Descripción General

Implementa un cliente WebSocket robusto con autenticación JWT, manejo de reconexión automática y sistema de suscripción a tópicos. Proporciona comunicación en tiempo real entre el frontend y el backend.

Clase Principal `RTClient`

Constructor e Inicialización

```javascript
export class RTClient {
  constructor() {
    this.ws = null;
    this.handlers = new Map(); // topic -> Set<fn>
    this.url = null;
    this.closing = false;
    this.backoff = { base: 500, max: 8000, factor: 2 };
  }
}
```

Propiedades:

- `ws`: Instancia de WebSocket activa
- `handlers`: Mapa de tópicos a funciones callback
- `url`: URL de conexión WebSocket con token
- `closing`: Flag para controlar cierre intencional
- `backoff`: Configuración de reintentos exponenciales

Método onMessage(ev)

```javascript
onMessage(ev) {
  try {
    const data = JSON.parse(ev.data);
    if (data?.topic) {
      const set = this.handlers.get(data.topic);
      if (set) set.forEach(fn => fn(data));
    }
  } catch {
    // ignore
  }
}
```

Propósito: Procesa mensajes entrantes del WebSocket

- Formato esperado: `{topic, ts, payload}`
- Distribución: Ejecuta todos los handlers registrados para el tópico
- Manejo de errores: Silencia errores de parsing

Método connect()

```javascript
async connect() {
  this.closing = false;
  const token = storage.get(TOKEN_KEY, null, true);
```

```
  if (!token) throw new Error("Sin token: inicie sesión");


  const base = CFG().WS_URL; // ej: ws://localhost:3000/ws

  this.url = `${base}?token=${encodeURIComponent(token)}`;


  let attempt = 0;

  while (!this.closing) {

    try {

      await this._connectOnce();

      return; // conectado: salir del bucle

    } catch (e) {

      attempt++;

      const delay = Math.min(this.backoff.base *
Math.pow(this.backoff.factor, attempt), this.backoff.max);

      const jitter = Math.random() * 150;

      await sleep(delay + jitter);

    }

  }

}
```

Propósito: Establece conexión WebSocket con reconexión automática

Flujo:

1. Verifica token: Requiere autenticación
2. Construye URL: `ws://host/ws?token=JWT`
3. Reintentos: Con backoff exponencial + jitter

4. Algoritmo de reconexión:
    - Base: 500ms, máximo: 8000ms, factor: 2
    - Jitter: ±150ms para evitar sincronización

Método `_connectOnce()`

```javascript
_connectOnce() {
  return new Promise((resolve, reject) => {
    const ws = new WebSocket(this.url);
    this.ws = ws;

    ws.onopen = () => resolve();
    ws.onmessage = (ev) => this.onMessage(ev);
    ws.onclose = () => {
      this.ws = null;
      if (!this.closing) this.connect(); // reconectar en background
    };
    ws.onerror = (err) => {
      try { ws.close(); } catch {}
      reject(err);
    };
  });
}
```

Propósito: Intento único de conexión WebSocket

Eventos:

- `onopen`: Resuelve la promesa
- `onmessage`: Delega al procesador de mensajes
- `onclose`: Trigger reconexión si no es cierre intencional
- `onerror`: Rechaza la promesa y limpura conexión

Sistema de Suscripciones

`subscribe(topic, fn)`

```
javascript

subscribe(topic, fn) {

  if (!this.handlers.has(topic)) this.handlers.set(topic, new Set());

  this.handlers.get(topic).add(fn);

  // Enviar sub al server

  this._send({ type: "sub", topic });

  // Retorna unsubscribe de conveniencia

  return () => this.unsubscribe(topic, fn);

}
```

Propósito: Suscribe una función a un tópico

- Registro: Añade handler al mapa interno
- Notificación al servidor: Envía comando `{type: "sub", topic}`
- Retorno: Función de unsuscribe para conveniencia

`unsubscribe(topic, fn)`

```
javascript

unsubscribe(topic, fn) {
```

```
  const set = this.handlers.get(topic);

  if (set) {

    set.delete(fn);

    if (set.size === 0) this.handlers.delete(topic);

  }

  this._send({ type: "unsub", topic });

}
```

Propósito: Remueve suscripción específica

- Limpieza: Elimina handler y tópico vacío
- Notificación al servidor: Envía comando `unsub`

Métodos de Soporte

`_send(obj)`

```
javascript

_send(obj) {

  if (this.ws && this.ws.readyState === WebSocket.OPEN) {

    this.ws.send(JSON.stringify(obj));

  }

}
```

Propósito: Envía datos al servidor de forma segura

- Verificación: Solo envía si conexión está abierta
- Serialización: Convierte objeto a JSON

`close()`

```
javascript
```

```
async close() {
  this.closing = true;
  if (this.ws) {
    try { this.ws.close(); } catch {}
    this.ws = null;
  }
}
```

Propósito: Cierra conexión intencionalmente

- Flag: Establece `closing = true` para evitar reconexión
- Limpieza: Cierra WebSocket y limpia referencia

Utilidades

Función `sleep(ms)`

```
javascript
function sleep(ms) { return new Promise(r => setTimeout(r, ms)); }
```

Propósito: Utilidad para delays asíncronos

Singleton `rtClient`

```
javascript
export const rtClient = new RTClient();
```

Propósito: Instancia única para uso global en la aplicación