Lecturer: Phd Sharon Yalov Handsle Noy Abecasis 206784266, Sharon Fogel 313118945, Lyr Ita 316136811

sTUDDY BUDDY Statement of work

**Introduction: Get to Know Study Buddy**

In today's fast-paced academic environment, students often find themselves inundated with a plethora of study materials, making it challenging to effectively manage and comprehend vast amounts of information. Our final project addresses this issue by developing an innovative AI Assistant tailored specifically for students across various degree programs. This AI Assistant is designed to enhance the learning experience by leveraging advanced artificial intelligence techniques to assist students in processing and understanding content from PDF files, such as summaries and articles.

The core functionality of our AI Assistant revolves around providing intelligent support for students. Users will have the ability to upload any PDF files into the system, which will then analyze the content to generate concise summaries. Additionally, the system will be capable of creating questions and answers based on the text, facilitating a deeper understanding and retention of the material. This feature is particularly beneficial for exam preparation and reviewing key concepts.

Our project will be realized as a user-friendly web application, ensuring accessibility and ease of use for students. The application will incorporate a robust backend system and a comprehensive database to securely store users' chat histories and interactions. The interactive user interface will be designed with students in mind, offering an intuitive and engaging experience.

By integrating these features, our AI Assistant aims to become an indispensable tool for students, helping them to efficiently navigate their academic journey and achieve their educational goals. The project not only showcases our technical skills and knowledge but also demonstrates our commitment to leveraging technology to solve real-world problems in the educational domain.

**Project Objectives**

The main objective of this project is to develop an AI Assistant that provides intelligent support for processing and understanding content from PDF files, facilitating a deeper understanding and retention of material through an interactive and user-friendly web application.

* **Note Summarization:** Extract text from uploaded documents and store it in a vector database. Utilize large language models (LLMs) to condense information into key points and summaries, making it easier for students to grasp essential concepts quickly.
* **Question Generation and Checking:** Automatically generate short-answer questions based on the content of the PDFs, along with matching answers. This feature allows students to test their understanding and use the system for self-assessment.
* **Query Response:** Analyze user queries and search through uploaded notes for relevant content. Present the information and respond to specific user requests related to the text, providing targeted and contextually accurate answers.

**Project Metrics**

* **Note Summarization:**
  + Summarization Quality: Use a simple percentage metric where a sample of summaries are reviewed by users or educators, and they rate them on a scale of 1 to 5 (1 being very poor and 5 being excellent). Calculate the average rating to gauge quality.
  + Processing Time: Track the time taken to generate summaries from the uploaded documents to ensure it is within an acceptable range for real-time use.
* **Question Generation and Checking:**
  + Relevance: Evaluate the relevance of the generated questions to the content in the PDFs. This can be assessed through expert reviews or user feedback.
  + Accuracy: Measure the correctness of the generated answers compared to a set of reference answers. This can be done using precision, recall, and F1-score metrics.
* **Query Response:**
  + Response Accuracy: Assess the accuracy of the responses to user queries by comparing them with expected answers. Metrics like precision, recall, and F1-score can be used.
  + Response Time: Monitor the time taken to analyze and respond to user queries to ensure the system provides timely assistance.

**Literature Review**

**Leveraging Large Language Models (LLMs) for Enhanced Learning Experiences**

In recent years, Large Language Models (LLMs) have emerged as transformative tools in education, showcasing remarkable capabilities in understanding and generating human-like text. Models such as T5, mT5, and GPT-3 have demonstrated versatility across various tasks, from language translation to text generation, highlighting their potential to revolutionize learning experiences. Our AI final project harnesses the power of LLMs to provide personalized explanations, analyses, and solutions to Computer Science (CS) students, enhancing their comprehension and retention of course material.

While LLMs offer immense potential, they also pose challenges, particularly in terms of computational resources and efficiency. The exponential growth in model parameters and training datasets has necessitated significant computational resources, hindering widespread adoption. However, researchers are actively addressing these challenges through techniques such as parameter-efficient tuning, model pruning, and knowledge distillation. By leveraging these advancements, we mitigate the computational overhead associated with LLMs, making our AI Assistant platform more accessible and scalable for CS students worldwide.

Moreover, LLMs exhibit emergent cognitive abilities that extend beyond traditional Natural Language Processing (NLP) tasks. These latent capabilities, including reasoning, planning, and context-aware learning, hold promise for revolutionizing the educational landscape. Through strategic integration within our AI Assistant platform, we aim to harness these capabilities to empower CS students with adaptive and personalized learning experiences. By providing tailored explanations, insightful analyses, and targeted solutions, LLMs enable our platform to cater to the diverse needs and learning styles of individual students, fostering a more engaging and effective learning environment.

In conclusion, the integration of LLMs within our AI Assistant project represents a paradigm shift in education, offering CS students a personalized and immersive learning experience that transcends traditional methodologies. [1]

**Leveraging Retrieval-Augmented Generation (RAG) for Enhanced Learning Experiences**

In our endeavor to develop an AI Assistant tailored for Computer Science (CS) students, we recognize the critical role of Retrieval-Augmented Generation (RAG) in revolutionizing the learning experience and deepening understanding of course material. By integrating RAG into our project, we aim to augment the internal knowledge base of the assistant with external sources, ensuring accurate, relevant, and contextually appropriate responses to student queries. [2]

RAG, introduced by Meta (formerly Facebook) in 2020, enhances large language models (LLMs) by leveraging external knowledge sources. It operates through retrieval and generation phases, enriching the model's understanding and contextualization with information sourced externally. By grounding responses in up-to-date, verifiable facts, RAG reduces inaccuracies, bolstering user trust in AI-generated outputs.

The implementation of RAG across various domains underscores its transformative potential, enabling personalized responses while ensuring accuracy and reliability. Challenges persist in its implementation and refinement, but as advancements in RAG continue, its impact is poised to revolutionize the way we harness AI technology. [3]

In summary, Retrieval-Augmented Generation (RAG) stands at the forefront of AI innovation, offering a transformative solution to enhance the reliability of generative models. By bridging the gap between internal knowledge and external sources, RAG unlocks new possibilities for accurate and trustworthy AI-generated content, revolutionizing the learning experience for CS students.

**Leveraging Prompt Engineering for Enhanced AI Assistance**

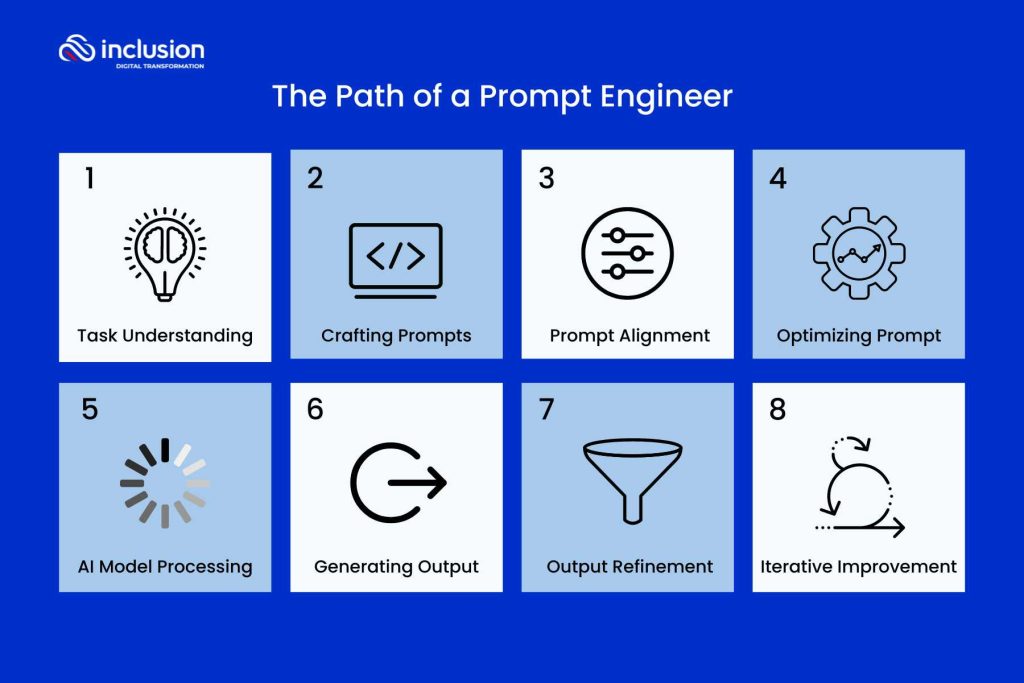
The integration of prompt engineering techniques within our CS AI project represents a significant advancement in refining the capabilities of our AI Assistant platform. Prompt engineering has directly influenced our project in several key ways, enhancing note summarization, question generation, and query response mechanisms for computer science students.

Prompt engineering has guided our approach to note summarization, enabling us to use precise prompts to effectively guide language models in generating concise and accurate summaries of class notes. By crafting specific prompts tailored to the content and structure of the notes, we ensure that our AI Assistant delivers summaries that capture the key concepts and insights comprehensively, enhancing students' comprehension and retention of course material.

Applying prompt engineering techniques has empowered us to generate relevant self-assessment questions from class notes with precision. By formulating precise prompts, we guide language models in generating questions that accurately assess students' understanding of the material. Additionally, prompt engineering enables us to implement accurate answer checking mechanisms, ensuring that students receive informative feedback on their responses, further enhancing their learning outcomes.

Prompt engineering plays a crucial role in formulating specific prompts for analyzing queries and providing references from uploaded notes. By crafting precise prompts, we enable our AI Assistant to understand and respond effectively to students' queries, ensuring that the information provided is accurate, informative, and relevant to their needs. This enhances the overall user experience and enables students to access relevant resources efficiently.

In essence, the integration of prompt engineering techniques into our CS Tutor AI project has guided our approach, enabling us to create a more effective and comprehensive AI assistant for computer science students. By leveraging insights from the article, we optimize the quality of AI-generated outputs, reduce post-processing efforts, and enhance the efficiency of our platform. As prompt engineering continues to evolve, we remain committed to leveraging advanced techniques to further improve the capabilities of our AI Assistant, ultimately empowering students to excel in their academic endeavors. [4]



**Competitors Overview**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Feature | Study Buddy |  | Socratic | Claude |  | GPT |  | Gemini | SummarizeBot |
| Summarization with AI | ✔ |  | - | ✔ |  | ✔ |  | ✔ | ✔ |
| Ability to Upload Files | ✔ |  | - | ✔ |  | ✔ |  | - | ✔ |
| Question Generation & Explanations | ✔ |  | ✔ | ✔ |  | ✔ |  | ✔ | - |
| Interactive Interface | ✔ |  | ✔ | ✔ |  | ✔ |  | ✔ | ✔ |
| Internet Access | X |  | ✔ | ✔ |  | ✔ |  | ✔ | ✔ |

A logo with black text

Description automatically generatedתמונה שמכילה אומנות קליפיפם, סרט מצויר

התיאור נוצר באופן אוטומטיA blue and black robot with a pencil and a speech bubble

Description automatically generatedA blue and white logo

Description automatically generatedA logo of a company

Description automatically generated

**Functional Requirements**

* The user will be able to register to the system. During registration, the system will ask the user for the following details: name and email address.
* The registered users' details will be saved in a database.
* The user will be able to upload PDF filescontaining summaries related to the student’s study subjects. The files will have a size limit of 200 MB.
* The system will be able to convert PDF files to text files.
* The system will be able to divide the text files into text chunks (for vector usage).
* When the user interacts with the chat, they will have two possible request types:
  + Create a united summary from the PDF files.
  + Generate questions according to the amount requested by the user, in different levels about the study materials, and provide matching answers for self-checking.
* The user will be able to provide feedback on the system’s output, such as: requesting a specific summarization length, including/excluding specific topics, asking for more details, requesting different/more questions, etc.
* The system’s output will be saved in a database and will be visible to the user on their next visits.
* The system will be able to export all the generated questions to a PDF file.

**Non-Functional Requirements**

* Design Requirements: The interface should be clear and user-friendly, allowing new users without a technological background to operate the web application independently from the first use.
* Performance Requirements: The web application should provide a response to the user about the generated summaries or questions in minimal time, ideally within 2 minutes.
* Operational Requirements:
  + The application should support file uploads of PDF documents up to 200 MB in size.
  + The application should be able to connect to and utilize external resources, such as a a large language model (LLM) service.
* Maintenance Requirements: The system should include a robust database capable of storing user data, summaries, questions, and chat histories without causing the application to crash.
* Reliability Requirements: The system should be able to detect errors in server connections and notify users when there is an issue preventing a response.

**System Architecture**

High Level Architecture & Components Diagram:

A diagram of a software process

Description automatically generated

Technology Stack Selection

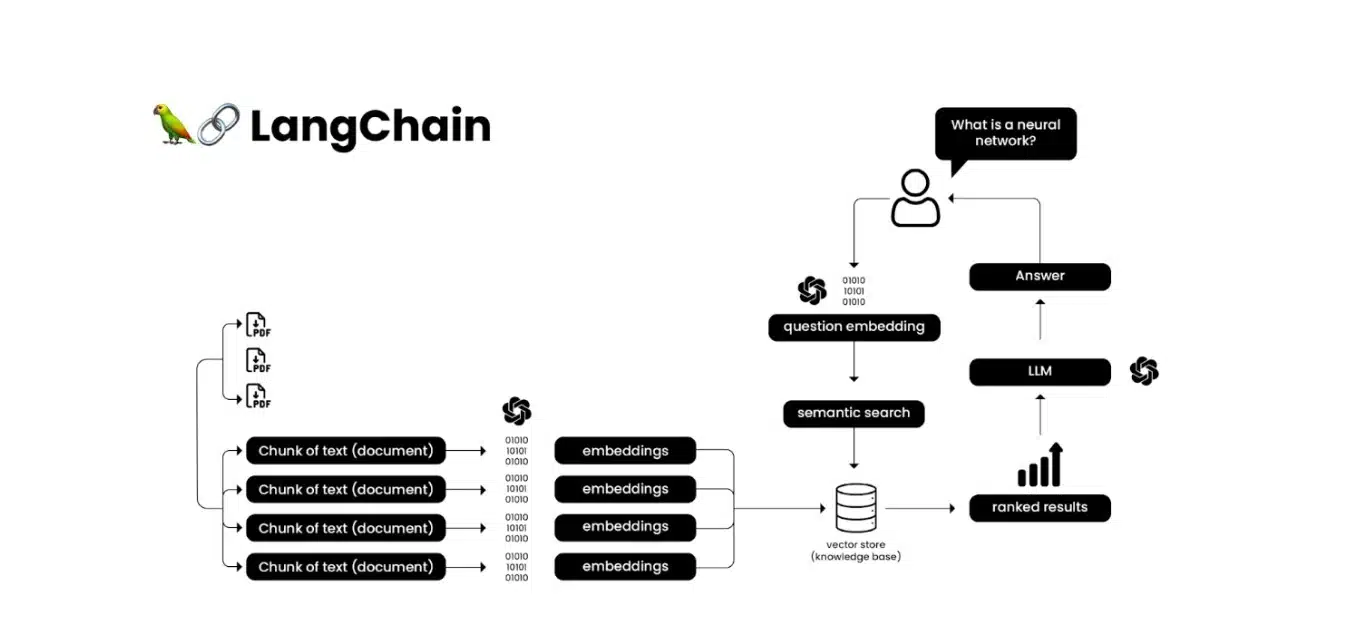
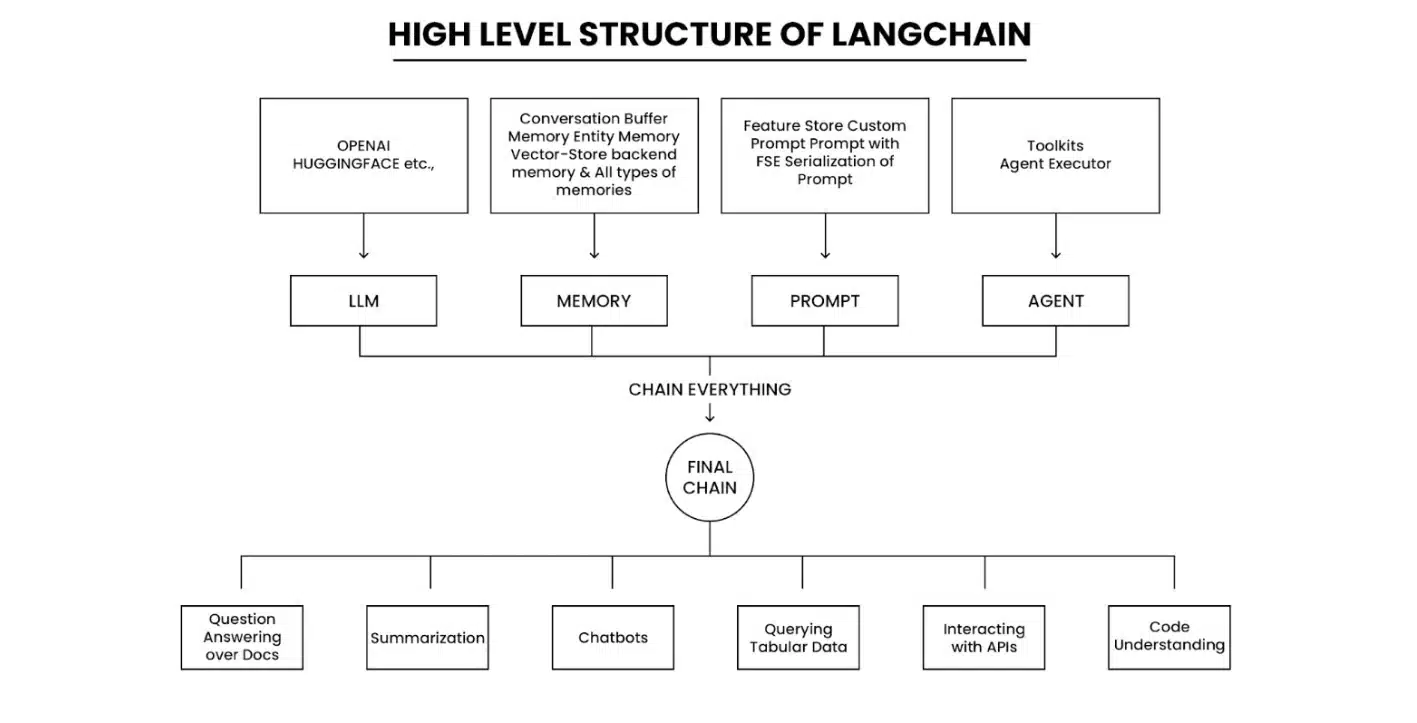
**Frontend Platform:**

**Streamlit:** Streamlit is an open-source Python library used for creating interactive data visualization and user interfaces. In our project, Streamlit is employed to develop the interactive user interfaces, enabling users to upload PDF files, interact with the AI Assistant, and view generated summaries and questions.

**Backend Application:**

Python: The project's backend logic is written in Python, handling all essential processes including user login and authentication, uploading and converting PDF files to text, connecting to the database (which stores user data and chat histories), interfacing with the LLM model, and retrieving chat responses.

LangChain: LangChain is a framework for developing applications powered by large language models (LLMs). In our project, LangChain is used to interact with OpenAI's API. LangChain manages the input to the model, enabling it to function as a chatbot capable of answering questions, generating text, and creating summaries. By leveraging LangChain, we streamline the process of sending queries to OpenAI's LLM and handling the responses, ensuring efficient and effective interaction with the AI model.



LLM - OpenAI API: The core of our AI Assistant is powered by OpenAI's large language model (LLM). The OpenAI API provides advanced natural language processing capabilities, allowing the system to generate high-quality summaries, formulate questions, and provide accurate responses based on the content of uploaded PDF files. The integration with OpenAI's LLM ensures that the AI Assistant can understand and process complex academic material, making it an invaluable tool for students.

**Database:**

MongoDB: MongoDB is the database used to store all user data, including user profiles, uploaded PDF files, and all chat queries and responses. MongoDB's flexible schema and scalability make it ideal for managing the diverse and growing data needs of our application, ensuring that user information and interactions are securely and efficiently stored.

**A diagram of a computer program

Description automatically generated**

Integration and communication protocols

**Integrations**

Our AI Assistant project integrates several key technologies and services to provide an efficient user experience.

**OpenAI API:** powers the large language model (LLM) and used for natural language processing. This integration enables the system to generate high-quality summaries, formulate questions, and provide accurate responses based on the content of uploaded PDF files. The OpenAI API is accessed via **secure HTTP requests**, with **JSON payloads** used to send queries and receive responses.

**LangChain:** serves as the intermediary framework between our backend logic and the OpenAI API. LangChain facilitates communication with the OpenAI API by securely transmitting user queries and inputs through HTTPS using RESTful API calls. User input is formatted into a JSON payload and sent to the OpenAI API, which processes the request and returns a JSON response.

**MongoDB:** database solution used to store all user-related data. This includes user profiles, uploaded PDF files, chat queries, and responses as messages or generated questions. The integration with MongoDB is achieved through the use of the PyMongo library, which allows for seamless communication between our Python backend and the database.

**Streamlit:** Streamlit is used for building the interactive user interfaces of our web application. Streamlit interacts with the backend via **HTTP requests**, ensuring real-time communication and data exchange between the frontend and backend components.

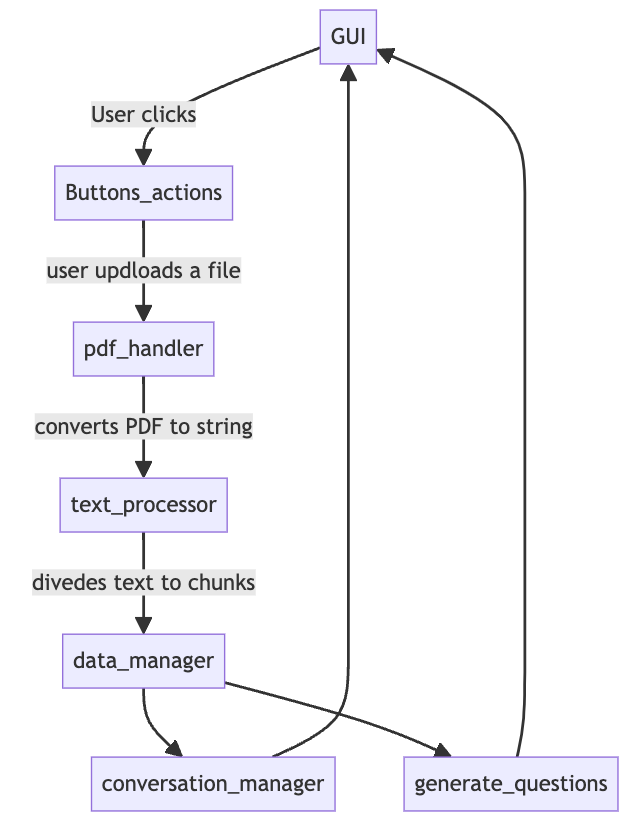
**Communication Protocols**

**HTTP/HTTPS:** All communications between the frontend (Streamlit) and backend services are conducted over HTTP/HTTPS protocols. HTTPS is used to ensure secure transmission of data, protecting user information and interactions from potential security threats.

**RESTful APIs:** The backend services expose RESTful APIs that handle various operations, such as user authentication, file uploads, text conversion, and database interactions. These APIs are designed to be stateless, providing a clear separation between client and server operations and ensuring scalability.

**JSON:** JSON (JavaScript Object Notation) is the primary format for data exchange between the frontend, backend, and external services (like OpenAI). JSON is lightweight and easy to parse, making it an ideal choice for transmitting structured data.

**Detailed Design**

Modules Diagram

* **GUI:** Handles user interface and interacts with buttons\_actions to process user inputs.
* **Buttons\_Actions:**
  + Manages the creation and handling of button interactions within the Streamlit app.
  + Includes functions that define what happens when buttons are clicked, including triggering different workflows for summarization, chat, and question generation.
* **PDF\_Handler:** Converts PDFs to strings.
* **Text\_processor:** Divides string to chunks for embeddings.
* **Data\_manager:**
  + Manages the storage and retrieval of user data, PDF files, chat histories, and responses.
  + Interacts with the MongoDB database to save and retrieve conversation data and text chunks
* **Conversation\_manager:** integrates with the language model and maintains conversation context through ConversationBufferMemory. It processes user inputs for both chatting and summarizing by leveraging the ConversationalRetrievalChain, ensuring that interactions are context-aware and the session state is consistently updated to reflect the ongoing conversation or summarization.
* **Generate\_Question:** responsible for generating questions based on the text provided. It uses LangChain and OpenAI's models to analyze text chunks and create questions of varying difficulty to help users prepare for exams or tests.

Objects

1. **Chat:**

The chat class represents an individual chat session between a user and the system.

Attributes:

* id (str): A unique identifier for the chat session.
* user\_id (str): The identifier of the user who owns this chat session.

Purpose:

To encapsulate the details of a chat session, uniquely identifying it and associating it with a specific user.

1. **Text File:**

The text\_file class represents the text extracted from a file (e.g., PDF) that is associated with a specific chat session.

Attributes:

* text (list): A list of text chunks extracted from the file.
* uid (int): The user identifier to whom this text file belongs.
* chat\_id (int): The identifier of the chat session associated with this text file.

Purpose:

To store and manage text data extracted from files, linking it to a specific user and chat session for further processing and question generation.

**3. User:**

The user class represents an individual user of the system, including their personal details and the chats they have initiated.

Attributes:

* chats (dict): A dictionary of chat objects, keyed by chat ID.
* name (str): The name of the user.
* uid (str): A unique identifier for the user.
* mail (str): The email address of the user.
* current\_chat (str): The ID of the current active chat session.

Methods:

* add\_new\_chat() -> str: Creates a new chat session, adds it to the user's chat list, and sets it as the current chat.
* add\_chat\_by\_id(chat\_id): Adds an existing chat session by its ID to the user's chat list.
* update\_current\_chat(chat\_id: int): Updates the current active chat session for the user.
* update\_session\_from\_db(): Updates the user's chat sessions by fetching the latest sessions from the database.

Purpose:

To manage user-related data and operations, including the creation and updating of chat sessions, and maintaining the user's current chat context.

Database Schema Design**A diagram of a computer program

Description automatically generated**

**1. Message**

The Message entity stores individual messages exchanged in chat sessions.

Attributes:

* message\_id (string): A unique identifier for the message.
* session\_id (string): The identifier of the chat session to which this message belongs.
* content (string): The actual text content of the message.
* type (string): Indicates whether the message was sent by the user or the AI (e.g., "user" or "assistant").

Purpose:

To store and manage individual messages within chat sessions, facilitating retrieval and display of chat history.

**2. User**

The User entity stores information about individual users of the system.

Attributes:

* user\_id (string): A unique identifier for the user.
* name (string): The name of the user.
* email (string): The email address of the user.
* password\_hash (string): A hashed version of the user's password for secure authentication.

Purpose:

To manage user accounts, including authentication, user information storage, and account maintenance.

**3. Question**

The Question entity stores questions generated by the system for the users.

Attributes:

* question\_id (string): A unique identifier for the question.
* session\_id (string): The identifier of the session during which the question was generated.
* difficulty (string): The difficulty level of the question (e.g., "easy", "medium", "hard").
* content (dictionary): The actual text content of the question and the answer.

Purpose:

To store and manage questions and their corresponding answers, facilitating the preparation and delivery of questions to users.

**4. Session**

The Session entity stores information about individual chat sessions.

Attributes:

session\_id (string): A unique identifier for the session.

user\_id (string): The identifier of the user who owns the session.

text\_file\_id (string): The identifier of the text file associated with this session (if any).

Purpose:

To manage chat sessions, associated chat history, and any linked text files.

**Algorithm Description**

A diagram of a diagram

Description automatically generated**System Flows**

1. User Authentication and Initialization:

- Users register or log in to the system, providing necessary details such as name, email, and password.

- Upon successful login, the user's data is fetched, including any previous chat sessions, questions, and messages.

2. Starting a New Chat Session:

- Users can start a new chat session or continue an existing one.

- When a new session is initiated, a unique session ID is generated, and a `chat` object is created.

- The system initializes a conversation chain using OpenAI's language model for interactive responses.

3. User Interaction:

- Users input questions or statements into the chat interface.

- The system processes the input and retrieves relevant information from the vector store if necessary (Retrieval-Augmented Generation, RAG).

- The response is generated using OpenAI's language model, considering both the current input and the conversation history.

4. Question Generation:

- Users can request the generation of exam preparation questions in different difficulty levels based on provided text.

- The text is processed into manageable chunks, and a question generation pipeline using OpenAI's model is employed.

- Generated questions are refined and stored in the database, linked to the current session and user.

5. Saving and Retrieving Data:

- Chat messages, generated questions, and session data are saved to the database.

- The `message` entity records individual chat exchanges, while the `question` entity records generated questions and answers.

- Sessions are tracked with start and end times, maintaining a history of user interactions.

6. Session Resumption:

- Users can resume previous sessions, which involves fetching the session data and reconstructing the chat history.

- The vector store is reinitialized with relevant text chunks to ensure continuity in the conversation.

**OpenAI Model**

he core of the system's conversational capabilities is powered by OpenAI's GPT-3.5-turbo model.  
It handles natural language understanding and generation, providing interactive and coherent responses to user inputs.

**Question Generation:**

The system utilizes OpenAI's language model to generate and refine questions based on input text.

Prompt templates guide the model to produce relevant questions with specified difficulty levels, aiding users in their exam preparation.

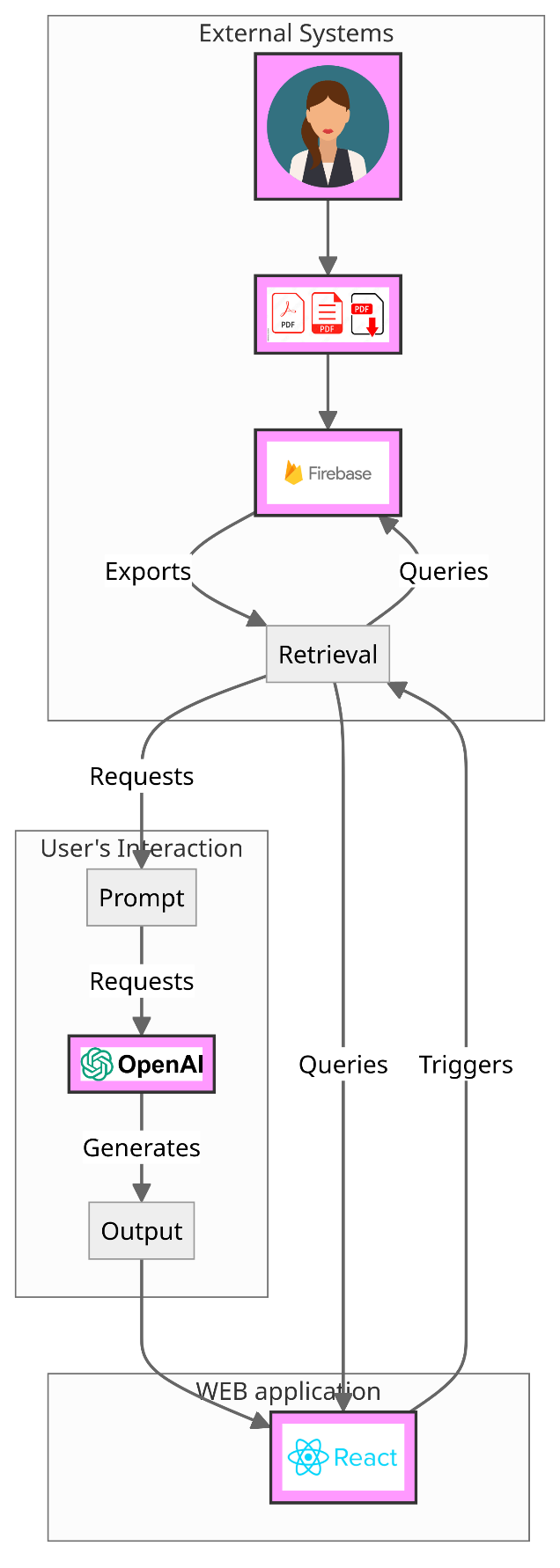
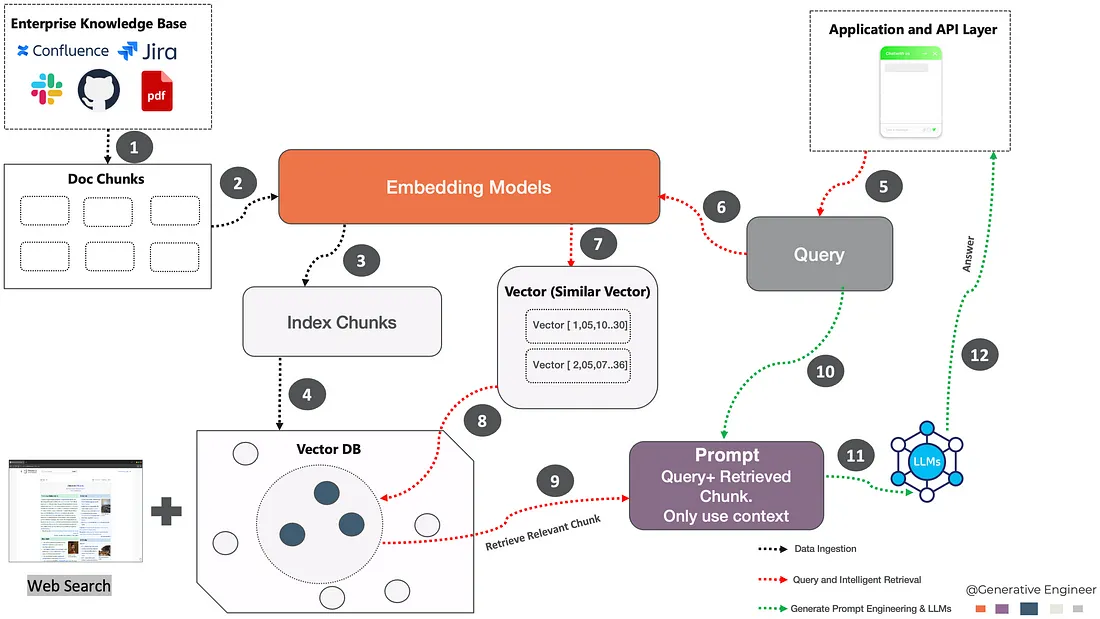
**Retrieval-Augmented Generation (RAG):**

Vector Store:

Text chunks are processed and stored in a vector store using FAISS, which facilitates efficient similarity searches.

When a user inputs a query, the system retrieves relevant text chunks from the vector store to augment the generation process.

This ensures that responses and generated questions are contextually relevant and accurate.



**Complexity**

Text Processing: Text is split into chunks using the `TokenTextSplitter`, which ensures that the chunks are of manageable size and overlap sufficiently for context retention.

Data Management: The schema involves multiple entities and relationships, which require efficient handling and retrieval of data to maintain performance.  
The integration of MongoDB and FAISS for storage and retrieval adds a layer of complexity in managing data consistency and query efficiency.

OpenAI Model Integration: Prompt engineering and the integration of multiple OpenAI API calls for different functionalities (chat response, question generation, and refinement) require careful orchestration. The system ensures low latency and high relevance in responses, leveraging the power of advanced language models.

**Project Timeline**

|  |  |  |
| --- | --- | --- |
| **Task** | **Dates** | **Description** |
| Literature Review + Competitor Analysis | Feb 13, 2024 |  |
| Functional and Non-Functional Requirements | Feb 27, 2024 |  |
| Detailed Design Document | Mar 5, 2024 |  |
| Meeting with Mentor | Mar 3, 2024 | Discuss project progress and receive guidance. Review timeline and adjust tasks if necessary. |
| Preparation of Project Presentation | Mar 6 - Mar 8, 2024 | Create slides to communicate project objectives, scope, features, technologies, progress, etc. |
| Research and Technology Study | Mar 8th - Mar 19th, 2024 | * Study and evaluate various frontend frameworks (e.g., Streamlit) * Research backend technologies and programming languages (e.g., Python) * Investigate large language models (LLMs) and APIs (e.g., OpenAI API) * Study database options |
| Design and Architecture | Mar 20th - Mar 24th | * Develop high-level architecture and component diagrams * Design database schema and data models * Plan for integration points between frontend, backend, and external services |
| Backend Development with Python | Mar 25th - Apr 10th, 2024 | * Set up backend infrastructure (e.g., servers, databases) * Implement user authentication and database connectivity * Develop initial backend API endpoints |
| Frontend Development with Streamlit | Apr 11 - Apr 20, 2024 | * Build frontend components and integrate with backend APIs * Implement PDF upload and text conversion functionalities * Start developing LangChain integration with OpenAI API |
| Integration of AI Models | Apr 20th – May 10th, 2024 | * Continue LangChain integration, focusing on query handling and response generation * Implement chat functionalities and user interaction features * Develop features for saving and retrieving user data from MongoDB * Integrate frontend with backend services and ensure seamless communication |
| Testing and Debugging | May 11th - May 20th, 2024 | Identify and fix bugs or issues. Ensure functionalities work as intendedIdentify and fix bugs   * Test and improve output quality- verify data integrity * Prepare a detailed test report and documentation |
| Documentation | May 21th - May 28th, 2024 | Creating detailed documentation for all project’s goals and components. |
| Final Checks and Submission | May 28th - June 2nd, 2024 | Perform final checks and submit the completed project. |

**Tests & Code Results**

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case** | **Input/Action** | **Expected Outcome** | **Outcome** |
| **User Registration** | | | |
| Register New User | Name, email address | User account created, details saved in the database. | As expected |
| Register with Existing Email | Name that already exists | Error message: username already registered. |  |
| Register with missing fields | Empty fields | Error message: some fields are missing |  |
| **User Login** | | | |
| Login with Correct Credentials | Registered email and correct password | User successfully logs in, redirected to main dashboard | As expected. |
| Login with Incorrect Credentials | Registered email and incorrect password | Error message: incorrect email or password. |  |
| **PDF Upload** | | | |
| Upload Valid PDF Files | PDF files | Files successfully uploaded and stored. | As expected. |
| Upload Exceeding Size Limit | PDF file larger than 200 MB | PDF file larger than 200 MB | Error message: file size exceeds limit. |
| **PDF Conversion to Text** | | | |
| Convert Uploaded PDF to Text | Valid PDF file | PDF content correctly converted to text. | As expected. |
| **Text Chunking** | | | |
| Divide Text into Chunks | Text from PDF file | Text successfully divided into chunks for vector usage. | As expected. |
| **Generating Summaries** | | | |
| Generate United Summary | 2 different articles- one about Theodore Roosevelt and the other about the white house. | Concise, coherent summary generated. |  |
| Specify Summarization Length | Entred 2 different articles- one about Theodore Roosevelt and the other about the white house  and asked for a longer summary. | Longer summary |  |
| **Generating Questions and Answers** | | | |
| Generate Questions and Answers | Request for questions from specific PDF content | Relevant questions and matching answers generated. |  |
| Generate Questions at Different Levels | Request for questions at varying difficulty levels | Request for questions at varying difficulty levels |  |
| **Query Response** | | | |
| Query Specific Information | User query related to uploaded PDF content | Accurate and relevant information retrieved and displayed. |  |
| **User Feedback** | | | |
| Provide Feedback on Summary | User requests adjustments to summarization | Summary updated based on feedback. |  |
| **Data Persistence and Retrieval** | | | |
| Save System Outputs to Database | Generated summaries, questions, and answers | Outputs saved, retrievable in future sessions. | As expected. |
| Export questions outpt to PDF | Request to export questions | Output successfully exported to PDF file | As expected. |
| **Error Handling** | | | |
| Server Connection Error |  |  |  |
| Invalid File Format | Invalid File Format | Specified error message |  |
| Unexpected System Error | System encounters an unhandled exception | Generic error message: an unexpected error occurred, please try again later. |  |
| **Security** | | | |
| User Authentication | Ensure only authenticated users can access system | Unauthorized access prevented. | As expected. |
|  |  |  |  |

**Analysis of Results**

* **Note Summarization:**
  + Summarization Quality:
    - Metric: Average user rating on a scale of 1 to 5.
    - Analysis: Collected a sample of summaries and have them reviewed by users. We calculated the average score to determine overall quality.
    - Result: An average rating of 3.2 - the summaries are generally adequate but not exceptional, not all the relevant information is included in part of the summaries. In some of the cases the summaries were too short.
  + Processing Time:
    - Metric: Tracked the time from when the user uploads a document and the time to generate a summary.
    - Result: average time to upload a file – 0.3 sec, average time to process uploaded file- 4.16 sec (depends on the file size!), average time to generate a summary- 5.3 sec.
* **Question Generation and Checking:**
  + Relevance and accuracy:
    - Metric: Rate of the relevance of the generated questions to the content in the PDFs by user feedback.
    - Analysis: Collected a sample of questions and have them reviewed by users. We calculated the average score to determine overall relevance.
    - Result: In case the text is short, our users found part of the questions repetitive but indeed matches the difficulty level. The questions and the answers content in most of the cases were relevant and accurate. The final average rate by our users was 4.1.
  + Processing Time:
    - Metric: Tracked the time that generating questions takes.
    - Result: average time to generate questions -24.7 seconds.  
      The time changes dramatically according to the PDF length – short texts usually takes around 13 seconds, longer text can take even more then 50 seconds.
* **Query Response:**
  + Response Accuracy:
    - Metric: Rate the accuracy of the responses to user queries by comparing them with expected answers.
    - Analysis: Collected responses for many different query type – some weren’t even related to the text.
    - Result: Most of the query responses had high accuracy rate of 4.7, for non related to the text queries the system was able to respond accordingly and decline the query with proper answer.
  + Response Time:
    - Metric: Monitor the time taken to analyze and respond to user queries.
    - Result: average time to response to queries is 7.3 seconds.

**Suggestions for Future Improvements**

1. Improve Summarization Quality by collecting users feedbacks.
2. Enhance Question Generation by choosing the exact questions amount and choose a specific questions topic.
3. Optimize response time by using vector database instead of vector store.
4. Add recommendations feature that will suggest related articles and documents based on the content of uploaded PDFs.
5. Add learning goals to users and rate their answers to the systems generated questions.

**Bibliography**

1- Anwar, S., Barnes, N., Khan, A. U., Mian, A., Naveed, H., Qiu, S., Saqib, M., Usman, M., & Akhtar, N. (2023). ”A Comprehensive Overview of Large Language Models”. arXiv.

2- Martineau K., (2023). "What is retrieval-augmented generation?". IBM

3- Singhal R., (2023). "The Power Of RAG: How Retrieval-Augmented Generation Enhances Generative AI". Forbes Technology Council (https://www.forbes.com/sites/forbestechcouncil/2023/11/30/the-power-of-rag-how-retrieval-augmented-generation-enhances-generative-ai/?sh=14a0fb0ea7a6)

(https://research.ibm.com/blog/retrieval-augmented-generation-RAG)  
4- “What is prompt engineering?“. IBM (<https://www.ibm.com/topics/prompt-engineering>) (Anonymous writer)