**SHL Assessment Recommendation System – Implementation & Deployment Overview**

**Objective**

The goal of the project was to build a Generative AI-powered SHL Assessment Recommendation System. Given a job description or a query (including URLs), the system identifies and recommends the most relevant SHL assessments (up to 10) from the SHL product catalogue.

**Approach**

**1. Problem Understanding & Flow Design**

We began by breaking the system down into key components:

* **Input Handler:** Accepts free-text queries or URLs containing job descriptions.
* **Text Extractor:** Extracts meaningful job description text from a webpage if a URL is detected.
* **LLM-powered Categorization:** Uses Generative AI (Google Gemini) to understand the query and extract relevant job skills and context.
* **Assessment Matcher:** Finds the best-matching SHL assessments using semantic similarity (vector search via FAISS).
* **Frontend Interface:** Streamlit-based UI that takes user input and displays recommended assessments.

**2. Key Tools, Technologies & Libraries**

| **Layer** | **Tools/Libraries Used** |
| --- | --- |
| Backend | FastAPI, FAISS, Requests |
| Frontend | Streamlit |
| LLM API | Google Generative AI (google-generativeai Python SDK) |
| Vector Search | FAISS (Facebook AI Similarity Search) |
| Deployment | Render (for FastAPI backend), Streamlit Cloud/Vercel |
| Storage | CSV for SHL Assessment Catalog (initial version) |
| Others | BeautifulSoup, urllib, re, dotenv |

**3. Backend Logic (FastAPI)**

* **Data Preparation**:
  + The SHL catalogue (CSV) is pre-processed to extract test titles, descriptions, and categories.
  + Each test’s description is embedded using a text embedding model (Gemini/OpenAI) and indexed using **FAISS** for efficient similarity search.
* **Input Handling**:
  + If the input is a URL, the backend fetches its content using requests, then parses and extracts text using BeautifulSoup.
  + The extracted or raw input text is then passed to the LLM for job description summarization.
* **LLM Integration**:
  + We used **Google’s Gemini API** (google.genai) for job role classification and to extract the core job requirements or skills.
* **Recommendation Engine**:
  + The extracted job content is embedded and compared with the SHL catalogue embeddings using **cosine similarity** through FAISS.
  + Top 10 matches are selected and returned in structured JSON format.

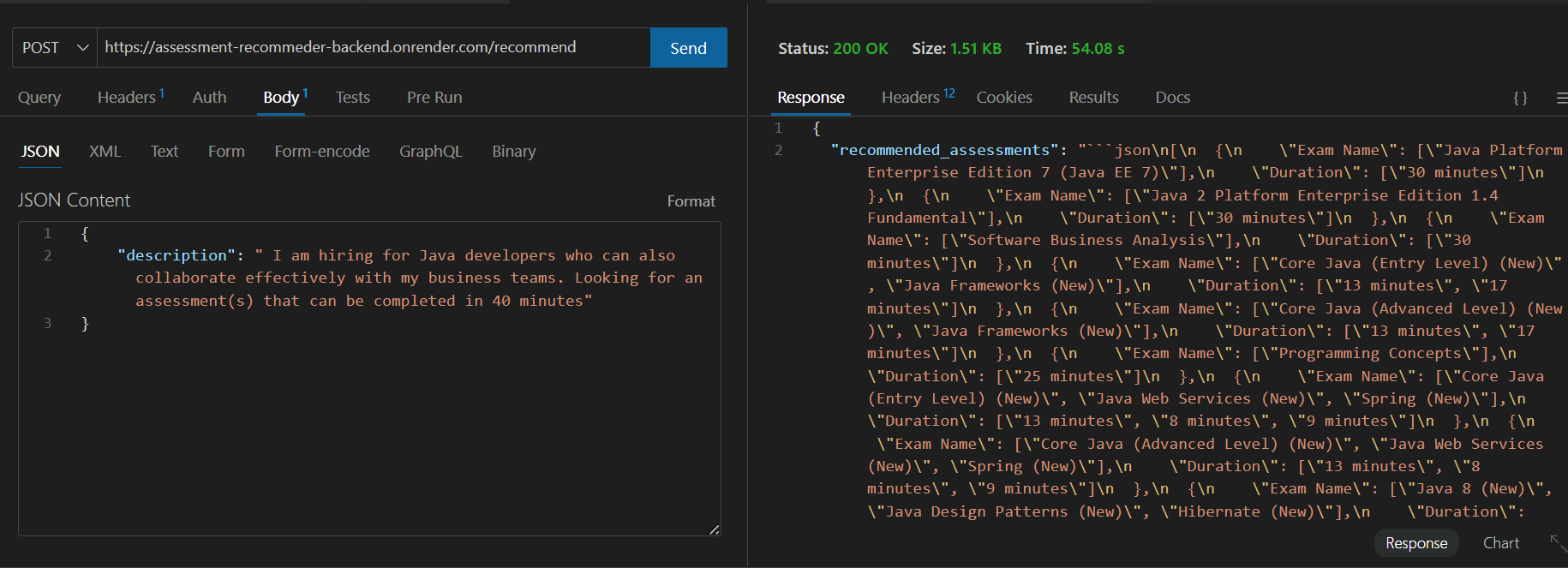
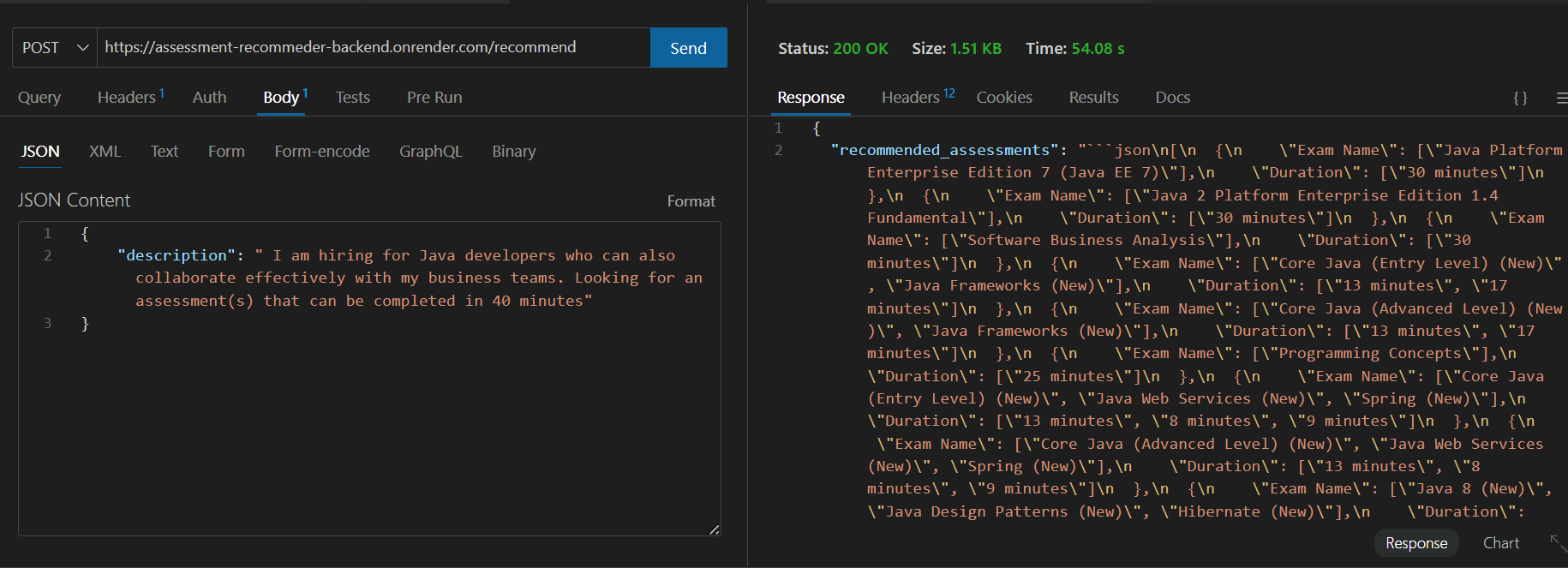
**4. Frontend (Streamlit App)**

* A simple and responsive **Streamlit UI** lets users:
  + Paste job descriptions or URLs
  + See real-time recommendations
  + Interact with formatted result cards (showing test names and summaries)
* Added enhancements:
  + URL detection and parsing
  + LLM response display for transparency
  + Error handling for broken URLs or empty responses

**5. Deployment Strategy**

* **Backend**:
  + Hosted on **Render**, configured to bind to 0.0.0.0 on the PORT environment variable.
  + Included a custom startup event to load documents and initialize the FAISS index at boot time.
* **Frontend**:
  + Hosted separately via **Streamlit Cloud** for better separation of concerns.
* **Environment Configuration**:
  + Stored sensitive credentials like API\_KEY securely using os.environ to access environment variables within the application code, ensuring secure and configurable deployment across environments.

**Sample Request**

****