VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi-590018

A MAJOR PROJECT REPORT

ON

"AI Powered Student Assistant Chatbot Using RAG and LangChain"

Submitted in the partial fulfillment of the requirements for the award of

BACHELOR OF ENGINEERING DEGREE

In

COMPUTER SCIENCE & ENGINEERING (DATA SCIENCE)

Submitted by,

DHARSHAN S	(4AD22CD013)
PUNEETH K S	(4AD22CD035)
SANJAY G J	(4AD22CD042)
VIVEK V	(4AD22CD057)

Under the guidance of

Dr. Anitha D B

Associate Professor &HoD

Computer Science & Engineering (Data Science)

Computer Science & Engineering (Data Science)

ATME College of Engineering,

13th Kilometer, Mysore-Kanakapura-Bangalore Road

Mysore-570028

2025-26

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Belagavi-590018

ATME College of Engineering

13th Kilometer, Mysore-Kanakapura-Bangalore Road
Computer Science & Engineering (Data Science)

CERTIFICATE

This is to certify that the project work entitled "AI Powered Student Assistant Chatbot Using RAG and LangChain" is the Bonafide work carried out by the students **Dharshan S (4AD22CD013), Puneeth K S (4AD22CD035), Sanjay G J (4AD22CD042), Vivek V (4AD22CD057)** in partial fulfillment for the award of degree of Bachelor of Engineering in CSE-Data Science from the Visvesvaraya Technological University, Belagavi during the year 2025-2026. It is certified that all the corrections and suggestions indicated for internal assessment have been incorporated in the Major-project report deposited in the department library. The Major-Project report has been approved and satisfies the academic requirement with respect to Major-project work prescribed for Bachelor of Engineering degree.

Signature of Guide	Signature of HOD	Signature of Principal
Dr Anitha D B	**Dr. Anitha D B**	**Dr. L Basavaraj**
Associate Professor & Head	Associate Professor & Head	Principal

External Viva

Name of Examiners	**Signature with Date**
1. …………………..	……………………
2. …………………..	…………………...

ACKNOWLEDGEMENT

The successful completion of our Major-project would be incomplete without the mention of the names of the people who have made it possible. We are indebted to several individuals who have helped us to complete the project report.

We are thankful to **Dr. L Basavaraj**, **Principal**, ATME College of Engineering for having granted us permission and extended full use of the college facilities to carry out this project successfully.

At the outset we express our profound gratitude to our guide **Dr Anitha D B**, Associate Professor & HoD, Department of Computer Science and Engineering (Data Science) for her consistent co-operation and support.

We are greatly indebted to our project coordinator **Dr. Vinod Kumar P,** Associate Professor, Department of Computer Science and Engineering (Data Science) for their timely inquiries into the progress of the project.

Lastly we would like to thank our family and friends for their cooperation and support for success full completion of our project.

DHARSHAN S (4AD22CD013)
PUNEETH K S (4AD22CD035)
SANJAY G J (4AD22CD042)
VIVEK V (4AD22CD057)

DECLARATION

We, the undersigned students, hereby declare that the project entitled "**AI-Powered Student Assistant Chatbot using Retrieval-Augmented Generation (RAG) and Lang Chain**", submitted in partial fulfilment of the requirements for the Degree of Bachelor of Engineering. We further declare that this project has not been submitted to any other university or institution for the award of any degree or diploma. All sources of information used in this report have been properly acknowledged.

Place: Mysuru
Date:

DHARSHAN S
PUNEETH K S
SANJAY G J
VIVEK V

ABSTRACT

In modern academic environments, students often struggle to access accurate and timely information related to courses, examinations, schedules, and institutional resources. Traditional search methods and static chatbots fail to deliver context-aware responses, especially when information is scattered across documents such as syllabi, PDFs, notices, and timetables. To address this challenge, this project presents an AI-powered Student Assistant Chatbot built using the Retrieval-Augmented Generation (RAG) architecture integrated with the LangChain framework and MERN stack.

The system combines document retrieval and large language model (LLM) generation to deliver precise, factual, and context-grounded answers. Academic documents are uploaded, processed into embeddings, and stored in a vector database to enable semantic search. When a student asks a question, the system retrieves the most relevant text chunks and generates accurate, user-friendly responses using the Gemini API. The platform also provides additional features such as quiz generation, study plan creation, document upload, chat history storage, and an admin dashboard for managing academic resources.

The proposed system significantly improves information accessibility, reduces manual workload for faculty, and enhances the overall learning experience for students. Through an intelligent RAG-based pipeline and a user-friendly interface, this chatbot provides a scalable, reliable, and modern solution for academic support within educational institutions.

Keywords- RAG, LangChain, Student Chatbot, LLMs, Vector Database, Semantic Search, MERN Stack

CONTENTS

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1 Preamble

Technical documentation is essential for developers, engineers, and end users to comprehend and make use of complicated software, hardware, and digital services in today's technologically advanced world. Finding accurate answers quickly is challenging, though, due to the sheer volume and complexity of technical manuals, API references, and troubleshooting guides. Knowledge discovery is frequently inefficient due to fragmented or irrelevant results from traditional search engines and keyword-based retrieval systems.

Retrieval-Augmented Generation (RAG), which combines retrieval-based and generative approaches, provides a solid solution to the growing need for intelligent question-answering systems in educational settings. This project uses the LangChain framework to implement a RAG-based chatbot that allows students to ask questions about college life, academics, and campus services and receive precise, context-aware answers. The system guarantees that responses are based on reliable, institution-specific data by combining a vector database, document retriever, and large language model (LLM). This greatly increases the relevance and dependability of information given to students.

A popular framework for putting such systems into practice is called Langchain, which combines custom knowledge sources with large language models. It is appropriate for creating instructional chatbots since it enables developers to link LLMs with vector stores and retrieval pipelines.

Finding precise answers to inquiries concerning classes, schedules, facilities, and other academic resources can be challenging for college students. In order to address this, we created an AI-powered chatbot that functions as a virtual assistant for students using Retrieval-Augmented Generation (RAG) and Lang Chain. The chatbot uses a language model to provide concise, informative answers after retrieving data from official college documents. The goal of this project is to improve the overall campus experience for students, lessen reliance on manual support, and streamline information access.

Students' access to information and academic support is changing as a result of the growing use of artificial intelligence in education. With the proliferation of online resources and digital learning materials, students frequently find it difficult to swiftly discern pertinent information from vast amounts of disorganized content. Because they rely on predetermined rules or responses, traditional chatbots are unable to comprehend complex academic queries or retrieve information from dynamic college documents, which limits their ability to provide assistance.

This project presents an AI-Powered Student Assistant Chatbot that uses the LangChain framework and Retrieval-Augmented Generation (RAG) to close this gap. RAG enables the system to integrate real-time retrieval from college-related documents, including syllabi, PDFs, circulars, and FAQs, with the reasoning power of Large Language Models (LLMs). By combining vector databases, embeddings, and document processing modules, Lang Chain facilitates this process and allows for precise and context-aware responses. The objective of this project is to develop an intelligent, dependable, and scalable academic assistant that improves overall learning efficiency, saves students' manual labor, and offers immediate answers.

1.2 Motivation

In modern academic institutions, students are frequently overwhelmed by the need to navigate through large volumes of scattered, often complex information ranging from course details to administrative services. The traditional modes of seeking assistance, such as FAQs, forums, or manual support desks, are often time-consuming and inefficient.
The motivation behind this project arises from the pressing need to simplify and enhance the way students access institutional knowledge. By leveraging the power of Retrieval-Augmented Generation (RAG) and frameworks like Lang Chain, this project aims to build a virtual assistant that can understand user queries, retrieve contextually relevant information from official documents, and generate accurate, conversational responses.

The goal is to reduce the cognitive load on students, minimize delays in obtaining critical information, and improve the overall efficiency of student services. By integrating advanced AI models with institution-specific data, the chatbot promises to transform the traditional information retrieval process into a more intuitive and intelligent experience ultimately contributing to a smarter and more student-friendly academic environment.

1.3 Problem statement

AI Powered Student Assistant Chatbot Using RAG and LangChain with this rapid advancement of Generative AI and Large Language Models (LLMs) has transformed the interaction between humans and intelligent systems, making conversations more natural, contextual, and reliable. In educational environments, students often struggle to access timely information about courses, faculty, facilities, and academic processes. Traditional rule-based chatbots are limited because they rely on fixed responses and cannot handle large or frequently updated academic data. To address these challenges, Retrieval-Augmented Generation (RAG) combines the strengths of LLMs with real-time information retrieval, enabling the system to fetch relevant content from external sources and generate accurate, fact-based responses.

This project focuses on developing an AI-Powered Student Assistant Chatbot using RAG and LangChain, designed specifically for academic institutions. LangChain provides powerful tools for integrating vector databases, retrievers, embeddings, and document loaders, enabling the chatbot to access and understand college-related documents such as syllabi, notices, FAQs, and PDFs. By retrieving semantically relevant information and generating context-aware answers, the system aims to reduce the workload on administrative staff, improve student engagement, and offer an efficient, scalable solution for delivering instant academic support.

1.4 Objectives of the Project

The proposed RAG-Based Student Chatbot using Lang Chain aims to build an intelligent and responsive academic assistant capable of answering student queries with contextual and document-based information.

The main objectives of the project are as follows

i. **Design a Conversational AI System Using RAG**: Implement a chatbot architecture that combines the power of Large Language Models (LLMs) with context-specific technical documentation using Retrieval-Augmented Generation

ii. **Integrate Lang Chain Framework for Modular Pipeline:** Use LangChain modular components to build a scalable and customizable chatbot pipeline.

iii. **Build an Interactive User Interface:** Develop a user-friendly chat interface using tools for real-time question answering and document interaction.

1.5 Organization of the Report

This project report is divided into eight chapters, with each chapter explaining an important part of the development of the AI-Powered Student Assistant Chatbot. The structure of the report is as follows:

Chapter 1 - Introduction: This chapter presents an overview of the project by discussing the background, motivation, problem statement, objectives, and organization of the report.

Chapter 2 - Literature Survey: This chapter reviews existing research works, RAG frameworks, chatbot architectures, LLM-based systems, and prior educational chatbots. It summarizes 14 academic papers and other Documentation of the resources, comparing their approaches, strengths, limitations, and contributions. The chapter highlights how insights from previous studies influenced the design of this system.

Chapter 3 - System Requirements and Specification: This chapter outlines the hardware and software requirements necessary to build and deploy the chatbot. It describes the tools and a technology used including MERN, Lang Chain, MongoDB, AWS S3, and Gemini AI and details the functional and non-functional requirements that guided system development.

Chapter 4 - System Analysis and Design: This chapter presents the overall system analysis, feasibility study, software architecture, and module-level design. It explains how students and admins interact with the system, provides diagrams such as system architecture, database schema, and use-case models.

Chapter 5 - Methodology: This chapter details the step-by-step methodology of the project, including frontend development, backend development, AI pipeline integration, document processing, RAG workflow, database design, and overall system implementation.

Chapter 6 - Testing: This chapter describes the testing techniques applied to validate the system, including unit testing, system testing, integration testing, and validation testing. It provides a module-wise test table showing test cases.

Chapter 7 - Experimentation & Results: This chapter presents the results of the developed chatbot, including screenshots of the student dashboard, admin dashboard, chat interface, quiz module, study planner, and backend server.

Chapter 8 - Conclusion and Future Scope: This chapter summarizes the overall achievements of the project and highlights how the system improves academic support using AI. It also lists future scope of the project.

CHAPTER 2

LITERATURE SURVEY

2.1 Introduction

The goal of this study is to understand how existing RAG frameworks and chatbot architectures can be leveraged to build an intelligent college assistant chatbot. Such a chatbot aims to assist students by answering academic queries, providing course-related information, and facilitating administrative support using institution-specific documents like syllabus, timetables, and FAQs. By analyzing existing works and tools, we aim to identify best practices, current limitations, and potential improvements for implementing a robust and efficient student assistance chatbot using Lang Chain and RAG methodologies.

2.2 Literature Review

The increasing complexity of student information systems in educational institutions has led to a growing interest in intelligent conversational agents that can simplify access to academic and administrative information. One of the most promising approaches in this domain is Retrieval-Augmented Generation (RAG), which combines the strengths of traditional information retrieval with the generative capabilities of large language models.

The paper titled **"Design and Implementation of LangChain-based Chatbot"** by Huaxiao Zhang, Zhigang Li, and Yunbo Zheng describes a chatbot built using Lang Chain, integrated with LLMs and local knowledge sources like PDFs and videos.The system uses Retrieval-Augmented Generation (RAG), combining user queries with relevant content from a vector database (via OpenAI embeddings and FAISS).It also enables video retrieval through subtitle matching and can play specific segments. The chatbot generates test questions from prior chats and maintains user context using LangChain memory. A Gradio interface makes it user-friendly.It limits only on depends on cloud-based LLMs, which may pose privacy and data security risks.Future work aims to use local LLMs to enhance privacy and reduce external dependents [1].

The paper titled **"Development of Interactive Assistance for Academic Preparation Using Large Language Models"** by Kumar P, Haresh M and et al presents a conversational learning platform designed to assist students through the integration of large language models (LLMs) deployed locally.The system is built using open source tools such as Ollama, LangChain, and Meta’s LLM 3 model, with the aim of avoiding the limitations and privacy concerns associated with closed-source APIs like OpenAI's ChatGPT. The architecture comprises four key modules: local LLM deployment using the lightweight 7B LLM model; vector embedding of educational content using FAISS and Milvus databases; a backend API developed with Flask and LangChain to handle queries; and a web-based user interface for interaction. The model uses Retrieval-Augmented Generation (RAG) to dynamically retrieve relevant information from embedded content, allowing it to provide accurate, context-aware responses [2].

The paper titled **“A Client-Server Based Educational Chatbot for Academic Institutions”** by Rohan Paul Richard, Joel Mathew et al presents a robust solution for enhancing educational support through AI-driven technologies. The authors propose a chatbot system built on a client-server architecture that utilizes Large Language Models (LLMs) combined with Retrieval-Augmented Generation (RAG) to provide accurate and context-aware responses to student queries. By integrating trusted academic resources such as PDFs and lecture slides into a vector store, the chatbot ensures the reliability of its outputs. The system leverages LangChain for orchestrating interactions between the user, model, and knowledge base, and uses FastAPI to build scalable APIs. A key innovation in the system is the use of the Mixture of Experts model, which enhances performance and efficiency through specialized model routing. The application architecture also features a secure JWT-based authentication system, making it stateless and scalable. The chatbot can be deployed within an institution’s infrastructure and is designed to support multiple concurrent users. Experimental results show that the system is effective in delivering informative and context-rich responses, making it a valuable tool for both students and educators.The paper concludes by highlighting future enhancements, including support for multimodal inputs to further improve the interactivity and educational value of the system [3].

The paper titled **“An Interactive Question-Answering System using Large Language Model and Retrieval-Augmented Generation in an Intelligent Tutoring System on the Programming Domain”** by Owen Christian Wijaya and Ayu Purwarianti presents the development of a web-based intelligent tutoring system (ITS) to enhance student engagement in programming education. Addressing the lack of interaction in current learning platforms, the authors propose a QA system combining Large Language Models (LLMs) with Retrieval-Augmented Generation (RAG) to deliver contextual, accurate responses based on user queries and chat history. The system is built using the LangChain framework, integrated with a history-aware retriever, a vector database, and a local Redis-based memory for persistent chat logs. The authors also apply CO-STAR prompt engineering to guide the model’s tone, audience, and response structure. Multiple LLMs, including Llama 3, Zephyr, and Code Gemma, are evaluated on qualitative metrics such as faithfulness, helpfulness, and friendliness. Results indicate that Llama 3 consistently performs best across single-turn and multi-turn conversations. The study concludes that combining RAG with LLMs significantly enhances the QA experience and suggests broader application in intelligent tutoring environments [4].

The paper titled **“Unleashing AI in Education: A Pre-Trained LLMs for Accurate and Efficient Question-Answering Systems”** by Josie C. Calfoforo et al represents a novel question answering (QA) system that integrates RAG, the LangChain framework, and the large language model (LLM) to enhance information access for faculty and staff at Iloilo Science and Technology University, Philippines. The system is specifically designed to answer policy-related questions using documents like handbooks and FAQs in PDF format. To improve efficiency and reduce hardware requirements, the authors employed the Quantized Low-Rank Adapter technique, enabling 4-bit fine-tuning of the model while maintaining high performance. Key components of the system include PDF text extraction, document chunking, embedding generation using the Sentence Transformer all-mpnet-base-v2, and vector storage via FAISS. The study demonstrates significant improvements in user satisfaction, accuracy, and speed of information retrieval when compared to traditional manual methods. Experimental results showed enhanced performance after fine-tuning, with reduced training loss and efficient runtime metrics. The system also supports conversational queries, maintains chat history for context, and returns source documents with each answer. The authors conclude that combining LLMs, RAG, LangChain can revolutionize support systems in academic environments and propose developing a web-based interface as the next step. Future work may incorporate sentiment analysis, domain-specific adaptation, and advanced NLP techniques to expand applicability across sectors like healthcare, legal services, and customer support [5].

The paper titled **“RAG based Chatbot using LLMs”** by Ananya G and Dr. Vanishree K presents the development of a chatbot that leverages Retrieval-Augmented Generation (RAG) combined with Large Language Models (LLMs) to answer queries based on PDF content. The chatbot integrates tools like LangChain, PyTorch, and Hugging Face Transformers, specifically using models like Llama2 and GPT-3.5. The system extracts and preprocesses data from PDF files, segments the text, and converts it into semantic embeddings. These embeddings are stored in vector databases (e.g., FAISS) and retrieved based on user queries using semantic similarity. A web-based interface built with Chainlit allows real-time interaction with users. The chatbot performs context-aware searches and generates human-like responses. The project shows how RAG can significantly enhance response accuracy and relevance by grounding the LLM with external documents. The model also employs reinforcement learning to optimize token usage and reduce computational costs. Results confirm improved performance in query resolution across uploaded PDFs. The framework demonstrates a scalable solution adaptable to various domains, including corporate and academic sectors. By combining retrieval and generation, the system bridges the gap between unstructured data and meaningful user interaction. The approach also supports future extensions like domain adaptation and multilingual capabilities. Overall, the project showcases the effectiveness of modern NLP in building intelligent and responsive chatbot systems [6].

The paper titled **“ScrapeTalk: Chatbot Conversation with Web Scraped Insights”** by Sriram G, Adaline Suji R, et al introduces ScrapeTalk, an advanced AI chatbot designed to enhance information retrieval using web scraping and natural language processing (NLP).

The system integrates multiple technologies including LangChain, OpenAI API, Streamlit, BeautifulSoup4, WebBaseLoader, and ChromaDB to create a conversational interface capable of answering user queries with real-time, web-sourced insights. Built on the Retrieval-Augmented Generation (RAG) methodology, ScrapeTalk retrieves and processes contextually relevant data from across the internet, improving the accuracy and relevance of its responses. The use of OpenAI's GPT-3.5 Turbo enhances the chatbot's conversational intelligence, while ChromaDB ensures efficient storage and access to scraped data. The paper emphasizes the importance of responsive UI through Streamlit and demonstrates how ScrapeTalk can save users significant time by replacing manual web searches with dynamic, context-aware dialogue. Designed to be scalable and continuously improving, ScrapeTalk showcases the powerful intersection of AI, web data, and user-focused design in modern conversational systems [7].

The paper titled **"Enhancing Question-Answering with Knowledge Graph Retrieval and Generation using LLMs"** by Supraj Gijre, Priyash Laddha et al presents an innovative approach to improving large language models (LLMs) for document-based question answering. The proposed system integrates knowledge graphs with Retrieval-Augmented Generation (RAG) to enhance the context awareness, accuracy, and reliability of responses. The methodology involves chunking long documents, generating vector embeddings using Google's Vertex AI, and structuring them into a knowledge graph with relationships like "NEXT," "PART-OF," and "SECTION." This structure helps maintain context and semantic relationships. When a user asks a question, the system retrieves the most relevant nodes and surrounding context from the graph, feeding them into an LLM for accurate response generation. The backend uses LangChain for contextual processing and Neo4j for graph database management. Evaluation through the RAGAS framework shows high scores in faithfulness (1.0), answer relevancy (0.83), and context recall (0.99). The research demonstrates that combining knowledge graphs with LLMs can significantly improve performance for domain-specific, complex document analysis. The system is also integrated with a user-friendly Streamlit interface. Future work includes refining query refinement capabilities and expanding to industries like law, finance, and healthcare [8].

The paper titled **“Retrieval Augmented Generation Approach for Multipdf Chatbot using LangChain”** by Anuj Jaiswal, Aakash Jha et al explores the development of an AI-driven chatbot system capable of processing and interacting with multiple PDF documents. The authors present a novel application that facilitates seamless information retrieval and conversational interactions with content extracted from uploaded PDFs. Addressing challenges associated with manual parsing and limited contextual understanding in existing tools, the study integrates advanced technologies such as Google's Generative AI, LangChain, FAISS, and Streamlit. The methodology involves extracting text using PyPDF2, segmenting it via RecursiveCharacterTextSplitter, embedding it through Generative AI, and indexing it with FAISS for rapid semantic search. A conversational chain is built using the Gemini 1.0 Pro model to interpret and respond to user queries contextually. The interface, created in Streamlit, enables real-time chat interactions and preserves chat history for reference. The paper reviews prior works on RAG systems and identifies limitations in contextual understanding, scalability, multi-modal integration, and user feedback mechanisms. This work addresses those gaps by combining document processing with a chatbot that enables dynamic querying. Results highlight the system’s efficiency, user-friendliness, and potential for diverse applications in academic and professional domains. Limitations such as dependency on document quality and potential model biases are acknowledged. Overall, the paper marks a significant step in improving knowledge accessibility through intelligent, AI-powered PDF interaction [9].

The paper titled **"Retrieval-Augmented Generation (RAG) Chatbots for Education: A Survey of Applications"** by Jakub Swacha and Michał Gracel provides a comprehensive overview of how RAG-based chatbots are applied in the educational domain. It highlights that RAG architecture helps mitigate hallucinations in LLMs by retrieving accurate external knowledge, making it suitable for educational use. The authors reviewed 47 works published between 2023 and early 2025, categorized by their target support such as learning, organizational support, or access to domain knowledge. The study found that most chatbots were designed to assist students directly, while a few catered to teachers or administrative functions. Thematic areas included health sciences, computer science, law, and language learning, with OpenAI's GPT being the most widely used. Evaluations varied, involving both automated metrics and human feedback. Notably, the RAGAS framework was mentioned as a promising but underutilized evaluation tool. The study emphasizes the lack of performance assessments tied directly to learning outcomes, calling this a critical research gap. Furthermore, the paper recommends future work focus on rigorous, goal specific evaluations. It also underscores the practical and ethical challenges in deploying such systems, including cost and data privacy. The work concludes by mapping current research directions and offering a framework for future studies to build upon this contributes significantly to understanding the role of AI-powered chatbots in enhancing education [10].

This paper titled by **"Optimizing RAG techniques for Automotive industry PDF Chatbots: A case Study with Locally Deployed Models"** Kaiyi Zhang et al. investigates whether Large Language Model (LLM) agents truly understand the prompts they are given. The authors approach this question using a reasoning-theoretic framework, focusing on how models interpret and respond to instructions. They analyse LLM behaviours across different tasks, testing whether models adhere to logical and semantic rules implied by prompts.The study introduces formal reasoning principles and evaluates LLMs based on their consistency and soundness in prompt comprehension. Experiments reveal that while LLMs often generate correct answers, they may fail to follow prompt logic, especially in complex or multi-step tasks. This suggests a superficial understanding rather than genuine reasoning. The authors argue for more rigorous evaluation strategies to assess prompt comprehension. Their findings imply that current LLMs may lack deep understanding, even when performance appears strong. The paper contributes important insights into AI interpretability and the future development of reliable LLM agents [11].

This paper titled **"From Questions to Insightful Answers: Building an Informed Chatbot for University Resources"** by subhash Neupane et all presents the design and implementation of an intelligent chatbot tailored to help university students access campus resources efficiently. The chatbot is developed to address common queries related to academic services, administrative procedures, and campus facilities. The authors focus on building a system that not only retrieves information but provides contextually insightful and user-friendly responses. The architecture integrates natural language processing (NLP), information retrieval, and domain-specific knowledge bases to handle diverse student queries. A modular approach is used, allowing the chatbot to be updated easily with new resource data. The system is evaluated through real-world testing with students, showing high satisfaction rates and improved access to information. To enhance answer quality, the chatbot employs ranking and filtering techniques to prioritize relevant responses. It also supports follow-up questions, enabling a more natural conversational flow. The authors highlight the importance of transparency and reliability in educational AI systems and propose future enhancements, including multilingual support and integration with campus mobile apps [12].

This paper titled by **"Unimib Assistant: Designing a student-friendly RAG-Based chatbot for all their needs"** by Chiara Antico, et all presents the design and development of a university focused chatbot that delivers insightful and context-aware answers to student inquiries. Authored by Safa AbuJarour et al the work aims to bridge the gap between students and the vast array of academic and administrative university resources.
The proposed chatbot uses advanced natural language processing techniques to understand user questions and retrieve the most relevant information. It is specifically tailored for educational environments, providing accurate, timely, and helpful responses related to admissions, course registration, campus services, and more. The system is also capable of handling ambiguous queries by engaging in follow-up questions to clarify user intent. Built with a modular and scalable architecture, the chatbot integrates with existing university data systems, allowing real-time updates and customization.
The paper details a successful pilot implementation and evaluation, which shows high levels of user satisfaction and efficiency gains. The authors highlight its potential as a digital assistant for improving student and administrative support. Future enhancements include multilingual support, improved personalization, and AI-driven insights for proactive engagement [13].

The paper titled **“FACTS About Building Retrieval Augmented Generation-based Chatbots”** by Sebastian Gehrmann et al presents a comprehensive overview of the challenges and best practices in developing Retrieval-Augmented Generation (RAG) chatbots. It highlights how the effectiveness of a chatbot heavily depends on the quality of the retrieval system, as it directly influences the relevance and accuracy of generated responses. The authors analyse the impact of various retrievers such as dense and sparse models and encoding strategies like chunk size and overlap, which can significantly affect performance. Prompt engineering is discussed as a key factor in ensuring that responses are well-grounded in the retrieved content. The paper investigates how proper grounding reduces hallucinations in generative models and emphasizes the importance of carefully balancing brevity, completeness, and factuality in chatbot responses. Through extensive experimentation, the authors reveal common failure points in retrieval, ranking, and generation components. They also warn about the risks of feedback loops, where generated answers may bias future retrievals. To measure performance, the study proposes new evaluation metrics focused on answer grounding and factual consistency. Additionally, the authors introduce a benchmark dataset named FACTS to assess RAG chatbot systems across various tasks. Ultimately, the paper advocates for an end-to-end optimization approach that considers the interactions between all system components, aiming to build more reliable and trustworthy chatbots [14].

2.3 Outcome of the literature review

The literature reviewed reflects a rapidly growing interest in the application of Retrieval-Augmented Generation (RAG) combined with Large Language Models (LLMs) for developing intelligent, context-aware chatbot systems. A common trend across the works is the adoption of LangChain as a versatile orchestration framework that seamlessly integrates retrieval, generation, memory, and external tools, making it central to most implementations.

The literature survey shows that Retrieval-Augmented Generation (RAG) is one of the most effective methods for building accurate and reliable chatbots, especially in education. Existing research highlights that traditional rule-based chatbots cannot handle large or unstructured academic data, while RAG-based systems can retrieve information directly from documents and provide fact-based answers. Studies also show that LangChain is widely used because it simplifies the development of RAG applications through its tools for document loading, vector storage, embeddings, and retrievers.

The survey further reveals that the accuracy of a RAG chatbot depends mainly on the quality of document processing, chunking strategies, and retrieval performance. Overall, the literature supports that a RAG + LangChain approach is well-suited for creating a student assistant chatbot, as it improves response accuracy, reduces hallucinations, and provides more personalized and efficient academic support.

CHAPTER 3

SYSTEM REQUIREMENTS AND SPECIFICATION

3.1 Hardware Requirements

The hardware requirements specify the physical resources needed to develop and run the AI-Powered Student Assistant Chatbot effectively. At a minimum, the system requires a computer with 8 GB RAM, a dual-core processor, and around 20 GB of free storage to handle essential tasks such as document uploading, text extraction, and basic API operations. A stable internet connection is also necessary since the chatbot relies on external LLM APIs for generating responses. For better performance, especially when working with larger documents or multiple users, it is recommended to have 8–16 GB RAM, a quad-core processor, and SSD storage. These enhanced specifications ensure faster document embedding, quicker retrieval from the vector database, and smoother multitasking during development. Overall, the project does not require heavy hardware or GPUs because all AI model computations occur on cloud-based APIs, making the system lightweight and easy to run on standard machines.

3.2 Software Requirements

The AI-Powered Student Assistant Chatbot requires several software tools to function smoothly. The frontend of the system is developed using React.js, which provides an easy and interactive interface for both students and admins. Node.js and npm are used to run the frontend and manage the required libraries. The backend is built using Node.js with Express.js for handling APIs, authentication, PDF uploads, and communication with the AI engine. MongoDB serves as the main database for storing user details, extracted document content, and chat history, while Mongoose helps manage database operations more easily.

For AI and RAG processing, the system uses LangChain to load documents, create embeddings, and retrieve relevant information. Vector databases such as ChromaDB or FAISS store these embeddings for fast similarity search. The chatbot also connects to LLM

APIs like Google Gemini for generating accurate, context-based answers. Additional tools like PyMuPDF are used for PDF text extraction, while VS Code, GitHub, and Postman support development, testing, and version control. These software components together create a reliable and efficient environment for building and deploying the chatbot.

3.3 Tools and Technologies Used

The AI Student Assistant system is built using a combination of modern web technologies, cloud services, and advanced AI frameworks to deliver an intelligent and scalable learning platform. The project follows the MERN stack architecture, integrating Node.js and Express.js in the backend, MongoDB for database management, and React.js for a responsive and user-friendly frontend. Real-time communication is enabled using Socket.IO, allowing instant messaging between the student and the AI assistant. The backend also incorporates LangChain to implement the Retrieval-Augmented Generation (RAG) process, which handles document chunking, vector embedding creation, semantic search, and context-based AI responses. For storing uploaded documents securely, AWS S3 is used, while JWT authentication ensures secure user access.

LangChain is an open-source framework designed to build applications powered by Large Language Models (LLMs). It provides structured components that simplify the development of advanced AI systems such as chatbots, RAG pipelines, intelligent agents, and document-based question-answering systems. LangChain acts as a bridge between LLMs and real-world data, allowing developers to integrate models with external tools, databases, documents, and APIs. Instead of prompting an LLM directly every time, LangChain helps build modular, reusable, and scalable AI pipelines.

Amazon S3 (Simple Storage Service) is used in the system for securely storing all uploaded files such as PDFs, DOCX, PPTs, and study materials. When a student or admin uploads a document, it is directly stored in an S3 bucket, and a secure file URL is generated. This URL is saved in MongoDB and used by the backend whenever the document needs to be processed by the RAG engine. S3 provides high scalability, fast file access, and reliable data storage, making it ideal for handling large academic files without putting load on the main server.

Socket.IO is used in the system to enable real-time communication between the React frontend and the Node.js backend. It provides instant message transfer, allowing students to receive AI-generated responses immediately without refreshing the page.In the chatbot module, Socket.IO ensures smooth, live chat interactions by sending user queries and receiving AI responses instantly. This creates a fast, interactive, and engaging chatbot experience similar to modern messaging apps.

On the frontend, react 18, React Router, and Axios manage user interactions, routing, and API requests, supported by CSS animations and Lucide icons for a polished user experience. The AI and machine learning functionalities are powered by the Gemini, which generates embeddings, processes uploaded documents, and supports semantic search.

MongoDB and Mongoose manage all backend data, including user information, chatlogs, MCQ exams, and study materials. Development tools like Visual Studio Code**,** Postman**,** Git**,** and GitHub help streamline development, API testing, debugging, and version control. Overall, this combination of tools and technologies ensures efficient data handling, accurate AI responses, secure document manager.

The development of the AI Student Assistant system is supported by a combination of modern web technologies and AI tools that ensure smooth performance, intelligent responses, and efficient data management. As shown in Table 3.1, the project utilizes React and CSS3 on the frontend to create an interactive user interface, while Node.js and Express.js manage the backend logic and API handling.

MongoDB is used as the primary database for storing student data, documents, and chat history. The intelligence of the system is powered by LangChain and the Hugging Face API, which enable Retrieval-Augmented Generation (RAG) and semantic search. AWS S3 provides secure file storage, and development tools such as VS Code, GitHub, and Postman support coding, version control, and API testing throughout the project.

Table 3.1: Tools and Technologies Used in the System

Category	**Technology**	**Description**
Frontend	React, CSS3	Builds the user interface.
Backend	Node.js, Express.js	Handles server logic and APIs.
Database	MongoDB	Stores user data and documents.
AI / ML	LangChain, Hugging Face API	Provides intelligent responses using RAG.
File Storage	AWS S3	Stores uploaded study files.
Tools	VS Code, GitHub, Postman	Used for coding, version control, and API testing.

3.4 Functional Requirements

The functional requirements explain what the system should do and how it should work for both students and admins. These requirements describe the main features of the chatbot, such as logging in, uploading documents, asking questions, viewing chat history, and managing content. They help ensure the system performs all the necessary tasks needed for smooth operation. The functional requirements of the system are as follows

a) **User Authentication**: The system must allow both students and admins to register or log in securely. Role-based access control ensures users only access their respective dashboards. Passwords and credentials should be validated before granting access. The system should prevent unauthorized access through proper session handling. Logged-in users must remain authenticated throughout their interaction.

b) **Student Dashboard**: Students should be able to ask academic questions directly from the dashboard. They must be able to upload PDFs to receive answers based on the content. The dashboard should show recent chats and previously answered questions. A clean and simple UI should help students navigate without confusion. All interactions should be stored to improve the overall learning experience.

c) **Admin Dashboard**: Admins should manage students, documents, and chatbot usage. They must be able to view system statistics, performance, and logs. Admin panel should allow updating academic documents used for training. Admins can review chat histories or remove irrelevant content. It ensures full control over the system's backend operations.

d) **PDF Upload & Text Extraction**: Students can upload PDFs related to academic subjects. The system extracts text using tools like PyMuPDF or pdf-parse. Extracted text must be processed into smaller chunks for retrieval.These chunks are converted into embeddings stored in the vector DB. The feature ensures accurate knowledge retrieval during question answering.

e) **Retrieval-Augmented Generation (RAG)**: The RAG pipeline retrieves the most relevant content from the database. The LLM uses this retrieved context to generate accurate answers. This ensures responses are not only AI-generated but context-driven. Helps reduce hallucinations and improves answer accuracy. It forms the core intelligence behind your chatbot.

f) **Chat History Storage**: Every question asked by a student must be saved in the database. Corresponding answers should also be stored for future reference. History helps students revise academic topics previously covered. The system should allow easy viewing of past conversations. This feature enhances continuity and learning effectiveness.

g) **Notifications & Error Handling:** System must display clear messages for invalid PDF uploads or unsupported formats. It should also notify users about missing inputs or system errors. Error messages should guide the user on how to correct the issue. Smooth handling prevents crashes and maintains system stability. Notifications should be user-friendly and easy to understand.

3.5 Non-Functional Requirements

The non-functional requirements describe how the system should behave and the quality it must maintain. These include performance, security, usability, reliability, and compatibility. They make sure the system is fast, safe, easy to use, and works well on different devices without issues. The non-functional requirements of the system are as follows

a) **Performance**: The system should return chatbot responses within 2–4 seconds. Vector search should be optimized to retrieve results quickly. Performance must remain stable even with large documents. Backend APIs should handle requests efficiently. Smooth performance ensures a positive user experience.

b) **Usability:** The interface should be simple, intuitive, and easy for students to use. Navigation should be straightforward with clear buttons and instructions. Uploading PDFs and asking questions must be hassle-free. UI elements must be designed for both technical and non-technical users. The goal is to reduce confusion and enhance productivity.

c) **Scalability:** The system should support multiple students using it simultaneously. More PDFs and documents should be handled without slowing down. Database and vector DB must scale as academic content grows. Backend architecture should allow adding new features easily. Scalability ensures the system remains future-ready.

d) **Security**: All user credentials must be encrypted and stored securely. Only authenticated users should gain access to dashboards. Admin privileges must be protected from unauthorized access. APIs should use secure communication methods like HTTPS. Security ensures trust and protects sensitive information.

e) **Reliability:** The system must work consistently without frequent downtime. Features like PDF upload and chat generation should work smoothly. Server failures should not result in loss of chat history or data. Proper backups must be maintained to restore information if needed. A reliable system increases user satisfaction and trust.

f) **Maintainability**: The codebase should follow modular and well-structured patterns. Developers should be able to update or fix modules easily. Documentation must support maintenance activities Admin should manage documents or database entries effortlessly. A maintainable system reduces future workload and errors.

g) **Compatibility**: The system should run on all modern browsers like Chrome and Edge. Frontend must adapt to different screen sizes and devices.Node.js and React-based application should support various OS environments. API communication should remain consistent across platforms. Compatibility ensures users can access the system anywhere.

CHAPTER 4

SYSTEM ANALYSIS AND DESIGN

4.1 System Analysis

The proposed Student Assistant Chatbot aims to solve the problem of students struggling to access quick and accurate academic information. Traditional methods like searching textbooks, browsing the internet, or waiting for faculty responses are slow and inefficient. The system provides an intelligent and centralized platform where students can instantly ask questions and receive accurate answers. It uses the RAG (Retrieval-Augmented Generation) approach to ensure responses come from relevant academic documents.

The MERN stack supports smooth web application development, with React for the user interface, Node.js and Express for backend APIs, and MongoDB for storing chat history, users, and documents. Uploaded PDFs are processed, converted into embeddings, and stored in a vector database for efficient retrieval. The system retrieves the most relevant content and uses an LLM (Gemini) to generate accurate, contextual answers. Students also get features like PDF upload, chat history, and a clean dashboard.

Admins are provided with a separate dashboard to manage users, documents, and overall system performance. The system is designed to be scalable, user-friendly, and secure for academic use. Overall, this chatbot improves learning efficiency by offering instant academic help, reducing manual workload, and providing a modern, intelligent solution for student support.

4.2 Feasibility study

The feasibility of the proposed Student Assistant Chatbot project is examined in this phase, and a clear proposal is presented with an overall plan and expected cost considerations. During system analysis, the feasibility study helps determine whether the new system can be developed and implemented without causing unnecessary burden. This ensures that the chatbot is practical, affordable, and suitable for the academic environment. For this analysis, it is important to understand the major requirements of the system, including technical tools, development cost, and user acceptance.

The AI Student Assistant system is feasible across all important areas. Technically, it uses strong and well-supported technologies like the MERN stack, LangChain, Gemini API, and AWS S3, making the system reliable and easy to develop. Economically, the project is highly affordable since most tools are open source, and only minimal costs are required for cloud storage or AI API usage. Socially, the system supports students by helping them access study materials quickly and improving their learning experience, making it widely acceptable in educational settings. Operationally, the platform is simple for both students and admins to use, with easy document upload, smooth navigation, and well-organized dashboards [17].

Additionally, the system can be maintained and updated easily due to its modular design. It also supports scalability, meaning more users and documents can be added without affecting performance. Data security is ensured through proper authentication and protected file storage. The system's modern interface encourages students to interact more actively with digital learning tools. Since the technologies used are common and well-documented, training new users or developers becomes simple. Overall, the project is practical, achievable, and beneficial for both academic institutions and students.

4.3 System Architecture

The system architecture of the AI Student Assistant Chatbot is designed to manage interactions between two main types of users: Students and Admins. The architecture follows a structured flow starting from user authentication and branching into separate modules based on user roles.

The system begins when a user accesses the platform and logs in using their credentials. Once the login request is made, the system checks whether the user is an Admin or a Student. If the user is a student, they are directed to the Student Dashboard, where they can upload PDFs, ask academic questions, and interact with the AI-based assistant. The system processes the uploaded document using text extraction and RAG-based AI models to generate accurate, contextual responses. The student then receives the answer directly on their dashboard.

If the user is an admin, they are redirected to the Admin Dashboard, which provides multiple management functions. Admins can upload notes, send notifications, create MCQ tests, and upload academic timetables. These operations help maintain and update the academic content that students rely on. Admin actions also support the backend knowledge base, ensuring the AI system has updated and relevant data.

Overall, the system architecture efficiently separates functionalities based on user roles while maintaining a smooth data flow. Students interact mainly with AI services and document processing modules, whereas admins handle content management and system updates. The entire framework ensures a user-friendly experience, secure data handling, and intelligent academic assistance through a well-organized dashboard structure.

The detailed design describes the internal working structure of the system, including module-level designs, data flow, component interactions, database structure, and the working of the AI pipeline. It explains how each part of the system operates and how all modules work together to deliver the final output.

Here it presents the detailed design of the Student Assistant Chatbot, focusing on module-level descriptions, interface design, data flow, and system integration. Each component of the system such as the student dashboard, admin module, backend server, PDF processing unit, and vector search engine is described in detail to ensure clarity in functionality. The design ensures that the system is scalable, maintainable, and aligned with the project's requirements. It also defines how the system transforms user input into meaningful output using the RAG pipeline.

The system architecture of the AI Student Assistant, presented in Figure 4.1, demonstrates the complete end-to-end workflow for both students and administrators. The process begins with user authentication, where the system identifies the user role and routes them accordingly. Students are directed to the Student Dashboard, where they can upload PDFs, ask academic questions, and access chat history, while admins are directed to a separate

Admin Dashboard used for uploading notes, managing timetables, posting announcements, and creating MCQ tests. These two distinct pathways ensure proper role separation, secure access, and a streamlined experience for all users.

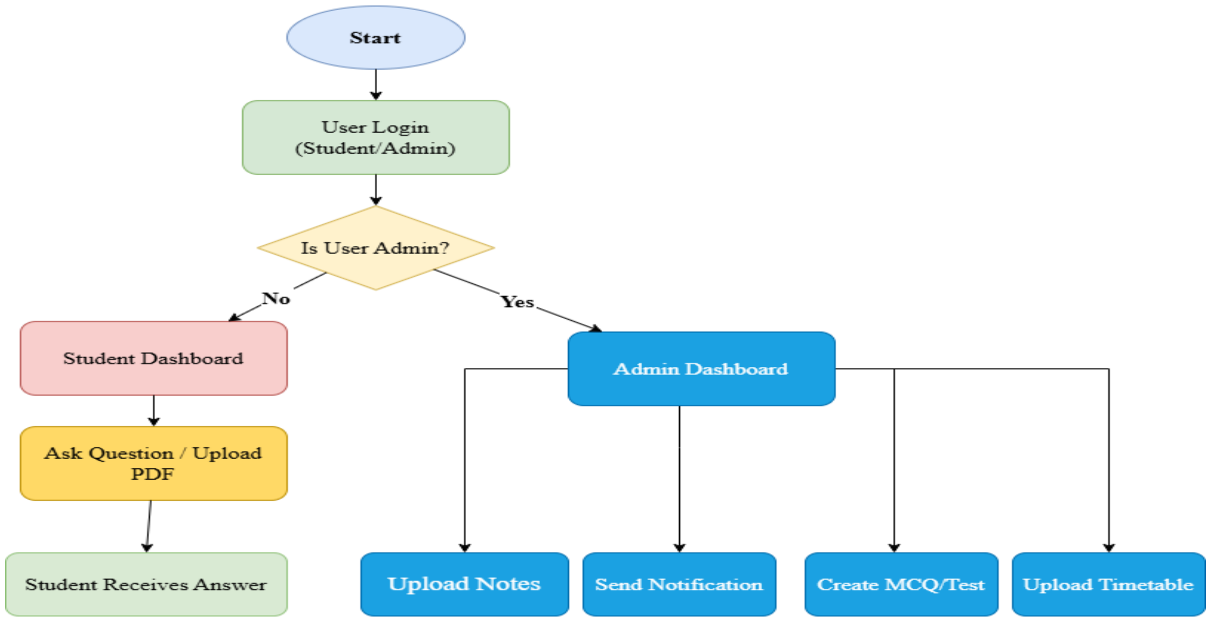


Fig 4.1 System Architecture

At the top layer, the Presentation Layer (Frontend) is built using React.js, which provides an interactive and responsive interface for both user roles. Through dynamic React components and REST API communication, users can perform tasks such as uploading documents, sending queries, navigating dashboards, and viewing AI-generated responses. This layer ensures smooth interaction and visually structured access across devices.

The frontend communicates with the Application Layer (Backend), which is developed using Node.js and Express.js. This layer manages all server-side operations, including routing APIs, handling user authentication, managing file uploads, and coordinating the flow of data between the frontend, database, and AI modules.

When a PDF is uploaded, the backend forwards it to the document-processing pipeline, and when a query is received, it triggers the retrieval and LLM generation process. This application layer acts as the central controller that links all other system components.

All structured data flows into the Database Layer (MongoDB), where user details, login credentials, document metadata, extracted text references, and chat histories are securely stored. MongoDB's flexible document schema allows efficient storage of both structured and semi-structured content, making it ideal for academic materials. It also supports fast data retrieval, ensuring smooth system performance even when the number of documents grows.

Once files are uploaded, they are processed through the Document Processing Layer, where PDF text extraction takes place using PyMuPDF. The extracted content is then cleaned, segmented, and prepared for embedding. This ensures that academic notes, question papers, and syllabus documents become part of the system's searchable knowledge base.

The cleaned text is converted into numerical vector representations in the Embedding and Vector Storage Layer, using transformer-based embedding models implemented through LangChain. These embeddings are stored in a vector database such as ChromaDB or FAISS. During a student query, the system performs similarity search across these stored vectors to retrieve the most relevant sections of the uploaded documents. This retrieval mechanism forms the core of the system's accuracy.The retrieved text then enters the RAG Pipeline (Retrieval-Augmented Generation), where LangChain orchestrates both the retrieval and generation processes. The relevant chunks are combined with the query and sent to the LLM, ensuring that all answers are grounded in real academic content rather than model assumptions. This significantly reduces hallucinations and improves reliability.

Next, the LLM Interaction Layer utilizes powerful language models such as Gemini, Groq LLaMA, or GPT to generate a coherent and context-specific response. The model processes both the question and the retrieved supporting text, producing accurate academic answers tailored to the student's needs. These responses are sent back to the backend, which finalizes the output.

Finally, the processed answer is passed to the Response Delivery Layer, where it appears on the React-based chat interface for the student. The interaction is simultaneously saved in MongoDB for future reference. This completes the full functional loop of the system, demonstrating smooth integration between user interfaces, backend logic, AI processing, retrieval mechanisms, and data management.

Overall, the system architecture ensures a well-structured, efficient, and scalable workflow that seamlessly integrates document processing, vector-based retrieval, and LLM-powered response generation. By clearly separating the roles of students and admins, and by connecting all layers from the frontend interface to the backend, database, RAG pipeline, and LLM interaction the architecture delivers a reliable, secure, and user-friendly academic assistance platform. This layered design not only improves accuracy and system performance but also guarantees smooth communication between components, resulting in an intelligent and responsive AI Student Assistant system suitable for real-world academic environments.

CHAPTER 5

METHODOLOGY

This chapter presents the detailed methodology adopted for the design and development of the AI Student Assistant, an intelligent web-based learning system capable of answering academic queries, generating quizzes, and creating personalized study plans using Retrieval-Augmented Generation (RAG). The methodology follows a structured approach comprising requirement analysis, system architecture design, backend and frontend development, AI pipeline integration, real-time communication, database design, testing, and deployment.

The project utilizes a modular MERN stack infrastructure combined with modern AI technologies such as Gemini API, LangChain, vector embeddings, and cloud-based file storage using Amazon S3. Each component has been systematically developed to ensure scalability, accuracy, security, and usability.

5.1 Methodological Workflow

The Methodological workflow illustrates in Figure 5.1 this complete methodological workflow of the AI-Powered Student Assistant Chatbot, showing how the frontend, backend, database, cloud storage, and RAG (Retrieval-Augmented Generation) engine interact to deliver features such as chatting, quiz generation, study planning, and document-based question answering.

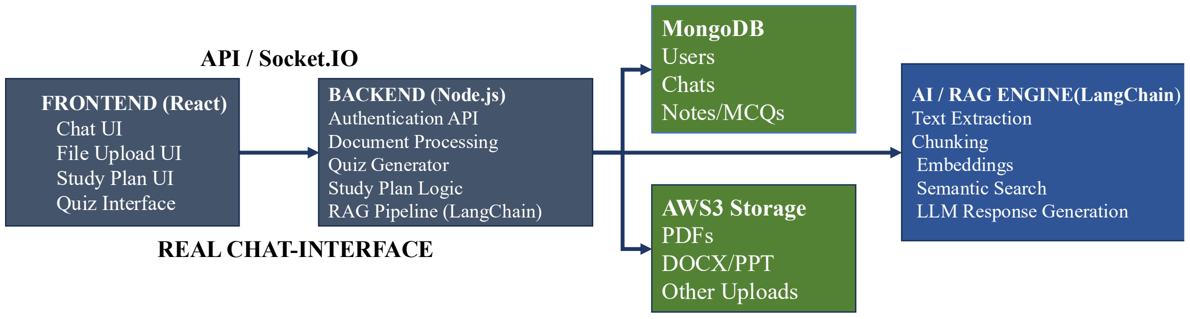


Figure 5.1 Methodological Workflow

The proposed system architecture integrates multiple components that work together to deliver an intelligent, real-time student assistance chatbot. The frontend, developed using React, provides all user-facing interfaces such as the chat window, file upload screen, study-plan interface, and quiz module. These components communicate with the backend through APIs or Socket.IO for real-time message exchange. The backend, built using Node.js, handles core logical operations including user authentication, document processing, quiz generation, study-plan creation, and the implementation of the RAG pipeline using Lang Chain. Uploaded files such as PDFs, DOCX, and PPTs are securely stored in AWS S3 Storage, where they can be retrieved for further processing. The system also connects to MongoDB, which manages all persistent data including user accounts, chat history, stored notes, and MCQs created by the system.

The backend further interacts with the AI/RAG Engine, powered by LangChain, which performs advanced tasks such as text extraction, document chunking, embedding generation, semantic similarity search, and final LLM-based response generation. Together, these components form a seamless pipeline that allows students to upload notes, ask academic questions, generate quizzes, and receive personalized study assistance through an intelligent, interactive conversational interface.

The Retrieval-Augmented Generation (RAG) combined with LangChain forms the core intelligence layer of the AI Student Assistant system. As illustrated in Figure 5.2, this architecture ensures that all responses generated by the chatbot are grounded in actual academic content such as uploaded PDFs, notes, syllabus, and institutional documents. The entire pipeline operates in two major stages are Indexing Phase and Query Processing Phase both of which are orchestrated using LangChain components like document loaders, text splitters, embedding models, vector stores, retrievers, and LLM-based chains.

In the Indexing Phase, academic documents uploaded by students or administrators are ingested and converted into a structured, searchable knowledge base. The uploaded files (PDF, DOCX, PPT, TXT) are first stored in cloud storage such as AWS S3, after which Langn Chain-supported loaders extract raw text from these documents. This text undergoes preprocessing to remove unwanted symbols, normalize spacing, and handle elements like headings and bullet points, creating clean and consistent textual data. LangChain RecursiveCharacterTextSplitter then divides the text into smaller, meaningful chunks typically 500–1000 characters with slight overlap to preserve context. Each chunk is transformed into a high-dimensional vector using embedding models such as Gemini embeddings. These embeddings, along with metadata like document source and page number, are stored in a vector database enabling fast semantic search and efficient retrieval.

The Query Processing Phase, also shown in Figure 5.2 begins when a user submits a question through the chatbot interface. The backend converts this question into an embedding using the same embedding model used during indexing. This ensures semantic compatibility between the query and document vectors. Lang Chain as Retriever module performs a similarity search to locate the top-k chunks that are most relevant to the user's question. These retrieved chunks form the contextual evidence used to generate an accurate response.

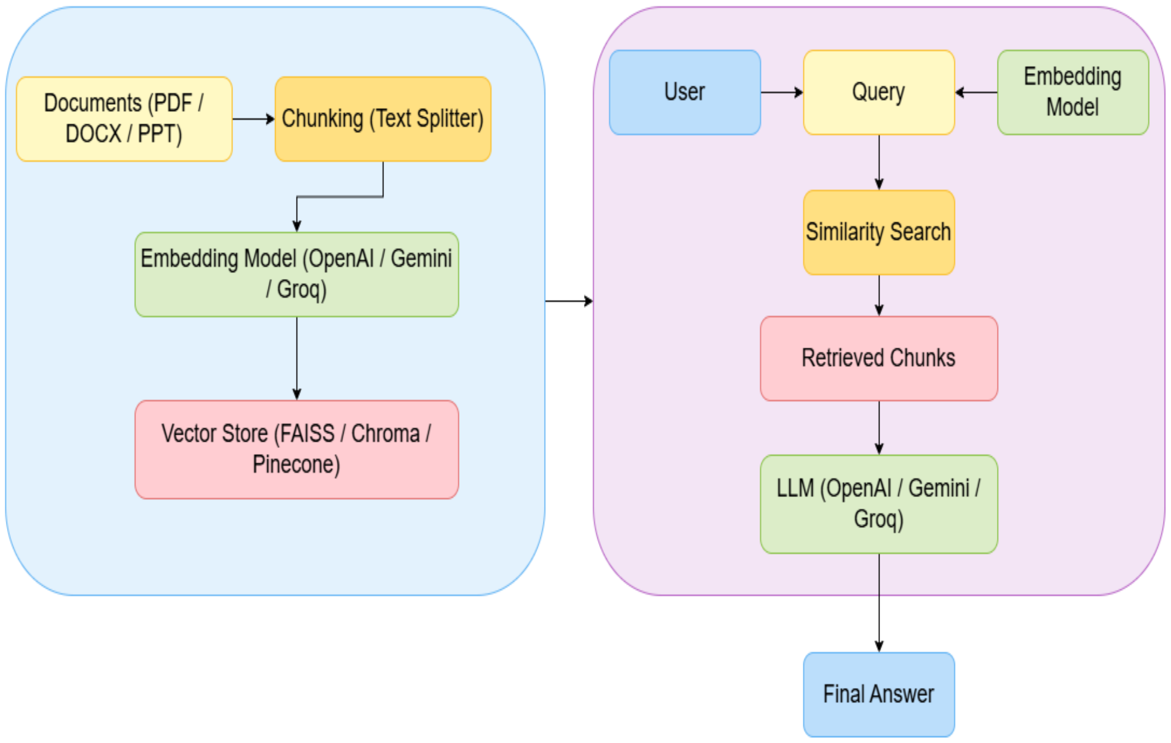

Documents (PDF / DOCX / PPT)
Chunking (Text Splitter)
Embedding Model (OpenAI / Gemini / Groq)
Vector Store (FAISS / Chroma / Pinecone)
User
Query
Embedding Model
Similarity Search
Retrieved Chunks
LLM (OpenAI / Gemini / Groq)
Final Answer

Figure 5.2 RAG and LangChain Architecture

Once the LLM produces the response, additional processing such as formatting, summarization, study-plan generation, or MCQ creation may be applied. The final answer is delivered back to the user interface, and the system stores the query, retrieved chunks, generated response, and timestamps in MongoDB for chat history and personalization. Overall, the pipeline illustrated in Figure 5.2 ensures that the AI Student Assistant produces highly accurate, document-grounded, and reliable academic responses.

The common pipeline used in Lang Chain for a RAG-based chatbot as follows

a. **User Query Input:** User query input process begins when a user inputs a query or a question into the system. This query serves as the starting point that initiates the entire retrieval-augmented generation (RAG) workflow.
b. **Query Embedding:** Next, Query Embedding is raw text query is passed to an embedding model, often based on a pretrained transformer architecture. The embedding model transforms the query into a dense vector representation within a semantic space. This vector encapsulates the meaning of the query, making it suitable for similarity search against stored knowledge.
c. **Retriever: Vector Search:** The query embedding is then sent to the Retriever component. The Retriever performs a similarity search within a vector database, which stores precomputed vector embeddings of documents, passages, or knowledge chunks. Using cosine similarity or another distance metric, the Retriever identifies and returns the top-k documents that are semantically closest to the user's query.
d. **Documents Returned:** Retriever system collects these top relevant documents or text chunks retrieved by the Retriever. These documents contain information that is most related to the user's query and serve as the contextual knowledge for the subsequent generation step.
e. **Prompt Construction:** After retrieving the documents, the system constructs a prompt by combining the user's original query with the retrieved documents. This usually involves concatenating the retrieved texts with additional instructions or templates to guide the language model response.

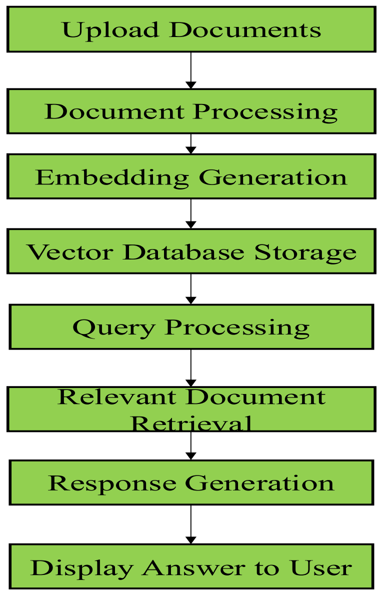


Figure 5.3 RAG Pipeline

f. **Language Model (LLM) Generation**: Constructed prompt is then provided to the Language Model. The LLM generates a response by leveraging both the retrieved documents and the original query. The presence of external knowledge from the retrieved documents allows the LLM to produce more accurate, relevant, and factually grounded answers.

g. **Generated Answer Returned:** Finally, the generated answer from the LLM is presented to the user as the system's response. This response is designed to be relevant, context-aware, and informative, providing the user with a high-quality answer based on both the LLM's capabilities and the retrieved knowledge.

5.2 System Module

In this project, two primary user interfaces are provided to support different roles within the system: the Admin Dashboard and the Student Dashboard. Each dashboard is designed with specific functionalities based on the responsibilities and requirements of its users. The separation of dashboards ensures better security, organized workflow, and a clear distinction between management tasks and learning activities.

The system follows a role-based access structure

i. **Student Module:** Students interact with learning resources and AI-based assistance.
ii. **Admin Module:** Admins handle the management and control operations.

The Student Module, as illustrated in Figure 5.4, begins with a secure User Authentication process where students can log in or register for a new account to gain personalized access to the system. After successful authentication, users are directed to the Student Dashboard, where they can choose from various available services, making this the primary decision point of the module. One of the key functionalities is the Document Upload Module, which enables students to upload study materials such as PDF or DOCX files, these documents are then securely stored in the system database or cloud storage for later use.

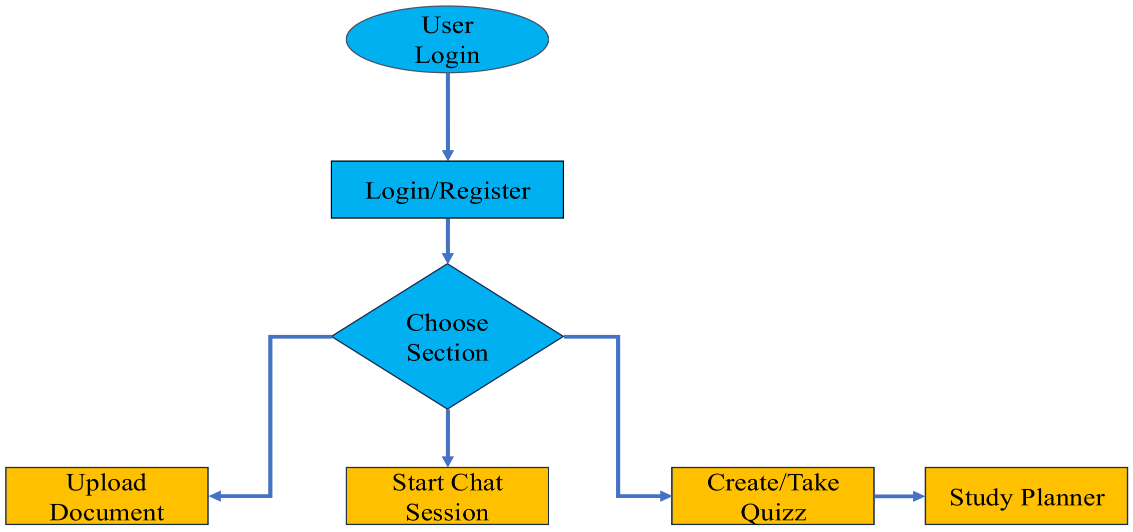


Figure 5.4 User Interface

The system also provides an intelligent Chat Module, which features an AI-based conversational interface allowing students to ask questions related to uploaded content or general academic topics. All chat interactions are securely stored to support future retrieval and personalized responses.

Furthermore, the Quiz Module lets students generate quizzes from their uploaded notes or attempt existing quizzes, all of which are saved within the Quiz Collection for later review or retakes. Lastly, the Study Planner Module offers personalized study schedules based on selected subjects, deadlines, and available time

Finally, selecting Study Planner enables the student to create a personalized study schedule by entering subjects, hours, and deadlines, after which the system generates an optimized study plan. This flow ensures a smooth, interactive, and well-organized learning experience, guiding the user step-by-step through all major features of the MERN-based AI Student Assistant system. Ensuringa that the generated study plan is saved and accessible whenever required. Overall, the student module provides a comprehensive and interactive learning environment, enhancing student engagement and academic support.

The Admin Module, as illustrated in Figure 5.5 represents the management operations performed by the system administrator through the Admin Dashboard.

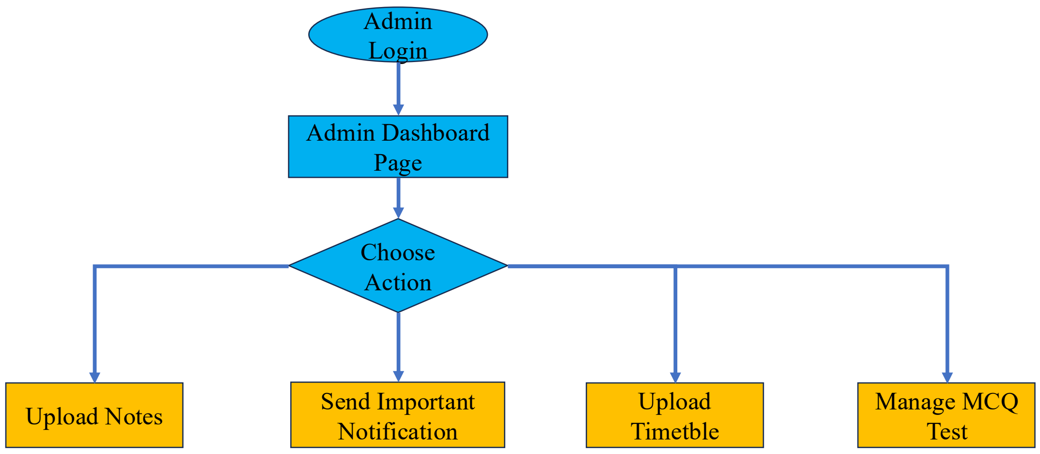


Figure 5.5 Admin Module

Begins with a secure Admin Authentication process, ensuring that only authorized personnel can access the administrative features of the system. Once successfully logged in, the administrator is directed to the Admin Dashboard, where multiple management functionalities are available, making it the central decision point for all administrative operations. One of the key features is the Upload Notes option, which allows the admin to upload official study materials or academic resources that become accessible to all students within the system.

Additionally, the admin can use the Send Important Notifications feature to broadcast essential announcements such as examination schedules, deadlines, event updates, or system-related alerts to every registered user. Through the Upload Timetable module, administrators can upload class schedules, exam timetables, or institutional calendars, ensuring that students always have access to updated academic information.

Furthermore, the Manage MCQ Test function enables the admin to create and delete multiple-choice quizzes, ensuring that assessment content remains accurate, relevant, and up to date. Collectively, these functionalities empower administrators to efficiently manage academic content, communication, and assessments within the system.

5.3 Database Schema

This diagram represents the main use cases of the Student Assistant Chatbot for both Students and Admins. Students interact with the system to perform academic tasks such as uploading documents and asking AI-powered questions. They can also generate quizzes based on their notes, create personalized study plans, and view timetables uploaded by the admin. These features help students learn efficiently and access academic resources quickly.

Admins manage the academic content and system data. They can upload notes and study materials for student access. Admins also manage semesters, ensuring subjects and resources are organized correctly. They upload class timetables and exam schedules for students to view. Additionally, admins can create MCQ tests and post important announcements. This keeps the system updated and ensures smooth academic communication between the institution and students.

The database for the Student Assistant Chatbot, as illustrated in Figure 5.6, is implemented using MongoDB, a NoSQL document-oriented database designed for flexible, scalable, and high-performance data storage. MongoDB was selected due to its ability to efficiently manage unstructured and semi-structured data, making it ideal for storing diverse information such as user profiles, chat history, uploaded notes, extracted text, quiz data, and study materials.

The backend communicates with the database through MongoDB, which enables schema-based modelling, validation, and structured interactions with the collections. This integration ensures reliable data handling, smooth CRUD operations, and robust management of all user- and admin-related records within the system

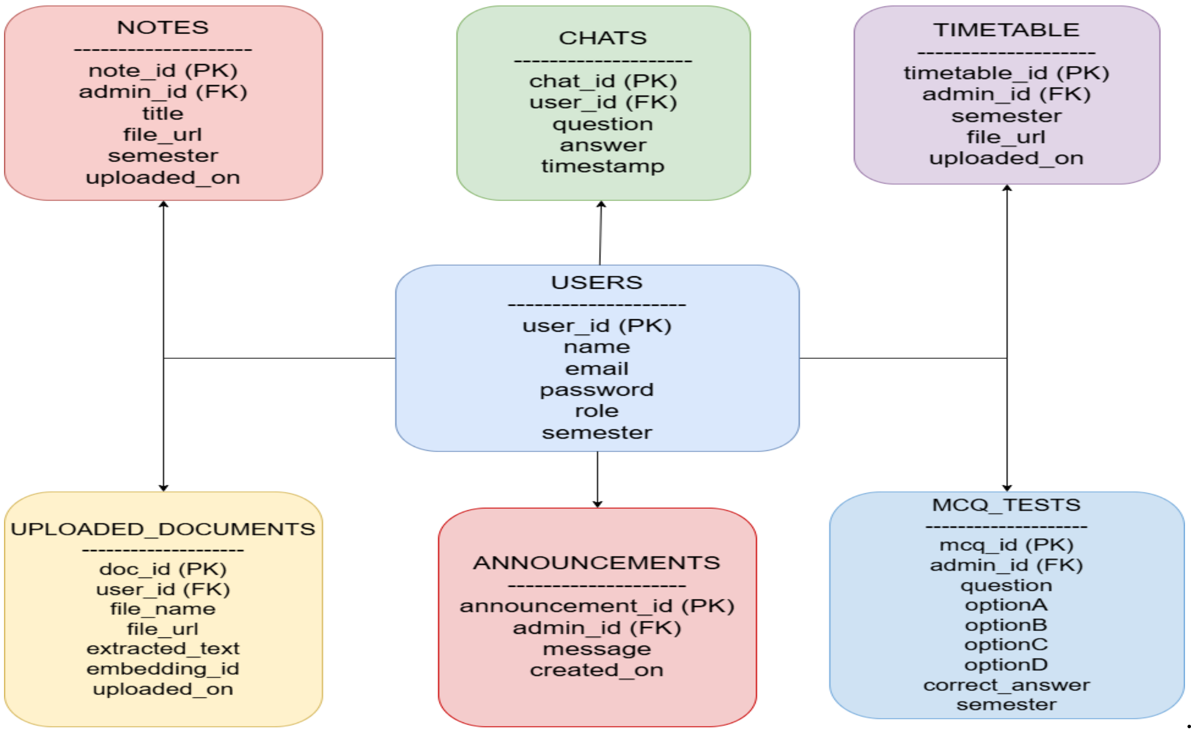


Figure 5.6 Database Schema

The use case shown in Figure 5.7 presents the primary interactions of both Students and Admins within the system. For students, the system provides key learning-oriented features such as uploading academic documents, asking questions through the AI chat module, generating quizzes from their notes, creating personalized study plans, and viewing published timetables. These functionalities aim to enhance learning efficiency and provide instant academic assistance. On the other hand, the admin module focuses on system and content management.

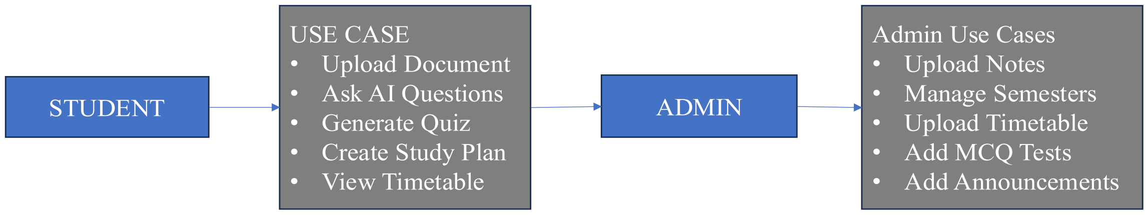


Figure 5.7 Use Case

Admins can upload official notes, manage semester details, update academic timetables, add or modify MCQ-based tests, and publish important announcements for all users. This Figure clearly outlines how each user role interacts with specific system features, ensuring a distinct separation between student activities and administrative responsibilities.

CHAPTER 6

TESTING

Testing is an essential phase of the project that ensures the Student Assistant Chatbot functions correctly, handles user input properly, and produces accurate AI responses. The testing process verifies each module of the MERN + RAG system, including the frontend, backend APIs, database operations, PDF processing, and the AI response pipeline. This chapter explains the different testing methods used during development.

6.1 Unit Testing

Unit testing was carried out on all major components and functions of the system to verify their correctness and stability. On the frontend, each React component including the Chat Interface, PDF Upload module, Quiz Generator, and Study Plan section was tested individually to ensure proper input handling, correct output rendering, and smooth UI behavior. These tests helped validate component-level logic and prevented issues related to state management or user interaction.

On the backend, unit tests were applied to Node.js API endpoints such as user login, document upload, chat response generation, quiz creation, and study plan retrieval. MongoDB-related operations, including user creation, chat storage, note retrieval, and document metadata saving, were also tested to confirm reliable database communication. Additionally, the RAG pipeline components text chunking, embedding generation, and semantic vector search were tested independently to ensure accurate retrieval and correct processing of academic content. Through unit testing, issues like mismatched API responses, faulty logic, and UI inconsistencies were detected and resolved early, resulting in a more robust and error-free system.

6.2 System Testing

System testing was conducted to verify that the entire chatbot system works seamlessly after integrating all modules. Frontend and backend communication was tested using actual student workflows, such as logging in, uploading PDFs, generating quizzes, and interacting with the chatbot. The backend was checked for correct processing of requests, proper routing, secure authentication, and error-free execution of APIs. The system was also tested for real-time behavior such as chat flow, response loading, and study plan generate.

Database testing ensured that all information users, chats, notes, MCQs, announcements is stored and retrieved correctly. System testing confirmed that the integrated MERN + RAG environment works smoothly without crashes or mismatches.

Performance checks were done by testing system response under multiple requests (simulated using Postman). System testing confirmed proper communication between React, Node.js, MongoDB, AWS S3, and the RAG engine without breaking any module.

6.3 Validation Testing

Validation testing ensured that the system meets user requirements and behaves exactly as intended. Students validated features like document upload, quiz generation, timetable viewing, and AI question-answering by performing real tasks. Admins validated the accuracy of backend functions such as uploading notes, adding MCQ tests, managing semesters, and creating announcements.

The AI-generated answers were checked for correctness, clarity, and relevance to uploaded academic content.PDF-based question answering was validated by comparing chatbot responses with the content extracted from the PDFs. User feedback was collected to ensure the interface was easy to use and all features were accessible. Validation confirmed that the chatbot meets functional, performance, and usability expectations outlined in project requirements. This ensured the final system is reliable, user-friendly, a best real academic use.

6.4 Testing Summary

The testing phase ensured that every module of the Student Assistant Chatbot functioned correctly, reliably, and according to the project requirements. Unit testing verified the accuracy of individual components such as login, file upload, API responses, chat processing, and database operations. Each function performed as expected, and any minor issues were fixed during development.

System testing confirmed that the integrated MERN + RAG architecture worked smoothly as a whole. Complete workflows such as uploading a PDF, generating embeddings, asking questions, creating quizzes, and viewing timetables were tested end-to-end. The system handled multiple operations consistently without failures, proving its stability and correctness

The detailed module-wise testing outcomes for the Student Assistant Chatbot are presented in Table 6.1, which includes inputs, expected outputs, actual outputs, and the final status.

Table 6.1 Module-wise Testing Results

TESTING ID	Module	Input	Expected Output	Actual Output	Status
01	Login Module	Valid email & password	User logged in	Works as expected	Pass
02	File Upload	PDF file upload	File stored & processed	Extracted successfully	Pass
03	Ask Question API	User query	AI response generated	Response received	Pass
04	MongoDB Insert	New chat message	Chat stored in DB	Stored correctly	Pass
05	RAG Embedding	Text input	Embedding generated	Embedding created	Pass
06	Quiz Generator	Notes/PDF text	5-MCQs generated	Questions created	Pass
07	Study Plan	Subject & time	Study plan generated	Works properly	Pass

Validation testing ensured that the system met all user requirements and delivered accurate academic assistance. Students and Admins tested real tasks, and the AI responses were compared with actual study material to confirm correctness. The system was easy to use, produced reliable outputs, and satisfied all functional and usability criteria.

Overall, the testing phase demonstrated that the chatbot is stable, accurate, user-friendly, and ready for deployment, meeting all expected academic and technical requirements.

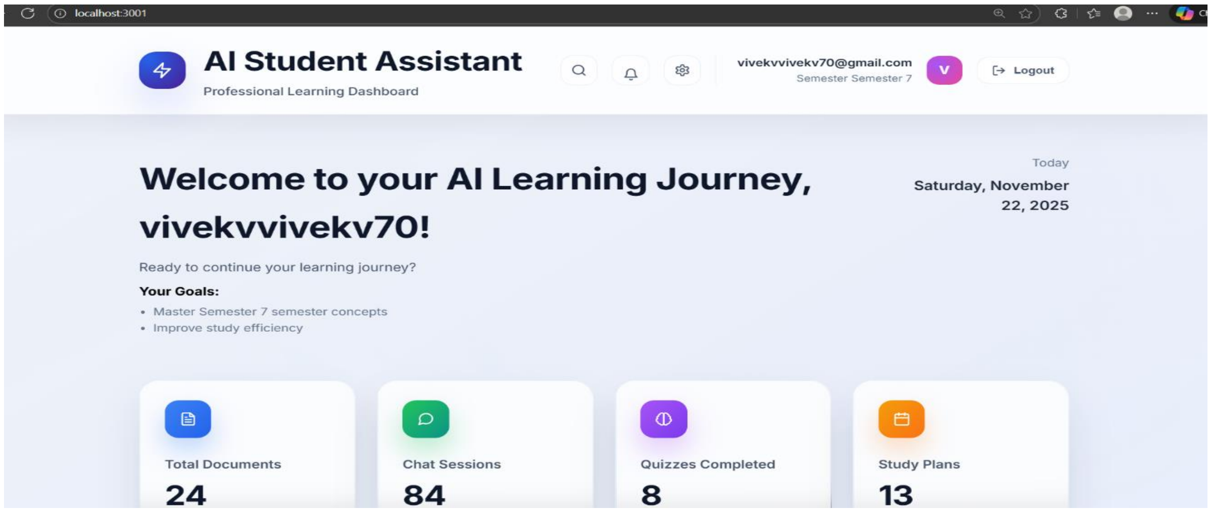
CHAPTER 7

EXPERIMENTATION RESULTS

The results obtained from the Student Assistant Chatbot demonstrate that the system performs effectively in providing academic support to students. The chatbot successfully answered user questions based on uploaded PDF content and internal notes, proving that the RAG (Retrieval-Augmented Generation) approach works accurately and reliably. The integration between React frontend, Node.js backend, MongoDB database, and the AI engine functioned smoothly, ensuring fast responses and a seamless user experience.

The results also show that students were able to upload documents, generate quizzes, create study plans, and view timetables without any difficulty. The AI-produced answers were consistent with syllabus-based content and matched the expectations set during validation testing. Admin results showed that features such as uploading notes, managing semesters, adding MCQs, and posting announcements worked as intended, making the academic management system more efficient.

Figure 7.1 presents the main Home Page or Dashboard of the AI Student Assistant. This interface acts as the entry point for students, offering essential options such as document upload, AI chat mode, quiz generation, and study plan creation. The sidebar contains navigation shortcuts, and the AI features become active only after the student uploads relevant documents, ensuring accurate and context-aware responses.



localhost:3001

AI Student Assistant
Professional Learning Dashboard

vivekvvivekv70@gmail.com
Semester Semester 7

Logout

Welcome to your AI Learning Journey, vivekvvivekv70!

Today
Saturday, November 22, 2025

Ready to continue your learning journey?

Your Goals:
- Master Semester 7 semester concepts
- Improve study efficiency

Total Documents 24
Chat Sessions 84
Quizzes Completed 8
Study Plans 13

Figure 7.1: Home Page / Dashboard of AI Student Assistant

Figure 7.2 represents the main Student Dashboard of the AI Student Assistant system. This Dashboard serves as the central hub for all student activities, offering an organized and user-friendly layout that allows students to access academic tools quickly. The Quick Actions panel on the left provides shortcuts to key features such as the AI Chat Assistant, Document Manager, AI Quiz Generator, Study Planner, Upload Notes, and Study Files. Each feature is visually highlighted with icons and brief descriptions, helping users easily understand its purpose.

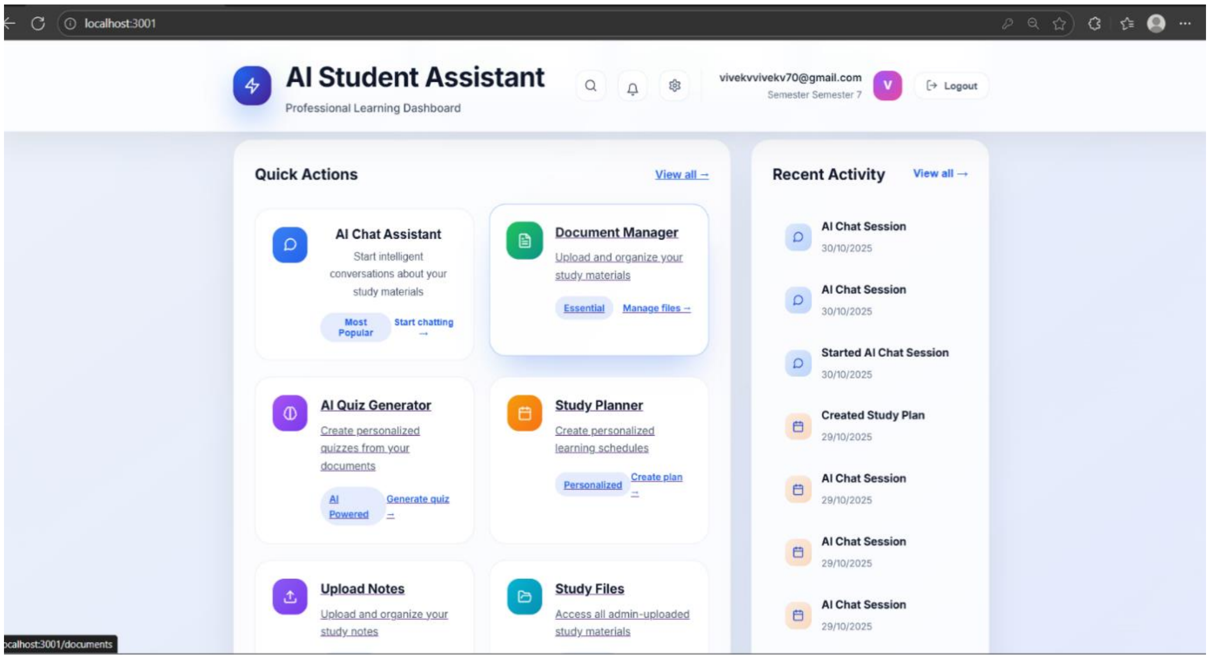
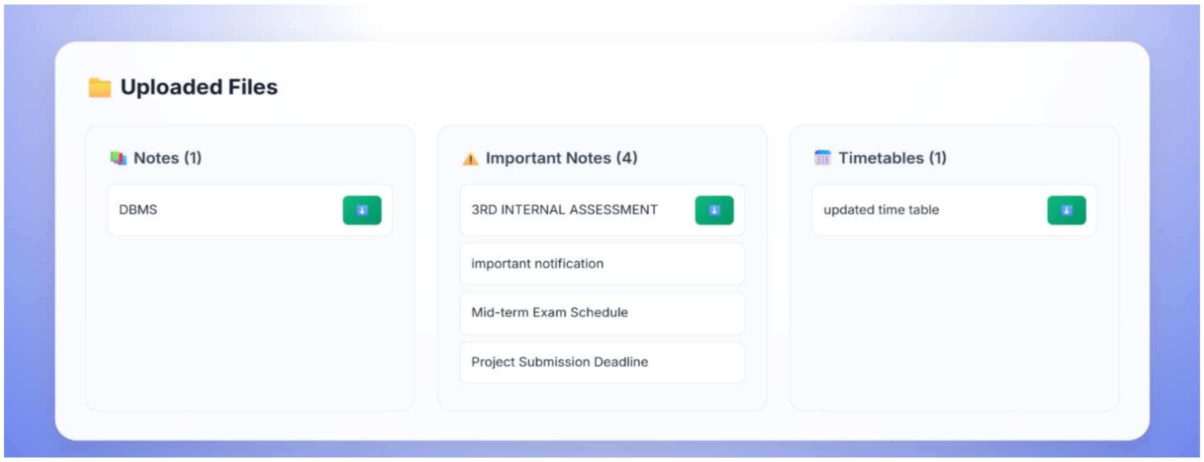


Figure 7.2: Student Dashboard Interface

On the right side, the Recent Activity section displays the student’s latest interactions, including AI chat sessions, created study plans, and other system activities. This enables users to track their progress and revisit previous actions. At the top, the navigation bar shows the student’s email, semester details, notification icons, and logout option, ensuring secure and personalized access. Overall, the dashboard is designed for simplicity, efficiency, and smooth navigation, enabling students to manage their learning materials and AI-powered tools with ease.

Figure 7.3 shows the Document Upload Section. This panel enables students to upload academic files, including PDF, DOCX, PPT, and text formats. These documents are processed by the RAG (Retrieval-Augmented Generation) engine to generate accurate academic responses, provide references during chat queries, and create meaningful quizzes based on the content.

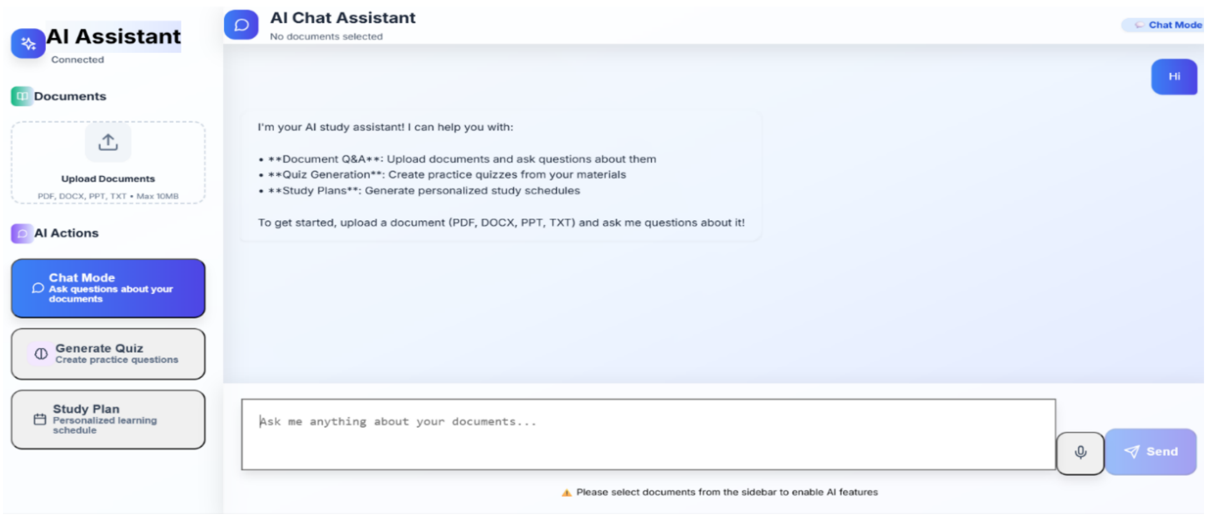


Uploaded Files

Notes (1)	Important Notes (4)	Timetables (1)
DBMS	3RD INTERNAL ASSESSMENT	updated time table
	important notification	
	Mid-term Exam Schedule	
	Project Submission Deadline	

Figure 7.3: Document Upload Section

Figure 7.4 represents the Chat Mode Interface, where students can ask questions related to their uploaded documents. The AI utilizes the extracted document content to deliver precise, syllabus-aligned, and context-based answers. This feature helps students clarify doubts quickly and interactively.


AI Assistant
Connected
Documents
Upload Documents
PDF, DOCX, PPT, TXT • Max 10MB
AI Actions
Chat Mode
Ask questions about your documents
Generate Quiz
Create practice questions
Study Plan
Personalized learning schedule
AI Chat Assistant
No documents selected
Chat Mode
Hi
Ask me anything about your documents...
Send
Please select documents from the sidebar to enable AI features

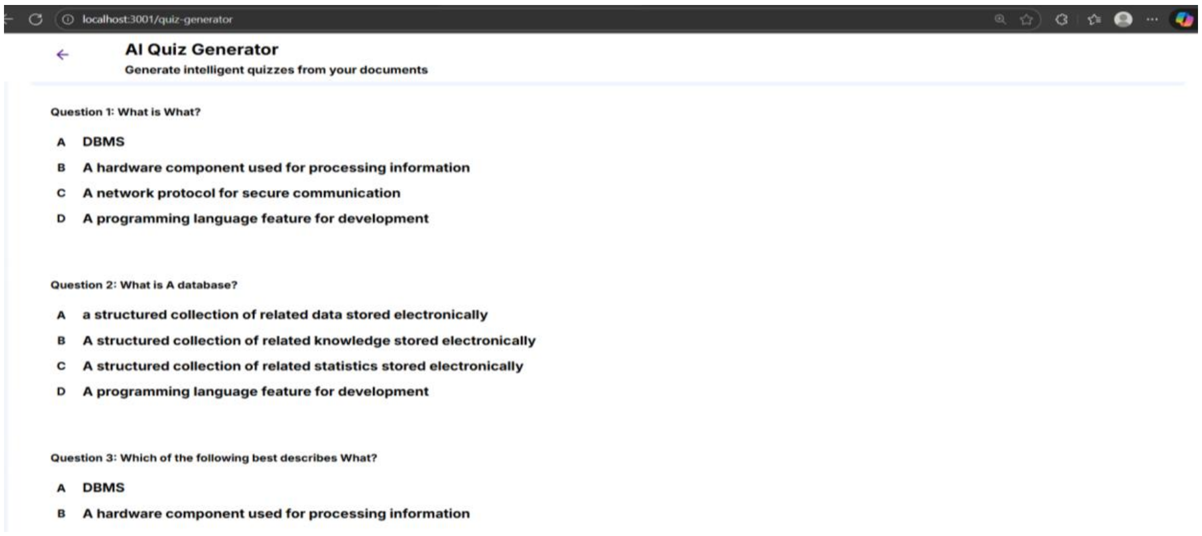
I'm your AI study assistant! I can help you with:

- **Document Q&A**: Upload documents and ask questions about them
- **Quiz Generation**: Create practice quizzes from your materials
- **Study Plans**: Generate personalized study schedules

To get started, upload a document (PDF, DOCX, PPT, TXT) and ask me questions about it!

Figure 7.4: Chat Mode Interface

Figure 7.5 demonstrates the Quiz Generator Module. This feature automatically generates multiple-choice questions from the content of uploaded notes or documents. It assists students in practicing key concepts, evaluating their understanding, and preparing effectively for exams.


localhost:3001/quiz-generator

AI Quiz Generator
Generate intelligent quizzes from your documents

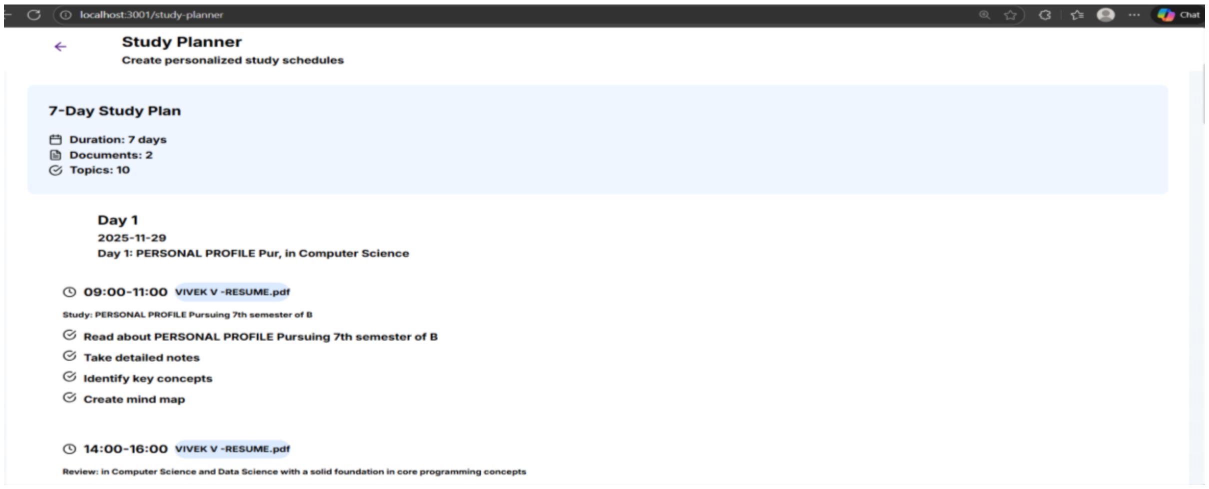
Question 1: What is What?
A DBMS
B A hardware component used for processing information
C A network protocol for secure communication
D A programming language feature for development

Question 2: What is A database?
A a structured collection of related data stored electronically
B A structured collection of related knowledge stored electronically
C A structured collection of related statistics stored electronically
D A programming language feature for development

Question 3: Which of the following best describes What?
A DBMS
B A hardware component used for processing information

Figure 7.5: Generate Quiz Module

Figure 7.6 displays the Study Plan Generator. Based on factors such as subject difficulty, available study time, and user preferences, the system creates a personalized study plan for each student. This functionality helps students manage their time efficiently and maintain a structured learning schedule.


localhost:3001/study-planner

Study Planner
Create personalized study schedules

7-Day Study Plan
Duration: 7 days
Documents: 2
Topics: 10

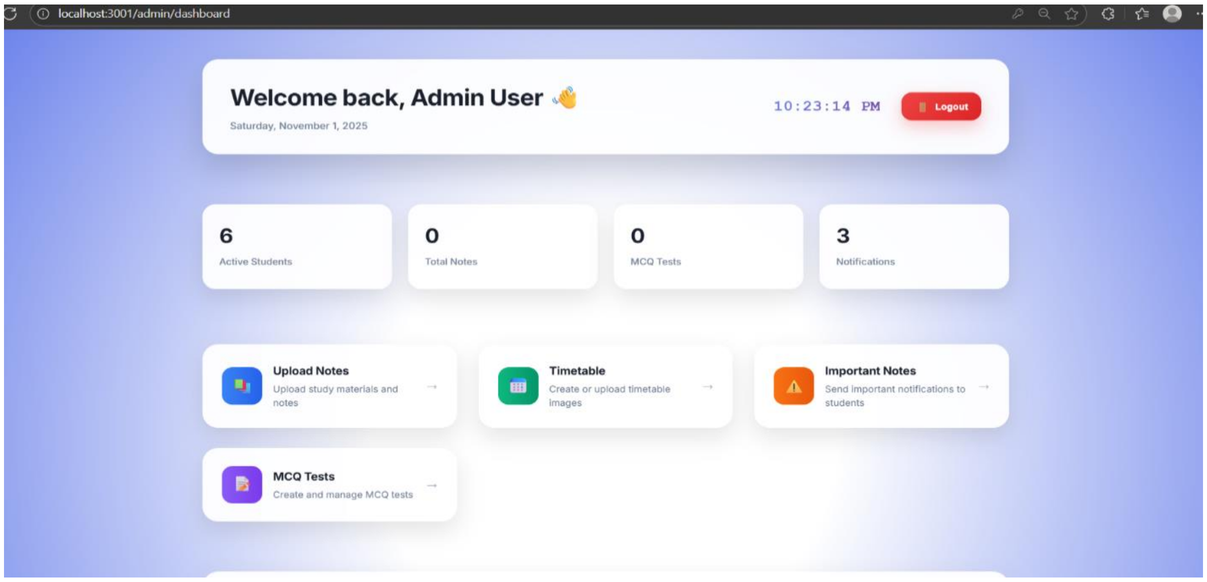
Day 1
2025-11-29
Day 1: PERSONAL PROFILE Pur, in Computer Science

09:00-11:00 VIVEK V -RESUME.pdf
Study: PERSONAL PROFILE Pursuing 7th semester of B
Read about PERSONAL PROFILE Pursuing 7th semester of B
Take detailed notes
Identify key concepts
Create mind map

14:00-16:00 VIVEK V -RESUME.pdf
Review: in Computer Science and Data Science with a solid foundation in core programming concepts

Figure 7.6: Study Plan Generator

Figure 7.7 illustrates the Admin Dashboard. Administrators can upload study notes, add timetables, manage semester configurations, and create MCQ tests for students. This dashboard centralizes academic management, making it easier for faculty or administrators to maintain updated academic resources.


Welcome back, Admin User 👋
Saturday, November 1, 2025
10:23:14 PM

6	0	0	3
Active Students	Total Notes	MCQ Tests	Notifications

Upload Notes Upload study materials and notes

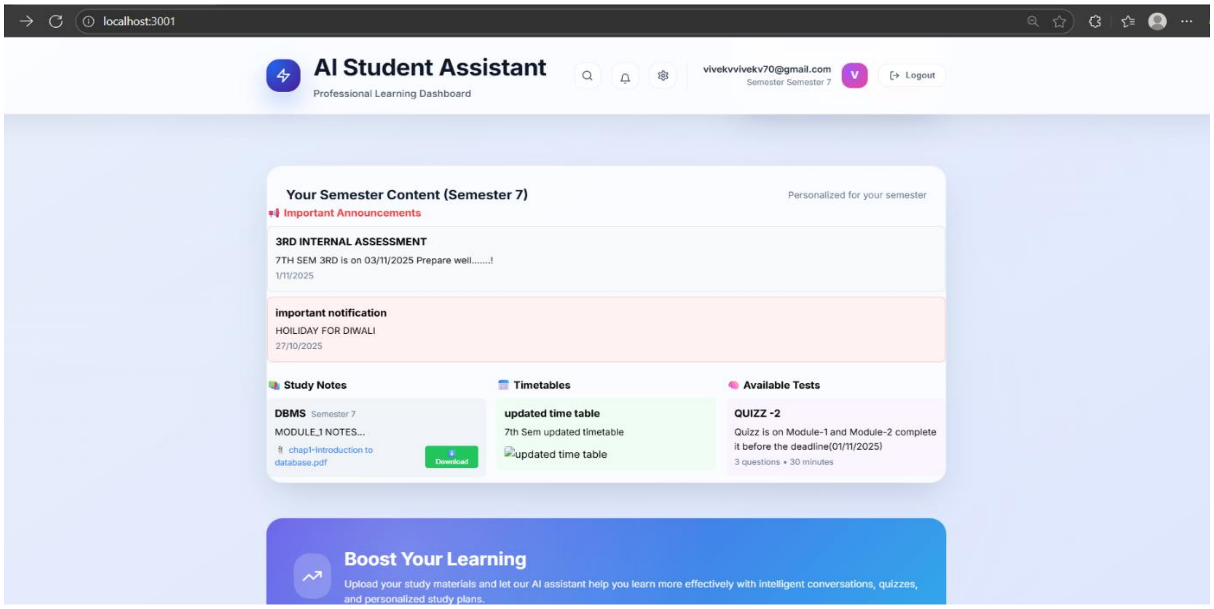
Timetable Create or upload timetable images

Important Notes Send important notifications to students

MCQ Tests Create and manage MCQ tests

Figure 7.7: Admin Dashboard

Figure 7.8 presents the Announcements Module. This panel allows administrators to broadcast important academic updates such as exam dates, deadlines, events, and institutional notices. The announcements appear on the student dashboard to ensure timely visibility.


AI Student Assistant
Professional Learning Dashboard
vivekvvivekv70@gmail.com
Semester Semester 7

Your Semester Content (Semester 7) Personalized for your semester

Important Announcements

3RD INTERNAL ASSESSMENT
7TH SEM 3RD is on 03/11/2025 Prepare well.......!
1/11/2025

important notification
HOILIDAY FOR DIWALI
27/10/2025

Study Notes
DBMS Semester 7
MODULE_1 NOTES...
chap1-Introduction to database.pdf

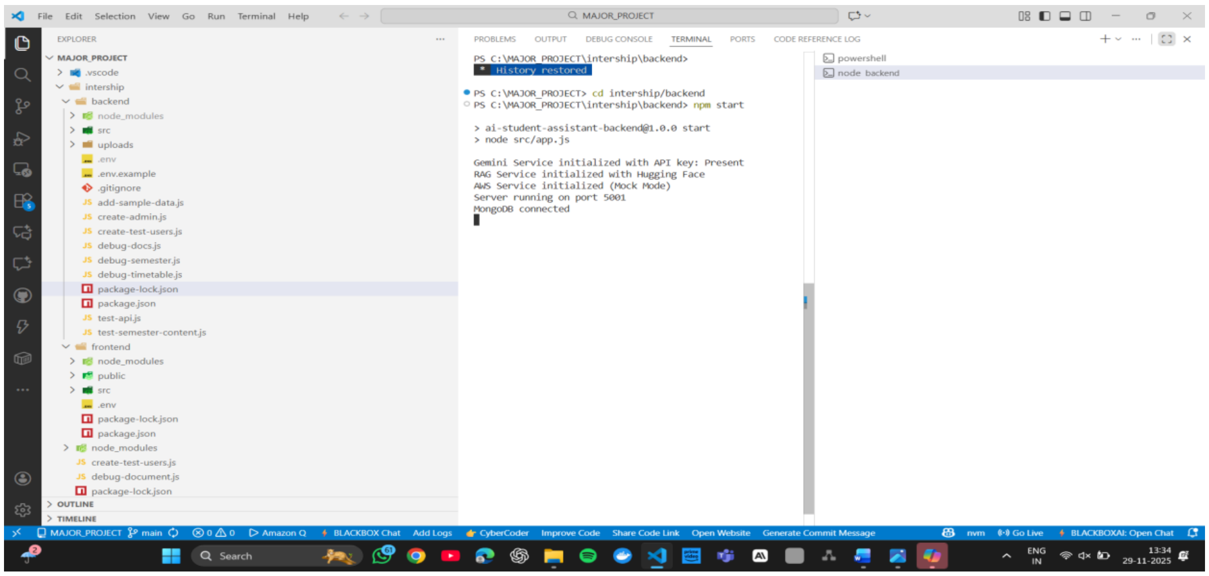
Timetables
updated time table
7th Sem updated timetable
updated time table

Available Tests
QUIZZ -2
Quizz is on Module-1 and Module-2 complete it before the deadline(01/11/2025)
3 questions • 30 minutes

Boost Your Learning
Upload your study materials and let our AI assistant help you learn more effectively with intelligent conversations, quizzes, and personalized study plans.

Figure 7.8: Announcements Module

Figure 7.9 shows the Backend Server Execution in VS Code. This confirms that the backend API, AI modules, and database connections are running successfully. It ensures that all student-facing features such as chat responses, quiz creation, study plan generation, and document processing can interact seamlessly with the backend services.



```
PS C:\MAJOR_PROJECT\intership\backend>
 *  History restored

PS C:\MAJOR_PROJECT> cd intership/backend
PS C:\MAJOR_PROJECT\intership\backend> npm start

> ai-student-assistant-backend@1.0.0 start
> node src/app.js

Gemini Service initialized with API key: Present
RAG Service initialized with Hugging Face
AWS Service initialized (Mock Mode)
Server running on port 5001
MongoDB connected
```

Figure 7.9: Backend Server Execution in VS Code

The results of the AI Student Assistant system demonstrate that the chatbot effectively supports students by providing accurate academic assistance and seamless interaction across all modules. Testing across various components confirms that the system successfully integrates the React frontend, Node.js backend, MongoDB database, and Lang Chain-powered RAG engine to deliver reliable and context-aware responses. Students were able to upload study materials, interact with the AI chat module, generate quizzes, create personalized study plans, and view academic timetables without encountering any functional issues. The system consistently produced relevant, syllabus-aligned answers, proving the effectiveness of the embedding model and retrieval mechanism used in the RAG pipeline.

The results from the admin module further validate the system's stability and usability. Admins could efficiently upload notes, manage semesters, create MCQ tests, post announcements, and update timetables, all of which were reflected instantly on the student dashboard. The backend server executed smoothly, with all APIs responding correctly for chat processing, document extraction, quiz generation, database operations, and study planning. This screenshots Figures from 7.1 to 7.9 illustrates the successful implementation of each module, highlighting well-designed dashboards and intuitive navigation for both students and administrators.

Performance testing confirmed that the system maintains fast response times and stable performance even under multiple requests. The AI-generated responses were accurate, contextually appropriate, and matched the expected results during validation. The system also demonstrated strong error handling capabilities and maintained operational stability throughout testing. Overall, the results confirm that the AI Student Assistant meets all the project objectives by providing a practical, user-friendly, and intelligent academic assistance platform. The integration of MERN, Lang Chain, and RAG technologies results in a robust solution that enhances learning efficiency for students while streamlining academic management tasks for administrators.

Table 7.1 Comparative Analysis of Existing Systems

Aspect	Design and Implementation of LangChain-Based Chatbot	Development of Interactive Assistance for Academic Preparation	Proposed Model
Core Approach	RAG with document and video retrieval	RAG with locally deployed LLM	RAG with Academic Document
Deployment	Cloud-based LLM	Local LLM deployment	Flexible
Domain Focus	General-purpose	Academic preparation	College academic assistance
Data Sources	PDFs and videos	Educational documents	Syllabus, notices, academic PDFs
Privacy	Limited	Improved	Moderate with controlled data

CHAPTER 8

CONCLUSION & FUTURE SCOPE

The AI Student Assistant project successfully demonstrates how modern technologies such as the MERN stack, LangChain, and Retrieval-Augmented Generation (RAG) can be combined to create an effective academic support system. The application provides students with intelligent features including AI-powered question answering, document-based learning, quiz generation, and personalized study planning. These capabilities help students learn more efficiently and reduce their reliance on traditional resources. The developed system also supports administrators by enabling them to upload notes, manage semesters, create MCQ tests, and publish announcements in an organized manner. Through extensive testing, the system proved to be stable, accurate, and user-friendly, offering a smooth experience for both students and admins. The integration of MongoDB for data management, Node.js for backend services, react for the user interface, and LLM-powered AI for academic support ensures that the system is scalable and future-ready.Overall, the project meets its objectives of enhancing student learning through intelligent automation and providing a centralized digital academic platform. It serves as a practical solution for educational institutions seeking to adopt AI-driven learning enhancements and opens the door for further improvements and future expansion.

The AI Student Assistant system can be improved further by adding more advanced features and expanding its capabilities. Although the current version meets the core requirements, there are several potential enhancements that can make the application even more powerful and user-friendly.The AI Student Assistant can be enhanced in several ways to improve accessibility, usability, and effectiveness. A dedicated mobile application for both Android and iOS would allow students to conveniently access chat features, upload documents, and generate quizzes directly from their phones. Integrating voice-based interaction through speech-to-text and text-to-speech technologies would make the system more interactive and especially useful for visually impaired students or those who prefer speaking over typing. Future development can also include seamless integration with college ERP and LMS platforms, enabling automatic synchronization of study materials, timetables, attendance, assignments, and announcements, thereby transforming the chatbot into a complete academic hub.An advanced analytics dashboard could further help students track their learning progress, quiz performance, and study habits, while enabling admins to analyse student engagement, commonly asked questions, and learning trends. Additionally, incorporating multi-language support such as Kannada, Hindi, Tamil, and Telugu would make the system more inclusive and accessible for students who prefer using regional languages.

REFERENCES

[1] H. Zhang, Z. Li, F. Liu, Y. He, Z. Cao, and Y. Zheng, “Design and Implementation of LangChain-based Chatbot,” in Proc. 2024 Int. Seminar on Artificial Intelligence, Computer Technology and Control Engineering (ACTCE)-2024

[2] Kumar P, Harish M et al Department of CSE – “Development of Interactive Assistance for Academic Preparation Using Large Language Models”, Rajalakshmi Engineering College Chennai, India.

[3] Rohan Paul Richard, Ebenezer Veemaraj, et al – “A Client-Server Based Educational Chatbot for Academic Institutions”, Karunya Institute of Technology and Sciences Coimbatore, India

[4] Owen Christian Wijayaand and Ayu Purwarianti-“An Interactive Question-Answering System using Large Language Model and Retrieval-Augmented Generation in An Intelligent Tutoring System on the Programming Domain”,School of Electrical Engineering and Informatics Bandung Institute of Technology Bandung, Indonesia.

[5] Josie C. Calfoforo and Dr. Rodolfo C. Raga, Jr. – “Unleashing AI in Education: A Pre-Trained LLMs for Accurate and Efficient Question-Answering Systems”, National University Manila, Philippines.

[6] Ananya G & Dr. Vanishree K - “RAG based Chatbot using LLMs”, Department of ISE R V College of Engineering Bengaluru, India

[7] Sri Ram G,Rahul S et al- “SCRAPETALK: CHATBOT CONVERSATION WITH WEB SCRAPED INSIGHTS” School of Computer Science and Engineering Vellore Institute of Technology Vellore, India.

[8] Supraj Gijre, Rishi Agrawal and et al –"Enhancing Question-Answering with Knowledge Graph Retrieval and Generation using LLMs" School of Computer Science and Engineering Ramdeobaba University Nagpur, India.

[9] Anuj Jaiswal, Garima Tiwari and et al-" Retrieval Augmented Generation Approach for Multipdf Chatbot using LangChain", Computer Department G H Raisoni College of Engineering & Management, Pune ,India.

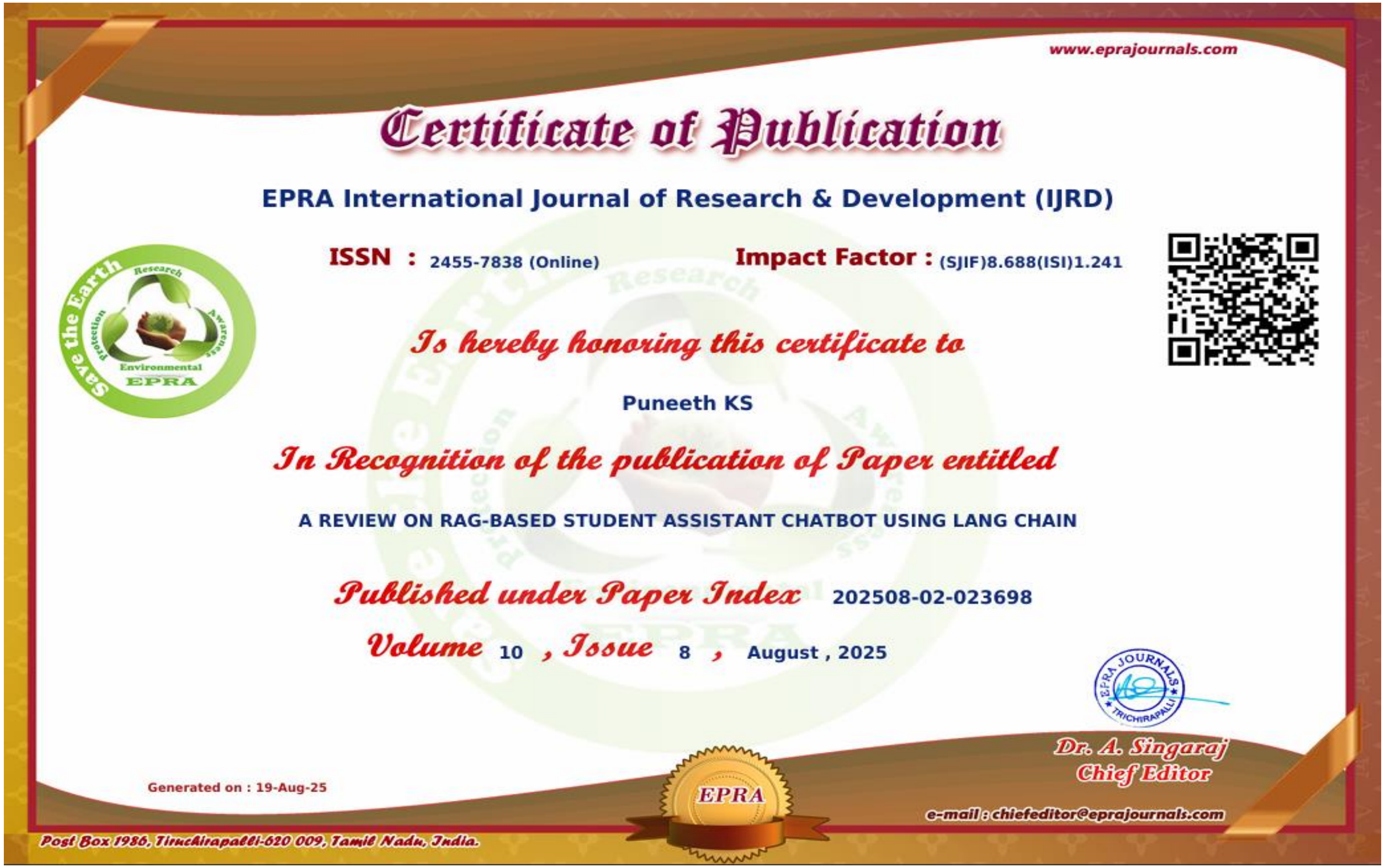
[10] Mandar Kulkarni,Praveen Tangarajan,kyung Kim–"Reinforcement Learning for Optimizing RAG for Domain Chatbots",Flipkart Data Science,Seattle,Washington,USA.

[11] Fel Liu,Zejun Kang and Xing Han –"Optimizing RAG techniques for Automotive industry PDF Chatbots: A case Study With Locally Depolyed ollama Models",china Automotive Technology & Research center.

[12] Subash Neupane, Elias Hossain,Jason Keith et al – "From Questions to Insightfull Answers:Building an informed Chatbot for University Resources",Department of computer science & Engineering, Mississippi State University.

[13] Chiara Antico,stefano Giordano,Cansu koyuturk-"Unimib Assistant: Designing a student-friendly RAG-Based chatbot for all their needs", Milan, Italy.

[14] Rama Akkiraju,Anbang Xu, Deepak Bora et al –"FACTS About Building Retrieval Augmented Generation – Based Chatbots",NVIDIA-2024



www.eprajournals.com

Certificate of Publication

EPRA International Journal of Research & Development (IJRD)

ISSN : 2455-7838 (Online) **Impact Factor :** (SJIF)8.688(ISI)1.241

Is hereby honoring this certificate to

Puneeth KS

In Recognition of the publication of Paper entitled

A REVIEW ON RAG-BASED STUDENT ASSISTANT CHATBOT USING LANG CHAIN

Published under Paper Index 202508-02-023698

Volume 10 , *Issue* 8 , August , 2025

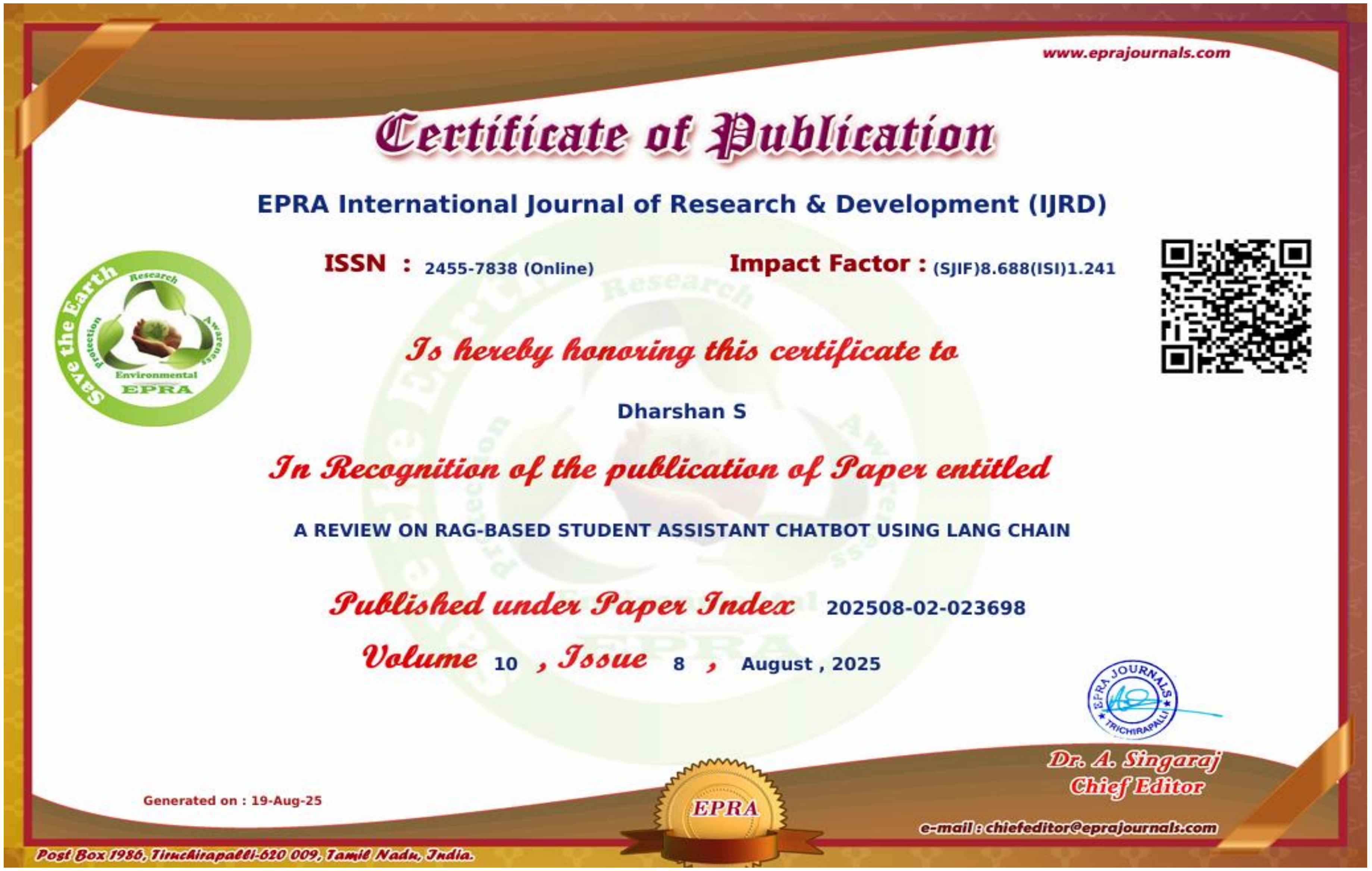
EPRA JOURNALS TRICHIRAPALLI

Dr. A. Singaraj
Chief Editor

Generated on : 19-Aug-25

e-mail : chiefeditor@eprajournals.com

Post Box 1986, Tiruchirapalli-620 009, Tamil Nadu, India.



www.eprajournals.com

Certificate of Publication

EPRA International Journal of Research & Development (IJRD)

ISSN : 2455-7838 (Online) **Impact Factor :** (SJIF)8.688(ISI)1.241

Is hereby honoring this certificate to

Dharshan S

In Recognition of the publication of Paper entitled

A REVIEW ON RAG-BASED STUDENT ASSISTANT CHATBOT USING LANG CHAIN

Published under Paper Index 202508-02-023698

Volume 10 , *Issue* 8 , August , 2025

EPRA JOURNALS TRICHIRAPALLI

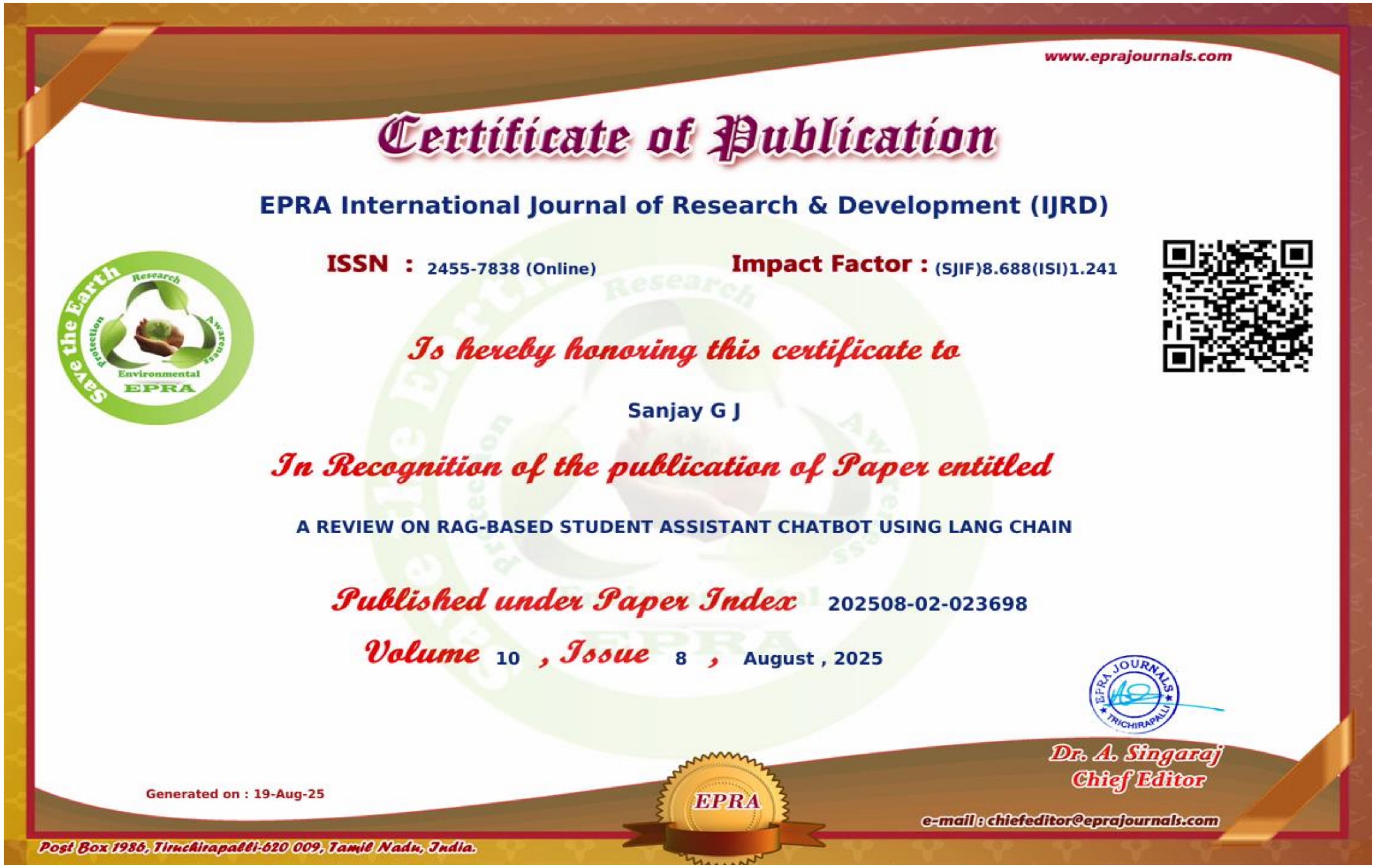
Dr. A. Singaraj
Chief Editor

Generated on : 19-Aug-25

EPRA

e-mail : chiefeditor@eprajournals.com

Post Box 1986, Tiruchirapalli-620 009, Tamil Nadu, India.



www.eprajournals.com

Certificate of Publication

EPRA International Journal of Research & Development (IJRD)

ISSN : 2455-7838 (Online) Impact Factor : (SJIF)8.688(ISI)1.241

Is hereby honoring this certificate to

Sanjay G J

In Recognition of the publication of Paper entitled

A REVIEW ON RAG-BASED STUDENT ASSISTANT CHATBOT USING LANG CHAIN

Published under Paper Index 202508-02-023698

Volume 10 , Issue 8 , August , 2025

EPRA JOURNALS ★ TRICHIRAPALLI ★

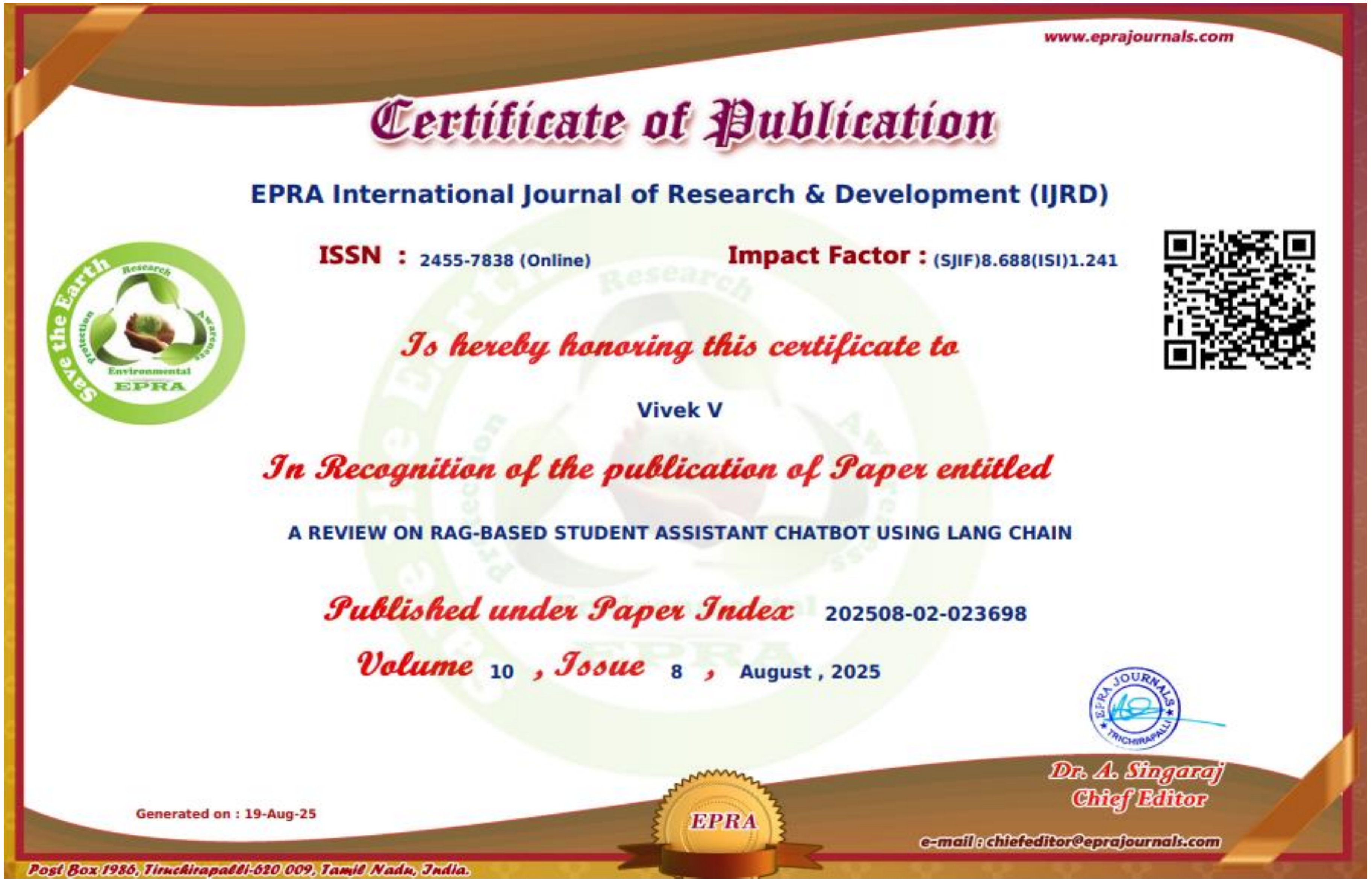
Dr. A. Singaraj
Chief Editor

Generated on : 19-Aug-25

EPRA

e-mail : chiefeditor@eprajournals.com

Post Box 1986, Tiruchirapalli-620 009, Tamil Nadu, India.



www.eprajournals.com

Certificate of Publication

EPRA International Journal of Research & Development (IJRD)

ISSN : 2455-7838 (Online) **Impact Factor :** (SJIF)8.688(ISI)1.241

Is hereby honoring this certificate to

Vivek V

In Recognition of the publication of Paper entitled

A REVIEW ON RAG-BASED STUDENT ASSISTANT CHATBOT USING LANG CHAIN

Published under Paper Index 202508-02-023698

Volume 10 , *Issue* 8 , August , 2025

EPRA JOURNALS TRICHIRAPALLI

Dr. A. Singaraj
Chief Editor

Generated on : 19-Aug-25

EPRA

e-mail : chiefeditor@eprajournals.com

Post Box 1986, Tiruchirapalli-620 009, Tamil Nadu, India.

CHAPTER 1

CHAPTER 2

CHAPTER 3

CHAPTER 4

CHAPTER 5

CHAPTER 6

CHAPTER 7

CHAPTER 8