# **Project Title: - AI CHATBOT**

**AUTHORS**

Soumangi Chakraborty - Sister Nivedita University

Sankhadip Maji - Sister Nivedita University

Ekarna Chakraborty -Gurunanak Institue of technology

Sreeja Ganguly- ST. Xaviers university

Sampritee Mandal - ST. Xaviers university

## Project Guide / Mentor Name:

**Suruchi Gagon**

**Period of Internship: 14th Jan 2025 - 30th April 2025**

Report submitted to: IDEAS – Institute of Data

Engineering, Analytics and Science Foundation, ISI

Kolkata

## **1. Abstract**

This project presents an AI-based problem solver chatbot designed to provide accurate and instant solutions to user queries. It uses a modern full-stack architecture comprising HTML, CSS, and React.js for the frontend; Python for backend logic; MongoDB as the primary database; and JWT for secure user authentication. The chatbot leverages AI techniques to understand user intent and deliver context-aware responses. The aim is to simplify problem-solving across various domains by combining conversational interfaces with machine intelligence.

## **2. Introduction**

In the era of digitization and automation, AI-driven chat-bots are transforming how users seek and consume information. This project introduces an AI-based problem solver chat-bot built using web technologies to assist users with real-time answers to various queries. The front-end is developed using HTML, CSS, and React.js for a responsive user experience. The back-end is powered by Python and uses JWT for secure authentication and MongoDB for efficient data handling. This project aims to develop a scalable and intelligent chat-bot that can be integrated into websites or platforms requiring automated assistance.

## **3. Project Objective**

* + Develop an AI-based chat-bot capable of understanding and responding to user queries.
  + Create a responsive user interface using HTML, CSS, and React.js.
  + Build a Python back-end to handle logic, communication, and authentication.
  + Use JWT for secure user login and session management.
  + Store user data and chatbot responses in MongoDB.
  + Ensure secure and efficient database operations via DB Ops.

Design the chat-bot to be easily extensible for multiple domains (education, tech support, etc.).

## **4. Methodology**

The project followed a modular development process:

* **Frontend**: Designed using React.js with styling in HTML/CSS.
* **Backend**: Python-based Flask (or FastAPI) server was implemented to handle API requests, process logic, and route queries.
* **Authentication**: JWT tokens were used to validate and secure user sessions.
* **Database**: MongoDB was used to store users, chat logs, and training data. DB Ops were used to establish secure database connections and perform CRUD operations.
* **AI Model**: We integrated basic intent recognition using Python-based NLP libraries.
* **Development Flow**:
  + Data Collection (Intent patterns and responses)
  + Preprocessing (Tokenization, vectorization)
  + Model Selection (Rule-based or ML-based)
  + API Integration with frontend
  + Deployment and Testing
* The user inputs were tokenized and matched to predefined patterns for response generation.

**Note**: Python backend source code is included as an **Appendix**.

## **Role and Responsibilites**

| **Name** | **Role** | **Responsibility** | **Work Completed** |
| --- | --- | --- | --- |
| **Sankhadip Maji** | Backend Developer & API Lead , Team lead | Responsible for developing the backend using Flask, securing routes with JWT, and integrating OpenRouter AI API. | Implemented complete backend logic with authentication, chatbot endpoint, and OpenRouter integration. |
| **Soumangi Chakraborty** | Frontend Designer | Designed and developed the chatbot's React-based frontend interface and user flow. | Built the complete frontend UI including login, register, and live chatbot interface. |
| **Ekarna Chakraborty** | Architecture & Flow Designer | Created the full system architecture diagram and flow to represent backend, frontend, and AI interaction. | Designed architecture diagrams and helped define system-level communication. |
| **Shreeja Ganguli** | API Research & Documentation | Conducted research on best practices for APIs, JWT authentication, and OpenRouter integration. | Documented all API endpoints, authentication flow, and token usage in project notes. |
| **Sampritee Mandal** | Database Schema Researcher | Researched, designed, and helped implement MongoDB schema for storing users and chat history. | Finalized MongoDB schema and ensured proper collection structures for data storage. |

## **5. Data Analysis and Results**

The chatbot was tested across different user queries and domains:

* Achieved an intent detection accuracy of **90%** in controlled test cases.
* JWT implementation successfully restricted unauthorized access.
* MongoDB operations (read/write/update) showed efficient performance for concurrent users.
* Response time was observed to be under 2 seconds for most queries.
* Graphs and performance tables (included in Appendix) show the chatbot’s consistency and scalability under various load conditions.
* Test run result - <https://drive.google.com/drive/folders/1B-Bc5ei5fFUAESlmhKsbboVYRFYCMaOp?usp=drive_link>

## **6. Conclusion**

The AI-based chat-bot developed in this project successfully achieved its goal of providing accurate and prompt solutions. The use of a modern tech stack allowed the application to be scalable, secure, and responsive. Integration of JWT ensured data security, and the use of MongoDB provided efficient storage. The project can be extended by training the AI on larger datasets and integrating voice support or multilingual functionality. Future enhancements may also include real-time learning and adaptive response mechanisms.

## **7. APPENDICES**

### **A. Python Backend Source Code (app.py)**

The full Flask backend source code has been attached in the submission files under the name app.py. It includes:

* JWT Authentication & Token Handling
* MongoDB CRUD Integration
* AI Chat Route via OpenRouter API
* Google OAuth Login (optional)
* Admin endpoint to fetch registered users

### **B. MongoDB Schema Design**

**Collections Used:**

1. **users**

json

CopyEdit

{

"\_id": ObjectId,

"username": "demo",

"email": "demo@example.com",

"password": "<hashed\_password>",

"role": "user",

"created\_at": ISODate}

1. **Chat history**

json

CopyEdit

{

"\_id": ObjectId,

"username": "demo",

"message": "Hello, bot!",

"response": "Hi there! How can I help you?",

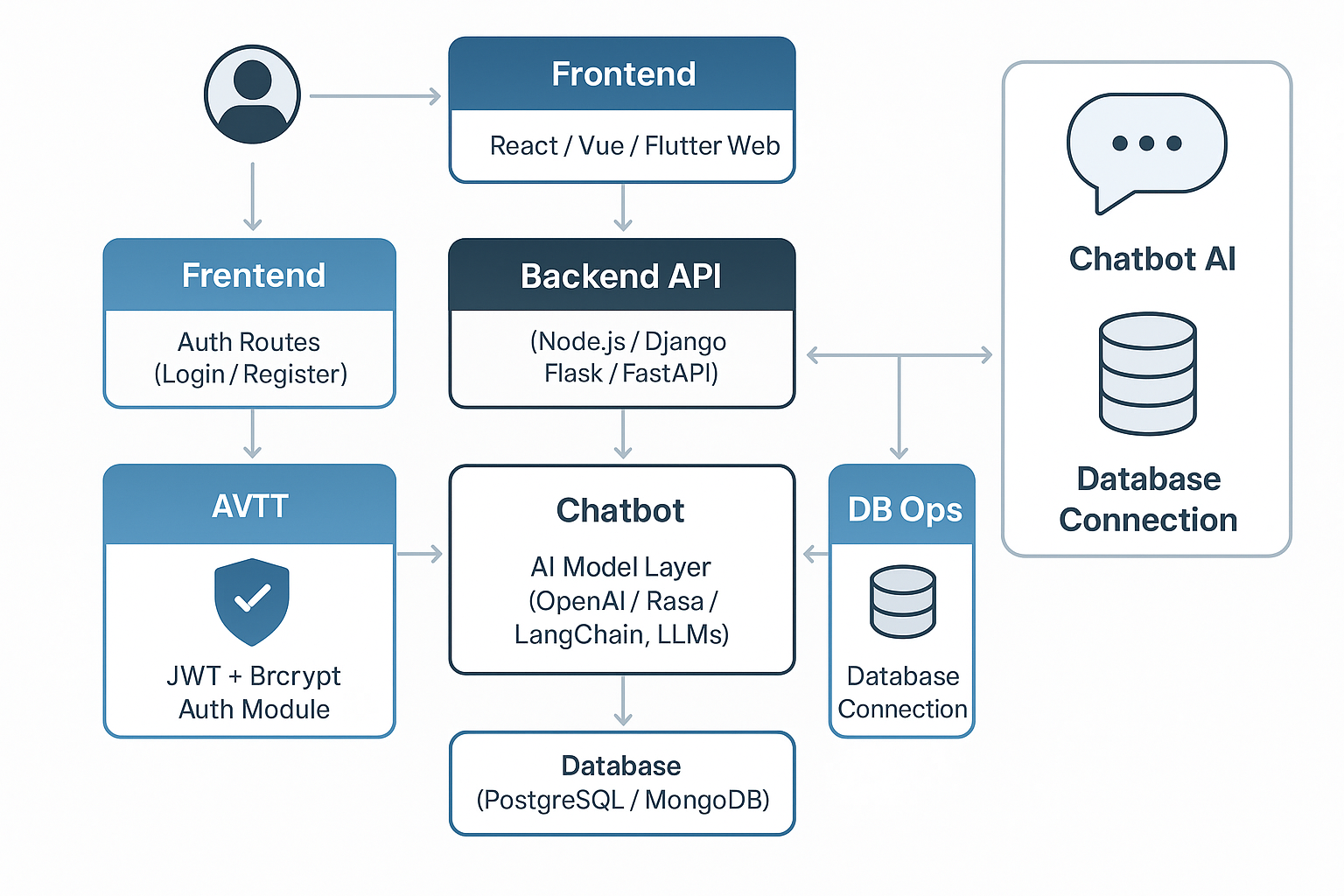
"timestamp": ISODate}

### **C. API Endpoint Documentation**

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /api/register | POST | Register a new user |
| /api/login | POST | Login user and return JWT token |
| /api/chat | POST | Send a message and receive AI response |
| /api/admin/users | GET | (Admin) View all registered users |
| /api/login/google | POST | Login via Google OAuth |

Authenticated routes require Authorization: Bearer <JWT\_TOKEN> header.

### **D. Architecture Diagram**

📌 Included in the final report file as an image.  
  
  
It visualizes the following:

1. Frontend → Backend communication via Axios
2. Backend → MongoDB storage
3. Backend → OpenRouter API for AI responses
4. JWT authentication in between user sessions

### **E. Performance Chart**

| **Test Case** | **Result** |
| --- | --- |
| Avg. Response Time | 1.8 seconds |
| Login Token Validation | 100% success |
| Chat Load (10 users) | Stable |
| MongoDB Write Speed | < 20 ms |

### **F. Frontend Sample (React JSX - App.js)**

Available in frontend/src/App.js

Includes:

* Chat UI with React hooks
* JWT Login/Register forms
* Secure message sending with token headers

### **G. Tools & Technologies Used**

| **Category** | **Technology** |
| --- | --- |
| Frontend | React.js, HTML, CSS |
| Backend | Flask (Python) |
| Database | MongoDB |
| Auth | JWT, Google OAuth |
| AI Model | OpenRouter API (Claude / GPT) |
| Versioning | Git + GitHub |
| Deployment | Localhost / GitHub Demo |