

DOCUMENTACIÓN TÉCNICA

SOMA

Centro de Entrenamiento

Documentación Técnica — Sistema SOMA

1. Introducción

SOMA es una aplicación de escritorio para la gestión de un gimnasio. Permite administrar los turnos semanales, los pagos de los socios y los datos de los usuarios. Reemplaza el registro manual en papel que se usaba antes, con el que era lento ver quién ya pagó el mes y quién no, o actualizar los datos de un socio.

La aplicación corre en Windows y macOS, y está construida con Flutter (un framework para crear aplicaciones que funcionan en varias plataformas con un mismo código). Los datos del sistema se guardan en un servidor en la nube, con el motor de Bases de Datos PostgreSQL. Este documento describe cómo está armado el sistema por dentro.

2. Arquitectura general

El sistema se compone de dos partes principales. Por un lado, la aplicación de escritorio, que se ejecuta, principalmente, en la computadora del gimnasio y es con la que interactúa el personal. Por otro lado, todo el backend vive en la nube, hosteado por el servicio Supabase, en su versión gratuita. La aplicación no almacena datos de negocio localmente, toda la información se consulta y se modifica contra ese servidor.

Un aspecto central del diseño es que la aplicación nunca accede de manera directa a las tablas de la base de datos. Cada vez que necesita leer o escribir información, solicita al servidor la ejecución de una función de la Base de Datos, previamente definida, perteneciente al esquema `public`, y recibe el resultado que esta retorna. Estas funciones, destinadas a ser llamadas desde fuera de la DB, se identifican con el prefijo `fc_` en su nombre.

Otro esquema en la DB (`internal`) maneja la lógica más crítica de la Base de Datos, como los roles, permisos, etc. Las funciones internas no llevan el prefijo `fc_` y no pueden ser llamadas desde el exterior.

La ventaja de este esquema es que centraliza toda la lógica sensible del sistema, tal como las validaciones y las reglas del negocio, en un único lugar.

El frontend de la aplicación está organizado en módulos, correspondientes a las distintas áreas de gestión del gimnasio: pagos, horarios, turnos y usuarios, entre otras. Dentro de cada módulo, el código se estructura en tres capas, siguiendo la “*clean architecture*”. La capa de presentación comprende las pantallas y todos los elementos con los que interactúa el usuario. La capa de dominio describe los conceptos propios del gimnasio junto con sus reglas, con independencia de la tecnología empleada. La capa de datos contiene el código encargado de comunicarse con el servidor, es decir, el que realiza las llamadas RPC (llamadas a las funciones `fc_` de la DB).

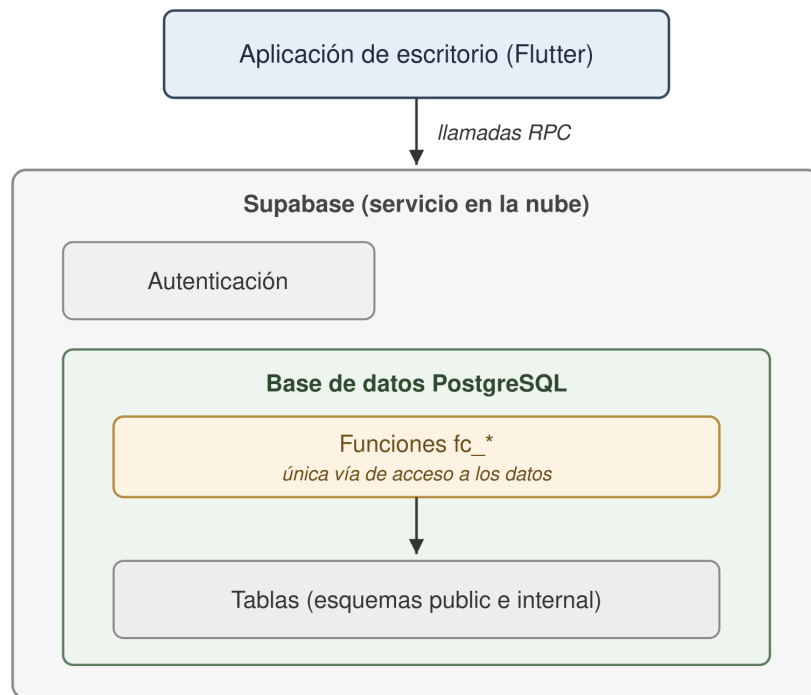


Figura 1. Arquitectura general del sistema.

Existen dos copias independientes del backend: un entorno de desarrollo, destinado a probar cambios sin riesgo, y un entorno de producción, que es el que utiliza efectivamente el gimnasio. Al compilar la aplicación se determina contra cuál de los dos entornos operará.

3. Modelo de datos

Los datos del sistema residen en la base PostgreSQL, organizados en los dos esquemas mencionados: **public**, con la información de la operación cotidiana, e **internal**, con la configuración interna. En esta sección se describen las tablas principales, agrupadas por área. Se omiten las columnas por simplificar.

Área	Tabla	Qué representa
Usuarios	usuarios	Personas del sistema clientes y administradores; el campo rol define sus permisos
Pagos	pagos	Cada pago registrado de un socio
Pagos	tipos_cuota	Modalidades de cuota: precio, cantidad de días por semana, recargo

Área	Tabla	Qué representa
Pagos	<code>metodos_pago</code>	Formas de pago disponibles (efectivo, transferencia, entre otras)
Horarios	<code>actividades</code>	Actividades o clases que ofrece el gimnasio
Horarios	<code>horario_actividad</code>	Grilla semanal: qué actividad se dicta cada día y en qué franja horaria
Turnos	<code>turnos</code>	Instancias concretas de una actividad en una fecha y hora determinadas
Reservas	<code>reservas</code>	Reserva de un socio para un turno
Horarios	<code>dias_especiales</code>	Fechas de cierre o con un horario diferente al habitual
Reservas	<code>reservas_huerfanas</code>	Reservas que quedaron sin turno válido tras un cambio de grilla
Trazabilidad	<code>eventos</code>	Registro de los cambios importantes, a modo de auditoría
Configuración	<code>app_config</code>	Parámetros ajustables del sistema (en el esquema internal)

Cada persona registrada en el sistema es una fila de la tabla `usuarios`, y su campo `rol` determina qué puede hacer: un `cliente` tiene permitidas pocas acciones (y por ende funciones en la DB), un `admin` tendrá permitidas la mayoría de las acciones y un `superadmin` no tiene restricciones en el sistema.

Cada cliente tiene asignado un tipo de cuota, que define el precio que paga y la cantidad de días por semana a los que puede asistir.

El registro de pagos gira en torno a la tabla `pagos`. Cada fila corresponde a un pago de un socio e indica con qué método se abonó, tomado de `metodos_pago`, y a qué tipo de cuota corresponde, tomado de `tipos_cuota`.

La gestión de asistencias se organiza en cascada. El gimnasio ofrece un conjunto de actividades, que son las clases disponibles. La tabla `horario_actividad` define la grilla semanal: qué actividad se dicta cada día y en qué horario. A partir de esa grilla se generan los turnos, que son las instancias concretas de una actividad en una fecha y hora puntuales. Cada reserva vincula a un socio con un turno.

Los días especiales registran fechas en las que el gimnasio cierra o funciona con un horario distinto del habitual. Las reservas huérfanas son reservas que quedan sin un turno válido cuando cambia la grilla horaria; el sistema las conserva para poder reubicarlas más adelante en lugar de descartarlas en silencio.

La tabla `eventos` funciona como un registro de auditoría: guarda los cambios importantes, por ejemplo, la anulación de un pago, junto con su valor anterior y posterior y el usuario que lo hizo, de modo que siempre sea posible reconstruir qué ocurrió.

4. Módulos funcionales

El sistema se organiza en módulos, cada uno correspondiente a un área de gestión del gimnasio. Esta sección describe cada módulo desde tres perspectivas: qué permite hacer, cuáles son sus conceptos centrales y qué decisiones de diseño lo caracterizan. No se detalla el recorrido de las pantallas, que corresponde al manual de usuario.

4.1 Usuarios, roles y autenticación

Todas las personas vinculadas al gimnasio, tanto los clientes como el personal administrativo, se registran en la tabla de `usuarios`. Cada usuario tiene asignado un rol, que puede ser uno de tres: `cliente`, `admin` o `superadmin`. Usuarios con rol `cliente` no pueden ingresar a la app de escritorio. Los roles `admin` y `superadmin` corresponden al personal que gestiona el sistema desde la aplicación de escritorio, y es `superadmin` el que dispone del mayor nivel de permisos.

El sistema resuelve por su cuenta la autenticación. El proceso funciona de la siguiente manera. Al iniciar sesión, el usuario ingresa su DNI y su contraseña, y la aplicación invoca una función del servidor que verifica esas credenciales contra los datos almacenados. La contraseña no se guarda en texto plano, sino que se guarda su hash (más salt). Si las credenciales son correctas, la función devuelve un token de sesión, que es un número aleatorio muy grande con el que el servidor identifica esa sesión en particular.

Cada request que la aplicación hace al servidor incluye ese token como prueba de identidad. El servidor lo valida y aplica los permisos que correspondan al rol del usuario. Las sesiones tienen una duración limitada: pasado cierto tiempo de inactividad, el token expira y el usuario debe volver a iniciar sesión.

4.2 Pagos

El módulo de pagos digitaliza el registro de las cuotas mensuales de los socios, que antes se llevaba a mano en un cuaderno. Permite registrar un pago, consultarlos, corregirlos y anularlos (estos últimos dos bajo ciertas condiciones). En el estado actual todos los pagos corresponden al mismo concepto, la cuota mensual de acceso al gimnasio, aunque el

diseño prevé que en el futuro puedan sumarse otros. Cada pago deja constancia de a qué socio se le cobró, a qué mes corresponde, cuánto se cobró, con qué método, qué tipo de cuota estaba pagando, en qué fecha y quién registró el cobro.

El requerimiento que gobierna todo el módulo es que **la información de pagos no se pierde**. Ningún pago se elimina realmente: a lo sumo se oculta de la vista cotidiana, pero siempre queda consultable, gracias a la tabla de eventos. Toda operación sobre un pago debe poder reconstruirse después, incluyendo quién la hizo, cuándo y qué cambió.

El sistema permite cuatro operaciones: registrar un pago, consultarlos, corregir uno ya registrado y anularlo. No existe la operación de borrar. La corrección de un pago está permitida sólo durante una ventana de tiempo acotada tras su registro; pasada esa ventana, el pago queda congelado y ya no admite correcciones en su lugar. La ventana existe para enmendar errores recientes (un monto mal tipeado, un método de pago equivocado detectado minutos después). Toda corrección queda registrada.

La anulación es el equivalente digital de tachar el pago en el cuaderno: el pago deja de contar como un cobro vigente y se muestra visiblemente marcado como anulado, pero sigue existiendo y pudiéndose consultar, junto con el motivo de la anulación, quién la hizo y cuándo. La anulación es terminal: un pago anulado no vuelve a corregirse ni a "desanularse". Para enmendar un pago viejo que ya quedó fuera de la ventana de corrección, se lo anula y se registra uno nuevo.

Para el caso de los usuarios de rol **admin**, tanto la corrección de un pago como la anulación del mismo, es posible sólo si el usuario quien realiza la acción es quien hizo el registro en un primer lugar. Los **superadmin** pueden anular, en cualquier momento, incluso fuera de la ventana y además pueden anular o editar pagos registrados por cualquier otro usuario (pero aún queda registrado quién hace la edición/anulación).

Operación	Efecto sobre el pago	Fuera de la ventana de tiempo
Corregir	Reescribe el pago en su lugar	No se permite a ningún rol
Anular	Preserva el pago y lo marca como anulado	Permitido a un superadmin

Todo este esquema se apoya en la tabla **eventos** descrita en el modelo de datos: cada corrección y cada anulación se guarda allí, con el valor anterior y el posterior. Esa es la pieza que permite cumplir el requerimiento de no perder información aun cuando los datos cambian.

4.3 Horarios, turnos y reservas

Este módulo administra las reservas de los socios. Su estructura es la siguiente: En la base están las actividades, que son las categorías de uso del gimnasio, como Musculación o Funcional. Cada actividad define su propio cupo y la duración de sus clases. Luego, los planes determinan a qué actividades tiene acceso cada socio. Sobre las actividades se define el horario regular, es decir, la grilla semanal que establece qué actividad se dicta cada día y en qué franja horaria. A partir de esa grilla, el sistema genera los turnos, que son las instancias concretas de una actividad en una fecha y hora determinadas. Y sobre cada turno se registran las reservas, que vinculan a un socio con un turno puntual.

El horario regular no es fijo en el tiempo. El personal puede programar un cambio de grilla con una fecha futura de entrada en vigencia, de manera de anunciarlo con anticipación: el horario nuevo queda cargado y convive con el vigente hasta que llega su fecha. Además, los días especiales permiten registrar excepciones sobre una fecha concreta. Ejemplo, un feriado en que el gimnasio cierra, una mañana sin actividad, o incluso un único turno que no se dicta.

Para que un cliente pueda reservar un turno, el sistema verifica varias condiciones de manera simultánea: que su plan habilite esa actividad, que el turno tenga cupo disponible, que el socio esté al día con sus pagos, que no haya agotado la cantidad de reservas semanales que su plan permite, y que no tenga ya otra reserva en el mismo horario. Si alguna condición no se cumple, el sistema le indica el motivo específico en lugar de un rechazo genérico. El personal administrativo verá el incumplimiento de estas reglas en forma de advertencias, pero se le permitirá ignorarlas, por ejemplo para anotar a un socio en un turno lleno. Es decir, las advertencias son informativas, no bloqueantes.

Ningún cambio de horario se borra reservas en silencio. Cuando una modificación deja a una reserva sin un turno válido, esa reserva se convierte en una reserva huérfana, que es un registro explícito de que un socio se quedó sin su turno. Desde ahí, el operador puede reasignar un turno equivalente a la reserva huérfana y/o dar un aviso al cliente con un mensaje de whatsapp.

4.4 Configuración

Varios comportamientos del sistema dependen de valores que conviene poder ajustar sin modificar el código. En lugar de fijarlos directamente en la programación, el sistema los concentra en un conjunto de parámetros de configuración guardados en el esquema `internal`. Cada parámetro tiene un valor por defecto, de manera que el sistema funciona aunque no se lo configure de forma explícita, pero ese valor puede cambiarse cuando la operación del gimnasio lo requiere.

Algunos ejemplos de estos parámetros son el umbral de morosidad (cuántos meses sin pagar hacen que un socio quede impedido de reservar, con un valor por defecto de dos meses) y la ventana de retención de pagos, que indica por cuánto tiempo se conservan los pagos en la base.

5. Entorno y operación

El sistema se despliega sobre dos entornos independientes, cada uno un proyecto de Supabase separado: uno de desarrollo y uno de producción. La razón de la separación es poder introducir y probar cambios sin poner en riesgo el sistema que el gimnasio usa a diario. El entorno de desarrollo se creó como una copia del de producción, en la misma región y con la misma versión de PostgreSQL, para que lo que se prueba en uno se comporte igual en el otro. El entorno de producción es el proyecto original, ya validado por el uso real.

Todo cambio en la base de datos, una tabla nueva, una función, un trigger o cualquier otra modificación; se promueve de desarrollo a producción por medio de migraciones. Una migración es un archivo con las instrucciones SQL del cambio (lo que se llama Delta SQL), que queda versionado junto con el código del sistema. Los dos entornos están registrados en este mecanismo de migraciones, de modo que la historia de cambios de la base es la misma para ambos y un entorno puede reconstruirse desde cero aplicando las migraciones en orden. La regla que ordena el proceso es que producción nunca recibe un cambio que no haya pasado antes por desarrollo con esta misma forma.

Del lado de la aplicación, el entorno contra el que se ejecuta se decide en el momento de compilarla, a partir de un archivo de configuración que indica a qué servidor apuntar. Ese archivo contiene únicamente datos públicos: la dirección del servidor y una clave de acceso pensada de antemano para ser pública. Cuando la aplicación corre contra el entorno de desarrollo, muestra una cinta roja con la leyenda “DEV”, para que nunca haya dudas sobre si se está trabajando con datos reales o de prueba.

Dado que los mismos comandos podrían ejecutarse contra cualquiera de los dos entornos, el sistema incorpora varias defensas para no operar sobre producción por error. El comportamiento por defecto de las herramientas apunta siempre a desarrollo, y actuar sobre producción exige indicarlo de forma explícita. Además, cada base de datos guarda su propia identidad (“dev” o “prod”) en una tabla interna que las herramientas pueden consultar antes de actuar, y toda modificación sobre producción requiere confirmarla escribiendo la palabra “PROD”.