UNIVERSIDAD DEL VALLE DE GUATEMALA

Ingeniería de Software – Sección 21

Ing. Pablo Barreno

Tarea 7 - Integración Continua

Osman Emanuel de León García – 23428

Milton Giovanni Polanco Serrano - 23471

Gadiel Amir Ocaña Véliz - 231270

Guatemala, 07 de octubre de 2025

INTEGRACIÓN CONTINUA: Pipeline CI/CD con GitHub Actions y Vercel

1. INVESTIGACIÓN SOBRE GITHUB ACTIONS PARA CI/CD

1.1 ¿Qué es GitHub Actions?

GitHub Actions es una plataforma de automatización integrada en GitHub que permite crear workflows personalizados para CI/CD (Integración Continua y Despliegue Continuo). Los workflows se definen mediante archivos YAML ubicados en el directorio .github/workflows/ del repositorio.

1.2 Componentes Fundamentales

- **Workflows:** Procesos automatizados configurables que se ejecutan en respuesta a eventos específicos (push, pull request, schedule, etc.)
- **Jobs:** Conjunto de pasos que se ejecutan en el mismo runner. Un workflow puede contener múltiples jobs que se ejecutan en paralelo o secuencialmente.
- **Steps:** Acciones individuales dentro de un job. Pueden ser comandos de shell o actions reutilizables.
- **Runners:** Máquinas virtuales (Ubuntu, Windows, macOS) donde se ejecutan los workflows.
- **Secrets:** Variables de entorno encriptadas para almacenar información sensible (API keys, tokens, credenciales).

1.3 Integración con Vercel

Para nuestro proyecto, utilizamos la integración de GitHub Actions con Vercel CLI, que permite:

- Deployments automáticos por branch
- Preview deployments para cada commit
- Gestión de variables de entorno por ambiente
- Despliegue de aplicaciones serverless (Frontend y Backend)

2. DESCRIPCIÓN DE LAS ETAPAS DEL PIPELINE CI/CD

2.1 Arquitectura del Sistema

Nuestro proyecto MoneyFlow utiliza una arquitectura de microservicios con:

- Frontend: React + TypeScript + Vite
- Backend: Node.js + Express + Supabase
- Deploy: Repositorio principal con submodules

2.2 Ambientes Implementados

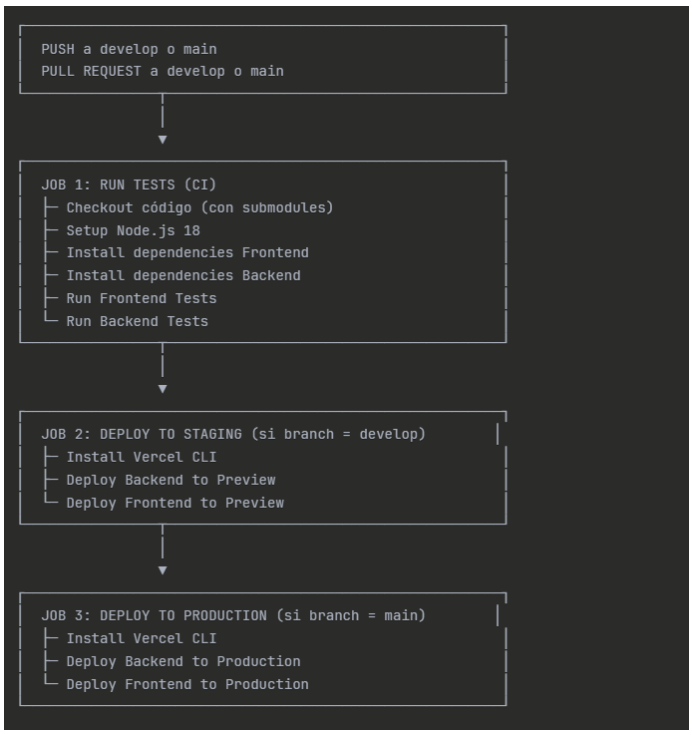
Ambiente de Staging (Preview/Develop)

- Branch: *develop*
- Frontend URL: https://moneyflow-frontend-lhie7ntxw-emadlgs-projects.vercel.app/
- Backend URL: https://moneyflow-backend-i9txmh185-emadlgs-projects.vercel.app/
- Propósito: Testing y validación de nuevas funcionalidades antes de producción

Ambiente de Producción

- Branch: *main*
- Frontend URL: https://moneyflow-frontend-five.vercel.app/
- Backend URL: https://moneyflow-backend.vercel.app/
- Propósito: Versión estable accesible a usuarios finales

2.3 Flujo del Pipeline



2.4 Etapas Detalladas

Etapa 1: Checkout del Código

- Se clona el repositorio Deploy que contiene los submodules de Frontend y Backend
- Se actualizan los submodules recursivamente para obtener el código más reciente

Etapa 2: Configuración del Entorno

- Instalación de Node.js versión 18
- Configuración de caché para node_modules (optimiza tiempos)

Etapa 3: Instalación de Dependencias

- Frontend: npm install en MoneyFlow_Frontend
- Backend: npm install en MoneyFlow_Backend

Etapa 4: Ejecución de Tests (CI)

- Tests del Frontend: Validación de componentes React
- Tests del Backend: Validación de rutas API y lógica de negocio
- Si algún test falla, el pipeline se detiene

Etapa 5: Deploy Condicional

- Si es *develop* → Deploy a Staging con Vercel CLI (preview)
- Si es *main* → Deploy a Production con Vercel CLI (--prod)

Etapa 6: Verificación Post-Deploy

- Health check del Backend
- Validación de Frontend (carga correctamente)

3. CONFIGURACIÓN DEL PIPELINE DE INTEGRACIÓN CONTINUA

3.1 Ubicación y Estructura del Pipeline

El pipeline de integración continua se encuentra configurado en el archivo deploy.yml ubicado en .github/workflows/ del repositorio MoneyFlow_Deploy. Este archivo define el flujo automatizado que se ejecuta cada vez que hay un push o pull request hacia las ramas main o develop.

3.2 Proceso de Testing

El job de testing se ejecuta en un runner de Ubuntu con Node.js 18. Primero hace checkout del código incluyendo los submodules de Frontend y Backend, luego instala las dependencias de ambos proyectos con npm ci, y finalmente ejecuta las pruebas de ambos. Si cualquier prueba falla, el pipeline se detiene inmediatamente y no se realiza ningún deploy, protegiendo así los ambientes de código defectuoso.

Para el Frontend se utilizan las pruebas de React Testing Library que validan componentes y funcionalidad. Para el Backend se prueban las rutas API y la lógica de negocio. El pipeline implementa caché de dependencias para reducir el tiempo de ejecución en aproximadamente 80%, permitiendo feedback más rápido a los desarrolladores.

4. CONFIGURACIÓN DEL PIPELINE DE DESPLIEGUE CONTINUO

4.1 Estrategia de Deploy Multi-Ambiente

El pipeline asocia automáticamente cada rama con un ambiente: develop se despliega a staging (preview) y main a producción. Esta separación permite probar cambios en un entorno controlado antes de llegar a usuarios finales.

4.2 Deploy a Staging

El deploy a staging se ejecuta solo cuando hay cambios en develop y todas las pruebas pasan. El pipeline instala Vercel CLI y despliega Backend y Frontend secuencialmente usando tres comandos: pull (obtener configuración), build (compilar), y deploy (publicar). Cada commit genera una URL única de preview, permitiendo que múltiples features coexistan simultáneamente para pruebas independientes. Las variables de entorno para staging están configuradas en el dashboard de Vercel y apuntan a las URLs de preview y servicios de desarrollo.

4.3 Deploy a Producción

El deploy a producción solo ocurre con cambios en main después de pasar todas las pruebas. Utiliza los mismos comandos de Vercel CLI pero con flags --prod y --prebuilt para optimizar el despliegue. Las variables sensibles (claves de Supabase, JWT secret, API keys) se inyectan desde los secrets de GitHub durante el build. Las URLs de producción son fijas: moneyflow-frontend.vercel.app y moneyflow-backend.vercel.app.

4.4 Gestión de Variables de Entorno

Las variables se gestionan en dos niveles. Las variables públicas del Frontend (prefijo VITE_) están en el dashboard de Vercel e incluyen URLs del Backend y credenciales públicas de Supabase. Las variables sensibles del Backend están como secrets en GitHub y se inyectan durante el build, incluyendo service role key de Supabase, JWT secret, y API key de Gemini. Para staging, las variables apuntan a URLs de preview; para producción, a URLs estables. Esta configuración garantiza aislamiento completo entre ambientes.

4.5 Secrets de GitHub

Los secrets actúan como puente seguro entre el pipeline y servicios externos. VERCEL_TOKEN permite autenticación con Vercel, los IDs de organización y proyecto identifican dónde desplegar, y GH_PAT permite acceso a los submodules privados. Los secrets de Supabase, JWT y Gemini se inyectan en el build del Backend. Todos están configurados en Settings → Secrets del repo Deploy y GitHub los enmascara automáticamente en logs para seguridad.

5. DECISIONES TÉCNICAS Y JUSTIFICACIÓN

5.1 GitHub Actions como Plataforma CI/CD

Elegimos GitHub Actions por su integración nativa con GitHub, eliminando configuraciones externas complejas. Es gratuito para repos públicos, tiene sintaxis YAML simple, y ofrece un ecosistema de acciones reutilizables que aceleró el desarrollo. Acciones como checkout y setup-node son mantenidas por GitHub y probadas por la comunidad, reduciendo riesgos.

5.2 Vercel como Plataforma de Hosting

Vercel se especializó en frontend y serverless, detectando automáticamente Vite y Node.js sin configuración manual. Los preview deployments generan URLs únicas por commit, permitiendo testing paralelo de múltiples features. Su CDN global distribuye automáticamente assets para mejor rendimiento, y las Serverless Functions eliminan gestión de servidores escalando automáticamente. La integración con tres comandos simples (pull, build, deploy) y la gestión de variables por ambiente desde el dashboard simplificaron el proceso.

5.3 Arquitectura de Submodules

Estructuramos el proyecto con submodules para mantener Frontend y Backend como entidades independientes con historial propio. Esto permite que diferentes miembros trabajen simultáneamente sin conflictos complejos, y cada repo puede clonarse individualmente según necesidad. El repo Deploy orquesta ambos mediante referencias SHA específicas, garantizando deployments reproducibles. Esta estructura también facilita versionado independiente y reutilización futura.

5.4 Estrategia de Dos Ambientes

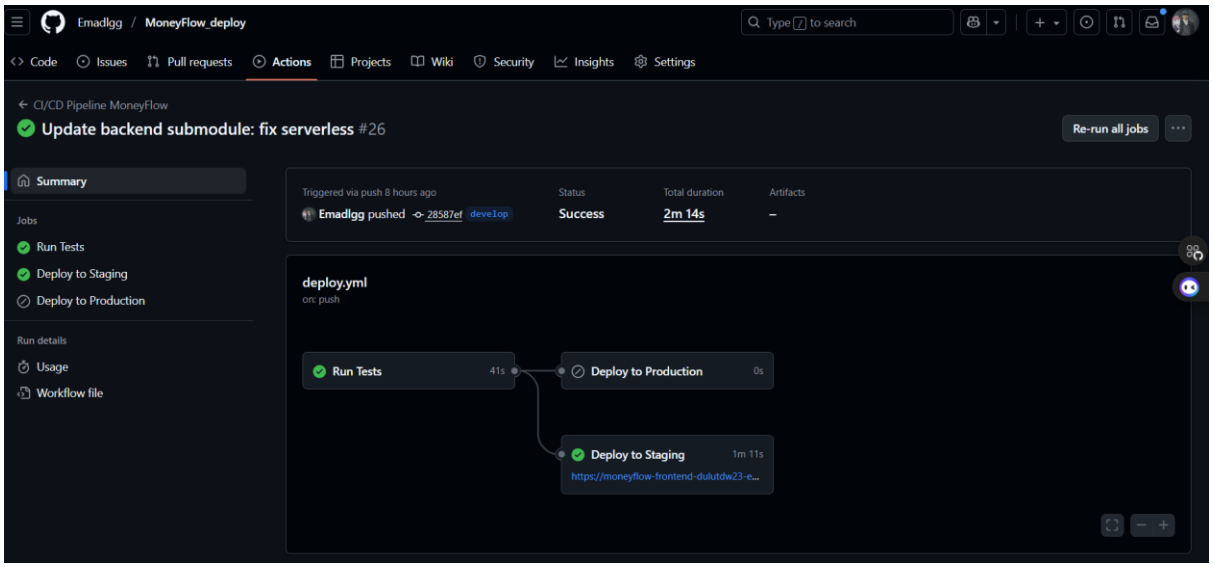
Implementamos staging y production para probar cambios en condiciones realistas sin afectar usuarios. Staging usa base de datos de prueba donde se pueden hacer cambios libremente, y los preview deployments permiten compartir URLs con stakeholders para validación temprana. Si algo falla en staging, el impacto es mínimo. La configuración dual también permite optimizaciones específicas: staging con logging verbose para debugging, production con logging minimal para rendimiento.

5.5 Framework de Pruebas

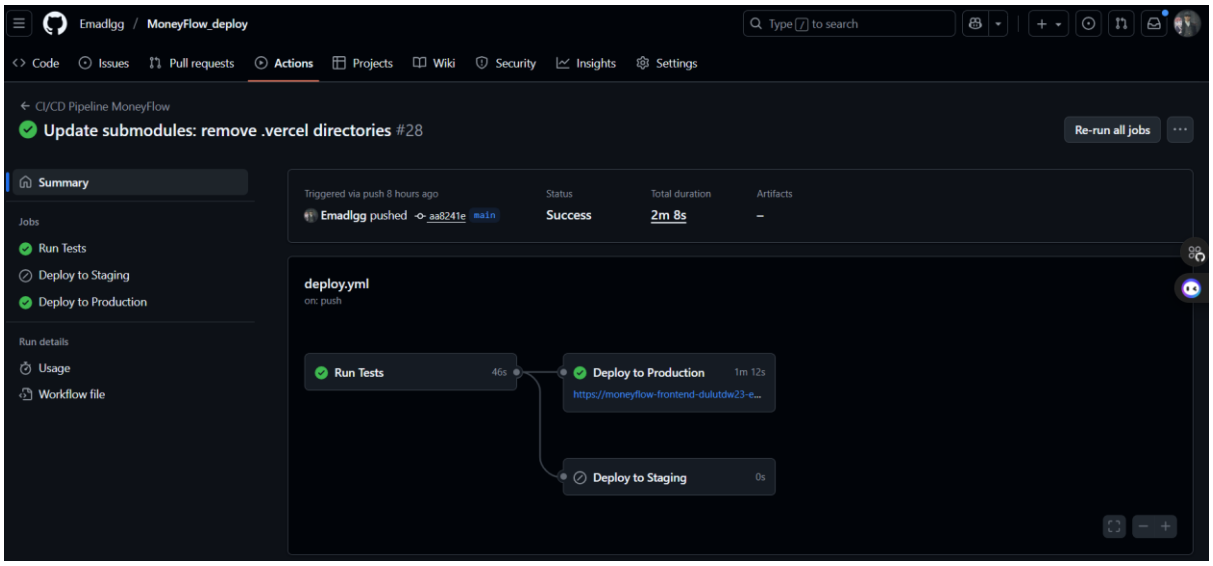
React Testing Library se eligió por su enfoque centrado en el usuario, probando comportamiento visible en lugar de implementación interna. Esto hace los tests resistentes a refactorizaciones y aumenta confianza en que se prueba lo importante. Para el Backend, Jest/Mocha permiten mockear dependencias externas (Supabase) para tests rápidos y deterministas. La cobertura automática de código ayuda a identificar áreas sin protección, aunque no es el único indicador de calidad.

6. EVIDENCIAS DE EJECUCIÓN EXITOSA

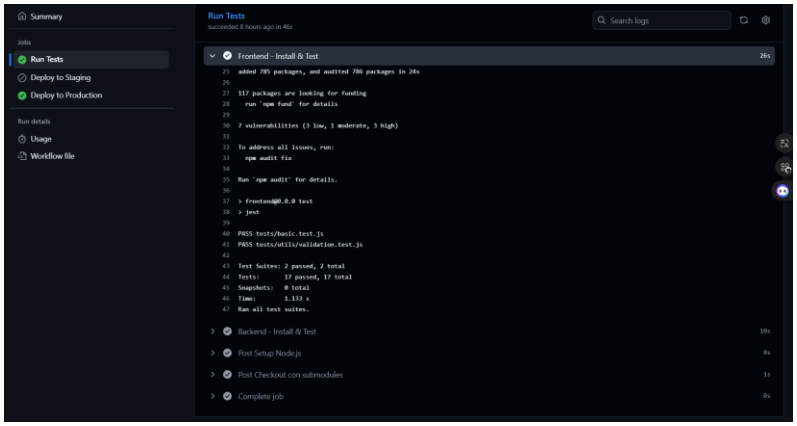
6.1 GitHub Actions - Pipeline Ejecutado



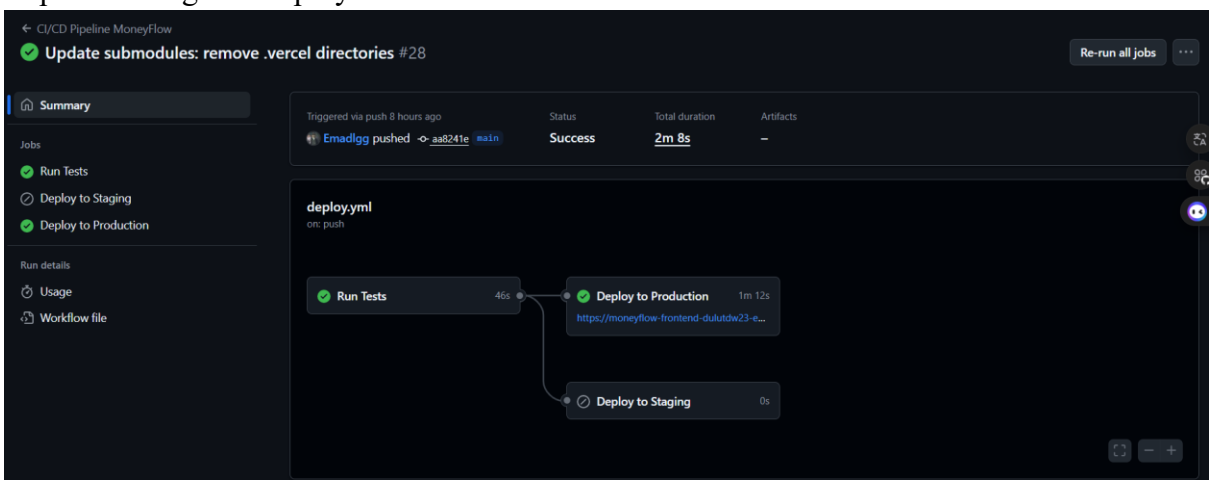
Captura 2: Workflow exitoso en rama main



Captura 3: Logs de Tests

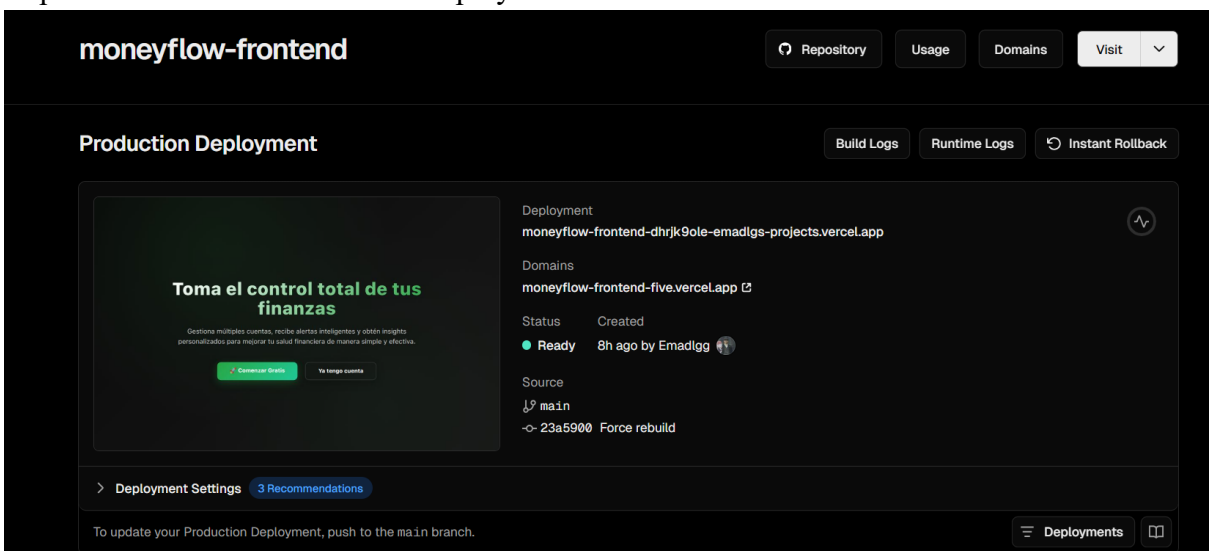


Captura 4: Logs de Deploy

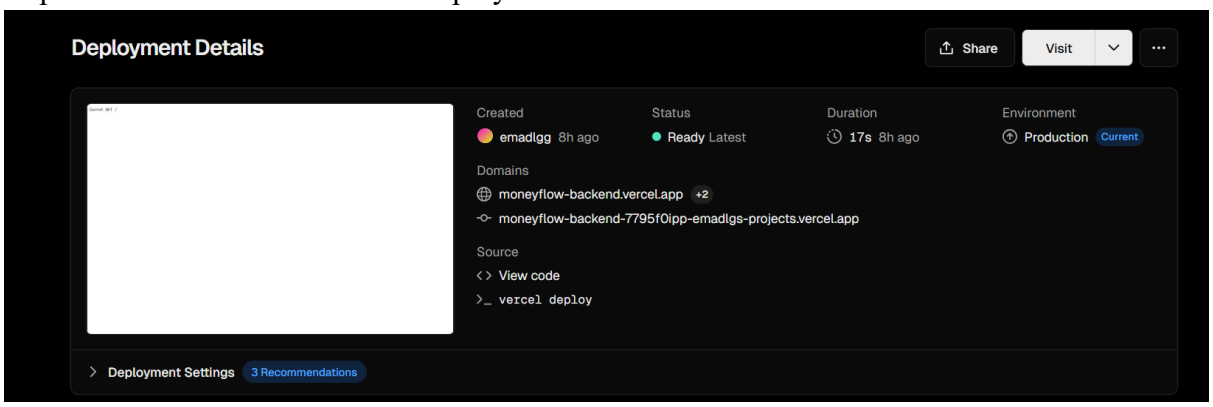


6.2 Vercel Dashboard - Deployments

Captura 5: Frontend Production Deployment

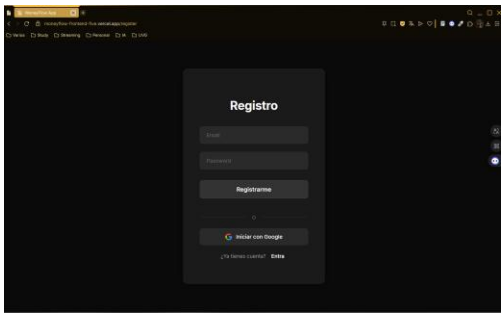


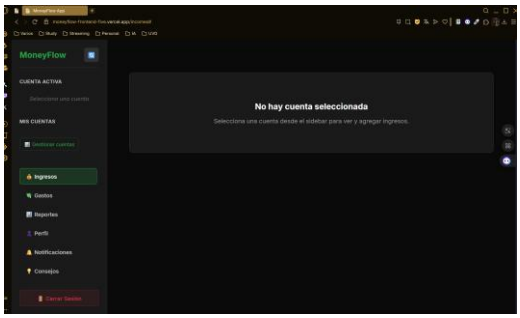
Captura 7: Backend Production Deployment



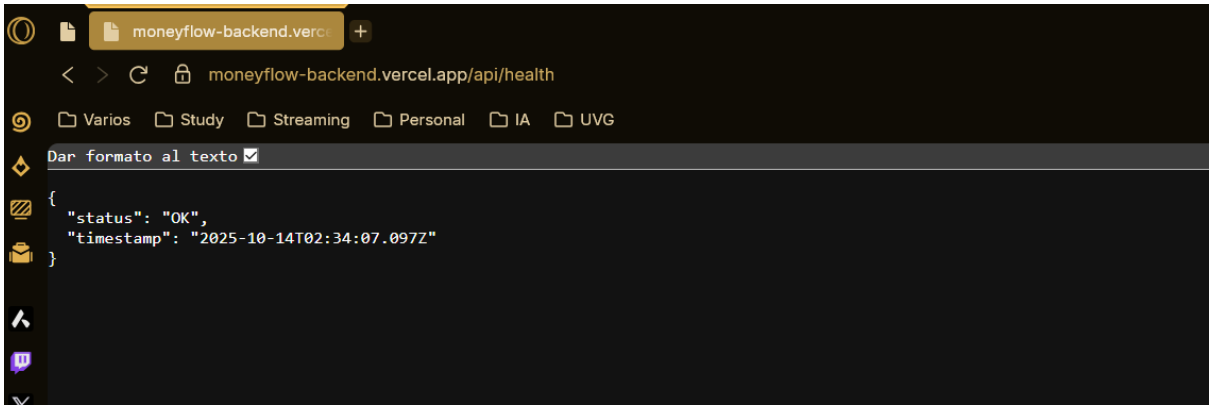
6.3 Aplicación Funcionando

Captura 8: Homepage en Production





Captura 9: Backend API funcionando



Repositorios

- Deploy (Principal): https://github.com/Emadlgg/MoneyFlow_Deploy
- Frontend: https://github.com/Emadlgg/MoneyFlow_Frontend
- Backend: https://github.com/Emadlgg/MoneyFlow_Backend

8. REFERENCIAS

- GitHub. (2024). *GitHub Actions documentation*. GitHub Docs. https://docs.github.com/en/actions

- GitHub. (2024). *Workflow syntax for GitHub Actions*. GitHub Docs. https://docs.github.com/en/actions/writing-workflows/workflow-syntax-for-github-actions

- Vercel. (2024). *Vercel CLI documentation*. Vercel Docs. https://vercel.com/docs/cli

- Vercel. (2024). *Deploying with GitHub Actions*. Vercel Docs. https://vercel.com/guides/how-can-i-use-github-actions-with-vercel

- Atlassian. (2024). *Continuous Integration Tools*. Atlassian. https://www.atlassian.com/continuous-delivery/continuous-integration/tools

- Supabase. (2024). *Supabase Documentation*. Supabase Docs. https://supabase.com/docs