NutriScope AI

An AI-powered diet & health companion that transforms lab reports into clear summaries and personalized meal plans.

🚀 Features

- **Authentication** — JWT-based sign-in/sign-up.
- **Secure Uploads** — Upload lab reports in PDF or image format.
- **OCR** — Extracts text from digital or scanned reports.
- **Lab Parsing** — Automatically parses key values (e.g., Vitamin D3, Glucose).
- **Reference Comparison** — Flags abnormal results using JSON-driven ranges.
- **RAG (Retrieval-Augmented Generation)** — Retrieves relevant medical/diet context for more reliable AI responses.
- **AI-Generated Insights** — Groq LLM produces concise summaries and structured meal plans.
- **Dashboard** — View, paginate, and delete past reports.
- **Frontend** — Modern React + Material-UI interface for a smooth UX.

🛠️ Tech Stack

Backend

- Python 3.11
- Flask + Flask-JWT-Extended + Flask-CORS

- SQLAlchemy (SQLite for persistence)
- Tesseract OCR + pdf2image (report parsing)
- FAISS + SentenceTransformers (RAG)
- Groq API (LLM inference)

Frontend

- React + TypeScript + Vite
- Material-UI (MUI) components
- React Router
- Custom API integration with JWT token handling

Infrastructure

- Local dev server (`localhost:8000` backend, `localhost:5173` frontend)
- dotenv for environment variables

📂 Project Structure

```
backend/
├── app.py            # Flask app entrypoint
├── auth.py           # Auth routes (signup/signin)
├── models.py         # SQLAlchemy models (User, Report)
├── database.py       # DB session/engine setup
├── refs.py           # JSON-driven reference ranges + annotate()
├── parsers.py        # Extract lab values from OCR text
├── ocr.py            # OCR helper for PDFs/images
├── rag.py            # RAG retrieval pipeline (FAISS + embeddings)
├── groq_client.py    # Groq API integration
└── uploads/          # Saved uploaded files
```

```
frontend/
├── src/
│   ├── api.tsx        # Typed API helper with JWT support
│   ├── store.tsx      # Auth state (token, user)
│   ├── pages/         # SignIn, SignUp, Upload, Reports, ReportDetail
│   ├── components/    # Navbar, Footer, FileDrop, MealPlan, UI elements
│   └── App.tsx        # Routing
└── public/
```

⚙️ Setup

Backend

```
cd backend
python -m venv .venv
.venv\Scripts\activate   # (Windows)
pip install -r requirements.txt

# Set environment variables in .env
SECRET_KEY=your_secret
JWT_SECRET_KEY=your_jwt_secret
GROQ_API_KEY=your_groq_api_key

# Run server
python app.py
```

Frontend

```
cd frontend
npm install
npm run dev
```

🔐 Authentication Flow

1. User **signs up** (`/api/auth/signup`) with name, email, password.
2. User **signs in** (`/api/auth/signin`) → receives a JWT token.
3. Token is stored in `localStorage` and attached to every request via `Authorization: Bearer <token>`.

📄 API Endpoints

Auth

- `POST /api/auth/signup` — Register user.
- `POST /api/auth/signin` — Authenticate user and return JWT.

Reports

- `POST /api/upload-report` — Upload PDF/image → OCR → parse → RAG → AI summary + meal plan.
- `GET /api/report/<id>` — Fetch one report with parsed values, flags, summary, meal plan.
- `GET /api/reports?page=N&page_size=M` — List reports (paginated).
- `DELETE /api/report/<id>` — Delete a report.

Utilities

- `GET /api/health` — Check DB + RAG KB status.
- `GET /api/search?q=term` — Query KB manually (dev/debug).

🤖 AI Logic

1. **OCR** extracts text from reports.
2. **Parser** converts text into structured values.
3. **Annotator** flags abnormalities using reference JSON.
4. **RAG Retriever** gathers relevant passages (medical/diet knowledge).
5. **Groq LLM** generates:
    - **Summary** (abnormalities + significance).
    - **Meal Plan** (structured JSON, 3 meals only for abnormal values).

📊 Example Workflow

1. Upload `report3.pdf`.
2. OCR → values: `{"Vitamin D3": 7.3}`.
3. Annotator → `{"Vitamin D3": "low"}`.
4. RAG → context on Vitamin D deficiency.
5. AI generates:
    - Summary: "Vitamin D3 is low, may affect bone health..."
    - Meal Plan: Salmon, fortified dairy, mushrooms.

👤 Author

Venkata Sai Krishna Aditya Vatturi

- Full-stack developer & AI systems engineer.
- Built the project solo: backend (Flask + AI pipeline), frontend (React + MUI), and integration with Groq API.