FACE

 Perplexity Prompt – “Designing FACE (Front-End Intelligence Layer)”
You are a senior B2B SaaS product + frontend architect with deep experience in:
marketing analytics dashboards
executive/CMO decision surfaces
React/TypeScript/Next.js front-ends
You are helping a team build FACE, the front-end “intelligence surface” for Omnify, an AI marketing intelligence layer.

 Context (Product + Market)
Use the attached documents as ground truth (do not contradict them):
28. Unified_Market_Intelligence_Validation_OmnifyBrain_V3
29. Unified 80/20 Pain Summary & Module Mapping (Validated)
30. Validated ICP + Subsector Targeting Summary
31. Final Top 5 Personas (Validated for MVP Build)
32–33. GTM Wedge Strategy — $50–100M Lower Mid-Market
34. Master Blueprint
35. The 24-Hour Hackathon Execution Blueprint
37. FACE UI Wireframe → UI
50. Unified Persona Pack — Builder Edition
51. Unified Customer Journey Map — Personas × Story × UX
52. FACE UI Functional Summary — Metrics, Predictions, Actions (Build Spec)
53. MVP Data Flow Summary — MEMORY → ORACLE → CURIOSITY → FACE
44. DataSeeds_Memory_Oracle_Curiosity_MVP_V1
From these documents, assume the following is true:
Target users:
CMO, VP Growth, Director of Performance
Mid-market DTC/subscription brands doing $50–$350M in revenue
First wedge: $50–$100M Beauty, Supplements, Wellness
Core problems:
Can’t see clearly what’s working across channels
Can’t see what’s about to break (fatigue, ROI decay, LTV drift)
Waste 20–40% of media budget from misallocation and late reaction
Too many dashboards, conflicting numbers, no clear action list
Omnify modules:
MEMORY → single source of truth (what happened)
ORACLE → predictions and risk (what might break next)
CURIOSITY → top 3 recommended actions (what to do now)
FACE → one simple UI that shows truth, risk, and actions for humans
MVP data:
FACE consumes three JSON outputs: memory_output, oracle_output, curiosity_output as described

in the docs.
For the hackathon, data is simulated via Data Seeds (CSV + JSON).

 Your task
Design the best practical way to build FACE (the front-end) for a 24-hour MVP, while also being a solid foundation for the actual product.
Break your answer into clear sections.

1. Front-end stack recommendation (short term + long term)
   For a 24-hour hackathon, what is the most realistic, high-leverage front-end stack to build FACE?
   Likely options: React + Vite, Next.js, plain React SPA, etc.
   For longer-term product, would you adjust to Next.js / Remix / tRPC / server components, etc.?
   Include:
   Suggested main stack (framework + language, e.g. React + TypeScript)
   Styling (e.g. Tailwind CSS vs CSS-in-JS)
   Charting library for simple time-series & bar charts
   Why this choice works best for:
   speed of build (24h)
   clarity of code
   ongoing maintainability
   Use citations for any performance or best-practice claims.
2. FACE screen architecture (components + layout)
   Using the attached FACE documents (37, 50, 51, 52, 53):
   Propose a component breakdown for the FACE screen, e.g.:

   (MER, ROAS, LTV-ROAS, risk, time window)
   (channel table: winners/losers)
   (fatigue, decay, LTV drift risks)
   (Top 3 actions, impact)
   (thumbnails + status)
   ("View as CMO / VP Growth / Performance")
   For each component, define:
   What props it receives (from memory_output, oracle_output, curiosity_output)
   What UI pattern it uses (table, tiles, list, inline chart)
   How it should express:
   severity (High/Med/Low)
   confidence (High/Med/Low)
   urgency (Now / This week)
   Show how to keep the full FACE UI as one page, with clear visual hierarchy:
   Truth (what happened)
   Risk (what might break)
   Actions (what to do)
   Impact (why this matters in money terms)
3. State management & data flow (MVP reality)
   Assume:
   Backend (Supabase) will expose or pre-generate a single demo JSON state with:

memory_output
oracle_output
curiosity_output
For the hackathon MVP, recommend:
How FACE should fetch and manage that state:
local JSON file vs simple REST endpoint vs Supabase client
minimal state management approach (e.g. React Query vs plain useEffect + useState)
How to structure TypeScript types/interfaces for:
MemoryOutput
OracleOutput
CuriosityOutput
How to handle:
loading state
error display (even if basic)
persona view toggle (CMO / VP Growth / Director Perf) as simple local state
Focus on simplicity and reliability over complexity.

4. UX & copy guidelines (Elon-level ruthless simplicity)
   Using the persona and journey docs:
   Propose microcopy rules for each persona:
   For CMO – plain business language and money impact
   For VP Growth – more explanation of "why things changed"
   For Performance Director – more tactical "pause/boost this" wording
   Suggest:
   How to phrase actions (e.g. "Shift 10% of budget from TikTok to Meta Retargeting")
   How to phrase risk (e.g. "TikTok is overspending with falling returns")
   How to phrase impact (e.g. "Save ≈ $18.5K this week by following these steps")
   Show how to make FACE usable in 5 seconds:
   The user should see:
   What's happening
   What's at risk
   What to do
   Why it matters (in money)

5. 24-hour implementation plan
   Given all of the above:
   Outline a realistic, hour-by-hour or step-by-step build plan for a small team over 24 hours, including:
   Setup (repo, framework, Tailwind, chart lib)
   Coding the layout and components
   Wiring to demo JSON / Supabase
   Testing the demo flow
   Polishing copy and states
   Suggest how to split responsibilities:
   1 "Structure + Data" dev (wiring JSON + logic)
   1 "UI + Components" dev (layout + styling)
   1 "Copy + Demo" owner (microcopy, narrative, final tweaks)
   Include risk points and what to cut first if short on time.

6. Musk-level critique of your own plan
   Finally:
   Critique your own proposed solution from the POV of:
   Elon Musk's UX team: is anything too complex? too slow to read? cluttered?
   Sales & Marketing: does FACE make the value obvious in under 30 seconds?
   Engineering: is this feasible in a 24-hour hackathon without overengineering?
   Then:
   Suggest one simplified version of FACE if we had to cut scope even further but still tell the full story (truth → risk → action → impact).
   Return:
   A structured, skimmable answer
   Tables where helpful
   Concrete recommendations, not just theory

FACE (Front-End Intelligence Layer) — Practical 24-Hour MVP Build, Scalable for Product

1. Front-end Stack Recommendation (Short Term + Long Term)

Timeframe	Main Stack	Styling	Charting	State/Data	Why
24h MVP	**React + Vite + TypeScript**	Tailwind CSS	Recharts (or Chart.js)	Local JSON / REST, `useState`	⚡ Fast setup, opinionated, instant HMR, lean
Product/Future	**Next.js (App Router) + TypeScript**	Tailwind CSS	Recharts or ECharts	React Query, maybe tRPC	SSR/SSG, API routes, scalability, maintainability

Why React + Vite + TypeScript & Tailwind (24h MVP)?

- **Speed**: Vite is fastest for greenfield React. No overkill (no routing needed for one-page FACE).
- **Clarity**: TypeScript enforces interface contracts for memory/oracle/curiosity outputs.
- **Maintainability**: Tailwind CSS is single-source styling, no context-switching.
- **Charts**: Recharts is React-first, declarative, supports time series/bar/inline sparklines (battle-tested).
- **Long Term**: Next.js for real product — server components, API routes, edge deployability, SSR/SSG, built-in auth/SEO .

2. FACE Screen Architecture (Components + Layout)

Visual Hierarchy:

- **Top**: Truth (summary)
- **Middle**: Risk (oracle predictions)
- **Lower**: Actions (what to do, why it matters)

Component	Props Received	UI Pattern	Expresses	Notes
FaceLayout	All outputs, persona, state	Page/layout	N/A	Main container, flex/column

Component	Props Received	UI Pattern	Expresses	Notes
TopSummaryStrip	KPIs (MER, ROAS, LTV), risk, persona	Tiles/strip	Severity, Confidence	Color: risk (red/orange/yellow/green bubbles/icons)
MemoryCard	memory_output	Table/list	High/Med/Low	Channels ranked, sparkline charts per channel
OracleCard	oracle_output	Inline cards	High/Med/Low	Decay/fatigue shown w/ urgency “Now/This week” badges
CuriosityActionsCard	curiosity_output	List	Urgency, Impact	3-item “do now” steps, money impact badge (`≈ $Xk`)
CreativeSnapshot	memory_output.creative_snapshot	Gallery/list	Status (On/At Risk)	Thumbnails, status chips
PersonaToggle	persona state	Toggle/tabs	—	3 persona buttons w/ icon, affects copy/context

How to express levels:

- **Severity/Confidence**: Color chips, icon (e.g. ▯, ▯, ▯, ▯), tooltip for details
- **Urgency**: “Now” = bold, “This week” = medium badge, muted if “Low”

Single Page Principle:
All main FACE blocks visible, scrollable, minimal nesting.
Top shows: “What’s working” (truth/KPIs), “What’s at risk” (risks), “What to do/impact” (recommended actions).

3. State Management & Data Flow (MVP Reality)

Backend/Source:

- For hackathon, fetch single JSON (simulated “Supabase” API or local file — easier for demo/handoff).

DATA FLOW SIMPLEST PATH:

- **Load**: useEffect to fetch and validate JSON
- **State**: useState for outputs; fail gracefully if null/undefined
- **Persona Toggle**: useState or context (`persona = "CMO" | "VP Growth" | "Director Perf"`)

What	How	Why
Fetch data	Local JSON, or REST endpoint	No need for direct DB integration; file fetch is enough
State mgmt	useState/useEffect	Don’t overengineer — React Query is overkill for static/dummy data

What	How	Why
Types	Define TS interfaces for each output, optional Zod/similar for parse	
Loading/Error	Basic spinner while loading, error boundary @ root	No silent failures—at least visible message
Persona View	Button toggle, sets local state, re-renders copy	Simplicity and rapid switch

Example TS Typings:

```
type Severity = "High" | "Medium" | "Low";
type Urgency = "Now" | "This week" | "Low";

interface MemoryOutput {
  channels: Array<{
    name: string;
    spend: number;
    roi: number;
    winner: boolean;
    loser: boolean;
    sparkline: number[];
  }>;
  creative_snapshot: Array<{ thumbnail: string; status: string; }>;
}

interface OracleOutput {
  risks: Array<{
    channel: string;
    risk_type: string;
    severity: Severity;
    confidence: Severity;
    urgency: Urgency;
    details: string;
  }>;
}

interface CuriosityOutput {
  actions: Array<{
    title: string;
    description: string;
    impact: string;
    urgency: Urgency;
  }>;
}
```

4. UX & Copy Guidelines (Personas × Simplicity)

- **CMO**
  - **Language:** Direct, dollars-first.
  - “Your ROAS is slipping on TikTok — risk $18.5K lost if unaddressed.”
  - Actions: “Shift 10% media budget to Meta Retargeting today.”

- Impact: “Saves ≈ $18.5K this week.”

- **VP Growth**
  - **Language:** Explain *why* numbers changed, what to investigate.
  - “TikTok returns are down 13% in 7d; fatigue & higher CPM identified.”
  - Actions: “Audit creatives, review CPM trend.”
- **Performance Director**
  - **Language:** Tactic/order-driven.
  - “Pause losing TikTok creatives now. Boost Meta Retargeting. Review UGC pipeline.”
  - Actions: “Pause ‘TikTok Spring Set B’ immediately,” “Increase Meta Retargeting by $10K.”
- **General Principles:**
  - **Risk:** “Meta Retargeting risk: Fatigue. Potential loss $8K,”
  - **Action:** “Pause TikTok losers now.”
  - **Impact:** “Prevents wasting $8K this week.”
  - **Usable in 5s:** Top shows “Truth” (summary KPIs), “Risk” (alerts/badges), “Action” (list), “Impact” (money → bold).

5. 24-Hour Implementation Plan

Hours	Task	Owner
0–1	Repo init (GitHub), install Vite (React + TS), Tailwind, Recharts/Chart.js	Structure
1–2	Base layout scaffold ()	Structure, UI
2–4	Define TS interfaces, load demo JSON data (local first)	Structure
4–7	Build , persona toggle	UI
7–10	(table, sparkline),	UI
10–12	, + badges/color	UI
12–13	Wire up persona-driven microcopy everywhere	Copy/demo
13–15	Loading/error states, handle missing JSON gracefully	Structure
15–17	Full demo/test, copy polish	All
17–18	Polish styles, empty/error/fallback cards	UI
18–22	QA: 5s onboarding, narrative, single “story” pass	Copy/demo
22–24	Buffer, screenshots, final fixes, showcase video	All

Split:

- *Structure + Data*: Setup, types, data fetch, persona toggle, error states.
- *UI + Components*: Layouts, visuals, Tailwind, charts, responsiveness.
- *Copy + Demo*: Microcopy for each persona, onboarding story, final UX test.

Cut First If Short:

- Skip CreativeSnapshot and detailed charting
- Focus on: TopSummary, MemoryCard w/ "winners/losers", OracleCard "risks", Curiosity "actions" — do not build personas toggle, keep just CMO mode and MVP copy.

6. Musk-level Critique & Simplified Scope

- **Elon's UX team?**
    - "Too much? Make everything bold and 'one-glance'. Remove clicks, hide secondary info by default. Use color, not numbers, for risk unless it's $."
    - Absolutely minimize word count above fold.
- **Sales/Marketing?**
    - "Is value clear in 10sec?" If not: dollar impact bigger, risk phrased as urgent, bold "You will lose $18K if you don't act today."
- **Engineering?**
    - "Can this actually be done in 24h?" Yes, as 1-page React app if scope creep is killed, data is local, and everyone pushes on final copy.

SIMPLEST VERSION if even further cut:

- Keep:
    - **1 summary tile strip:** MER/ROAS/LTV + "$ at risk" (big red badge if any)
    - **1 "What Broke" box:** plain list (e.g. "TikTok at risk: pause now")
    - **1 "Top Action" callout:** e.g. "Shift $10K to Meta Retargeting — Save $18K/7d"
- No persona toggle, just dollars-first CMO copy. No charts, just tiles + lists.
- **Truth** → **Risk** → **Action** → **Impact** all visible, no scrolling.

Concrete Recommendations:

- **Stack (MVP):** React + Vite + TypeScript + Tailwind + Recharts, fetch flat local JSON, basic useState. Move to Next.js and tRPC only for product.
- **FACE Components:** Compose layout w/ summary tiles, ranked tables, inline charts, urgent lists. Use color, badges, and dollars for urgency/severity.
- **State:** Keep minimal, fetch once; focus on reliable error+loading.
- **UX/Copy:** Ruthlessly direct, persona-tailored, action-oriented. Money impact always bold.
- **Plan:** Timebox wickedly, cut everything beyond "Is the value clear in 5s?"

Citations:

- Vite performance: , React with TypeScript best practice: , Next.js for scalable B2B products: