Project Assignment Document

Project Title

StudyBuddy — Personal RAG Learning Assistant

1. Project Overview

StudyBuddy is a Retrieval-Augmented Generation (RAG) based learning assistant that allows students to upload their study materials (PDF or text files) and ask questions based strictly on those materials.

The system must:

- Extract document content
- Split into structured chunks
- Generate embeddings
- Store embeddings in a local vector database
- Retrieve relevant context
- Generate answers grounded only in retrieved content
- Generate summaries and flashcards

This project must be completed in **3 working days (8 hours/day = 24 total hours).**

2. Objectives

The assigned developer must build a fully functional RAG pipeline i, including:

1. Document ingestion
2. Embeddings generation
3. Vector database storage
4. Retrieval system
5. Context-grounded QA
6. Document summarization
7. Flashcard generation
8. Simple UI (Streamlit preferred)

3. Technical Requirements

Programming Language:

- Python 3.10+

Recommended Libraries:

- LangChain (optional but preferred)
- FAISS (local vector store)
- PyPDF
- Streamlit
- Pandas
- python-dotenv

LLM:

- OpenAI / OpenRouter / equivalent
- Temperature: 0–0.2

4. Required Features

4.1 Document Upload & Ingestion

The system must:

- Accept PDF or TXT files
- Extract text
- Split into chunks (500–800 tokens, overlap 100)
- Store metadata:
    - filename
    - page number
    - chunk index

Acceptance:

- Chunking works correctly
- Metadata stored for each chunk

4.2 Embeddings & Vector Storage

The system must:

- Convert chunks to embeddings
- Store embeddings + metadata in FAISS
- Save index locally
- Implement retrieval function (top-K search)

Acceptance:

- Query returns relevant chunks
- Retrieval tested independently

4.3 Retrieval + Question Answering (RAG)

The system must:

- Embed user question
- Retrieve top 3–5 chunks
- Send context + question to LLM
- Generate a grounded answer

Strict Requirement: If the answer is not found in the retrieved context, the system must reply:

> "Not found in document."

Acceptance:

- Answers are based only on the uploaded material
- Hallucination minimized

4.4 Document Summary

The system must:

- Generate a concise summary
- Use only document content

- Support full document summary

Acceptance:

- Summary is accurate and grounded

4.5 Flashcard Generator

The system must:

- Generate Q/A flashcards from the document
- Return structured output
- Export flashcards as CSV
- Compatible with Anki

Acceptance:

- CSV correctly formatted
- Minimum 10 flashcards generated

5. Optional (Bonus – Required for Full Marks)

- Show citations (page numbers/chunk index)
- Streamlit UI
- File filters
- Export summaries

6. Project Structure (Mandatory)

```
studybuddy/

├── README.md

├── PROJECT.md

├── requirements.txt

├── .env.example
├── data/

├── ingest.py

├── embeddings.py

├── vectorstore.py

├── rag.py

├── app.py

└── utils.py
```

Structure must be clean and modular.

7. Timeline (3 Days – 24 Hours Total)

Day 1 – Backend Core (8 Hours)

- Setup project
- Implement ingestion
- Implement chunking
- Implement embeddings
- Implement vector storage
- Test retrieval independently

Day 2 – RAG + Features (8 Hours)

- Implement QA pipeline
- Add grounding enforcement
- Implement the summary feature
- Implement flashcard generation
- Add citation logic

Day 3 – UI + Documentation (8 Hours)

- Build a Streamlit interface
- Add upload functionality
- Add question interface
- Add summary + flashcard buttons

- Implement CSV export
- Testing
- Write README
- Write PROJECT.md
- Write a reflection document
- Capture demo screenshots

8. Deliverables

The developer must submit:

1. GitHub repository (or ZIP)
2. Working application
3. requirements.txt
4. README.md (clear setup + usage)
5. PROJECT.md (technical explanation)
6. Reflection document (1 page)
7. Demo (screenshots or short video)

9. Acceptance Criteria

Basic Completion (60%)

- Upload works
- Chunking correct
- Retrieval works
- QA grounded
- Summary works
- Flashcards export works

Code Quality (20%)

- Modular structure
- Clean functions
- Clear comments
- No hardcoded secrets
- Proper .env usage

Bonus (20%)

- UI implemented
- Citations displayed
- Export options included

10. Quality Expectations

- No unnecessary duplication
- Functions clearly separated
- Meaningful variable names
- Proper error handling
- No hardcoded API keys
- Clear commit history

11. Risks & Constraints

- The developer must test the retrieval before LLM integration
- Chunk size must be tuned properly
- Ensure the temperature is low
- Prevent hallucination

12. Definition of Done

The project is considered complete when:

- End-to-end flow works: Upload → Ingest → Ask Question → Answer → Summary → Flashcards
- All required files exist
- Demo shows functionality
- Code is clean and documented
- No runtime errors