#backend/logic/qa.py

```
import re
import logging
import numpy as np
import google.generativeai as genai
from .config import GEMINI_CHAT_MODEL, GEMINI_EMBED_MODEL

logger = logging.getLogger(__name__)

def search_and_answer(query, index, texts, metadata):
    """
    Improved search_and_answer using Gemini:
    - Handles "New question:" prefix
    - Supports slide/page-specific retrieval
    - Uses Gemini embeddings + FAISS search
    - Dedupes and safely falls back if index fails
    - Builds a safe, concise prompt for Gemini chat
    """
    print("RAW USER QUERY:", query)

    # --- parse question ---
    try:
        match_new = re.search(r"New question:\s*(.+)", query, re.IGNORECASE | re.DOTALL)
        actual_question = match_new.group(1).strip() if match_new else query.strip()
    except Exception:
        logger.exception("Failed to parse question")
```

```
    actual_question = query.strip()


if not index or not texts:
    return "Sorry, I don't have data to answer that yet. Upload a file first."


# --- retrieval ---
retrieved = []
try:
    # Slide/page-based retrieval
    match = re.search(r"(slide|page)\s+(\d+)", actual_question.lower())
    if match:
        num = int(match.group(2))
        retrieved = [
            texts[i] for i, meta in enumerate(metadata or [])
            if str(meta.get("slide") or meta.get("page", "")) == str(num)
        ]
        if not retrieved:
            retrieved = texts[:min(3, len(texts))]


    else:
        # Gemini embeddings
        try:
            emb_resp = genai.embed_content(
                model=GEMINI_EMBED_MODEL,
                content=actual_question,
                task_type="retrieval_query"
```

```
            )
            emb = emb_resp["embedding"]
            print("EMBEDDING CREATED FOR:", actual_question)
        except Exception:
            logger.exception("Embedding API failed")
            return "Embedding service failed. Please try again later."

        # FAISS search
        try:
            vec = np.array([emb], dtype="float32")
            D, I = index.search(vec, 3)
            ids = [int(i) for i in I[0] if i is not None and 0 <= int(i) < len(texts)]
            # dedupe while preserving order
            seen, valid_ids = set(), []
            for i in ids:
                if i not in seen:
                    valid_ids.append(i)
                    seen.add(i)
            retrieved = [texts[i] for i in valid_ids] or texts[:min(3, len(texts))]
        except Exception:
            logger.exception("Index search failed")
            retrieved = texts[:min(3, len(texts))]

    except Exception:
        logger.exception("Retrieval step failed")
        retrieved = texts[:min(3, len(texts))]
```

```
# --- prompt building ---
try:
    MAX_CHARS = 16000
    joined = "\n\n".join(str(r) for r in retrieved)
    if len(joined) > MAX_CHARS:
        joined = joined[:MAX_CHARS]

    prompt = f"""
```

You are an assistant helping the user understand their uploaded files.

File Content:

{joined}

Conversation context (last Q&A + new question):

{query}

Guidelines:

1. Always provide a clear, concise and accurate answer based only on the retrieved File content.

2. If the file text is unclear, still extract the most accurate and meaningful information possible — do not mention that the content is messy, incomplete, or mismatched.

3. If the question is a follow-up, treat it as a request to expand or clarify your last answer.

4. Keep tone factual, simple, and user-friendly.

5. Do not include disclaimers about file quality — just answer directly from the content.

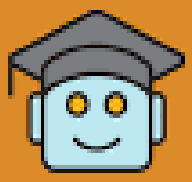
Answer:

```
"""
    print("FINAL PROMPT TO GEMINI:", prompt[:800], "..." if len(prompt) > 800 else "")
  except Exception:
    logger.exception("Prompt construction failed")
    return "I couldn't build a response. Try again later."

  # --- Gemini LLM call ---
  try:
    model = genai.GenerativeModel(GEMINI_CHAT_MODEL)
    response = model.generate_content(prompt)
    text = getattr(response, "text", None) or ""
    print("RAW GEMINI RESPONSE:", text)
    return text if text else "I couldn't generate an answer this time."
  except Exception:
    logger.exception("Gemini chat completion failed")
    return "Sorry, I'm having trouble answering that question right now."
```

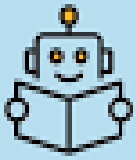
WHAT ARE LARGE LANGUAGE MODELS (LLMS)?

A type of artificial intelligence (AI) that's trained to create sentences and paragraphs out of its training dataset.



The main LLMS include:

Zero-shot models are for general purposes.



Domain-specific models are given extra training.

Edge models do one job well.

Payment Details

PREPAID
9384298954

₹349.00

Payment Successful
21 Jul 2025 · 11:11 AM

Payment type
Recharge

Airtel transaction id
7352935623803576320

Airtel posting id
7352935728076529664

Bank reference id
288589033712

Payment channel
Thanks App

Payment mode
UPI