AUREO

Hackathon de Acepta Bitcoin - 09/02/26

Taller de vibe coding con IA para Bitcoin

De idea → prompt → MVP en ~40 minutos

Tristan Borges Solari

Lo que te llevarás

Un flujo de trabajo práctico que puedes reutilizar para cualquier app pequeña con IA

Resultados

1. Un ciclo de “vibe coding” repetible
2. Una plantilla de prompt maestro para construir aplicaciones
3. Un MVP funcional de alertas BTC → Telegram
4. Un checklist para evitar errores comunes

Premisa clave

El vibe coding reduce el costo de iterar, pero aún necesitas intención clara, restricciones bien definidas y verificación.

¿Qué es Vibe Coding?

- Construcción rápida e iterativa con un agente de IA que genera código
- La IA hace el boilerplate; tú diriges intención de producto, restricciones y revisión
- La ventaja: baja radicalmente el costo de probar ideas

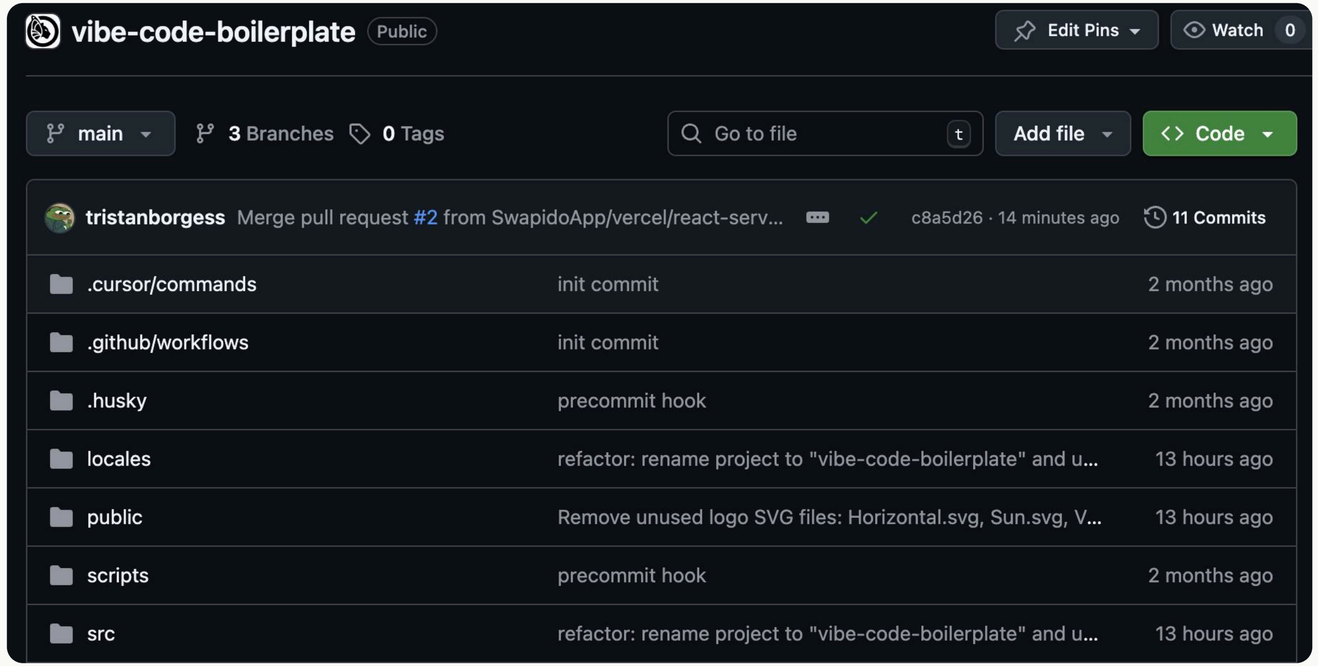
Lo que NO es Vibe Coding

- No es “una frase → app lista para producción”
- No es “nunca leo el código”
- No es “me salto pruebas/seguridad”
- Sí es: prototipado veloz + iteración disciplinada

Setup y herramientas

Setup del taller (asumimos listo):

- Cursor
- Clonar repo base:
- https://vibe-code-boilerplate.vercel.app/es



Alternativas

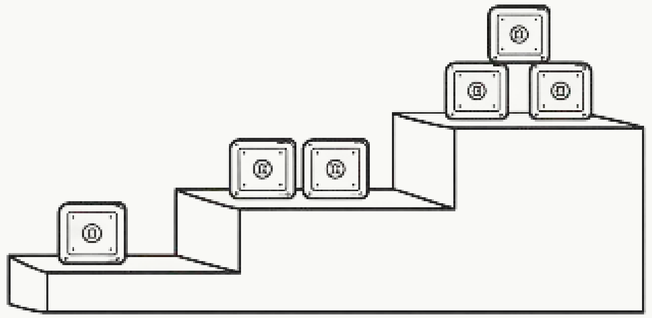
- Replit (correr y desplegar rápido)
- Claude Code (buen flujo en terminal/ planeación)
- Lovable.dev (generación rápida de UI)
- n8n (automatizaciones y orquestación)

Hosting

- Vercel para desplegar después (no es requisito hoy)
- Alternativas: Netlify / Render / Replit

¿Necesito saber programar?

No, puedes llegar a un MVP funcional



Pero ayuda porque:

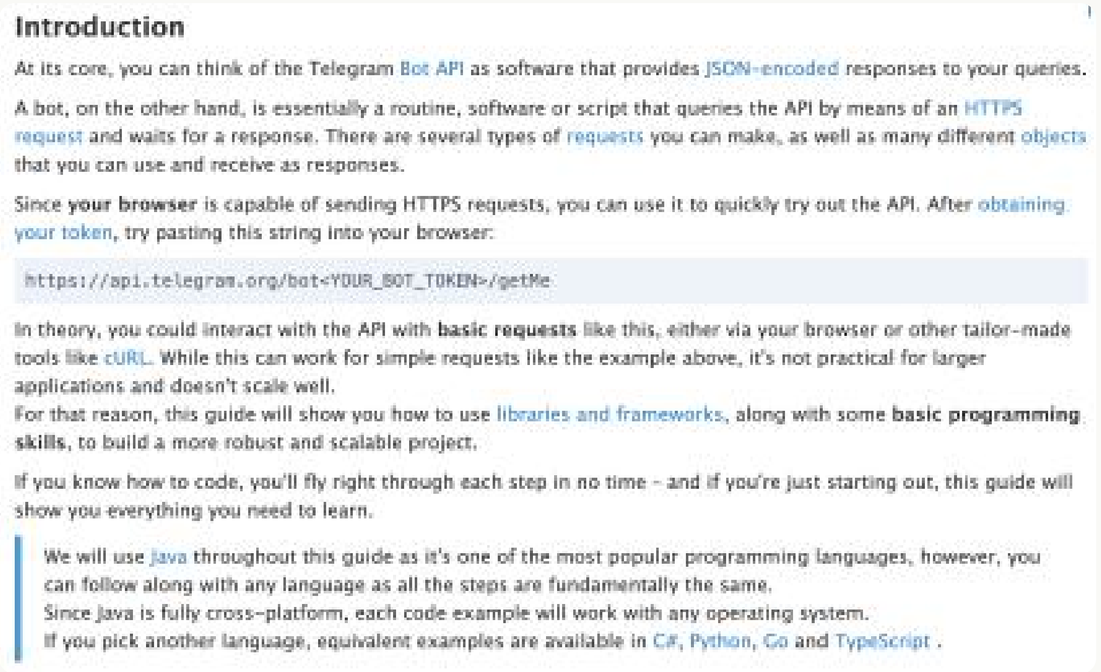
- Depuras más rápido
- Revisas con mayor seguridad
- Haces ajustes pequeños sin re-prompting

La habilidad principal: **requisitos claros + alcance acotado + disciplina de prueba**

“Cómo pensar como ingeniero sin escribir código.”

Mini marco “ingenieril”

1. Lee primero la documentación del API (endpoints, auth, límites)
2. Modulariza: UI / hooks / ruta server / helpers
3. Secretos en variables de entorno (solo servidor)
4. Prueba temprano, por fase (primero happy path)
5. Documenta el happy path (README con quickstart)



Introduction

At its core, you can think of the Telegram Bot API as software that provides JSON-encoded responses to your queries.

A bot, on the other hand, is essentially a routine, software or script that queries the API by means of an HTTPS request and waits for a response. There are several types of requests you can make, as well as many different objects that you can use and receive as responses.

Since **your browser** is capable of sending HTTPS requests, you can use it to quickly try out the API. After obtaining your token, try pasting this string into your browser:

```
https://api.telegram.org/bot<YOUR_BOT_TOKEN>/getMe
```

In theory, you could interact with the API with **basic requests** like this, either via your browser or other tailor-made tools like cURL. While this can work for simple requests like the example above, it's not practical for larger applications and doesn't scale well.
For that reason, this guide will show you how to use libraries and frameworks, along with some **basic programming skills**, to build a more robust and scalable project.

If you know how to code, you'll fly right through each step in no time - and if you're just starting out, this guide will show you everything you need to learn.

> We will use Java throughout this guide as it's one of the most popular programming languages, however, you can follow along with any language as all the steps are fundamentally the same.
> Since Java is fully cross-platform, each code example will work with any operating system.
> If you pick another language, equivalent examples are available in C#, Python, Go and TypeScript .

Tip de alto impacto

Cursor rules.md

- Un archivo rules.md define “reglas de la casa” para el agente:
  - stack, patrones, qué sí/no, seguridad, sin DB, etc.
- Mejora consistencia y reduce iteraciones

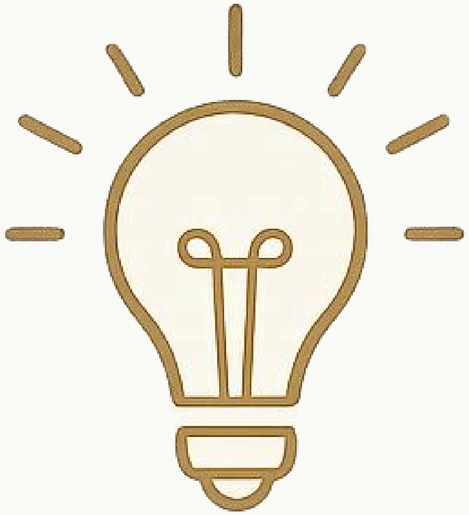
```
# Project Rules
- Next.js App Router + TypeScript
- shadcn/ui only (no new UI libs)
- No DB for MVP (localStorage only)
- Secrets: env vars server-side only
- Prefer small diffs; no refactors without asking
```

Nota: Mantén esto breve: es una palanca, no el tema central.

Cómo elegir una gran idea

Vibe coding baja el costo de experimentar, así que el filtro de ideas importa:

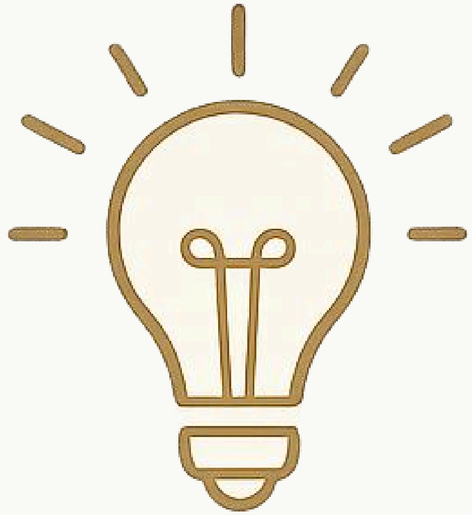
- Divertida de construir
- Te duele a ti (o te lo piden seguido)
- Resultado medible y claro
- MVP en una pantalla / un flujo
- Ideal: una integración simple (un API)



Heurísticas + ejemplos

Ideas perfectas para taller:

- Alertas/notificaciones (Telegram/email/SMS)
- “Calculadora + explicación”
- Dashboards mínimos (una sola fuente de datos)
- Automatizaciones (n8n)
- Asistentes de decisión (salidas estructuradas)



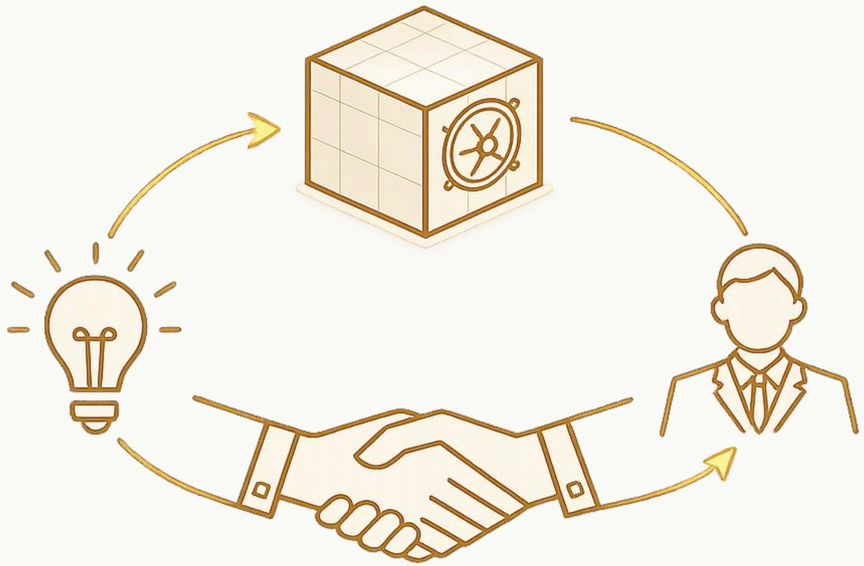
Ejemplo de hoy: Alertas de precio BTC en MXN → Telegram

Modelo mental

Qué hace grande a un prompt

Un gran prompt es:

- un **mini Product Requirements Document (PRD)**
- más **restricciones de ingeniería**
- más un **proceso de entrega**



Tres pilares:

1. Especificación clara (qué construir)
2. Control (fases, tamaño de cambios, sin refactors)
3. Verificación (criterios de aceptación + pruebas)

1/3: Estructura

Buenas prácticas

Incluye:

- Rol (“Eres un senior full-stack…”)
- Contexto (stack del repo + restricciones)
- Requisitos (features + UI + contratos de datos)
- Formato de salida (plan primero, por fases)

```
GOAL: …
CONTEXT: Next.js + shadcn/ui …
CONSTRAINTS: minimal files …
UI: screens/states …
ACCEPTANCE: checks/tests …
```

2/3: Control

Buenas prácticas

Agrega “guardrails” para evitar caos:

- Plan primero; no código hasta aprobación
- Fases (Fase 1/2/3)
- Puntos de alto después de cada fase
- Diffs pequeños; sin refactors
- Sin dependencias nuevas sin pedir permiso

```
Contrato de API: endpoints, estructura de
requests/responses, límites de tasa

Contrato de UI: componentes a reutilizar
(shadcn/ui), restricciones de layout

Contrato de estado: comportamientos de
carga/error/vacío/éxito

Contrato de datos: almacenamiento
(localStorage vs base de datos),
intervalo de actualización

Criterios de aceptación: “puedo hacer X y
ver Y” + casos de error
```

3/3: Verificación

Buenas prácticas

- Criterios de aceptación (“esto debe funcionar sí o sí”)
- Checklist de pruebas manuales por fase
- Manejo de errores y rate limits (UX básica)

Auto-revisión del agente al final:

- “¿Qué podría salir mal?”
- “¿Dónde están los riesgos de seguridad?”
- “Top 3 bugs probables aquí”

```
GOAL: …
CONTEXT: Next.js + shadcn/ui …
CONSTRAINTS: minimal files …
UI: screens/states …
ACCEPTANCE: checks/tests …
```

Evita estos errores comunes

- Hardcodear secretos (en cliente, logs, screenshots)
- Dejar que el agente “limpie” el repo con refactors grandes
- Crecimiento descontrolado de dependencias
- Prompts sin límites (“hazme una app completa”)
- No probar hasta el final
- APIs inventadas o endpoints desactualizados (verifica documentación)

(Bitcoin → Telegram)

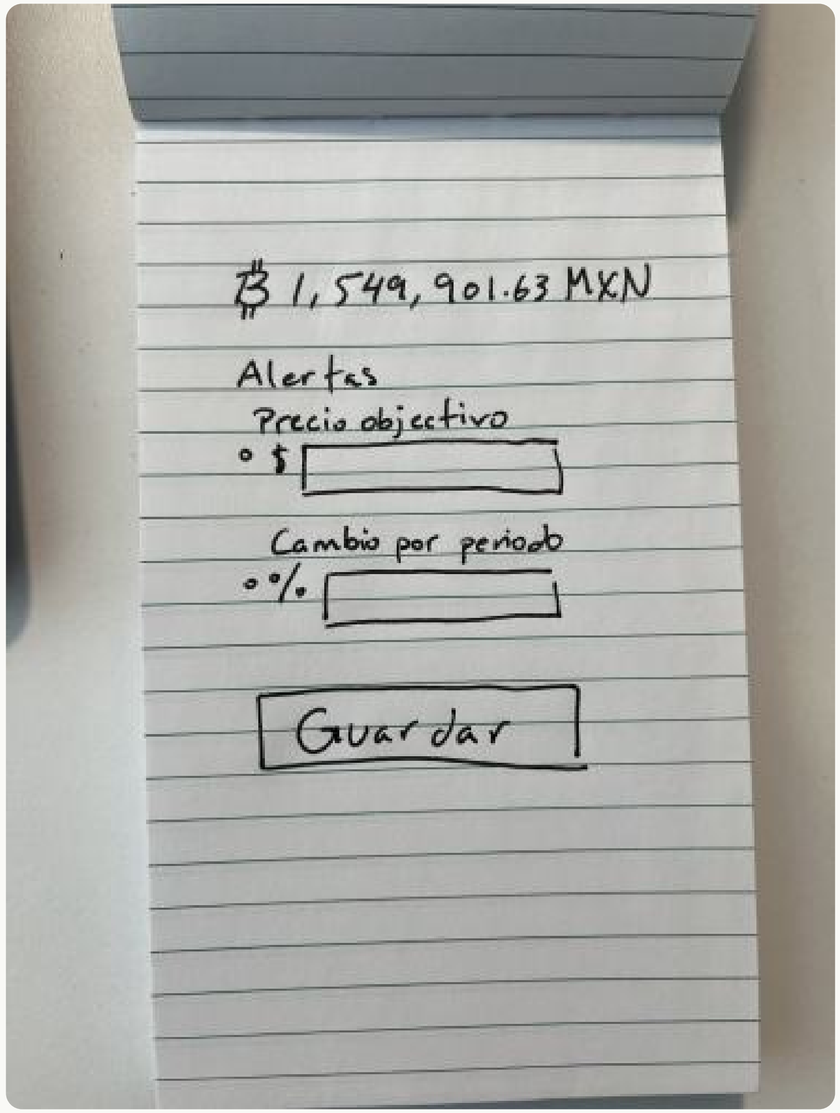
Demo: la idea

Historia de usuario

“Notificarme en Telegram cuando BTC alcance mi objetivo, ya sea un precio en MXN o un cambio porcentual”.

Requisitos de UI (MVP)

- Muestra precio de BTC en MXN (CoinGecko)
- Alertas:
  - precio objetivo arriba/abajo
  - % cambio (+/–) por periodo
- Envía notificación a Telegram cuando se cumpla
- Guardado MVP: localStorage
- Polling ~30 segundos



(Bitcoin → Telegram)

APIs + contratos

CoinGecko API

`simple/price BTC en MXN + cambio 24h`

→

Next.js UI

`(price + form)`

→

Alert evaluator

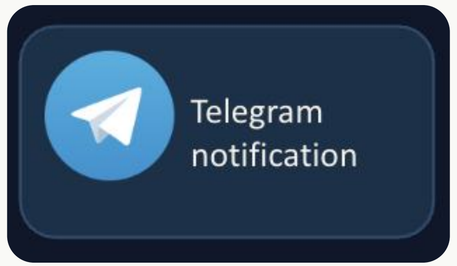
`(poll ~30s)`

→

API route

`POST /api/telegram`

↓



¿Por qué esta división?

Mantén los secretos en el servidor (token de Telegram).
Mantén todo lo demás en el cliente para el MVP.

Elección de modelo

Elija en función de: calidad del código, velocidad, costo y su tolerancia a la revisión

	Pros	Contras
Claude Opus 4.5	Codificación sólida y razonamiento; ideal para indicaciones importantes	Puede over engineer; necesita barandillas de “mantenerlo mínimo”
GPT-5.2 (Cursor)	Iteración rápida; fuerte seguimiento de instrucciones en los editores	Aún necesita revisión; esté atento a las suposiciones silenciosas
Claude Sonnet 4.5	Great speed/quality balance; reliable refactors	Es posible que se necesiten detalles de interfaz de usuario más explícitos que Opus

Consejo práctico: utiliza tu “mejor” modelo para escribir el mensaje principal y luego utiliza un modelo más rápido para la iteración.

El “Master Prompt”

Structura

```
ROLE: senior full-stack engineer
GOAL: …
STACK: Next.js App Router + TS + shadcn/
ui
CONSTRAINTS: minimal files, no new deps
FEATURES: price card, alert form, list
API: CoinGecko + Telegram route
ACCEPTANCE: manual steps + edge cases
OUTPUT: plan first, then implement in
phases
```

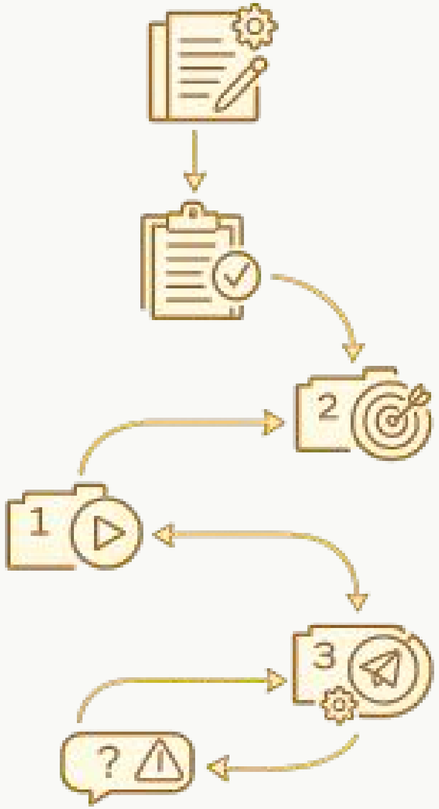
Ejemplo

```
CONSTRAINTS:
- Do not expose secrets in client code.
- Avoid over-engineering; MVP only.
- Use existing shadcn/ui components.

ACCEPTANCE:
- I can enter a price target in MXN.
- I can enter a % threshold (+/-).
- I can send a test Telegram message.
- Alerts persist in localStorage.
- UI shows loading/error states.
```

Pasos para vibe codear en vivo

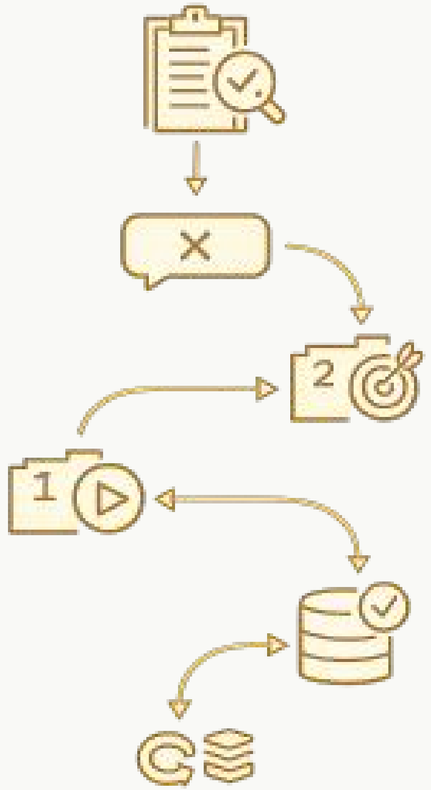
1. Pegar el master prompt en Plan mode
2. Aprobar el plan
3. Implementar Fase 1 → correr → corregir
4. Fase 2 → correr → corregir
5. Fase 3 → probar Telegram + lógica de disparo
6. Usar “Ask” para revisión: “¿Qué podría salir mal?”



Ciclo de prueba e iteración

Después de cada fase:

- Verifica 1 criterio de aceptación a la vez
- Si falla:
  - pega el error exacto
  - pide arreglo mínimo
- Haz commits frecuentes (rollback fácil)



Conclusion

- Una buena indicación es una especificación: contexto + restricciones + criterios de aceptación.
- Planifique primero el trabajo con varios archivos; desarrolle por fases; verifique cada fase.
- Si se desvía: revierta → ajuste el plan → vuelva a ejecutar.
- Nunca pegue secretos; mantenga los tokens en el servidor.
- Guarde las indicaciones que funcionen (se convertirán en su manual de estrategias).

Gracias

Contacto

@tristanborgess

tristan@aureobitcoin.com

aureobitcoin.com

https://vibe-code-boilerplate.vercel.app/es

Recursos

Utilidades

- Cursor IDE
- n8n
- Codex
- Claude Code
- Replit
- Lovable
- Bolt
- Base44

Programación

- Bitcoin Dev Kit
- Lightning Dev Kit
- Nostr Dev Kit
- Libreria de Satoshi
- Learn me a bitcoin
- Roadmap.sh
- Base 58 School