Cubezon

Monté un marketplace tipo e-commerce… dentro de Minecraft.

Compras en una web, pagas con dinero del juego y recoges el pedido en una taquilla del servidor. Objetos reales, sin duplicados.



Proyecto de portfolio

desliza →

EL RETO

Dos sistemas, un mismo dinero.

La web vende. El servidor de juego guarda los saldos y los inventarios. Entre medias, una base de datos compartida.

Si el servidor se cae **a mitad de una compra**…

✕ ¿Se cobra dos veces al comprador?
✕ ¿Se paga dos veces al vendedor?
✕ ¿Aparecen objetos de la nada?

En un juego, un duplicado rompe la economía de todo el servidor. Había que diseñarlo como un sistema de pagos.

ARQUITECTURA

El plugin es la única autoridad.

Web

`Next.js 16 · React 19 · TypeScript · Tailwind 4`

- Catálogo, carrito, checkout y seguimiento
- Panel de vendedor y de administración

↓ `solo crea pedidos PENDING`
`nunca toca saldos ni inventarios`

PostgreSQL (Supabase)

`24 migraciones · RLS en todas las tablas · triggers · RPC`

- Estados del pedido con historial por trigger
- Escrow: una fila = una unidad depositada

↑ `reclama, cobra y entrega pedidos`
`transacciones + bloqueo de filas`

Plugin de Paper

`Java 21 · Paper 1.21 · Vault · HikariCP · JUnit`

- Cobra con Vault en el hilo principal
- Depósitos, taquillas, buzones y repartos

CICLO DE UN PEDIDO

Cada paso, su transacción.

PENDING

La web crea el pedido. El plugin lo **reclama** en una transacción: reserva unidades y valida tienda, taquilla e ítem.

↓

PAID

Cobro con Vault en el hilo principal del servidor. Cualquier fallo antes de cobrar → FAILED sin mover dinero.

↓

READY

En la taquilla. El pago al vendedor se **marca antes de pagar**: un fallo deja un pago pendiente, nunca uno doble.

↓

COLLECTED

El jugador lo recoge. Las unidades pasan a DELIVERED en la misma transacción.

Al arrancar, un pedido a medio procesar se marca **`FAILED / INTERRUPTED`** y se registra para revisión. Nunca se reintenta un cobro a ciegas.

DECISIONES DE DISEÑO

Cuatro reglas que no se rompen.

01

Stock solo por depósito

Un objeto se vende solo si el vendedor lo metió antes en el juego. Se serializa con todo su NBT. Nada se crea de la nada.

02

Mejor perder recuperable que duplicar

Si algo falla al entregar, los objetos quedan registrados en Base64 para devolverlos a mano.

03

La web no toca el dinero

La clave pública solo lee. La clave secreta vive en el servidor. Los permisos se vuelven a comprobar en SQL.

04

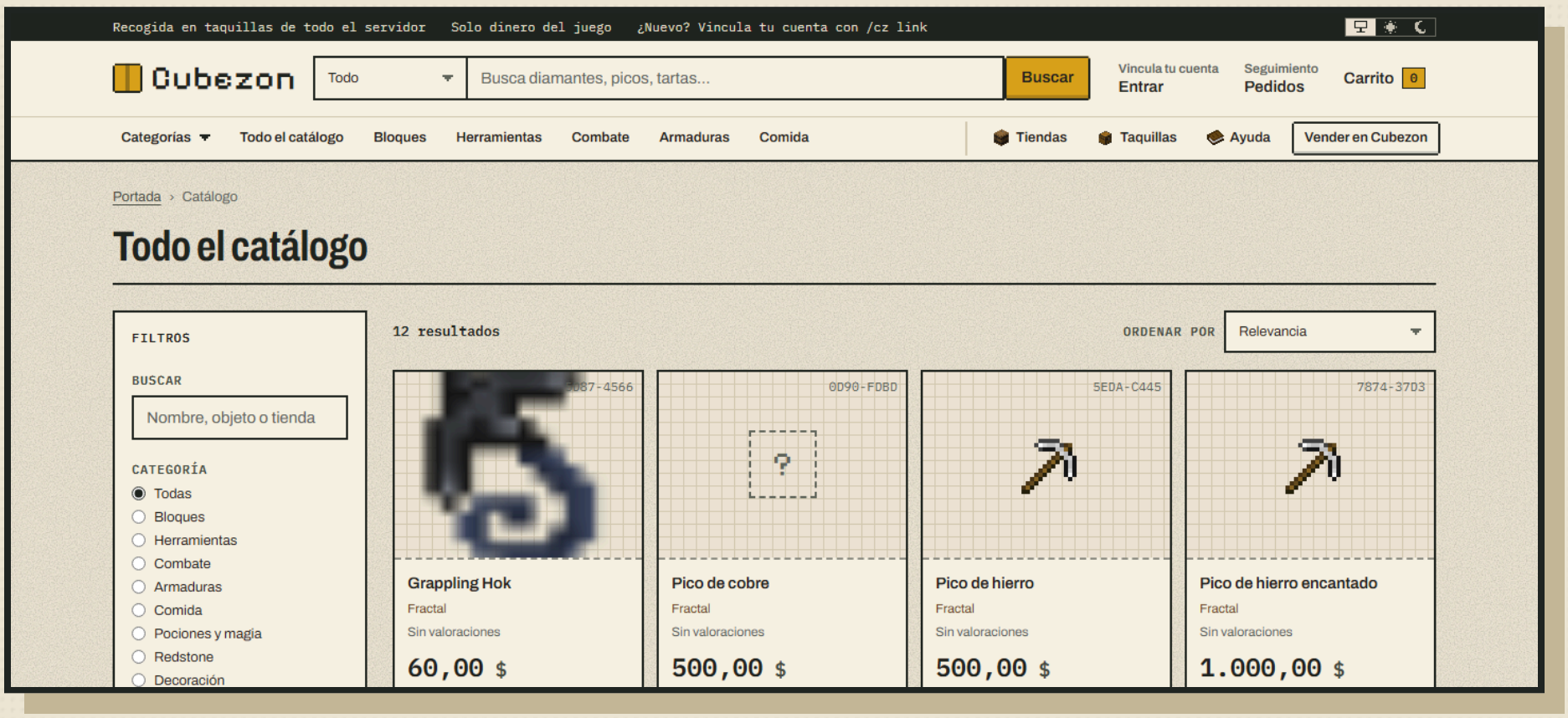
Todo hilo en su sitio

Base de datos en un executor propio. Vault e inventarios en el hilo principal. Al apagar, se espera a que terminen los depósitos.

EL PRODUCTO

Una tienda online de verdad.

Búsqueda, categorías, ficha con «vendido por», carrito, checkout con propina y envío exprés, seguimiento hasta la taquilla.





MODO ROLEPLAY

Las tiendas se gestionan jugando.

Un modo opcional donde el personal de cada tienda hace el trabajo dentro del juego.

▣ ORDENADOR

Panel de empresa

Aceptar o rechazar pedidos, productos, stock y empleados desde un bloque en el mundo.

▤ RANGOS

Permisos finos

7 permisos por rango, comprobados en SQL. Nadie se asciende a sí mismo.

▥ PAQUETES

Repartidores

Se imprime un paquete y alguien lo lleva a la taquilla. Cobra comisión y propina.

▦ ALMACÉN

Capacidad física

Cuánto stock admite una tienda depende de los bloques de almacén colocados en su zona.

EN NÚMEROS

Hecho para durar.

336
tests automáticos
(110 JUnit + 226 Vitest)

24
migraciones SQL versionadas,
RLS en cada tabla

12k
líneas de Java
en el plugin

24k
líneas de TypeScript
en la web

`Java 21` `Paper` `Vault` `Gradle` `Next.js 16` `React 19` `TypeScript` `Tailwind 4` `PostgreSQL` `Supabase` `GitHub Actions`

Cubezon

El código es abierto.

Plugin, web y esquema SQL, con tests, CI y documentación. En el repositorio están las capturas y la guía para montarlo.

github.com/IOxee/Cubezon

Java · TypeScript · PostgreSQL

