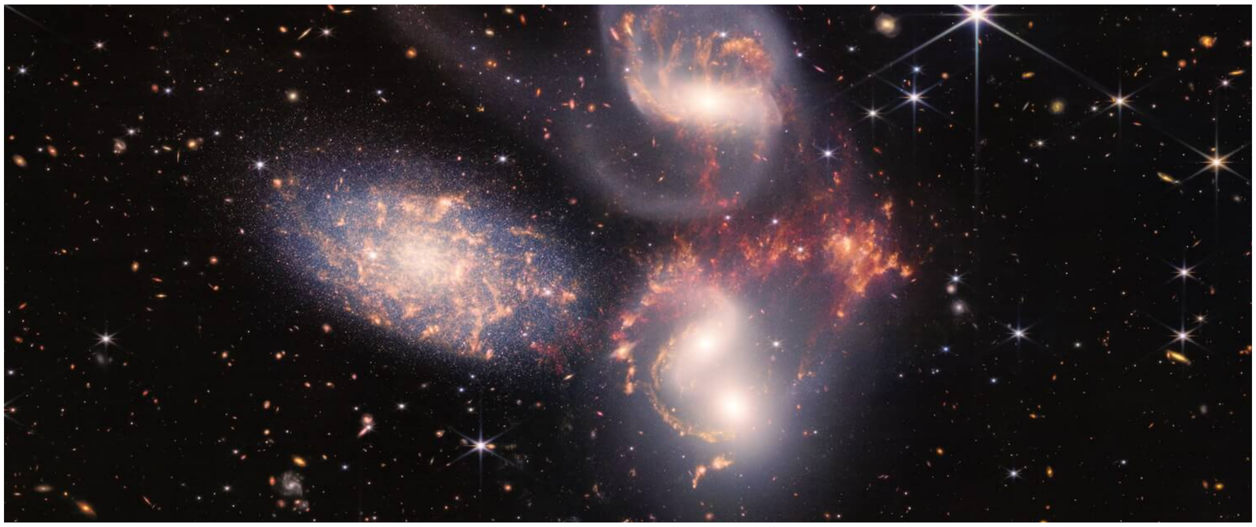

Task-2 Make It Real-Time

This task extends our API with real-time capabilities using WebSockets. The implementation leverages the Socket.IO library for bidirectional communication between clients and the server, while integrating with our existing GraphQL API to push updates in real time.

Features

- **JWT Authentication & Authorization**

  Uses JSON Web Tokens (JWT) to verify users for both GraphQL HTTP requests and Socket.IO connections.

- **Bidirectional Real-Time Communication**

  Enables real-time chat, message broadcasting, and room-based messaging through Socket.IO.

- **GraphQL Integration with Real-Time Updates**

  Utilizes a PubSub mechanism (from the `graphql-subscriptions` package) to emit updates (e.g., when a post is updated) to connected clients.

- **Structured API**

  Implements the API using Express and Apollo Server, and organizes the code for scalability and maintainability.

- **Extra Credits with Frameworks**

  Although the current implementation uses Express, the code is structured in a way that it can be easily adapted to frameworks like Nest.js or LoopBack for extra credits.

Architecture Overview

1. **Express & Apollo Server**
   - Sets up the main HTTP server.
   - Integrates Apollo Server to handle GraphQL queries and mutations.
   - Applies authentication middleware on the `/graphql` endpoint.
2. **Socket.IO Integration**
   - Initializes a Socket.IO server on top of the HTTP server.
   - Uses middleware to authenticate socket connections with JWT tokens.
   - Handles various real-time events such as private messaging, broadcasting, and room management.
3. **GraphQL PubSub Integration**
   - Uses the `graphql-subscriptions` library to publish and subscribe to real-time events.
   - For example, when a post is updated, an event is published that Socket.IO listens for and emits to clients.

Code Walkthrough

1. Server Initialization

The server uses Express, Apollo Server, and Socket.IO. The key steps include:

- **Connecting to the Database:**

  The `connectDB()` function is called before starting the server.
- **GraphQL Setup:**

The Apollo Server is configured with type definitions, resolvers, and a context that includes the authenticated user and PubSub instance.

- **HTTP and Socket Server:**

  An HTTP server is created from the Express app. The Socket.IO server is initialized on this HTTP server and configured to handle real-time events.

```
```

2. Socket.IO Initialization

The `initSocket` function configures Socket.IO with the following features:

- **JWT Authentication:**

  Uses a middleware to intercept the socket connection handshake. It extracts the token from the query parameters, verifies it using the same authentication middleware as the HTTP server, and stores the authenticated user in the socket's data.

- **Event Listeners:**
  - **sendMessage:** Sends a private message to the client's own room.
  - **broadcastMessage:** Emits a broadcast message to all connected clients.
  - **Room Management:** Allows users to join or leave rooms. When joining or leaving, notifications are sent to other room members.
  - **GraphQL PubSub:** Subscribes to the `POST_UPDATED` event to emit post updates in real time.

```
```

3. Client-Side Implementation

The HTML page demonstrates how clients can connect to the Socket.IO server, authenticate using a JWT token, and interact in real time:

- **Connecting to Socket.IO:**

  The client connects to the server with a token passed in the query string.

- **Handling Real-Time Events:**

Listeners are set up for receiving private messages, broadcasts, and post updates.

- **Sending Messages:**

  Buttons and input fields allow users to send messages, broadcast messages, and manage room membership.

```
```

Setup & Running Instructions

1. **Clone the Repository:**

   ```
   ```

2. **Install Dependencies:**

   Make sure you have Node.js installed, then run:

   ```
   ```

3. **Environment Variables:**

   Create a `.env` file in the root directory with the necessary configuration (e.g., database connection string, JWT secret, etc.).

4. **Start the Server:**

   ```
   ```

   The server will run on http://localhost:4000. The GraphQL endpoint will be available at `/graphql`.

5. **Test the Real-Time Features:**

   Open the provided HTML file (or serve it using a static server) and interact with the chat interface. Use your JWT token to authenticate.

Final Notes

- **Error Handling:**

  Custom errors are formatted in the GraphQL server, and errors in socket authentication are clearly communicated.

- **Security:**

  Ensure your JWT tokens are securely stored and transmitted, and adjust the CORS policy as necessary for your production environment.

This documentation should help you understand the implementation and serve as a guide for further enhancements. Enjoy building your real-time application!