International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Real Time Chat Application Aws Websocket API

[1]Karthikeyan S, [2]Sathish Kumar M

[1]Assistant Professor, II MCA
[1,2]Department of Master of Computer Applications,
[1,2]Er.Perumal Manimekalai College of Engineering, Hosur,
[1]skkeyan@gmail.com, [2]madevsathishkumar6502@gmail.com

Abstract:
This project presents a real-time, AI-powered chatbot application that integrates Flask, Socket.IO, and Open Router's AI models to offer intelligent conversational support across multiple domains. The system is designed with a modular architecture that allows users to engage with a virtual assistant specializing in gadget troubleshooting, healthcare guidance, or sentiment analysis. Utilizing WebSocket technology, the application achieves seamless bi-directional communication between the client and server, enabling an instant and dynamic user experience.The frontend is developed using HTML5, CSS3, and vanilla JavaScript, enriched with interactive features such as live typing animations and optional image upload. On the backend, Flask is used in conjunction with Flask-SocketIO to manage real-time events, while the OpenRouter API provides context-aware AI responses through the DeepSeek language model. The chatbot is highly customizable and serves as a scalable prototype for intelligent customer support and personal assistant systems.
This report explores the design, implementation, and testing of the application, with a focus on leveraging WebSocket APIs and AI-powered chat interfaces. It also discusses the advantages of using cloud-based services like AWS for scalability and performance in real-world deployment scenarios.

Keywords: Authentication Security, One Time Password, Visual Cryptography.

1. INTRODUCTION

The rapid advancement of digital communication and AI technologies has increased the demand for intelligent chat systems. Traditional customer service and support tools, such as phone-based helpdesks and static web forms, are becoming inefficient and slow, requiring human intervention for most queries. These systems fail to provide real-time, personalized responses to users and lack scalability to handle large volumes of queries simultaneously. There is a need for an efficient, scalable, and reliable chat application that can respond to diverse user queries in real time, providing a consistent experience across various platforms and domains.

This project aims to address these issues by developing an AI-powered chat application that supports real-time communication and intelligent conversation across multiple domains such as gadget troubleshooting, healthcare advice, and sentiment analysis.

1. To develop a real-time chat application capable of engaging users in meaningful conversations using AI models.

2. To implement a scalable architecture utilizing WebSocket technology for continuous communication between the client and server.

3. To integrate multiple specialized AI models (healthcare, gadget support, sentiment analysis) for domain-specific responses.

4. To use cloud-based infrastructure, leveraging AWS WebSocket APIs for robust and scalable backend services.

5. To provide a user-friendly web interface that supports seamless chat interactions and optional image uploads.

2. PROPOSED WORK

The proposed system replaces HTTP polling with WebSocket-based communication, enabling persistent bi-directional connections between client and server. Key features:

- **WebSocket Communication**: Low-latency, full-duplex communication channels for instant message transfer.
- **AI Integration**: Uses OpenRouter's DeepSeek model to generate context-aware replies in real time.
- **Multi-Role Assistants**: Domain-specific system prompts for gadget support, healthcare advice, and sentiment analysis.
- **Cloud Scalability**: Deployed on AWS WebSocket API Gateway and EC2 (or equivalent) for handling large user volumes.

3. METHODS

User Interface:

- A web-based chat interface for users to interact with the system.
- Text input, image upload functionality, and dropdown menu to select the domain (e.g., gadget support, healthcare, sentiment analysis).

Real-Time Communication:

- Use WebSocket technology for continuous, low-latency communication between the client and server.
- Chat history should be dynamically updated with both user and bot messages.

AI Integration:

- The system should integrate with an AI model (DeepSeek from OpenRouter) that generates responses based on the user's input and selected role.

- AI should support multiple domains such as gadget troubleshooting, healthcare advice, and sentiment analysis.

Image Upload:

- Allow users to upload images (e.g., screenshots or product images) alongside text queries.
- Images should be displayed in the chat interface.

User Authentication (Optional):

- Allow users to register and log in (if needed) for personalized sessions and storing chat history.

Error Handling:

- Provide clear messages when errors occur (e.g., in AI response, network issues).

- **Additional Roles**: Adding more specialized roles, such as tech support, legal advice, or customer service, would broaden the application's use cases.
- **User Authentication**: Implementing user authentication and saving chat histories for personalized experiences and tracking user interactions.
- **Natural Language Processing Improvements**: Enhance the AI model's ability to handle complex conversations and context switching, making the bot more versatile.
- **Mobile App**: Extend the chat interface to mobile platforms, allowing users to chat from their smartphones or tablets.
- **Performance Optimization**: Further fine-tune the backend, especially WebSocket handling, to improve performance for high-traffic environments.

- **Access the Chat Interface**
  - Open a web browser and navigate to http://localhost:5000.
  - You will see the chat window with an input box and a role selector.
- **Using the Chatbot**
  - **Select Role**: Choose between Gadget Support, Healthcare Assistant, or Sentiment Analysis.
  - **Enter Message**: Type your question or issue in the text box.
  - **Upload Image (Optional)**: Click the image icon to attach a photo.
  - **Send**: Press the Send button or hit Enter.
- **Viewing Responses**
  - The bot's response appears with a typing animation.
  - Uploaded images display inline above the bot's reply.

• **Error Messages**

○ If the AI service is unavailable, you'll see:

"Sorry, something went wrong. Please try again later."

4. FUTURE ENHANCEMENTS

• **Additional Roles**: Adding more specialized roles, such as tech support, legal advice, or customer service, would broaden the application's use cases.

• **User Authentication**: Implementing user authentication and saving chat histories for personalized experiences and tracking user interactions.

• **Natural Language Processing Improvements**: Enhance the AI model's ability to handle complex conversations and context switching, making the bot more versatile.

• **Mobile App**: Extend the chat interface to mobile platforms, allowing users to chat from their smartphones or tablets.

• **Performance Optimization**: Further fine-tune the backend, especially WebSocket handling, to improve performance for high-traffic environments.

5. LIMITATIONS

• **Dependency on OpenRouter API**: The bot's responses are limited by the capabilities and availability of the OpenRouter AI API. Any downtime or changes in the API could impact user experience.

• **Limited Role Scope**: Currently, the bot can only respond in three predefined roles. The system's flexibility for other roles or customization is restricted.

• **Error Handling**: While basic error handling has been implemented, more extensive exception management, particularly around user inputs, can be added to improve robustness.

Conflict of Interest: The authors have verified that this study lacks any conflicting interests.

Funding Disclosure: There was no financial support or third-party funding used for performing this research project.

Ethical Approval: The researchers performed this study devoid of experiments which included human or animal participants.

Consent Disclosure: The study omitted patient-related data so consent procedures were not required.

6. ACKNOWLEDGMENT

The authors declare that they have no reports of acknowledgments for this

REFERENCES

1. Flask Documentation. “Flask Web Development.” https://flask.palletsprojects.com/
2. Flask-SocketIO Documentation. “Flask-SocketIO.” https://flask-socketio.readthedocs.io/
3. Socket.IO. “Socket.IO – Real-time Engine.” https://socket.io/
4. OpenRouter AI. “DeepSeek Chat API.” https://openrouter.ai/
5. AWS WebSocket API. “Amazon API Gateway WebSocket APIs.” https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-websocket-api.html
6. Mozilla Developer Network. “Using the FileReader API.” https://developer.mozilla.org/en-US/docs/Web/API/FileReader
7. OWASP Foundation. “OWASP Testing Guide.” https://owasp.org/www-project-web-security-testing-guide/