BUILD DIARY / 01

If I Won.

How a late-night idea became a live, AI-powered EuroMillions simulator in ten days.

BY

Arjun Chana

BUILT

26 May – 5 June 2026

LIVE AT

ifiwon.app

01 / THE IDEA

The Idea

It started at five to eleven on a Tuesday night with a question I half expected to talk myself out of: would an app where you simulate spending a EuroMillions jackpot actually do well, or was it just a fun thought?

The first thought I had wasn't encouragement. The honest assessment was that the core concept was *a novelty with a short shelf life.* Lottery spending simulators already existed in various forms. Without a sharp differentiator it would be something people use once, share once, and forget. Retention would be near zero. The presence-building angle would only work if there was a content engine wrapped around it.

That pushback turned out to be the most useful thing that happened in the whole project, because it forced the idea to earn its existence before a single line of anything was written. My response was to flesh out what I actually meant. Not *buy a yacht for x millions.* A jackpot tied to the real EuroMillions draw, updating twice a week. AI personalisation that asks about your actual life and writes something back that is genuinely yours. An investment mini game where your money moves, where you make decisions and feel the consequences. A house, but which house, with which rooms. A business, and why. Giving money to named friends and family, and what that moment feels like.

Somewhere in that exchange the idea stopped being a novelty and became a product. The recurring hook was the real draw data. The depth was the branching. The emotional pull was the personalisation. The shareability was a results card people would actually want to post.

Then came the question that decided everything about how this would get made: is this possible through vibe coding? I am not a developer. My background is UX writing, copywriting, and content. The answer was yes, with the right tools, and that set the constraint that shaped the entire build: I would own every creative decision and every word, and AI would handle the construction.

The name

The working title was Jackpot Life, which never felt right. We ran through options: Windfall, Spend the Lot, Rollover, Flush. The one that stuck was **If I Won**. Three words, and it is the exact phrase everyone already says when the jackpot rolls over. It needs no explanation because people have been saying it their whole lives.

The creative direction

Two principals were locked in before the build started and never changed.

First, the prices had to be real. A private chef costs what a private chef costs. A scholarship endowment is ~£500,000 because that is roughly what one takes to fund. The accuracy is what makes the absurdity land: you buy the villa, the jet, the castle, you clear every debt you have ever had, and the balance barely moves. That is the actual experience of a nine-figure jackpot, and the app only communicates it if the numbers are honest.

Second, the voice had to be mine. Every line of copy in the app went through me. AI drafted structure and starting points; I rewrote until it sounded like a person. No em dashes. No synthetic enthusiasm. No constructions that smell like a language model. This rule later got encoded directly into the product itself, in a way I will come back to.

> **Somewhere in that exchange the idea stopped being a novelty and became a product.**

02 / THE DESIGN PROCESS

The Design Process

The single best process decision of the project: nothing went into the build tool until it had been designed, written, and signed off first.

Prototype before product

Before touching Lovable, a full clickable HTML prototype was built in the chat. Landing page, onboarding, hub, category screens. This cost almost nothing and meant that by the time real building started, the questions being answered were *does this work* rather than *what is this.* The same pattern repeated at every major design moment. The summary screen was prototyped as a standalone HTML file before a prompt was written. The share card went through two full prototype versions. The colour overhaul was explored as four working mock-ups of the actual hub layout.

Prototyping in HTML first meant I could react to something real instead of imagining from a description, and it meant every Lovable prompt described something already decided rather than something being figured out live in an expensive build loop. This also helped keep the cost of Lovable tokens down and be more effective with prompting.

The colour journey

The app launched its build phase in a default dark navy with gold. Functional, but partway through I asked the question that had been nagging me: is this design too AI basic? The base palette you see on every AI-generated landing page. The same gradient, the same font feel, the same midnight blue.

The answer was to finish the functionality first and do one dedicated design pass at the end, which is exactly what happened. When the time came, three completely different directions were mocked up (*near-black editorial, deep forest green, warm amber brown*), then four targeted refinements of the actual app layout. The winner was **Deep Plum**: the existing gold kept exactly as it was, but the base shifted from cold navy to a plum-black (*#0a0810*). It was the most subtle option on the table and the most effective. It removed the generic AI dark-mode feel in one move without touching anything that already worked.

One detail from that pass taught me something about glow. An attempt to reduce the background radial glow made everything look flat and dead, and I reverted it within minutes. The glow was not decoration. It was doing structural work, giving the dark canvas depth and making the jackpot number feel like it was radiating significance. Some things look excessive in isolation and correct in context.

Typography

Three typefaces, three jobs. **Fraunces**, a high-contrast serif, for headlines and the big money numbers; it makes a jackpot figure read like a newspaper front page rather than a calculator output. **Hanken Grotesk** for body copy. **DM Mono** for prices, stats, and anything that needs to feel like data. One font experiment late in the build got instantly rejected (my actual message was "*i hate the font! please make it go back*"), which is its own lesson: taste is a veto, and the undo button is part of the design process.

03 / THE BUILD

The Build

The stack

Lovable as the builder. The entire app was constructed through written prompts, with no code written by hand. **Supabase** behind it for data. The **Claude API** (*claude-sonnet-4-6, I'm not made of money here!*) for the personalised narratives, which became the soul of the product. **Vercel** for hosting, handled invisibly through Lovable's publish flow. And a free community **EuroMillions API** for the live jackpot data, which meant the app's headline number updates itself after every Tuesday and Friday draw with zero maintenance.

The working pattern

A repeatable loop emerged within the first two days and carried the whole project:

1. Decide and design the screen in conversation first, including all copy.
2. Write one detailed, structured prompt for Lovable describing exactly what to build.
3. Build, screenshot, review.
4. Send small, targeted fix prompts rather than broad rebuild requests.
5. Bookmark in Lovable's version history after every confirmed working feature.

The bookmarking habit started after the first scary moment and became non-negotiable. Lovable's history restore saved the project at least twice, once from a spending bar redesign I disliked and once from the font change. Cheap insurance, taken constantly.

The other pattern worth naming: when a feature was complex, the prompt was split into parts. The investment game went in as three separate prompts (*setup screens, Set and Forget mode, Active Investor mode*) because a single prompt covering all of it would have overwhelmed the tool and produced mush. Sequencing the build is a skill in itself.

> **Lovable executed decisions; it did not make them.**

04 / THE NARRATIVE LAYER

The Narrative Layer

This is the feature that turns the app from a spending tracker into something people describe to other people.

Without it, every category ends in a preset line, the same for everyone. With it, completing a category triggers a live call to the Claude API carrying everything the app knows: your age bracket, living situation, dream location, risk appetite, country, and the exact choices you just made. Claude writes back three to five sentences that exist for you alone. Finish the run and a longer summary narrative ties the whole life together.

Making this work took three pieces of real engineering, all built through prompts.

A label resolver. The app stores choices as code keys: *clifftop, chef, scholarship.* Those mean nothing to a language model. A resolver file maps every ID in the entire app to readable language (*Clifftop Villa, Live-in Private Chef*) before anything is sent to the API. Writing that mapping meant auditing every selectable item in the product, which doubled as the most thorough QA pass of the build.

A system prompt that enforces my voice. This is where the copy rules became code. The system prompt that travels with every narrative request bans em dashes outright. It bans *journey, transform, elevate, unlock, next chapter,* exclamation marks, and synthetic enthusiasm. It includes a worked example of good output against bad output, where the bad example is explicitly labelled a brochure. And it carries the line that did the most work of any single instruction in the project: **the narrative is not a summary of what they did, it is a portrait of who they are based on what they chose.**

An anti-hallucination clamp. The first playtests exposed the failure mode immediately. The narratives were beautiful and partly fictional. One told me about the ~£200,000 monthly salary I had set up; I had set up no such thing. Another praised my donation to mental health causes I had not yet visited. The fix was a hard instruction prepended to every category prompt: only reference choices explicitly present in the data, never invent, never extrapolate, treat empty fields as non-existent. Paired with null checks in the prompt templates, the inventions stopped.

I was honestly sceptical of this layer until I saw it run on real choices. The business narrative was the one that won me over. Reading something that referenced the company name I had typed, the location I had chosen, and the fact that I had told the app I was waiting for a reason to quit my job, all woven together rather than listed back, was the moment the product justified itself.

05 / THE PROBLEMS SOLVED

The Problems Solved

Every build has its bug log. These were the ones that mattered, and what each taught.

The silent jackpot fallback

For days the landing page showed a fixed figure and I assumed the live API was working. It was not. The fetch call existed, but the API returns an array of historical draws with a prize's breakdown rather than a top-level jackpot field, so the extraction logic never matched and the code silently fell back to its hardcoded default. Nothing errored. Nothing looked broken. The fix read the latest draw and pulled the top-tier 5+2 prize with sensible fallbacks. The lesson sits with me as a fallback that works too smoothly can hide a failure for as long as you let it. If a number never changes, get suspicious.

The multi-select plague

The same bug family appeared in Property, Lifestyle, and Business in different costumes: options that should allow several selections capping at one, model pickers collapsing when a sibling was chosen. Business was the worst because its randomised outcome cards

refused to trigger, and the diagnosis took several rounds: multi-select screens never produced a single committed final state for the outcome to hang off. The structural fix converted the path steps to single-select with an explicit lock-in, which is a case where the bug fix was actually a design correction.

The spend of zero

Friends and family playtesting (*always the most honest QA*) found that a profitable investment run made the summary screen report total spend as ~£0. The net investment gain was being subtracted from spending. The repair separated the concepts permanently: total spent is the sum of all allocations excluding investments, and the investment result lives on its own line with its own colour, mint for profit, coral for loss.

There was also a one-character bug worth recording for humility: the share card shipped pointing to the wrong domain. One find-and-replace before anyone noticed.

06 / WHAT I LEARNED

What I Learned

Vibe coding is real, and it is not magic. I built and shipped a full product with branching logic, live external data, an AI integration, and a canvas-rendered share system without writing code. But every successful prompt was successful because the thinking had already been done. Lovable executed decisions; it did not make them. The clearer and more structured the prompt, the better the output, and the moments it went sideways were almost always moments where I had asked for something I had not fully specified.

The skill is decomposition. Big features go in as parts. Bugs get isolated with single targeted prompts, not pleas to fix everything. Version bookmarks happen after every confirmed win. None of this is coding, and all of it is engineering discipline. I came out of this with a genuine new competency that has nothing to do with syntax.

Creative ownership is the differentiator. Anyone can prompt an app into existence now. What they cannot prompt into existence is taste. Every piece of copy in If I Won sounds like one person because one person rewrote all of it, batch by batch, line by line. The voice rules were so central they got compiled into the product itself as a system prompt. As the technical barriers keep falling, the originality and the judgment are the product. This is the thesis I keep coming back to, and this build is its proof.

The last message I sent on launch day was simple: I feel so proud of myself, I did something. Ten days earlier the whole thing was a question I expected to get talked out of. *Not bad for a Tuesday night thought.*

```
Built with Lovable, Supabase, the Claude API, and a free EuroMillions data feed.
Every word of product copy written and edited by hand.
Total build time: ten days, no prior coding experience, one bookmark at a time.
Live at ifiwon.app — Arjun Chana, 2026.
```