NutriScopeAI V2 Workflow Chart

The workflow for NutriScopeAI V2 can be visualized as a sequential process with distinct modules handling specific tasks. The diagram below illustrates the flow from user interaction to the final report display.

---

1. Frontend Interaction

The process begins with the **frontend**, specifically `frontend/pages/upload.py`, where a user like Alice uploads her lab report PDF. This file, along with metadata (name, age, sex), is submitted to the backend through the `/api/analyze` endpoint. The `frontend/pages/utils.py` file ensures that API requests are properly authenticated. Once the analysis is complete, `frontend/pages/report_details.py` fetches and displays the final report, including a summary table and meal plan cards.

---

2. Backend API Orchestration

The **backend API**, managed by `backend/app/api/routes.py`, acts as the central hub. It receives the uploaded file from the frontend and orchestrates the entire analysis pipeline. This endpoint handles the initial request, triggering the series of subsequent steps, including ingestion, normalization, and AI summarization.

---

3. Ingestion & Parsing

This step is critical for reading the user's data. The `backend/app/ingest/parser.py` module extracts the essential information—test names, values, and units—from the PDF. If the PDF is a low-quality scan, the `backend/app/ingest/extract.py` provides an **OCR (Optical Character Recognition) fallback**, ensuring data can be extracted from image-based reports.

---

4. Normalization

After extraction, the raw data is cleaned and standardized. The `backend/app/normalize/normalized_values.py` maps extracted test names to a canonical form (e.g., "Hb" to "Hemoglobin"). Simultaneously, `backend/app/normalize/unit_normalization.py` standardizes all units, converting them to a consistent format (e.g., mg/dL to mmol/L), while `backend/app/normalize/aliases.py` handles various synonyms for test names.

---

5. Knowledge Base & Evaluation

With the data normalized, it's cross-referenced against a **knowledge base (KB)**. The `backend/app/kb/loader.py` module loads standard reference ranges for each lab test. The backend/app/core/evaluator.py then flags each value as **Low**, **Normal**, or **High** by comparing the user's result to these reference ranges. This evaluation provides the foundational context for the AI summarization step.

---

6. AI Summarization & Meal Planning

This is the core of the NutriScopeAI. The backend/app/summarize/llm.py module uses the Groq LLM API to generate a personalized report. Using the flagged test results and the knowledge base context, it produces:

- An explanation of **why a test is important**.
- Potential **reasons** for high or low values.
- Associated **risks** if the condition is untreated.
- A personalized **meal plan** with food suggestions to address the flagged issues (e.g., iron-rich foods for low hemoglobin).

---

7. Storage & Persistence

Once the AI-generated report is complete, the backend/app/storage/reports_store.py module saves the full report—including the original data and the AI-generated summaries—to a **JSON file** (backend/app/storage/reports.json). This ensures the data is durable and can be retrieved later by the user or the system.

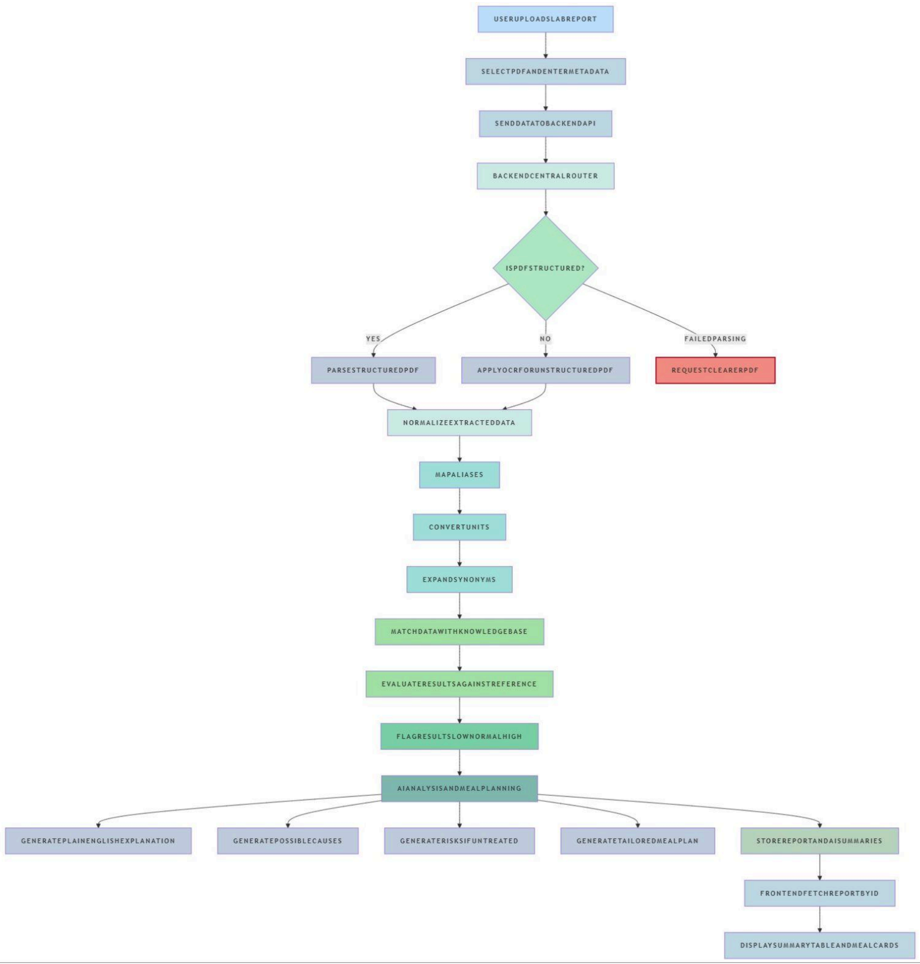
---

8. Frontend Display

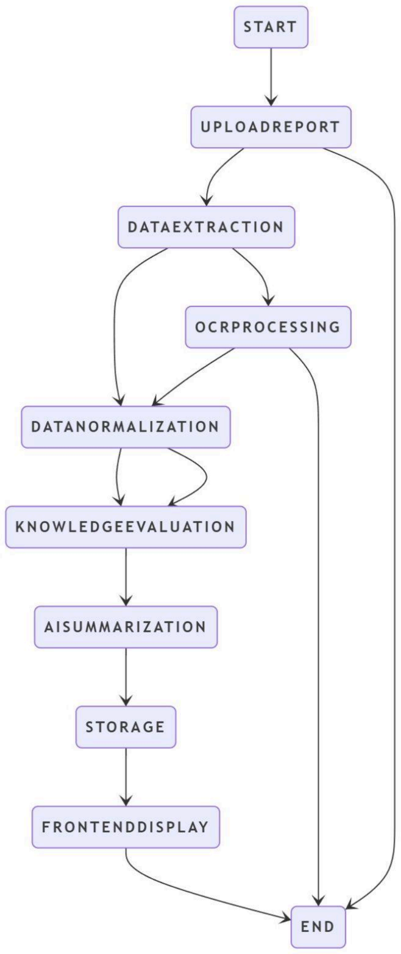
Finally, the completed report is ready to be displayed. The frontend fetches the stored report via the /api/report/{rid} endpoint. The frontend/pages/report_details.py file then formats and presents the information in a user-friendly manner, showing a structured summary table and a clear, easy-to-read meal plan.

This shows the flow from **Frontend → API → Parsing → Normalization → KB → Core Evaluation → AI Summarization → Storage → Back to Frontend** with test reports feeding into parsing.

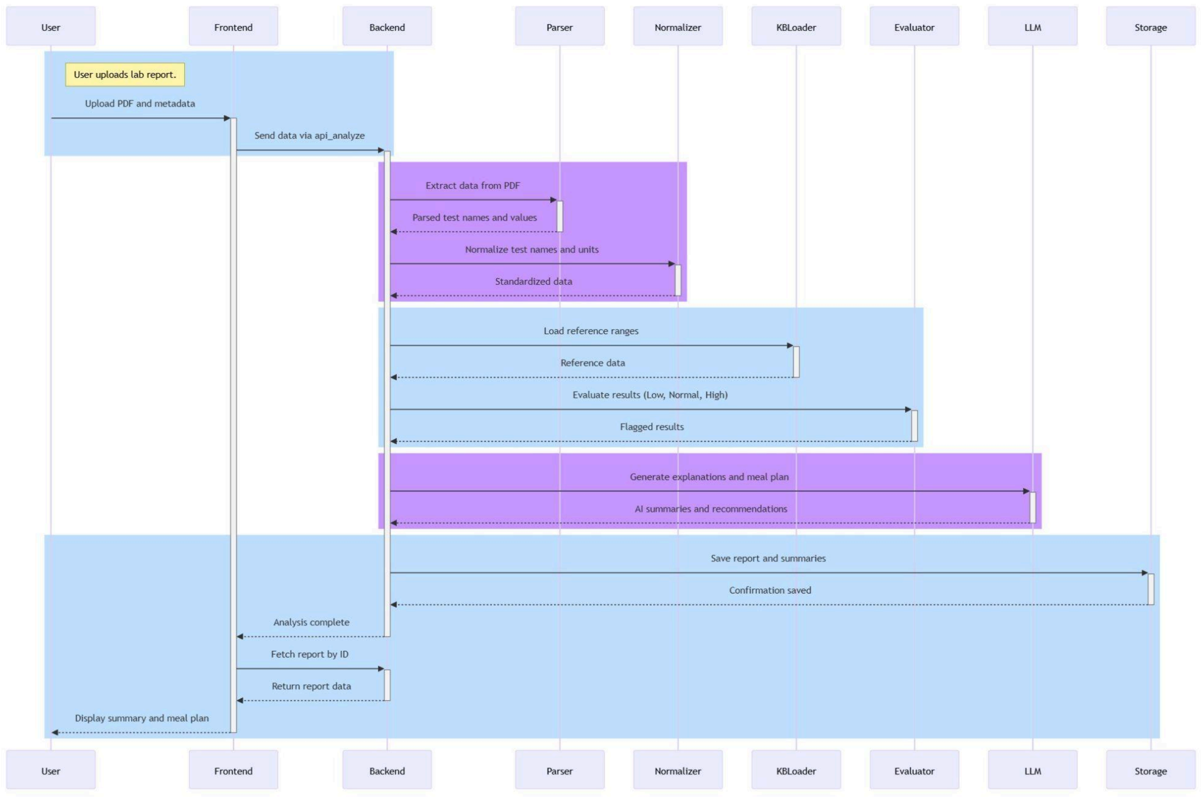
9. Flow Chart


USERUPLOADSLABREPORT
SELECTPDFANDENTERMETADATA
SENDDATATOBACKENDAPI
BACKENDCENTRALROUTER
ISPDFSTRUCTURED?
YES
NO
FAILEDPARSING
PARSESTRUCTUREDPDF
APPLYOCRFORUNSTRUCTUREDPDF
REQUESTCLEARERPDF
NORMALIZEEXTRACTEDDATA
MAPALIASES
CONVERTUNITS
EXPANDSYNONYMS
MATCHDATAWITHKNOWLEDGEBASE
EVALUATERESULTSAGAINSTREFERENCE
FLAGRESULTSLOWNORMALHIGH
AIANALYSISANDMEALPLANNING
GENERATEPLAINENGLISHEXPLANATION
GENERATEPOSSIBLECAUSES
GENERATERISKSIFUNTREATED
GENERATETAILOREDMEALPLAN
STOREREPORTANDAISUMMARIES
FRONTENDFETCHREPORTBYID
DISPLAYSUMMARYTABLEANDMEALCARDS

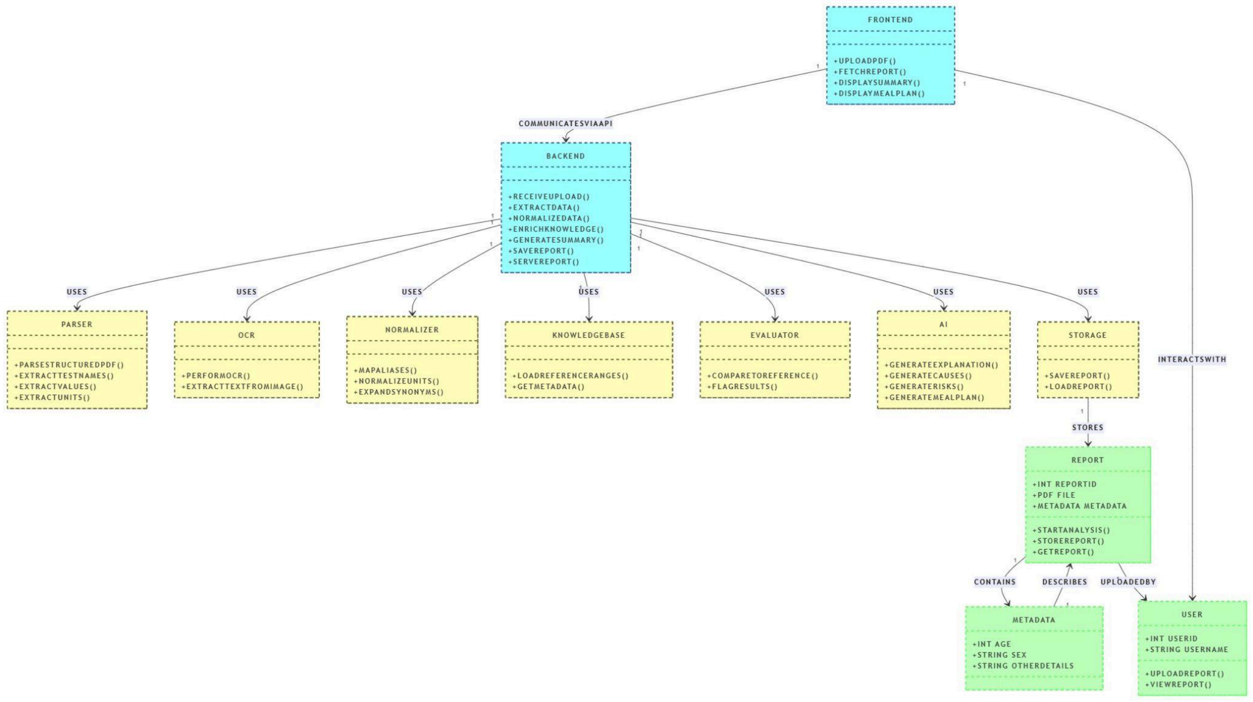
10. State Diagram



11. Sequence Diagram



12. Class Diagram


FRONTEND
+UPLOADPDF()
+FETCHREPORT()
+DISPLAYSUMMARY()
+DISPLAYMEALPLAN()
COMMUNICATESVIAAPI
INTERACTSWITH
BACKEND
+RECEIVEUPLOAD()
+EXTRACTDATA()
+NORMALIZEDATA()
+ENRICHKNOWLEDGE()
+GENERATESUMMARY()
+SAVEREPORT()
+SERVEREPORT()
USES
PARSER
+PARSESTRUCTUREDPDF()
+EXTRACTTESTNAMES()
+EXTRACTVALUES()
+EXTRACTUNITS()
OCR
+PERFORMOCR()
+EXTRACTTEXTFROMIMAGE()
NORMALIZER
+MAPALIASES()
+NORMALIZEUNITS()
+EXPANDSYNONYMS()
KNOWLEDGEBASE
+LOADREFERENCERANGES()
+GETMETADATA()
EVALUATOR
+COMPARETOREFERENCE()
+FLAGRESULTS()
AI
+GENERATEEXPLANATION()
+GENERATECAUSES()
+GENERATERISKS()
+GENERATEMEALPLAN()
STORAGE
+SAVEREPORT()
+LOADREPORT()
STORES
REPORT
+INT REPORTID
+PDF FILE
+METADATA METADATA
+STARTANALYSIS()
+STOREREPORT()
+GETREPORT()
CONTAINS
DESCRIBES
UPLOADEDBY
METADATA
+INT AGE
+STRING SEX
+STRING OTHERDETAILS
USER
+INT USERID
+STRING USERNAME
+UPLOADREPORT()
+VIEWREPORT()
1