**🧠 Supplement Recommendation Engine – Backend for Personalized Health**

This project is a comprehensive **Python-based supplement recommendation system**. It's built for a **personalized supplement subscription service**, capable of adapting to user input over time, integrating wearable data, and assigning safe, effective supplement protocols.

**📁 Project Structure**

app/

├── api.py # Main entry for API calls

├── cluster\_engine.py # ML clustering logic

├── cluster\_logger.py # Tracks cluster assignment changes

├── cluster\_runner.py # Executes clustering for all users

├── data\_model.py # Defines the data schema for users

├── data\_storage.py # Handles persistent user data (JSON-based)

├── dosage\_calculator.py # Assigns dosages using scientific guidelines

├── drug\_interaction\_checker.py # Checks for supplement-drug conflicts

├── explanation\_utils.py # Generates human-readable reasoning

├── feedback\_loop.py # Adapts recommendations over time

├── generate\_mock\_users.py # Test data generation

├── protocol\_log\_utils.py # Logs protocol changes

├── safety\_checks.py # NIH/EFSA safety validations

├── supplement\_engine.py # Main recommendation logic

├── supplement\_utils.py # Helper functions for supplements

├── symptom\_scorer.py # Converts symptoms to nutrient scores

├── unit\_converter.py # Handles unit standardization

├── user\_update\_pipeline.py # Processes feedback to update users

├── wearable\_middleware.py # Interfaces with wearable APIs

├── \*.json # Supplement DB, user data, logs, protocols

└── tests/ # Extensive test coverage

**🚀 Features**

**✅ 1. Input Types Supported**

* **Health Quiz**: diet, symptoms, history, lifestyle, gender, age.
* **Wearables**: Apple Health, Oura, WHOOP, Garmin (via wearable\_middleware.py)
* **Feedback Check-ins**: “How do you feel today?” (via feedback\_loop.py)

**🧪 2. Personalized Supplement Recommendations**

**Core Logic:**

* symptom\_scorer.py: turns user answers into **nutrient need scores**.
* supplement\_engine.py: uses those scores and user traits to:
  + Choose **which supplements**
  + Compute **personalized dosage**
  + Apply **scientific & safety constraints**
  + Link **reasoning and trigger inputs**

**Dosage Estimation:**

* dosage\_calculator.py uses:
  + RDA/UL from NIH/EFSA
  + Dynamic scaling based on goals/health status
  + Clustering fallback (see below)

**🔄 3. Adaptive Feedback Loops**

* feedback\_loop.py: integrates ongoing input (e.g. “better sleep”) to **adjust dosage or switch supplements**
* user\_update\_pipeline.py: handles periodic updates (weekly, monthly)

**📊 4. User Clustering & Machine Learning**

* cluster\_engine.py: converts user vectors into KMeans-like clusters
* cluster\_runner.py: runs clustering pipeline
* cluster\_protocols.json: stores typical protocols per cluster
* cluster\_logger.py & protocol\_log\_utils.py: track evolution and overrides

**🛡️ 5. Safety & Interactions**

* drug\_interaction\_checker.py: cross-checks user medications with supplement interactions (drug\_supp\_interactions.json)
* safety\_checks.py: ensures dosages never exceed known tolerable limits (UL)
* Contraindications (e.g., male with normal ferritin skips Iron)

**🧬 6. Output Format**

Each recommendation includes:

json

{

"supplement": "Vitamin D3",

"dosage": 2000,

"unit": "IU",

"reason": "Low sun exposure, poor sleep",

"triggers": ["lifestyle.sun\_exposure", "sleep\_quality"],

"contraindications": []

}

**📦 Supplements in the Engine**

Already included:

* Vitamins: A, B6, B9 (Folate), B12, D, E, K
* Minerals: Iron, Magnesium, Zinc, Copper, Calcium, Selenium, Iodine, Chromium
* Others: Omega-3, Choline, Fiber, Potassium
* Formulas: Multivitamin
* Adaptogens: Ashwagandha, Rhodiola
* Hormonal: Melatonin

Dynamic extension supported via supplement\_db.json.

**🔧 How To Run**

You can test the pipeline end-to-end using:

python api.py # or integrate with FastAPI/Flask later

To simulate test users:

python generate\_mock\_users.py

To update clusters and assign recommendations:

python cluster\_runner.py

**🧪 Testing**

Run unit tests with:

pytest app/tests/

Test coverage includes:

* Dosage calculator
* Symptom scoring
* Drug interactions
* API integration
* Feedback adjustments

**🔮 Future Improvements**

* Add NLP for parsing freeform feedback
* Integrate real wearable APIs (placeholders exist)
* Migrate to database (currently uses JSON files)
* Wrap with FastAPI and expose endpoints
* Train custom ML models for dosage prediction

A black screen with white text

AI-generated content may be incorrect.

