# Entrega 4: Entrega final

Fecha: 27/11/2025

## Entregables

1. ***Implementación*** de todas las funcionalidades completas, con la correspondiente integración con el Backend.
2. ***Despliegue*** del aplicativo en la nube (Backend + Frontend).
3. ***Documentación*** acerca del despliegue del aplicativo que contenga el detalle de los pasos a seguir para desplegar el aplicativo por primera vez y por cada vez que se quiere subir a producción una nueva release.
4. ***Tests de integración*** en la Capa de Controladores del Backend: solamente es necesario realizar 1 test, cuyo endpoint a testear queda a criterio del equipo.
5. ***Tests E2E*** en el Frontend: solamente es necesario realizar 1 test, cuya funcionalidad a testear queda a criterio del equipo.

## Tecnologías a utilizar

- Todas las mencionadas en las anteriores entregas.
- Render como Cloud Application Platform para el despliegue de nuestro Backend (se pueden utilizar algunas otras alternativas, a criterio del equipo y en común acuerdo con el equipo docente).
- Netlify como plataforma para despliegue de nuestro Frontend (se pueden utilizar algunas otras alternativas, a criterio del equipo y en común acuerdo con el equipo docente).
- Cypress como herramienta de Testing E2E. Está permitido buscar y utilizar alguna otra herramienta alternativa (como Jest, por ejemplo).
- Jest como framework para Testing de Integración

## 1. Despliegue del Frontend

Para el frontend utilizamos Vercel debido a que la plataforma está optimizada para aplicaciones basadas en Next.js y ofrecen integración nativa con GitHub. Implementamos un GitHub Action que, en cada push a main, ejecuta la CLI de Vercel y dispara automáticamente el deploy. Solo fue necesario generar una API_KEY desde la plataforma y configurarla como variable de entorno en GitHub.

https://vercel.com/kb/guide/how-can-i-use-github-actions-with-vercel

Esto garantiza despliegues rápidos y reproducibles sin intervención manual.
Ventajas:

- Deploy automático por CI/CD sin pasos manuales.
- Soporte nativo para Next.js (renderizado híbrido SSR/ISR).
- Rollbacks instantáneos.

- Manejo fácil de variables de entorno.

Desventajas:
- Límites en planes gratuitos (tiempos de build).
- Dependencia de una plataforma propietaria.

## 2. Despliegue del Backend

El backend se desplegó en Render, utilizando nuestra imagen Docker. Render permite subir un servicio basado en un contenedor propio sin modificar el código, lo cual simplifica el pipeline. Render nos ofrece 512 MB de RAM, lo cual es justo para nuestro backend y por eso elegimos delegar la autenticación en Clerk.

## 3. Base de Datos

La base de datos es MongoDB Atlas, porque:
- Tiene alta disponibilidad por defecto.
- Fácil conexión desde Render mediante IP allowlist.
- No requiere mantenimiento local y escala automáticamente.

## 4. Autenticación con Clerk (SSO en la nube)

Elegimos Clerk para autenticación y manejo de sesiones.
Se justificó principalmente porque:
- Al estar 100% gestionado en la nube, evita que tengamos que almacenar sesiones en memoria local.
- Esto reduce el consumo de RAM del backend, algo crítico porque Render solo da 512 MB en el plan gratuito.
- Clerk maneja tokens, sesiones, refresh, MFA y roles sin procesamiento local.

Ventajas:
- Ahorro de RAM (evitamos almacenar sesiones/server state).
- Seguridad y escalabilidad con mínima configuración.
- SDK simple para frontend y backend.

Desventajas:
- Servicio externo (dependencia).
- Requiere configurar claves públicas y privadas en ambos entornos.

## 5. CI/CD con GitHub Actions

Implementamos un flujo CI/CD simple:
- Cada vez que mergeamos a main, GitHub Actions:
    - construye el proyecto,
    - valida que compile
    - ejecuta la CLI de Vercel
    - despliega automáticamente la nueva versión.

Esto garantiza que el deploy siempre refleje el estado real del repositorio y evita divergencias.

## 6. Test E2E con Cypress (Frontend)

Para el test E2E implementamos un flujo completo en Cypress ya que Clerk posee soporte para el mismo. Dicha prueba valida la interacción real de un vendedor con la plataforma ejecutada de manera local

Flujo real del test E2E (Cypress)

1. Logueo del vendedor mediante Clerk.
2. Creación de un producto desde la UI.
3. Agregar ese producto al carrito desde el catálogo.
4. Realizar la compra con el usuario correspondiente.
5. Validar que el stock se reduce correctamente después del pedido.
6. Cancelar el pedido desde la vista del vendedor.
7. Verificar que el stock se repone luego de la cancelación.
8. Eliminar el producto para finalizar el ciclo.

Este test asegura la consistencia del flujo completo: creación → venta → actualización de stock → cancelación → reposición → limpieza final.