Submission: Git repository (public or private) or ZIP archive with setup instructions

We will know if you have completed the assessment when we receive your submission. Please ensure that your code is well-structured, documented, and adheres to best practices. We will review your code for correctness, architecture, API design, DevOps practices, GenAI logic, frontend usability, testing coverage, and documentation quality.

We expect you to complete this assessment independently—without external assistance or copied code—and to ensure your submission fully meets the requirements below; responsibly used AI tools are welcome, but any non-original code or deviations will be evident during our review.

---

1. Overview

Your task is to design, implement, and containerize a complete solution—including authentication, a GenAI contract analysis API, core backend services, and a minimal frontend. Bonus points for asynchronous design, thorough testing, and comprehensive documentation.

---

2. Deliverables

- **Source Code & Configuration**
- Git repository (or ZIP) containing all code
- `README.md` with clear setup and run instructions
- `.env.example` listing all environment variables
- Database seeding or test-data scripts, if applicable
- **Dockerized Environment**
- `docker-compose.yml` orchestrating:
- MongoDB
- Redis
- Backend application (and frontend, if containerized)
- **API Endpoints**

1. **Authentication** (no token required)

- `POST /auth/register` — Create a new user
- `POST /auth/login` — Issue JWT/session token

2. **GenAI Contract Analysis** (JWT required)

- `POST /genai/analyze-contract`
- Accept a PDF upload
- Extract and classify clauses
- Return structured JSON of clause types & contents
- `POST /genai/evaluate-contract`
- Assess contract health
- Return `{ approved: boolean, reasoning: string }`

3. **Backend Services** (JWT required)

- `GET /logs` — Paginated retrieval; filters: user, endpoint, date, status
- `GET /metrics` — System metrics (request count, avg. latency)
- `GET /healthz` — Liveness probe
- `GET /readyz` — Readiness probe
- **Clients & Contracts**
- `POST /clients` — Create client record
- `GET /clients/:id/contracts` — List contracts for a client
- Full CRUD endpoints for `/contracts`
- `POST /contracts/:id/init-genai` — Trigger analysis pipeline; store results
- **Frontend**
- **Next.js** (preferred) *or* **Streamlit**
- Features:
- User registration & login
- PDF upload for contract analysis
- Display categorized clause results
- View evaluated contracts
- Admin panel: metrics & logs dashboard

---

3. Technical Requirements

- **Containerization & DevOps**
- Use Docker best practices (small images, multi-stage builds)
- Environment variables for all secrets/configuration
- **Authentication**
- Stateless JWT or session-based auth

- Secure password storage
- **GenAI Integration**
- OpenAI, Hugging Face, LangChain, or similar
- Modular design: separate service or library
- **Asynchronous Processing** *(Bonus)*
- FastAPI or equivalent
- Background tasks for PDF parsing & AI calls
- **Testing**
- Unit tests for core modules (auth, GenAI, logging)
- Load testing scripts (e.g., Locust, k6) *(Bonus)*
- **Documentation**
- OpenAPI/Swagger spec or equivalent
- Clear code comments and architecture overview

---

4. Bonus Opportunities

- Async I/O framework (e.g., FastAPI, asyncio)
- LangChain or reusable AI-tooling integration
- Containerized frontend in `docker-compose.yml`
- Comprehensive unit & integration tests
- Load testing suite (Locust, k6, Artillery)
- Auto-generated API docs (OpenAPI, Redoc)

---

5. Submission Checklist

- [ ] Git repo URL or ZIP file delivered
- [ ] `README.md` with setup & run instructions
- [ ] `.env.example` present
- [ ] `docker-compose.yml` with MongoDB, Redis, backend (±frontend)
- [ ] Database seed or sample data scripts (if needed)
- [ ] Unit tests & load test scripts (if included)
- [ ] API documentation (OpenAPI schema or README section)

---

6. Evaluation Criteria

Category	What We’re Looking For
Correctness	All endpoints work as specified; GenAI integration functions
Clean Architecture	Modular, readable code; clear separation of concerns
API Design	Consistent, RESTful routes; meaningful request/response schemas
DevOps & Docker	Reliable, minimal Docker images; sensible environment setup
GenAI Logic	Accurate clause extraction; clear evaluation reasoning
Frontend Usability	Intuitive UI; smooth user flows for upload & results
Testing & QA	Coverage of critical paths; reliable load tests
Documentation	Easy onboarding; comprehensive API & architecture docs