API Reference: WhatsApp Management - Clevio Pro

Overview

This document provides a comprehensive API reference for the WhatsApp Multi-Session Dashboard, developed by the Clevio AI Team. This project is a modern solution for managing multiple WhatsApp Web sessions, offering robust features for businesses and developers. It is built using a technology stack including React for the frontend, Express for the backend API, and PostgreSQL for data storage, leveraging `whatsapp-web.js` for WhatsApp Web integration.

Key Features:

- **Multi-session WhatsApp Web**: Supports multiple independent WhatsApp Web sessions, each with its own login and multi-device capabilities.
- **QR Code Scanning**: Facilitates easy session initialization through QR code scanning.
- **Automated Message Forwarding**: Automatically forwards incoming messages to a per-session webhook (e.g., n8n) for custom processing.
- **Webhook Reply Functionality**: Webhooks can reply to messages with various content types, including text, media, and documents.
- **Live Dashboard**: Provides a live display of session status and message logs.
- **Secure Authentication**: Implements JWT-based authentication for secure access to the dashboard and API.
- **Responsive UI**: Features a blue and white, elegant, and responsive user interface optimized for desktop.

This document will detail the backend API endpoints, authentication mechanisms, data models, and provide examples for effective integration and usage.

Table of Contents

---

1. Authentication

All API endpoints, except for `/login`, are protected and require a JSON Web Token (JWT) for authentication. The JWT must be included in the `Authorization` header of each request as a Bearer token.

`POST /login`

Authenticates a user and returns a JWT token. This token should be used for subsequent authenticated requests.

Request Body:

```
{
  "username": "your_username",
  "password": "your_password"
}
```

Response:

```
{
  "token": "your_jwt_bearer_token"
}
```

Example (using `curl`):

```
curl -X POST -H "Content-Type: application/json" -d '{"username": "admin",
"password": "yourpassword"}' http://localhost:3001/login
```

---

2. Session Management

These endpoints allow for the creation, retrieval, updating, and deletion of WhatsApp sessions, as well as managing their lifecycle and status.

`GET /sessions`

Retrieves a list of all active WhatsApp sessions.

Authentication: Required (JWT Bearer Token)

Response:

```
[
  {
    "id": 1,
    "session_id": "628123456789",
    "webhook_url": "https://your-webhook-url.com/session1",
    "created_at": "2023-01-01T12:00:00.000Z",
    "updated_at": "2023-01-01T12:05:00.000Z"
  },
  // ... more sessions
]
```

Example:

```
curl -X GET -H "Authorization: Bearer your_jwt_token"
http://localhost:3001/sessions
```

`POST /sessions`

Creates a new WhatsApp session or updates an existing one. If a `sessionId` already exists, its `webhookUrl` will be updated.

Authentication: Required (JWT Bearer Token)

Request Body:

```
{
  "sessionId": "628123456789",
  "webhookUrl": "https://your-new-webhook-url.com/session1"
}
```

Response:

```
{
  "message": "Session created/updated successfully",
  "session": {
    "id": 1,
    "session_id": "628123456789",
    "webhook_url": "https://your-new-webhook-url.com/session1",
    "created_at": "2023-01-01T12:00:00.000Z",
    "updated_at": "2023-01-01T12:10:00.000Z"
  }
}
```

Example:

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Bearer your_jwt_token" -d '{"sessionId": "628123456789", "webhookUrl": "https://your-new-webhook-url.com/session1"}' http://localhost:3001/sessions
```

`PUT /sessions/:sessionId/webhook`

Updates the webhook URL for a specific session.

Authentication: Required (JWT Bearer Token)

Path Parameters: - `sessionId` : The ID of the session to update (e.g., `628123456789`).

Request Body:

```
{
  "webhookUrl": "https://another-webhook-url.com/session1"
}
```

Response:

```
{
  "message": "Webhook URL updated successfully",
  "session": {
    "id": 1,
    "session_id": "628123456789",
    "webhook_url": "https://another-webhook-url.com/session1",
    "created_at": "2023-01-01T12:00:00.000Z",
    "updated_at": "2023-01-01T12:15:00.000Z"
  }
}
```

Example:

```
curl -X PUT -H "Content-Type: application/json" -H "Authorization: Bearer
your_jwt_token" -d '{"webhookUrl": "https://another-webhook-url.com/session1"}'
http://localhost:3001/sessions/628123456789/webhook
```

`DELETE /sessions/:sessionId`

Removes a specific WhatsApp session.

Authentication: Required (JWT Bearer Token)

Path Parameters: - `sessionId` : The ID of the session to remove.

Response:

```
{
  "message": "Session removed successfully"
}
```

Example:

```
curl -X DELETE -H "Authorization: Bearer your_jwt_token"
http://localhost:3001/sessions/628123456789
```

`POST /sessions/:sessionId/init`

Initializes a WhatsApp session. This will trigger the QR code generation or login process if the session is not already connected.

Authentication: Required (JWT Bearer Token)

Path Parameters: - `sessionId` : The ID of the session to initialize.

Response:

```
{
  "message": "Session initialization triggered"
}
```

Example:

```
curl -X POST -H "Authorization: Bearer your_jwt_token"
http://localhost:3001/sessions/628123456789/init
```

`GET /sessions/:sessionId/qr`

Retrieves the QR code for a session if it is not yet connected. This endpoint is typically polled by the frontend to display the QR code for user scanning.

Authentication: Required (JWT Bearer Token)

Path Parameters: - `sessionId`: The ID of the session to get the QR code for.

Response:

```
{
  "qrCode": "data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAA...", // Base64
encoded QR code image
  "status": "initializing"
}
```

Example:

```
curl -X GET -H "Authorization: Bearer your_jwt_token"
http://localhost:3001/sessions/628123456789/qr
```

`GET /sessions/:sessionId/status`

Retrieves the current status and information of a WhatsApp session.

Authentication: Required (JWT Bearer Token)

Path Parameters: - `sessionId`: The ID of the session to get the status for.

Response:

```
{
  "status": "connected", // connected, initializing, disconnected
  "info": {
    "name": "WhatsApp User Name",
    "number": "628123456789"
  }
}
```

Example:

```
curl -X GET -H "Authorization: Bearer your_jwt_token"
http://localhost:3001/sessions/628123456789/status
```

`GET /sessions/:sessionId/logs`

Retrieves in-memory message logs for a specific session. These logs are live and provide a real-time view of message activity.

Authentication: Required (JWT Bearer Token)

Path Parameters: - `sessionId`: The ID of the session to get logs for.

Response:

```
[
  {
    "type": "incoming",
    "from": "628xxxx@c.us",
    "to": "628xxxx@c.us",
    "body": "Hello",
    "timestamp": 1753078922
  },
  {
    "type": "outgoing",
    "to": "628xxxx@c.us",
    "body": "Reply message",
    "timestamp": 1753078950
  }
  // ... more logs
]
```

Example:

```
curl -X GET -H "Authorization: Bearer your_jwt_token"
http://localhost:3001/sessions/628123456789/logs
```

---

3. Webhook System

The system automatically POSTs incoming WhatsApp messages to the `webhookUrl` configured for each session in the database. This allows for custom processing and automated responses.

Incoming Webhook Payload Format

When a new message is received by a WhatsApp session, the system will send an HTTP POST request to the configured `webhookUrl` with the following JSON payload:

```
{
  "sessionId": "628xxxx",  // Sender's number (without @c.us)
  "from": "628xxxx@c.us",
  "to": "628xxxx@c.us",
  "body": "Hello",
  "type": "chat", // chat, image, document, audio, etc.
  "timestamp": 1753078922,
  "group": false,
  "media": "...base64...",   // if media is present
  "filename": "photo.jpg",   // if media is present
  "mimetype": "image/jpeg",  // if media is present
  "caption": "..."           // if media is present
}
```

Webhook Response for Replying

Your webhook can reply to the incoming message by returning a specific JSON format in its HTTP response. The system will then send the message back to the original sender via WhatsApp.

Reply with Plain Text: To reply with a simple text message, your webhook should return a plain text string in the response body.

Example (Plain Text Response):

```
This is a plain text reply from your webhook.
```

Reply with JSON (Text or Media): For more complex replies, including media, your webhook should return a JSON object. The JSON structure varies based on whether you are sending a text message or a media message.

Example (JSON Text Response):

```
{
  "text": "Balasan text..."
}
```

Example (JSON Media Response):

```
{
  "media": "...base64...", // Base64 encoded media content
  "mimetype": "image/png",
  "caption": "Keterangan foto"
}
```

---

4. Database Schema

The backend uses PostgreSQL to store session and user information. Below are the SQL schemas for the primary tables.

`wa_sessions` Table

This table stores information about each WhatsApp session.

```sql
CREATE TABLE wa_sessions (
    id serial PRIMARY KEY,
    session_id varchar(100) UNIQUE NOT NULL,
    webhook_url text,
    created_at timestamp DEFAULT NOW(),
    updated_at timestamp DEFAULT NOW()
);
```

`users` Table (Optional)

This table is for user authentication for the admin dashboard. The `password` field should store a bcrypt hash of the user's password.

```sql
CREATE TABLE users (
    id serial PRIMARY KEY,
    username varchar(40) UNIQUE NOT NULL,
    password varchar(200) NOT NULL, -- bcrypt hash
    created_at timestamp DEFAULT NOW()
);
```

---

5. Setup and Deployment Guide

This section outlines the steps to set up and run the WhatsApp Multi-Session Dashboard project.

Prerequisites

- Node.js (LTS version recommended)
- PostgreSQL database server
- `npm` (Node Package Manager)

Installation Steps

1. **Clone the Project:** `bash git clone <repository_url> cd <project_directory>`
2. **Database Setup:** Create a PostgreSQL database and the necessary tables (`wa_sessions` and `users`) as defined in the [Database Schema](#) section.
3. **Environment Configuration:** Create a `.env` file in the `/backend` directory and configure the environment variables. An example `.env` file is provided below:

   `PORT=3001 PGHOST=localhost PGUSER=postgres PGPASSWORD=yourpassword PGDATABASE=whatsapp_dashboard PGPORT=5432 JWT_SECRET=your_jwt_secret` * `PORT`: The port on which the Express API server will run. * `PGHOST`, `PGUSER`, `PGPASSWORD`, `PGDATABASE`, `PGPORT`: PostgreSQL database connection details. * `JWT_SECRET`: A strong, secret key used for signing JWT tokens. **Ensure this is kept confidential and is sufficiently complex.**
4. **Install Dependencies:** Navigate to the `backend` and `frontend` directories and install their respective dependencies:
   - **Backend:** `bash cd backend npm install`
   - **Frontend:** `bash cd ../frontend npm install`

Running the Application

1. **Start the Backend Server:** From the `backend` directory, run: `bash npm start` The backend API server will typically run on `http://localhost:3001` (or the port specified in your `.env` file).

2. **Start the Frontend Application:** From the `frontend` directory, run: `bash npm start` The React dashboard UI will typically run on `http://localhost:3000`.

3. **Login to the Dashboard:** Access the frontend application in your web browser (e.g., `http://localhost:3000`). Use the username and password configured in your `users` table in the PostgreSQL database to log in.

---

6. Security Considerations

- **Dashboard Access:** The dashboard should not be exposed to the public internet without strong authentication and proper security measures. Ensure robust, unique passwords are used for all user accounts.
- **JWT Secret:** The `JWT_SECRET` in your `.env` file must be kept confidential. Compromise of this secret could lead to unauthorized access.
- **API Protection:** All API endpoints (except `/login`) are protected by JWT Bearer token authentication. Ensure that client-side applications correctly send the `Authorization` header with the valid token.
- **Webhook Security:** If your webhooks are publicly accessible, ensure they are secured against unauthorized access. Consider implementing signature verification or IP whitelisting if sensitive data is being processed.
- **Database Security:** Secure your PostgreSQL database with strong credentials and restrict network access to only trusted sources.

---

7. Credits

This project leverages the following key technologies and contributions: - `whatsapp-web.js`: For WhatsApp Web integration. - React: Frontend JavaScript library. - Express: Backend web application framework for Node.js. - PostgreSQL: Relational database system. - n8n: (Optional) Workflow automation tool, often used for webhook processing.

Developed by: Clevio AI Team

Date: 21 July 2025 **Author:** Manus AI