Practical Test: Multi-Tenant WhatsApp Messaging Service

Overview

We know this is most complex test, here we want to evaluate how best you can utilise your knowledge, skill and AI tools to finish the core multi-device WhatsApp connection feature with message and chat history storage. You are tasked with building a multi-tenant WhatsApp messaging service using either the **Baileys TypeScript library** (https://baileys.wiki/docs/intro/) or the **WAHA WhatsApp HTTP API** (https://waha.devlike.pro/docs-overview/introduction/). The service should allow different tenants (e.g., organizations) to manage multiple users, assign users to user groups with distinct permissions, authenticate with WhatsApp, link multiple WhatsApp devices, and send messages to specified phone numbers or chat groups. The solution must be RESTful, documented with Swagger, and follow best practices for API security and scalability, reflecting the requirements of a Senior Backend Developer role. **Feel free to use AI tools - that's the key** to assist in development (e.g., GitHub Copilot, ChatGPT), but ensure you understand and explain your code in the submission video. **Feel free to use any readymade platform that covers side features like authentication, UI etc...** to focus on the core WhatsApp integration. Optionally, you may create a frontend in any language or framework you prefer (e.g., React, Vue, Angular, Svelte, or plain HTML/CSS/JS) to interact with the backend APIs.

You have the weekend to complete this task. Please submit your solution as a GitHub repository and include a **Loom video** (5-10 minutes) demonstrating the working application, explaining your code structure, and showcasing the key features, including the database schema, multi-device WhatsApp connections, message sending, chat history storage, and contact/group management.

Requirements

1. **Multi-Tenant Architecture**:
    - Implement a multi-tenant system where each tenant (organization) has isolated data and WhatsApp sessions.
    - Use a `tenantId` to segregate data in the database, ensuring one tenant's data (users, sessions, messages, contacts, groups) cannot be accessed by another.
    - Each tenant can have multiple users, organized into user groups (e.g., Admins, Editors, Viewers) with different permissions.
2. **User and Group Management**:

   - Support multiple users per tenant, each with a unique `userId`, `username`, and `password`.
   - Implement user groups with distinct permissions (e.g., Admins can manage users, Editors can send messages, Viewers can only view logs).
   - Example permissions:
     - Admins: Create/delete users, manage groups, link devices, send messages, view logs.
     - Editors: Link devices, send messages, view logs.
     - Viewers: View logs only.
   - Provide endpoints to manage users and groups (e.g., create user, assign user to group).

3. **WhatsApp Integration (Baileys or WAHA)**:

   - Choose either **Baileys** (`@whiskeysockets/baileys`) or **WAHA** (https://waha.devlike.pro/) for WhatsApp integration.
     - **Baileys**: Implement WebSocket-based connection, custom auth state (stored in database), and QR code generation for multi-device support.
     - **WAHA**: Use the WAHA Docker image (`devlikeapro/waha` for Core, or `devlikeapro/waha-plus` for advanced features) to simplify WhatsApp integration with RESTful APIs. Follow WAHA's Quick Start guide (https://waha.devlike.pro/docs-quick-start/) to set up and authenticate sessions.
   - Allow users to link multiple WhatsApp devices and switch between them on the platform.
   - Implement a feature to generate a QR code for WhatsApp authentication for each device.
   - Enable users to send text messages to a specified phone number or WhatsApp chat group from any linked device.
   - Store WhatsApp session data per user securely in the database (for Baileys, implement custom auth state; for WAHA, use built-in session storage with MongoDB/PostgreSQL).
   - Store all contacts and chat groups in the database, retrieved via Baileys' APIs (e.g., `sock.groupFetchAllParticipating`, `sock.fetchContacts`) or WAHA's APIs (e.g., `GET /api/{session}/contacts`, `GET /api/{session}/groups`).

4. **Authentication and Authorization**:

   - Implement JWT-based authentication for users to securely access the service (you may use a readymade platform like Auth0 or Firebase Authentication if preferred).
   - Each user should have credentials (e.g., `username`, `password`) stored in the database, scoped to their tenant, unless using a third-party auth service.
   - Provide an endpoint for users to log in and receive a JWT token containing `tenantId`, `userId`, and `groupId`.
   - Enforce group-based permissions using middleware to restrict access to endpoints based on the user's group.

5. **Database**:

- Use **PostgreSQL** or **MongoDB** to store:
  - Tenant information (e.g., `tenantId`, `name`).
  - User information (e.g., `userId`, `tenantId`, `username`, `password`, `groupId`) unless using a third-party auth service.
  - User group information (e.g., `groupId`, `tenantId`, `permissions`).
  - WhatsApp session data per user (e.g., `userId`, `deviceId`, session data).
  - Message logs (e.g., `tenantId`, `userId`, recipient phone number or group ID, message content, timestamp).
  - Contacts (e.g., `tenantId`, `userId`, contact phone number, contact name).
  - Chat groups (e.g., `tenantId`, `userId`, group ID, group name, participants).
- Design a **scalable and standardized database schema** with:
  - Tenant isolation (e.g., include `tenantId` in all tables/collections).
  - Efficient indexing for frequent queries (e.g., on `tenantId`, `userId`, `groupId`).
  - Normalized structure for relational databases (PostgreSQL) or optimized document structure for NoSQL (MongoDB).
  - Support for high read/write throughput (e.g., connection pooling, sharding if applicable).

6. **RESTful API**:

- Build RESTful APIs using **NestJS** (preferred for its TypeScript support and modularity).
- Endpoints (all scoped to the authenticated user's `tenantId`):
  - `POST /auth/login`: Authenticate a user and return a JWT token.
  - `POST /users`: Create a new user (Admin-only).
  - `PUT /users/:userId/group`: Assign a user to a group (Admin-only).
  - `GET /whatsapp/devices`: List linked WhatsApp devices for the authenticated user.
  - `POST /whatsapp/devices`: Generate a QR code for WhatsApp authentication for a new device.
  - `POST /whatsapp/send`: Send a WhatsApp message to a phone number or group from a specified linked device.
  - `GET /messages`: Retrieve message logs for the authenticated user's tenant.
  - `GET /contacts`: Retrieve stored contacts for the authenticated user.
  - `GET /groups`: Retrieve stored chat groups for the authenticated user.
- Secure all endpoints (except login) with JWT middleware and group-based permission checks.
- Document APIs using **Swagger** (https://docs.nestjs.com/openapi/introduction) with detailed request/response schemas.

7. **Optional Frontend**:

- You may create a frontend in any language or framework you prefer (e.g., React, Vue, Angular, Svelte, or plain HTML/CSS/JS), or use a readymade platform (e.g., Retool, Bubble) for UI components.
- The frontend should interact with the backend APIs to:
  - Log in as a user and display the JWT token.
  - Display a QR code for WhatsApp device authentication.
  - Allow users to select a linked WhatsApp device and send messages to contacts or groups.
  - Show message logs, contacts, and chat groups for the tenant.
- Ensure the frontend is responsive and user-friendly, with basic error handling (e.g., display API error messages).

8. **API Security Best Practices**:

   - Validate and sanitize all inputs to prevent injection attacks (e.g., use NestJS validation pipes with `class-validator`).
   - Implement rate-limiting to prevent abuse (e.g., using `@nestjs/throttler`).
   - Use secure password hashing (e.g., bcrypt) for user credentials if not using a third-party auth service.
   - Ensure JWT tokens have a short expiration time (e.g., 1 hour) and support refresh tokens.
   - Use HTTPS for all API communications (configure in Docker setup; WAHA supports HTTPS natively with environment variables).
   - Log sensitive operations (e.g., user creation, message sending) without exposing sensitive data.

9. **Scalability Best Practices**:

   - Design the application with a microservices-ready architecture (e.g., separate modules for auth, user management, WhatsApp integration).
   - Use connection pooling for database interactions.
   - Implement Redis for caching frequently accessed data (e.g., user sessions, tenant metadata, contacts).
   - Optimize database queries with proper indexing and avoid N+1 query issues.
   - Use asynchronous processing for WhatsApp message sending to handle high throughput.

10. **Deployment**:

    - Containerize the application (and WAHA, if used) using **Docker**.
    - Provide a `docker-compose.yml` file to run the application, database, Redis, and WAHA (if used) locally.
    - Ensure the application can be deployed with a single command (e.g., `docker-compose up`).

11. **Testing**:
    - Write unit tests for at least three critical components (e.g., authentication middleware, permission checks, message sending logic) using **Jest**.
    - Include instructions for running tests.
12. **Bonus (Optional)**:
    - Implement an event-driven system (e.g., emit events for message sending or user creation using NestJS event emitters, or use WAHA's webhook support).
    - Add support for WebSocket notifications (e.g., real-time updates when a message is sent) using NestJS WebSockets or WAHA's WebSocket events.
    - Integrate a logging framework (e.g., Winston) for structured logging.
    - Use WAHA Plus for advanced features like group APIs or media support.

Tech Stack

- **Backend**: NestJS, TypeScript
- **Database**: PostgreSQL or MongoDB
- **Caching**: Redis
- **WhatsApp API**: Baileys (`@whiskeysockets/baileys`) or WAHA (`devlikeapro/waha` or `devlikeapro/waha-plus`)
- **Authentication**: JWT (or third-party like Auth0, Firebase Authentication)
- **Containerization**: Docker
- **Testing**: Jest
- **Documentation**: Swagger
- **Optional Frontend**: Any language/framework (e.g., React, Vue, Angular, Svelte, HTML/CSS/JS) or readymade platform (e.g., Retool, Bubble)
- **Optional**: Winston (logging), NestJS WebSockets

Submission Guidelines

1. **Code**:
    - Push your code to a **public GitHub repository**.
    - Include a `README.md` with:
        - Instructions to set up and run the application locally (including prerequisites like Node.js 17+, Docker, etc.).
        - Steps to run unit tests.
        - A brief explanation of your architecture, design decisions (especially database schema), security/scalability practices, frontend (if implemented), and use of AI tools,

        readymade platforms, or WAHA.
        - Swagger documentation (e.g., link to `/api` endpoint or exported Swagger JSON).

2. **Loom Video**:
    - Record a 5-10 minute video on Loom.
    - Demonstrate:
        - Setting up the application using `docker-compose up`.
        - Creating a tenant and multiple users with different groups.
        - Logging in as a user and obtaining a JWT token.
        - Linking multiple WhatsApp devices (generating QR codes and switching between devices using Baileys or WAHA).
        - Sending a WhatsApp message to a contact or chat group from a specific device.
        - Retrieving message logs, contacts, and chat groups (demonstrating tenant isolation).
        - Explaining the database schema (e.g., via a diagram or SQL/MongoDB queries) and how it ensures scalability and standardization.
        - Running unit tests.
        - Accessing Swagger documentation.
        - (If implemented) Using the frontend to interact with the APIs.
    - Briefly explain your code structure, database schema design, security measures, scalability considerations, frontend (if applicable), and how you used AI tools, readymade platforms, or WAHA.
    - Share the Loom video link in your `README.md`.

3. **Deliverables**:
    - Submit the GitHub repository URL and Loom video link via email by **Monday, 8 AM IST**.
    - Ensure the repository includes all necessary files (source code, Docker files, tests, Swagger documentation, optional frontend).

Evaluation Criteria

Overall, candidates will be evaluated based on:

- **Scalable and Standardized Database Schema**: Is the database schema designed for tenant isolation, efficient indexing, and scalability? Does it follow best practices (e.g., normalization for PostgreSQL, optimized documents for MongoDB, connection pooling)?
- **WhatsApp Web Features**: Are core WhatsApp features (multi-device support, QR code authentication, message sending to contacts/groups, chat history storage) correctly implemented using Baileys or WAHA?
- **Multi-Tenant Mode**: Does the solution ensure strict data isolation between tenants and support multiple users and groups per tenant?

- **Functionality**: Does the application meet all requirements (user/group management, contact/group storage, etc.)?
- **Code Quality**: Is the code clean, modular, and well-documented? Are design patterns (e.g., Repository, CQRS) used appropriately?
- **Security**: Are tenant/user data isolated? Is JWT authentication secure? Are inputs validated and sanitized?
- **Scalability**: Does the architecture support high throughput? Are caching and async processing implemented?
- **API Design**: Are the APIs RESTful and well-documented with Swagger?
- **Testing**: Are unit tests comprehensive and cover critical components?
- **Deployment**: Can the application be easily deployed using Docker?
- **Documentation**: Is the `README.md` clear? Is Swagger documentation complete and usable?
- **Frontend (Optional)**: If implemented, is the frontend functional, responsive, and integrated with the backend?
- **Use of AI Tools/Platforms**: Did you effectively leverage AI tools, readymade platforms, or WAHA to streamline development while maintaining code quality?
- **Bonus Points**: Implementation of optional features (event-driven system, WebSocket notifications, structured logging, WAHA Plus).

Notes

- **Baileys**: Do not use `useMultiFileAuthState` for session storage in production. Implement a custom auth state using the database (e.g., MongoDB/PostgreSQL).
- **WAHA**: Use the Core version for free, or Plus for advanced features (requires donation). Follow WAHA's security guidelines (e.g., API key, HTTPS) and avoid exposing the instance publicly without a proxy like Nginx.
- **Disclaimer**: Both Baileys and WAHA are unofficial WhatsApp APIs and carry risks of account blocking. Ensure compliance with WhatsApp's terms and avoid spamming or bulk messaging. For production, consider official APIs like Twilio.
- If you encounter issues with Baileys or WAHA, document any workarounds or assumptions in your `README.md`.
- Focus on a production-ready solution with robust error handling, logging, and scalability.
- You may use AI tools (e.g., GitHub Copilot, ChatGPT) and readymade platforms (e.g., Auth0, Retool) to assist, but ensure you understand and can explain your code.
- If implementing a frontend, choose a language/framework you are comfortable with, or use a readymade platform, ensuring it integrates seamlessly with the backend APIs.

Good luck! We look forward to seeing your solution.