how to reproduce build this site for my own dashboard

You can get very close to Stand’s Octobox by combining a modern React dashboard frontend with Polymarket’s APIs (or agents) and a real-time data layer, then layering your own layout/preset logic on top.[1] [2]

High-level architecture

- **Frontend:** Next.js or plain React + a UI kit (e.g. Tailwind + Tremor / ShadCN) to build the dashboard grid, cards, tables, and charts.[1]
- **Backend:** Lightweight Node/Next.js API routes or Python FastAPI service that proxies Polymarket data, handles auth, and normalizes markets/positions.[3] [2]
- **Data/real-time:** WebSockets or periodic polling for market/order book updates; optional analytics backend if you want historical charts and backtests.[3] [1]

Polymarket integration options

- **Direct API:** Use Polymarket’s public endpoints (and private trading endpoints if you want execution) to fetch markets, order books, and fills, then map them into your own “Octobox-like” tiles.[2]
- **Polymarket Agents repo:** Clone the open-source Polymarket Agents framework and reuse its Python code for authentication, market sync, and order routing, exposing a thin REST/WebSocket layer to your frontend.[2]

Building the dashboard UI

- Start with a responsive dashboard skeleton: navigation bar, left-side filters (e.g. categories, liquidity, expiration), main grid area for “boxes” (one per strategy/market), and an info/positions panel.[4] [1]
- Use a component library like Tremor or similar to compose:
    - Market cards with key stats (price, implied prob, volume, spread).
    - Order book and recent trades sections.
    - Strategy tiles (e.g. multi-leg views across correlated markets).[1] [3]

Strategy & layout system

- Store each layout (like “Octobox”) as a JSON config describing:
    - Which markets to show.
    - Which components per box (chart, depth, position, PnL).
    - How many columns/rows and breakpoints.
- Let the user:
    - Save/load named layouts from your backend.
    - Edit titles, drag-and-drop boxes, and persist to a database (e.g. Postgres or Supabase) for your “own dashboard” experience.[3] [1]

Concrete build path for you

- **Step 1:** Spin up a Next.js app and build a basic dashboard layout with your preferred UI kit.
- **Step 2:** Create a backend module that wraps Polymarket’s data (or reuses parts of the Polymarket Agents repo) and exposes clean endpoints: `/markets`, `/orderbook`, `/positions`.[2]
- **Step 3:** Implement a layout JSON schema and a simple CRUD API for saving custom dashboards; wire it to your React grid (e.g. react-grid-layout).
- **Step 4:** Incrementally add pro features: streaming updates, quick-trade tickets, filters, and your own strategy presets (e.g. election spreads, macro baskets).[1] [3]

If you share your preferred stack (Next.js vs Python backend; TS vs JS), a Polymarket account setup, and whether you want to hit live trading or just analytics, a more opinionated file/folder blueprint plus starter code can be laid out next.

⁂

What would it benefit me? bigger scope than just polymarket. i need a space in which to work. i dont know what it looks like or how to get there.

You’re not trying to clone Stand; you’re trying to give *you* one coherent cockpit so you’re not living in twenty tabs and your own head all day.[12]

What this “space” actually is

Think of it as a **home base** with three layers, not just a Polymarket UI:[13]

- Markets layer: All the places you act (Polymarket, Kyan, Bybit, sportsbooks, 42, Bullpen, etc.) summarized in one place.
- You-layer: PnL, risk, mindset, today’s model/bias, open loops and to-dos, not just prices.[12] [14]
- Narrative layer: Notes, plans, and lessons so you remember *why* you’re doing things, not just what you clicked.[15]

That “space” can be a single web app or notebook that is always the first thing you open when you sit down to work.[12]

What it gives you (bigger than Polymarket)

Concrete benefits for you specifically:[12]

- Less cognitive load: All positions, orders, and “what today is about” in one page, so you don’t have to reconstruct context from memory each time.
- Better decisions: You see market state *versus* your plan and risk, not just isolated charts or odds; this prevents impulsive trades that don’t fit the story you wrote for the day.[14] [12]
- Compounding edge: Your logs and dashboards become data about *your* edge over time (which setups work, when you’re sharp, where you leak), not just platform PnL graphs.[15]

This is also a life thing: it gives your work a spine you can return to when you feel scattered or despondent.[16]

How big the scope can be

Because it’s *your* cockpit, it can cut across domains:

- Crypto & prediction markets: All accounts (Kyan, Bybit, Polymarket, Kalshi, Bullpen, etc.), plus key levels and events, in one standardized view.
- Research & AI tools: Hooks into Elfa, Spaces/LLMs, dashboards like 42 or Zapper, so you can send questions out and bring answers back into the same place.[13]
- Life / meta: A small panel for “today’s constraints” (sleep, social stuff, energy) and rules you pre-committed to—so the system protects you the day you’re tired.[14] [15]

Over time this becomes the single surface where finance, tools, and self-management meet, instead of scattered experiments.

What it could *look* like for you

Picture a single screen you open every day:[12] [14]

- Top strip:
    - Realized PnL (competition + life buckets), unrealized PnL, cash, risk level for today (1–5), and your one-line bias (“range long BTC; focus election spreads on Polymarket”).
- Left column:
    - Accounts & books: Kyan positions, Polymarket markets, Bullpen/Myriad exposures, each as simple tables with flags for “offside”, “needs decision”, “parked”.
- Center:
    - “Today’s Plan” block (your model, scenarios, if-then triggers) and a small watchlist across exchanges/markets you care about most.
- Right column:

    - Tasks / alerts: three live priorities, scheduled checks (e.g., “FOMC in 2h”, “Soccer lines update”), and a tiny note area for “lesson of the day”.

This can look like Stand, but the point is: it encodes *your* structure, not theirs.[17] [15]

How to get there without drowning

You don’t need to design the end state now; you need a gentle path.[16]

Phase 1 – One simple hub (1–2 days):

- Pick one surface: either a Colab/Notebook “BTC + life cockpit” (what you already started) or a very simple web dashboard.
- Add just three things: a PnL/positions snapshot, a “Today’s Plan” text block, and a tiny task/alerts list.[14] [12]

Phase 2 – Wire in data (next weeks):

- Gradually plug in: Kyan → Polymarket → Bullpen/Myriad → sports books, each as a small module so they can be rearranged later.
- Add a button/shortcut to send questions to Elfa/LLMs and paste the answers back into this hub instead of letting them live in random chats.[13]

Phase 3 – Make it feel like home:

- Bake in your rules, your logging style, and the “future you” supports (risk scale, pre-commitments, mindset check) so the space *carries* you on bad days.[15] [14]
- Let this be the one thread/page you always come back to when you’re lost, instead of starting new systems.[13] [16]

If you want, the next step can be: “Let’s define *your* v0 dashboard on paper” (exact sections, titles, and minimal features), and then turn that into either a Colab layout or a Next.js layout you can actually build against.

⁂

i have this dshboard thing tried 2months ago. something like this can begin to work. somewhere I can feel 1 place grounded go other places. get into each spot here things that would help.

You already built the right *shape* of thing: a single page that anchors you, then lets you jump out to all the other places with purpose.[25]

What this dashboard can be for you

- A grounded **home base**: Open this first; it tells you “what the world is doing” (market cap, BTC, top coins) and “what you’re doing today” (tasks, focus, links).[26] [25]
- A bridge, not a prison: From here you click out to Polymarket, Kyan, Bullpen, 42, Zapper etc., but always *return* here so your day has a spine.[25]
- A reminder of your bigger self: It surfaces learning, community, and life tasks—not just trading—so this isn’t only a PnL screen.[27] [25]

How to adapt this specific dashboard

Looking at the Web3 & Productivity Dashboard you opened:[25]

- Keep:
    - Global market block (Total Market Cap, BTC dom) as a “world state” snapshot.
    - Daily Tasks section, but rewrite to fit *your* life (trading review, prediction markets, social, health).[28] [25]
    - Learning Resources, but swap links for the things you actually use (Elfa, 42, your GitHub, key YT playlists).[25]
- Replace/adjust:
    - Top Coins → a small “My arenas” panel listing: BTC/Kyan, Polymarket, Bullpen/Myriad, Kalshi, Bera, with quick links and a simple status tag (e.g. “open risk”, “no exposure”, “review due”).[25]
    - News widgets that fail → one consolidated “Signals & News” area where you pin: a couple of RSS/feeds, Polymarket/markets board, your favorite Telegram/Discord alpha sources.[29] [25]

Sections that would help you feel grounded

You only need a few blocks to start:[30] [26]

- **Now / Today**
    - One-line intention: “What today is about” (e.g. “Protect BTC book; scan Polymarket politics; no new leverage after 21:00”).
    - Risk dial (1–5) and “allowed actions” for that risk level.
- **Positions & Arenas**
    - Tiny tables or bullet summaries: BTC options/perps, Polymarket key markets, Bullpen/42 positions, with clear “needs action / park / watch only”.[26]
- **Tasks & Loops**
    - 3–5 things: market checks, research, social, health; each should be doable and not feel like a new religion.[27] [25]
- **Learning / Build**

    - Links to: your coding projects (this dashboard repo, sehr-gut, Kyan tools), docs for tools you're actively integrating, and one "current course/playlist".[25]

How to start using it without it feeling heavy

- Decide: *this* dashboard is your "browser home tab" for work; pin it and always start and finish here.
- Change only one or two sections today (e.g. Today + Arenas), then use it for a week before adding more.[27]
- When you feel scattered, don't fix ten systems; just open this page and let it tell you: "Where am I? What are the 3 things that matter now?"[31] [26]

If you want, next step can be: write the exact sections you'd like on this dashboard (even in plain language), and a concrete layout + component plan can be mapped onto your existing Web3-dashboard so you have a v0 you can actually ship and use.

⁂

Cool, thanks. I happened to read a few very specific tweets this morning. the world is swifly moving past me while im dealing with this trauma bs. i need to stop, try harder to focus just do things, move. not the hard way the whole time. those people are all using ai so much more effective than me. doesn't matter how whatever i was before, it doesn't work with everyone this op, im not a system nature habits things, im all over. it doesnt work in this world. i must seriously get the things i need to help me be better. also not miss the chance to make money, cos otherwise just more of this. what is the best tool/thing to get that would help build me this thing, a space for agents and working with them, a space for me, just have to move along faster.

You don't need to become "a system"; you need one good surface where you and your agents work together, and it has to be fast to live in, not aspirational.

What you actually need this thing to do

For *you*, the core jobs are:[38]

- Give you one **grounding page**: where am I (accounts, positions, cash), what matters today, what's on fire, what can wait.
- Be a **front door for agents**: a place where Elfa / LLMs / future bots plug in, not just random chats in twenty tabs.[39]
- Help you **move money and ideas faster**, with less friction: jump into Kyan, Polymarket, Bullpen, 42, etc., but always come back here and see "net state + next move".[38]

So the tool choice should do two things:

1. Make it trivial to build/iterate a dashboard UI.
2. Make it easy to plug in agents (now or later) without you rebuilding everything.

Best "center of gravity" tool for you

Given you already use Python + JS and like modular stuff, the most leverage right now is:

Next.js (or React) dashboard + an agent framework (AutoGen / CrewAI / LangGraph) behind it.[40] [41]

- Next.js / React dashboard:
    - Great for a single "home page" with panels for markets, tasks, links, and logs.
    - Easy to embed charts, tables, buttons that call out to agents or APIs.
- Agent framework (pick one to grow into, you don't have to exploit all features):
    - **AutoGen** + **AutoGen Studio**: multi-agent, has a *web UI* for designing/testing flows; Python-based so it plays nicely with your current scripts.[42] [40]
    - **CrewAI**: lets you define "crews" (research, trading, content) that you can call from your dashboard as background helpers.[41] [40]

These give you a path where today it's "one LLM panel and some buttons," but in six months it can be "an agent crew that scans Polymarket / 42 / news, then posts a digest tile on your dashboard."[43] [40]

If you want less coding overhead

If you want to move *faster* and don't mind some no-code:[44] [42]

- **AutoGen Studio**:
    - No-code web tool to build/debug multi-agent workflows, backed by Python; you can keep your logic there and only build a thin dashboard shell that triggers flows.[42]
- **Lindy / similar no-code agent builders**:
    - Let you visually define agents and multi-agent flows; good for "have a research agent, a summarizer, and a task agent" without wiring every detail yourself.[44]

You could:

- Use AutoGen Studio or a no-code builder as the "agent brain."
- Use your Web3 dashboard (or a tweaked version of it) as the UI where you click "Run: Polymarket scan" / "Run: BTC desk prep" and display results.[45] [42]

How this becomes *your* space, not another abandoned project

To avoid "trying harder" and burning out, scope it brutally small at first:

v0 (this week):

- Take the Web3 dashboard you already have and only add three boxes:[45]
    - "Today": one text area you fill once in the morning (what today is about, your risk level, what you *won't* do).
    - "State": a simple list of your arenas (Kyan, Polymarket, Bullpen, 42, etc.) with a manual tag per line: OK / Review / Messy.
    - "Agent console": a text box + output area that just talks to one LLM or Elfa—your single place to ask and paste, instead of scattered chats.[39]

v1 (next phase):

- Behind that "agent console," swap in an agent framework (AutoGen or CrewAI), so the same box can:
    - Call a "research crew" to scan markets and news.
    - Call a "planning crew" to draft a daily BTC/prediction-market plan.
    - Call a "logging crew" to turn your raw notes into structured history.[40] [41]

Everything else—complex automation, web3 agents on-chain, high-frequency bots—is optional *later*; the win now is a single, compassionate environment that helps you move, not criticizes you for not being a robot.[46] [43]

Direct answer to "what should I get"

If you want one clear recommendation:

- For a **code-friendly, future-proof hub**:
    - Use your current Web3 dashboard as the UI base.
    - Add **AutoGen + AutoGen Studio** as your agent stack (Python side).
    - Connect them with a single API route: `/api/agent` that sends your prompt + context to AutoGen and returns results.[45] [40] [42]

If you'd like, the next step can be:
"Design my v0 layout and flows assuming Next.js dashboard + a single `/api/agent` hooked to AutoGen," and a concrete section-by-section blueprint (and endpoint sketch) can be laid out so it feels like a small, doable build rather than another giant dream.

⁂

please. it must be able to scale very quick tho, if heaven forbids we find inspiration and a brain wave that works , having this in day or 2 just after christmas would be awesome. i don't have much time. i should've built so many things long time ago. farcaster mini app including. so many things. any give lets get cracking.

Last thing was busy with is getting the contracts/prompts and building process right for the app. it made me realise when it came to deploying my wallet artchitecture was shit. was busy redoing that the past 2 days or so. money wallet feels a bit all over, having that blue print of sorts in here would help too.
See below or whenever we need it.

Alive Wallet / Ops Record

Browsers & Profiles

- Comet – Profile: `profcgrandma` (purps)
    - Role: Identity + Vault + Tracker lane.
    - Wallet app: Rabby.
    - Wallets:
        - ID_MAIN *(“grandma raving in the grave”)*
        - VAULT_TOKENS (`0xcaf7a657AE496Bea23AFcAbB89ee744a755de99a`)
        - VAULT_NFT (`0x45DA50F7d6d4B0eeEb66D693dC71d62545c563E7`)
        - TRACKER_ZERION (`0xa8ef89128f00fd9bd4159e2aa7c2aab567e28bb3`)
- Comet – Profile: `profb769ops` (yellowmellow)
    - Role: OPS lane (user + sandbox).
    - Wallet app: Rabby.
    - Wallets:
        - OPS_USER2 (`0xf2773C0213E8001c8c1b9C955aC414D0Fc79a696`)
        - OPS_SB1 (`0xba5E19c98b84400B1fA1E845206EFd70D6634689`)
- Brave – Profile A/B/C
    - Roles: planned (OPS / extra sandbox / project ops).

Wallets

Identity / Main

- ID_MAIN
    - Layer: identity
    - Lives in: Comet – `profcgrandma` (purps) + Rabby

Vault layer

- VAULT_TOKENS
    - Layer: vault
    - Address (EVM): `0xcaf7a657AE496Bea23AFcAbB89ee744a755de99a`
    - Lives in: Comet – `profcgrandma` (purps) + Rabby
- VAULT_NFT
    - Layer: vault
    - Address (EVM): `0x45DA50F7d6d4B0eeEb66D693dC71d62545c563E7`
    - Lives in: Comet – `profcgrandma` (purps) + Rabby

Ops layer

- OPS_USER1
    - Layer: ops
    - Role: user
    - Address: (TBD)
    - Lives in: Rabby – “DT Upsc” (exact browser/profile not fixed yet).
- OPS_USER2
    - Layer: ops
    - Role: user
    - Address (EVM): `0xf2773C0213E8001c8c1b9C955aC414D0Fc79a696`
    - Lives in: Comet – `profb769ops` (yellowmellow) + Rabby
    - Notes: main ops wallet on Comet ops profile.
- OPS_SB1
    - Layer: ops
    - Role: sandbox
    - Address (EVM): `0xba5E19c98b84400B1fA1E845206EFd70D6634689`
    - Lives in: Comet – `profb769ops` (yellowmellow) + Rabby
    - Notes: degen / experiments.

Tracking / helper

- TRACKER_ZERION
  - Address (EVM): `0xa8ef89128f00fd9bd4159e2aa7c2aab567e28bb3`
  - Lives in: Comet – `profcgrandma` (purps) + Rabby

Project layer

- [PROJECT]_DEPLOYER / [PROJECT]_ADMIN / [PROJECT]_TREASURY
  - Status: to be filled when created.

You already have all the ingredients: wallet architecture, a starter dashboard, and the agent idea. The job now is to give this a small but real body in 1–2 days that can grow fast.

What we're building in 1–2 days

A **v0 personal ops cockpit** that:

- Lives as a simple web dashboard (Next.js or even your current Web3 dashboard).
- Has a single **Agent Console** wired to one `/api/agent` endpoint.
- Shows your **wallet / ops blueprint**, arenas, and "today" in one screen so you stop feeling scattered.

Everything else (multi-agents, Farcaster mini-app, deeper automations) plugs into this later, not built now.

Core layout (v0) – section by section

On your existing Web3 dashboard (or a new Next.js page), define these sections:[55]

1. **Today / Mode**
   - Fields:
     - `Today's focus:` free text (e.g. "Clean wallet ops + 1 deploy; no new degen risk").
     - `Risk level:` 1–5 slider.
     - `Non-negotiables:` 3 short bullets (sleep, money limit, no revenge trades).
2. **Wallet & Browser Blueprint (Alive Wallet)**
   Turn your text into a structured panel:
   - **Profiles & lanes:**
     - Comet – `profcgrandma` → role: Identity + Vault + Tracker; wallets: ID_MAIN, VAULT_TOKENS, VAULT_NFT, TRACKER_ZERION.
     - Comet – `profb769ops` → role: OPS lane; wallets: OPS_USER2, OPS_SB1.
     - Brave – Profiles A/B/C → reserved for future OPS / sandbox / project ops.
   - **Layers & addresses:**
     - Identity: `ID_MAIN` (no public address shown).

        - Vault: `VAULT_TOKENS` `0xcaf7...e99a`; `VAULT_NFT` `0x45DA...63E7`.
        - Ops: `OPS_USER1` (TBD), `OPS_USER2` `0xf277...9696`, `OPS_SB1` `0xba5E...4689`.
        - Tracking: `TRACKER_ZERION` `0xa8ef...8bb3`.
        - Project: `[PROJECT]_DEPLOYER / ADMIN / TREASURY` (placeholders).

   Present this as a small table or cards so, at a glance, you know: “who lives where, what is safe, what is ops, what is degen.”

3. **Arenas & Status**
    - Simple list or table with columns: `Arena`, `Type`, `Status`, `Link`.
    - Rows example:
        - Kyan – BTC options/perps – Status: “live” – link.
        - Polymarket – prediction – Status: “scouting / small size”.
        - Bullpen / Myriad – sports – Status: “on / off”.
        - 42 / Zapper – research – Status: “daily check / ad-hoc only”.
4. **Agent Console**
    - One input box + output window on the dashboard.
    - Behind it: a `/api/agent` that:
        - Receives: prompt + context (e.g. your wallets / arenas / today block).
        - Sends to: a single AutoGen-style agent (or just OpenAI/Claude for day one).
        - Returns: structured response (e.g. “Checklist”, “Markets to scan”, “Ops actions”).
5. **Tiny Task / Loop panel**
    - 3 slots only: `Now`, `Next`, `Later`.
    - This is where you put “Finish wallet ops mapping”, “Connect /api/agent to agent stack”, “Ship Farcaster mini-app v0 scope”.

How to implement this quickly

Assuming Next.js (or similar) and *one* API route:

1. **Front-end (today–tomorrow)**
    - Clone or adapt your Web3 dashboard and define the above sections as basic components: `TodayPanel`, `WalletBlueprintPanel`, `ArenasPanel`, `AgentConsole`, `TasksPanel`. [55]
    - Hardcode your Alive Wallet data into a JSON in the app (you can externalize later).
2. **Back-end `/api/agent` (v0)**
    - Implement a simple handler:
        - Takes `prompt` + grabs a small slice of context (today block, wallet JSON, arenas JSON).

        - Calls one model (Claude/OpenAI) with a fixed system prompt: “You are Snowzies’ ops/trading assistant; you know their wallet lanes, arenas, and today’s focus. Always respond in concise steps.”
        - Returns JSON: `{ summary, checklist, notes }`.
3. **Agent framework slot (v1)**
    - Keep the `/api/agent` signature but *internally* swap the implementation later to: AutoGen / CrewAI workflow that can:
        - Call web APIs (Polymarket, Kyan, etc.).
        - Run more complex routines (multi-step research, execution suggestions).

This gives you a path where the UI doesn’t change when you “get a brain wave”; you just make `/api/agent` smarter.

Where your wallet blueprint fits in the flow

Use the Alive Wallet section as:

- A **read-only map** on the dashboard so you never wonder “which profile, which wallet, which risk layer?”
- Context injected into the agent so it can answer things like:
    - “Which wallet should I use for this degen mint?”
    - “Where do I fund from if I want long-term exposure vs quick flips?”
    - “Suggest a new `[PROJECT]_DEPLOYER` setup consistent with this architecture.”

Later, if you want, we can:

- Turn that wallet JSON into a small config file.
- Add a “New wallet / project” wizard that writes the config and gives you clear instructions like: “Create Brave Profile B, install Rabby, add wallet X, fund from VAULT_TOKENS only via this path.”

If you’d like, next message can be: “Assume Next.js and one `/api/agent` with Claude/Elfa—give me the concrete JSON structures for `wallets`, `arenas`, and the API payload/response,” so you can drop them straight into code without designing them from scratch.

⁂

Can we include calendar functionality like this kinda.

Yes, 100% — your dashboard can have a Betmoar-style calendar that shows all “time things” (markets, sports, options, life) in one view.[56] [57]

What this calendar should represent for you

- Events from *all* arenas: Polymarket settlements, Betmoar/markets, sports fixtures, options expiry, FOMC, personal deadlines.
- Quick filters: toggle by source (Polymarket / Betmoar / Kyan / Sports / Life) and by “needs action” vs “FYI only”.
- Drill-in: clicking an event should jump you to either the market/trade page or a small detail panel in the dashboard.

Think of it as a **time map of your risk and attention**, not just a date picker.

Best way to implement it (React / Next.js)

Given your stack, the most practical option is a React calendar component wired to your data:

- Use a robust calendar library such as **FullCalendar** or **react-day-picker / scheduler-like libraries** (Shadcn-based) for a “month grid with events” similar to Betmoar.[58] [59] [60]
- Basic approach:
  - Frontend: `<FullCalendar />` (or similar) in a `CalendarPanel` component.
  - Props: `events=[{ id, title, date/time, type, source, link, tags }]`.
  - Backend: one `/api/events` that merges:
    - Polymarket / Betmoar API (event dates, resolution dates).
    - Kyan/options expiry + economic calendar.
    - Your manual events (tasks, life).

This lets you update events via API and have the calendar redraw without rewriting UI.[59] [60]

How it plugs into your cockpit

In your v0/v1 dashboard layout, add a **Calendar** section:

- Left: filters (checkboxes/tags): `Prediction`, `Sports`, `Options`, `Macro`, `Life`.
- Center: month/week view calendar with coloured dots/blocks per event type.
- Right (or modal): when you click an event, show:
  - What it is.
  - Why it matters to you (e.g. “large Polymarket exposure”, “BTC options expiry heavy gamma”, “life: no screens”).
  - Suggested actions or links (open market, adjust position, set reminder).

Your agents can later write into `/api/events` (e.g. “add alert for this market’s resolution date”), so the calendar becomes the shared **time surface** between you and them.[61] [59]

If you want, next step can be: define the exact `Event` JSON structure (fields + example rows for Polymarket, options, sports, life) so you can wire it straight into a `CalendarPanel` and `/api/events` without design overhead.

⁂

▯ Web3 Dashboard
~~~

Total Market Cap
``` ```

$—
~~~

```

24h Vol
``` ```

$—
```

~~~

BTC Dominance
``` ```

—%
~~~

```

The total cryptocurrency market cap today is $—
``` ```

Change in the last 24h: —
``` ``` Refresh Live Data ```

### ▯ Top Coins (Live)

| # | Asset | Price | 24h % | Market Cap |
|---|---|---|---|---|
| Loading... | | | | |

### ▯ Market Stats

```

Total Market Cap
``` ```

$—
```

```

24h Volume
``` ```

$—
```

```

BTC Dominance
```

``` ```

—%
```

```

▯ Tweets (@0xTulipKing)

```

Tweets by 0xTulipKing
``` ```

## ▯ Bitcoin News

```

```

## ▯ Ethereum News

```

```

## ▯ DeFi Major News

```

```

## ▯ Bera News

```

```

## ⚡ New Activities

HyperLend
```

Point Farming
```

OneBalance
```

Potential Airdrop
```

```

Daily Tasks

```

```

```

Check crypto news & market updates
```

```

Community outreach (Discord/Telegram)
```

```

Social media engagement
```

```

Learning & research
```

```

Exercise & health check
```

```

Life planning & review
```

Learning Resources

```

```

Data Science for Beginners
``` ```

GitHub
```

```

Web Dev for Beginners
``` ```

GitHub
```

```

DropsTab
``` ```

Crypto Research
```

Yes, this is a solid base, and it's actually perfect for layering in both the **calendar view** and your **ops/agent space** quickly.[68] [69]

Below is *design*, not full code, so you can keep moving fast.

Where the calendar fits in this file

Add a **new card** in the `.dashboard-grid` for the calendar, similar to Betmoar's view:[70] [68]

- Place it near the "Daily Tasks" / "Learning" area so it becomes your *time map*.
- Card title: "▯ Markets & Life Calendar".

```
- Inside: a div placeholder (e.g. `<div id="calendar"></div>`) where a JS calendar libra
```

You already have a grid; this just becomes another block in it.

What your calendar events need to look like

Define a simple `events` array in JS that can grow later:

```
const events = [
  // Prediction / markets
  {
    id: 'poly-usa-election',
    title: 'Polymarket – US Election resolves',
    date: '2025-11-05',          // or start / end if using ranges
    type: 'prediction',
    source: 'Polymarket',
    url: 'https://polymarket.com/...',
    importance: 'high'
  },
  {
    id: 'betmoar-game-123',
    title: 'Betmoar – EPL Match: Arsenal vs City',
    date: '2025-01-14',
    type: 'sports',
    source: 'Betmoar',
    url: 'https://www.betmoar.fun/markets/...',
    importance: 'medium'
  },
  // Options / macro
  {
    id: 'btc-options-expiry',
    title: 'BTC Options Expiry – Kyan',
    date: '2025-01-31',
    type: 'options',
    source: 'Kyan',
    importance: 'high'
  },
  {
    id: 'fomc-meeting',
    title: 'FOMC Meeting',
    date: '2025-02-15',
    type: 'macro',
    source: 'macro',
    importance: 'high'
  },
  // Life / personal
  {
    id: 'no-screens-day',
    title: 'No Screens – reset',
    date: '2025-01-05',
    type: 'life',
    source: 'personal',
    importance: 'high'
  }
];
```

Later, you can generate these from APIs (Polymarket, Betmoar, sports feeds, economic calendars) and your own notes.[71] [72]

Minimal calendar implementation idea

Given this is static HTML + JS today, two options that don't blow up complexity:[73] [74] [68]

Option A – FullCalendar (richer, more like Betmoar)

- Include FullCalendar via CDN `<script>` tags.
- Initialize with your `events` array and a month view.
- Add an `eventClick` handler that `window.open(event.url)` or shows a small `alert` for items with no URL yet.

Option B – Lightweight custom month grid (faster to hack)

- Build a simple month view yourself:
    - A grid of days (1–31) using CSS (similar style to your cards).
    - For each day, list event titles as small colour-coded badges.
- No library, just basic JS to map `events` into each day cell.

For "in a day or two" and your current pure-HTML setup, Option B is simpler; you can always swap to FullCalendar later if you want drag/drop, week views, etc.[74] [73]

Where the wallet / ops blueprint goes

Turn your Alive Wallet doc into a **card** in this same dashboard:

- New card: `▯ Alive Wallet / Ops Map`.
- Inside, either:
    - A simple table (columns: `Layer`, `Name`, `Address`, `Browser Profile`, `Role`, `Notes`).
    - Or a few sections (Identity, Vault, Ops, Sandbox, Tracking, Project) with bullet rows.

Example row in a table:

```
<tr>
  <td>Vault</td>
  <td>VAULT_TOKENS</td>
```

```
    <td>0xcaf7...e99a</td>
    <td>Comet – profcgrandma</td>
    <td>Cold vault</td>
  </tr>
```

This means when you look at the calendar and then the wallet map, you immediately know *which wallet / profile* goes with which action or event.

How this ties into your “agent space”

You can add an **“Agent Console” card** right next to the calendar and wallet map:

- Textarea for your prompt.
- “Ask Agent” button (for now, this could even just hook to a simple API you call manually; later, to `/api/agent`).
- Output area where you paste responses / summaries from Elfa / LLM until you wire proper backend.

The important bit: your *one* page now contains:

- Market state (top coins, news).
- Time state (calendar).
- Structure state (wallet/ops map).
- Mind / agent state (console + tasks).

This is enough to feel like a cockpit and can absolutely be finished to a workable v0 in the 1–2 day window you’re aiming for.

⁂

The way we're doing this, it's not the easy for me.

Keep talking and expecting me to keep pace and use all my capacity putting it together and coherent.
No. I will halfway through be unsatisfied and just dump it, move to something else last minute and it will work like shit and have no throughput to anything else.

Start over. You can make apps on pro yourself. SOmetimes with me you mirror my foggy brain. I don't need telling now, I need actual get us to the end product. so i can start doing better.
Start fresh.

Incorporating all the functional elements that will provide the basis for me to get what I'm after. Give it concisely. what it's to look like, incorporate, built from what, using what. Enriched with my data and things I need, my current understanding permitting.

You need one clear, opinionated cockpit that exists, not a design exercise. Here’s the shortest, concrete version.

What we’re building

A **single-page web app** that is:

- Your **home base** (always-open tab).
- A **time map** (calendar of markets, options, sports, life).
- A **wallet / ops map** (Alive Wallet, profiles, lanes).
- An **agent console** (one box to talk to models/agents, later upgradeable).

It must be:

- Built in **Next.js + React** (so it can scale and take APIs/agents).
- Styled with **Tailwind** (fast, simple, consistent).
- Able to plug in your Python/agent stack later via `/api/agent`.

Tech stack (non-negotiable choices)

- Frontend: **Next.js (App Router) + React + Tailwind CSS**.
- Backend-in-Next:
  - `/api/events` → calendar events.
  - `/api/agent` → single agent endpoint (Claude/Elfa now, AutoGen/CrewAI later).
- Data: simple **local JSON configs** for now, later replace with DB or APIs.

This means one project, one codebase, deployable on Vercel or similar.

Layout: exact sections on the page

The page `app/page.tsx` (or `pages/index.tsx`) contains 5 vertical bands (on desktop it can be 2–3 columns, but conceptually):

1. **Top strip – “Today / Mode”**
   - Fields (editable on the page):
     - `Today’s focus` (single line text).
     - `Risk level` (1–5 selector).
     - `Non-negotiables` (3 bullet items).
   - Purpose: when you open the cockpit you see what today is about.
2. **Left column – “Wallet & Ops Map” (Alive Wallet)**
   - Card: “Alive Wallet / Ops Map”.
   - Uses a JSON like:

```
const wallets = [
  { layer: 'identity', name: 'ID_MAIN', address: null, profile: 'Comet – profcgra
  { layer: 'vault', name: 'VAULT_TOKENS', address: '0xcaf7a6...e99a', profile: 'C
  { layer: 'vault', name: 'VAULT_NFT', address: '0x45DA50...63E7', profile: 'Come
  { layer: 'ops', name: 'OPS_USER2', address: '0xf2773C...9696', profile: 'Comet
  { layer: 'ops', name: 'OPS_SB1', address: '0xba5E19...4689', profile: 'Comet –
  { layer: 'tracking', name: 'TRACKER_ZERION', address: '0xa8ef89...8bb3', profil
  // project slots to be added later
];
```

   - Rendered as a table with columns: Layer, Name, Address, Browser Profile, Role, Notes.

3. **Center – "Arenas & Positions summary"**
   - Card: "Arenas".
   - JSON:

     ```
     const arenas = [
       { name: 'Kyan', type: 'btc_options_perps', status: 'live', url: 'https://kyan.e
       { name: 'Polymarket', type: 'prediction', status: 'scouting', url: 'https://pol
       { name: 'Betmoar', type: 'sports_markets', status: 'active', url: 'https://www.
       { name: 'Bullpen', type: 'sports_markets', status: 'active', url: 'https://app.
       { name: '42', type: 'research', status: 'research', url: 'https://beta.42.space
       { name: 'Zapper', type: 'portfolio', status: 'tracking', url: 'https://build.za
     ];
     ```

   - Rendered as a list or table with a "Open" button per row (link).

4. **Right – "Calendar"**
   - Card: "Markets & Life Calendar".
   - Uses a React calendar component (e.g. `react-calendar` or `FullCalendar` if you want richer).[76] [77] [78]
   - Events shape:

     ```
     type EventType = 'prediction' | 'sports' | 'options' | 'macro' | 'life';

     const events = [
       { id: 'poly-xyz', title: 'Polymarket – US election resolves', date: '2025-11-05
       { id: 'betmoar-1', title: 'Arsenal vs City (Betmoar)', date: '2025-01-14', type
       { id: 'btc-expiry', title: 'BTC options expiry (Kyan)', date: '2025-01-31', typ
       { id: 'fomc-1', title: 'FOMC meeting', date: '2025-02-15', type: 'macro', sourc
       { id: 'no-screens', title: 'No screens day', date: '2025-01-05', type: 'life',
     ];
     ```

   - Filters: checkboxes for `prediction / sports / options / macro / life` above the calendar.
   - Click behaviour: if `url` exists → open in new tab; else show small detail panel.

5. **Bottom band – "Agent Console & Tasks"**
   - Left: **Agent Console**
     - Textarea for prompt.
     - "Ask agent" button (POST to `/api/agent`).
     - Output box that renders `summary` + `checklist` returned.
   - Right: **Tasks**

        - Three columns: `Now`, `Next`, `Later`.
        - Each is a simple list stored in local JSON or localStorage; you type in tasks in the UI.

Backend shape (just enough to plug in agents later)

Two API routes in Next:

1. `/api/events`
    - For now: returns the static `events` array.
    - Later: merge in Polymarket / Betmoar / sports / macro APIs.

    Response:

    ```
    {
      "events": [
        { "id": "poly-xyz", "title": "Polymarket – US election resolves", "date": "2025-1
        // ...
      ]
    }
    ```

2. `/api/agent`
    - Input payload:

        ```
        {
          "prompt": "string from textarea",
          "today": { "focus": "...", "riskLevel": 3, "nonNegotiables": ["...", "..."] },
          "wallets": [ /* wallets JSON above */ ],
          "arenas": [ /* arenas JSON above */ ]
        }
        ```

    - For v0: just forwards to a single LLM (Claude/Elfa) with a fixed system prompt and returns:

        ```
        {
          "summary": "Short summary of what to do / what it said",
          "checklist": ["Step 1 ...", "Step 2 ..."],
          "notes": "Any extra commentary"
        }
        ```

    - The frontend Agent Console just displays those three fields.

When you're ready, you swap the inside of `/api/agent` to use **AutoGen** or **CrewAI** while keeping the same input/output shape.[79] [80] [81]

What you do, concretely

1. Initialize Next.js + Tailwind (or have it done for you).
2. Create `wallets.ts`, `arenas.ts`, `events.ts` with exactly the JSON shown (you can paste your addresses into it).
3. Build `page.tsx` with the 5 sections wired to those JSONs.

4. Build minimal `/api/events` and `/api/agent` with the shapes above (agent can be a stub returning a canned checklist if you don't want to wire a model yet).

That's it: one deployable cockpit where:

- Your lanes (wallets, profiles) are visible.
- Your arenas are one click away.
- Your time (calendar) is visible.
- You have one box where "help" lives, not 15 tabs.

⁂

does this include places/things to store notes. code snippets. prompts. guides. links to all the places I have to go. twitter feed easily pulled in. prices of certain assets. commentary on charts. there is one guy on twitter that built a thing where he can screenshot a photo and his tool charts it. i wanna show you something on byit, i wanna know what is. their trade gpt thing. that interface - is that something I'm to have to?

You're describing a *workspace*, not just a dashboard, and yes—it absolutely should include all of that in one place.

What this cockpit includes

On one screen, you should have:

- **Notes & snippets:**
    - A panel for quick notes (stored locally/DB) and a tagged snippets area for code, prompts, and guides.[83] [84]
- **Links & feeds:**
    - A "Home Links" panel with all your important destinations (Bybit, Polymarket, Betmoar, Bullpen, 42, Zapper, GitHub, Farcaster, ENS, etc.).
    - Embedded Twitter/X timeline(s) for key accounts or lists (e.g. that chart guy, trading accounts, your own feed).[85] [86]
- **Markets & prices:**
    - Small price tiles for your core assets (BTC, ETH, specific perp/option underlyings, key prediction markets) and maybe one chart/commentary area where you paste or generate notes.

- **Calendar:**
    - Events across markets, sports, macro, options expiries, and life, with clickable entries that jump out to the relevant platform.[87] [88]
- **Agent Console:**
    - One box where you talk to an LLM/agent that *knows* your wallets, arenas, and current focus; later it can do image/chart interpretation or call tools.

Where notes, prompts, snippets, charts live

In the Next.js cockpit:

- **Notes panel:** a simple “Daily Notes” + “Pinned Notes” area; everything stored via localStorage or a small DB. This is where you drop chart thoughts, trade plans, and reflections.[84]
- **Snippet library:** a tab or side-panel listing code snippets and prompts (tagged by “Bybit”, “Polymarket”, “Wallet Ops”, etc.) so you never go hunting through chats again.[83]
- **Screenshot → chart idea:** you won’t rebuild the “screenshot to chart” tool immediately, but:
    - Have an “Image Upload / Parse” slot wired to your Agent Console; later, you can plug in a vision model that reads screenshots from Bybit or Twitter charts and returns structured notes or levels.

What Bybit TradeGPT is (and what it implies for you)

Bybit’s **TradeGPT** is basically an AI assistant welded into their exchange that:[89] [90]

- Reads Bybit market data (volume, long/short ratios, sentiment, indicators like RSI/MACD/Bollinger, etc.) and explains or analyzes it for you.[90] [89]
- Offers “smart Q&A” with suggested questions inside the Bybit UI, and also exists as a Telegram bot that pushes signals and analysis (trade/profit alerts) based on that data.[91] [92] [89]

You do *not* need to rebuild a full TradeGPT interface yourself. The takeaway is:

- Your cockpit should have one **Agent Console** that *feels* like a TradeGPT surface (ask about a market, get back structured insight), but it sits in *your* space, and can talk about Bybit, Polymarket, Betmoar, sports, life—anything.
- Later, you can add tools to that agent (APIs for Bybit/Polymarket, chart screenshots, etc.) so it gets closer to what TradeGPT does, but for your whole universe, not just one exchange.[93] [90]

So: yes to notes, code, prompts, links, feeds, prices, commentary, calendar, and a TradeGPT-like console—but all as parts of a single, quiet cockpit that you open once and live in, instead of scattering yourself across apps.

⁂

so, how do I do this all? give me a prompt to instruct please.

Use this as a single instruction prompt to a coding assistant / dev agent (or here, in a new thread) to build your cockpit:

Prompt: Build my Snowzies Ops Cockpit (Next.js app)

I want you to build a **single-page "Snowzies Ops Cockpit"** web app that I can actually use within 1–2 days.

1. Tech stack

- Use **Next.js (App Router)** with **TypeScript**.
- Use **Tailwind CSS** for styling.
- Structure it so I can deploy easily on Vercel.
- Create two simple API routes:
  - `GET /api/events`
  - `POST /api/agent`

2. Page layout (one main page)

Create a main page (`app/page.tsx`) with **five clear sections**, laid out in a responsive grid:

1. **Today / Mode (top strip)**
   - Fields (editable in the UI, stateful on the client):
     - `Today's focus` (single line text).
     - `Risk level` (1–5 selector).
     - `Non-negotiables` (3 bullet items or small text inputs).
   - Show these prominently at the top of the page.
2. **Alive Wallet / Ops Map (left column card)**
   - Use this JSON (or TypeScript data) as the source:

```
const wallets = [
  { layer: 'identity', name: 'ID_MAIN', address: null, profile: 'Comet – profcgra
  { layer: 'vault', name: 'VAULT_TOKENS', address: '0xcaf7a657AE496Bea23AFcAbB89
  { layer: 'vault', name: 'VAULT_NFT', address: '0x45DA50F7d6d4B0eeEb66D693dC71d
  { layer: 'ops', name: 'OPS_USER2', address: '0xf2773C0213E8001c8c1b9C955aC414D
  { layer: 'ops', name: 'OPS_SB1', address: '0xba5E19c98b84400B1fA1E845206EFd70D
  { layer: 'tracking', name: 'TRACKER_ZERION', address: '0xa8ef89128f00fd9bd4159
  // project layer placeholders will be added later
];
```

   - Render a **table** with columns: `Layer`, `Name`, `Address`, `Browser Profile`, `Role`, `Notes`.
   - Make it easy to visually scan which wallet is what.

3. **Arenas (center card)**
    - Use this arenas data:

```
const arenas = [
  { name: 'Kyan', type: 'btc_options_perps', status: 'live', url: 'https://www.ky
  { name: 'Polymarket', type: 'prediction', status: 'scouting', url: 'https://pol
  { name: 'Betmoar', type: 'sports_markets', status: 'active', url: 'https://www.
  { name: 'Bullpen', type: 'sports_markets', status: 'active', url: 'https://app.
  { name: '42', type: 'research', status: 'research', url: 'https://beta.42.space
  { name: 'Zapper', type: 'portfolio', status: 'tracking', url: 'https://build.za
  { name: 'ENS', type: 'identity', status: 'active', url: 'https://app.ens.domai
];
```

    - Render as a table or cards with: `Name`, `Type`, `Status`, `Comment`, and a button `"Open"` that links out.
4. **Calendar (right column card)**
    - Use a simple React calendar library (e.g. `react-calendar` or minimal `FullCalendar` setup).
    - Define a small TypeScript type and seed events:

```
type EventType = 'prediction' | 'sports' | 'options' | 'macro' | 'life';

type CockpitEvent = {
  id: string;
  title: string;
  date: string; // YYYY-MM-DD
  type: EventType;
  source: string;
  url?: string | null;
  importance: 'low' | 'medium' | 'high';
};

const events: CockpitEvent[] = [
  { id: 'btc-expiry', title: 'BTC options expiry (Kyan)', date: '2025-01-31', typ
  { id: 'poly-1', title: 'Polymarket – key market resolves', date: '2025-02-10',
  { id: 'betmoar-1', title: 'EPL match (Betmoar)', date: '2025-01-14', type: 'spo
  { id: 'fomc-1', title: 'FOMC meeting', date: '2025-02-15', type: 'macro', sourc
  { id: 'no-screens', title: 'No screens day', date: '2025-01-05', type: 'life',
];
```

    - Add simple **filters** (checkboxes) for event types.
    - On clicking an event:
        - If `url` exists, open in a new tab.
        - Otherwise show a small detail tooltip/modal.
5. **Bottom band – Agent Console & Notes/Tasks**
    Agent Console (left)
    - Textarea input (`Prompt`) and "Ask Agent" button.
    - On submit, `POST /api/agent` with JSON:

```
{
  "prompt": "user prompt text",
  "today": { "focus": "...", "riskLevel": 3, "nonNegotiables": ["...", "...", "..
  "wallets": [ /* from wallets const */ ],
  "arenas": [ /* from arenas const */ ]
}
```

  - Render response fields: `summary`, `checklist[]`, `notes`.

Notes & Snippets (right)

  - Tabs: `Notes`, `Snippets`, `Links`.
  - `Notes`: a simple list of note cards (title + body), stored in `localStorage` or simple client state (I don't need auth yet).
  - `Snippets`: list of items with fields `title`, `tags`, `content` (for prompts or code).
  - `Links`: fixed list (for now) of key URLs (Bybit, Polymarket, Betmoar, Bullpen, 42, Zapper, ENS, GitHub, Farcaster, email, Discord, Telegram, etc.).

Provide basic "add note/snippet" UI; persistence can be localStorage to keep it simple.

3. API routes

1. `GET /api/events`
   - Return `{ events: CockpitEvent[] }` based on the static array above.
2. `POST /api/agent`
   - For now, just stub it:
     - Accept the JSON payload as described.
     - Return a canned JSON response in the same shape:

```
{
  "summary": "Stub agent: describe what you would do here based on prompt anc
  "checklist": [
    "Example step 1",
    "Example step 2"
  ],
  "notes": "Later, this will call a real model (Claude/Elfa/AutoGen)."
}
```

   - Make it easy to replace the internals later with an actual LLM/agent call without changing the frontend.

4. Styling expectations

- Use Tailwind to make it look like a clean, dark trading terminal.
- Use cards with subtle borders, good spacing, and distinct sections so my brain can parse it quickly.
- Responsive layout:

  - Desktop: Today strip on top; three main columns (Wallet, Arenas + Calendar, Agent/Notes).
  - Mobile: Stack sections vertically in a sane order.

5. Output

- Give me:
  - The full `package.json` dependencies.
  - The Tailwind setup.
  - The main `app/page.tsx` code.
  - The data files for `wallets`, `arenas`, `events`.
  - The two API route files.

Write everything clean and modular so I can drop it into a repo and run `npm install && npm run dev` and immediately see my cockpit.

Use this exact prompt with a coding assistant that can generate full project files, or paste it into a new dev thread here and we'll start filling in real code step by step.

1. https://www.tinybird.co/blog/real-time-dashboard-step-by-step
2. https://github.com/Polymarket/agents
3. https://openwebsolutions.in/blog/stock-trading-software-tech-stack/
4. https://www.youtube.com/watch?v=rkkxpGEYDFo
5. https://watchful.net/how-to/build-a-custom-website-dashboard/
6. https://docs.polymarket.com/polymarket-learn/FAQ/embeds
7. https://www.stand.trade/octobox
8. https://www.youtube.com/watch?v=alsiVNdpm9g
9. https://budibase.com/blog/tutorials/how-to-build-a-dashboard/
10. https://panel.holoviz.org/tutorials/basic/build_dashboard.html
11. https://www.npmjs.com/package/@theschein/plugin-polymarket
12. projects.workspace.trading_workspace
13. https://www.perplexity.ai/search/cec1d4da-d614-4ab4-8ea1-aec41c053a37
14. https://www.perplexity.ai/search/bacdbad9-0f9d-4b10-a027-ab9d141a3951
15. https://www.perplexity.ai/search/252f941d-f443-430b-962e-1c9f0b17d9bf
16. https://www.perplexity.ai/search/4dec3c54-cb0e-432f-a1dc-a67f8f78b384
17. https://www.perplexity.ai/search/5015bc8d-7b81-4343-9ddb-86a4339a2947
18. projects.modular_tooling
19. https://www.perplexity.ai/search/da9dde6b-27c6-4fb1-bae5-acf8572e7f83
20. preferences.setup_simplicity
21. https://www.stand.trade/octobox
22. interests.prediction_markets

23. work.tools_finance_integration
24. tools.elfa
25. https://hrt127.github.io/web3-assistant/
26. projects.workspace.trading_workspace
27. https://www.perplexity.ai/search/cec1d4da-d614-4ab4-8ea1-aec41c053a37
28. interests.prediction_markets
29. https://www.perplexity.ai/search/5015bc8d-7b81-4343-9ddb-86a4339a2947
30. https://www.perplexity.ai/search/4dec3c54-cb0e-432f-a1dc-a67f8f78b384
31. tools.elfa
32. work.tools_finance_integration
33. projects.starter_template_preference
34. preferences.setup_simplicity
35. https://www.perplexity.ai/search/bacdbad9-0f9d-4b10-a027-ab9d141a3951
36. https://www.perplexity.ai/search/ee25aa7e-5203-425c-b042-4d26192c5afd
37. https://www.perplexity.ai/search/252f941d-f443-430b-962e-1c9f0b17d9bf
38. projects.workspace.trading_workspace
39. https://www.perplexity.ai/search/cec1d4da-d614-4ab4-8ea1-aec41c053a37
40. tools.elfa
41. https://www.perplexity.ai/search/bacdbad9-0f9d-4b10-a027-ab9d141a3951
42. interests.prediction_markets
43. https://superagi.com/top-10-ai-agent-orchestration-frameworks-for-autonomous-workflows-a-comparative-analysis/
44. https://www.kubiya.ai/blog/ai-agent-orchestration-frameworks
45. https://www.microsoft.com/en-us/research/publication/autogen-studio-a-no-code-developer-tool-for-building-and-debugging-multi-agent-systems/
46. https://morsoftware.com/blog/web3-ai-agents
47. https://www.lindy.ai/blog/no-code-ai-agent-builder
48. https://hrt127.github.io/web3-assistant/
49. https://www.perplexity.ai/search/5015bc8d-7b81-4343-9ddb-86a4339a2947
50. https://www.domo.com/learn/article/best-ai-orchestration-platforms
51. https://www.kubiya.ai/blog/ai-orchestration-tools
52. https://akka.io/blog/ai-orchestration-tools
53. https://www.reddit.com/r/AI_Agents/comments/1hq9il6/best_ai_agent_frameworks_in_2025_a_comprehensive/
54. https://github.com/lorine93s/Web3-AI-Trading-Agent
55. https://hrt127.github.io/web3-assistant/
56. https://www.betmoar.fun
57. https://hubseer.com/site/betmoar
58. https://www.builder.io/blog/best-react-calendar-component-ai

59. https://docubear.com/use-fullcalendar-6-in-react-and-get-data-from-api-using-rtk-query/
60. https://dev.to/lberge17/fullcalendar-with-react-3hnl
61. https://morsoftware.com/blog/web3-ai-agents
62. https://polymarket.com
63. https://polymarketanalytics.com
64. https://www.syncfusion.com/react-components/react-scheduler
65. https://www.betmoar.fun/markets?view=calendar
66. https://www.betmoar.fun/
67. https://x.com/betmoardotfun
68. https://docubear.com/use-fullcalendar-6-in-react-and-get-data-from-api-using-rtk-query/
69. https://hrt127.github.io/web3-assistant/
70. https://www.betmoar.fun
71. https://polymarket.com
72. https://polymarketanalytics.com
73. https://www.builder.io/blog/best-react-calendar-component-ai
74. https://dev.to/lberge17/fullcalendar-with-react-3hnl
75. https://github.com/hrt127/web3-assistant/blob/main/index.html
76. https://www.builder.io/blog/best-react-calendar-component-ai
77. https://docubear.com/use-fullcalendar-6-in-react-and-get-data-from-api-using-rtk-query/
78. https://dev.to/lberge17/fullcalendar-with-react-3hnl
79. https://superagi.com/top-10-ai-agent-orchestration-frameworks-for-autonomous-workflows-a-comparative-analysis/
80. https://www.kubiya.ai/blog/ai-agent-orchestration-frameworks
81. https://www.microsoft.com/en-us/research/publication/autogen-studio-a-no-code-developer-tool-for-building-and-debugging-multi-agent-systems/
82. https://github.com/hrt127/web3-assistant/blob/main/index.html
83. https://www.youtube.com/watch?v=v2V6sYUrkAs
84. https://www.youtube.com/watch?v=O2DAy4AkFo4
85. https://www.youtube.com/watch?v=vEdtAWP_Umo
86. https://smashballoon.com/twitter-embed-timeline/
87. https://www.builder.io/blog/best-react-calendar-component-ai
88. https://docubear.com/use-fullcalendar-6-in-react-and-get-data-from-api-using-rtk-query/
89. https://www.bybit.com/en/help-center/article/Introduction-to-Bybit-TradeGPT
90. https://cointelegraph.com/news/bybit-launches-ai-trade-gpt-for-market-analysis
91. https://www.bybit.com/en/help-center/article/Introduction-to-Bybit-TradeGPT-Telegram-Bot
92. https://www.bybit.nl/en-NL/help-center/article/Introduction-to-Bybit-TradeGPT-Telegram-Bot
93. https://morsoftware.com/blog/web3-ai-agents
94. https://www.youtube.com/watch?v=4X0zI30Teng
95. https://www.youtube.com/watch?v=oYPvsFG6jyk