Building an Intelligent QA/Chatbot with LLMs

Team 3

Submitted to: Professor Mohammed Farag

CS4624: Multimedia, Hypertext, Information Access

Patrick Cross, Sean Scott, Mikail Syed, Aditya Singh, Maokun Zhang

Client: Dr. Mohammed Farag

December 2024

Contents

1 Abstract

This project developed a web-application Q/A chatbot that enables users to interact with Large Language models (LLMs) through a collection format. The system implemented a Retrieval Augmented Generation (RAG) pipeline to provide context-specific responses based on either user-uploaded documents (.txt, .html, and .zip formats) or user uploaded URLs. The appliaction features secure user authentication, multiple-instances of chat/document contexts through collections, document upload, and standard LLM chatbot functionalities, including the ability to switch between LLMs.

This report will give readers an understanding of how the application was designed and developed; how to install and use the application; how to continue development of the application; lessons learned during development; and future plans for the project.

Github:
LangchainRAGRepository

2 Introduction

2.1 Problem Statement

In our increasingly connected world, crisis events - from natural disasters to man-made emergencies - generate massive amounts of information across multiple platforms and sources. This information overload presents several critical challenges:

2.1.1 Information Processing Challenges

During crisis events, stakeholders face numerous obstacles in processing and understanding available information:

- Volume: The sheer quantity of information from news sources, social media, official reports, and other channels can be overwhelming
- Velocity: Information updates occur rapidly, making it difficult to track and process new developments

2.2 Existing Solutions and Their Limitations

2.2.1 Manual Search Limitations

Manual searching through documentation presents significant inefficiencies:

- Time-consuming process of finding specific information across multiple documents
- Difficulty extracting relevant details from large volumes of text. It i easy to accidentally look over critical pieces of information

2.2.2 Current AI Solution Limitations

General-purpose AI systems also have notable shortcomings:

- File uploading is often unavailable. When it is, user's are unable to upload a large set of documents and URLs for analysis all at once.
- Web search systems pose a risk of introducing irrelevant misinformation from highly unvetted sources
- General AI systems will often rely on outdated training data from months or years ago.

2.3 Project Goals and Innovation

Our project addresses these limitations through several development decisions. For more information on this, look in section 3 [Requirements] and section 4 [General Design]

2.3.1 Our Solution

Our system addresses these limitations by allowing users to create their own knowledge bases:

- Users can bulk upload documents and URLs to create custom collections
- RAG architecture ensures responses are drawn directly from user-provided documents
- Multiple collections enable separate contexts for different topics or projects

2.3.2 Technical Implementation

The system is built using:

- Django web framework for the application backend
- LangChain for LLM integration and RAG implementation
- ChromaDB for vector storage and efficient document retrieval
- Local deployment options for enhanced privacy and control

2.4 Application Domains

While initially focused on crisis event information management, the system's architecture supports broader applications:

- Emergency response and disaster management
- Corporate knowledge base management
- Educational resource navigation
- Research document analysis
- Legal document processing
- Technical documentation support

2.5 Report Organization

This report is structured to serve both as comprehensive project documentation and as a guide for future development:

- **Requirements** (Section 3): Details business requirements, functional specifications, and technical requirements for the system.
- **General Design** (Section 4): Documents system architecture, including frontend, backend, and database layers.
- **Tooling and Motivations** (Section 5): Details technology choices and implementation specifics, including RAG architecture and deployment.
- **Testing/Evaluation** (Section 6): Covers testing methodology for RAG pipeline, document processing, and user interface.
- **User's Manual** (Section 7): Comprehensive guide for end users, including account management and core functionality.
- **Developer's Manual** (Section 8): Technical documentation covering setup, architecture, and continuing development guidelines.
- **Lessons Learned** (Section 9): Insights from development process, including technical challenges and project management considerations.
- **Timeline** (Section 10): Details timeline of work completed.
- **Acknowledgments and References** (Sections 11-12): Credits and documentation of resources used.

3 Requirements

3.1 1. Project Scope Requirements

- **RAG architecture:** Deliver a QA/chat bot which pulls information from provided User Documents. These documents must be stored and accessed in a RAG type infrastructure. It must be able to support many many total documents. Types include
  - txt files
  - html files
  - zip files of txt and html
  - list of URLs
- **Pipeline Development:** Establish a pipeline with the following core components. A prompt retriever, LLM, memory module, and conversational chatbot.
- **Collections & Retrieval:** Users must be able to create, view, and manage document collections to tailor how the AI responds for particular crises event's and use cases. These collections must be persistent, and must retain separate context spaces.

3.2 2. Technical Specifications

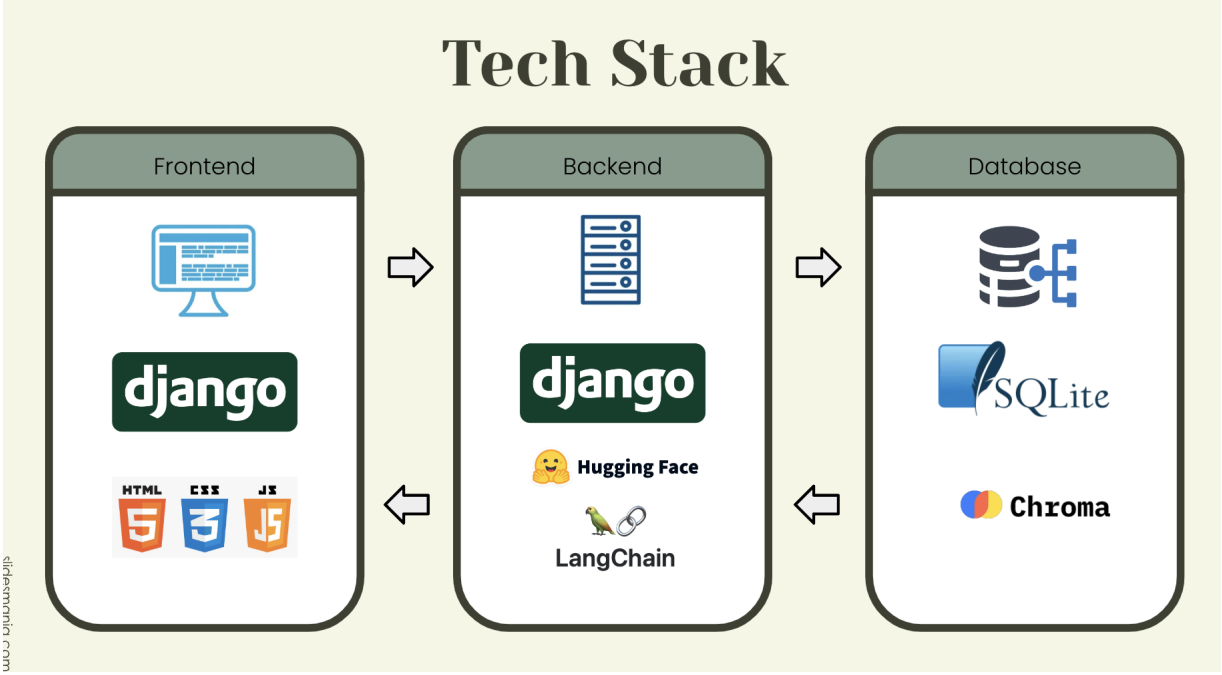
- **LangChain for LLM Management:** The QA/chatbot should use LangChain's various modules for prompt templates, tokenizers, vector stores (ChromaDB), and retrievers.
- **Embedding Model & Vector Store:** Select an appropriate embedding model that can efficiently work with ChromaDB for retrieval.
- **Local Model Hosting:** Open-source LLMs (e.g., Zypher Quantized) should be loaded and run locally rather than relying on cloud-based solutions, considering performance on available hardware (CPU/GPU).
- **Database Requirements:** Use ChromaDB for the vector database and MySQL for storing regular data, like chat histories.
- **Frontend Stack:** Django is the preferred framework for developing the web app, aligning with the full stack requirements of the project.

3.3 3. Client Expectations

- **Structured Timeline with Deliverables:** Adherence to milestones for deliverables, such as initial MVP completion, RAG pipeline development, and finalized documentation, is crucial. Each stage (e.g., RAG pipeline, collections, UI) should reflect progress aligned with the client's timeline.
- **Documentation & Presentation:** Each deliverable phase should include detailed documentation (source code, scripts, manuals) and a presentation update to reflect on current progress.
- **Final Deliverables:** Source code for the web app, preprocessing scripts, and training/testing models must be handed over along with comprehensive user and developer documentation.
- **Learning Outcomes:** The project is also an educational experience, so knowledge and understanding of NLP, LangChain, and LLM development are implicit deliverables.

4 General Design

The intelligent QA bot system is structured as a three-tier architecture, which ensures a clean separation of responsibilities and scalability. This architecture comprises the Frontend Layer, Backend Layer, and Database Layer, each fulfilling a unique role in delivering seamless user interactions and efficient document-based question-answering.



4.1 Frontend Layer:

The frontend is designed to be the primary interface through which users interact with the system. It provides intuitive functionalities such as user registration, login, file upload, and a chat interface for conducting document-based queries. Using responsive design principles, the frontend ensures a consistent and adaptable experience across devices. By serving as a bridge between users and the system's logic, the frontend layer captures inputs that are critical to the backend's operations, such as document uploads and user-submitted questions. This layer focuses on user experience, prioritizing ease of navigation, real-time feedback on processing status, and responsiveness to different screen sizes.

4.2 Backend Layer:

The backend layer is the core of the system's logic, connecting the frontend interface with the database and language model (LLM). This layer manages all data processing workflows, ensuring that uploaded

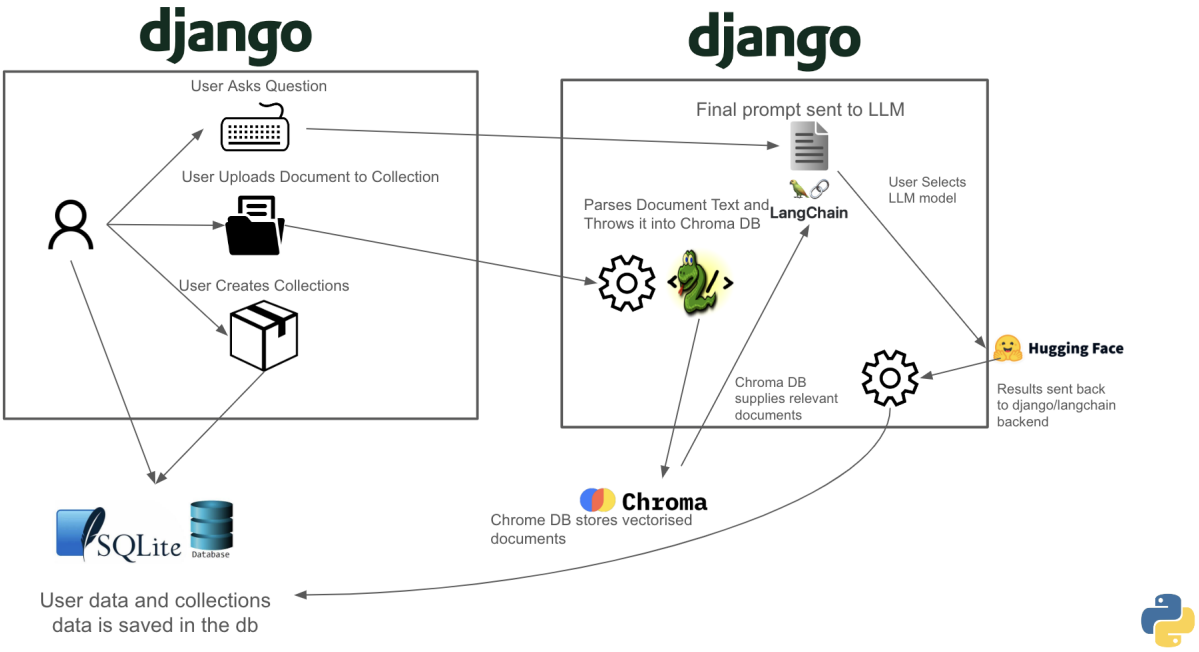
documents are securely stored, indexed, and processed to generate meaningful responses. It coordinates tasks like document vectorization, prompt management, and query analysis. In addition, the backend leverages advanced prompt management tools to integrate the language model for generating accurate and contextually relevant responses. This layer functions as the processing hub, streamlining the flow of data between the user-facing frontend and the structured and unstructured data in the database.

4.3 Database Layer:

The database layer is responsible for storing and managing user data, uploaded documents, and vectorized document embeddings. This includes both relational data, such as user information and collection metadata, and vector embeddings, which support semantic search for relevant content within uploaded files. The database is essential for efficiently retrieving and contextualizing information, as it organizes document embeddings that can be queried and retrieved during user interactions. The system's use of vector embeddings enables high-speed access to relevant document sections, allowing the LLM to answer questions accurately based on the document context.

4.4 Data Flow Architecture

The data flow within the system is designed to support efficient document processing and question-answering. The architecture facilitates seamless communication across layers, maintaining a structured and optimized workflow.



- **User Interaction Flow:**

  Users interact with the system by uploading documents and submitting questions. The system organizes uploaded documents into collections, which are then vectorized and stored in a vector database. When a question is submitted, the backend retrieves relevant document chunks based on semantic similarity, constructs a prompt with context, and sends it to the language model to generate a response.

- **Backend Processing Pipeline:**

  This pipeline is responsible for handling document processing, query analysis, and response generation. Uploaded documents are parsed, vectorized, and stored in the vector database. When a query is submitted, the pipeline retrieves relevant document embeddings, assembles them into prompts, and generates responses using the LLM.

5 Tooling and Motivations

The intelligent QA bot system was developed using a range of tools chosen for their ability to meet specific requirements related to scalability, performance, and ease of integration. This section discusses the tools used and the motivations behind each choice, illustrating how each tool contributes to the overall system functionality.

5.1 Frontend

- **Tooling:**
  - Django: We chose Django as the frontend framework due to its robust templating engine, which supports server-side rendering and simplifies the handling of form submissions and user authentication. Django's built-in features allow for seamless session management and user data handling, enabling a secure and intuitive user interface.
- **Motivation:**

  Django's strong documentation and community support made it a practical choice for frontend development. Its MVC architecture aligns well with our system's needs, allowing us to easily manage the user-facing interface while integrating with backend services. Furthermore, Django's flexibility in rendering HTML templates enabled us to design a responsive interface that works effectively across desktop and mobile devices.

5.2 Database

- **Tooling:**
  - Django: The backend framework is also built on Django, leveraging its modular and scalable design for handling HTTP requests, URL routing, and database connections. Django's ORM simplifies data manipulation and enables efficient CRUD operations.
  - LangChain: To manage interactions with the LLM, LangChain was incorporated as an orchestration tool for constructing dynamic prompts. LangChain allows us to handle complex, document-based queries and ensures that the LLM receives relevant context for generating responses.

  - Hugging Face: We deployed open-source LLM models through Hugging Face, opting for local deployment to enhance privacy, control, and latency. By hosting the models locally, we retain flexibility in tuning and adapting the LLMs, ensuring they align with our specific QA needs.

- **Motivation:**

  This combination of Django, LangChain, and Hugging Face supports the system's need for scalable backend logic and sophisticated LLM integration. Django provides stability and ease of development, while LangChain and Hugging Face enable advanced prompt management and response generation in real-time, fostering accurate and efficient question-answering.

5.3 Backend

- **Tooling:**
  - SQLite: For relational data, we selected SQLite due to its simplicity and minimal configuration requirements. SQLite handles user data, collection metadata, and basic file organization, providing a lightweight solution for prototyping and testing.
  - ChromaDB: For vector storage, we employed ChromaDB, which is optimized for handling vector embeddings. ChromaDB allows the system to efficiently store and retrieve vectorized document content, enabling semantic search and contextualized responses to user queries.

- **Motivation:**

  SQLite's ease of use and low overhead made it ideal for managing relational data during development, while ChromaDB's capabilities in vector storage ensure high-performance retrieval for complex queries. This dual-database approach allows us to balance simplicity and efficiency in handling structured and unstructured data, respectively.

5.4 LLM Integration

The integration of large language models focuses on local deployment, which enhances privacy, control, and latency. Model selection was guided by the project's requirement for open-source LLMs capable of handling

document-specific question-answering tasks. Our local deployment setup allows for quick adjustments to models and configurations based on real-time user interactions, making the system adaptable and responsive.

5.5 RAG Implementation Details

The Retrieval-Augmented Generation (RAG) setup combines document retrieval, memory management, and response generation, ensuring that users receive answers with contextually relevant information from their uploaded documents. The query processor uses LangChain to parse user questions and select relevant document chunks stored in ChromaDB, allowing for accurate and specific responses.

5.6 Deployment Configuration

The deployment leverages Docker containers, which streamline environment consistency across development and production setups. Separate environments ensure stability in production while allowing continuous improvement in the development environment.

6 Testing/ Evaluation

This section provides a detailed overview of the testing methodologies used to ensure the functionality, reliability, and performance of key components in the Intelligent QA/Chatbot system. For each critical part of the project, we describe the specific testing approaches employed, tools used, and metrics evaluated.

6.1 RAG Pipeline Testing

The Retrieval-Augmented Generation (RAG) pipeline is essential to the chatbot's capability to provide accurate, context-aware responses. Testing this component involved evaluating various LLMs, prompt styles, and retrieval mechanisms to optimize accuracy, relevance, and consistency in responses.

- **LLM Comparison and Analysis:**
  - We tested multiple open-source LLMs to understand how each model performed in terms of response relevance and accuracy. Models were evaluated based on their ability to handle complex queries and deliver nuanced answers.
  - *Evaluation Criteria:* Accuracy, relevance, speed of response, and ability to understand context.
  - *Observations:* We noted the strengths and limitations of each model, focusing on how well each handled event-specific queries, particularly those in crisis scenarios.
  - *Decision:* From these tests, we decided on a set of models to use in our actual application. It was a balance of factors judgement call based on what we as users found to be the most seemless and usable experience.
- **Prompt Engineering:**
  - To optimize the quality of responses, we experimented with different prompt structures, including variations in language and phrasing. This helped to identify patterns that yielded the most accurate and contextually appropriate responses. We also implmented and tested various stop phrases, which upon utterance would end an AI's output.
  - *Testing Scenarios:* Each prompt variation was tested to see how phrasing affected response quality and relevance.

  - *Results and Patterns:* Certain prompts produced more concise and accurate responses, especially when explicit context clues were provided. We refined prompts based on these findings. We also found certain stop phrases produced more accurate results and reduced hallucinations.

- **Context and Memory Testing:**
  - Memory management was tested to ensure that the chatbot could maintain coherence in multi-turn conversations, especially for follow-up questions. This was critical to confirm that the system could retain and use previous information in generating subsequent responses.
  - *Testing Method:* We tracked memory accuracy by prompting the chatbot with follow-up questions and evaluated its ability to retain previous conversation context.

6.2 Document Processing and Vector Storage Testing

Document processing is fundamental to the chatbot's ability to understand and retrieve relevant information, especially when dealing with large datasets and various document formats. Testing focused on the accuracy and efficiency of text extraction, vectorization, and retrieval.

- **Format Compatibility Testing:**
  - We tested the system's ability to process and extract information from different file types (.txt, .html, .zip) and URL sources.
  - *Tools Used:* We employed validation scripts to check for data consistency and completeness after extraction.
  - *Outcomes:* Minor discrepancies in text extraction were observed for HTML files with heavy formatting. We optimized extraction functions to ensure consistency across file types.

- **Vector Storage and Retrieval Accuracy:**
  - To ensure efficient and accurate retrieval, we tested the system with various embedding models and vector storage configurations.
  - *Evaluation Metrics:* Semantic similarity, retrieval speed, and response accuracy.

- *Observations:* Embedding models that focused on semantic understanding (e.g., sentence transformers) provided more accurate and contextually relevant retrieval, particularly in multi-document scenarios.

6.3 Document Upload Functionality Testing

Ensuring smooth and reliable document upload is crucial for user experience and system functionality, as it enables users to add content to collections for accurate response generation. Testing in this area focused on verifying file compatibility, error handling, and upload efficiency.

- **File Compatibility and Validation:**
  - We tested the upload functionality with multiple file types, including .txt, .html, and .zip formats, to ensure the system correctly identified and processed each type.
  - *Testing Scenarios:* Files of varying sizes and content complexity were uploaded to confirm that text extraction and vectorization were handled consistently across all formats.
  - *Results:* The system successfully processed each supported file type, although minor adjustments were made to improve handling of large zip files.
- **Error Handling:**
  - We simulated various upload errors, such as uploading unsupported file formats, empty files, and corrupted files, to ensure the system could identify issues and provide users with meaningful feedback.
  - *Findings:* The error handling functioned as expected, displaying clear error messages when unsupported files or corrupt data were uploaded.
- **User Experience and Efficiency:**
  - We evaluated the upload experience for users by testing upload times for various file sizes and ensuring that the UI displayed progress indicators for larger uploads.
  - *Observations:* The system performed well with small- to medium-sized files, but larger files occasionally took longer than expected. To improve efficiency, asynchronous processing was implemented,

allowing users to continue interacting with the system during longer uploads.

7 Users' Manual

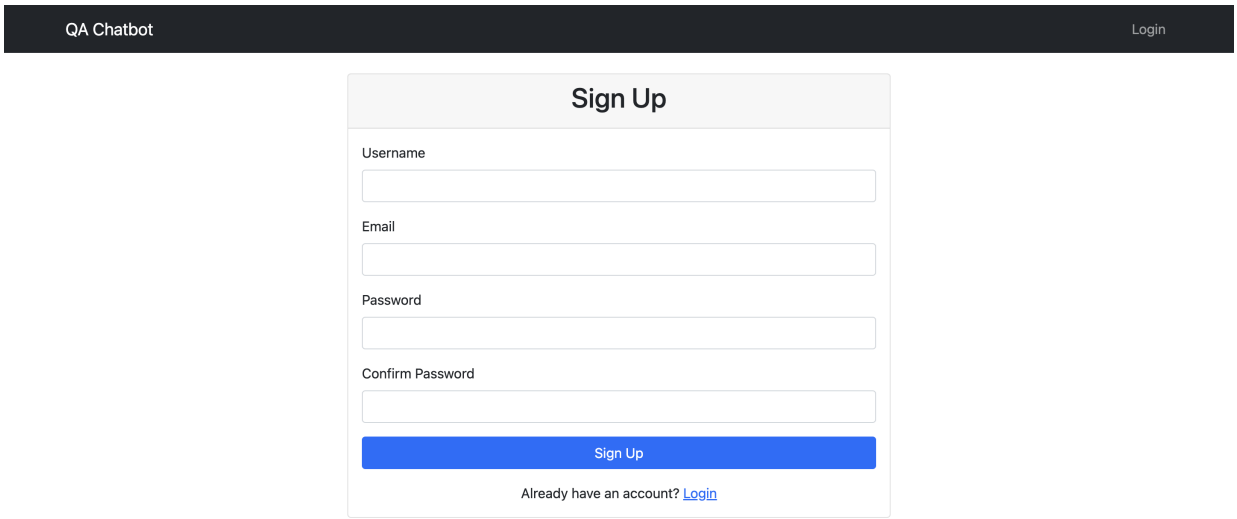
This section details how an end user can interact with the web application. It is split into two main sections: what to do when accessing the site for the first time, and then core functionality. Sections will be accompanied by screen dumps of the associated functions/actions/steps. (to be added)

7.1 Getting Started

7.1.1 Account Creation

When first accessing the application, users will need to create an account:

1. Navigate to the registration page via the ”Sign Up” button
2. Enter a valid email address
3. Create a secure password (minimum 8 characters, including numbers and special characters)
4. Confirm password
5. Click ”Create Account”

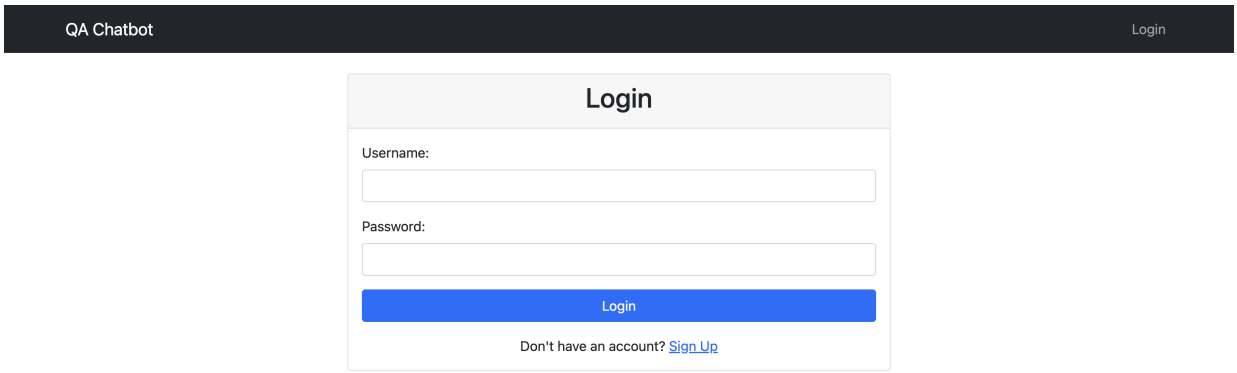


7.1.2 Log In

After account creation, users can log in:

1. Enter registered email address

2. Enter password
3. Click ”Log In”
4. For forgotten passwords, use the ”Reset Password” link

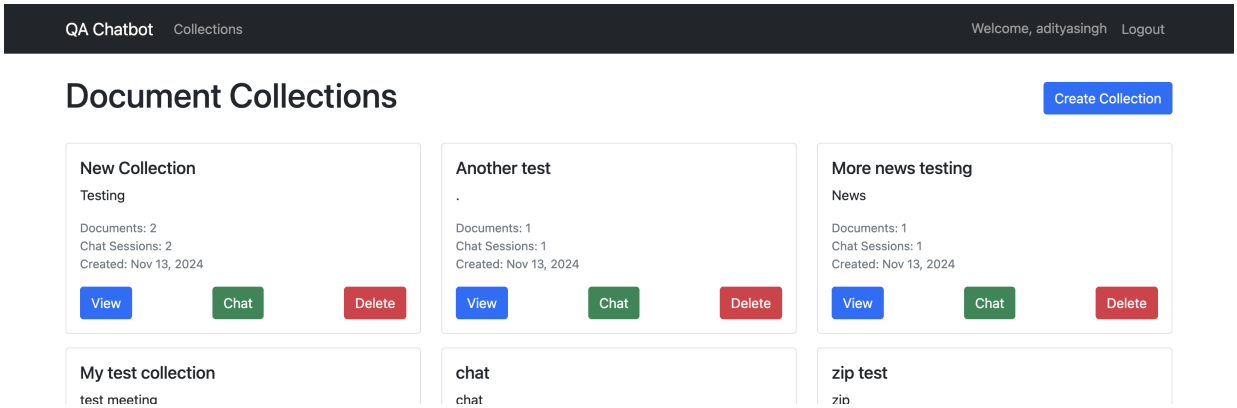


7.1.3 Interface Overview

Upon successful login, users will see the main dashboard:

Navigation Bar

- Collections - Access to all your document collections
- Create Collection - Create a new collection
- Logout - Ability to log out



Dashboard Layout

- Recent Collections - Quick access to recently used collections

- Collection Interactions - Overview of all ways of interacting with collection

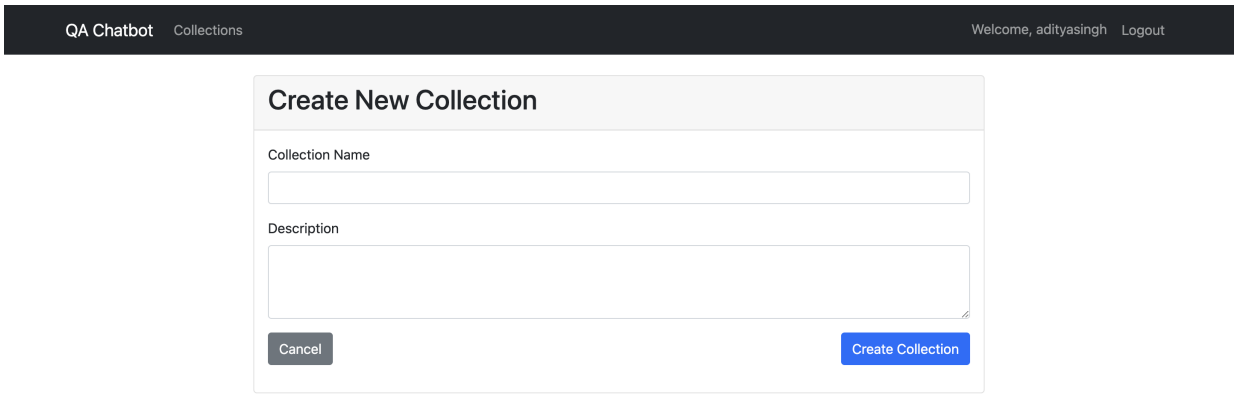
7.2 Core Functions

7.2.1 Managing Collections

Collections are the primary way to organize documents and conversations:

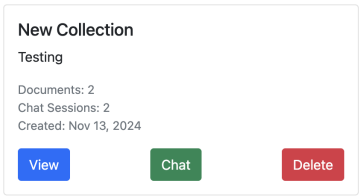
Creating a New Collection

1. Click ”New Collection” in the navigation bar
2. Enter collection name
3. (Optional) Add description
4. Click ”Create”



Viewing Collections

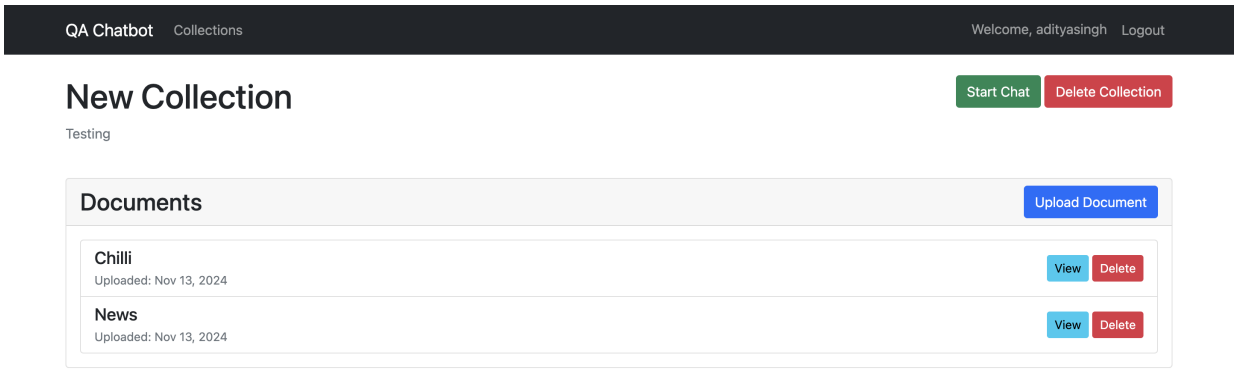
- Collections are displayed in a grid/list view
- Each collection shows:
  - Name
  - Creation date
  - Number of documents
  - Last accessed date



7.2.2 Document Management

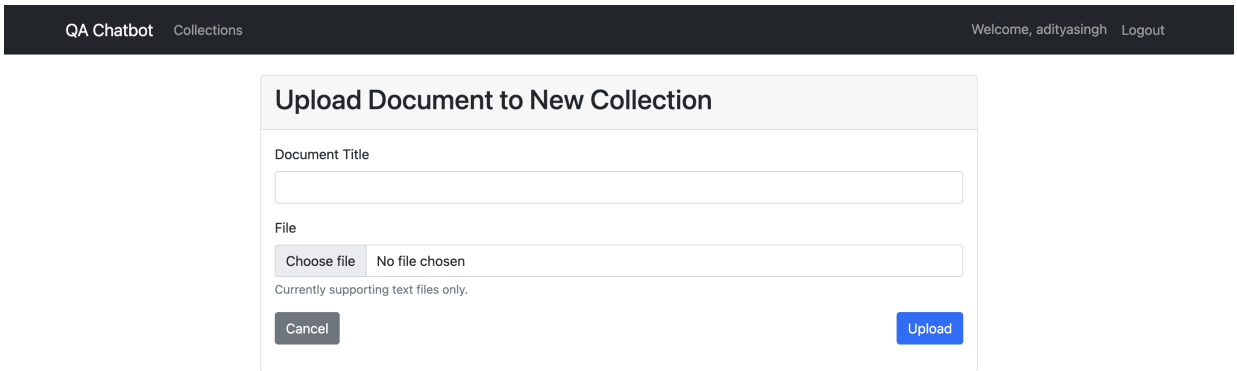
Supported File Types The system accepts the following formats:

- Text files (.txt)
- HTML documents (.html)
- ZIP archives containing text files
- URLs (processed and stored as text)



Upload Process

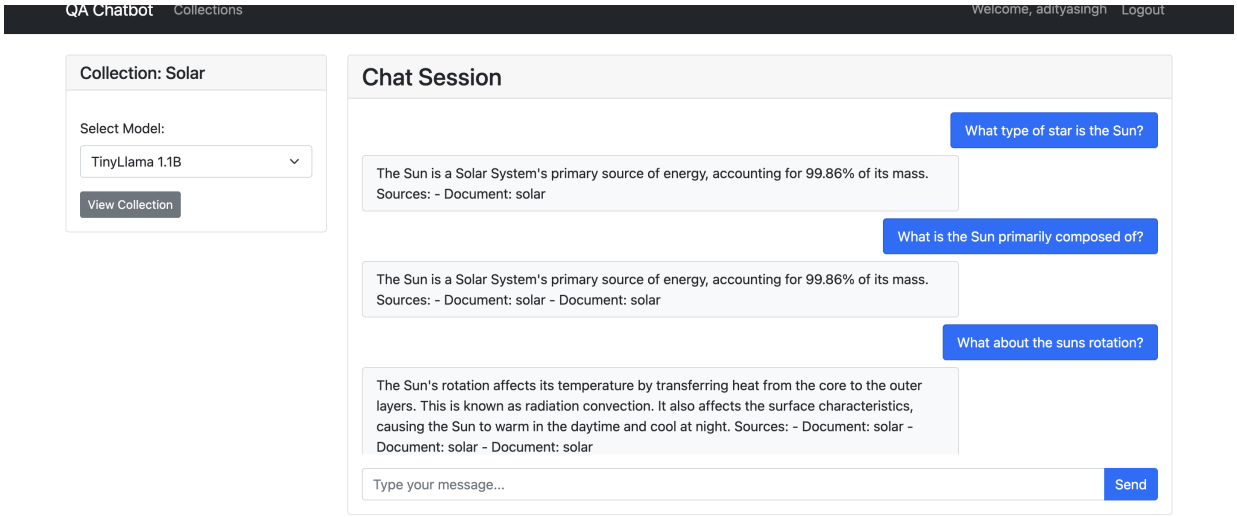
1. Select a collection
2. Click ”Upload Documents”
3. Choose upload method:
    - Single file upload
    - Multiple file upload
    - ZIP file upload
    - URL input
4. Wait for processing confirmation



7.2.3 Chat Interface

Starting a Chat

1. Select a collection from the dashboard
2. Click ”Start Chat” or ”Continue Chat”
3. The interface displays your chat history (if any) and question input field

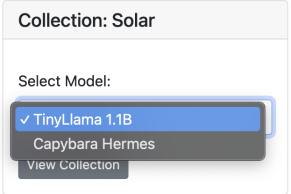


Using the Chat

- Type your question in the input field
- System responds with:
  - Answer based on collection context
  - Referenced source documents
- Previous conversations remain visible for context
- Switch collections or LLM models using the dropdown menus

Change LLM

- Be able to change which LLM model is used to generate answers



8 Developers Manual

This section accomplishes multiple goals. It will inform how to set up and install the application as a host user. It will detail the various program and data files contained in the application. And it will provide necessary details for a future team wishes to continue development.

8.1 Installation and Setup

8.1.1 System Requirements

We plan to add implementation for MacOS, Windows, and Linux systems. OS will absolutely be Linux. As several LLMs rely on the use of a GPU (typically CUDA), but utilize a CPU for lack of a better option, we will be assuming the same for our project; CUDA will be the preferred GPU device, but if all else, we will resort to CPU usage. This will be determined once we finalize local LLM options. Although we will want around 8 to 16 GB of RAM for local runs, if we wanted a larger scale operation we may need to upgrade to 16 to 32 GB of RAM.

8.1.2 Development Environment Setup

IDE Setup Use a Python-compatible IDE. Ex: PyCharm, Visual Studio Code

- Recommended settings: Python interpreter version 3.8+
- Install Python extensions for your IDE

Repository Access Pull, download, or fork the repository from: `https://github.com/ady2303/QA_Chatbot`

- Using git: git clone `https://github.com/ady2303/QA_Chatbot`

Directory Navigation Go to the QA_Chatbot directory in your terminal

- Windows: cd path/to/QA_Chatbot
- Linux/Mac: cd path/to/QA_Chatbot

Conda Enviroment

- It is highly recommended that you use a Conda environment for this project. Please download Miniconda from this: click here.
- Create a Conda Environment with Python 3.10.2 and activate that environment

Dependencies Installation Once you created your Conda environment run ”pip install -r requirements.txt”.

Installation Errors If you run into errors, here is a list of things to consider

- If you receive the error ”Failed building wheel for mysqlclient” you need to install necessary tools which can handle .c and .h file dependencies. Try these commands to install necessariy dependencies
  - Ubuntu/Debian: sudo apt-get install python3-dev default-libmysqlclient-dev build-essential pkg-config
  - CentOS/RHEL: sudo yum install python3-devel mysql-devel gcc
  - macbrew install mysql-connector-c
  - pip install wheel
  - pip install mysqlclient –only-binary :all:
- If you receive the error ”Failed building wheel for llama-cpp-python,” you may need to use a different building wheel for the python package. In this case, run the following commands:
  - Ubuntu/Debian: pip install ninja
    sudo apt-get update
    sudo apt-get install ninja-build
    pip install llama-cpp-python
  - CentOS/RHEL: sudo yum install epel-release
    sudo yum install ninja-build
    pip install llama-cpp-python
  - macOS: brew install ninja
    pip install llama-cpp-python or rerun pip install-r requirements.txt

Additional Dependencies Install any remaining dependencies

- MySQL client libraries
- Development tools and compilers

Server Deployment Run ”python manage.py runserver”

- Default address: http://127.0.0.1:8000/
- For network access: python manage.py runserver 0.0.0.0:8000

8.1.3 Application Deployment

On the server in order to deploy it we want to download the Github repository onto the server that you plan to deploy it on whether it is Google Cloud, Amazon Web Services, or tml.cs.vt.edu. When you do this, change directories to QA_Chatbot and run the following command in the terminal: gunicorn –bind tml.cs.vt.edu:3333 –timeout 120 config.wsgi:application. This example works if you are currently on tml.cs.vt.edu, but if you were to replace it with your own IP address and port, it will work the same way.

8.2 System Architecture and Files

8.2.1 Project Structure

The project is organized into a series of folders. This section details the purpose of each folder to help new developers familiarize themselves with the system's structure.

- **apps**: Contains individual Django applications, each responsible for specific functionalities within the project.
  - **chat**: Manages chat-related features, including models, views, and templates for user interactions.
  - **document_collections**: Handles collections of documents, providing organization and management capabilities.
  - **documents**: Manages individual documents, including their storage, retrieval, and processing.
- **config**: Houses configuration files and settings for the project, such as environment variables and deployment configurations.
- **core**: Contains core functionalities and services utilized across the project.
  - **llm**: Integrates Large Language Models (LLMs) to enhance natural language processing capabilities.
  - **memory**: Implements memory storage mechanisms to retain and recall information during interactions.
  - **vectorstore**: Manages vector storage and similarity search operations, essential for efficient information retrieval.
- **documents**: Stores static document files or serves as a repository for document-related data.

- **templates**: Contains HTML templates used to render the frontend of the application.
  - **chat**: Includes templates specific to chat interfaces and user interactions.
  - **document_collections**: Holds templates for displaying and managing document collections.
  - **documents**: Contains templates related to individual document views and operations.
  - **registration**: Manages templates for user registration and authentication processes.
- **vectorstore**: Handles the storage and management of vectorized data, facilitating efficient similarity searches and retrieval operations.
- **config**: Contains essential Django files such as settings.py, urls.py, wsgi.py, and asgi.py
- **llm_models**: Contains all information including the LLM Model usage
- **documents**: Contains all documents uploaded
- **my_rag.py**: RAG Pipeline Python File
- **manage.py**: File used to manage the Django project and run the project

8.2.2 Data Files

Migrations are written in init.py and 0001-initial.py in the apps/chat/migrations folder. Otherwise, databases are handled in the Django-native SQLite, which currently does not have a visual interface, but one will be made and required to be viewed. Langchain vector data is held inside of ChromaDB. The documents are to be organized according to their collection for simplicity of database access.

8.2.3 Core Components

- We have a fully functioning authentication system that manages user creation/signup, user login and user log out.
- Users can create a Collection, and they can also delete it.

- To each Collection, they can add Documents (as many as they'd wish) for the particular Collection they have added it to. They can delete Documents and add more as they wish.
- The project uses RAG to analyze the documents and feed it to the LangChain, where they can interact with a Chatbot and ask it questions about the Documents it has been fed.
- More information as to core components can be found in sections 3 and 4.

8.3 Known Issues and Limitations

One important issue is the relationship between the web interface and the terminal, as the RAG pipeline hogs the control of the web server and forces the application to run in the terminal. We were able to fix this eventually and complete the RAG Pipeline migration. Some issues with how the RAG Pipeline works may exist and I highly suggest anyone who takes on this project really think about how their RAG pipeline works so that it does not output nonsense with your prompts.

9 Lessons Learned

Throughout the development of this project, the team encountered several challenges and learning opportunities that shaped our approach and understanding of working with LLMs, vector databases, and full-stack development. This section summarizes these lessons, organized into key areas for reflection and improvement.

9.1 1. Technical Challenges and Solutions

- **Efficient Data Processing:** Processing large datasets for vector storage and retrieval was initially slower than expected. Optimizing chunk sizes for document splitting and selecting a high-performance embedding model were key improvements that enhanced the retrieval pipeline's efficiency. To make this possible, research on common machine learning optimization techniques had to be done, and some additional research on different LLM models and their functionality was especially crucial to the development and the resolution of the inefficiency in the data processing. However, this was solved in a timely and workmanlike manner, which meant that while this was an unexpected challenge, we fortunately had the time and expertise to resolve the issue.
- **Model Selection and Quantization:** Finding models that balanced accuracy with processing speed led us to explore quantized models found on HuggingFace and other platforms. This wasn't significant in terms of challenge, but time was especially necessary. There are an abundance of models that can suit the purpose of the project, but finding the right one meant searching for a model, researching said model, and experimenting with the model to determine which LLM model best fit our needs for the project, and whether it could deliver a worthwhile output. Because of this, while intellectual challenge was of no concern, time and tediousness had to play a role and can be identified as our maximum cost, and could abstractly generate the question of whether or not there is a computational method to efficiently find a model for our needs, or whether an artificial intelligence could help us detect a potential model without needing to exhaust ourselves to a tedious process such as this.
- **Memory Management in Retrieval Augmented Generation (RAG):** Integrating memory to handle follow-up questions was more com-

plex than anticipated. Comparing memory types in LangChain provided insights into achieving coherent multi-turn conversations and allowed us to better meet the client's expectations for continuity in QA. Since memory was also a foundational part of computer programming, and a lack of knowledge regarding memory was equivalent to a lack of knowledge regarding computing in general, it was increasingly important to understand how memory had worked in the space of LangChain. Doing some extra research and reading up on the important memory types moved the needle forward towards not only understanding our program at a high level, but also what is going on at a lower level, which gave us significantly more free reign on how to operate the RAG and additionally helped us contextualize any further issues that may have occurred in the program. Therefore, understanding the memory management was not only crucial for the program to move forward in this project and comply with the professor's suggestions, we also needed to know this conceptually in order to gain the skill for our benefit as it is an important, relevant, and yet highly demanded skill. Thus, like all challenges in this project, this ended up serving us with much-needed gifts on how to advance further.

9.2 2. Project Management Insights

- **Time Management and Milestones:** Setting and adhering to project milestones was crucial for timely progress. The structured timeline enabled us to stay on track, but unexpected technical challenges occasionally required adjustments. Allocating time for unplanned testing and debugging proved essential. Spacing out the time effectively was not terribly difficult to do, but allotting that much time in case of slip-up (and slip-up did indeed occur) was crucial to our final success in the project. What was greatly important was the team ensuring that they had done a reasonable job and effectively spaced out their work on the project application as cleanly as they could, avoiding last-minute operations as much as possible. The team did in fact overall avoid any procrastination and it was because of the teams' proactive cooperation that scheduling and time management issues were largely avoided. In order to ensure that the time was managed, partnership with task delegation meant that the two qualities were not mutually exclusive, therefore tasks being effectively performed by teams was just as crucial as time manage-

ment in itself, and it was because the result of task delegation was so excellent that time management in itself did not pose a significant issue to the team's development of the project.

- **Collaborative Development:** With team members working on different components (e.g., frontend, backend, document processing), clear communication and task delegation were critical. Using version control effectively and holding regular sync meetings helped manage dependencies and ensured integration went smoothly. Time management and collaborative development were not mutually exclusive, meaning that proper time management allowed for true freedom in issues related to collaborative development; however, this was simply a band-aid and not a potion to a lack of collaboration, meaning that a lack of collaboration would penalize the group no matter how much time was allotted. Meetings, active chats, and cooperation in setting up times and going out of the way to get the project done were ultimately incredibly crucial in speeding up the project development; even meetings that didn't appear to serve a purpose initially ultimately ended up making a heavy difference in the planning and revising of the project. In essence, even bad planning had to lead to good planning, and the good planning is what ultimately took the team to the success of the project.

9.3 3. Design and Usability Considerations

- **User Interface and Experience (UI/UX):** Designing a user-friendly interface for a technical tool required additional thought, particularly around document upload, collection management, and chat interactions. Feedback from testing sessions emphasized the need for intuitive navigation and minimal steps to access the system's core functionalities. When designing a User Interface, it is important to consider the human and computer interaction between the two. It is important to keep the application ambiguous towards users of all backgrounds, and should support a common interface that makes the application easy to use and involve language that any user can recognize. Essentially, the design should be such that it is relatively easy to navigate the application.
- **Document Format Compatibility:** Supporting various document formats (e.g., .txt, .html, .zip, URLs) added value to the system, but it also introduced complexity in text extraction. Implementing fallback

mechanisms and consistent pre-processing steps made the system more robust. It was important to support as many basic formats as possible to make the application as versatile as possible. Therefore, having a file type that wasn't readable by the application wouldn't be such a deal breaker to the point that a user may use a competing product, and affect all shareholders of the application. Lacking functionality for an advanced or significantly less common file type may have been excusable, but doing so for a basic type would have ruined the entire reputation of the project and was thus incredibly risky. Therefore, we considered the document format compatibility to be an absolutely essential cornerstone of the user experience and a necessary feature to implement properly.

9.4 4. Planning

- **Prioritization:** Initially, the work was mostly split, with members working separately on the backend functionality of the project, while other members worked on the RAG pipeline. This turned out to be an inefficient and dangerous strategy, as despite our communication with one another and meetings to resolve the two portions and merge them together, merging became problematic as neither implementation really bore the other in mind. Because of this, we had to soft-reboot the project, creating a fresh implementation that spoke with both the backend and the RAG pipeline, without any need for merging as the implementation was merged together from the beginning. Had we considered the ambitious task of the merge beforehand, we could have saved a significant amount of time in our team planning, merging the content immediately and would have had time for further adjustments that extended beyond a basic implementation.
- **Extended Research Time:** The team spent a significant time researching and planning, as the team deluded themselves into believing that the project was rather ambitious and needed extra support and material in order to craft up a proper project; however, this was ill-founded, and the group eventually needed something to show for the project. The time spent on research was so great that pre-project planning such as wireframes were entirely scrapped; instead, the program structure had to be made as it was going through, which meant the project was initially poorly planned. After

the first basic step was created, more time was allotted to fully focus on the project, but this had taken enough time away that any further planning would have to be optimized in the future. In essence, the project spent so much time in one phase, other phases had to be skipped entirely and we had to jump straight to the project and plan as we created. Although we had successfully managed this over time, this was a widely unprofessional move, and the project could have gone significantly more efficiently had we not made this critical error.

9.5 5. Future Considerations and Improvements

- **Scalability and Deployment:** Hosting models locally worked well for testing, but future deployments may require more scalable solutions, such as distributed processing or GPU support, to improve response times and handle larger datasets. Therefore, the data that we had used was relatively primitive and was designed on a scale small enough for a group in a Capstone course; however, upscaling this project may require more working capital to satisfy the needs of the application and the shareholders accordingly. It should be significantly easier to upscale the project towards the future, in case expansion is to be made, in which case any necessary expansion and polishing can be properly implemented.
- **Enhanced Security Measures:** As we integrated user authentication, ensuring data privacy became a priority. Additional security features, like encryption for sensitive data could be further enhanced in future iterations. Because the current scale of the application assumes a small number of potential users, security is relatively small with obvious changes that can be further implemented to ensure complete user and user content privacy. As the project is upscaled and improved, extra security measures may need to be implemented. Fortunately, the current security techniques provide enough of a base implementation that further security implementations should be significantly easier to implement.
- **Advanced Retrieval Techniques:** Exploring hybrid retrieval techniques—combining semantic search with rule-based filtering—might improve the accuracy of responses, especially for multi-document contexts. Perhaps using more efficient execution flow to speed up searching techniques, such as using multithreading or thread pool-

ing techniques, may in fact significantly speed up our application. Doing so can also speed up the application if mutex locks and recursive locks are ignored and rather, an atomic lockless implementation is done instead, as lockless implementations are more efficient and a pooling technique can effectively serve as a pure optimization to simply creating more threads. This should be able to build upon further searching and scraping techniques as an obvious solution, but should a further optimization be desired, it would be imperative to consider other retrieval techniques.

In conclusion, the project has been a valuable learning experience, reinforcing the importance of planning, collaboration, and adaptability in developing a complex, multi-functional QA/chatbot system. These lessons lay a strong foundation for further advancements and optimizations in future work.

10 Timeline

The following section outlines the timeline of work completed

10.1 Development Timeline

- **October 27**:
  - Complete LangChain pipeline implementation
  - Finish component integration
  - Initial CSS design implementation
- **November 10**:
  - Finalize CSS design
  - Complete LLM integration and localization
  - Deploy containerized application
- **November 17**:
  - Complete web scraping functionality
  - Finish HTML filtering implementation
  - Verify all core functionality
- **November 22**:
  - Bug fixing and system optimization
  - Finalize presentation materials
  - Complete 50% of final documentation
- **December 8**:
  - Submit final documentation
  - Complete all deliverables
  - Project completion

11 Acknowledgments

We would like to acknowledge our client, Dr. Mohammed Farag for their consistent feedback, which significantly affected the direction of our project and consequently the final product. The lecture by Elnady has provided some inspiration to the structure of our project, and provided useful sample code to be run in Jupyter notebooks that we have inspected, but instead opted for a different approach written in Python. We would also like to thank the previous team who implemented a version of this project. We redesigned much of the application, but the problems they ran into, and certain code sections served as a useful guide.

12 References

- https://colab.research.google.com/drive/1tw1PzkMe6TlIZX9NSu9rhLNIfU0W8w7M
- https://cratedb.com/use-cases/ai-ml-database/retrieval-augmented-generation-pipelines
- https://github.com/PromtEngineer/localGPT
- https://github.com/MaokunZhang/Intelligent-QA-Chatbot-Old
- https://vtechworks.lib.vt.edu/items/976c5deb-cae3-4654-a86f-2fd6a668990e