

# Home Exam | Junior AI Engineer Course

## Overview

You've passed the initial screening exam. Congratulations! Now we'd like to assess your practical skills in building a Retrieval-Augmented Generation (RAG) system. This assignment focuses on the **retrieval** component of RAG. You'll build a system that can find relevant context from a dataset based on user queries.

**Important:** This assignment does NOT require implementing the LLM generation step. You'll focus on building the retrieval pipeline and demonstrating your understanding of RAG fundamentals.

---

## Objectives

Build a simple RAG retrieval system that:

1. Ingests data from a Kaggle dataset.
  2. Creates embeddings and stores them in a vector database.
  3. Retrieves relevant context chunks based on user queries.
  4. Displays the retrieved context in a minimal UI.
- 

## Requirements

### 1. Dataset Selection

- **Source:** Choose any *small* dataset from <https://www.kaggle.com/datasets> or another open, freely available site.

- **Size:** Very Small
    - **Unstructured text datasets:** Total text under **30,000 characters** (e.g., short stories, <30 short articles, 50–100 short reviews)
    - **Structured/tabular datasets:** No more than **200 rows** and **≤10 columns** (e.g., product descriptions, trivia Q&A).
  
  - **Type:** Data must be text-based, not numerical.
  
  - **Documentation:** Briefly explain:
    - Why did you choose this dataset.
    - What kind of user questions you'd expect.
- 

## 2. Vector Database Selection

Choose one: **Chroma, Pinecone, Weaviate, Mongo, pgvector**, or any other option of your choosing.

Document: Why did you choose this vector DB?

---

## 3. Embeddings

- Use OpenAI's embedding model (e.g., text-embedding-3-small).
- Cost must remain under \$1 USD.

**Tip:** With a small dataset such as suggested above, embeddings should cost far far below \$1.

---

## 4. RAG Implementation

### Chunking Strategy

- Decide chunk size and overlap.

### Retrieval Parameters

- **Similarity threshold:** Minimum cosine similarity score to accept results.
  - **Top-K:** Number of chunks to return per query.
- 

## 5. Minimal UI

### Requirements

- Input field for user queries.
- Display retrieved context chunks.
- Show similarity scores (optional but recommended).
- Show which chunks were retrieved (IDs or indices).

### Tech

- **Backend:** Flask or FastAPI.
- **Frontend:** React or plain HTML/CSS/JavaScript.

### No LLM required

The UI must only display retrieved context — **no generated responses.**

---

## Deliverables

### 1. Code Repository

- Clean, readable structure
- README with setup instructions

## 2. Documentation

Include:

- **Usage Instructions:**
  - How to set up
  - How to interact with UI

## 3. Demo/Video (Optional)

2 minute screen recording showing:

- System usage
  - Example queries and results
- 

## Evaluation Criteria

### 1. Technical Implementation (25%)

- Code quality & structure
- Working retrieval pipeline

### 2. Reasoning & Choices (75%)

- Clarity of decisions
  - Justification of decision
  - Overall communication of your thought process
-

## Constraints & Guidelines

### Must NOT Have:

- LLM generation or chatbot functionality
- Costs exceeding \$1

### Recommended:

- Use environment variables for API keys
  - Include proper error handling
  - Test multiple query types
  - Consider edge cases (empty queries, no matches, etc.)
- 

## Submission

Submit via email:

- GitHub repository link (public or with access)
- Optional: Demo video link

**Deadline:** One week after receiving this task.

---

Good luck — we're excited to see what you build!