Descripción General

Funciona como el cargador de configuración de la aplicación. Su propósito principal es cargar la configuración desde múltiples fuentes antes de iniciar la aplicación principal, proporcionando valores por defecto y manejo de fallos robusto.

Estructura y Flujo

1. Configuración por Defecto

```
javascript
```

```
const defaults = {

  API_URL: `http://${location.hostname}:4000/api`,

  WS_URL:  `ws://${location.hostname}:4000/ws`,

  GOOGLE_CLIENT_ID: "",

  ROLES: { ADMIN: "admin", ACTION: "action", READONLY: "readonly", GUEST: "guest" }

};
```

Propósito: Define valores predeterminados para cuando no haya configuración disponible.

Valores por defecto:

- `API_URL`: Endpoint de API en puerto 4000 del mismo host
- `WS_URL`: WebSocket endpoint en puerto 4000
- `GOOGLE_CLIENT_ID`: Vacío (se obtendrá del backend)
- `ROLES`: Diccionario de roles del sistema

2. Función Principal `loadConfig()`

```
javascript
```

```
async function loadConfig() {

  // Flujo de carga en 2 etapas principales

}
```

Etapa 1: Carga de config.json

```
javascript

let cfgFromJson = null;

try {

  const rCfg = await fetch("/config.json", { cache: "no-store" });

  if (rCfg.ok) cfgFromJson = await rCfg.json().catch(() => null);

} catch {}



const base = cfgFromJson && typeof cfgFromJson === "object" ?

  { ...defaults, ...cfgFromJson } :

  { ...defaults };
```

Propósito: Intentar cargar configuración desde archivo estático `config.json`

Características:

- Cache: `no-store` para evitar cache no deseado
- Fallback: Si falla, usa valores por defecto
- Merge: Combina configuración JSON con defaults

Etapa 2: Obtención de Google Client ID

```
javascript

const apiUrl = (base.API_URL || defaults.API_URL).replace(/\/$/, "");

const backendConfigUrl = `${apiUrl}/config`;
```

```
let googleClientId = "";
```

Intento 2a: Desde API del backend

```
javascript
```

```
try {
  const r = await fetch(backendConfigUrl, { cache: "no-store" });
  if (r.ok) {
    const j = await r.json().catch(() => ({}));
    if (j && j.GOOGLE_CLIENT_ID) googleClientId = j.GOOGLE_CLIENT_ID;
  }
} catch {}
```

Intento 2b: Desde proxy local

```
javascript
```

```
if (!googleClientId) {
  try {
    const rLocal = await fetch("/api/config", { cache: "no-store" });
    if (rLocal.ok) {
      const j = await rLocal.json().catch(() => ({}));
      if (j && j.GOOGLE_CLIENT_ID) googleClientId = j.GOOGLE_CLIENT_ID;
    }
  } catch {}
}
```

Estrategia de obtención:

1. Backend directo: Usando `API_URL/config`
2. Proxy local: Usando `/api/config` (para entornos con proxy reverso)

3. Configuración Final y Exportación

```
javascript

const finalCfg = { ...base };

if (googleClientId) finalCfg.GOOGLE_CLIENT_ID = googleClientId;



return finalCfg;
```

Inicialización Global

```
javascript

window.__CONFIG = await loadConfig();

await import("./app.js");
```

Propósito: Hace la configuración globalmente disponible e inicia la aplicación.

Flujo Completo de Carga

1. Definir defaults → Valores base de respaldo
2. Cargar config.json → Configuración estática personalizada
3. Obtener Google Client ID → Desde backend o proxy
4. Combinar configuración → Merge inteligente de fuentes
5. Exponer globalmente → `window.__CONFIG` disponible para toda la app
6. Iniciar aplicación → Carga dinámica de `app.js`