IES ALBARREGAS

TÉCNICO SUPERIOR EN DESARROLLO DE APLICACIONES WEB

Departamento de Informática

PROYECTO

Manual Técnico

Autor/es: Agustín Gómez Piñero
Curso Académico: 2024/25

Índice

1. Introducción **3**
2. Arquitectura de la aplicación **4**
2.1 Introducción 4
2.2 Backend 4
2.3 Frontend 5
3. Documentación técnica **6**
3.1 Análisis 6
3.1.1 Requisitos funcionales 6
3.1.2 Requisitos no funcionales 8
3.2 Desarrollo 8
3.2.1 Diseño de la base de datos 8
3.3 Pruebas realizadas 12
4. Proceso de despliegue **13**
4.1 Despliegue cliente 13
4.2 Despliegue servidor 15
4.3 Despliegue BBDD + configuración Resend 17
5. Propuestas de mejora **22**
5.1 Requisitos funcionales pendientes 22
5.2 Requisitos técnicos 22
5.2.1 Backend **22**
5.2.1 Frontend **22**
6. Webgrafía **23**
6.1 Conversaciones del foro de supabase en reddit **23**
6.2 Vídeos de YouTube **23**
6.3 Documentación de tecnologías **23**

1. Introducción

FinanzApp es una aplicación web de gestión financiera personal diseñada para ofrecer a los usuarios un control sobre sus finanzas domésticas.

Su finalidad es proporcionar una herramienta completa que permita la administración de múltiples cuentas bancarias, la categorización automática de transacciones, el establecimiento de presupuestos y la generación de análisis financieros detallados.

A continuación expondremos qué objetivos se pretenden alcanzar con esta aplicación y por qué:

- El objetivo principal era desarrollar una aplicación de gestión de finanzas que fuera simple a la hora de usarla para anotar movimientos, pero que a la vez, gracias al aporte de la información del usuario, se pudieran crear análisis y estadísticas avanzadas para dar una visión clara de patrones de gasto y tendencias financieras. Esta dualidad entre simplicidad de uso y profundidad analítica es lo que creo que le da un gran valor a la aplicación.
- Definir una arquitectura híbrida escalable que combine las ventajas de un Backend-as-a-Service (Supabase) con la flexibilidad de un backend personalizado (Node.js). Esta aproximación permite aprovechar servicios robustos de infraestructura mientras se mantiene control total sobre la lógica de negocio específica.
- Ofrecer una experiencia de usuario de calidad mediante la implementación de interfaces intuitivas, diseño responsivo y funcionalidades como el modo privacidad para ocultar saldos. La aplicación se adapta completamente a dispositivos móviles y de escritorio, garantizando accesibilidad y usabilidad en cualquier contexto de uso.

2. Arquitectura de la aplicación

2.1 Introducción

En cuanto a las herramientas empleadas para el desarrollo del proyecto se han empleado las siguientes:

- Un equipo con el sistema operativo de Windows 11.
- Los entornos de desarrollo:
  - Visual Studio Code > v1.84.0
  - WebStorm > v2025.1.1
- Aplicaciones de escritorio:
  - Git > v2.49.0
  - Node.js > v22.16.0
  - npm > v9.6.7
- Aplicaciones en la nube:
  - Supabase (Base de datos y autenticación)
  - Vercel (Despliegue frontend)
  - Railway (Despliegue backend)
  - GitHub (Control de versiones)
  - Notion (Gestión de proyecto)
  - Resend (SMTP para envío de emails)
  - Ionos (dominio personalizado)
- Aplicaciones en consola de comandos:
  - Vite > v6.3.5
  - React > v19
  - Express > v4.21.2

2.2 Backend

Se han tomado como referencia diversos conceptos a la hora de implementar su arquitectura. Partiendo del precepto de separar en capas la gestión de los procesos que vinculan a la parte cliente y servidor, se ha tenido en cuenta:

- Para la estructuración del código se ha optado por implementar una **arquitectura en capas** organizando la lógica en controladores (manejo de peticiones HTTP), servicios (lógica de negocio) y middlewares (funciones transversales) claramente diferenciados, siguiendo los principios de separación de responsabilidades.

- Para la comunicación entre servidor y cliente de nuestra API se han tomado los límites de arquitectura definidos por el estilo REST (Representational State Transfer), implementando endpoints que siguen las convenciones HTTP

estándar.

- Para la autenticación y autorización se ha implementado un sistema basado en JSON Web Tokens (JWT) junto con Supabase Auth, que ofrece un estándar de creación de tokens para el intercambio seguro de información entre el navegador y el servidor. Entre sus ventajas: permite el transporte de metadatos y no está ligado a estados.

Para la implementación de estos conceptos, se ha utilizado Node.js con el framework Express.js. La utilidad del mismo excede lo dicho hasta el momento. Para mostrar esto, paso a exponer las dependencias implementadas en este proyecto:

Dependencias principales:

- **Express** : Framework web minimalista para Node.js que facilita la creación de APIs RESTful y el manejo de rutas HTTP.

- **Supabase Client**: Cliente oficial de Supabase para la gestión de base de datos PostgreSQL, autenticación y almacenamiento.

Utilidades y middleware:

- **morgan**: Middleware de logging HTTP para monitoreo de peticiones.

- **multer**: Middleware para el manejo de uploads de archivos multipart/form-data.

Automatización:

- **node-cron**: Scheduler para tareas programadas como limpieza de logs y actualización de presupuestos.

2.3 Frontend

Para el desarrollo de la interfaz de usuario, se ha trabajado con los siguientes lenguajes y tecnologías web:

- **JavaScript ES6+** (con sintaxis JSX)
- **HTML5**
- **CSS3** (Mediante el framework Tailwind)

Por otro lado, se ha implementado el framework **React** en su versión 19, junto con Vite como herramienta de construcción y desarrollo, acompañado de las siguientes librerías y dependencias principales:

- **React Router DOM**: Gestión del enrutamiento SPA y navegación entre páginas

- **React Query**: Manejo del estado del servidor, caché inteligente y sincronización de datos
- **Tailwind CSS**: Framework de utilidades CSS para diseño responsivo y consistente
- **Lucide React**: Librería de iconos SVG optimizados y escalables
- **React Hot Toast**: Sistema ligero de notificaciones toast
- **ExcelJS**: Generación y manipulación de archivos Excel
- **Recharts**: Librería para renderización de gráficas basado en componentes React

La estructura del frontend sigue una arquitectura por capas, partiendo de la **estructura base generada por Vite** y extendida con una **separación de responsabilidades** bien definida:

Para la estructuración del código se ha optado por implementar una arquitectura por capas organizando la lógica en **contextos** (estado global), **hooks** (lógica de datos con React Query), **servicios API** (comunicación con backend), **componentes** (interfaz reutilizable) y **utilidades/utils**, siguiendo los principios de separación de responsabilidades y reutilización de código.

3. Documentación técnica

3.1 Análisis

A continuación exponemos los requisitos funcionales y no funcionales implementados adecuadamente en la aplicación[3]. Para organizar el contenido, tomamos como referencia los roles existentes en la aplicación.

3.1.1 Requisitos funcionales

Usuario anónimo

1. Acceder a la página principal de la aplicación.
2. Acceder al formulario de registro de usuario.
3. Crear perfil como usuario con rol USUARIO.
4. Confirmar registro mediante correo electrónico
5. Iniciar sesión
6. Recuperar contraseña mediante correo electrónico

Rol USUARIO

1. Gestión de cuentas bancarias
    a. Corriente, Ahorro, Crédito e Inversión.
    b. Visualización de listado de cuentas con fecha de última actualización y ordenadas por balance total.
    c. Edición de datos de cuenta: sólo nombre por motivos de consistencia.

   d. Eliminación de cuentas con todas sus transacciones asociadas.
2. Gestión de transacciones/movimientos:
   a. Tres tipos de operación sobre una cuenta: ingreso, gasto y transferencia
   b. Registro de ingresos o gastos con categorización y posibilidad de rectificación en caso de error.
   c. Edición de datos de la transacción: descripción y categoría (con restricciones)
   d. Creación de transferencias entre cuentas propias.
   e. Visualización de movimientos ordenados por orden cronológico y filtrado avanzado por fecha, tipo, categoría, búsqueda textual y cuenta.
3. Sistema de presupuestos:
   a. Creación de presupuestos por categorías de gasto.
   b. Definición de fecha final del presupuesto y monto presupuestado
   c. Seguimiento automático del progreso con una barra de progreso por porcentaje.
   d. Gestión de presupuestos activos y expirados.

4. Análisis financiero:
   a. Gráfica con resumen de ingresos vs gastos en diferentes rangos de fecha predefinidos.
   b. Gráficos de distribución de gastos por categorías.
   c. Alertas inteligentes para notificar del comportamiento de los gastos del usuario.

Rol ADMIN

1. Gestión de usuarios:
   a. Visualización de listado completo de usuarios registrados.
   b. Acceso al perfil detallado de cada usuario juntos con sus cuentas y presupuestos.
   c. Activación/desactivación de cuentas de usuario.
   d. Monitoreo de último acceso y actividad del usuario.

2. Estadísticas del sistema:
   a. Métricas de usuarios activos mensualmente.
   b. Análisis de crecimiento de transacciones.
   c. Indicadores de salud del sistema.
   d. Actividad reciente de la plataforma.

Usuario autenticado (Ambos roles)

1. Opciones de configuración:
   a. Actualización de datos personales.
   b. Cambio de contraseña con verificación.
   c. Gestión de avatar (subida, actualización, eliminación).
   d. Modo privacidad para ocultar/mostrar saldos.

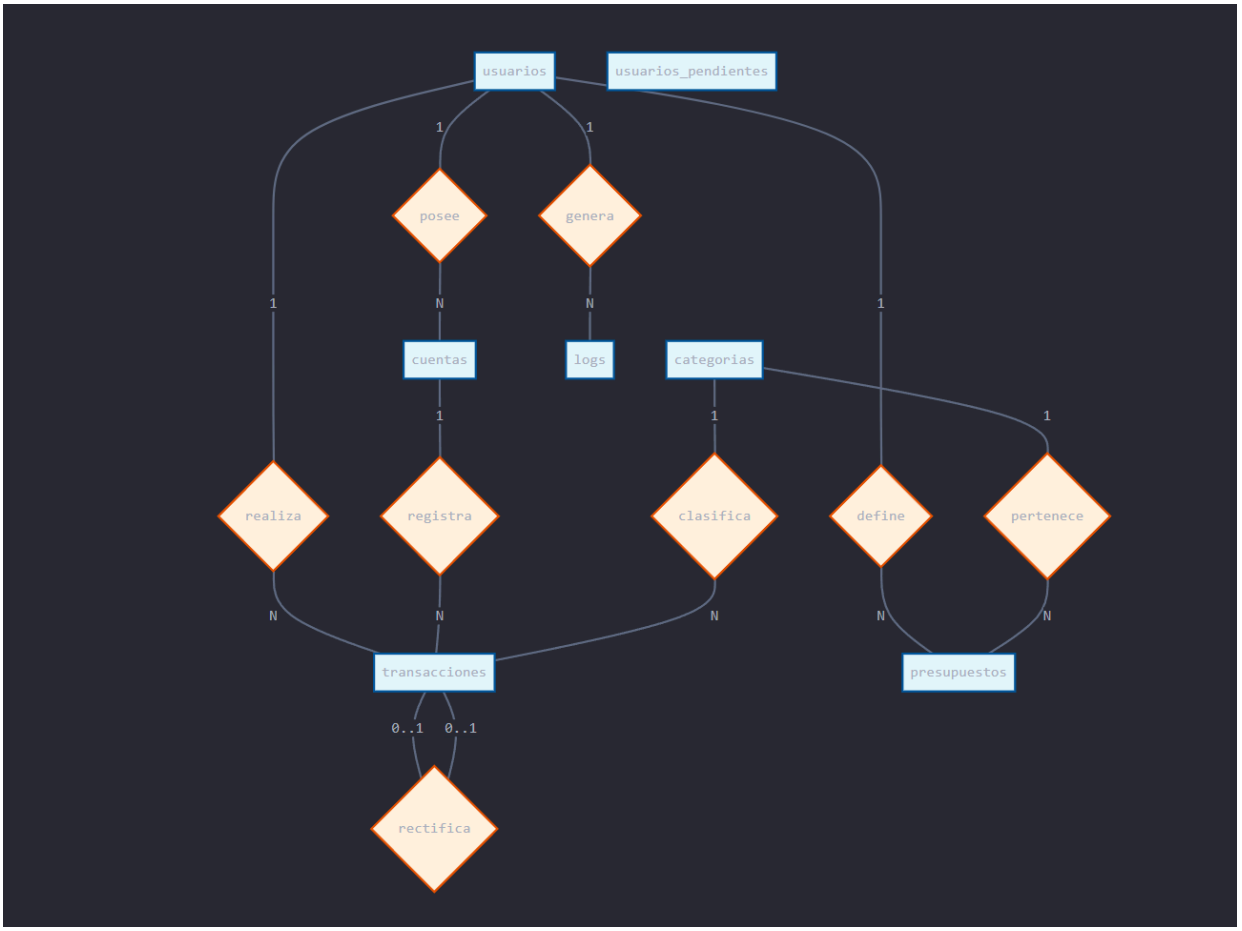
3.1.2 Requisitos no funcionales

1. **Diseño responsivo**: Interfaz adaptable a dispositivos móviles, tablets y escritorio.
2. **Seguridad robusta**: Autenticación JWT, validación de datos tanto en frontend como en el servicio Supabase, y control de acceso por roles.
3. **Rendimiento optimizado**: Carga diferida de transacciones, caché inteligente con React Query y uso de 0 imágenes, sólo SVGs optimizados.

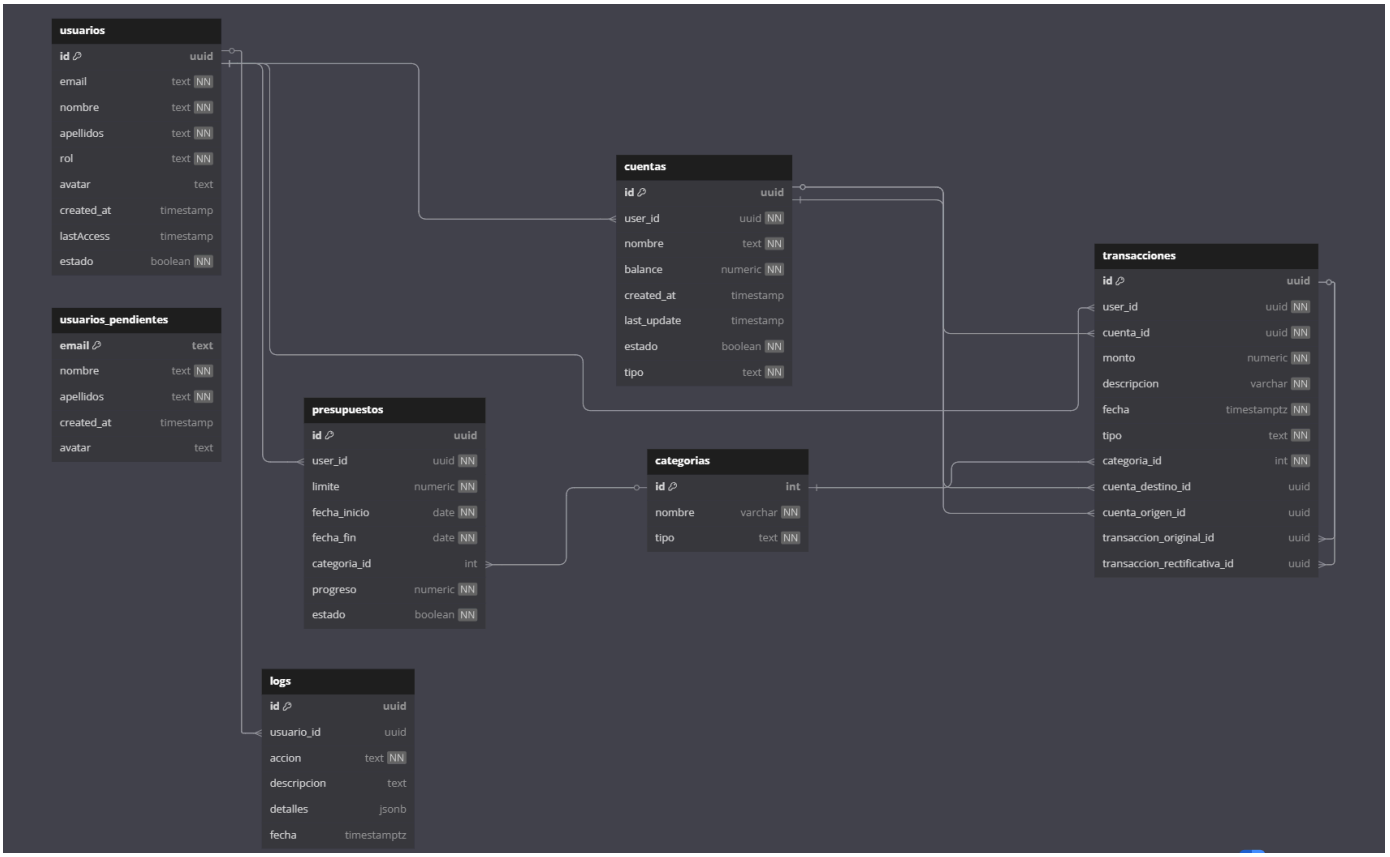
3.2 Desarrollo

3.2.1 Diseño de la base de datos

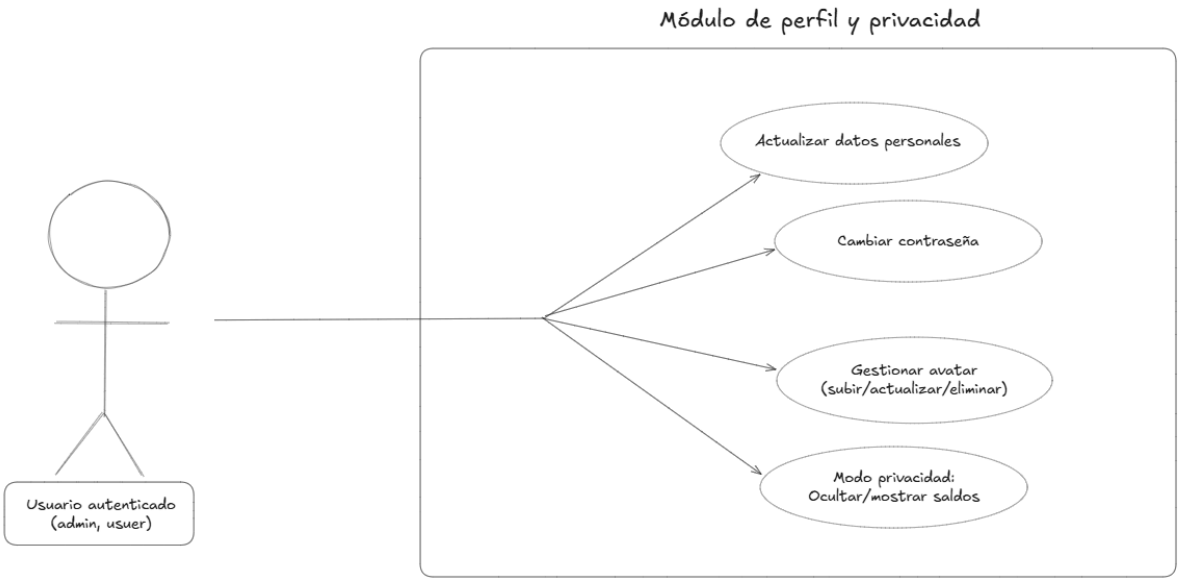
Modelo entidad relación de la bbdd:

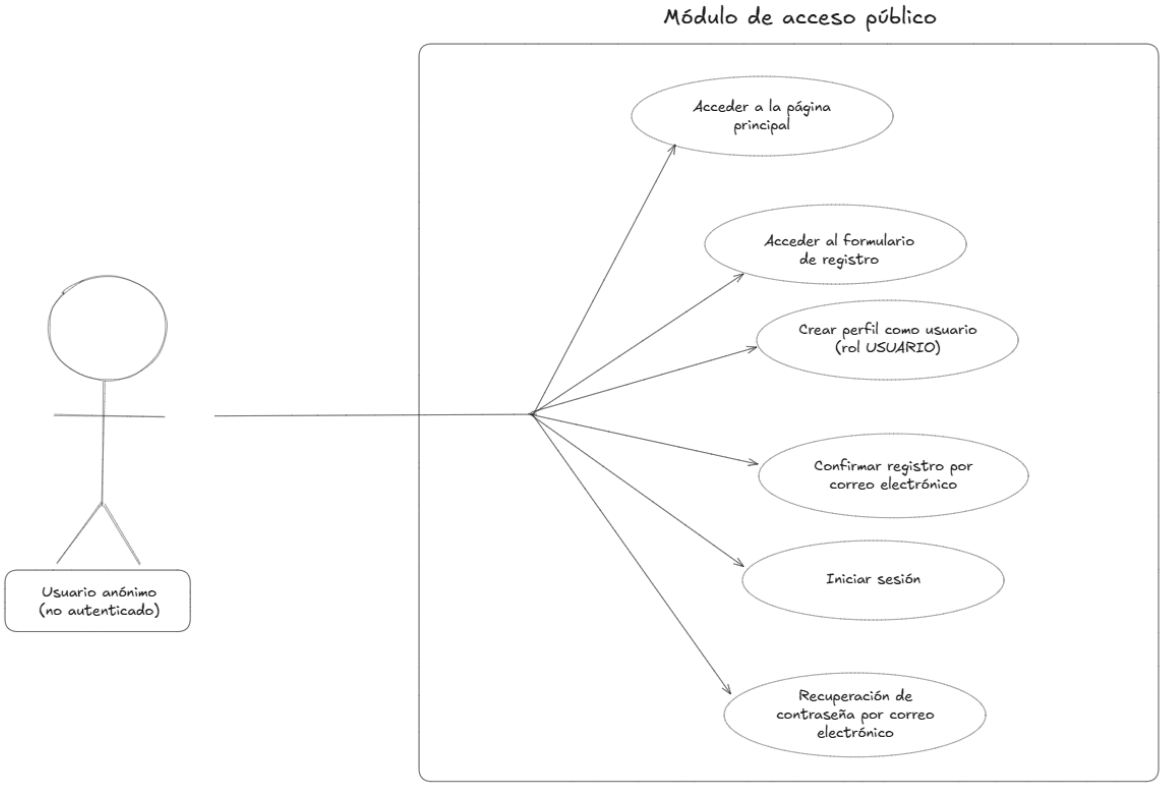


Modelo relacional de la bbdd:

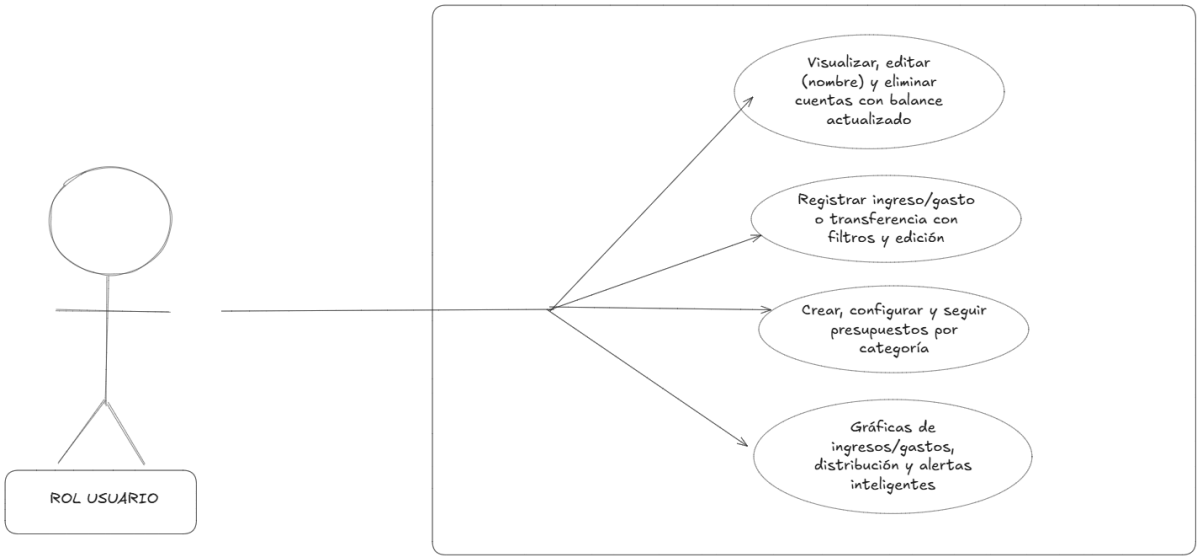


Diagramas de casos de uso:

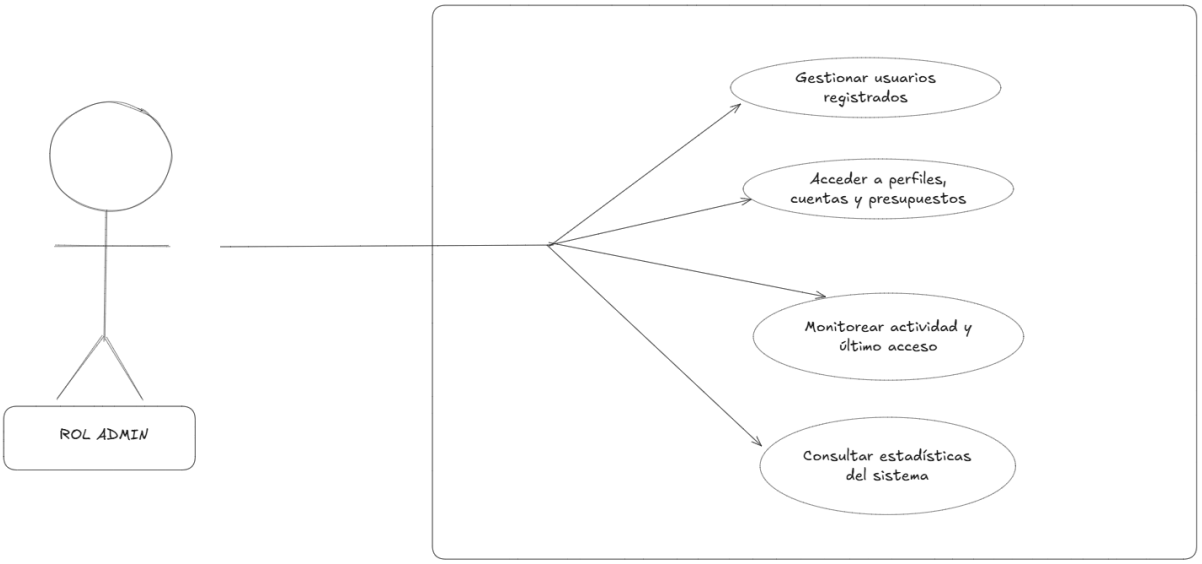

Módulo de perfil y privacidad
Usuario autenticado
(admin, usuer)
Actualizar datos personales
Cambiar contraseña
Gestionar avatar
(subir/actualizar/eliminar)
Modo privacidad:
Ocultar/mostrar saldos


Módulo de acceso público
Usuario anónimo
(no autenticado)
Acceder a la página
principal
Acceder al formulario
de registro
Crear perfil como usuario
(rol USUARIO)
Confirmar registro por
correo electrónico
Iniciar sesión
Recuperación de
contraseña por correo
electrónico

Módulo de gestión financiera



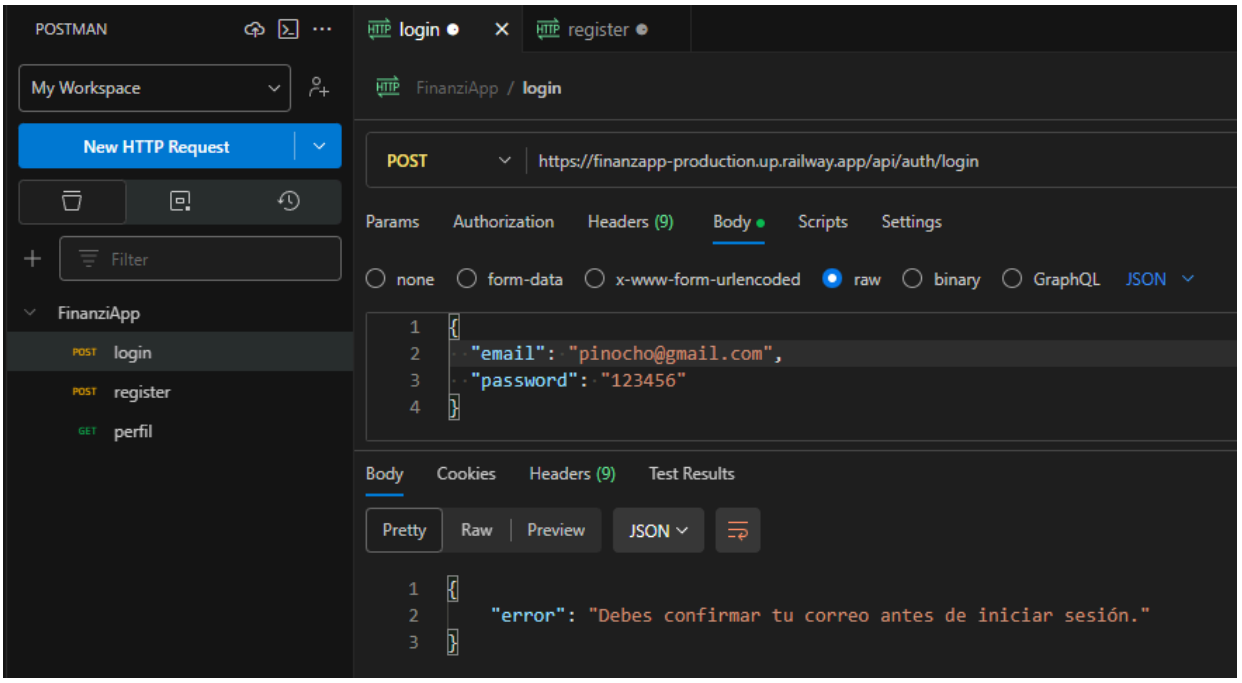
Módulo de administración



3.3 Pruebas realizadas

Para garantizar el correcto funcionamiento y la seguridad de la aplicación, se han llevado a cabo diferentes tipos de pruebas centradas en la validación de funcionalidades y la verificación de los controles de acceso.

Pruebas de Seguridad con Postman
Se utilizó la extensión de Postman para Visual Studio Code para realizar pruebas exhaustivas de seguridad del backend, enfocándose en la verificación de los controles de acceso y autenticación.



Pruebas Funcionales y de Usabilidad

Para el resto de pruebas se adoptó un enfoque de testing manual exhaustivo desde la interfaz de usuario.

Pruebas de flujo completo: Se simularon todos los posibles recorridos que un usuario podría realizar en la aplicación, desde el registro hasta la exportación de datos, incluyendo escenarios de uso típicos y casos extremos.

Validación de formularios: Se probaron todas las validaciones implementadas introduciendo datos incorrectos, campos vacíos y valores límite para verificar que los esquemas Zod funcionan correctamente tanto en frontend como backend.

Pruebas de integración: Se verificó la comunicación correcta entre frontend y backend, así como la sincronización de datos con React Query y la persistencia en Supabase.

Depuración de código: Durante todo el proceso se utilizaron las herramientas de

desarrollo del navegador y el debugger de VS Code para identificar y corregir errores en tiempo de ejecución, optimizar consultas y mejorar el rendimiento general de la aplicación.

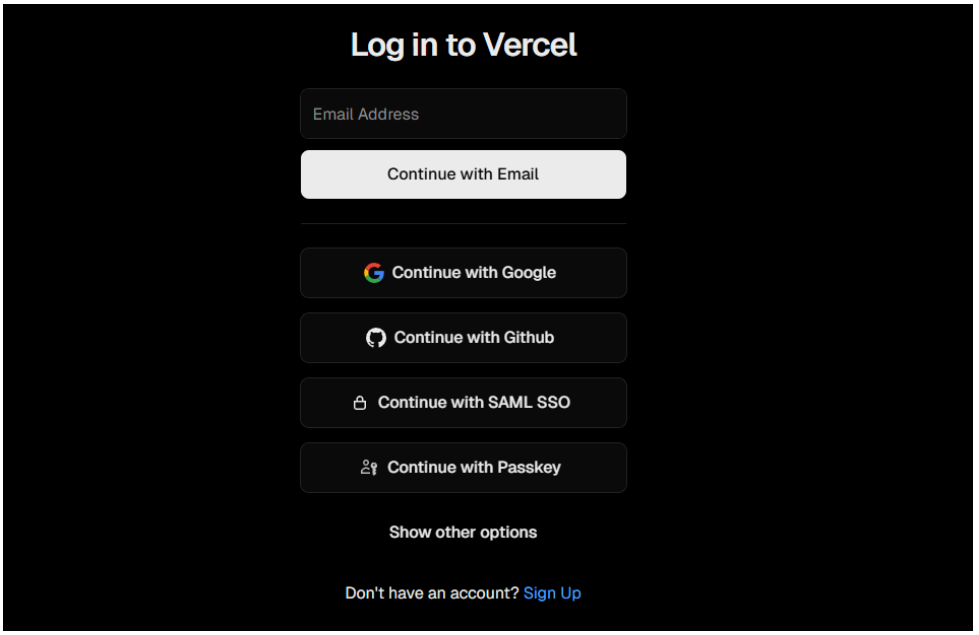
4. Proceso de despliegue

Para el despliegue completo de la aplicación, opté por una arquitectura basada en servicios especializados:

- Frontend: se aloja en **Vercel**, aprovechando su integración nativa con frameworks modernos y su despliegue continuo desde el repositorio.
- Backend: está desplegado en **Railway**, que facilita la gestión del entorno Node.js y permite una integración ágil con bases de datos y otros servicios.
- Base de datos: como ya mencioné, utilizo **Supabase**, que ofrece una solución completa basada en PostgreSQL, con autenticación integrada y una API REST generada automáticamente.
- Servicio de correo electrónico: a través de **Resend**, vinculado directamente con Supabase para gestionar el envío de correos de verificación y recuperación de contraseña.

4.1 Despliegue cliente

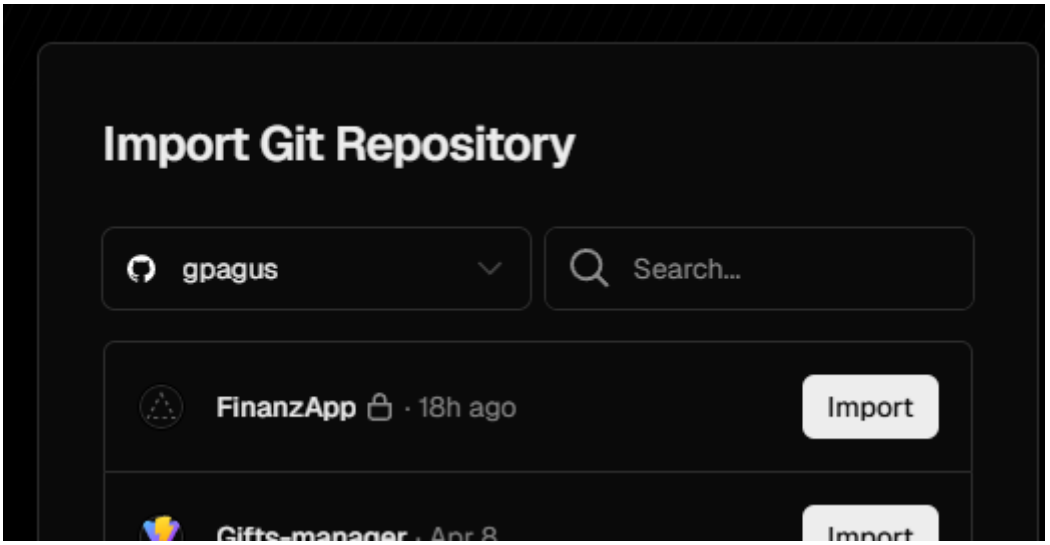
1. Creamos una cuenta en vercel, preferiblemente con github para ya tener acceso desde vercel a nuestros repositorios directamente.



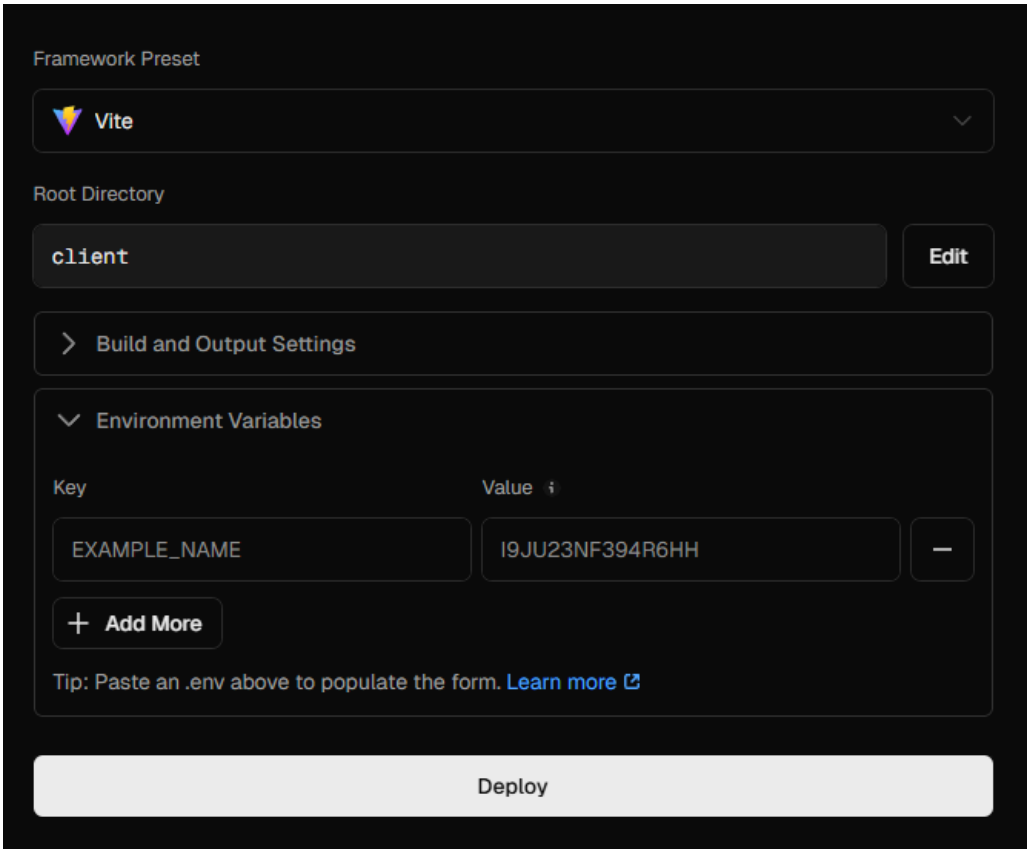
2. Una vez accedido, desde la página de overview hacemos clic en add new > new

project

3. Seleccionamos el repositorio de la aplicación



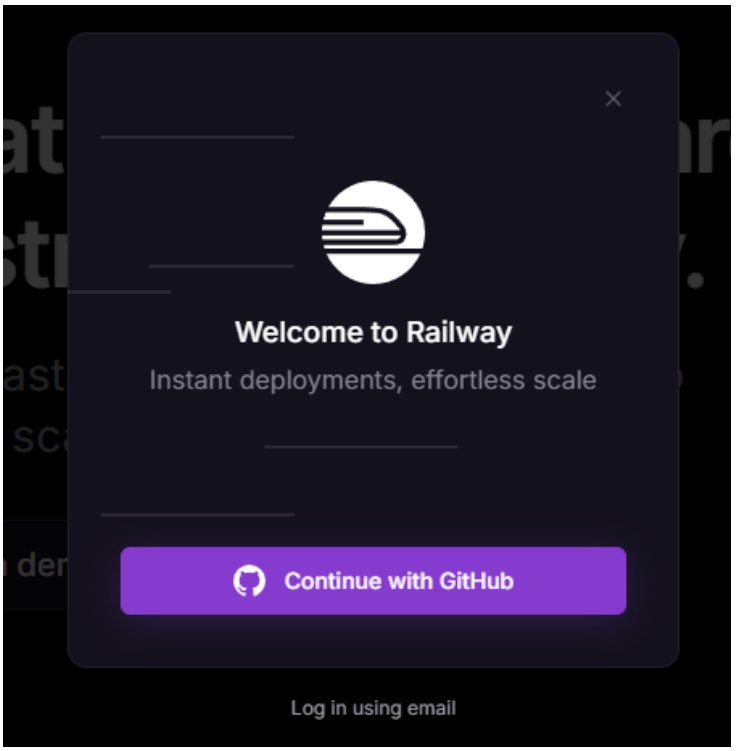
4. en ‘root directory’ seleccionamos la carpeta del entorno cliente, (en mi caso se trata de un proyecto vite+react jsx) y más abajo en ‘Enviroment variables’ pondremos nuestras variables de entorno, que en local tendriamos que tener en un fichero .env.



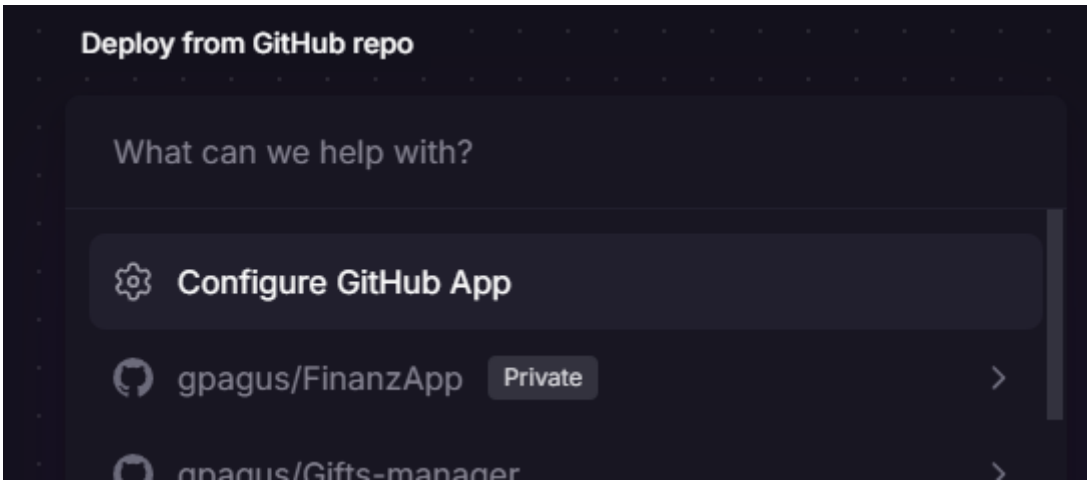
Hecho esto le daremos a ‘Deploy’ y ya tendremos el cliente desplegado.

4.2 Despliegue servidor

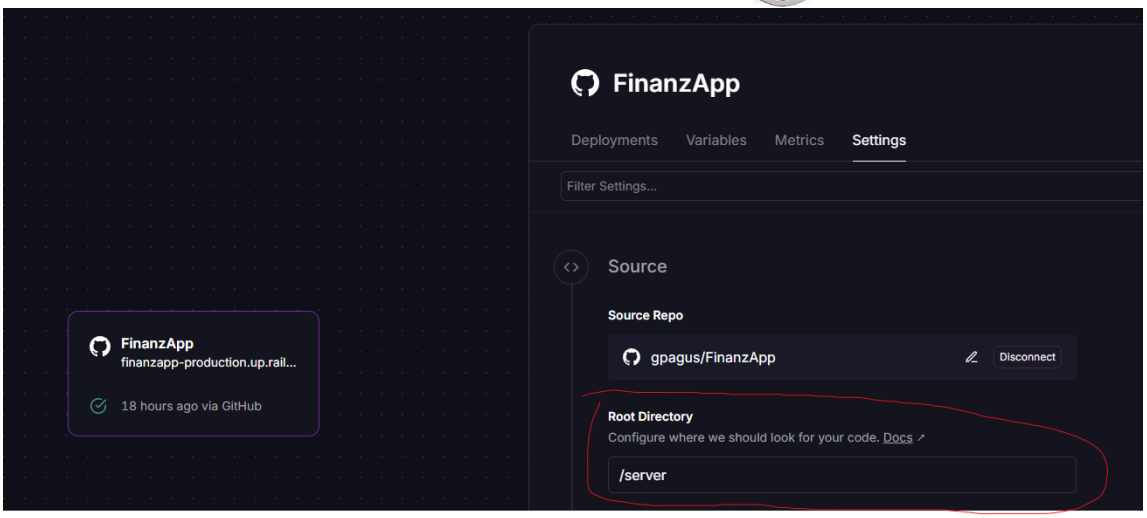
1. Nos creamos una cuenta en **Railway** con nuestra cuenta de GitHub



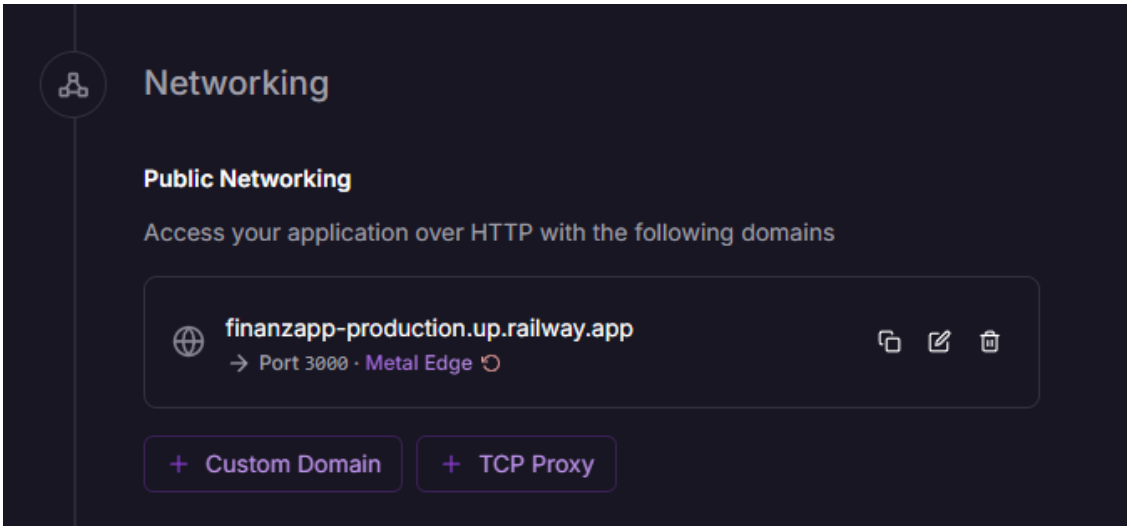
2. Una vez accedido hacemos clic en new > Deploy from GitHub repo y seleccionamos el repositorio



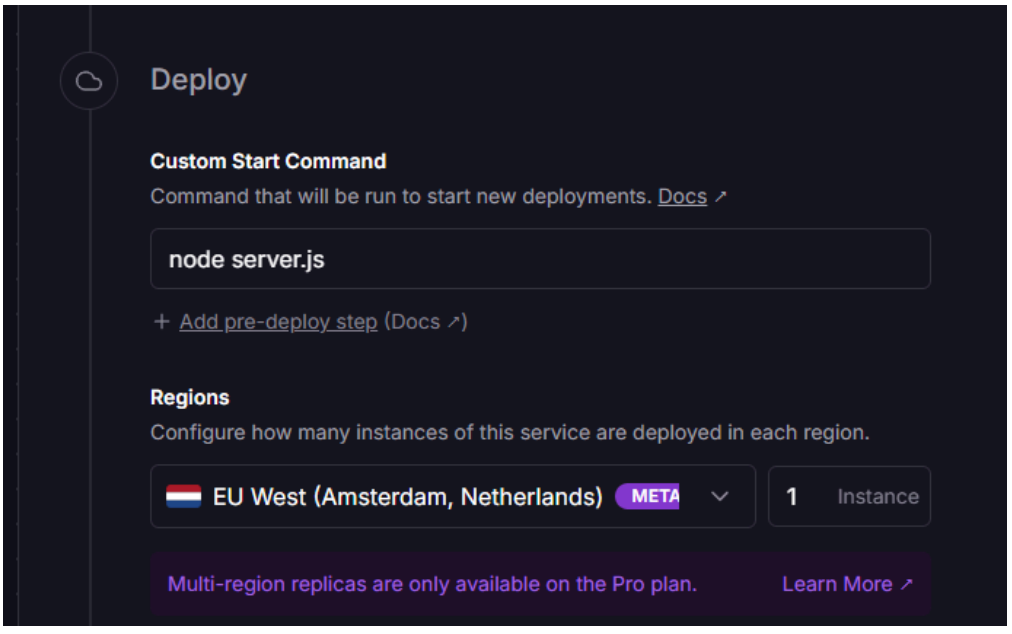
3. Hecho esto se nos creará el proyecto en el workspace el cual deberemos configurar antes de desplegarlo:
4. Primero seleccionamos la tarjeta de proyecto y en ‘settings’ ponemos que el directorio sea la carpeta del servidor



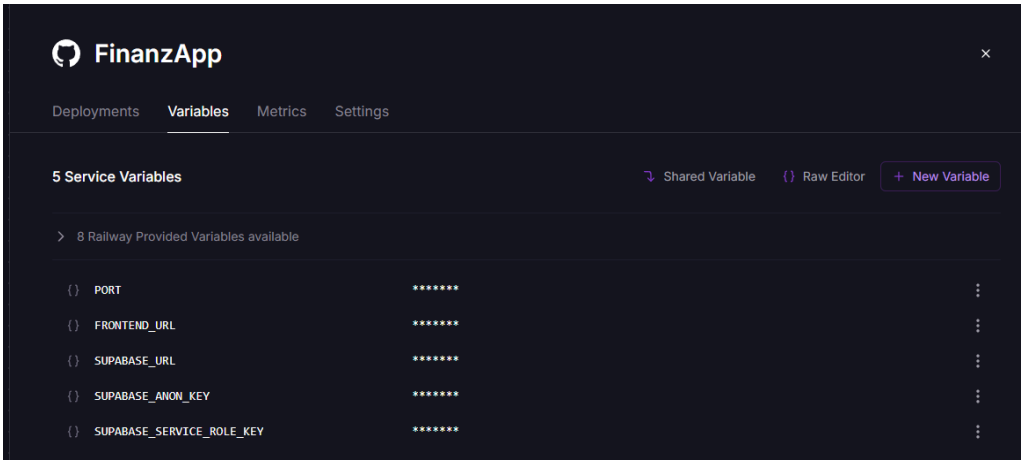
5. En la sección de networking, generamos el dominio que será el de la API



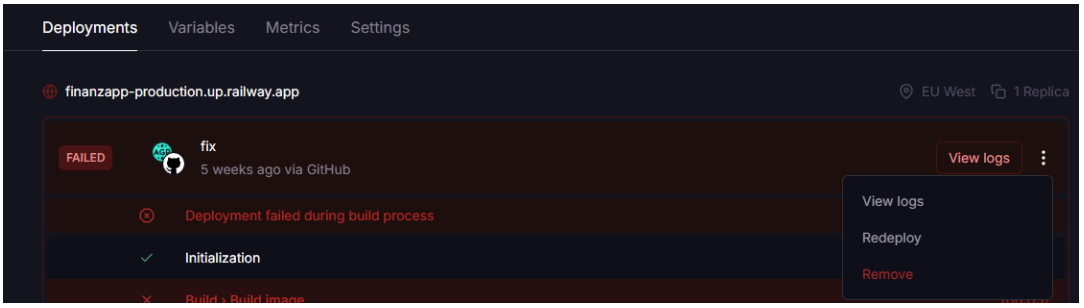
6. En la sección ‘deploy’ seleccionamos la región más cercana de las que hay, donde se aloja el servidor y escribimos el comando de node para arrancar el servidor



7. En la sección ‘variables’ definidos las variables de entorno, para que el servidor contacte tanto con el cliente en vercel como la bbdd en supabase

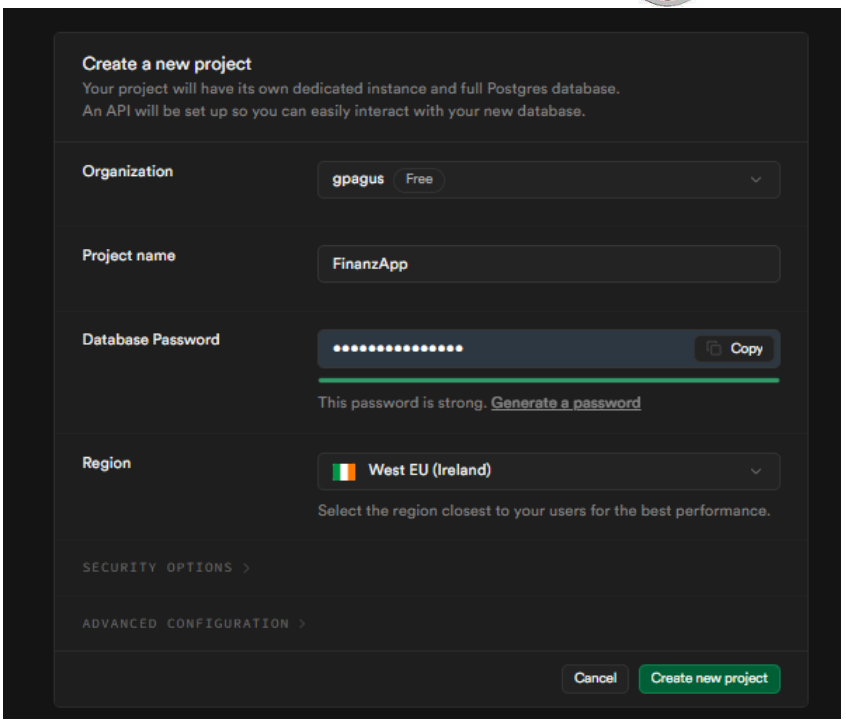


8. Por último sólo nos quedará hacer un redeploy del despliegue que se hizo automáticamente, el cual lógicamente falló porque no se configuró nada para su correcto despliegue.



4.3 Despliegue BBDD + configuración Resend

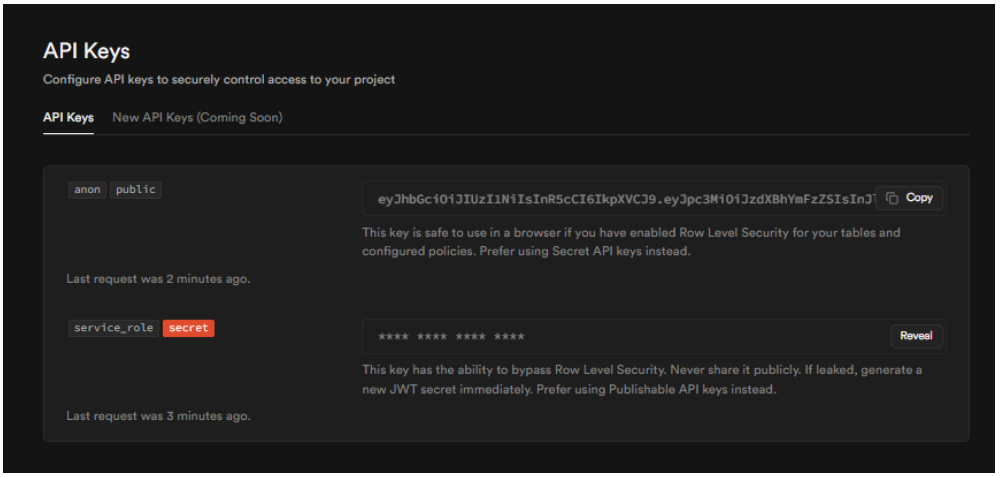
1. Nos creamos una cuenta en supabase y procedemos a crearnos un nuevo



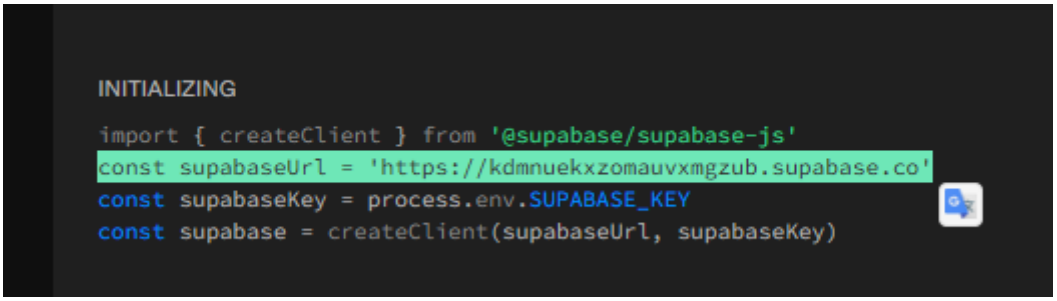
proyecto

Con simplemente crear el proyecto, tendriamos desplegada la base de datos.

2. Para acceder desde la aplicación a la base de datos, necesitaremos las API keys para las peticiones, que encontraremos en ‘project settings’ > ‘API keys’



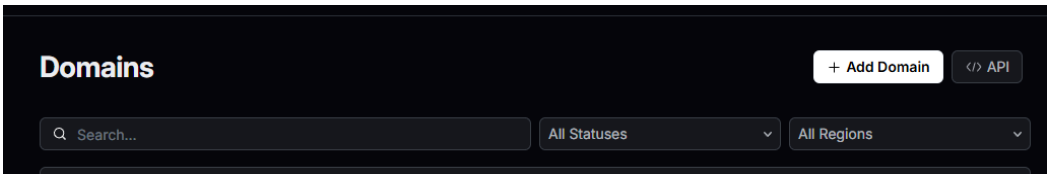
3. Y también la url de supabase que se encuentra en ‘API Docs’



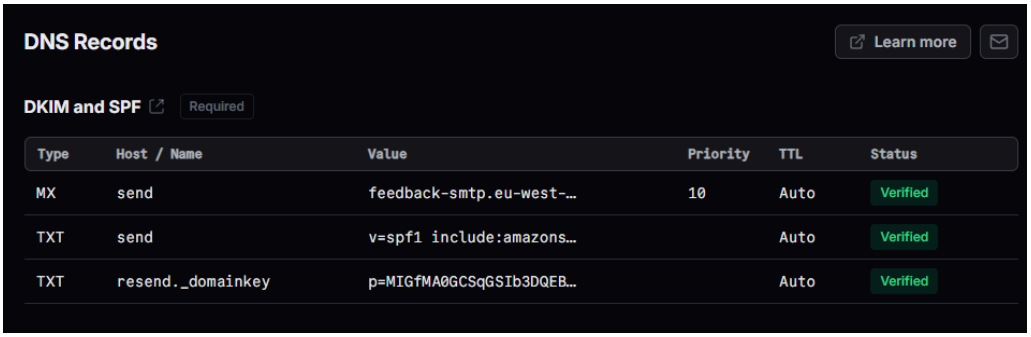
Ahora vamos paso a paso con la configuración de servicio de envío de emails, para la cual previamente necesitaremos ser dueños de un dominio (en mi caso compré un dominio con el **ionos**)

1. Nos creamos una cuenta en **Resend** y en la sección ‘Domains’ del sidebar

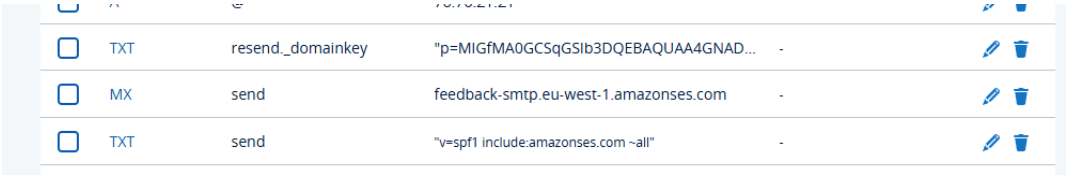
hacemos clic en ‘new domain’



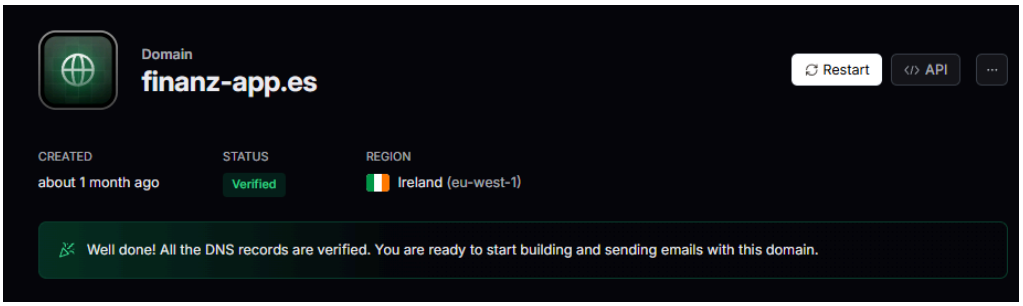
2. Nos mostrará un formulario donde tendremos que escribir nuestro dominio y una serie de DNS records que deberemos registrar en el panel de configuración de nuestro dominio para verificar que somos dueños.



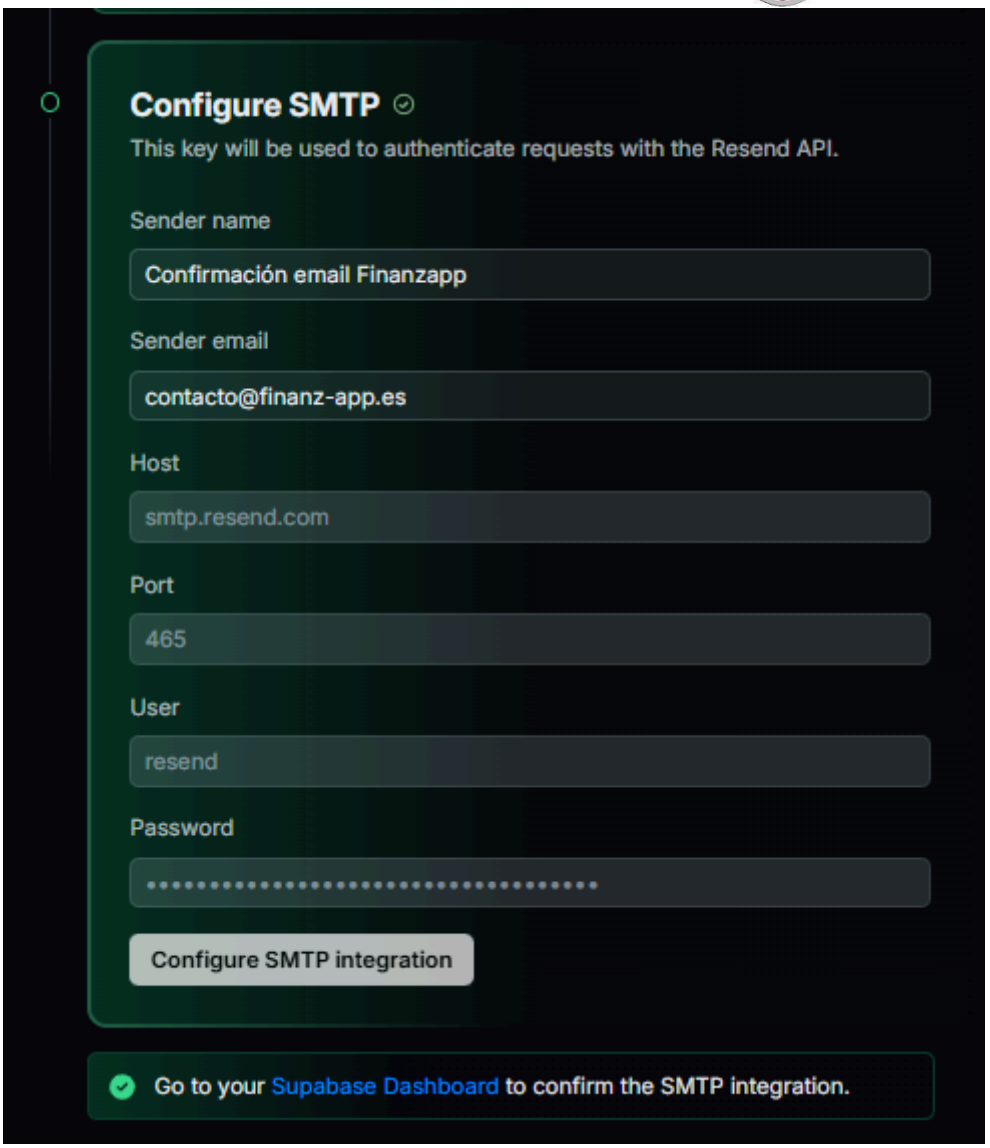
y aquí los dns configurados en el panel de DNS de ionos:



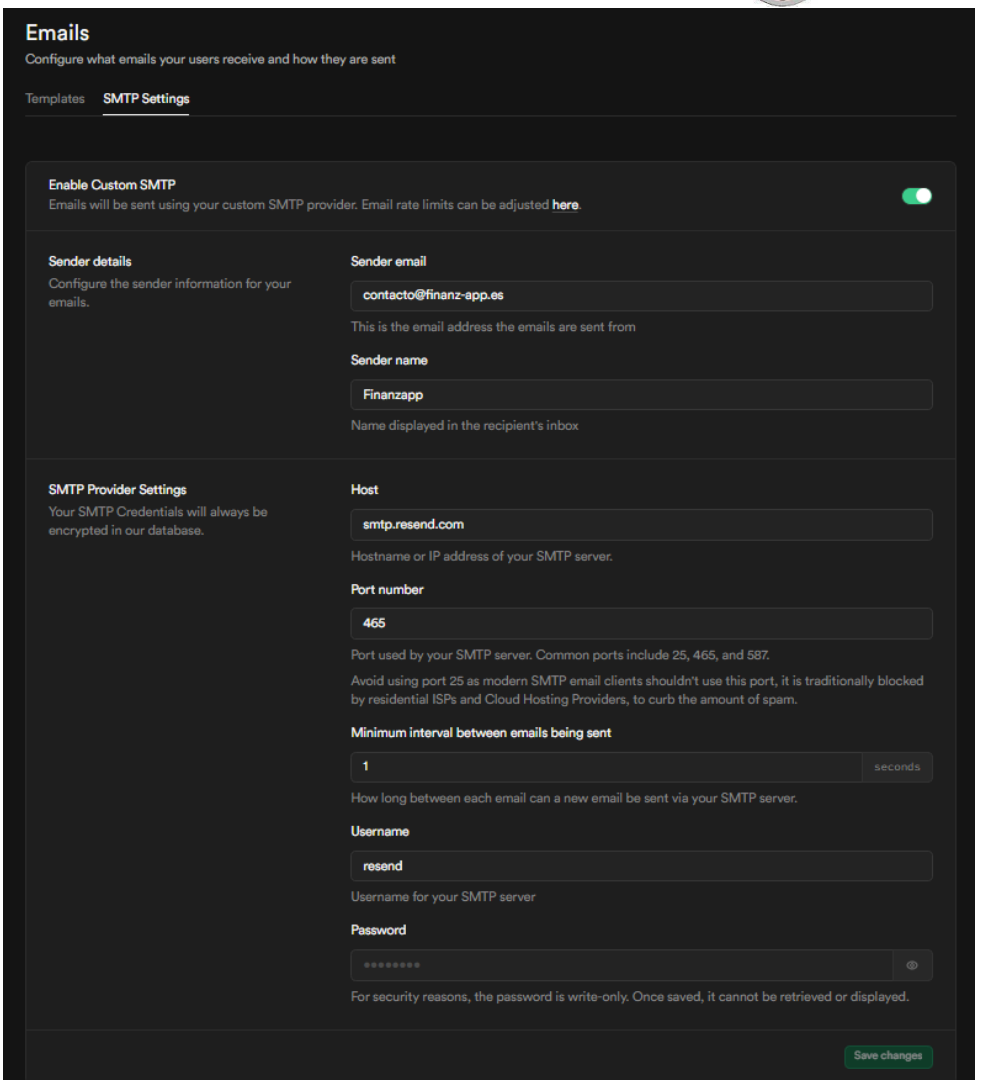
Hecho esto le daremos click a ‘check domains’ y esperaremos unos minutos hasta que acabe la comprobación.



3. Ahora nos iremos a ‘settings’ > ‘integrations’, seleccionamos la integración con supabase y seguiremos los pasos de configuración



4. Una vez completado los formularios confirmamos la integración desde supabase y veremos desde ‘Emails’ > ‘SMTP settings’ como se nos configuró solo el envio de emails de confirmacion usando el servicio personalizado de Resend


Emails
Configure what emails your users receive and how they are sent
Templates
SMTP Settings
Enable Custom SMTP
Emails will be sent using your custom SMTP provider. Email rate limits can be adjusted here.
Sender details
Configure the sender information for your emails.
Sender email
contacto@finanz-app.es
This is the email address the emails are sent from
Sender name
Finanzapp
Name displayed in the recipient's inbox
SMTP Provider Settings
Your SMTP Credentials will always be encrypted in our database.
Host
smtp.resend.com
Hostname or IP address of your SMTP server.
Port number
465
Port used by your SMTP server. Common ports include 25, 465, and 587.
Avoid using port 25 as modern SMTP email clients shouldn't use this port, it is traditionally blocked by residential ISPs and Cloud Hosting Providers, to curb the amount of spam.
Minimum interval between emails being sent
1
seconds
How long between each email can a new email be sent via your SMTP server.
Username
resend
Username for your SMTP server
Password
For security reasons, the password is write-only. Once saved, it cannot be retrieved or displayed.
Save changes

5. Propuestas de mejora

5.1 Requisitos funcionales pendientes

Sistema de Suscripciones y Gastos Recurrentes
Se planteó implementar un sistema de gestión de gastos recurrentes que permita a los usuarios registrar transacciones que se repiten de manera periódica (semanal, mensual, anual).

Evolución hacia un Sistema Colaborativo (Estilo Splitwise)

La aplicación está diseñada para evolucionar hacia una plataforma colaborativa donde múltiples usuarios puedan interactuar y compartir gastos

5.2 Requisitos técnicos

5.2.1 Backend

Reestructuración del Sistema de Autenticación
La principal mejora técnica identificada en el backend es la reestructuración completa del sistema de autenticación. Actualmente, al manejar la autenticación a través del backend personalizado en lugar de utilizar directamente el cliente de Supabase en el frontend, se perdieron funcionalidades automáticas del servicio, por lo que tuve que hacer una gestión manual de tokens JWT y refresh tokens, con un middleware personalizado y lógica de renovación de sesión implementada manualmente.

5.2.1 Frontend

Mejora en la Componentización
En ciertas vistas de la aplicación hay repetición de código debido a una componentización insuficiente, lo cual va en contra de uno de los principios fundamentales de React.

Reorganización de la Estructura del Proyecto
La estructura actual, aunque funcional, podría beneficiarse de una organización más ramificada con directorios.

6. Webgrafía

6.1 Conversaciones del foro de supabase en reddit

- ¿Cómo detectar si un email de Supabase ya existe pero aún no se ha confirmado?
- Auth emails : r/Supabase
- smtp service recommendation : r/Supabase
- Integrating Supabase Auth with custom Node.js API

6.2 Vídeos de YouTube

- 12. Integración de Resend con Supabase para Envio de Email - Curso FlutterFlow …
- APRENDE React Query DESDE CERO: Paginación, Infinite Scroll, DevTools
- Desarrollando una API con Express desde cero
- Caching OG Images with Supabase Storage CDN

6.3 Documentación de tecnologías

- Use Supabase with React
- Auth | Supabase Docs
- Integrating With Supabase Auth
- TanStack Query React Docs
- React Hot Toast
- https://recharts.org/en-US
- Tailwind CSS Cheat Shee