Descripción General

El archivo `api.js` implementa un wrapper completo para manejo de API con autenticación JWT, gestión de sesiones y endpoints organizados por funcionalidad.

Estructura del Módulo

1. Constantes y Configuración

```
javascript
```

```javascript
const CFG = () => window.__CONFIG || {};

const TOKEN_KEY = "auth_token";

const USER_KEY = "auth_user";

const ROLE_KEY = "auth_role";
```

Propósito: Define las claves para almacenamiento y función para acceder a configuración global.

2. Gestión de Sesión

`getToken()`

```
javascript
```

```javascript
export function getToken() {

  return storage.get(TOKEN_KEY);

}
```

Función: Recupera el token JWT del almacenamiento.

`setSession({ accessToken, user, role })`

```
javascript
```

```
export function setSession({ accessToken, user, role }) {
  storage.set(TOKEN_KEY, accessToken, true); // sessionStorage
  storage.set(USER_KEY, user, true);
  storage.set(ROLE_KEY, role, true);
  setState({ user, role });
}
```

Propósito: Establece una nueva sesión de usuario.

- Parámetros:
  - `accessToken`: Token JWT
  - `user`: Objeto con datos del usuario
  - `role`: Rol del usuario
- Almacenamiento: Usa sessionStorage para persistencia durante la sesión del navegador.

`clearSession()`

```
javascript
export function clearSession() {
  storage.del(TOKEN_KEY, true);
  storage.del(USER_KEY, true);
  storage.del(ROLE_KEY, true);
  // También limpiar localStorage por seguridad
  storage.del(TOKEN_KEY, false);
  storage.del(USER_KEY, false);
  storage.del(ROLE_KEY, false);
  setState({ user: null, role: ROLES_CONST.GUEST, currentProject: null });
}
```

Propósito: Limpia completamente la sesión del usuario.

- Seguridad: Elimina datos tanto de sessionStorage como localStorage.

`initSession()`

```
javascript
export async function initSession() {
  const token = getToken();
  const storedUser = storage.get(USER_KEY, null, true);
  const storedRole = storage.get(ROLE_KEY, null, true);

  if (token && storedUser && storedRole) {
    try {
      await request("/config/general", { auth: true });
      setSession({ accessToken: token, user: storedUser, role: storedRole
});
      return true;
    } catch (error) {
      clearSession();
      return false;
    }
  }
  return false;
}
```

Propósito: Inicializa la sesión al cargar la aplicación.

- Validación: Verifica que el token sea válido haciendo una petición autenticada.
- Retorno: `true` si la sesión se restauró correctamente, `false` en caso contrario.

3. Función Principal `request()`

```javascript
async function request(path, { method = "GET", body, auth = false } = {}) {
  const headers = { "Content-Type": "application/json" };
  if (auth) {
    const t = getToken();
    if (t) headers["Authorization"] = `Bearer ${t}`;
  }
  const res = await fetch(`${CFG().API_URL}${path}`, {
    method,
    headers,
    credentials: "include",
    body: body ? JSON.stringify(body) : undefined
  });
  const data = await res.json().catch(() => ({}));
  if (!res.ok) throw Object.assign(new Error("API error"), { status:
res.status, data });
  return data;
}
```

Características:

- Autenticación: Soporte para JWT Bearer tokens
- Manejo de errores: Lanza excepciones con detalles del error
- Configuración: Usa URL base desde configuración global

4. APIs Específicas

AuthAPI - Autenticación

```javascript
export const AuthAPI = {
```

```
  googleLogin(credential) {
    return request("/auth/google", { method: "POST", body: { credential } });
  }
};
```

ConfigAPI - Configuración del Sistema

```
javascript
export const ConfigAPI = {
  // Configuración general - todos los autenticados
  getGeneralConfig(),

  // Configuración avanzada - solo admin
  getAdvancedConfig(),
  updateAdvancedConfig(config),

  // Administración MQTT
  getMQTTStatus(),
  getMQTTTopics(),
  restartMQTTConnection(),
  reloadMQTTTopics(),
  clearDataCache(),

  // CRUD de tópicos MQTT (solo admin)
  createMQTTTopic(topicData),
  updateMQTTTopic(topicId, topicData),
  deleteMQTTTopic(topicId)
```

```
};
```

DevicesAPI - Gestión de Dispositivos

```
javascript

export const DevicesAPI = {

  getAllDevices(),

  getDeviceById(deviceId),

  getDeviceSensorData(deviceId, limit = 100)

};
```

Flujo de Autenticación

1. Login: `AuthAPI.googleLogin(credential)` → Retorna token y datos de usuario
2. Establecer sesión: `setSession()` → Almacena en sessionStorage y estado global
3. Peticiones autenticadas: Incluyen header `Authorization: Bearer <token>`
4. Verificación de sesión: `initSession()` al cargar la aplicación
5. Logout: `clearSession()` → Limpia todos los datos de sesión

Características de Seguridad

- Tokens JWT: Autenticación stateless con tokens Bearer
- Almacenamiento seguro: sessionStorage para datos sensibles
- Limpieza completa: Eliminación de datos en ambos storage types
- Validación de token: Verificación automática al inicializar sesión