Tentu, berdasarkan file-file yang Anda berikan, berikut adalah dokumentasi lengkap untuk menjalankan dan berintegrasi dengan server WebSocket Anda.

---

## Ringkasan Proyek

Ini adalah server WebSocket performa tinggi yang dibangun menggunakan Node.js dan `ws`. Server ini menggunakan **Redis Streams** sebagai message broker utama. Fitur utamanya meliputi:

- **Otentikasi berbasis Token**: Koneksi WebSocket diamankan menggunakan `WEBSOCKET_SECRET_KEY`.
- **Pub/Sub Real-time**: Klien dapat mem-publish pesan dan subscribe ke berbagai "channel".
- **Message History (Catch-up)**: Saat klien baru subscribe, server secara otomatis mengirimkan riwayat pesan (catch-up) dari Redis Stream, berdasarkan `lastMessageId` yang diberikan klien.
- **Acknowledgment**: Klien akan menerima konfirmasi (`ack`) setelah berhasil mem-publish pesan.
- **Docker-ready**: Dilengkapi dengan `Dockerfile` dan `docker-compose.yaml` untuk deployment yang mudah.

---

## Prasyarat

- Node.js (direkomendasikan v20, sesuai `Dockerfile`)
- NPM (terbawa oleh Node.js)
- Server Redis yang sedang berjalan (v5 ke atas, sesuai `package.json`)
- Docker dan Docker Compose (jika menggunakan metode Docker)

---

## Konfigurasi (Wajib)

Sebelum menjalankan, Anda harus membuat file `.env` di root proyek. Salin konten dari `.env.example` dan isi nilainya.

```
# Salin dari contoh
cp .env.example .env
```

**Isi file `.env`:**

- `WEBSOCKET_SECRET_KEY`: Kunci rahasia unik Anda untuk mengotentikasi koneksi WebSocket.
- `REDIS_URL`: URL koneksi ke server Redis Anda.
  - Contoh lokal: `redis://localhost:6379`
  - Contoh Docker (lihat di bawah): `redis://nama_container_redis:6379`
- `PORT`: Port yang akan digunakan server. (Default 8080 jika tidak diatur).

---

## Cara Menjalankan (Lokal)

Metode ini menjalankan aplikasi secara langsung di mesin Anda menggunakan Node.js.

1. **Instal Dependensi**: Buka terminal di folder proyek dan jalankan:

   ```
   npm install
   ```

   (Atau `npm ci` untuk instalasi yang lebih cepat sesuai `package-lock.json`)

2. **Pastikan Redis Berjalan**: Pastikan server Redis Anda aktif dan dapat diakses sesuai `REDIS_URL` yang Anda atur di `.env`.

3. **Siapkan `.env`**: Pastikan file `.env` Anda sudah dibuat dan diisi dengan benar. Contoh untuk lokal:

   ```
   WEBSOCKET_SECRET_KEY=kunci_rahasia_lokal_123
   REDIS_URL=redis://localhost:6379
   PORT=8080
   ```

4. **Jalankan Server**:

   ```
   npm start
   ```

   Server akan berjalan di `ws://localhost:8080` (atau port yang Anda tentukan).

---

## Cara Menjalankan (Docker)

Metode ini menggunakan Docker dan Docker Compose sesuai dengan file `Dockerfile` dan `docker-compose.yaml` Anda.

Konfigurasi `docker-compose.yaml` Anda dirancang untuk terhubung ke container Redis lain pada *network kustom*.

1. **Buat Network Docker**: (Anda hanya perlu melakukan ini sekali)

   ```
   docker network create my-shared-network
   ```

2. **Jalankan Redis di Network**: Jalankan container Redis (jika Anda belum memilikinya) dan hubungkan ke network yang sama.

   ```
   docker run -d --name my-redis-container --network my-shared-network redis:latest
   ```

3. **Siapkan `.env` untuk Docker**: Buat file `.env`. **PENTING**: `REDIS_URL` harus menggunakan nama container Redis (`my-redis-container`) sebagai host, sesuai petunjuk di `.env.example`.

   ```
   WEBSOCKET_SECRET_KEY=kunci_rahasia_docker_456
   # Gunakan nama container 'my-redis-container' sebagai host
   REDIS_URL=redis://my-redis-container:6379
   PORT=8080
   ```

4. **Jalankan Server WebSocket**: Dari direktori proyek Anda, jalankan:

   ```
   docker-compose up --build -d
   ```

   Perintah ini akan membangun image Anda (sesuai `Dockerfile`) dan menjalankan container (`websocket_server`). Server akan dapat diakses di `ws://localhost:8080` (karena port `8080:8080` di-mapping).

---

# Protokol API & Integrasi

Komunikasi dengan server WebSocket ini menggunakan format JSON.

## 1. Koneksi

Klien harus terhubung ke URL server dengan menyertakan token sebagai query parameter.

`ws://<HOST>:<PORT>?token=<WEBSOCKET_SECRET_KEY>`

**Contoh:** `ws://localhost:8080?token=kunci_rahasia_lokal_123`

Jika token salah, server akan menolak koneksi.

## 2. Protokol (JSON)

Berikut adalah aksi (`action`) yang dipahami oleh server:

### A. Client → Server: `subscribe`

Untuk mulai mendengarkan pesan di sebuah channel dan mendapatkan riwayat pesan.

```
{
  "action": "subscribe",
  "channel": "chat-room-1",
  "lastMessageId": "0"
}
```

- `channel` : Nama channel (Redis Stream) yang ingin didengarkan.
- `lastMessageId` :
  - `"0"` : (Atau "0-0") Meminta semua riwayat pesan dari awal.
  - `"1700000000000-0"` : (Contoh Stream ID) Meminta riwayat pesan *setelah* ID ini (catch-up).
  - `"$"` : Hanya mendengarkan pesan baru, **tidak** meminta riwayat.

### B. Client → Server: `publish`

Untuk mempublikasikan pesan ke sebuah channel.

```
{
  "action": "publish",
  "channel": "chat-room-1",
  "data": { "user": "alice", "text": "Halo semua!" },
  "messageId": "client-generated-uuid-12345"
}
```

- `channel` : Channel tujuan pesan.
- `data` : Konten pesan (objek JSON apapun).
- `messageId` : ID unik yang dibuat oleh klien. Server akan menggunakan ID ini untuk mengirim konfirmasi ( `ack` ).

---

### C. Server → Client: (Berbagai Respon)

Server akan mengirim pesan-pesan berikut ke klien:

- **Pesan Baru / Riwayat**:

  ```
  {
    "event": "message",
    "channel": "chat-room-1",
    "streamId": "1700000001234-0",
    "data": { "user": "alice", "text": "Halo semua!" }
  }
  ```

- **Konfirmasi Publish Berhasil**:

  ```
  {
    "status": "ack",
    "messageId": "client-generated-uuid-12345"
  }
  ```

- **Konfirmasi Publish Gagal**:

  ```
  {
    "status": "error_ack",
    "messageId": "client-generated-uuid-12345",
    "error": "Server Redis tidak siap"
  }
  ```

- **Status Koneksi & Subskripsi**:

  ```
  { "status": "connected", "message": "Welcome!" }
  ```

  ```
  { "status": "subscribed", "channel": "chat-room-1" }
  ```

---

## Contoh Integrasi Frontend (JavaScript)

Berikut adalah contoh sederhana klien WebSocket di browser.

```
<!DOCTYPE html>
<html>
<head><title>WS Test</title></head>
<body>
    <h1>WebSocket Client</h1>
    <input type="text" id="channel" placeholder="Channel (e.g., 'news')" value="news">
    <button id="btnSub">Subscribe</button>
    <hr>
    <input type="text" id="message" placeholder="Message">
    <button id="btnPub">Publish</button>
    <pre id="logs"></pre>

    <script>
        const logs = document.getElementById('logs');
        const wsToken = "kunci_rahasia_lokal_123"; // Ganti dengan secret key Anda
        const wsUrl = `ws://localhost:8080?token=${wsToken}`;
```

```
        const wsUrl = `ws://localhost:8080?token=${wsToken}`;
        let ws;

        function log(message) {
            logs.textContent += `${new Date().toLocaleTimeString()}: ${message}\n`;
        }

        function connect() {
            log("Menyambungkan...");
            ws = new WebSocket(wsUrl);

            ws.onopen = () => {
                log("✅ Terhubung ke server.");
            };

            ws.onmessage = (event) => {
                const msg = JSON.parse(event.data);
                log(`IN: ${JSON.stringify(msg)}`);

                // Cek jika ini adalah pesan yang kita publish
                if (msg.status === 'ack') {
                    log(`PUBLISH SUKSES (ID: ${msg.messageId})`);
                }
            };

            ws.onclose = () => {
                log("❌ Koneksi terputus.");
            };

            ws.onerror = (err) => {
                log(`ERROR: ${err.message}`);
            };
        }

        // Tombol Subscribe
        document.getElementById('btnSub').onclick = () => {
            const channel = document.getElementById('channel').value;
            log(`Subscribing ke channel '${channel}'...`);
            ws.send(JSON.stringify({
                action: "subscribe",
                channel: channel,
                lastMessageId: "0" // Ambil semua riwayat
            }));
        };

        // Tombol Publish
        document.getElementById('btnPub').onclick = () => {
            const channel = document.getElementById('channel').value;
            const message = document.getElementById('message').value;
            const clientMsgId = `msg-${Date.now()}`;

            log(`Publishing ke '${channel}'...`);
            ws.send(JSON.stringify({
                action: "publish",
                channel: channel,
                data: { from: "browser", text: message },
                messageId: clientMsgId
            }));
            document.getElementById('message').value = '';
        };

        connect(); // Langsung sambungkan saat halaman dimuat
    </script>
</body>
</html>
```

## Contoh Integrasi Backend (Node.js)

Karena arsitektur Anda menggunakan Redis Streams sebagai pusat, layanan backend lain (misalnya, API server) tidak perlu terhubung ke WebSocket server.

Alih-alih, layanan backend Anda dapat **langsung mem-publish pesan ke Redis Stream**. Server WebSocket akan otomatis mendeteksi pesan baru tersebut dan menyiarkannya ke semua klien yang subscribe.

Berikut adalah contoh skrip Node.js (sebagai backend service terpisah) yang mem-publish pesan ke Redis.

```
// backend_publisher.js
// Skrip ini BUKAN bagian dari server.js,
// tapi contoh bagaimana service lain (misal: API) bisa mem-publish pesan.

const { createClient } = require('redis');

// Pastikan ini terhubung ke Redis yang SAMA dengan server WS Anda
const REDIS_URL = process.env.REDIS_URL || "redis://localhost:6379";
const redisClient = createClient({ url: REDIS_URL });

async function publishMessage(channel, data) {
    if (!redisClient.isReady) {
        await redisClient.connect();
    }

    try {
        const dataToStream = {
            // Sesuai dengan format di server.js
            messageData: JSON.stringify(data)
        };

        // Gunakan XADD untuk mem-publish ke Stream
        const streamId = await redisClient.xAdd(
            channel,
            '*', // '*' berarti ID digenerate otomatis oleh Redis
            dataToStream
        );

        console.log(`Pesan berhasil dipublish ke channel '${channel}' dengan ID: ${streamId}`);
        console.log(`Data:`, data);

    } catch (err) {
        console.error("Gagal publish ke Redis Stream:", err);
    } finally {
        await redisClient.quit();
    }
}

// --- Contoh Penggunaan ---
const channel = 'penting';
const message = {
    type: 'ALERT',
    timestamp: new Date().toISOString(),
    details: 'Server A sedang maintenance.'
};

publishMessage(channel, message);
```