Gold Investment Detection & Recommendation System in Kuber AI Chatbot

Name: Nikita Fulsundar Pawale
Contact: 8080756853 | nikitapawale782002@gmail.com
Assignment Title: API Development
Date of Submission: 27-08-2025

1. Introduction

This project develops a specialized API system that enhances the "Simplify Money" financial application by detecting gold investment-related queries in the Kuber AI chatbot. The API identifies when users ask about gold investments, provides accurate answers using Retrieval Augmented Generation (RAG) technology, and suggests appropriate gold investment plans from a curated database.

The system integrates multiple specialized components:

- A RAG-based intent detection service for identifying gold investment queries
- A knowledge-based answering system using Mistral AI and FAISS vector search
- A gold plan suggestion service with filtering capabilities

The primary objective was to create a focused API system that can accurately detect gold investment queries, provide contextually relevant answers, and recommend personalized investment plans based on user parameters.

2. Project Requirements

- **Programming Language:** Python 3.x
- **Web Framework:** FastAPI (for microservices), Django
- **Key Libraries:**
    - FastAPI, SentenceTransformers, FAISS, Mistral AI client, SQLAlchemy
    - Additional dependencies in requirements.txt files

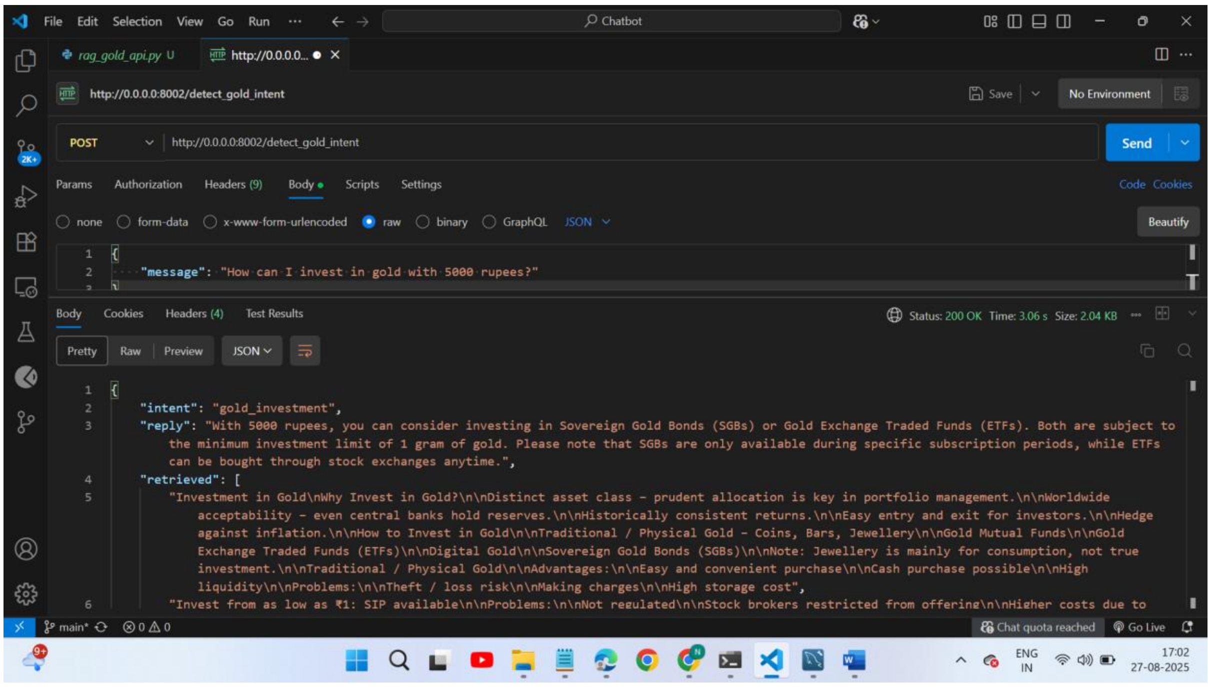
- **External APIs:** Mistral AI API (for response generation)
- **Data Storage:** MySQL database for gold investment plans
- **Vector Database:** FAISS for efficient similarity search

3. API Design

API 1: Gold Intent Detection Service

- **Purpose:** Detect gold investment queries using RAG methodology
- **Endpoint:** http://localhost:8002/detect_gold_intent
- **Method:** POST
- **Start command :** python -m uvicorn rag_gold_api:APP --host 0.0.0.0 --port 8002
- **Input Format:** JSON {"message": "user query string"}
- **Output Format:** JSON {"intent": "gold_investment" | "other", "reply": "response string", "retrieved": ["context chunks"]}
- **Technology:** FAISS vector database, SentenceTransformers embeddings, Mistral AI generation.
- **Knowledge Base:** The system relies on a single, curated text file (gold_doc.txt) containing all information about gold investments.

Testing using Postman


File Edit Selection View Go Run
Chatbot
rag_gold_api.py U
http://0.0.0.0...
Save
No Environment
Send
Params Authorization Headers (9) Body Scripts Settings
Code Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (4) Test Results
Pretty Raw Preview JSON
main*
Chat quota reached
Go Live
ENG IN
17:02
27-08-2025

http://0.0.0.0:8002/detect_gold_intent

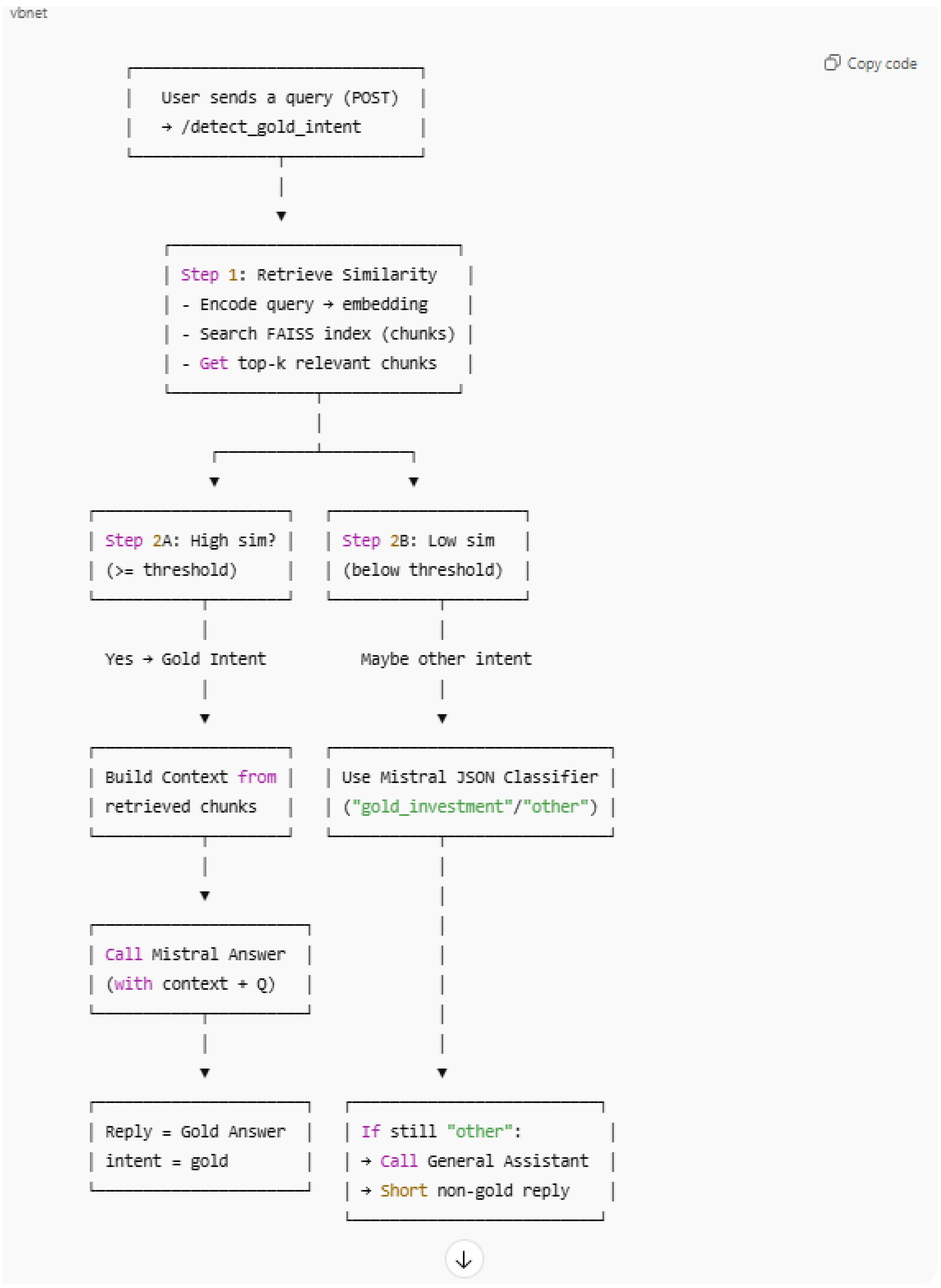
POST http://0.0.0.0:8002/detect_gold_intent

```
{
    "message": "How can I invest in gold with 5000 rupees?"
```

Status: 200 OK Time: 3.06 s Size: 2.04 KB

```
{
    "intent": "gold_investment",
    "reply": "With 5000 rupees, you can consider investing in Sovereign Gold Bonds (SGBs) or Gold Exchange Traded Funds (ETFs). Both are subject to
        the minimum investment limit of 1 gram of gold. Please note that SGBs are only available during specific subscription periods, while ETFs
        can be bought through stock exchanges anytime.",
    "retrieved": [
        "Investment in Gold\nWhy Invest in Gold?\n\nDistinct asset class - prudent allocation is key in portfolio management.\n\nWorldwide
            acceptability - even central banks hold reserves.\n\nHistorically consistent returns.\n\nEasy entry and exit for investors.\n\nHedge
            against inflation.\n\nHow to Invest in Gold\n\nTraditional / Physical Gold - Coins, Bars, Jewellery\n\nGold Mutual Funds\n\nGold
            Exchange Traded Funds (ETFs)\n\nDigital Gold\n\nSovereign Gold Bonds (SGBs)\n\nNote: Jewellery is mainly for consumption, not true
            investment.\n\nTraditional / Physical Gold\n\nAdvantages:\n\nEasy and convenient purchase\n\nCash purchase possible\n\nHigh
            liquidity\n\nProblems:\n\nTheft / loss risk\n\nMaking charges\n\nHigh storage cost",
        "Invest from as low as ₹1: SIP available\n\nProblems:\n\nNot regulated\n\nStock brokers restricted from offering\n\nHigher costs due to
```

Flow of Gold Intent detection API:


vbnet

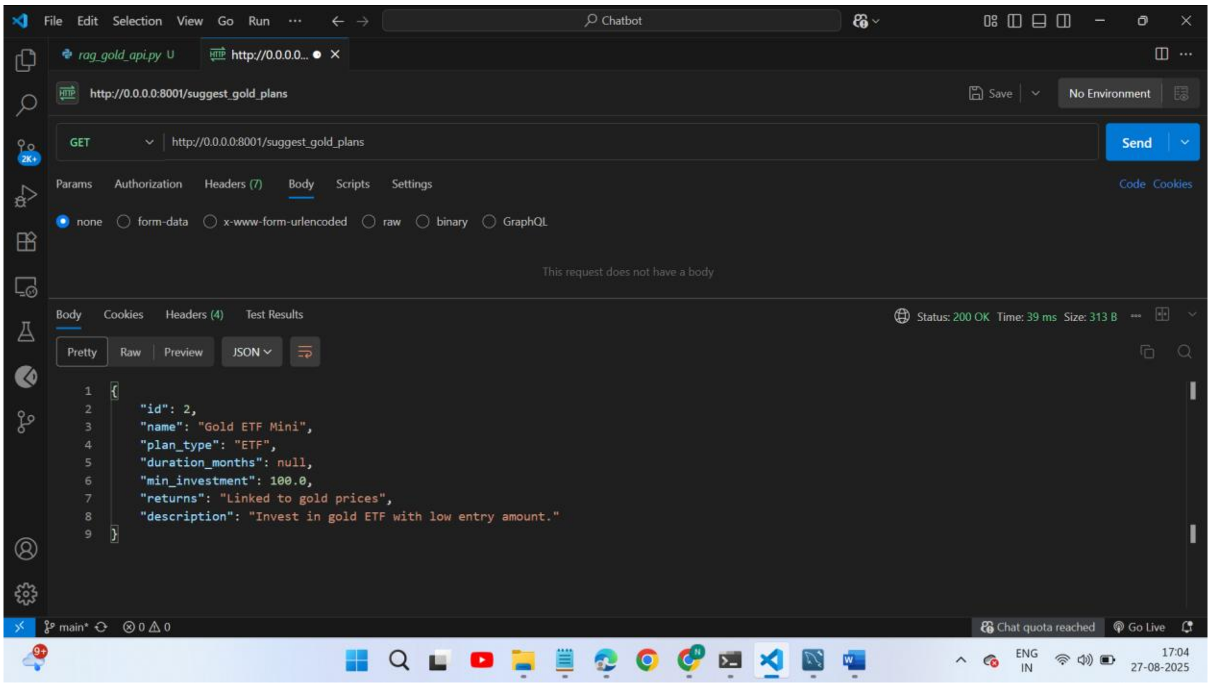
Copy code

```
        ┌──────────────────────────────┐
        │  User sends a query (POST)   │
        │  → /detect_gold_intent       │
        └───────────────┬──────────────┘
                        │
                        ▼
            ┌──────────────────────────────┐
            │ Step 1: Retrieve Similarity  │
            │ - Encode query → embedding   │
            │ - Search FAISS index (chunks)│
            │ - Get top-k relevant chunks  │
            └───────────────┬──────────────┘
                            │
                 ┌──────────┴──────────┐
                 ▼                     ▼
┌────────────────────┐     ┌────────────────────┐
│ Step 2A: High sim? │     │ Step 2B: Low sim   │
│ (>= threshold)     │     │ (below threshold)  │
└───────────┬────────┘     └───────────┬────────┘
            │                          │
     Yes → Gold Intent          Maybe other intent
            │                          │
            ▼                          ▼
┌────────────────────┐     ┌─────────────────────────────┐
│ Build Context from │     │ Use Mistral JSON Classifier │
│ retrieved chunks   │     │ ("gold_investment"/"other") │
└───────────┬────────┘     └───────────┬─────────────────┘
            │                          │
            ▼                          │
┌──────────────────────┐               │
│ Call Mistral Answer  │               │
│ (with context + Q)   │               │
└───────────┬──────────┘               │
            │                          │
            ▼                          ▼
┌──────────────────────┐     ┌──────────────────────────┐
│ Reply = Gold Answer  │     │ If still "other":         │
│ intent = gold        │     │ → Call General Assistant  │
└──────────────────────┘     │ → Short non-gold reply    │
                             └──────────────────────────┘
```

API 2: Gold Plan Suggestion Service

- **Purpose:** Provide filtered gold investment recommendations
- **Endpoint:** http://localhost:8001/suggest_gold_plans
- **Method:** GET
- **Start command:** python -m uvicorn main:app --host 0.0.0.0 --port 8001
- **Query Parameters:** budget, duration_months, plan_type (optional filters)
- **Output Format:** JSON with plan details
- **Technology:** SQLAlchemy, MySQL database

Testing using postman


File Edit Selection View Go Run
Chatbot
rag_gold_api.py U
http://0.0.0.0...
Save
No Environment
Send
Params Authorization Headers (7) Body Scripts Settings
Code Cookies
none form-data x-www-form-urlencoded raw binary GraphQL
Body Cookies Headers (4) Test Results
Pretty Raw Preview JSON
main*
Chat quota reached
Go Live
ENG IN
17:04
27-08-2025

http://0.0.0.0:8001/suggest_gold_plans

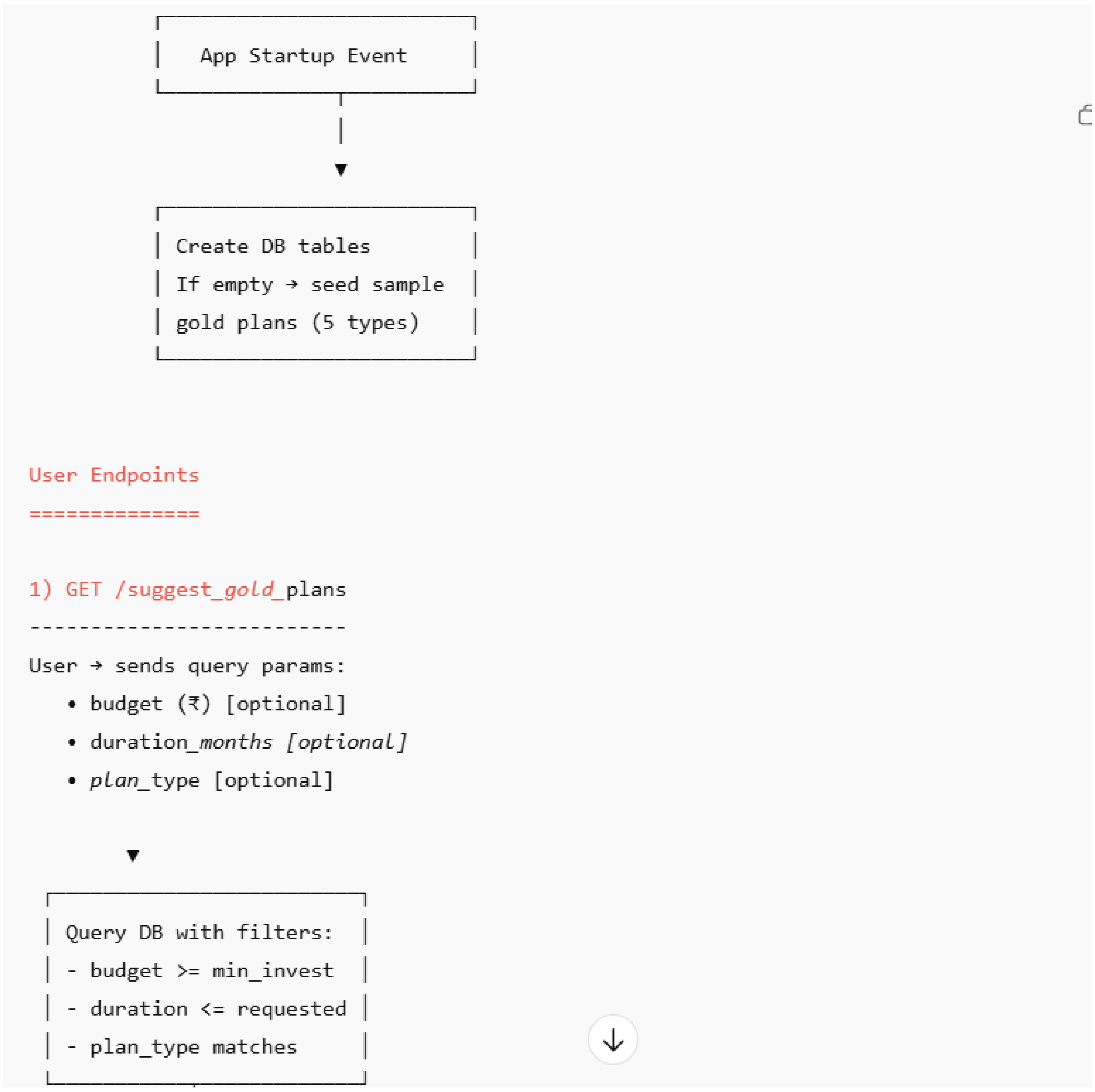
GET http://0.0.0.0:8001/suggest_gold_plans

This request does not have a body

Status: 200 OK Time: 39 ms Size: 313 B

```
{
    "id": 2,
    "name": "Gold ETF Mini",
    "plan_type": "ETF",
    "duration_months": null,
    "min_investment": 100.0,
    "returns": "Linked to gold prices",
    "description": "Invest in gold ETF with low entry amount."
}
```

Flow of Gold Plans Suggestion API:

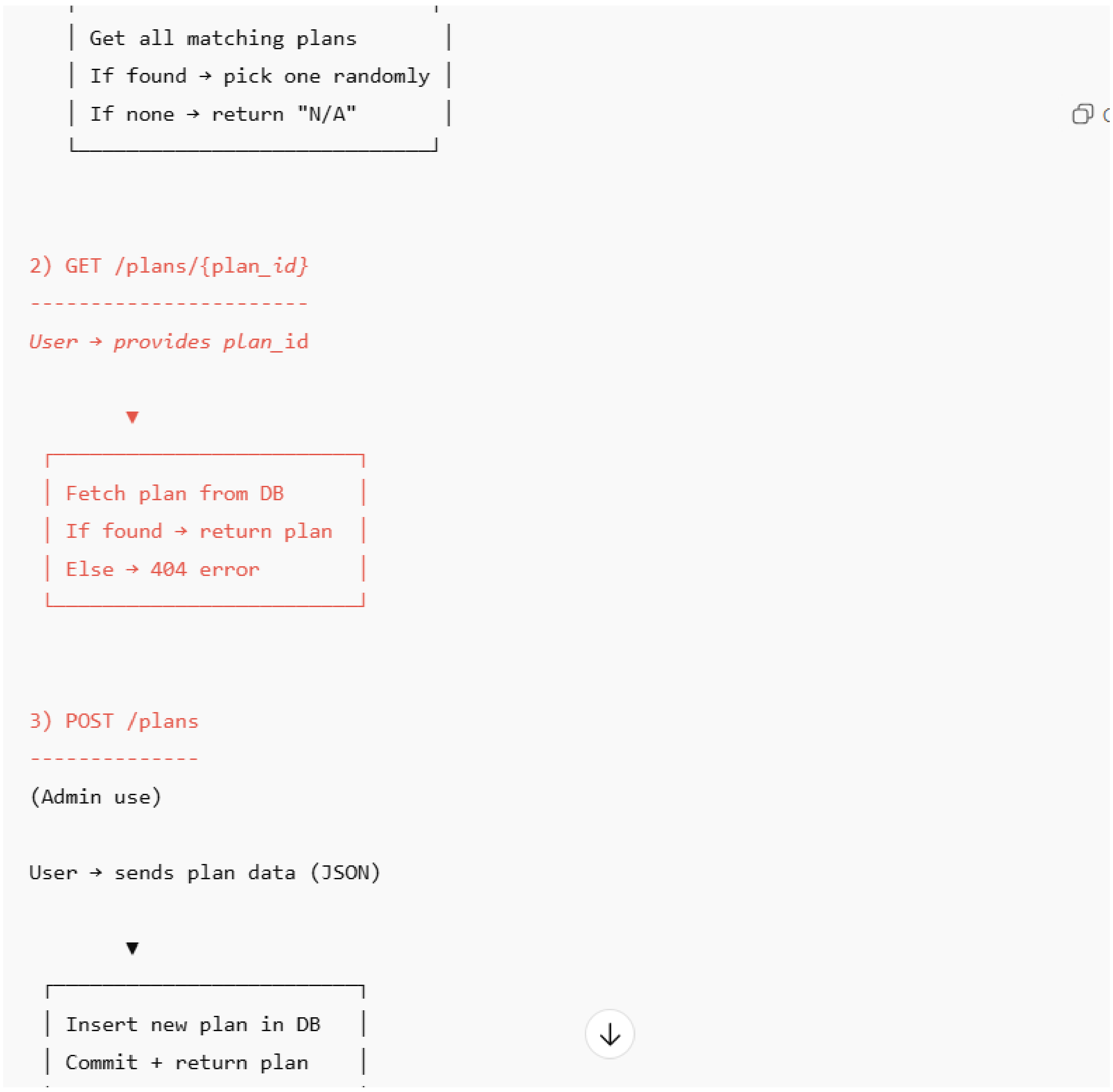


```
            ┌─────────────────────────┐
            │   App Startup Event     │
            └──────────────┬──────────┘
                           │
                           ▼
            ┌─────────────────────────┐
            │ Create DB tables        │
            │ If empty → seed sample  │
            │ gold plans (5 types)    │
            └─────────────────────────┘


  User Endpoints
  ==============

  1) GET /suggest_gold_plans
  --------------------------
  User → sends query params:
     • budget (₹) [optional]
     • duration_months [optional]
     • plan_type [optional]

          ▼
   ┌─────────────────────────┐
   │ Query DB with filters:  │
   │ - budget >= min_invest  │
   │ - duration <= requested │
   │ - plan_type matches     │
   └───────────┬─────────────┘
```



```
    │ Get all matching plans      │
    │ If found → pick one randomly │
    │ If none → return "N/A"      │
    └─────────────────────────────┘


2) GET /plans/{plan_id}
-----------------------
User → provides plan_id

        ▼
  ┌──────────────────────────┐
  │ Fetch plan from DB       │
  │ If found → return plan   │
  │ Else → 404 error         │
  └──────────────────────────┘


3) POST /plans
--------------
(Admin use)

User → sends plan data (JSON)

        ▼
  ┌──────────────────────────┐
  │ Insert new plan in DB    │
  │ Commit + return plan     │
```

4. Project Structure

```
chatbot/
├── backend/                 # Django project directory
|    ├── __init__.py
|    ├── asgi.py
|    ├── settings.py        # Contains API keys and app config
|    ├── urls.py           # Root URL configuration
|    └── wsgi.py
├── frontend/               # Django app directory
|    ├── __init__.py
|    ├── admin.py
|    ├── apps.py
|    ├── models.py
|    ├── tests.py
|    ├── urls.py           # Defines path to 'get_chatbot_response'
|    ├── views.py          # Contains the core logic provided
|    └── templates/
|        └── chatbot.html   # The frontend chat interface
├── db.sqlite3              # SQLite database for storing conversation
├── API/                    # Gold Plan Suggestion Service
|
|    ├── rag_gold_api.py   # FastAPI app for intent detection
|    ├── schemas.py        # Pydantic schemas
|    ├── requirements.txt  # Dependencies
|    └── gold_doc.txt      # Knowledge base for gold investments
└── API1/                   # Intent Detection Service
    ├── main.py            # fast_api for gold plan suggetion
    ├── models.py          # SQLAlchemy models
    ├── schemas.py         # Pydantic schemas
    ├── database.py        # Database connection to mysql
    └── requirements.txt   # Dependencies
```

6. Implementation Steps

1. **Project Setup:** Created Django project and app structure along with two FastAPI microservices
2. **Gold Plan Suggetion (API1):**
    - Implemented database model for gold investment plans
    - Created RESTful endpoints for plan suggestions
    - Added filtering capabilities based on budget, duration, and plan type
    - Implemented seeding of sample data
3. **RAG-based Intent Detection Service (API):**
    - Implemented FAISS vector database for efficient similarity search

  - Created text chunking and embedding pipeline using SentenceTransformers
  - Integrated Mistral AI for response generation with context from knowledge base
  - Implemented round-robin API key rotation for handling rate limits

4. **Django Chatbot Integration:**
   - Implemented get_chatbot_response view in frontend/views.py to handle decision-making workflow
   - Integrated with both external services
   - Added routing logic:
     - For gold investment intent: Fetches suggested plan from gold plan service
     - For other intents: Queries Mistral AI API with tailored prompt
   - Implemented comprehensive error handling
5. **Knowledge Base Preparation:**
   - Created gold_doc.txt with comprehensive information about gold investments
   - Implemented text chunking optimized for retrieval

7. Technical Challenges:

- Handling multiple external API dependencies and potential failures
- Implementing efficient similarity search with FAISS
- Managing API rate limits with key rotation
- Ensuring proper context retrieval for RAG responses
- Deployment was hindered by the high memory requirements of SentenceTransformers/FAISS on free cloud tiers and payment barriers for scalable services

8. Testing Procedures

1. **Gold Intent Detection Testing:**
   - Test with various gold-related queries
   - Verify context retrieval accuracy
   - Check response relevance
2. **Plan Suggestion Testing:**
   - Test filtering with different parameters

    - Verify plan data accuracy
    - Check error handling for edge cases

3. **Integration Testing:**
    - Test end-to-end query processing
    - Verify data flow between components

8. Results

The API system successfully:

- Accurately detects gold investment queries with high precision
- Provides contextually relevant answers using RAG methodology
- Suggests appropriate investment plans based on user parameters
- Handles API rate limits through key rotation
- Maintains robust performance through comprehensive error handling

9. Conclusion

The Gold Investment Detection & Recommendation System successfully enhances the **Kuber AI chatbot** within the *Simplify Money* application by integrating specialized APIs for gold-related query handling. Through the combination of **RAG-based intent detection, FAISS-powered knowledge retrieval, and Mistral AI response generation**, the system accurately identifies user intent, delivers contextually relevant answers, and recommends gold investment plans.

The modular design with **two FastAPI microservices** ensures scalability, maintainability, and clear separation of responsibilities between intent detection and plan recommendation. Despite challenges such as memory constraints, external API dependencies, and deployment limitations, the system demonstrates strong performance in query detection, contextual response generation, and plan filtering.

Overall, this project highlights the effectiveness of combining **knowledge-based retrieval, generative AI, and database-driven recommendations** to create intelligent financial advisory systems. It provides a reliable foundation for future improvements, such as advanced personalization, cloud deployment optimization, and integration with broader investment domains beyond gold.

10. Future Scope

1. **Advanced Personalization** – Use user profiles, risk preferences, and history to give more tailored investment suggestions.
2. **Multi-Asset Expansion** – Extend recommendations beyond gold to include stocks, mutual funds, and other asset classes.
3. **Real-Time Market Data** – Integrate live gold prices and financial APIs for up-to-date advice.
4. **Cloud Deployment** – Optimize the system for scalable cloud deployment to handle higher traffic and real-world usage.

Setup & Usage Guide

1. **Clone the Repository** and navigate to the project folder. (https://github.com/pawalenikita78/Kuber_AI.git)
2. **Install dependencies** from requirements.txt in both API folders (api1, api2).
3. **Start FastAPI services**:
4. Intent Detection API → uvicorn rag_gold_api:APP --host 0.0.0.0 --port 8002
5. Plan Suggestion API → uvicorn main:app --host 0.0.0.0 --port 8001
6. **Run Django server** using python manage.py runserver.
7. **Access the chatbot** at http://127.0.0.1:8000 to interact with gold query detection and recommendations.