The Demand Instrument

Nine ideas on one template, one Vercel project per launch, and the three ways a validation test gets misread.

Michael Kaminski 2026-09-23 michael-kaminski.io/papers White paper 03

Summary

Nine business ideas were put on nine landing pages built from one template, each with its own pass/fail bar, so that copy quality was a constant and the only variable was the idea. Running ten launches cost roughly what running one does. This paper documents the instrument: the launch template that stands up a new idea in under fifteen minutes, the shared-services design that keeps ten ideas comparable in one view, the funnel plumbing most templates get wrong, and the three ways a validation test gets misread that the read model was built to refuse.

Sources of record: the public `launch-template` repository and its decision log; the private `demand-desk` repository that reads the results; and the nine landing pages of the Demand Test that ran 2026-08-17 to 2026-08-31. This paper covers the instrument. The read-out is in Demand Desk and is not restated here.

1. Why an instrument

Validating one idea is a project. Validating nine is a product, and the product is the instrument, not any of the nine. If each idea gets its own site, its own analytics account, its own database and its own payment setup, then two things go wrong. The cost of the test scales with the number of ideas, so the ninth idea does not get tested. And the results are not comparable, because each page was built differently and the difference in conversion could be the copy, the layout, the load time or the idea.

The instrument fixes both. One template renders every landing page, so copy quality and page performance are held constant. One set of shared services sits behind every launch, so the cost of the tenth idea is the cost of a new hosting project and nothing else.

2. One project per launch, everything else shared

Figure 1. The sharing decision by layer. Only hosting is per launch; the other four are shared and joined by a single `launch_slug`.

Each launch is its own hosting project, so it gets its own domain and its own deploy cadence. Everything underneath is shared and joined by one key:

Layer	Shared	Joined by
Hosting	No, one project per launch	—
Analytics	One project	`launch_slug` super property on every event
Database	One project	`launch_<slug>` Postgres schema per launch
Payments	One account	`metadata.launch_slug` on every session
Scheduling	One account	One event type per launch

The template runs with zero credentials. No environment file, no accounts, no setup: every page renders, and each unconfigured integration degrades to a no-op and logs exactly one warning. A health endpoint reports what is wired. Integrations are filled in one at a time, in any order, and nothing is a prerequisite for anything else. That is what "under fifteen minutes" means in practice: the time to a rendering site is the time to run the install, and the integrations are added while the copy is being written.

3. The funnel plumbing most templates get wrong

A validation test is only as good as its ability to say that the person who converted is the person who arrived. Three decisions in the template exist for that reason, and each one contradicts something the original brief asked for.

The browser's identity is threaded through checkout

The analytics library assigns each browser a distinct id. That id is passed into the payment session's metadata and the scheduling booking's metadata, so the server-side `purchase_completed` event, fired from the payment webhook, lands on the same person who started checkout. Without this, every purchase is a new anonymous person and the funnel breaks at the step that matters.

The thank-you page does not fire the purchase event

The brief described the thank-you page as firing `purchase_completed`, and also marked the event server-side from the webhook. Both cannot be right. The webhook is authoritative for three reasons: a browser event is spoofable, since anyone can open the thank-you URL; it is missed entirely whenever the customer closes the tab before the redirect, which is common; and firing in both places double-counts revenue. The thank-you page renders confirmation only.

Webhooks cannot double-charge

Both providers share one idempotency ledger. The claim is an atomic unique insert, keyed on provider and external id together (an event id is only unique within the provider that issued it), and it runs before any business logic. A failed event keeps its claim; re-opening it would let the provider's retry re-run half-finished work, so failures are recorded and found by query for manual replay.

Two smaller rules

Prices cannot be tampered with from the browser: the client sends a plan key and the server resolves the price id. And the database is locked down from migration one: row-level security on every table, deny by default, with public lead capture going through a rate-limited definer-rights function rather than a table grant. The function pins its search path, because a caller controlling the search path could otherwise shadow the leads table and hijack it.

4. The nine ideas and their bars

Each idea had a brand, a proof of demand and a bar defined before the test started. The bar is a conversion rate against a view count, and the proof is a costly action: a booked call, a one-dollar refundable deposit, or an email submitted with a real qualifier answer.

Idea	Brand	Proof of demand	Bar	Conversions needed
hire-a-founder	Founders Bench	Booked call	1.5% of 400 views	6
exec-search-flat	Tenflat	Booked call	1.5% of 400 views	6
provenance	Provenance	Booked call	1.5% of 400 views	6
pet-furniture	Hearth & Hound	$1 refundable deposit	1.0% of 600 views	6
curated-antiques	Estate	$1 refundable deposit	1.0% of 600 views	6
verified-tech-dating	Verified	Email + qualifier	6% of 800 views	48
made-in-america	Wholly American	Email + qualifier	4% of 800 views	32
look-alikes	Look-Alikes	Email + qualifier	4% of 800 views	32
off-peak	Off-Peak	Booked call	Supply test, no bar	—

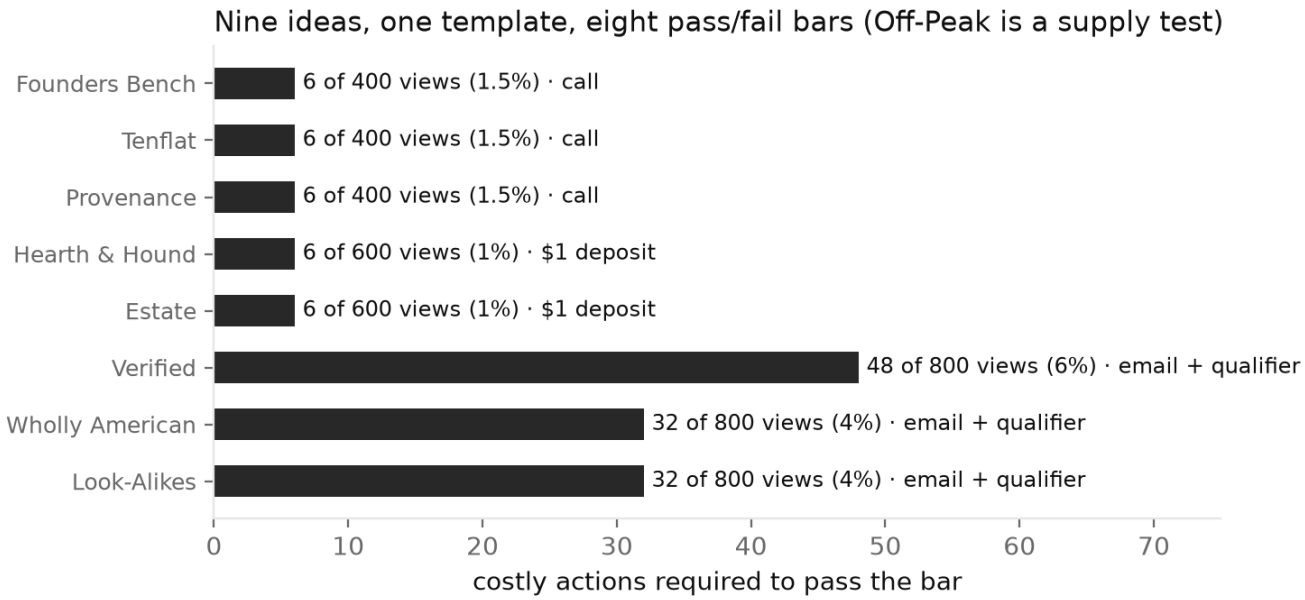

Figure 2. The bars, converted to the count of costly actions each idea needs. A booked call is a higher-cost proof than a qualified email, so it needs fewer of them.

The arithmetic in the last column is the point of the table. A booked call is expensive for the visitor, so six of them on 400 views is a real signal. A qualified email is cheap, so the bar is 32 to 48 on 800 views. **The proof of demand and the bar move together: the cheaper the action, the higher the rate it has to clear.**

Off-Peak has no consumer bar because it measures whether operators will take a call at all. It is a supply test and is never marked pass or fail.

5. Three ways the test gets misread

The read model, Demand Desk, is a fork of the template with the marketing removed and a gated dashboard added. It is opinionated about three distinctions, because each one is a way this kind of test is usually misread.

An email is not demand

On the call and deposit pages the email box is deliberately secondary. It records curiosity. Only the costly action counts toward the bar. Emails are shown but never headlined, and sub-1% rates render with two decimals so 0.4% and 0.04% do not both read as 0.0%.

A rate without a sample is noise

One conversion on eight views is 12.5% and means nothing. Every idea shows progress toward the view count its bar was defined against and reads "Not enough data" until it gets there. It does not get to look like a winner early, however flattering or damning the early rate.

Zero views is not 0%

An unmeasured idea shows a dash, not 0.0%. Those are different claims, and conflating them makes a page that has simply had no traffic look like a failure. This one had a concrete cause in the database: the project's existing scoreboard view was `FROM pageviews GROUP BY idea`, so an idea with no pageviews did not appear in the result at all. On day one that would have hidden eight of the nine. The nine come from code, and counts are joined onto them; a unit test asserts the nine slugs match the live pages exactly.

6. Access is enforced by the database

Registration to the dashboard is open, and being authenticated is not the same as being authorised. The obvious build, reading with a privileged key and checking an allowlist in the application, was rejected. A privileged key bypasses row security, which makes the application code the only thing between a bug and every row of live experiment data. And an application-level allowlist is theatre when registration is open: anyone could request a magic link, become authenticated, and query the REST API directly with their own token, never touching the application or its check.

So the allowlist lives in the database, in a table with row security on and no policy (deny-all, so knowing who has access is itself not published). A definer-rights function with a pinned search path answers whether the caller is on it, and the read policies on the three data tables call that function. **An unauthorised token gets zero rows from the database, not merely a hidden UI.** The deployment holds no privileged key at all; metrics are read with the signed-in user's own session. Verified empirically: allowlisted true, random signed-in user false, no token false, mixed-case email true.

Sign-in is magic-link only. There is no password field anywhere in the application, nothing to store, nothing to rotate.

7. Decisions the live tooling forced

The template's decision log records every place the original brief was contradicted by the platform, so the next launch does not re-derive them. Four are worth carrying to any project.

- **Proxy rewrites belong in the framework config, not the hosting config.** Hosting-level rewrites only apply in production. Locally the analytics proxy would serve nothing, every event would 404, and the template would fail its own works-on-a-fresh-clone requirement silently, because the analytics library swallows transport errors.
- **Placeholder secrets count as unset.** The launch initialiser writes every unfilled secret as `TODO_<NAME>`. Those are non-empty strings, so a fresh launch reported every integration as wired, the health endpoint agreed, and the payment client would have attempted a call with a literal placeholder key. The environment module treats the prefix as absent.
- **Generate the config file, do not patch it.** A regex rewrite matched the interface declaration before the object literal and swallowed the entire config export, emitting a file that did not compile, silently, because the script reported success. The file is now rendered from a template.
- **Latest is not always compatible.** The TypeScript and linter versions are held one major behind the registry's latest because the lint stack's peer ranges top out below them. The reasons are written next to the pins, with a dependency-bot ignore rule that names the condition for removing it.

8. What this paper does not report

The results. The Demand Test ran from 2026-08-17 to 2026-08-31 with the rule that one idea gets built and the data behind the pick is published with it. That read-out belongs to Demand Desk and to the launch of whichever idea cleared its bar, not to a paper about the instrument. Conflating the two would commit exactly the misreading section 5 warns about: quoting a rate before the sample it was defined against.

9. What to copy

- **Hold the page constant so the idea is the only variable.** One template, one set of shared services, one key joining them.
- **Define the bar before the traffic.** A proof of demand and a rate against a view count, with the cheaper proof carrying the higher rate.
- **Thread the browser identity through checkout, and let the webhook be the truth.**
- **Refuse the three misreads in the read model, not in the analyst.** Emails are not demand, a rate needs its sample, and a dash is not a zero.
- **Put authorisation in the database.** An application check with open registration protects nothing.
