Project Document: FinBot - An RAG-Powered Financial Literacy Chatbot

1. Project Proposal

1.1 Project Title

FinBot: A Trustworthy Financial Literacy Chatbot with Retrieval-Augmented Generation (RAG)

1.2 Problem Statement

Financial literacy is a critical skill for economic well-being, yet a significant portion of the population finds financial concepts (e.g., investing, mortgages, taxes, and retirement plans) to be intimidating and complex.

While people often turn to the internet, they face two major problems:

1. **Information Overload:** It's difficult to find simple, direct answers from a sea of dense articles.
2. **Misinformation:** Generic Large Language Models (LLMs) can "hallucinate" and provide incorrect, outdated, or even dangerous financial advice.

This creates a high-risk environment for individuals seeking to improve their financial knowledge.

1.3 Proposed Solution

We propose **FinBot**, a specialized conversational AI designed to provide **safe, accurate, and accessible** financial education.

The core of FinBot is a **Retrieval-Augmented Generation (RAG)** architecture. Unlike a standard chatbot, FinBot does not rely on its internal (and potentially flawed) knowledge. Instead, it follows a strict process:

1. **Listen:** The user asks a question (e.g., "What is the difference between a 401k and an IRA?").
2. **Retrieve:** FinBot instantly searches a pre-approved, curated **Knowledge Base** of trusted financial documents (from sources like the Consumer Financial Protection Bureau and Investopedia).
3. **Augment:** It "retrieves" the most relevant, factual passages from this knowledge base.
4. **Generate:** It feeds these passages—along with the user's question—to a generative AI with a single instruction: "**Answer the user's question *only* using the provided text.**"

This "grounding" technique ensures that the bot's answers are factual, verifiable, and free from the risk of hallucination.

1.4 Impact and Innovation

- **Impact (Financial Inclusion):** FinBot democratizes financial knowledge by providing a non-judgmental, easy-to-use tool for anyone to ask "basic" questions without feeling intimidated. It acts as a patient, 24/7 financial tutor.
- **Innovation (Trust & Safety):** The true innovation is not the chat interface, but its **trustworthiness**. By restricting the AI to a "walled garden" of factual data, we solve the primary problem of LLM-based misinformation, making it a reliable tool for an education-focused domain.

1.5 Technical Stack

- **Programming Language:** Python
- **Core AI Framework:** LangChain or LlamaIndex (for building the RAG pipeline)
- **LLM Model:** An open-source model (e.g., **Gemma**, **Llama 3 8B**, or **Mistral 7B**) for generation.
- **Embedding Model:** A sentence-transformer model (e.g., `all-MiniLM-L6-v2`) to convert text into vector embeddings.
- **Vector Database: FAISS** or **ChromaDB** (fast, free, and runs locally) to store and query the document embeddings.
- **Frontend: Streamlit** (to build a simple, interactive web app demo in pure Python).

---

2. Data Sources

The RAG system is composed of two main data parts:

1. **The Knowledge Base (For Retrieval):** This is the corpus of documents that FinBot will "read" to find answers. This is the most critical component. We will create it by scraping and cleaning data from highly reputable sources:
   - **Consumer Financial Protection Bureau (CFPB):** The CFPB's "Consumer Tools" and "Financial Education" sections are written in plain English and are an ideal, authoritative source.
   - **Investopedia.com:** We will focus on the core "Terms" and "Guides" sections, which provide clear definitions of thousands of financial concepts.
   - **Investor.gov:** An official U.S. government site from the SEC, designed to educate investors.
2. **The Q&A Dataset (For Evaluation):** This dataset will not be *in* the knowledge base. Instead, we will use it to *test* our completed FinBot to see if it gives correct answers.
   - **FiQA (Financial Question Answering) Dataset:** This is the perfect public dataset for this. It contains thousands of financial questions paired with expert-written answers. We will use the questions from FiQA as "prompts" to evaluate our bot's accuracy.

---

3. Step-by-Step Methodology

This methodology breaks the project into a logical flow, perfect for a competition.

Phase 1: Data Curation & Preprocessing (The "Knowledge Base")

1. **Collect Data:** Write simple Python scripts using libraries like `requests` and `BeautifulSoup` to scrape the text from the "Data Sources" listed above.
2. **Clean Data:** Process the raw HTML to extract clean text. Remove all navigation bars, advertisements, footers, and JavaScript code, leaving only the paragraphs of an article.
3. **Chunk Data:** This is a key step for RAG. A single article is too large to fit into an AI's context window. We will split the clean text into small, overlapping **chunks** (e.g., 500 characters per chunk with a 100-character overlap) using LangChain's `RecursiveCharacterTextSplitter`. Each chunk is a potential "fact" to be retrieved.

Phase 2: Indexing (The "Retrieval" Engine)

1. **Select Embedding Model:** We will use the `all-MiniLM-L6-v2` model from the `sentence-transformers` library. This model is small, fast, and excellent at understanding the semantic meaning of text.
2. **Generate Embeddings:** We will feed every single text chunk from Phase 1 through this model to convert it into a numerical vector (an "embedding"). This vector represents the chunk's *meaning*.
3. **Create Vector Store:** We will use **FAISS** (Facebook AI Similarity Search) to create a vector database. We will load all (embedding, text chunk) pairs into this database.
4. **Save Index:** We will save this completed FAISS index to disk. This is FinBot's "brain" or "long-term memory."

Phase 3: Building the RAG Pipeline (The "Generation" Engine)

1. **Load the LLM:** We will load a pre-trained, instruction-tuned LLM (like `Gemma-2B-it` or `Mistral-7B`).

2. **Design the Master Prompt:** This is the *most important* part of the project. We will create a **prompt template** that forces the LLM to be factual.

   > **Prompt Template:**
   >
   > "You are a helpful and neutral financial assistant. Answer the user's question *only* based on the context provided below. Do not use any outside information. If the answer is not in the context, state that you do not have enough information to answer.
   >
   > **CONTEXT:** {context}
   >
   > **QUESTION:** {question}
   >
   > **ANSWER:**"

3. **Create the RAG Chain:** Using LangChain, we will tie everything together into a single chain that executes the full RAG process:

   - **Input:** User's question.
   - **Step 1:** The question is sent to the embedding model (from Phase 2) to create a query vector.
   - **Step 2:** The FAISS index is queried with this vector to find the top 4 most semantically similar text chunks.
   - **Step 3:** These 4 chunks are "stuffed" into the `{context}` section of our Master Prompt.
   - **Step 4:** The user's question is put into the `{question}` section.
   - **Step 5:** This completed prompt is sent to the LLM.
   - **Output:** The LLM's generated answer, which is (by instruction) based only on the retrieved facts.

Phase 4: Interface & Evaluation

1. **Build UI:** Use **Streamlit** to create a simple chat interface. This involves just a few lines of code to create a text input box and a display area for the conversation.
2. **Qualitative Evaluation:** We will perform manual "Red Teaming." We will ask it questions we know are in the database, questions we know are *not* in the database (to ensure it refuses to answer), and "trick" questions (e.g., "Should I buy Tesla stock?"—it should refuse to give advice).
3. **Quantitative Evaluation (Time Permitting):** We will use the questions from the **FiQA dataset** as inputs and have a human judge (or another LLM) score the quality and accuracy of the generated

answers.

---

4. Work Plan (Example for a 1-Week Competition)

This plan is aggressive but feasible.

- **Day 1: Setup & Data Curation**
    - **Morning:** Project setup (Git repo, Python environment, install libraries: `langchain`, `streamlit`, `faiss-cpu`, `sentence-transformers`, `beautifulsoup4`).
    - **Afternoon:** Write and run scraping scripts for CFPB and Investopedia.
    - **Evening:** Begin writing the data cleaning and chunking scripts.
- **Day 2: Indexing & Vector Store**
    - **Morning:** Finish cleaning and chunking all scraped text.
    - **Afternoon:** Run the embedding model over all chunks and build the FAISS vector store.
    - **Evening:** Save the final index to disk. *Milestone: Knowledge Base is complete.*
- **Day 3: Core RAG Pipeline**
    - **Morning:** Download the LLM (e.g., Gemma) and test it standalone.
    - **Afternoon:** Build the core LangChain RAG chain (linking the prompt, LLM, and retriever).
    - **Evening:** Fine-tune the Master Prompt Template. This is critical—test it repeatedly.
- **Day 4: Frontend Development**
    - **Morning:** Build the Streamlit chat interface.
    - **Afternoon:** Connect the Streamlit UI to the RAG chain backend.
    - **Evening:** End-to-end testing and debugging. *Milestone: Functional Prototype is complete.*
- **Day 5: Evaluation & Refinement**
    - **Morning:** "Red Team" the bot. Ask it dozens of questions (in-domain, out-of-domain, advice-seeking).
    - **Afternoon:** Log all failures and edge cases.
    - **Evening:** Tweak the prompt or retrieval settings to fix the most common errors.
- **Day 6: Presentation Prep**
    - **Morning:** Build the slide deck (Problem, Solution, Architecture Diagram, Demo, Impact).
    - **Afternoon:** Record a video of the demo (critically important backup in case of live-demo failure).
    - **Evening:** Write the presentation script.
- **Day 7: Final Polish & Pitch**
    - **Morning:** Final code cleanup and commenting.
    - **Afternoon:** Full presentation dry-run.
    - **Evening: Compete!**