Milestone - 5

Project Title: SmartDocQ

Team ID: G - 573

Team Members:

Sarika - 23BD1A664K

Sougandhika - 23BD1A665L

Raniya Rida - 23BD1A665G

Sankeertana - 23BD1A665J

Shishir Gella - 23BD1A6751

Date: 01-09-2025

1. Introduction

SmartDocQ is a Python application designed to let users upload PDF documents and interact with them using Google Gemini AI. It demonstrates text extraction, AI-powered question answering, and console-based user interaction.

The main objectives of this milestone are:

- o Implement PDF upload and storage
- o Extract text from uploaded documents
- o Enable querying through Google Gemini AI
- o Handle errors gracefully (file not found, no document uploaded, invalid input)
- o Provide a simple console-based interface for interaction

2. Project Setup

Language: Python 3.12

Libraries used:

- o `os` → file handling and environment variables

- `dotenv` → load environment variables from `.env`
- `PyPDF2` → extract text from PDF files
- `google.generativeai` → interact with Google Gemini AI

3. API Key Management

Store the Gemini API key inside the `.env` file as `GOOGLE_API_KEY`.

Load with `dotenv`.

Raise an error if the key is not found.

4. Functional Endpoints

2. **upload PDF**

   Function: `upload_pdf(file_path)`

   Purpose: Extracts text from the provided PDF and saves it in memory.

   Steps:

   - Open the PDF in binary mode.
   - Use `PyPDF2.PdfReader` to read page text.
   - Concatenate extracted text into a string.
   - Store the filename and text inside the `documents` dictionary.
   - Print a success message.

3. **Validation**

   - Check if the file exists before processing..
   - Handle unreadable PDFs gracefully.

   Output Example:

   ```
   \n✅ sample.pdf uploaded successfully!\n
   ```

4. **Ask Gemini a Question**
   - Function: `ask_gemini(question)`
   - Purpose: Query Gemini AI using the text from uploaded PDFs.

   Steps:

   1. Verify that at least one PDF is uploaded.
   2. Merge content from all uploaded documents.
   3. Construct a prompt combining document text and user's question.
   4. Send the prompt to Gemini (`gemini-2.0-flash`).
   5. Return Gemini's generated answer.

Validation:

- Warn the user if no documents are uploaded.

Output Example:

```
\n🤖 Gemini Answer:\n <AI response here>\n
```

5. Main menu

Function: `main()`

Purpose: Provides a console-driven menu for user interaction.

Options:

1. Upload PDF
2. Ask a Question

3. Exit

Features:

1. Runs continuously until the user exits.
2. Handles invalid choices gracefully.
3. Prints informative messages for success/warnings/errors.

6. Business Logic

1. Maintains uploaded documents in an in-memory dictionary.
2. Supports multiple PDFs (merges their text into one context).
3. Ensures AI responses are generated based on provided document content.

7. Input Validation & Error Handling

- Check file path before upload.
- Ensure the database (in this case, dictionary) is not empty before answering.
- Manage invalid menu options with a warning message.

8. Response Formatting

- Upload: ✅ Success / ❌ Error messages in console.
- AI Answer: Clearly displayed under "🤖 Gemini Answer".
- Warnings: ⚠ No documents available.

9. Deliverables

- Updated SRS Document (PDF) with implementation details.
- A demo video showing :
    1. Uploading a PDF
    2. Asking a question
    3. Getting AI-generated output
- Full Python source code submission.

10. Future Improvements

- Add a web-based frontend for easier interaction.
- Replace in-memory storage with a database.
- Extend to multiple AI models.
- Improve preprocessing (summarization, noise removal).
- Build FastAPI endpoints for real-time integration.

11. Model Output

Reserved for Gemini AI’s answers during execution.