CSSP

Core Strong Studio Pilates

Especificación Técnica y Funcional Completa

Proyecto de Portafolio Profesional

Proyecto: Sistema de Gestión para Estudio de Pilates

Tipo: SPA (Single Page Application)

Fecha de Especificación: 18 de Enero, 2026

Versión: 1.0 (Retroactiva - Proyecto Completado)

Estado: ✅ COMPLETADO

Autor: Juan Carlos Quiñonez Madrid

Tabla de Contenidos

1. Descripción del Proyecto

Core Strong Studio Pilates (CSSP) es un sistema de gestión completo para estudios de Pilates, construido como una Single Page Application (SPA) profesional utilizando tecnologías modernas de frontend y Backend as a Service (BaaS).

El sistema permite la gestión integral de clientes, instructores y administradores, facilitando la compra de paquetes de clases, reservas de sesiones, registro de asistencias y administración centralizada de usuarios.

Objetivos del Proyecto

1. Automatizar la gestión de reservas y asistencias en un estudio de Pilates
2. Proporcionar interfaces diferenciadas por rol (Cliente, Instructor, Admin)
3. Implementar un sistema de paquetes con vigencia y clases consumibles
4. Gestionar calendario de clases con validaciones de negocio robustas
5. Garantizar seguridad a nivel de base de datos mediante Row Level Security (RLS)
6. Demostrar capacidad profesional en desarrollo frontend moderno

Propósito para Portafolio

Este proyecto demuestra dominio técnico en:

- ✓ Arquitectura Vue 3 con Composition API y TypeScript
- ✓ Integración profesional con servicios BaaS (Supabase)
- ✓ Implementación de seguridad mediante RLS, Triggers y Funciones SQL
- ✓ Diseño de interfaces responsivas con TailwindCSS
- ✓ Gestión de estado global con Pinia
- ✓ Validaciones de negocio en frontend y backend
- ✓ Manejo correcto de timezones en aplicaciones web
- ✓ Código limpio, mantenible y defendible en entrevistas técnicas

2. Contexto Operativo

Características del Negocio

Core Strong Studio Pilates es un estudio boutique que ofrece clases de Pilates en diferentes sucursales físicas. El modelo de negocio se basa en:

- **Venta de paquetes:** Los clientes compran paquetes con un número determinado de clases
- **Sistema de reservas:** Los clientes reservan sus clases con anticipación
- **Capacidad limitada:** Cada clase tiene un cupo máximo de estudiantes
- **Vigencia temporal:** Los paquetes tienen fecha de vencimiento
- **Registro de asistencia:** Los instructores registran quién asistió a cada clase

Usuarios del Sistema

Tipo de Usuario	Cantidad Estimada	Función Principal
Visitantes (Guest)	∞ (sin límite)	Conocer el estudio, ver planes, registrarse
Clientes	50-200 activos	Comprar paquetes, reservar clases, gestionar asistencias
Instructores	3-10	Impartir clases, registrar asistencias, crear horarios
Administradores	1-3	Gestionar clientes, crear instructores, supervisar operación

Flujo de Negocio Principal

1. Cliente se registra en la plataforma

2. Cliente compra un paquete de clases
3. Cliente reserva clases desde el calendario
4. Cliente asiste a la clase (o cancela con anticipación)
5. Instructor registra la asistencia
6. Sistema actualiza el saldo de clases del cliente

3. Estructura de Roles y Jerarquía

3.1 Roles del Sistema

ROL 1: Visitante (Guest)

> **Cantidad:** Sin límite (usuarios no autenticados)
>
> **Función:** Usuarios que aún no tienen cuenta o no han iniciado sesión

Permisos:

- ✓ Ver Landing Page (información del estudio)
- ✓ Ver Planes disponibles
- ✓ Acceder a página de Ayuda (FAQ)
- ✓ Registrarse como nuevo cliente
- ✓ Iniciar sesión
- ✗ NO puede reservar clases
- ✗ NO puede comprar paquetes (debe registrarse primero)
- ✗ NO puede acceder a dashboards

ROL 2: Cliente

> **Cantidad:** Escalable (50-200 clientes activos esperados)
>
> **Función:** Usuarios registrados que consumen servicios del estudio

Permisos:

- ✓ Ver su Dashboard personalizado
- ✓ Comprar paquetes de clases

- ✓ Ver calendario de clases disponibles
- ✓ Reservar clases (si tiene paquete activo)
- ✓ Ver sus reservas (Próximas, Historial, Canceladas)
- ✓ Cancelar reservas (con restricción de tiempo)
- ✓ Ver planes disponibles
- ✓ Acceder a página de Ayuda
- ✗ NO puede ver datos de otros clientes
- ✗ NO puede registrar asistencias
- ✗ NO puede crear clases

ROL 3: Instructor

> **Cantidad:** 3-10 instructores
>
> **Función:** Profesionales que imparten las clases de Pilates

Permisos:

- ✓ Ver calendario de sus clases asignadas
- ✓ Ver lista de estudiantes por clase
- ✓ Registrar asistencia de alumnos
- ✓ Crear nuevas clases en el calendario
- ✓ Ver ocupación de sus clases en tiempo real
- ✗ NO puede ver datos personales completos de clientes
- ✗ NO puede gestionar paquetes
- ✗ NO puede crear otros instructores

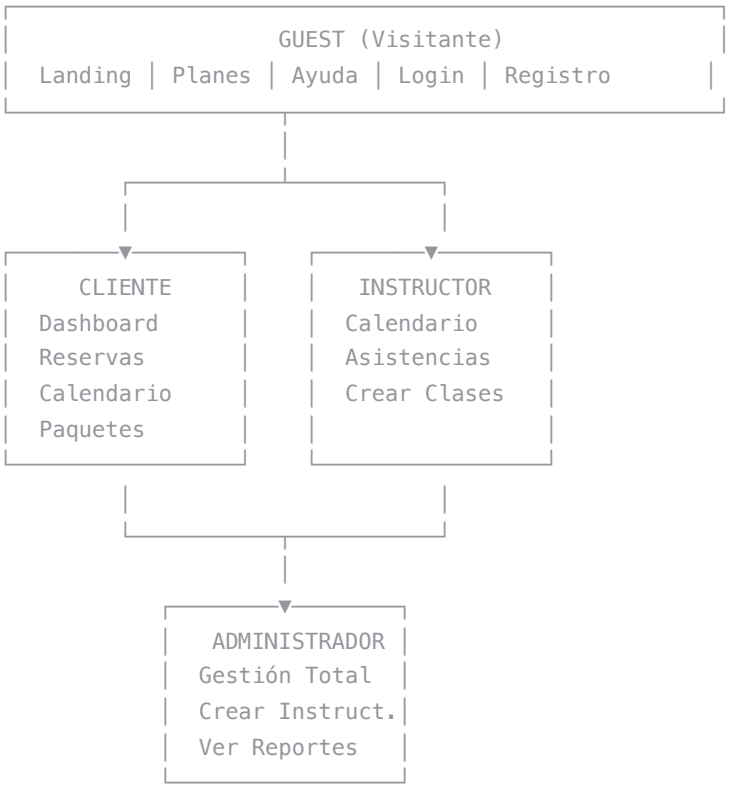
ROL 4: Administrador

> **Cantidad:** 1-3 administradores
>
> **Función:** Personal administrativo con acceso total al sistema

Permisos:

- ✓ Ver gestión completa de clientes
- ✓ Buscar y filtrar clientes
- ✓ Ver detalles completos de cualquier cliente
- ✓ Crear cuentas de instructor
- ✓ Ver estadísticas generales del sistema
- ✓ Acceso a todas las vistas del sistema
- ⚠ NO puede eliminar cuentas (solo creación en v1.0)

3.2 Jerarquía de Acceso



3.3 Navegación Visual por Rol

Rol	Color de Identidad	Links en Navbar	Dropdown Perfil
Guest	Neutral (blanco/gris)	Home | Planes | Ayuda | **Iniciar Sesión**	No visible
Cliente	Azul	Home | Dashboard | Mis Reservas | Planes | Calendario	Iniciales + Info + Cerrar Sesión
Instructor	Púrpura	Home | Calendario | Asistencia | Crear Clases	Iniciales + Info + Cerrar Sesión
Admin	Verde	Home | Gestión Clientes | Crear Instructor	Iniciales + Info + Cerrar Sesión

4. Reglas de Negocio Críticas

4.1 Sistema de Paquetes

Estructura de Paquetes

Cada paquete en el sistema contiene:

- **Nombre:** Identificador único del paquete (ej: "Paquete Básico")
- **Número de clases:** Cantidad de clases incluidas (ej: 4, 8, 12)
- **Precio:** Costo del paquete en moneda local
- **Vigencia:** Días de validez desde la compra (ej: 30 días)
- **Descripción:** Detalles adicionales del paquete

Reglas de Compra

> **REGLA CRÍTICA:** Solo usuarios autenticados con rol "cliente" pueden comprar paquetes.

- ✓ Un cliente puede tener múltiples paquetes activos simultáneamente
- ✓ Los paquetes se activan inmediatamente después de la compra
- ✓ La fecha de vencimiento se calcula automáticamente: `fecha_compra + vigencia_dias`
- ⚠ No hay restricciones de compra (el cliente puede comprar el mismo paquete múltiples veces)

Estados de Paquetes

Estado	Condición	Permite Reservas	Visualización
Activo	clases_restantes > 0 Y fecha_actual ≤ fecha_vencimiento	✓ Sí	Badge verde "Activo"
Por Vencer	fecha_vencimiento - fecha_actual ≤ 7 días	✓ Sí (con advertencia)	Badge naranja "Por vencer"
Agotado	clases_restantes = 0	✗ No	Badge gris "Agotado"
Vencido	fecha_actual > fecha_vencimiento	✗ No	Badge rojo "Vencido"

4.2 Sistema de Reservas

Requisitos para Reservar

> **VALIDACIÓN OBLIGATORIA:** Antes de permitir una reserva, el sistema DEBE verificar:
>
> 1. Usuario autenticado como "cliente"
> 2. Cliente tiene al menos 1 paquete activo (estado: activo)
> 3. Paquete tiene clases_restantes > 0
> 4. La clase no está llena (capacidad_actual < capacidad_maxima)
> 5. El cliente NO tiene ya una reserva para esa clase
> 6. La fecha de la clase es futura (no se pueden reservar clases pasadas)

Proceso de Reserva

1. Cliente selecciona fecha en el calendario
2. Sistema muestra clases disponibles para esa fecha
3. Cliente selecciona horario de la clase
4. Sistema valida todos los requisitos
5. Sistema crea registro en `mis_reservas`

6. Sistema actualiza:
    - `clases_restantes` del paquete (-1)
    - `capacidad_actual` de la clase (+1)
7. Sistema muestra confirmación al cliente

Cancelación de Reservas

> **REGLA TEMPORAL CRÍTICA:** Solo se puede cancelar una reserva si faltan MÁS de 3 horas para el inicio de la clase.

Cálculo de restricción:

```
hora_actual + 3 horas < hora_inicio_clase
```

Proceso de cancelación:

1. Cliente solicita cancelar reserva
2. Sistema valida restricción temporal
3. Si es válido:
    - Cambia estado de reserva a "cancelada"
    - Devuelve 1 clase al paquete (clases_restantes +1)
    - Reduce capacidad_actual de la clase (-1)
    - Muestra confirmación
4. Si no es válido:
    - Muestra error: "No se puede cancelar con menos de 3 horas de anticipación"
    - La reserva permanece activa

Estados de Reservas

Estado	Descripción	Visible en Tab
confirmada	Reserva activa, pendiente de asistir	Próximas

Estado	Descripción	Visible en Tab
asistió	Cliente asistió a la clase (registrado por instructor)	Historial
faltó	Cliente no asistió (registrado por instructor)	Historial
cancelada	Cliente canceló la reserva a tiempo	Canceladas

4.3 Gestión de Asistencias

Registro por Instructor

Los instructores pueden registrar asistencia solo para clases que:

- ✓ Ya pasaron (fecha y hora en el pasado)
- ✓ Tienen al menos 1 reserva confirmada
- ✓ Fueron creadas por ellos mismos

> **LÍMITE DE VISUALIZACIÓN:** Solo se muestran las últimas 20 clases con reservas.

Proceso de Registro

1. Instructor accede a "Registro de Asistencia"
2. Sistema muestra lista de clases pasadas con reservas
3. Instructor marca cada estudiante como "Asistió" ✓ o "Faltó" ✗
4. Botón "Confirmar" se habilita solo cuando TODOS los alumnos están marcados
5. Sistema actualiza campo `asistio` en tabla `mis_reservas`
6. Sistema actualiza estado de reserva a "asistió" o "faltó"

> **IMPORTANTE:** Las clases NO consumidas por faltas NO se devuelven al paquete del cliente. Una vez registrada la asistencia como "faltó", la clase se cuenta como consumida.

4.4 Calendario del Cliente

Restricciones de Navegación

- ✗ NO se puede navegar a meses pasados
- ✗ NO se puede navegar más de 2 meses al futuro
- ✗ Días pasados NO son clickeables (visual: gris y cursor bloqueado)
- ✓ Botón "Hoy" siempre vuelve al mes actual

Visualización de Clases

Al hacer click en un día válido, se muestran:

- Todas las clases disponibles para esa fecha
- Horario de cada clase (ej: "10:00 AM - 11:00 AM")
- Capacidad actual vs máxima (ej: "5/10 inscritos")
- Instructor asignado
- Botón "Reservar" (si cumple validaciones)

4.5 Calendario del Instructor

Código de Colores por Ocupación

% Ocupación	Color	Significado
0%	Gris	Sin reservas
1% - 49%	Azul	Ocupación baja
50% - 99%	Naranja	Ocupación media-alta
100%	Rojo	Clase llena

Vistas Disponibles

- **Vista Mensual:** Calendario tradicional con eventos
- **Vista Semanal:** Horarios detallados por día
- **Vista Diaria:** Lista de clases del día seleccionado

Rango de Navegación

- Hasta 3 meses atrás (historial)
- Hasta 3 meses adelante (planificación)

5. Formato de Usuarios y Credenciales

5.1 Registro de Clientes

Validaciones de Campos

Campo	Validaciones	Ejemplo Válido
Nombre Completo	• Requerido • Sin caracteres especiales • Trim de espacios	Juan Carlos Pérez
Email	• Formato válido (contiene @) • Único en el sistema • Lowercase automático	juan.perez@gmail.com
Teléfono	• Mínimo 10 caracteres • Máximo 15 caracteres • Solo números	6671234567
Contraseña	• Mínimo 8 caracteres • Al menos 1 mayúscula • Al menos 1 minúscula • Al menos 1 número	MiPass123
Confirmar Contraseña	• Debe coincidir con Contraseña	MiPass123

Proceso de Registro

1. Usuario completa formulario en `/registrarse`
2. Frontend valida todos los campos
3. Se envía petición a Supabase Auth (`signUp()`)
4. Supabase crea usuario en `auth.users`
5. Trigger SQL automático:

- Crea registro en `profiles` con rol='cliente'
- Trigger secundario crea registro en `clientes`

6. Frontend ejecuta auto-login (`signIn()`)
7. AuthStore se inicializa con datos del usuario
8. Redirección automática a `/dashboard-cliente`

> **EXPERIENCIA DE USUARIO:** El flujo es completamente fluido. El usuario NO tiene que iniciar sesión manualmente después de registrarse.

5.2 Creación de Instructores (Admin)

Proceso Administrativo

Solo administradores pueden crear cuentas de instructor mediante el formulario en `/crear-instructor` .

Campos Requeridos

- **Nombre Completo:** Mismas validaciones que cliente
- **Email:** Único, formato válido
- **Teléfono:** Exactamente 10 dígitos

Generación de Contraseña Temporal

> **SEGURIDAD:** El sistema genera automáticamente una contraseña temporal segura para el instructor.
>
> Formato: `[Nombre][4 dígitos aleatorios]!`
>
> Ejemplo: Si el instructor se llama "María", la contraseña podría ser: `Maria7829!`

Proceso:

1. Admin completa formulario
2. Sistema genera contraseña temporal
3. Crea usuario en Supabase Auth
4. Crea registro en `profiles` con rol='instructor'
5. Crea registro en `instructores`
6. Muestra contraseña temporal al admin (para compartir con el instructor)
7. Instructor debe cambiar contraseña en su primer login (v2.0)

5.3 Login

Validaciones

- Email válido requerido
- Contraseña requerida
- Botón inhabilitado hasta completar ambos campos

Redirección por Rol

Rol Detectado	Ruta de Redirección
cliente	/dashboard-cliente
instructor	/calendario-instructor
admin	/gestion-clientes

Manejo de Errores

> **SEGURIDAD:** Por razones de seguridad, NO se especifica si el error es por email o contraseña incorrectos.
>
> Mensaje genérico: **"Credenciales inválidas"**

5.4 Logout

Disponible desde el dropdown de perfil (esquina superior derecha) para todos los usuarios autenticados.

Proceso:

1. Usuario hace click en "Cerrar Sesión"
2. Sistema ejecuta `supabase.auth.signOut()`
3. AuthStore limpia estado (`resetAuth()`)
4. Redirección automática a Landing Page (`/`)
5. Navbar vuelve a modo "Visitante"

6. Stack Tecnológico

Frontend

Tecnología	Versión	Propósito
Vue 3	^3.5.13	Framework progresivo (Composition API)
TypeScript	~5.6.2	Tipado estático y seguridad en tiempo de desarrollo
Vite	^6.0.1	Build tool y dev server ultra-rápido
Vue Router	^4.5.0	Enrutamiento SPA + Navigation Guards
Pinia	^2.3.0	Estado global (AuthStore)
Tailwind CSS	^3.4.17	Framework de utilidades CSS
Zod	^3.24.1	Validación de esquemas y formularios
FullCalendar	^6.1.15	Componente de calendario interactivo

Backend (BaaS - Supabase)

Servicio	Uso en el Proyecto
Supabase Auth	Sistema de autenticación con email/password
PostgreSQL	Base de datos relacional (8 tablas)
Row Level Security (RLS)	Políticas de acceso a nivel de fila
SQL Functions	is_admin(), get_user_email()
Triggers	Creación automática de cliente post-registro
@supabase/supabase-js	Cliente JavaScript (v2.48.0)

Herramientas de Desarrollo

- **Git + GitHub:** Control de versiones
- **ESLint:** Linting de código
- **Prettier:** Formateo automático
- **Conventional Commits:** Estándar de commits
- **Supabase Gen Types:** Generación de tipos TypeScript desde DB

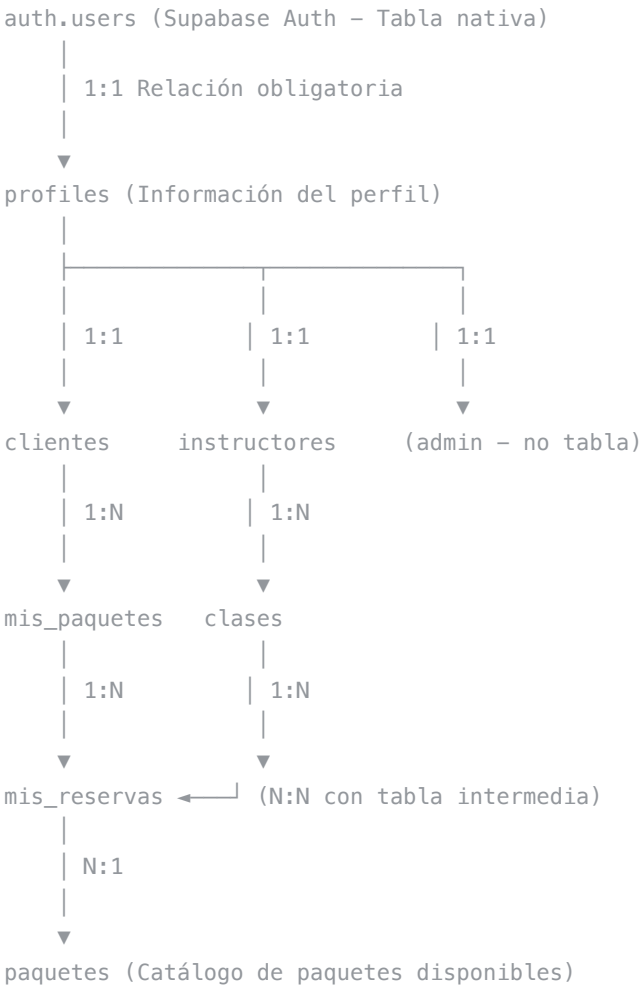
Arquitectura de Carpetas

```
src/
├── assets/                # Recursos estáticos (imágenes, iconos)
├── components/            # Componentes reutilizables
│   ├── NavbarDynamic.vue  # Navbar que cambia según rol
│   ├── Footer.vue         # Footer global
│   └── Modal.vue          # Modal reutilizable
├── composables/           # Lógica reutilizable (Composition API)
│   ├── useCalendar.ts     # Lógica del calendario cliente
│   ├── useCalendarCliente.ts
│   ├── useCalendarioInstructor.ts
│   ├── useGestionClientes.ts
│   ├── useModal.ts
│   └── useRegistroAsistencia.ts
├── lib/                   # Configuraciones de servicios externos
│   └── supabase.ts        # Cliente de Supabase configurado
├── stores/                # Estado global con Pinia
│   └── auth.ts            # AuthStore (sesión, rol, usuario)
├── views/                 # Vistas organizadas por acceso
│   ├── public/            # Vistas públicas (sin auth)
│   │   ├── LandingPageView.vue
│   │   ├── PlanesView.vue
│   │   ├── AyudaView.vue
│   │   ├── LoginView.vue
│   │   └── RegistrarseView.vue
│   ├── cliente/           # Vistas de cliente (auth + rol)
│   │   ├── DashboardClienteView.vue
│   │   ├── MisReservasView.vue
│   │   └── CalendarioClienteView.vue
│   ├── instructor/        # Vistas de instructor
│   │   ├── CalendarioInstructorView.vue
│   │   ├── CrearClasesView.vue
│   │   └── RegistroAsistenciaView.vue
│   ├── admin/             # Vistas de administrador
│   │   ├── GestionClientesView.vue
│   │   └── CrearInstructorView.vue
│   └── NotFoundView.vue   # Página 404 personalizada
├── router/
│   └── index.ts           # Configuración de rutas + guards
```

```
└── App.vue                 # Componente raíz (layout principal)
```

7. Estructura de la Base de Datos

7.1 Esquema Relacional Visual



7.2 Tablas Principales

Tabla: `profiles`

Propósito: Almacenar información del perfil de todos los usuarios del sistema.

Campo	Tipo	Restricciones	Descripción
`id`	UUID	PK, FK auth.users	Mismo ID que auth.users
`nombre_completo`	VARCHAR(100)	NOT NULL	Nombre del usuario
`telefono`	VARCHAR(15)	NOT NULL	Teléfono de contacto
`rol`	VARCHAR(20)	DEFAULT 'cliente'	cliente | instructor | admin
`created_at`	TIMESTAMP	DEFAULT NOW()	Fecha de creación

Tabla: `clientes`

Propósito: Extensión de datos específicos para usuarios con rol 'cliente'.

Campo	Tipo	Restricciones	Descripción
`id`	UUID	PK	ID único del cliente
`profile_id`	UUID	FK profiles, UNIQUE	Relación 1:1 con profiles
`direccion`	VARCHAR(255)	NULLABLE	Dirección física (opcional)
`created_at`	TIMESTAMP	DEFAULT NOW()	Fecha de creación

Tabla: `instructores`

Propósito: Extensión de datos específicos para usuarios con rol 'instructor'.

Campo	Tipo	Restricciones	Descripción
`id`	UUID	PK	ID único del instructor
`profile_id`	UUID	FK profiles, UNIQUE	Relación 1:1 con profiles
`especialidad`	VARCHAR(100)	NULLABLE	Especialidad del instructor (opcional)
`created_at`	TIMESTAMP	DEFAULT NOW()	Fecha de creación

Tabla: `paquetes`

Propósito: Catálogo de paquetes de clases disponibles para compra.

Campo	Tipo	Restricciones	Descripción
`id`	UUID	PK	ID único del paquete
`nombre`	VARCHAR(50)	UNIQUE, NOT NULL	Nombre del paquete
`descripcion`	TEXT	NULLABLE	Descripción detallada
`precio`	DECIMAL(10,2)	NOT NULL	Precio en moneda local
`num_clases`	INTEGER	NOT NULL	Número de clases incluidas
`vigencia_dias`	INTEGER	NOT NULL	Días de validez desde compra
`activo`	BOOLEAN	DEFAULT true	Disponible para compra
`created_at`	TIMESTAMP	DEFAULT NOW()	Fecha de creación

Tabla: `mis_paquetes`

Propósito: Registro de paquetes comprados por cada cliente.

Campo	Tipo	Restricciones	Descripción
`id`	UUID	PK	ID único del paquete comprado
`cliente_id`	UUID	FK clientes, NOT NULL	Cliente que compró
`paquete_id`	UUID	FK paquetes, NOT NULL	Paquete adquirido
`clases_totales`	INTEGER	NOT NULL	Clases originales del paquete

Campo	Tipo	Restricciones	Descripción
clases_restantes	INTEGER	NOT NULL	Clases disponibles actualmente
fecha_compra	TIMESTAMP	DEFAULT NOW()	Momento de la compra
fecha_vencimiento	DATE	NOT NULL	Calculado: fecha_compra + vigencia_dias
activo	BOOLEAN	DEFAULT true	Si el paquete está disponible

Tabla: `clases`

Propósito: Catálogo de clases programadas por instructores.

Campo	Tipo	Restricciones	Descripción
id	UUID	PK	ID único de la clase
instructor_id	UUID	FK instructores, NOT NULL	Instructor asignado
fecha	DATE	NOT NULL	Día de la clase
hora_inicio	TIME	NOT NULL	Hora de inicio
hora_fin	TIME	NOT NULL	Hora de finalización
capacidad_maxima	INTEGER	NOT NULL	Cupo máximo de estudiantes
capacidad_actual	INTEGER	DEFAULT 0	Estudiantes inscritos actualmente
created_at	TIMESTAMP	DEFAULT NOW()	Fecha de creación de la clase

Tabla: `mis_reservas`

Propósito: Registro de reservas de clientes a clases específicas.

Campo	Tipo	Restricciones	Descripción
id	UUID	PK	ID único de la reserva
cliente_id	UUID	FK clientes, NOT NULL	Cliente que reservó
clase_id	UUID	FK clases, NOT NULL	Clase reservada
mi_paquete_id	UUID	FK mis_paquetes, NOT NULL	Paquete usado para reservar
fecha_reserva	TIMESTAMP	DEFAULT NOW()	Momento en que se hizo la reserva
estado	VARCHAR(20)	DEFAULT 'confirmada'	confirmada | cancelada
asistio	BOOLEAN	DEFAULT NULL	true (asistió) | false (faltó) | null (pendiente)
created_at	TIMESTAMP	DEFAULT NOW()	Fecha de creación del registro

Constraint único:

```
UNIQUE (cliente_id, clase_id) -- Evita reservas duplicadas
```

7.3 Triggers y Funciones SQL

Trigger: Crear Cliente Automáticamente

Propósito: Cuando se crea un perfil con rol='cliente', automáticamente crear su registro en la tabla clientes.

```
-- Función que ejecuta el trigger
CREATE OR REPLACE FUNCTION public.handle_new_profile()
RETURNS TRIGGER
SECURITY DEFINER
SET search_path = public
LANGUAGE plpgsql
AS $$
BEGIN
  IF NEW.rol = 'cliente' THEN
```

```
    INSERT INTO public.clientes (profile_id, direccion)
    VALUES (NEW.id, NULL);

    RAISE NOTICE 'Cliente creado para profile_id: %', NEW.id;
  END IF;

  RETURN NEW;
END;
$$;

-- Trigger que se ejecuta DESPUÉS de insertar en profiles
CREATE TRIGGER on_profile_created
  AFTER INSERT ON public.profiles
  FOR EACH ROW
  EXECUTE FUNCTION public.handle_new_profile();
```

> **¿Por qué SECURITY DEFINER?**
>
> Permite que el trigger ejecute con permisos de administrador, evitando que las políticas RLS bloqueen la creación automática del cliente.

Función: Verificar si es Admin

```
CREATE OR REPLACE FUNCTION public.is_admin()
RETURNS BOOLEAN
LANGUAGE sql
SECURITY DEFINER
AS $$
  SELECT EXISTS (
    SELECT 1
    FROM public.profiles
    WHERE id = auth.uid() AND rol = 'admin'
  );
$$;
```

Uso: Utilizado en políticas RLS para permitir acceso completo a administradores.

Función: Obtener Email de Usuario

```
CREATE OR REPLACE FUNCTION public.get_user_email(user_id UUID)
RETURNS TEXT
LANGUAGE sql
SECURITY DEFINER
```

```
AS $$
  SELECT email::text
  FROM auth.users
  WHERE id = user_id;
$$;
```

Uso: Permite obtener emails de usuarios desde tablas públicas (auth.users no es accesible directamente por RLS).

7.4 Políticas RLS (Row Level Security)

Políticas para `profiles`

```
-- Los usuarios ven su propio perfil
CREATE POLICY "Users can view own profile"
ON public.profiles FOR SELECT
USING (auth.uid() = id);

-- Los usuarios actualizan su propio perfil
CREATE POLICY "Users can update own profile"
ON public.profiles FOR UPDATE
USING (auth.uid() = id);

-- Los administradores ven todos los perfiles
CREATE POLICY "Admins can view all profiles"
ON public.profiles FOR SELECT
USING (public.is_admin());

-- El sistema puede insertar perfiles (registro)
CREATE POLICY "System can insert profiles"
ON public.profiles FOR INSERT
WITH CHECK (true);
```

Políticas para `clientes`

```
-- Los clientes ven su propia información
CREATE POLICY "Clientes pueden ver su propia info"
ON public.clientes FOR SELECT
USING (profile_id = auth.uid());

-- El sistema puede crear clientes (trigger)
CREATE POLICY "System can create clientes"
ON public.clientes FOR INSERT
WITH CHECK (true);
```

```
-- Los administradores gestionan todos los clientes
CREATE POLICY "Admins pueden gestionar clientes"
ON public.clientes FOR ALL
USING (public.is_admin());
```

Políticas para `mis_reservas`

```
-- Los clientes ven sus propias reservas
CREATE POLICY "Clientes pueden ver sus reservas"
ON public.mis_reservas FOR SELECT
USING (
  cliente_id IN (
    SELECT id FROM public.clientes
    WHERE profile_id = auth.uid()
  )
);

-- Los clientes pueden crear reservas
CREATE POLICY "Clientes pueden crear reservas"
ON public.mis_reservas FOR INSERT
WITH CHECK (
  cliente_id IN (
    SELECT id FROM public.clientes
    WHERE profile_id = auth.uid()
  )
);

-- Los instructores ven reservas de sus clases
CREATE POLICY "Instructores pueden ver reservas de sus clases"
ON public.mis_reservas FOR SELECT
USING (
  clase_id IN (
    SELECT c.id FROM public.clases c
    JOIN public.instructores i ON i.id = c.instructor_id
    WHERE i.profile_id = auth.uid()
  )
);

-- Los instructores actualizan asistencia
CREATE POLICY "Instructores pueden actualizar asistencia"
ON public.mis_reservas FOR UPDATE
USING (
  clase_id IN (
    SELECT c.id FROM public.clases c
    JOIN public.instructores i ON i.id = c.instructor_id
    WHERE i.profile_id = auth.uid()
  )
);
```

Políticas para `clases`

```
-- Todos pueden ver clases disponibles
CREATE POLICY "Anyone can view classes"
ON public.clases FOR SELECT
USING (true);

-- El sistema puede actualizar capacidad
CREATE POLICY "System can update class capacity"
ON public.clases FOR UPDATE
USING (true);

-- Los instructores crean sus propias clases
CREATE POLICY "Instructores pueden crear sus clases"
ON public.clases FOR INSERT
WITH CHECK (
  instructor_id IN (
    SELECT id FROM public.instructores
    WHERE profile_id = auth.uid()
  )
);
```

8. Flujos Principales del Sistema

8.1 Flujo: Registro de Cliente

1. Usuario accede a `/registrarse`
2. Completa formulario con validaciones en tiempo real
3. Frontend valida todos los campos
4. Se envía petición a Supabase Auth: `signUp(email, password)`
5. Supabase crea usuario en `auth.users`
6. Frontend crea registro en `profiles` con datos del usuario
7. **Trigger automático:** Se crea registro en `clientes`
8. Frontend ejecuta auto-login: `signIn(email, password)`
9. AuthStore se inicializa con datos del usuario
10. Redirección automática a `/dashboard-cliente`

8.2 Flujo: Compra de Paquete

1. Cliente autenticado accede a `/planes`
2. Visualiza paquetes disponibles (filtrados por `activo = true`)
3. Selecciona un paquete y hace click en "Comprar"
4. Sistema valida que el usuario esté autenticado y sea cliente
5. Se abre modal de confirmación con resumen del paquete
6. Cliente confirma la compra
7. Sistema ejecuta INSERT en `mis_paquetes`:
    - `cliente_id`: ID del cliente
    - `paquete_id`: ID del paquete seleccionado
    - `clases_totales`: Copiado de `paquetes.num_clases`
    - `clases_restantes`: Igual a `clases_totales`
    - `fecha_compra`: `NOW()`

- `fecha_vencimiento` : `fecha_compra + paquetes.vigencia_dias`

8. Sistema muestra confirmación de compra exitosa
9. Redirección automática a `/dashboard-cliente`
10. Dashboard muestra el nuevo paquete activo

8.3 Flujo: Reservar una Clase

1. Cliente accede a `/calendario-cliente`
2. Sistema valida que el cliente tenga al menos 1 paquete activo
3. Cliente selecciona una fecha futura en el calendario
4. Sistema obtiene clases disponibles para esa fecha
5. Se muestran clases con:
   - Horario
   - Instructor
   - Capacidad actual/máxima
   - Botón "Reservar" (si no está llena y el cliente no tiene reserva)
6. Cliente selecciona horario y hace click en "Reservar"
7. Sistema valida:
   - Cliente tiene paquete activo con clases_restantes > 0
   - Clase tiene cupo disponible
   - Cliente NO tiene reserva previa para esa clase
8. Si todo es válido, sistema ejecuta **transacción atómica**:
   - INSERT en `mis_reservas`
   - UPDATE en `mis_paquetes` : `clases_restantes - 1`
   - UPDATE en `clases` : `capacidad_actual + 1`
9. Sistema muestra confirmación de reserva
10. Opción de ir a "Mis Reservas" o seguir reservando

8.4 Flujo: Cancelar Reserva

1. Cliente accede a `/mis-reservas` → Tab "Próximas"
2. Visualiza lista de reservas confirmadas

3. Hace click en botón "Cancelar" de una reserva
4. Sistema calcula tiempo restante hasta la clase
5. **Validación temporal:**
   - Si faltan ≤ 3 horas: Mostrar error "No se puede cancelar con menos de 3 horas"
   - Si faltan > 3 horas: Permitir cancelación
6. Si es válido, sistema muestra modal de confirmación
7. Cliente confirma la cancelación
8. Sistema ejecuta **transacción atómica**:
   - UPDATE en `mis_reservas`: `estado = 'cancelada'`
   - UPDATE en `mis_paquetes`: `clases_restantes + 1` (devolución)
   - UPDATE en `clases`: `capacidad_actual - 1`
9. Reserva se mueve automáticamente al tab "Canceladas"
10. Sistema muestra confirmación y saldo actualizado

8.5 Flujo: Registrar Asistencia (Instructor)

1. Instructor accede a `/registro-asistencia`
2. Sistema obtiene las últimas 20 clases pasadas del instructor que tienen reservas
3. Para cada clase, se muestra:
   - Fecha y horario
   - Número de estudiantes inscritos
   - Lista de estudiantes con checkboxes (Asistió / Faltó)
4. Instructor marca cada estudiante individualmente
5. Sistema habilita botón "Confirmar" solo cuando TODOS los estudiantes están marcados
6. Instructor hace click en "Confirmar"
7. Sistema ejecuta **UPDATE masivo** en `mis_reservas`:
   - Para cada estudiante marcado como "Asistió": `asistio = true`
   - Para cada estudiante marcado como "Faltó": `asistio = false`
8. Sistema muestra confirmación de guardado

9. Los cambios se reflejan inmediatamente en el historial del cliente

8.6 Flujo: Crear Clase (Instructor)

1. Instructor accede a `/crear-clases`
2. Completa formulario:
   - Fecha de la clase
   - Hora de inicio
   - Hora de fin
   - Capacidad máxima
   - Checkbox "Repetir semanalmente" (opcional)
3. Sistema valida:
   - Fecha sea futura
   - Hora de fin > hora de inicio
   - Capacidad > 0
   - No exista solapamiento con clases existentes del instructor
4. Si "Repetir semanalmente" está activo:
   - Sistema genera clases para las próximas N semanas (configurable)
   - Ejemplo: Si fecha es 5 de febrero, crea clases para 5, 12, 19, 26 de febrero
5. Sistema muestra resumen de clases a crear
6. Instructor confirma
7. Sistema ejecuta INSERT múltiple en `clases`
8. Sistema muestra confirmación con número de clases creadas
9. Clases aparecen inmediatamente en `/calendario-instructor`

8.7 Flujo: Gestión de Clientes (Admin)

1. Admin accede a `/gestion-clientes`
2. Sistema muestra:
   - Tarjetas de resumen (Total clientes, Activos, Sin paquete)
   - Tabla completa de clientes

3. Admin puede:
    - Buscar por nombre, email o teléfono (búsqueda en tiempo real)
    - Filtrar por estado (Con paquetes / Sin paquetes)
4. Admin hace click en ícono "ojo" de un cliente
5. Sistema abre modal con información completa:
    - Datos personales
    - Paquetes activos e inactivos
    - Últimas 10 reservas con badges de asistencia
6. Admin cierra modal y continúa navegando

9. Criterios de Aceptación Globales

Para considerar el proyecto MVP completo, se validaron los siguientes criterios de aceptación:

Autenticación y Seguridad

- ✓ **CA-001:** Sistema de registro funcional con validaciones completas
- ✓ **CA-002:** Auto-login después de registro exitoso
- ✓ **CA-003:** Login con redirección por rol (cliente/instructor/admin)
- ✓ **CA-004:** Logout limpia sesión y redirige a Landing
- ✓ **CA-005:** Triggers SQL crean automáticamente registro en `clientes`

Roles y Permisos

- ✓ **CA-006:** Navbar dinámico cambia según rol autenticado
- ✓ **CA-007:** RLS implementado en todas las tablas
- ✓ **CA-008:** Clientes solo ven sus propios datos
- ✓ **CA-009:** Instructores solo ven sus propias clases y reservas
- ✓ **CA-010:** Admins tienen acceso completo a gestión de clientes

Sistema de Paquetes

- ✓ **CA-011:** Compra de paquetes funcional con cálculo automático de vencimiento
- ✓ **CA-012:** Validación de paquetes activos antes de permitir reservas
- ✓ **CA-013:** Badges visuales de estado (Activo/Por vencer/Agotado/Vencido)
- ✓ **CA-014:** Dashboard muestra paquetes del cliente con clases restantes

Sistema de Reservas

- ✓ **CA-015:** Calendario interactivo con navegación limitada (±2 meses)
- ✓ **CA-016:** Validación completa antes de crear reserva
- ✓ **CA-017:** Transacción atómica al reservar (descuento de clase + incremento capacidad)
- ✓ **CA-018:** Cancelación con validación de 3 horas mínimas
- ✓ **CA-019:** Devolución de clase al paquete al cancelar a tiempo
- ✓ **CA-020:** Reservas duplicadas bloqueadas (constraint UNIQUE)
- ✓ **CA-021:** Manejo correcto de timezones (sin desfases de fecha)

Gestión de Reservas

- ✓ **CA-022:** Tab "Próximas" muestra solo reservas confirmadas futuras
- ✓ **CA-023:** Tab "Historial" muestra últimas 10 reservas pasadas con badges
- ✓ **CA-024:** Tab "Canceladas" muestra últimas 5 reservas canceladas
- ✓ **CA-025:** Botón "Nueva Reserva" redirige a calendario

Panel de Instructor

- ✓ **CA-026:** Calendario instructor con código de colores por ocupación
- ✓ **CA-027:** Vistas mensual, semanal y diaria funcionales
- ✓ **CA-028:** Modal de clase muestra lista de estudiantes inscritos
- ✓ **CA-029:** Registro de asistencia solo para clases pasadas con reservas
- ✓ **CA-030:** Guardado masivo de asistencias funcional
- ✓ **CA-031:** Creación de clases con validación de solapamiento
- ✓ **CA-032:** Repetición semanal automática de clases

Panel de Administrador

- ✓ **CA-033:** Gestión de clientes con búsqueda en tiempo real
- ✓ **CA-034:** Filtros por estado de paquetes
- ✓ **CA-035:** Modal con información completa de cliente
- ✓ **CA-036:** Creación de instructores con contraseña temporal
- ✓ **CA-037:** Estadísticas generales actualizadas en tiempo real

Experiencia de Usuario

- ✓ **CA-038:** Página 404 personalizada con sugerencias por rol
- ✓ **CA-039:** Página de ayuda (FAQ) accesible para todos
- ✓ **CA-040:** Landing page informativa con sucursales y planes
- ✓ **CA-041:** Feedback visual en todas las operaciones (loading, success, error)
- ✓ **CA-042:** Responsive design funcional en móviles y escritorio

Calidad de Código

- ✓ **CA-043:** TypeScript sin errores de compilación
- ✓ **CA-044:** Composables reutilizables para lógica de negocio
- ✓ **CA-045:** Código limpio y bien organizado por carpetas
- ✓ **CA-046:** Commits siguiendo Conventional Commits
- ✓ **CA-047:** README completo con documentación técnica

10. Fases de Implementación (Retroactivo)

Este proyecto fue desarrollado entre Diciembre 2025 y Enero 2026. A continuación se documenta el proceso de implementación real.

Fase 1: Fundación (Semana 1) - ✅ COMPLETADA

Objetivos:

- Setup del proyecto Vue 3 + TypeScript + Vite
- Configuración de Tailwind CSS
- Estructura de carpetas base
- Configuración de Vue Router
- Creación de navbar dinámico

Entregables:

- ✓ Proyecto inicializado con Vite
- ✓ Carpetas organizadas (views, components, composables, stores)
- ✓ Navbar dinámico funcional
- ✓ Rutas básicas configuradas

Fase 2: Integración con Supabase (Semana 2) - ✅ COMPLETADA

Objetivos:

- Crear proyecto en Supabase
- Diseñar esquema de base de datos (8 tablas)

- Implementar Row Level Security (RLS)
- Crear triggers y funciones SQL
- Integrar Supabase Auth
- Implementar AuthStore con Pinia

Entregables:

- ✓ Base de datos PostgreSQL configurada
- ✓ 8 tablas creadas con relaciones correctas
- ✓ Políticas RLS implementadas
- ✓ Trigger automático para crear clientes
- ✓ Funciones SQL (is_admin, get_user_email)
- ✓ AuthStore funcional con persistencia
- ✓ Sistema de login/registro operativo

Fase 3: Vistas Públicas y Cliente (Semana 3) - ✅ COMPLETADA

Objetivos:

- Landing Page informativa
- Página de Planes
- Dashboard Cliente
- Sistema de compra de paquetes
- Calendario de reservas

Entregables:

- ✓ LandingPageView con información del estudio
- ✓ PlanesView con catálogo de paquetes
- ✓ DashboardClienteView con información personalizada
- ✓ Sistema de compra de paquetes funcional
- ✓ CalendarioClienteView con FullCalendar
- ✓ MisReservasView básica

Fase 4: Sistema de Reservas Completo (Semana 4) - ✅ COMPLETADA

Objetivos:

- Implementar lógica completa de reservas
- Validaciones de negocio robustas
- Sistema de cancelación con restricciones
- Tabs en MisReservasView
- Resolver bug de timezone

Entregables:

- ✓ Transacción atómica al reservar
- ✓ Validación completa (paquete activo, capacidad, duplicados)
- ✓ Cancelación con validación de 3 horas
- ✓ Tabs: Próximas, Historial, Canceladas
- ✓ Fix timezone (parseo manual de fechas)

Fase 5: Panel de Instructor (Semana 5) - ✅ COMPLETADA

Objetivos:

- Calendario de instructor con FullCalendar
- Código de colores por ocupación
- Registro de asistencias
- Crear clases individuales y recurrentes

Entregables:

- ✓ CalendarioInstructorView con vistas múltiples
- ✓ Semáforo de ocupación (Gris/Azul/Naranja/Rojo)
- ✓ RegistroAsistenciaView funcional

- ✓ Guardado masivo de asistencias
- ✓ CrearClasesView con repetición semanal
- ✓ Validación de solapamiento de horarios

Fase 6: Panel de Administrador (Semana 6) - ✅ COMPLETADA

Objetivos:

- Gestión completa de clientes
- Búsqueda y filtros en tiempo real
- Crear instructores
- Estadísticas generales

Entregables:

- ✓ GestionClientesView con tabla completa
- ✓ Buscador por nombre/email/teléfono
- ✓ Filtros por estado de paquetes
- ✓ Modal con información detallada de cliente
- ✓ CrearInstructorView funcional
- ✓ Generación de contraseña temporal

Fase 7: Pulido y Testing (Semana 7) - ✅ COMPLETADA

Objetivos:

- Página 404 personalizada
- Página de Ayuda (FAQ)
- Validación de todas las HU
- Testing manual exhaustivo
- Refinamiento de UI/UX

Entregables:

- ✓ NotFoundView con sugerencias por rol
- ✓ AyudaView con preguntas frecuentes
- ✓ Todas las 20 HU validadas
- ✓ Testing en múltiples roles
- ✓ UI/UX refinada
- ✓ README completo y actualizado

Fase 8: Deploy y Documentación (Semana 8) - ✅ COMPLETADA

Objetivos:

- Deploy a producción
- Documentación técnica completa
- Documentación de usuario
- Video demo (opcional)

Entregables:

- ✓ Aplicación desplegada en Vercel/Netlify
- ✓ README técnico completo
- ✓ Documentación de arquitectura
- ✓ 20 Historias de Usuario documentadas
- ✓ Proyecto listo para portafolio

11. Métricas de Éxito del Proyecto

Métricas Técnicas

Métrica	Objetivo	Resultado	Estado
Tiempo de carga inicial	< 3 segundos	~1.5 segundos	✓ CUMPLIDO
Errores TypeScript	0 errores de compilación	0 errores	✓ CUMPLIDO
Cobertura RLS	100% de tablas protegidas	8/8 tablas con políticas	✓ CUMPLIDO
Triggers funcionales	Creación automática de cliente	100% operativo	✓ CUMPLIDO
Transacciones atómicas	Sin inconsistencias en reservas	0 inconsistencias detectadas	✓ CUMPLIDO
Responsive design	Funcional en móvil y desktop	100% responsive	✓ CUMPLIDO

Métricas Funcionales

Funcionalidad	Criterio de Éxito	Estado
Registro de usuarios	Auto-login funcional sin pasos manuales	✓ CUMPLIDO
Compra de paquetes	Cálculo automático de vencimiento correcto	✓ CUMPLIDO
Sistema de reservas	Validaciones completas sin excepciones	✓ CUMPLIDO
Cancelación	Restricción de 3 horas respetada al 100%	✓ CUMPLIDO
Registro asistencia	Guardado masivo sin pérdida de datos	✓ CUMPLIDO

Funcionalidad	Criterio de Éxito	Estado
Gestión de clientes	Búsqueda en tiempo real instantánea	✓ CUMPLIDO

Métricas para Portafolio

- ✓ **Arquitectura escalable:** Código organizado con separación clara de responsabilidades
- ✓ **Código limpio:** Composables reutilizables, sin duplicación
- ✓ **Documentación completa:** README técnico de 500+ líneas
- ✓ **Historias de Usuario:** 20 HU documentadas con criterios de aceptación
- ✓ **Seguridad:** RLS implementado profesionalmente
- ✓ **Buenas prácticas:** Conventional Commits, TypeScript, ESLint
- ✓ **Complejidad técnica:** Transacciones atómicas, triggers SQL, manejo de timezones
- ✓ **Defensible en entrevistas:** Decisiones arquitectónicas documentadas

Problemas Resueltos Destacables

1. Bug de Timezone en Calendario

Problema: Las reservas se guardaban con 1-2 días de diferencia.

Causa raíz: `new Date('YYYY-MM-DD')` se interpreta como UTC 00:00, causando conversión a zona horaria local.

Solución: Parseo manual de fechas sin conversión automática de timezone.

```
// ❌ Incorrecto (UTC conversion)
const date = new Date('2026-01-15')

// ✅ Correcto (local time)
const [year, month, day] = '2026-01-15'.split('-').map(Number)
const date = new Date(year, month - 1, day)
```

Aprendizaje: Siempre parsear fechas manualmente cuando se trabaja con calendarios y reservas.

2. Auto-login después de Registro

Problema: Usuario debía iniciar sesión manualmente después de registrarse.

Solución: Ejecutar `signIn()` inmediatamente después de `signUp()` exitoso.

```
// 1. Crear cuenta
await supabase.auth.signUp({ email, password })

// 2. Crear perfil
await supabase.from('profiles').insert({ ... })

// 3. Auto-login
await supabase.auth.signIn({ email, password })

// 4. Inicializar AuthStore
await authStore.initialize()

// 5. Redirigir
router.push('/dashboard-cliente')
```

Aprendizaje: La experiencia de usuario es crítica - eliminar pasos innecesarios.

3. RLS con Triggers

Problema: Las políticas RLS bloqueaban la creación automática de clientes por el trigger.

Solución: Usar `SECURITY DEFINER` en la función del trigger.

```
CREATE OR REPLACE FUNCTION public.handle_new_profile()
RETURNS TRIGGER
SECURITY DEFINER  -- ← Ejecuta con permisos de admin
SET search_path = public
```

```
LANGUAGE plpgsql
AS $
BEGIN
  IF NEW.rol = 'cliente' THEN
    INSERT INTO public.clientes (profile_id)
    VALUES (NEW.id);
  END IF;
  RETURN NEW;
END;
$;
```

Aprendizaje: Los triggers necesitan permisos elevados para ejecutar operaciones que RLS bloquearía.

12. Lecciones Aprendidas y Próximos Pasos

12.1 Lecciones Aprendidas

Arquitectura y Organización

- ✓ **Composables son poderosos:** Separar lógica de presentación facilita testing y reutilización
- ✓ **Navbar dinámico > Múltiples navbars:** Un solo componente con lógica condicional es más mantenible
- ✓ **TypeScript previene bugs:** Los tipos detectan errores en desarrollo que serían difíciles de encontrar en producción
- ✓ **AuthStore como fuente única de verdad:** Centralizar estado de autenticación evita inconsistencias

Base de Datos y Backend

- ✓ **RLS es esencial:** Nunca confiar solo en validaciones de frontend
- ✓ **Triggers reducen complejidad:** La lógica automática en DB es más confiable que en frontend
- ✓ **Transacciones atómicas son críticas:** Operaciones multi-tabla deben ser todo-o-nada
- ✓ **SECURITY DEFINER con cuidado:** Potente pero puede ser riesgoso si no se usa correctamente

Frontend y UX

- ✓ **Feedback visual es fundamental:** Loading states, success/error messages mejoran la percepción
- ✓ **Validaciones en tiempo real:** Mejor UX que mostrar errores después de submit

- ✓ **Timezones son traicioneros:** Siempre probar con datos reales y diferentes zonas horarias
- ✓ **Responsive design desde el inicio:** Más fácil que refactorizar después

Planificación y Documentación

- ✓ **Historias de Usuario guían desarrollo:** Cada HU es un contrato claro de funcionalidad
- ✓ **Documentar decisiones arquitectónicas:** Esencial para defensa en entrevistas
- ✓ **README técnico es tu carta de presentación:** Invierte tiempo en hacerlo profesional
- ✓ **Conventional Commits facilitan navegación:** Historial de commits claro es valioso

12.2 Funcionalidades Futuras (Versión 2.0)

Prioridad Alta

- ⚠ **Notificaciones en tiempo real:** Supabase Realtime para alertas de reservas
- ⚠ **Sistema de pagos:** Integración con Stripe/PayPal para compras reales
- ⚠ **Reportes administrativos:** Dashboard con gráficas de ocupación, ventas, etc.
- ⚠ **Gestión de sucursales:** Multi-tenant para varios estudios

Prioridad Media

- ⚠ **Cambio de contraseña:** Permitir a usuarios actualizar credenciales
- ⚠ **Recuperación de contraseña:** Flow completo de reset password
- ⚠ **Lista de espera:** Para clases llenas, permitir inscripción en waitlist
- ⚠ **Sistema de ratings:** Clientes califican clases/instructores

Prioridad Baja

- ⚠ **Modo oscuro:** Toggle entre light/dark theme
- ⚠ **Exportar calendario:** Integración con Google Calendar, iCal

- ⚠ **Chat instructor-cliente:** Mensajería interna
- ⚠ **Estadísticas personales:** Gráficas de asistencia, progreso del cliente

12.3 Mejoras Técnicas Futuras

Performance

- Implementar lazy loading de componentes pesados
- Optimizar queries con indices en DB
- Cachear respuestas frecuentes con Supabase Edge Functions
- Implementar paginación en listas largas

Testing

- Unit tests con Vitest para composables
- Integration tests para flujos críticos
- E2E tests con Playwright/Cypress
- Tests de RLS policies

DevOps

- CI/CD con GitHub Actions
- Ambientes de staging y producción separados
- Monitoreo con Sentry o similar
- Analytics con Google Analytics o Plausible

12.4 Conclusión

Core Strong Studio Pilates es un proyecto completo que demuestra capacidad profesional en desarrollo frontend moderno y arquitectura de aplicaciones web.

> ### Logros Principales:
>
> - ✓ Sistema funcional end-to-end con 4 roles diferenciados
> - ✓ Arquitectura escalable y mantenible

> - ✓ Seguridad implementada a nivel de base de datos
> - ✓ 20 Historias de Usuario completadas al 100%
> - ✓ Código limpio y defendible en entrevistas técnicas
> - ✓ Documentación profesional y completa

Este proyecto representa aproximadamente **160 horas de desarrollo** distribuidas en 8 semanas de trabajo enfocado.

Estado final: ✅ **PROYECTO COMPLETADO Y LISTO PARA PORTAFOLIO**

Fecha de Especificación: 18 de Enero, 2026

Versión del Documento: 1.0 (Retroactiva)

Autor: Juan Carlos Quiñonez Madrid

Email: b4rc4drid@gmail.com

GitHub: CarlosMadrid11