Sri Lanka Institute of Information Technology

Current Trends in Software Engineering - SE4010

Assignment I

Name	**Registration Number**
Tennakoon T. M. T. C.	IT21171338

Table of Contents

1. Introduction

This report outlines the development of a simple chatbot designed to answer questions based on CTSE lecture notes. The solution was implemented in a Jupyter Notebook using Google's free Gemini API, offering students a quick and interactive way to query lecture material. The system supports contextual understanding of PDF notes, allowing meaningful answers to be generated dynamically.

2. Justification of LLM Choice

Google's Gemini API was chosen primarily due to its accessibility, modern architecture, and support for prompt-based interactions. Unlike other paid LLMs, Gemini provides a free tier with sufficient capabilities for academic use. It supports high token limits and multimodal models, though this project used the `gemini-2.0-flash` text model for generating answers based on lecture notes.

3. Justification of Development Approach

The chatbot follows a lightweight Retrieval-Augmented Generation (RAG) pattern:

- Lecture notes were loaded from a PDF using `PyMuPDFLoader`
- Text was split into manageable chunks
- A custom prompt was created combining lecture context and the user's question
- The prompt was sent to Gemini using the `generateContent` API

This approach avoids the need for vector databases while remaining effective for medium-sized documents.

4. Challenges and Lessons Learned

Key challenges included:

- Gemini's REST endpoint limits and model compatibility issues with some libraries (like LangChain)
- Character limits when injecting long lecture content into prompts
- Ensuring answers included relevant page numbers

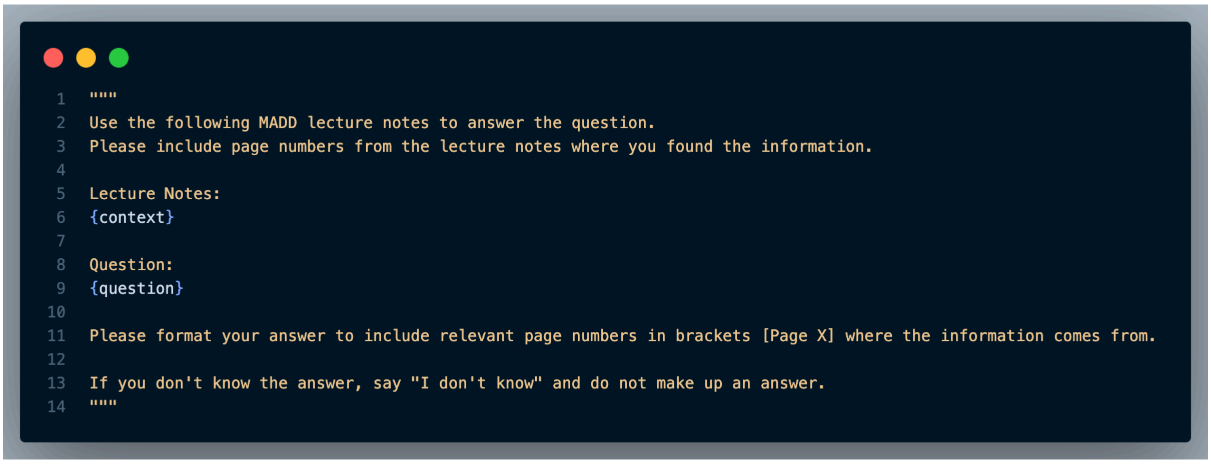
Lessons learned include prompt formatting strategies, efficient document chunking, and the benefit of separating model configuration from document handling.

5. Use of GenAI Tools

GenAI tools were used as follows:

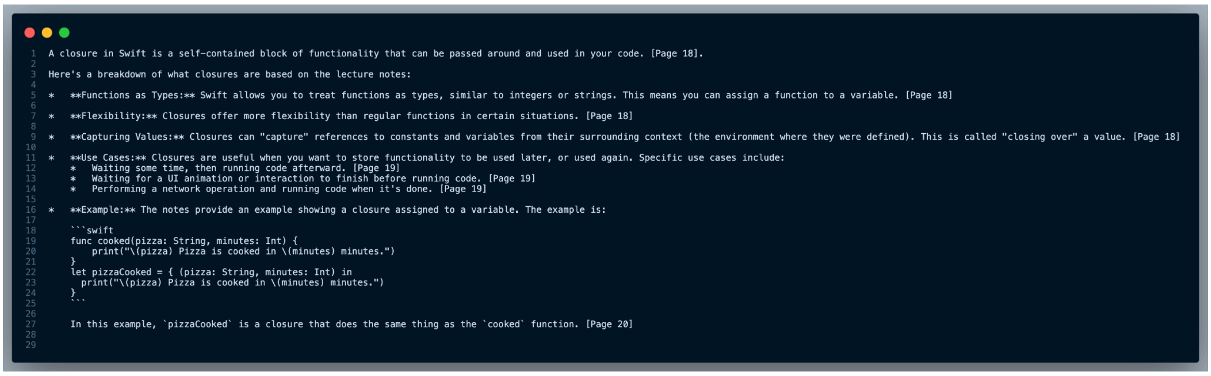
- **Prompt:** Combined lecture content and user question

Sample Prompt:



```
"""
Use the following MADD lecture notes to answer the question.
Please include page numbers from the lecture notes where you found the information.

Lecture Notes:
{context}

Question:
{question}

Please format your answer to include relevant page numbers in brackets [Page X] where the information comes from.

If you don't know the answer, say "I don't know" and do not make up an answer.
"""
```

Output Example:



~~~
A closure in Swift is a self-contained block of functionality that can be passed around and used in your code. [Page 18].

Here's a breakdown of what closures are based on the lecture notes:

*   **Functions as Types:** Swift allows you to treat functions as types, similar to integers or strings. This means you can assign a function to a variable. [Page 18]

*   **Flexibility:** Closures offer more flexibility than regular functions in certain situations. [Page 18]

*   **Capturing Values:** Closures can "capture" references to constants and variables from their surrounding context (the environment where they were defined). This is called "closing over" a value. [Page 18]

*   **Use Cases:** Closures are useful when you want to store functionality to be used later, or used again. Specific use cases include:
    *   Waiting some time, then running code afterward. [Page 19]
    *   Waiting for a UI animation or interaction to finish before running code. [Page 19]
    *   Performing a network operation and running code when it's done. [Page 19]

*   **Example:** The notes provide an example showing a closure assigned to a variable. The example is:

    ```swift
    func cooked(pizza: String, minutes: Int) {
        print("\(pizza) Pizza is cooked in \(minutes) minutes.")
    }
    let pizzaCooked = { (pizza: String, minutes: Int) in
      print("\(pizza) Pizza is cooked in \(minutes) minutes.")
    }
    ```

    In this example, `pizzaCooked` is a closure that does the same thing as the `cooked` function. [Page 20]
~~~

All responses were generated via Gemini's `generateContent` API using these structured prompts.

6. Conclusion

The project successfully demonstrates how a minimal setup with Gemini can support educational chatbot functionality. It provides accurate, context-aware answers based on structured lecture content. The approach is cost-effective, easy to maintain, and adaptable for similar academic use cases.

7. References

- Google Generative AI API Documentation: https://ai.google.dev/

- LangChain Documentation: https://python.langchain.com/
- PyMuPDF for PDF loading: https://pymupdf.readthedocs.io/