EXAMPLE REPORT

Not a real client. FocusFlow is a fictional product used to illustrate what a PreThrow review actually looks like in your hands. The client name, answers, and recommendations are fabricated. Real PreThrow reviews look like this but are written for specific, paying clients.

FocusFlow

Mobile app for adults with ADHD · fictional product, example report

VERDICT

Resolve first

ENGAGEMENT

PreThrow · $750 flat

PREPARED FOR

FocusFlow *(fictional)*

STATUS

Confidential

AT A GLANCE · EXAMPLE

Summary in one page

If you only read one page of this report, read this one. The pages that follow show the evidence and the work. If you read one more thing, jump to the Recommendation.

REMINDER

FocusFlow is a fictional product used to demonstrate the PreThrow deliverable. The verdict and recommendations below are illustrative. Your real report would use your own product, your own intake answers, and your own decision points.

VERDICT

Resolve first

Becomes "Simplify then build" after two quick decisions.

FocusFlow is buildable. Two questions need answers first. Each one is roughly a week of work, mostly looking at vendor demos plus one short founder conversation. Skip them and you'll pay a developer to make them for you, usually the safe way, and that caution shows up in the bill.

PreThrow is a decision document — not a build plan, vendor pick, or budget estimate. Scope in §9.

TOP THREE THINGS DRIVING COST AND RISK

1. **How you handle user data.** Doing it yourself in a database is a much bigger lift than letting a sign-in service handle it. (Especially since you said European users are in scope.)
2. **Two platforms (iOS + Android).** Either pick one to start, or use a framework that handles both at once. This is the call that moves the budget most.
3. **Designing for payments now, even if launch is free.** Bolting payments on later commonly costs several times more than designing for them up front.

TWO DECISIONS TO MAKE BEFORE HIRING A DEVELOPER

1. **Pick a sign-in service.** An hour each with Auth0, Clerk, and Supabase. Whichever feels easiest. Most of your privacy obligation goes with that choice.
2. **Commit that paid is coming.** You don't have to launch with payments. You do have to commit they're part of the plan, so the developer designs for them now.

RECOMMENDED NEXT STEP

Make the two decisions above (about two weeks). Then either book a free 30-min walkthrough of this report, or move on to a Discovery engagement that produces a real build plan and budget tailored to your decisions.

Why I'm not putting a dollar figure on the build here. The two open decisions change the budget a lot — quoting a number now would be misleading. A Discovery engagement, scoped separately, produces a real number once those decisions are locked in.

The pages that follow show how I got here →

SECTION 01

Executive Summary

FocusFlow is a real, well-shaped product idea. The user research is solid (12 ADHD-adult interviews). What's not solved yet is two specific decisions about how the product gets built. Those decisions are easy to make in a couple of weeks. Skipping them and going straight to a developer is the expensive path.

The recommendation is **Resolve first**. Not because FocusFlow is a bad idea. Because two questions need answers before you start writing checks.

Question one: who handles user data? If you store names and emails in your own database, you take on real European privacy obligations (because you said you'll launch in Europe). If you use a sign-in service like Auth0 or Clerk, they handle most of it for you and the obligation shrinks dramatically. This is a vendor pick. A couple of demos and you're done.

Question two: when do you turn on paid subscriptions? You marked the revenue model as "freemium" but said no payments in v1. That's fine. But the paid tier needs to be designed in from day one, even if it ships later. Bolting payments on six months in commonly costs several times more than building for them up front. So you don't need to know exactly when paid launches. You need to commit that paid is coming.

Answer both and "Resolve first" becomes "Simplify then build" with a realistic budget you can hand to a vendor. About two weeks of research plus one decision conversation.

What's driving cost and risk, top to bottom

- **Two platforms (iOS + Android).** Each one is a separate build, separate testing, separate app-store reviews. Most apps either pick one to start, or use a framework (React Native, Flutter) that handles both at once. This is the call that moves the budget most.
- **Privacy posture (user data + EU users).** Buildable either way. The choice in "question one" above is what decides whether this is a quick policy doc or a few months of legal and engineering work.
- **Sign-in / accounts.** Solvable. The question is build it yourself or use a service. Use a service.
- **Freemium with no payments yet.** The trap is "we'll add payments later." Even if you ship free, design as if paid is coming. Cheaper to do up front than to come back to.

SECTION 02

What Is Being Proposed

Neutral restatement of the product as captured in the intake. This is what the report evaluates; if anything here is wrong, please correct it before we proceed to the walkthrough call.

- **Core product:** FocusFlow, a mobile app for adults with ADHD. Helps users break tasks into 5-minute "sprint blocks" with completion badges, streak tracking, and adaptive break suggestions. Includes habit-style daily check-ins and a weekly "wins recap" email summary.
- **Platforms:** iOS and Android (cross-platform from launch).
- **Authentication:** Required. User accounts.
- **Offline support:** Listed as nice-to-have rather than required.
- **Integrations:** Apple HealthKit for screen-time data (optional, user opt-in). No payment integration in v1 (free during beta).
- **Sensitive data:** Personally identifiable information (name, email).
- **Scale expectation:** 1,000-10,000 users in the first year.
- **Reliability target:** Customer-facing — must work reliably.
- **Revenue model:** Freemium (free with paid pro tier; pro tier not in v1 scope).
- **Geographic reach:** North America + Europe (GDPR applies).
- **Update cadence:** Frequent — monthly or more.

- **Timeline expectation:** 3-6 months to launch.

SECTION 03

Feasibility Assessment

Each area is rated *Feasible*, *Feasible with risk*, *High complexity*, or *Not recommended*. The rating reflects implementation difficulty for a competent solo developer or small team, not whether the area should be in the product at all.

AREA	ASSESSMENT	HEADLINE
Core functionality	FEASIBLE	Sprint blocks + badges + streaks are well-understood patterns.
Platform scope	HIGH COMPLEXITY	Cross-platform doubles testing and store-review surface.
Data integrity	FEASIBLE WITH RISK	Auth + nice-to-have offline = decisions to lock down early.
Authentication	FEASIBLE	Use a vendored identity provider; don't build your own.
Compliance	HIGH COMPLEXITY	PII + EU users = GDPR work. Solvable; needs a chosen path.
Operations	FEASIBLE WITH RISK	External dep (HealthKit) + freemium + frequent updates require CI/CD.

> **Total complexity score: 16 points** (out of an internal max of ~30). **Interactions detected: 2.** "Multi-platform + frequent updates" and "Auth + compliance" each compound risk beyond the sum of their parts.

SECTION 04

Primary Cost & Complexity Drivers

The following items materially increase development and maintenance cost. They are not red flags; they are the places where a smart vendor will charge fairly and a less-smart vendor will under-quote and then bill change orders.

A few directional ranges appear below. They're industry rough-orders for sizing decisions, not your specific build estimate — your vendor and scope can pull the number either way.

Platform Scope

- **Multi-platform (Q4).** iOS + Android from launch. Materially adds to the build cost vs. iOS-only. Cross-platform frameworks (React Native, Flutter) commonly cut the duplicate-implementation effort by roughly half compared to building each platform separately. (Industry pattern, not a guarantee.)

Data Integrity

- **Auth required (Q5).** Session management, password reset, account recovery, security surface. Vendored solutions (Auth0, Clerk, Supabase) take a few days of integration; rolling your own takes weeks of build plus permanent ongoing security work.
- **Offline support — nice-to-have (Q6).** Flagged as optional. If pursued later, it would add sync logic and conflict resolution; if scoped out at launch, this risk drops materially. Worth pricing both paths in your build SOW.

Compliance & Security

- **Compliance required (Q9).** PII handling. Architecture choice (self-handle vs. vendor-delegate) determines the size of this work.
- **Multi-region (Q14).** Europe-included means GDPR. At your scale, off-the-shelf privacy generators (Termly, iubenda) cover ~90% of what's needed; the remaining 10% is a deletion-on-request flow and a clear consent gate.

Ongoing Operations

- **External dependencies (Q8).** Apple HealthKit. Stable API but requires physical-device testing.
- **High reliability (Q12).** Customer-facing means monitoring + crash reporting from day 1 (Sentry, Datadog).
- **Payments / freemium (Q13).** Even if v1 ships free, design the entitlement architecture in. Retrofitting payments mid-build commonly costs several times more than designing them in.
- **Frequent updates (Q16).** Monthly+ release cadence requires CI/CD, automated testing, and capacity for hotfixes.

Where the cost drivers compound (worse than the sum):

- **Two platforms + frequent updates** — each release passes both app-store review queues. A regression in one can block a hotfix in the other.
- **User accounts + privacy obligation** — your login system becomes part of what an auditor would care about, not just a login screen.

Note on this section in the sample: in a real engagement, this section also includes a per-driver cost-range read tied to your specific intake, comparable-app benchmarks where relevant, and a few mitigation notes for the riskiest drivers. Kept lighter here to fit the sample.

SECTION 05

Risk Assessment

Technical Risk

Biggest technical risk is the combo of two platforms plus a frequent release cycle. Every release means passing both Apple and Google review queues. Their policies sometimes diverge, so a regression in Android can block an iOS hotfix. A cross-platform framework (React Native, Flutter) cuts this down a lot in exchange for a little less native polish.

Offline is the quiet one. You marked it "nice-to-have," which I'd take seriously. Nice-to-have offline tends to become required offline a few months in, when users on planes and subways complain. A vendor who designs offline in from day one charges meaningfully more than one who doesn't. The one who didn't will charge you the difference plus a premium as a "phase 2" later. Decide which path you want before you sign anything.

Cost Risk

Two specific cost risks to think about before picking a vendor:

- **Sign-in: build or buy.** Buying (Auth0, Clerk, Supabase) is a modest monthly fee and a few days of setup. Building your own is "free" in fees but takes weeks to build and never stops needing security attention. At your stage, buy.
- **Payments later.** If payments get bolted on mid-build instead of designed in from day one, expect the billing work to cost meaningfully more — commonly 2-3x what it would have cost

to design in up front. The hard part isn't the integration itself. It's reworking how the app thinks about who has paid for what, which is hard to retrofit. Cheaper to plan for now.

Operational Risk

Apple HealthKit is well-documented and stable, but a few features only work on real iOS hardware, not the simulator. Plan to keep a test device or two on hand.

Frequent updates plus a customer-facing product means you need automated build and release tools from day one. Not "I'll archive in Xcode when I have time." Without that, your ability to ship a hotfix when a user reports a bug degrades pretty quickly.

Legal & Privacy Risk

The "user data + EU users" combination is the main legal question. Two paths:

- **Handle data yourself.** You become "the data controller" under GDPR — privacy policy, deletion-on-request, breach reporting (72-hour deadline), legal review. Material annual cost.
- **Delegate to a sign-in service.** Auth0, Clerk, Supabase hold the user data; your app just remembers an ID. Day-to-day work shrinks dramatically. Recommended for FocusFlow at this stage.

One note on Apple Health: it sounds like health data, but Apple keeps it on the device unless you explicitly send it to your servers. If you don't, HIPAA doesn't apply.

SECTION 06

Maintenance & Long-Term Reality

This product should be assumed to require ongoing maintenance, not a one-time build. The intake's "frequently" update cadence is honest about that.

> *The maintenance breakdown in this sample is intentionally abbreviated. In a real engagement, this section names the specific ongoing burdens for your product — app-store review cadence, third-party API churn, compliance posture upkeep, and product-specific gotchas — with a realistic annual maintenance percentage based on apps of similar shape.*

Annual maintenance, all-in, commonly runs to **15-25% of original build cost per year**. (Long-standing industry rule of thumb for actively-developed consumer mobile apps; your specific number depends on update cadence, regulatory load, and how many third-party

dependencies you carry.) For FocusFlow's shape — auth, planned payments, one third-party integration, monthly+ release cadence — expect the upper half of that range.

SECTION 07

Recommendation

> **Verdict: Resolve first.** Don't start the build until two questions are answered. Roughly two weeks of work, mostly vendor demos and one conversation. Out-of-pocket cost to resolve: near zero.

Two things to do before signing anything with a developer

1. **Pick a sign-in service.** Spend an hour each looking at Auth0, Clerk, and Supabase Auth. Pick whichever feels easiest. That choice handles most of your user-data obligation and removes a fork from the build conversation.
2. **Decide payments are coming.** You don't have to launch with payments. You do have to commit that paid is part of the plan, so the developer designs for it. Building it later from scratch is the expensive path.

Both decisions made, the verdict moves from "Resolve first" to "Simplify then build" — and you'll have a real budget to take to vendors.

If you'd rather just start building anyway

You can. FocusFlow is buildable. The cost and timeline will run higher because the developer will have to make these calls for you, usually the safe way, and you pay for that caution. Not a disaster. Just not the cheapest path.

Two open questions for the walkthrough call

Two things worth pinning down on the call. Pre-deciding either is fine too.

COMPLIANCE SCOPE

Does the sensitive data flow through your system, or only through a vetted third party (e.g., Stripe for cards, Auth0 for identity)?

Why it matters: Outsourcing the compliance surface materially changes feasibility. The intake does not distinguish between "we handle PII" and "we delegate to a SOC2 vendor."

OFFLINE SCOPE

Is offline "nice if it falls out of the architecture" or "we will spend time on it later"? The cost difference is substantial.

Why it matters: Nice-to-have offline often becomes load-bearing post-launch when users complain. Worth pricing both paths.

SECTION 08

Next-Step Guidance

When you go to hire a developer, here's what to look for. You don't need to know any of these by heart — share this section with them and ask if they're comfortable with all of it.

1. **Cross-platform mobile experience.** Either React Native, Flutter, or separate iOS+Android teams. Pick based on who's actually available in your network, not which framework sounds best on a sales call.
2. **Comfort using a sign-in service.** Auth0, Clerk, or Supabase Auth. If they push back and want to build their own login system from scratch, that's a flag for an app your size.
3. **Designing for payments from day one.** Even if launch is free, the app should be structured so that turning paid on later is a small change, not a months-long rebuild.
4. **Apple HealthKit.** Well-documented but a couple of features only work on real iOS devices. Make sure they have one.
5. **European privacy basics.** A privacy policy, a way for users to request deletion, and clear consent before collecting anything. You don't need a specialized lawyer at this scale — services like Termly or iubenda handle most of it.
6. **Automated build and release.** Because you'll be updating often, manual releases will fall apart fast. Make sure they set this up early, not as an afterthought.

What you should NOT pay a developer to do

- **Build a custom backend from scratch.** Use Supabase or Firebase. They're cheaper, faster, and what most apps your size run on.
- **Build a custom marketing website.** Use Webflow or Framer. A developer is the wrong tool for that.
- **Redesign your screens.** Your Figma is fine for v1 once user testing wraps. Don't pay for a polish pass yet.

This review is a decision document, not a vendor recommendation or a build plan. If you'd like help with either, that's a separate conversation.

SECTION 09

Engagement Boundary

This review provides a decision framework, not implementation support. Any follow-up analysis would be scoped separately.

What this PreThrow engagement included:

- 17-question intake review
- Written feasibility report (this document)
- Optional 30-minute walkthrough call to discuss findings

What this engagement did not include:

- Wireframes, architecture diagrams, or technical implementation
- Vendor selection or developer recommendations beyond category-level guidance
- Ongoing advisory or coaching
- Build SOW drafting

If you'd like to engage further: a follow-on Discovery engagement (typically 2-3 weeks, $1,800-$2,500) would produce a build SOW including tech-stack selection, architecture sketch, MVP scope definition, compliance-posture decision, and a real build cost estimate. That conversation happens only at your request, after this report's recommendations have been considered.

A few clean boundaries on what this is

This report is Throw It In Main LLC's professional opinion based on the answers you provided. It's decision-support information, not legal, financial, or regulatory advice. You're responsible for verifying that specific regulations, vendors, and recommendations fit your situation. References to outside services (Auth0, Supabase, Termly, etc.) are informational; they're not endorsements and we don't get a kickback from any of them.

Thank you for trusting Throw It In Main with this review. The intent of PreThrow is to save you more money than the $750 it cost. If this report achieved that — even by recommending you not build — please pass our name along to one person who needs the same clarity.

SECTION 10

Sources

The specific claims in this report are grounded in public, citable sources. Each one was selected because the relevant claim is at the cited link, not merely adjacent to it. If you have time for nothing else from this report, the links below let you verify the load-bearing facts independently.

Compliance & Privacy

1. **GDPR Article 33 — Notification of personal data breach.** 72-hour breach-notification rule + documentation duties for data controllers. Backs the §5 Legal/Compliance Risk discussion of EU-user obligations. gdpr-info.eu/art-33-gdpr
2. **COPPA — Children's Online Privacy Protection Rule (FTC).** Verifiable parental consent required before collecting personal information from children under 13. Backs the Kids-Category complexity discussion in §4. ftc.gov/legal-library/browse/rules/childrens-online-privacy-protection-rule-coppa
3. **COPPA Rule — eCFR 16 CFR Part 312.** The full authoritative regulation text. The 2025 amendments (effective compliance deadline April 22, 2026) are worth a read if your product touches anyone under 13. ecfr.gov/current/title-16/chapter-I/subchapter-C/part-312

Apple App Store

4. **Apple Developer — Design safe and age-appropriate experiences (Kids Category).** Constraints on PII handling, third-party analytics, third-party ads, and parental gates for apps in the Kids Category. Material to any literacy/learning app aimed at minors. developer.apple.com/kids
5. **Apple App Store Review Guidelines.** Authoritative guidelines for App Store submission, including kids-category rules, content moderation, and privacy requirements. Read once before vendor selection. developer.apple.com/app-store/review/guidelines
6. **Apple Developer — Updated age ratings in App Store Connect.** New age rating system effective January 31, 2026 adds 13+, 16+, and 18+ ratings; existing apps must update their

responses to avoid submission interruption. Worth scheduling a 30-min review for any app currently live. developer.apple.com/news

7. **Runway — Live App Store and TestFlight review times.** Crowdsourced live tracker of current App Store and TestFlight review times. Useful for realistic hotfix-cycle planning; review times have been trending up in 2026 (24-72 hours typical, 5-7 days for new submissions). runway.team/appreviewtimes

Identity / Auth vendors

8. **Auth0 — Pricing & plans.** Reference pricing for vendored authentication; the free tier covers small consumer apps. auth0.com/pricing
9. **Clerk — Pricing.** An alternative vendored identity provider with strong React/Next.js integration. clerk.com/pricing
10. **Supabase — Authentication.** Open-source backend that includes auth + database + storage; common choice for indie builds where consolidating vendors matters. supabase.com/auth

Architecture background

11. **CRDTs — A primer (Martin Kleppmann).** Conflict-free replicated data types are the architectural foundation for serious offline-capable apps with sync. Background reading for understanding why "offline as a first-class feature" is materially more expensive than feature parity would suggest. youtube.com (45-min talk)

> **A note on what's cited here.** The links above are only ones that back something I actually said earlier in the report. Padding a sources section with extra "related reading" looks thorough but isn't. If you click a link and what I claimed isn't there, that's on me — flag it on the walkthrough call and I'll fix it.