System flow :

Here is the exact, step-by-step execution flow of what happens under the hood the moment a user drops a VCF file into your Next.js frontend:

Step 1: Client-Side Pre-Validation (Next.js / Browser)

- **The Action:** The user drags a VCF file and types a drug name (e.g., "SIMVASTATIN, CODEINE") into the Next.js UI.
- **The Execution:** Before the file ever leaves the user's computer, a JavaScript `FileReader` function slices the first few kilobytes of the file.
- **The Check:** It verifies the file size is strictly `< 5MB`. It reads the first line to ensure `##fileformat=VCFv4.2` is present, and checks that the header row contains `#CHROM POS ID REF ALT`.
- **The Result:** If it fails, the UI throws a fast, user-friendly error without wasting server bandwidth. If it passes, the file is temporarily uploaded to your Firebase Storage bucket, and a secure download URL is generated.

Step 2: API Handshake (Next.js to FastAPI)

- **The Action:** Next.js makes a POST request to your Python FastAPI backend hosted on Render.
- **The Payload:** It sends the Firebase VCF download URL and the array of requested drugs `["SIMVASTATIN", "CODEINE"]`.

Step 3: Bioinformatics Parsing (FastAPI + `cyvcf2`)

- **The Action:** FastAPI downloads the VCF into memory and feeds it to the `cyvcf2` library.
- **The Execution:** The parser ignores the millions of irrelevant genetic lines and specifically filters for coordinates matching our 6 target genes (CYP2D6, CYP2C19, CYP2C9, SLCO1B1, TPMT, DPYD).
- **Data Extraction:** It pulls the `rsID` (e.g., rs4149056) and the `GENE` symbol from the `INFO` column. It checks the `GT` (Genotype) column to see if the mutations are phased (`0|1`) or unphased (`0/1`).

Step 4: Deterministic Rules Engine (The Medical Logic)

- **The Action:** The backend maps the extracted mutations against hardcoded CPIC tables.
- **The Execution:** 1. **Allele Matching:** It sees the `rs4149056` mutation and maps it to the SLCO1B1 `*5` Star Allele. 2. **Diplotype & Phenotype:** It combines the maternal and paternal alleles into a Diplotype (e.g., `*5/*5`) and calculates the Phenotype ("Poor Function"). 3. **Risk Assessment:** It checks the requested drug (Simvastatin) against the Phenotype ("Poor Function"). The CPIC rules engine immediately outputs: `Risk: Toxic`, `Severity: Critical`, and pulls the strict clinical recommendation ("Prescribe alternative statin...").

Step 5: MEGA-RAG Explanation Generation (Pinecone + Gemini API)

- **The Action:** The backend asks the AI to explain the deterministic result.
- **The Execution:** 1. **Vector Search:** FastAPI sends a query ("SLCO1B1 Poor Function Simvastatin") to Pinecone. 2. **Retrieval:** Pinecone instantly returns the official CPIC PDF paragraph explaining the biological mechanism (OATP1B1 transporter failure). 3. **Generation:** FastAPI sends this specific paragraph, the patient's `*5/*5` diplotype, and a strict guardrail prompt to the **Gemini API**. Gemini returns a concise, 3-sentence biological explanation.

Step 6: Strict JSON Assembly & Cleanup (FastAPI)

- **The Action:** FastAPI structures the final response.
- **The Execution:** It maps all the generated data into the **EXACT** JSON schema requested in the problem statement (patient_id, risk_label, confidence_score, rsid, llm_generated_explanation, quality_metrics). It then sends a command to Firebase to delete the temporary VCF file to keep your storage clean.

Step 7: Visual Triage UI Rendering (Next.js)

- **The Action:** Next.js receives the structured JSON from FastAPI.
- **The Execution:** The dashboard updates. Because the risk label is "Toxic," it renders a Red alert card for Simvastatin. The CPIC dosing recommendation is displayed boldly. The dense genetic data and the Gemini-generated biological mechanism are hidden neatly inside an expandable accordion. The user is presented with a button to "Download JSON" or "Copy to Clipboard."

This flow ensures your app hits every single evaluation criterion: It handles the exact files, uses deterministic CPIC logic for accuracy, integrates LLMs safely for explainability, outputs the exact JSON, and provides a polished, clinical UX.