WhatsApp Chatbot Full-Stack Application Documentation

Table of Contents

---

1. Project Overview

This project is a **full-stack WhatsApp chatbot** that integrates with the **Meta WhatsApp API** to send and receive messages automatically based on user input. All chat messages are stored in **MongoDB** for persistence and can be viewed using a **React frontend**. The user interface allows easy access to all stored chats and organizes messages efficiently.

This chatbot is designed to assist users in interacting with an automated response system, allowing them to select different options for various categories, such as admissions, contact details, and more.

---

2. Features

1. **WhatsApp Chatbot Integration**: The chatbot interacts with users through the Meta WhatsApp API, responding to their queries and guiding them through predefined options.

2. **Persistent Chat Storage**: Every interaction between users and the chatbot is stored in a MongoDB database for future retrieval and analysis.

3. **Frontend Display**: The stored chats are displayed on a React frontend, which allows users to view and analyze conversations between the bot and the users.
4. **Session Management**: Each user session is managed with a timeout feature. If a user is inactive for 2 hours, their session is reset, and they are greeted with a fresh start when they resume.
5. **Interactive Buttons**: The chatbot offers options through interactive buttons that allow users to easily navigate the chatbot's offerings.

---

3. Tech Stack

Frontend:

- **React**: JavaScript library for building user interfaces.
- **Tailwind CSS**: Utility-first CSS framework for styling.

Backend:

- **Node.js & Express**: Backend framework for building the API and handling requests.
- **MongoDB & Mongoose**: NoSQL database and its ORM for storing chat messages.
- **Axios**: Used for making HTTP requests, especially for WhatsApp API.
- **WhatsApp Meta API**: To interact with WhatsApp messages programmatically.

---

4. Backend Setup

The backend is powered by Node.js and Express, and it interacts with MongoDB to store and retrieve chat data. The WhatsApp Meta API is used for sending and receiving messages. The backend is also responsible for managing sessions and handling webhook events from the WhatsApp API.

Backend File Structure

```
backend/
├── controllers/
│   ├── messageController.js   // Handles incoming messages and processes responses
│   ├── sessionManager.js      // Manages session timeouts and expiration
│   └── messageFormatter.js    // Formats interactive messages and buttons
├── models/
│   └── chatMessage.js         // Mongoose schema for chat messages
├── routes/
│   └── chatRoutes.js          // API routes for fetching stored chats
├── .env                       // Environment variables for backend
├── index.js                   // Main Express server setup
├── messageSender.js           // Handles sending messages via WhatsApp API
└── package.json               // Backend dependencies
```

Setting Up MongoDB

MongoDB is used to persist the chat messages exchanged with the WhatsApp bot. You can either set up a local instance of MongoDB or use a cloud-based MongoDB service such as **MongoDB Atlas**.

1. **Install MongoDB**:
   - Installation Guide for MongoDB.
2. **Add MongoDB URI to Environment Variables**: Add the following line to your .env file:

   ```
   MONGODB_URI=mongodb://localhost:27017/whatsapp_bot
   ```

3. **Connect to MongoDB in index.js**: In the main server file, ensure MongoDB is connected when the server starts.

   ```
   mongoose.connect(process.env.MONGODB_URI, {
     useNewUrlParser: true,
     useUnifiedTopology: true,
   })
   .then(() => console.log('MongoDB connected'))
   .catch(err => console.error('MongoDB connection error:', err));
   ```

API Endpoints

1. GET /api/chats

Fetch all chat messages stored in the database.

- **Request**:
  No request parameters.
- **Response**:

  ```
  [
    {
      "_id": "614d1c8897b7a920fcd4d0e6",
      "from": "918765432100",
      "to": "WhatsAppBot",
      "message": "Hello",
      "timestamp": "2022-01-12T07:35:00Z"
    },
    {
      "_id": "614d1c8897b7a920fcd4d0e7",
      "from": "918765432100",
      "to": "WhatsAppBot",
      "message": "Contact Details",
      "timestamp": "2022-01-12T07:36:00Z"
    }
  ]
  ```

2. POST /webhook
Webhook that listens to incoming WhatsApp messages and processes them based on the user's input.

- **Request**:
  Example of incoming data from WhatsApp:

  ```
  {
    "from": "918765432100",
    "to": "WhatsAppBot",
    "text": {
      "body": "Hello"
    }
  }
  ```

- **Response**:
  A 200 status code is returned if the message is successfully processed.

---

5. Frontend Setup

The frontend is built using **React** for dynamic rendering and **Tailwind CSS** for styling. It fetches chat data from the backend API and displays it in a clean, easy-to-read format.

Frontend File Structure

```
frontend/
├── src/
│   ├── components/
│   │   ├── ChatDisplay.js     // Component to display all chat messages
│   │   ├── ChatItem.js        // Component for individual chat message
│   ├── App.js                 // Main React component
│   └── index.js               // Entry point for React application
├── .env                       // Environment variables for frontend (optional)
├── tailwind.config.js         // Tailwind CSS configuration
├── index.css                  // Tailwind CSS imports
└── package.json               // Frontend dependencies
```

React Components Overview

1. **ChatDisplay Component (ChatDisplay.js)**: Fetches and displays all the chat messages from the backend. This is the main display component for all chat conversations.

2. **ChatItem Component (ChatItem.js)**: Handles the display of individual chat messages within the chat list. It provides a clear separation of concerns and allows customization of how each message appears.

```
const ChatItem = ({ chat }) => {
  return (
    <div className="mb-4 border-b pb-2 hover:bg-gray-100 transition duration-200 ease-in-
out">
```

```
      <p className="font-semibold text-gray-800">{chat.from} to {chat.to}</p>
      <p className="text-gray-700">{chat.message}</p>
      <p className="text-xs text-gray-500">{new Date(chat.timestamp).toLocaleString()}</p>
    </div>
  );
};
```

Fetching Chat Messages

The **ChatDisplay** component fetches messages from the backend's /api/chats endpoint and displays them in a formatted list.

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import ChatItem from './ChatItem';

const ChatDisplay = () => {
  const [chats, setChats] = useState([]);

  useEffect(() => {
    const fetchChats = async () => {
      try {
        const response = await axios.get('http://localhost:5000/api/chats');
        setChats(response.data);
      } catch (err) {
        console.error('Failed to fetch chats:', err);
      }
    };

    fetchChats();
  }, []);

  return (
    <div className="container mx-auto p-6">
      <h1 className="text-3xl font-bold mb-4 text-center">WhatsApp Chat Messages</h1>
      <div className="bg-white rounded-lg shadow-lg p-4 max-h-80vh overflow-y-scroll">
        {chats.length > 0 ? (
          chats.map((chat) => (
            <ChatItem key={chat._id} chat={chat} />
          ))
        ) : (
          <p className="text-center text-gray-500">No chats found.</p>
        )}
      </div>
    </div>
  );


};

export default ChatDisplay;
```

6. Session Management

The chatbot uses session management to keep track of users' interactions. Each session lasts for **2 hours**, after which it resets if the user is inactive. If a session expires, the next interaction will start with a fresh greeting and main menu.

- **sessionManager.js** handles the session logic:

```
const userSessions = {};

// Check if the session has expired
export const isSessionExpired = (from) => {
  const currentTime = Date.now();
  if (userSessions[from]) {
    const lastInteractionTime = userSessions[from].lastInteraction;
    const timeDiff = (currentTime - lastInteractionTime) / (1000 * 60 * 60);  // Convert ms to hours
    return timeDiff >= 2;
  }
  return true;
};

// Update session on user interaction
export const updateSession = (from) => {
  userSessions[from] = { lastInteraction: Date.now() };
};

// Reset session after expiration
export const resetSession = (from) => {
  delete userSessions[from];
};
```

7. How to Run the Application

Step 1: Clone the Repository

```
git clone <repo-url>
cd whatsapp-chat-app
```

Step 2: Backend Setup

1. Install backend dependencies:

```
cd backend
npm install
```

2. Create a .env file in the backend/ directory:

```
MONGODB_URI=mongodb://localhost:27017/whatsapp_bot
VERIFY_TOKEN=your_verify_token
WHATSAPP_PHONE_NUMBER_ID=your_phone_number_id
```

```
WHATSAPP_ACCESS_TOKEN=your_access_token
PORT=5000
```

3. Start the backend server:

```
npm start
```

Step 3: Frontend Setup

1. Navigate to the `frontend` directory and install dependencies:

```
cd frontend
npm install
```

2. Start the React development server:

```
npm start
```

---

8. Environment Variables

To ensure security and flexibility, the application uses environment variables to store sensitive information. Below is the list of required environment variables.

Backend `.env` File

```
MONGODB_URI=mongodb://localhost:27017/whatsapp_bot
VERIFY_TOKEN=your_verify_token
WHATSAPP_PHONE_NUMBER_ID=your_phone_number_id
WHATSAPP_ACCESS_TOKEN=your_access_token
PORT=5000
```

---

9. WhatsApp Meta API Integration

The backend uses the **Meta WhatsApp API** to send and receive messages. When a message is received via the API, it is processed by the backend, and a response is sent back based on the user's input.

- **Incoming Webhook**: The WhatsApp API sends incoming messages to the `/webhook` endpoint on the backend. These messages are then stored in MongoDB, and appropriate responses are sent.
- **Message Sending**: The backend uses **Axios** to send messages to users through the WhatsApp API.

---

10. Extending the Project

This project provides a strong foundation for building a chatbot with WhatsApp API integration. Here are some potential ways to extend the project:

1. **Advanced Chat Analytics**: Add features that analyze the stored chat messages for patterns or frequent queries.
2. **Sending Messages from Frontend**: Extend the frontend to allow sending messages directly from the UI.
3. **Multilingual Support**: Implement translation features to handle conversations in multiple languages.
4. **Rich Media Support**: Integrate the capability to send images, documents, and other media types through the WhatsApp API.

This concludes the documentation for the WhatsApp chatbot project. With this setup, you have a working chatbot that interacts with users, stores conversations, and displays them through a user-friendly interface.