Contents

List of Figures

Chapter 1

Introduction

1.1 Background

With the rise of artificial intelligence (AI) and machine learning (ML), chatbots have become essential across industries, enhancing interactive user engagement. However, traditional chatbots, often reliant on rule-based systems or generic language models, struggle with niche or domain-specific inquiries, limiting their usefulness where users seek detailed, contextually accurate information beyond everyday conversational topics.

To address these limitations, the Retrieval-Augmented Generation (RAG) framework was introduced. RAG combines retrieval methods with generative language models (LLMs) to fetch relevant, high-quality information from external sources before generating responses. This hybrid approach helps overcome the shortcomings of generic LLMs, which may provide incorrect or incomplete information in specialized areas. By incorporating specific, up-to-date knowledge through retrieval, RAG enables chatbots to respond with greater precision, making them valuable for specialized applications, such as educational content delivery and other targeted knowledge domains.

1.2 Problem Statement

In educational settings, users often need detailed, topic-specific responses beyond what standard chatbots or generic LLMs can offer. Traditional models may lack the precision required for subject-specific queries, limiting their effectiveness in learning environments. This project addresses this gap by developing a RAG-based chatbot tailored to educational needs. By retrieving relevant information from vectorized educational materials, the chatbot ensures precise, contextually relevant responses, enhancing user engagement and supporting better learning outcomes.

1.3 Objectives

1. Develop a RAG-Based Chatbot for Educational Purposes Design and implement a chatbot that uses the RAG framework to answer educational queries accurately by retrieving and generating contextually relevant responses.

2. Implement a Flask-Driven Backend with a Flutter Interface Create a backend server with Flask for processing queries and responses, and design a user-friendly frontend interface with Flutter for seamless user interaction.

3. Pinecone and SQLite for Vector and User Data Management Utilize Pinecone for efficient vector storage of embedded data and SQLite for managing user information, ensuring smooth and scalable data retrieval and storage.

1.4 Literature Survey

To develop an educational chatbot, several resources were referenced to implement core functions such as login, project setup, and RAG integration with LLMs and a vector database.

User Authentication in Flutter The repository by Mohammed Hashim on (Flu) provided code for building a login interface in Flutter with a Flask backend. This was used to create a secure login system for both users and admins in the chatbot application.

Setting up Flutter in VS Code The (Vis) provided guidance on setting up Visual Studio Code as an IDE for Flutter. This resource was used to configure the development environment for coding, debugging, and testing the app.

RAG Chatbot Implementation with LangChain Real Python's tutorial on building an LLM RAG chatbot with (Python) gave a clear structure for combining retrieval with generative responses, guiding the development of a chatbot that retrieves accurate responses based on stored data.

End-to-End RAG Chatbot and Pinecone Database Kong Nopwattanapong's article on (Nopwattanapong) outlined RAG model setup and Pinecone database configuration for vector storage and retrieval. This code was adapted to enable the chatbot to retrieve relevant educational information accurately.

Chapter 2

Methodology

The development of this educational chatbot application follows a systematic methodology, utilizing the Retrieval-Augmented Generation (RAG) framework along with a robust backend server and frontend application. The following sections outline the key stages and components of the methodology used.

2.1 Data Preparation, Embedding, and Retrieval-Augmented Generation (RAG) Pipeline

The chatbot's knowledge base is built from educational content provided in the form of texts or PDFs. This content is preprocessed by splitting it into smaller chunks using a TextSplitter to facilitate efficient retrieval. Each chunk is then transformed into a vector representation using embedding techniques, making it suitable for similarity-based search. These embeddings are stored in Pinecone, a vector database, which allows fast access to relevant chunks when a user submits a query.

When a query is received, the RAG pipeline initiates two main processes:

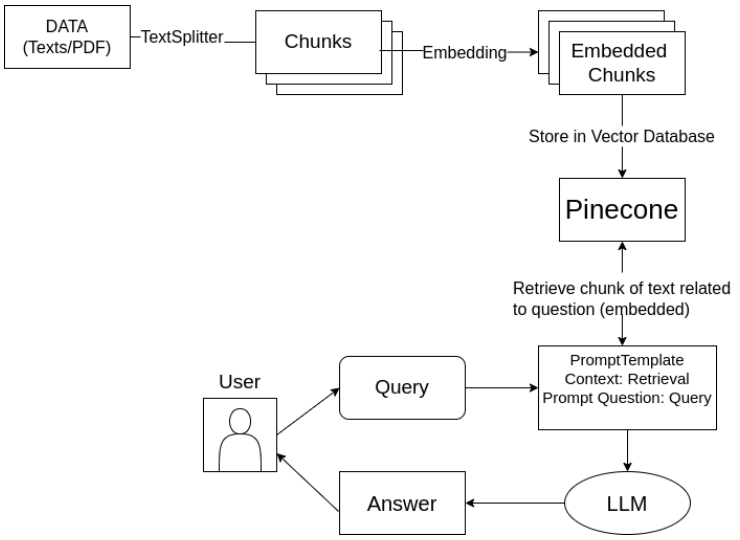


FIGURE 2.1: Rag chatbot integrated with pinecone database

2.1.1 Retrieval

The Flask server searches Pinecone for the most relevant text chunks based on the query's embedding. This retrieval step narrows down the content to the most pertinent information, which is then used as context for generating responses.

2.1.2 Generation

The retrieved content, along with the user's query, is formatted using a prompt template and sent to a Large Language Model (LLM) for response generation. The LLM leverages this context to produce accurate, topic-relevant responses, which are then sent back to the Flask server and displayed in the user interface

2.2 Backend: Flask and SQLite3 Integration

The backend server, built using Flask, serves as the core of the application, connecting the frontend, database, and RAG components. Flask handles requests from the Flutter application, processes user queries, retrieves information, and manages user

	id	name	email	username	password	role
	Fi...	Filter	Filter	Filter	Filter	Filter
1	1	Admin User	admin@example.com	admin	adminpass	admin
2	9	user1	user1@gmail.com	username1	userpass	user

FIGURE 2.2: Few entries of user table in the SQLite3 database

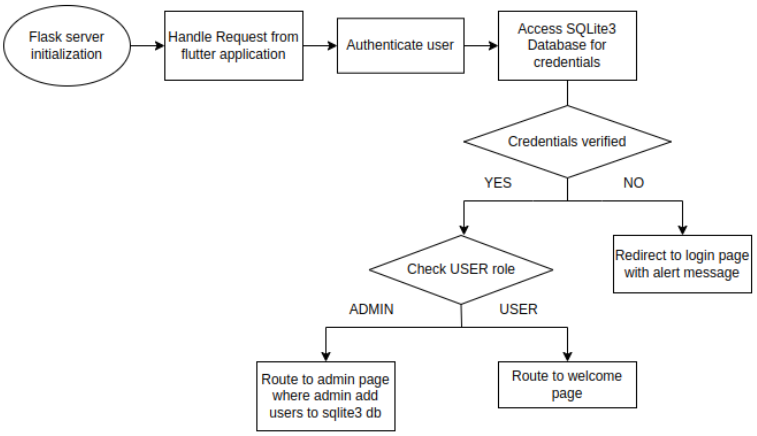


FIGURE 2.3: Flask server and SQLite3 database integration

authentication. An SQLite3 database is integrated into the backend, storing user credentials and roles (Admin or User). When a user logs in, Flask verifies credentials against the SQLite3 database and redirects the user based on their role.

2.2.1 Admin Access

If an Admin logs in, they are routed to an Admin page where they can add or manage users within the SQLite3 database.

2.2.2 User Access

If a regular User logs in, they are taken to a welcome page with an option to interact with the chatbot.

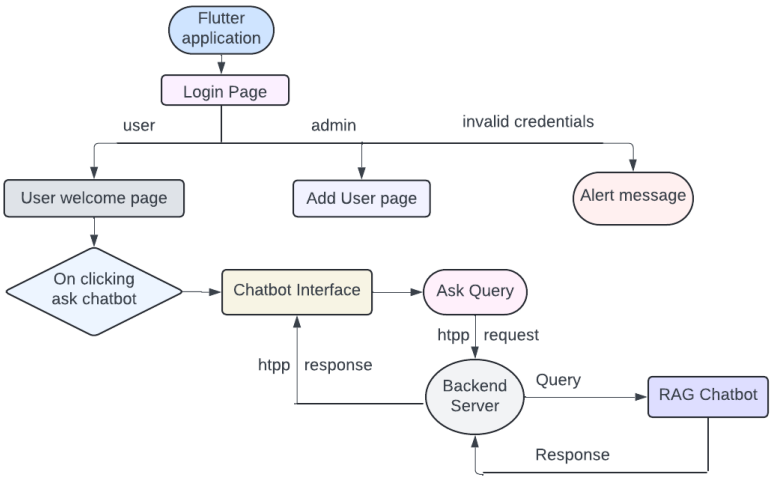


FIGURE 2.4: Frontend-Backend-RAG Chatbot

2.3 Flutter Application and Query Handling

The Flutter frontend offers an intuitive interface for both Admins and Users, with role-specific pathways:

2.3.1 Login and Role-Based Routing

Users start at the login page, where they enter their credentials. Based on their role as stored in the SQLite3 database, users are redirected either to the Admin page or the User welcome page.

2.3.2 Admin Interface

Admins have access to a user management page, where they can add users in the database. This interface allows for streamlined database management within the app itself.

2.3.3 User Interface with Chatbot

Regular users are directed to a welcome page with a button leading to the chatbot interface. Here, users can type queries related to educational content. Once submitted, each query is sent to the Flask server, processed through the RAG pipeline, and the generated response is returned to the app.

2.4 Testing and Evaluation

To ensure optimal functionality and user experience, the application undergoes thorough testing at each stage of development:

Functional Testing: Validates login flows, role-based routing, and response generation to ensure that each feature operates as expected.

Performance Testing: Assesses retrieval and response generation speed, particularly for handling complex or detailed queries in a timely manner.

Usability Testing: Evaluates the user interface for a seamless and intuitive experience, ensuring ease of use for both admins and users.

Chapter 3

Results and Discussions

This outlines the step-by-step user experience within the educational chatbot application, highlighting key functionalities, user flow, and the response handling mechanisms. Screenshots are provided to illustrate the interface at each step, from login through interaction with the chatbot.

3.1 Login Process and Role-Based Access

Upon launching the application, users are directed to a login page where they are required to enter their username and password.

User/Admin Login: Users are prompted to enter valid credentials to proceed. The application distinguishes between regular users and admins based on the role stored in the SQLite3 database.

Invalid Credentials: If a user inputs incorrect credentials, an alert message appears, indicating "Invalid username or password," preventing access until valid information is provided.

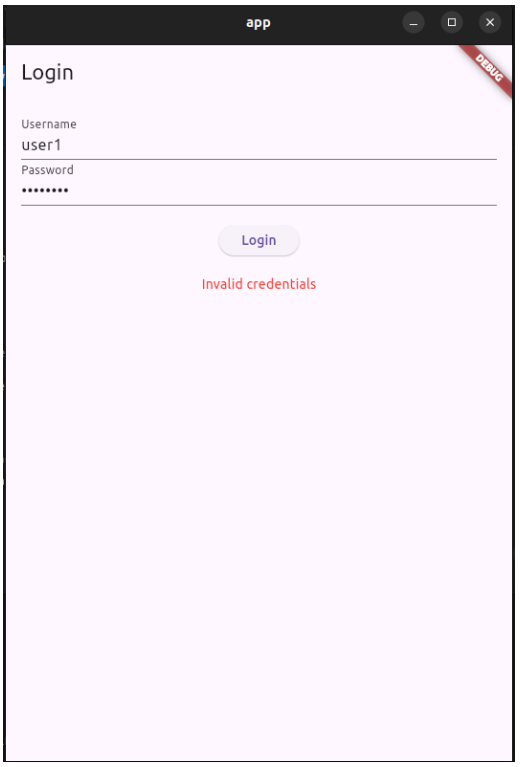


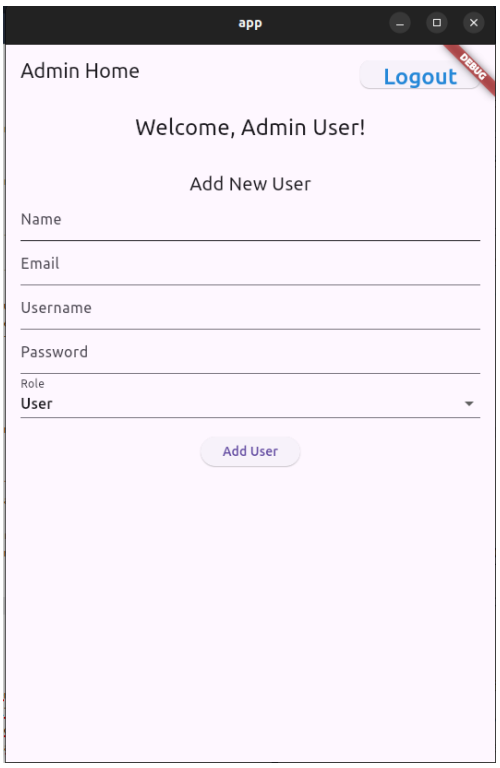
FIGURE 3.1: Error message for invalid login credentials

3.2 Admin Role: User Management Interface

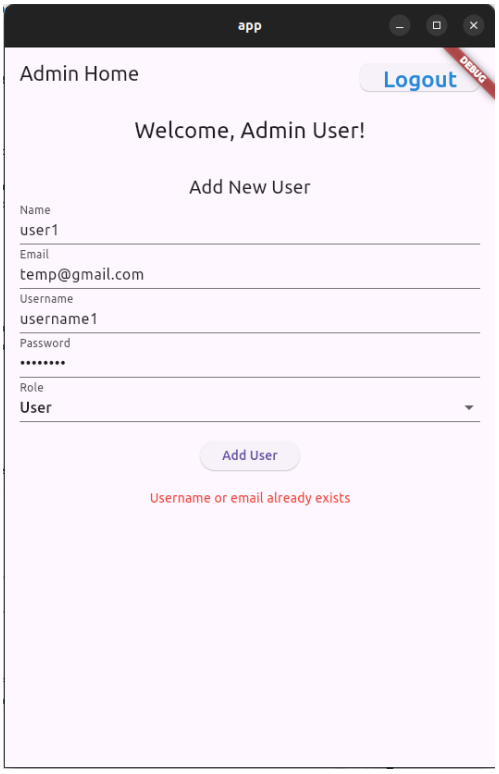
When an admin logs in successfully, they are directed to a user management page. This interface provides admins with options to manage user access, including adding new users.

Adding Users: Admins can input details such as username, email ID, and password for new users. Upon successful addition, a confirmation message appears.

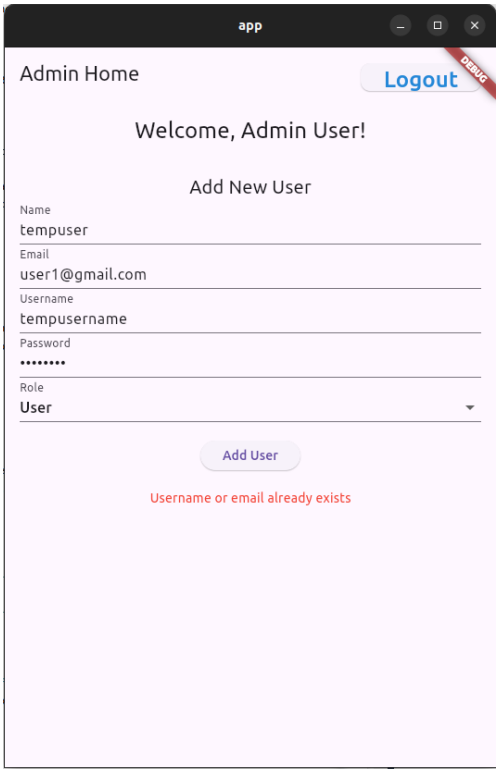
Feedback for Invalid Input: If required fields are left blank or if the username already exists, the application provides a prompt indicating the specific error.



(a) Admin home page



(b) Username exists error



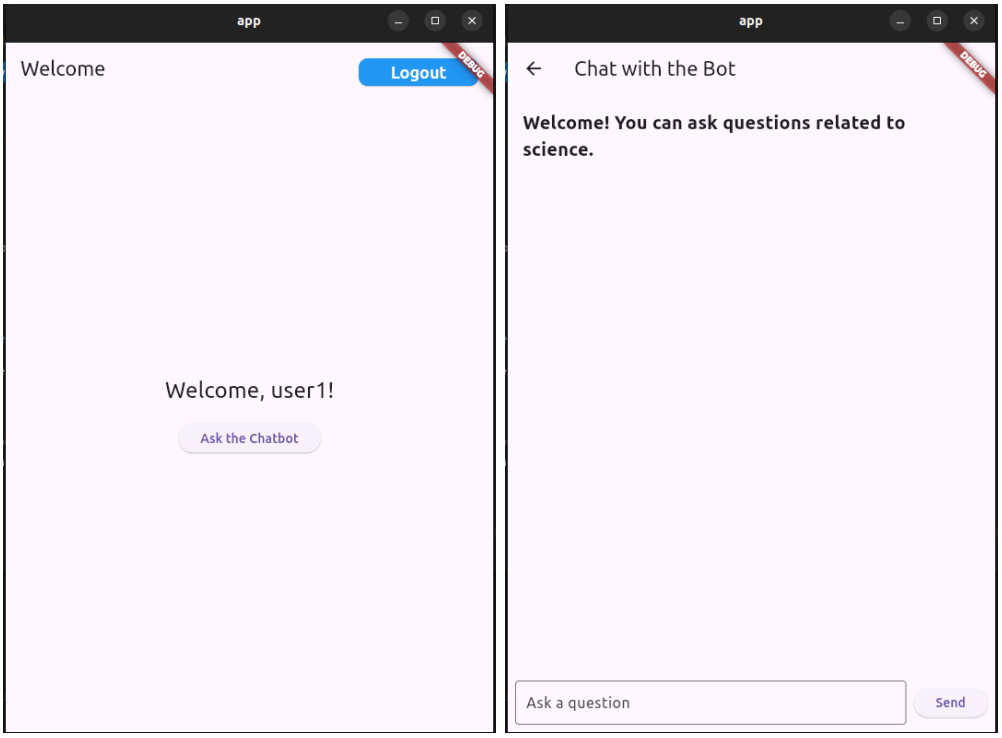
(c) Email exists error

3.3 User Role: Welcome Page, Chatbot Access

For regular users, a successful login redirects them to a personalized welcome page. Here, users can access a button labeled "Ask Chatbot," which directs them to the chatbot interface.

Welcome Message: The page displays a personalized greeting based on the user's name from the database, enhancing user experience.

Navigation to Chatbot: Users can click on "Ask Chatbot" to initiate a session with the chatbot interface, where they can enter queries and receive responses



(d) User home page (e) On clicking "Ask the Chatbot"

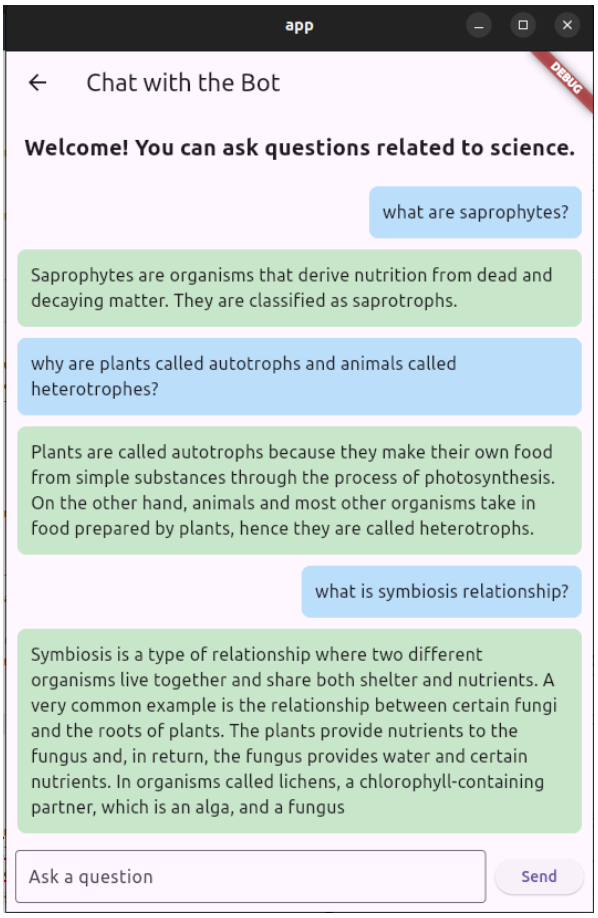


FIGURE 3.2: Conversational flow with the chatbot

3.4 Chatbot Interaction

Upon entering the chatbot interface, users are presented with a text input field where they can type their queries. This section leverages the Retrieval-Augmented Generation (RAG) framework to fetch relevant information from the Pinecone vector database, delivering precise responses based on the input query.

Query Submission: Once the user submits a query, the chatbot retrieves the most relevant information from the database and generates a response.

Conversational Flow: The chatbot maintains conversational flow, displaying past interactions on the interface. User queries appear on right, while the chatbot responses are shown on left, creating conversational experience similar to ChatGPT.

Response Accuracy: Based on the retrieval and generation process, responses are accurate and contextually relevant, addressing the educational needs of the user.

Out Of Context Query Prompts If a user asks a question that is outside the scope of the educational content stored in the database, the chatbot provides a polite response indicating that it cannot answer the query.

Chapter 4

Future Work

While the current chatbot successfully utilizes the RAG framework to provide relevant responses based on external educational content, there are several promising avenues for future development that could further enhance the chatbot's capabilities:

4.0.1 Fine-tuning the LLM with Domain-Specific Data

To improve the accuracy and relevance of responses, future work could focus on fine-tuning the language model used in the RAG framework with domain-specific data. This would involve training the model on a carefully curated dataset relevant to the educational topics covered by the chatbot. By doing so, the LLM could become more adept at generating precise and nuanced responses tailored to the educational domain, reducing dependency on external retrieval and increasing response coherence.

4.0.2 Implementing Contextual Learning

Enhancing the chatbot to learn from previous interactions by retaining user preferences and frequently asked questions could make it more effective. By implementing

contextual learning, the chatbot could refine its understanding of recurring queries and progressively improve its response quality based on user interactions over time.

4.0.3 Multi-lingual Support:

Expanding the chatbot's capabilities to support multiple languages would make it accessible to a broader user base. This could involve training or fine-tuning the model with multilingual data and adjusting the RAG framework to handle retrieval across multiple languages, enabling students from diverse linguistic backgrounds to benefit from the tool.

Bibliography

[Flu] Flutter-Flask-Login/flutter_app/lib/src/repositories/network/api_repository.dart at master · mohammedhashim44/Flutter-Flask-Login — github.com. https://github.com/mohammedhashim44/Flutter-Flask-Login/blob/master/flutter_app/lib/src/repositories/network/api_repository.dart. [Accessed 29-10-2024].

[Vis] Visual Studio Code — docs.flutter.dev. https://docs.flutter.dev/tools/vs-code. [Accessed 29-10-2024].

[Nopwattanapong] Nopwattanapong, K. Build a simple RAG chatbot with LangChain — medium.com. https://medium.com/credera-engineering/build-a-simple-rag-chatbot-with-langchain-b96b233e1b2a. [Accessed 29-10-2024].

[Python] Python, R. Build an LLM RAG Chatbot With LangChain – Real Python — realpython.com. https://realpython.com/build-llm-rag-chatbot-with-langchain/. [Accessed 29-10-2024].