Cartpanda — Front-end Engineer Practical Test

This test has **two parts**:

1. Build a small **drag-and-drop upsell funnel builder** (visual editor only).
2. Answer a **dashboard architecture question** to show how you build scalable, modern front-ends and propagate patterns across a team.

What you must submit (required)

1. **Public demo URL** (no login required)
2. **GitHub repository** (with commit history)
3. **README** explaining setup + architecture decisions + accessibility notes
4. **Written dashboard answer** (in README or `/docs/dashboard-architecture.md`)

> Important: We are evaluating **front-end UX, code quality, and accessibility**.
>
> Not evaluating payments, auth, internal services, or real funnel page editing.

Part 1 — Build: Basic Upsell Funnel Builder (Public URL)

Goal

Create a **simple funnel builder**:

- Nodes (pages) can be **dragged onto a canvas**, **moved**, and **connected** with arrows.
- Funnel example: **Sales Page → Order Page → Upsell 1 → Upsell 2 → Thank You**, with optional **Downsells**.

This is **visual only**. No “real” pages required.

What you must deliver

- **Public URL** (e.g., Vercel / Netlify / Cloudflare Pages)
- **GitHub repo**
- A short README with:
  - how to run locally
  - main architecture decisions
  - tradeoffs / what you’d improve next

Core requirements (MVP)

A) Canvas

- Infinite-ish canvas feel (pan). Zoom is optional but nice.
- Grid background (optional but nice).
- Nodes are draggable.

B) Node types

Create node templates at least for:

- Sales Page
- Order Page
- Upsell
- Downsell
- Thank You

Each node shows:

- title (e.g. “Upsell 1”)
- small icon or placeholder thumbnail (can be simple)
- a primary button label (static)

C) Add nodes (drag from palette)

- Left sidebar “Palette” with the node types.
- Drag a type into the canvas to create a new node.

D) Connections (arrows)

- Connect nodes visually with arrows/edges.
- User can create a connection from a node to another (handle/connector).
- Show arrow direction.

E) Basic funnel rules

Implement these basic constraints:

- “Thank You” has **no outgoing** edges.
- “Sales Page” should have **one** outgoing edge (to Order Page) — allow editing but warn visually if invalid.
- Node labels like “Upsell 1”, “Upsell 2” should auto-increment when adding Upsells/Downsells.

F) Persist the funnel

- Save and load the funnel state in **localStorage** (good enough).
- Provide **Export JSON** and **Import JSON** buttons.

Nice-to-have (bonus)

- Zoom in/out
- Snap to grid
- Mini-map
- Undo/redo
- Edge deletion + node deletion
- Simple validation panel: “This funnel has 1 orphan node”, etc.

Suggested libraries (not required)

- React + TypeScript
- A graph library like React Flow (or your own)
- Tailwind / CSS modules / styled-components (your choice)

What we care about when reviewing

- Is the UX clean and obvious?
- Is the code modular and readable?
- Are state + interactions handled cleanly?
- Do you make good tradeoffs (not overbuilding)?
- Is it easy for someone else to extend?

Part 2 — Written Question: Modern Dashboard Architecture

Prompt

Imagine you are joining Cartpanda to build a modern admin dashboard for a funnels + checkout product:

- funnels, orders, customers, subscriptions
- analytics, disputes, settings, permissions

Write **1–2 pages** describing how you would build this dashboard in a way that:

1. stays fast and consistent as it grows
2. supports multiple engineers shipping in parallel without chaos
3. avoids “big rewrite” traps
4. meets **WCAG** standards for accessibility

Your answer must cover

1. Architecture

- How do you structure routes/pages, shared UI, feature modules?
- What patterns do you use to avoid spaghetti (feature folders, domain modules, etc.)?

2. Design system

- Do you use a component library (build vs buy)?
- How do you enforce consistency (tokens, theming, typography, spacing, accessibility)?

3. Data fetching + state

- Query caching strategy (server state vs client state)
- How do you handle loading/error/empty states?
- How do you manage filters/sorts/pagination on tables?

4. Performance

- Bundle splitting, virtualization, memoization, avoiding rerenders
- Instrumentation (how would you measure “dashboard feels slow”?)

5. DX (developer experience) & scaling to a team

- How do you onboard engineers into the patterns?
- What conventions do you enforce? (linting, formatting, PR templates, component guidelines)
- How do you prevent one-off UI and inconsistent behavior?

6. Testing strategy

- What gets unit tested vs integration tested vs E2E?
- What’s the minimum testing you’d require before moving fast?

7. Release & quality

- Feature flags, staged rollouts, error monitoring
- How do you handle “ship fast but safe”?

What a strong answer looks like

- Concrete choices (“I’d use TanStack Query for server state + Zod for runtime validation + Storybook for UI docs…”)
- Clear boundaries (“feature modules own their routes + queries + components”)
- Team scalability (conventions, docs, templates, PR review rules)
- Pragmatic tradeoffs (what they’d skip now vs later)

Scoring rubric (simple + effective)

Part 1 (Funnel builder) — 70%

- **Usability & correctness (25%)**: drag/drop feels right, connections work, basic rules enforced
- **Code quality (25%)**: structure, naming, components, state management
- **State & persistence (10%)**: export/import + localStorage done cleanly
- **Polish (10%)**: small UX touches, responsiveness, empty states, etc.

Part 2 (Dashboard thinking) — 30%

- **Clarity & depth (15%)**: can they communicate a plan clearly?
- **Scalability (10%)**: how they replicate the approach to the team
- **Pragmatism (5%)**: avoids overengineering, sensible tradeoffs

---

Notes

- Keep it simple. Don’t build internal features.

- We prefer a clean MVP with good accessibility over complex extra features.
- If something is intentionally skipped, explain the tradeoff in the README.