Contents

DOCUMENTACIÓN COMPLETA DEL SISTEMA

CERTUS Drive - Sistema de Gestión de Recursos Administrativos

---

ÍNDICE

RESUMEN EJECUTIVO

Descripción del Proyecto

CERTUS Drive es un sistema web de gestión de recursos internos desarrollado para el personal administrativo y operativo de la institución CERTUS. El sistema permite organizar, buscar y administrar recursos compartidos de Google Drive distribuidos por 8 áreas operativas diferentes.

Objetivos del Sistema

Centralizar recursos de Google Drive dispersos en diferentes áreas
Facilitar la búsqueda y acceso a documentos institucionales internos
Administrar contenido con panel de control intuitivo
Organizar recursos por áreas operativas con correos de contacto
Escalar el sistema para futuras necesidades institucionales

Características Principales

- **Búsqueda avanzada** con filtros por área y palabras clave
- **Panel de administración** para gestión CRUD de recursos y áreas
- **Interfaz moderna** con diseño responsivo
- **Sistema de autenticación** con roles de usuario
- **Información de contacto** por área operativa

- **Alto rendimiento** con lazy loading y paginación

Áreas Operativas

1. **ATENCIÓN - FRONT**
2. **ATENCIÓN - CANALES**
3. **CRÉDITO Y COBRANZAS**
4. **FACTURACIÓN**
5. **RR.AA** (Recursos Administrativos)
6. **PEC** (Programa de Educación Continua)
7. **REINGRESO**
8. **OPERACIONES**

ARQUITECTURA DEL SISTEMA

Tipo de Arquitectura

Arquitectura JAMstack (JavaScript, APIs, Markup)

- **Frontend**: Single Page Application (SPA) con Vanilla JavaScript
- **Backend**: Serverless con Supabase (PostgreSQL + Auth)
- **Hosting**: Netlify CDN
- **Assets**: Almacenamiento estático optimizado

Diagrama de Arquitectura

```
                    USUARIO
```

```
                NETLIFY CDN
```

- `Hosting estático`
- `SSL/TLS automático`
- `Compresión Gzip/Brotli`
- `Cache inteligente`

```
          FRONTEND (Vanilla JavaScript)

  index.html       admin.html       login.html
   (Público)         (Admin)          (Auth)
```

JavaScript Modules

- script.js
- admin.js
- areas.js
- auth.js
- login.js
- config.js

SUPABASE (Backend)

PostgreSQL DB

- recursos table
- areas table
- profiles table
- RLS Policies

Auth Service

- Email/Password
- JWT Tokens
- Session Mgmt

REST API + Real-time

GOOGLE DRIVE

- Almacenamiento de archivos institucionales
- Enlaces compartidos internos

Flujo de Datos

1. **Usuario accede** → Netlify CDN sirve archivos estáticos
2. **JavaScript carga** → Se inicializa cliente de Supabase
3. **Autenticación** → Verificación de sesión con JWT
4. **Consultas DB** → REST API de Supabase con RLS
5. **Renderizado** → DOM manipulation con Vanilla JS
6. **Enlaces Drive** → Redirección a Google Drive

STACK TECNOLÓGICO

Frontend

HTML5

- **Versión**: HTML5 (Estándar moderno)
- **Uso**: Estructura semántica de las páginas
- **Características**:
  - Tags semánticos (`<section>`, `<nav>`, `<main>`)

  - Accesibilidad ARIA
  - Formularios interactivos

CSS3

- **Versión**: CSS3 con Custom Properties
- **Metodología**: BEM (Block Element Modifier)
- **Características**:
  - Variables CSS (`:root`)
  - Flexbox y Grid Layout
  - Media Queries responsivas
  - Animaciones y transiciones
  - Box-shadow y gradientes

Archivos CSS:

```
css/
  styles.css      # Estilos principales (index)
  admin.css       # Estilos del panel admin
```

JavaScript (Vanilla)

- **Versión**: ES6+ (ECMAScript 2015+)
- **Características utilizadas**:
  - `async/await` para operaciones asíncronas
  - Arrow functions
  - Template literals
  - Destructuring
  - Spread operator
  - Array methods (map, filter, reduce)
  - Optional chaining (`?.`)
  - Nullish coalescing (`??`)

Módulos JavaScript:

```
js/
  config.js       # Configuración de Supabase
  auth.js         # Sistema de autenticación
  login.js        # Lógica de login
  script.js       # Lógica principal (index)
  admin.js        # Panel de administración
  areas.js        # Gestión de áreas (CRUD)
```

Backend as a Service (BaaS)

Supabase

- **Versión**: 2.x (Latest)
- **Tipo**: PostgreSQL + APIs REST + Auth
- **Características**:
  - Base de datos PostgreSQL 15

  - Row Level Security (RLS)
  - Autenticación JWT
  - REST API auto-generada
  - Real-time subscriptions (no usado actualmente)

Cliente de Supabase:

```
// Carga desde CDN
https://cdn.jsdelivr.net/npm/@supabase/supabase-js@2
```

Hosting y CDN

Netlify

- **Plan**: Free Tier
- **Características utilizadas**:
  - Deploy automático desde Git
  - SSL/TLS gratuito (Let's Encrypt)
  - CDN global distribuido
  - Compresión automática (Gzip/Brotli)
  - Headers de seguridad personalizados
  - Redirects y rewrites
  - Deploy previews (opcional)

Configuración: - Archivo: `netlify.toml` - Redirects: `_redirects`

Almacenamiento Externo

Google Drive

- **Uso**: Almacenamiento de archivos institucionales internos
- **Tipo de enlaces**: Compartidos con permisos restringidos al personal
- **Formato de enlaces**:

  `https://drive.google.com/file/d/{FILE_ID}/view`
  `https://drive.google.com/drive/folders/{FOLDER_ID}`

Control de Versiones

Git & GitHub

- **Repositorio**: `certusdrives-gif/CERTUSDRIVES`
- **Branch principal**: `main`
- **Estrategia**: Git Flow simplificado

INFRAESTRUCTURA Y SERVICIOS

1. Netlify (Hosting y CDN)

Configuración de Proyecto URL de Producción: `https://[tu-sitio].netlify.app`

Build Settings:

```
[build]
  publish = "."
  command = ""
```

```
[[headers]]
  for = "/*"
  [headers.values]
    Strict-Transport-Security = "max-age=31536000; includeSubDomains; preload"
    X-Frame-Options = "SAMEORIGIN"
    X-XSS-Protection = "1; mode=block"
    X-Content-Type-Options = "nosniff"
    Referrer-Policy = "strict-origin-when-cross-origin"
    Permissions-Policy = "geolocation=(), microphone=(), camera=()"
```

Headers de Seguridad

Características Activas HTTPS forzado
Compresión automática
Cache optimizado
Deploy continuo desde Git
Minificación automática

2. Supabase (Backend)

Proyecto Supabase **URL del Proyecto**: `https://[tu-proyecto].supabase.co`
Región: Seleccionable (recomendado: `us-east-1` o más cercano)

Servicios Utilizados

PostgreSQL Database

- **Versión**: PostgreSQL 15
- **Conexión**: REST API + JavaScript Client
- **Tablas**: 3 principales (recursos, areas, profiles)
- **Storage**: Incluido en plan free (500MB)

Authentication

- **Métodos habilitados**:
  - Email/Password
  - Magic Links
  - OAuth (Google, GitHub, etc.)
- **JWT**: Tokens con expiración de 1 hora
- **Refresh Tokens**: Automáticos

Row Level Security (RLS)

- Políticas personalizadas por tabla
- Restricciones basadas en roles
- Acceso público para lectura
- Acceso admin para escritura

```
# Supabase
SUPABASE_URL=https://[tu-proyecto].supabase.co
SUPABASE_ANON_KEY=[tu-anon-key]

# No exponer en frontend:
SUPABASE_SERVICE_ROLE_KEY=[solo-backend]
```

Variables de Entorno Archivo `.env.example`:

```
SUPABASE_URL=tu_supabase_url_aqui
SUPABASE_ANON_KEY=tu_supabase_anon_key_aqui
```

3. Google Drive (Almacenamiento)

Organización de Archivos

```
CERTUS Drive/
   ATENCIÓN - FRONT/
       Manuales/
       Procedimientos/
       Formatos/
   ATENCIÓN - CANALES/
   CRÉDITO Y COBRANZAS/
   FACTURACIÓN/
   RR.AA/
   PEC/
   REINGRESO/
   OPERACIONES/
```

Permisos de Enlaces

- **Tipo**: Cualquiera con el enlace puede ver
- **Seguridad**: Enlaces no indexables por Google
- **Gestión**: Desde Google Drive corporativo

ESTRUCTURA DEL PROYECTO

Árbol de Directorios

```
temp-deploy/
    index.html                    # Página principal (pública)
```

```
admin.html                    # Panel de administración
login.html                    # Página de inicio de sesión
README.md                     # Documentación básica
netlify.toml                  # Configuración de Netlify
_redirects                    # Reglas de redirección
.env.example                  # Plantilla de variables de entorno
.gitignore                    # Archivos ignorados por Git

css/                          # Hojas de estilo
  styles.css                   # Estilos principales (1000+ líneas)
  admin.css                    # Estilos del admin (1000+ líneas)

js/                           # Scripts JavaScript
  config.js                    # Configuración Supabase
  auth.js                      # Sistema de autenticación
  login.js                     # Lógica de login
  script.js                    # Lógica principal index (800+ líneas)
  admin.js                     # Panel admin (600+ líneas)
  areas.js                     # CRUD de áreas (470+ líneas)

assets/                       # Recursos estáticos
  logo.svg                     # Logo de CERTUS
  favicon.ico                  # Favicon del sitio

docs/                         # Documentación
   DOCUMENTACION_COMPLETA.md   # Este documento
   GUIA_USUARIO.md             # Guía de usuario
   GUIA_TECNICA.md             # Guía técnica
   SQL_CREATE_AREAS_TABLE.sql  # Script de creación de tabla areas
   SQL_MIGRAR_RECURSOS_AREAS_SEGURO.sql  # Migración FK
   [otros archivos de documentación]
```

Descripción de Archivos Principales

HTML `index.html` (Página Principal) - Barra de navegación con logo y botones - Sección de búsqueda avanzada - Carrusel de áreas operativas - Grid de resultados con paginación - Skeleton loaders para UX - Footer institucional

`admin.html` (Panel de Administración) - Sistema de pestañas (Recursos / Áreas) - Estadísticas en tiempo real - Tablas con paginación - Modales para CRUD - Filtros y búsqueda - Notificaciones toast

`login.html` (Autenticación) - Formulario de login - Validación de campos - Manejo de errores - Redirección automática

JavaScript `config.js` (Configuración)

```
// Inicialización de Supabase
const supabaseUrl = 'https://[proyecto].supabase.co'
```

```
const supabaseKey = '[anon-key]'
const supabase = window.supabase.createClient(supabaseUrl, supabaseKey)
```

auth.js (Sistema de Autenticación) - Verificación de sesión - Protección de rutas admin - Manejo de JWT - Cierre de sesión

script.js (Lógica Principal) - Carga de recursos desde BD - Sistema de búsqueda - Filtrado por área - Paginación de resultados - Renderizado dinámico

admin.js (Panel Admin) - CRUD de recursos - Gestión de estados - Estadísticas - Validaciones de formularios - Filtros y búsqueda admin

areas.js (Gestión de Áreas) - CRUD completo de áreas - Sistema de paginación - Búsqueda de áreas - Modales y notificaciones - Validación de datos

CSS **styles.css** (Estilos Principales) - Variables CSS personalizadas - Layout responsivo - Componentes reutilizables - Animaciones - Media queries

admin.css (Estilos Admin) - Layout de panel - Tablas responsivas - Formularios estilizados - Modales - Estados interactivos

---

MÓDULOS Y COMPONENTES

1. Sistema de Autenticación

Componentes **Login Form** (`login.html`)

```
<form id="loginForm">
  <input type="email" id="loginEmail" required />
  <input type="password" id="loginPassword" required />
  <button type="submit">Iniciar Sesión</button>
</form>
```

Auth Guard (`auth.js`)

```
async function checkAuth() {
  const { data: { session } } = await supabase.auth.getSession()
  if (!session) window.location.href = '/login.html'
}
```

Flujo de Autenticación

1. Usuario ingresa credenciales
2. `supabase.auth.signInWithPassword()`
3. Verificación en tabla `profiles`
4. Validación de rol `admin`
5. Almacenamiento de sesión JWT
6. Redirección a admin o index

2. Gestión de Recursos

Componentes **Resource Card** (Tarjeta de Recurso)

```
<div class="result-card">
  <div class="card-icon">[ICONO]</div>
  <span class="card-badge">[ÁREA]</span>
  <h4>[TÍTULO]</h4>
  <p>[DESCRIPCIÓN]</p>
  <a href="[ENLACE DRIVE]">Abrir enlace</a>
</div>
```

Search Bar (Barra de Búsqueda)

```
<input
  type="text"
  id="searchInput"
  placeholder="Buscar recursos..."
  oninput="buscarRecursos()"
/>
```

Area Filter (Filtro por Área)

```
<div class="area-card" onclick="filtrarPorArea('FACTURACIÓN')">
  <div class="area-icon">[ICONO]</div>
  <h4>FACTURACIÓN</h4>
  <p>[EMAIL]</p>
</div>
```

Funciones Principales **Cargar Recursos**

```
async function cargarRecursosDesdeSupabase() {
  const { data, error } = await supabase
    .from('recursos')
    .select(`*, areas:area_id(*)`)
    .eq('is_active', true)
    .order('created_at', { ascending: false })

  recursosDatabase = data
  displayResults()
}
```

Búsqueda

```
function buscarRecursos() {
  const searchTerm = document.getElementById('searchInput').value
  const filtrados = recursosDatabase.filter(r =>
    r.titulo.includes(searchTerm) ||
    r.descripcion.includes(searchTerm) ||
    r.palabras_clave.some(p => p.includes(searchTerm))
  )
```

```
  displayResults(filtrados)
}
```

Paginación

```
function mostrarPaginacion() {
  const totalPaginas = Math.ceil(recursos.length / RECURSOS_POR_PAGINA)
  // Generar botones de navegación
}
```

3. Panel de Administración

Componentes **Stats Cards** (Tarjetas de Estadísticas)

```
<div class="stats-card">
  <div class="stat-icon">[ICONO]</div>
  <p>Total Recursos</p>
  <h3 id="totalRecursos">0</h3>
</div>
```

Resources Table (Tabla de Recursos)

```
<table class="recursos-table">
  <thead>
    <tr>
      <th>Área</th>
      <th>Título</th>
      <th>Descripción</th>
      <th>Estado</th>
      <th>Acciones</th>
    </tr>
  </thead>
  <tbody id="tablaRecursos"></tbody>
</table>
```

Modal Form (Formulario Modal)

```
<div id="modalRecurso" class="modal">
  <div class="modal-content">
    <form id="formRecurso" onsubmit="guardarRecurso(event)">
      <!-- Campos del formulario -->
    </form>
  </div>
</div>
```

Funciones Admin **CRUD de Recursos**

```
// CREATE
async function guardarRecurso(event) {
  const { data, error } = await supabase
    .from('recursos')
```

```
    .insert([nuevoRecurso])
}

// READ
async function cargarRecursos() {
  const { data } = await supabase
    .from('recursos')
    .select('*, areas:area_id(*)')
}

// UPDATE
async function editarRecurso(id) {
  const { error } = await supabase
    .from('recursos')
    .update(recursoActualizado)
    .eq('id', id)
}

// DELETE
async function eliminarRecurso(id) {
  const { error } = await supabase
    .from('recursos')
    .delete()
    .eq('id', id)
}
```

4. Gestión de Áreas

Componentes **Areas Table** (Tabla de Áreas)

```
<table id="areasTable">
  <thead>
    <tr>
      <th>Orden</th>
      <th>Nombre</th>
      <th>Correo</th>
      <th>Estado</th>
      <th>Acciones</th>
    </tr>
  </thead>
  <tbody id="tablaAreas"></tbody>
</table>
```

Area Modal (Modal de Área)

```
<div id="modalArea" class="modal">
  <form id="formArea" onsubmit="guardarArea(event)">
    <input type="text" id="inputNombre" required />
    <input type="email" id="inputCorreo" />
```

```
    <input type="text" id="inputIcono" />
    <!-- más campos -->
  </form>
</div>
```

Funciones de Áreas CRUD Completo

```
// Cargar áreas
async function cargarAreas() {
  const { data } = await supabase
    .from('areas')
    .select('*')
    .order('orden', { ascending: true })

  areasGlobales = data
  mostrarAreas()
}

// Guardar área
async function guardarArea(event) {
  if (areaEditando) {
    // UPDATE
    await supabase.from('areas').update(area).eq('id', areaEditando)
  } else {
    // INSERT
    await supabase.from('areas').insert([area])
  }
}

// Eliminar área
async function eliminarArea(id) {
  await supabase.from('areas').delete().eq('id', id)
}
```

Búsqueda y Paginación

```
function buscarAreas() {
  const term = document.getElementById('searchAreaInput').value
  areasFiltradas = areasGlobales.filter(a =>
    a.nombre.includes(term) || a.correo.includes(term)
  )
  mostrarAreas()
}

function mostrarPaginacionAreas() {
  const totalPaginas = Math.ceil(areasFiltradas.length / AREAS_POR_PAGINA)
  // Renderizar navegación
}
```

5. Sistema de Notificaciones

Toast Notifications Componente Toast

```
function showToast(mensaje, tipo = 'success') {
  const toast = document.createElement('div')
  toast.style.cssText = `
    background: ${tipo === 'success' ? '#10b981' : '#ef4444'};
    color: white;
    padding: 16px 24px;
    border-radius: 8px;
    animation: slideIn 0.3s ease-out;
  `
  toast.textContent = mensaje

  document.body.appendChild(toast)

  setTimeout(() => {
    toast.remove()
  }, 3000)
}
```

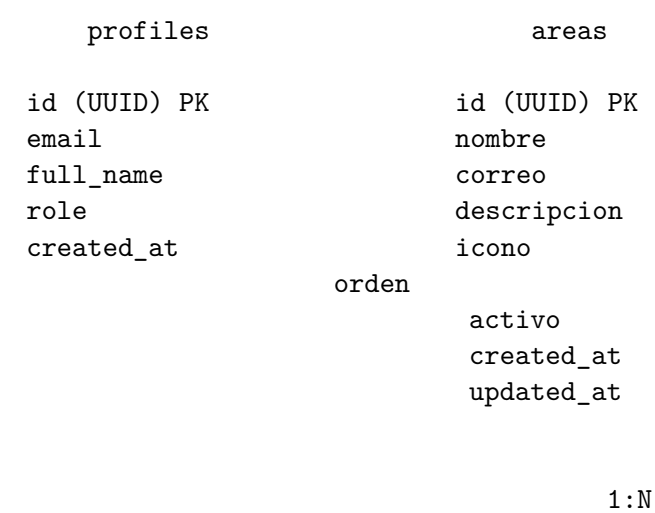
Uso

```
showToast('Recurso guardado exitosamente', 'success')
showToast('Error al eliminar área', 'error')
showToast('Cargando datos...', 'info')
```

BASE DE DATOS

Esquema de Base de Datos

Diagrama ER (Entity-Relationship)



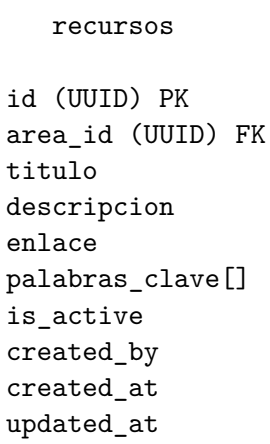


Tabla: `recursos`

Descripción: Almacena todos los recursos (enlaces de Google Drive) del sistema.

Campos:

Campo	Tipo	Restricciones	Descripción
`id`	UUID	PK, NOT NULL	Identificador único
`area_id`	UUID	FK, NULL	Relación con tabla `areas`
`titulo`	VARCHAR(255)	NOT NULL	Nombre del recurso
`descripcion`	TEXT	NULL	Descripción detallada
`enlace`	TEXT	NOT NULL	URL de Google Drive
`palabras_clave`	TEXT[]	NULL	Array de palabras clave
`is_active`	BOOLEAN	DEFAULT true	Estado activo/inactivo
`created_by`	UUID	NULL	Usuario creador (profiles.id)
`created_at`	TIMESTAMPTZ	DEFAULT now()	Fecha de creación
`updated_at`	TIMESTAMPTZ	DEFAULT now()	Última actualización

Índices:

```
CREATE INDEX idx_recursos_area_id ON recursos(area_id);
CREATE INDEX idx_recursos_is_active ON recursos(is_active);
CREATE INDEX idx_recursos_created_at ON recursos(created_at DESC);
```

Foreign Keys:

```
ALTER TABLE recursos
ADD CONSTRAINT fk_recursos_areas
FOREIGN KEY (area_id) REFERENCES areas(id)
```

```
ON DELETE SET NULL
ON UPDATE CASCADE;
```

RLS Policies:

```
-- Lectura pública de recursos activos
CREATE POLICY "Public read active recursos" ON recursos
FOR SELECT USING (is_active = true);

-- Admin puede hacer todo
CREATE POLICY "Admin full access recursos" ON recursos
FOR ALL USING (
  EXISTS (
    SELECT 1 FROM profiles
    WHERE profiles.id = auth.uid()
    AND profiles.role = 'admin'
  )
);
```

Tabla: `areas`

Descripción: Define las 8 áreas operativas con su información de contacto.

Campos:

Campo	Tipo	Restricciones	Descripción
`id`	UUID	PK, NOT NULL	Identificador único
`nombre`	VARCHAR(100)	NOT NULL, UNIQUE	Nombre del área
`correo`	VARCHAR(255)	NULL	Email de contacto
`descripcion`	TEXT	NULL	Descripción del área
`icono`	TEXT	NULL	Emoji o ícono representativo
`orden`	INTEGER	NOT NULL	Orden de visualización (1-8)
`activo`	BOOLEAN	DEFAULT true	Estado activo/inactivo
`created_at`	TIMESTAMPTZ	DEFAULT now()	Fecha de creación
`updated_at`	TIMESTAMPTZ	DEFAULT now()	Última actualización

Índices:

```
CREATE UNIQUE INDEX idx_areas_nombre ON areas(nombre);
CREATE INDEX idx_areas_orden ON areas(orden);
CREATE INDEX idx_areas_activo ON areas(activo);
```

RLS Policies:

```
-- Lectura pública de áreas activas
CREATE POLICY "Public read active areas" ON areas
FOR SELECT USING (activo = true);
```

```sql
-- Admin puede hacer todo
CREATE POLICY "Admin full access areas" ON areas
FOR ALL USING (
  EXISTS (
    SELECT 1 FROM profiles
    WHERE profiles.id = auth.uid()
    AND profiles.role = 'admin'
  )
);
```

Datos Iniciales (8 áreas):

```sql
INSERT INTO areas (nombre, correo, icono, orden, activo) VALUES
('ATENCIÓN - FRONT', 'atencion.front@certus.edu.pe', ' ', 1, true),
('ATENCIÓN - CANALES', 'atencion.canales@certus.edu.pe', ' ', 2, true),
('CRÉDITO Y COBRANZAS', 'credito.cobranzas@certus.edu.pe', ' ', 3, true),
('FACTURACIÓN', 'facturacion@certus.edu.pe', ' ', 4, true),
('RR.AA', 'recursos.academicos@certus.edu.pe', ' ', 5, true),
('PEC', 'pec@certus.edu.pe', ' ', 6, true),
('REINGRESO', 'reingreso@certus.edu.pe', ' ', 7, true),
('OPERACIONES', 'operaciones@certus.edu.pe', ' ', 8, true);
```

Tabla: `profiles`

Descripción: Extiende la tabla de usuarios de Supabase Auth con información adicional.

Campos:

Campo	Tipo	Restricciones	Descripción
`id`	UUID	PK, FK	Referencia a auth.users(id)
`email`	VARCHAR(255)	NOT NULL	Email del usuario
`full_name`	VARCHAR(255)	NULL	Nombre completo
`role`	VARCHAR(20)	DEFAULT ‘user’	Rol: ‘admin’ o ‘user’
`created_at`	TIMESTAMPTZ	DEFAULT now()	Fecha de registro

RLS Policies:

```sql
-- Usuarios pueden ver su propio perfil
CREATE POLICY "Users can view own profile" ON profiles
FOR SELECT USING (auth.uid() = id);

-- Admin puede ver todos los perfiles
CREATE POLICY "Admin can view all profiles" ON profiles
FOR SELECT USING (
  EXISTS (
    SELECT 1 FROM profiles
    WHERE profiles.id = auth.uid()
    AND profiles.role = 'admin'
```

```
  )
);
```

Consultas Comunes

Obtener recursos con información de área:

```sql
SELECT
  r.*,
  a.nombre as area_nombre,
  a.correo as area_correo,
  a.icono as area_icono
FROM recursos r
LEFT JOIN areas a ON r.area_id = a.id
WHERE r.is_active = true
ORDER BY r.created_at DESC;
```

Estadísticas del sistema:

```sql
SELECT
  COUNT(*) as total_recursos,
  COUNT(*) FILTER (WHERE is_active = true) as recursos_activos,
  COUNT(DISTINCT area_id) as areas_diferentes
FROM recursos;
```

Recursos por área:

```sql
SELECT
  a.nombre,
  COUNT(r.id) as cantidad_recursos
FROM areas a
LEFT JOIN recursos r ON a.id = r.area_id AND r.is_active = true
GROUP BY a.id, a.nombre
ORDER BY a.orden;
```

AUTENTICACIÓN Y SEGURIDAD

Sistema de Autenticación

Tecnología: Supabase Auth Características: - Autenticación basada en JWT (JSON Web Tokens) - Sesiones persistentes con refresh tokens - Expiración automática de tokens - Protección CSRF

```javascript

// 1. Usuario envía credenciales
const { data, error } = await supabase.auth.signInWithPassword({
  email: email,
  password: password
})
```

```
// 2. Supabase valida y retorna session
if (data.session) {
  // 3. Verificar rol en tabla profiles
  const { data: profile } = await supabase
    .from('profiles')
    .select('role')
    .eq('id', data.user.id)
    .single()

  // 4. Redirigir según rol
  if (profile.role === 'admin') {
    window.location.href = '/admin.html'
  } else {
    window.location.href = '/index.html'
  }
}
```

Flujo de Login

Protección de Rutas Auth Guard (`auth.js`):

```
async function checkAuth() {
  const { data: { session } } = await supabase.auth.getSession()

  if (!session) {
    // No hay sesión, redirigir a login
    window.location.href = '/login.html'
    return false
  }

  // Verificar rol admin
  const { data: profile } = await supabase
    .from('profiles')
    .select('role')
    .eq('id', session.user.id)
    .single()

  if (profile.role !== 'admin') {
    // No es admin, redirigir a index
    window.location.href = '/index.html'
    return false
  }

  return true
}

// Ejecutar al cargar admin.html
```

```
checkAuth()
```

```
async function cerrarSesion() {
  await supabase.auth.signOut()
  window.location.href = '/login.html'
}
```

Cierre de Sesión

Row Level Security (RLS)

Concepto RLS es una característica de PostgreSQL que permite definir políticas de acceso a nivel de fila, controlando qué usuarios pueden ver o modificar qué datos.

Políticas Implementadas **Tabla recursos:**

```
-- Política 1: Lectura pública de recursos activos
CREATE POLICY "Public read active recursos"
ON recursos FOR SELECT
USING (is_active = true);

-- Política 2: Admin tiene acceso total
CREATE POLICY "Admin full access recursos"
ON recursos FOR ALL
USING (
  EXISTS (
    SELECT 1 FROM profiles
    WHERE profiles.id = auth.uid()
    AND profiles.role = 'admin'
  )
);
```

Tabla areas:

```
-- Política 1: Lectura pública de áreas activas
CREATE POLICY "Public read active areas"
ON areas FOR SELECT
USING (activo = true);

-- Política 2: Admin tiene acceso total
CREATE POLICY "Admin full access areas"
ON areas FOR ALL
USING (
  EXISTS (
    SELECT 1 FROM profiles
    WHERE profiles.id = auth.uid()
    AND profiles.role = 'admin'
```

```
  )
);
```

Tabla `profiles`:

```
-- Política 1: Usuarios ven su propio perfil
CREATE POLICY "Users can view own profile"
ON profiles FOR SELECT
USING (auth.uid() = id);

-- Política 2: Admin ve todos los perfiles
CREATE POLICY "Admin can view all profiles"
ON profiles FOR SELECT
USING (
  EXISTS (
    SELECT 1 FROM profiles
    WHERE profiles.id = auth.uid()
    AND profiles.role = 'admin'
  )
);
```

Seguridad del Frontend

Headers HTTP Configuración en `netlify.toml`:

```
[[headers]]
  for = "/*"
  [headers.values]
    # HTTPS estricto
    Strict-Transport-Security = "max-age=31536000; includeSubDomains; preload"

    # Prevenir clickjacking
    X-Frame-Options = "SAMEORIGIN"

    # Protección XSS
    X-XSS-Protection = "1; mode=block"

    # Prevenir MIME sniffing
    X-Content-Type-Options = "nosniff"

    # Política de referrer
    Referrer-Policy = "strict-origin-when-cross-origin"

    # Permisos restrictivos
    Permissions-Policy = "geolocation=(), microphone=(), camera=()"
```

Validación de Datos En formularios:

```javascript
function validarRecurso(recurso) {
  // Validar título
  if (!recurso.titulo || recurso.titulo.trim().length < 3) {
    throw new Error('El título debe tener al menos 3 caracteres')
  }

  // Validar enlace
  if (!recurso.enlace || !recurso.enlace.startsWith('https://drive.google.com')) {
    throw new Error('El enlace debe ser de Google Drive')
  }

  // Validar área
  if (!recurso.area_id) {
    throw new Error('Debe seleccionar un área')
  }

  return true
}
```

```javascript
function sanitizeHTML(html) {
  const temp = document.createElement('div')
  temp.textContent = html
  return temp.innerHTML
}

// Uso en renderizado
element.textContent = sanitizeHTML(userInput)
```

Sanitización de HTML

Mejores Prácticas Implementadas

Contraseñas: Nunca almacenadas en código (manejadas por Supabase)
HTTPS: Forzado en producción
JWT: Tokens con expiración corta
RLS: Control de acceso a nivel de base de datos
Validación: En frontend y backend (RLS)
Sanitización: Prevención de XSS
Headers: Configurados para máxima seguridad

FLUJOS DE TRABAJO

Flujo 1: Usuario Busca un Recurso

```
1. Usuario accede a index.html
```

```
    → 2. JavaScript carga recursos desde BD

         → SELECT * FROM recursos
           WHERE is_active = true
           LEFT JOIN areas ON recursos.area_id = areas.id

    → 3. Se muestran todas las áreas en carrusel

    → 4. Usuario puede:
         → a) Buscar por texto
              → Filtrado en JavaScript por título/descripción/palabras clave

         → b) Filtrar por área
              → Filtrado por area_id

    → 5. Resultados se muestran en grid
         → Con paginación (12 recursos por página)

    → 6. Usuario hace clic en "Abrir enlace"
         → Redirección a Google Drive en nueva pestaña
```

Flujo 2: Admin Agrega un Recurso

```
1. Admin ingresa a admin.html

    → 2. checkAuth() verifica sesión

         →  No autenticado → Redirigir a login.html

         →  Autenticado y admin → Continuar

    → 3. Admin hace clic en "Nuevo Recurso"
         → Se abre modal con formulario

    → 4. Admin completa formulario:
         → Selecciona área (dropdown desde BD)
         → Ingresa título
         → Ingresa descripción
         → Pega enlace de Drive
         → Agrega palabras clave (separadas por comas)

    → 5. Admin hace clic en "Guardar"

         → Validación frontend

         → INSERT en tabla recursos

              → RLS verifica rol admin
```

```
        → Éxito → Toast "Recurso guardado"
            → Recargar tabla

        → Error → Toast "Error al guardar"

    → 6. Tabla se actualiza con nuevo recurso
```

Flujo 3: Admin Gestiona Áreas

```
1. Admin en admin.html → Pestaña "Gestión de Áreas"

    → 2. Carga áreas desde BD
        → SELECT * FROM areas ORDER BY orden

    → 3. Admin puede:

        → a) Crear nueva área
            → Abrir modal
            → Completar formulario
            → INSERT en tabla areas

        → b) Editar área existente
            → Cargar datos en modal
            → Modificar campos
            → UPDATE en tabla areas

        → c) Activar/Desactivar área
            → UPDATE areas SET activo = !activo

        → d) Eliminar área
            → Confirmación en modal
            → DELETE FROM areas
                → Los recursos con area_id se ponen en NULL (ON DELETE SET NULL)

    → 4. Tabla se actualiza con cambios
```

Flujo 4: Autenticación de Usuario

```
1. Usuario accede a cualquier página

    → 2. checkAuth() se ejecuta

        → ¿Hay sesión activa?

            → No → Redirigir a login.html

            → Sí → Verificar rol
```

```
                → SELECT role FROM profiles WHERE id = auth.uid()

                → role = 'admin' → Permitir acceso a admin.html

                → role = 'user' → Solo index.html

        → Token expirado
              → Refresh automático o redirigir a login

    → 3. Usuario navega normalmente
```

Flujo 5: Búsqueda Avanzada

```
1. Usuario en index.html

    → 2. Escribe en barra de búsqueda

        → oninput="buscarRecursos()"

              → Obtener texto de búsqueda

              → Filtrar recursosDatabase:
                    → Por título
                    → Por descripción
                    → Por palabras_clave

              → Actualizar grid con resultados

    → 3. Selecciona área en carrusel

        → onclick="filtrarPorArea(areaId)"

              → Filtrar por area_id

              → Actualizar grid

    → 4. Combina búsqueda + filtro de área
        → Filtrado acumulativo

    → 5. Navega por páginas de resultados
         → Paginación dinámica
```

Flujo 6: Carga Inicial de la Aplicación

```
1. Usuario accede a index.html

    → 2. HTML carga completamente
```

```
      → DOMContentLoaded event

→ 3. Se cargan scripts:
      → Supabase Client (CDN)
      → config.js (inicializa cliente)
      → auth.js (verifica sesión - opcional para index)
      → script.js (lógica principal)

→ 4. script.js ejecuta:

      → Mostrar skeleton loader

      → cargarAreasDesdeSupabase()
            → SELECT * FROM areas WHERE activo = true

      → cargarRecursosDesdeSupabase()
            → SELECT recursos.*, areas.*
               FROM recursos
               LEFT JOIN areas ON recursos.area_id = areas.id
               WHERE recursos.is_active = true

      → Ocultar skeleton loader

      → displayAreasCarousel()
            → Renderizar tarjetas de áreas

      → displayResults()
             → Renderizar grid de recursos

→ 5. Interfaz lista para interacción
```

Este es el primer bloque de la documentación completa. ¿Quieres que continúe con las siguientes secciones (Guía de Usuario, Configuración y Deployment, Mantenimiento, etc.)?