Manejo de la navegación de la aplicación SPA (Single Page Application) con control de acceso por roles de usuario.

Características Principales

- Enrutamiento basado en hash (ej: `#/dashboard`)
- Carga dinámica de páginas (code splitting)
- Guardias de ruta por roles de usuario
- Protección de rutas privadas/públicas
- Manejo de rutas no encontradas (404)

Configuración de Rutas

Definición del Objeto `routes`

```javascript
      const routes = {
        "": {
          view: () => import("../pages/login.js").then(m => m.render),
          meta: { public: true }
        },
        "login": {
          view: () => import("../pages/login.js").then(m => m.render),
          meta: { public: true }
        },
        "dispositivos": {
```

```
    view: () => import("../pages/dispositivos.js").then(m =>
m.render),

    meta: { roles: [ROLES_CONST.ADMIN, ROLES_CONST.ACTION,
ROLES_CONST.READONLY] }

  },

  "dashboard": {

    view: () => import("../pages/dashboard.js").then(m => m.render),

    meta: { roles: [ROLES_CONST.ADMIN, ROLES_CONST.ACTION,
ROLES_CONST.READONLY] }

  },

  "sobre-nosotros": {

    view: () => import("../pages/sobreNosotros.js").then(m =>
m.render),

    meta: { public: true }

  },

  "configuracion": {

    view: () => import("../pages/configuracion.js").then(m =>
m.render),

    meta: { roles: [ROLES_CONST.ADMIN, ROLES_CONST.ACTION,
ROLES_CONST.READONLY] }

  },

  "configuracion/avanzada": {

    view: () => import("../pages/configuracionAvanzada.js").then(m =>
m.render),

    meta: { roles: [ROLES_CONST.ADMIN] }
```

```
  },
  "404": {
    view: () => import("../pages/notFound.js").then(m => m.render),
    meta: { public: true }
  }
};
```

Propiedades de Ruta

Propiedad	Tipo	Descripción
`view`	`Function`	Función que importa y devuelve el render de la página
`meta.public`	`Boolean`	Si es `true`, acceso sin autenticación
`meta.roles`	`Array`	Lista de roles permitidos para la ruta

Funciones del Router

`parseHash()`

Propósito: Extrae y normaliza el hash de la URL.

```javascript
function parseHash() {
  const h = location.hash.replace(/^#\/?/, "");
  return h || "login"; // Default: login
}
```

Ejemplos:

- `#/dashboard` → `"dashboard"`
- `#/` → `"login"`
- `#` → `"login"`

`canAccess(hash, role, isLogged)`

Propósito: Determina si un usuario puede acceder a una ruta.

```javascript
function canAccess(hash, role, isLogged) {
  const def = routes[hash] || routes["404"];
  const { meta = {} } = def;

  if (meta.public) return true;           // Rutas públicas
  if (!isLogged) return false;            // Usuario no autenticado
  if (!meta.roles) return true;           // Sin restricciones de rol
  return meta.roles.includes(role);       // Verificación de rol
```

```
}
```

`initRouter()`

Propósito: Inicializa el router y configura los listeners de navegación.

```
javascript
```

```javascript
export function initRouter() {
  const app = document.getElementById("app");

  const navigate = async () => {
    const { role, user } = getState();
    const hash = parseHash();

    // Verificación de acceso
    if (!canAccess(hash, role, !!user)) {
      location.hash = "#/login";
      return;
    }

    // Carga y renderizado de página
    const def = routes[hash] || routes["404"];
    const render = await def.view();
```

```
    mount(app, await render());


    // Actualización de navegación activa
    document.querySelectorAll('[data-nav]').forEach(a => {
      a.classList.toggle("active", a.getAttribute("data-nav") === hash);
    });
  };

  window.addEventListener("hashchange", navigate);
  navigate(); // Navegación inicial
}
```

`navigate(path)`

Propósito: Navegación programática entre rutas.

```
javascript
export function navigate(path) {
  location.hash = `#/${path}`;
}
```

Uso:

```
javascript
// Desde cualquier componente
```

```
navigate("dashboard");
navigate("configuracion/avanzada");
```

Sistema de Roles

Constantes de Roles (ROLES_CONST)

```
javascript
// En store.js (asumido)
ROLES_CONST = {
  ADMIN: "admin",
  ACTION: "action",
  READONLY: "readonly"
}
```

Flujo de Navegación

1. Usuario hace clic en enlace → Cambia `location.hash`
2. Evento `hashchange` → Dispara `navigate()`
3. Verificación de acceso → `canAccess()`
4. Carga de página → `import()` dinámico
5. Renderizado → `mount()` en el DOM
6. Actualización UI → Clase `active` en navegación

Manejo de Errores

- Ruta no encontrada: Redirige automáticamente a página 404
- Acceso denegado: Redirige a login
- Error de carga: Manejo nativo de `import()` (Promise rejection)

Consideraciones de Seguridad

1. Protección del lado cliente: Este router proporciona protección UX, pero se debe implementar validación de roles en el backend.
2. Rutas anidadas: El sistema soporta rutas con `/` (ej: `configuracion/avanzada`).
3. Estado persistente: El estado de autenticación debe persistir entre recargas.

Dependencias

- `dom.js`: Utilidades `mount()` y `el()` para manipulación DOM
- `store.js`: Estado global y constantes de roles (`getState()`, `ROLES_CONST`)

Este router proporciona una solución completa para aplicaciones SPA con control de acceso granular basado en roles de usuario.