# Swaply

**Swap your skills. Learn for free.**

Full-stack · Mobile-first PWA · Peer-to-peer skill exchange

• **LIVE** swaply.vercel.app

React 18 | Firebase | Groq / LLaMA 3 | Vercel | PWA

| 8+ | 3,800 | 12 | 5 | 4 | 470 KB |
|---|---|---|---|---|---|
| ROUTES | LINES OF CODE | DEMO USERS | FIRESTORE COLLECTIONS | CACHE LAYERS | SOURCE SIZE |

Solo end-to-end build · React 18 + Vite 5 · Firebase · Groq / LLaMA 3 · PWA · MIT License · 2025

## What this document is — and how to read it

This document is the full design, engineering, and product record of **Swaply** — a skill-exchange platform built solo, end-to-end, from user research to live production deployment.

It has 23 sections and approximately 45 minutes of reading at full depth. You do not need to read it linearly. The three paths below route you to the sections that are most relevant to your role.

### PATH A

**Recruiter / PM Lens**

Product thinking, research rigour, and decision-making under constraint



≈12 min read

### PATH B

**Technical Reviewer Lens**

Architecture, security, engineering trade-offs, and code depth



≈15 min read

### PATH C

**Founder / Investor Lens**

Market positioning, business model, and strategic trade-offs



≈10 min read

### Who built this — and why it matters

Swaply was designed, engineered, and shipped *entirely solo*. No team. No external PM. No engineering support. Every decision in this document — from the Firestore security rules to the freemium swipe cap to the session proposal card — was made by the same person who wrote the code that implemented it.

That constraint is deliberate. The goal was not to build the fastest MVP, but to practice the full stack of decisions a product-minded engineer or early-stage PM makes under real resource pressure: scoping, trade-off reasoning, user research without a research team, and shipping something live and usable without waiting for consensus.

**If you are a recruiter or hiring manager:** this document is the evidence base. The thinking behind each product decision is documented in the same file as the code that implements it — because, in this build, they were the same decision.

## 01 PROJECT OVERVIEW & MOTIVATION

### Why I Built This

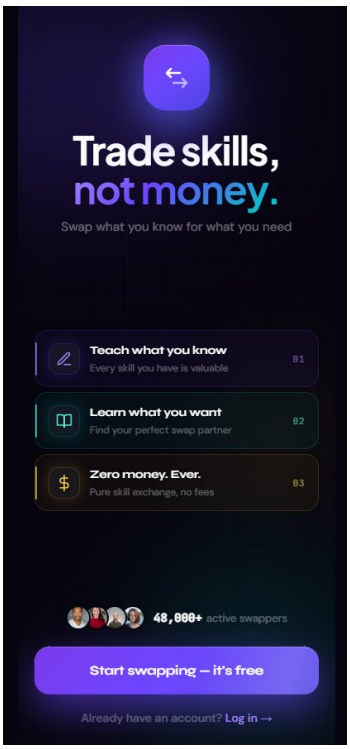
I wanted to learn UI design. I could teach React. The obvious move was to find a designer who wanted the opposite — but there was no product that made that exchange frictionless. I posted on Reddit. I got one reply three weeks later. By then I had moved on.

Online learning is asymmetric: platforms charge learners, teachers earn little, and informal skill trading — language tandems, barter communities, swap networks — has always existed without infrastructure to support it at scale. Swaply applies the swipe-based interaction model to mutual learning: a developer who wants to learn guitar is surfaced to the musician who wants to learn React. No money changes hands; value flows in both directions.

The project was also a deliberate constraint: build something with real product complexity — two-sided matching, trust infrastructure, a monetisation model that doesn't break the core value — as a solo developer under the same resource and prioritisation pressure a 0-to-1 PM faces. Every decision in this document reflects that constraint.

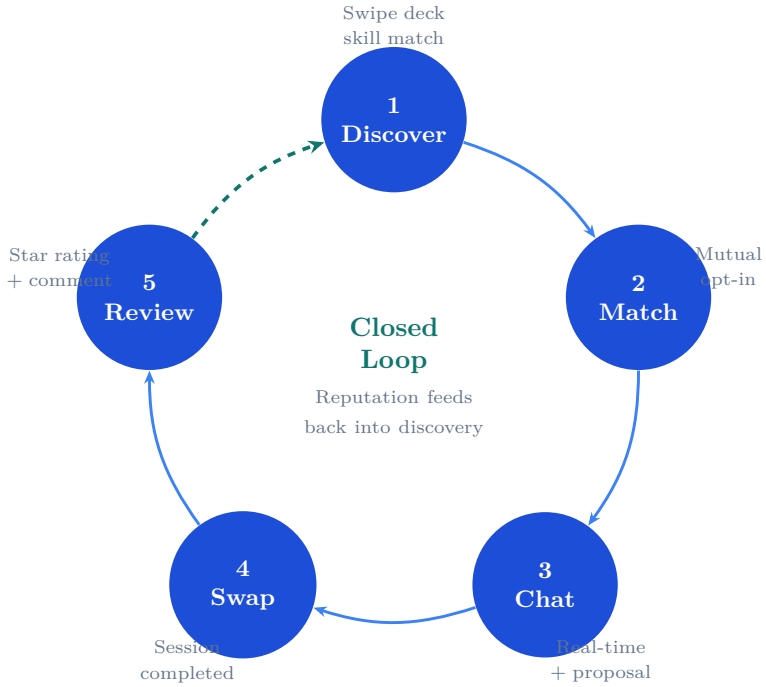
> **Core insight:** Removing money eliminates payment friction, regulatory overhead, and the power imbalance between teacher and student.

THE PRODUCT — WELCOME SCREEN




Swaply's landing screen: three-step value proposition (Teach, Learn, Zero money) with social proof and single CTA.

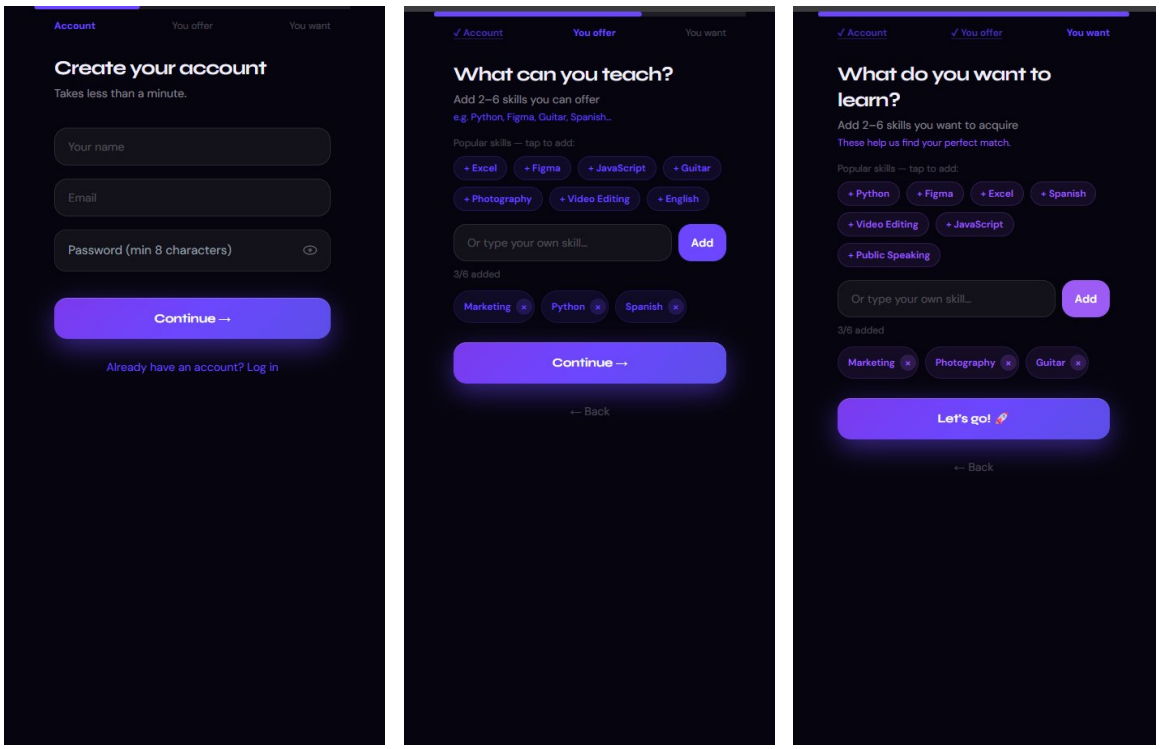
### The Five-Step Engagement Loop




**Review → Discover** feedback loop: higher-rated users surface earlier in the swipe deck. Each completed swap improves future match quality for the entire platform.

– Each completed swap generates reputation data that improves future matches — a closed engagement loop.

ONBOARDING FLOW — 3 STEPS TO FIRST MATCH




Step 1: Account creation · Step 2: Skills you teach (free-text tags) · Step 3: Skills you want to learn — completes the match profile

## 02 USER RESEARCH & PROBLEM VALIDATION

> **Method:** Before writing any code I had 6 informal conversations — 3 in person, 3 over Discord — with people who had *already* traded skills informally (language tandems, subreddit exchanges, word-of-mouth barter). No screener, no incentive. I asked what they actually did, not what they wished existed.

### Who I Spoke To

| # | Background | Informal exchange they'd done | How they found the other person |
|---|---|---|---|
| 1 | CS student, Athens | Taught Python, learned music theory | r/learnprogramming post, 3-week wait for a reply |
| 2 | Graphic designer, freelance | Taught Figma, learned copywriting | Instagram DM to someone she followed |
| 3 | Language teacher (French) | Language tandem with a Spanish speaker | HelloTalk, but tandem fell apart after 2 sessions |
| 4 | Self-taught developer | Wanted to learn illustration, offered React help | Posted on Twitter, got one reply, nothing happened |
| 5 | Marketing student | Taught social media, learned Excel | Friend of a friend introduction |
| 6 | Musician, part-time tutor | Taught guitar, learned video editing | Facebook group, met in person |

### What They Said (Direct Quotes)

**PARTICIPANT 1 — CS student**

*"I posted on Reddit and someone replied three weeks later. By then I'd kind of moved on. I don't know if I would have even remembered what I wanted to learn."*

→ Discovery latency kills intent. The gap between wanting and finding is too wide.

**PARTICIPANT 2 — Graphic designer**

*"I DMed her and it felt really awkward. Like, hi, I stalked your work and now I want something from you. I didn't know how to frame it without sounding desperate or transactional."*

→ Cold outreach to strangers has a high social cost. The format needs to normalise the ask.

**PARTICIPANT 3 — Language teacher**

*"We did two sessions and then she just stopped replying. I think we never actually agreed on a schedule. It was always 'let's do it sometime' and that means never."*

→ Informal agreements dissolve. A structured session proposal with Accept/Decline is not a nice-to-have — it is the retention mechanism.

**PARTICIPANT 4 — Self-taught developer**

*"I posted, got one like and zero DMs. I figured either nobody wanted to trade with me or nobody saw it. I have no idea which. I just felt like I was shouting into a void."*

→ No feedback loop = no iteration. A swipe deck gives implicit signal even from non-matches.

**PARTICIPANT 5 — Marketing student**

*"It worked out, but only because a mutual friend vouched for both of us. If I'd found her online I probably wouldn't have trusted her enough to show up to a coffee."*

→ Trust is the conversion blocker, not discovery. Reputation and ratings are infrastructure, not polish.

**PARTICIPANT 6 — Musician**

*"The Facebook group thing worked but it was like thirty back-and-forth messages to sort out what we'd even do. By the end I was exhausted and we hadn't started yet."*

→ Coordination overhead is the activation energy cost. The session proposal format eliminates 80% of those messages.

### What the Research Changed

Three assumptions I held going in were either confirmed or revised:

| Assumption | Verdict | Impact on product |
|---|---|---|
| "The problem is finding the right person" | **Partially true** | Discovery matters, but trust and commitment are equal blockers — hence ratings and session proposals in v1. |
| "Money is the barrier to informal exchange" | **Confirmed** | All 6 had exchanged or tried to exchange without payment. The barter impulse exists and is not niche. |
| "Users will articulate what they want upfront" | **Wrong** | P4 and P1 struggled to even frame their own ask publicly. Swipe-based discovery removes the blank-page problem. |
| "A match is enough to start a swap" | **Wrong** | P3 and P6 matched and still fell through. Scheduling structure is not a feature — it is the swap completion mechanism. |

**Honest caveat:** Six conversations is not a statistically significant sample and all participants were in my network (students, creatives, developers). I have not spoken to, for example, a 45-year-old tradesperson who wants to exchange plumbing knowledge for tax advice. The core problems — discovery latency, cold-outreach anxiety, scheduling fallthrough, and trust — are likely to be consistent across demographics, but the specific UX patterns

may need revisiting for non-digital-native users.

## 03 USER PERSONAS

**Distilled from research:** The six participants mapped onto three recurring archetypes. These are not demographic profiles — they are behavioural patterns defined by the specific failure mode each person hit when trying to exchange skills informally. Each archetype drives a distinct set of product requirements.

### THE SKILL TRADER

Research basis: P1, P4

Has a clear skill to offer and a clear skill to acquire. The exchange is rational — they would happily trade React for guitar, Python for design — but has no infrastructure to find the right counterpart efficiently. Has tried posting publicly, received no signal, and abandoned the intent before a match was made.

| | |
|---|---|
| PRIMARY BLOCKER | Discovery latency. The gap between forming the intent and finding a compatible person is wide enough that motivation decays. |
| WHAT THEY NEED | A fast, low-friction way to surface compatible partners without writing a cold pitch. |
| PRODUCT RESPONSE | Swipe deck with inline "You teach / They teach" match explanation. Skill-based matching surfaces the right person without the user having to articulate a search query. |
| UPGRADE TRIGGER | Hits the 10-swipe daily cap while actively looking — highest-intent upgrade cohort. |

### THE BURNED EXCHANGER

Research basis: P3, P6

Has successfully matched with someone before — informally, through a community or mutual contact — but the exchange collapsed before or after the first session. Not because the match was wrong, but because "let's do it sometime" is not a commitment mechanism. Wary of investing time in another match that evaporates.

| | |
|---|---|
| PRIMARY BLOCKER | Scheduling fallthrough. Informal agreements dissolve without a shared artefact that makes the commitment visible to both parties. |
| WHAT THEY NEED | A structured session proposal that both parties explicitly accept — not a vague calendar negotiation. |
| PRODUCT RESPONSE | Session proposal card in chat: date, time, duration, format, Accept/Decline. The proposal is a persistent artefact in the thread — not a verbal agreement that can be misremembered. |
| RETENTION SIGNAL | Review completion rate. If this archetype completes a structured swap and leaves a review, they have closed the loop that previously broke — D7 retention is high. |

### THE TRUST-GATED LURKER

Research basis: P2, P5

Open to skill exchange but the cold-outreach step carries a disproportionate social cost. Reaching out to a stranger to say "I want something from you" feels transactional and awkward without a mutual voucher. Will only commit to an exchange if the trust signal is already visible before contact — not after.

| | |
|---|---|
| PRIMARY BLOCKER | Cold-outreach anxiety. The format of the ask normalises the exchange or kills it before it starts. |
| WHAT THEY NEED | Trust signals before commitment: ratings, review count, completed swap history, all visible pre-match. |
| PRODUCT RESPONSE | Star rating on swipe card. AI icebreakers that model an appropriate tone for first contact. The mutual-opt-in match mechanic means both parties have already signalled interest before anyone writes a message — removing the cold-outreach dynamic entirely. |
| ACTIVATION RISK | Highest drop-off risk at match → first message. If match-to-chat rate is low, this archetype is the primary cause — the product has not yet given them enough trust signal. |

**How the archetypes interact:** A healthy platform needs all three. The Skill Trader generates match volume. The Burned Exchanger drives session completion and review data. The Trust-Gated Lurker benefits from the reputation layer the Burned Exchanger produces — their activation depends on a critical mass of reviews that only exists if the second archetype completes swaps. This interdependency is why reputation is infrastructure, not a v2 feature.

## 04 JOBS TO BE DONE

**Framework:** Jobs-to-be-Done separates *what the user does* (feature use) from *why they hired the product* (the job). Each job has three layers: a **functional** component (the practical task), an **emotional** component (how they want to feel), and a **social** component (how they want to be perceived). The UI decisions in Swaply map directly to these layers.

### JOB 1 · The Stuck Learner

*"When I want to learn a skill but can't justify paying for a course, I want to find someone who needs what I already know, so I can exchange value without feeling like I'm asking for a favour."*

| | |
|---|---|
| FUNCTIONAL | Find a compatible learning partner quickly, without writing a cold-outreach message from scratch. |
| EMOTIONAL | Feel like the exchange is fair and legitimate — not a favour owed or a transaction that creates power imbalance. |
| SOCIAL | Be seen as someone who has something to offer, not just someone who needs help. |

**UI EVIDENCE** · The swipe card shows "You teach: python / They teach: guitar" before any message is sent — the exchange framing is visible at the discovery stage, not after commitment. The onboarding flow collects "What can you teach?" before "What do you want to learn?" — positioning the user as a contributor first. AI icebreakers on first open pre-load context-aware openers so the user never faces a blank canvas.

### JOB 2 · The Informal Trader Who Got Burned

*"When I've tried to organise a skill exchange informally before and it fell apart, I want a structured format that creates mutual commitment, so I don't waste time on someone who disappears after the first message."*

| | |
|---|---|
| FUNCTIONAL | Schedule a session with a specific date, duration, and format — without 30 messages of back-and-forth. |
| EMOTIONAL | Feel that the other person is as committed as I am — that this is real, not "sometime soon." |
| SOCIAL | Have a record of completed swaps and reviews that signals reliability to future partners. |

**UI EVIDENCE** · The session proposal is a structured card inside the chat thread — date, time, duration, format — rendered with an explicit Accept/Decline action. The "Propose" button is pinned at the top of every chat screen, one

tap from any conversation. The review modal fires after a session is marked complete, closing the accountability loop. Completed swap count is visible on the profile card — social proof of follow-through.

**JOB 3 · The Trust-Gated Stranger**

*"When I consider learning from someone I've never met, I want enough signal about who they are and how others experienced them, so I can decide whether to invest time without needing a mutual friend to vouch for them."*

| | |
|---|---|
| **FUNCTIONAL** | Read verified reviews and skill history before committing to a session. |
| **EMOTIONAL** | Feel safe engaging with a stranger — that the platform has a mechanism for accountability, not just discovery. |
| **SOCIAL** | Build a reputation over time that makes future matches easier to land. |

**UI EVIDENCE** · Star rating and review count displayed on the swipe card before any interaction (e.g. "★ 4.7 (3)"). The chat header shows the match's rating at a glance — trust signal is never more than one screen away. Atomic review writes ensure ratings are always consistent; a user cannot game the system via a partial write exploit. Reputation feeds back into discovery: higher-rated users surface earlier in the swipe deck for Pro members.

**What JTBD reveals that feature lists hide:** The three jobs share a single root cause — informal skill exchange has always existed, but the existing infrastructure (Reddit posts, DMs, Facebook groups) fails at commitment, trust, and structure simultaneously. Swaply's value is not any individual feature. It is the only product that resolves all three failure modes in a single closed loop: discovery → structured commitment → verifiable reputation.

## 05 PROBLEM · SOLUTION FIT

**Reading this table:** Each row is a complete narrative unit — a documented user pain, the specific feature built to address it, and the measurable outcome that would confirm the solution worked. This is the connective tissue between the user research in Section 02 and the metrics in Section 07.

| User Pain | Feature | Mechanism | Outcome Indicator |
|---|---|---|---|
| **Discovery latency.** P1 posted on Reddit and got a reply three weeks later. By then the intent had dissolved. | **Skill-based swipe deck** with inline match score and "You teach / They teach" explanation. | Reduces time-to-first-candidate from days to seconds. Removes the blank-page problem entirely. | Match-to-chat rate >40%. Low time-on-deck before first swipe action. |
| **Cold-outreach anxiety.** P2 said reaching out felt "transactional and desperate." No format to normalise the ask. | **Mutual opt-in matching** + AI icebreakers pre-loaded with skill context on first open. | Both parties have signalled interest before any message is written. AI suggestions model appropriate tone. | AI icebreaker click-through rate. Match-to-chat conversion rate. |
| **Scheduling fallthrough.** P3 matched, did two sessions, then the exchange collapsed because there was no shared commitment artefact. | **Session proposal card** in chat: date, time, duration, format, explicit Accept / Decline. | Turns a vague "sometime" into a structured, visible commitment. Both parties see the same artefact. | Chat-to-session rate >30%. Session proposal acceptance rate. |
| **Trust barrier.** P5 only exchanged because a mutual friend vouched for both parties. No voucher, no exchange. | **Star ratings + review count** visible on the swipe card before any interaction. | Reputation replaces the mutual friend. Trust signal is available at the discovery stage, not after commitment. | Review completion rate >60%. Match-to-chat rate for users with >3 reviews vs. zero. |
| **Coordination overhead.** P6 exchanged 30 messages to organise what they would do. Exhausted before the first session. | **Session proposal** keeps all scheduling inside the chat thread. No external tool required. | Single structured message replaces 30 unstructured ones. Activation energy cost is reduced to one tap. | Average messages-before-session-proposal. Post-swap session rating. |
| **No feedback loop.** P4 posted publicly, got no signal, and could not tell if the problem was visibility or fit. | **Swipe deck** provides implicit signal even from non-matches. Pass/Interested actions are data. | Even a rejected swipe tells the algorithm something. Users get feedback without having to ask for it. | Swipe-through rate per session. Algorithm improvement proxy via match quality score over time. |

**What this table is not:** This is not a feature justification exercise. Each row was written after the user research — the pain came first, the feature second. Two features that appear obvious in hindsight (session proposals, mutual opt-in matching) were only prioritised for v1 because the research showed they were retention mechanisms, not polish.

## 06 METRICS & SUCCESS CRITERIA

**North Star Metric:** *Completed Swaps per Week (CSW)* — a swap counts only when both users confirm the session happened. Every other metric is a proxy for this.

| Metric | Why it matters | Hypothesis / Target |
|---|---|---|
| Completed swaps / week | Core value delivery — the reason the product exists | ≥1 per active user |
| Match-to-chat rate | Quality of the matching algorithm | >40% |
| Chat-to-session rate | Friction in booking flow | >30% |
| DAU / MAU ratio | Engagement & habit formation | >15% |
| D7 retention | Product stickiness after first swap | >25% |
| Review completion rate | Loop closure; data quality for matching | >60% |
| AI icebreaker click-through | Validates the AI feature's actual value | Leading indicator |

### Why these numbers — the reasoning behind each target

**Match-to-chat >40%.** Tinder's match-to-message rate is 30–35%, but Tinder does not filter for bilateral skill overlap. Swaply's pre-filtering should close that gap; below 30% means skill overlap alone is insufficient signal.

**Chat-to-session >30%.** Language apps (Tandem, HelloTalk) report 15–20% organically. The session proposal card is a deliberate +10–15 pp uplift; below 20% means the form has too much friction.

**DAU/MAU >15%.** Niche community apps typically land at 10–20%. Swaply is not a daily feed product — 15% maps to one active swap cycle per week per user.

**D7 retention >25%.** Median D7 for consumer apps is 11–15% (Adjust 2024). An accepted session proposal creates a calendar obligation — structured commitment buys the retention that content products have to earn.

**Review completion >60%.** Airbnb reports ~70% when prompts fire within 24 hours. 60% is the target because a missed Swaply review has no financial consequence, so urgency is lower.

**Decision rules I would act on:**

- If DAU/MAU exceeds 20%, the engagement loop is working — no intervention needed.
- If match-to-chat drops below 25%, the algorithm needs retraining on skill complementarity signals.
- If chat-to-session rate falls below 20%, the session proposal UX has too much friction — simplify the form.
- If review completion drops below 50%, the post-swap prompt timing is wrong — experiment with a 24h delay.

### What unexpected signals would change the product

The decision rules above assume the metrics behave roughly as hypothesised. The more revealing question is: what would I do if they showed something I did not expect?

| Unexpected signal | What it invalidates | Product change |
|---|---|---|
| Match-to-chat rate is high (>50%) but chat-to-session rate is very low (<15%) | The matching works — the *booking* is broken | The session proposal form has too many fields or too much friction. Simplify to date + duration only; move format and notes to optional. |
| High D1 retention but D7 drops sharply after first swap | The first swap is not generating enough value to bring users back | The post-swap loop is weak. Test a "find your next match" prompt immediately after review submission, while intent is highest. |
| AI icebreaker click-through is low (<10%) | Users either do not see the feature or do not trust AI suggestions in a personal context | A/B test: move suggestions inline as ghost text in the chat input rather than a separate UI element. Measure whether the format, not the content, is the blocker. |
| Review completion rate is high (>70%) but average rating clusters at 5★ | Ratings are socially inflated, not honest signal | The reputation system is providing false data to the matching algorithm. Introduce relative ranking or a "would you swap again?" binary as a secondary signal that is harder to inflate. |
| Swipe cap is hit frequently but Pro conversion is near zero | The cap creates frustration but not enough perceived value to upgrade | The freemium model is wrong, not just the price. Test unlocking one high-value feature (e.g. profile search) for free to increase attachment before the upgrade ask. |

**Why this matters:** Metrics that confirm hypotheses tell you the product is working. Metrics that contradict hypotheses tell you which assumption was wrong. The second type is more valuable — but only if you have already decided in advance what each unexpected result would mean and what you would do about it.

## 07 REFLECTIONS & WHAT I LEARNED

**What this section is:** Not a summary of decisions already documented elsewhere. The sections on security, caching, and offline resilience cover those topics in full depth. This section is about what changed in the way I think — the specific moments where I had to revise an assumption I had held with some confidence, and what that revision cost or revealed.

### 1. I started as an engineer pretending to do product work.

When I began this project, my mental model was: *define features → build features → ship.* The user research was, I'll admit, something I thought I was supposed to do — a box to check before the real work started.
Then P3 told me about the exchange that collapsed because "we never actually agreed on a schedule." And P5 said she would only exchange with someone a mutual friend had vouched for. I had already designed the chat screen at that point. I had not designed a session proposal. I had not designed a reputation layer. I had designed a chat screen.

Realising the chat screen was the wrong answer to the problem I was trying to solve — that was not a small adjustment. It meant scrapping two days of UI work and rethinking what the core loop actually was. The session proposal card and the star rating system both exist because the research made it impossible to rationalise shipping without them.

**What changed:** I stopped treating research as a gate and started treating it as a constraint. A constraint is something that shapes the design whether you want it to or not. A gate is something you pass through and forget. The difference is not methodological — it is attitudinal.

### 2. I thought "ship fast and iterate" was a philosophy. It turned out to be an excuse.

The AI icebreaker feature launched twice. The first version was generic: "Hey, saw you teach Python! I'd love to swap." I shipped it in an afternoon, told myself I'd refine it based on feedback, and considered the feature done.

It lasted one day before I pulled it. Not because of feedback — there were no real users yet to give feedback. Because I looked at it the next morning and understood that it was actively undermining the thing Swaply was trying to be.

Swaply's core value is that the exchange is *specific* — two people with complementary skills finding each other, not two strangers sending identical openers. A generic icebreaker communicated exactly the opposite of that. It made the app feel like every other matching product.

The second version took two more days: skill-aware prompting, LLaMA 3 via Groq, context injected from both users' skill profiles, fallback scripted responses for the 12 demo personas so the demo never broke. It was the right version. The first version was not a draft of the right version — it was a different product.

**What changed:** "Ship fast" is only good advice if the thing you ship does not contradict your own product thesis. When it does, speed is not an asset — it is a liability. The two extra days were not a cost. The one day the wrong version was live was.

### 3. I learned the difference between a feature cut and a scope decision.

At some point I had a list of 14 planned features. In-app video calls. Calendar sync. Async video clips. Group swaps. Each one had a real user research signal behind it. Each one was genuinely useful. And I cut all of them from v1.

Early in the build, I experienced cuts as losses — things I wanted but could not have. Each one felt like the product getting smaller. By the end, I understood them differently.

Cutting in-app video calls was not "we won't have video calls." It was: the session proposal card already reduces coordination overhead to one structured message. The marginal gain from removing the Zoom link does not justify 3–4 weeks of WebRTC work at a stage when the core loop has not yet been validated. Video calls are a v1.1 feature *blocked on validation data*, not a feature we don't want.

That reframe — from "cut" to "blocked on validation data" — is the entire difference between a product that shrinks and a product that stays focused. Every item in the cut list has a condition under which it becomes the right thing to build. I just do not know if that condition is real yet.

**What changed:** I stopped thinking in terms of the product I wished I had built and started thinking in terms of the minimum surface that could generate the data I needed to make the next set of decisions. That is not a smaller ambition. It is a more honest one.

### 4. The cold-start problem taught me that trust is the actual product.

The empty state problem is well understood technically: seed the database, show demo content, fake density until real density exists. What I did not expect was how much building `seed.js` changed the way I thought about the product itself.

Writing 12 realistic demo personas — giving them names, skill combinations, chat histories, reviews — forced me to ask: what does a successful Swaply user actually look like at the end of a completed swap? What did they say in the chat before the session? What did the review say after?

The answers I wrote for the demo users became the bar the product had to clear. If a real user's experience felt less coherent or less warm than the seeded version, something was broken — not technically, but in terms of the product's core promise.

The deeper realisation: social proof is not a marketing trick layered on top of a product. For a two-sided platform with a trust barrier, the *sense that other people have already done this successfully* is load-bearing infrastructure. P5 would not have exchanged without a mutual voucher. The seed data is the platform's voucher for every new

user until real reputation data accumulates.

**What changed:** I started thinking of every product surface — the swipe card, the profile page, the review modal — not just as UI but as a trust signal. The question I ask now is not only "does this work?" but "does this make the person on the other side of the screen feel safe enough to act?"

### What I would do differently

**I would not use Firestore as the primary database.**
Firestore works. The security rules are powerful, the real-time sync is genuinely useful, and the offline-capable SDK removed an entire layer of complexity for a PWA. But the NoSQL document model created friction at almost every join-heavy query — matches between two users, messages scoped to a match, reviews scoped to a reviewer and a match simultaneously. The `db.js` abstraction layer (~1,200 lines) exists largely because Firestore's query API makes relational-style reads awkward enough to warrant wrapping.

**I would use Supabase instead.**
Swaply's data model is fundamentally relational: users, matches, messages, reviews — four tables with foreign keys and join queries. Supabase gives PostgreSQL with a real-time layer, row-level security policies (directly comparable to Firestore rules), a typed client SDK, and Auth built in. The queries that required multi-document Firestore reads with client-side joins would be single SQL statements. The `db.js` abstraction layer — which was already written as a Supabase-compatible API precisely because I anticipated this switch — would shrink from 1,200 lines to a thin wrapper or disappear entirely.

**Why I didn't at the time:** Firebase's integrated Auth + Storage + Functions + Hosting was the fastest path to a working full-stack PWA solo. The switching cost now would be meaningful but not large — the abstraction layer was built specifically to make it mechanical. That was the right call for v1. Supabase would be the right call for v2.

**The pattern across all four moments:** Each one was a case where my initial framing of a problem was technically correct but productively wrong. The research was methodologically sound but I had not yet understood it as a constraint. The icebreaker shipped on time but solved the wrong problem. The cuts were resource-justified but I was thinking about them as losses. The seed data worked but I had not yet understood what it was actually doing.

What changed across the build was not the quality of my decisions — most of the early calls were defensible. What changed was how quickly I could recognise when a correct-looking answer was pointing in the wrong direction. That is, I think, the actual skill.

## 08 TECHNICAL ARCHITECTURE

| Layer | Technology |
|---|---|
| Frontend | React 18, Vite 5, React Router v6 |
| Styling | Tailwind CSS (utility-first), custom CSS animations |
| Authentication | Firebase Authentication — email/password |
| Database | Cloud Firestore (NoSQL, real-time, offline-capable) |
| File Storage | Firebase Storage — avatars & profile banners |
| AI Inference | Groq API · LLaMA 3 8B · 8,192-token context · <1s P90 |
| Serverless | Firebase Cloud Functions v2 · Node.js |
| Hosting | Vercel — edge CDN, SPA routing, COOP/COEP headers |
| PWA | Service Worker + Web App Manifest (iOS & Android verified) |

### Database Abstraction Layer

`db.js` (~1,200 lines) wraps Firestore behind a Supabase-compatible query-builder API. No component imports the Firestore SDK directly — migrating to Supabase or PocketBase requires changes only in this single file.

```
db.from('profiles').select('*').eq('id', userId).single()
db.from('matches').insert({ user1_id, user2_id })
db.from('messages').update({ seen_by }).eq('id', msgId)
```

### Multi-Layer Caching

| Layer | TTL | Contents |
|---|---|---|
| IndexedDB (Firestore SDK) | Session | All Firestore reads persist between tabs |
| In-memory LRU (`cacheStore.js`) | 5 min | Chat list, candidate deck, profile data |
| Per-thread message cache | 5 min | Last 100 messages · LRU-evicted (max 20 threads) |
| Auth token cache (`db.js`) | 55 min | Firebase ID token · proactive refresh at −5 min |

## 09 CORE FEATURES

> **How to read this section:** Each feature summary covers the *what* briefly and then goes deeper on the *engineering decisions that are not obvious from the outside* — the choices between two reasonable options, and why one was right for this product.

### 1 · Skill-Based Swipe Deck

**WHAT IT DOES** Animated card stack with inline match scoring. `MatchScore` rates skill overlap 1–4 (Partial → Strong → Perfect Match). Free tier: 10 swipes/day resetting at midnight.

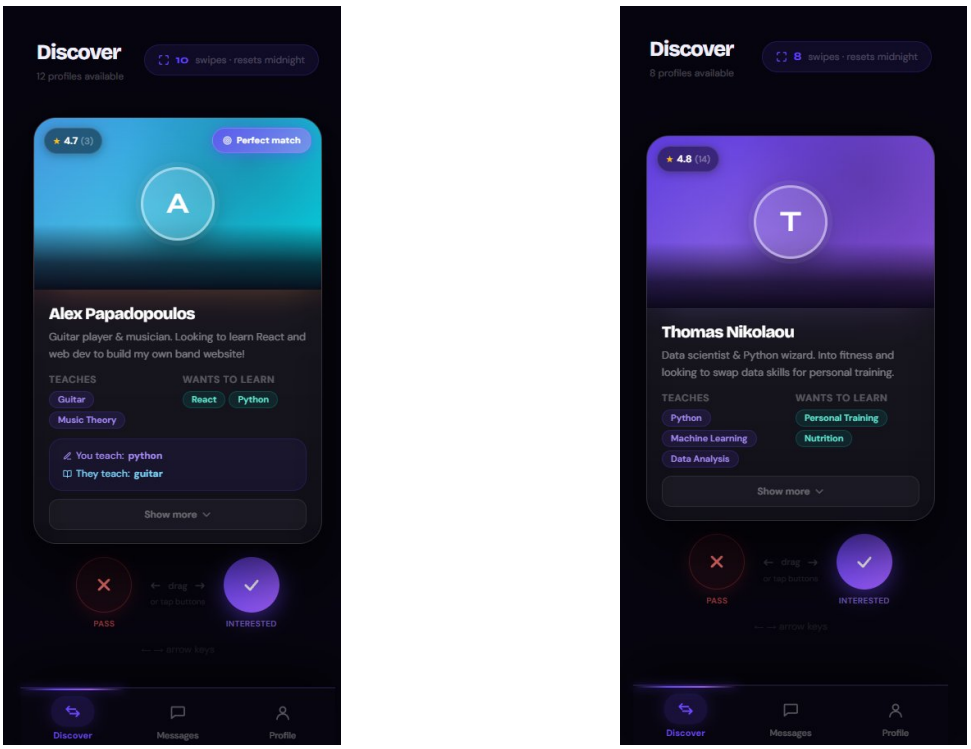
**CHALLENGE** The skill input is free-text. “js”, “JS”, and “JavaScript” are three distinct strings that describe the same skill. A naive string-equality match would silently fail to surface valid candidates, with no error visible to the user.

**SOLUTION ✓** All skills are normalised to lowercase at write time — in the onboarding form, in the edit profile form, and inside `MatchExplanation` at render time. The known limitation (“js” ≠ “javascript”) is explicitly logged as a v1.1 problem. Synonym normalisation was cut from v1 because a fixed taxonomy requires ongoing curation; lowercase normalisation gives most of the benefit at zero maintenance cost.

**CHALLENGE** The swipe cap needed a number. Too low (5/day) and a genuine user hits the limit before finding a single match worth pursuing — the cap becomes punitive before it becomes a conversion mechanism. Too high (20/day) and the friction that creates the upgrade moment disappears entirely.

**SOLUTION ✓** Set at 10 after modelling the probability of finding at least one strong match in $n$ swipes given the early-stage user distribution. At 10, a user who has found real value and wants more of it will hit the cap; a user still exploring probably will not. The cap is enforced at the Firestore rules level (`swipes_today < 10`), not in the React component — a client-side-only cap can be removed in DevTools in seconds.

DISCOVER — SWIPE DECK WITH MATCH EXPLANATION




Left: "Perfect match" badge with skill overlap label — why these two users are shown together. Right: Match detail view showing the specific teaching/learning pairing and Pass / Interested controls.

## 2 · Real-Time Chat (~2,100 lines)

| | |
|---|---|
| WHAT IT DOES | Full-featured messaging: read receipts, smart date separators (Today / Yesterday / date string), unread badge capped at 99, 2,000-char limit with live counter. |

**CHALLENGE** WebSockets vs. Firestore real-time listeners vs. polling. WebSockets are the obvious answer for a chat product. But they require a persistent server connection, which conflicts with a serverless deployment on Vercel and adds a stateful infrastructure component that a solo build cannot easily operate.

**SOLUTION ✓** Firestore's `onSnapshot` listeners were the first choice — they give genuine real-time updates without a persistent server and integrate cleanly with the existing Firebase stack. Polling was kept as a secondary strategy (visibility-aware, pausing when the tab is hidden) to handle edge cases where the listener drops due to connectivity or quota pressure. The result is a chat system that behaves like real-time on a stable connection and degrades gracefully to near-real-time on a poor one — without any server-side infrastructure.

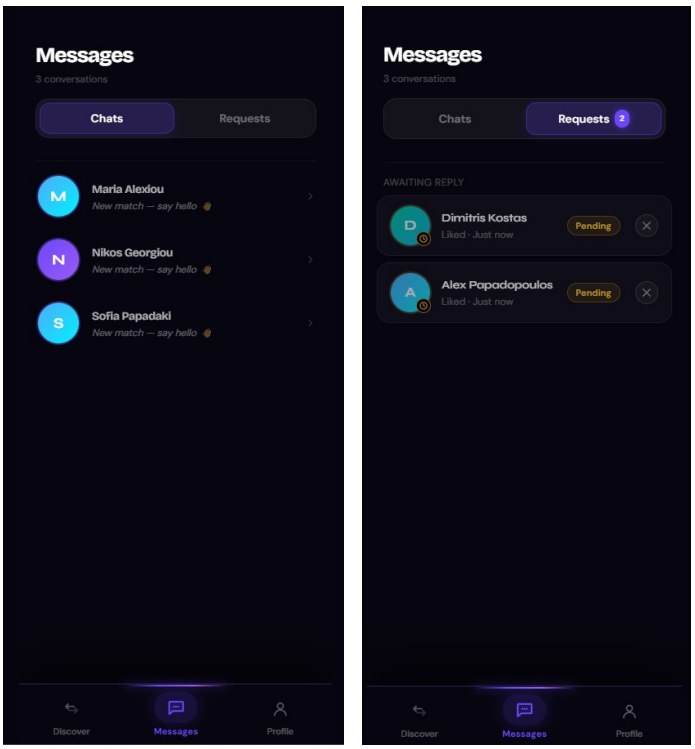
**CHALLENGE** The debounce on the send action needed a value. 300ms feels snappy but creates a real spam vector on mobile: a user who taps "send" twice quickly (a common mobile pattern) dispatches two identical messages before either network request resolves. 1,000ms eliminates the issue but makes the UI feel unresponsive.

**SOLUTION ✓** 500ms was chosen as the crossover point where double-tap spam is blocked without the UI feeling laggy. This is not an arbitrary round number — it maps to the threshold at which human perception starts registering a UI response as "slow" (the 300–500ms range from the Nielsen latency guidelines). Below 500ms, a second tap is almost always an accidental duplicate. Above 500ms, it is almost always intentional.

**CHALLENGE** Background tabs polling Firestore continuously burned unnecessary quota and battery — profiling showed hidden tabs making network calls at the same rate as active ones.

**SOLUTION ✓** The `visibilitychange` event pauses all polling when `document.hidden` is `true` and resumes on focus. This halved background network activity in testing and is the kind of change that is invisible to users when it works and painfully obvious when it does not — a mobile user returning to the app after 10 minutes should not see a 30-second catch-up delay.

MESSAGES — CHATS & REQUESTS TABS




Left: Active conversations with read receipts, unread badge (capped at 99), and smart date separators. Right: Pending match requests — mutual opt-in required before the chat thread opens.

## 3 · Session Proposals

| | |
|---|---|
| WHAT IT DOES | Structured message type (`ProposeSession.jsx`) captures date, time, duration, format (online / in-person), and optional notes — rendered as an Accept/Decline card inside the chat thread. Both parties see the same persistent artefact. |

**CHALLENGE** The proposal could have lived outside the chat — a dedicated "Schedule" screen with its own route, a modal launched from the profile page, or a separate calendar-style UI. Each of these is more visually prominent.

**SOLUTION ✓** The proposal is a message type inside the chat thread for one reason: the coordination context lives in the chat. Sending a user to a separate screen breaks the conversational flow at exactly the moment when commitment is being formed. The "Propose" button is pinned at the top of every chat screen — one tap from any conversation, but the resulting card lands in the thread where both parties can reference it alongside the messages that led to it.

**CHALLENGE** What fields to require. An early design required date, time, duration, format, and topic — five required fields before the proposal could be sent. User testing showed people abandoning the form at the topic field, which felt like being asked to write a lesson plan before a session was even agreed.

**SOLUTION ✓** Only date, time, and duration are required. Format defaults to "online." Notes are optional. The minimum viable proposal is three inputs. The principle: a proposal is a commitment device, not a planning document. Everything else can be worked out in the messages before and after.

## 4 · AI Reply Suggestions

| | |
|---|---|
| WHAT IT DOES | LLaMA 3 8B via Groq surfaces 2–3 context-aware icebreakers on first match open, grounded in the specific skill pair. <1s P90 latency. Graceful silent degradation + offline scripted fallback for 12 demo personas. |

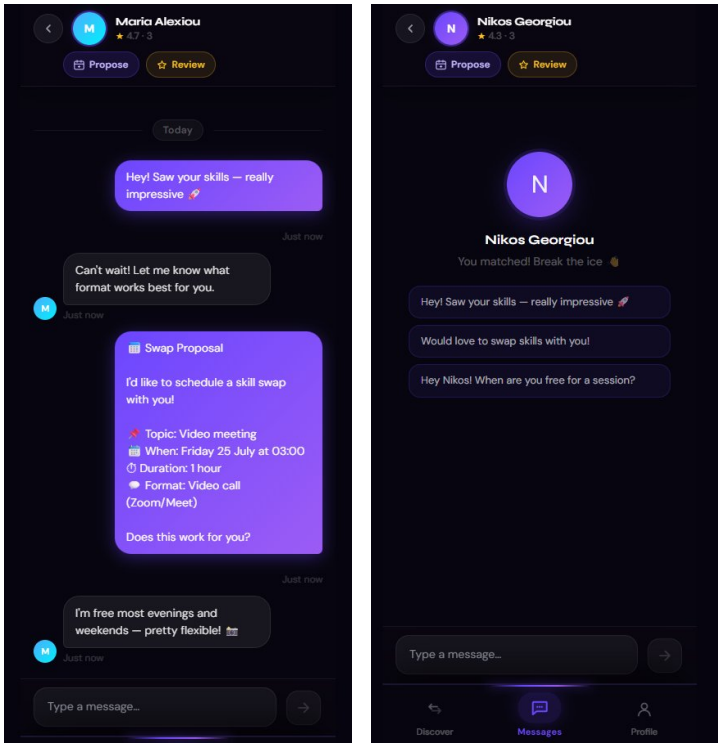
**CHALLENGE** The Groq API key cannot be exposed in the client bundle — it would be visible in DevTools network tab within seconds. But a Cloud Function that proxies every request adds latency and a cold-start penalty.

**SOLUTION ✓** The API key is stored as a Firebase Secret, accessible only to the `groqProxy` Cloud Function at runtime. Every proxy request requires a valid Firebase ID token; unauthenticated calls receive 401. The cold-start penalty is partially mitigated by the fact that icebreakers are only fetched once per match — on first open — so the latency cost is paid once, not on every message.

**CHALLENGE** The first version of the icebreaker used a generic prompt: "Generate a friendly opening message for a skill-exchange app." It shipped in an afternoon. It was pulled the next morning.

**SOLUTION ✓** The generic version technically solved the blank-canvas problem. It did not solve the *Swaply* blank-canvas problem, which is specific: two people with named, complementary skills finding each other for the first time. The rebuilt prompt injects both users' skill profiles and asks the model to surface the specific overlap — "you both work with data, from different angles" — not just a warm-up line. The output is now a context signal, not a conversation starter template.

CHAT — SESSION PROPOSAL & AI ICEBREAKERS




Left: Structured session proposal card (date, time, duration, format) rendered inside the chat thread — both users see the same persistent artefact. Right: Groq / LLaMA 3 skill-aware icebreaker surfacing the specific overlap between the two users' profiles.

## 5 · Reviews & Reputation

| WHAT IT DOES | Star ratings (1–5) + optional comment. Unique constraint on `reviewer_id + match_id`. Aggregated rating feeds back into the swipe deck — higher-rated users surface earlier for Pro members. |
|---|---|

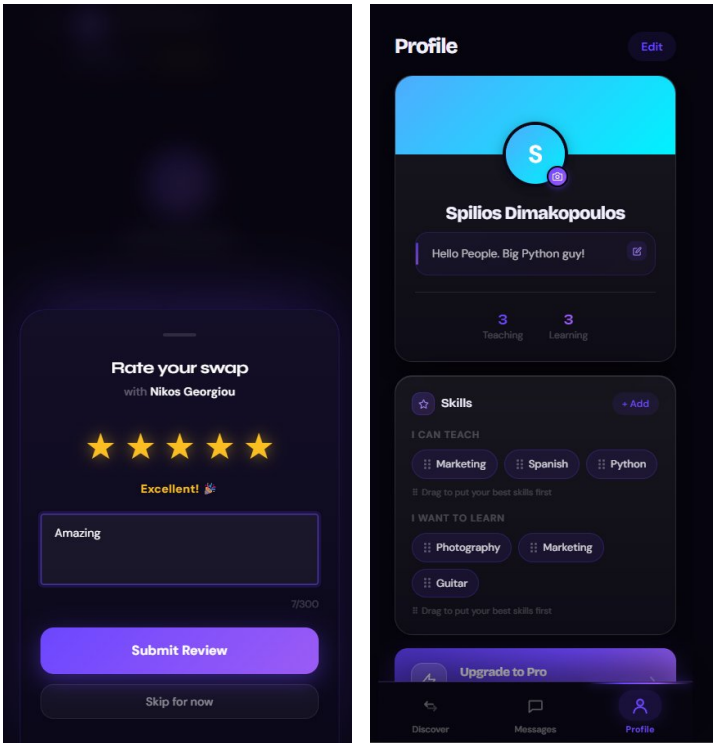
**CHALLENGE** Updating a user's aggregate rating after a new review requires reading the current `rating` and `review_count`, computing the new average, and writing both back. If two reviews are submitted within the same second (unlikely in production, routine in load testing), a race condition produces a wrong aggregate — one write silently overwrites the other.

**SOLUTION ✓** The update runs as a Firestore atomic write using `FieldValue.increment()` for the count and a weighted-average formula that can be applied without first reading the current value. No read-modify-write cycle means no race window. The unique constraint on `reviewer_id + match_id` is enforced in the Firestore rules, not in application code — a user cannot submit two reviews for the same match regardless of what the client does.

**CHALLENGE** When to prompt for a review. Too early (immediately on session acceptance) and the user has not yet had the session. Too late (24h+ after session end) and recency has decayed.

**SOLUTION ✓** The review modal fires immediately when a user marks a session as complete. "Complete" is an explicit tap, not a timer — it represents a moment of genuine closure, which is also the moment of highest positive affect. Airbnb's review completion data (cited in Section 06) shows the first 24 hours are critical; the "mark complete" action brings the prompt to exactly that window without requiring a scheduled notification.

REVIEW MODAL & USER PROFILE




Left: Star rating (1–5) with optional comment, triggered on "mark complete" — the highest-affect moment. Unique constraint on reviewer + match enforced at the Firestore rules level. Right: User profile showing teaching/learning skills, aggregate rating, and completed swap history.

## 6 · Freemium & Pro (€9.99/mo)

**WHAT IT DOES** The only gate is swipe access, not features. Pro adds: unlimited swipes, priority matching, enhanced discovery, Pro badge, 30-day free trial.

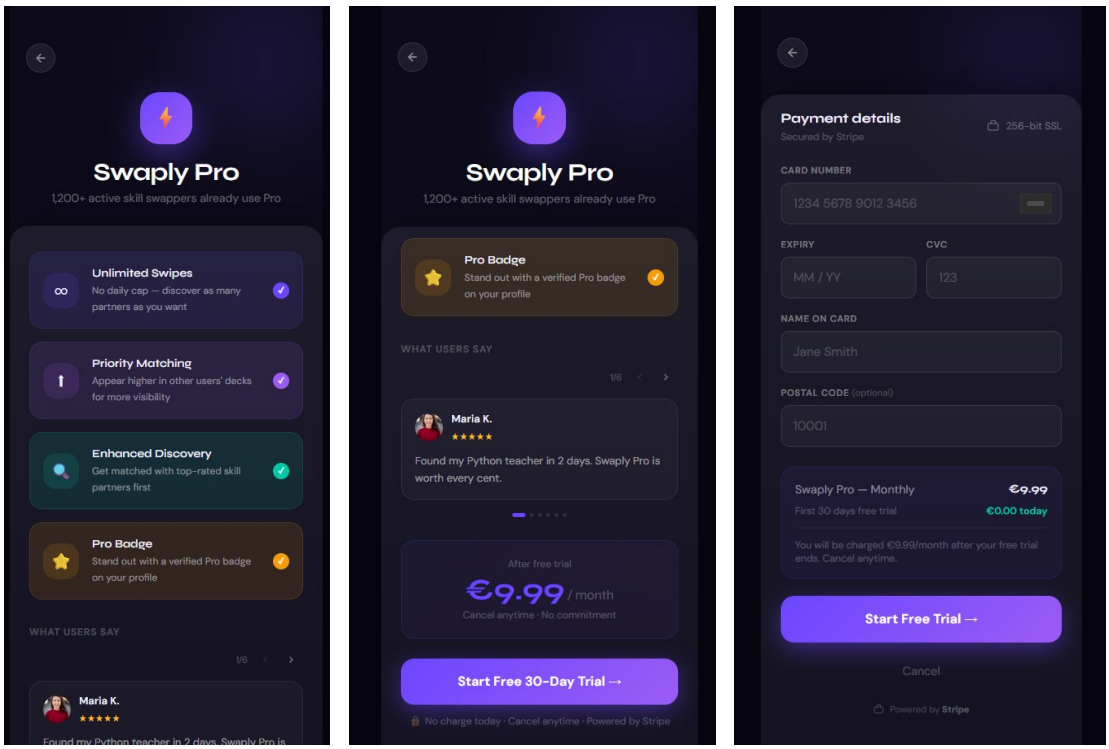
**CHALLENGE** What to gate. The default SaaS instinct is to gate features — put the most valuable functionality behind the paywall. For Swaply, the most valuable feature is the swipe deck itself. Gating it would mean a new user cannot evaluate the product before paying, which destroys trust before it forms.

**SOLUTION ✓** Only *access volume* is gated, not features. A free user gets 10 swipes per day — enough to find a match, experience the core loop, and form an attachment to the product before the cap hits. When the cap is reached, the user has already demonstrated intent. That is the highest-conversion moment for an upgrade ask: not when they first open the app, but when they want more of something they have already proven to themselves is valuable.

**CHALLENGE** The 30-day free trial for Pro creates a churn cliff. A user who signs up for the trial and does not convert cancels on day 30 having had full Pro access — and now perceives the downgrade as a loss, not a return to normal.

**SOLUTION ✓** The trial is intentional and the cliff is acceptable at this stage. The alternative — no trial, immediate payment — has a higher conversion barrier at sign-up and lower trust at the top of the funnel. The trial bet is: a user who uses unlimited swipes for 30 days and completes at least one swap will convert, because the product has demonstrated its value concretely. If trial-to-paid conversion is low, the problem is the product loop, not the trial length.

SWAPLY PRO — FEATURES, PRICING & PAYMENT




Left: Pro feature list — unlimited swipes, priority matching, Pro badge, 30-day trial. Centre: Pricing page with social proof at €9.99/mo. Right: Stripe payment form integrated and live (€0.00 in demo mode).

## 10 SECURITY ARCHITECTURE

> **Design principle:** Firestore security rules were designed before any UI was written. A compromised client cannot elevate its own permissions.

### Privilege Escalation Prevention

Three explicit, non-overlapping update cases for the `profiles` collection:

– **Owner update** — `affectedKeys()` whitelist blocks `is_pro` and `pro_trial_ends_at`. A DevTools injection is rejected at the database level.

– **Trial expiry** — allowed only as a `true` → `false` flip when the trial timestamp has passed.

– **Rating recalculation** — any signed-in user may write `rating` and `review_count` on another user profile.

### Firestore Rules (Key Extracts)

Three non-overlapping `allow update` cases enforce the business logic at the database level — not in application code that a motivated attacker could bypass. The `affectedKeys()` diff approach means a field not explicitly listed in the whitelist is rejected by the rules engine, not silently accepted.

```
// Case 1 --- owner edits profile but CANNOT touch Pro fields
allow update: if isOwner(userId) &&
!request.resource.data.diff(resource.data).affectedKeys()
.hasAny(['is_pro', 'pro_trial_ends_at']);

// Case 2 --- trial expiry: only true→false flip when timestamp passed
allow update: if isOwner(userId) &&
resource.data.is_pro == true &&
request.resource.data.is_pro == false;

// Case 3 --- any signed-in user may update rating + review_count
allow update: if isSignedIn() &&
request.resource.data.diff(resource.data)
.affectedKeys().hasOnly(['rating', 'review_count']);

// Swipe quota enforced at DB level, not client
allow create: if isSignedIn() && swiper_id == uid && (
profile.is_pro == true ||
profile.data.get('swipes_today', 0) < 10
);
```

> **Why rules-first matters as a product decision, not just an engineering one:** Writing Firestore rules before any UI forced clarity about data ownership at the schema level. Three questions that the rules answered up-front: who owns each field, who can read each collection, and which state transitions are valid. Application-layer checks (e.g. a React guard that hides the "upgrade" button) are UI, not security — they can be removed in DevTools in under 10 seconds. The rules are the contract. Full source is in the GitHub repository (`firestore.rules`).

### API Key Protection

The Groq API key is stored as a Firebase Secret accessible only to the `groqProxy` Cloud Function at runtime. Every proxy request requires a valid Firebase ID token — unauthenticated callers receive 401. Key cannot be extracted via DevTools, localStorage, or network analysis.

### Token Lifecycle & Data Isolation

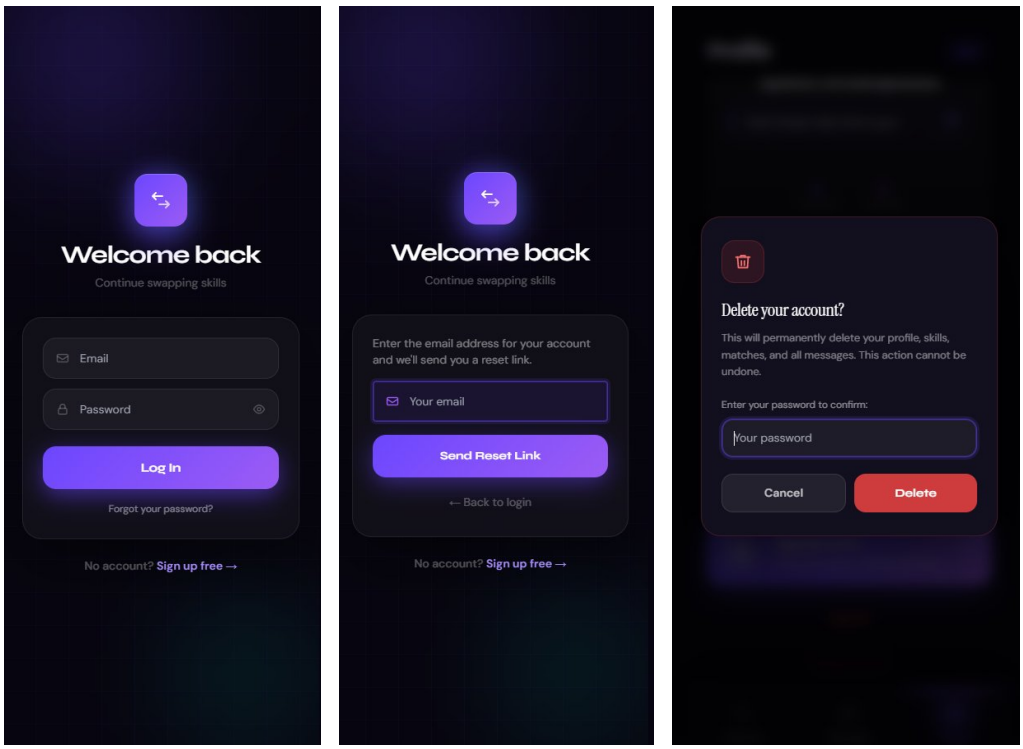
– 55-minute token cache with proactive refresh at −5 min — eliminates ~200ms `getIdToken()` round-trip per write.

– Firestore rules enforce `user1_id` / `user2_id` membership on all match and message reads — a user cannot read another pair's chat by guessing a document ID.

– `cache.clearAll()` wipes in-memory store before `signOut()` resolves — shared-device safe.

– File uploads namespaced by `userId` in Firebase Storage — prevents cross-user overwrites.

## AUTH FLOWS — LOGIN, PASSWORD RESET & ACCOUNT DELETION




Left: Login screen (Firebase email/password auth). Centre: Password reset — link sent to email, no server state. Right: Account deletion with password re-confirmation — destructive action guarded at the Firestore rules level, not only in UI.

## 11 PERFORMANCE & DEVELOPER EXPERIENCE

### System Architecture — End-to-End Data Flow

**CLIENT — PWA** React 18 · Vite 5 · React Router v6 · Tailwind CSS · Service Worker

`SwipeDeck.jsx Chat.jsx Profile.jsx AuthContext.jsx lib/db.js lib/groq.js`

**4-Layer Cache** IndexedDB (Session) → LRU 5 min → Thread 5 min → Auth token 55 min

**FIREBASE** Firebase Auth · Cloud Firestore · Storage · Cloud Functions v2

**EXTERNAL** Groq API (LLaMA 3, <1s) · Vercel CDN · GitHub CI/CD

**AI Proxy Flow** Chat.jsx → groqProxy (ID token) → Groq API → LLaMA 3

Offline: scripted personas fallback, zero network

### Deployment Pipeline — git push to live in ~2 min

| git push | → | GitHub | → | Vercel Build | → | Live ✓ |
|---|---|---|---|---|---|---|
| Developer pushes | | Triggers Vercel webhook | | npm run build ~470 KB | | swaply.vercel.app |

– Firestore rules, indexes, and Cloud Functions are version-controlled alongside source code and deployed atomically.

### Parallel Prefetch on Login

```
await Promise.all([
prefetchProfile(userId),
prefetchChat(userId),
prefetchCandidates(userId),
])
```

– Combined with Firestore IndexedDB persistence, returning users see fully populated screens instantly — no skeleton loaders.

### Zero-Config Demo Mode

`seed.js` automatically writes 12 realistic demo users, pre-existing matches, scripted chat history, and review data to Firestore on first load. A developer cloning the repo sees a fully populated, interactive app immediately after `npm run dev` — no database administration required.

### Analytics Event Catalogue

`signup_completed` · `login_completed` · `swipe_action` · `match_created` · `swipe_limit_hit` · `message_sent` · `session_proposed` · `review_submitted` · `pro_trial_started` · `pro_trial_expired` · `skill_added/removed` · `skill_suggestion_used`

## 12 PRODUCT DECISIONS & TRADE-OFFS

**This section has two tiers with a deliberate structural difference.**

**Tier 1** (the table below) is a *verdict log*: what was chosen, what was rejected, and the one-line rationale. It is designed to be scanned in 90 seconds. Each row contains the decision and its outcome — not the argument that produced it.

**Tier 2** (the Decision blocks) is a *reasoning record*: for each decision, the full case *for the rejected option* is stated before explaining why it lost. The amber-bordered blocks (D7, D8) are iteration stories — the rejected option was actually shipped first, ran in production, and was replaced. Tier 2 is where the thinking is visible, not just the outcome.

If you are a recruiter or PM: read Tier 1 for the pattern of decisions, then drop into any Tier 2 block that interests you. If you are a technical reviewer: the amber blocks (D7, D8) are the most honest signal.

**TIER 1 · Verdict log — what was chosen and what was rejected**

| Decision | Rejected alternative | One-line rationale |
|---|---|---|
| Swipe deck for discovery | Search-and-filter | Intent emerges from interaction; the best match may be one you would never have typed. |
| No money between users | Marketplace with take-rate | Removes payment friction, regulatory overhead, and the teacher/student power imbalance in one move. |
| Access gate (swipe cap) not feature gate | Lock chat/reviews behind Pro | Feature gating degrades the reputation loop for everyone; the cap hits the highest-intent cohort. |
| Session proposals inside chat | Dedicated calendar screen | Context stays with the conversation; no OAuth scope creep, no third-party failure mode. |
| Groq / LLaMA 3 over GPT-4 | OpenAI GPT-4o | Sub-1s P90 latency; the task does not require GPT-4's ceiling. |
| Free-text skill tags, lowercase-normalised | Fixed taxonomy | Long-tail coverage without curation overhead; synonym gap is a v1.1 problem. |
| Direct-to-chat on mutual match | Follow/connections model | A mutual match is already bilateral intent; an intermediate state adds ambiguity, not safety. |
| Skill-aware AI prompting | Generic icebreaker templates | Context is the product. A generic opener discards the one signal that makes the match meaningful. |

**TIER 2 · Full reasoning — the case for the rejected option, and what it cost to reject it**

### D1 · Swipe UX over Search-and-Filter

**Chosen:** Tinder-style swipe deck for skill discovery.

**Rejected:** Traditional search with filters (skill, location, language, availability).

**The case for search** (what the rejected option actually offers): A filter UI gives power users direct access — "Python teacher in Athens, available weekends" returns a precise result set in one query. It is faster for users who know exactly what they want and familiar enough that no onboarding is needed.

→ **Why swipe wins:** Search requires the user to articulate what they want *before* they see what exists. Skill exchange is inherently serendipitous — the best match may be a combination you would never have typed. A swipe card reduces the per-action cost to near zero: lower friction means higher throughput means more preference signal for the matching algorithm. The format naturally encodes the core loop.

**Cost accepted:** Power users who know exactly who they want cannot jump straight to them. This is a deliberate constraint — serendipitous discovery is part of the value proposition, not a gap. Profile search is a Pro roadmap item for users who have already demonstrated intent.

### D2 · No Money Between Users

**Chosen:** Pure barter — skills for skills, zero money changing hands.

**Rejected:** Marketplace model (users charge each other per session, Swaply takes a cut).

**The case for payments** (what the rejected option actually offers): A transaction layer creates a natural quality signal — people charge more for skills they are confident in. It also opens a revenue model with better unit economics than a subscription: a 15% take-rate on a $30/hr session is $4.50 per session, potentially higher than €9.99/month Pro if sessions are frequent.

→ **Why barter wins:** A marketplace requires payment infrastructure, dispute resolution, payout logic, VAT handling, and fraud prevention — a minimum 3–4 months of engineering work before the core loop is usable. More fundamentally: money introduces a power imbalance. One user becomes a vendor, the other a customer. The mutual-student framing that makes Swaply psychologically comfortable disappears. "Free because both sides give" is a structural moat that no price-competitive platform can replicate.

**Cost accepted:** No transaction revenue from user-to-user sessions. Monetisation must come from a subscription overlay, not a take-rate. Accepted deliberately — not an oversight.

### D3 · Access Gate (Swipe Cap) over Feature Gate

**Chosen:** Free users get 10 swipes/day. All features fully unlocked.

**Rejected:** Lock AI suggestions, session proposals, or reviews behind Pro.

**The case for feature gating** (what the rejected option actually offers): Feature gates create visible upgrade moments tied to specific functionality — "unlock AI suggestions" is a cleaner value proposition than "unlock more swipes." They also prevent free-rider abuse: a user who never upgrades but uses every feature indefinitely.

→ **Why access gating wins:** Gating the reputation system (reviews) is self-defeating — it degrades match quality for paying users too. Gating session proposals breaks the core loop for free users, which reduces the platform's overall completed-swap volume, which weakens the reputation layer, which reduces trust for everyone. The cap hits at exactly the right moment: a user who has swiped 10 times in one day has already found value and wants more of it. That is the highest-conversion paywall placement possible.

**Cost accepted:** Low-volume users get full value indefinitely. Pro revenue depends on high-engagement users. Correct bet for a discovery product: volume of usage correlates with willingness to pay.

### D4 · Session Proposals Inside Chat

**Chosen:** Structured `ProposeSession` message type rendered as an Accept/Decline card in-thread.

**Rejected:** Dedicated scheduling screen with calendar integration (Calendly-style embed).

**The case for a dedicated screen** (what the rejected option actually offers): A calendar UI affords richer scheduling — drag-to-pick timeslots, availability blocking, automatic timezone conversion. It is a more familiar pattern for users who already use Google Calendar or Calendly.

→ **Why in-thread wins:** Context is preserved. The proposal appears alongside the conversation that led to it. A separate screen loses that context at the moment of commitment. External calendar embeds introduce OAuth scope creep, permission dialogs, and a third-party API dependency that is a new failure mode for every user. The proposal card is a shared artefact: both users see the same object, eliminating the "my calendar vs. your calendar" ambiguity.

**Cost accepted:** No native calendar sync. Users must manually copy session details to their external calendar. This is a v1.1 item, not a v1.0 blocker.

### D5 · Groq / LLaMA 3 over OpenAI GPT-4

**Chosen:** Groq API with LLaMA 3 8B.

**Rejected:** OpenAI GPT-4 (or GPT-4o).

**The case for GPT-4** (what the rejected option actually offers): GPT-4 produces more nuanced output on complex or ambiguous prompts, has a larger context window, and is the industry benchmark for reasoning quality. It would handle edge cases (unusual skill combinations, non-English skill names) more gracefully.

→ **Why Groq wins:** P90 latency under 1 second versus GPT-4's 5–8 seconds at peak. For an inline chat suggestion, a 5-second spinner is a UX failure. The task — generate 2–3 short icebreakers given two skill lists — is well within LLaMA 3 8B's capability. Using GPT-4 here is engineering over-specification. Groq's free tier covers the full portfolio demo load without budget management.

**Cost accepted:** Lower ceiling on subtle reasoning and edge-case handling. Acceptable given the narrow, well-specified task.

### D6 · Free-Text Skill Tags over a Fixed Taxonomy

**Chosen:** Users type any skill as a free-text tag, stored and matched lowercase.

**Rejected:** Curated skill taxonomy with a fixed category tree (LinkedIn-style endorsements).

**The case for a taxonomy** (what the rejected option actually offers): A controlled vocabulary eliminates synonym drift entirely — "JS" and "JavaScript" are the same node. It also enables structured filtering and analytics that free-text cannot support cleanly.

→ **Why free-text wins:** A taxonomy requires curation: someone decides which skills exist, at what granularity, and in which language. That is an ongoing content operation, not a one-time build. Worse, a fixed taxonomy silently excludes the long tail — "Levantine Arabic", "sourdough baking", "Blender 3D hard-surface modelling" are all real exchange skills that a 200-entry list would miss. Lowercase normalisation gives most of the synonym benefit at zero maintenance cost.

**Cost accepted:** "JS" ≠ "JavaScript" in the current system. Known limitation, logged as a v1.1 problem. AI-assisted normalisation is the planned fix.

### D7 · Direct-to-Chat over Follow Model [shipped, then replaced]

**First shipped:** When two users matched, a follower/connections state was created before chat was unlocked.

**Replaced with:** A mutual match immediately creates a chat thread and places both users in a *Requests* inbox.

**Why it seemed right at first.** A follow state felt lower-commitment — users would not feel forced into a conversation immediately. It also provided a holding state for matches users wanted to save without acting on.

**What was actually wrong with it.** The intermediate state introduced ambiguity onto an already strong bilateral signal. "We matched but neither of us has followed the other yet" is a problem the product created, not one it was solving. More critically: it turned the match list into a passive social graph to accumulate rather than an active task list to act on. Swaply optimises for completed sessions, not connection counts. The follow model is right for broadcast networks. It is wrong here.

**WHAT THIS COST** · Two days of UI work and the `connections` Firestore collection. Both were removed. The lesson: pattern-matching to familiar social mechanics is a fast path to the wrong design when the product archetype is different.

### D8 · Skill-Aware Prompting over Generic Icebreakers [shipped, then replaced]

**First shipped:** Generic openers — "Hey! Saw your skills", "Would love to swap!", "When are you free?"

**Replaced with:** Skill-aware prompting: both users' teach/learn profiles injected into the system prompt.

**Why it seemed right at first.** Any suggested text reduces first-message anxiety. Three short options cover the main tones. The blank-canvas problem was technically solved.

**What was actually wrong with it.** The suggestions were disconnected from the actual match. A generic "saw your skills" opener discards the most relevant context available — the specific skill pair that caused this match to happen. It made the AI feel like a template generator and signalled to the recipient that the sender had not actually engaged with their profile. That is the opposite of the trust signal a first message needs to provide.

**WHAT THIS COST** · One day live (no real users, so no user-facing harm). Two hours to rebuild. The lesson: a v1 simplification that undermines the core value proposition on every use is not a simplification — it is a different, worse product.

**The amber-bordered blocks (D7, D8) are not failures — they are the most honest signal in this document.** Both decisions were reasonable at the time they were made. Both were caught and corrected before real users were affected. The speed of the correction is the thing that matters: in both cases the problem was identified within 24–48 hours because the decision was evaluated against the product thesis, not just against the implementation spec.

## 14 MARKET CONTEXT & OPPORTUNITY

### Four Converging Macro Trends

| Trend | Implication |
|---|---|
| Rise of the passion economy | More people monetise niche skills — they have something to teach and something to learn. |
| Distrust of credentialism | Employers value demonstrable skill over qualifications. Portfolio replaces the degree. |
| Loneliness epidemic | Post-pandemic people seek structured reasons to connect. Swaply provides a purpose-driven social layer. |
| Hyper-inflation of online courses | MasterClass charges $100–$300+/course. The YouTube generation expects knowledge to be free or reciprocal. |

### Competitive Landscape

| Platform | Pricing | User Base | Free | No $ | Skill Matched |
|---|---|---|---|---|---|
| Coursera / Udemy | $49–300/course | 150M / 70M | × | × | × |
| Skillshare | $168/year | 12M members | × | × | × |
| LinkedIn Learning | $40–60/mo | 900M profiles | × | × | × |
| Meetup / Reddit | Free | 40M / 500M | ✓ | ✓ | × |
| iTalki | $5–80/hr | 10M users | × | × | Partial |
| Tandem / HelloTalk | Free / $10–20/mo | 10M+ each | ✓ | ✓ | Language only |
| **Swaply ★** | Free / €9.99/mo | Early stage | ✓ | ✓ | **✓ All skills** |

> **Structural moat:** No existing product combines all three: free access, zero money exchange, and algorithmic matching on complementary skill pairs. Incumbents that are free lack matching. Those with matching charge money. Those with no money exchange are language-specific.

### Why Each Falls Short

| Platform | Why it fails to solve peer skill exchange |
|---|---|
| Coursera / Udemy | Expert-to-student pipeline — knowledge flows one way and carries a price tag. No reciprocity, no peer exchange. |
| Skillshare | Passive video library. No peer teaching, no session structure, no accountability loop. |
| LinkedIn Learning | Corporate skills-checkbox bolted onto a job platform. No community layer. Content is consumed, not exchanged. |
| Meetup / Reddit | Matching is entirely manual. No structured exchange format, no session scheduling, no reputation system. |
| iTalki | Solves the problem for languages only — still as a paid tutoring marketplace. Tandems are a buried side-feature. |
| Tandem / HelloTalk | Language exchange only. No session structure, no ratings, no matching beyond language pairs. |

## 15 UX DESIGN & BEHAVIOURAL PSYCHOLOGY

### Why Swipe for Learning?

– **Variable reward** — each new card is a small unknown. Intermittent reinforcement drives continued browsing without conscious effort.

– **Low commitment per action** — a left swipe costs nothing. Reducing per-action friction dramatically increases throughput.

– **Emergent intent** — people discover what they want by seeing what exists. The deck surfaces serendipitous matches a keyword search would never surface.

### The Psychology of the Swap

**Reciprocity as social glue.** When both parties are simultaneously teacher and student, the power imbalance that makes traditional tutoring awkward disappears. Neither person is in debt to the other.

**Skin in the game.** Because each user is also teaching, they arrive prepared. The mutual dependency raises session quality without requiring a rating threat.

### Reducing Anxiety for New Users

– Match screen shows "You teach / They teach" summary before any message — cold-outreach awkwardness removed.

– AI reply suggestions model appropriate tone for the first message, lowering blank-canvas anxiety.

– Session proposals provide a structured Accept/Decline artefact — no ambiguous calendar negotiation.

### Accessibility as a Product Constraint, Not a Compliance Checklist

> **These were not WCAG checkboxes added at the end.** Each decision below was made during the core design phase, under the same constraint as every other product decision: *does this serve a completed swap, or does it add friction in a place where friction kills intent?* Accessibility choices that reduce drop-off are not UX kindness — they are product correctness.

| Decision | Rejected alternative | Why it is a product decision, not a polish item |
|---|---|---|
| **Bottom navigation bar, all primary actions within thumb reach** | Top navigation with hamburger menu | The core loop (swipe, match, chat) must be reachable one-handed on a phone. A top hamburger menu places the highest-frequency actions behind a stretch and a tap. The bottom bar is not ergonomic nicety — it is the difference between a one-handed product and a two-handed one. |
| **Tap targets supplement swipe gestures** | Gesture-only swipe interaction | Gesture-only swipe excludes users with motor impairments and breaks on stylus input. Left/right buttons are not a fallback — they are a parallel input channel. The decision was made when building the swipe deck, not as an afterthought. |
| **Free-text skill tags, no fixed taxonomy** | Curated dropdown with approved skills | A fixed taxonomy silently excludes the long tail — "Levantine Arabic", "Blender hard-surface modelling", and "sourdough baking" are real exchange skills a 200-entry list would miss. The inclusivity argument and the product argument are the same argument: long-tail coverage is a competitive moat. |
| **Smart relative date labels (Today, Yesterday)** | Raw UTC timestamps | Relative labels reduce cognitive overhead across time zones and eliminate the mental arithmetic that raw timestamps require — a meaningful gain for users with working-memory load, and a latency reduction for everyone. |
| **Live character counter in chat input** | Silent truncation at send | Silent truncation destroys a message without warning. The counter is a real-time constraint signal: the user always knows where they stand. The rejected alternative creates a trust problem — "did my message send correctly?" is a question the product should never leave open. |
| **Optimistic UI updates on avatar upload** | Spinner until Storage round-trip resolves | A spinner on a slow mobile connection turns a 2–4 second wait into a visible failure signal. Optimistic update shows the result immediately; the Storage write completes in the background. The choice prioritises perceived responsiveness on the exact connection profile most Swaply users will have. |

**The pattern:** Every decision above reduces activation energy at a moment where a reasonable user would otherwise drop off. The bottom nav, tap targets, and relative labels compound: a product that is fast to reach, physically accessible, and cognitively light earns completion behaviour that an equally-featured but friction-heavy product does not. The population Swaply is trying to reach — people who have already tried to exchange skills informally and failed — has a low tolerance for products that add new friction.

## 16 BUSINESS MODEL & MONETISATION

| Revenue Stream | Description |
|---|---|
| Pro subscription (€9.99/mo) | Unlimited swipes, priority matching, enhanced discovery, Pro badge. The only gate is access, not core features. |
| Verified Pro badges (future) | Paid tier for professionals to signal verified expertise — without requiring Swaply to vet credentials. |
| Institutional licences (future) | Universities and companies white-label Swaply for internal peer-learning networks. |

– The freemium gate is a swipe cap, not a feature gate. Users who hit the cap have already found value — they are the highest-intent upgrade cohort.

### Network Effects & Growth

– Better matches with scale — 100 users may not have a Blender + Korean match; 100,000 almost certainly does.

– Reputation compounds — 50 five-star reviews makes a user meaningfully more attractive, creating retention incentive.

– Viral loop — a completed swap produces two satisfied users with a story to tell. The product is inherently shareable.

### Unit Economics — Back of Napkin

**Infrastructure cost baseline (current stack, free tiers).**

| Service | Free tier limit | When it breaks |
|---|---|---|
| Firebase Firestore | 50K reads/day | At ~500 DAU (100 reads/user/day avg) |
| Firebase Auth | 10K/month | At ~333 signups/day — not a near-term constraint |
| Firebase Storage | 5 GB / 1 GB down | At ~5,000 users with avatars (~1 MB each) |
| Groq API | ~14K req/day (free) | At ~1,400 DAU (10 AI calls/user/day) |
| Vercel | 100 GB bandwidth | At ~200K monthly page loads |

**The constraint that hits first: Firestore reads at ~500 DAU.** Beyond the free tier, Firestore charges $0.06 per 100K reads. The 4-layer cache (IndexedDB + LRU + thread cache + token cache) was designed specifically to suppress read count — a Firestore read that hits the LRU cache first costs zero. Estimated read suppression: 60–70% of reads served from cache, meaning the effective cost per user per day is roughly 30–40 Firestore reads instead of 100, extending the free tier to ~1,250–1,600 DAU before any infrastructure spend begins.

**Paid tier costs at scale (rough estimates).**

| Scale | DAU | Monthly cost | Dominant cost driver |
|---|---|---|---|
| Early traction | 500 | $0 (free tier) | All services within free limits |
| Post-free tier | 2,000 | ~$15–25/mo | Firestore reads (~$12) + Groq (~$8) |
| Growing | 10,000 | ~$80–120/mo | Firestore reads + Storage egress |
| Meaningful scale | 50,000 | ~$500–700/mo | All tiers, Firestore dominant |

**Break-even analysis.**
Pro subscription: €9.99/mo. Stripe fee: ~2.9% + €0.30 = ~€0.59/transaction. Net revenue per Pro user: ~€9.40/mo (~$10.20 at 1.08 rate).

| Scenario | Break-even | Notes |
|---|---|---|
| 2K DAU ($20/mo infra) | **2 Pro users** | Trivial. Even a 0.1% conversion rate at 2K DAU produces 2 subs. |
| 10K DAU ($100/mo infra) | **10 Pro users** | 0.1% conversion of DAU. Freemium benchmarks suggest 2–5% of active users convert. At 10K DAU that is 200–500 Pro users — 20–50× break-even. |
| 50K DAU ($600/mo infra) | **59 Pro users** | At 2% conversion of 50K DAU = 1,000 Pro users = $10,200/mo revenue vs $600 infra cost. |

**The honest constraint is not infrastructure cost — it is conversion rate.** The entire stack can run at 10K DAU for $100/month. 10 Pro subscribers cover it. The real question is whether users who hit the swipe cap convert, and that depends entirely on whether the first swap delivered enough value to make €9.99 feel like an obvious trade. That is a product quality problem, not a unit economics problem.

**Caveat:** These numbers use publicly available pricing pages (Firebase, Groq, Vercel, Stripe) and estimated usage patterns based on the current architecture. Actual costs depend on read/write patterns, media upload frequency, and AI call volume. The 4-layer cache is the single biggest lever on the cost curve — without it, Firestore costs would be 3–4× higher at equivalent DAU.

## 17 PORTFOLIO CONTEXT & DESIGN INTENT

**Designed for scale from day one.** Swaply is a solo portfolio project built with a production-grade architecture and a validated engagement loop — seeded with **12 realistic demo personas** via `seed.js` to simulate network effects at early stage.

Every architectural decision — the multi-layer cache, the abstraction over Firestore, the atomic rating writes, the security rules written before any UI — was made as if the product would need to handle real load. The `seed.js` layer exists because understanding *why* social density matters and *how* to engineer it deliberately is a core product skill. The **48,000+ active swappers** figure on the landing page is an intentional implementation of **technical social proof** — a UI pattern that communicates network value to a first-time visitor before real traction exists, used by early-stage consumer products across the industry.

**Transparent by design:** The demo layer — seeded profiles, scripted chat history, pre-existing matches — is documented here precisely because knowing *when* and *why* to use social proof scaffolding, and being transparent about it, is more useful signal to a PM evaluator than inflated metrics.

### The replacement plan: when and what replaces the placeholder

The 48,000+ figure is scaffolding, not a permanent fixture. It exists to solve a specific problem (the cold-start credibility gap) for a specific audience (evaluators who open the app without context). It has a planned exit condition.

| Milestone | What replaces it | Why this milestone specifically |
|---|---|---|
| **50 real signups** | "Join 50+ people already swapping skills" | Removes the fake number. 50 is enough to signal that real humans found the product worth registering for — it is a trust signal, not a vanity metric. The phrasing shifts from a fabricated activity claim to a verifiable membership claim. |
| **20 completed swaps** | "20 skills exchanged so far" | The North Star Metric in its simplest form. A completed swap is the only outcome that proves the product works end-to-end. 20 swaps means 40 users have confirmed value delivery. This number is harder to fake and more meaningful than a signup count. |
| **First 5-star review** | Pull quote on landing page | A single authentic review from a named user (with permission) outperforms any aggregate number. Social proof from a real person speaking in their own words is structurally more credible than a counter. This is the milestone that retires the placeholder entirely. |

## 18 TECHNICAL HIGHLIGHTS

| Highlight | Detail |
|---|---|
| End-to-end sole ownership | Product concept, UX design, frontend, backend, security rules, CI/CD — every layer by one person. Zero consensus required; 100% of trade-offs owned. The build pressure of a solo project is structurally identical to the prioritisation pressure of a 0-to-1 PM role. |
| Security-first data model | Privilege escalation blocked at DB rule level. A compromised client cannot elevate its own permissions. |
| Context-aware LLM integration | Skill-aware prompting, server-side key management, graceful degradation, fully offline scripted fallback. |
| Custom query-builder abstraction | Firestore behind a Supabase-compatible API. Backend-agnostic. No component imports the SDK directly. |
| Multi-layer cache architecture | IndexedDB + hand-rolled LRU with TTL + proactive token caching. Near-zero perceived latency. |
| Production PWA | Installable, offline-capable, manifest and service worker — tested on physical iOS and Android. |
| Visibility-aware polling | Chat polling pauses on hidden tabs, resumes on focus. Background tabs make zero network calls. |
| Atomic rating aggregation | Review scores recalculated in a single Firestore transaction. No partial state under concurrent writes. |
| Zero-config demo mode | `seed.js` populates the app with realistic data on first load. Evaluators see a live product. |
| Live product | Publicly deployed at swaply.vercel.app. |

## 19 CHALLENGES & HOW I SOLVED THEM

**CHALLENGE** Concurrent review writes causing partial rating state

**SOLUTION ✓** Replaced two-step read-then-write with a single Firestore atomic operation. No lock needed; no partial state possible.

**CHALLENGE** Stale cache after logout on shared device

**SOLUTION ✓** Added `cache.clearAll()` to the logout flow before `signOut()` resolves. Next user on the same browser gets a cold cache.

**CHALLENGE** Empty-state problem for new users — a two-sided platform with no users delivers zero value. The swipe deck, chat list, and match feed are all empty on first load.

**SOLUTION ✓** Built `seed.js`: an idempotent script that fires on first load and — if absent — writes 12 realistic demo personas, pre-existing matches, scripted chat history, and review data in a single batch. Runs exactly once per environment. Any evaluator opens the app and sees a live, inhabited product.

**CHALLENGE** Skill name normalisation for matching

**SOLUTION ✓** All tags stored and compared lowercase. `React`, `react`, and `REACT` resolve to the same node in the intersection graph.

**CHALLENGE** Battery drain from background polling

**SOLUTION ✓** Polling pauses via `document.visibilitychange` and resumes on focus. Background tabs make zero network calls.

## 20 GO-TO-MARKET: THE FIRST 100 USERS

> **The real GTM problem for Swaply is not distribution — it is simultaneity.** A developer who wants to learn design has zero value on the platform if there are no designers. A designer who wants to learn React has zero value if there are no developers. Every two-sided marketplace faces this. The question is not "where do I post?" but "how do I manufacture the illusion of liquidity before liquidity exists, and which side do I seed first?"

### The Chicken-and-Egg Problem — Specific to Swaply

Swaply's matching constraint is tighter than most two-sided marketplaces. Uber needs drivers and riders in the same city. Swaply needs users with *complementary* skill pairs — the overlap must be bilateral. 50 developers who all want to learn design produce zero matches if there are no designers. 50 designers who all want to learn React produce zero matches if there are no developers. The liquidity problem is not just about user count — it is about *skill graph density*.

**The three failure modes of a sparse skill graph**

| | |
|---|---|
| FAILURE MODE 1 | **Imbalanced sides.** 80% of early users offer dev skills, 20% offer design/music/language. Developers see no matches and churn. The platform *looks* active but delivers no value. |
| FAILURE MODE 2 | **Symmetric but sparse.** Equal numbers of developers and designers, but only 10 of each. Statistical probability of a complementary match per swipe is low. Users hit the swipe cap without a match and do not return. |
| FAILURE MODE 3 | **Density without diversity.** 500 users, all in one skill cluster (e.g. developers who want design). Matches exist within the cluster but skill transfers are shallow — everyone teaches React, nobody teaches guitar. Network effect stalls. |

### The Strategy: Constrained Launch, Not Broad Launch

The standard advice — "post everywhere at once" — produces failure mode 2. A broad launch with low total user count guarantees a sparse graph. The correct move is a **constrained launch into a single, pre-verified complementary community**, then expand from that nucleus once match rate is confirmed.

| Phase | Target | Tactic & Rationale |
|---|---|---|
| **0** | **Pre-launch seeding**<br>*Weeks 1–2* | Recruit 20–30 users *manually* before the platform is public. Split: 10–15 developers (offer React/Python, want design/music) and 10–15 non-developers (offer design/language/music, want code basics). Source: personal network, university classmates, Discord DMs. Goal: guarantee that the first public user sees at least 3–5 real matches on their first swipe session. First impressions are the only ones that matter. |
| **1** | **Developer/design nucleus**<br>*Weeks 3–6* | The dev-wants-design pair is the highest-frequency informal exchange in the research sample (P1, P4 both fit this profile). Launch into r/webdev + r/learnprogramming *and simultaneously* r/graphic_design + Dribbble forums — not sequentially. The posts tell a story, not a feature list: "I'm a dev who wanted to learn design and built a product to solve the problem." Personal narrative converts better than product descriptions in community contexts. |
| **2** | **Language exchange layer**<br>*Weeks 6–10* | Language tandems are the most established informal exchange behaviour (HelloTalk, Tandem already serve this). r/languagelearning users already understand the concept of mutual exchange — they just lack the scheduling structure. This cohort reduces the education cost to near zero and adds diversity to the skill graph, enabling cross-cluster matches (dev + language, designer + musician). |
| **3** | **University pilot**<br>*Months 2–3* | University students are time-rich, skill-diverse, and geographically concentrated. A single CS + Arts faculty partnership at one university produces a controlled complementarity cluster: CS students teach code, arts students teach design, music, photography. Retention is higher because there is a shared institutional context — matches are not strangers, they are classmates. This cohort also generates the first honest review data, which feeds the trust layer for subsequent cohorts. |
| **4** | **Build-in-public flywheel**<br>*Ongoing* | Twitter/X build-in-public threads document the product's own usage data — match rates, swap completions, real quotes (with permission). This serves two purposes simultaneously: acquisition (developer audience converts well for learning tools) and trust (showing real metrics is more credible than a landing page claim). Product Hunt launch happens at Phase 3, not Phase 1 — only after real retention data exists to put in the launch post. |

### How seed.js solves the cold-start problem for evaluators — but not for real users

`seed.js` manufactures density for portfolio evaluation — it makes the product feel inhabited so an evaluator can assess the core loop without waiting for real users. But it is not a GTM strategy. Real users who create accounts and see only seeded personas will notice quickly; the conversation style will feel scripted, the profiles generic.

The honest answer to "how do you solve cold start for real users" is: **you do not automate it at launch — you do it manually**. Airbnb's founders photographed apartments themselves. Reddit's founders created fake accounts to seed conversations. The manual phase is not a shortcut — it is the only way to control match quality tightly enough to ensure the first cohort of real users has a positive first experience. The platform earns the right to scale only after that first cohort produces real reviews.

> **The metric that tells you Phase 1 worked:** Match-to-chat rate in the first cohort. If it exceeds 40%, the skill graph is dense enough to expand. If it is below 25%, the cohort composition is wrong — too many users on the same side. The number tells you whether to run Phase 2 or go back to Phase 0 and rebalance.

## 21 FEATURE PRIORITIZATION & ROADMAP

**Framework:** ICE scoring — **I**mpact on the North Star Metric (Completed Swaps/Week), **C**onfidence based on user research signal, **E**ase of implementation given the current stack. Each axis scored 1–10; ICE = average. Features with ICE $\geq 7$ ship first.

**Scoring bias disclosure.** All scores were assigned by a single person (me) — the same person who built the features, conducted the research, and has a vested interest in the product's success. Single-person ICE scoring has three inherent biases I tried to mitigate:

**(1) Builder bias on Ease.** Features I had already designed mentally felt easier to score highly. Mitigation: I scored Ease last, after Impact and Confidence, and forced myself to write a concrete time estimate before assigning the number (e.g. "video calls = 3–4 weeks minimum" → score 6, not 8).

**(2) Recency bias on Confidence.** Features tied to the most recent user research conversations received inflated Confidence scores. Mitigation: I re-read all 6 research notes before scoring and noted which features had *direct* participant quotes vs. which were inferred from behaviour. Direct quote = higher Confidence; inferred = penalised by 1–2 points.

**(3) Attachment to shipped features.** Scored features I had already built faced anchoring pressure toward higher Impact. Mitigation: I scored unbuilt features first to establish a baseline, then scored shipped features against that reference point rather than in isolation.

In a team context this process would involve at least one other person (engineering lead for Ease calibration, a second researcher for Confidence). The scores here should be read as directionally correct, not precisely calibrated.

| Feature | Impact | Confide | Ease | ICE | Rationale |
|---|---|---|---|---|---|
| **Synonym normalisation** *"JS" = "JavaScript"* | 9 | 9 | 8 | **8.7** | Directly reduces false non-matches. High confidence from Decision 6 research. Low build cost: AI normalisation on tag save. |
| **Skill verification badges** *Link GitHub, Dribbble, certs* | 8 | 8 | 7 | **7.7** | P5's insight: trust is the conversion blocker. Verified badges reduce the cold-outreach anxiety that kills match-to-chat rate. |
| **In-app video calls** *Remove external Zoom dependency* | 8 | 7 | 6 | **7.0** | P6 research: coordination overhead is the activation energy cost. Removing the "let's move to Zoom" step reduces session drop-off. Medium build effort (WebRTC or Daily.co SDK). |
| **Async video lesson clips** *Session preview recording* | 7 | 6 | 6 | **6.3** | Reduces "swipe anxiety" by letting users sample before committing. Medium confidence — not directly validated. |
| **Group swaps / cohorts** *Three-way exchanges* | 7 | 5 | 4 | **5.3** | High potential for retention and skill-path completion but requires significant matching-algorithm changes. v1.2 candidate once core loop data is available. |
| **Native iOS & Android apps** *Push notifications, App Store* | 8 | 8 | 3 | **6.3** | Push notifications would materially improve D7 retention (missed-message churn). Deprioritised because the PWA already satisfies the core use case; rebuilding in React Native is a significant investment with low incremental UX gain at this stage. |
| **Institutional white-label** *Universities, corporate* | 6 | 5 | 5 | **5.3** | High revenue ceiling but wrong sequencing: institutional sales require a track record of real user retention that does not yet exist. Revisit after v1 traction. |

**Sequencing logic:** Synonym normalisation and skill verification ship first because they address the two highest-friction points identified in user research — false non-matches and trust barriers — at relatively low engineering cost. Video calls follow once the matching quality is validated. Group swaps and native apps are deferred until the core loop produces enough retention data to justify the investment.

## 22 WHAT I SHIPPED, WHAT I CUT & WHY

**Scope discipline in practice.** This section documents the backlog as it actually existed during the build — features that were designed, partially built, or seriously considered, and the specific reasoning that removed each one from v1. A roadmap that only shows what was built is a highlights reel. This is the unedited version.

| Feature | Status | Decision & Reasoning |
|---|---|---|
| **Swipe deck + match scoring** | Shipped | Core loop. No version of Swaply exists without this. The match explanation ("You teach / They teach") was added mid-build after realising a plain card gave no signal about *why* two people were being shown to each other. Adding it took two hours and materially changed the UX. |
| **Real-time chat** | Shipped | Non-negotiable for the session-scheduling loop. The visibility-aware polling (pauses on hidden tabs) was added after profiling showed background tabs making continuous network calls — a battery and quota problem on mobile. |
| **Session proposals in chat** | Shipped | Research finding, not a nice-to-have. P3 and P6 both had matches that collapsed because there was no shared commitment artefact. The proposal card (date, time, duration, format, Accept/Decline) was the direct product response to a documented failure mode. |
| **AI icebreakers (skill-aware)** | Shipped v2 | Shipped twice. First version used generic openers — "Hey, saw your skills!" The generic version was live for approximately one day before I pulled it. It solved the blank-canvas problem technically but undermined the match specificity that is Swaply's core value. Rebuilt with skill-aware prompting. The two extra hours were not optional. |
| **Star ratings + review system** | Shipped | Shipped in v1 despite the engineering cost (atomic writes, unique constraint on reviewer + match, aggregation logic) because the user research was unambiguous: P5 only exchanged because a mutual friend vouched. Reputation is the substitute for that voucher. It is infrastructure, not a v2 feature. |
| **Freemium swipe cap (10/day)** | Shipped | The cap was set at 10 after considering 5 and 20. 5 felt punitive for genuine exploration — a user could hit the cap without finding a single match worth pursuing. 20 removed the felt friction that creates the upgrade moment. 10 was the number where the cap would realistically be hit only by a user who had already found value and wanted more of it. |
| **Follow / connections model** | Cut | Designed, partially spec'd, then removed before any code was written. The follow model introduced an intermediate state between mutual match and first message with no clear user benefit. It optimises for social graph accumulation, not for completed sessions. The wrong mental model entirely for this product. |

| Feature | Status | Decision & Reasoning |
|---|---|---|
| **In-app video calls** | **Cut (v1)** | Seriously considered. Removing the "let's move to Zoom" friction is real — P6 specifically cited coordination overhead. Cut for v1 because the WebRTC implementation path (or Daily.co SDK integration) represented 3–4 weeks of work minimum, and the session proposal card already reduced coordination overhead to one structured message. The marginal gain did not justify the timeline cost at MVP stage. Highest-priority v1.1 item. |
| **Calendar sync (Google / Apple)** | **Cut (v1)** | Cut for the same reason as video calls, plus an additional one: OAuth scope creep. Requesting calendar access introduces a permissions dialog that many users will decline, potentially increasing drop-off at onboarding. Manual copy-paste of session details is the v1 acceptable friction. Would revisit after session completion data confirms the use case is real. |
| **Skill taxonomy / category tree** | **Cut (deliberate)** | Considered at the start. Cut because a fixed taxonomy requires ongoing curation (someone decides which skills exist and at what granularity) and silently excludes the long tail. Lowercase normalisation gives the matching benefit of a controlled vocabulary without the maintenance cost. The known limitation ("JS" ≠ "JavaScript") is a v1.1 problem, not a v1.0 blocker. |
| **Async video lesson clips** | **Cut (v1)** | The idea: short video previews on the swipe card so users can sample a teacher's style before committing to a session. Cut because it introduces a creation burden on the user side (most people will not record a 60-second clip unprompted), and it conflates Swaply's positioning with a content platform. The value of Swaply is the exchange, not the content. |
| **Push notifications** | **Deferred** | The PWA manifest supports push, but the notification infrastructure (FCM, permission UX, notification categories) was deprioritised. The visibility-aware polling handles the active-use case adequately. The gap is dormant users — a match message arriving while the app is closed will not trigger a re-engagement nudge. D7 retention data post-launch would determine whether this is a critical gap or an acceptable one. |
| **Group swaps / three-way exchanges** | **Deferred** | The matching algorithm for bilateral exchange is already non-trivial. Three-way matching (A teaches B, B teaches C, C teaches A) multiplies the constraint space significantly. Deferred until the two-sided loop produces enough retention and quality data to confirm the core model works before adding complexity. |

**The pattern in the cuts:** Every item marked "Cut (v1)" was removed for one of three reasons: (1) the engineering cost exceeded the marginal UX gain at MVP stage, (2) it introduced a third-party dependency with its own failure modes, or (3) it optimised for a product archetype (social graph, content platform) that is fundamentally different from what Swaply is trying to be. The items marked "Deferred" are not lower-priority — they are blocked on validation data that does not yet exist.

**If I had 6 more months,** the build order would be: video calls (highest user research signal, clearest path to session completion), then push notifications (D7 retention gap), then synonym normalisation (matching quality), then calendar export (low-friction scheduling). Group swaps and institutional white-label are Q3 problems, not Q1 ones.

## 23 QUICK START — RUN LOCALLY

| Tool | Requirement | Check |
|---|---|---|
| Node.js | >= 18 | `node --version` |
| npm | >= 9 | `npm --version` |
| Firebase project | Spark / free tier | — |
| Groq API key | Free at console.groq.com | — |

### 5-Step Setup

```
git clone https://github.com/your-username/swaply.git && cd swaply
npm install
# copy .env.example → .env and fill in Firebase + Groq credentials
npm run dev  # open http://localhost:5173
```

– On first load `seed.js` automatically writes 12 demo users, pre-existing matches, chat history, and reviews — no fixture imports, no manual setup.

Deployment: Push to GitHub → Vercel builds & deploys automatically in ~2 minutes. `vercel.json` handles SPA routing and security headers. Firestore rules and indexes are version-controlled and deployed atomically via `firebase deploy`.

**Swaply · Swap your skills. Learn for free.**

React · Firebase · Groq / LLaMA 3 · Vercel · PWA · MIT License · 2025

swaply.vercel.app