Alcancía — spec del MVP

> Un agente que te arma el plan de ahorro conversando, reparte cada peso que te entra en alcancías onchain (una por meta) y te frena los impulsos antes de que rompas el chanchito. Si la meta es un viaje, ahorra en la moneda en la que vas a gastar.

Para: demo en vivo en la builder session del sábado 17/10 (18–20 h ART), y ejemplo de referencia del Track 1 (Stable Agents: LatAm) de Agents on Open Rails. **Chain:** Celo mainnet (42220). **Estado:** spec, pre-build. **Fecha:** 2026-10-15.

---

1. Problema

Ahorrar en LatAm es difícil por dos razones que se refuerzan:

1. **Disciplina.** La plata que queda en la cuenta se gasta. El impulso de las 2 am en Mercado Libre le gana a la meta de julio.
2. **Moneda.** Si vas a gastar en otra moneda (un viaje a Brasil), ahorrar en la tuya te expone al tipo de cambio. La salida habitual es dolarizarse, que es otro paso, otra cuenta y otro costo.

2. Solución en una frase

Conectás tu wallet, le contás al agente tus metas del año, tu ingreso y tus gastos fijos; te propone cuánto apartar para cada meta; crea una alcancía (un contrato propio) por meta y, desde ahí, cada vez que te entran wARS reparte el porcentaje de cada una sola. Sacar antes de tiempo tiene fricción: un acertijo sobre tu meta, una reflexión y, si no, 72 h de espera. Nunca te puede retener la plata.

3. Experiencia (la demo)

Todo es un chat gamificado. El agente responde con componentes, no solo texto (AI SDK + tool calls que renderizan UI).

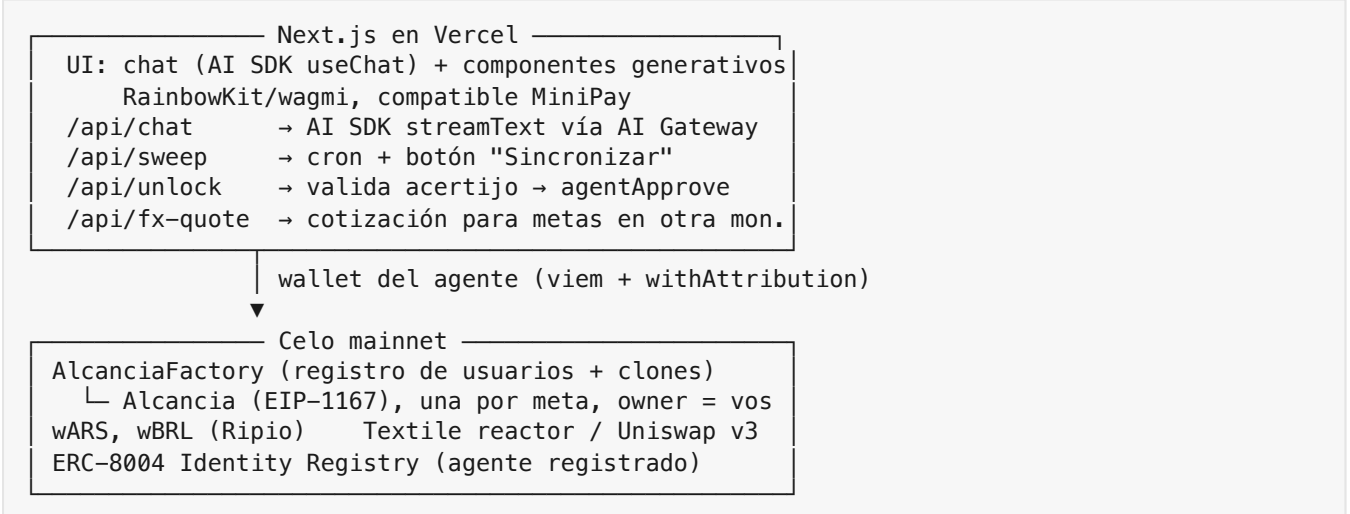
Paso	Qué ve el usuario	Qué pasa por debajo
0	Pantalla única: "Conectá tu wallet para empezar".	RainbowKit / MiniPay. Se lee el saldo de los 6 wFiat.
1	"¿Cuáles son tus metas este año?" → escribe "cruce de los Andes, Ironman, viaje a Florianópolis".	El LLM extrae metas.
2	Una **tarjeta por meta**: nombre, emoji, monto, selector de moneda (ARS / USD / BRL…), fecha.	Si la meta es un viaje, el agente sugiere la moneda del destino (Florianópolis → BRL).
3	"¿Cuánto te entra por mes?" + **tabla de gastos fijos** editable (alquiler, comida, internet, gimnasio…).	Se calcula el excedente mensual.

Paso	Qué ve el usuario	Qué pasa por debajo
4	**El plan:** gráfico de torta + sliders. "Te quedan ARS 400.000 libres. Ironman 35%, Cruce 25%, Floripa 15%, libre 25%." Avisos si una meta no llega a tiempo.	Cálculo determinístico (no LLM) de `% = (meta − ahorrado) / (meses × excedente)`, con el LLM explicando y ajustando.
5	Botón **"Crear mis alcancías"** → 1 firma de `approve` + 1 tx de creación. Confetti.	`AlcanciaFactory.createBatch(...)` despliega un clon por meta.
6	Dashboard: una alcancía por meta, con barra de progreso, racha 🔥, nivel, y la dirección onchain de cada una. Botón **"Cargar wARS"**.	Link a Ripio Ramp con `address=` del usuario.
7	Entran wARS → al rato: "Entraron 200.000. Aparté 70.000 para Ironman (42%), 50.000 para el Cruce y R$ 210 para Floripa."	Cron/agent: detecta el ingreso, `sweep`, y para la alcancía en BRL hace el FX.
8	"Quiero sacar 50.000 del Ironman." → **ritual**: reflexión ("Ironman se corre de julio a septiembre"), "contame por qué", **acertijo** sobre la meta. Si lo resolvés → sale ya. Si no → 72 h de espera, cancelable.	`requestEarlyWithdraw` + `agentApprove` firmado por el agente.
9	"Resumen de la semana" (tarjeta): cuánto entró, cuánto se apartó, avance por meta, cambio en el tipo de cambio para metas en otra moneda.	Calculado de eventos onchain.

Gamificación (MVP liviano)

- Hitos 25 / 50 / 75 / 100% con confetti y mensaje del agente.
- Racha: semanas seguidas con al menos un sweep.
- Nivel del ahorrista según % cumplido del plan.
- El acertijo es sobre tu propia meta ("¿Cuántos km nada un Ironman?", "¿En qué estado de Brasil está Florianópolis?"). Lo genera el LLM con la respuesta guardada server-side; tres intentos.

4. Arquitectura



Sin base de datos para el MVP: el estado de verdad está onchain (metas, saldos, cursor de sweep). Lo que no es onchain (gastos fijos, ingreso declarado) vive en `localStorage` del usuario; es solo input del plan.

5. Contratos

Foundry, Solidity 0.8.x. Dos contratos.

5.1 `AlcanciaFactory`

- `createBatch(GoalInput[] goals)` — despliega un clon de `Alcancia` por meta con `owner = msg.sender`, `agent = AGENT`. Guarda la lista de alcancías por usuario. `GoalInput = { token, name, target, deadline, bps }`.
- `sweep(address user, uint256 income, uint256 toBlock)` — **onlyAgent**. Lee las alcancías del usuario y, para cada una en el token de ingreso, hace `transferFrom(user → alcancía, income * bps / 10_000)`. Exige `toBlock > lastSweptBlock[user]` y guarda `lastSweptBlock[user] = toBlock` (el cursor vive onchain, así el backend es stateless y no se puede barrer dos veces el mismo ingreso).
- `pullForFx(address user, uint256 amount)` — **onlyAgent**. Trae al agente la porción destinada a alcancías en otra moneda (ver §6.2).
- `setPlan(...)` / `pause()` del usuario: cambiar porcentajes o pausar el ahorro automático es siempre del owner, sin fricción.
- Tope de seguridad: la suma de `bps` de un usuario ≤ 10.000, así `sweep` nunca mueve más que `income`. El contrato no puede verificar que `income` sea real: eso lo calcula el agente desde los logs (ver riesgos §10.1).

5.2 `Alcancia` (una por meta)

Estado: `owner, agent, token, name, target, deadline, pendingWithdraw{amount, unlockAt}`.

- `deposit(amount)` — cualquiera (para aportes manuales).
- `withdraw(amount)` — owner, **libre** si `balance ≥ target` o `now ≥ deadline`.
- `requestEarlyWithdraw(amount)` — owner; arranca la espera de **72 h**.
- `executeEarlyWithdraw()` — owner, después de `unlockAt`.
- `cancelEarlyWithdraw()` — owner.
- `agentApprove()` — **onlyAgent**: deja ejecutar el pedido pendiente ya (después del acertijo). El agente **solo puede acortar** la espera; no puede mover fondos a ningún lado salvo al owner.
- `raiseTarget()` inmediato; **bajar** el target o adelantar el deadline pasa por la misma espera de 72 h (si no, bajar la meta sería la puerta trasera).
- `setAgent(0)` — owner puede desconectar al agente en cualquier momento; las alcancías siguen funcionando solas (retiros con espera).

Invariante central: **los fondos de una alcancía solo pueden salir hacia su owner.**

Cada alcancía es una dirección propia → actividad visible por meta en el explorer y una tx por movimiento.

6. El agente

6.1 Chat (AI SDK + Vercel AI Gateway)

- `streamText` con modelo vía AI Gateway (p. ej. `anthropic/claude-sonnet-5-5`, configurable por env).
- Tools que devuelven datos que la UI renderiza como componentes: `proposeGoals`, `askIncome`, `askFixedCosts`, `buildPlan`, `explainPlan`, `startUnlockRitual`, `weeklySummary`, `getFxQuote`.
- Las tools que escriben onchain **no** las ejecuta el LLM: el chat arma la propuesta y la firma la hace el usuario desde la UI (`createBatch`, retiros). Lo único que firma la wallet del agente es `sweep`, `pullForFx`, el FX y `agentApprove`, cada uno con reglas determinísticas.

6.2 Sweep automático + FX

1. Cron (Vercel Cron) + botón "Sincronizar" para la demo.
2. Para cada usuario de la factory: leer logs `Transfer(to = user)` de wARS desde `lastSweptBlock`. Se excluyen transferencias desde sus propias alcancías (retiros) y desde la factory/agente.
3. `sweep(user, income, toBlock)` → reparte a las alcancías en wARS.
4. Alcancías en otra moneda (MVP: **wBRL**): `pullForFx` trae la porción al agente → RFQ en **Textile v2** (`/v2/rfq/request`, keyless con `takerProof` o con API key) wARS → USD₮ → wBRL (dos patas; hoy no hay mercado directo) → `deposit` en la alcancía wBRL. Fallback: Uniswap v3 multihop wARS/USD₮ 0,01% → USD₮/wBRL 0,01%, ambos pools existen en Celo. La custodia del agente dura los segundos entre `pullForFx` y `deposit`; se explica en la entrega.
5. Si el monto de FX es < ~US$1 (mínimo de Textile), se acumula para el próximo.

Monedas: el sistema acepta cualquier wFiat como moneda de ingreso y de alcancía, pero liquidez onchain en Celo hoy solo hay para **wARS y wBRL** (Textile + Uniswap). wMXN/wCOP/wPEN/wCLP: alcancías en la misma moneda funcionan; FX hacia ellas queda fuera hasta que haya liquidez.

6.3 Lectura de la cadena vía buy (usebuy.ai)

El agente paga por la infraestructura que usa, a través del gateway de buy (`https://gateway.usebuy.ai`), con x402 en Celo (USDC/USD₮/USA₮; el gateway paga el gas).

Servicio	Costo	Uso
SQD `/sqd/datasets/celo-mainnet/finalized-stream`	Gratis (120 req/min por IP)	Detección de ingresos: logs `Transfer` de wARS hacia los usuarios desde `lastSweptBlock`, solo bloques finalizados (sin riesgo de reorg).
Chainstack Celo RPC `/chainstack/celo`	US$0,01 por request (lote de hasta 2000 llamadas)	**Un lote pago por ciclo**: saldos de todas las alcancías, allowances, `slot0` de los pools wARS/USD₮ y wBRL/USD₮ (precio para FX y metas en USD), nonce y gas.

Reglas:

- Una llamada paga por ciclo de sweep, no por usuario: integración real, no spam.
- Cron cada 15–30 min + botón "Sincronizar" → centavos por día.
- Nunca reintentar a ciegas un request pago (pagos finales, beta). Si Chainstack falla, fallback a Forno para no frenar el ahorro.

- Cada recibo (token, monto, tx hash) se guarda y se muestra en la app: "este ciclo le costó al agente US$0,01".
- La wallet del agente necesita ~US$2–3 en USDC/USD₮ para x402.

Esto suma el **Track 3 (Build with buy)** a la historia de la demo.

6.4 Metas en USD guardadas en pesos

Si elegís USD para una meta sin viaje (Ironman), la alcancía es en wARS y el target se convierte a ARS con el precio del pool wARS/USD₮. El agente revisa el tipo de cambio y propone subir el target o el % ("el dólar subió, ahora te conviene 38%"). El usuario confirma (`raiseTarget`).

6.5 Ritual de desbloqueo

`/api/unlock`: el usuario hace `requestEarlyWithdraw` → el chat abre el ritual → reflexión con números reales (cuánto se corre la meta) → "contame por qué" → acertijo (respuesta guardada server-side, 3 intentos) → si acierta, el agente manda `agentApprove()`. Si no, queda la espera de 72 h. Nunca se bloquea.

7. Cumplimiento de la hackathon

Regla	Cómo
Celo mainnet 42220	Contratos y todas las tx en mainnet; Celo Sepolia solo para tests.
Attribution tag ERC-8021 en **cada** tx	`@celo/attribution-tags`: `withAttribution(CODE)` sobre el wallet client del agente **y** sobre el del usuario en el frontend (wagmi). Assert al arrancar con `codeFromRepo` si el código sale del repo.
ERC-8004	Registrar la identidad desde la wallet del agente (skill `8004`), agent card servida por la app.
Repo público	`github.com/<user>/alcancia`, público desde el día 1.
Gas	El usuario paga su gas (approve, createBatch, retiros). El agente paga el suyo (sweep, FX, approve). No hay relayer.
Track 1	wFiat real (wARS, wBRL), agente autónomo (sweep + FX + ritual), usable por cualquiera con wallet, impacto LatAm. El cruce por USD₮ en el FX se explica como puente técnico sin exposición para el usuario.

8. Stack

- Next.js (App Router) en Vercel · AI SDK + Vercel AI Gateway · Tailwind + shadcn.
- wagmi + viem + RainbowKit (detección MiniPay).
- Foundry para contratos, verificados en Celoscan/Blockscout.
- `@celo/attribution-tags`.
- Vercel Cron.
- buy gateway: SQD (gratis) + Chainstack Celo RPC (x402).

9. Alcance

MVP (3 h):

- [ ] Contratos + tests + deploy mainnet + verificados
- [ ] Chat con los pasos 0–5 (metas, ingreso, gastos fijos, plan, crear)
- [ ] Dashboard de alcancías + botón "Cargar wARS" (Ripio Ramp)
- [ ] Sweep en wARS (botón + cron)
- [ ] Alcancía en wBRL con FX (Textile; Uniswap si Textile no responde)
- [ ] Ritual de desbloqueo con acertijo + espera de 72 h
- [ ] Attribution tag en todas las tx + agente registrado en ERC-8004
- [ ] Deploy en Vercel

Después:

- Resumen semanal por email/push, guardián (un amigo aprueba), rendimiento (no hay hoy mercado de wARS en Celo: ver §11), más monedas de FX, MiniPay listing.

10. Riesgos y decisiones abiertas

1. **Confianza en `income.sweep`** recibe `income` del agente; un agente comprometido podría barrer de más (hasta el allowance). Mitigaciones: el `approve` es por un monto acotado (no infinito) y renovable; los fondos solo pueden ir a alcancías del mismo usuario (siguen siendo suyos); `setAgent(0)`.
2. **Vercel Cron**: en plan Hobby solo corre una vez por día. Para la demo, botón "Sincronizar"; en Pro, cada minuto.
3. **Textile**: RFQ puede tardar hasta ~70 s en corredores fríos o no cotizar (`no_makers_online`). Fallback Uniswap.
4. **Liquidez wBRL/USD₮ en Uniswap v3**: ~US$29k. Alcanza para la demo; montos grandes requieren Textile.
5. **El acertijo es UX, no seguridad**: la seguridad real es la espera onchain.
6. Ingreso que entra como USD₮ u otro token: fuera del MVP (solo wFiat).

11. Rendimiento (no entra al MVP)

Hoy no hay dónde poner wARS a rendir en Celo sin dolarizarse: el único rendimiento real en wARS es el vault de Morpho `KPK-WARS-YIELD` en **Ethereum** (~~24% APY). En Celo solo hay LP de Uniswap contra USD₮~~ (0,2%). En la app se muestra "Rendimiento: próximamente" y, como mucho, una proyección rotulada. Para la charla: un mercado de crédito en wARS en Celo (Morpho Blue es permissionless) es una oportunidad abierta.

12. Lo que falta de Arturo

- Attribution tag (o derivarlo del repo con `codeFromRepo`).
- API key de Vercel AI Gateway / proyecto vinculado.

- OK para crear el repo público.
- Fondear la wallet del agente (la genero yo) con CELO para deploy + gas y ~US$2–3 en USDC/USD₮ para pagar buy (x402).
- Unos wARS en tu wallet para la demo (Ripio Ramp).

13. Plan (3 horas)

Bloque	Qué	En paralelo
0:00–0:20	Repo, scaffolding Next.js, wallet del agente, Foundry	Arturo: tag, API key AI Gateway, fondear wallet
0:20–1:10	Contratos + tests → deploy mainnet	App: conexión de wallet + chat (metas, ingreso, gastos, plan)
1:10–2:00	Crear alcancías desde la app + dashboard + Ripio Ramp	Agente: sweep (SQD + Chainstack) + ritual de desbloqueo
2:00–2:30	FX wBRL (Textile, fallback Uniswap) + ERC-8004 + deploy Vercel	
2:30–3:00	Pasada punta a punta con plata real, ajustes, ensayo	

Si algo se atrasa, se corta en este orden: FX → gamificación → resumen semanal. Lo intocable: crear alcancías, sweep, ritual, tag en cada tx, 8004.

Direcciones

	Dirección
wARS	`0x0dc4f92879b7670e5f4e4e6e3c801d229129d90d`
wBRL	`0xd76f5faf6888e24d9f04bf92a0c8b921fe4390e0`
USD₮	`0x48065fbbe25f71c9282ddf5e1cd6d6a887483d5e`
Uniswap v3 wARS/USD₮ 0,01%	`0x5d8ef8b839be522b9e3d60a51edb5837cd0b2391`
Uniswap v3 wBRL/USD₮ 0,01%	`0x01c3cf91cdf7b34181c7ec8566c0236951fea40f`