PRODUCT CASE STUDY

Return the Favor

Product thinking, scoping, and trade-off decisions on a solo-built native iOS app. Shipped in three weeks by directing AI coding tools.

AUTHOR	Nicholas White
ROLE	Founder & Product Owner
PRODUCT	Return the Favor · Native iOS application
SHIPPED	April 23, 2026 · Live on the App Store
APP STORE	apps.apple.com/us/app/return-the-favor
LIVE SITE	rtf-case-study.vercel.app
DOCUMENT	Product case study · Prepared April 2026

The problem

Personal gift-giving is a recurring, high-frequency life event that most people manage badly. Birthdays, anniversaries, weddings, baby showers, and holidays arrive on a predictable schedule, yet they consistently produce the same friction: forgotten dates, inconsistent budgets, duplicate gifts, and the quiet social debt of not remembering what someone gave you a year ago. The cost is rarely catastrophic in any single instance, which is why it goes unsolved. It compounds quietly across a year and across a relationship.

The users who feel this most acutely are already attempting to solve it. They keep spreadsheets of gifts given. They scribble notes in the Notes app after every birthday party. They set calendar reminders that fire too late to be useful. They are not looking for a novel category of software; they are looking for one place to consolidate a workflow they are currently running across three or four apps.

Existing tools fail this user in a specific way. Calendars track dates without context. Notes apps store context without structure. Budgeting apps track spending without relationships. Generic gift-tracker apps solve a single problem and require the user to maintain the surrounding issues themselves. The result is a fractured system where the user does more administrative work than the tool saves.

Return the Favor was built on a specific bet: that the answer to this problem is not a better calendar, a better notes app, or a better budget tracker, but a single workspace that treats people, dates, gifts, and budget as one connected object. The product's job is to remove administrative work, not add a new surface for it. Every decision that follows was tested against it.

Initial vision and early build

The original v1 spec was deliberately small. Three tabs (Birthdays, Events, Holidays), each opening with a countdown to the next item. Budget tracking was a single dollar value attached to each gift. The product was meant to feel like a focused utility, not a platform.

That spec changed materially within the first week. The largest change came from a question asked by a non-technical user on day five of the build:

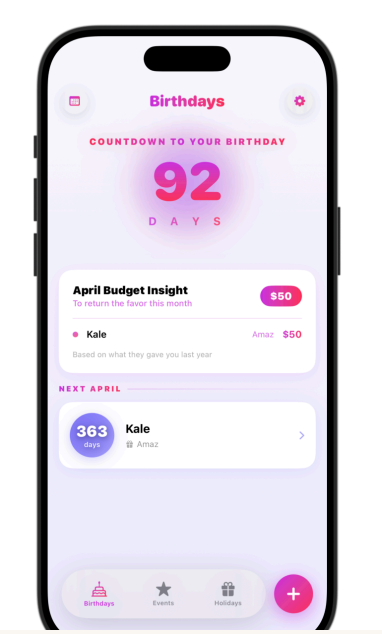
> *“So where can I see everything?”*

The original navigation model assumed users would move between the three tabs based on what they were trying to remember. She did not think that way. She wanted a single landing surface that showed her the next thing happening, the budget pressure for the current month, and the people involved, without making her choose a tab first. The Home tab was designed and built that night. Adding it meant merging Birthdays and Events into one conceptual tab to keep the navigation count to three. That conversation reshaped the information architecture of the entire app.

Two other early decisions are worth a brief note. The Anniversary tab was renamed to Events when it became clear that weddings, baby showers, and work milestones would not fit under a narrower label. Two features were cut outright before launch: a share button that depended on an App Store link that did not yet exist, and a Reset All Data button that conflicted with the way iCloud sync was repushing deleted records.

From day five to App Store

The screenshot to the right shows the first working version of the Home tab, built the night of the conversation that prompted it. Day five of a twenty-one day build.

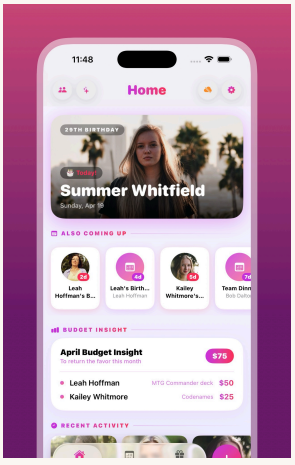


Home tab, day 5

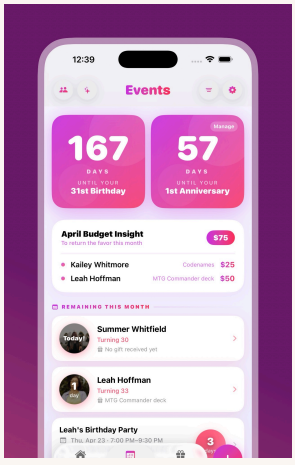
The version shown here shipped largely intact. The architectural decision to make the Home tab the primary landing surface did not change after this night.

By the end of the build, the feature set had grown beyond the original spec but stayed inside the original thesis. Each addition was justified against whether it removed administrative work for the user.

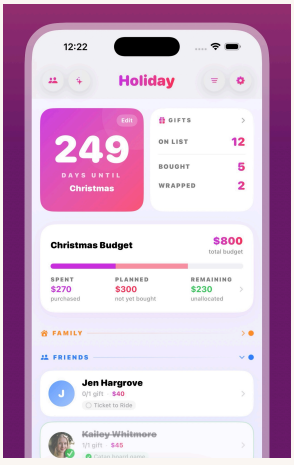
Three weeks later, this is what shipped to the App Store.



Home



Events



Holiday

Key product decisions

Four decisions in the v1 build are worth examining in detail. Each represents a different category of product judgment: a values decision made before any code was written, a platform-constraint decision driven by the operating system, an architectural decision made under time pressure, and a scope decision that absorbed real cost to protect a quality bar.

Accessibility as v1 scope, not a 1.1 nice-to-have

Accessibility was a hard scope requirement for v1: full VoiceOver labels on every interactive element, Dynamic Type across all text styles, Reduce Motion support, ADA and EU Accessibility Act compliance. Retrofitting accessibility into a SwiftUI app after the fact is significantly harder than building it in from the start, and every custom component, animation, and layout decision had to account for it from day one. The lesson learned the hard way: accessibility is a foundation, not a coat of paint.

Capping advance reminders at 14 days

iOS allows a maximum of 64 scheduled local notifications per app. With dozens of birthdays, anniversaries, and holidays in a typical user's data, allowing 90-day advance reminders would silently exceed the system budget. Reminders would be dropped without warning, and the app would fail at its core promise. The decision was to cap user-configurable advance reminders at 14 days and structure the rest of the notification budget around a daily digest. Users cannot set a 30-day reminder for a wedding. In return, every reminder the app promises is a reminder the system actually delivers.

Key product decisions

Privacy by design, with no servers

Built as CloudKit-only from the first commit. No backend server, no user accounts, no analytics, no tracking, no advertising. User data lives in their own iCloud and never touches a third party. The cost: no cross-platform support, no usage analytics to inform the roadmap, no admin dashboard to debug user issues. The benefit: the app cannot leak data it does not have.

Past-date handling, fixed by spending half a day on invisible work

Late in the build, a bug surfaced where the app was silently rewriting the dates users had entered. A birthday saved as 2023 would, after a sync, show as 2027. The cause was an automatic cleanup routine that moved past events out of the active list by overwriting their dates instead of just flagging them as archived. The shortcut fix was to patch that one routine. The right fix was to lock event dates so that no part of the app could ever change them after they were entered, and to track archive status in a separate field. Half a day of refactoring across seventeen places in the code, no visible feature change, the entire class of bug gone for good. I took the expensive fix. A memory app that silently rewrites the user’s memories has nothing left to sell.

How it got built

I built Return the Favor with AI coding agents because that was the team I could afford. The product manager's job did not change because of it. Define what good looks like. Break work into reviewable units. Catch when execution drifts from intent. Make the call when something needs to be cut, redesigned, or pushed. The same skills apply whether the implementation is done by people, by AI, or by a mix of both.

AI agents are strong at boilerplate and consistent patterns across many files. They struggle with subtle architectural conflicts, platform edge cases, and debugging that requires holding more than one system in mind. My role was to provide architectural intent, define acceptance criteria, review what was produced, and intervene when an agent could not solve a problem.

A specific example. One debugging session involved a Liquid Glass effect on the iCloud backup page. The agent kept producing code that compiled but rendered incorrectly. After several attempts returned variations of the same broken output, I read through the code myself and traced the call chain. The rendering relied on an API that only existed in iOS 18 and later. The project was set to a deployment target of iOS 17. The agent had no way to know this without being told. Once I pointed at the version mismatch, the fix was straightforward: raise the deployment target, restrict the user base slightly, keep the product looking the way the design called for.

That kind of intervention is what leading any team looks like. The next one I lead might be human, AI, or both. The work is the same.

Launch and what's next

Return the Favor shipped to the App Store on April 23, 2026, on the original 21-day timeline. The launch outcomes I can speak to are process outcomes: full WCAG and EU Accessibility Act compliance at v1, zero P0 bugs in the first week post-launch, and a 1.1 release candidate already scoped from real-user feedback collected in the first 72 hours.

User metrics are intentionally limited. The product ships with no analytics, no behavioral tracking, and no telemetry. That decision is consistent with the privacy posture in Section three, and it is reinforced every time I watch a non-technical user pause before granting an app permission to their contacts or calendar. The cost is that I cannot measure install-to-first-event conversion, retention curves, or feature adoption. The benefit is that every user knows the app cannot watch them, because it cannot.

The signals I am working from instead are App Store reviews, direct feedback from real users, and crash reports from Apple. Those will tell me whether the v1 thesis is correct before I commit engineering time to the 1.2 widget release.

Roadmap

The roadmap is structured around four releases. The 1.1 release is small, focused, and close. The 1.2 release adds platform integration the v1 deliberately deferred. The 1.3 release introduces a quiet rewards layer. The 2.0 release is a longer-horizon rebuild that revisits the visual language of the entire product.

NEXT RELEASE

1.1

Returns the share button cut from v1, now that the App Store link is stable. Incorporates bug fixes from real-user feedback in the first weeks after launch.

FOLLOWING RELEASE

1.2

Adds Home Screen widget support. Widgets were deferred from v1 to keep scope tight, extending the product's core value onto the user's lock screen and home screen.

AFTER WIDGETS

1.3

A quiet badge system that rewards data entry, profile completeness, and acknowledging important dates on the day. No streaks, no levels, no leaderboards.

LONG-HORIZON RELEASE

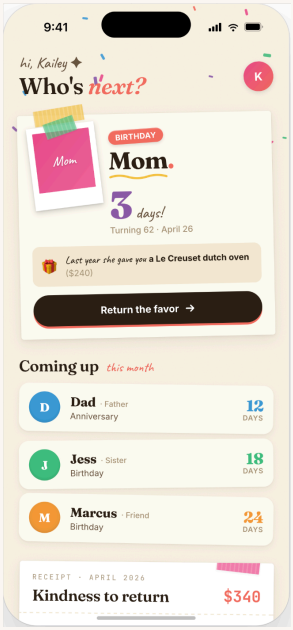
2.0 · Paper Party concept

An end-to-end visual rebuild on top of the same data foundation. The current visual language is a polished iOS utility. The 2.0 concept replaces it with an editorial, scrapbook-style aesthetic: cream paper backgrounds, mixed serif and script typography, polaroid frames, and hand-drawn accents. The data foundation does not change. The surface presented to the user does. Three early concepts shown on the next page.

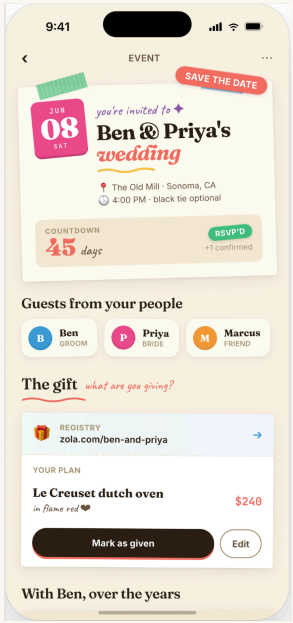
A note on the 1.3 release. The badges cannot compete with the user's actual reason for using the app. Return the Favor exists to help people show up for the people who showed up for them. The reward for that is the relationship, not the badge. The badges acknowledge the effort without ever framing the gesture as transactional.

Paper Party

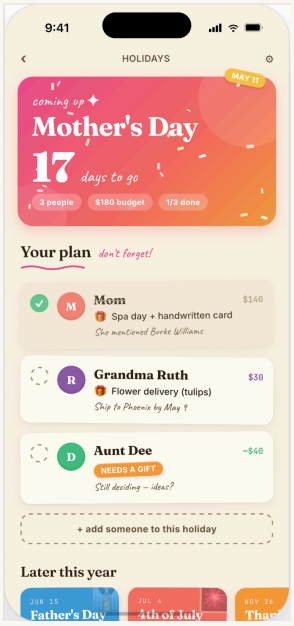
Three early concept mockups for the Home, Event Detail, and Holiday surfaces. Same data foundation. Different surface.



Home: "Who's next?"



Event as invitation



Holiday as task list

Beyond 2.0, an Android port is on the table if the product gains traction. The largest open question on Android is the loss of CloudKit as a sync layer, which is the foundation of the privacy-by-design commitment. A port would either require a different sync architecture or a different privacy contract with the user.

What I would do differently

Three things I would build into the next project from day one. None of them cost the v1 ship date, but each would have saved real time.

Listen earlier. The Home tab existed because a non-technical user asked the question I should have been asking myself. The v1 navigation was internally consistent but did not match how a real user wanted to interact with the product. The lesson is to put the product in front of users earlier and treat their confusion as data, not as a teaching moment.

Validate sync on day one. CloudKit sync issues surfaced late in the build. By the time the conflict was discovered, code had already been built around assumptions that turned out to be wrong. A two-device test setup running from the first commit would have caught the issues before they compounded.

Justify architectural choices in writing before any code gets built. AI agents propose patterns that are familiar from training data, not necessarily patterns that fit the project. Early in the build I accepted several of these without enough scrutiny, and some had to be unwound later. The habit I am taking forward: every architectural decision gets a one-paragraph written rationale before delegation.

Return the Favor exists because of a thesis: that thoughtful gift-giving is a workflow, not a memory test. The v1 build is the proof. Shipped in three weeks, accessible from day one, private by design, and built solo by directing AI agents end-to-end. The next product I lead will be larger. The discipline will be the same.