

Implementation plan :

The Hackathon Pivot: Speed & Stability over Academic Theory

Team, before we start, we need to address a critical pivot from the V2 research architecture. The paper recommends a "MEGA-RAG" system utilizing a local Biomedical Knowledge Graph and cross-encoder reranking models.

We are dropping the Knowledge Graph and cross-encoders. Attempting to run those alongside a bioinformatics parser and a local LLM will instantly max out our 6GB mobile GPUs and crash our app during the live demo.

The Fix: We are building a hyper-optimized, decoupled cloud architecture. We will use **Pinecone** for ultra-fast semantic vector retrieval and offload the generation and hallucination-prevention loops entirely to the **Gemini API** using rigid system prompts. This guarantees sub-second response times, zero local GPU bottlenecks, and an enterprise-grade result.

Here is your end-to-end implementation blueprint.

Phase 1: The Frontend (Next.js & Firebase)

Assigned to: Teammate 1 Deployment: Vercel

Your goal is to build a reactive, highly secure portal that handles user authentication, file ingestion, and renders the complex genomic data without triggering "alert fatigue."

- **Authentication & Storage:** Initialize a Next.js (App Router) project. Set up Firebase Authentication (Google Sign-in) and Firebase Cloud Storage.
- **The Uploader & Pre-validation (Crucial):** Build a drag-and-drop file uploader. Do not use the WebAssembly approach mentioned in the paper; it's a time-sink for a hackathon. Instead, use the native JavaScript `FileReader` and `File.slice()` API to read the first few kilobytes of the file.
 - *Validation 1:* Enforce the strict 5MB limit.
 - *Validation 2:* Check that the first line contains `##fileformat=VCFv4.2` and the header contains `#CHROM POS ID REF ALT`.
 - If valid, upload the VCF to Firebase Storage and send the secure download URL to our Python backend.
- **Visual Triage Dashboard:** When the JSON response returns from the backend, build a "Traffic Light" UI.
 - **Green:** Safe (Hide details).
 - **Yellow:** Adjust Dosage.
 - **Red:** Toxic / Ineffective (Forcefully command attention).
 - Use **Progressive Disclosure:** Show only the drug name, the color-coded risk label, and the CPIC recommendation on the main card. Hide the dense genetic diplotypes and the LLM explanation inside an expandable UI accordion.

Phase 2: Bioinformatics Backend & Deterministic Rules Engine

Assigned to: Teammate 2 (Frosty Coder - Lead Developer) **Deployment:** Docker container on Render (Free Tier)

This is the core engine. You will build a Python FastAPI service that executes the exact mathematical determinism required by CPIC guidelines.

- **VCF Parsing with `cyvcf2`:** Your FastAPI endpoint will receive the Firebase URL, download the VCF, and parse it using `cyvcf2`.
 - Extract the `GENE` and `RS` (rsID) from the `INFO` tags.
 - Extract phasing data from the `FORMAT` and `GT` (genotype) columns (e.g., differentiating `0|1` phased vs `0/1` unphased).
- **The CYP2D6 Structural Variant Trap:** Write a specific function to scan the `INFO` column for Structural Variant (SV) or Copy Number Variation (CNV) markers when evaluating CYP2D6. If none are found, you must dynamically lower the JSON `confidence_score` and flag a warning that whole-gene deletions might be missed.
- **The Rule Engine:** Map the genetic variants to CPIC "Star Alleles" and calculate the Activity Score (AS).
 - *Example Logic:* If `cyvcf2` detects a DPYD `*1/*2A` diplotype, calculate the AS as 1.0 (Normal function + No function). Map AS 1.0 to the "Intermediate Metabolizer" phenotype. Output "Adjust Dosage" (Reduce 5-FU by 25-50%).
 - *The Warfarin Exception:* Hardcode the IWPC algorithm to evaluate CYP2C9 alongside VKORC1 and CYP4F2 simultaneously, rather than strictly relying on CYP2C9 alone.

Phase 3: The Intelligence Layer (Pinecone + Gemini API)

Assigned to: Teammate 3 **Deployment:** API Driven (No local hosting)

Your job is to build the RAG pipeline that explains the deterministic math to the doctors without hallucinating dosages.

- **Vector Database Setup:** Before the demo, take the official CPIC PDF guidelines for our 6 target drugs, chunk the text into paragraphs, embed them, and upload them to a free Pinecone index.
- **Context Retrieval:** When the Python backend determines a phenotype (e.g., "SLCO1B1 Poor Function and Simvastatin"), your script queries Pinecone to retrieve the exact paragraph explaining the OATP1B1 transporter failure.
- **The Gemini API Guardrails:** Send the retrieved Pinecone text and the patient data to the Gemini API using this strict system prompt template:
 - *"You are an expert clinical pharmacogenomics consultant. You must ONLY use the provided retrieved literature. Explicitly cite the patient's specific Diplotype. Under NO circumstances are you to recommend a specific"*

numerical dosage... If the retrieved literature does not explain the biological mechanism, state 'Mechanism unclear in available guidelines'."

- **Schema Assembly:** Finally, the Python backend takes the Gemini-generated explanation string and injects it into the "llm_generated_explanation" block of our strict, HL7 FHIR-compliant JSON schema. Send this JSON back to the Next.js frontend.