PRODUCT TEARDOWN · QUICK COMMERCE · INDIA

Blinkit and the empty shelf

There is no control anywhere in this app for what should happen when a line cannot be picked, and that absence is the most interesting thing in it.

Independent teardown, written from the public storefront and the published terms. No affiliation with the company and no internal data.

12 min read

---

backpocket.website/studies

IN SHORT

- Blinkit promises groceries in about ten minutes from a small neighbourhood store. That promise prices everything under it: floor space, lines carried, rider radius, and what the system does when the shelf and the database disagree.
- What I verified is an absence. Nowhere in the flow can I say what I would want if a line cannot be picked, and nowhere during an order does the app come back to ask. What the platform then does on a miss is reasoned from its own published refund terms, not from having watched it, and it is marked as inferred wherever it appears.
- The preference has to exist before the interrupt does. A live prompt shipped first, with no stated default to expire into, is an expensive notification that changes nothing, which is why the cheap build goes first.
- I would not manage this against average delivery time. I would manage it against intact orders delivered on promise, keep unserved demand beside that number, and name what the pair still cannot see.

Ten minutes is a ceiling, and it prices everything below it

Blinkit sells groceries and everyday household goods in Indian cities, delivered from a small neighbourhood store in about ten minutes. Open the app and it asks for a location before it shows a single product. There is no catalogue in the ordinary sense. There is only what the nearest store is holding right now.

Two jobs share one login. One is a top up: milk, bread, a forgotten charger, small basket, urgent, not price sensitive. The other is a weekly shop done here because ten minutes beats forty, so the basket is bigger and the brand requirements are firmer. Both are served by the same screens and the same failure policy.

The ten minutes is a ceiling and it squeezes everything below it. A small store cannot carry what a supermarket carries, and a rider on a ten minute clock cannot go far. So the product has to decide what not to sell, neighbourhood by neighbourhood, and then live with that decision in front of whoever is shopping.

What the storefront shows, and what it does not

Everything below was read off blinkit.com, which is the same catalogue and the same ordering flow in a browser. It is deliberately dull. The interesting part turned out to be which controls are absent rather than which are present.

What I have not done, and will not claim: I have not run an order through to a line failing after payment. So the post payment path in this piece is reasoned from the published cancellation and refund text rather than from having watched it happen. That is the weakest link in the argument and it is named again where it matters.

- The storefront asks for a location before it shows a single product. Nothing about the catalogue exists independently of which store is nearest.
- A delivery estimate sits at the top of the screen before the cart holds anything. It belongs to the store, not to an order.
- A product card offers exactly one action, which is to add the item. There is no per line preference anywhere on it.
- Lines that cannot be supplied are greyed out or absent from the listing. Neither state says why, or when the item might return.
- The cancellation and refund terms are published, and the window closes once an order is out for delivery. That is what the timing argument rests on.

Nowhere in the flow can I say what to do if a line is missing

The verified part is an absence. From the first tap to the confirmation screen there is no control, no toggle and no note field where I can say what should happen if a line cannot be picked. No account setting carries such a preference between orders. That is checkable by anyone in a few minutes, and it is the whole of what I know for certain.

What the platform does next I am reasoning from its own published refund text: the line comes off the order, the money comes back, and nothing in that text describes the customer being asked first. If it does ask somebody, somewhere, I have not found where.

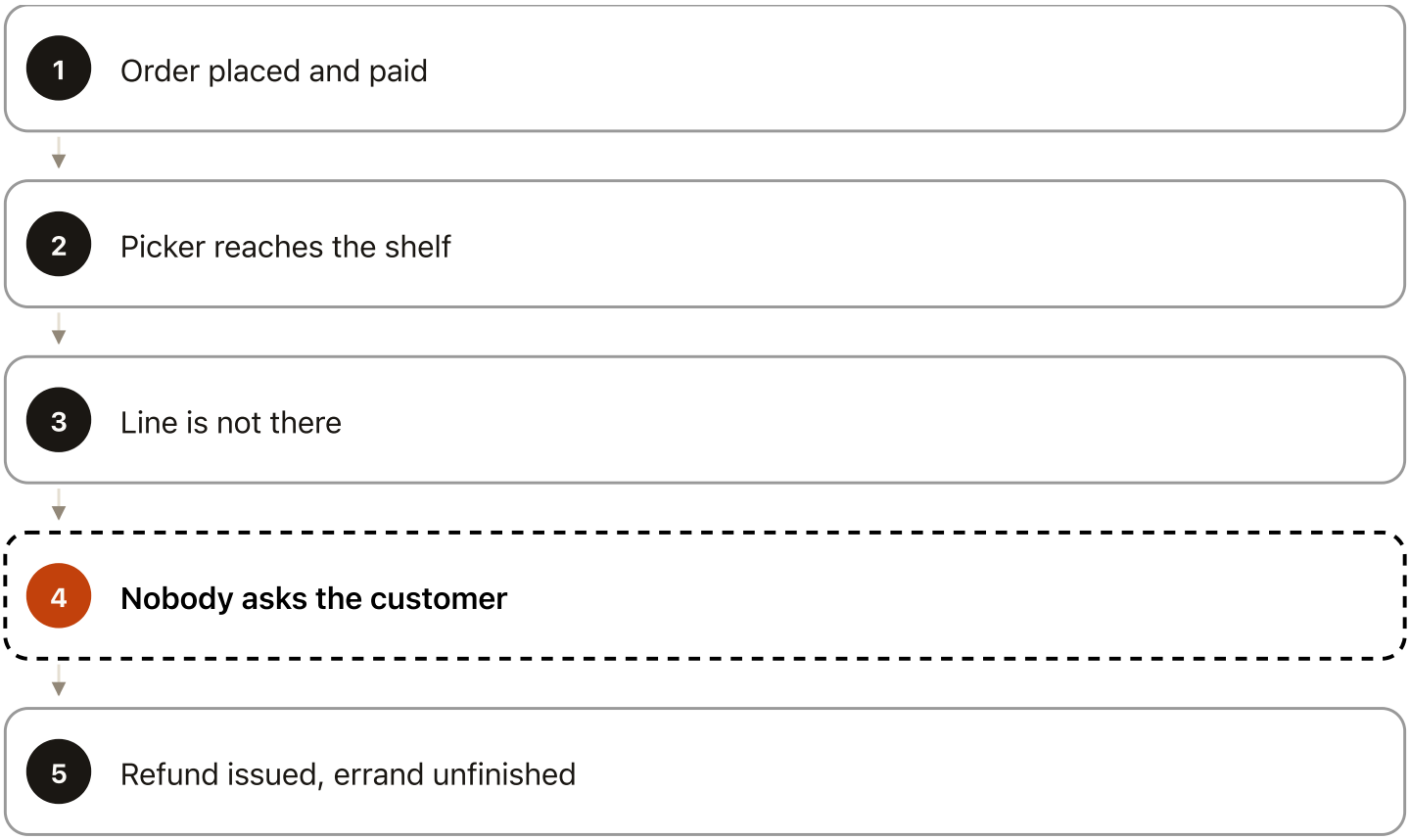
Take it as a preferences problem first. I know when a line goes into the cart whether it is fungible: toned milk, any brand, fine, and a named infant formula, no. A 500g pack in place of a 100g one is a different purchase, and I would rather have nothing.

The app has no way to hear any of that, so one policy has to cover every line. Returning the money is the policy that is safest for the platform rather than the one that finishes the errand.

Timing is the second problem, and here I am on thinner ground. The information that a line cannot be supplied exists at the shelf, and it exists while the bag is still open. Whether it exists early enough for somebody to be asked a question, I cannot tell from outside the app.

One assumption of mine, and I want it labelled as mine: a ten minute promise has to pay for rider assignment, the ride and the handover, so the time inside the store is short and a miss found there is found early. Nothing in the app confirms that. Every build below rests on it.

Why this rather than something more visible. The cost of the failure is not the refund. It is that the errand is unfinished, so a second trip is needed, which is the outcome the product exists to remove. The order record shows a delivery and a refund.



The post-payment miss, as the app handles it today. The gap is not the refund, it is that nobody is asked.

The catalogue boundary stays hidden until I walk into it

The assortment constraint is genuine and I would not argue it away. A small store cannot carry everything, and it should carry what turns over fastest. That is the correct call.

What I object to is that the boundary stays hidden until I hit it. A search with no results returns a blank space or a scatter of loosely related things. It does not say that this store does not stock the line, or that the 200g pack is here and the 500g is not. It does not offer to remember that I asked.

My side of that cost is plain. I learn the boundary by failing at it, so I stop trusting the app for anything specific, and the basket shrinks to the four things I know are always there.

The organisation's side has to be stated carefully. Search misses are standard to record at any retailer this size, so I will not claim nobody is recording them. The claim is narrower: nothing in the storefront tells the person searching that the item is not carried here, and nothing offers to tell them when it is.

Say plainly that a line is not carried here. Name the nearest thing that is and label why it is near. Offer to say when the real one comes back.

Why I would not manage this against average delivery time

Average delivery time would be on a wall somewhere if I ran this. I would not manage against it.

It averages the wrong population. Only delivered orders are in it. An order never placed because the pincode is unserved or the line is not carried is silently excluded, and those absences are what the assortment constraint produces. So is every order cancelled during packing.

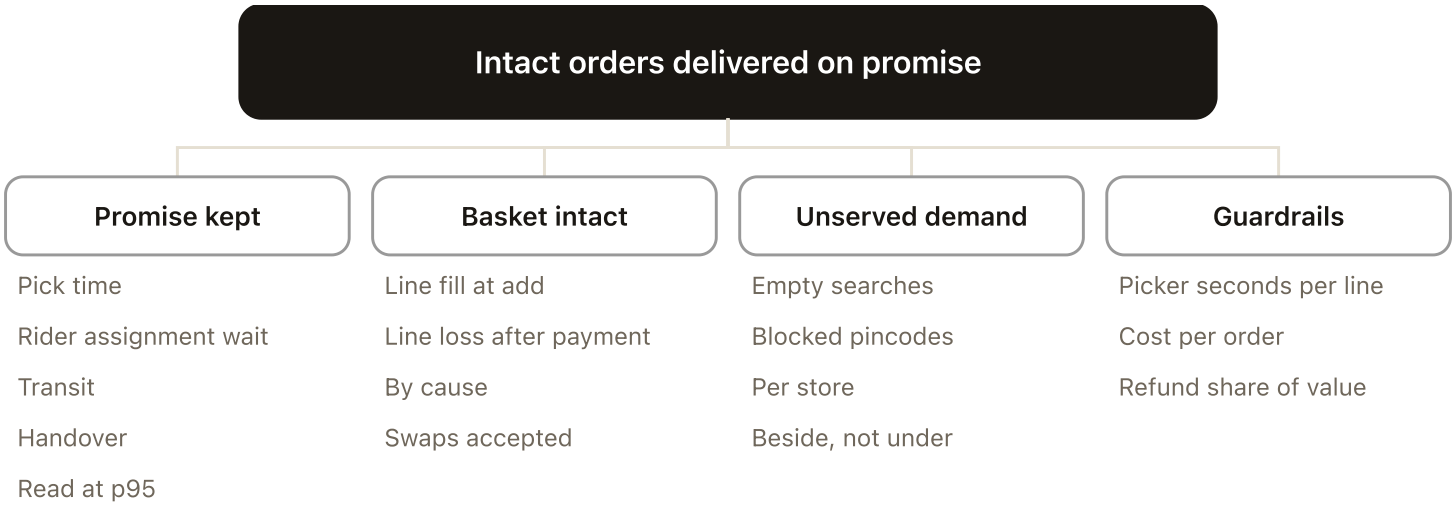
It is a mean measured against a ceiling, and the distribution has a long right tail. A store can sit under the promise on average and still break it often enough to matter. The mean cannot say how often.

Aggregated upward it becomes weighted by volume and dominated by the densest, easiest stores, so a struggling store in a hard pincode disappears inside it.

The last objection is the object being measured. Nine minutes with two of six lines refunded is a nine minute failure, and the average records it as a success.

So the top line I would choose is intact orders delivered on promise. An order counts only if every line ordered arrived, as ordered, inside the window quoted at checkout.

I would read it per store over a window long enough to hold a few hundred orders rather than per hour. A store doing a handful of orders in an hour swings on a single miss. The store manager is held to the intact rate. Timing sits underneath it as a diagnostic rather than as a target.



The tree I would run instead of average delivery time, with unserved demand beside the top line rather than under it.

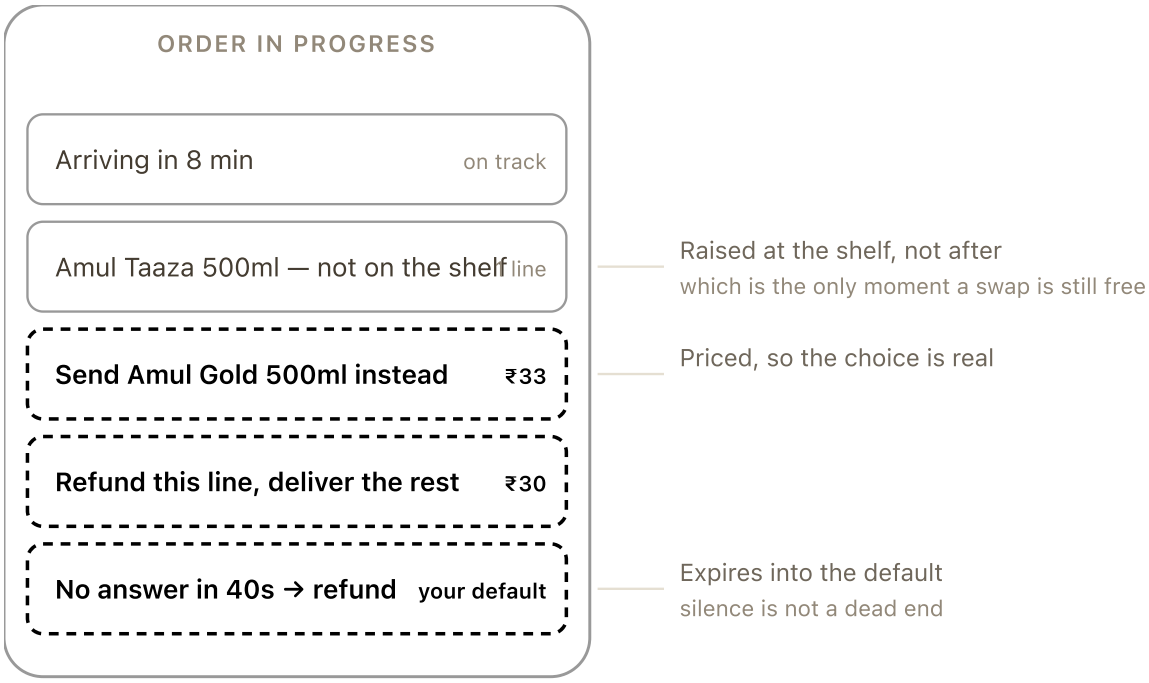
- It cannot see whether the thing that arrived was the thing wanted. A swap that completes the basket can still be the wrong purchase, and this number scores it as a win.
- The denominator is orders placed, so a store can improve the number by carrying less. That is why unserved demand sits beside it. If the two ever move in opposite directions I want to watch it happen.
- It treats a refund and a substitution as the same miss, and the two are not experienced as remotely similar. I would report them separately.
- It will get worse in the short run under the change I argue for below, because asking somebody a question takes time.

What I would build, and what I would cut

The order here is set by dependency first and by cost second, and I would rather say that than pretend failure mode is doing all the work.

The preference has to exist before the interrupt does. If the live window ships first and nobody answers it, there is no stated default to fall back on. The system returns the money anyway, and the build is an expensive notification that changes nothing.

The cost column is my estimate from outside. I have not seen the codebase, the picker app or the team, so it is an ordering from cheapest to most expensive and nothing finer than that.



The prompt that does not exist. It appears at the shelf rather than after the bag is sealed, it carries a price against each option so the choice is real, and it expires into whatever default was set at add-to-cart so a silent customer still gets an answer.

WHAT I WOULD BUILD	WHY THIS, AND WHY NOW	COST, ESTIMATED FROM OUTSIDE	VERDICT
Per line substitution preference, set as a line goes into the cart, with a default per category	The failure policy lives nowhere in the product today, so one policy has to cover every line. Stating a preference is the thing that makes any of the rest possible.	Cheapest of the five. The design risk is the real cost, because it must not add a tap for the common case.	First
Live substitution window during picking: notify when a miss is flagged, two priced options, expires on its own into the stated default	The shelf is where the information first exists, and while the bag is open a swap is still physically possible. It turns a refund into a finished errand.	Most expensive of the five. The picker app has to emit a miss as it happens and the basket has to stay open for a beat.	Second, one city, one store cluster
Honest zero result and near miss messaging, with the miss logged against store and pincode	Turns a hidden constraint into a stated one, and gives assortment a record of what was asked for and not carried.	Second cheapest, and it touches search and messaging without waiting on the picker app.	In parallel
Delivery estimate composed from the basket, when the cart holds lines that are slow to pick	Makes the promise honest for the weekly shop rather than only for the top up.	Middle of the five, and it will quote longer times, which nobody in the building will enjoy.	Later, once the timing data exists
Rider side hold so the store can check a back shelf	It reads as helpful. It spends the one thing this product cannot spend, on every order it touches.	Low to build. Expensive on the clock, every time.	Cut

The objection I have to answer, and the trade I would take

The strongest objection is not the engineering. It is that a live prompt moves a purchase decision onto somebody under a short clock, at a bad moment, and creates a failure mode that does not exist today: a swap accepted in a hurry, unwanted on arrival, returned, disputed.

My answer is the sequencing, and it is why the preference ships first. A line marked as not substitutable never generates a prompt at all, so the named infant formula is settled before anybody is under a clock. The prompt is only ever for lines already declared fungible, and if that narrows it to a small set then the build is smaller than it looks.

The top up order is the common case, and that is my inference rather than something I can count. It should not pay for this. A prompt fires only where a line is missing, so a complete three item order never sees one. If testing shows that the preference control slows the ordinary add, I would rather not collect the preference at all.

The live window makes delivery slower on the orders that use it. That is not a rounding error I can bury. Any hold at all comes out of a ten minute budget, and it comes out on the orders that were already going badly. Average delivery time will get worse. Someone will bring a chart.

I would take that trade. What is being sold is a complete order in ten minutes, and today the app holds the ten and says nothing about the completeness. Eleven minutes with everything in the bag beats nine minutes and a refund note.

The bound has to be mechanical rather than a promise in a document. The timer starts when the notification is sent rather than when it is opened. The picker carries on with the rest of the basket while it runs, and expiry needs no input from anybody. If the wait cannot be capped that way, the feature is not worth building.

The smaller thing I am giving up is surface area on the most used control in the app. If the only way to collect a per line preference is something sitting on top of the add button, I have made the common case worse to improve the rare one.

What would change my mind, and what I can test from outside

These would stop the plan, and I would want them settled before a ticket gets written. Most of them need numbers I do not have. Two of the questions I can start on from outside the app, and I would rather run those than ask anyone for access.

- How often it happens. If line loss after payment is rare enough, this is machinery for a tail and the effort belongs in assortment breadth. I will not invent the threshold: it is build cost divided by errands rescued, set against what a retained order is worth, and those are their numbers rather than mine.
- When the miss surfaces. The plan assumes the empty shelf is found early. If misses surface late in the pick there is no room to put a decision in. The right product is then a faster refund with one tap reordering of only the missing lines.
- What people choose when asked. If most set the default to return the money, substitution is my preference rather than theirs. A single city test of the preference control answers that before anything expensive is built.
- The cause. If most misses are stock the system thought it had rather than genuine sell through, this is an inventory accuracy problem. The money belongs in cycle counts and shelf discipline. That is a different team and a different budget.

- A study I can run from outside: search the same brand and pack size across a spread of pincodes and count how often it returns nothing. That sizes the assortment boundary with no internal data at all.
- A second study I can run from outside: order one fixed basket from one store repeatedly and log every miss. It will not give me a rate I would publish, and it will tell me whether the failure is common enough to chase.

What I take from it

- The absence is the finding. There is no control anywhere in this app for what should happen when a line cannot be picked, which is checkable in minutes, and it means one failure policy has to cover a basket of milk and a basket of infant formula.
- A delivery time average cannot see that failure at all, which is why I would move the top line to intact orders delivered on promise and keep unserved demand beside it.
- The build I want is slower on the orders that use it, and I would defend that in a review, because what is being sold is a complete order and not a fast one.
- If the misses turn out to be stock the system thought it had, none of this is the right work, and I would want that settled before the budget is spent.

What this is based on

Written from the outside, with no access to anything internal. Specifically:

- blinkit.com, read in August 2026. That is the same catalogue and the same ordering flow as the app, in a browser.
- Blinkit's own published cancellation and refund terms, which is what the timing argument rests on.
- No order run to the point of a line failing after payment. That path is reasoned from published terms rather than observed, and it is the weakest link in the piece.
- No internal data of any kind. No revenue, order volume, store count or fulfilment figures, because I do not have them and a wrong number is worse than no number.
- Cost in the prioritisation table is an ordering from cheapest to most expensive, estimated from outside. I have not seen the codebase, the picker app or the team.
- Independent analysis. I have never worked on this product or for this company, and nothing here comes from any employer of mine.

> The app changes often. Everything here describes what it did in August 2026, and any observation in it can be out of date by the time it is read.