JUNTA DE EXTREMADURA

Consejería de Educación y Empleo

I E S A L B A R R E G A S

TÉCNICO SUPERIOR EN DESARROLLO DE APLICACIONES WEB

Departamento de Informática

PROYECTO

CONTROL-DIARIO

Manual Técnico

Autor/es: Francisco Javier Gómez García

Curso Académico: 2024/2025

CONTROL-DIARIO

JUNTA DE EXTREMADURA
Consejería de Educación y Empleo

Introducción

App **Control-Diario** es un prototipo de aplicación para adaptarse a empresas, tanto públicas como privadas, cuyos servicios requieren una operatividad continua y una coordinación precisa entre sus integrantes. Se trata de una herramienta especialmente útil para organizaciones en las que los turnos, la supervisión del personal y la trazabilidad de las actuaciones resultan críticos para garantizar un funcionamiento eficaz.
En su concepción inicial, la aplicación ha sido enfocada a los **servicios de prevención y extinción de incendios forestales de Extremadura**, un entorno donde la inmediatez, la disponibilidad del personal y la correcta gestión de recursos son fundamentales. No obstante, siendo solo un prototipo y gracias a su estructura modular y flexible del sistema, permite que pueda ser adaptada con facilidad a otros cuerpos y entidades con dinámicas similares, como los **Agentes del Medio Natural**, equipos de emergencias, o empresas con plantillas numerosas distribuidas en diferentes ubicaciones geográficas.

El objetivo principal de Control-Diario es proporcionar a cada trabajador una herramienta accesible desde cualquier dispositivo móvil que le permita consultar su situación laboral en tiempo real: turnos, anotaciones diarias, salidas de servicio, incidencias, horas extra, disponibilidad y cualquier otra información relevante. Al mismo tiempo, dota a los responsables y coordinadores de un sistema centralizado que automatiza gran parte de la gestión, reduce errores humanos y mejora considerablemente la eficiencia administrativa.

Control-Diario garantiza un almacenamiento seguro de datos, autenticación fiable y comunicación inmediata entre trabajadores y gestión. Esto permite que la supervisión del personal sea prácticamente automática, ofreciendo una visión global en cuestión de segundos.

En resumen, Control-Diario nace como una solución moderna, escalable y altamente adaptable, pensada para cubrir las necesidades reales de organizaciones que requieren un control operativo constante, mejorando la coordinación interna y aumentando la capacidad de respuesta ante cualquier situación.

Arquitectura de la aplicación

La aplicación Control-Diario se ha desarrollado siguiendo una arquitectura web moderna de tres capas bien diferenciadas:

1. **Capa de presentación** (Frontend): construida con Next.js 15.5.5 y React 19.1.0, escrita íntegramente en TypeScript 5.9.x, y maquetada con Tailwind CSS 4.x y componentes accesibles mediante la librería Radix UI.

2. **Capa de lógica de negocio y API:** implementada sobre el propio runtime de Node.js a través de los *route handlers* de Next.js (src/app/supabase/...) y la middleware de Next para control de acceso por roles.

3. **Capa de datos:** resuelta mediante Supabase, que proporciona una base de datos PostgreSQL gestionada, autenticación de usuarios y un SDK completo para el acceso seguro a los datos.

Esta separación facilita la mantenibilidad, la escalabilidad y la posibilidad de evolucionar cada capa de forma independiente (por ejemplo, cambiando el proveedor de base de datos sin tocar la interfaz de usuario)

Estructura del proyecto

La estructura de carpetas principal del proyecto es la siguiente:

src/app

- layout.tsx y page.tsx: layout general y página de inicio.
- admin/...: vistas y lógica específicas para el rol administrador.
- bf/...: vistas para brigadas forestales.
- jr/...: vistas para jefes de reten/responsables.
- supabase/...: *route handlers* que exponen la lógica de negocio como endpoints HTTP.
- globals.css: estilos globales de la aplicación.

src/components: componentes reutilizables de interfaz (formularios, tablas, tarjetas, etc.).

src/server:

- server.ts: creación del cliente de Supabase en servidor.
- client.ts: creación del cliente de Supabase en navegador.

src/lib: utilidades generales (utils.ts, helpers para estilos, etc.).

public/: recursos estáticos (imágenes, logos como logoGreen.svg, etc.).

Archivos de configuración en raíz:

- next.config.ts, tsconfig.json, eslint.config.mjs, postcss.config.mjs, tailwind.config (a través de components.json), etc.

Esta organización facilita localizar rápidamente cualquier parte del sistema.

Frontend

Está basado en el App Router de Next.js (carpeta `src/app`), lo que permite combinar páginas server components con client components según las necesidades de cada vista. La interfaz está organizada por roles y áreas funcionales:

- `/admin` – zona de administración (gestión de usuarios, listados mensuales, etc.).
- `/bf` – interfaz para el personal de base (brigadas forestales).
- `/jr` – interfaz para los jefes de reten / responsables.
- `/` – pantalla de inicio y acceso.
- `src/app/supabase/...` – puntos de entrada de la API que consumen el frontend

Tecnologías usadas(Frontend)

Control-Diario está desarrollado con un stack moderno, optimizado para el rendimiento, la experiencia de usuario y la mantenibilidad del código

--Next.js 15.5.5: Framework principal del frontend que permite:

- Renderizado híbrido (SSR, SSG y CSR según la página).
- Routing basado en el *App Router* (src/app/*).
- Middleware nativa para filtrado por roles.
- Optimización automática de imágenes, datos y carga.

Su elección se debe a su integración perfecta con React y Vercel, además de ofrecer un rendimiento excelente en aplicaciones con múltiples vistas y autenticación.

--React 19.1.0: Librería principal para crear componentes reutilizables.

Ventajas:

- Componentes declarativos.
- Manejo avanzado del estado.
- Ecosistema amplio y estable.

-- TypeScript 5.9: El proyecto se ha tipado completamente para eliminar errores comunes en tiempo de ejecución, mejorar la autocompletación y facilitar la mantenibilidad.

--Tailwind CSS 5: Framework de utilidades para estilado.

Se eligió porque:

- Permite construir interfaces rápidamente.
- Garantiza consistencia en diseño.
- Reduce CSS innecesario. Hace el código más legible y modular.

--Radix UI y Shadcn UI: Usadas para modales, selects, dropdowns,

formularios y elementos interactivos con accesibilidad garantizada (WAI-ARIA).
Ofrecen componentes avanzados sin necesidad de reinventar comportamientos complejos.
--Lucide-react: Paquete de iconos modernos SVG optimizados para React.
date-fns y react-day-picker: Manejo avanzado de fechas: calendarios, selección de días, cálculos de horas, diferencias, etc.
Esencial para gestionar anotaciones diarias, turnos y salidas de servicio.
--Supabase JS (cliente navegador) :Control de sesión, login, logout y lectura específica de datos desde el cliente cuando es necesario.

En conjunto, el frontend está construido con una arquitectura clara, modular, escalable y orientada a componentes, optimizada para ser rápida, accesible y fácil de mantener.

Entorno de desarrollo (Frontend)

-- Visual Studio Code como editor principal, junto con sus extensiones para
-- React, Tailwind, Supabase y control de errores en TypeScript.
-- Node.js ≥ 18.x LTS utilizado para ejecutar el servidor de desarrollo.
-- Husky 9.x + lint-staged 16.x para asegurar que el código está formateado y sin errores antes de ser versionado.
-- ESLint 9.x + Prettier 3.x + eslint-config-next para mantener un estilo de código homogéneo y sin errores.
--Commitizen + cz-emoji para generar mensajes de commit limpios y estandarizados.
--Hot Reloading de Next.js para desarrollo rápido.

Este entorno facilita un ciclo de desarrollo ágil, seguro y con un control estricto de la calidad del código

Backend

Aunque la aplicación no utiliza un backend tradicional separado, Next.js proporciona un backend integrado a través de:

Route Handlers de Next.js

Ubicados en src/app/supabase/*, actúan como API REST del sistema:

Tecnologías usadas(Backend)

-- Supabase (PostgreSQL + Auth + Storage). Es el núcleo del backend:
Base de datos PostgreSQL con tablas relacionales (usuarios, anotaciones, casetas, municipios, salidas, unidades)
Sistema de autenticación ligado a auth.users
Policies RLS para controlar qué datos puede ver o modificar cada usuario
Supabase Server Client para el backend (createServerClient)Supabase Browser Client para el frontend (createBrowserClient)

La elección de Supabase se justifica por:
Su integración nativa con Next.js y Vercel.
Autenticación segura sin necesidad de un backend adicional.
SQL real y potente (PostgreSQL).
RLS para control fino del acceso.
SDK sencillo para desarrollar y mantener
Next.js Middleware:
En src/middleware.ts se define:
Autenticación basada en cookie de Supabase.
Redirecciones según rol: admin, jr, bf.
Protección automática de rutas internas.
Esto ofrece un backend robusto sin necesidad de Express o NestJS.

Entorno de desarrollo (Backend)

-- Node.js LTS (18+) para ejecutar los handlers, middleware y la lógica de servidor.

-- Visual Studio Code para escribir toda la lógica backend dentro del mismo repositorio que el frontend.

-- Postman para probar manualmente los endpoints de /supabase/*.

Documentación técnica

Análisis

A continuación se describen los requisitos funcionales implementados en la aplicación Control-Diario.

Estos requisitos describen las funcionalidades que el sistema proporciona a cada tipo de usuario, así como los procesos principales de la aplicación.

Requisitos Funcionales

Autenticación y acceso:

1. El sistema debe permitir el inicio de sesión mediante email y contraseña.

2. El sistema debe validar las credenciales contra el servicio de autenticación de Supabase.

3. El sistema debe mantener la sesión del usuario mediante cookies seguras.

4. El sistema debe redirigir automáticamente a la vista correspondiente según el rol del usuario:

Administrador → /admin

Jefe de Retén (JR) → /jr

Brigada Forestal (BF) → /bf

5. El sistema debe impedir el acceso a rutas no autorizadas según el rol.

Gestión de usuarios (Administrador):

6. El administrador debe poder crear nuevos usuarios desde una interfaz dedicada.

7. El administrador debe poder eliminar usuarios del sistema.

8. El administrador debe poder ver la lista completa de usuarios registrados.

9. El sistema debe vincular cada usuario con su unidad, caseta y rol correspondiente.

10. El sistema debe permitir consultar la información detallada de cada usuario.

Gestión de anotaciones diarias JR:

11. El usuario debe poder registrar su anotación diaria indicando:
    Fecha
    Hora de entrada
    Hora de salida
    Horas extra (si las hubiera)

12. Código del día (servicio normal, guardia, descanso, etc.)
13. El sistema debe almacenar cada anotación en la tabla anotaciones.
14. El usuario debe poder consultar y editar las anotaciones del día.
15 .El sistema debe impedir introducir más de una anotación por usuario y día.
16 .El usuario debe poder registrar salidas asociadas a una anotación diaria.
    -- Cada salida debe incluir:
    -- Tipo de salida
    -- Lugar
    -- Hora de salida
    -- Hora de entrada
    -- Número de intervinientes
    -- Las salidas deben quedar vinculadas a la anotación correspondiente mediante anotacion_id.

Consultas y listados mensuales:

17. El usuario debe poder visualizar un listado de sus anotaciones mensuales.
18. El administrador debe poder generar un listado mensual de todo el personal.
19. El sistema debe mostrar para cada trabajador:
    -- Horas totales
    -- Número de servicios
    -- Horas extra
    -- Estadísticas del mes

Gestión de la sesión:

20. El usuario debe poder cerrar sesión desde cualquier página.
21. El sistema debe limpiar las cookies de autenticación al cerrar sesión.
22. La sesión caducada debe detectarse automáticamente y redirigir al login.

Requisitos NO Funcionales

Seguridad:

1. Los datos sensibles deben almacenarse en Supabase con políticas RLS activas.
2. Solo los administradores pueden gestionar usuarios.
3. No se debe permitir el acceso a endpoints sin autenticar.
4. La base de datos Supabase debe soportar crecimiento en volumen de datos y número de usuarios

Mantenibilidad:

5. El código debe estar estructurado en componentes reutilizables.
6. La lógica del servidor debe estar separada en endpoints independientes.
7. El proyecto debe cumplir reglas de estilo mediante ESLint y Prettier.
8. Debe existir una organización clara en carpetas (app, components, server, lib, etc.).

Fiabilidad:

9. El sistema debe mantener la integridad de los datos gracias a claves foráneas y validaciones.
10. No debe permitir anotaciones duplicadas para el mismo usuario y día.
11. Los fallos deben registrarse en los logs de Supabase y Vercel.

Disponibilidad:

12. La aplicación debe estar accesible 24/7 mediante su despliegue serverless.
13. Supabase y Vercel garantizan alta disponibilidad por encima del 99%.

Desarrollo (diagrama de secuencias,…

En primer lugar el esquema principal de la base de datos que proporciona automáticamente Supabase:

En esta base de datos las tablas municipios, unidades y casetas están cargadas desde un archivo .json, creado para asignar a cada usuario su lugar de trabajo, que a su vez dependen de dos tipos enumerados que son las provincias Cáceres y Badajoz , y las zonas de coordinación que se asigna a cada municipio, que en total son once.

Para consultar el archivo .json pulse Aquí

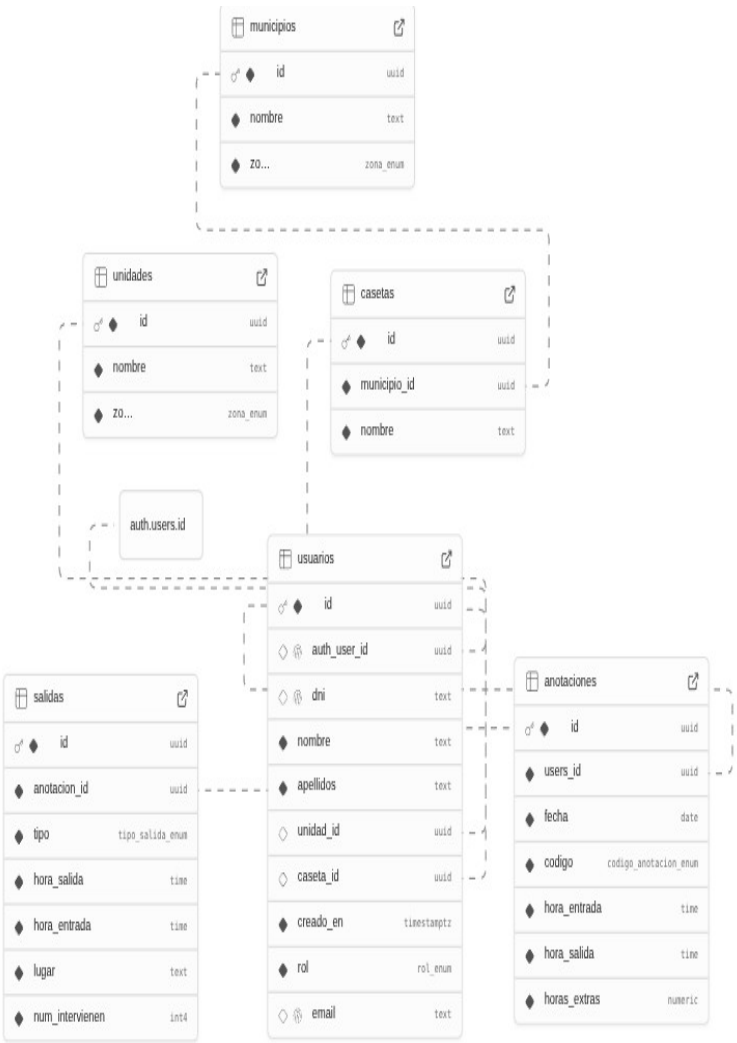


Diagrama relacional

Se exponen también los valores que tienen los enumerados en Supabase.

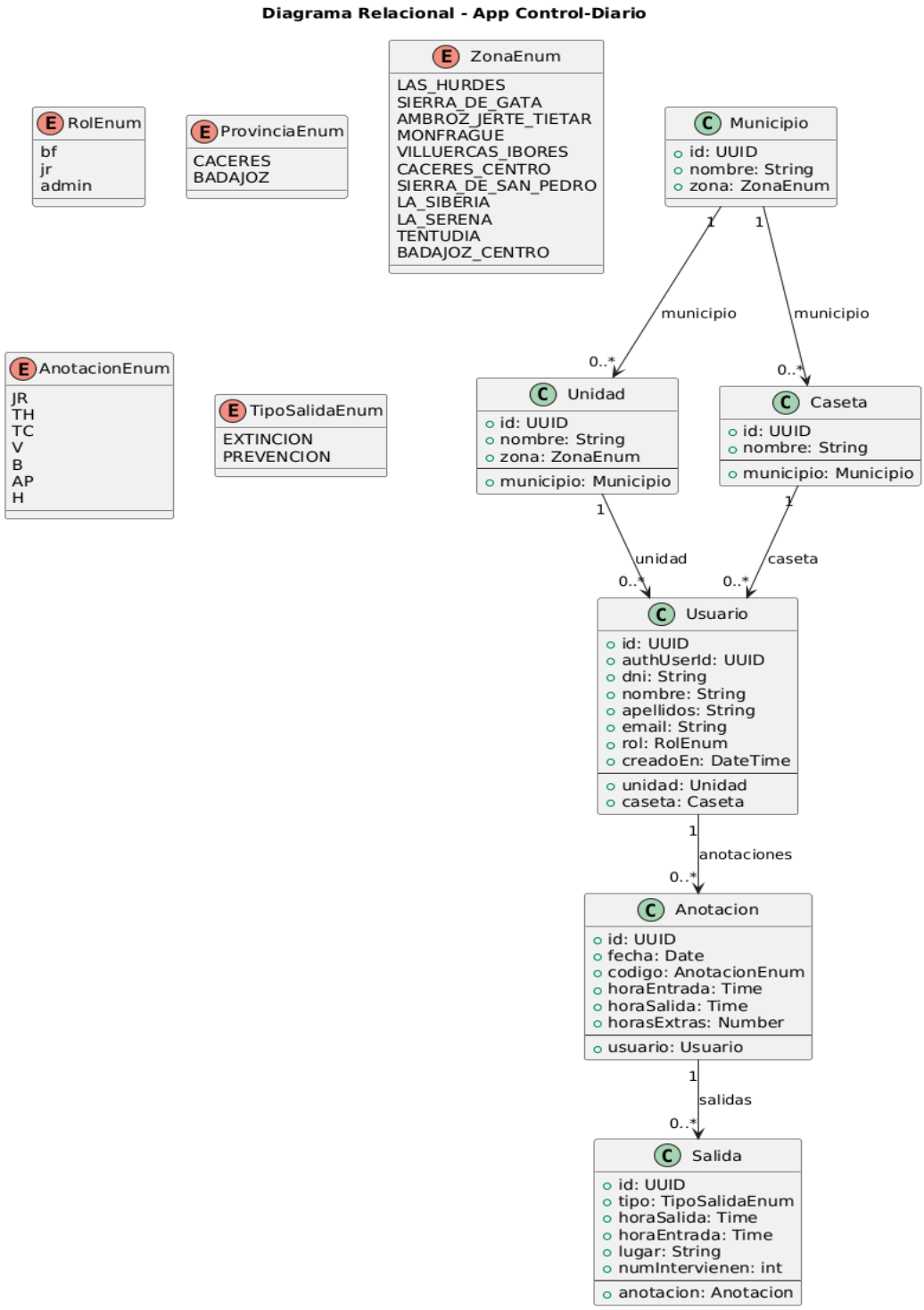


Diagrama de despliegue

Este diagrama especifica como se comporta mi -App Control-Diario y como se conecta con las herramientas que la forman

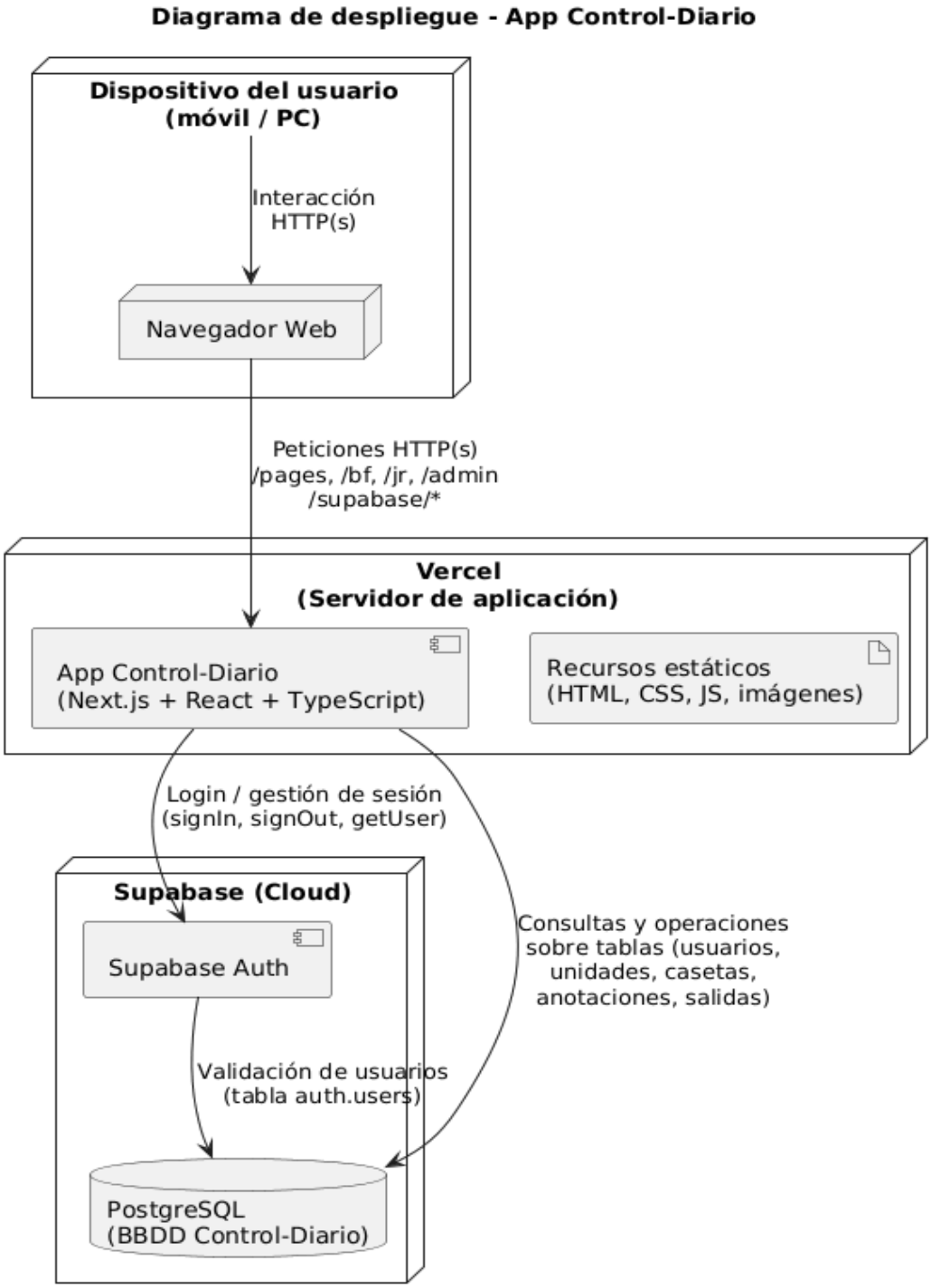
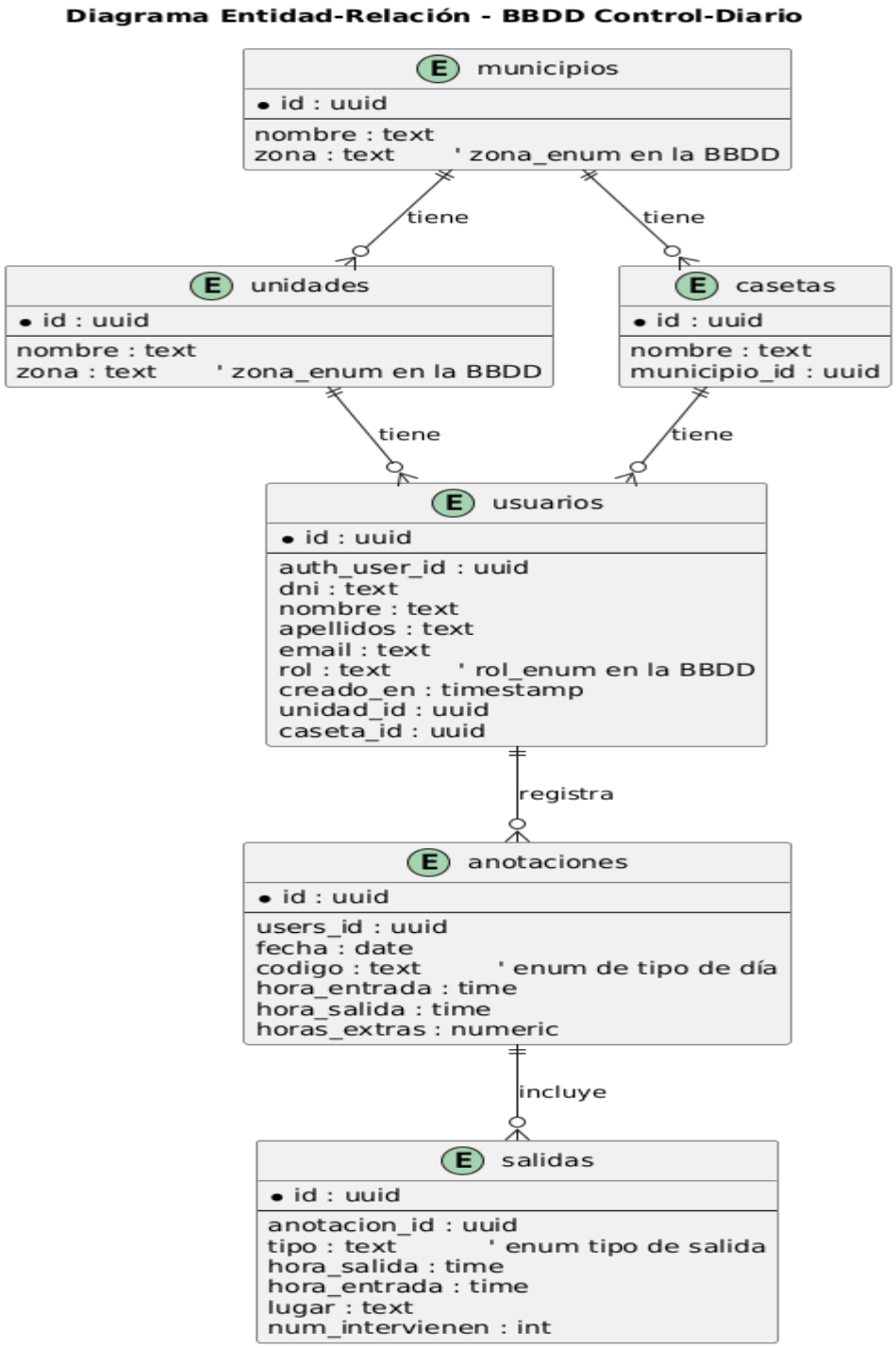
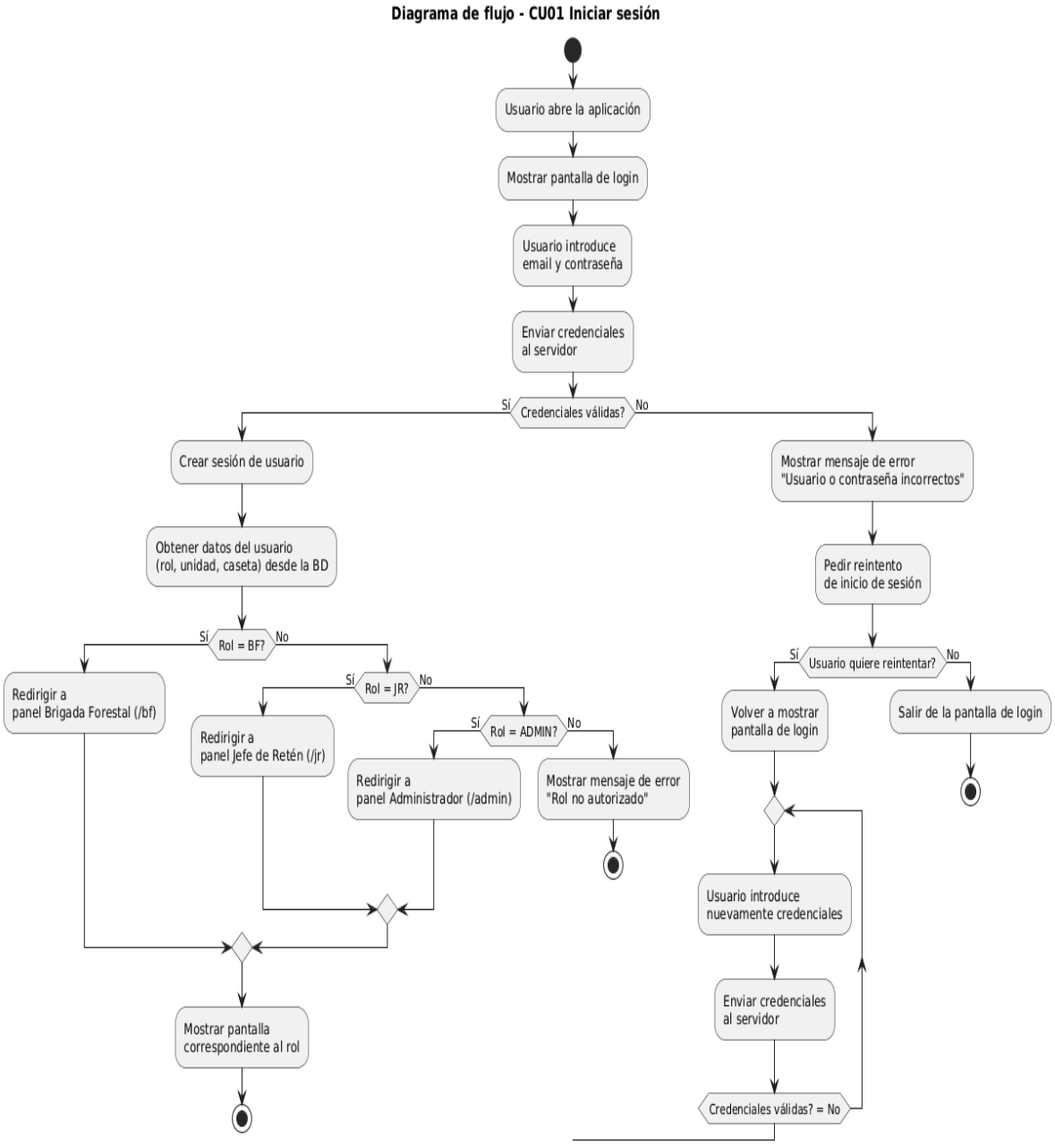


Diagrama de Entidad-Relación, que nos informa de las relaciones que existen entre las entidades de mi aplicación.



Flujo de inicio de sesión controlado por el archivo middleware.ts, en que controla el acceso según el tipo de rol asignado a cada usuario. Además de la seguridad que ofrece Supabse con la tabla de autenticación de usuarios.



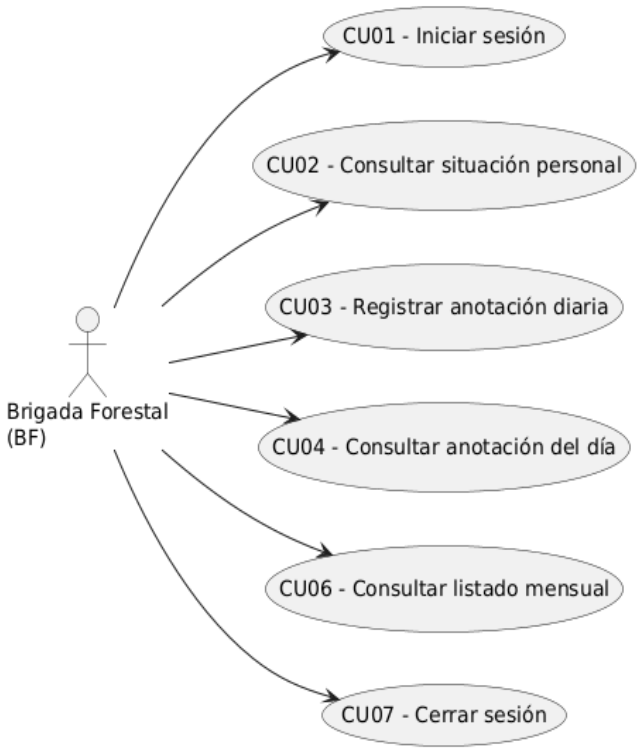
Casos de Uso

A continuación enumeramos los casos de uso y los exponemos según su tipo de rol.

CU01 – Iniciar sesión
CU02 – Consultar situación personal
CU03 – Registrar anotación diaria
CU04 – Consultar anotación del día
CU05 – Registrar salidas de servicio *(solo JR o personal autorizado)*
CU06 – Consultar listado mensual
CU07 – Cerrar sesión
CU08 – Consultar anotaciones del equipo *(JR)*
CU09 – Consultar salidas del equipo *(JR)*
CU10 – Generar resumen de actividad *(JR)*
CU11 – Gestionar usuarios *(Admin)*
CU12 – Gestionar estructura organizativa *(Admin)*
CU13 – Consultar listados globales *(Admin)*

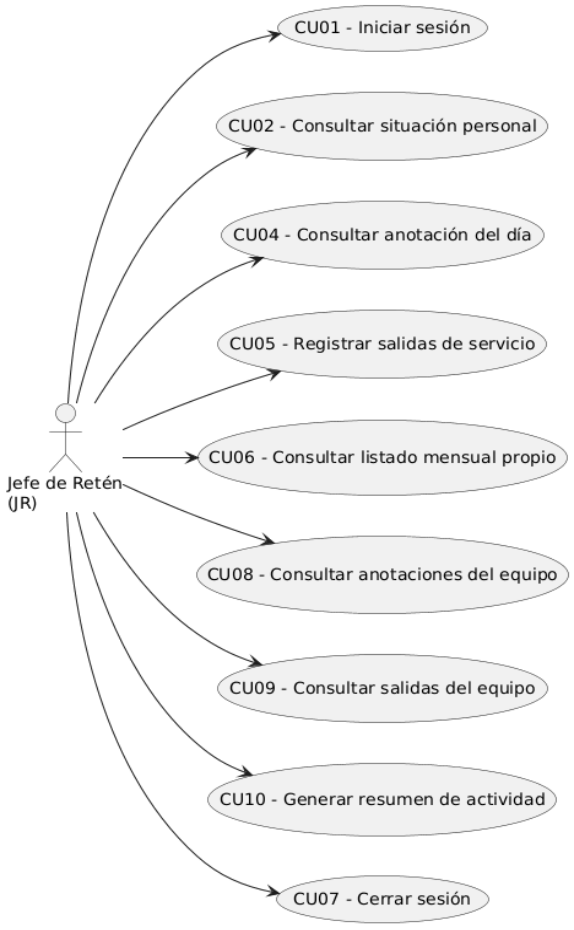
Casos de uso según su rol:

Bombero Forestal



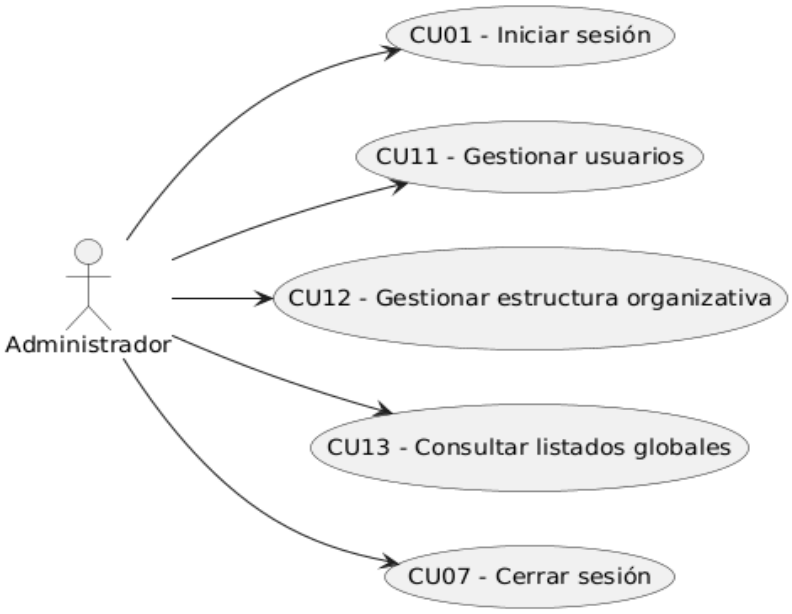
- **CU01 – Iniciar sesión**
  El usuario introduce sus credenciales (correo y contraseña) para acceder a la aplicación.
- **CU02 – Consultar situación personal**
  Permite al usuario ver su información básica: unidad, caseta, rol y estado laboral actual.
- **CU03 – Registrar anotación diaria**
  El usuario envía la solicitud del día, indicando fecha, código del día y, en su caso, horas extra, vacaciones o asuntos propios
- **CU04 – Consultar anotación del día**
  El usuario consulta los datos de la anotación correspondiente al día actual (o a un día seleccionado), comprobando la información registrada.
- **CU06 – Consultar listado mensual**
  Muestra un listado de las anotaciones del usuario en un mes determinado, incluyendo horarios y horas extra.
- **CU07 – Cerrar sesión**
  El usuario finaliza la sesión en la aplicación, cerrando su acceso y limpiando la sesión activa

Jefe de Servicio



- **CU01 – Iniciar sesión**
  Permite al JR autenticarse en el sistema y acceder a su panel de trabajo.
- **CU02 – Consultar situación personal**
  El JR puede ver su información personal y asignación dentro de la organización.
- **CU04 – Consultar anotación del día**
  Permite al JR revisar su anotación propia del día actual o de un día concreto.
- **CU05 – Registrar salidas de servicio**
  El JR registra las salidas/intervenciones realizadas durante la jornada, indicando tipo de salida, lugar, horario y personas que intervienen.
- **CU06 – Consultar listado mensual propio**
  El JR puede consultar sus anotaciones de un mes específico.
- **CU08 – Consultar anotaciones del equipo**
  Permite al JR ver las anotaciones diarias del personal que pertenece a su unidad.
- **CU09 – Consultar salidas del equipo**
  El JR visualiza las salidas registradas por su equipo, con detalle de lugar, horario y número de intervinientes.
- **CU10 – Generar resumen de actividad**
  El sistema genera un resumen global de la actividad del equipo (por ejemplo, por día o por mes: número de servicios, horas totales, etc.).
- **CU07 – Cerrar sesión**
  El JR finaliza su sesión en la aplicación.

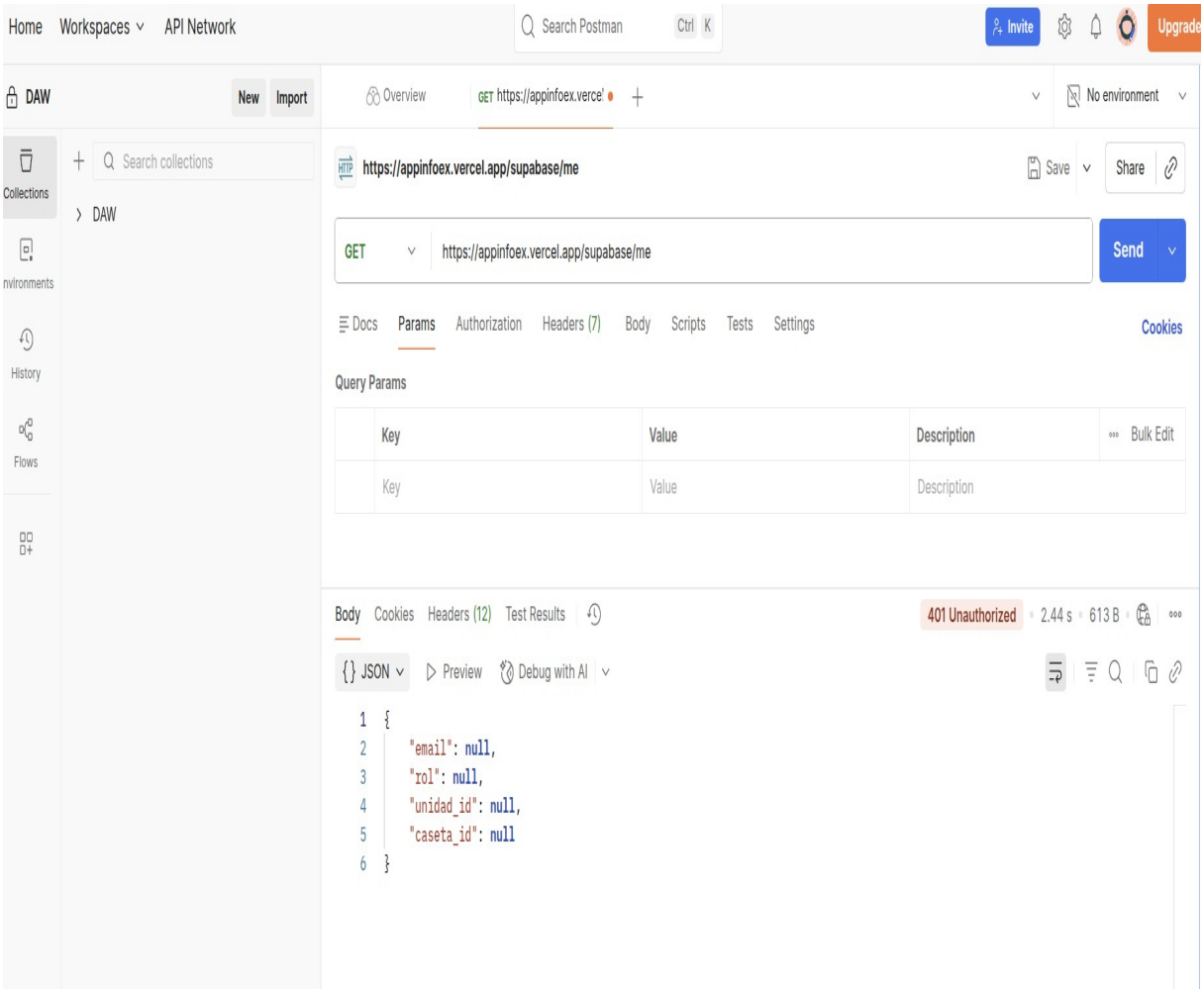
Administrador



- **CU01 – Iniciar sesión**
  Permite al administrador identificarse en el sistema y acceder al panel de administración.
- **CU11 – Gestionar usuarios**
  El administrador puede crear nuevos usuarios, modificar sus datos, asignarles rol (BF, JR, ADMIN), unidad y caseta, así como eliminar usuarios del sistema cuando sea necesario.
- **CU12 – Gestionar estructura organizativa**
  Permite administrar la jerarquía de la organización: unidades, casetas, municipios y zonas. El administrador puede crear, editar o eliminar usuarios perteneciendo a unos u otros elementos.
- **CU13 – Consultar listados globales**
  Proporciona acceso a listados e informes globales de la organización: anotaciones mensuales, horas trabajadas, salidas registradas, etc., filtrados por zona, unidad, caseta o trabajador.
- **CU07 – Cerrar sesión**
  El administrador cierra su sesión y abandona el sistema de forma segura.

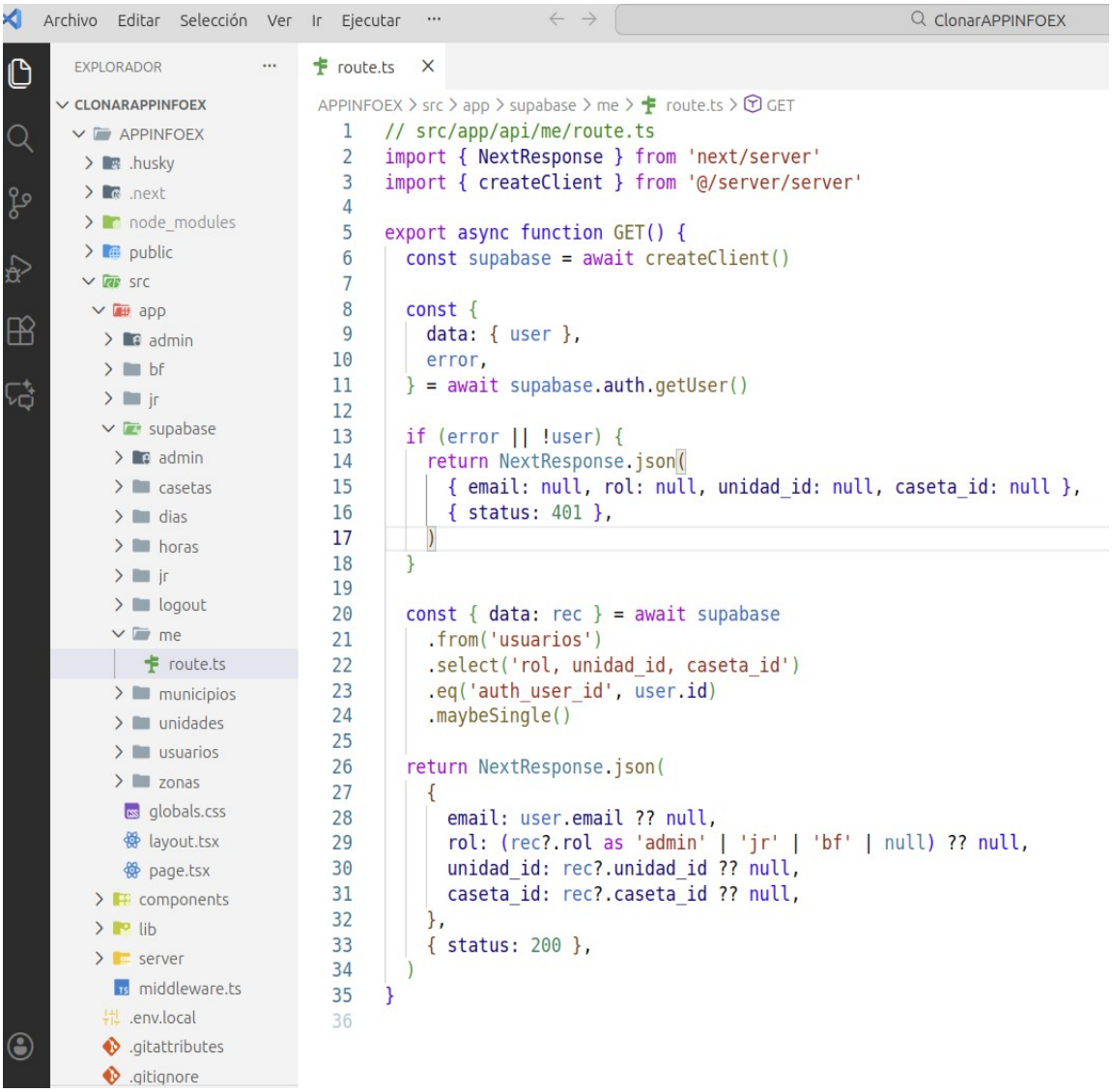
Pruebas realizadas

Para comprobar las funcionalidades de la aplicación, se han realizado pruebas manuales introduciendo usuarios con emails y contraseñas aleatorias directamente desde Supabase , pero voy a mostrar una de ellas que se realiza de forma sencilla con la app Postman, realizando las peticiones sobre las url.



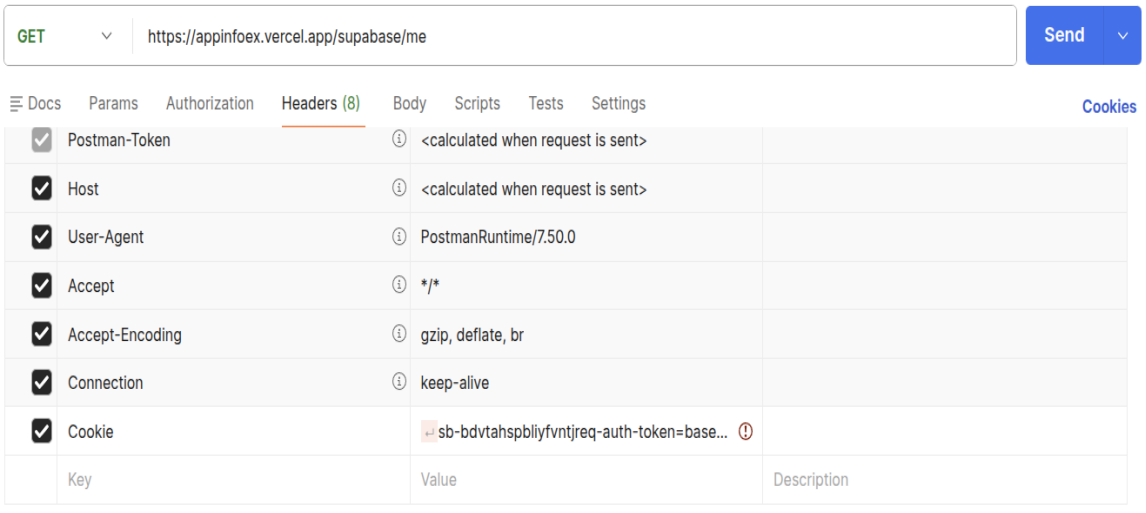
En esta captura he realizado el envío de la petición GET a través de Postman pero recibo un 401 (no estoy autorizado), y el email, rol y la unidad o caseta a la que pertenece este usuario me dan resultado ‘null’.

La url es https://appinfoex.vercel/supabase/me donde supabase/me/ es :

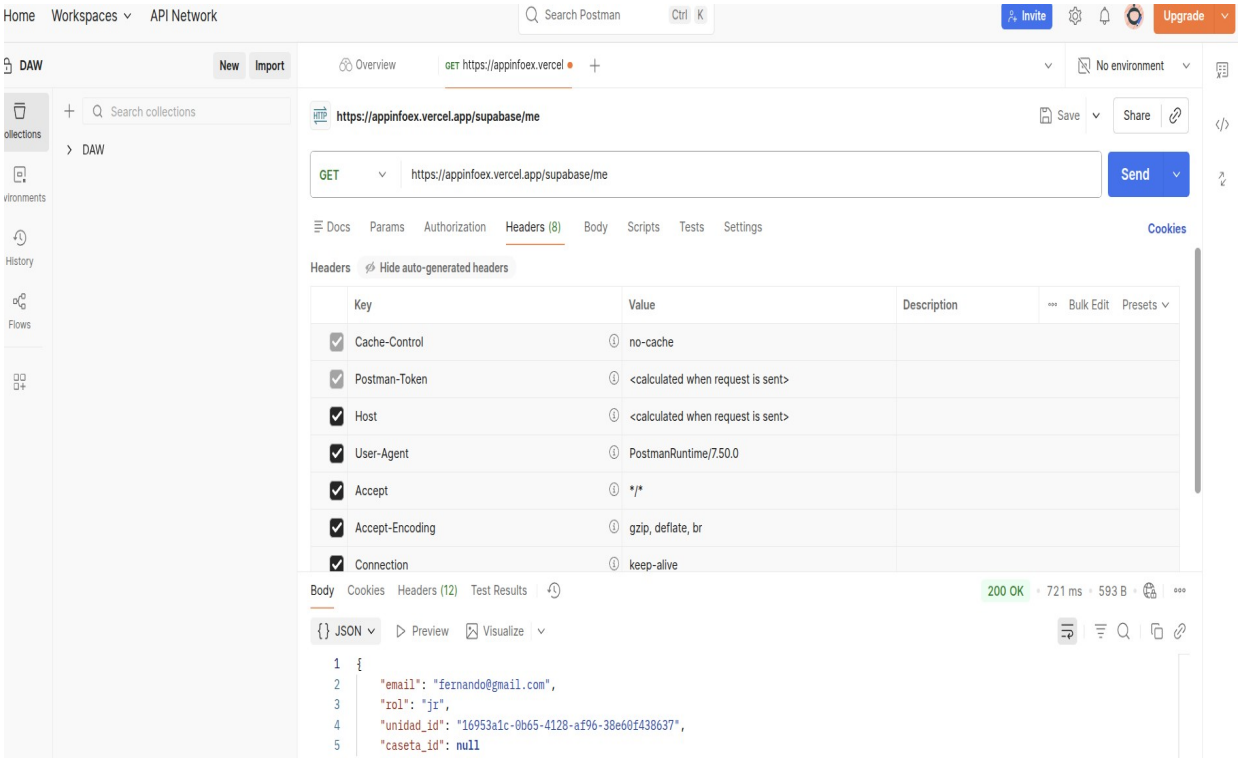


Ahora voy a entrar en la app con las claves y contraseñas y voy a copiar la cookie que se genera en el navegador:

Las voy a pegar en Postman en las Headers: Key es : Cookie y Value: todo el texto que he copiado del navegador, en la imagen solo se aprecia le primera linea



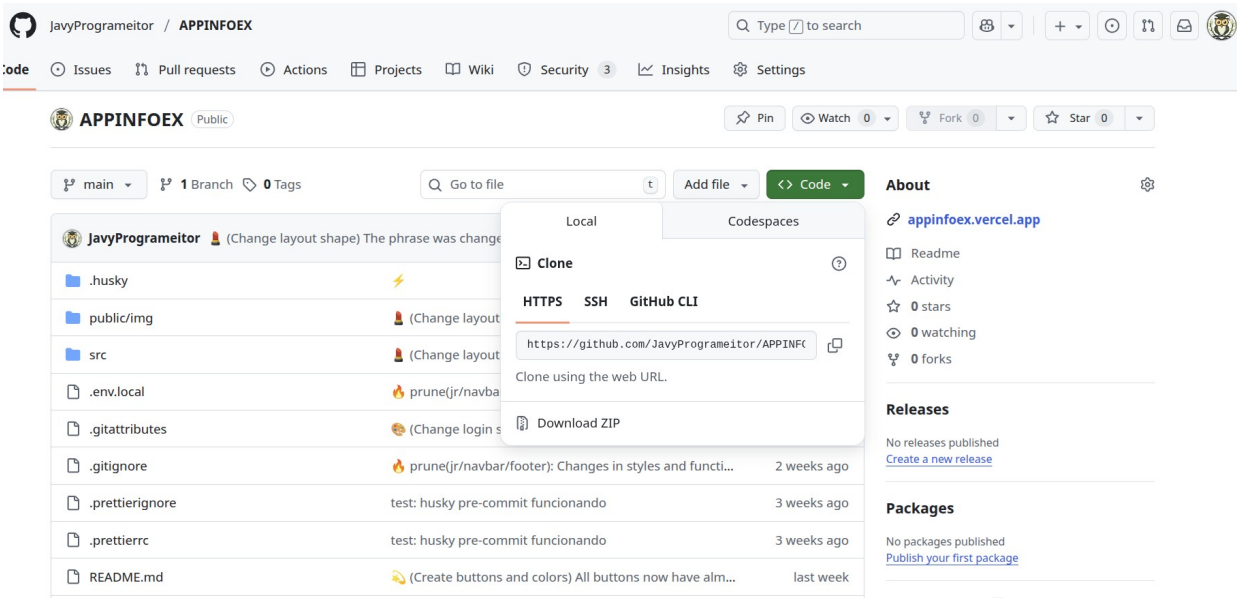
Y ahora tenemos acceso desde Postman, por lo tanto podemos decir que este acceso esta controlado:



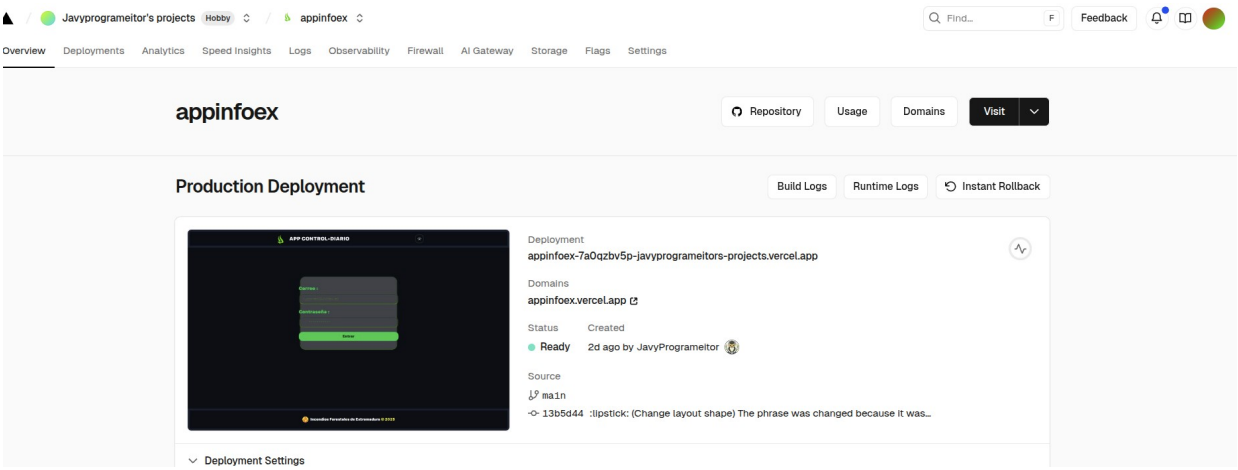
De esta forma se pueden testear todas las peticiones que se realizan desde mi app . Como vemos el resultado ahora es un: 200 OK

Proceso de despliegue

El despliegue de la aplicación *Control-Diario* se llevó a cabo utilizando un entorno completamente basado en la nube, combinando los servicios de **Vercel** para el servidor de aplicación y **Supabase** para la base de datos y autenticación. La aplicación, construida en **Next.js + React + TypeScript**, se versionó mediante GitHub.

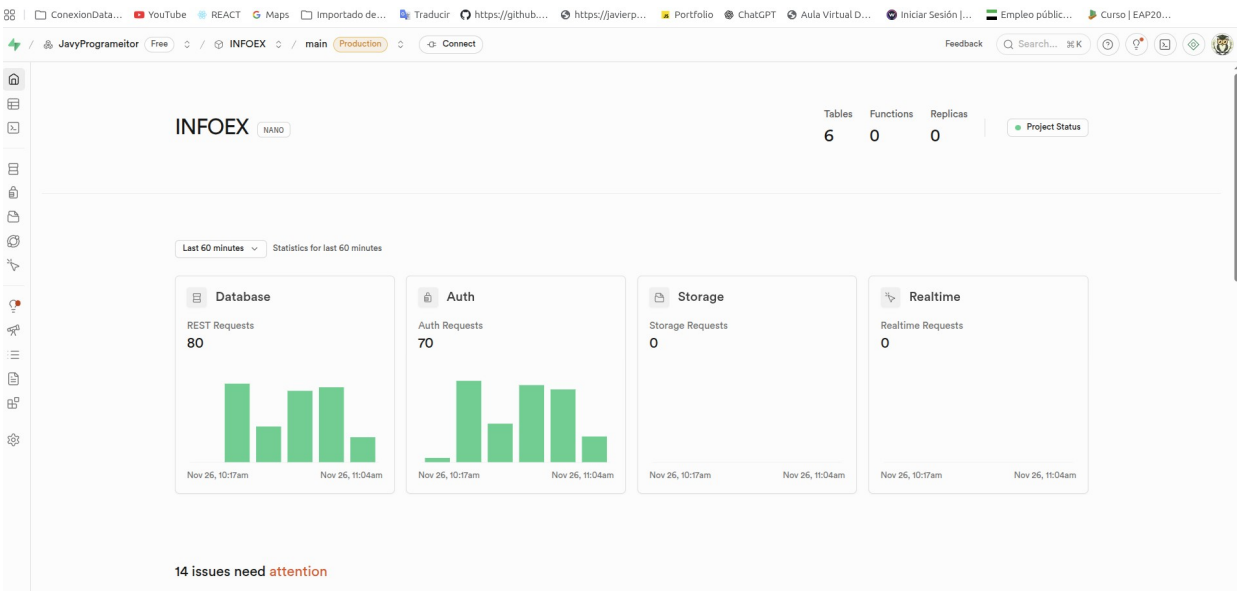


Desde este repositorio se conectó directamente a Vercel, plataforma especializada en despliegues automáticos de aplicaciones JavaScript (https://vercel.com).

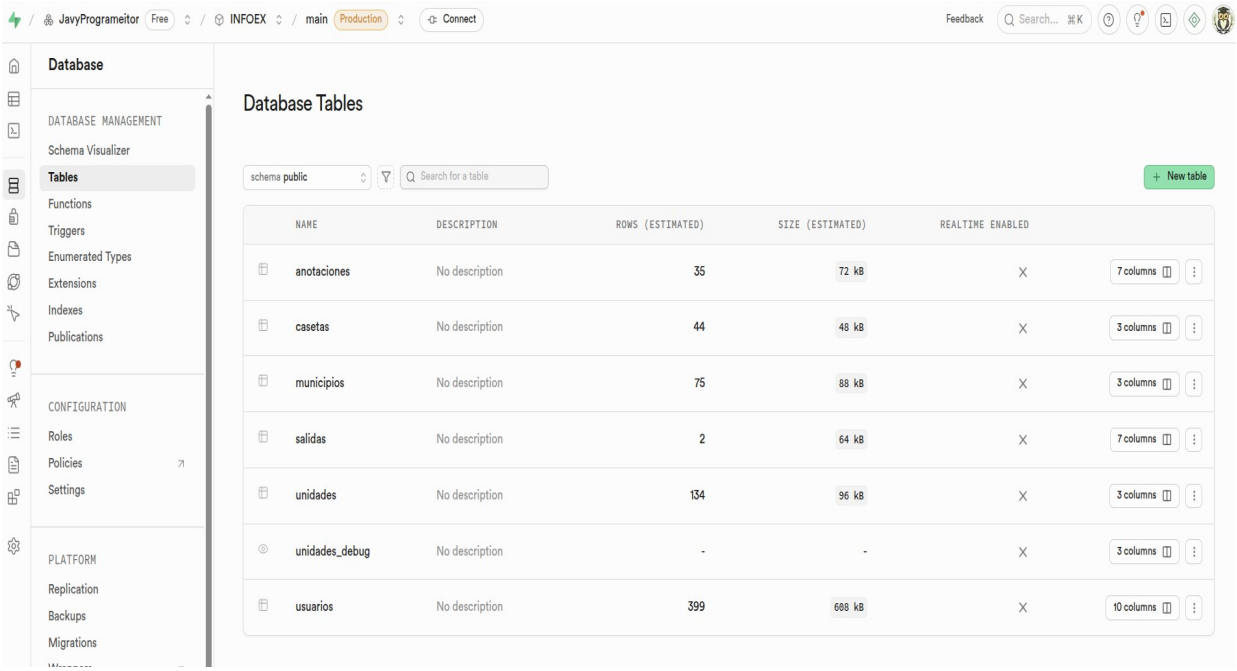


Una vez vinculado el proyecto, Vercel detectó automáticamente la configuración de Next.js y ejecutó el proceso de *build*, generando las páginas, los recursos estáticos y las funciones serverless que dan soporte a los *route handlers* situados en `/supabase/*`.

Paralelamente, se configuró en **Supabase** (https://supabase.com) un proyecto que actúa como backend de datos. Supabase proporciona una base de datos PostgreSQL gestionada, junto con un sistema de autenticación basado en `auth.users`. Desde su panel web se definieron las tablas del dominio (`usuarios`, `unidades`, `municipios`, `casetas`, `anotaciones`, `salidas`) y se configuraron las políticas de seguridad (RLS) que restringen el acceso a los datos según el rol del usuario. También se habilitó el servicio **Supabase Auth**, encargado de validar credenciales y generar sesiones seguras.



Durante el despliegue en Vercel se configuraron las variables de entorno `NEXT_PUBLIC_SUPABASE_URL` y `NEXT_PUBLIC_SUPABASE_ANON_KEY`, necesarias para que la aplicación pueda conectarse a Supabase tanto desde el navegador como desde el servidor. Gracias a la librería `@supabase/ssr`, el servidor de Vercel integra estas credenciales dentro del sistema de cookies de Next.js, lo que permite gestionar sesiones, validar tokens y obtener el usuario actual en cada petición.



Una vez desplegada, la arquitectura funciona del siguiente modo: el usuario accede desde un dispositivo móvil o PC a la URL pública del proyecto en Vercel, la cual sirve los recursos estáticos y las vistas de la aplicación. Cuando el usuario realiza operaciones como iniciar sesión, consultar anotaciones o registrar salidas, el navegador envía peticiones HTTPS hacia Vercel. Si la operación requiere acceso a datos, Vercel contacta con Supabase, que procesa la consulta o modificación sobre PostgreSQL y devuelve la respuesta. Este enfoque desacopla completamente la capa de aplicación y la capa de datos, proporcionando un sistema moderno, escalable y altamente disponible, optimizado para un uso intensivo desde dispositivos móviles y entornos distribuidos.

Propuestas de mejoras

Aunque *Control-Diario* cumple con las funciones básicas de registro de jornadas, salidas y administración del personal, existen diversas mejoras que permitirían convertirla en una herramienta más completa y alineada con las necesidades reales del servicio.
Una primera mejora sería la unificación del diseño visual, ya que algunas vistas presentan estilos diferentes.

También es recomendable la ampliación del catálogo de códigos de jornada, incorporando todos los códigos oficiales y permitiendo su gestión desde la administración para asegurar precisión en el registro.

En el área administrativa, se propone añadir paneles de estadísticas y visualización de datos, incluyendo gráficos de actividad por zonas, horas trabajadas y número de intervinientes, lo que facilitaría la toma de decisiones.
Otra mejora clave es adaptar la aplicación a los distintos niveles de peligrosidad (alto, medio, bajo), ajustando automáticamente el calendario, los códigos disponibles y la validación de jornadas según el ciclo laboral real del servicio.

Además, sería útil incorporar una función para anular peticiones dentro de un periodo de tiempo, permitiendo corregir errores o replanificaciones del personal.
A nivel técnico, la aplicación se beneficiaría de una refactorización del código, reduciendo duplicidades y facilitando el mantenimiento.

Finalmente, como mejoras opcionales a medio plazo, podrían añadirse notificaciones internas, exportación de datos en PDF/CSV y un modo offline para zonas con poca cobertura.

En conjunto, estas mejoras incrementarían la precisión, la usabilidad y la capacidad de análisis del sistema, beneficiando tanto a los operativos como a los administradores del servicio.

Bibliografía

Para el desarrollo de esta aplicación se han utilizado los siguientes frameworks, herramientas, tecnologías y fuentes consultadas:

Frameworks y Lenguajes

- Next.js – Documentación oficial: https://nextjs.org/docs
- React – Documentación oficial: https://react.dev/
- TypeScript – Documentación oficial: https://www.typescriptlang.org/docs/
- Node.js – Documentación oficial: https://nodejs.org/en/docs

Base de datos, Backend y Autenticación

- Supabase – Documentación general: https://supabase.com/docs
- Supabase Auth – Autenticación y gestión de sesiones: https://supabase.com/docs/guides/auth
- Supabase PostgreSQL – Gestión de base de datos: https://supabase.com/docs/guides/database
- Supabase JavaScript Client (@supabase/supabase-js): https://supabase.com/docs/reference/javascript
- Supabase SSR y Next.js: https://supabase.com/docs/guides/auth/server-side/nextjs

Despliegue y DevOps

- Vercel – Plataforma de despliegue: https://vercel.com/docs

Estilos y diseño UI

- Tailwind CSS – Documentación completa: https://tailwindcss.com/docs
- Radix UI – Componentes accesibles: https://www.radix-ui.com/docs
- Lucide Icons – Iconos SVG: https://lucide.dev/
- Shadcn UI – Componentes basados en Radix: https://ui.shadcn.com/docs

Control de calidad, desarrollo y automatización

- ESLint – Análisis de código: https://eslint.org/docs/latest
- Prettier – Formateador: https://prettier.io/docs/en/
- Husky – Hooks de Git: https://typicode.github.io/husky/
- lint-staged – Formateo previo al commit: https://github.com/okonet/lint-staged
- Commitizen – Convencionalización de commits: https://commitizen.github.io/cz-cli/
- cz-emoji – Estilo de commits con emojis: https://github.com/cz-emoji/cz-emoji

Herramientas de desarrollo

- Visual Studio Code: https://code.visualstudio.com/docs
- Postman – Pruebas de API: https://www.postman.com/
- dbdiagram.io – Diagramas E-R: https://dbdiagram.io/d

Herramientas auxiliares para diagramas

- Graphviz – Generación de diagramas: https://graphviz.org/documentation/
- diagrams.net (Draw.io) – Diagramas UML: https://www.diagrams.net/doc

Tecnologías web generales

- MDN Web Docs – HTML, CSS y JS: https://developer.mozilla.org/
- ECMAScript – Estándar JavaScript: https://tc39.es/ecma262/

Blogs recomendados

. Alexander Vega: https://alexvega.net/blog