Design and Implementation of a ChatBot: InteliTalk

by

Saiful Islam
ID: CSE2001019037

Munaimun Bari Fahad
ID: CSE2001019148

Israt Jahan
ID: CSE2001019088

Robinsion Hasda
ID: CSE1901016142

Farhana Sumi
ID: CSE1901016087

Supervised by
Arifur Rahaman

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SONARGAON UNIVERSITY (SU)

May 2024

Design and Implementation of a ChatBot: InteliTalk

by

Saiful Islam
ID: CSE2001019037

Munaimun Bari Fahad
ID: CSE2001019148

Israt Jahan
ID: CSE2001019088

Robinsion Hasda
ID: CSE1901016142

Farhana Sumi
ID: CSE1901016087

Supervised by
Arifur Rahaman

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SONARGAON UNIVERSITY (SU)

May 2024

APPROVAL

The project titled "**Design and Implementation of a ChatBot: InteliTalk**" submitted by Saiful Islam(CSE2001019037), Munaimun Bari Fahad(CSE2001019148), Israt Jahan (CSE2001019088),Robinsion Hasda (CSE1901016142),Farhana Sumi (CSE1901016087) to the Department of Computer Science and Engineering, Sonargaon University (SU), has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and approved as to its style and contents.

Board of Examiners

Arifur Rahaman **Supervisor**
Assistant Professor
Department of Computer Science and Engineering
Sonargaon University (SU)

---------- **Examiner1**
(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name and Signature) **Examiner2**
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name and Signature) **Examiner3**
Department of Computer Science and Engineering
Sonargaon University (SU)

DECLARATION

We, hereby, declare that the work presented in this report is the outcome of the investigation performed by us under the supervision of **Arifur Rahaman, Assistant Professor and Coordinator,** Department of Computer Science and Engineering, Sonargaon University, Dhaka, Bangladesh. We reaffirm that no part of this project has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

(Arifur Rahaman)
Supervisor

Signature

Saiful Islam
ID:CSE2001019037

Munaimun Bari Fahad
ID:CSE2001019148

Israt Jahan
ID:CSE2001019088

Robinsion Hasda
ID:CSE1901016142

Farhana Sumi
ID:CSE1901016087

ABSTRACT

InteliTalk is a cutting-edge chatbot system revolutionizing digital interactions by providing automated responses to user queries. Powered by Langchain and OpenAI Embedding systems, it continuously learns from interactions to improve accuracy. It has separate interfaces for students and guests, ensuring personalized experiences. Administrators can fine-tune conversation flows through an intuitive admin panel, optimizing performance. The system offers an admin panel that allows administrators to manage and customize the system. Technologically, InteliTalk employs the MERN stack for seamless back-end and frontend development, offering real-time updates and scalability. Git/GitHub aids collaborative development, while Docker ensures consistent deployment. CI/CD pipelines enable rapid iteration. Integration with Microsoft Azure enhances scalability and reliability, ensuring high availability across devices. Overall, InteliTalk enhances university communication, providing efficient service and insights for an enhanced user experience.

ACKNOWLEDGMENT

At the very beginning, we would like to express my deepest gratitude to the Almighty Allah for giving us the ability and the strength to finish the task successfully within the schedule time.

We are auspicious that we had the kind association as well as supervision of **Arifur Rahaman,** Assistant Professor and Coordinator, Department of Computer Science and Engineering, Sonargaon University whose hearted and valuable support with best concern and direction acted as necessary recourse to carry out our project.

We are also thankful to all our teachers during our whole education, for exposing us to the beauty of learning.

Finally, our deepest gratitude and love to my parents for their support, encouragement, and endless love.

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
CI/CD	Continuous Integration/Continuous Delivery
BPMN	Business Process Modeling and Notation
CSS	Cascading Style Sheets
DFD	Data Flow Diagram
DOM	Document Object Model
ESM	ECMA Script Modules
GUI	Graphical User Interface
HMR	Hot Module Replacement
HTTP	Hyper Text Transfer Protocol
JSX	JavaScript XML
JWT	Json Web Token
LLM	Large Language Model
MVC	Model-View-Controller
NLP	Natural Language Processing
PFD	Process Flow Diagram
SDLC	Software Development Life Cycle
UML	Unified Modeling Language
UI/UX	User Interface/User Experience

TABLE OF CONTENTS

Title	Page No.
APPROVAL…….	ii
DECLARATION……..……....	iii
ABSTRACT……...……..	iv
ACKNOWLEDGMENT…….…….	v
LIST OF ABBREVIATION ……..……..	vi
CHAPTER 1	1 – 3
INTRODUCTION TO CHATBOT SYSTEM	
1.1 Background and Context……..…….	1
1.2 Problem Statement……...……..	1
1.3 Objectives of the Project ……..…….....	1-2
1.4 Scope and Limitations …….…….	2
1.5 Benefits of chatbot…….……	2-3
1.6 Significance of the Project…….	3
1.7 Organization of the Report……	3
CHAPTER 2	4 – 9
SYSTEM ARCHITECTURE	
2.1 What is System Architecture?……	4
2.2 Monolithic Architecture …….……..	4
2.3 Design Pattern……	5
2.3.1 MVC(Model-View-Controller) Architectural Pattern…….	5-6
2.4 Front-End Technologies……	7
2.5 Back-End Technologies…….	7 – 8
2.6 Working Procedure…….....	9
CHAPTER 3	10 – 12
SYSTEM REQUIREMENTS AND ANALYSIS	
3.1 System Requirements ……...…	10
3.2 Hardware and Software Requirement ……..	10

3.2.1 Hardware ……..…….. 10
3.2.2 Software …….…….... 10
3.3 Functional Requirements…….. 11
3.4 Non-Functional Requirements…….. 11–12

CHAPTER 4 13– 25
SYSTEM DESIGN AND IMPLEMENTATION
4.1 Flowchart …….……. 13-14
4.2 User Interface Design ……... 15– 20
4.3 Integration of Admin Panel……..... 21– 23
4.4 Front-End Implementation……. 24
4.5 Back-End Implementation……. 25

CHAPTER 5 26 – 29
TESTING AND VALIDATION
5.1 Testing Strategies …….…….…… 26
5.2 Test Cases and Results ……..…… 26– 27
5.2.1 Front-End Test Cases……... 26 – 27
5.2.2 Back-End Test Cases……... 27– 28
5.2.3 Additional Testing……... 28
5.3 Bug Tracking …….……... 29

CHAPTER 6 30– 31
SYSTEM EVOLUTION
6.1 Performance Evaluation ……..... 30
6.2 User Feedback …….….. 30
6.3 Scalability and Maintenance…….. 30 – 31
6.4 Comparison with Objectives…….. 31
CHAPTER 7 32– 33

CONCLUSION AND FUTURE WORKS
7.1 Conclusion…… 32

7.2 Limitations……... 32– 33
7.3 Future Works…… 33
REFERENCES ……..…… 34

LIST OF FIGURES

Figure No.	Title	Page No.
Fig.2.1	Monolithic Architecture	4
Fig.2.2	MVC Architecture Pattern	6
Fig.2.3	Technologies	8
Fig.2.4	SDLC Model	9
Fig.4.1	System Flowchart	14
Fig.4.2	Home Page	15
Fig.4.3	Guest UI	16
Fig.4.4	Guest Q n A	17
Fig.4.5	Login UI	18
Fig.4.6	Student Dashboard	19
Fig.4.7	Student Q n A	20
Fig: 4.8	Admin Panel(1)	21
Fig: 4.9	Admin Panel(2)	22
Fig: 4.10	Admin Panel(3)	22
Fig: 4.11	Registration UI	23

CHAPTER 1

INTRODUCTION TO CHATBOT SYSTEM

1.1 Background and Context

In recent years, chatbot technology has rapidly advanced, offering new ways to improve user interactions on websites and applications. These computer programs, able to understand human language, provide instant responses, streamline customer support, and boost user engagement. From answering questions to scheduling appointments, chatbots handle diverse online tasks, allowing brands to automate processes, expedite customer service, and cut support costs.

Chatbots are mainly used in dialog systems for various practical purposes including customer services or information acquisition. There are two main types of chatbots available, one whose functions are based on a set of rules and other is the more advanced version which uses artificial intelligence. The former one tends to be limited and their smartness depends upon the complexity of the program. The more complex the program is, the more is the smartness of the bot. The one that uses artificial intelligence, understands language, not just commands, and continuously gets smarter as it learns from the conversation with the people.[1]

1.2 Problem Statement

Many websites and apps aren't great at helping users find what they need. People often get frustrated when they can't find information quickly, and they might stop using the site or app altogether. So , we're creating a Simple Chatbot System that gives instant answers to user questions. This way, everyone can easily get the information they're looking for without any hassle.

1.3 Objectives of the Project

The primary objective of this project is to develop a chatbot that can answer university student’s and also the outsider’s questions about various university-related topics. The chatbot aims to provide a user-friendly interface that allows students to engage in natural, human-like conversations to receive quick and accurate responses. By offering a reliable communication channel available at any time, the chatbot enhances the student experience by providing

access to important information about academic programs, faculty information, resources, and other aspects of university life.

Through this project, the university aims to automate common queries, improve accessibility, and personalize interactions based on individual student needs. The chatbot will be continuously evaluated and improved based on user feedback to maintain high-quality service.

1.4 Scope and Limitations

The project involves creating a chatbot system and an admin panel to manage responses. However, it doesn't include advanced AI features for natural language processing (NLP). The chatbot's capabilities are limited to answering pre-set questions and cannot handle more complex queries beyond its programmed responses.

1.5 Benefits of Chatbot

The benefits of a chatbot are numerous and include:

1. 24/7 Availability: Chatbots can provide instant responses to user queries at any time of the day or night, enhancing accessibility and user satisfaction.

2. Improved Customer Service: Chatbots streamline customer support processes by offering immediate assistance, reducing wait times, and resolving issues efficiently.

3. Cost Savings: By automating routine tasks and reducing the need for human intervention, chatbots help organisations lower operational costs associated with customer service and support.

4. Scalability: Chatbots can handle multiple inquiries simultaneously, making them scalable solutions for businesses experiencing high volumes of user interactions.

5. Consistent Responses: Chatbots deliver consistent and accurate responses to user queries, ensuring a uniform standard of service across all interactions.

6. Enhanced User Experience: Chatbots provide personalised assistance and tailored recommendations, enhancing the overall user experience and engagement.

7. Data Collection and Analysis: Chatbots can gather valuable user data and feedback, enabling organisations to gain insights into user preferences, behaviours, and trends.

8. Increased Efficiency: Chatbots automate repetitive tasks, such as answering FAQs and processing orders, allowing employees to focus on more complex and strategic activities.

9. Cross-Platform Integration: Chatbots can be integrated with various messaging platforms and websites, providing a seamless user experience across multiple channels.

1.6 Significance of the Project

InteliTalk holds significant value as it offers a cost-effective solution to enrich user experiences on websites. By incorporating this tool, websites can become more engaging and interactive, catering to the needs and queries of users effectively. Beyond just answering common questions, InteliTalk has the potential to serve as a versatile asset for businesses and organizations, facilitating smoother communication and fostering deeper engagement with their audience. Its implementation can lead to improved customer satisfaction, streamlined support processes, and ultimately, a more successful online presence for businesses of all sizes.

1.7 Organization of the Report

The report is structured into several chapters, providing a comprehensive overview of the project journey. Each chapter delves into specific aspects, starting from the foundational system architecture and requirements analysis, progressing through the design and implementation phases, and culminating in thorough testing, validation, and system evaluation. Additionally, insights into project management methodologies are provided, offering valuable lessons learned and best practices employed throughout the project lifecycle. Finally, the report concludes with key findings, recommendations, and reflections, summarizing the project's outcomes and implications for future endeavors.

CHAPTER 2

SYSTEM ARCHITECTURE

2.1 What is System Architecture?

System architecture has been defined in various ways. It has been defined as a structure composed of components, and rules characterizing the interaction of these components. It has been defined as components, connections, constraints and rationale. It has also been defined as elements, form, and rationale and as components, connectors, and configurations.[2]

2.2 Monolithic Architecture

A monolithic application is an application that uses a single codebase to serve numerous different services and different interfaces such as REST (Representationalstate transfer), APIs and HTML pages. The monolithic approach is considered as the standard way to start developing applications. A single codebase eases the development, deployment and scaling of the application as long as the size of the codebase is relatively small. A monolithic codebase is a good choice at the start of the project because of the aforementioned qualities and because there is not any distribution of code which would add complexity.[3]
This project follows the monolithic architecture.

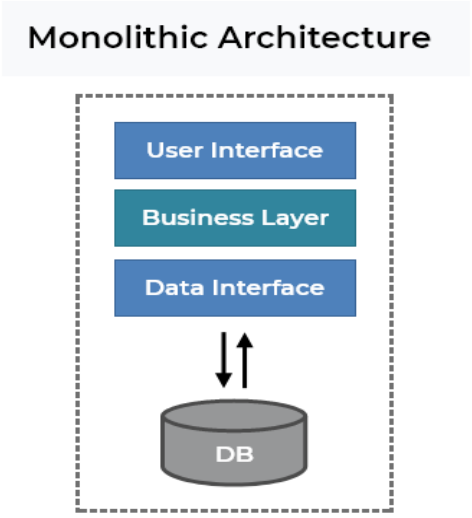


Fig : 2.1 Monolithic Architecture

2.3 Design Pattern

Software design patterns are recurring solutions to common problems encountered during software development. They provide a structured approach for designing software systems that are flexible, maintainable, and scalable.

2.3.1 MVC(Model-View-Controller) Architectural Pattern

MVC stands for model-view-controller. Model-View-Controller (MVC) is an architectural pattern used in software development for graphical user interfaces. It was one of the first proposed solutions in the late 1970s to the Smart UI anti-pattern, which refers to the act of writing all domain logic into a user interface. The original MVC pattern has since evolved in multiple directions, with various names and may confuse many. The goal of this thesis is to present the origin of the MVC pattern and how it has changed over time. Software architecture in general and the MVC's evolution within web applications are not the primary focus. Fundamental designs are abstracted, and then used to examine the more recent versions. Problems with the subject and its terminology are also presented.[4]

Here's what each of those components mean:

- **Model:** the Model serves as the backbone of the application, managing the data storage, retrieval, and manipulation processes. It encapsulates the business logic and ensures data integrity and consistency.
- **View:** The View component represents the user interface, presenting information to users in a visually appealing and understandable manner. It handles the display of data and interacts with users to gather input or convey information.
- **Controller:** The Controller acts as the intermediary between the Model and View, orchestrating the flow of data and controlling the application's behavior. It receives input from the user via the View, processes it, and updates the Model accordingly. The Controller also determines how the updated data should be presented in the View, ensuring a seamless user experience.

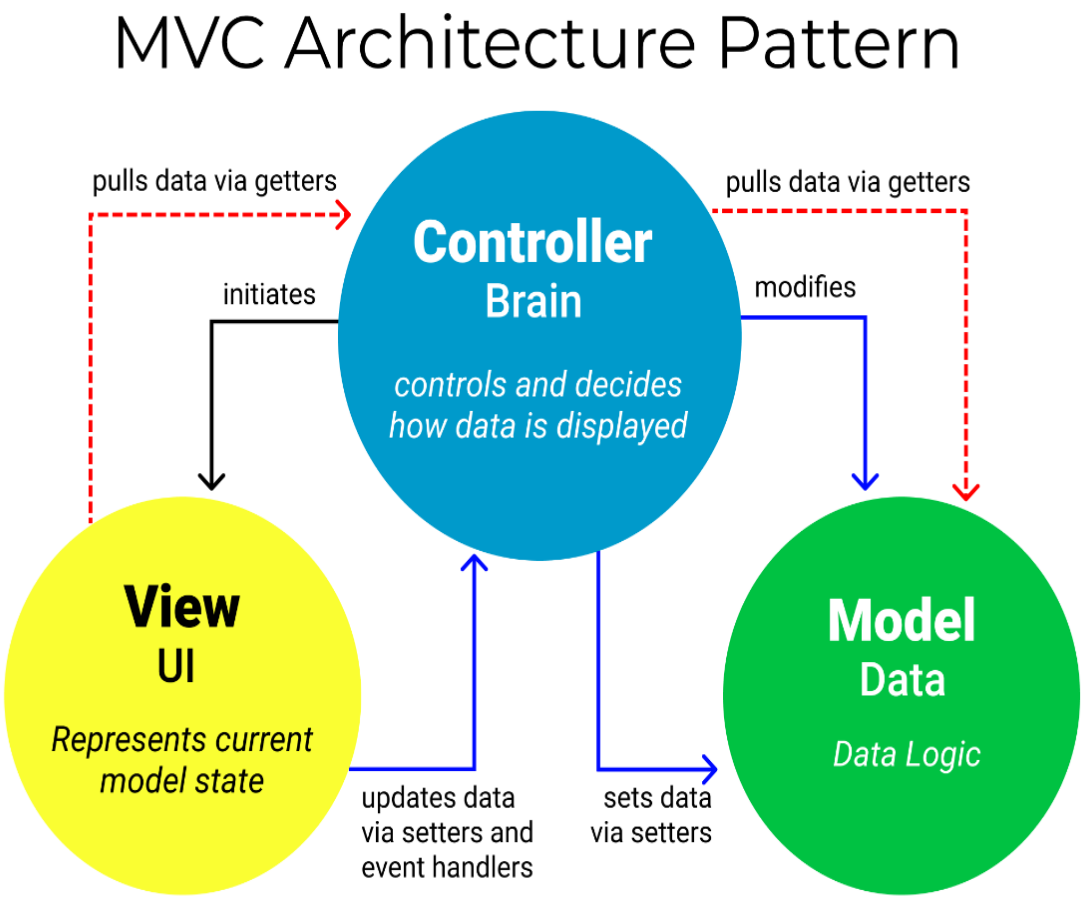


Fig: 2.2 MVC Architecture Pattern

Together, these components work in harmony to create a well-structured and efficient software architecture. The Model, View, and Controller collaborate to provide users with a cohesive and intuitive application interface, enhancing usability and facilitating smooth interaction.

2.4 Front-End Technologies

- **React.js:** The library for web and native user interfaces.
- **Vite:** Vite is a JavaScript tool that provides a fast, rich and typed frontend development environment. It supports ESM, HMR, TypeScript, JSX, CSS and more features for web developers.
- **TailwindCSS:** A utility-first CSS framework for rapidly building custom designs.
- **Axios:** Axios is a popular library for making XMLHttpRequests and http requests from the browser and node.js. It supports the Promise API, interceptors, transformers, cancellation, and automatic serialization of data.
- **React-Router-Dom:** React Router is a library for routing and navigation in React applications.

2.5 Back-End Technologies

- **Node.js:** Node.js is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts.
- **ExpressJS:** Fast, unopinionated, minimalist web framework for Node.js.
- **MongoDB:** MongoDB is a source-available, cross-platform, document-oriented database program. Classified as a NoSQL database product, MongoDB utilizes JSON-like documents with optional schemas.
- **Langchain:** LangChain is a framework for developing applications powered by large language models (LLMs).
- **OpenAI API:** OpenAI API to create an intelligence layer in your applications, powered by state of the art models like GPT-4.
- **Git & GitHub:**Git is a distributed version control system that tracks the history of changes in projects. GitHub hosts Git repositories and provides tools to collaborate, ship, and manage code with Git.
- **Postman:**Postman is a software that helps you test and document your APIs. It offers a set of tools, a repository, workspaces, and governance to simplify the API lifecycle and collaboration.
- **CI/CD:**CI/CD, which stands for continuous integration and continuous delivery/deployment, aims to streamline and accelerate the software development lifecycle.Continuous integration (CI) refers to the practice of automatically and

frequently integrating code changes into a shared source code repository. Continuous delivery and/or deployment (CD) is a 2 part process that refers to the integration, testing, and delivery of code changes. Continuous delivery stops short of automatic production deployment, while continuous deployment automatically releases the updates into the production environment.

- **Docker:** Docker helps developers build, share, run, and verify applications anywhere — without tedious environment configuration or management.
- **Render:** Render is a unified cloud to build and run all your apps and websites with free TLS certificates, global CDN, private networks and auto deploys from Git.



Fig : 2.3 Technologies

2.6 Working Procedure

In any project, it's important to have a clear plan and set of rules to follow. We use something called the Software Development Life Cycle (SDLC) to guide us through our work. This means we break our project into steps and follow them carefully from start to finish. Each step helps us move closer to reaching our goals. By sticking to this plan, we make sure we're organized and don't miss anything important. It helps us do our work effectively and deliver the best results possible.

Diagrammatic representation of these steps of agile model is given below:

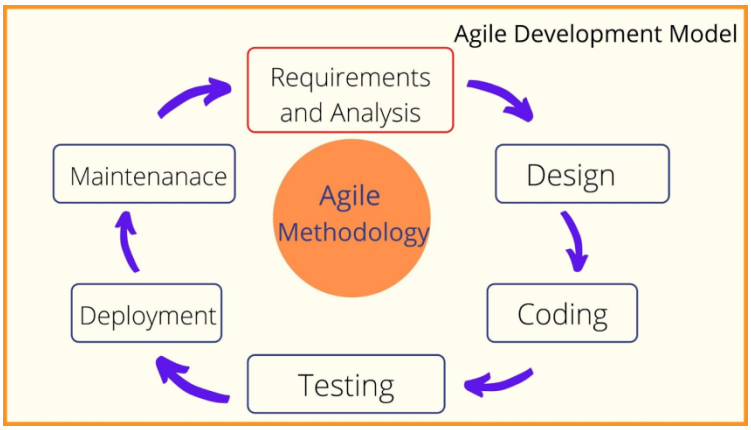


Fig: 2.4 SDLC Model

CHAPTER 3

SYSTEM REQUIREMENTS AND ANALYSIS

3.1 System Requirements

System requirements refer to the functionalities necessary for a system to meet user needs effectively. System requirements are a crucial aspect of software development, as they define the foundation upon which your software will run. They encompass a wide range of aspects critical for the system's performance. Without clear system requirements, a project is destined for failure, risking the success of the company involved.

Conversely, having well-defined system requirements greatly enhances the likelihood of success, as it ensures alignment with customer needs and expectations. Effective system requirements not only contribute to project success but also facilitate cost and time savings. By clearly outlining what the system needs to achieve, organizations can streamline development processes, minimize errors, and
allocate resources efficiently.

Robust system requirements serve as the foundation for successful project outcomes, enabling companies to deliver solutions that meet customer needs while optimizing operational efficiency and resource utilization.

3.2 Hardware and Software Requirement

3.2.1 Hardware:

Processor	Pentium-II or higher
Processor Speed	533 MHZ
Hard Disk Space	20 GB (min.)
Ram Memory	32 MB (64 MB recommended)

3.2.2 Software:

Operating System	Windows 95/98/NT/2000 /10/7/8/11
Text Editor	Visual Studio Code
Testing Software	Postman

3.3 Functional Requirements

Functional requirements are specifications that describe the behavior of a system. They detail what the system should do, how it should work internally, and how it should interact with users.

Functional requirements detail:

- **System Tasks:** Describing the tasks the system should perform.
- **Processes:** Outlining the steps involved in completing these tasks.
- **Data Requirements:** Specifying the data the system should manage.
- **User Interfaces:** Defining how users interact with the system.
- **Registration:** Users can create accounts with required details.
- **Login:** Registered users can access their accounts securely.
- **Feedback:** Users can leave feedback on their experiences, enhancing system improvement.

3.4 Non-Functional Requirements

Non-functional requirements (NFRs) define critical system attributes such as security, reliability, performance, maintainability, scalability, and usability. These requirements act as constraints or guidelines for the system's design, ensuring its effectiveness and usability.
Non-functional requirements are essential for ensuring the reliability, usability, and overall effectiveness of the system. They complement functional requirements and contribute to the successful implementation and operation of the system.

Key non-functional requirements include:

- **Security:** The system must maintain a high level of security and data integrity. Only users with valid credentials should be able to access sensitive information.

- **Performance and Response Time:** The system should demonstrate high performance, providing prompt responses to user inputs. Response times for complex tasks should be within 50 seconds, while less complicated tasks should receive responses within 20 to 25 seconds.

- **Error Handling:** The system must validate user inputs effectively and efficiently. Additionally, the system should be capable of recovering from errors within 15 to 20 seconds, minimizing disruptions to user experience.

CHAPTER 4

SYSTEM DESIGN AND IMPLEMENTATION

4.1 Flowchart

A flowchart is a diagram that depicts a process, system or computer algorithm. They are widely used in multiple fields to document, study, plan, improve and communicate often complex processes in clear, easy-to-understand diagrams. Flowcharts, sometimes spelled as flow charts, use rectangles, ovals, diamonds and potentially numerous other shapes to define the type of step, along with connecting arrows to define flow and sequence. They can range from simple, hand-drawn charts to comprehensive computer-drawn diagrams depicting multiple steps and routes. If we consider all the various forms of flowcharts, they are one of the most common diagrams on the planet, used by both technical and non-technical people in numerous fields.

Flowcharts are sometimes called by more specialized names such as Process Flowchart, Process Map, Functional Flowchart, Business Process Mapping, Business Process Modeling and Notation (BPMN), or Process Flow Diagram (PFD). They are related to other popular diagrams, such as Data Flow Diagrams (DFDs) and Unified Modeling Language (UML) Activity Diagrams.

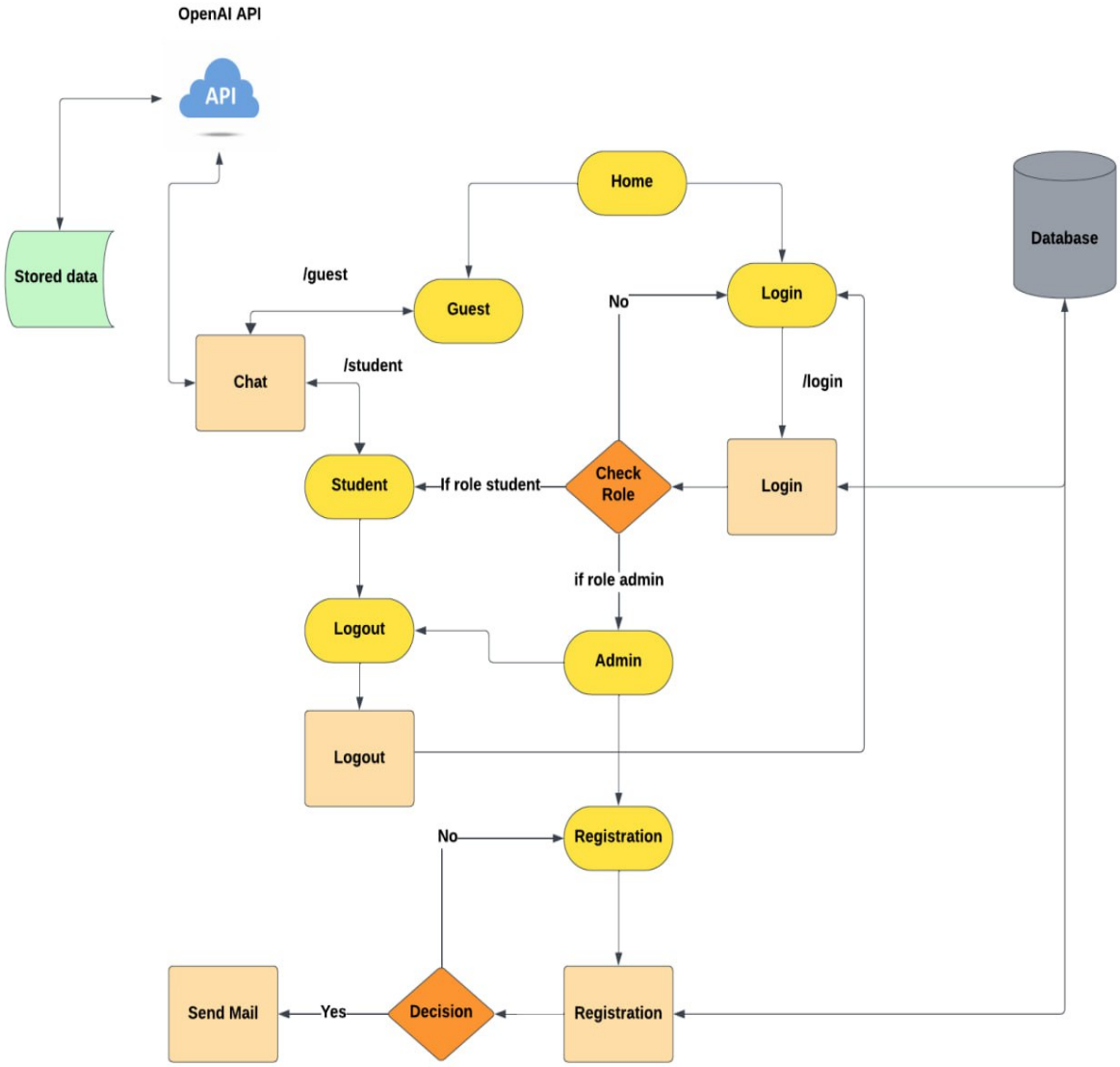


Fig: 4.1 System Flowchart

4.2 User Interface Design

The user interface is designed to be user-friendly, with a chat-like conversation flow. React is great for building dynamic user interfaces, and with the addition of React Router DOM, can easily handle navigation within app. Tailwind CSS is excellent for quickly styling components without having to write custom CSS, and react-toast notifications can provide a sleek way to communicate with users. Overall, combining these technologies should result in a smooth, responsive, and visually appealing user experience.

User Interface (UI) :

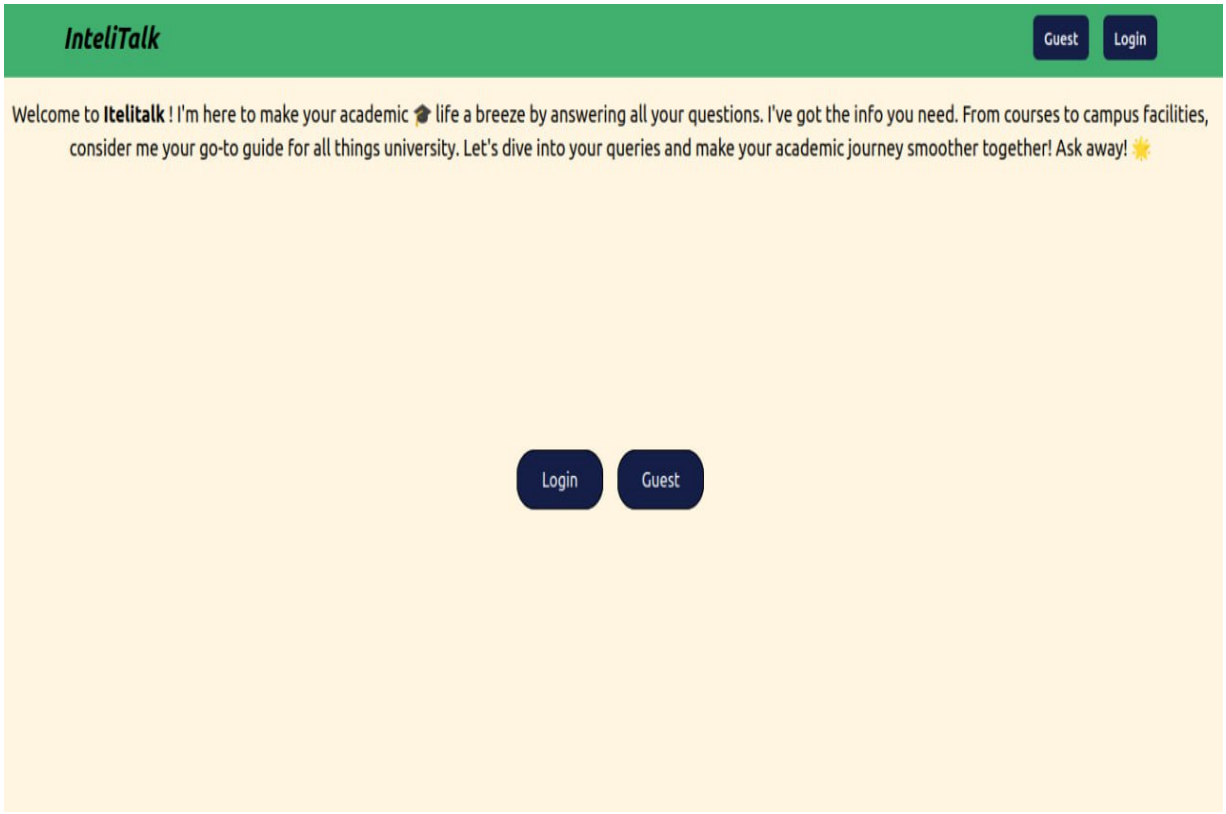


Fig: 4.2 Home Page

Guest UI :

The chatbot's guest UI lets outsiders ask university questions without logging in. It provides quick and easy access to information about the university

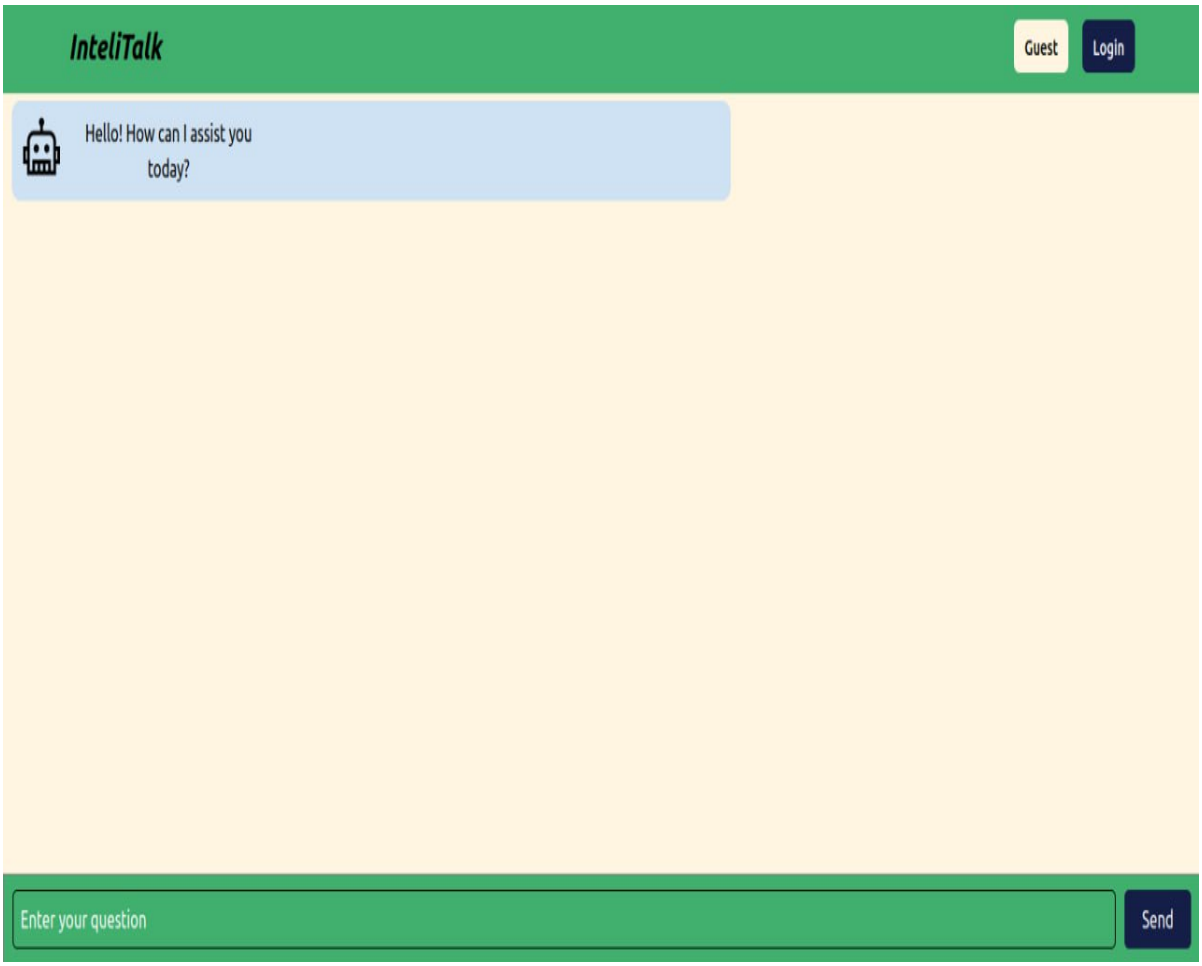


Fig: 4.3 Guest UI

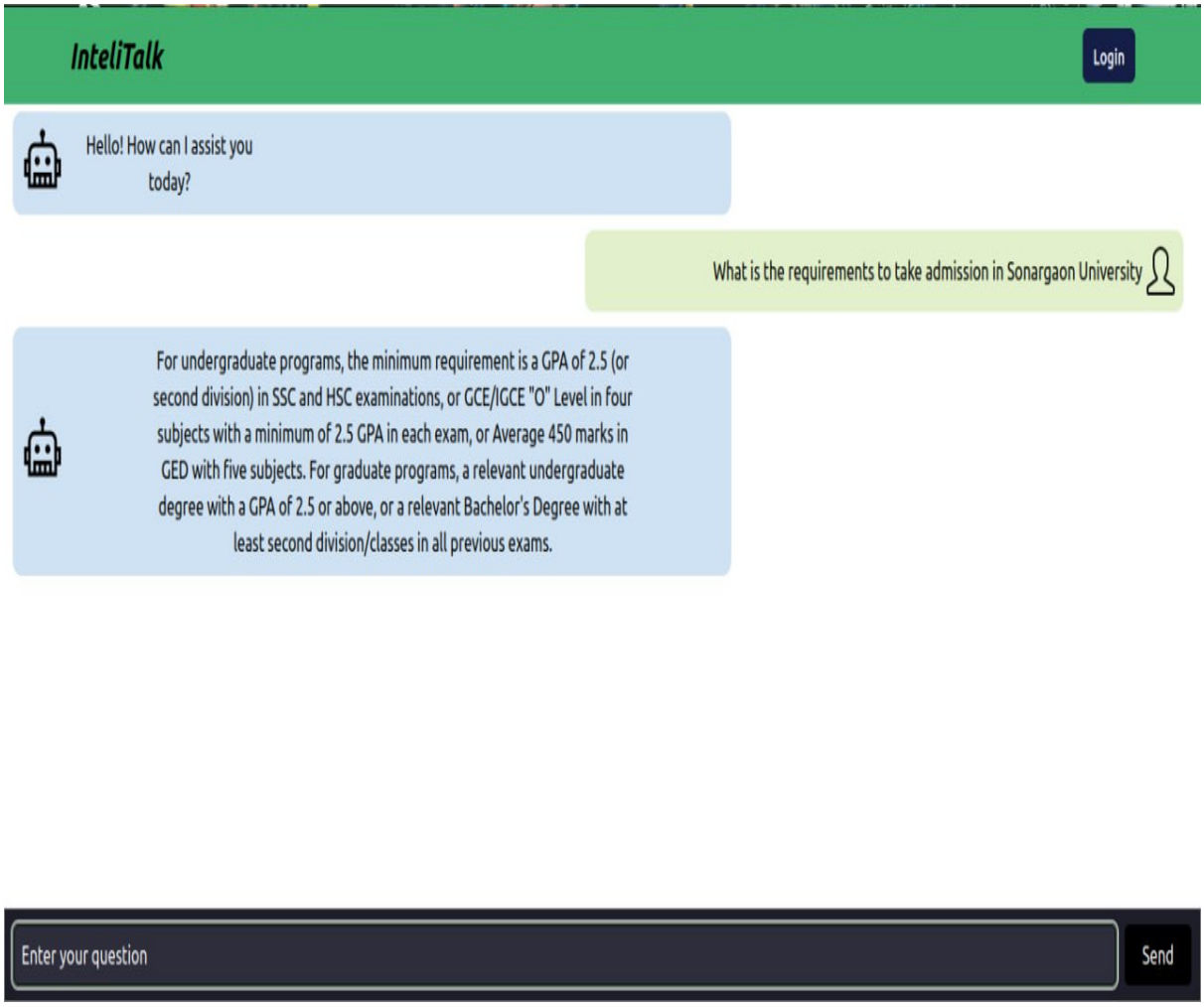


Fig: 4.4 Guest Q n A

Login UI :

The chatbot has a login UI where users can log in and access either the student or admin interface based on their role. Students can ask questions and get responses, while admins can manage content.

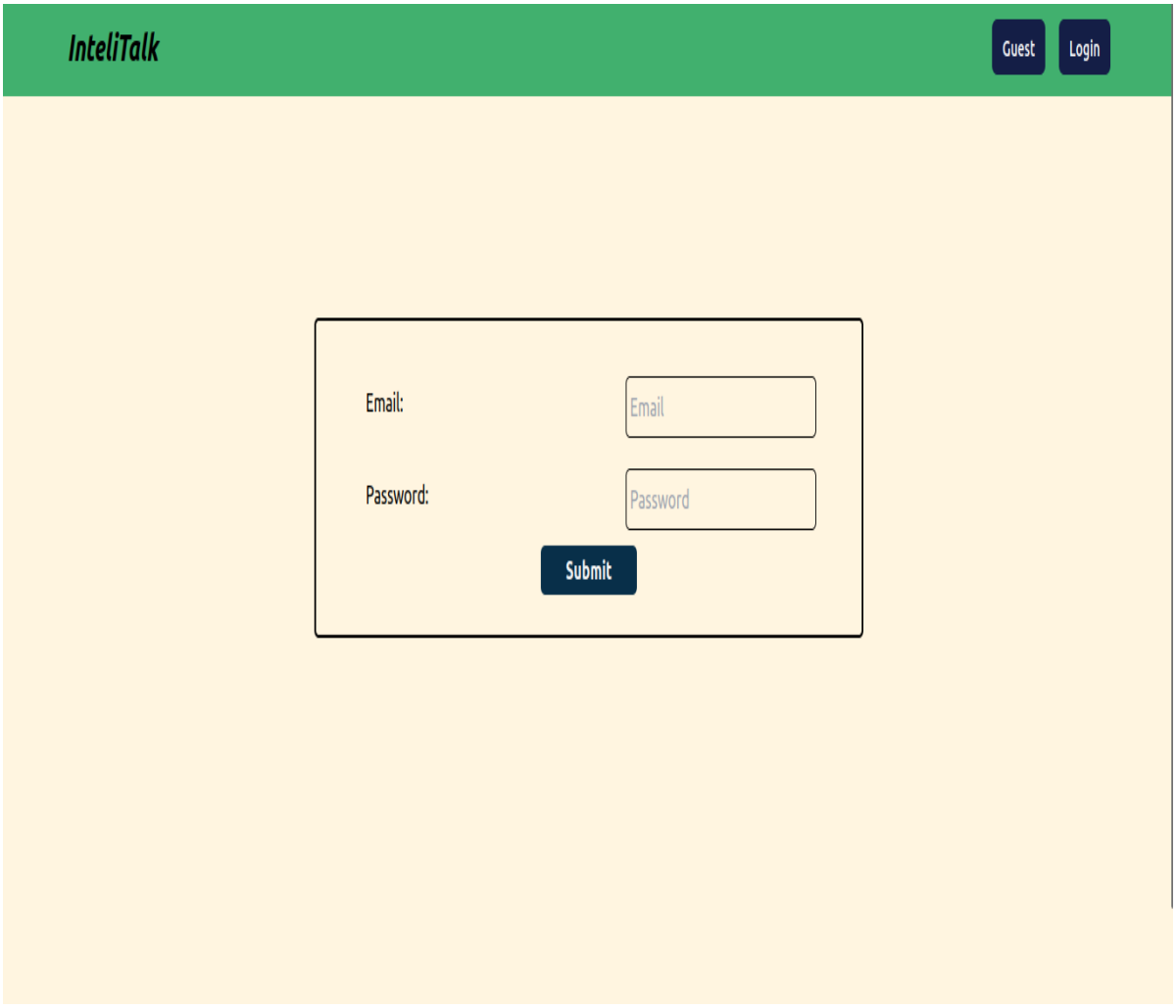


Fig: 4.5 Login UI

Student UI :

The student UI allows logged-in students to ask questions and receive personalized answers about the university.

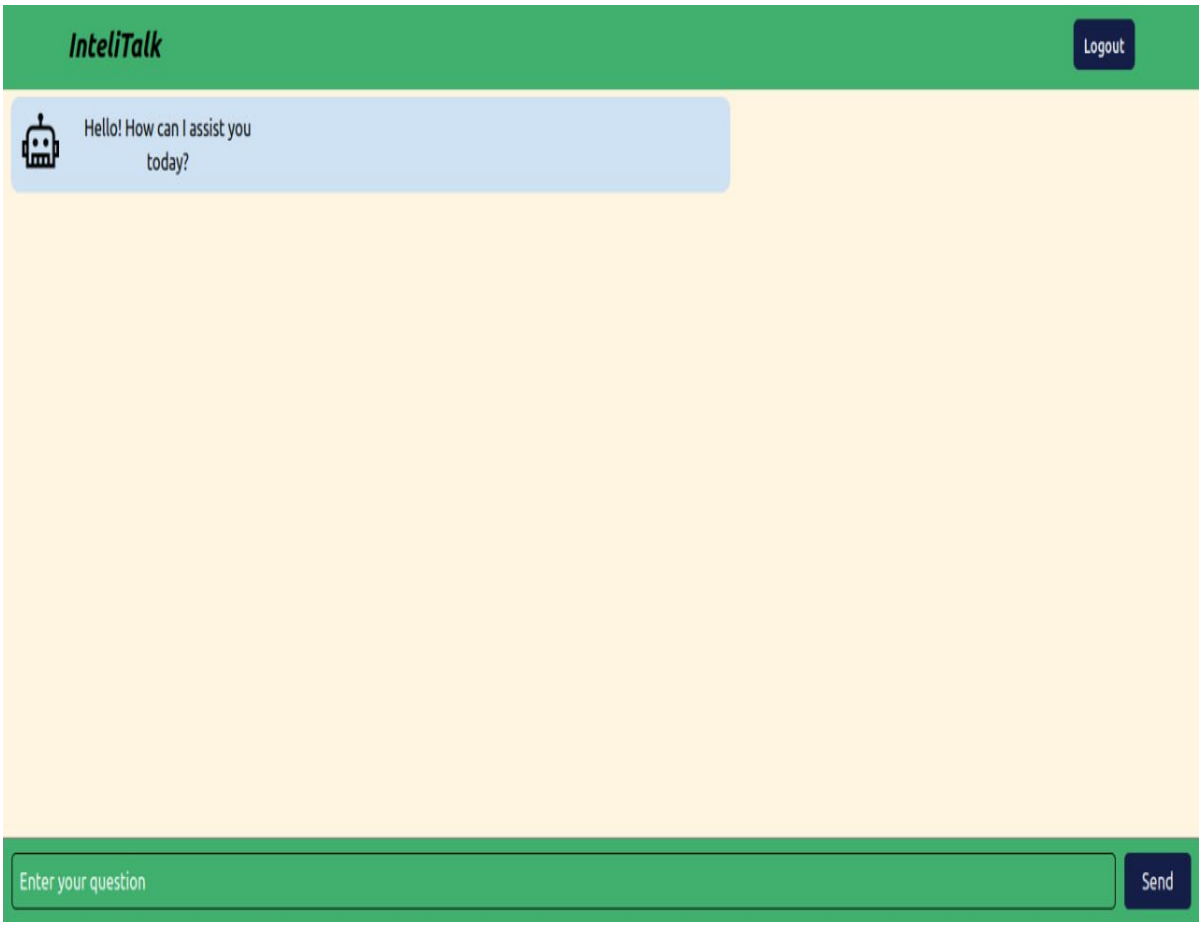


Fig: 4.6 Student Dashboard



Fig: 4.7 Student Q n A

4.3 Integration of Admin Panel :

The admin panel lets admins register new students and view a list of all users. It helps manage student accounts and keep track of the user database.

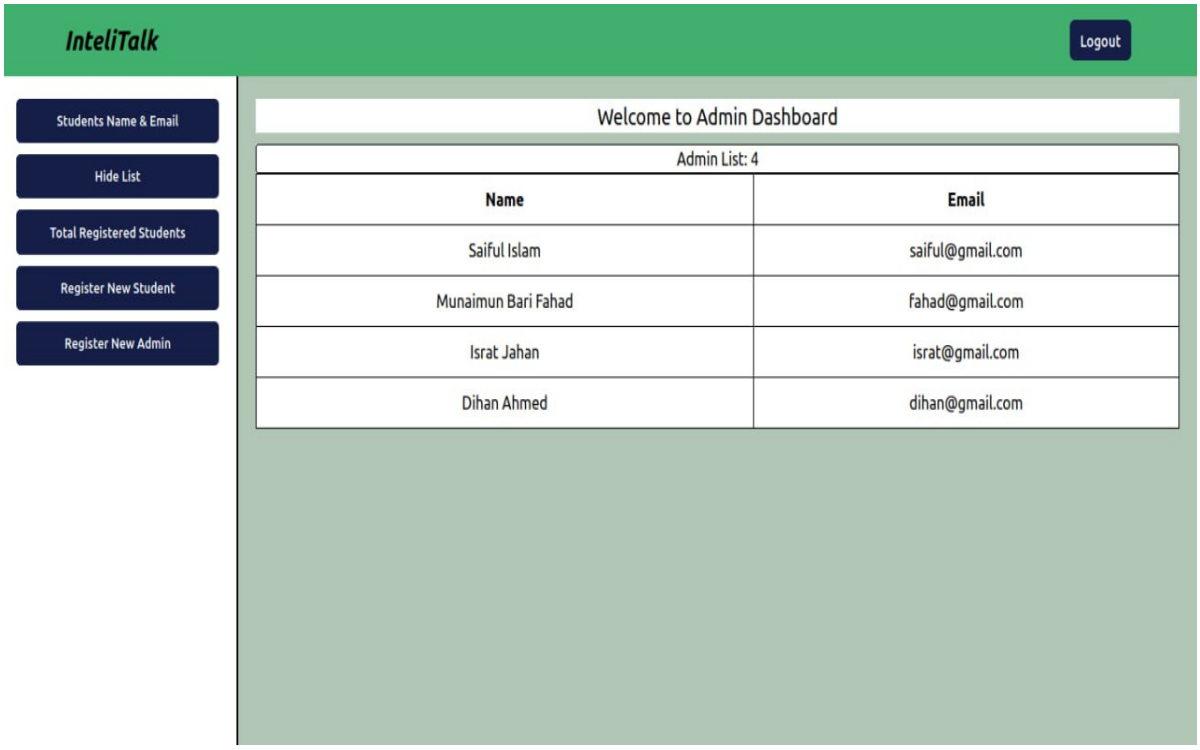


Fig: 4.8 Admin Panel(1)



Fig: 4.9 Admin Panel(2)

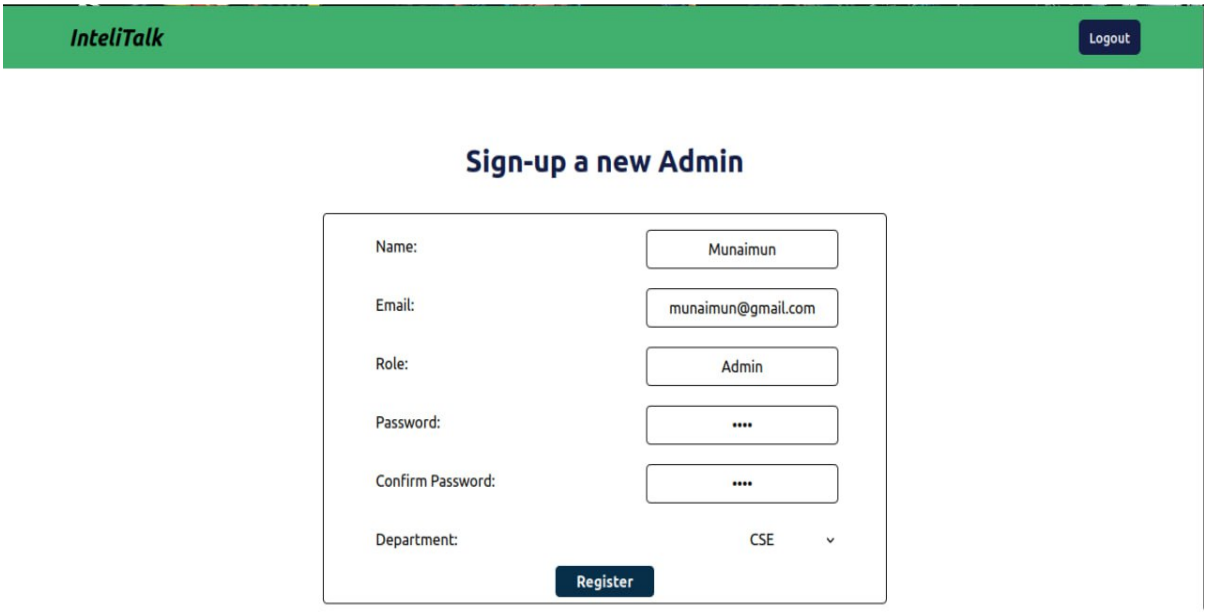


Fig: 4.10 Admin Panel(3)

Registration UI :

The registration UI lets admins quickly register a new student by entering their details and saving them in the system. This makes adding new students easy and efficient.

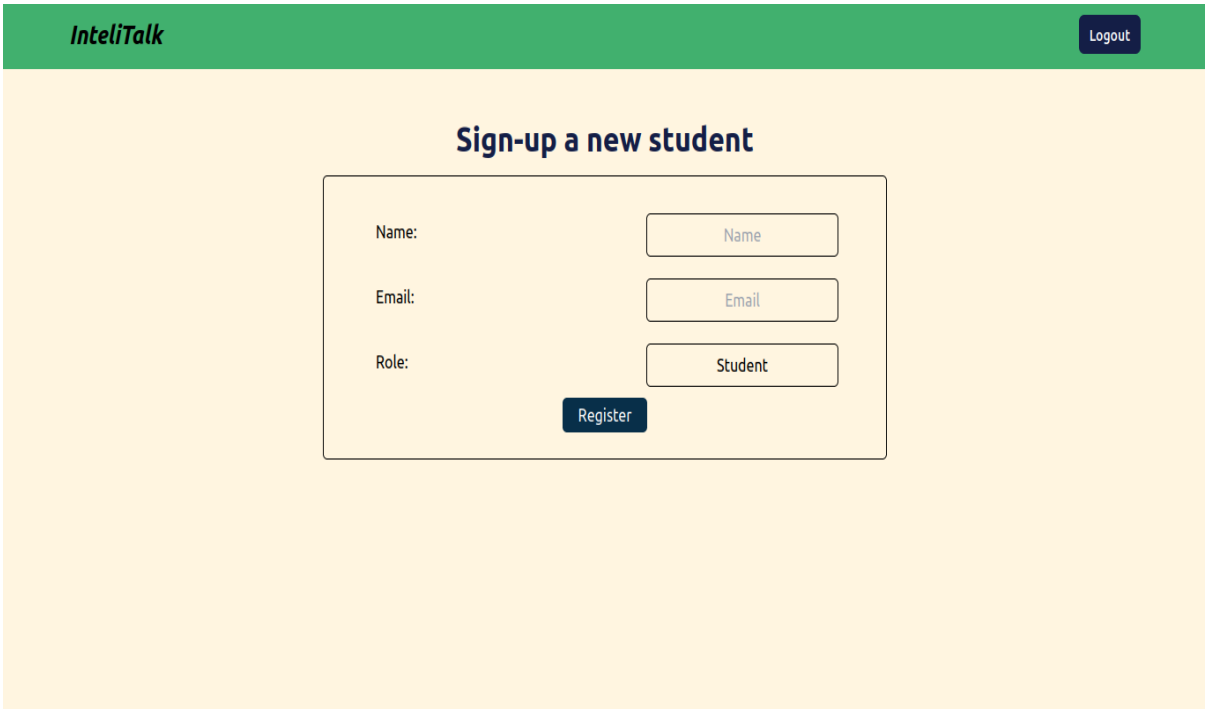


Fig: 4.11 Registration UI

4.4 Front-End Implementation

Front-end development is the process of creating the visible parts of a website or web application that users interact with directly. It focuses on crafting the user interface (UI) to ensure it's visually appealing, intuitive, and easy to navigate. This aspect of web development is crucial as it directly impacts the user experience (UX) and influences how users perceive and interact with the website.

Front-end developers utilize various tools and technologies to build and enhance the user interface. For instance, ReactJS, a popular JavaScript library, enables developers to create dynamic and interactive UI components. CSS frameworks like Tailwind CSS are used to streamline the styling process by providing pre-built components and utility classes for consistent styling.

Navigation is another key aspect of front-end development, and libraries like React Router are commonly used to handle routing and rendering of components based on the URL. Additionally, features like React Toast notifications provide feedback to users, informing them about important events or actions on the website.

Communication between the front-end and back-end of a web application is facilitated by tools like Axios, a popular HTTP client for JavaScript, enabling seamless data exchange without page reloads. Moreover, accessibility considerations play a crucial role in front-end development, ensuring that websites are accessible to users of all abilities by following web accessibility standards and guidelines.

Overall, front-end development aims to create a harmonious blend of aesthetics and functionality, catering to diverse user needs and preferences. By prioritizing design, responsiveness, and accessibility, front-end developers strive to deliver a captivating and user-friendly interface that leaves a lasting impression on visitors. Integrating additional features like React Table components can further enhance the presentation and interaction of data within the user interface, contributing to an enhanced overall user experience..

4.5 Back-End Implementation

The back-end of a website is like the engine of a car. You might not see it, but it's what keeps everything running smoothly behind the scenes. It's made up of different parts, each with its own job.

First, there's Node.js and Express.js. These are like the mechanics that make sure the server, or the main computer of the website, works properly. They help with managing requests from users and sending back the right responses.

Then, there's MongoDB, which is like a storage room for all the website's data. It's where information like user profiles and posts are kept safe and organized.

Next up are OpenAI's API and Langchain. These are like the brainy assistants that add cool features to the website, such as chatbots or language translation. They use smart algorithms to make the website more helpful and interactive.

For keeping things secure, there's JSON Web Tokens (JWT). Think of JWT as the secret code that lets users access certain parts of the website securely. It makes sure only the right people can see and do what they're supposed to.

Docker is like the shipping container for the website. It packages everything up neatly so it can be easily moved around and deployed, or launched, on different computers.

Lastly, there's Continuous Integration/Continuous Deployment (CI/CD). This is like the assembly line that automates the process of updating and improving the website. It makes sure changes are tested and rolled out smoothly without causing any hiccups.

Even though users don't directly see or interact with the back-end, it's the backbone that supports everything they do on the website, making sure it all works smoothly and securely.

CHAPTER 5

TESTING AND VALIDATION

5.1 Testing Strategies

Testing strategies for chatbot development typically involve three key phases: unit testing, integration testing, and user acceptance testing. These tests ensure that the application functions as intended and provides a satisfactory user experience.

Testing simply refers to the validation and verification specially to built a good quality software. Testing also defined as “Software testing is technique of evaluating the attributes (i.e. correctness, completeness, security, consistency, unambiguous-ness, quality etc.) of software and determining that whether it meets its required functionality or not”. Purpose of testing is to find out Defects and causes
and fixed them as early as possible.[5]

Conversational interfaces, such as chatbots, offer the opportunity to engage with customers at scale and around the clock. However, if they fail to deliver a great customer experience, they can do so at scale as well. Therefore, testing should be a fundamental aspect of any chatbot development process.Testing helps evaluate the chatbot's performance in terms of accuracy, speed, and usefulness. It provides insights into areas for improvement, such as language refinement, personality adjustments, and feature enhancements. Additionally, testing helps streamline the conversational flow and identify any potential issues or bottlenecks in the user interaction process.

5.2 Test Cases and Results

Test cases include scenarios where users interact with the chatbot, and expected outcomes are defined. Testing results indicate the success of these scenarios.

5.2.1 Front-End Test Cases:

1. UI/UX Testing:

- **Test Case:** Verify that the navigation menu works correctly.
    - **Steps:** 1. Click on each menu item in the navigation menu.

- **Expected Result:** The corresponding page is displayed upon clicking each menu item.

- **Test Case:** Verify that toast notifications are displayed correctly.
    - **Steps:** 1. Trigger a toast notification event.
    - **Expected Result:** Toast notification is displayed at the specified position with the correct message.

2. API Integration:

- **Test Case:** Verify that data is fetched from the server when the dashboard loads.
    - **Steps:** 1. Login with valid credentials.

2. Check if data is loaded on the dashboard.

    - **Expected Result:** Dashboard displays relevant data fetched from the server.

- **Test Case:** Verify that data can be submitted to the server.
    - **Steps:** 1. Fill in the required fields in a form.

2. Submit the form.

    - **Expected Result:** Data is successfully submitted to the server and reflected in the UI.

3. User Authentication:

- **Test Case:** Verify that users can register with valid credentials.
    - **Steps:** 1. Navigate to the registration page.
        2. Enter valid user credentials.
        3. Click on the register button.
    - **Expected Result:** User is successfully registered.
- **Test Case:** Verify that users can login with valid credentials.
    - **Steps:** 1. Navigate to the login page.
        2. Enter valid login credentials.
        3. Click on the login button.
    - **Expected Result:** User is successfully logged in and redirected to the dashboard.

5.2.2 Back-End Test Cases:

1. Authentication and Authorization:

- **Test Case:** Verify that users can register with valid credentials.

- **Steps:** 1. Send a POST request to the registration endpoint with valid user credentials.
- **Expected Result:** User is successfully registered, and a corresponding entry is created in the database.

- **Test Case:** Verify that users can login with valid credentials.
  - **Steps:** 1. Send a POST request to the login endpoint with valid login credentials.
  - **Expected Result:** User receives a JWT token upon successful login.

2. API Endpoints Testing:

- **Test Case:** Verify that the API endpoint for fetching data works correctly.
  - **Steps:** 1. Send a GET request to the data endpoint.
  - **Expected Result:** Server responds with the requested data.
- **Test Case:** Verify that the API endpoint for submitting data works correctly.
  - **Steps:** 1. Send a POST request with valid data to the submission endpoint.
  - **Expected Result:** Data is successfully stored in the database.

5.2.3 Additional Testing:

1. Integration Testing:

- **Test Case:** Verify the integration between front-end and back-end.
  - **Steps:** 1. Perform user actions on the front-end UI.
    2. Verify the corresponding changes in the back-end database.
  - **Expected Result:** Front-end actions are reflected correctly in the back-end.

2. Security Testing:

- **Test Case:** Verify that authentication tokens are securely stored.
  - **Steps:** 1. Inspect browser storage for authentication tokens.
  - **Expected Result:** Authentication tokens are securely stored and not accessible to unauthorized parties.

5.3 Bug Tracking

Bug tracking is like having a to-do list for fixing problems in a program. It helps keep track of all the issues and makes sure they get fixed in the right order. A bug tracking system usually has a database where it keeps information about each problem. This includes when it was found, how serious it is, what went wrong, and how to make it happen again. It also keeps track of who found the problem and who's working on fixing it.

Bug tracking is super important because it helps make sure the program works smoothly without any hiccups. It's like having a detective to solve mysteries in the program and make it better for everyone to use.

CHAPTER 6

SYSTEM EVOLUTION

6.1 Performance Evaluation

Performance evaluation is like checking the health of your computer. You want to make sure it's running smoothly and not slowing down. Similarly, with chatbots, we check following things:

- **Response Time:** How quickly the chatbot responds to messages.
- **System Up-time:** How long the chatbot stays up and running without any breaks.
- **Server Resource Utilization:** How much of the computer's power the chatbot uses.

Just as you want your computer to handle multiple tasks without crashing, we design chatbots to handle a certain number of users at once. Whether customers reach out through phone, email, or website, chatbots aim to provide a great online experience. To ensure they're doing their job effectively, we measure and track their performance regularly.

6.2 User Feedback

User feedback is like getting advice from your customers about how to improve your business. We collect feedback to see if the chatbot is easy to use and if it's doing its job well. Surveys and interviews with users give us important information. Just like listening to customers helps businesses grow, collecting feedback helps online businesses understand what their customers want.

There are different ways to gather feedback, but sometimes the usual forms don't work well for online shops. So, we find new ways to make sure we hear from our customers and make our chatbots even better for them.

6.3 Scalability and Maintenance

Scalability and maintenance are crucial aspects of the chatbot system. We evaluate scalability to ensure the system can handle a growing number of users and increased traffic without slowing down or crashing.

Software maintenance is one of the most expensive and resource requiring phase of the software development process. Maintainability is the capability of the software to be modified from the perspective of component based systems, wherein the concept of maintainability is of major importance. Maintainability of component based systems is a complex issue, since maintenance personnel does not have access to the source code of the component, which needs to be modified.[6]

Maintenance is essential for several reasons:

- **Improved Accuracy:** Regular maintenance ensures the chatbot stays up-to-date with the latest information, reducing the chances of giving incorrect or irrelevant responses.

- **Better Engagement:** Without maintenance, the chatbot's ability to engage with users diminishes over time, affecting the quality of interactions.

- **Efficiency and Promptness:** An updated chatbot is more efficient in answering user queries promptly, leading to higher satisfaction levels.

- **Seamless User Experience:** Regular maintenance keeps the chatbot optimized and free of minor issues, ensuring a smooth user experience.

- **Cost Savings:** Maintenance not only improves the chatbot's performance but also increases the productivity of support agents, leading to cost savings.

6.4 Comparison with Objectives

We check to see if we did what we set out to do in the project. If there are any differences or things that turned out better than expected, we take note. This helps us understand what worked well and what we can do better next time.

CHAPTER 7

CONCLUSION AND FUTURE WORKS

7.1 Conclusion

our chatbot system for universities is a real breakthrough. Think of it as having a knowledgeable friend on campus whenever you need help. With its ability to understand natural language and learn from each interaction, it's like having a personal assistant for all your university-related queries. While there's still room for growth, especially in tackling more intricate issues, our chatbot system represents a significant stride towards enhancing the student experience and streamlining university services.

7.2 Limitations

While InteliTalk offers significant benefits and enhancements to university communication, it's essential to acknowledge potential limitations to ensure realistic expectations and continuous improvement.

Here are some potential limitations of InteliTalk:

Dependency on External APIs: InteliTalk relies on external APIs such as OpenAI and Langchain for natural language understanding and generation. Any downtime or changes to these APIs could affect the chatbot's functionality and reliability.

1. **Training and Integration Complexity**: Integrating and training the chatbot with external APIs like OpenAI and Langchain can be complex and time-consuming. The effectiveness of the chatbot's responses may depend on the quality and specificity of the training data.
2. **Scalability Challenges**: As the user base and data volume grow, maintaining scalability with the MERN stack can become challenging. Managing large volumes of user interactions and data processing may require additional optimization and resources.
3. **Limited Customization**: While the MERN stack provides flexibility, customization options may be limited when integrating external APIs like OpenAI and Langchain. Customizing the chatbot's responses or behavior beyond the capabilities of these APIs may require additional development effort.
4. **Privacy and Security Concerns**: Utilizing external APIs for natural language processing raises privacy and security concerns, especially when handling sensitive

user data. Ensuring compliance with data protection regulations and securing data transmission is crucial to maintain user trust.

5. **Cost Considerations**: Integrating and utilizing external APIs like OpenAI and Langchain may incur additional costs, especially for high-volume usage or premium features. Managing and optimizing costs while ensuring the chatbot's effectiveness is essential for long-term sustainability.
6. **Dependency on Internet Connectivity**: InteliTalk requires a stable internet connection to access external APIs like OpenAI and Langchain. Any disruptions in internet connectivity could impact the chatbot's functionality and availability.
7. **Learning Curve for Developers**: Developing and maintaining a chatbot system with the MERN stack along with external APIs like OpenAI and Langchain requires expertise in multiple technologies. The learning curve for developers may be steep, especially for those unfamiliar with these technologies.

Addressing these limitations requires careful planning, continuous monitoring, and adaptation to ensure InteliTalk remains effective, scalable, and reliable in meeting user needs within the university environment.

7.3 Future Works

- InteliTalk poised for innovation in university environments.
- Enhance understanding of natural language and personalize interactions.
- Expand knowledge base to include university-specific information like schedules and events.
- Collaborate with other campus systems to provide comprehensive support.
- Become a helpful guide for everything happening on campus.
- Empower administrators to add their own information for enhanced utility.
- Commitment to listening to feedback and implementing improvements.
- Aim to keep InteliTalk ahead of the curve in assisting students and staff at universities.

REFERENCES

[1] Sharma, V., Goyal, M. and Malik, D., 2017. An intelligent behaviour shown by chatbot system. International Journal of New Technology and Research, 3(4), p.263312. G. O. Young, "Synthetic structure of industrial plastics," in Plastics, 2nd ed., vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15-64.

[2] Gacek, C., Abd-Allah, A., Clark, B. and Boehm, B., 1995, April. On the definition of software system architecture. In Proceedings of the First International Workshop on Architectures for Software Systems (pp. 85-94). Seattle, Wa.G. Liu, K. Y. Lee, and H. F. Jordan, "TDM and TWDM de Bruijn networks and shufflenets for optical communications," IEEE Trans. Comp., vol. 46, pp. 695-701, June 1997.

[3] Kalske, M., 2017. Transforming monolithic architecture towards microservice architecture. University of Helsinki. Altun, “Understanding hypertext in the context of reading on the web: Language learners’ experience,” Current Issues in Education, vol. 6, no. 12, July, 2005. [Online]. Available: http://cie.ed.asu.edu/volume6/number12/. [Accessed Dec. 2, 2007].

[4] Bragge, M., 2013. Model-View-Controller architectural pattern and its evolution in graphical user interface frameworks.T. J. van Weert and R. K. Munro, Eds., Informatics and the Digital Society: Social, ethical and cognitive issues: IFIP TC3/WG3.1&3.2 Open Conference on Social, Ethical and Cognitive Issues of Informatics and ICT, July 22-26, 2002, Dortmund, Germany. Boston: Kluwer Academic, 2003.

[5] Jangra, A., Singh, G., Singh, J. and Verma, R., 2011. Exploring testing strategies. International Journal of Information Technology and Knowledge Management, 4, pp.297-299K. E. Elliott and C.M. Greene, "A local adaptive protocol," Argonne National Laboratory, Argonne, France, Tech. Rep. 916-1010-BB, 1997.

[6] Sharma, V. and Baliyan, P., 2011. Maintainability analysis of component based systems. International Journal of Software Engineering and Its Applications, 5(3), pp.107-118.IEEE Criteria for Class IE Electric Systems, IEEE Standard 308, 1969.