Universidad de Alcalá
Escuela Politécnica Superior

Máster Universitario en Desarrollo Ágil de Software para la Web

Trabajo Fin de Máster

Desarrollo de una aplicación móvil sin backend con el uso de PostgREST

Autor: Joswald Leonardo Martínez Bautista
Tutor: José María Gutiérrez Martínez
Cotutor: Sergio de la Mata Moratilla

2023

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Máster Universitario en Desarrollo Ágil de Software para la Web

Trabajo Fin de Máster
Desarrollo de una plataforma de gamificación en el entorno universitario

Autor: Joswald Leonardo Martínez Bautista
Tutor: José María Gutiérrez Martínez
Cotutor: Sergio de la Mata Moratilla

TRIBUNAL:

Presidente:

Vocal 1º:

Vocal 2º:

FECHA: __ de _____ de 20__

Agradecimientos

Quiero expresar mis agradecimientos a todas las personas que contribuyeron en el desarrollo de este trabajo de fin de master por su inquebrantable apoyo y orientación Su dedicación, conocimiento y paciencia fueron fundamentales para el éxito de este proyecto. Gracias a su guía constante, pude crecer académicamente y desarrollar nuevas habilidades que sin duda serán valiosas en mi futuro profesional. Agradezco sinceramente a mis tutores, José María Gutiérrez Martínez y Sergio De La Mata Moratilla, por su invaluable orientación y apoyo durante la realización de mi Trabajo de Fin de Máster.

No puedo dejar de mencionar el apoyo incondicional de mi familia. Su amor y aliento fueron mi fuente de fortaleza durante este desafiante período. Agradezco profundamente su sacrificio y comprensión en todo momento.

También quiero reconocer a mis compañeros de clase, cuya colaboración y amistad fueron esenciales en este viaje académico. Sus perspectivas y consejos contribuyeron en gran medida a enriquecer mi trabajo.

Por último, pero no menos importante, deseo expresar mi gratitud a la Universidad de Alcalá por brindarme la oportunidad de llevar a cabo este proyecto y por ofrecerme una educación de calidad que ha sido fundamental para mi crecimiento personal y profesional.

Índice

Contenido

Índice de imágenes

Índice de tablas

Resumen

Para la creación de una página web era necesario desarrollar tanto las interfaces de usuario (Frontend) como la capa que desarrolla los cálculos y la conexión a la base de datos (Backend) lo que influye en la eficacia del proceso de desarrollo del software.

Utilizando la herramienta PostgREST y el framework Ionic hemos desarrollado una aplicación móvil sin necesidad de programar un backend tradicional. Esta aplicación nos permite gestionar planes estratégicos, generar informes y facilitar la toma de decisiones.

Palabras Clave

Backend, Frontend, PostgREST, Ionic, API.

Abstract

To create a website, you had to develop both the user interfaces (Frontend) and the layer that develops the calculations and the connection to the database (Backend), which influences the effectiveness of the software development process.

Using the PostgREST tool and the Ionic framework we have developed a mobile application without the need to program a traditional backend. This application allows us to manage strategic plans, generate reports and facilitate decision making.

Keywords

Backend, Frontend, PostgREST, Ionic, API.

1. Introducción

En las acciones vinculadas a la tecnología y el desarrollo de software, las interfaces de programación de aplicaciones (Apis) desempeñan un papel fundamental en la evolución de la manera en que las aplicaciones se comunican entre sí.

Hace algunos años, cuando la WEB comenzaba a tomar forma, la comunicación entre aplicaciones y sistemas se basaba en métodos rudimentarios, como SSR, MVC y otros que aún se utilizan como AJAX y SOCKETS, cada uno de ellos tiene sus ventajas, pero en cada caso existen limitaciones; como son la complejidad en el desarrollo, flexibilidad limitada y poca interoperabilidad con otros sistemas[15], por lo que en las aplicaciones desarrolladas que tomaban como fundamento el intercambio de datos en formatos personalizados la interoperabilidad era un desafío constante.

Se requería dar respuestas a estas limitaciones, fue entonces cuando surgió en el año 2000 un enfoque arquitectónico revolucionario llamado Transferencia de Estado Representacional (REST), ésta dio vida a las APIs RESTful, que prometían superar los desafíos planteados los demás métodos [2],[3].

El PostgREST, según veremos en el desarrollo de este trabajo es una herramienta que simplifica y potencializa la creación de las APIs RESTful, se basa en la idea audaz de que la estructura de una base de datos PostgreSQL puede convertirse automáticamente en una API REST ful completa [11].

En este proyecto exploramos cómo PostgREST allana el camino para un acceso seguro a los datos almacenados y como su uso disminuye el tiempo de programación, elementos esenciales para incidir en altos niveles de eficiencia.

Existen áreas en que la eficiencia de un software es altamente valorada, específicamente la empresarial y ejecutiva, sobre todo en la planificación, seguimiento y monitoreo de sus procesos, por ello el desarrollo de una herramienta basada PostgREST podría cumplir con esos estándares de eficiencia.

La creación de una aplicación móvil utilizando esta herramienta nos permitirá comprobar su eficacia en el manejo de datos y en ahorro de tiempo del desarrollo, propiciando el terreno para un emocionante futuro de aplicaciones móviles y servicios web interconectados.

2. Objetivo

El objetivo principal de este proyecto es verificar la viabilidad para desarrollar una aplicación móvil sin la necesidad de crear un backend tradicional, aprovechando las ventajas de la herramienta PostgREST y empleando el framework Ionic.

2.1 Objetivos específicos

Para dar cumplimiento a nuestro objetivo general de este proyecto se requieren acciones puntuales, que delimiten las actividades que vamos a desarrollar, estas acciones están planteadas como objetivos específicos a ser logrados, están organizados de manera secuencial y cada uno de ellos representa un paso en el camino de lograr el resultado final.

- Configurar y establecer una base de datos utilizando PostgreSQL, con tablas y funciones necesarias para respaldar la funcionalidad de la aplicación.

- Implementar la herramienta PostgREST para convertir la base de datos en un apiREST funcional.

- Desarrollar una aplicación móvil que permita a los usuarios gestionar y administrar planes o programas específicos.

- Generar informes a partir de los datos almacenados en la base de datos a través de la aplicación móvil, proporcionando a los usuarios información útil en formatos amigables.

3. Estado del arte

Para este proyecto realizamos una revisión documental de las informaciones disponibles en la WEB acerca de los temas esenciales del trabajo, Las informaciones fueron revisadas, analizadas y sintetizadas para describirlas en este acápite.

3.1 Breve historia de las APIs

Una API, o *interfaz de programación de aplicaciones*, es un conjunto de reglas que indican cómo las aplicaciones o los dispositivos pueden conectarse y comunicarse entre sí.[23]

Las APIs ofrecen una serie de ventajas a otras alternativas del mundo de la tecnología y los datos; como garantizar la seguridad, al permitir el cifrado de extremo a extremo**,** facilitan el seguimiento de quién accede a la información y brindan la capacidad de auditar esos accesos, lo que mejora la confianza de los clientes [5].

El origen de las APIs se remonta a finales de la década de los 60 cuando los primeros sistemas de computación se empezaron a desarrollar. Aunque el primer protocolo que empezó a ganar popularidad fue llamado a Procedimiento Remoto (RPC) desarrollado por SUN Microsystems ya en 1988[6].

A lo largo de la historia, las interfaces de comunicación han experimentado notables transformaciones. La revolución llegó con la creación de SOAP por parte de Microsoft, que es un protocolo ligero para el intercambio de información en entornos descentralizados y distribuidos [19]. Esto marcó un cambio significativo en comparación con CORBA, que previamente había dominado la escena y empleaba protocolos binarios para la comunicación [6].

A pesar de que SOAP tenía un rendimiento inferior en comparación con los protocolos binarios, su simplicidad y facilidad de uso le dieron principalía. Esto resultó en un crecimiento exponencial de los servicios web, permitiendo a muchas empresas exponer sus servicios a terceros o consumir servicios de éstos. A partir de este punto, el modelo continuó evolucionando [6].

Con la popularización de Transferencia de Estado Representacional (REST), se habilitó la utilización de múltiples formatos de datos, destacando JSON por encima de otros. Esto simplificó aún más la creación de servicios web y dio lugar a un ecosistema de integraciones que agilizan las operaciones y facilitan el intercambio de información entre diversas organizaciones.[6]

3.2 Importancia de las APIs

Las APIs son mecanismos que permiten a dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos de manera eficiente [16]. Aunque no son perceptibles para los usuarios finales de una aplicación, desempeñan un papel fundamental en la experiencia que éstos experimentan. Las APIs son componentes esenciales en el ámbito del desarrollo web, simplificando la vida de los usuarios y de los desarrolladores [17].

3.2.1 Api funcional

Una API debe ser versátil, funcional y debe cumplir eficazmente su propósito, por ello según la página https://hostingato.com es esencial que reúna al menos los siguientes atributos [12]:

- Cada API debe incluir su propia documentación, dado que existe una variedad de tipos de API, y cada una requiere su propio manual para comprender su funcionamiento y utilizarla correctamente.

- Debe proporcionar una biblioteca de clientes, lo que significa que la API debe contener un código modular reutilizable. Esto permitirá a los desarrolladores acceder a la API de manera eficiente.

- Es fundamental indicar claramente la versión de la API, ya que esto garantiza la compatibilidad y facilita las actualizaciones futuras.

- La API debe ser rápida, ya que, aunque sea funcional, la lentitud puede sobrecargar el servidor y afectar negativamente la experiencia del usuario.

- La alta disponibilidad es un requisito esencial, lo que significa que la API debe estar siempre disponible, los 365 días del año y las 24 horas del día.

- Es importante realizar una monitorización constante y análisis de los usuarios de la API. Esto proporcionará información valiosa, como quién la utiliza y con qué propósito.

- Fomentar la creación de una comunidad en torno a la API es beneficioso en el entorno tecnológico actual. Una comunidad activa brinda apoyo mutuo a los usuarios y desarrolladores, lo que puede mejorar significativamente el desarrollo y uso de la API.

3.2.2 Evolución de las APIs RESTful

Una API REST es un estilo arquitectónico para una interfaz de programa de aplicación (API) que usa solicitudes HTTP a la hora de acceder y utilizar datos.[25] Las API de RESTful han ganado popularidad masiva debido a su interoperabilidad y flexibilidad en la web [18]. A lo largo de los años, hemos visto la transición desde la exposición directa de bases de datos a través de CGI hasta el enfoque más estructurado y orientado a recursos de REST.

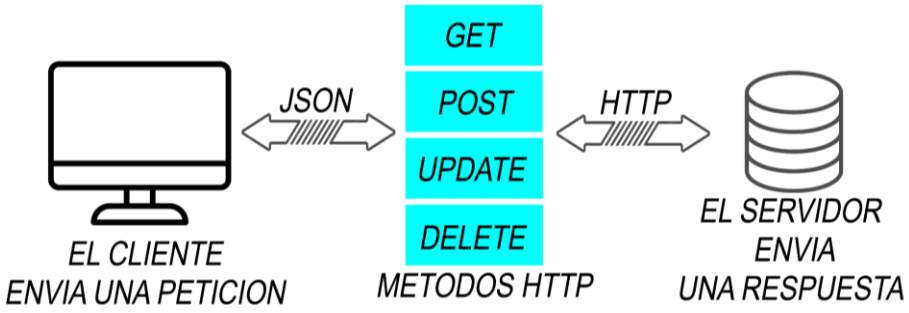


Ilustración 1. diagrama funcionamiento de un api

Las APIs basadas en la arquitectura REST, dada su naturaleza ligera, se presentan como una elección idónea para entornos emergentes y contemporáneos, tales como el Internet de las cosas (IOT), el desarrollo de aplicaciones móviles y la informática sin servidor [2].

3.3 Herramienta PostgREST

Para el desarrollo de una API se utilizan diferentes lenguajes de programación y herramientas apropiadas, cada una de ellas tiene sus fortalezas y desafíos, aunque el resultado esperado es siempre que ésta pueda realizar las funciones para las que fue diseñada. Revisemos a continuación algunos elementos conceptuales y características de una API utilizando el PostgREST, alternativa diferente para desarrollar una API desde cero.

3.3.1 PostgREST

PostgREST se basa en la idea de que la estructura de la base de datos PostgreSQL puede utilizarse para definir automáticamente una API RESTful completa [11]. Algunos de los conceptos fundamentales incluyen:

- **Tablas como recursos:** Las tablas de la base de datos se convierten en recursos accesibles a través de la API, lo que simplifica la exposición de datos.
- **Autenticación y autorización:** PostgREST ofrece opciones para gestionar la autenticación y autorización de manera flexible, incluyendo la integración con sistemas de seguridad existentes.

Ilustración 2. diagrama funcionamiento de PostgREST

Un tema que hay que tener en cuenta es que los desarrolladores deben colocar restricciones declarativas directamente en su base de datos para evitar cualquier tipo de corrupción de datos ya que PostgREST no se basa en un asignador relacional de objetos (ORM) ni en una codificación imperativa personalizada. [24]. Para mantener tiempos de respuesta rápidos, delega la mayor parte de la parte de cálculo a la base de datos, incluida la serialización de respuestas JSON directamente en SQL, la validación de datos y más. muestra tiempos de respuesta inferiores a un segundo para hasta 2000 solicitudes/segundo [24].

Estos tiempos se ven impulsados debido a 3 factores:

1. Primero, el servidor está escrito en Haskell usando el servidor HTTP Warp (un lenguaje compilado).
2. Luego, delega tantos cálculos como sea posible a la base de datos, incluida la serialización de respuestas JSON directamente en SQL, validando datos, etc.
3. Finalmente, utiliza la biblioteca Hasql para mantener un grupo de conexiones de base de datos, el protocolo binario PostgreSQL, y permanece sin estado para permitir el escalado horizontal.

PostgREST maneja la autenticación a través de JSON Web Tokens y delega la autorización de los datos a los roles definidos en la base de datos. Esto asegura que solo haya una fuente declarativa de verdad para la seguridad [26].

PostgREST está disponible para los principales sistemas operativos (Linux, Windows y Mac) [11].

A continuación, describimos una empresa que utiliza PostgREST.

Caso de uso (Nimbus)

Nimbus, una destacada empresa especializada en limpieza de oficinas y soluciones de movilidad ha adoptado PostgREST como parte fundamental de su infraestructura de aplicaciones y páginas web. Esta elección estratégica ha permitido a Nimbus optimizar sus operaciones y mejorar la experiencia de sus clientes en una variedad de aspectos clave [10].

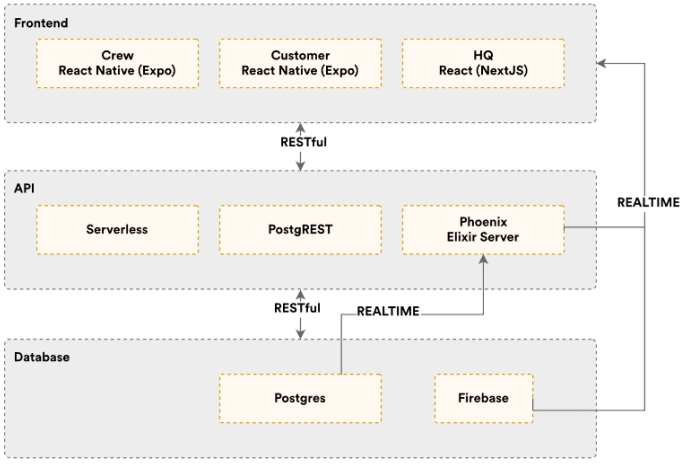


Ilustración 3. Estructura de Nimbus

Uso de base de datos con PostgreSQL para respaldar funcionalidad de una aplicación

La gestión de base de datos es uno de los componentes esenciales en el desarrollo de software. PostgreSQL es uno de los gestores de bases de datos más utilizadas por su flexibilidad y robustez. La exposición eficiente de datos almacenados en una base de datos es un desafío crítico en el desarrollo de aplicaciones web, desarrollo móvil y servicios RESTful. En este contexto, PostgREST aparece como una herramienta para facilitar el acceso a estos datos almacenados de una manera fácil y sencilla [8].

Entre las características notables de PostgreSQL se encuentran la capacidad de implementar replicación asíncrona y el soporte nativo para el almacenamiento de documentos en formatos como JSON o XML. Además, PostgreSQL ofrece funcionalidades avanzadas, como búsquedas de texto completas en la base de datos y una amplia gama de tipos de datos integrados, como rangos, arrays y capacidades de geolocalización [8].

3.4 Ionic

Es un framework diseñado para el desarrollo de aplicaciones híbridas que pueden funcionar en dispositivos móviles y computadoras. Su enfoque en la creación de aplicaciones multiplataforma permite a los desarrolladores escribir código una vez y utilizarlo en múltiples sistemas operativos, como Android, iOS y navegadores web [14]. Esto ofrece una gran ventaja al reducir la necesidad de realizar desarrollos separados para cada plataforma, lo que ahorra tiempo y recursos.

3.4.1 Características

Ionic tiene varias características que lo hacen ideales para este proyecto, de las cuales se destacan las siguientes:

- **Multiplataforma:** Con este framework es posible implementar aplicaciones que funcionen en múltiples plataformas, como iOS nativo, Android y la web como una aplicación web progresiva, todo con una única base de código [14]
- **Basado en estándares WEB:** Ionic se basa en tecnologías web confiables y estandarizadas: HTML, CSS y JavaScript, utilizando API modernas como Custom Elements y Shadow DOM [14].
- **Simplicidad:** Ionic está diseñado pensando en la simplicidad, de modo que la creación de aplicaciones sea divertida, fácil de aprender y accesible para casi cualquier persona con habilidades de desarrollo web.[14]

3.5 Aplicaciones móviles para gestionar o administrar planes

En la actualidad, existen diversas aplicaciones de gestión de proyectos que pueden facilitar y mejorar significativamente la administración de tus proyectos. Algunas de las más destacadas incluyen:

1. **Notion**: Notion es una herramienta multiuso en línea que combina funciones de gestión de proyectos, bases de datos y notas en una sola plataforma. [21].

2. **Trello**: Trello es una herramienta visual que permite a los equipos gestionar cualquier tipo de proyecto y flujo de trabajo, así como supervisar tareas.[20] Esta aplicación se basa en tableros con columnas y tarjetas para representar tareas y facilitar la colaboración en equipo. Además, se integra con otros servicios como Google Drive y Google Calendar [1].

3. **Coda**: Es una plataforma colaborativa todo en uno que combina lo mejor de documentos, hojas de cálculo y aplicaciones [22]. Aunque menos conocida, Coda es una aplicación versátil que permite crear documentos que evolucionan en herramientas complejas con fórmulas, tablas y más [1].

Cada una de estas aplicaciones tiene ventajas y desventajas distintivas que las hacen adecuadas para diferentes situaciones.

Notion se destaca por su versatilidad, ya que ofrece una amplia gama de funciones que abarcan desde la creación de documentos hasta la gestión de proyectos. Sin embargo, su versatilidad puede llevar a una curva de aprendizaje más empinada, ya que los usuarios pueden necesitar tiempo para explorar y aprovechar todas las características [1].

Trello, se enfoca en la simplicidad y la visualización de proyectos a través de tarjetas y tableros. Aunque ofrece integraciones útiles, su simplicidad puede limitar su utilidad en proyectos más complejos [1].

Coda se diferencia por su potencia en la personalización y la capacidad de evolucionar documentos en herramientas complejas. Sin embargo, su relativa falta de notoriedad podría dificultar su adopción, y su enfoque en la personalización puede requerir una curva de aprendizaje para sacar el máximo provecho de la herramienta [1].

3.5.1 Uso de PostgREST en generar aplicaciones de gestión de planes

Para una aplicación gestión de planes es necesario que ésta cumpla ciertos parámetros de interoperabilidad, versatilidad y robustez, además de que sea fácil de utilizar. Este tipo de herramienta requiere un alto nivel de interoperabilidad debido a que se requieren varios niveles que interactúan entre sí, esta interoperabilidad es uno de los factores que PostgREST facilita, además, que gracias a su uso sencillo y facilidad al momento de presentar los datos la convierte en una herramienta ideal para este proyecto; pues estas características son necesarias para establecer el monitoreo de las tareas con visión panorámica y en tiempo real del estado de los planes y sus componentes.

4. Desarrollo

En este apartado se definen las tecnologías y herramientas utilizadas para la creación de la aplicación. Además, se describe cómo fue utilizada la herramienta PostgREST para comprobar su eficacia, también se analiza la estructura de la base de datos que respalda nuestro proyecto, y la forma cómo utilizamos Ionic una tecnología esencial en la creación del frontend de nuestra aplicación, se describen además las acciones ejecutadas para lograr la comunicación con la herramienta PostgREST.

4.1 Tecnología y herramientas

Para el desarrollo de este software de gestión de planes estratégicos utilizamos diferentes tecnologías y herramientas, que nos permitieron el desarrollo de una aplicación sin backend, por lo que fue necesario utilizar las tecnologías más apropiadas para el desarrollo del frontend y la configuración de la base de datos las cuales describimos a continuación.

4.1.1 Base de datos PostgreSQL

Utilizamos un modelo de base de datos PostgreSQL para almacenar la información necesaria para el funcionamiento de la aplicación. PostgreSQL es un potente sistema de base de datos relacional de objetos, de código abierto, con más de 35 años de desarrollo activo, que le ha permitido ganar una sólida reputación por su confiabilidad, solidez de funciones y rendimiento [15].

Para que el proyecto funcione de manera óptima es esencial llevar a cabo una serie de tareas de configuración y desarrollo en PostgreSQL. Estas tareas incluyen:

- **Creación de roles:** que definen el nivel de acceso a los datos.
- **Definición de esquemas (schemas):** que sirven para organizar las tablas.
- **Creación de tablas:** para almacenar la información.
- **Desarrollo de funciones:** estas funciones sirven para presentar información personalizadas y llevar a cabo procesos como el inicio de sesión.
- **Configuración de triggers:** para automatizar procesos de edición e inserción.
- **Definición de tipos (types):** para parametrizar los tipos de datos que devuelven las funciones.
- **Uso de pgcrypto:** para encriptar información sensible como contraseñas.
- **Uso de pgjwt:** para generar el token de acceso.

Roles

Para poder definir qué usuarios puede acceder a un dato definido se utilizan los roles en la base de datos. Para este proyecto son necesarios 3 roles

- **Usuario anónimo**: el usuario anónimo es el usuario sin iniciar sesión y solo tiene acceso a las funciones de creación de usuario e inicio de sesión.
- **Autenticado:** el usuario autenticado como su nombre lo dice es el usuario que se ha autenticado en la aplicación y tiene acceso a todos sus datos y los planes compartidos.
- **Administrador:** tiene todo el acceso a toda la información.

Schemas

Para una mejor gestión de los datos se utilizaron dos schemas, uno para la gestión de usuarios y otro para los datos de los planes

Diagrama base de datos

Como este proyecto trata de la elaboración de planes estratégicos, en las siguientes descripciones y diagramas representamos gráficamente la base de datos. cuya característica es que está centrada en la tabla de planes y en las tablas de los componentes que integran esos planes, ambas se describen de manera escalonada.

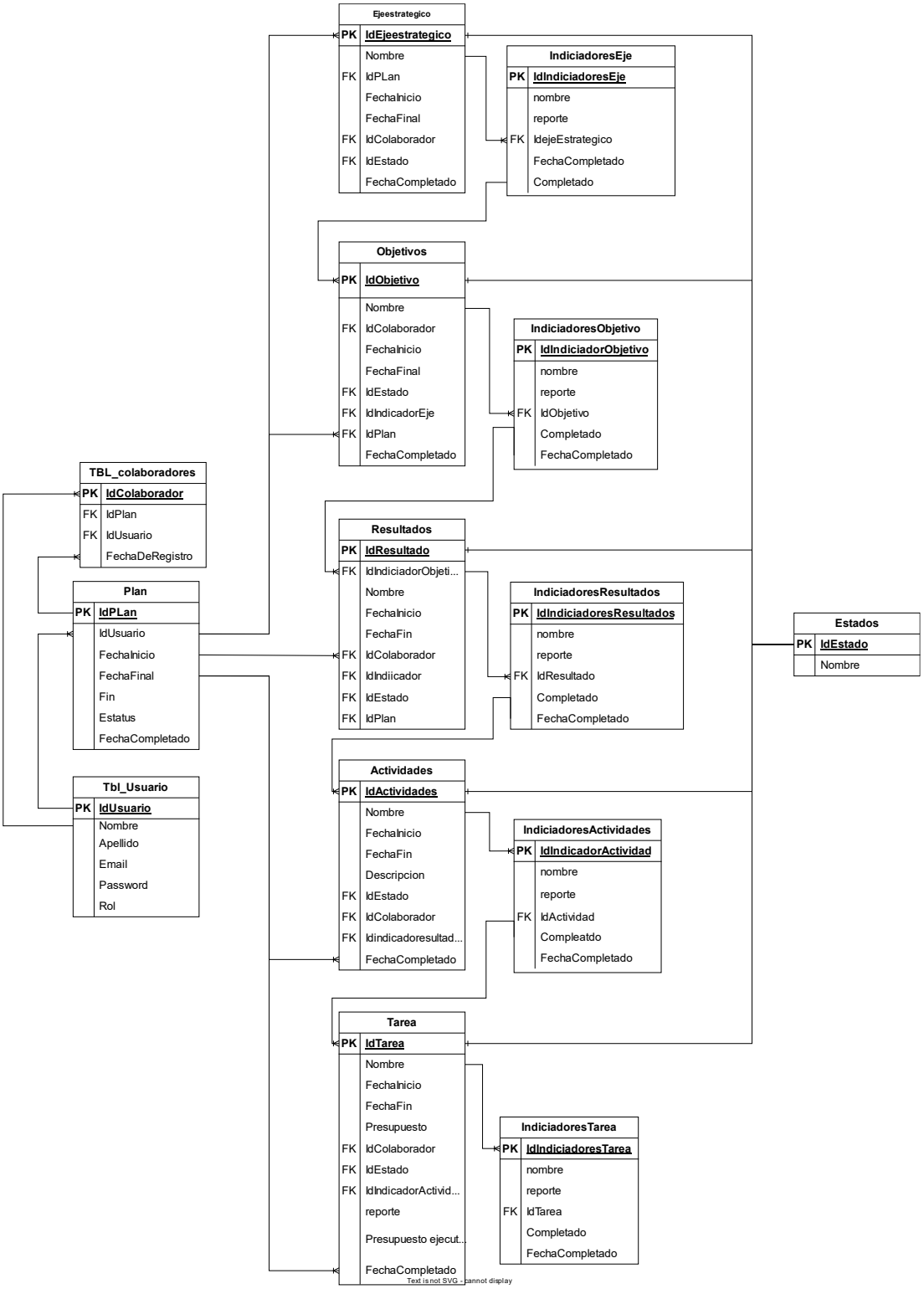


Ilustración 4. Diagrama entidad relación de base de datos

Tablas principales

En este acápite del proyecto se describen los diferentes componentes que lo integran, incluyendo las tablas de entidades que componen el proyecto.

Plan

La tabla de planes es la tabla central de la base de datos en donde se guardan los datos de los planes. En su descripción incluye; usuario creador, fecha inicio, fecha final, su fin y su estado.

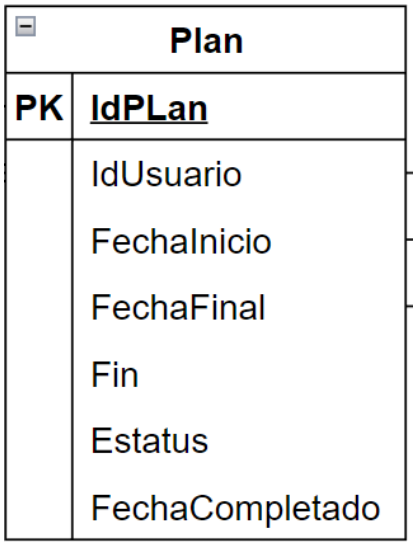


Ilustración 5. Entidad de plan

Ejes estratégicos

Los ejes estratégicos son el primer componente de un plan, ellos determinan los demás componentes del proyecto. Las tablas que lo integran almacenan la información de los ejes, así como los indicadores que permiten medir su avance y evolución.

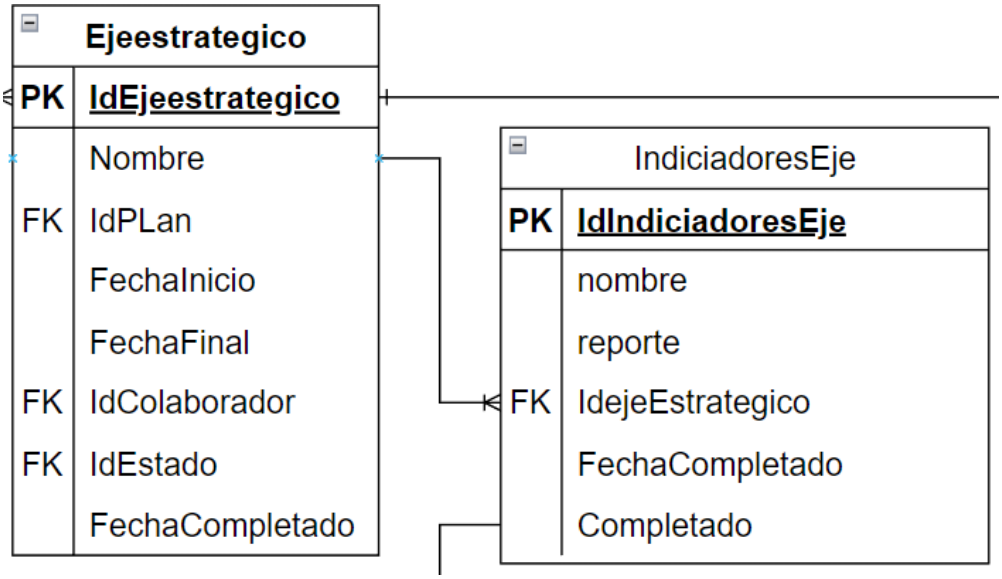


Ilustración 6. Entidad de eje estratégico e indicador

Objetivos

Los objetivos son los resultados específicos que se pretenden alcanzar dentro de cada eje estratégico, estás tabla tiene una relación de dependencia con la de los ejes estratégicos y permiten guardar los datos de los objetivos y sus indicadores, incluyendo las características de éstos como; estado, nombre, fecha de inicio, fecha fin, plan al que pertenece y la fecha cuando se completó.

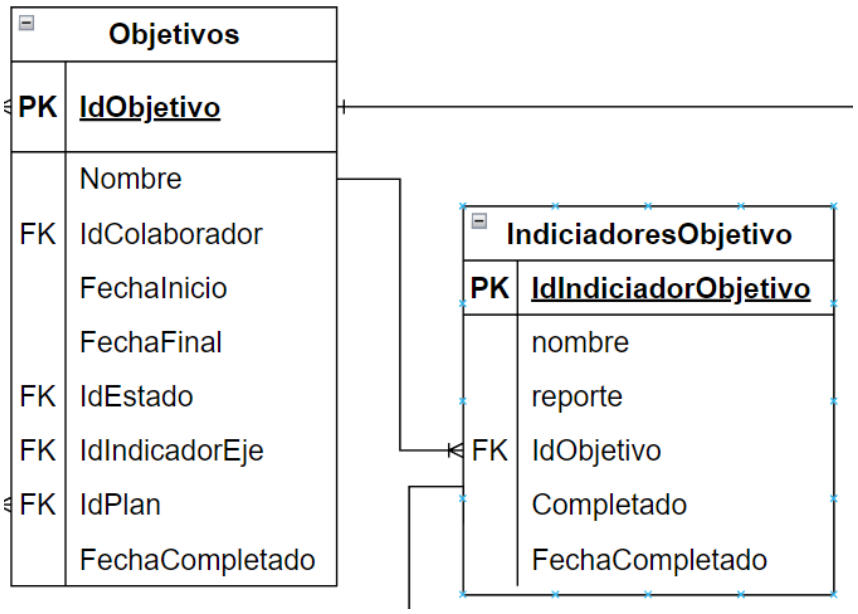


Ilustración 7. Entidad de objetivos e indicador

Resultados

Los resultados permiten conocer el impacto logrado al alcanzar los objetivos establecidos. Este conjunto de tablas guarda la información de los resultados, los indicadores y sus características; estado, nombre, fecha de inicio, fecha fin, plan al que pertenece y la fecha cuando se completó.

Ilustración 8. Entidad de resultados e indicador

Actividades y tareas

Las actividades y tareas representan las acciones necesarias para cumplir con los objetivos. Las tablas que lo describen guardan la información respectiva de estos componentes y las características de sus indicadores (estado, nombre, fecha de inicio, fecha fin, plan al que pertenece, la fecha cuando se completó, las tareas y el coste de estas).

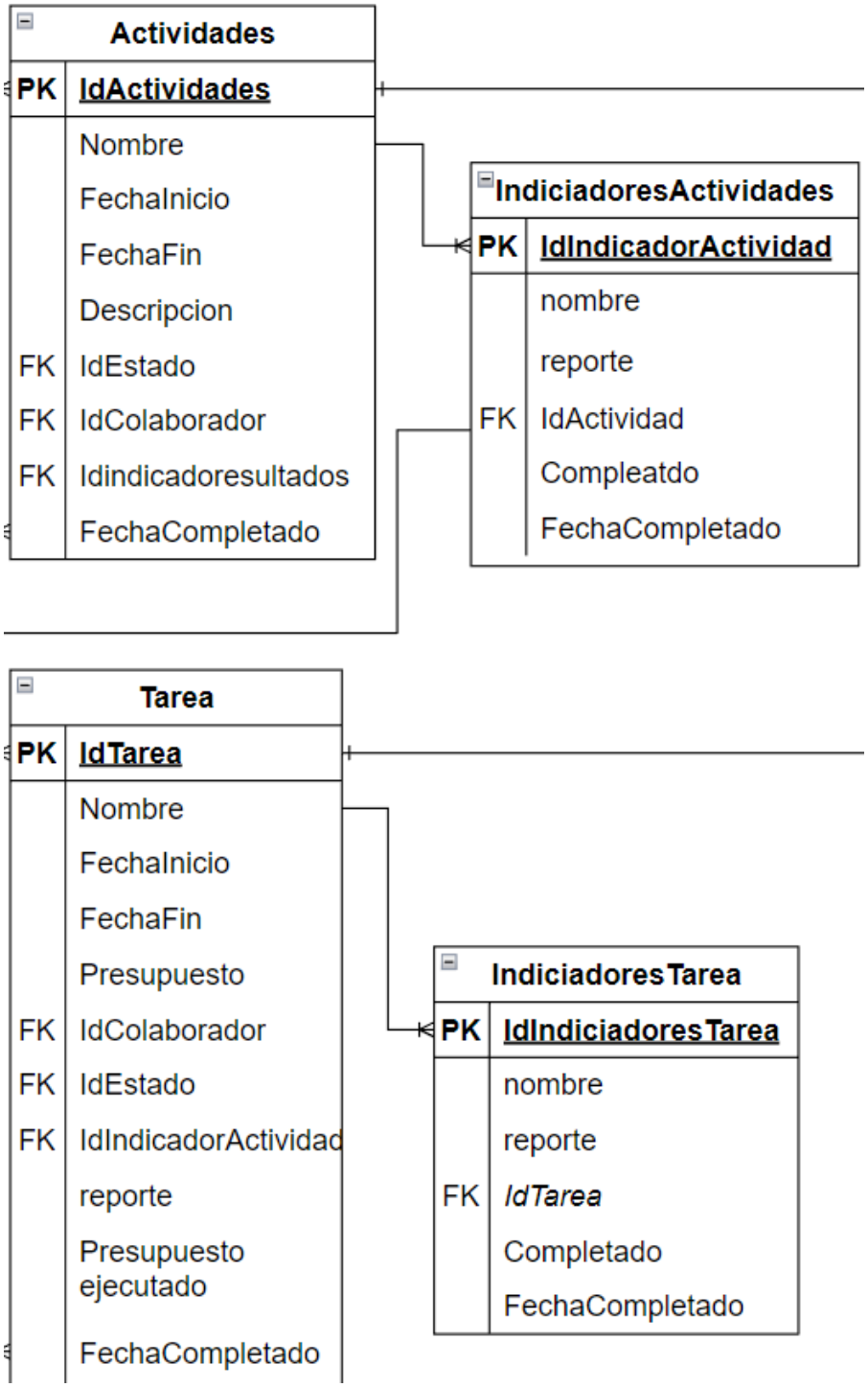


Ilustración 9. Entidad de actividades y tareas e indicadores

Usuarios
La tabla de usuarios es una de las más importantes para el proyecto ya que guarda la información de los autores de los planes y sus colaboradores

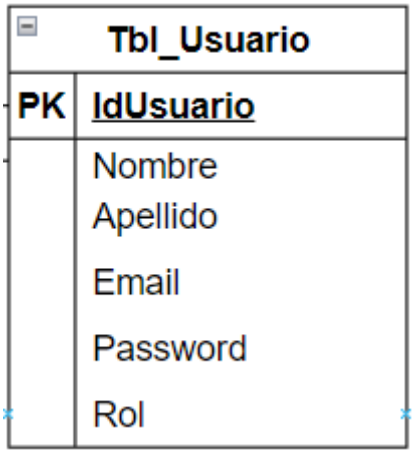


Ilustración 10. Entidad de usuarios

Funciones

Además de las tablas se deben crear un conjunto de funciones. PostgREST tiene la capacidad de ejecutar funciones y utilizar el resultado de éste si existe un tipo con ese formato. Esta aplicación necesitó varias de estas para su correcto funcionamiento como serían las que envían la información de la progresión de los componentes, la búsqueda de usuarios, siendo las más importantes las siguientes:

- **Registro de usuario**: los datos de los usuarios se encuentran en un schema diferente, ya que la herramienta solo se conecta a un schema a la vez, fue necesario crear funciones personalizadas para la gestión de los usuarios.

```
CREATE OR REPLACE FUNCTION public.insert_user(
    p_email text,
    p_pass text,
    p_nombre text,
    p_apellido text)
    RETURNS void
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$
BEGIN
  INSERT INTO basic_auth.users (email, pass, role, nombre, apellido)
  VALUES (p_email, p_pass,  'authenticator', p_nombre, p_apellido);
END;
$BODY$;

ALTER FUNCTION public.insert_user(text, text, text, text)
    OWNER TO postgres;
```

Ilustración 11. función de registro de usuarios

- **Obtener plan**: ya que el plan tiene varias llaves foráneas es necesario crear una función para que en vez de recibir los identificadores éstos sean sustituidos por el valor de éstas

```
CREATE OR REPLACE FUNCTION public.get_planes_por_usuario(
    user_id integer)
    RETURNS SETOF planestype
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
DECLARE
  rec public.planesType;
BEGIN
  FOR rec IN
    SELECT
      p.idplan as id,
      p.nombre as nombre_plan,
      p.fechainicio as fecha_inicio,
      p.fechafinal as fecha_final,
      e.nombre as estado,
      p."FechaCompletado" as fechaFinalizado
    FROM plan p
    INNER JOIN estados e ON p.idestado = e.idestado
    WHERE p.idusuario = user_id
  LOOP
    RETURN NEXT rec;
  END LOOP;
  RETURN;
END;
$BODY$;
```

Ilustración 12. función para obtener un plan

- **Inicio de sesión**: Al igual que la función de registro de usuarios la función de inicio de sesión utiliza datos del schema de usuarios, además de que la función debe enviar el token de autorización.

```
CREATE OR REPLACE FUNCTION public.login(
    email text,
    pass text)
    RETURNS basic_auth.jwt_token
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE SECURITY DEFINER PARALLEL UNSAFE
AS $BODY$
declare
  _role name;
  _id integer;

  result basic_auth.jwt_token;
begin
  -- check email and password
  select basic_auth.user_role(email, pass) into _role;
  if _role is null then
    raise invalid_password using message = 'invalid user or password';
  end if;

 SELECT idusuario INTO _id FROM basic_auth.users WHERE basic_auth.users.email = login.email;

  select sign(
      row_to_json(r), 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890ConElLapiNo12!'
    ) as token
    from (
      select _role as role, login.email as email, _id as id,
         extract(epoch from now())::integer + 60*60 as exp
    ) r
    into result;
  return result;
end;
$BODY$;
```

Ilustración 13. función de inicio de sesión

Types

Los types son parecido a los modelos de datos y sirvieron para parametrizar los resultados de las funciones. Este proyecto utiliza varios de estos siendo lo más importantes los siguientes:

- Planes: El type de planes contiene: el nombre del plan, las fechas plazos, el estado y la fecha que se finalizó

```
CREATE TYPE public.planestype AS
(
    id integer,
    nombre_plan text,
    fecha_inicio date,
    fecha_final date,
    estado text,
    "fechaFinalizado" date
);

ALTER TYPE public.planestype
    OWNER TO postgres;
```

Ilustración 14. type de planes

- Token: Este es el type que retorna la función de inicio de sesión

```
CREATE TYPE public.jwt_token AS
(
    token text
);

ALTER TYPE public.jwt_token
    OWNER TO postgres;
```

Ilustración 15. type de token

Triggers

Para este proyecto fue necesario crear varios trigger para automatizar varias funciones de inserción y edición. La paliación tiene 2 triggers principales:

- Encriptación de contraseña: al momento del registro es necesario encriptar las contraseñas utilizando pgcrypto.

```
CREATE OR REPLACE FUNCTION basic_auth.encrypt_pass()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
begin
  if tg_op = 'INSERT' or new.pass <> old.pass then
    new.pass = crypt(new.pass, gen_salt('bf'));
  end if;
  return new;
end
$BODY$;

ALTER FUNCTION basic_auth.encrypt_pass()
    OWNER TO postgres;
```

Ilustración 16. Script de creación de trigger para encriptación de contraseña

- Comprobación de roles: para comprobar si un rol insertado en un usuario existe y no tener problemas con el acceso a los datos. Utilizamos este trigger

```
CREATE OR REPLACE FUNCTION basic_auth.check_role_exists()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
begin
  if not exists (select 1 from pg_roles as r where r.rolname = new.role) then
    raise foreign_key_violation using message =
      'unknown database role: ' || new.role;
    return null;
  end if;
  return new;
end
$BODY$;

ALTER FUNCTION basic_auth.check_role_exists()
    OWNER TO postgres;
```

Ilustración 17. Script de creación de trigger para comprobar el si el rol existe

4.1.2 PostgREST

PostgREST desempeña un papel fundamental en este proyecto al asumir la responsabilidad del backend, y por lo que su función principal es gestionar y proporcionar acceso a los datos que requiere la aplicación móvil de manera eficiente y segura.

Es un servidor web independiente que convierte su base de datos PostgreSQL directamente en una API RESTful. Las restricciones estructurales y los permisos en la base de datos determinan los puntos finales y las operaciones de la API [11].

Esta herramienta se destaca por su capacidad para simplificar la exposición de datos almacenados en una base de datos PostgreSQL mediante la creación automática de una API RESTful completa.

Instalación

Para la obtención de la herramienta se debe ir al GitHub oficial de PostgREST https://github.com/PostgREST/postgrest/releases/latest , e ir al apartado de assets y descargar la versión del sistema operativo al cual se usará.

Ilustración 18. Listado de versiones de PostgREST

Configuración

Para la implementación de esta herramienta se debe crear un archivo de configuración definiendo varios parámetros básicos:

- db-uri
  Es la url de conexión de la base de datos, con el usuario, seguido de la contraseña y el puerto donde se aloja la base de datos.
- db-schema
  Este es el esquema en el cual se conectará la herramienta PostgREST.
- db-anon-role
  Rol que se le asignará al usuario si no está logueado.
- jwt-secret
  String utilizado para generar el token de autorización.

```
db-uri = "postgres://postgres:root@127.0.0.1:5432/postgres"

db-schema = "public"
db-anon-role = "anon"


jwt-secret = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890ConElLapiNo12!"
```

Ilustración 19. Ejemplo configuración básica de PostgREST

Peticiones HTTP

Las peticiones con la herramienta PostgREST tienen una estructura predefinida dependiendo lo que se desea ejecutar, ya sea la interacción con una tabla de la base de datos o la ejecución de funciones.

URL Base: La URL base es la dirección raíz del servidor donde está alojada la API de PostgREST.

Por ejemplo:

```
http://localhost:3000/resultados
```

Ilustración 20. Ejemplo url base de PostgREST

Ruta de tabla: La ruta del recurso es la parte específica de la URL que identifica el recurso o la tabla en la base de datos que deseas consultar o modificar. Por lo general se coloca después de la URL base y puede incluir sub-rutas si es necesario.

Por ejemplo:

```
http://localhost:3000/resultados?idresultados=eq.1
```

Ilustración 21. Ejemplo url para obtención de datos de tabla

Funciones: Para ejecutar una función, en la URL se debe colocar las letras rcp, luego el nombre de la función y por último el parámetro si se solicita.

```
http://localhost:3000/rpc/get_resultados_show_by_idplan?id_plan=1
```

Ilustración 22. ejemplo url ejecución de función

4.1.3 Ionic

Ionic fue utilizado para la creación de las interfaces de la aplicación. Este framework fue escogido para la creación de las interfaces de la aplicación, no solo por su sencillez de desarrollo, sino también por su capacidad multiplataforma, lo que permitió aprovechar el mismo código base para alcanzar audiencias tanto en dispositivos Android como en ordenadores, reduciendo significativamente el esfuerzo de desarrollo.

Arquitectura

La aplicación está desarrollada con una arquitectura basada en componentes de Ionic, se basa en un enfoque modular y utiliza el conjunto de herramientas de Ionic y Angular para proporcionar una estructura sólida y escalable. Está diseñada para mantener un código limpio, promover la reutilización de componentes y garantizar un rendimiento óptimo.

```
<ion-card>

  <ion-card-content>
    <ion-row size="12">

      <ion-col size="12">
        <div>
          <h1>{{ titulo }}</h1></div>
        <div class="container">
          <p class="small-text" [ngClass]="{
            'estatus-verde': estado === 'completado',
            'estatus-rojo': estado === 'sin completar'
          }">{{ estado }}</p>
        </div>
      </ion-col>

    </ion-row>
  </ion-card-content>
</ion-card>
```

Ilustración 23. Ejemplo de componente utilizado en el desarrollo

La aplicación realiza solicitudes HTTP para interactuar con PostgREST. Utilizando los servicios RESTful para enviar y recibir datos, esto garantiza una comunicación eficiente y segura entre la aplicación y PostgREST.

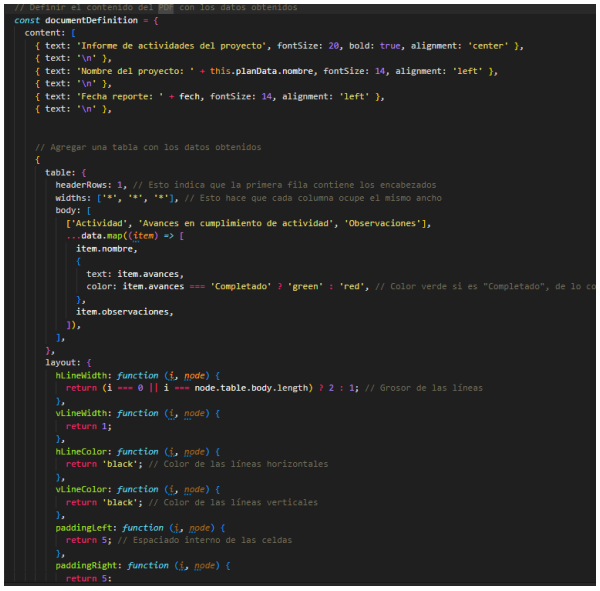
Ilustración 24. Arquitectura de la aplicación

Para hacer las peticiones HTTP se utilizó el módulo HttpClient de la librería common de angular, además de utilizar HttpHeaders de la misma librería para enviar el token de autorización

```
const url = `${apiUrl}plan?idplan=eq.${this.id}`;
const token = sessionStorage.getItem('token');
const headers = new HttpHeaders({
  'Content-Type': 'application/json',
  Authorization: `Bearer ${token}`
});

if(getHeader()!= null){
  this.httpClient.get(url, {headers} ).subscribe(
    (response: any) => {
      this.planData = response[0]; // Guarda los datos en la variable
      console.log('Datos del plan:', this.planData);

    },
    error => {
      this.presentAlert('Error',error);
        logout(this.router);

      // Manejo de errores
    }
  );
}
```

Ilustración 25. ejemplo petición HTTP en Ionic

Otra librería utilizada en este proyecto es PDFMake que se utiliza para la generación de pdf para los informes de los planes.



```
// Definir el contenido del PDF con los datos obtenidos
const documentDefinition = {
  content: [
    { text: 'Informe de actividades del proyecto', fontSize: 20, bold: true, alignment: 'center' },
    { text: '\n' },
    { text: 'Nombre del proyecto: ' + this.planData.nombre, fontSize: 14, alignment: 'left' },
    { text: '\n' },
    { text: 'Fecha reporte: ' + fech, fontSize: 14, alignment: 'left' },
    { text: '\n' },


    // Agregar una tabla con los datos obtenidos
    {
      table: {
        headerRows: 1, // Esto indica que la primera fila contiene los encabezados
        widths: ['*', '*', '*'], // Esto hace que cada columna ocupe el mismo ancho
        body: [
          ['Actividad', 'Avances en cumplimiento de actividad', 'Observaciones'],
          ...data.map((item) => [
            item.nombre,
            {
              text: item.avances,
              color: item.avances === 'Completado' ? 'green' : 'red', // Color verde si es "Completado", de lo co
            },
            item.observaciones,
          ]),
        ],
      },
      layout: {
        hLineWidth: function (i, node) {
          return (i === 0 || i === node.table.body.length) ? 2 : 1; // Grosor de las líneas
        },
        vLineWidth: function (i, node) {
          return 1;
        },
        hLineColor: function (i, node) {
          return 'black'; // Color de las líneas horizontales
        },
        vLineColor: function (i, node) {
          return 'black'; // Color de las líneas verticales
        },
        paddingLeft: function (i, node) {
          return 5; // Espaciado interno de las celdas
        },
        paddingRight: function (i, node) {
          return 5;
```

Ilustración 26. ejemplo implementación pdfmake

Para mas detalles sobre la instalación y configuración de PostgREST y el frontend, es necesario ir al anexo 2 manual de despliegue

4.2 Funcionamiento del software

El software incluye diferentes niveles y componentes que estructuran un proyecto, para cada uno de los componentes se integran indicadores que permiten medir el avance en su ejecución, a los niveles del proyecto y a sus componentes se le definen los responsables, acceso y roles. Estos colaboradores reportan los avances de los indicadores y componentes mediante un conjunto de formularios, con la información una vez acumulada nos permitirá tener un informe final en la ejecución programática del proyecto.

4.2.1 Niveles

La planificación se estructura tomando en cuenta diferentes estados en el desarrollo de un producto o servicio. Estos estados se vinculan a niveles de avances en la elaboración, y en cada uno de ellos intervienen procesos, recursos y actores claves, los cuales hay que organizar para mejorar su operatividad y eficacia.

Una de las herramientas de importancia en la planificación es el organigrama o estructura organizacional, pues a cada una de ellas le corresponde un nivel que lo vincula al proceso.
Para el caso de este proyecto los recursos han sido organizados en varios niveles, que han facilitado el desarrollo de la herramienta.

Los niveles establecidos según veremos en la siguiente descripción son, el estratégico, el programático y el operativo.

Nivel estratégico

En la estructura de este plan el nivel estratégico es quien define las líneas de trabajo, establece las metas según los resultados planteados. Bajo la responsabilidad de este nivel está el logro final del proyecto, por ello debe tener una panorámica completa de cómo se avanza en los diferentes niveles y como cada uno de ellos va agregando sus procesos para el obtener el resultado final.

Componente ejes estratégicos:

Descripción: los ejes estratégicos representan las grandes líneas de acción del proyecto.
Indicadores clave: definición de indicadores medibles para evaluar el cumplimiento de cada eje estratégico.

Nivel programático

Acápite de la estructura en el que se programan las acciones a ser ejecutadas, este nivel está vinculado de manera estrecha al logro de objetivos organizacionales, reporta al nivel estratégico los diferentes avances en la consecución de estos. De sus reportes el nivel estratégico toma decisiones que marcan el rumbo de la ejecución.

Componente objetivos:

Descripción: los objetivos son los resultados específicos que se pretenden alcanzar dentro de cada eje estratégico.
Indicadores clave: establecimiento de indicadores que permitan medir el progreso hacia cada objetivo.

Componente resultados:

Descripción: los resultados reflejan el impacto logrado al alcanzar los objetivos establecidos.
Indicadores clave: definición de indicadores para medir la consecución de los resultados esperados.

Nivel operativo

Este nivel se corresponde con la parte de la estructura en la que reposan la implementación de las acciones programadas, reporta al nivel programático el desarrollo de las diferentes actividades y tareas programadas, así como sus niveles de avance.

Componente actividades y tareas:

Descripción: Las actividades y tareas representan las acciones específicas necesarias para cumplir con los objetivos y, en última instancia, alcanzar los resultados.
Indicadores clave: establecimiento de indicadores de progreso para cada actividad y tarea.

4.2.2 Informes y análisis

La aplicación generará informes conceptuales y gráficos para facilitar el análisis del progreso del proyecto:

A. Cumplimiento de Tareas y Actividades:

Reportes que mostrarán el avance logrado en el cumplimiento de los indicadores de las tareas y actividades en los tiempos establecidos.

B. Cumplimiento de resultados y objetivos:

Informes que permitan observar el cumplimiento de los indicadores de resultados y la forma cómo éstos van contribuyendo a los indicadores de los objetivos.

C. Alcance de los ejes estratégicos:

Informe que permite observar el avance en el alcance de los ejes estratégicos.

D. Logro final del proyecto:

Informe final en el que se refleja el reporte del cumplimiento de los diferentes componentes.

4.2.3 Interfaces principales

- Manejo de planes

Este panel de control se erige como un elemento central para proporcionar a los usuarios una visión panorámica y en tiempo real del estado de sus planes y componentes. El Dashboard se destaca por su capacidad para representar de manera gráfica e intuitiva la información esencial que los usuarios necesitan para tomar decisiones informadas y realizar un seguimiento efectivo de sus proyectos.

Ilustración 27 Dashboard de planes

- Manejo de colaboradores

Dentro de la aplicación, se encuentra una funcionalidad crucial que se presenta en forma de una tabla de colaboradores activos. Esta tabla sirve como un componente central para la gestión de los colaboradores que participan en los planes y proyectos. Su diseño y funcionalidad están diseñados para proporcionar a los usuarios un medio eficiente y efectivo para supervisar y administrar a su equipo de colaboradores de manera conveniente

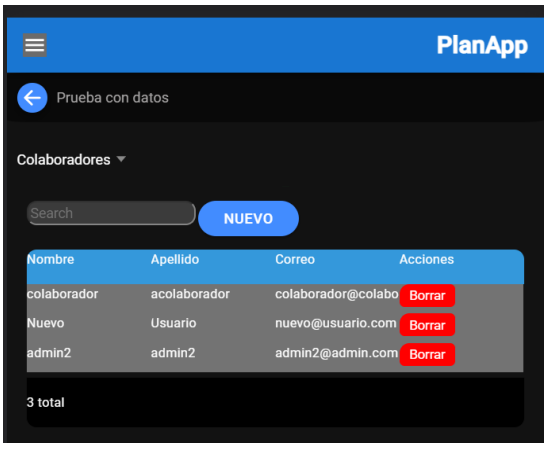


Ilustración 28 Tabla de colaboradores

- Generación de PDF

La aplicación cuenta con una funcionalidad esencial que permite la generación de informes en formato PDF. La generación de informes se encuentra estrechamente vinculada a la estructura jerárquica de los planes y sus componentes.

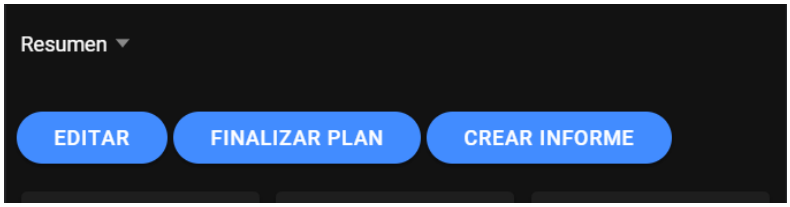
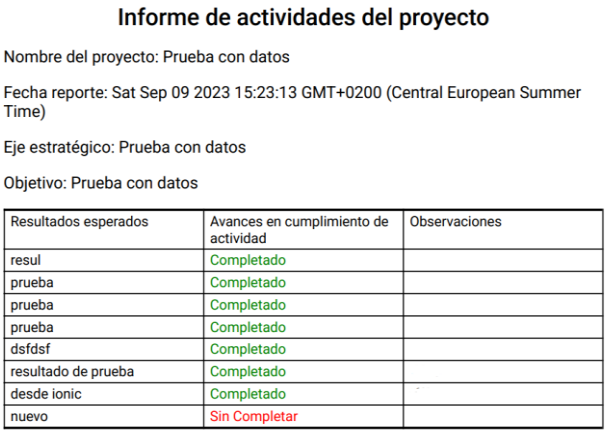


Ilustración 29 menú de opciones de informe



Informe de actividades del proyecto

Nombre del proyecto: Prueba con datos

Fecha reporte: Sat Sep 09 2023 15:23:13 GMT+0200 (Central European Summer Time)

Eje estratégico: Prueba con datos

Objetivo: Prueba con datos

Resultados esperados	Avances en cumplimiento de actividad	Observaciones
resul	Completado	
prueba	Completado	
prueba	Completado	
prueba	Completado	
dsfdsf	Completado	
resultado de prueba	Completado	
desde ionic	Completado	
nuevo	Sin Completar	

Ilustración 30 Ejemplo de informe

5. Conclusiones

El diseño y elaboración de este proyecto tuvo como objetivo principal verificar la viabilidad para desarrollar una aplicación móvil sin la necesidad de crear un backend tradicional, para lo cual utilizamos la herramienta PostgREST y el framework Ionic.

Aplicando una secuencia de pasos pudimos desarrollar una aplicación funcional, mediante el uso de un modelo de base de datos PostgreSQL, y de la herramienta PostgREST que nos permitió convertir la base de datos en un apiREST funcional.

Desarrollamos una aplicación móvil para la gestión y administración de planes estratégicos, esto nos permitió confirmar que PostgREST es una herramienta altamente accesible y conveniente, ideal para usuarios de SQL y administración de bases de datos. En el proceso verificamos que esta herramienta permite una gestión eficaz de roles y permisos al asignar tablas específicas a roles de base de datos.

PostgREST probó ser una herramienta robusta, pues con la simple creación de una base de datos, se elimina la necesidad de desarrollar los cuatro tipos de solicitudes básicas (CRUD) tradicionalmente requeridos en el desarrollo de aplicaciones.

Mejoró nuestra eficacia pues nos permitió enfocarnos principalmente en la implementación de las funciones que se encargaban de mostrar datos y en la configuración de los desencadenadores (triggers) para las operaciones de edición y adición de datos.

Adicional a esto, como las funciones encargadas de insertar, editar y eliminar datos ya estaban disponibles de manera predeterminada en la herramienta y no requerían modificaciones, pudimos acelerar significativamente el proceso de creación de la aplicación y concentrarnos en aspectos más específicos y personalizados, como la presentación de datos y la experiencia del usuario. Por lo cual, la herramienta simplificó la fase de desarrollo y permitió lograr resultados más rápidos y precisos.

La combinación del uso de la herramienta PostgREST, junto con el framework Ionic, resultó ser efectiva, pues se logró realizar una aplicación móvil y página web sin problemas y con mucha facilidad al momento de desarrollarla. Esta sinergia entre PostgREST y Ionic permitió aprovechar la generación automática de Apis RESTful a partir de la base de datos PostgreSQL, acelerando significativamente el proceso de desarrollo y garantizando una gestión eficiente de los datos tanto en la aplicación móvil como en la versión web.

Es importante destacar que no es una solución perfecta en todos los casos, algunas limitaciones fueron evidentes durante el proceso de desarrollo. En aplicaciones con numerosas relaciones entre tablas, puede volverse más difícil al momento de desarrollar, ya que se requiere la creación de modelos y funciones personalizadas para cada respuesta, un proceso que es más directo en un backend convencional. Además, en entornos de este tipo, herramientas como los ORM (Object-Relational Mapping) facilitan la gestión de modelos, lo que no está disponible de manera nativa en PostgREST.

6. Trabajo futuro

Algunas limitaciones actuales de este proyecto se deben superar en próximas versiones, cuestión de hacerlo mucho más completo, versátil y amigable al usuario. En el desarrollo de la aplicación se dejan los elementos para vincular lo realizado a los procesos de mejora pretendidos para el futuro. Presentamos a continuación las acciones de mejoras a ser implementadas en un futuro próximo.

6.1 Soporte para gráficos

El Dashboard de la aplicación presenta varias barras de progreso para identificar de manera grafica el estado de un plan. Pero para el futuro será necesario la utilización de otro tipo de gráficos. Tanto para el dashboard principal como también para el informe. Mediante la implementación de chart.js para la parte del dashboard y pdfmake-graphs para los informes.

6.2 Soporte para indicadores cuantitativos

En el estado actual, los indicadores se registran simplemente como completados o no completados. Sin embargo, para futuras mejoras, se plantea la necesidad de permitir que los indicadores se registren no solo como completados o no completados, sino también en términos de una cantidad específica o un estado de completitud parcial. Esto permitirá una medición más precisa y detallada de los indicadores, brindando una visión más completa de su progreso y rendimiento.

6.3 Reporte costo/efectividad del proyecto:

Para una nueva versión de la aplicación se plantea el realizar un nuevo informe que haga el análisis que evaluará el cumplimiento del proyecto en relación al coste planificado.

7. Coste del proyecto

Para este proyecto se utilizarán tres tipos de recursos:

a. Mano de obra especializada
b. Equipos y servicios
c. Software

Para el cálculo de los costos de la mano de obra especializada, se identifican los diferentes recursos a ser utilizados y sobre la base de tiempo de trabajo dedicado, para el cálculo de los costos de equipos se calculó el valor de su tiempo de uso y para los servicios, se utiliza igual metodología.
En el caso del Software, éste es de código abierto, por lo tanto, no tiene costos.

a. Coste de mano de obra

Para el cálculo de los costos de la mano de obra, se describe el tipo de recursos humanos utilizados para el desarrollo de la aplicación, teniendo entre otros; el uso de dos desarrolladores cuya funciones son el desarrollo el Frontend, la configuración del PostgREST; y la creación de la base de datos, también utilizamos un personal para testing, éste debe comprobar que todos los aspecto de la aplicación funcionen y una persona que funja como gerente, que es quien debe gestionar y supervisar los diferentes procesos en el desarrollo y prueba de la aplicación. Para todos los casos el cálculo de costos se realiza tomando en cuenta el tiempo de dedicación.

Concepto	Cantidad	Horas	Coste/Hora	Coste total
Desarrolladores	2	160	10.10	3,232
Tester	1	80	10.10	808
Gerente	1	160	15	2,400
Total	**4**	400	35.20	**6,440**

Tabla 1: Resultados de costes de mano de obra.

b. Coste de materiales

Como recursos materiales se usarán equipos de computadoras y servicios de electricidad e internet, cuyos costes se calcularán tomando en cuenta su tiempo de uso.

Costes de Hardware

Descripción	Número	Coste	Total
Ordenador i7 16gb ram 500gb	3	750	750
Total	3	750	**2250**

Tabla 2: Resultados de costes de Hardware.

Costes de Servicios

Descripción	Horas	Coste	Total
Uso de electricidad	400	0.30	120
Uso de internet	400	0.02	80
Total			**200**

Tabla 3: Resultados de costes de servicios.

Descripción	Total
Costes de Hardware	2250
Costes de servicios	200
Total	**2450**

Tabla 4: Resultados de costes totales de materiales.

c. Gastos generales

Descripción	Total
Costes de mano de obra	6640
Costes de materiales	2540
Gastos generales	**9180**

Tabla 5: Resultados de gastos generales.

8. Bibliografía

[1] C. Collado, «11 aplicaciones gratuitas y muy útiles para gestionar proyectos», *Andro4all*, 4 de mayo de 2022. https://www.lavanguardia.com/andro4all/aplicaciones-gratis/mejores-apps-gestionar-proyectos (accedido 11 de septiembre de 2023).

[2] «Architectural styles and the design of network-based software architectures | Guidebooks». https://dl.acm.org/doi/book/10.5555/932295 (accedido 11 de septiembre de 2023).

[3] «What is REST - REST API Tutorial». https://restfulapi.net/ (accedido 11 de septiembre de 2023).

[4] «What is REST», *REST API Tutorial*, 7 de abril de 2022. https://restfulapi.net/ (accedido 11 de septiembre de 2023).

[5] «Diferencias entre REST y SOAP». https://www.redhat.com/es/topics/integration/whats-the-difference-between-soap-rest (accedido 11 de septiembre de 2023).

[6] «El mundo de las API», *Innovation*, 1 de octubre de 2020. https://www.innovaxians.es/el-mundo-de-las-api/ (accedido 11 de septiembre de 2023).

[7] «Frontend Definition». https://techterms.com/definition/frontend (accedido 11 de septiembre de 2023).

[8]C. Valdeolmillos, «Las mejores bases de datos open source y sus ventajas frente a las propietarias», *MuyComputerPRO*, 15 de octubre de 2022. https://www.muycomputerpro.com/2022/10/15/mejores-bases-datos-open-source-ventajas-propietarias (accedido 11 de septiembre de 2023).

[9]P. G. D. Group, «PostgreSQL», *PostgreSQL*, 12 de septiembre de 2023. https://www.postgresql.org/ (accedido 12 de septiembre de 2023).

[10] «Nimbus Tech Stack (2019) | Paul Copplestone». https://paul.copplest.one/blog/nimbus-tech-2019-04.html#tech-stack (accedido 11 de septiembre de 2023).

[11] «PostgREST Documentation — PostgREST 11.2.0 documentation». https://postgrest.org/en/latest/index.html (accedido 11 de septiembre de 2023).

[12] «¿Qué es una API y que debe de poseer para que sea funcional? - Hostingato». https://hostingato.com/blog/57-que-es-una-api-y-que-debe-de-poseer-para-que-sea-funcional.html (accedido 11 de septiembre de 2023).

[13] «¿Qué es una API? 5 ventajas de una interfaz de programación», *Zendesk MX*. https://www.zendesk.com.mx/blog/que-es-una-api/ (accedido 11 de septiembre de 2023).

[14] «Introduction to Ionic | Ionic Documentation», Ionic Framework Docs. https://ionicframework.com/docs (accedido 12 de septiembre de 2023).

[15] «Websocket vs REST - Coconauts». https://coconauts.net/blog/2017/11/20/websocket-vs-rest/ (accedido 13 de septiembre de 2023).

[16] «¿Qué es una API? - Explicación de interfaz de programación de aplicaciones - AWS», *Amazon Web Services, Inc.* https://aws.amazon.com/es/what-is/api/ (accedido 13 de septiembre de 2023).

[17] «El papel de las API en el desarrollo web | AppMaster». https://appmaster.io/es/blog/apis-en-el-desarrollo-web (accedido 14 de septiembre de 2023).

[18] «REST frente a SOAP: creación de aplicaciones modernas», *Auth0*. https://auth0.com/es/learn/rest-vs-soap (accedido 14 de septiembre de 2023).

[19] «IBM Documentation», 5 de marzo de 2021. https://www.ibm.com/docs/es/rsas/7.5.0?topic=standards-soap (accedido 14 de septiembre de 2023).
[20] «Qué es Trello: descubre sus funciones, usos y todo lo que ofrece | Trello». https://trello.com/es/tour (accedido 14 de septiembre de 2023).

[21] «Qué es Notion | Definición, ventajas y características», 6 de abril de 2023. https://www.arimetrics.com/glosario-digital/notion (accedido 14 de septiembre de 2023).

[22] «What is Coda? The all-in-one platform for teams. - Coda», *Coda | Everything evolves, the evolution of documents.* https://coda.io/product (accedido 14 de septiembre de 2023).

[23] «¿Qué es una API REST? | IBM». https://www.ibm.com/es-es/topics/rest-apis (accedido 15 de septiembre de 2023).

[24] B. R, «Introducing PostgREST, a REST API for any PostgreSQL database written in Haskell», *Packt Hub*, 4 de noviembre de 2019. https://hub.packtpub.com/introducing-postgrest-a-rest-api-for-any-postgresql-database-written-in-haskell/ (accedido 15 de septiembre de 2023).

[25]M. D. Agency, «Todo sobre API REST: Qué es, características, usos y ventajas», *Epitech Spain*, 13 de octubre de 2021. https://www.epitech-it.es/api-rest/ (accedido 15 de septiembre de 2023).

[26]Darkcrizt, «PostgREST, una API REST para cualquier base de datos PostgreSQL escrita en Haskell», *Desde Linux*, 7 de noviembre de 2019. https://blog.desdelinux.net/postgrest-una-api-rest-para-cualquier-base-de-datos-postgresql-escrita-en-haskell/ (accedido 15 de septiembre de 2023).

9. Anexo 1: Manual de Usuario

9.1 Introducción

Este Manual de Usuario proporciona una visión completa de todas las funciones y características de nuestra aplicación de gestión de proyectos. Aquí, se detalla cada sección y se explica cómo utilizarlas para optimizar los procesos de planificación y gestión de proyectos.
La aplicación está diseñada para facilitar la planificación, supervisión y control de proyectos de manera efectiva y organizada. Desde el inicio de sesión hasta la gestión de tareas, colaboradores, ejes estratégicos, objetivos, resultados, actividades y tareas, este manual ofrece instrucciones paso a paso para aprovechar al máximo todas las funcionalidades disponibles.

9.2 Interfaces

Se describen a continuación las diferentes interfaces de la aplicación.

9.2.1 Inicio de sesión

Esta es la primera pantalla en ella los usuarios pueden iniciar su sesión para luego comprobar sus planes y responsabilidades

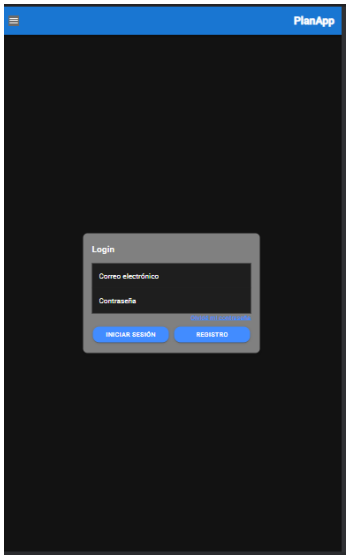


Ilustración 31 Interfaz de inicio de sesión

9.2.2 Registro

En esta interfaz se introducen los datos para crear un usuario y utilizar la aplicación

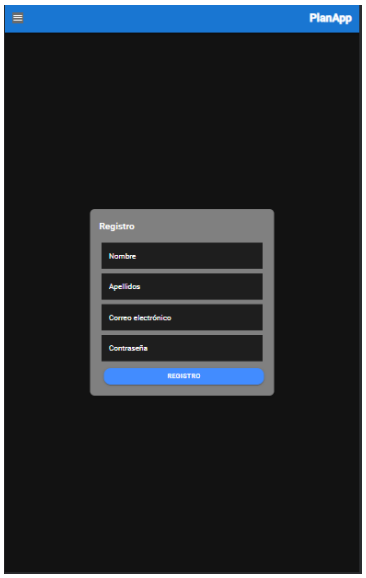


Ilustración 32 Interfaz de registro

9.2.3 Listado de planes

Se muestran todos los planes, los concluidos y los que no lo están, además contiene una interfaz de búsqueda para buscar por nombres los planes.

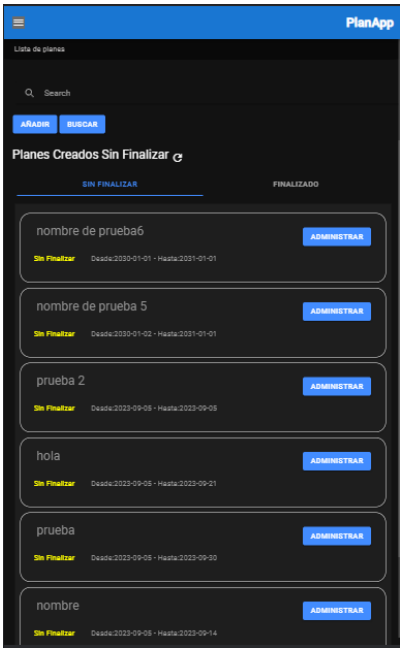


Ilustración 33 Interfaz de listado de planes

9.2.4 Registro de planes

Formulario para crear un nuevo plan.
para crear un plan es necesario el nombre, la fecha de inicio y la fecha de finalización.

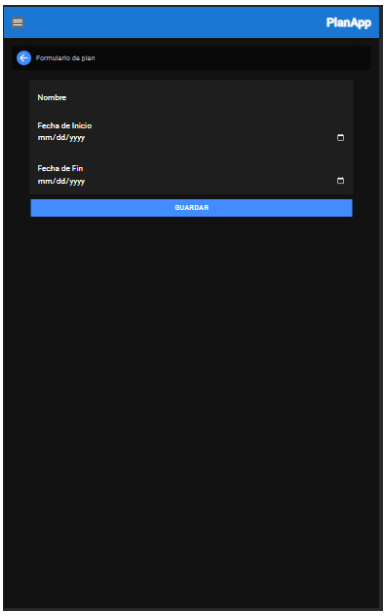


Ilustración 34 Formulario de creación de e planes

9.2.5 Resumen

En esta interfaz se presenta un resumen de avances del proyecto mostrando una barra de progreso por cada elemento del plan y del presupuesto planificado y el utilizado.

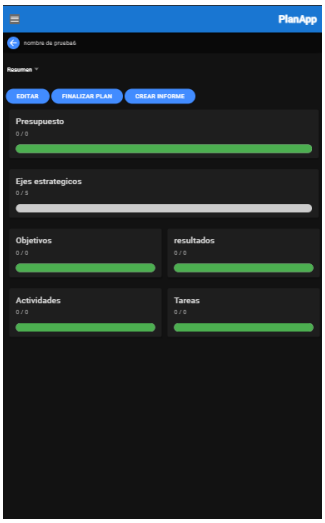


Ilustración 36 Interfaz resumen del plan

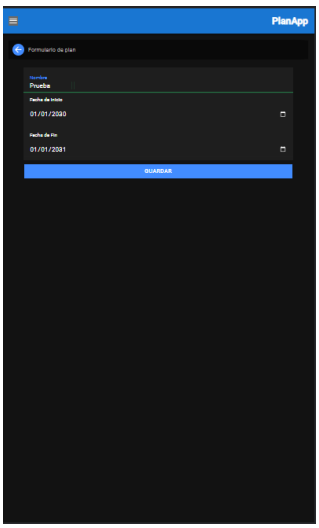


Ilustración 35 formulario edición de información del plan

9.2.6 Colaboradores

En esta interfaz se muestran y administran los colaboradores a los que se le asignará cada elemento de la planificación. Mostrando sus nombres y el botón para eliminarlo de la planificación si no se le ha asignado uno de los elementos.

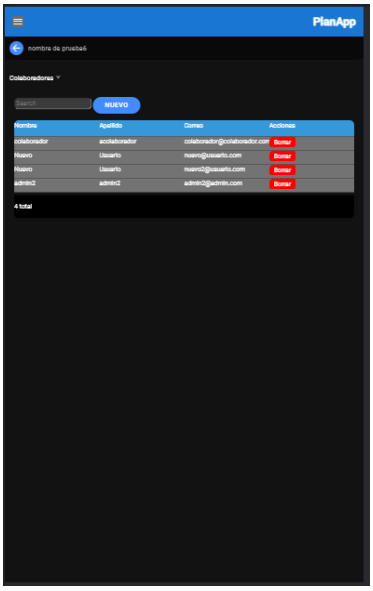


Ilustración 38 Interfaz tabla de colaboradores

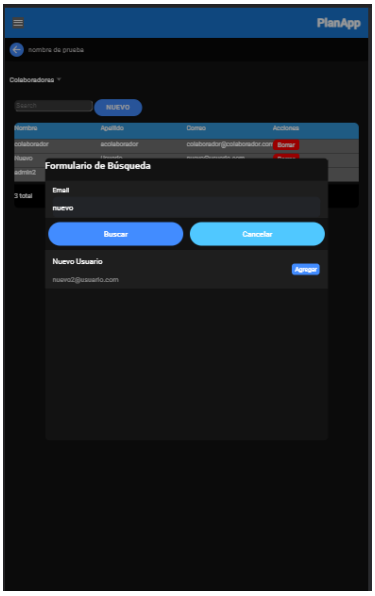


Ilustración 37 formulario de adición de colaboradores

9.2.7 Eje estratégico

Esta interfaz permite listar todos los ejes estratégicos del plan seleccionado. Con un botón para crear los ejes del plan y organizándolos en dos secciones: los completos y los incompletos Cuenta con un botón para ver los indicadores de los ejes listado. Estos ejes cuentan con un botón para ver los indicadores.

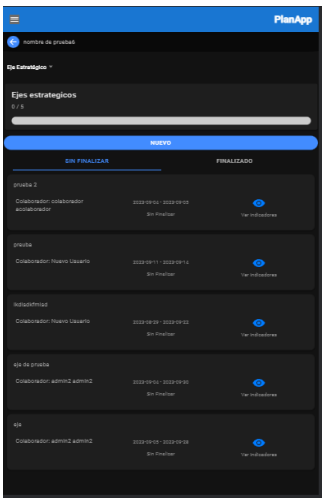


Ilustración 39 Listado de ejes estratégicos

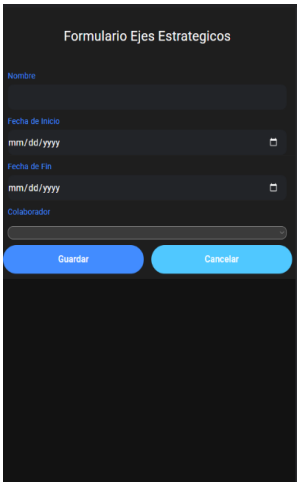


Ilustración 40 Formulario de creación de eje estratégicos

9.2.8 Indicadores de los ejes estratégicos

Esta interfaz muestra el listado de indicadores que tiene un eje estratégico. Además de contener un textbox para agregar los indicadores necesarios y un botón para dar por concluido el eje estratégico. Al momento de pulsar cada indicador se abrirá un formulario para realizar el informe de ese indicador seleccionado

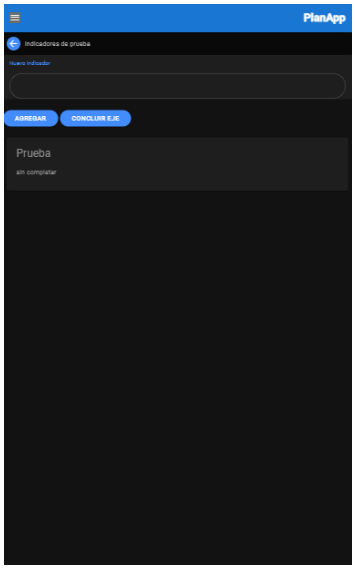


Ilustración 41 Interfaz de gestión de ejes estratégico

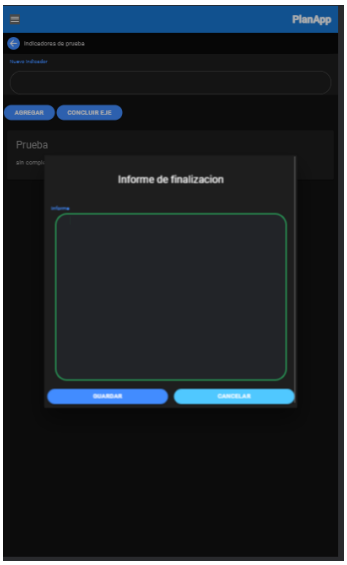


Ilustración 42 Formulario de informe de finalización de indicador

9.2.9 Objetivos

Esta interfaz muestra el listado de indicadores que tiene un eje estratégico. Además de contener un textbox para agregar los indicadores necesarios y un botón para dar por concluido. Al momento de pulsar cada indicador se abrirá un formulario para realizar el informe de ese indicador seleccionado.

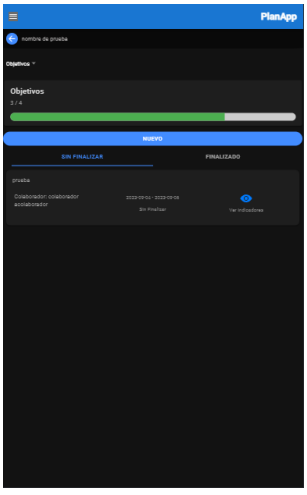


Ilustración 43 Listado de objetivos

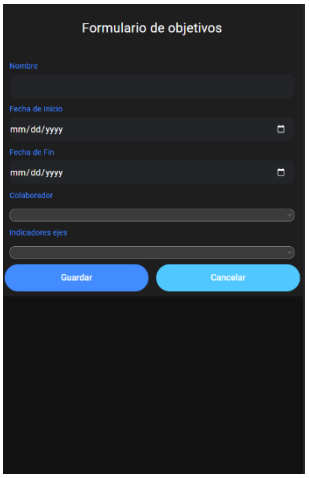


Ilustración 44 Formulario de creación de objetivos

9.2.10 Indicadores de objetivo

Esta interfaz presenta el listado de indicadores asociados a un objetivo. También incluye un cuadro de texto para agregar nuevos indicadores y un botón para marcar el objetivo como completado. Al hacer clic en cada indicador, se abrirá un formulario para completar el informe correspondiente a ese indicador seleccionado.

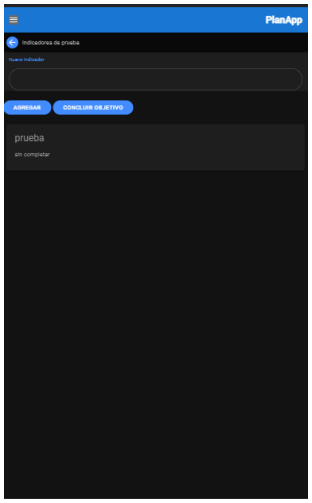


Ilustración 46 Interfaz de gestión de indicadores de objetivos

Ilustración 45 Formulario de informe de finalización de indicador

9.2.11 Resultados

En esta interfaz, se exhiben todos los resultados del plan seleccionado, y se brinda la opción de crear nuevos resultados. Estos resultados se organizan en dos categorías: los logrados y los pendientes. Asimismo, se facilita la visualización de los indicadores relacionados con cada uno de ellos a través de un botón específico. Además, para cada resultado, existe la posibilidad de acceder a sus indicadores particulares mediante un botón adicional.

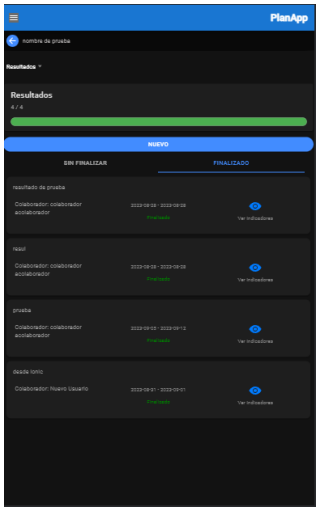


Ilustración 48 Listado de resultados

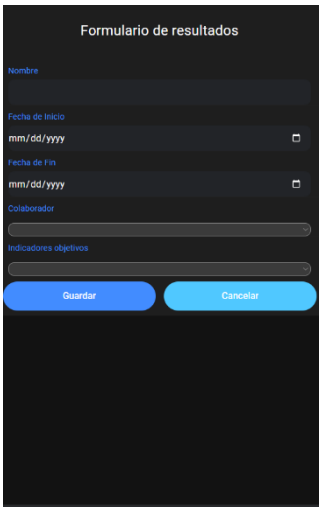


Ilustración 47 Formulario de inserción de resultados esperados

9.2.12 Indicadores Resultados

Esta interfaz muestra el listado de indicadores vinculados a un resultado. También dispone de un cuadro de texto para añadir nuevos indicadores y un botón para marcar el resultado como concluido. Al seleccionar un indicador, se desplegará un formulario para completar el informe correspondiente a dicho indicador.

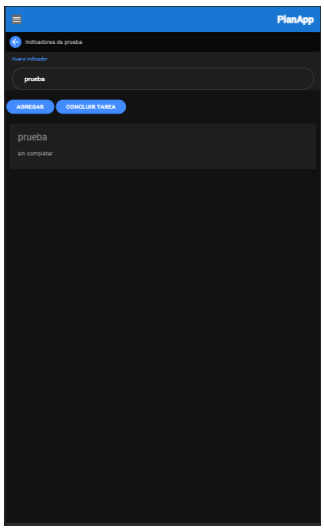


Ilustración 50 Interfaz de gestión de indicadores de resultados

Ilustración 49 Formulario de informe de finalización de indicador

9.2.13 Actividades

En esta interfaz, se presentan todas las actividades del plan seleccionado, con la opción de crear nuevas actividades. Estas actividades se clasifican en dos categorías: las completadas y las pendientes. Además, se proporciona un botón específico para ver los indicadores asociados a cada una de ellas de manera sencilla.

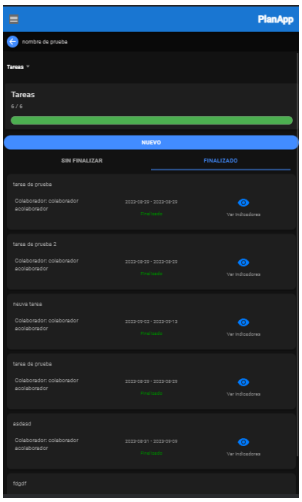


Ilustración 51 Listado de actividades

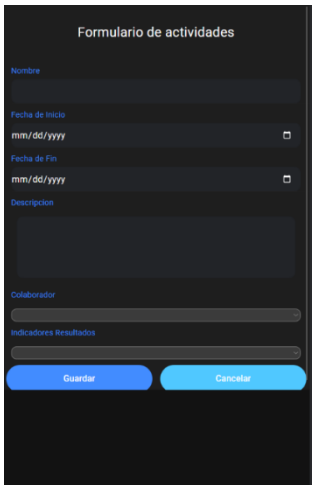


Ilustración 52 Formulario de inserción de actividades

9.2.14 Indicadores actividades

En esta interfaz, se presenta el listado de indicadores relacionados con una actividad. También se incluye un campo de texto para agregar nuevos indicadores y un botón para marcar la actividad como finalizada. Cuando se selecciona un indicador, se abrirá un formulario para completar el informe asociado a ese indicador específico.

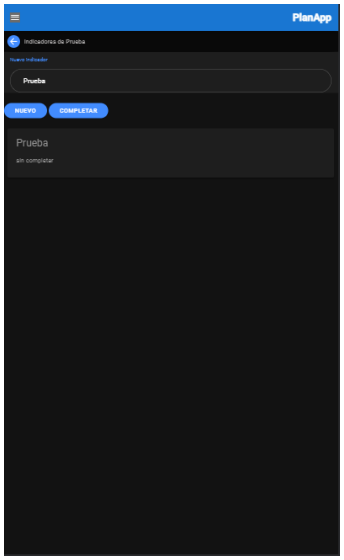


Ilustración 53 Interfaz de gestión de indicadores de actividades

Ilustración 54 Formulario de informe de finalización de indicador

9.2.15 Tareas

En esta interfaz, se muestran todas las tareas del plan seleccionado, junto con la opción de crear nuevas tareas. Estas tareas se dividen en dos categorías: las que han sido finalizadas y las que están pendientes. Además, se proporciona un botón específico para visualizar los indicadores relacionados con cada tarea de manera fácil y accesible.

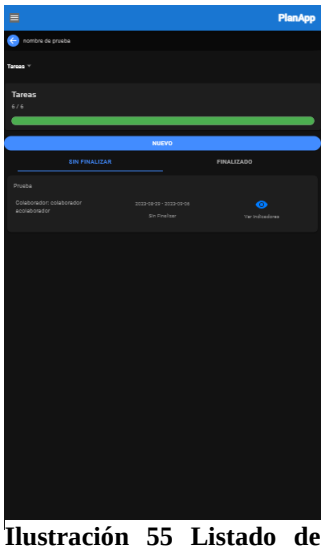


Ilustración 55 Listado de tareas

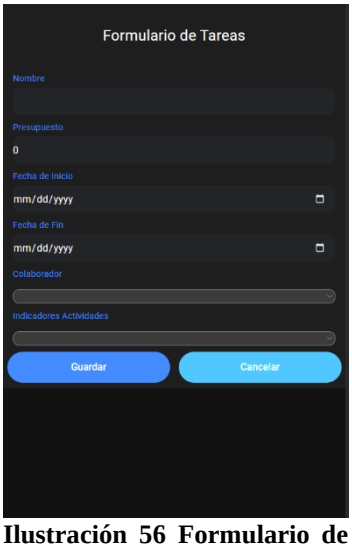


Ilustración 56 Formulario de inserción de tareas

9.2.16 Indicadores tareas

En esta interfaz, se exhibe el catálogo de indicadores vinculados a una tarea. Asimismo, se incorpora un espacio de texto para la adición de indicadores nuevos y un botón para señalar la tarea como completada. Al elegir un indicador, se activará un formulario para completar el informe correspondiente a ese indicador particular.

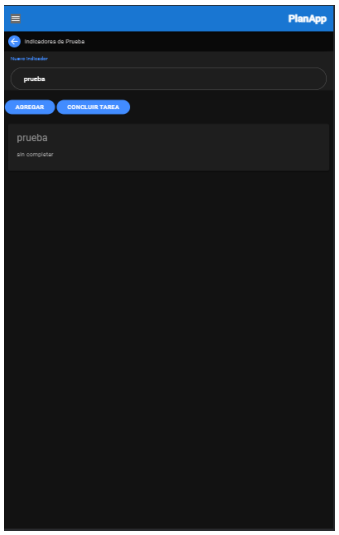


Ilustración 58 Interfaz de gestión de indicadores de tareas

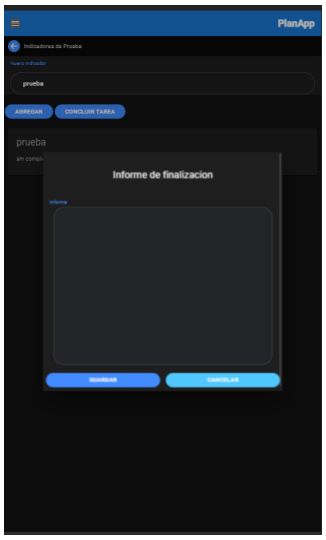


Ilustración 57 Formulario de informe de finalización de indicador

9.3 Funcionalidades

Para poder operar de manera correcta, la aplicación tiene diferentes funcionalidades, que son descritas a continuación.

9.3.1 Registro

Pulsar la opción de Registrarse en el menú de hamburguesa y rellenar los campos correspondientes

- Nombre
- Apellido
- Correo electrónico
- Contraseña

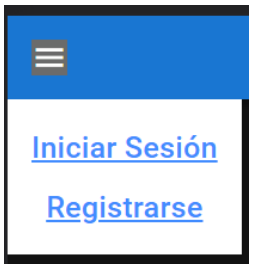
1. 

Ilustración 59 menú principal

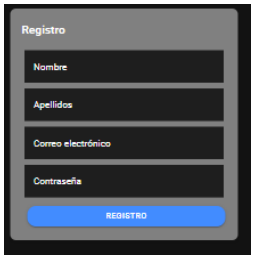
2. 

Ilustración 60 Formulario de registro

9.3.2 Inicio de sesión

Luego de registrase, se debe pulsar la opción de inicio de sesión y colocar los datos correspondientes; correo electrónico y contraseña y pulsar el botón inicio de sesión

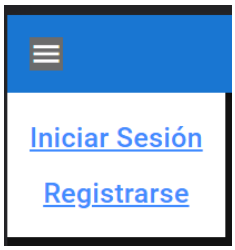
1. 

Ilustración 62 menú principal

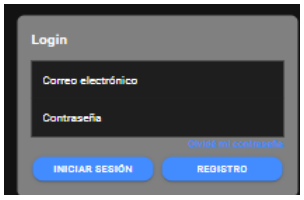
2. 

Ilustración 61 Formulario de inicio de sesión

9.3.3 Cerrar sesión

Luego de iniciar sesión se debe pulsar la opción de cerrar sesión en el menú hamburguesa

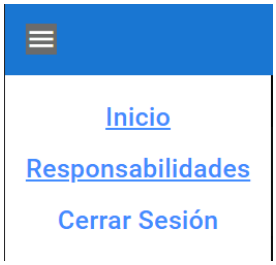


Ilustración 63 menú principal extendido

9.3.4 Crear plan

Para crear un plan se debe pulsar el botón Añadir en la interfaz de listado de planes. Esto lo llevará al formulario de Creación de planes. Para luego rellenar los datos para la creación del plan.

- Nombre
- Fecha inicio
- Fecha final

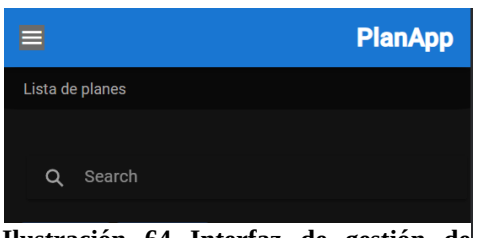


Ilustración 64 Interfaz de gestión de planes

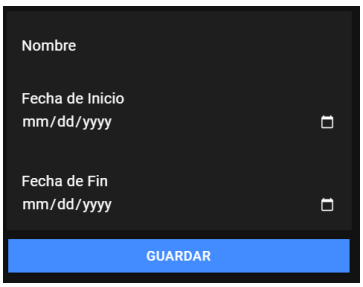


Ilustración 65 Formulario de inserción de planes

9.3.5 Administrar plan

Para administrar un plan se debe pulsar el botón Administrar del plan correspondiente en la interfaz de lista de planes

Ilustración 66 Tarjeta de información de planes

9.3.6 Editar plan

Para editar la información de un plan se debe pulsar el botón Editar. Esto lo llevará a la interfaz del formulario de creación de planes, con la información actual del plan ya incluida

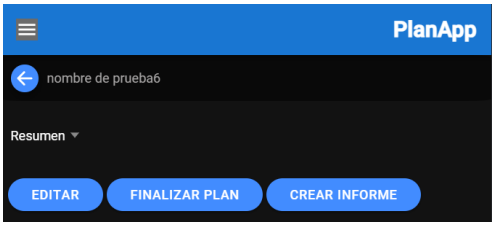


Ilustración 68 Interfaz de gestión de plan

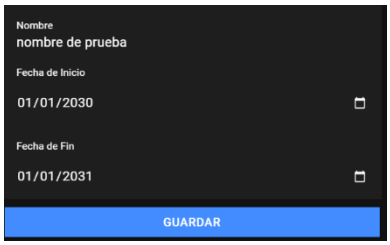


Ilustración 67 Formulario de edición de planes

9.3.7 Ver colaboradores

Para ver los colaboradores se debe seleccionar la opción Colaboradores de la interfaz de administración de planes y esto lo llevará a la interfaz de colaboradores.

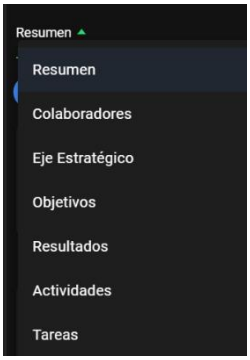


Ilustración 70 menú de gestión de planes

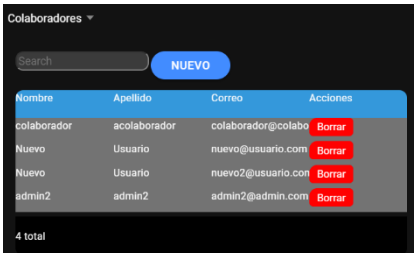


Ilustración 69 Tabla de colaboradores

9.3.8 Añadir colaborador

Para añadir un nuevo colaborador hay que pulsar botón nuevo en la interfaz y colaboradores. Esto abrirá el formulario de búsqueda, se debe escribir el correo electrónico del usuario y pulsar Buscar. Luego pulsar Añadir el usuario que se desee.

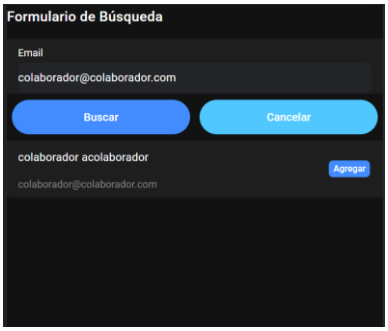


Ilustración 71 Interfaz de inserción de colaborador

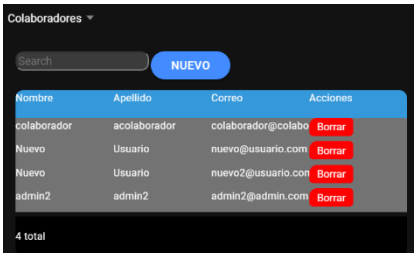


Ilustración 72 Tabla de colaboradores

9.3.9 Eliminar colaborador

En la interfaz de colaborador se debe pulsar el botón Eliminar en el usuario deseado, si el colaborador no tiene asignaciones se eliminará de la planificación.

Ilustración 73 Tabla de colaboradores

9.3.10 Ver Componente

Para ver algún componente de la aplicación se debe seleccionar el componente de la interfaz de administración de planes y esto lo llevara a la interfaz del correspondiente componente.

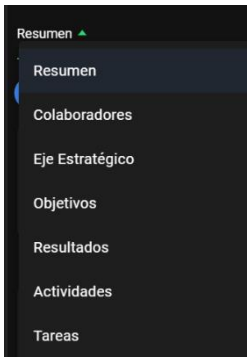


Ilustración 74 menú gestión de planes

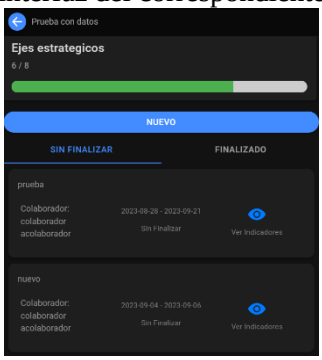


Ilustración 75 Listado de ejes estratégicos

9.3.11 Añadir Componente

En la interfaz del componente se debe presionar el botón de nuevo, esto abrirá el formulario del componente seleccionado. Luego rellenar los datos correspondientes

- Nombre
- Fecha inicio
- Fecha fin
- Colaborador responsable

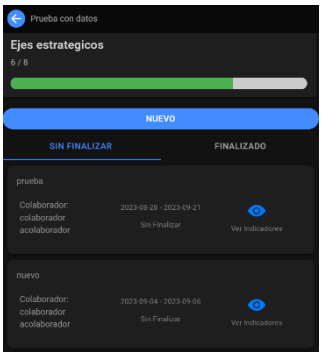


Ilustración 77 Listado de ejes estratégicos

Ilustración 76 Formulario de inserción de ejes estratégicos

9.3.12 Concluir componente

En la interfaz del componente se debe presionar Ver indicadores luego se abrirá la interfaz de indicadores del componente. Luego presionar el botón Concluir.

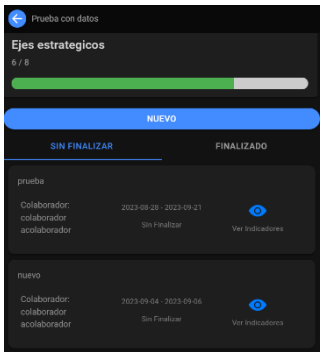


Ilustración 78 Listado de ejes estratégicos

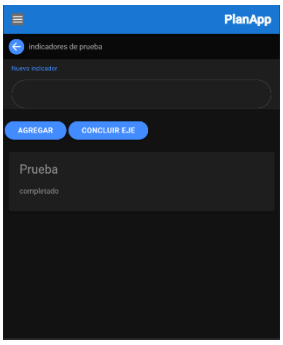


Ilustración 79 Interfaz de gestión de indicadores

9.3.13 Ver Indicadores

Para ver los indicadores de algún componente se debe presionar el botón de Ver indicadores Luego se abrirá la interfaz de los indicadores del componente seleccionado en donde se listan los indicadores.

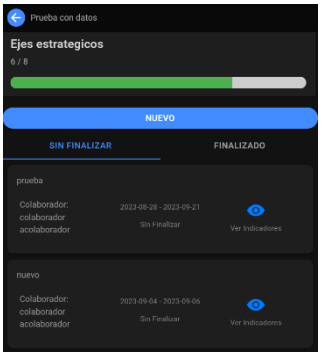


Ilustración 81 Listado de ejes estratégicos

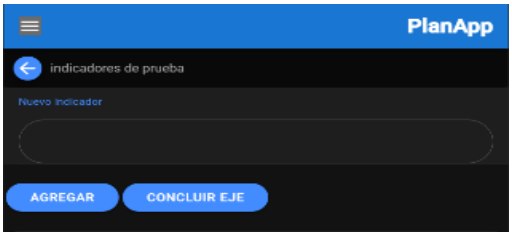


Ilustración 80 Interfaz de gestión de indicadores

9.3.14 Añadir Indicador

Para añadir un nuevo indicador del componente seleccionado se debe ir a la interfaz de indicadores y se debe escribir en el textbox el nombre del indicador para luego presionar al botón de Agregar.

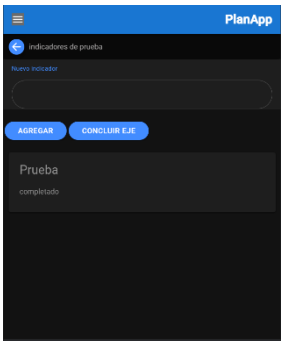


Ilustración 82 Interfaz de gestión de indicadores

9.3.15 Concluir Indicador

Para concluir un indicador se debe pulsar un indicador en la interfaz de indicadores, esto abrirá un formulario de informe, se debe rellenar la información y luego presionar Guardar.

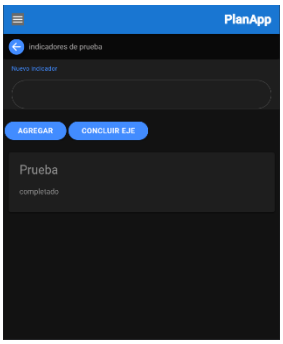


Ilustración 84 Interfaz de gestión de indicadores

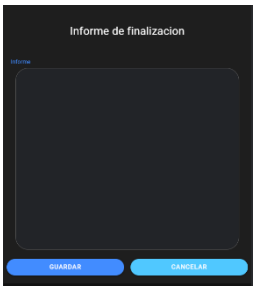


Ilustración 83 Formulario de conclusión de indicador

9.3.16 Responsabilidades en otros planes

Para ver las asignaciones que tiene su usuario en otros planes se debe pulsar la opción Responsabilidades esto llevar a la interfaz de responsabilidades

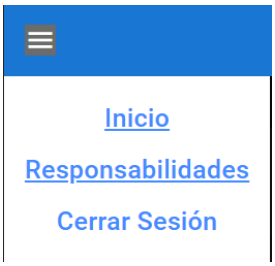


Ilustración 86 menú principal

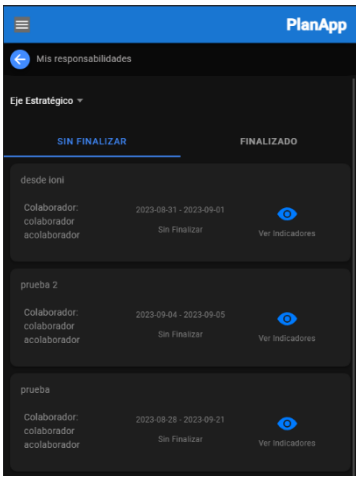


Ilustración 85 Listado de ejes estratégicos

9.3.17 Crear informe

Para la crear un informe se debe presionar le botón de Crear un informe esto descargara un PDF con el informe de la situación actual del plan. Cada componente contiene un informe específico. El informe contiene el listado de componentes dependiente del componente elegido, su estado actual y las observaciones de conclusión de cada componente dependiente.

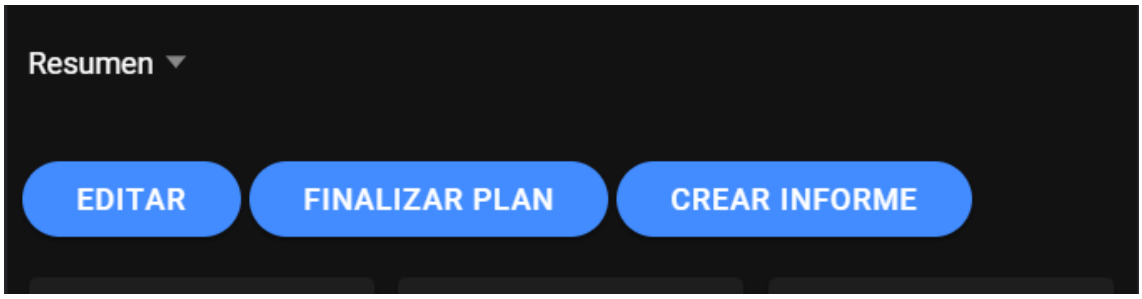


Ilustración 88 menú de gestión de planes

Informe de actividades del proyecto

Nombre del proyecto: Prueba con datos

Fecha reporte: Sat Sep 09 2023 15:23:13 GMT+0200 (Central European Summer Time)

Eje estratégico: Prueba con datos

Objetivo: Prueba con datos

Resultados esperados	Avances en cumplimiento de actividad	Observaciones
resul	Completado	
prueba	Completado	
prueba	Completado	
prueba	Completado	
dsfdsf	Completado	
resultado de prueba	Completado	
desde ionic	Completado	
nuevo	Sin Completar	

Ilustración 87 Informe de actividades

10. Anexo 2: Manual de Despliegue

En este documento encontrarás una guía detallada de como configurar el backend y el frontend de la aplicación de planificación de proyectos

10.1 Backend

Como el proyecto está hecho en base a PostgREST la configuración del backend es la configuración de la base de datos. En esta sección se configurarán, los roles, tablas funciones y triggers de la base de datos que a su vez es configurar el backend

10.1.1 Configuración de base de datos

10.1.1.1 Creación de roles

En la base de datos se deben crear 3 roles que cumplirán diferentes funciones en la aplicación, estos roles son los que al momento de iniciar sesión se le asigna a un usuario, estos roles son:

- Anónimo
- Autenticado
- Administrador

10.1.1.2 Scripts

Para crear estos roles se debe ejecutar los siguientes scripts

1. Para el rol anónimo(anon)

```
CREATE ROLE anon WITH
  NOLOGIN
  NOSUPERUSER
  NOINHERIT
  NOCREATEDB
  NOCREATEROLE
  NOREPLICATION;
```

2. Para el rol de autenticado(authenticator)

```
CREATE ROLE authenticator WITH
  NOLOGIN
  NOSUPERUSER
  NOINHERIT
  NOCREATEDB
  NOCREATEROLE
  NOREPLICATION;

GRANT anon TO authenticator;
```

3. Para el rol administrador(postgres)

```
CREATE ROLE postgres WITH
  LOGIN
  SUPERUSER
  INHERIT
  CREATEDB
  CREATEROLE
  REPLICATION
  ENCRYPTED PASSWORD 'SCRAM-SHA-256$4096:xWVUa/P8P+caiomHYh/bjg==$1akvJycKfahXK+Ho9UT6LaDOvHQI1A8CWxWMl4xJAwM=:Eu5JmLpo04x1p7lcOPsAyYyrfQP2pqinFu+91kv/MKU=';
```

10.1.2 Creación de schemas

Para el proyecto se crearon 2 schemas, uno para manejar los usuarios y otro para el manejo de los datos de planes.

10.1.2.1 Scripts

- Schema de autenticación(basic_auth) Este esquema se encarga de guardar los datos de los usuarios y el inicio de sesión. A este esquema debe tener acceso todos los roles

```
CREATE SCHEMA IF NOT EXISTS basic_auth
    AUTHORIZATION postgres;
```

- Schema de datos(public), este esquema se encarga de todo los datos y funciones de los planes y sus componentes. A este esquema solo debe tener acceso el rol administrador y el usuario autenticado.

```
CREATE SCHEMA IF NOT EXISTS public
    AUTHORIZATION postgres;
```

10.1.3 Creación de tablas

Para este proyecto es necesario crear varias tablas que sirvan para almacenar los datos correspondientes. Se deben crear las siguientes tablas:

- Tabla de usuarios
- Tabla de actividades
- Tabla de colaboradores
- Tabla de eje estratégico
- Tabla de estados
- Tabla de indicadores de actividades
- Tabla de indicadores de ejes estratégico
- Tabla de indicadores de objetivos
- Tabla de indicadores de resultados
- Tabla de indicadores de tareas
- Tabla de objetivo
- Tabla de plan
- Tabla de resultados
- Tabla de tareas

10.1.3.1 Scripts

Para crear las tablas se deben ejecutar los siguientes scripts.:

- Tabla de Usuarios(users): esta contiene los datos de los usuarios con sus respectivos roles

```
CREATE TABLE IF NOT EXISTS basic_auth.users
(
    email text COLLATE pg_catalog."default" NOT NULL,
    pass text COLLATE pg_catalog."default" NOT NULL,
    role name COLLATE pg_catalog."C" NOT NULL,
    idusuario integer NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),
    nombre text COLLATE pg_catalog."default",
    apellido text COLLATE pg_catalog."default",
    CONSTRAINT idusuarioprimary PRIMARY KEY (idusuario),
    CONSTRAINT email_unico UNIQUE (email),
    CONSTRAINT unique_idusuario UNIQUE (idusuario),
    CONSTRAINT users_email_check CHECK (email ~* '^.+@.+\..+$'::text),
    CONSTRAINT users_pass_check CHECK (length(pass) < 512),
    CONSTRAINT users_role_check CHECK (length(role::text) < 512)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS basic_auth.users
    OWNER to postgres;

REVOKE ALL ON TABLE basic_auth.users FROM anon;
REVOKE ALL ON TABLE basic_auth.users FROM authenticator;

GRANT INSERT ON TABLE basic_auth.users TO anon;

GRANT SELECT, UPDATE ON TABLE basic_auth.users TO authenticator;

GRANT ALL ON TABLE basic_auth.users TO postgres;
```

- Tabla de plan: Esta tabla almacena los planes creados por los usuarios.

```
CREATE TABLE IF NOT EXISTS public.plan
(
    idplan integer NOT NULL DEFAULT nextval('plan_idplan_seq'::regclass),
    idusuario integer,
    idestado integer,
    nombre character varying(100) COLLATE pg_catalog."default",
    fechainicio date,
    fechafinal date,
    "FechaCompletado" date,
    "informeFinal" text COLLATE pg_catalog."default",
    CONSTRAINT plan_pkey PRIMARY KEY (idplan),
    CONSTRAINT plan_idestado_fkey FOREIGN KEY (idestado)
        REFERENCES public.estados (idestado) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.plan
    OWNER to postgres;

REVOKE ALL ON TABLE public.plan FROM authenticator;

GRANT DELETE, INSERT, SELECT, UPDATE ON TABLE public.plan TO authenticator;

GRANT ALL ON TABLE public.plan TO postgres;
```

- Tabla de estados, aquí se parametrizan los estados en los cuales se puede encontrar un plan, sus componentes e indicadores.

```
CREATE TABLE IF NOT EXISTS public.estados
(
    idestado integer NOT NULL DEFAULT nextval('estados_idestado_seq'::regclass),
    nombre text COLLATE pg_catalog."default",
    CONSTRAINT estados_pkey PRIMARY KEY (idestado)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.estados
    OWNER to postgres;

REVOKE ALL ON TABLE public.estados FROM anon;
REVOKE ALL ON TABLE public.estados FROM authenticator;

GRANT SELECT ON TABLE public.estados TO anon;

GRANT SELECT ON TABLE public.estados TO authenticator;

GRANT ALL ON TABLE public.estados TO postgres;
```

- Tabla de colaboradores, Los colaboradores son los usuarios que se encargan del cumplimiento de los diferentes planes, esta tabla se encarga de almacenar la unión de los colaboradores y los planes.

```
CREATE TABLE IF NOT EXISTS public.colaboradores
(
    idcolaborador integer NOT NULL DEFAULT nextval('colaboradores_idcolaborador_seq'::regclass),
    idplan integer,
    idusuario integer,
    fecha date,
    CONSTRAINT colaboradores_pkey PRIMARY KEY (idcolaborador),
    CONSTRAINT colaboradores_idplan_fkey FOREIGN KEY (idplan)
        REFERENCES public.plan (idplan) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT colaboradores_idusuario_fkey FOREIGN KEY (idusuario)
        REFERENCES basic_auth.users (idusuario) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.colaboradores
    OWNER to postgres;

REVOKE ALL ON TABLE public.colaboradores FROM authenticator;

GRANT UPDATE, INSERT, SELECT ON TABLE public.colaboradores TO authenticator;

GRANT ALL ON TABLE public.colaboradores TO postgres;
```

- Tabla de eje estratégico, los ejes estratégicos son el primer componente en el escalafón de los planes y esta tabla contiene los datos de estos.

```
CREATE TABLE IF NOT EXISTS public.ejeestrategico
(
    ideje integer NOT NULL DEFAULT nextval('ejeestrategico_ideje_seq'::regclass),
    nombre text COLLATE pg_catalog."default",
    idplan integer,
    fechainicio date,
    fechafinal date,
    idcolaborador integer,
    esstado integer,
    "FechaCompletado" date,
    CONSTRAINT ejeestrategico_pkey PRIMARY KEY (ideje),
    CONSTRAINT ejeestrategico_esstado_fkey FOREIGN KEY (esstado)
        REFERENCES public.estados (idestado) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT ejeestrategico_idcolaborador_fkey FOREIGN KEY (idcolaborador)
        REFERENCES public.colaboradores (idcolaborador) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT ejeestrategico_idplan_fkey FOREIGN KEY (idplan)
        REFERENCES public.plan (idplan) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.ejeestrategico
    OWNER to postgres;
```

- Tabla de indicadores de ejes estratégicos, Esta tabla contiene los indicadores de los ejes estratégicos, que sirven para medir si se cumple o no un eje estratégico

```
CREATE TABLE IF NOT EXISTS public.indicadoreseje
(
    idindicadoreje integer NOT NULL DEFAULT nextval('indicadoreseje_idindicadoreje_seq'::regclass),
    nombre text COLLATE pg_catalog."default",
    reporte text COLLATE pg_catalog."default",
    ideje integer,
    completado boolean,
    "FechaCompletado" date,
    CONSTRAINT indicadoreseje_pkey PRIMARY KEY (idindicadoreje),
    CONSTRAINT indicadoreseje_ideje_fkey FOREIGN KEY (ideje)
        REFERENCES public.ejeestrategico (ideje) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.indicadoreseje
    OWNER to postgres;

REVOKE ALL ON TABLE public.indicadoreseje FROM authenticator;

GRANT TRIGGER, INSERT, SELECT, UPDATE, REFERENCES ON TABLE public.indicadoreseje TO authenticator;

GRANT ALL ON TABLE public.indicadoreseje TO postgres;
```

- Tabla de objetivo, almacena los objetivos de los planes almacenados.

```
CREATE TABLE IF NOT EXISTS public.objetivo
(
    idobjetivos integer NOT NULL DEFAULT nextval('objetivo_idobjetivos_seq'::regclass),
    nombre text COLLATE pg_catalog."default",
    idcolaborador integer,
    fechainicio date,
    fechafinal date,
    estado integer,
    idindicadoreje integer,
    idplan integer,
    "FechaCompletado" date,
    CONSTRAINT objetivo_pkey PRIMARY KEY (idobjetivos),
    CONSTRAINT objetivo_estado_fkey FOREIGN KEY (estado)
        REFERENCES public.estados (idestado) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT objetivo_idcolaborador_fkey FOREIGN KEY (idcolaborador)
        REFERENCES public.colaboradores (idcolaborador) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT objetivo_idindicadoreje_fkey FOREIGN KEY (idindicadoreje)
        REFERENCES public.indicadoreseje (idindicadoreje) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT objetivo_idplan_fkey FOREIGN KEY (idplan)
        REFERENCES public.plan (idplan) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;
```

- Tabla de indicadores de objetivos, los indicadores de los objetivos sirven para verificar el cumplimento de los objetivos.

```
CREATE TABLE IF NOT EXISTS public.indicadoresobjetivos
(
    idindicadorobjetivos integer NOT NULL DEFAULT nextval('indicadoresobjetivos_idindicadorobjetivos_seq'::regclass),
    nombre text COLLATE pg_catalog."default" NOT NULL,
    reporte text COLLATE pg_catalog."default",
    idobjetivo integer NOT NULL,
    completado boolean,
    "FechaCompletado" date,
    CONSTRAINT indicadoresobjetivos_pkey PRIMARY KEY (idindicadorobjetivos),
    CONSTRAINT indicadoresobjetivos_idobjetivo_fkey FOREIGN KEY (idobjetivo)
        REFERENCES public.objetivo (idobjetivos) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.indicadoresobjetivos
    OWNER to postgres;

REVOKE ALL ON TABLE public.indicadoresobjetivos FROM authenticator;

GRANT INSERT, SELECT, TRUNCATE, REFERENCES, UPDATE ON TABLE public.indicadoresobjetivos TO authenticator;

GRANT ALL ON TABLE public.indicadoresobjetivos TO postgres;
```

- Tabla de resultados, esta tabla tiene como función el almacenamiento de sus datos de los resultados esperados de los planes

```
CREATE TABLE IF NOT EXISTS public.resultados
(
    idresultados integer NOT NULL DEFAULT nextval('resultados_idresultados_seq'::regclass),
    nombre text COLLATE pg_catalog."default",
    fechainicio date,
    fechafin date,
    idcolaborador integer,
    idindicadorobjetivo integer,
    estado integer,
    idplan integer,
    "FechaCompletado" date,
    CONSTRAINT resultados_pkey PRIMARY KEY (idresultados),
    CONSTRAINT resultados_estado_fkey FOREIGN KEY (estado)
        REFERENCES public.estados (idestado) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT resultados_idcolaborador_fkey FOREIGN KEY (idcolaborador)
        REFERENCES public.colaboradores (idcolaborador) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT resultados_idindicadorobjetivo_fkey FOREIGN KEY (idindicadorobjetivo)
        REFERENCES public.indicadoresobjetivos (idindicadorobjetivos) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT resultados_idplan_fkey FOREIGN KEY (idplan)
        REFERENCES public.plan (idplan) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;
```

- Tabla de indicadores de resultados, esta tabla almacena los indicadores de los resultados esperados.

```
CREATE TABLE IF NOT EXISTS public.indicadoresresultados
(
    idindicadorresultado integer NOT NULL DEFAULT nextval('indicadoresresultados_idindicadorresultado_seq'::regclass),
    nombre text COLLATE pg_catalog."default",
    reporte text COLLATE pg_catalog."default",
    idresultado integer,
    completado boolean,
    "FechaCompletado" date,
    CONSTRAINT indicadoresresultados_pkey PRIMARY KEY (idindicadorresultado),
    CONSTRAINT indicadoresresultados_idresultado_fkey FOREIGN KEY (idresultado)
        REFERENCES public.resultados (idresultados) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.indicadoresresultados
    OWNER to postgres;

REVOKE ALL ON TABLE public.indicadoresresultados FROM authenticator;

GRANT INSERT, SELECT, TRUNCATE, REFERENCES, UPDATE ON TABLE public.indicadoresresultados TO authenticator;

GRANT ALL ON TABLE public.indicadoresresultados TO postgres;
```

- Tabla de actividades, El siguiente script crea la tabla que contendrá los datos de las actividades a realizar para el cumplimiento del plan

```
CREATE TABLE IF NOT EXISTS public.actividades
(
    idactividad integer NOT NULL DEFAULT nextval('actividades_idactividad_seq'::regclass),
    nombre text COLLATE pg_catalog."default",
    fechainicio date,
    fechafin date,
    descripcion text COLLATE pg_catalog."default",
    idestado integer,
    idcolaborador integer,
    idindicadorresultado integer,
    idplan integer,
    "FechaCompletado" date,
    CONSTRAINT actividades_pkey PRIMARY KEY (idactividad),
    CONSTRAINT actividades_idcolaborador_fkey FOREIGN KEY (idcolaborador)
        REFERENCES public.colaboradores (idcolaborador) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT actividades_idestado_fkey FOREIGN KEY (idestado)
        REFERENCES public.estados (idestado) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT actividades_idindicadorresultado_fkey FOREIGN KEY (idindicadorresultado)
        REFERENCES public.indicadoresresultados (idindicadorresultado) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

- Tabla de indicadores de actividades, Este script crea la tabla de los indicadores de las actividades.

```
CREATE TABLE IF NOT EXISTS public.indicadoresactividades
(
    idindicadoractividades integer NOT NULL DEFAULT nextval('indicadoresactividades_idindicadoractividades_seq'::regclass),
    nombre text COLLATE pg_catalog."default",
    reporte text COLLATE pg_catalog."default",
    idactividad integer,
    completado boolean,
    "FechaCompletado" date,
    CONSTRAINT indicadoresactividades_pkey PRIMARY KEY (idindicadoractividades),
    CONSTRAINT indicadoresactividades_idactividad_fkey FOREIGN KEY (idactividad)
        REFERENCES public.actividades (idactividad) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.indicadoresactividades
    OWNER to postgres;

REVOKE ALL ON TABLE public.indicadoresactividades FROM authenticator;

GRANT TRIGGER, INSERT, TRUNCATE, SELECT, UPDATE, REFERENCES ON TABLE public.indicadoresactividades TO authenticator;

GRANT ALL ON TABLE public.indicadoresactividades TO postgres;
```

- Tabla de tareas, esta tabla contiene las diferentes tareas que se deben cumplir para la conclusión del plan

```
CREATE TABLE IF NOT EXISTS public.tareas
(
    idtarea integer NOT NULL DEFAULT nextval('tareas_idtarea_seq'::regclass),
    nombre text COLLATE pg_catalog."default",
    fechainicio date,
    fechafin date,
    presupuesto numeric,
    idcolaborador integer,
    idestado integer,
    idindicadoractividad integer,
    reporte text COLLATE pg_catalog."default",
    presupuestoejecutado numeric,
    idplan integer,
    "FechaCompletado" date,
    CONSTRAINT tareas_pkey PRIMARY KEY (idtarea),
    CONSTRAINT tareas_idcolaborador_fkey FOREIGN KEY (idcolaborador)
        REFERENCES public.colaboradores (idcolaborador) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT tareas_idestado_fkey FOREIGN KEY (idestado)
        REFERENCES public.estados (idestado) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT tareas_idindicadoractividad_fkey FOREIGN KEY (idindicadoractividad)
        REFERENCES public.indicadoresactividades (idindicadoractividades) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT tareas_idplan_fkey FOREIGN KEY (idplan)
        REFERENCES public.plan (idplan) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

- Tabla de indicadores de tareas, Este script crea la tabla para guardar los datos de los indicadores de las diferentes tareas de un plan.

```
CREATE TABLE IF NOT EXISTS public.indicadorestareas
(
    idindicadortareas integer NOT NULL DEFAULT nextval('indicadorestareas_idindicadortareas_seq'::regclass),
    nombre text COLLATE pg_catalog."default",
    reporte text COLLATE pg_catalog."default",
    idtarea integer,
    completado boolean,
    "FechaCompletado" date,
    CONSTRAINT indicadorestareas_pkey PRIMARY KEY (idindicadortareas),
    CONSTRAINT indicadorestareas_idtarea_fkey FOREIGN KEY (idtarea)
        REFERENCES public.tareas (idtarea) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.indicadorestareas
    OWNER to postgres;

REVOKE ALL ON TABLE public.indicadorestareas FROM authenticator;

GRANT TRIGGER, INSERT, TRUNCATE, SELECT, UPDATE, REFERENCES ON TABLE public.indicadorestareas TO authenticator;

GRANT ALL ON TABLE public.indicadorestareas TO postgres;
```

10.1.4 Creación de trigger

Para este proyecto fue necesario crear varios trigger para automatizar varias funciones y así asegurar el correcto funcionamiento de la aplicación. A continuación, los scripts para crearlos:

10.1.4.1 Scripts

- Es necesario crear un script para comprobar si el rol existe al momento de registrar un usuario y así evitar mal funcionamientos de la aplicación

```
CREATE OR REPLACE FUNCTION basic_auth.check_role_exists()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
begin
  if not exists (select 1 from pg_roles as r where r.rolname = new.role) then
    raise foreign_key_violation using message =
      'unknown database role: ' || new.role;
    return null;
  end if;
  return new;
end
$BODY$;

ALTER FUNCTION basic_auth.check_role_exists()
    OWNER TO postgres;
```

- Es necesario encriptar la contraseña para la protección contra acceso no autorizado y mantener la privacidad de los usuarios

```
CREATE OR REPLACE FUNCTION basic_auth.encrypt_pass()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
begin
  if tg_op = 'INSERT' or new.pass <> old.pass then
    new.pass = crypt(new.pass, gen_salt('bf'));
  end if;
  return new;
end
$BODY$;

ALTER FUNCTION basic_auth.encrypt_pass()
    OWNER TO postgres;
```

10.1.5 Creación de types

Los types para esta aplicación sirven para devolver un dato en un formato personalizado. Para este proyecto fue necesario crear varios types ya que es necesario crear endpoint con varios cruces de datos y algunos datos deben ser visibles y otros no, además que las funciones deben devolver un tipo registrado para que PostgREST.

10.1.5.1 Scripts

- Type para mostrar las actividades

```
CREATE TYPE public.actividadesshow AS
(
    idactividad integer,
    nombre text,
    fechainicio date,
    fechafin date,
    descripcion text,
    estado text,
    colaborador text
);

ALTER TYPE public.actividadesshow
    OWNER TO postgres;
```

- Type para mostrar los colaboradores

```
CREATE TYPE public.colaborador AS
(
    idusuario integer,
    nombre text,
    apellido text,
    email text,
    idcolaborador integer
);

ALTER TYPE public.colaborador
    OWNER TO postgres;
```

- Type para mostrar los ejes estratégicos

```
CREATE TYPE public.ejetype AS
(
    ideje integer,
    nombre text,
    fechainicio date,
    fechafinal date,
    colaborador text,
    idestado integer,
    estado text
);

ALTER TYPE public.ejetype
    OWNER TO postgres;
```

- Type para mostrar los indicadores

```
CREATE TYPE public.indicadoresejeshow AS
(
    idindicadoreje integer,
    nombre text,
    reporte text,
    ideje integer,
    completado text
);

ALTER TYPE public.indicadoresejeshow
    OWNER TO postgres;
```

- Type para mostrar el token

```
CREATE TYPE public.jwt_token AS
(
    token text
);

ALTER TYPE public.jwt_token
    OWNER TO postgres;
```

- Type para mostrar los objetivos

```
CREATE TYPE public.objetivotype AS
(
    ideje integer,
    nombre text,
    fechainicio date,
    fechafinal date,
    colaborador text,
    idestado integer,
    estado text
);

ALTER TYPE public.objetivotype
    OWNER TO postgres;
```

- Type para mostrar los planes

```
CREATE TYPE public.planestype AS
(
    id integer,
    nombre_plan text,
    fecha_inicio date,
    fecha_final date,
    estado text,
    "fechaFinalizado" date
);

ALTER TYPE public.planestype
    OWNER TO postgres;
```

- Type para mostrar el progreso de un componente

```
CREATE TYPE public.progreso AS
(
    totales integer,
    completados integer
);

ALTER TYPE public.progreso
    OWNER TO postgres;
```

- Type para mostrar el uso del presupuesto

```
CREATE TYPE public.progresopresupuesto AS
(
    totalplaneado numeric,
    total_ejecutado numeric
);

ALTER TYPE public.progresopresupuesto
    OWNER TO postgres;
```

- Type para mostrar el resultado

```
CREATE TYPE public.resultadosshow AS
(
    idresultados integer,
    nombre text,
    fechainicio date,
    fechafin date,
    idcolaborador integer,
    colaborador text,
    estado text
);

ALTER TYPE public.resultadosshow
    OWNER TO postgres;

GRANT USAGE ON TYPE public.resultadosshow TO PUBLIC;

GRANT USAGE ON TYPE public.resultadosshow TO authenticator;

GRANT USAGE ON TYPE public.resultadosshow TO postgres;
```

- Formato en el cual se mostrarán las tareas

```
CREATE TYPE public.tareasshow AS
(
    idtarea integer,
    nombre text,
    fechainicio date,
    fechafin date,
    presupuesto numeric,
    colaborador text,
    estado text,
    reporte text,
    presupuestoejecutado numeric
);

ALTER TYPE public.tareasshow
    OWNER TO postgres;
```

- Formato de Visualización de Usuarios al Buscar por Correo Electrónico

```
CREATE TYPE public.user_search_result AS
(
    email text,
    idusuario integer,
    nombre text,
    apellido text
);

ALTER TYPE public.user_search_result
    OWNER TO postgres;
```

10.1.6 Creación de funciones

Algunas de las funciones generadas automáticamente por la herramienta no resultan tan prácticas debido a la necesidad de mostrar información adicional que no se encuentra en la tabla de origen o la importancia de evitar la exposición de datos sensibles, como contraseñas.

10.1.6.1 Scripts

10.1.6.1.1 Funciones para el manejo de usuarios

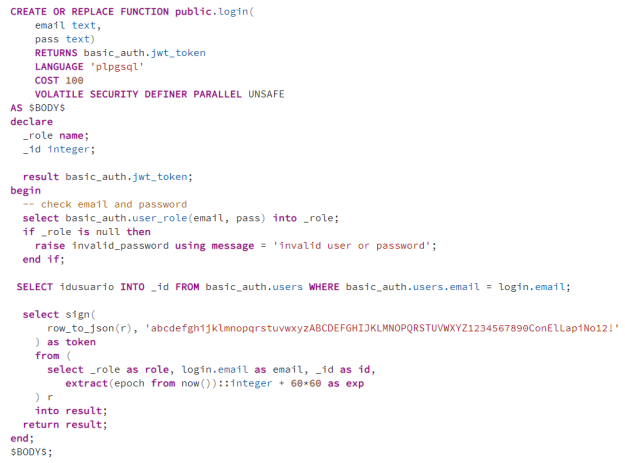
- Registro de usuario, Esta función sirve para insertar un usuario a la base de datos, debido a que está en un esquema diferente al de los planes, el cual es el que se conecta PostgREST

```
CREATE OR REPLACE FUNCTION public.insert_user(
    p_email text,
    p_pass text,
    p_nombre text,
    p_apellido text)
    RETURNS void
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$
BEGIN
  INSERT INTO basic_auth.users (email, pass, role, nombre, apellido)
  VALUES (p_email, p_pass,  'authenticator', p_nombre, p_apellido);
END;
$BODY$;

ALTER FUNCTION public.insert_user(text, text, text, text)
    OWNER TO postgres;
```

- Búsqueda de usuario por email, Al igual que el script pasado es necesario crearlo ya que los datos de usuario están almacenados en un esquema diferente al que se conecta PostgREST

```
CREATE OR REPLACE FUNCTION public.search_users_by_email(
    email_search text)
    RETURNS SETOF user_search_result
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
BEGIN
  RETURN QUERY SELECT email, idusuario, nombre, apellido FROM basic_auth.users WHERE email LIKE '%' || email_search || '%';
END;
$BODY$;
```

- Inicio de sesión, Esta función realiza varias tareas. La primera es autenticar al usuario y la segunda la creación del token que se utilizara para la mayoría de las peticiones de la aplicación

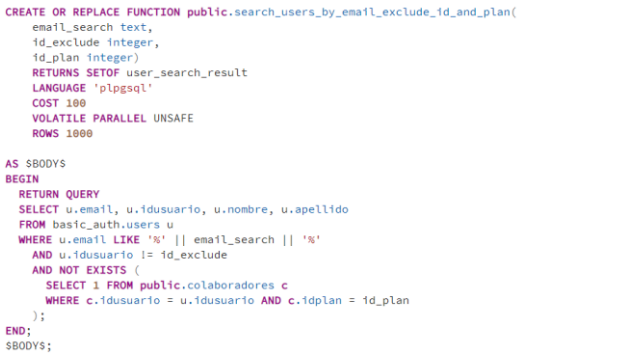


```
CREATE OR REPLACE FUNCTION public.login(
    email text,
    pass text)
    RETURNS basic_auth.jwt_token
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE SECURITY DEFINER PARALLEL UNSAFE
AS $BODY$
declare
  _role name;
  _id integer;

  result basic_auth.jwt_token;
begin
  -- check email and password
  select basic_auth.user_role(email, pass) into _role;
  if _role is null then
    raise invalid_password using message = 'invalid user or password';
  end if;

  SELECT idusuario INTO _id FROM basic_auth.users WHERE basic_auth.users.email = login.email;

  select sign(
      row_to_json(r), 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890ConElLap1No12!'
    ) as token
    from (
      select _role as role, login.email as email, _id as id,
         extract(epoch from now())::integer + 60*60 as exp
    ) r
    into result;
  return result;
end;
$BODY$;
```

- Verificar token, esta función sirve para verificar la autenticidad del token enviado

```
CREATE OR REPLACE FUNCTION public.verify(
    token text,
    secret text,
    algorithm text DEFAULT 'HS256'::text)
    RETURNS TABLE(header json, payload json, valid boolean)
    LANGUAGE 'sql'
    COST 100
    IMMUTABLE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
  SELECT
    jwt.header AS header,
    jwt.payload AS payload,
    jwt.signature_ok AND tstzrange(
      to_timestamp(public.try_cast_double(jwt.payload->>'nbf')),
      to_timestamp(public.try_cast_double(jwt.payload->>'exp'))
    ) @> CURRENT_TIMESTAMP AS valid
  FROM (
    SELECT
      convert_from(public.url_decode(r[1]), 'utf8')::json AS header,
      convert_from(public.url_decode(r[2]), 'utf8')::json AS payload,
      r[3] = public.algorithm_sign(r[1] || '.' || r[2], secret, algorithm) AS signature_ok
    FROM regexp_split_to_array(token, '\.') r
  ) jwt
$BODY$;
```

- Buscar colaboradores para asignar a al plan, esta función busca por el campo de correo electrónico usuarios para colaborar en el plan exceptuando los usuarios ya colaboradores del plan y el usuario creador del plan



```
CREATE OR REPLACE FUNCTION public.search_users_by_email_exclude_id_and_plan(
    email_search text,
    id_exclude integer,
    id_plan integer)
    RETURNS SETOF user_search_result
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
BEGIN
  RETURN QUERY
  SELECT u.email, u.idusuario, u.nombre, u.apellido
  FROM basic_auth.users u
  WHERE u.email LIKE '%' || email_search || '%'
    AND u.idusuario != id_exclude
    AND NOT EXISTS (
      SELECT 1 FROM public.colaboradores c
      WHERE c.idusuario = u.idusuario AND c.idplan = id_plan
    );
END;
$BODY$;
```

10.1.6.1.2 Funciones para el manejo de planes

- Obtener planes de usuario, la función sirve para buscar los planes y que muestre esto en el formato adecuado para presentarlo en la aplicación.

```
CREATE OR REPLACE FUNCTION public.get_planes_por_usuario(
    user_id integer)
    RETURNS SETOF planestype
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
DECLARE
  rec public.planesType;
BEGIN
  FOR rec IN
    SELECT
      p.idplan as id,
      p.nombre as nombre_plan,
      p.fechainicio as fecha_inicio,
      p.fechafinal as fecha_final,
      e.nombre as estado,
      p."FechaCompletado" as fechaFinalizado
    FROM plan p
    INNER JOIN estados e ON p.idestado = e.idestado
    WHERE p.idusuario = user_id
  LOOP
    RETURN NEXT rec;
  END LOOP;
  RETURN;
END;
$BODY$;
```

- Presupuesto del plan, como su nombre lo dice esta función calcula el presupuesto del plan

```
CREATE OR REPLACE FUNCTION public.sumar_presupuesto_por_idplan(
    id_plan integer)
    RETURNS numeric
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$
DECLARE
    total_presupuesto numeric;
BEGIN
    SELECT SUM(presupuesto) INTO total_presupuesto
    FROM tareas
    WHERE idplan = id_plan;

    RETURN total_presupuesto;
END;
$BODY$;
```

- Función para editar el estado del plan, esta función se utiliza para cambiar el estado y añadir el informe de conclusión del plan.

```
CREATE OR REPLACE FUNCTION public.actualizar_estado_plan(
    id_plan integer,
    informe text)
    RETURNS void
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$
BEGIN
  -- Actualiza el estado del plan a 3 (completado)
 UPDATE public.plan
    SET
    idestado=3,
    "informeFinal"=informe,
    "FechaCompletado"=now()
    WHERE idplan=id_plan;
  -- Puedes realizar otras acciones aquí si es necesario

END;
$BODY$;

ALTER FUNCTION public.actualizar_estado_plan(integer, text)
    OWNER TO postgres;

GRANT EXECUTE ON FUNCTION public.actualizar_estado_plan(integer, text) TO PUBLIC;

GRANT EXECUTE ON FUNCTION public.actualizar_estado_plan(integer, text) TO authenticator;

GRANT EXECUTE ON FUNCTION public.actualizar_estado_plan(integer, text) TO postgres;
```

- Obtener los colaboradores de un plan, Como indica su nombre esta función obtiene la información de los colaboradores de un plan y gestionarlos en la aplicación.

```
CREATE OR REPLACE FUNCTION public.get_colaboradores_por_idplan(
    id_plan integer)
    RETURNS SETOF colaborador
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
DECLARE
  rec public.colaborador;
BEGIN
  FOR rec IN
    SELECT

      u.idusuario,
      u.nombre,
      u.apellido,
      u.email,
      c.idcolaborador
    FROM basic_auth.users u
    INNER JOIN public.colaboradores c ON u.idusuario = c.idusuario
    WHERE c.idplan = id_plan
  LOOP
    RETURN NEXT rec;
  END LOOP;
  RETURN;
END;
$BODY$;

ALTER FUNCTION public.get_colaboradores_por_idplan(integer)
    OWNER TO postgres;
```

10.1.6.1.3 Funciones para el manejo de componentes de planes

- Funciones para obtener los componentes del plan. Utilizando el identificador del plan, estas funciones muestran el conjunto de componentes de dicho plan.

```
CREATE OR REPLACE FUNCTION public.get_ejes_por_idplan(
    id_plan integer)
    RETURNS SETOF ejetype
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
DECLARE
  eje public.ejeType;
BEGIN
  FOR eje IN
    SELECT
      ej.ideje,
      ej.nombre,
      ej.fechainicio,
      ej.fechafinal,
      CONCAT(u.nombre, ' ', u.apellido) AS colaborador,
      ej.esstado,
      est.nombre AS estado
    FROM ejeestrategico ej
    INNER JOIN public.colaboradores col ON ej.idcolaborador = col.idcolaborador
    INNER JOIN basic_auth.users u ON col.idusuario = u.idusuario
    INNER JOIN public.estados est ON ej.esstado = est.idestado
    WHERE col.idplan = id_plan
  LOOP
    RETURN NEXT eje;
  END LOOP;
  RETURN;
END;
$BODY$;
```

```
CREATE OR REPLACE FUNCTION public.get_resultados_show_by_idplan(
    id_plan integer)
    RETURNS SETOF resultadosshow
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
BEGIN
  RETURN QUERY
  SELECT
    r.IdResultados ,
    r.Nombre,
    r.FechaInicio,
    r.FechaFin,
    c.IdColaborador,
    u.nombre || ' ' || u.apellido AS colaborador,
    e.nombre AS estado
  FROM public.resultados r
  INNER JOIN public.colaboradores c ON r.IdColaborador = c.IdColaborador
  INNER JOIN basic_auth.users u ON c.IdUsuario = u.idusuario
  INNER JOIN public.estados e ON r.estado = e.idestado
  WHERE r.IdPlan = id_plan;
END;
$BODY$;
```

```
CREATE OR REPLACE FUNCTION public.get_actividades_show_by_idplan(
    id_plan integer)
    RETURNS SETOF actividadesshow
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
BEGIN
  RETURN QUERY
  SELECT
    a.IdActividad,
    a.Nombre,
    a.FechaInicio,
    a.FechaFin,
    a.Descripcion,
    e.nombre AS estado,
    u.nombre || ' ' || u.apellido AS colaborador

  FROM public.Actividades a
  INNER JOIN public.estados e ON a.IdEstado = e.idestado
  INNER JOIN public.Colaboradores c ON a.IdColaborador = c.IdColaborador
  INNER JOIN basic_auth.users u ON c.IdUsuario = u.idusuario

  WHERE a.IdPlan = id_plan;
END;
$BODY$;

ALTER FUNCTION public.get_actividades_show_by_idplan(integer)
    OWNER TO postgres;
```

```
CREATE OR REPLACE FUNCTION public.get_objetivos_por_idplan(
    id_plan integer)
    RETURNS SETOF objetivotype
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
DECLARE
  eje public.ejeType;
BEGIN
  FOR eje IN
    SELECT
      ej.idobjetivos,
      ej.nombre,
      ej.fechainicio,
      ej.fechafinal,
      CONCAT(u.nombre, ' ', u.apellido) AS colaborador,
      ej.estado,
      est.nombre AS estado
    FROM public.objetivo ej
    INNER JOIN public.colaboradores col ON ej.idcolaborador = col.idcolaborador
    INNER JOIN basic_auth.users u ON col.idusuario = u.idusuario
    INNER JOIN public.estados est ON ej.estado = est.idestado
    WHERE col.idplan = id_plan
  LOOP
    RETURN NEXT eje;
  END LOOP;
  RETURN;
END;
$BODY$;
```

```
CREATE OR REPLACE FUNCTION public.get_tareas_show_by_idplan(
    id_plan integer)
    RETURNS SETOF tareasshow
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
BEGIN
    RETURN QUERY
    SELECT
        t.IdTarea,
        t.Nombre,
        t.FechaInicio,
        t.FechaFin,
        t.Presupuesto,
        u.nombre || ' ' || u.apellido AS colaborador,
        e.nombre AS estado,
        t.Reporte,
        t.PresupuestoEjecutado
    FROM public.tareas t
    INNER JOIN public.Colaboradores c ON t.IdColaborador = c.IdColaborador
    INNER JOIN basic_auth.users u ON c.IdUsuario = u.idusuario
    INNER JOIN public.estados e ON t.idestado = e.idestado
    WHERE t.IdPlan = id_plan;
END;
$BODY$;
```

- Obtener los componentes por el colaborador designado, Al igual que las funciones anteriores muestran el conjunto de componentes de un plan, pero esta realiza la búsqueda por el colaborador

```
CREATE OR REPLACE FUNCTION public.get_ejes_por_colaborador(
    id_usuario integer)
    RETURNS SETOF ejetype
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
DECLARE
  eje public.ejeType;
BEGIN
  FOR eje IN
    SELECT
      ej.ideje,
      ej.nombre,
      ej.fechainicio,
      ej.fechafinal,
      CONCAT(u.nombre, ' ', u.apellido) AS colaborador,
      ej.esstado,
      est.nombre AS estado
    FROM ejeestrategico ej
    INNER JOIN public.colaboradores col ON ej.idcolaborador = col.idcolaborador
    INNER JOIN basic_auth.users u ON col.idusuario = u.idusuario
    INNER JOIN public.estados est ON ej.esstado = est.idestado
    WHERE u.idusuario = id_usuario
  LOOP
    RETURN NEXT eje;
  END LOOP;
  RETURN;
END;
$BODY$;
```

```
CREATE OR REPLACE FUNCTION public.get_resultados_show_by_idusuario(
    id_usuario integer)
    RETURNS SETOF resultadosshow
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
BEGIN
  RETURN QUERY
  SELECT
    r.IdResultados ,
    r.Nombre,
    r.FechaInicio,
    r.FechaFin,
    c.IdColaborador,
    u.nombre || ' ' || u.apellido AS colaborador,
    e.nombre AS estado
  FROM public.resultados r
  INNER JOIN public.colaboradores c ON r.IdColaborador = c.IdColaborador
  INNER JOIN basic_auth.users u ON c.IdUsuario = u.idusuario
  INNER JOIN public.estados e ON r.estado = e.idestado
  WHERE  u.idusuario = id_usuario;
END;
$BODY$;
```

```
CREATE OR REPLACE FUNCTION public.get_actividades_show_by_usuario(
    id_usuario integer)
    RETURNS SETOF actividadesshow
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
BEGIN
  RETURN QUERY
  SELECT
    a.IdActividad,
    a.Nombre,
    a.FechaInicio,
    a.FechaFin,
    a.Descripcion,
    e.nombre AS estado,
    u.nombre || ' ' || u.apellido AS colaborador

  FROM public.Actividades a
  INNER JOIN public.estados e ON a.IdEstado = e.idestado
  INNER JOIN public.Colaboradores c ON a.IdColaborador = c.IdColaborador
  INNER JOIN basic_auth.users u ON c.IdUsuario = u.idusuario

  WHERE u.idusuario = id_usuario;
END;
$BODY$;
```

```
CREATE OR REPLACE FUNCTION public.get_tareas_show_by_usuario(
    id_usuario integer)
    RETURNS SETOF tareasshow
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000

AS $BODY$
BEGIN
    RETURN QUERY
    SELECT
        t.IdTarea,
        t.Nombre,
        t.FechaInicio,
        t.FechaFin,
        t.Presupuesto,
        u.nombre || ' ' || u.apellido AS colaborador,
        e.nombre AS estado,
        t.Reporte,
        t.PresupuestoEjecutado
    FROM public.tareas t
    INNER JOIN public.Colaboradores c ON t.IdColaborador = c.IdColaborador
    INNER JOIN basic_auth.users u ON c.IdUsuario = u.idusuario
    INNER JOIN public.estados e ON t.idestado = e.idestado
    WHERE u.idusuario = id_usuario;
END;
$BODY$;
```

```
    FOR eje IN
      SELECT
        ej.idobjetivos,
        ej.nombre,
        ej.fechainicio,
        ej.fechafinal,
        CONCAT(u.nombre, ' ', u.apellido) AS colaborador,
        ej.estado,
        est.nombre AS estado
      FROM public.objetivo ej
      INNER JOIN public.colaboradores col ON ej.idcolaborador = col.idcolaborador
      INNER JOIN basic_auth.users u ON col.idusuario = u.idusuario
      INNER JOIN public.estados est ON ej.estado = est.idestado
      WHERE  u.idusuario = id_usuario
    LOOP
      RETURN NEXT eje;
    END LOOP;
    RETURN;
  END;
  $BODY$;
```

- Obtener el progreso de cada componente, Estos scripts sirven para crear las funciones que muestran los componentes concluidos de un plan comparados con los totales.

```
CREATE OR REPLACE FUNCTION public.get_progreso_actividades_by_idplan(
    id_plan integer)
    RETURNS progreso
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$
DECLARE
    total_actividades integer;
    completadas integer;
    progreso public.Progreso;
BEGIN
    SELECT COUNT(*) INTO total_actividades
    FROM public.actividades
    WHERE IdPlan = id_plan;

    SELECT COUNT(*) INTO completadas
    FROM public.actividades
    WHERE IdPlan = id_plan AND IdEstado = 3;

    progreso := ROW(total_actividades,completadas  );

    RETURN progreso;
END;
$BODY$;
```

```
CREATE OR REPLACE FUNCTION public.get_progreso_objetivos(
    idplan_param integer)
    RETURNS progreso
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$
DECLARE
  totales int;
  completados int;
BEGIN
  -- Obtener el total de objetivos por IdPlan
  SELECT COUNT(*) INTO totales
  FROM public.objetivo o
  WHERE o.idplan = idplan_param;

  -- Obtener el total de objetivos completados por IdPlan
  SELECT COUNT(*) INTO completados
  FROM public.objetivo o
  WHERE o.idplan = idplan_param AND o.estado = 3;

  RETURN (totales, completados)::public.Progreso;
END;
$BODY$;
```

```
CREATE OR REPLACE FUNCTION public.get_progreso_resultado(
    id_plan integer)
    RETURNS progreso
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$
DECLARE
  totales int;
  completados int;
BEGIN
  SELECT COUNT(*) INTO totales
  FROM public.resultados
  WHERE IdPlan = id_plan;

  SELECT COUNT(*) INTO completados
  FROM public.resultados
  WHERE IdPlan = id_plan AND estado = 3;

  RETURN (totales, completados)::public.Progreso;
END;
$BODY$;
```

```
CREATE OR REPLACE FUNCTION public.get_progreso_ejes(
    id_plan integer)
    RETURNS progreso
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$
DECLARE
  total_ejes integer;
  completados integer;
BEGIN
  SELECT COUNT(*) INTO total_ejes FROM public.ejeestrategico WHERE idplan = id_plan;
  SELECT COUNT(*) INTO completados FROM public.ejeestrategico WHERE idplan = id_plan AND esstado = 3;

  RETURN (total_ejes, completados)::public.progreso;
END;
$BODY$;
```

```
CREATE OR REPLACE FUNCTION public.get_progreso_tareas_by_idplan(
    id_plan integer)
    RETURNS progreso
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$
DECLARE
    completados integer;
    totales integer;
BEGIN
    SELECT COUNT(*) INTO completados FROM public.tareas WHERE IdPlan = id_plan;
    SELECT COUNT(*) INTO totales FROM public.tareas WHERE IdPlan = id_plan AND IdEstado = 3;

    RETURN (completados,totales);
END;
$BODY$;
```

10.2 Instalación de PostgREST

Luego de concluir la creación y configuración de la base de datos es necesario descargar y configurar la herramienta PostgREST, el cual convertirá la base de datos previamente creada en una apiREST funcional. Para ello se debe ir al GitHub oficial de PostgREST https://github.com/PostgREST/postgrest/releases/latest , e ir al apartado de assets y descargar la versión del sistema operativo que se usará para ejecutar el api.

Ilustración 89 Enlaces de descargas PostgREST

10.3 Configuración PostgREST

Luego de descargar el software, se debe crear un fichero y llamarlo “postgrest.conf” y colocar los siguientes datos

- db-uri
  Es la url de conexión de la base de datos, con el usuario, seguido de la contraseña y el puerto donde se aloja la base de datos.
- db-schema
  Este es el esquema en el cual se conectará la herramienta PostgREST.
- db-anon-role
  Rol que se le asignará al usuario si no está logueado.
- jwt-secret
  Debe ser el mismo string que se utiliza en la función de login para generar el token, pero este se utiliza para comprobar si el token es válido.

```
db-uri = "postgres://postgres:root@127.0.0.1:5432/postgres"

db-schema = "public"
db-anon-role = "anon"


jwt-secret = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890ConElLapiNo12!"
```

10.4 Ejecutar Proyecto web

Para ejecutar el proyecto se debe abrir una ventana de comandos en el directorio donde se encuentra el proyecto y utilizar el comando “Ionic serve”, esto creará un servidor que ejecuta el proyecto en la url http://localhost:8100

Ilustración 91 Ejemplo comando ejecutar proyecto web

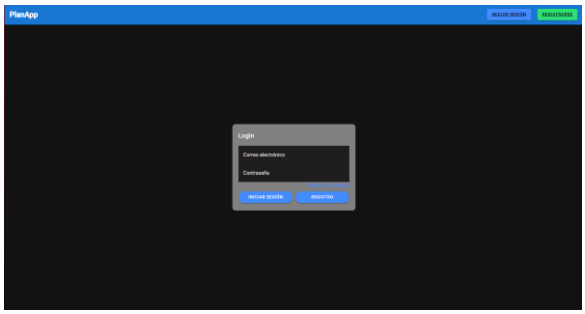


Ilustración 90 Login versión web

10.5 Ejecutar proyecto móvil

Para ejecutar la aplicación en un móvil Android es necesario ejecutar los siguientes comandos.

- Para construir la aplicación y agregar la plataforma Android

```
PS E:\master\tfm Files\TFMProyect\TFMApp> ionic cordova platform add android
```

Ilustración 92 Comando agregar plataforma Android

- Para compilar la aplicación para Android

```
PS E:\master\tfm Files\TFMProyect\TFMApp> ionic cordova run android
```

Ilustración 93 Comando compilar versión Android

- Instalar la aplicación en el dispositivo Android

```
PS E:\master\tfm Files\TFMProyect\TFMApp> ionic cordova build android
```

Ilustración 94 comando instalación dispositivo

ESCUELA POLITECNICA
SUPERIOR