-801',
    rule: 'FRE 801',
    category: 'HEARSAY',
    description: 'Hearsay is an out-of-court statement offered to prove the truth of the matte
    examples: ['Tell us what your colleague said...', 'What did the report say?'],
  },
  // ... all 5 FRE categories
];

async function main() {
  for (const rule of FRE_RULES) {
    await indexDocument({
      indexId: process.env.NIA_FRE_CORPUS_INDEX_ID!,
      documentId: rule.id,
      content: `${rule.rule}: ${rule.description}\n\nExamples: ${rule.examples.join('; ')}`,
      metadata: { rule: rule.rule, category: rule.category },
    });
    console.log(`✅ Indexed: ${rule.rule}`);
  }
}

main();
```

# Index FRE corpus (run before hackathon, or Hour 0) npx tsx scripts/nia/index-fre-corpus.ts # Test query npx tsx -e " const { searchIndex } = require('./apps/backend/src/lib/nia'); searchIndex({ indexId: process.env.NIA_FRE_CORPUS_INDEX_ID, query: 'Isn\\'t it true you knew about the dosage?' }).then(r => console.log('Nia results:', r)); "

Step 2.3 — Interrogator Agent (P0.1)

Owner: Aman
Duration: 90 minutes
Goal: `POST /sessions/:id/agents/question` returning streamed question text within 2 seconds.

```
// apps/backend/src/modules/agents/interrogator.agent.ts
import { claudeStream } from '../../lib/claude';
import { searchIndex } from '../../lib/nia';

const INTERROGATOR_SYSTEM = `You are a highly skilled opposing counsel conducting a deposition
Your goal is to expose inconsistencies in the witness's testimony.
You ask ONE focused question at a time. Questions are precise, legally professional.
You adapt based on the witness's prior answers and detected hesitations.
NEVER ask compound questions. NEVER reveal your strategy.
Format: Return only the question text, no preamble.`;

export async function generateQuestion(params: {
  caseType: string;
  witnessRole: string;
  currentTopic: string;
  aggressionLevel: 'STANDARD' | 'ELEVATED' | 'HIGH_STAKES';
  priorAnswer?: string;
  questionNumber: number;
  hesitationDetected: boolean;
  recentInconsistencyFlag: boolean;
  niaSessionContextId: string;
  priorWeakAreas?: string[];
}): AsyncGenerator<string> {
  // Retrieve relevant prior statements from Nia
  const niaContext = params.priorAnswer
    ? await searchIndex({
        indexId: params.niaSessionContextId,
        query: params.priorAnswer,
        topK: 3,
      })
    : [];

  const aggressionInstructions = {
    STANDARD: 'Ask methodically. Allow witness to elaborate.',
    ELEVATED: 'Press on contradictions. Use controlled silence.',
    HIGH_STAKES: 'Maximum pressure. Expose inconsistencies directly. Demand specifics.',
  }[params.aggressionLevel];

  const userMessage = `
Case type: ${params.caseType}
Witness role: ${params.witnessRole}
Current focus topic: ${params.currentTopic}
Question number: ${params.questionNumber}
${params.priorAnswer ? `Witness last answered: "${params.priorAnswer}"` : 'First question on t
${params.hesitationDetected ? '⚠️ Witness hesitated significantly before answering.' : ''}
${params.recentInconsistencyFlag ? '🚨 Inconsistency detected in last answer — probe this area
${niaContext.length > 0 ? `Relevant prior sworn statements:\n${niaContext.map(r => `- "${r.con
Prior weak areas for this witness: ${params.priorWeakAreas?.join(', ') ?? 'None (first session
Aggression instruction: ${aggressionInstructions}

Generate the next deposition question:`.trim();

  return claudeStream(INTERROGATOR_SYSTEM, userMessage, 200);
}
```

```
// Route: POST /api/v1/sessions/:sessionId/agents/question
// In sessions module routes file:
fastify.post('/:sessionId/agents/question', {
  preHandler: [requireAuth],
}, async (request, reply) => {
  const { sessionId } = request.params as { sessionId: string };
  const body = request.body as {
    questionNumber: number;
    priorAnswer?: string;
    hesitationDetected: boolean;
    recentInconsistencyFlag: boolean;
    currentTopic: string;
  };

  const session = await db.session.findUnique({
    where: { id: sessionId },
    include: { case: true },
  });
  if (!session) return reply.code(404).send({ error: 'NOT_FOUND' });

  // Stream response as Server-Sent Events
  reply.raw.setHeader('Content-Type', 'text/event-stream');
  reply.raw.setHeader('Cache-Control', 'no-cache');
  reply.raw.setHeader('Connection', 'keep-alive');

  const start = Date.now();
  let fullText = '';

  reply.raw.write(`data: ${JSON.stringify({ type: 'QUESTION_START', questionNumber: body.quest

  const stream = generateQuestion({
    caseType: session.case.caseType,
    witnessRole: 'DEFENDANT',
    currentTopic: body.currentTopic,
    aggressionLevel: session.aggressionLevel as any,
    priorAnswer: body.priorAnswer,
    questionNumber: body.questionNumber,
    hesitationDetected: body.hesitationDetected,
    recentInconsistencyFlag: body.recentInconsistencyFlag,
    niaSessionContextId: session.niaSe ssionContextId ?? session.id,
    priorWeakAreas: (session.priorWeakAreas as any)?.lowestAxes,
  });

  for await (const chunk of stream) {
    fullText += chunk;
    reply.raw.write(`data: ${JSON.stringify({ type: 'QUESTION_CHUNK', text: chunk })}\n\n`);
  }

  const latencyMs = Date.now() - start;
  reply.raw.write(`data: ${JSON.stringify({ type: 'QUESTION_END', fullText, latencyMs })}\n\n`
  reply.raw.end();

  // Log event
  await db.sessionEvent.create({
    data: {
      sessionId,
      firmId: session.firmId,
      eventType: 'QUESTION_DELIVERED',
      questionNumber: body.questionNumber,
      speaker: 'INTERROGATOR',
      textContent: fullText,
      metadata: { latencyMs },
      occurredAt: new Date(),
    },
  });
});
```

# Test Interrogator via curl curl -N -X POST http://localhost:4000/api/v1/sessions/SEED_SESSION_ID/agents/question \ -H "Authorization: Bearer YOUR_JWT" \ -H "Content-Type: application/json" \ -d '{ "questionNumber": 1, "currentTopic": "TIMELINE_CHRONOLOGY", "hesitationDetected": false, "recentInconsistencyFlag": false }' # Expected: stream of SSE events with question chunks, ending with QUESTION_END

✅ **Success Criteria:**

- ☐ First question chunk arrives within 2 seconds of request
- ☐ Full question arrives within 4 seconds
- ☐ `session_events` row created for `QUESTION_DELIVERED`
- ☐ Question is legally appropriate for the case type
- ☐ With `recentInconsistencyFlag: true`, question presses harder on the topic

Step 2.4 — ElevenLabs TTS/STT Integration

Owner: [Member 4]
Duration: 90 minutes
Goal: Interrogator question text → ElevenLabs audio → WebSocket to witness. STT transcribing witness audio → text.

```
// apps/backend/src/lib/elevenlabs.ts
import { ElevenLabsClient } from 'elevenlabs';
import { Readable } from 'stream';

export const eleven = new ElevenLabsClient({ apiKey: process.env.ELEVENLABS_API_KEY! });

export const VOICES = {
  INTERROGATOR: process.env.ELEVENLABS_INTERROGATOR_VOICE_ID!, // 'Adam'
  COACH: process.env.ELEVENLABS_COACH_VOICE_ID!,               // 'Rachel'
};

/**
 * Convert text to audio stream using ElevenLabs TTS
 * Returns a ReadableStream of audio bytes (mp3)
 */
export async function textToSpeech(text: string, voiceId: string): Promise<Readable> {
  const audioStream = await eleven.generate({
    voice: voiceId,
    text,
    model_id: 'eleven_turbo_v2_5', // lowest latency model
    stream: true,
  });
  return audioStream as unknown as Readable;
}
```

```
/**
 * Convert audio buffer to text using ElevenLabs STT
 */
export async function speechToText(audioBuffer: Buffer): Promise<string> {
  const transcription = await eleven.speechToText.convert({
    audio: audioBuffer,
    model_id: 'scribe_v1',
  });
  return transcription.text;
}
```

```
// apps/backend/src/modules/sessions/sessions.websocket.ts
// After Interrogator generates question text → pipe TTS audio to witness via WebSocket

io.on('connection', (socket) => {
  const { sessionId, role } = socket.handshake.query; // 'attorney' or 'witness'
  socket.join(`session:${sessionId}`);

  if (role === 'witness') {
    socket.on('answer_audio', async (data: { audioBuffer: ArrayBuffer; questionNumber: number
      const buffer = Buffer.from(data.audioBuffer);
      const startTime = Date.now();

      try {
        const transcribedText = await speechToText(buffer);
        const sttLatencyMs = Date.now() - startTime;

        // Emit transcription to attorney
        io.to(`session:${sessionId}:attorney`).emit('answer_received', {
          questionNumber: data.questionNumber,
          transcribedText,
          sttLatencyMs,
        });

        // Trigger Objection + Inconsistency analysis
        // (see Step 2.5 and Phase 3 Step 3.1)
      } catch (err) {
        // STT fallback: request text input from witness
        socket.emit('stt_fallback', { message: 'Speech recognition unavailable. Please type yo
      }
    });
  }
});
```

✅ **Success Criteria:**

- ☐ `textToSpeech("What was the dosage?", VOICES.INTERROGATOR)` returns audio stream
- ☐ Audio plays correctly in browser when streamed via WebSocket
- ☐ `speechToText(audioBuffer)` returns accurate text within 3 seconds
- ☐ STT fallback message received when audio is too noisy

Step 2.5 — Objection Copilot (P0.2)

Owner: Aman
Duration: 60 minutes
Goal: `POST /sessions/:id/agents/objection` classifying questions in ≤1.5s. Must fire alert via WebSocket.

```
// apps/backend/src/modules/agents/objection.agent.ts
import { claudeChat } from '../../lib/claude';
import { searchIndex } from '../../lib/nia';
```

```
const OBJECTION_SYSTEM = `You are an expert attorney specializing in evidence law and Federal
Analyze the given deposition question for objectionable content.
Respond ONLY with valid JSON. No preamble, no markdown.

JSON format:
{
  "isObjectionable": boolean,
  "category": "LEADING" | "HEARSAY" | "COMPOUND" | "ASSUMES_FACTS" | "SPECULATION" | null,
  "freRule": string | null,
  "explanation": string | null,
  "confidence": number
}`;

export async function analyzeForObjections(params: {
  questionText: string;
  sessionId: string;
}): Promise<{
  isObjectionable: boolean;
  category: string | null;
  freRule: string | null;
  explanation: string | null;
  confidence: number;
}> {
  // Retrieve relevant FRE rules from Nia for context
  const [niaClaude] = await Promise.all([
    claudeChat(
      OBJECTION_SYSTEM,
      `Analyze this deposition question for FRE objections:\n\n"${params.questionText}"`,
      256
    ),
    searchIndex({
      indexId: process.env.NIA_FRE_CORPUS_INDEX_ID!,
      query: params.questionText,
      topK: 2,
    }),
  ]);

  const result = JSON.parse(niaClaude);
  return result;
}
```

# Test Objection Copilot curl -X POST http://localhost:4000/api/v1/sessions/SEED_ID/agents/objection \ -H "Authorization: Bearer JWT" \ -H "Content-Type: application/json" \ -d '{ "questionNumber": 3, "questionText": "Isn'\"t it true you had completely forgotten about the dosage by then?", "questionTimestamp": "2026-02-21T15:14:22.000Z" }' # Expected in <1500ms: # { "isObjectionable": true, "category": "LEADING", "freRule": "FRE 611(c)", "confidence": 0.94 } # + WebSocket objection_alert event fired to attorney room

✅ Success Criteria:

- ☐ Response arrives within 1,500ms (measure `processingMs` in response)
- ☐ "Isn't it true..." correctly flagged as LEADING (FRE 611c)
- ☐ "What did the doctor tell you?" correctly flagged as HEARSAY (FRE 801)
- ☐ "Where were you on Tuesday?" correctly returns `isObjectionable: false`
- ☐ `alerts` row created in database
- ☐ WebSocket `objection_alert` event received in browser console

Phase 2 Gate Check ✅ (Hour 12 — Sat 6 AM)

- ☐ Interrogator question streams from Claude → ElevenLabs TTS → audio to witness browser
- ☐ Objection Copilot fires within 1.5s — `objection_alert` visible in browser console
- ☐ Nia FRE corpus returns results for objectionable questions
- ☐ Login flow works end-to-end (browser form → JWT → protected API call)
- ☐ Dashboard page renders with seed case data

- [ ] Cases API: GET /cases, POST /cases working

---

5. PHASE 3 — FULL PIPELINE (Hour 12–24, Sat 6 AM–6 PM)

Goal: Complete end-to-end session flow: upload document → Nia ingestion → configure session → live session with all 3 P0 agents → Inconsistency Detector detecting the demo contradiction.

This is the highest-risk phase. If any step takes >2x estimated time, escalate immediately.

Step 3.1 — Inconsistency Detector + Nemotron (P0.3)

Owner: Aman
Duration: 120 minutes
Goal: Witness answer compared against Nia-retrieved prior statements, Nemotron scoring, alert fired at ≥0.75 confidence.

```
// apps/backend/src/lib/nemotron.ts
import axios from 'axios';

const nemotron = axios.create({
  baseURL: process.env.NEMOTRON_BASE_URL,
  headers: { Authorization: `Bearer ${process.env.NEMOTRON_API_KEY}` },
  timeout: parseInt(process.env.NEMOTRON_TIMEOUT_MS ?? '5000'),
});

export async function scoreContradiction(params: {
  witnessAnswer: string;
  priorStatements: Array<{ content: string; metadata: Record<string, unknown> }>;
  caseContext: string;
}): Promise<{
  contradiction_confidence: number;
  best_match_index: number;
  reasoning: string;
}> {
  const prompt = `You are analyzing a witness deposition for contradictions.

Case context: ${params.caseContext}

Witness answer just given:
"${params.witnessAnswer}"

Prior sworn statements on record:
${params.priorStatements.map((s, i) => `[${i}] "${s.content}"`).join('\n')}

Analyze whether the witness answer contradicts any prior statement.
Respond ONLY with JSON:
{
  "contradiction_confidence": <float 0.0-1.0>,
  "best_match_index": <integer index of most contradicted statement, or -1>,
  "reasoning": "<one sentence explanation>"
}`;

  const { data } = await nemotron.post('/chat/completions', {
    model: process.env.NEMOTRON_MODEL,
    messages: [{ role: 'user', content: prompt }],
    max_tokens: 200,
    temperature: 0.1, // Low temperature for consistent scoring
  });

  const text = data.choices[0].message.content;
  return JSON.parse(text);
}
```

```
// apps/backend/src/modules/agents/inconsistency.agent.ts
import { scoreContradiction } from '../../lib/nemotron';
import { claudeChat } from '../../lib/claude';
import { searchIndex } from '../../lib/nia';

const CONFIDENCE_THRESHOLD_LIVE = 0.75;
const CONFIDENCE_THRESHOLD_SECONDARY = 0.50;
const CONFIDENCE_THRESHOLD_CLAUDE_FALLBACK = 0.85; // raised when Nemotron unavailable

export async function detectInconsistency(params: {
  questionText: string;
  answerText: string;
  sessionId: string;
  niaSessionContextId: string;
  caseType: string;
  behavioralCorroboration?: {
    emotionCategory: string;
    durationMs: number;
  };
}): Promise<{
  flagFound: boolean;
  isLiveFired: boolean;
  contradictionConfidence: number;
  priorQuote: string | null;
  priorDocumentPage: number | null;
  priorDocumentLine: number | null;
  impeachmentRisk: 'STANDARD' | 'HIGH';
}> {
  // Step 1: Semantic search for related prior statements via Nia
  const priorStatements = await searchIndex({
    indexId: params.niaSessionContextId,
    query: params.answerText,
    topK: 5,
  });

  if (priorStatements.length === 0) {
    return { flagFound: false, isLiveFired: false, contradictionConfidence: 0, priorQuote: nul
  }

  // Step 2: Nemotron scoring (with Claude fallback)
  let score: { contradiction_confidence: number; best_match_index: number };
  let usingFallback = false;

  try {
    score = await scoreContradiction({
      witnessAnswer: params.answerText,
      priorStatements,
      caseContext: `${params.caseType} deposition`,
    });
  } catch {
    // Nemotron unavailable — fallback to Claude with raised threshold
    usingFallback = true;
    const claudeResult = await claudeChat(
      'Score contradiction confidence 0-1. Return only JSON: {"contradiction_confidence": floa
      `Answer: "${params.answerText}"\nPrior statements:\n${priorStatements.map((s, i) => `[${
    );
    score = JSON.parse(claudeResult);
  }

  const threshold = usingFallback ? CONFIDENCE_THRESHOLD_CLAUDE_FALLBACK : CONFIDENCE_THRESHOL
  const confidence = score.contradiction_confidence;
  const bestMatch = priorStatements[score.best_match_index];

  // Step 3: Routing by confidence
  if (confidence < CONFIDENCE_THRESHOLD_SECONDARY) {
    return { flagFound: false, isLiveFired: false, contradictionConfidence: confidence, priorQ
  }

  const isLiveFired = confidence >= threshold;

  // Step 4: Impeachment risk — upgrade if Behavioral Sentinel corroborates
  let impeachmentRisk: 'STANDARD' | 'HIGH' = 'STANDARD';
  let adjustedConfidence = confidence;

  if (params.behavioralCorroboration && params.behavioralCorroboration.durationMs >= 800) {
    impeachmentRisk = 'HIGH';
    adjustedConfidence = Math.min(1.0, confidence + 0.05);
  }

  return {
    flagFound: true,
    isLiveFired,
    contradictionConfidence: adjustedConfidence,
    priorQuote: bestMatch?.content ?? null,
    priorDocumentPage: (bestMatch?.metadata?.page as number) ?? null,
    priorDocumentLine: (bestMatch?.metadata?.line as number) ?? null,
    impeachmentRisk,
  };
}
```

# Test Inconsistency Detector curl -X POST http://localhost:4000/api/v1/sessions/SEED_ID/agents/inconsistency \ -H "Authorization: Bearer JWT" \ -H "Content-Type: application/json" \ -d '{ "questionNumber": 8, "questionText": "What was the exact dosage you administered?", "answerText": "Approximately $200, in that range", "answerTimestamp": "2026-02-21T15:15:03.000Z" }' # Expected (given "The dosage was exactly $217" is in the demo doc): # { "flagFound": true, "contradictionConfidence": 0.91, "priorQuote": "The dosage was exactly $217.", "impeachmentRisk": "STANDARD" } # And with behavioralCorroboration.durationMs >= 800: impeachmentRisk: "HIGH"

✅ **Success Criteria:**

- ☐ Demo contradiction ($200 vs $217) detected with confidence ≥0.75
- ☐ Response arrives within 4,000ms
- ☐ `isLiveFired: true` for confidence ≥0.75
- ☐ `isLiveFired: false` for confidence 0.50–0.74
- ☐ `flagFound: false` for unrelated answer
- ☐ Nemotron timeout → graceful Claude fallback, threshold raised to 0.85
- ☐ WebSocket `inconsistency_alert` fires to attorney room

Step 3.2 — Document Ingestion Pipeline (P0.4)

Owner: Nikhil
Duration: 120 minutes
Goal: Upload PDF → S3 → pre-screen → Nia indexing → fact extraction → ready in <3 min for 50-page demo doc.

```
// apps/backend/src/modules/documents/ingestion.worker.ts
import { db } from '../../lib/prisma';
import { redis } from '../../lib/redis';
import { indexDocument, searchIndex } from '../../lib/nia';
import { claudeChat } from '../../lib/claude';
import pdfParse from 'pdf-parse';
import mammoth from 'mammoth';
import { GetObjectCommand } from '@aws-sdk/client-s3';
import { s3, S3_BUCKET } from '../../lib/s3';

export async function runIngestionPipeline(documentId: string) {
  const doc = await db.document.findUnique({ where: { id: documentId } });
  if (!doc) return;

  const updateStatus = async (status: string, extra = {}) => {
    await db.document.update({ where: { id: documentId }, data: { ingestionStatus: status, ...
    await redis.set(`ingestion:${documentId}`, JSON.stringify({ status, ...extra }), 'EX', 600
  };

  try {
    await updateStatus('INDEXING', { ingestionStartedAt: new Date() });

    // Step 1: Fetch file from S3
    const s3Obj = await s3.send(new GetObjectCommand({ Bucket: S3_BUCKET, Key: doc.s3Key }));
    const fileBuffer = Buffer.from(await streamToBuffer(s3Obj.Body as any));

    // Step 2: Extract text based on MIME type
    let textContent: string;
    let pageCount = 1;

    if (doc.mimeType === 'application/pdf') {
      const parsed = await pdfParse(fileBuffer);
      textContent = parsed.text;
      pageCount = parsed.numpages;
    } else if (doc.mimeType.includes('wordprocessingml')) {
      const { value } = await mammoth.extractRawText({ buffer: fileBuffer });
      textContent = value;
    } else {
      textContent = fileBuffer.toString('utf-8');
    }

    if (!textContent.trim()) {
      throw new Error('No text content found in document. Scanned/image PDFs are not supported
    }

    await redis.set(`ingestion:${documentId}`, JSON.stringify({ status: 'INDEXING', progress:

    // Step 3: Send to Nia for indexing
    await indexDocument({
      indexId: `case:${doc.caseId}`,
      documentId: doc.id,
      content: textContent,
      metadata: { docType: doc.docType, caseId: doc.caseId, firmId: doc.firmId },
    });

    await redis.set(`ingestion:${documentId}`, JSON.stringify({ status: 'INDEXING', progress:

    // Step 4: Extract structured facts via Claude
    const factsJson = await claudeChat(
      'Extract structured facts from this legal document. Return ONLY valid JSON with keys: pa
      `Document type: ${doc.docType}\n\nContent:\n${textContent.slice(0, 12000)}`,
      2048
    );

    const extractedFacts = JSON.parse(factsJson);

    // Step 5: Mark as READY
    await updateStatus('READY', {
      pageCount,
      ingestionCompletedAt: new Date(),
      extractedFacts,
      niaIndexId: `case:${doc.caseId}:${doc.id}`,
    });

    console.log(`✅ Ingestion complete for ${documentId} (${pageCount} pages)`);
  } catch (err: any) {
    await updateStatus('FAILED', {
      ingestionError: err.message,
      ingestionCompletedAt: new Date(),
    });
    console.error(`❌ Ingestion failed for ${documentId}:`, err.message);
  }
}

async function streamToBuffer(stream: NodeJS.ReadableStream): Promise<Buffer> {
  const chunks: Buffer[] = [];
  for await (const chunk of stream) chunks.push(Buffer.from(chunk));
  return Buffer.concat(chunks);
}
```

# Test ingestion with demo PDF curl -X POST http://localhost:4000/api/v1/cases/DEMO_CASE_ID/documents/presign \ -H "Authorization: Bearer JWT" \ -H "Content-Type: application/json" \ -d '{"filename":"chen_depo_2024.pdf","mimeType":"application/pdf","fileSizeBytes":2097152}' # → get uploadUrl and documentId # Upload file directly to S3 presigned URL (frontend handles this, but test with curl) curl -X PUT "PRESIGNED_URL" \ -H "Content-Type: application/pdf" \ --data-binary @demo/chen_v_metropolitan.pdf # Confirm upload and trigger ingestion curl -X POST http://localhost:4000/api/v1/cases/DEMO_CASE_ID/documents/DOCUMENT_ID/confirm-upload \ -H "Authorization: Bearer JWT" # Poll status every 3 seconds watch -n 3 curl -s http://localhost:4000/api/v1/cases/DEMO_CASE_ID/documents/DOCUMENT_ID/ingestion-status \ -H "Authorization: Bearer JWT" # Expected progression: UPLOADING → INDEXING (progress 30% → 70%) → READY

✅ Success Criteria:

- ☐ Demo PDF (50 pages) reaches READY status in <3 minutes
- ☐ `extractedFacts` contains recognizable parties, dates, disputed facts from the demo doc
- ☐ Nia search on the demo document returns relevant prior statements
- ☐ Duplicate upload (same file hash) returns `409 DUPLICATE_DOCUMENT`
- ☐ Image-only PDF returns error with message about text extraction

---

Step 3.3 — Live Session WebSocket Plumbing

Owner: [Member 4]
Duration: 90 minutes
Goal: Full WebSocket session room — attorney and witness both connected, events flowing both directions.

```
// apps/backend/src/modules/sessions/sessions.websocket.ts — full implementation
import { Server, Socket } from 'socket.io';
import jwt from 'jsonwebtoken';
import { redis } from '../../lib/redis';
import { db } from '../../lib/prisma';
import { bufferSessionEvent } from './sessions.buffer';

export function registerSessionWebSocket(io: Server) {
  io.on('connection', async (socket: Socket) => {
    const { sessionId, token, role } = socket.handshake.query as {
      sessionId: string; token?: string; role?: string;
    };

    // Auth: attorney uses JWT, witness uses token
    let userId: string | null = null;
    if (role === 'witness' && token) {
      const data = await redis.get(`witness:${token}`);
      if (!data) return socket.disconnect();
      const { sessionId: tokenSessionId } = JSON.parse(data);
      if (tokenSessionId !== sessionId) return socket.disconnect();
    } else {
      // JWT auth for attorney
      const jwtToken = socket.handshake.auth.token || token;
      try {
        const payload = jwt.verify(jwtToken, process.env.JWT_SECRET!) as any;
        userId = payload.sub;
      } catch {
        return socket.disconnect();
      }
    }

    // Join appropriate rooms
    socket.join(`session:${sessionId}`);
    if (role === 'witness') {
      socket.join(`session:${sessionId}:witness`);
      // Notify attorney that witness connected
      io.to(`session:${sessionId}:attorney`).emit('witness_connected', { sessionId });
      await db.session.update({ where: { id: sessionId }, data: { witnessJoinedAt: new Date()
    } else {
      socket.join(`session:${sessionId}:attorney`);
    }

    // Handle witness audio answer
    socket.on('answer_audio', async (data) => {
      // Handled in STT integration (Step 2.4)
    });

    // Attorney: add timestamped note
    socket.on('annotation_add', async (data: { questionNumber: number; text: string }) => {
      const session = await db.session.findUnique({ where: { id: sessionId } });
      if (!session) return;
      await db.attorneyAnnotation.create({
        data: {
          sessionId,
          firmId: session.firmId,
          questionNumber: data.questionNumber,
          noteText: data.text,
          sessionTimestampMs: Date.now(),
        },
      });
    });

    // Handle disconnect
    socket.on('disconnect', () => {
      if (role === 'witness') {
        io.to(`session:${sessionId}:attorney`).emit('witness_disconnected', { sessionId });
      }
    });

    // Auto-flush session events every 60 seconds (PRD §8.4 — max 60s data loss)
    const flushInterval = setInterval(async () => {
      await flushSessionEventBuffer(sessionId);
    }, 60_000);

    socket.on('disconnect', () => clearInterval(flushInterval));
  });
}
```
# Test WebSocket connection with wscat npm install -g wscat # Attorney connection wscat -c "ws://localhost:4000/ws?sessionId=SEED_SESSION&role=attorney" \ --header "Authorization: Bearer JWT" # In second terminal — witness connection wscat -c "ws://localhost:4000/ws?sessionId=SEED_SESSION&role=witness&token=WITNESS_TOKEN" # Send a test annotation from attorney: # > {"type":"annotation_add","questionNumber":3,"text":"Good moment to flag"} # Expected: DB row created in attorney_annotations

✅ Success Criteria:

- ☐ Attorney and witness both connected to same session room
- ☐ `witness_connected` event fires to attorney when witness joins
- ☐ `objection_alert` event reaches only attorney socket (not witness)
- ☐ Annotation creates DB row
- ☐ Disconnect fires `witness_disconnected` to attorney room

Step 3.4 — Frontend: Design System + Core Screens

Owner: Dhanush
Duration: Full Phase 3 (parallel to Steps 3.1–3.3)
Goal: All P0 screens built and navigable. Attorney can log in, see dashboard, create a case, and enter the live session screen.

cd apps/frontend # Install UI dependencies npm install \ @radix-ui/react-alert-dialog@1.1.6 \ @radix-ui/react-dialog@1.1.6 \ @radix-ui/react-dropdown-menu@2.1.6 \ @radix-ui/react-label@2.1.2 \ @radix-ui/react-progress@1.1.2 \ @radix-ui/react-select@2.1.6 \ @radix-ui/react-switch@1.1.3 \ @radix-ui/react-tabs@1.1.3 \ @radix-ui/react-toast@1.2.6 \ @radix-ui/react-tooltip@1.1.8 \ class-variance-authority@0.7.1 \ clsx@2.1.1 \ tailwind-merge@2.6.0 \ lucide-react@0.477.0 \ framer-motion@12.4.7 \ zustand@5.0.2 \ @tanstack/react-query@5.66.0 \ axios@1.7.9 \ react-hook-form@7.54.2 \ @hookform/resolvers@3.10.0 \ zod@3.24.1 \ socket.io-client@4.8.1 \ recharts@2.15.0 \ wavesurfer.js@7.8.11 # Install shadcn/ui CLI and init npx shadcn@latest init # Select: New York style, Slate color, yes to CSS variables # Add all needed components npx shadcn@latest add button card badge tabs dialog progress npx shadcn@latest add select switch toast tooltip alert npx shadcn@latest add table input label form

```
// apps/frontend/src/lib/tailwind-config-additions.ts
// Add to tailwind.config.ts — VERDICT design tokens
const verdictColors = {
  'verdict-navy': '#0F1729',       // Primary dark — three-panel session bg
  'verdict-slate': '#1E2B3C',      // Secondary dark — card backgrounds
  'verdict-blue': '#3B82F6',       // Primary accent — attorney actions
  'verdict-red': {
    50: '#FFF0F0',
    500: '#EF4444',                // Alert rail inconsistency
    700: '#B91C1C',                // HIGH IMPEACHMENT RISK badge
  },
  'verdict-amber': '#F59E0B',      // Timer warning, objection alerts
  'verdict-green': '#10B981',      // Agent active, confirmed flags
  'verdict-purple': '#8B5CF6',     // ElevenLabs waveform
};
```

Key screens to build (in priority order):

1. `/login` — email/password form + JWT storage
2. `/dashboard` — case grid, session countdown badges
3. `/cases/new` — case creation form
4. `/cases/:id` — case detail with Documents/Witnesses/Sessions tabs
5. `/cases/:id/documents/facts` — fact review and confirmation
6. `/cases/:id/witnesses/new` — add witness modal
7. `/cases/:id/session/new` — session configuration
8. `/cases/:id/session/:id/lobby` — pre-session waiting room
9. `/cases/:id/session/:id/live` — **three-panel live session UI (most important)**
10. `/briefs/:id` — coaching brief viewer

```
// apps/frontend/src/stores/session.store.ts — Zustand store for live session
import { create } from 'zustand';

interface Alert {
  id: string;
  type: 'OBJECTION' | 'INCONSISTENCY' | 'COMPOSURE';
  questionNumber: number;
  firedAt: string;
  fre Rule?: string;
  priorQuote?: string;
  contradictionConfidence?: number;
  impeachmentRisk?: 'STANDARD' | 'HIGH';
  decision?: 'CONFIRMED' | 'REJECTED' | 'ANNOTATED';
```

```
}

interface SessionStore {
  sessionId: string | null;
  status: 'LOBBY' | 'ACTIVE' | 'PAUSED' | 'COMPLETE';
  questionCount: number;
  elapsedSeconds: number;
  alerts: Alert[];
  transcript: Array<{ speaker: 'INTERROGATOR' | 'WITNESS'; text: string; questionNumber: numbe
  agentStatus: { interrogator: string; objection: string; inconsistency: string; sentinel: str

  // Actions
  addAlert: (alert: Alert) => void;
  resolveAlert: (alertId: string, decision: Alert['decision']) => void;
  addTranscriptLine: (line: { speaker: string; text: string; questionNumber: number }) => void
  incrementQuestion: () => void;
  setStatus: (status: SessionStore['status']) => void;
}

export const useSessionStore = create<SessionStore>((set) => ({
  sessionId: null,
  status: 'LOBBY',
  questionCount: 0,
  elapsedSeconds: 0,
  alerts: [],
  transcript: [],
  agentStatus: { interrogator: 'STANDBY', objection: 'STANDBY', inconsistency: 'STANDBY', sent

  addAlert: (alert) => set((s) => ({ alerts: [alert, ...s.alerts] })),
  resolveAlert: (alertId, decision) => set((s) => ({
    alerts: s.alerts.map(a => a.id === alertId ? { ...a, decision } : a),
  })),
  addTranscriptLine: (line) => set((s) => ({ transcript: [...s.transcript, line] })),
  incrementQuestion: () => set((s) => ({ questionCount: s.questionCount + 1 })),
  setStatus: (status) => set({ status }),
}));
```

✅ **Success Criteria (Phase 3 frontend):**

- ☐ `/login` → submits → JWT stored → redirects to `/dashboard`
- ☐ `/dashboard` renders case cards from API
- ☐ `/cases/new` form creates a case, redirects to case detail
- ☐ Document upload UI shows progress while ingestion runs
- ☐ Fact review screen displays extracted parties, dates, statements
- ☐ Live session screen: three-panel layout renders correctly on 1280px+ desktop
- ☐ Alert card slides in when `objection_alert` WebSocket event arrives

Phase 3 Gate Check ✅ (Hour 24 — Sat 6 PM)

- ☐ Upload `demo/chen_v_metropolitan.pdf` → ingestion completes → facts extracted
- ☐ Create session → attorney and witness both connect via WebSocket
- ☐ Start session → Interrogator asks first question via ElevenLabs audio
- ☐ Witness answers → STT transcribes → transcript appears on attorney screen
- ☐ $217 demo contradiction → Inconsistency Detector fires → alert visible in alert rail
- ☐ Leading question → Objection Copilot fires within 1.5s → alert visible
- ☐ All P0 flows technically functional (polish TBD Phase 4)

---

6. PHASE 4 — POLISH + BRIEF (Hour 24–36, Sat 6 PM–Sun 6 AM)

Goal: P0.5 Coaching Brief fully working. Full session flow is smooth. ElevenLabs Coach voice narrating brief clips. All judge-facing interactions are polished.

Step 4.1 — Review Orchestrator + Coaching Brief (P0.5)

Owner: Aman
Duration: 90 minutes

```
// apps/backend/src/modules/agents/orchestrator.agent.ts
import { claudeChat } from '../../lib/claude';
import { eleven, VOICES } from '../../lib/elevenlabs';
import { db } from '../../lib/prisma';
import { s3, S3_BUCKET } from '../../lib/s3';
import { PutObjectCommand } from '@aws-sdk/client-s3';

export async function generateCoachingBrief(sessionId: string) {
  const session = await db.session.findUnique({
    where: { id: sessionId },
    include: {
      alerts: { where: { attorneyDecision: 'CONFIRMED' } },
      witness: true,
      case: true,
    },
  });
  if (!session) throw new Error('Session not found');

  const confirmedFlags = session.alerts.filter(a => a.alertType === 'INCONSISTENCY' || a.alert
  const objections = session.alerts.filter(a => a.alertType === 'OBJECTION');
  const composureAlerts = session.alerts.filter(a => a.alertType === 'COMPOSURE');

  // Compute topic sub-scores
  const weaknessMapScores = computeWeaknessMap(session.alerts, session.focusAreas);

  // Compute session score (0-100): weighted composite
  const sessionScore = Math.round(
    (1 - confirmedFlags.length / Math.max(session.questionCount, 1)) * 50 + // consistency wei
    (objections.length === 0 ? 25 : Math.max(0, 25 - objections.length * 5)) + // objection ha
    (Math.min(weaknessMapScores.timeline + weaknessMapScores.financial, 100) / 100) * 25 // to
  );

  // Generate narrative via Claude
  const narrativePrompt = `Generate a professional attorney coaching brief for this deposition

Case: ${session.case.name} (${session.case.caseType})
Witness: ${session.witness.name} (${session.witness.role})
Session Number: ${session.sessionNumber}
Session Score: ${sessionScore}/100
Questions asked: ${session.questionCount}
Confirmed inconsistencies: ${confirmedFlags.length}
Objection events: ${objections.length}

Confirmed inconsistencies:
${confirmedFlags.map(f => `- Q${f.questionNumber}: "${f.answerText}" contradicts prior stateme

Weakness scores by topic: ${JSON.stringify(weaknessMapScores)}

Write a 3-paragraph coaching brief. Be specific, actionable, and professional.
End with 3 numbered coaching recommendations.`;

  const narrativeText = await claudeChat(
    'You are a senior litigation coach writing coaching briefs for attorneys preparing witness
    narrativePrompt,
    1500
  );

  // Generate ElevenLabs coach voice clips for each confirmed flag
  const audioManifest = [];
  for (const flag of confirmedFlags.slice(0, 3)) { // max 3 audio clips for brief
    const clipText = `At question ${flag.questionNumber}, the witness said: "${flag.answerText
    const audioStream = await textToSpeech(clipText, VOICES.COACH);
    const s3Key = `firms/${session.firmId}/briefs/${sessionId}/alert_${flag.id}.mp3`;
    // Upload audio to S3
    const chunks: Buffer[] = [];
    for await (const chunk of audioStream) chunks.push(Buffer.from(chunk));
    const audioBuffer = Buffer.concat(chunks);
    await s3.send(new PutObjectCommand({ Bucket: S3_BUCKET, Key: s3Key, Body: audioBuffer, Con
    audioManifest.push({ alertId: flag.id, s3Key, durationMs: Math.round(audioBuffer.length /
  }

  // Extract top 3 recommendations from narrative
  const recommendations = extractRecommendations(narrativeText);

  // Save brief
  const brief = await db.brief.create({
    data: {
      sessionId,
      firmId: session.firmId,
      witnessId: session.witnessId,
      generationStatus: 'COMPLETE',
      generationCompletedAt: new Date(),
      sessionScore,
      consistencyRate: 1 - confirmedFlags.length / Math.max(session.questionCount, 1),
      confirmedFlags: confirmedFlags.length,
      objectionCount: objections.length,
      composureAlertCount: composureAlerts.length,
      topRecommendations: recommendations,
      narrativeText,
      weaknessMapScores,
      elevenlabsAudioManifest: audioManifest,
    },
  });

  // Update witness profile
  await db.witness.update({
    where: { id: session.witnessId },
    data: { latestScore: sessionScore, ...(session.sessionNumber === 1 ? { baselineScore: sess
  });

  return brief;
}

function computeWeaknessMap(alerts: any[], focusAreas: string[]) {
  // Score each axis 0-100 based on alert density per topic
  const axes = ['timeline', 'financial', 'communications', 'relationships', 'actions', 'prior_
  const scores: Record<string, number> = {};
  axes.forEach(axis => { scores[axis] = 75; }); // baseline

  alerts.filter(a => a.attorneyDecision === 'CONFIRMED').forEach(alert => {
    const topic = alert.metadata?.topic?.toLowerCase().replace('_', '') ?? '';
    const matchedAxis = axes.find(a => topic.includes(a.replace('_', '')));
    if (matchedAxis) scores[matchedAxis] = Math.max(0, scores[matchedAxis] - 15);
  });

  return scores;
}

function extractRecommendations(narrativeText: string): string[] {
  const lines = narrativeText.split('\n');
  const numbered = lines.filter(l => /^\d\./.test(l.trim())).slice(0, 3);
  return numbered.length >= 3 ? numbered : [
    'Practice precise, consistent answers for the key disputed facts.',
    'Pause before answering any question that begins with "Isn\'t it true..."',
    'Review all prior deposition testimony carefully before the next session.',
  ];
}
```

✅ **Success Criteria:**

- ☐ Brief generates within 3 minutes of session end
- ☐ `narrativeText` is coherent professional prose (not boilerplate)
- ☐ `topRecommendations` has 3 specific, actionable items
- ☐ ElevenLabs Coach voice audio clips uploaded to S3 and accessible
- ☐ `weaknessMapScores` correctly reflects topics where alerts fired
- ☐ `GET /briefs/:id` returns full brief JSON
- ☐ Brief viewer page renders score, inconsistencies, and recommendations

---

Step 4.2 — Coaching Brief Frontend (P0.5)

Owner: Dhanush
Duration: 60 minutes
Goal: Beautiful, polished brief viewer. Recharts radar chart rendering. Audio clip playback working.

```
// apps/frontend/src/app/briefs/[briefId]/page.tsx — key components

// Score card with animated count-up
'use client';
import { useEffect, useState } from 'react';
import { motion } from 'framer-motion';

function ScoreCard({ score, delta }: { score: number; delta?: number }) {
  const [displayed, setDisplayed] = useState(0);

  useEffect(() => {
    // Count up animation 1.2s
    const start = Date.now();
    const duration = 1200;
    const tick = () => {
      const elapsed = Date.now() - start;
      const progress = Math.min(elapsed / duration, 1);
      setDisplayed(Math.round(progress * score));
      if (progress < 1) requestAnimationFrame(tick);
    };
    requestAnimationFrame(tick);
  }, [score]);

  const color = score >= 75 ? 'text-green-500' : score >= 50 ? 'text-amber-500' : 'text-red-50

  return (
    <motion.div initial={{ opacity: 0, y: 20 }} animate={{ opacity: 1, y: 0 }} transition={{ d
      <div className={`text-7xl font-bold ${color}`}>{displayed}</div>
      <div className="text-gray-400 text-sm">/100</div>
      {delta !== undefined && (
        <div className={`text-sm font-medium ${delta >= 0 ? 'text-green-400' : 'text-red-400'}
          {delta >= 0 ? '▲' : '▼'} {Math.abs(delta)} pts vs Session 1
        </div>
      )}
    </motion.div>
  );
}

// Weakness Map Radar
import { RadarChart, PolarGrid, PolarAngleAxis, Radar, ResponsiveContainer } from 'recharts';

function WeaknessMap({ scores }: { scores: Record<string, number> }) {
  const data = Object.entries(scores).map(([key, value]) => ({
    subject: key.replace('_', ' ').replace(/\b\w/g, c => c.toUpperCase()),
    value,
    fullMark: 100,
  }));
  return (
    <ResponsiveContainer width="100%" height={400}>
      <RadarChart data={data}>
        <PolarGrid stroke="#1E2B3C" />
        <PolarAngleAxis dataKey="subject" tick={{ fill: '#94A3B8', fontSize: 12 }} />
        <Radar name="Score" dataKey="value" stroke="#3B82F6" fill="#3B82F6" fillOpacity={0.3}
      </RadarChart>
    </ResponsiveContainer>
  );
}
```

✅ **Success Criteria:**

- ☐ Score card number counts up from 0 → actual score in 1.2s
- ☐ Score color: green ≥75, amber 50–74, red <50
- ☐ Radar chart renders with 6 axes (7 if Sentinel active) with correct scores
- ☐ Audio clip plays when attorney clicks [▶ Play clip] on inconsistency row
- ☐ HIGH IMPEACHMENT RISK badge renders in red with pulse animation
- ☐ [Download PDF] button triggers PDF generation

Step 4.3 — UI Polish Pass

Owner: Dhanush + [Member 4]
Duration: 90 minutes
Goal: Live session screen looks professional. Alert rail animations working. All P0 screens are demo-ready.

Critical UI checklist for judges:

- ☐ Three-panel layout: left control (220px) | center transcript | right alert rail (320px)
- ☐ Alert card slides in from right with spring animation (300ms)
- ☐ Red border pulse on HIGH IMPEACHMENT RISK alerts (×2 cycles)
- ☐ Timer shows amber at <10 min, red at <5 min
- ☐ Waveform animates when Interrogator speaks (wavesurfer.js)
- ☐ "Monitoring active" green pulse in alert rail header
- ☐ [CONFIRMED] click → green flash → grays out card
- ☐ [REJECTED] click → slides out right 200ms
- ☐ Transcript auto-scrolls to latest exchange

Phase 4 Gate Check ✅ (Hour 36 — Sun 6 AM)

- ☐ Full demo run-through: login → create case → upload doc → add witness → configure session → start → 3 questions → contradiction detected → end → brief generated → brief viewed
- ☐ ElevenLabs coach voice plays in brief
- ☐ Recharts radar chart renders correctly
- ☐ All animations functional
- ☐ No console errors during demo flow
- ☐ Brief PDF download working

---

7. PHASE 5 — INTEGRATION + P1 FEATURES (Hour 36–44, Sun 6 AM–2 PM)

Goal: P1.4 Behavioral Sentinel, P1.1 multi-session witness profile, P1.2 Weakness Map in Databricks, Databricks Delta Live Tables streaming. These are sponsor prize differentiators — implement only what can be demo'd cleanly.

Step 5.1 — Behavioral Sentinel Frontend (P1.4)

Owner: Dhanush + Aman
Duration: 60 minutes
Goal: MediaPipe Face Mesh running in-browser, AU vectors sent to backend, composure alerts firing.

```
// apps/frontend/src/lib/mediapipe-sentinel.ts
import { FaceLandmarker, FilesetResolver } from '@mediapipe/tasks-vision';

let faceLandmarker: FaceLandmarker | null = null;

export async function initMediaPipe() {
  const vision = await FilesetResolver.forVisionTasks(
    'https://cdn.jsdelivr.net/npm/@mediapipe/tasks-vision@0.10.20/wasm'
  );
  faceLandmarker = await FaceLandmarker.createFromOptions(vision, {
    baseOptions: {
      modelAssetPath: '/models/face_landmarker.task',
      delegate: 'GPU',
    },
    runningMode: 'VIDEO',
    numFaces: 1,
    outputFaceBlendshapes: true,
  });
  return faceLandmarker;
}

export function extractAUVectors(landmarks: any[]): {
  au4: number; au6: number; au12: number; au20: number; au45: number;
} {
  // Map MediaPipe blendshape scores to FACS Action Units
  if (!landmarks || landmarks.length === 0) return { au4: 0, au6: 0, au12: 0, au20: 0, au45: 0

  const bs = landmarks[0].categories;
  const findScore = (name: string) => bs.find((c: any) => c.categoryName === name)?.score ?? 0

  return {
    au4: findScore('browInnerUp'),          // Brow furrow
    au6: findScore('cheekSquintLeft'),       // Cheek raise
    au12: findScore('mouthSmileLeft'),       // Lip corner
    au20: findScore('mouthStretchLeft'),     // Lip stretch (fear)
    au45: findScore('eyeBlinkLeft'),         // Blink rate
  };
}

// React hook for witness view
export function useBehavioralSentinel(sessionId: string, socket: any) {
  useEffect(() => {
    if (!sentinelEnabled) return;

    let videoEl: HTMLVideoElement;
    let animFrameId: number;
    let expressionStartTime: number | null = null;
    let currentExpression: string | null = null;

    const processFrame = async () => {
      if (!faceLandmarker || !videoEl) return animationFrameId = requestAnimationFrame(process

      const results = faceLandmarker.detectForVideo(videoEl, Date.now());
      const auVectors = extractAUVectors(results.faceBlendshapes ?? []);

      // Detect sustained Fear expression (AU4 + AU20 both > 0.6 for ≥800ms)
      const isFear = auVectors.au4 > 0.6 && auVectors.au20 > 0.6;
      if (isFear && !expressionStartTime) {
        expressionStartTime = Date.now();
        currentExpression = 'FEAR';
      } else if (!isFear) {
        if (expressionStartTime && currentExpression) {
          const durationMs = Date.now() - expressionStartTime;
          if (durationMs >= 800) {
            // Send to backend
            socket.emit('behavioral_vectors', { auVectors, durationMs, questionNumber: current
          }
        }
        expressionStartTime = null;
        currentExpression = null;
      }

      animFrameId = requestAnimationFrame(processFrame);
    };

    // Initialize camera
    navigator.mediaDevices.getUserMedia({ video: true }).then(stream => {
      videoEl = document.createElement('video');
      videoEl.srcObject = stream;
      videoEl.play();
      initMediaPipe().then(() => requestAnimationFrame(processFrame));
    }).catch(() => {
      console.warn('Camera denied — Behavioral Sentinel disabled');
    });

    return () => cancelAnimationFrame(animFrameId);
  }, [sentinelEnabled]);
}
```

✅ Success Criteria:

- ☐ Camera permission prompt appears when Sentinel is enabled
- ☐ AU vectors appear in backend logs when simulating Fear expression
- ☐ `POST /sessions/:id/agents/behavioral` creates COMPOSURE alert
- ☐ `composure_alert` WebSocket event appears in attorney alert rail
- ☐ Inconsistency flag upgraded to HIGH IMPEACHMENT RISK when Sentinel corroborates
- ☐ Camera denial → graceful silent degradation (no error, sentinel badge shows "Inactive")

Step 5.2 — Databricks Delta Lake Integration

Owner: Nikhil
Duration: 60 minutes
Goal: Session events streaming to Databricks, Weakness Map data queryable.

```
// apps/backend/src/lib/databricks.ts
import { DBSQLClient } from '@databricks/sql';

const databricksClient = new DBSQLClient();

let databricksSession: any = null;

async function getDatabricksSession() {
  if (databricksSession) return databricksSession;

  await databricksClient.connect({
    host: process.env.DATABRICKS_HOST!,
    path: `/sql/1.0/warehouses/${process.env.DATABRICKS_SQL_WAREHOUSE_ID}`,
    token: process.env.DATABRICKS_TOKEN!,
  });
  databricksSession = await databricksClient.openSession({
    initialCatalog: process.env.DATABRICKS_CATALOG,
    initialSchema: process.env.DATABRICKS_SCHEMA,
  });

  return databricksSession;
}

export async function streamSessionEventsToDelta(events: any[]) {
  if (!events.length) return;

  try {
    const session = await getDatabricksSession();
    const operation = await session.executeStatement(`
      INSERT INTO session_events_stream
      SELECT * FROM VALUES
      ${events.map(e => `('${e.sessionId}', '${e.firmId}', '${e.eventType}', '${e.occurredAt}'
    `);
    await operation.close();
    console.log(`✅ Streamed ${events.length} events to Databricks Delta`);
  } catch (err) {
    // Non-critical — log but don't break session
    console.warn('Databricks stream failed (non-critical):', err);
  }
}

export async function queryWeaknessMapScores(sessionId: string): Promise<Record<string, number
  try {
    const session = await getDatabricksSession();
    const operation = await session.executeStatement(`
      SELECT topic_axis, AVG(confidence_score) as avg_score
      FROM inconsistency_flags
      WHERE session_id = '${sessionId}'
      GROUP BY topic_axis
    `);
    const result = await operation.fetchAll();
    await operation.close();
    return Object.fromEntries(result.map((r: any) => [r.topic_axis, 100 - r.avg_score * 100]))
  } catch {
    return {}; // Graceful degradation
  }
}
```

✅ **Success Criteria:**

- ☐ Session events appear in Databricks Delta table after session ends
- ☐ Emotion vectors visible in Delta Live Tables UI (for judge demo)
- ☐ Weakness Map scores queryable via Databricks SQL

Step 5.3 — Multi-Session Witness Profile (P1.1)

Owner: Dhanush + Nikhil
Duration: 45 minutes
Goal: Witness profile page shows score trend chart, improvement delta, and persisting inconsistencies.

```
// apps/frontend/src/app/cases/[caseId]/witnesses/[witnessId]/page.tsx
// Score trend chart using Recharts LineChart
import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, ReferenceLine } from 'recharts

function ScoreTrendChart({ sessions, depositionDate }: { sessions: any[]; depositionDate?: str
  const data = sessions.map((s, i) => ({
    name: `Session ${i + 1}`,
    score: s.score,
    date: new Date(s.endedAt).toLocaleDateString(),
  }));

  if (depositionDate) {
    data.push({ name: 'Deposition', score: null as any, date: new Date(depositionDate).toLocal
  }

  return (
    <LineChart width={500} height={250} data={data}>
      <CartesianGrid strokeDasharray="3 3" stroke="#1E2B3C" />
      <XAxis dataKey="name" tick={{ fill: '#94A3B8' }} />
      <YAxis domain={[0, 100]} tick={{ fill: '#94A3B8' }} />
      <Tooltip />
      {depositionDate && <ReferenceLine x="Deposition" stroke="#EF4444" strokeDasharray="4 4"
      <Line type="monotone" dataKey="score" stroke="#3B82F6" strokeWidth={2} dot={{ fill: '#3B
    </LineChart>
  );
}
```

Phase 5 Gate Check ✅ (Hour 44 — Sun 2 PM)

- ☐ Behavioral Sentinel fires composure alert in live demo with real camera
- ☐ Composure corroboration upgrades inconsistency to HIGH IMPEACHMENT RISK
- ☐ Databricks table has session events visible in Delta Live Tables
- ☐ Witness profile shows score trend across 3 demo sessions
- ☐ Load test: 5 concurrent WebSocket connections (use Artillery or k6)
- ☐ All P0 flows still working after P1 additions (regression check)

---

8. PHASE 6 — DEMO PREPARATION (Hour 44–48, Sun 2–6 PM)

Goal: Win the hackathon. Every minute of this phase is invested in the demo experience, not new features.

Step 6.1 — Demo Scenario Setup

Owner: Aman (leads), all members verify

Demo Scenario A — Primary (5 minutes, live):

```
 1. Attorney (Aman) opens VERDICT on main screen
2. Shows pre-loaded case: "Chen v. Metropolitan Hospital"
3. Navigates to Dr. Emily Chen's witness profile — shows Session 1 score: 44/100
4. Clicks "Start Session 2" — witness (Dhanush on separate laptop) joins via token link
5. Interrogator asks: "Ms. Chen, what was the exact dosage you administered?"
6. Dhanush answers: "Approximately $200, in that range"
7. [DEMO MOMENT] Inconsistency Detector fires within 4 seconds:
   ▶ Alert rail: "INCONSISTENCY DETECTED — 91% confidence"
   ▶ Prior quote: "The dosage was exactly $217" (from uploaded depo PDF, page 47)
   ▶ Behavioral Sentinel: shows FEAR expression detected (pre-staged or live camera)
   ▶ Alert upgrades to HIGH IMPEACHMENT RISK
8. Session ends — Brief Generation animation plays
9. Opens Brief: score 79/100 (+35 pts from Session 1), coach voice narrates
10. Recharts Weakness Map: Financial axis is lowest (34/100)
11. "This is why attorneys choose VERDICT"
```

Demo Scenario B — Sponsor Prize Moments (woven in):

- ElevenLabs: "Listen to the coach brief in your own language" → switch coach voice locale

- Databricks: "Here's our Delta Live Tables real-time dashboard" → show Databricks workspace tab
- Claude: "Four agents, all orchestrated by Claude Sonnet 4" → show architecture slide

Pre-staged demo data (load before presentation):

# Run demo seed script (extends prisma/seed.ts for full demo) npx tsx scripts/demo/seed-demo-scenario.ts # Creates: # - Firm: "Kirkland & Ellis LLP" # - Attorney: Sarah Chen (partner) # - Case: "Chen v. Metropolitan Hospital" (Medical Malpractice) # - 1 uploaded + indexed document: chen_depo_2024.pdf (50 pages) # - Witness: Dr. Emily Chen with 3 completed sessions (scores: 44, 61, 79) # - 3 briefs with full data (weakness maps, confirmed flags)

Step 6.2 — Production Deployment

Owner: [Member 4]
Duration: 30 minutes

# 1. Final environment variable audit cat apps/backend/.env | grep -c "=" # count all vars # Verify all vars are in Railway dashboard # 2. Run database migration on production DATABASE_URL=$PRODUCTION_DATABASE_URL npx prisma migrate deploy # 3. Seed production with demo data DATABASE_URL=$PRODUCTION_DATABASE_URL npx tsx scripts/demo/seed-demo-scenario.ts # 4. Deploy backend to Railway # Railway auto-deploys on push to main — just push: git add -A && git commit -m "feat: hackathon submission - all P0 features complete" git push origin main # 5. Verify Railway build succeeds (watch build logs) # 6. Verify Vercel build succeeds # 7. Run smoke tests on production URLs # Smoke test checklist: curl https://api.verdict.law/api/v1/health # → { "status": "ok" } curl -X POST https://api.verdict.law/api/v1/auth/login \ -H "Content-Type: application/json" \ -d '{"email":"sarah.chen@demo.com","password":"Demo!Pass123"}' # → 200 + JWT # 8. Pre-load demo case in production browser session # Open verdict.law/dashboard → confirm case "Chen v. Metropolitan Hospital" visible # Click into witness Dr. Emily Chen → confirm 3 session history visible # Navigate to brief #3 → confirm score 79/100, radar chart, audio plays

---

Step 6.3 — Backup Demo Recording

Owner: All team members
Duration: 30 minutes
Goal: Record a 5-minute demo video in case of live demo failure.

# Tools: OBS Studio or QuickTime (Mac) # Recording checklist: # - Use production URL (not localhost) # - 1920×1080 minimum # - Record audio via screen capture # - Show: login → case → live session → contradiction alert → brief → radar chart # - No hesitations; rehearse 3× before recording # - Upload to Google Drive, have direct URL ready on phone # Backup URL in Devpost submission: # "Demo video: https://drive.google.com/file/d/..."

Step 6.4 — Devpost Submission

Owner: Aman
Deadline: 30 minutes before judging
Duration: 45 minutes

```
## Devpost Submission Checklist


### Required Fields

- [ ] Project name: VERDICT
- [ ] Tagline: "AI-powered deposition coaching. From 16 hours of prep to 6."
- [ ] Description: 400-word summary (see PRD §1 for narrative)
- [ ] Demo video link: [production demo URL]
- [ ] Live app URL: https://verdict.law
- [ ] GitHub repo: https://github.com/voiceflow-intelligence/verdict

### Sponsor Prize Categories to Select

- [ ] August.law AI Automation Track (primary)
- [ ] ElevenLabs Best Use of Voice AI
- [ ] Databricks Best Use of Delta Lake / Delta Live Tables
- [ ] Anthropic Best Use of Claude SDK

### Tech Stack Tags to Add

- ElevenLabs, Anthropic Claude, NVIDIA Nemotron, Databricks, Nia AI
- Next.js, Fastify, PostgreSQL, Redis, WebSocket
- MediaPipe, Framer Motion, Recharts

### Architecture Diagram

Include: 4-agent system diagram (from PRD Appendix)
Show: data flow from document upload → Nia → session → alerts → brief

### Sponsor Prize Justification (include in description)

ElevenLabs: Dual use — Interrogator voice + Coach narration. Real-time STT for witness answers
Databricks: Delta Live Tables streaming emotion vectors. Weakness Map powered by Databricks SQ
Claude: Orchestrates all 4 agents. Streaming question generation. Brief synthesis.
```

Step 6.5 — Pitch Rehearsal (×3)

Schedule: Hour 46, 47, and 30 min before judging

```
 Pitch Structure (5 minutes):
   0:00 — Hook: "What does a litigation partner do the night before a deposition?"
   0:30 — Problem: 16 hours of manual roleplay. No consistency scoring. No coaching.
   1:00 — Solution: VERDICT. 4 AI agents working together.
   1:30 — LIVE DEMO (Scenario A — 2.5 minutes)
   4:00 — Sponsor integrations: "Built on ElevenLabs, Databricks, Claude, Nia"
   4:30 — Traction: "August.law's exact use case. $14,400 addressable market at AmLaw 200."
   5:00 — Ask: "This is the future of trial preparation."

Key moments to nail:
   • The $217 inconsistency detection moment — pause, let the alert land
   • The HIGH IMPEACHMENT RISK upgrade — explain Behavioral Sentinel in one sentence
   • The radar chart — "Financial axis: 34/100. We know exactly where to coach next."
   • The coach voice narrating the brief — "This is ElevenLabs running in production"

Questions to prep for:
   Q: "Is this a lie detector?"
   A: "No. It's a consistency scorer. Same as what a skilled attorney notices — we just do it i

   Q: "What about attorney-client privilege?"
   A: "All data lives within the firm's tenant. Encrypted at rest and in transit. Optionally se

   Q: "Why wouldn't attorneys just use GPT?"
   A: "GPT can't hear the hesitation, can't cross-reference 500 pages in 4 seconds, and can't g
```

---

9. MILESTONES & TIMELINE

Milestone 1: Foundation Complete ✅

Target: Hour 4 (Fri 10 PM)

- [ ] Monorepo initialized, all 4 members building
- [ ] Fastify health check live
- [ ] PostgreSQL: all 11 tables created
- [ ] Redis connected
- [ ] S3 connected
- [ ] Login endpoint returning JWT
- [ ] Vercel auto-deploying frontend

Milestone 2: Agents Live ✅

Target: Hour 12 (Sat 6 AM)

- [ ] Interrogator Agent streaming questions via Claude
- [ ] ElevenLabs TTS delivering audio to witness browser
- [ ] ElevenLabs STT transcribing witness answers
- [ ] Objection Copilot firing within 1.5s
- [ ] Nia FRE corpus indexed and queryable
- [ ] WebSocket session room: attorney + witness connected

Milestone 3: Full Pipeline ✅

Target: Hour 24 (Sat 6 PM)

- [ ] Document upload → S3 → Nia ingestion → READY in <3 min
- [ ] Extracted facts displayed in Fact Review screen
- [ ] Inconsistency Detector: demo contradiction ($200 vs $217) detected
- [ ] Nemotron confidence score ≥0.75 triggering live alert
- [ ] Full session flow functional end-to-end
- [ ] All frontend screens navigable (design may still be rough)

Milestone 4: MVP Demo-Ready ✅

Target: Hour 36 (Sun 6 AM)

- [ ] P0.5 Coaching Brief generated with Claude narrative
- [ ] ElevenLabs Coach voice narrating flagged moments
- [ ] Recharts radar chart rendering in brief viewer
- [ ] Brief PDF downloadable
- [ ] All animations and transitions smooth
- [ ] No blocking console errors in demo flow

Milestone 5: Submission-Ready ✅

Target: Hour 48 (Sun 6 PM)

- [ ] All P0 features demoable in production (verdict.law)
- [ ] P1.4 Behavioral Sentinel: composure alerts fire on camera
- [ ] Databricks Delta Live Tables: session events streaming
- [ ] Multi-session witness profile: score trend chart working
- [ ] Devpost submission complete
- [ ] Backup demo video recorded and uploaded
- [ ] Pitch rehearsed ×3 with all team members

---

10. RISK MITIGATION

Technical Risks

Risk	Probability	Impact	Mitigation
ElevenLabs TTS latency >2s	Medium	High — core UX	Use `eleven_turbo_v2_5` model. Test latency in Phase 2. Fallback: text-only questions
Nemotron API rate limit hit	Medium	Medium — affects Inconsistency Detector	Cache embeddings in Redis. Use batch calls. Fallback: Claude-only scoring at threshold 0.85
Nia ingestion >5 min	Low	Medium — blocks demo	Pre-index demo document before hackathon. Cap at 50-page test doc
MediaPipe WASM not loading	Low	Low — P1.4 only	Host face_landmarker.task in /public/models. CDN fallback. Feature can be skipped
Databricks warehouse cold start	Medium	Low — demo only	Warm warehouse before demo. Keep alternative pre-queried static data
WebSocket drops on demo laptop WiFi	Medium	High — live demo failure	Test on hackathon venue WiFi. Have hotspot backup. Record demo video
Prisma migration fails in production	Low	Critical	Test migration on staging DB first. Pre-deploy snapshot. Rollback: `prisma migrate resolve --rolled-back`
JWT secret mismatch between frontend/backend	Low	High	Single source of truth: Railway env vars. Verify before demo
Claude API rate limit (heavy agent use)	Low	Medium	Queue requests. Distribute across session. Use streaming to reduce apparent latency

Timeline Risks

Risk	Impact	Mitigation
Phase 2 agents take 2× estimated time	All subsequent phases slip	Cut Behavioral Sentinel entirely if Phase 2 is late. Focus on P0 only
Frontend screens too rough for judges	Perception of incomplete product	Dhanush focuses on live session screen first — that's what judges see
Brief generation >3 min in demo	Demo dead air is fatal	Pre-generate brief in demo seed data. Just "reload" the brief page
Team member gets sick/unavailable	25% capacity loss	Each member writes a 1-paragraph handoff doc for their module after Phase 2
Demo laptop battery dies	Demo fails	Both demo laptops fully charged. Chargers on table. App is deployed — use any laptop

Scope Risks

Temptation	Decision
"Let's add voice-to-text real-time transcript display"	❌ Cut if not already built. Shows aren't scored on features not in PRD
"Let's integrate iManage Matter Management"	❌ P2. Not started
"Let's add multi-jurisdiction FRE rules"	❌ P2. Nia FRE corpus is sufficient
"Let's make a native iOS app"	❌ Web only. No time
"Let's add LangChain"	❌ Claude SDK is sufficient. Dependency risk not worth it
"Let's polish the marketing landing page"	⚠️ Only if all P0 features are shipped with time remaining

---

11. SUCCESS CRITERIA

P0 — Minimum Viable Demo (must all be true to submit)

#	Criterion	Test
1	Interrogator Agent asks legally appropriate questions via ElevenLabs voice	Manual: Start session, hear question audio within 2s
2	Objection Copilot fires within 1.5s for a leading question	Manual: Ask "Isn't it true..." → alert in <1500ms
3	Inconsistency Detector detects $200 vs $217 contradiction at ≥0.75 confidence	Manual: Demo script → see INCONSISTENCY alert
4	Document upload → Nia ingestion → READY in <3 min for 50-page PDF	Automated timing test
5	Coaching Brief generated with Claude narrative + ElevenLabs coach voice	Manual: End session → wait → hear coach voice in brief
6	Full session flow navigable without errors: login → case → session → brief	Manual E2E walkthrough
7	Attorney alert rail shows objection and inconsistency alerts in real-time	Manual: Live session with witness browser open
8	Brief deployed at production URL (verdict.law)	`curl https://verdict.law` → 200

Performance Targets (from PRD §8.1)

Metric	Target	Test Method
Objection Copilot response	≤ 1,500ms	`processingMs` field in response
Inconsistency Detector	≤ 4,000ms	`processingMs` field in response

response		
ElevenLabs TTS to audio start	≤ 2,000ms	`audio_latency_ms` in session_events
Document ingestion (50 pages)	≤ 3 minutes	Ingestion status polling
Concurrent WebSocket sessions	≥ 5 simultaneous	Artillery load test
Brief generation	≤ 3 minutes	Timer from session end to brief.generation_status = COMPLETE

Sponsor Prize Criteria Met

Sponsor	What's Required	Our Implementation
August.law (primary)	End-to-end AI legal workflow	Full 5-agent deposition prep pipeline
ElevenLabs	Voice AI as core feature	Interrogator TTS + STT + Coach narration
Databricks	Delta Lake or Delta Live Tables used meaningfully	Session events + emotion vectors streamed to Delta
Anthropic	Claude SDK with meaningful orchestration	4 agents all using Claude; streaming; tool use

---

12. POST-MVP ROADMAP

Immediate Post-Hackathon (Feb 23 – Mar 7)

1. **Production hardening** — Add comprehensive error boundaries, retry logic, connection pooling
2. **August.law partnership call** — Discuss integration requirements, compliance review
3. **AmLaw 200 pilot outreach** — 5 firms for paid pilot at $2,500/month
4. **SAML SSO** — Full Okta/AzureAD integration for enterprise pilot firms
5. **Puppeteer PDF export** — Full brief PDF with branded letterhead
6. **WCAG 2.1 AA audit** — Accessibility compliance for enterprise legal tools

P1 Feature Completion (Q2 2026)

Feature	PRD Reference	Priority
P1.1 Cross-session witness profile — full UI	BACKEND_STRUCTURE §witnesses	High
P1.2 Weakness Map — Databricks SQL-powered	PRD §P1.2	High
P1.3 Argument Strength Scoring — Nemotron per answer	PRD §P1.3	Medium
P1.4 Behavioral Sentinel — firm-wide rollout	PRD §P1.4	Medium
Firm Admin Panel — full user management	APP_FLOW §2.7	High

Multi-language Interrogator voices	ElevenLabs locale API	Medium

P2 Feature Roadmap (Q3 2026)

Feature	Description
P2.1 Whisper-in-Ear Mode	Real-time coaching cues to attorney earpiece via ElevenLabs spatial audio
P2.2 Multi-Jurisdiction FRE	Nia-indexed state evidence rules (CA, NY, TX, FL)
P2.3 Case Outcome Analytics	MLflow-powered prediction: deposition score → trial outcome correlation
P2.4 Telephony Integration	Twilio-based remote deposition support
P2.5 Matter Management	iManage + Clio integration for case file sync

User Feedback Loop

```
 Week 1 post-launch:
  → 5 pilot attorney interviews (2 partners, 2 associates, 1 paralegal)
  → NPS survey after first session
  → Session recording review (with consent): where do attorneys click most?

Month 1:
  → Track: ingestion success rate, brief generation time, score improvement across 3 sessions
  → Track: which alert types get confirmed vs rejected (calibrate Nemotron threshold)
  → Track: Behavioral Sentinel opt-in rate (measures comfort with feature)
```

Revenue Model Post-Hackathon

Tier	Price	Seats	Features
Starter	$500/month	5	P0 features, email auth
Professional	$2,500/month	25	P0+P1, SSO, Databricks
Enterprise	$10,000/month	Unlimited	Full stack, self-hosted option, SLA

Target ARR at 12 months: 20 firms × $2,500/month × 12 = $600,000
