**Brief técnico — Plataforma académica SDCS (Robótica + STEAM)**

**Fecha:** 28-08-2025  
**Alcance inicial (MVP):** Portal académico 1°–11° con autenticación por código de grupo, espacios por grado → 3 trimestres → (1) Información pedagógica, (2) Clases 1–15, (3) Proyecto del periodo. Dashboard de docente para administrar contenidos (videos, PDFs, textos, imágenes, audios). Hosting estático (Git) + Supabase (Auth, DB, Storage, Realtime).

**1) Objetivo**

Centralizar los procesos académicos de Robótica + STEAM del Seminario Diocesano Cristo Sacerdote para grados 1° a 11°, entregando:

* Acceso por **código de 6 dígitos** por grado (un solo código compartido por grupo).
* Estructura de **3 trimestres** con: metas de comprensión e indicadores, clases 1–15 y proyecto del periodo.
* **Dashboard docente** para crear/editar contenidos multimedia y texto.
* Entrega en un **sitio estático** (GitHub Pages/Netlify/Vercel) con datos en Supabase.

**2) Requisitos funcionales (MVP)**

1. **Autenticación por grupo**: el estudiante escribe un código de 6 dígitos y entra directo al espacio de su grado. Un código distinto por cada grado (1°–11°). Un código adicional para **administrador**.
2. **Estructura por grado**: 3 trimestres con:
   * (1) Información pedagógica: *metas de comprensión* + *indicadores de desempeño*.
   * (2) Clases 1–15 (título, descripción breve y materiales).
   * (3) Proyecto de periodo (descripción, recursos).
3. **Tipos de contenido** administrables:
   * **Videos**: almacenar **links** (YouTube/Vimeo u .mp4 de Storage) y **embeber** en la misma página.
   * **PDFs** (en Storage).
   * **Textos** (título + cuerpo, ej. “Competencias”).
   * **Imágenes** (Storage).
   * **Audios** (Storage).
4. **Dashboard docente** (solo admin): crear/editar/borrar términos, clases y materiales; **cargar archivos** a Storage en rutas por grado.
5. **Realtime**: que los cambios del docente se reflejen sin recargar (escucha en frontend).

**3) Requisitos no funcionales**

* **Simplicidad**: HTML + CSS + JS vanilla.
* **Seguridad de datos**: RLS en Supabase.
* **Rendimiento**: carga diferida de materiales por trimestre/clase.
* **Mantenibilidad**: esquema de BD claro y políticas documentadas.
* **Accesibilidad**: etiquetas ARIA básicas, contraste adecuado.

**4) Arquitectura**

**Frontend (estático):**

* index.html → login por código.
* grado.html → vista de contenidos del estudiante (única página dinámica por grado vía querystring/estado).
* admin.html → dashboard docente.
* assets/ (CSS) y js/ (módulos).

**Backend as-a-service: Supabase**

* **Auth**: cuentas por **grupo** (un usuario por grado) y un usuario **admin**. La **contraseña** de cada cuenta = **código de 6 dígitos**.
  + Ejemplo de emails: g1@sdcs.local … g11@sdcs.local y admin@sdcs.local.
  + Los estudiantes ingresan **solo el código**; el frontend intentará loguearse contra cuentas g1..g11 + admin hasta que una coincida.
* **Database**: tablas para perfiles, trimestres, clases y materiales.
* **Storage**: buckets img, video, pdf, audio con **carpetas por grado**:
  + img/1/..., video/1/..., pdf/1/..., audio/1/... (para 1°), etc.
  + *Nota:* Esto respeta los nombres de buckets solicitados y usa **carpetas** por grado; es la forma recomendada en Supabase (un bucket por tipo y rutas internas por grado).
* **Realtime**: suscripción a tablas materials, classes, terms para reflejar cambios en tiempo real.

**5) Modelo de datos (SQL — esquema)**

Ejecutar en **SQL Editor** de Supabase. Ajusta comentarios según necesites.

-- 1) Tipos y perfiles

grade int check (grade between 1 and 11)

);

alter table public.profiles enable row level security;

-- 2) Trimestres por grado

create table public.terms (

id bigserial primary key,

grade int not null check (grade between 1 and 11),

term\_no int not null check (term\_no between 1 and 3),

goals text, -- metas de comprensión

indicators text, -- indicadores de desempeño

unique (grade, term\_no)

);

alter table public.terms enable row level security;

-- 3) Clases por grado y trimestre

create table public.classes (

id bigserial primary key,

grade int not null check (grade between 1 and 11),

term\_no int not null check (term\_no between 1 and 3),

class\_no int not null check (class\_no between 1 and 15),

title text,

description text,

unique (grade, term\_no, class\_no)

);

alter table public.classes enable row level security;

-- 4) Materiales (videos, pdfs, imágenes, audios, textos)

create type material\_kind as enum ('video','pdf','image','audio','text');

create type material\_scope as enum ('pedagogical','class','project');

create table public.materials (

id bigserial primary key,

grade int not null check (grade between 1 and 11),

term\_no int not null check (term\_no between 1 and 3),

class\_no int, -- null cuando scope = 'pedagogical' o 'project'

scope material\_scope not null,

kind material\_kind not null,

title text,

body text, -- texto libre (competencias, etc.)

link\_url text, -- para videos externos (YouTube/Vimeo) o recursos externos

storage\_bucket text, -- 'img' | 'video' | 'pdf' | 'audio'

storage\_path text, -- ej. '1/unidad1/imagen1.png'

created\_by uuid references auth.users(id),

created\_at timestamp with time zone default now()

);

alter table public.materials enable row level security;

-- Índices sugeridos

create index on public.classes (grade, term\_no);

create index on public.materials (grade, term\_no, class\_no, scope, kind);

**6) Políticas RLS (SQL — seguridad)**

**Idea:** Solo **admin** puede escribir. Estudiantes (cuentas por grado) **leen** únicamente su grado.

-- PROFILES

create policy "read own profile" on profiles

for select using (auth.uid() = id);

create policy "admins manage profiles" on profiles

for all to authenticated using (

exists(select 1 from profiles p where p.id = auth.uid() and p.role = 'admin')

) with check (

exists(select 1 from profiles p where p.id = auth.uid() and p.role = 'admin')

);

-- TERMS (solo admin escribe; todos los autenticados leen su grado)

create policy "terms read by grade" on terms

for select to authenticated using (

exists(

select 1 from profiles p

where p.id = auth.uid()

and (p.role = 'admin' or p.grade = terms.grade)

)

);

create policy "terms admin write" on terms

for all to authenticated using (

exists(select 1 from profiles p where p.id = auth.uid() and p.role = 'admin')

) with check (

exists(select 1 from profiles p where p.id = auth.uid() and p.role = 'admin')

);

-- CLASSES

create policy "classes read by grade" on classes

for select to authenticated using (

exists(

select 1 from profiles p

where p.id = auth.uid()

and (p.role = 'admin' or p.grade = classes.grade)

)

);

create policy "classes admin write" on classes

for all to authenticated using (

exists(select 1 from profiles p where p.id = auth.uid() and p.role = 'admin')

) with check (

exists(select 1 from profiles p where p.id = auth.uid() and p.role = 'admin')

);

-- MATERIALS

create policy "materials read by grade" on materials

for select to authenticated using (

exists(

select 1 from profiles p

where p.id = auth.uid()

and (p.role = 'admin' or p.grade = materials.grade)

)

);

create policy "materials admin write" on materials

for all to authenticated using (

exists(select 1 from profiles p where p.id = auth.uid() and p.role = 'admin')

) with check (

exists(select 1 from profiles p where p.id = auth.uid() and p.role = 'admin')

);

**Storage (políticas)**

* Buckets: img, video, pdf, audio.
* **MVP simple:** lectura pública (para evitar fricción inicial) y **escritura solo admin**.
* **V1** (más seguro): lectura solo autenticados; opcionalmente por grado filtrando storage.objects.name con prefijo '{grade}/'.

**MVP (público READ, admin WRITE):**

-- Lectura pública

create policy "public read" on storage.objects

for select using (bucket\_id in ('img','video','pdf','audio'));

-- Solo admin sube/edita/borra

create policy "admin write" on storage.objects

for all to authenticated using (

exists(select 1 from profiles p where p.id = auth.uid() and p.role = 'admin')

) with check (

exists(select 1 from profiles p where p.id = auth.uid() and p.role = 'admin')

);

**V1 (lectura por grado; opcional):**

-- Lectura por grado: compara el prefijo de la ruta con el grade del perfil

create policy "read by grade" on storage.objects

for select to authenticated using (

bucket\_id in ('img','video','pdf','audio') and (

exists(

select 1 from profiles p

where p.id = auth.uid()

and (

p.role = 'admin' or

split\_part(storage.objects.name, '/', 1) = p.grade::text

)

)

)

);

**7) Flujo de autenticación por código de grupo (sin backend propio)**

**Estrategia recomendada (MVP):**

* Crear **1 usuario por grado** en Supabase Auth y **1 usuario admin**.
* **Password = código de 6 dígitos**.
* Desde index.html, el usuario escribe el código → el frontend intenta signInWithPassword primero con admin@sdcs.local, luego con g1@sdcs.local…g11@sdcs.local hasta acertar.
* Tras login, se consulta profiles para saber role y grade → redirección a admin.html (admin) o grado.html?g=GRADE (estudiante).

**Pasos para preparar cuentas en Supabase:**

1. **Authentication → Users → Add user** (manual).
2. Emails sugeridos: g1@sdcs.local … g11@sdcs.local, admin@sdcs.local.
3. Password = el **código** (6 dígitos) correspondiente.
4. Crear sus **profiles**: asignar role y grade (ver SQL abajo).

-- ejemplo: vincular perfiles a usuarios ya creados

-- reemplaza los UUIDs por los reales (desde la tabla auth.users)

insert into profiles (id, role, grade) values

('<uuid-admin>', 'admin', null),

('<uuid-g1>', 'student', 1),

('<uuid-g2>', 'student', 2);

-- ... hasta g11

**Nota:** Este enfoque cumple el requisito de “código por grupo” con la menor complejidad técnica. A futuro, si necesitas **códigos rotativos** sin cambiar password, podemos migrar a **Edge Functions** con verificación de códigos y **JWT con claims de grado**.

**8) Estructura de navegación (IA)**

* **Login (index)** → input de 6 dígitos.
* **Vista de grado (grado.html)**
  + Selector de **Trimestre 1 / 2 / 3** (tabs).
  + Secciones: *Información pedagógica*, *Clases 1–15*, *Proyecto del periodo*.
  + Cada clase: título, descripción, lista de materiales (embed video, visor PDF inline, imágenes en galería, audio en reproductor).
* **Dashboard (admin.html)**
  + Form de **Selección**: Grado, Trimestre, Clase (1–15 o “—” para info pedagógica/proyecto).
  + Pestañas de **Tipos**: Video (link), PDF (upload), Imagen (upload), Audio (upload), Texto (editor simple).
  + Botones: *Publicar*, *Editar*, *Eliminar*.
  + Lista de materiales existentes con acciones.
  + Realtime: refresh automático.

**9) Scaffolding Frontend (HTML/JS/CSS)**

**Supabase client**:

<!-- supabase-js v2 desde CDN -->

<script type="module">

import { createClient } from 'https://esm.sh/@supabase/supabase-js@2';

window.supabase = createClient(

'https://trfiejowmfgzudhsdpdx.supabase.co',

'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InRyZmllam93bWZnenVkaHNkcGR4Iiwicm9sZSI6ImFub24iLCJpYXQiOjE3NTY0MjM2MjksImV4cCI6MjA3MTk5OTYyOX0.SCNuSO9uKtaKfGBSi3WMIpvNFmsE66aRROJEgTUrx34'

);

</script>

**index.html (login por código)**:

<!doctype html>

<html lang="es">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>SDCS — Ingreso</title>

<link rel="stylesheet" href="/assets/styles.css" />

</head>

<body>

<main class="login">

<h1>Plataforma SDCS</h1>

<form id="login-form">

<label>Código de 6 dígitos</label>

<input id="code" inputmode="numeric" pattern="\\d{6}" maxlength="6" required />

<button type="submit">Entrar</button>

<p id="msg"></p>

</form>

</main>

<script type="module">

import { createClient } from 'https://esm.sh/@supabase/supabase-js@2';

const supabase = createClient(

'https://trfiejowmfgzudhsdpdx.supabase.co',

'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InRyZmllam93bWZnenVkaHNkcGR4Iiwicm9sZSI6ImFub24iLCJpYXQiOjE3NTY0MjM2MjksImV4cCI6MjA3MTk5OTYyOX0.SCNuSO9uKtaKfGBSi3WMIpvNFmsE66aRROJEgTUrx34'

);

const form = document.querySelector('#login-form');

const msg = document.querySelector('#msg');

form.addEventListener('submit', async (e) => {

e.preventDefault();

msg.textContent = 'Verificando…';

const code = document.querySelector('#code').value.trim();

const emails = ['admin@sdcs.local'];

for (let i=1;i<=11;i++) emails.push(`g${i}@sdcs.local`);

let ok = false;

for (const email of emails) {

const { data, error } = await supabase.auth.signInWithPassword({ email, password: code });

if (!error) { ok = true; break; }

}

if (!ok) { msg.textContent = 'Código inválido.'; return; }

const { data: { user } } = await supabase.auth.getUser();

const { data: profile } = await supabase.from('profiles').select('role, grade').eq('id', user.id).single();

if (!profile) { msg.textContent = 'Perfil no encontrado.'; return; }

if (profile.role === 'admin') location.href = '/admin.html';

else location.href = `/grado.html?g=${profile.grade}`;

});

</script>

</body>

</html>

**grado.html (vista estudiante, lectura)**:

<!doctype html>

// Proyecto del periodo

const { data: proj } = await supabase.from('materials').select('\*').match({ grade, term\_no: currentTerm, scope:'project' });

proyecto.innerHTML = '<h2>Proyecto del periodo</h2>' + (proj||[]).map(renderMaterial).join('') || '<p>Sin proyecto.</p>';

}

function renderMaterial(m){

if (m.kind === 'video') {

// Si link\_url es YouTube/Vimeo, embeber via iframe; si es mp4 de Storage, usar <video>

if (m.link\_url?.includes('youtube.com') || m.link\_url?.includes('youtu.be')) {

const id = (m.link\_url.split('v=')[1]||'').split('&')[0] || m.link\_url.split('/').pop();

return `<div class="video"><iframe src="https://www.youtube.com/embed/${id}" allowfullscreen loading="lazy"></iframe></div>`;

}

if (m.link\_url?.includes('vimeo.com')) {

const id = m.link\_url.split('/').pop();

return `<div class="video"><iframe src="https://player.vimeo.com/video/${id}" allowfullscreen loading="lazy"></iframe></div>`;

}

// MP4 en Storage

const url = m.link\_url || storagePublicURL(m);

return `<video controls preload="metadata" src="${url}"></video>`;

}

if (m.kind === 'pdf') {

const url = m.link\_url || storagePublicURL(m);

return `<div class="pdf"><iframe src="${url}#view=fitH" loading="lazy"></iframe></div>`;

}

if (m.kind === 'image') {

const url = m.link\_url || storagePublicURL(m);

return `<img src="${url}" alt="${m.title||''}" loading="lazy" />`;

}

if (m.kind === 'audio') {

const url = m.link\_url || storagePublicURL(m);

return `<audio controls src="${url}"></audio>`;

}

// texto

return `<div class="texto"><h4>${m.title||''}</h4><p>${(m.body||'').replaceAll('\n','<br>')}</p></div>`;

}

function storagePublicURL(m){

const { data } = supabase.storage.from(m.storage\_bucket).getPublicUrl(m.storage\_path);

return data.publicUrl;

}

document.querySelectorAll('nav [data-t]').forEach(btn=>{

btn.addEventListener('click', ()=>{ currentTerm = Number(btn.dataset.t); loadTerm(); });

});

document.querySelector('#logout').addEventListener('click', async ()=>{

await supabase.auth.signOut(); location.href = '/';

});

loadTerm();

// Realtime: refrescar en cambios

supabase.channel('realtime:materials')

.on('postgres\_changes', { event: '\*', schema: 'public', table: 'materials', filter: `grade=eq.${grade}` }, loadTerm)

.subscribe();

</script>

</body>

</html>

**admin.html (dashboard docente — escritura)** *(extracto base)*:

<!doctype html>

<textarea id="body" placeholder="Texto / descripción"></textarea>

<input id="link\_url" placeholder="Link de video (YouTube/Vimeo) o URL pública">

<input id="file" type="file" />

<button id="publish">Publicar</button>

</div>

<div id="list"></div>

</section>

<script type="module">

import { createClient } from 'https://esm.sh/@supabase/supabase-js@2';

const supabase = createClient('https://trfiejowmfgzudhsdpdx.supabase.co','eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InRyZmllam93bWZnenVkaHNkcGR4Iiwicm9sZSI6ImFub24iLCJpYXQiOjE3NTY0MjM2MjksImV4cCI6MjA3MTk5OTYyOX0.SCNuSO9uKtaKfGBSi3WMIpvNFmsE66aRROJEgTUrx34');

// Guardacalles: permitir solo admin

(async ()=>{

const { data: { user } } = await supabase.auth.getUser();

if (!user) return location.href = '/';

const { data: profile } = await supabase.from('profiles').select('role').eq('id', user.id).single();

if (!profile || profile.role !== 'admin') return location.href = '/';

})();

const el = id => document.getElementById(id);

el('logout').onclick = async ()=>{ await supabase.auth.signOut(); location.href='/'; };

el('publish').onclick = async ()=>{

const grade = Number(el('grade').value);

const term\_no = Number(el('term').value);

const class\_no = el('class\_no').value ? Number(el('class\_no').value) : null;

const scope = el('scope').value;

const kind = el('kind').value;

const title = el('title').value; const body = el('body').value; const link\_url = el('link\_url').value;

let storage\_bucket = null, storage\_path = null;

const file = el('file').files[0];

if (file && ['pdf','image','audio','video'].includes(kind)) {

const bucket = kind === 'image' ? 'img' : kind; // img | video | pdf | audio

const path = `${grade}/${Date.now()}-${file.name}`;

const { error: upErr } = await supabase.storage.from(bucket).upload(path, file, { upsert:false });

if (upErr) { alert(upErr.message); return; }

storage\_bucket = bucket; storage\_path = path;

}

const { error } = await supabase.from('materials').insert({

grade, term\_no, class\_no, scope, kind, title, body, link\_url, storage\_bucket, storage\_path

});

if (error) { alert(error.message); return; }

alert('Publicado');

loadList();

};

async function loadList(){

const grade = Number(el('grade').value); const term\_no = Number(el('term').value);

const { data } = await supabase.from('materials').select('\*').match({ grade, term\_no }).order('created\_at', { ascending: false });

el('list').innerHTML = (data||[]).map(m=>

`<div class="item">

<b>${m.kind}</b> [${m.scope}] ${m.title||''}

<button data-del="${m.id}">Eliminar</button>

</div>`

).join('');

el('list').querySelectorAll('[data-del]').forEach(btn=>{

btn.onclick = async ()=>{ await supabase.from('materials').delete().eq('id', Number(btn.dataset.del)); loadList(); };

});

}

['grade','term'].forEach(id=> el(id).addEventListener('change', loadList));

</script>

</body>

</html>

**styles.css (boceto mínimo):**

:root{ font-family: system-ui, sans-serif; color-scheme: light dark; }

main.login{ max-width: 360px; margin: 10vh auto; display: grid; gap: 12px; }

input, button, textarea, select{ padding: .6rem .8rem; border-radius: 10px; border: 1px solid #ccc; }

nav{ display: flex; gap: 8px; margin-bottom: 12px; }

section{ margin: 16px auto; max-width: 960px; }

.clase{ padding: 12px; border: 1px solid #ddd; border-radius: 14px; margin-bottom: 12px; }

.video iframe, .pdf iframe{ width: 100%; height: 360px; border: 0; border-radius: 12px; }

img{ max-width: 100%; border-radius: 12px; display: block; }

**10) Embebido de videos**

* **YouTube/Vimeo**: usar <iframe> con URL de **embed** para reproducir en la misma página.
* **Videos propios (.mp4)**: subir a bucket video y reproducir con <video src="PUBLIC\_URL" controls>.

**11) Realtime**

Suscribirse a cambios en materials, classes y terms por grado. El ejemplo en grado.html refresca materials cuando hay inserciones/updates/deletes del mismo grado.

**12) Despliegue**

* **Sitio estático** a Git (GitHub Pages/Netlify/Vercel).
* Configurar **dominio** si aplica.
* Asegurarse de permitir https://<tu-dominio> en **Auth → URL Redirects** (Supabase).
* Rotar códigos cuando sea necesario cambiando la **contraseña** del usuario por grado en Supabase Auth.

**13) Roadmap**

**Fase 0 (hoy):** Esquema BD + RLS + buckets + usuarios (admin y g1..g11) + profiles.  
**Fase 1:** Login por código + vista de grado (read) + carga de materiales desde dashboard (admin).  
**Fase 2:** Realtime + editor más rico (markdown) + filtros/búsqueda.  
**Fase 3:** Permisos avanzados (lectura por grado en Storage), auditoría de cambios, exportaciones (PDF).

**14) Riesgos y mitigaciones**

* **Compartición de credenciales** del grado: aceptado como requisito; mitigación: rotar códigos por periodo.
* **Almacenamiento público (MVP):** si se requiere mayor privacidad, activar políticas de lectura por grado en Storage (V1).
* **Escalabilidad de edición:** si aumentan docentes, crear roles adicionales y/o sub-roles por área.

**15) Próximos pasos**

1. Crear buckets y ejecutar SQL de **esquema** y **políticas**.
2. Crear usuarios (admin@sdcs.local, g1@sdcs.local … g11@sdcs.local) con sus contraseñas = códigos.
3. Rellenar profiles.
4. Subir el scaffolding (index.html, grado.html, admin.html, assets/styles.css).
5. Probar flujo completo: login → vista de grado → publicar desde dashboard → visualización en vivo.