Case Study AI Specialist

Hi 👋🏼

Congrats! If you’ve reached this point, it’s because we see potential in you and believe you can crush it with the Case Study. So, **good luck, you’ve got this**! 🍀 🚀

The Challenge: Intelligent Technical Support System with Personalized Recommendations

1. The Challenge

Develop an intelligent technical support system with personalized recommendation capabilities for the Shakers platform, and display its performance through a metrics dashboard.

2. The Rules

We want a system that helps our clients find the perfect talent. To do this, we need to be able to provide them with precise information about how our platform works ("How do payments work?" or "What is a freelancer?") and recommend the most suitable talents based on their requests ("I need an Android developer with 8 years of experience").

You can use any Python framework you prefer (recommended: FastAPI or Flask for the backend and Streamlit). Since this is for evaluating an AI Engineer, the solution should integrate language models, retrieval systems, and evaluation components. Remember to use Git to commit your progress as you go; we will review the history to understand your decision-making process and how you tackle the various challenges of the test.

The objective of the system is twofold:

1. Answer technical user questions based on a provided knowledge base.
2. Proactively recommend relevant resources (articles, tutorials) based on the user's query history.

Core Features (Minimum Requirements)

1. RAG Query Service

- **Input**: User query about Shakers and the freelance world
- **Process**:
  - Query vectorization
  - Retrieval of relevant documents from the knowledge base
  - Generation of a contextualized response using an LLM
- **Output**: Clear and concise response with references to the sources used
- **Specific Requirements**:
  - The system must detect when a query is out of scope and respond appropriately ("I don't have information on this...")
  - Responses must include links or references to the original documentation
  - Maximum response time: 5 seconds

2. Personalized Recommendation Service

- **Input**: User's query history and current query
- **Process**:
  - Analysis of the user's profile based on previous queries
  - Identification of topics/areas of interest
  - Selection of relevant, previously unconsulted resources
- **Output**: 2-3 personalized recommendations with a brief explanation of their relevance
- **Specific Requirements**:
  - Recommendations should be diverse (not all on the same topic)
  - Each recommendation must include a brief explanation of why it is being suggested
  - Recommendations should be updated after each new query

Considerations

Data you will need to simulate to run the tests:

- A collection of technical documents in markdown format about a fictional SaaS platform.
- A dataset of example questions and their ideal answers for evaluation.
- A set of user profiles with simulated query history.

All of your deliverables must be submitted via a Git repository, with documentation and organization that make it easy to understand and analyze. We don't mind if you use AI to help you code the technical test, but you are responsible for defending all the code you submit as part of your final solution. All code must be written in English, although the system can interact in Spanish.

Disclaimer:

To complete the test, you may access the following APIs, which include free tiers in their plans. Simply include the values in a .env.example file, and we will replace them with our own during evaluation:

- Google Gemini: https://ai.google.dev/gemini-api/docs/pricing
- Groq: https://groq.com/

3. Evaluation

3.1 Setup

Criterion	Description
Documentation	The project must be properly documented. This includes installation instructions, technologies used, system architecture, and how to run the tests.
Installation	After following the instructions, the project must start correctly without errors. You must include a virtual environment to ease installation.

3.2 RAG System

Criterion	Description
Document Indexing	The system correctly vectorizes and indexes the provided knowledge base. The indexing must be efficient and allow updates.

Query and Retrieval	The retrieval of relevant documents works correctly for different types of queries (direct, ambiguous, out of scope).
Response Generation	The generated responses are natural, accurate, and based on the retrieved information. The system must cite the sources used.
Uncertainty Management	The system recognizes when a query is beyond the knowledge base's scope and responds accordingly.

3.3 Recommendation System

Criterion	Description
User Profiles	The system builds user profiles based on their query history and preferences.
Personalized Recommendations	It generates relevant, personalized recommendations for additional resources based on the current context and user profile.
Explainability	The system provides understandable explanations for why each resource is recommended.
Recommendation Diversity	The recommendations maintain a proper balance between relevance and diversity.

3.3 Backend

Criterion	Description
Well-Structured API	The server's API is well-designed, with clear and documented endpoints (Swagger/OpenAPI).
Model Optimization	The system implements strategies to optimize the performance and cost of API calls to LLMs or local models.
Data Storage	User profiles, query history, and performance metrics are stored correctly.
Logging and Monitoring	The system logs relevant information for debugging and performance analysis.

4. Timing

You will have 7 days to code a valid solution. If you need more time, you can request it. The idea is to give you enough time to create a high-quality solution without unnecessary pressure.

When you submit your solution, we will analyze it and contact you to schedule a video call where we will discuss your approach, technical decisions, and potential improvements. Think of this video call as a technical conversation about the system you developed.

5. Extra Miles (Additional Features Valued)

Advanced Testing and Evaluation

- **Unit tests** for key components (retrieval, generation, recommendation)
- **Integration tests** for complete workflows
- **Automated evaluation** against a ground truth dataset using metrics such as:
    - Accuracy in retrieving key information
    - Relevance of recommendations via cross-validation

Enhanced Interaction

- **Conversational history** allowing follow-up queries with context
- **Sentiment detection** to adapt the tone of the response
- **Explicit feedback** ("helpful/not helpful" buttons) to improve future responses

Optimization and Performance

- **Smart caching** of frequent queries/responses
- **Advanced prompting strategies** with few-shot examples or chain-of-thought
- **Context compression** to optimize tokens in API calls to LLMs

Robustness and Security

- **Malicious query detection** or prompt injection attempts
- **Filtering inappropriate content** in generated responses
- **High load handling** via rate limiting or processing queues

Expense Tracking and Monitoring

- Latency, cost, and call metrics tracking
- Dynamic model comparison to choose the most suitable for each system

- Fallbacks, retries, and protection against failures

6. FAQ

What AI APIs or models can I use?

You can use any AI API (OpenAI, Anthropic, Gemini, etc.) or open-source models (Hugging Face). We recommend using the recommended free APIs.

Do I need to implement the entire system from scratch?

No, you can and should leverage libraries like LangChain, LlamaIndex, or similar frameworks. What matters is how you integrate these components and the added value you provide.

How complex should the recommendation system be?

We don’t expect extremely sophisticated algorithms. A basic approach based on embedding similarity or basic collaborative filtering is sufficient. What matters is the reasoning behind your implementation.

Do I need to implement all the functionalities?

Prioritize quality over quantity. It’s better to have a basic but functional RAG and recommendation system than to try to implement all features without completing any correctly.

Can I add extra features?

Of course! Some interesting ideas might include: customizing response tone, detecting user sentiment, feedback systems to improve future responses, etc. Just make sure the basic functionality works correctly.

How do I manage costs for AI APIs?

If you use paid APIs, implement caching and prompt optimization strategies to minimize costs. You can also provide mocks or tests that don’t require real API calls.

What technical aspects do you particularly value?

We’re interested in aspects such as: prompt structure, information retrieval strategy, the balance between accuracy and latency, and how you evaluate and improve the system’s performance.

If you have any further questions, feel free to contact us:
a.marin@shakersworks.com or e.garcia@shakersworks.com 🚀