MANUAL

Técnico

Una guía para la implementación y ejecución correcta del aplicativo.

CuidaDos

ÍNDICE

1. INTRODUCCIÓN

El presente manual técnico describe la arquitectura interna, el proceso de instalación y los protocolos de despliegue de **cuidaDos**, una plataforma web (Single Page Application) destinada a la gestión de servicios asistenciales.

El sistema ha sido construido desacoplando el frontend del backend, utilizando una arquitectura *serverless* [1]para la base de datos y un enfoque reactivo en la interfaz de usuario. Este documento está dirigido a desarrolladores y administradores de sistemas que requieran mantener, auditar o escalar la aplicación.

2. OBJETIVOS

Este manual técnico tiene como objetivo la especificación los componentes y recursos técnicos del sistema al usuario final, con la finalidad de apoyar situaciones de mantenimiento o mejoras aplicadas

2.1 Objetivos técnicos

Más allá de la funcionalidad de negocio, la implementación persigue los siguientes hitos de ingeniería:

- Rendimiento y Adaptabilidad:
  - Server-Side Rendering (SSR): Se ha implementado el renderizado en servidor nativo de Angular 21 para minimizar el *First Contentful Paint (FCP)* y mejorar la indexación (SEO).
  - Optimización del First Contentful Paint (FCP)
    El sistema prioriza la velocidad de percepción del usuario mediante la optimización de la métrica FCP. En aplicaciones SPA tradicionales, el usuario experimenta una pantalla en blanco hasta que el bundle de JavaScript se descarga y ejecuta. El navegador recibe un documento HTML completo, permitiendo que el FCP ocurra milisegundos después de la

[1] Modelo de ejecución en la nube donde la asignación de recursos es dinámica y gestionada por terceros (Supabase y Vercel). En este esquema, la base de datos PostgreSQL actúa como un servicio independiente capaz de exponer sus datos vía API REST y WebSockets sin requerir un servidor intermedio dedicado. Se delega la seguridad y el mantenimiento de la infraestructura, permitiendo una alta disponibilidad y tolerancia a fallos desde el primer despliegue.

solicitud, mejorando drásticamente la experiencia de usuario (UX) y el posicionamiento SEO, ya que los motores de búsqueda reciben contenido indexable de inmediato.

- Gestión de Entornos Isomórficos: Se ha configurado el servidor Express para manejar las diferencias entre el entorno de ejecución del servidor (Node.js) y el cliente (Browser). Específicamente, se ha implementado una estrategia de anulación de WebSockets en el lado del servidor para garantizar la compatibilidad de la librería cliente de Supabase durante la fase de prerenderizado, asegurando que la hidratación en el cliente ocurra sin bloqueos ni errores de conexión.

- Integridad: Garantizar la consistencia de datos mediante restricciones a nivel de base de datos y validación de esquemas en tiempo de ejecución (Zod).
- Escalabilidad: Implementación de un flujo de CI/CD (Integración y Despliegue Continuo) que permite actualizaciones automáticas sin tiempo de inactividad.
- Seguridad: Delegación de la autorización al motor de base de datos mediante políticas RLS (*Row Level Security*), asegurando que el aislamiento de datos sea inviolable desde el frontend.

3. REQUISITOS DEL SISTEMA

3.1 Requisitos para el Cliente (Usuario Final)

Para garantizar la correcta interpretación del código compilado y la estabilidad de la conexión en tiempo real, se establecen los siguientes requisitos:

1. Navegador Compatible con ESModules (ES2022): La aplicación ha sido compilada bajo el estándar ECMAScript 2022 para aprovechar las últimas optimizaciones de rendimiento de Angular 21. Se requiere soporte nativo para módulos de JavaScript.
    - *Google Chrome:* Versión 90 o superior.
    - *Mozilla Firefox:* Versión 90 o superior.
    - *Safari:* Versión 15 o superior.

- *Microsoft Edge:* (Chromium based) Versión 90 o superior.
- **Nota***:* Internet Explorer 11 no está soportado.

Motivo:

En el archivo tsconfig.json se ha configurado:

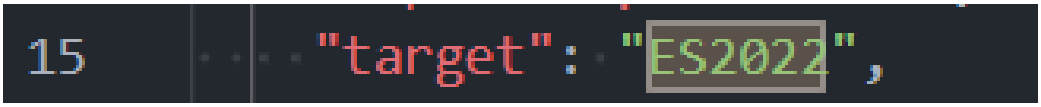


Ilustración 1: tsconfig.json

Angular está compilando el código a una versión muy moderna.

Si se usa un **navegador viejo** (ej. Chrome 80): La aplicación se romperá y saldrá una pantalla blanca porque el navegador no entenderá instrucciones modernas. ESModules: Angular 21 ya no genera scripts "clásicos", genera Módulos. Solo los navegadores modernos (Chrome 90+, etc.) saben cargar estos módulos de forma nativa.

- Conectividad de Red (WebSockets): Además del tráfico HTTP estándar (puerto 443), la red del usuario debe permitir conexiones WebSocket (WSS) salientes hacia el dominio de Supabase. El sistema implementa una arquitectura de Sincronización de Estado en Tiempo Real (Realtime Database), que, a diferencia de las aplicaciones web tradicionales que requieren recargar la página para ver cambios, **cuidaDos** mantiene una **conexión bidireccional** abierta para escuchar eventos INSERT, UPDATE y DELETE directamente desde el motor de base de datos (PostgreSQL WAL). Crítico para los siguientes módulos:

  1. Integridad de Catálogo (Servicios y Horarios): Si una Empresa elimina un hueco horario o modifica una tarifa, el cambio se refleja instantáneamente en la interfaz de los Clientes que están visualizando la oferta en ese momento, evitando errores de contratación por información obsoleta (*Race Conditions*).

  2. Gestión Administrativa (Usuarios y Empresas): Los paneles CRUD del administrador se actualizan en vivo. Si se registra una nueva empresa o se da de baja un usuario, la tabla de datos se refresca automáticamente sin intervención humana.

3. Gestión de los servicios, horarios y ofertas ofrecidas por las empresas: los administradores pueden gestionar los servicios y horarios que se ofrecen en tiempo real, así como visualizarlos y modificarlos en una tabla de gestión. Las empresas pueden gestionar (crear, ver, eliminar, modificar) las ofertas que ofrecen en tiempo real a su vez.

4. Mensajería y Notificaciones: Recepción instantánea de mensajes y actualizaciones de estado (leído/no leído).

Beneficio*:* Esta estrategia elimina la necesidad de realizar *Polling* (peticiones HTTP repetitivas cada X segundos para consultar cambios), reduciendo drásticamente el consumo de ancho de banda y la carga computacional tanto en el cliente como en el servidor.

3.2 Requisitos[2] para el Servidor de Desarrollo

Para transformar el código fuente (TypeScript, SASS) en una aplicación interpretable por los navegadores, se requiere un entorno de construcción local basado en las siguientes herramientas:

- Node.js (Runtime Environment)

  Versión: v18.13.0 (LTS) o superior. El proyecto cuenta actualmente con la versión: v22.17.0

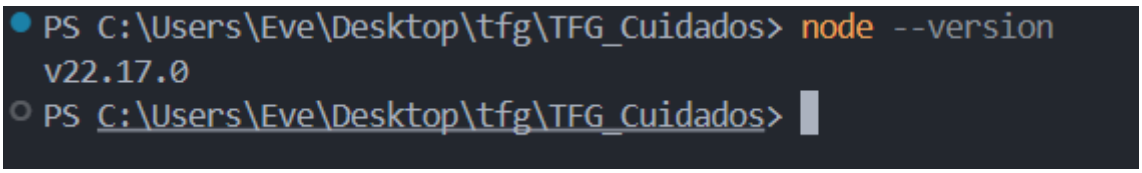


Ilustración 2. Node version

  Aunque **cuidaDos** no utiliza un servidor Node.js en producción (ya que el backend es Supabase), este entorno es indispensable durante la fase de desarrollo. Actúa como el motor de ejecución que permite al compilador de Angular

[2] Los comandos correspondientes para las instalaciones se encuentran en el apartado 5.1 Instalación de Prerrequisitos.

(Esbuild/Webpack) transpilat el código TypeScript a JavaScript (ECMAScript) y gestionar el renderizado del lado del servidor (SSR) en local.

- NPM (Node Package Manager)

  Versión: v10.0.0 o superior.

  Es el gestor de dependencias encargado de resolver e instalar las librerías listadas en el archivo package.json (como @angular/material, zod, supabase-js). Garantiza que todo el equipo de desarrollo y el servidor de despliegue (Vercel) utilicen exactamente las mismas versiones de cada paquete mediante el archivo de bloqueo package-lock.json.

  El proyecto actualmente cuenta con la versión: 11.4.2

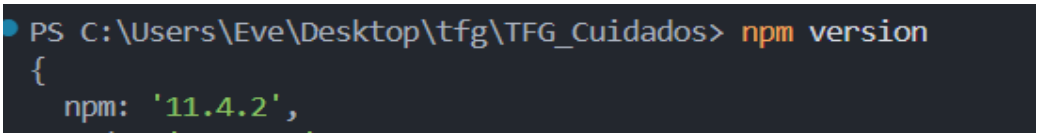


Ilustración 3: NPM versión.

Listado de dependencias:

Comando → npm list --depth=0

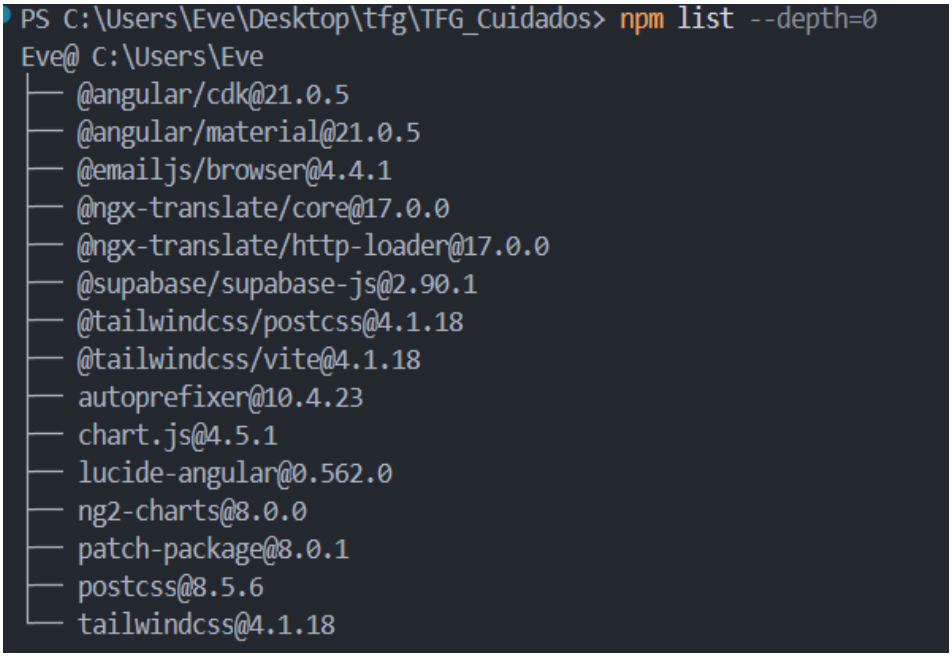


Ilustración 4: Lista de dependencias.

- Git (Control de Versiones)

  Herramienta fundamental para la Integración Continua (CI).

  Historial: Permite un control granular de los cambios en el código.

  Despliegue Automático: Vercel está conectado al repositorio remoto. Al detectar un git push en la rama main, Vercel descarga el código y ejecuta el script de construcción automáticamente. Sin Git, el flujo CI/CD no sería posible.

  Ejemplo de uso de git:

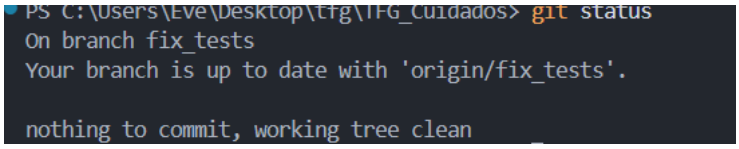


Ilustración 5: Git implementado.

  Git status demuestra que no hay cambios que realizar.

- Angular CLI (Command Line Interface)

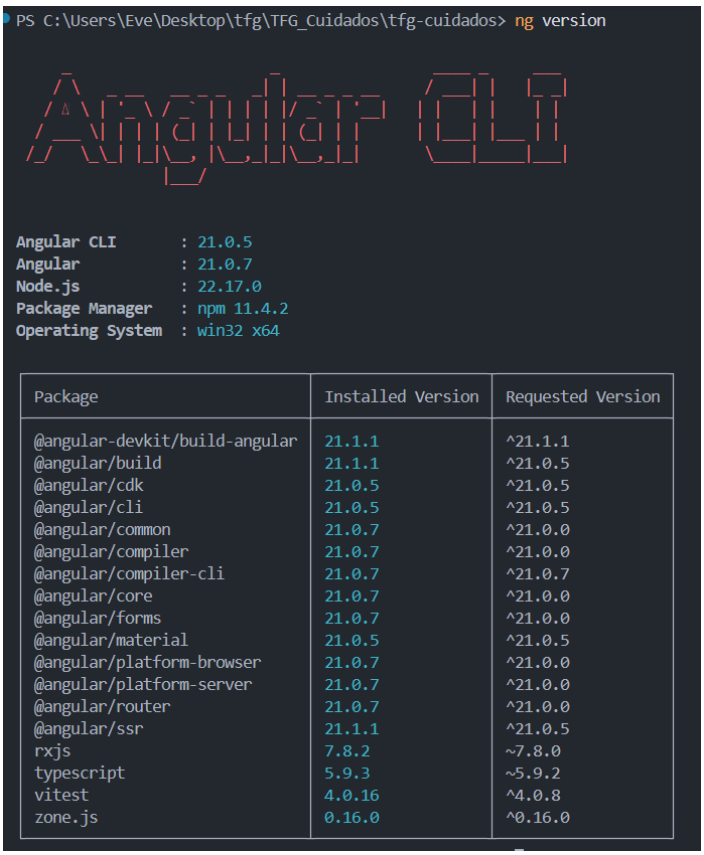


Ilustración 6: Angular cli.

Versión: v21.0.5.

Es la herramienta de ingeniería central del proyecto. Actúa como una capa de abstracción sobre el sistema de construcción (basado en *Esbuild* en las versiones modernas), gestionando el ciclo de vida completo del desarrollo de software sin necesidad de configurar manualmente herramientas complejas como Webpack

Es la herramienta de orquestación del framework. Abstrae la complejidad de la configuración de Webpack y permite:

- **vScaffolding (Generación de Estructura):** Automatiza la creación de artefactos de código (Componentes, Servicios, Interfaces) mediante el comando ng generate. Garantiza que todos los archivos sigan estrictamente la **Guía de Estilo oficial de Angular**, creando automáticamente los archivos de pruebas unitarias (.spec.ts) y actualizando las importaciones necesarias en los componentes *Standalone*.
- **vServing (Servidor de Desarrollo):** Despliega un servidor HTTP local en memoria a través del comando ng serve.

  Característica Clave: Implementa **HMR (Hot Module Replacement)**. Esta tecnología detecta cambios en el código TypeScript o CSS y actualiza el módulo específico en el navegador sin recargar la página completa, manteniendo el estado de la aplicación (como los datos de un formulario relleno) durante el desarrollo.

- **vBuilding (Compilación para Producción):** Ejecuta el pipeline de optimización mediante ng build. Este proceso transforma el código legible por humanos en código máquina eficiente para el navegador:
  - **Transpilación:** Convierte TypeScript a JavaScript (ECMAScript 2022).
  - **AOT (Ahead-of-Time):** Compila las plantillas HTML y CSS en instrucciones JavaScript durante la construcción, eliminando la necesidad de enviar el compilador de Angular al navegador.
  - **Tree-Shaking:** Elimina el código muerto (funciones o librerías importadas pero no utilizadas) para reducir el peso final.

- **Hashing:** Añade códigos únicos a los nombres de archivo (ej: main.7a2b3c.js) para gestionar la caché del navegador en futuras actualizaciones.

4. HERRAMIENTAS Y STACK TECNOLÓGICO

La arquitectura de cuidaDos se ha diseñado siguiendo el principio de Separation of Concerns (SoC), seleccionando librerías especializadas para cada capa de la aplicación. A continuación, se detalla la configuración técnica, la justificación de uso y los comandos de integración de cada componente.

4.1 Frontend: Core y Framework

Angular 21 (v21.0.5)

Framework principal del proyecto. Se ha configurado utilizando la arquitectura de Standalone Components, eliminando la complejidad de los NgModules tradicionales y reduciendo el *boilerplate*.

- Innovación (Signals & SSR):
  - Signals API: Se utiliza como primitiva reactiva principal para la gestión del estado, sustituyendo el uso intensivo de Zone.js. Esto permite una detección de cambios granular (solo se actualiza en el DOM lo que realmente cambia), mejorando el rendimiento en dispositivos de gama baja.
  - Hydration: Configurado con @angular/ssr para permitir una "hidratación no destructiva". El servidor envía el HTML pre-renderizado y Angular "despierta" la página sin parpadeos ni reconstrucción del DOM.
- Comando de Creación: El proyecto se inicializó habilitando SSR y el enrutamiento estricto:

ng new tfg-cuidados --ssr --style=css –routing

Verificación de la versión del framework (v21.0.5) y entorno Node.js.:

ng version

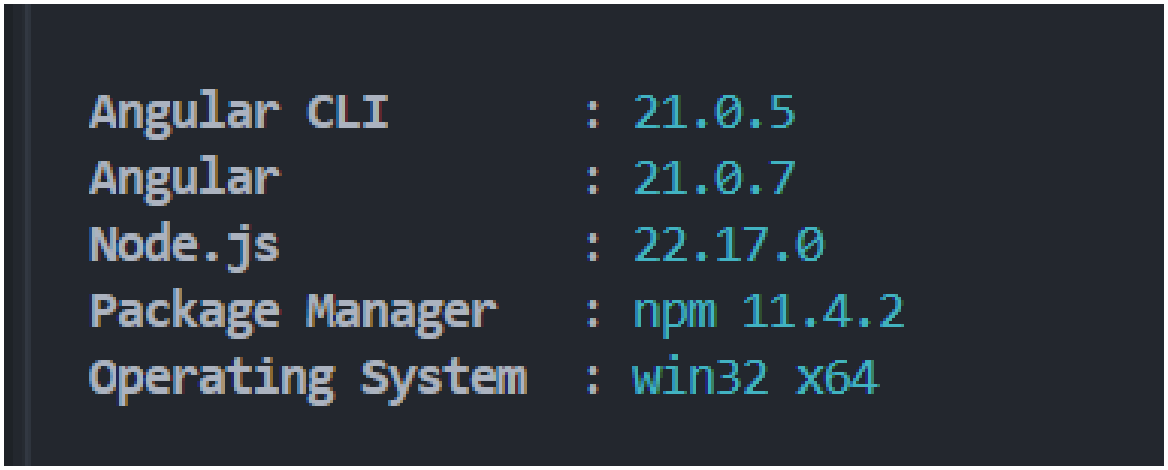


Ilustración 7: Versión de Angular.

4.2 Estilos e Interfaz de Usuario (UI)

4.2.1 Tailwind CSS (Motor de Estilos)

Framework *Utility-First*. A diferencia de las hojas de estilo tradicionales, Tailwind permite construir diseños personalizados directamente en el HTML. Se ha integrado mediante **PostCSS** (como se evidencia en el archivo .postcssrc.json), lo que permite purgar el CSS no utilizado (*PurgeCSS*) en tiempo de compilación, generando archivos de estilo extremadamente ligeros.

Comando de Instalación:

npm install -D tailwindcss postcss autoprefixer

npx tailwindcss init

Configuración del preprocesador PostCSS para la integración nativa de Tailwind:

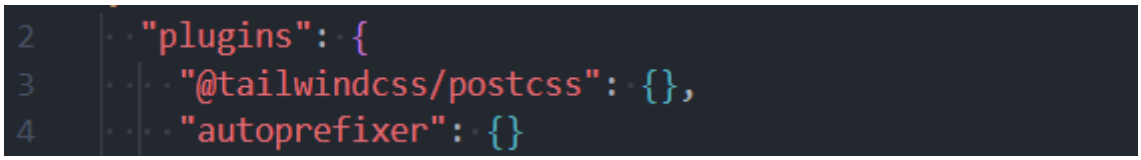


Ilustración 8: Archivo de Tailwind.

4.2.2 Angular Material (Componentes UI)

Biblioteca oficial utilizada para componentes complejos que requieren cumplir estrictamente con los estándares de accesibilidad (WCAG). Se emplea fundamentalmente en:

- MatDialog: Para ventanas modales de confirmación y formularios emergentes.
- MatTable: Para la visualización tabular de datos con paginación y ordenamiento en el panel de administración.

Comando de Instalación:

ng add @angular/material

4.2.3 Lucide Angular (Iconografía)

Set de iconos SVG optimizados. Se seleccionó frente a FontAwesome por su capacidad de **Tree-Shaking**: el compilador de Angular elimina automáticamente del *bundle* final cualquier icono que no se esté importando explícitamente, reduciendo el peso de la aplicación.

Comando de Instalación:

npm install lucide-angular

4.3 Backend & Persistencia (BaaS)

Supabase Client (@supabase/supabase-js)

SDK isomórfico que permite la comunicación con la infraestructura de Supabase.

- **Funcionalidad:** Gestiona la autenticación de usuarios (GoTrue), las consultas a la base de datos PostgreSQL (PostgREST) y mantiene los canales de WebSocket abiertos para la funcionalidad *Realtime* (chat y actualizaciones en vivo).
- **Integración:** Se ha implementado mediante un servicio inyectable (SupabaseService) que sigue el patrón **Singleton**, garantizando una única instancia de conexión durante el ciclo de vida de la sesión.

Comando de Instalación:

npm install @supabase/supabase-js

4.4 Librerías Auxiliares

- **Zod:** Validación estricta de esquemas TypeScript. Se usa para validar formularios antes de enviarlos a la API.:
  - **npm:** npm install zod

  - **yarn:** yarn add zod

- **EmailJS:** Servicio de notificaciones transaccionales.

  - **npm:** npm install @emailjs/browser
  - **yarn:** yarn add @emailjs/browser

- **Chart.js / ng2-charts:** Visualización de datos en el panel de administración.

  - **npm:** npm install chart.js ng2-charts
  - **yarn:** yarn add chart.js ng2-charts

4.5 Testing

El proyecto implementa una estrategia de pruebas en pirámide, asegurando que cada componente, desde la lógica interna hasta la interfaz de usuario y la comunicación con el servidor, funcione según los requerimientos.

4.5.1 Karma & Jasmine: Pruebas Unitarias

Las pruebas unitarias se centran en probar la unidad mínima de código (funciones, servicios o componentes aislados) para asegurar que la lógica de negocio sea robusta.

- **Jasmine:** Es el framework de desarrollo guiado por comportamiento (BDD) que proporciona la sintaxis para escribir las pruebas (describe, it, expect).
- **Karma:** Es el ejecutor de pruebas (*Test Runner*) que abre un navegador (Chrome por defecto) para ejecutar los tests y reportar los resultados en tiempo real.

Comandos de Instalación y Uso

Normalmente incluido en Angular CLI.

Si necesitas reinstalar:

npm install jasmine-core karma karma-jasmine karma-chrome-launcher --save-dev

Ejecución de pruebas (Modo Watch):

ng test

Ejecución para CI:

ng test --watch=false --browsers=ChromeHeadless

Detalles Técnicos

- **Ubicación:** Archivos con extensión .spec.ts junto al archivo fuente.
- **Aislamiento:** Se utiliza TestBed para crear entornos controlados, simulando dependencias mediante *Mocks* o *Spies* para no depender de servicios externos o APIs reales durante la prueba.

Ej. de página con test para auth:

Ilustración 9: Archivo para test Karma.

4.5.1.1 Reportes de karma

Reporte Visual en HTML (El más recomendado)

npm install karma-jasmine-html-reporter --save-dev

Generar el Reporte de Cobertura

Para "descargar" los resultados en un formato profesional (HTML):

ng test --watch=false --code-coverage

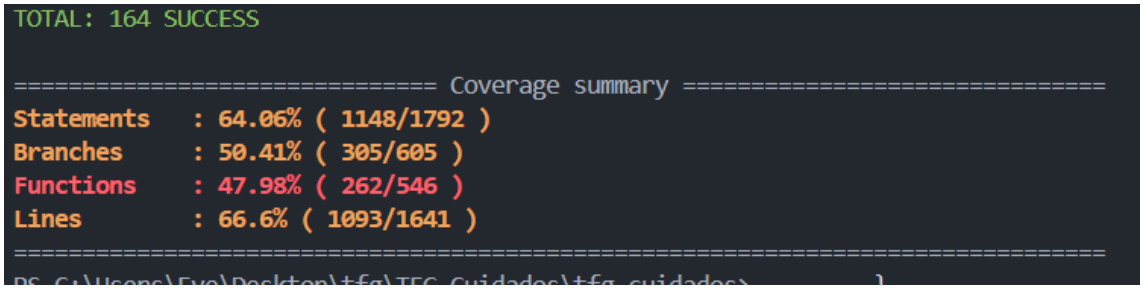


Ilustración 10. Test Karma.

1. Se creará una carpeta llamada /coverage en la raíz del proyecto.
2. Dentro habrá otra carpeta con el nombre del proyecto.
3. Debe abrir el archivo index.html que está dentro.

4. Verá un desglose detallado por líneas de código, funciones y ramas.

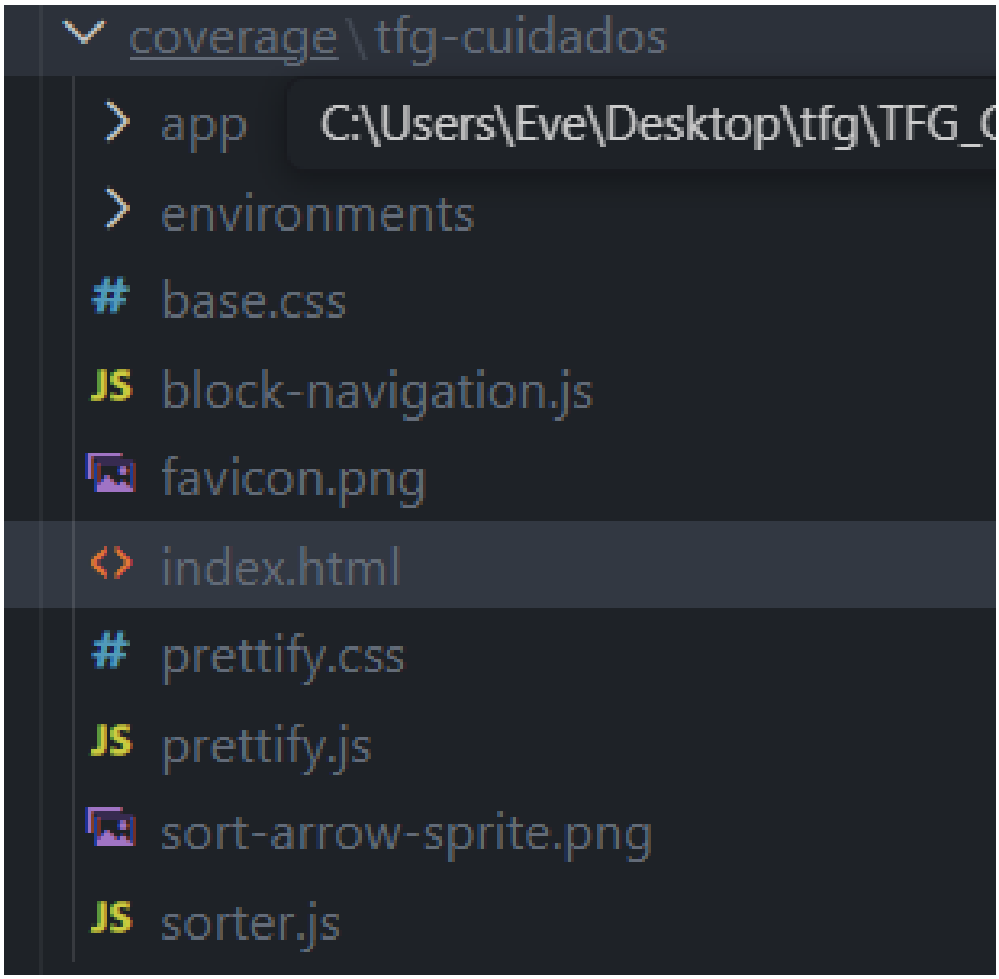


Ilustración 11: Ver test Karma.

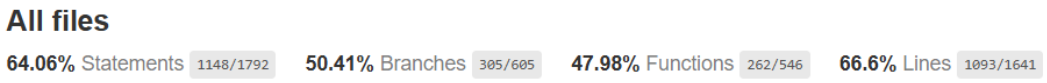


Ilustración 12: Test Karma genéricos.

5.2 Cypress: Pruebas de Extremo a Extremo (E2E)

Cypress es una herramienta moderna que simula el comportamiento real de un usuario en el navegador. A diferencia de las unitarias, estas pruebas validan flujos completos.

Comandos de Instalación y Uso

Se recomienda el uso del esquema oficial de Angular para una integración limpia:

Instalación e integración automática:

ng add @cypress/schematic

Abrir la interfaz interactiva de Cypress:

npx cypress open

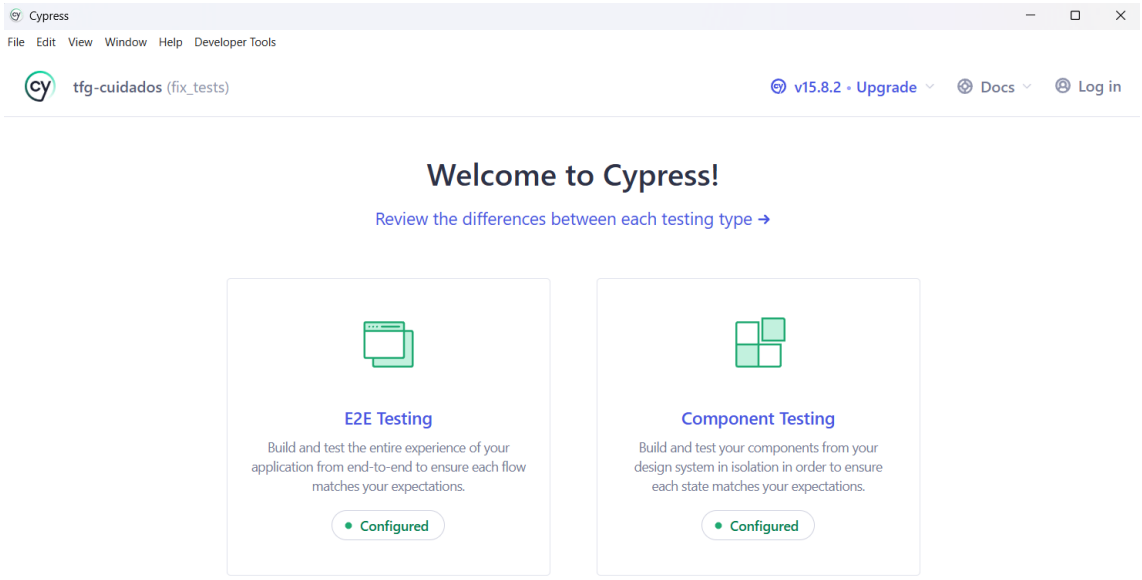


Ilustración 13: Test Cypress.

Ejecutar todas las pruebas en consola (Headless):

npx cypress run

Detalles Técnicos

Simulación Dinámica: Cypress "toma el control" del navegador. Puede hacer clic, escribir en formularios de registro y verificar que los gráficos de **Chart.js** se rendericen correctamente.

Persistencia: Es ideal para probar el flujo de **auto-registro** y la gestión de usuarios por parte del **Administrador**, validando que los permisos de acceso funcionen en la interfaz real.

5.3 Postman: Validación de API y Contratos de Datos

Postman es la herramienta de pruebas de capa de red. Se utiliza para garantizar que el intercambio de información entre el frontend (Angular) y el backend sea correcto.

Instalación

Software: Descarga la aplicación de escritorio en postman.com.

VS Code: Extensión oficial "Postman" para integración directa en el editor.

Ilustración 14: Logo Postman.

Detalles Técnicos

Validación de Esquemas: Antes de usar **Zod** en el frontend, se usa Postman para asegurar que el backend devuelve exactamente los campos esperados (evitando errores como el de la columna inexistente rol).

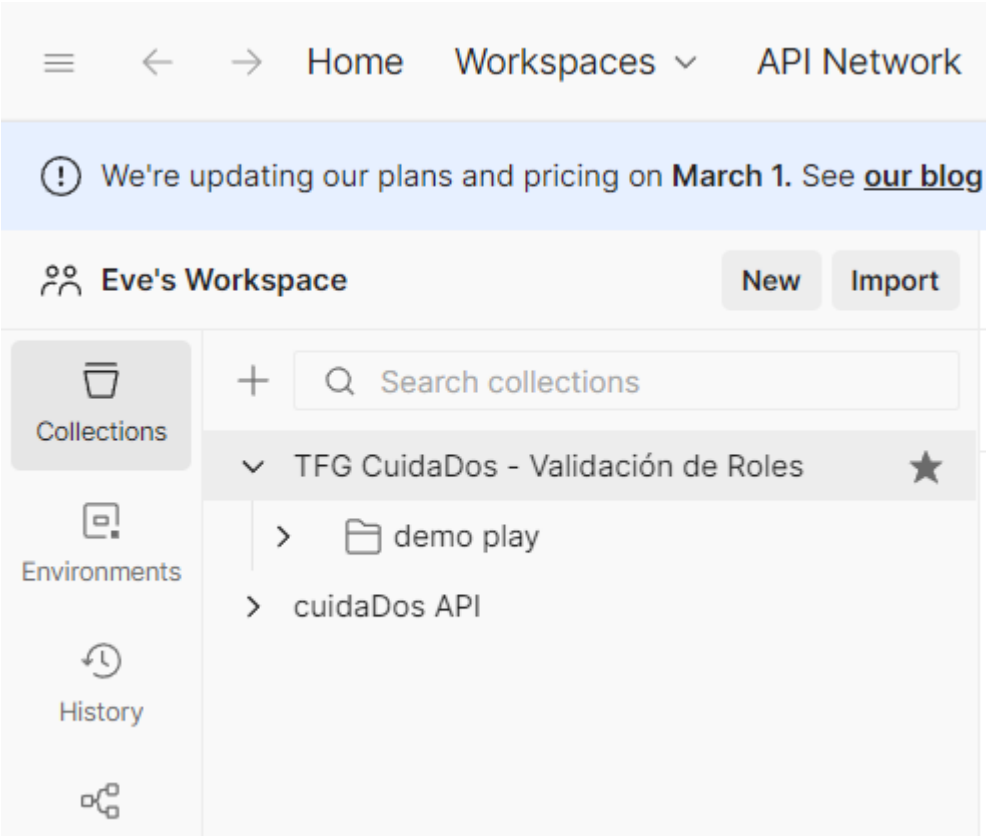


Ilustración 15: Tests Postman.

Automatización: Permite crear "Collections" con scripts de prueba en JavaScript para verificar códigos de estado (200 OK, 201 Created) y tiempos de respuesta.

5.3.1 Validación de API con Postman

Debido a que el proyecto utiliza una arquitectura distribuida (Frontend en Angular y Backend en Supabase), se ha implementado una suite de pruebas en **Postman**. Estas pruebas garantizan que la lógica de seguridad y el intercambio de datos sean correctos, complementando los tests unitarios de Karma.

5.3.1.1 Configuración del Entorno (Environment)

Para evitar la duplicación de datos y facilitar el cambio entre entornos, se utiliza el archivo TFG_CuidaDOS.postman_environment.json.

- **Variables Globales:** Se gestionan mediante variables dinámicas como {{URL_SUPABASE}}, {{ANON_KEY}} y {{SERVICE_KEY}}.
- **Gestión de Tokens:** El entorno almacena automáticamente el {{token_admin}} y {{token_cliente}} tras el inicio de sesión, permitiendo probar rutas protegidas sin intervención manual.
- **IDs Temporales:** Se utilizan variables como {{temp_id_servicio}} o {{temp_id_usuario}} para encadenar pruebas (ej: crear un usuario y usar su ID inmediatamente para borrarlo).

5.3.1.2 Estructura de la Colección: "Validación de Roles"

La colección principal organiza las peticiones por niveles de acceso, reflejando las reglas de negocio del TFG.

Ilustración 16: Colección test Postman.

1. Auto-registro: tests para el registro individual de cada usuario, uno por cada rol: administrador, empresa y cliente.
2. Log in: tests para el inicio de sesión de cada usuario, por roles.
3. Gestión usuarios por Admin: la gestión (CRUD) de empresas y clientes por parte del administrador de la aplicación.
4. Modificar perfil: modificación de los perfiles de los usuarios, por roles, de forma personal (sin intervención del administrador).
5. Acciones Admin, acciones empresas y acciones clientes: cada una de las funciones por roles.

   Administrador: comunicaciones, gestión de servicios y gestión de horarios.

   Empresa: gestión de ofertas, comunicaciones.

   Cliente: comunicaciones, gestión de contratos.
6. Eliminaciones: eliminaciones lógicas de la base de datos (mensajes, notificaciones…)
7. Baja: baja de la aplicación de los usuarios de prueba.

5.3.1.3 Automatización de Pruebas (Scripts)

Cada petición incluye scripts de validación en JavaScript para asegurar que la API responde según lo esperado. Ejemplos incluidos en los archivos:

5.3.1.4 Ejecución de Pruebas (Test Run Results)

Según el reporte de ejecución (postman_test_run.json), se han obtenido los siguientes resultados de calidad:

- **Total de Pruebas Ejecutadas:** 94 tests.
- **Tasa de Éxito:** 100% (94 pass / 0 fail).
- **Tiempo de Respuesta Promedio:** ~20ms - 150ms dependiendo del endpoint.

5. GUÍA DE INSTALACIÓN (ENTORNO LOCAL)

5.1 Instalación de Prerrequisitos (Paso a Paso)

Antes de desplegar el proyecto, es necesario preparar el entorno en la máquina anfitriona.

Paso 1: Instalación de Node.js y NPM

Node.js es el entorno de ejecución. Al instalarlo, se incluye automáticamente NPM.

1. Acceder al sitio oficial: https://nodejs.org/
2. Descargar la versión **LTS (Long Term Support)** recomendada (v18.13.0 o superior).
3. Ejecutar el instalador siguiendo el asistente predeterminado.

Comando de verificación: Abra una terminal (PowerShell o CMD) y ejecute:

```
node -v
npm -v
```

(Si escoge las versiones actuales, debe devolver versiones superiores a v22.x y v11.x respectivamente).

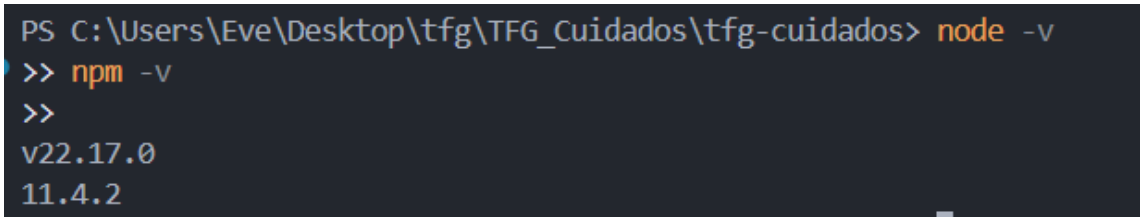


Ilustración 17: Verificación instalación node.js

Paso 2: Instalación de Git

Necesario para clonar el repositorio y gestionar versiones.

1. Descargar el instalador desde: https://git-scm.com/downloads
2. Instalar manteniendo las opciones por defecto (especialmente la opción "Git from the command line").

Configuración inicial (Comandos): Una vez instalado, ejecute en la terminal para configurar su identidad:

git config --global user.name "Tu Nombre"

git config --global user.email "tu@email.com"

Comando de verificación:

git --version

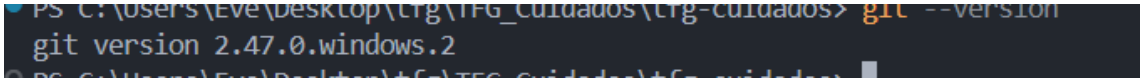


Ilustración 18: Instalación de git.

Paso 3: Instalación de Angular CLI (v21)

Esta herramienta se instala como un paquete global de NPM. Es el único paso que se realiza íntegramente por consola.

Comando de instalación: Ejecute el siguiente comando en la terminal con permisos de administrador (o usar sudo en Mac/Linux):

npm install -g @angular/cli@21

(Se especifica @21 para garantizar la compatibilidad con el proyecto, aunque @latest suele funcionar).

Comando de verificación:

ng version

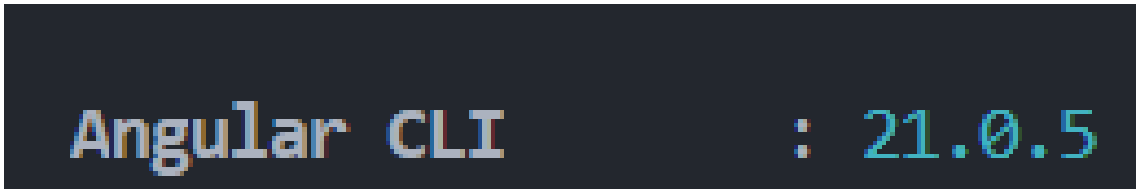


Ilustración 19: Versión Angular.

Paso 4: Instalación de Dependencias del Proyecto

Una vez instaladas las herramientas globales, se debe instalar las librerías específicas de **cuidaDos** (Supabase, Material, Tailwind, etc.).

1. Navegue hasta la carpeta del proyecto descargado:

cd ruta/a/tfg-cuidados

2. Ejecute el comando de instalación automatizada:

npm install

(Este comando leerá el archivo package.json e instalará todas las dependencias exactas en la carpeta node_modules).

5.2 Entorno de desarrollo

Siga estos pasos para levantar el entorno de desarrollo:

Paso 1: Clonado del Repositorio

git clone https://github.com/[TU_USUARIO]/tfg-cuidados.git

cd tfg-cuidados

Paso 2: Instalación de Dependencias El proyecto utiliza un package-lock.json para asegurar versiones exactas.

npm install

Paso 3: Configuración de Variables de Entorno Angular requiere las credenciales de conexión. Cree o modifique el archivo src/environments/environment.ts:

TypeScript

```
export const environment = {
  production: false,
  // URL del proyecto Supabase (Project Settings > API)
  supabaseUrl: 'https://xyzcompany.supabase.co',
  // Clave Anónima Pública (Project Settings > API > anon public)
  supabaseKey: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...'
};
```

Paso 4: Ejecución

npm start

El aplicativo estará disponible en http://localhost:4200/.

6. GUÍA DE DESPLIEGUE (ENTORNO CLOUD - VERCEL)

El despliegue en producción se gestiona a través de **Vercel**, aprovechando su red CDN global.

6.1 Configuración de Build

Vercel detecta automáticamente el framework Angular. Asegúrese de que la configuración de salida sea correcta:

- **Build Command:** npm run build
- **Output Directory:** dist/tfg-cuidados/browser (Importante: Angular 17+ y 21 cambian la carpeta de salida a /browser cuando se usa SSR).

6.2 Gestión de Rutas (SPA)

Para evitar errores 404 al recargar la página en rutas profundas (ej: /dashboard/contratos), se incluye el archivo de configuración vercel.json en la raíz:

JSON

```
{
```

```
 "rewrites": [
   { "source": "/(.*)", "destination": "/index.html" }
 ]
}
```

6.3 Variables de Entorno en Producción

Por seguridad, **nunca** suba las claves al repositorio. Configúrelas en el panel de Vercel (*Settings > Environment Variables*):

- NG_APP_SUPABASE_URL: Valor de producción.
- NG_APP_SUPABASE_KEY: Valor de producción.

7. ARQUITECTURA DE BASE DE DATOS (SUPABASE)

El sistema utiliza **Supabase** (PostgreSQL) como motor de base de datos relacional. La arquitectura se ha diseñado para garantizar la integridad referencial y la seguridad de los datos mediante lógica del lado del servidor.

7.1 Aislamiento de Capas y Seguridad de Esquemas

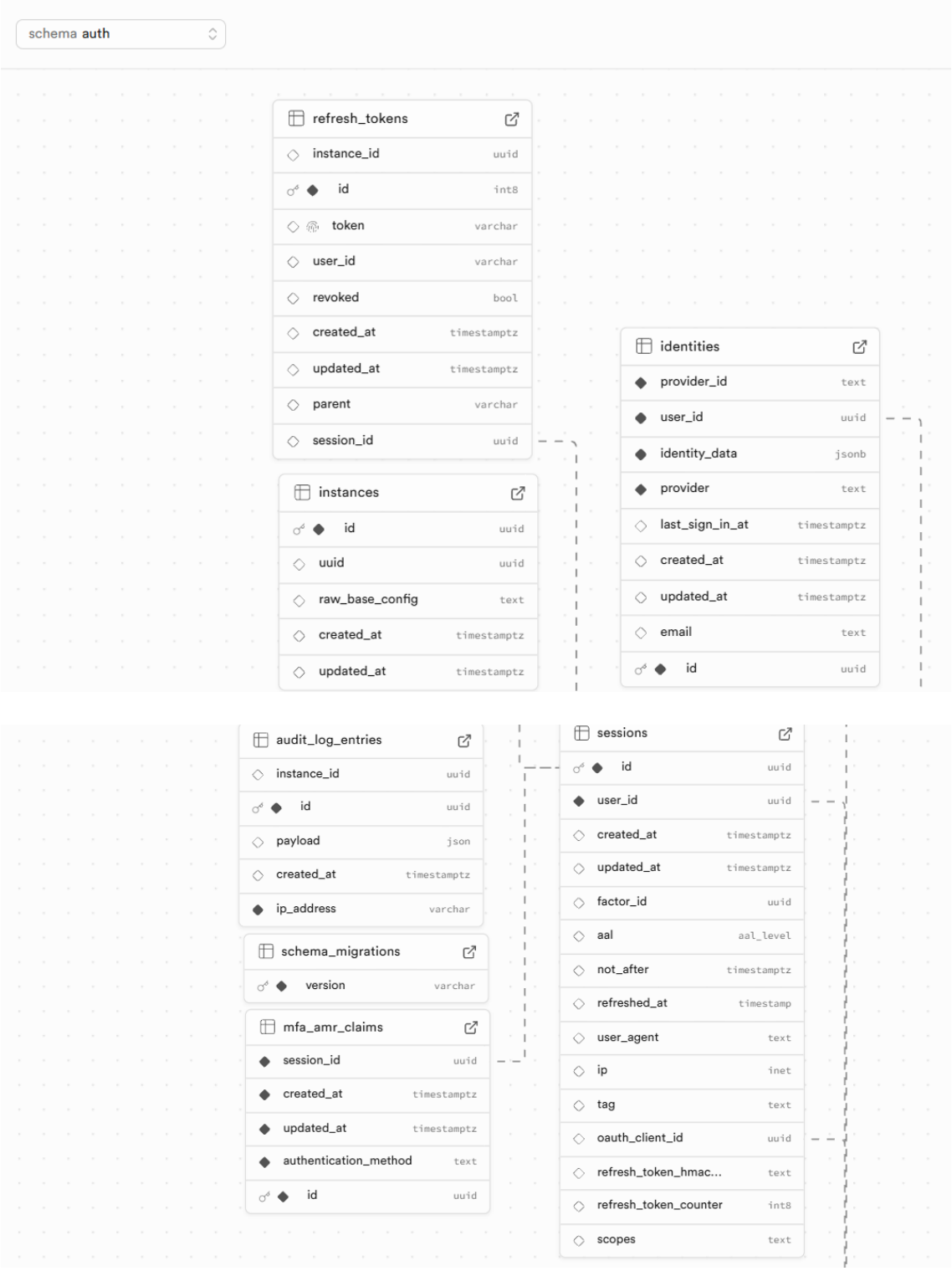
La base de datos se divide en dos áreas de responsabilidad:

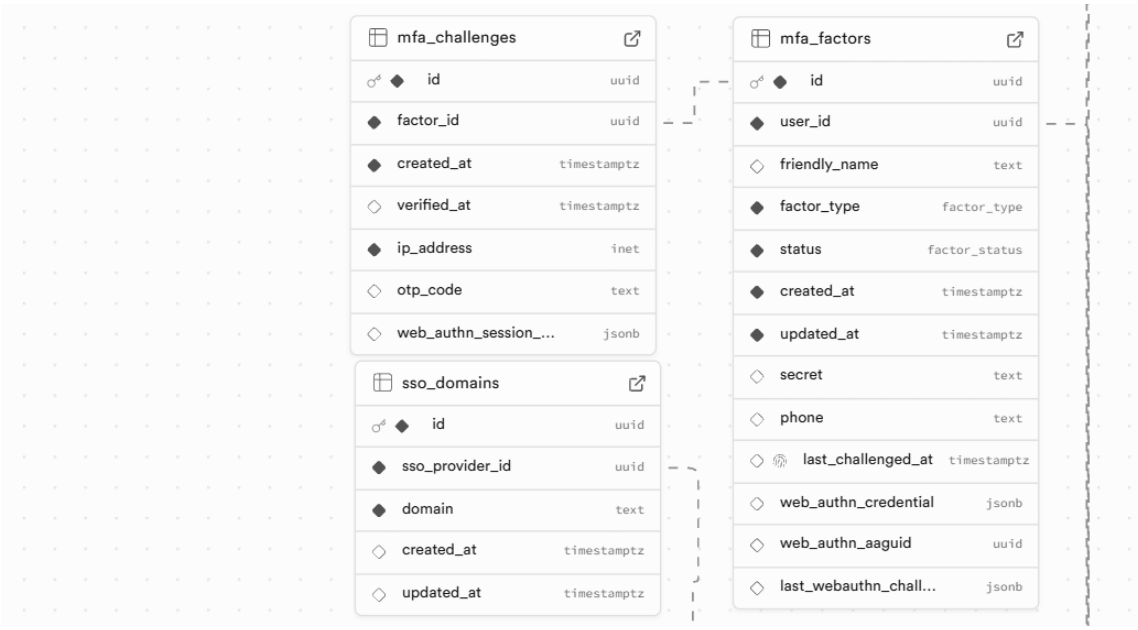
- Capa de Autenticación (auth): Gestionada por Supabase, es de solo lectura a través del panel principal para proteger las credenciales de los usuarios. Aquí residen los disparadores de pre-confirmación y creación inicial.
- Capa de Aplicación (public): Donde reside toda la lógica de negocio, servicios y contratos. Los disparadores en esta capa aseguran que los datos del frontend cumplan con las reglas de integridad antes de ser persistidos

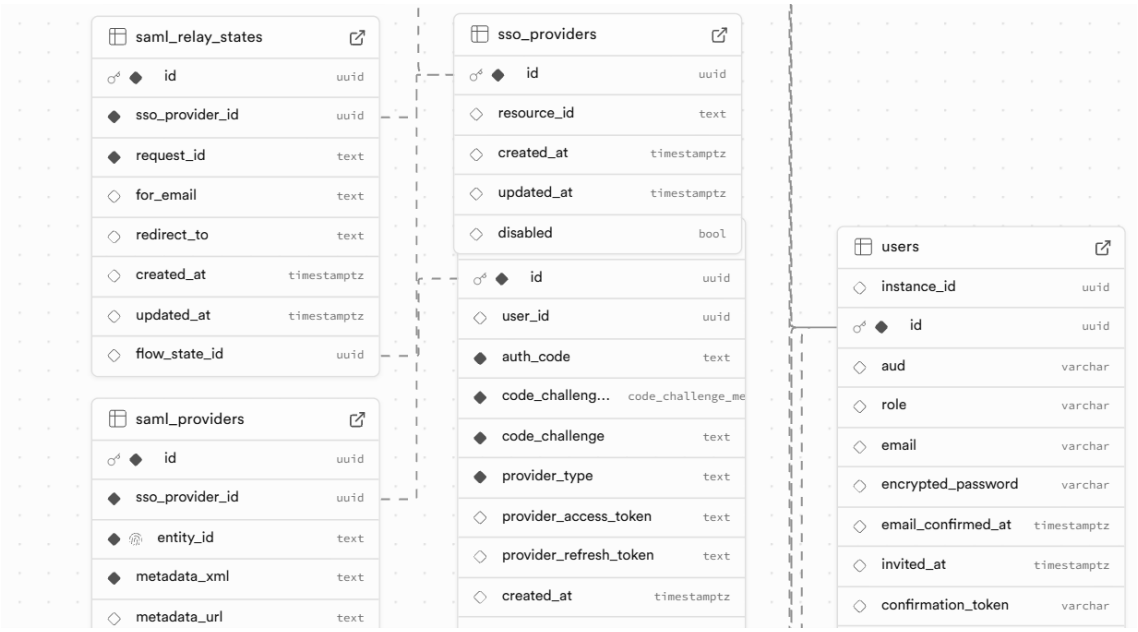
7.2 Modelo Relacional y Especialización de roles

La gestión de usuarios se divide en dos capas para separar la autenticación de la información de negocio:

- **Esquema auth (Interno):** Almacena las credenciales de acceso (email y contraseña) gestionadas por Supabase Auth.


schema auth
refresh_tokens
instance_id uuid
id int8
token varchar
user_id varchar
revoked bool
created_at timestamptz
updated_at timestamptz
parent varchar
session_id uuid
instances
id uuid
uuid uuid
raw_base_config text
created_at timestamptz
updated_at timestamptz
identities
provider_id text
user_id uuid
identity_data jsonb
provider text
last_sign_in_at timestamptz
created_at timestamptz
updated_at timestamptz
email text
id uuid
audit_log_entries
instance_id uuid
id uuid
payload json
created_at timestamptz
ip_address varchar
schema_migrations
version varchar
mfa_amr_claims
session_id uuid
created_at timestamptz
updated_at timestamptz
authentication_method text
id uuid
sessions
id uuid
user_id uuid
created_at timestamptz
updated_at timestamptz
factor_id uuid
aal aal_level
not_after timestamptz
refreshed_at timestamp
user_agent text
ip inet
tag text
oauth_client_id uuid
refresh_token_hmac... text
refresh_token_counter int8
scopes text


mfa_challenges
id uuid
factor_id uuid
created_at timestamptz
verified_at timestamptz
ip_address inet
otp_code text
web_authn_session_... jsonb
mfa_factors
id uuid
user_id uuid
friendly_name text
factor_type factor_type
status factor_status
created_at timestamptz
updated_at timestamptz
secret text
phone text
last_challenged_at timestamptz
web_authn_credential jsonb
web_authn_aaguid uuid
last_webauthn_chall... jsonb
sso_domains
id uuid
sso_provider_id uuid
domain text
created_at timestamptz
updated_at timestamptz


saml_relay_states
id uuid
sso_provider_id uuid
request_id text
for_email text
redirect_to text
created_at timestamptz
updated_at timestamptz
flow_state_id uuid
saml_providers
id uuid
sso_provider_id uuid
entity_id text
metadata_xml text
metadata_url text
sso_providers
id uuid
resource_id text
created_at timestamptz
updated_at timestamptz
disabled bool
id uuid
user_id uuid
auth_code text
code_challeng... code_challenge_me
code_challenge text
provider_type text
provider_access_token text
provider_refresh_token text
created_at timestamptz
users
instance_id uuid
id uuid
aud varchar
role varchar
email varchar
encrypted_password varchar
email_confirmed_at timestamptz
invited_at timestamptz
confirmation_token varchar

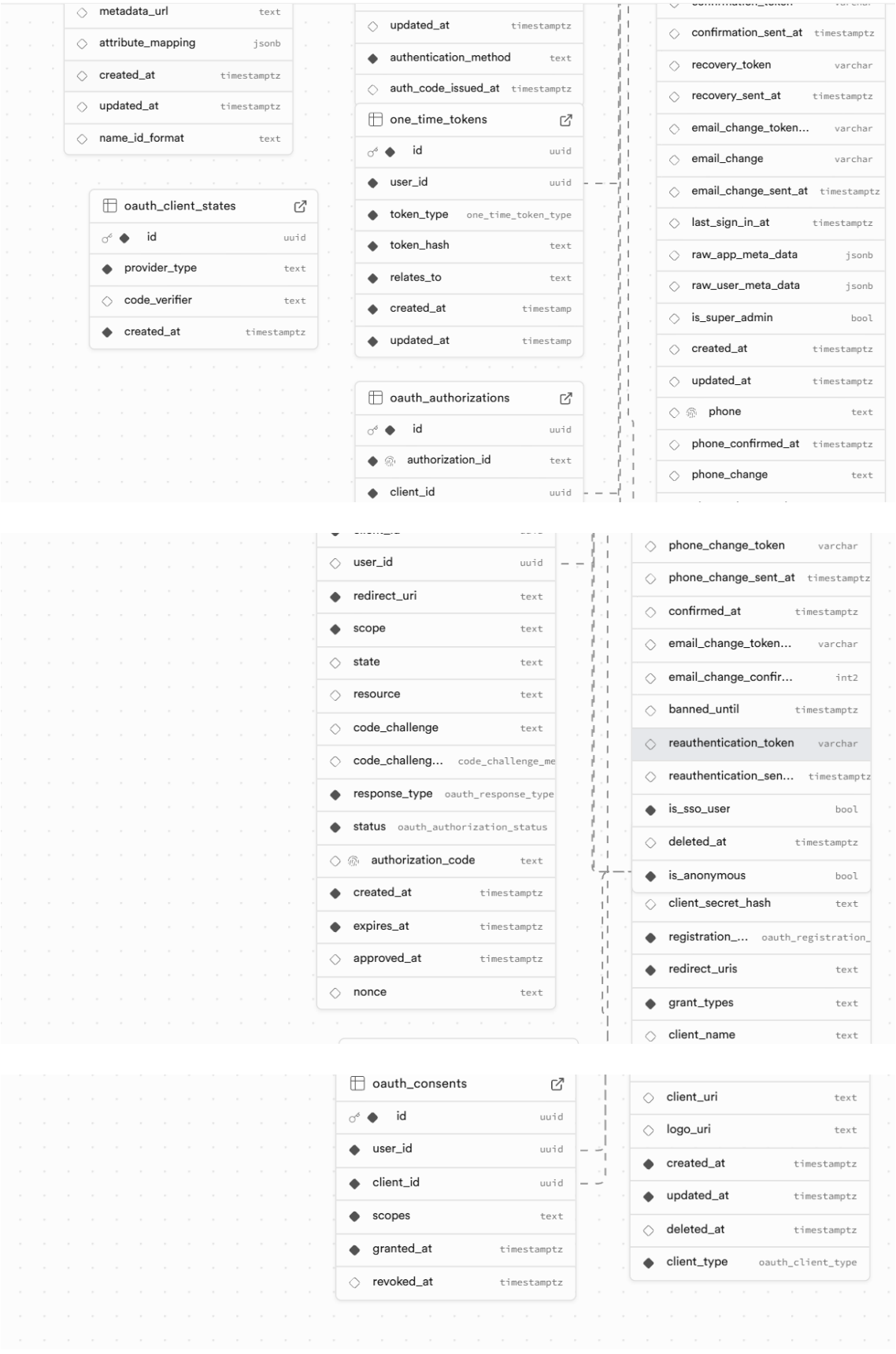


Ilustración 20: Esquema Auth Supabase.

- **Esquema public (Negocio):** La tabla public.usuarios extiende el perfil del usuario.
    - **Atributo rol:** Campo crítico que define los permisos del sistema mediante una especialización: Cliente, Empresa o Administrador.

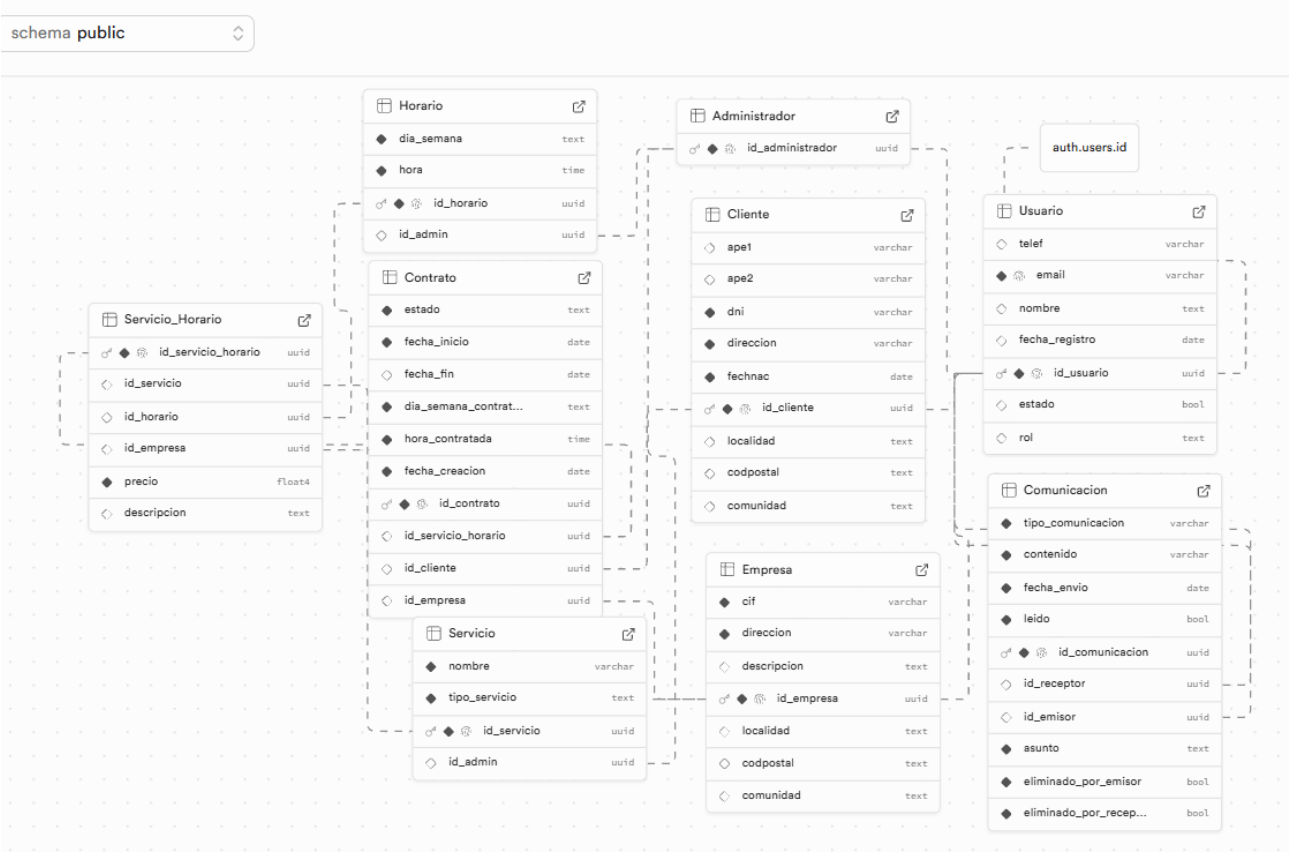


Ilustración 21: Esquema Public Spabase.

7.2.1 Diagrama Entidad-Relación (ERD)

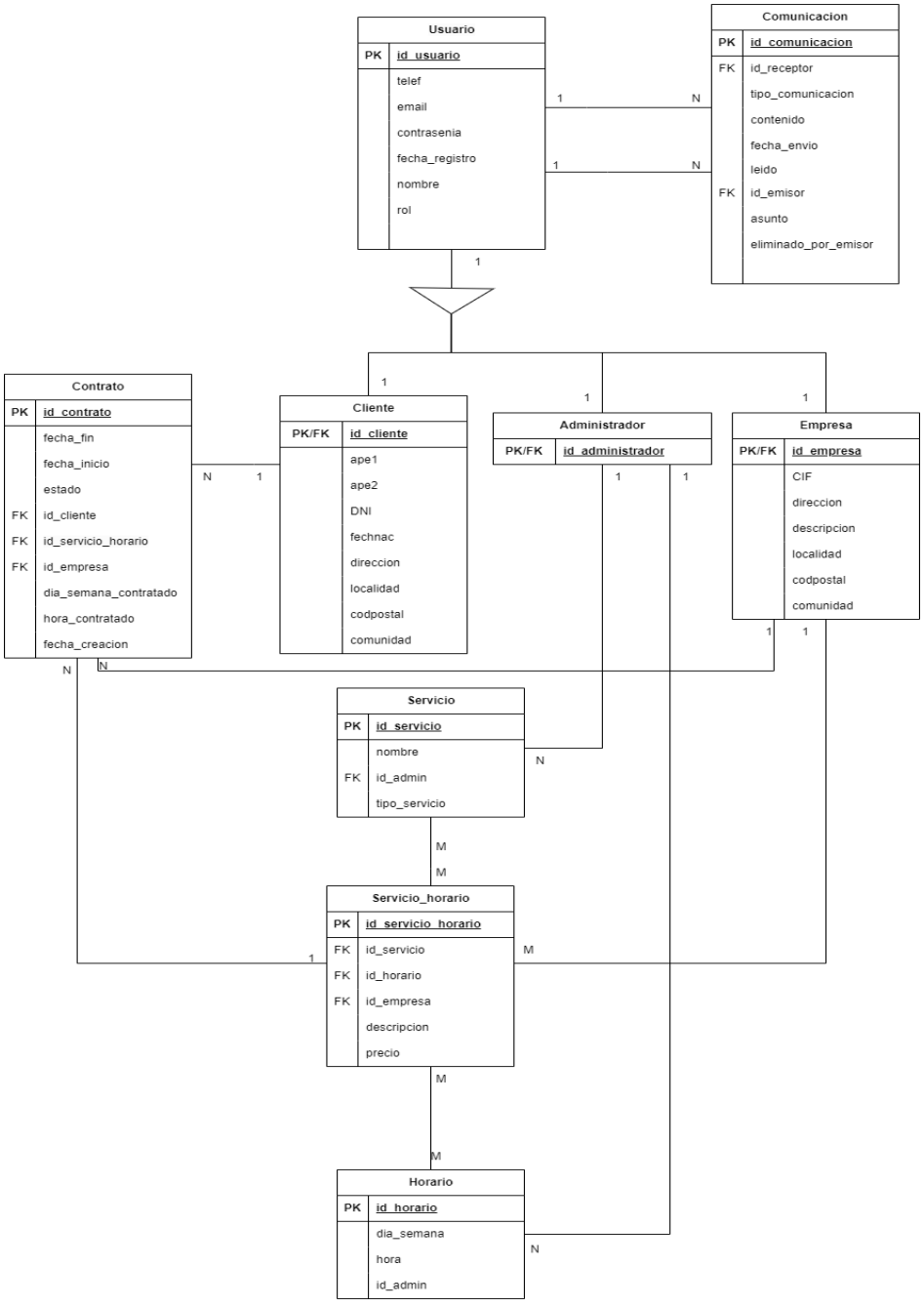


Ilustración 22: MER cuidaDos.

7.3 Lógica de Negocio

7.3.1 Lógica de Servidor: Triggers y Funciones RPC

El sistema delega la validación y la integridad de datos al motor de base de datos mediante funciones y disparadores. Esto garantiza que las reglas de negocio se cumplan incluso si se intenta manipular la API externamente.

A. Gestión de Usuarios y Perfiles

- **handle_new_user**: Trigger que automatiza la creación del perfil en la tabla public.usuarios inmediatamente después de un registro exitoso en auth.users.
- **update_profile_complete**: Procedimiento robusto para actualizar todos los campos del perfil (dirección, localidad, descripción, etc.) en una sola transacción, asegurando la consistencia de los datos del usuario.
- **activar_usuario_al_confirmar**: Gestiona el estado de activación del usuario una vez que se verifica su identidad.

B. Seguridad y Control de Acceso (RBAC)

- **is_admin**: Función de seguridad crítica que verifica si el UUID del usuario solicitante posee el rol de "Administrador" en la tabla de perfiles. Se utiliza como filtro en las políticas RLS y funciones sensibles.
- **check_user_exists** y **email_exists**: Funciones auxiliares para validar la disponibilidad de credenciales durante el registro o actualización, evitando duplicidad de datos.
- **eliminar_usuario_total**: Función de limpieza de los usuarios, elimina un usuario de forma lógica, estableciendo el estado del usuario a FALSE, de forma que no podrá conectarse al aplicativo al iniciar sesión, no se le tendrá en cuenta en el Dashboard del usuario y no podrá recuperar su cuenta (solo crearse otra cuenta nueva con otro email). Esta función fue validada con un 100% de éxito en las pruebas de Postman.

C. Automatización de Pruebas E2E

Para facilitar el testing continuo con **Cypress** y **Postman**, se han implementado mecanismos que evitan el bloqueo por flujos manuales:

- **pre_confirm_test_users**: Trigger que intercepta registros con dominios de prueba (ej. @test.com) para auto-confirmar el correo electrónico.
- **handle_test_user_confirmation**: Asegura que el flujo de confirmación para cuentas de prueba se complete sin intervención humana.

D. Integración de Limpieza y Contratos

- **rellenar_contrato_desde_angular**: Gestiona la lógica de inserción de contratos, validando parámetros antes de persistirlos en la base de datos.
- **limpieza_al_desactivar_usuario** y **limpieza_total_antes_borrado**: Garantizan que, al eliminar o desactivar una cuenta, se limpien en cascada las relaciones (mensajes, servicios, contratos), cumpliendo con las normativas de protección de datos y manteniendo la base de datos optimizada.

E. Triggers del Esquema auth (Gestionados por Supabase)

Estos disparadores reaccionan a eventos de autenticación y registro.

Trigger	Tabla	Función Asociada	Evento	Propósito
on_auth_user_ created_pre_ confirmation	users	pre_confirm_test_users	BEFORE INSERT	Auto-confirmar correos @test.com para testeo.
on_auth_user_created	users	handle_new_user	AFTER INSERT	Crear el perfil público tras el registro.
on_auth_user_created_test	users	handle_test_user_confirmation	AFTER INSERT	Flujo de confirmación interna para usuarios de prueba.
on_auth_user_verified	users	activar_usuario_al_confirmar	AFTER UPDATE	Activar cuenta cuando el email es verificado.

Tabla 1: Triggers Auth.

F. Triggers del Esquema public (Lógica de Negocio)

Estos disparadores aseguran la integridad de los datos de la aplicación.

Trigger	Tabla	Función Asociada	Evento	Propósito
tr_rellenar_contrato_angular	Contrato	rellenar_contrato_desde_angular	BEFORE INSERT	Validar y completar datos del contrato

				antes de guardar.
tr_reaccion_desactivar_usuario	Usuario	limpieza_al_desactivar_usuario	AFTER UPDATE	Limpiar datos vinculados al

Tabla 2: Triggers Public.

7.3.2 Seguridad (Row Level Security - RLS)

El sistema implementa políticas de seguridad a nivel de fila para garantizar que el acceso a la información esté estrictamente regulado según el rol y la relación del usuario con los datos.

A. Visibilidad de Perfiles y Catálogo

1. Usuarios (Clientes y Empresas): Los perfiles son visibles para todos los usuarios, pero únicamente si están autenticados en la plataforma. Esto permite que tanto clientes como empresas consulten la información necesaria para formalizar contratos y servicios.

2. Servicios y Horarios: Estas tablas actúan como un catálogo gestionado por los Administradores. Las Empresas tienen permiso para seleccionar y vincular estos servicios y horarios predefinidos a su oferta particular.

3. Servicios por Empresa: La gestión de la tabla servicios_horarios es responsabilidad de las empresas, pero el RLS restringe el acceso de modo que cada empresa solo puede gestionar sus propios registros.

B. Privacidad en Comunicaciones y Contratos

1. Mensajería Interna: La comunicación solo es accesible para las partes implicadas (emisor y receptor). Si una de las partes decide borrar el mensaje, este dejará de ser visible para ella, pero seguirá siendo accesible para la otra parte implicada que no lo haya eliminado.

2. Contratos: El acceso está limitado exclusivamente a las partes involucradas en el acuerdo. No obstante, un contrato deja de ser visible si se marca como inactivo para una de las partes, reforzando el control individual sobre la información histórica

7.4 Seguridad Avanzada y Ejecución de Funciones (Security Definer)

Para garantizar que ciertas operaciones críticas se realicen correctamente sin comprometer la seguridad general del sistema, se ha utilizado el modificador SECURITY DEFINER en funciones específicas como eliminar_usuario_total, update_profile_complete e is_admin.

¿Por qué es necesario?

Por defecto, las funciones en PostgreSQL se ejecutan como SECURITY INVOKER, lo que significa que tienen los mismos permisos que el usuario que las llama. Sin embargo, en CuidaDos se necesita:

1. Bypass de RLS: Un usuario normal no tiene permisos para borrar filas en esquemas protegidos o realizar limpiezas totales. Al definir la función como SECURITY DEFINER, la función se ejecuta con los privilegios del creador de la función (admin) y no del usuario actual.
2. Gestión de Autenticación: La función eliminar_usuario_total necesita interactuar con el esquema auth, el cual está restringido. Solo una función con privilegios elevados puede coordinar el borrado simultáneo en auth.users y public.usuarios.
3. Validación de Roles: La función is_admin requiere consultar tablas de sistema para verificar permisos; el uso de este modificador asegura que la validación sea infalible independientemente de quién la invoque.

8. EJECUCIÓN Y PRUEBAS DEL APLICATIVO

8.1 Comandos de Ciclo de Vida

El archivo package.json define los siguientes scripts:

- **npm start**: Levanta el servidor de desarrollo (HMR activo).
- **npm run build**: Compila la aplicación para producción con optimización AOT (Ahead-of-Time).

- **npm test**: Ejecuta la suite de pruebas unitarias con Karma.

8.2 Gestión de Bundles y Optimización (Caso de Estudio)

Durante el proceso de compilación para producción (ng build), se realiza un análisis del peso de los archivos JavaScript generados. Angular establece por defecto unos "presupuestos" (*Budgets*) muy restrictivos (1MB máximo) para garantizar el rendimiento en redes móviles 3G.

1. El Desafío (Problema Identificado)

Al integrar librerías robustas necesarias para la lógica de negocio, el tamaño del *bundle* inicial excedió el límite predeterminado, generando el siguiente error de compilación:

Error: bundle initial exceeded maximum budget. Budget 1.00 MB was not met by 313.55 kB.

Este incremento de tamaño se debe a la inclusión de dependencias críticas en el hilo principal:

- **@supabase/supabase-js:** Cliente completo para la conexión a base de datos y tiempo real.
- **@angular/material:** Componentes de interfaz de usuario complejos (tablas, diálogos).
- **Chart.js:** Motor de renderizado de gráficas para el panel de administración.

2. Solución Implementada

Para solucionar el bloqueo sin sacrificar funcionalidad, se reconfiguraron los umbrales de rendimiento en el archivo angular.json. Se estableció una política de presupuestos realista para una **Aplicación de Gestión (Enterprise SPA)**, diferenciándola de una web estática ligera.

Configuración aplicada (angular.json):

Tabla 3: Ajustes Test.

3. Justificación Técnica y Estrategia de Carga

A pesar de aumentar el tamaño permitido del archivo principal (main.js), el rendimiento de la aplicación se mantiene óptimo gracias a la implementación de dos estrategias compensatorias:

1. **Lazy Loading (Carga Perezosa):** Como se observa en la salida de la compilación (Figura X), no todo el código se carga al inicio. Módulos como la "Recuperación de Contraseña" se han separado en *chunks* diferidos (chunk-recover-password.js), que solo se descargan si el usuario los solicita.
2. **SSR (Server-Side Rendering):** Gracias al renderizado en servidor, el usuario visualiza la interfaz (FCP) antes de que termine de descargarse el bundle de JavaScript completo, mitigando la sensación de espera.

4. Resultado

```
PS C:\Users\Eve\Desktop\tfg\TFG_Cuidados\tfg-cuidados> npm run build

> tfg-cuidados@0.0.0 build
> ng build

Browser bundles
Initial chunk files   | Names           |  Raw size | Estimated transfer size
main-BERFKOR2.js      | main            | 701.27 kB |               156.18 kB
chunk-ZHKNJ2QP.js     | -               | 556.88 kB |               138.08 kB
styles-N56PI5UO.css   | styles          |  55.40 kB |                 6.95 kB

                      | Initial total   |   1.31 MB |               301.21 kB

Lazy chunk files      | Names           |  Raw size | Estimated transfer size
chunk-XHJCSMPV.js     | recover-password |  3.04 kB |                 1.30 kB


Server bundles
Initial chunk files   | Names           |  Raw size
server.mjs            | server          |   1.26 MB |
main.server.mjs       | main.server     |   1.15 MB |
chunk-3AZZQSYJ.mjs    | -               | 576.62 kB |
polyfills.server.mjs  | polyfills.server | 233.25 kB |
chunk-L2MVFUKD.mjs    | -               |   1.32 kB |
chunk-T55IDOPT.mjs    | -               | 976 bytes |

Lazy chunk files      | Names           |  Raw size
chunk-RHDGF2TI.mjs    | xhr2            |  12.14 kB |
chunk-N3SI5I4Q.mjs    | xhr2            |  12.07 kB |
chunk-74CEVUQZ.mjs    | recover-password |  3.10 kB |

Prerendered 19 static routes.
Application bundle generation complete. [11.481 seconds] - 2026-02-01T09:26:12.835Z
```

Tabla 4: Resultado test tras configuración.

Como se observa en la salida, el sistema genera dos tipos de bundles:

1. **Browser Bundles:** Archivos JavaScript optimizados para el cliente.
2. **Server Bundles:** Archivos generados para el SSR (Server-Side Rendering).

9. ARQUITECTURA DEL PROYECTO (ESTRUCTURA DE CARPETAS)

El proyecto sigue una arquitectura modular basada en las recomendaciones de estilo oficiales (*Angular Style Guide*), agrupando los archivos por funcionalidad (Feature-based).

Al utilizar la arquitectura de Standalone Components (nativa en Angular 21), se ha eliminado la complejidad de los NgModules, resultando en un árbol de directorios limpio y escalable situado en src/app:

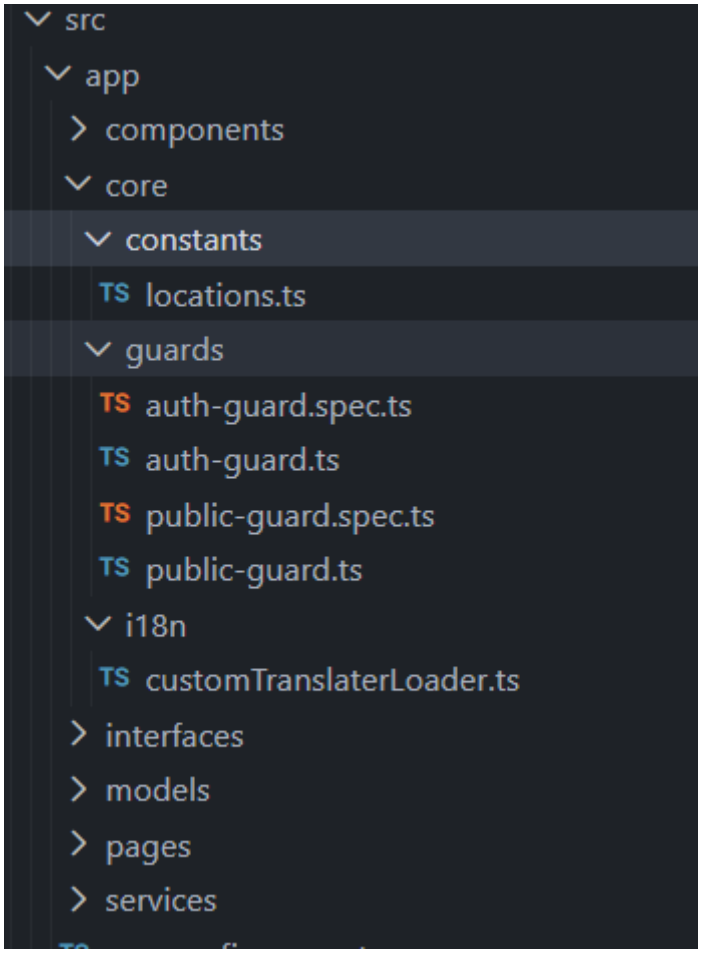


Ilustración 23: Arquitectura de carpetas.

- **Carpeta components:** Contiene los **Componentes de Presentación (Dumb Components)**.
  Son elementos reutilizables de la interfaz (botones, tarjetas, tablas) que no dependen de la lógica de negocio ni de servicios externos. Reciben datos únicamente a través de @Input y comunican eventos al padre mediante @Output.
- **Carpeta constants.**
  Almacena archivos con valores inmutables y de configuración global para evitar el uso de "números mágicos" o cadenas de texto dispersas.
- **Carpeta guards.**
  Aloja la lógica de **Seguridad Perimetral del Enrutador**. Implementa las interfaces que interceptan la navegación antes de cargar una ruta. Aquí residen AuthGuard (verificación de sesión) y PublicGuard (rutas de acceso a los usuarios sin sesión).

- **Carpeta interfaces.**
  Define los **Contratos de Datos (Data Contracts)** en TypeScript. Se utilizan para garantizar el tipado estricto de las respuestas de la base de datos (Supabase) y la estructura de los objetos que fluyen por la aplicación, facilitando el desarrollo y reduciendo errores en tiempo de ejecución.
- **Carpeta models.**
  Contiene clases o tipos complejos que requieren lógica interna o transformación de datos. A diferencia de las interfaces (que son solo contratos), los modelos pueden incluir constructores o métodos auxiliares para formatear la información antes de mostrarla en la vista.
- **Carpeta services.**
  Representa la **Capa de Lógica de Negocio y Acceso a Datos**.
  Función: Son clases inyectables (Singleton) encargadas de la comunicación HTTP, la interacción con el cliente de Supabase y la gestión del estado reactivo (usando Signals). Los componentes delegan en ellos cualquier operación compleja.

10. NAVEGACIÓN Y SEGURIDAD EN RUTAS

La gestión de vistas en **cuidaDos** se delega al **Angular Router**, configurado mediante un array de objetos Routes que mapean URLs específicas a componentes *Standalone*.

Estrategia de Carga Híbrida

El sistema implementa una arquitectura de carga mixta para equilibrar el tiempo de inicio y la fluidez de navegación:

1. Carga Temprana (Eager Loading):

Los módulos críticos de uso frecuente (como Home, Dashboard, Contracts) se importan estáticamente. Al cargar estos recursos al inicio, las transiciones entre las opciones del menú principal son instantáneas (0ms de latencia), mejorando la experiencia de usuario (UX) en sesiones activas.

2. Carga Perezosa (Lazy Loading):

Se aplica en módulos aislados o de uso esporádico, como se evidencia en la ruta recover-password.

Implementación: Uso de la función loadComponent con importación dinámica (import(...)).

El código de recuperación de contraseña no se descarga en el navegador del usuario a menos que este lo solicite explícitamente, reduciendo el peso del *bundle* inicial.

Seguridad Perimetral (Guards)

La protección de rutas se aplica mediante Guardianes Funcionales (Functional Guards) inyectados en la propiedad canActivate:

- **authGuard:**

Protege las rutas privadas (/home, /dashboard, /messages, etc.). Verifica que exista una sesión válida en Supabase. Si el token ha expirado o no existe, bloquea la navegación y redirige al usuario al Login.

- **publicGuard:**

Protege la ruta raíz (/) y el Landing. Su función es inversa: si un usuario ya tiene sesión iniciada e intenta acceder al Landing, el guardián lo redirige automáticamente a su área personal (/home), evitando que tenga que volver a loguearse.

- **Resolución Estricta (runGuardsAndResolvers):**

En la ruta /register, se ha configurado la opción 'always'. Esto fuerza la re-ejecución de los validadores incluso si la navegación proviene de la misma ruta, asegurando que el formulario de registro se reinicie correctamente ante reintentos.

Mapa de Rutas (Route Map)

A continuación, se detalla la estructura de navegación definida en app.routes.ts:

Ruta (URL)	Acceso	Componente / Función	Guardián
'' (Raíz)	Público	Landing Page (Página de Aterrizaje)	publicGuard
register	Público	Registro de Usuarios/Empresas	-
home	Privado	Vista Principal del Usuario	authGuard
dashboard	Privado	Panel de Control (Admin/Empresa)	authGuard
admin-gestion	Privado	Gestión de Usuarios (Admin)	authGuard
messages	Privado	Sistema de Mensajería Interna	authGuard
contract	Privado	Gestión de Contratos	authGuard
recover-password	Público	Recuperación (Lazy Loaded)	-
** (Wildcard)	-	Redirección por defecto a Raíz (Manejo 404)	-