The Reverse-Engineer Playbook

Turn a proven app's bad reviews into your build plan — worked on a real meal tracker

01 Pick an App That Already Sells

Do not look for an original idea. Look for an app in your niche with strong download numbers, a paid or subscription tier, and at least a few hundred written reviews. Demand, pricing and distribution are already proven, so the only question left is what the incumbent is doing badly. For this playbook we use the category that produced a $30 million exit in two years: photo-based meal trackers.

> 💡 **QUICK TIPS**
>
> - Filter the App Store or Google Play charts by category, not by what interests you.
> - Skip apps with under ~200 reviews — there is not enough complaint data to mine.
> - Boring beats exciting. Trackers, converters and loggers have the most fixable gaps.

---

02 Pull 50–100 Negative Reviews

Open the app's store listing, filter to one- and two-star reviews, and copy 50 to 100 of them into a plain text file. A one-star review is a paying customer describing the exact feature they would switch apps for, which makes it more useful than any survey you could run. Fifty is enough to see the patterns; past a hundred you stop learning anything new.

> 💡 **QUICK TIPS**
>
> - Sort by Most Recent so you catch complaints the developer has not fixed yet.
> - Also check the app's subreddit and its support forum — the angriest users post there.
> - Keep the review dates. A complaint repeating for two years is one they will never fix.

---

03 Cluster the Complaints Into Gaps

Paste the reviews into Claude Code and have it group them into ranked complaint themes instead of reading them yourself. You are looking for the two or three themes that repeat across dozens of reviews, because those are the ones worth building against.

PASTE THIS

```
Here are 80 negative reviews for [App Name], a photo-based meal
tracking app. Group them into recurring complaint themes.

For each theme give me:
- the theme in one line
- how many reviews mention it
- 2 verbatim quotes
- what a competing app would have to do to fix it
- whether the fix is a feature, a pricing change, or an accuracy problem

Rank themes by frequency. Ignore one-off complaints and anything
about a temporary bug or outage.
```

💡 QUICK TIPS

- Ask for verbatim quotes — they become your landing page copy later.
- Ignore complaints about outages and crashes. Those get fixed; design flaws do not.

---

04 Worked Example: The Meal Tracker Gaps

Run the prompt above on any leading photo meal tracker and the same four themes come back. Portion sizes are guessed badly from a single photo, the free tier caps scans so low the app is useless, restaurant and home-cooked meals are missing from the database, and the subscription is hard to cancel. Now pick ONE and be dramatically better at it — fixing all four gets you a worse version of the incumbent.

💡 QUICK TIPS

- Accuracy gap → win with one follow-up question: 'palm-sized or plate-sized?'
- Pricing gap → win with unlimited scans at a flat low price, no scan credits.
- Database gap → win by letting users photograph a menu, not just the plate.
- Write your pick as one sentence: 'The meal tracker that gets portions right because it asks one question.'

05

Turn the Gap Into a PRD

A Product Requirements Document is one file that defines your user, the single problem you beat the incumbent on, the screens, the data model and the libraries. It matters because Claude Code builds far more reliably from a spec than from a conversation — and because naming your auth, payment, image and database providers means roughly 20% of the app already exists as packages you are wiring together.

PASTE THIS

```
Using the complaint analysis above, write a full PRD for a photo meal
tracking app that wins on portion-size accuracy.

Include:
1. Target user in one paragraph (specific, not 'health-conscious people')
2. The one problem we beat the incumbent on, and how we prove it
3. Core feature list for v1 - maximum 5 features, ruthlessly cut
4. Screen-by-screen user flow
5. Data model
6. Recommended libraries and APIs, with why for each
7. What we are deliberately NOT building in v1

Save it as PRD.md in the project root.
```

💡 **QUICK TIPS**

- Keep PRD.md in the repo root so every later session reads it and stays on spec.
- Build one screen or one feature per session, checking against PRD.md each time.
- If a feature is not in the PRD, it does not go in v1. That rule is the whole point.