PRODUCT & ENGINEERING PROPOSAL

Vesopa Order
Takeaway, Delivery & QR Table Ordering

A commission-free ordering channel for Vesopa venues — native apps for Android and iOS, scan-to-order at the table, and every order landing straight on the till.

PREPARED FOR	PREPARED BY	DATE	STATUS
Meirion Davies, Director	**Software Development, Vesopa EPOS Ltd**	**28 July 2026**	**For decision**

1 Executive summary

We already own the hard part. Vesopa runs the till, the menu, the pricing rules, the floor plan and the card payments. What we do not yet own is the moment the customer decides what to eat. This proposal adds that: three new ordering channels that all feed the till we already ship.

3
New order channels — takeaway, delivery, and scan-at-table QR ordering

2
App stores — Android and iOS, from one shared Flutter codebase

8 wks
All three channels delivered — two months from go-ahead

~2%
Cost to the venue per order, versus 14–30% on the aggregators

What we are proposing to build

A consumer app for Android and iOS where a customer picks a Vesopa venue, orders for collection or delivery, pays with Apple Pay, Google Pay or a card, and watches the order progress. This is the main focus of this document.

A QR mini-site for table ordering. The customer scans the QR on their table, the menu opens in their phone browser with no app install, they order and either pay there or pay at the counter. Sub-second to first screen — nobody installs an app to order a coffee.

A new Orders tab on the till where takeaway, delivery and table orders arrive live, are accepted or rejected by a clerk, and print to the kitchen through the routing we already have.

Back-office controls so a venue can set its own menu per channel, generate and print a QR code for each table, and point its own website or domain at its ordering page.

Why it is worth doing

Independent UK restaurants pay **14% to 30% commission** to Deliveroo, Uber Eats and Just Eat. On £10,000 of monthly online orders that is £1,400–£3,000 a month leaving the business — recurring, forever, on customers the venue often introduced itself.

A Vesopa venue ordering through its own app and its own QR codes pays card processing and its Vesopa subscription. That is the entire pitch, and it is a pitch our competitors in the till market already make.

> **THE STRATEGIC POINT**
>
> This is not a side product. It is what stops Vesopa being *only* a till. A venue that takes its online orders through us has its menu, its customers, its payments and its ordering all in one place — and a much higher cost to leave.

What I need from you

Four things, set out in section 13: the App Store publishing model, whether we build delivery ourselves, the trial venue, and how the two parallel tracks are resourced — that last one is the main risk to the date.

The shape of the build — eight weeks

PHASE	WEEKS	WHAT IT DELIVERS
0 — Foundations	1–2	Order channels in the data model, server-side pricing, live push to the till, the till Orders tab
1 — QR table ordering	2–4	Scan-to-order mini-site, per-table QR codes, pay online or at the counter. **First venue live at week 4.**
2 — Mobile apps	3–7	**Android and iOS apps for takeaway**, accounts, payments, tracking, push. Submitted to both stores at week 7.
3 — Delivery	5–7	Delivery zones, fees, driver dispatch view, live status for the customer
4 — Own domain & hardening	7–8	Venue's own domain and branding, trial-venue support, store-review buffer

The phases overlap deliberately — that is what makes eight weeks possible, and it is the plan's main assumption. Phase 2 starts once phase 0 has settled the API it depends on, so the app and web tracks then run side by side rather than end to end. Store review and venue menu onboarding run alongside and are not development weeks.

2 The three channels

All three sell the same menu, price it with the same rules, and land in the same place on the till. They differ only in how the customer arrives and how the food leaves the building.

Table ordering (QR)

Customer is in the venue

- Scans the QR on the table
- Menu opens in the phone browser — **no install**
- Table number comes from the QR, not typed
- Pays online, or adds to the table tab and pays at the counter
- Order joins the existing table bill on the floor plan

Wins: fewer staff walking, larger baskets, faster turnover.

Takeaway / collection

Customer is at home, collecting

- Orders in the **Vesopa app** or on the venue's site
- Picks a collection time from real capacity
- Always paid up front — no no-shows
- Push notification when it is ready
- Prints to the kitchen marked `TAKEAWAY`

Wins: the highest-margin online order there is. No courier, no commission.

Delivery

Customer is at home, delivered

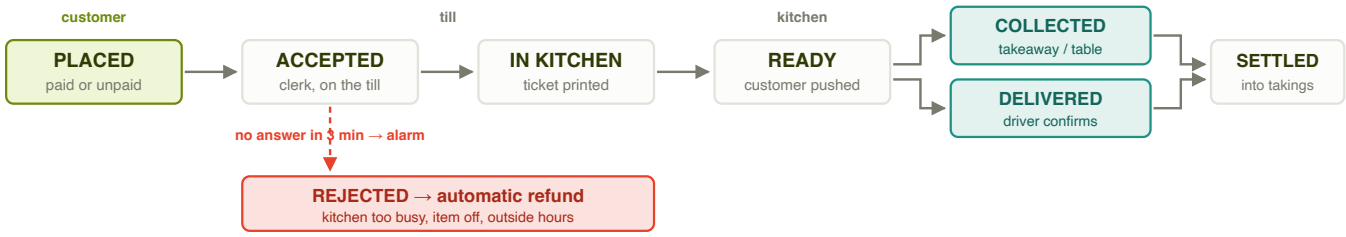
- Address checked against the venue's **delivery zone**
- Delivery fee by zone or distance
- Assigned to a driver from the till
- Customer sees status; driver marks delivered
- Prints a kitchen ticket *and* a driver docket

Wins: replaces the aggregator on the venue's own repeat customers.

One order, one lifecycle

Whatever the channel, an order moves through the same states. The till is the system of record for what happens to it; the customer's phone is only ever a view of that state.

FIGURE 1 — ORDER LIFECYCLE



The accept step is deliberate. An order that appears in the kitchen without a human agreeing to it is how a venue ends up cooking for a customer who has already left. A clerk accepts, gives a time, or rejects with a reason — and a rejection refunds the card automatically, because a venue that has to phone a customer to say no will simply stop using the feature.

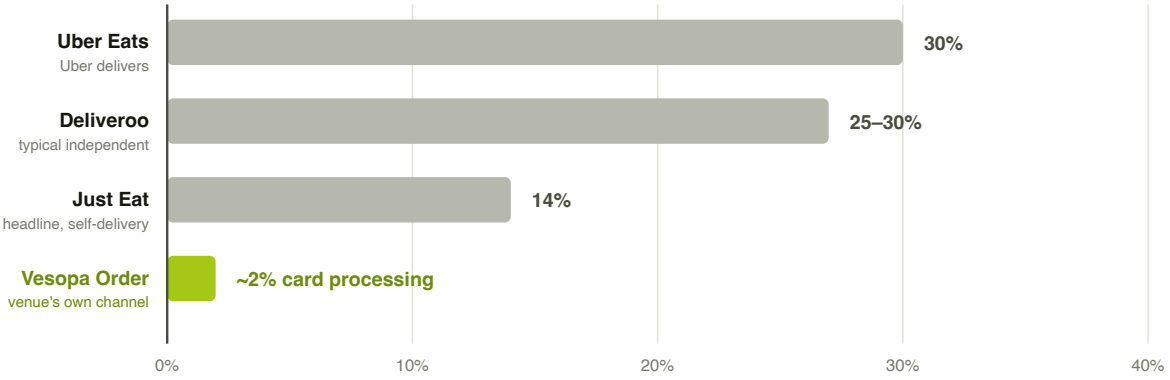
What the customer pays for, and when

CHANNEL	PAYMENT	TIMING	WHY
Table (QR)	Online, or on the table tab	Venue chooses	A pub wants a tab; a busy café wants the money before the coffee. Both are configured per venue.
Takeaway	Online only	Before the order is sent	Unpaid collection orders are a no-show problem. Card up front removes it entirely.
Delivery	Online only	Before the order is sent	Same, plus a driver is a cost we cannot recover if nobody answers the door.

3 The commercial case

The aggregators are the competitor here, not other till vendors. Their commission is the number a restaurant owner already resents, and it is the number this product is sold against.

FIGURE 2 — WHAT A UK INDEPENDENT PAYS PER ORDER



Commission taken from the restaurant, as a share of order value. Uber Eats publishes 30% when Uber delivers and 13% for self-delivery; Deliveroo does not publish a rate card and industry guides put independents at 25–30%; Just Eat's headline is 14%. Vesopa Order takes no commission — the venue pays card processing plus its existing subscription. **Rates are negotiable and vary by contract; treat these as published ranges, not quotes.**

What that means in money

A venue doing **£10,000 a month** of online orders — a modest number for a busy independent — pays:

ROUTE	MONTHLY COST	PER YEAR
Uber Eats (they deliver)	£3,000	£36,000
Deliveroo (typical)	£2,700	£32,400
Just Eat (self-delivery)	£1,400	£16,800
Vesopa Order	**~£200 + sub**	**~£2,400 + sub**

Even against Just Eat's cheapest tier the venue keeps roughly **£14,000 a year**. That is the sales conversation, and it is a conversation the salesperson does not have to be technical to win.

Where Vesopa earns

- **Subscription uplift.** Ordering is a paid module on top of the existing per-till subscription. Predictable, recurring, and it prices off the money the venue is already saving.
- **Payment processing.** Online orders route through our Dojo / Paymentsense relationship, on the same rails as the card machine.
- **Retention.** A venue whose customers have its app installed and its QR codes on the tables does not change till vendor casually.

BE HONEST ABOUT THIS

We are not replacing the aggregators. They bring *discovery* — new customers who did not know the venue existed — and we cannot. What we replace is the **repeat** order: the regular who already knows the venue and is being charged 27% to reorder from it.

The pitch is "keep Deliveroo for discovery, put your regulars on your own app" — which is what venues already try to do by hand with a phone number on the receipt.

The lever we already have

The till's receipt already supports a printed QR code (`show_qr` / `qr_url` in the branding table). Every receipt a venue prints today is free advertising space for its own ordering page.

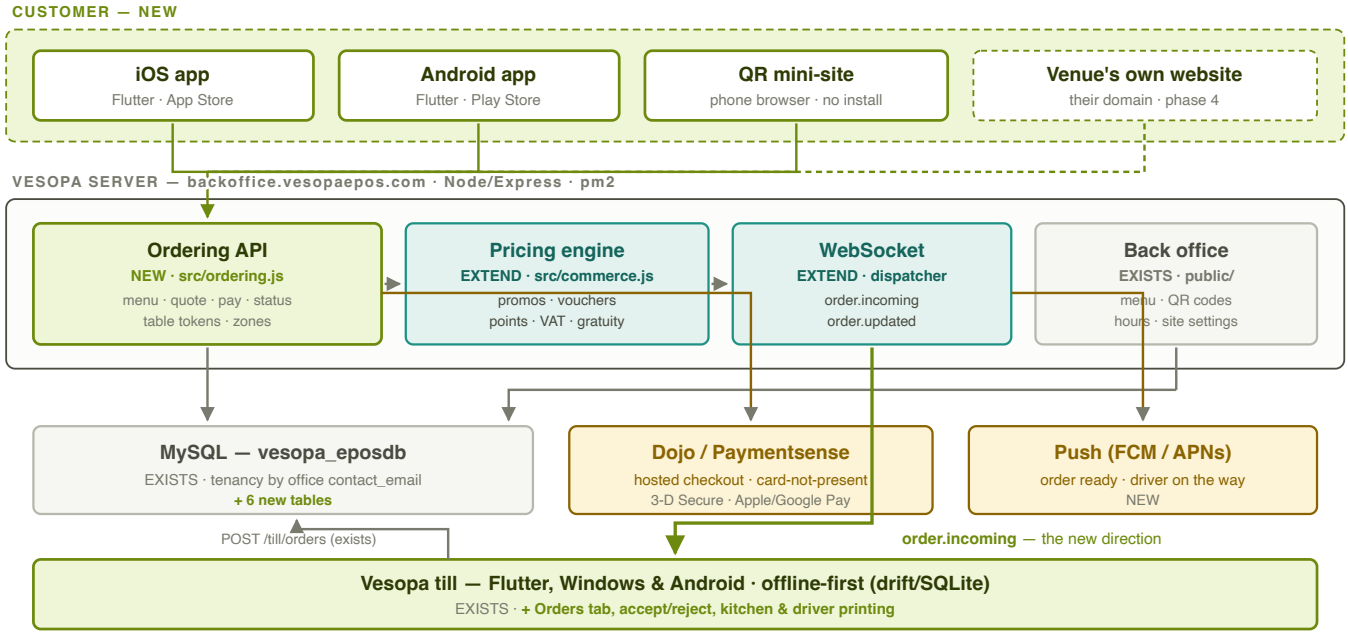
Competitive position

Square, Zonal, Lightspeed and Toast all ship online ordering with their tills. A UK EPOS vendor without it is increasingly disqualified from tenders before the demo — this is as much about **not losing deals** as about winning new revenue.

4 Where this fits in what we already have

Most of this is extension, not invention. The server, the database, the tenancy model, the menu, the pricing rules and the live socket to the till all exist and are in production. The genuinely new parts are the customer front-ends and making the order flow run *towards* the till as well as away from it.

FIGURE 3 — SYSTEM ARCHITECTURE



Teal blocks already exist and are extended. Lime blocks are new work. **The single most important change is the thick lime arrow:** today orders only travel from the till to the server. Online ordering makes that link two-way, which is what section 11 and the offline risk in section 12 are about.

Build on, don't rebuild

ASSET WE ALREADY HAVE	STATUS	HOW ORDERING USES IT
Menu & catalogue (`bo_products`)	EXISTS	The ordering menu is the till menu. One place to change a price. Needs per-channel price, availability and photo.
Multi-venue tenancy	EXISTS	Every read and write is already scoped by office contact email — the ordering API inherits that for free.
Floor plan (`floor_rooms`, `floor_tables`)	EXISTS	Every table already has an identity. The QR code is a signed token for a row that is already there.
Pricing rules	EXTEND	Promotions, vouchers, points, VAT and gratuity all exist. They must now run server-side for online orders — see 11.2.
Card payments (Dojo / Paymentsense)	EXTEND	Existing relationship and hosted-checkout page. Online is card-not-present rather than card-present.
Live socket to the till	EXTEND	The socket and reconnect logic exist and push catalogue changes. We add order events to the same channel.
Kitchen ticket printing	EXTEND	Printer routing per product and a ticket layout that already prints a `TAKEAWAY` header.
Back office SPA	EXTEND	Add menu-per-channel, opening hours, QR generator, delivery zones and site settings as new sections.

5 The mobile apps: the decision that comes first

Before a line of app code is written there is one decision that changes everything downstream: **do we ship one Vesopa app containing every venue, or a separately branded app per venue?** Getting this wrong does not cost us a sprint — it costs us the App Store listing.

THE CONSTRAINT

Apple's App Review Guideline **4.2.6** states that apps created from a commercialised template or app-generation service will be rejected *unless they are submitted directly by the provider of the app's content*. Fifty near-identical restaurant apps from our own developer account are rejected as spam. Enforced since 2017; it removes the obvious business model.

Apple names the acceptable alternative explicitly: **a single binary hosting all client content in an aggregated or "picker" model — for example a restaurant-finder app with a customised entry per client restaurant.** That is precisely the product described here.

The three options

OPTION	STORE RISK	COST	ASSESSMENT
A — One Vesopa app, venue picker RECOMMENDED	LOW	One build	Exactly the model Apple names as acceptable. One binary, one submission, one review cycle. Every new venue is a database row, live the same day — **no store review to onboard a customer.** Cross-venue discovery is a genuine benefit. *Cost:* the venue's brand sits inside ours rather than on the home screen.
B — One app per venue, Vesopa's account	HIGH	Very high	**Do not do this.** The exact pattern 4.2.6 exists to reject, and a rejection risks our whole developer account. Also unworkable: every venue's onboarding waits on Apple.
C — One app per venue, the *venue*'s account	MEDIUM	Per venue	Permitted — 4.2.6 is satisfied when the content provider submits from its own account. The venue pays Apple $99/yr and Google's fee and owns the listing; we build from the same codebase with their branding. Sell as a premium add-on. **Phase 4, not phase 2** — the per-venue release burden is real.

Recommendation

Ship option A now; offer option C as a paid add-on from phase 4.

Architecturally these are the same app: option C is option A with the venue picker removed and the venue's colours swapped by build flavour. Building A properly makes C a configuration exercise later rather than a second product — which is why branding must be data-driven from the first commit.

Two things that would otherwise surprise us

Apple's 30% in-app purchase cut does not apply. Physical goods and services consumed outside the app — which is what food is — are explicitly outside the IAP requirement. We take payment with our own processor. This is the same basis on which Deliveroo and Uber Eats operate.

Apple will want to see real content. An app submitted with two demo venues gets rejected as incomplete. We must have a handful of live venues with real menus before first submission — which is why phase 1 (QR ordering) deliberately comes first: it populates real menus.

Why Flutter for the consumer apps

The team already ships it

The till is Flutter on Dart 3.11, in production on Windows and Android, with a tested pricing engine. Anything else means a second language and a second toolchain for one developer.

The money code is reusable

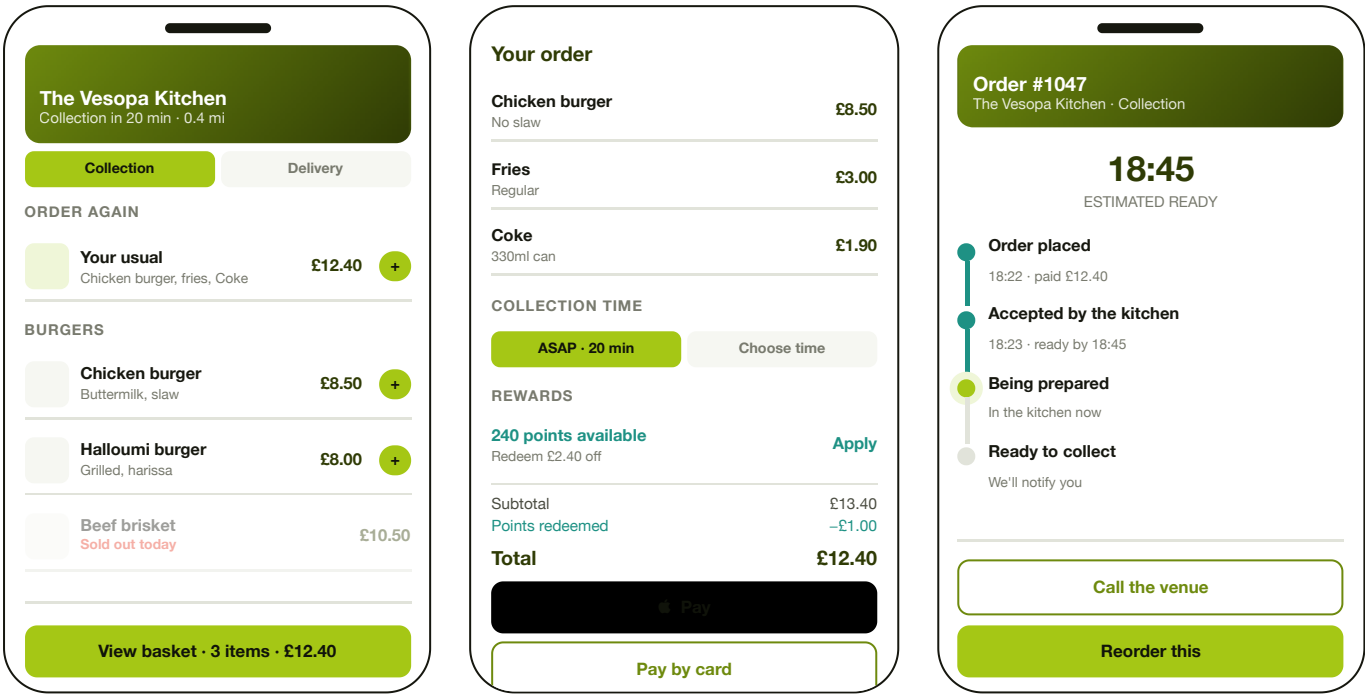
The money model, tender kinds and display formatting are already written in Dart and covered by tests. The app reuses those types rather than reimplementing them in a third language.

One codebase, both stores

Native iOS plus native Android is roughly double the effort for a UI this conventional. **Flutter is the obvious call and I would not spend long debating it** — with one exception, the QR mini-site, examined in section 8.

6 The app, screen by screen

The whole app is judged on one number: how many taps from opening it to having paid. The target is **a repeat customer reordering their usual in under fifteen seconds**, and every screen below is arranged around that.



1 · Menu | 2 · Checkout | 3 · Live tracking

Indicative layouts, not final design. The deliberate details: **"Your usual" is first on the menu** (the repeat order is the business case); **sold-out items are greyed, not hidden**; and **Apple Pay / Google Pay is the primary button** — it roughly halves checkout abandonment against typing a card number.

What ships in the first version

AREA	INCLUDED
Find a venue	Nearby list, search, recently ordered. Opening hours and "closed now" shown honestly.
Menu	Categories, photos, modifiers, allergen notes, per-item availability.
Basket	Quantities, item notes, live totals quoted by the server, promotions applied.
Account	Sign in by phone number and one-time code. No password to forget.
Payment	Apple Pay, Google Pay, saved cards, 3-D Secure. Card details never reach us.
Loyalty	Points earned and redeemed against the existing loyalty scheme.
Tracking	Live status, estimated time, push at accepted and ready.
Reorder	One tap from history. The highest-value screen in the app.

Explicitly not in version one

- **Table booking** — a separate product with its own complexity; do not let it in.
- **In-app chat with the venue** — needs staffing the venue does not have. A phone button instead.
- **Group ordering / split the bill in-app** — genuinely wanted, but it doubles the basket model. Post-launch.
- **Live driver map** — status updates in phase 3, a moving pin only if delivery volume justifies it.
- **Scheduled orders days ahead** — capacity modelling gets hard fast. Same-day time slots only at first.

THE THING MOST LIKELY TO SINK THIS

Not the code — **the menu data**. A till menu is written for a clerk who knows the products: terse names, no photos, no descriptions, no allergens. A consumer will not order "CHK BRG 2" from a blank row.

Every venue needs its menu enriched before it can go live. We must budget for this as an *onboarding service*, build the back-office tools to make it fast, and not discover it at the first venue trial.

7 Payments, push, and getting into the stores

7.1 Taking money without touching a card number

The till today takes **card-present** payments — a physical terminal, a customer standing there. Online ordering is **card-not-present**: a different integration, a different risk profile, and different rules.

CONCERN	HOW IT IS HANDLED
PCI scope	Card details are entered in the processor's own hosted fields and never reach a Vesopa server, app or database. This keeps us in the lightest self-assessment tier — a genuinely important cost and liability difference.
Strong Customer Authentication	3-D Secure is mandatory for UK card-not-present. The processor's SDK handles the challenge; we handle the "customer got interrupted mid-challenge" case, which is the one that actually breaks in production.
Apple Pay / Google Pay	Primary payment method. Both satisfy SCA without a challenge screen, so they are faster *and* convert better. Apple Pay needs a merchant certificate per environment — a set-up task with lead time, not a coding task.
Refunds	A rejected order must refund automatically, from the server, with no clerk action. Anything less and staff stop rejecting orders they should reject.
Chargebacks	New exposure the venue does not have today. Needs a back-office view of disputes and a documented policy before the first venue goes live.

OPEN QUESTION FOR THE PAYMENTS PARTNER

We have a working card-present integration and a hosted-checkout page already written for the till. What we need to confirm with Dojo / Paymentsense is the **card-not-present product**: mobile SDKs for iOS and Android, Apple Pay and Google Pay support, tokenised saved cards, server-side refunds, and the commercial rate for online transactions.

This is on the critical path for phase 1 and I would like to open that conversation now. If their online offering is weak, the fallback is a dedicated online processor alongside them — worse commercially, but not a blocker.

7.2 Push notifications

Firebase Cloud Messaging for Android, APNs for iOS, both driven from the server when an order changes state. Two rules: notify only on **accepted** and **ready** (plus **out for delivery**), and never send marketing pushes from the shared app without the venue's own opt-in — one venue spamming would get the app uninstalled for all of them.

7.3 Getting into the stores, and staying in

ITEM	LEAD TIME	NOTE
Apple Developer Program	Up to 3 weeks	Organisation account under Vesopa EPOS Ltd. Needs a D-U-N-S number — **this must be started in week 1**. On an eight-week plan it is the single likeliest cause of missing the date, and it needs no code from us.
Google Play Developer	Days	One-off fee. Organisation accounts now require identity verification too.
App review	1–3 days typical	First submission is the risky one. Budget two rejection cycles: real venues with real menus, a working demo account for the reviewer, and a privacy policy that matches what we actually collect.
Privacy labels	With submission	Both stores require a declaration of data collected and why. Location, contact details and order history all need declaring — and the declaration has to be true.
Ongoing releases	Every 2–4 weeks	Store review sits between us and every fix. **Anything that might need changing urgently must be server-driven, not compiled in** — menus, prices, opening hours, fees, copy.

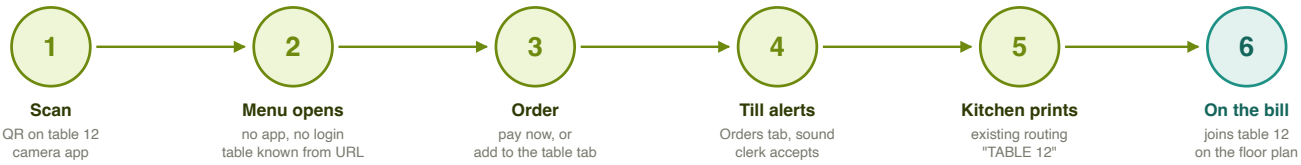
THE DESIGN RULE THAT FOLLOWS FROM THAT LAST ROW

Because an app release takes days and a server deploy takes minutes, the app ships as a *thin client*: it renders what the server tells it and computes as little as possible. Prices, availability, fees, opening hours, delivery zones and even which channels a venue offers are all server-supplied. The app should never need a release because a venue changed its mind.

8 QR ordering at the table

A customer sitting at a table will not install an app. This channel lives or dies on the time between pointing a camera at a sticker and seeing food — the budget is **under two seconds**, on a mid-range Android, on café Wi-Fi.

FIGURE 4 — FROM STICKER TO KITCHEN



Step 6 is the part competitors get wrong. A QR order that creates a separate, parallel bill leaves staff reconciling two tabs for one table. Ours joins the order the floor plan already has, so a customer can order two rounds on their phone, one at the bar, and still get a single bill.

The URL and the QR code

Phase 1 uses a Vesopa-hosted address, one path per venue, with a signed token identifying the table:

```
order.vesopaepos.com/the-vesopa-kitchen/t/A7K2M9
```

The token is short, opaque and signed. It **must not** be a guessable table number: a URL ending `/t/12` invites a customer to change it to 13 and order onto someone else's bill. Tokens are revocable per table, so a QR that leaks — printed on a sticker in a public room, photographed and posted — can be rotated without reprinting the whole venue.

Generating and printing the codes

A new back-office screen lists the venue's rooms and tables from the existing floor plan and produces a **print-ready A4 sheet** of table tents — venue logo, table number, QR, and a short "scan to order" line. Reprint one table or the whole room. The venue prints it on their own printer; no fulfilment, no stock, no shipping.

Why this is not the Flutter app compiled for web

It is the obvious idea and it is the wrong one. A Flutter web build ships a multi-megabyte engine before it renders anything; on a phone on café Wi-Fi that is several seconds of blank screen, and the customer has already put the phone down. A small hand-written web page is **an order of magnitude smaller** and paints almost immediately.

The honest cost: two customer front-ends to maintain rather than one. I think that is clearly the right trade, for three reasons — the mini-site is a much smaller surface than the app (menu, basket, pay: no accounts, no history, no push), both front-ends call the identical API so the logic lives in one place, and it matches how the back office is already built: plain HTML, CSS and JavaScript with no build step, which anyone on the team can pick up.

> **DESIGN CONSTRAINTS THAT FOLLOW**
>
> No framework bundle. Menu data cached so a second scan is instant. Works on an iPhone that has never seen the site and will never install anything. Readable one-handed. And a permanent, obvious **"call the waiter"** button — the goal is to remove the walk to take the order, not to remove the staff.

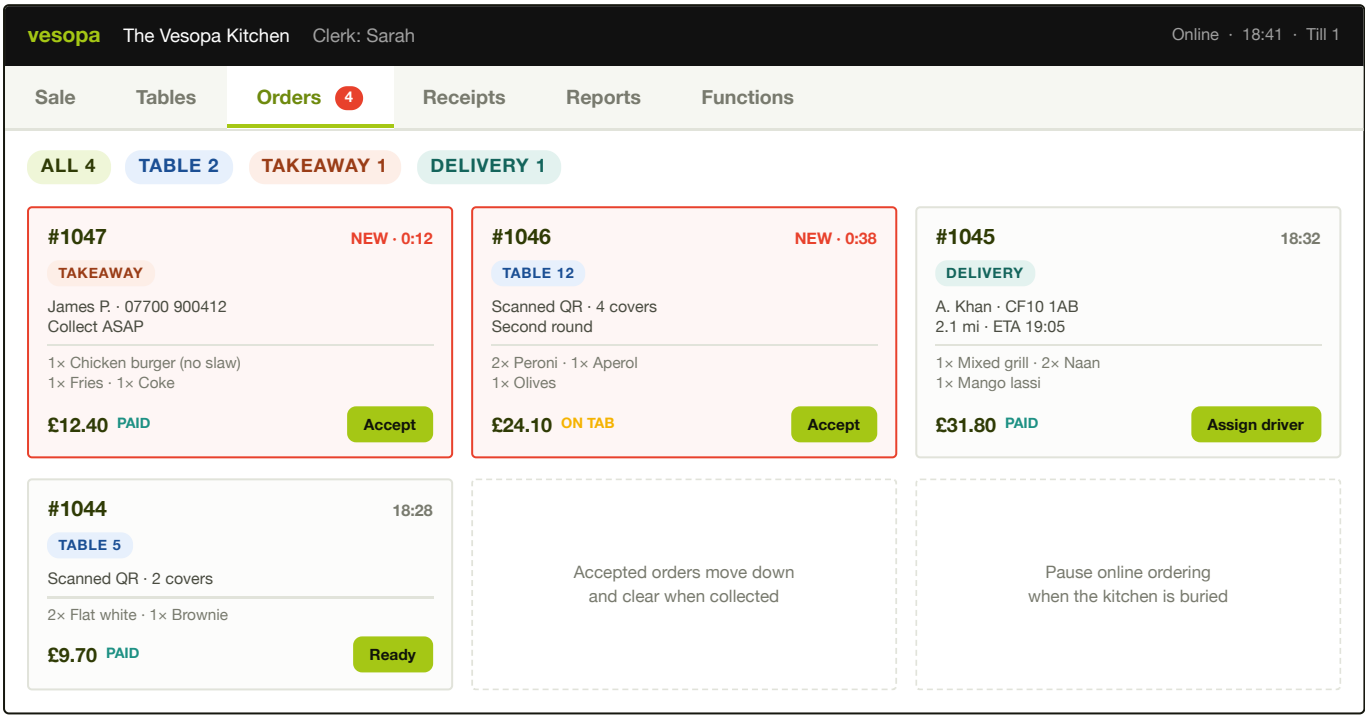
Pay now or pay at the counter

Set per venue. **Pay now** settles the order immediately and the table bill shows it as paid. **Pay at the counter** adds the items to the open table bill and the customer settles as they do today — which is what pubs and table-service restaurants will want, and what makes this sellable to venues that would otherwise refuse card-only ordering.

9 What changes on the till

From the clerk's point of view this is one new tab. Behind it is the most operationally sensitive part of the whole project: a kitchen that misses an order loses a customer, and blames us.

FIGURE 5 — THE NEW ORDERS TAB



Indicative layout on a standard till screen. **The count on the tab, the countdown on new orders, and the sound are the whole feature.** A clerk mid-service is not looking for orders — the till has to interrupt them, and keep interrupting until someone responds.

Behaviour that is not optional

- **Audible alert on arrival**, repeating until acknowledged. A silent badge in a busy kitchen is a missed order.
- **Escalation.** Unacknowledged after three minutes: louder, full-screen, and the customer is told there is a delay before they work it out.
- **One-tap "pause online ordering."** When the kitchen is thirty minutes behind, the venue must stop the tap instantly. Without it, staff turn the feature off permanently after one bad Saturday.
- **Accept with a time.** Not just yes — "ready in 25 minutes", pushed to the customer.
- **Reject with a reason,** refunding automatically.
- **Prints on accept, not on arrival,** so a rejected order never reaches the pass.

THE HARD PROBLEM

The till is deliberately **offline-first**: it keeps selling through a broken internet connection and syncs later. That design is a real strength — and it does not extend to *incoming* orders. If the venue's connection drops, orders customers have already paid for cannot reach the till.

This needs three defences, all in phase 0:

1. The till reports its connection state; the server knows within a minute.
2. An offline venue is **automatically taken off online ordering** — customers see "not accepting orders right now" rather than paying for food nobody will cook.
3. Orders taken in a gap are queued and replayed on reconnect, with stale ones flagged for a human rather than silently printed an hour late.

A second defence: the Orders tab runs on any Vesopa till, including the Android build — so a venue can put a cheap tablet in the kitchen on 4G, independent of the shop's broadband.

10 Delivery, and what the venue controls

10.1 Delivery

Delivery is deliberately phase 3. It is the channel with the most moving parts and the least of it is software — but for venues that already self-deliver it is the one that replaces an aggregator outright.

PIECE	WHAT WE BUILD
Delivery zones	Postcode-prefix list or a radius, each with its own fee and minimum order. Postcodes first — venues think in postcodes, not metres.
Address capture	Postcode lookup with a proper address database. Free-text addresses produce failed deliveries, and a failed delivery costs the whole order.
Fees and minimums	Per zone, quoted by the server before payment so the total the customer approves is the total charged.
Driver assignment	From the Orders tab: assign to a named driver, print a docket, mark dispatched. A simple list, not a routing engine.
Driver status	A minimal mobile view — the driver's drops and a "delivered" button. Same Flutter codebase, sign-in by PIN.
Customer view	Status only in phase 3: accepted, preparing, out for delivery, delivered. A live map is a post-launch decision.

DECISION NEEDED — SECTION 13

We should not build a courier network. The plan above supports venues that already have their own drivers, which is most independents doing delivery today. Venues without drivers are served later by plugging in a third-party courier API as an option — a partner integration, not a fleet.

Building dispatch, driver supply and delivery liability ourselves is a different company, not a feature. I want that ruled out explicitly rather than left to drift into scope.

The VAT trap

UK VAT on food depends on how it is sold: hot food and anything eaten in is standard-rated, while much cold takeaway food is zero-rated. **The same product needs a different VAT rate depending on the channel.** Our catalogue holds one rate per product today.

This is a data-model change, not a rounding detail — it is the venue's VAT return. It is in phase 0 for that reason, and it should be confirmed with the venue's accountant at the first trial.

10.2 What the venue controls in the back office

Menu per channel

Which products appear online; a separate online price if they want one; photo, description and allergens; a per-item availability switch; and the channel-specific VAT rate. A "sold out today" toggle that clears overnight.

Hours & capacity

Ordering hours per channel, separate from opening hours. Preparation times. A cap on orders per fifteen minutes so a rush cannot take more than the kitchen can cook — the setting that prevents most bad experiences.

QR codes

Generate, print and rotate codes for every table on the existing floor plan. Printable A4 table-tent sheets with the venue's logo. Reprint one table or a whole room.

Their site & domain

Phase 1 gives every venue a hosted page at `order.vesopaepos.com/<venue>`. Phase 4 adds their own domain with an automatic certificate, plus an embeddable "Order online" button.

Branding

Logo, colours and header image, applied to their ordering page, their entry in the app, and their table tents. Extends the branding table the receipt designer already uses.

Orders & reporting

Online orders in the existing reports, split by channel. This is what a venue looks at when deciding whether to keep paying for it.

On "restaurants can add their own website" — worth being precise, because it is easy to over-promise. We are not building a website builder; that is a large product unrelated to EPOS. We give the venue an ordering page they can point their own domain at, and a button they can paste into whatever site they already have. A venue with no site gets one decent ordering page, which is more than many have now.

11 The engineering work, honestly stated

Three changes carry most of the risk. All in the server, all in phase 0, all cheaper now than retrofitted.

11.1 Orders must travel towards the till

Today the till posts finished sales to the server and the server pushes only catalogue changes back. Online ordering needs the reverse: an order created on the server, delivered live to a venue's tills, acknowledged, and updated as it progresses. We extend the existing socket with `order.incoming` and `order.updated` rather than building a second channel.

> **KNOWN CONSTRAINT IN THE CURRENT DEPLOYMENT**
>
> The server runs as a **single process** because the socket dispatcher holds connections in memory — under multiple workers a push only reaches the worker holding that till's socket. Fine for the first venues; a scaling ceiling once inbound orders are business-critical. The fix is a shared pub/sub layer. **Flagging it now, not building it now** — but it should be a known item, not a surprise at the hundredth venue.

11.2 One pricing engine, not two

Money rules — automatic promotions, then manual discount, then vouchers, then points, then gratuity, with VAT apportioned across discounted values — are in Dart on the till, covered by a substantial test suite. An online order is priced by the *server*, so without care we get two implementations of the same rules drifting.

The server becomes canonical for online orders. The app and mini-site display an optimistic total, but every order is quoted and confirmed by the server before payment, and the existing Dart tests are ported against the server implementation as a contract suite so the two cannot silently diverge. The single highest-value piece of engineering discipline in the plan.

11.3 The catalogue has to grow up

A consumer-facing menu needs what a till menu does not: photographs, descriptions, allergens, modifiers, per-channel price and VAT, availability. Most does not exist.

CHANGE	TYPE	DETAIL
`epos_orders`	ALTER	Channel, state, requested time, contact, address, payment ref. Additive — till sales untouched.
`bo_products`	ALTER	Online visibility, online price, description, allergens, per-channel VAT, availability.
Product modifiers	NEW	Option groups with price deltas — "no slaw", "large". App and kitchen ticket both need them.
Table tokens	NEW	Signed, revocable token per existing floor-plan table row. Rotatable individually.
Venue ordering settings	NEW	Channels, hours, prep times, capacity cap, payment mode, slug, domain, branding.
Delivery zones	NEW	Postcode prefixes or radius, fee, minimum order, per venue.
Online customers	EXTEND	Existing table holds points and membership. Add credentials, addresses, device tokens.
Order events	NEW	Append-only log of every state change, who and when. This is what settles disputed orders.

Security, in one place

- **Public endpoints are genuinely public** — so they need rate limiting and abuse controls the till never did.
- **Prices come from the server, always.** A client that submits a price can submit £0.01; the order carries ids and quantities only.
- **Table tokens are opaque and revocable** — never a bare table number.
- **Card data never reaches us.** Hosted fields only.
- **Personal data** puts us in scope for UK GDPR: retention, deletion, and a privacy policy matching the store declarations.

Testing this properly

- The ported pricing contract suite is the backbone — if online and till totals ever differ, the build fails.
- End-to-end tests through the real lifecycle including reject and refund, the path least likely to be exercised by hand.
- Payment testing against the processor's sandbox, including 3-D Secure challenge and abandonment.
- **A deliberate offline drill** at the trial venue: pull the internet mid-service and confirm customers are turned away cleanly.
- Device testing on cheap Android hardware, not just new iPhones.

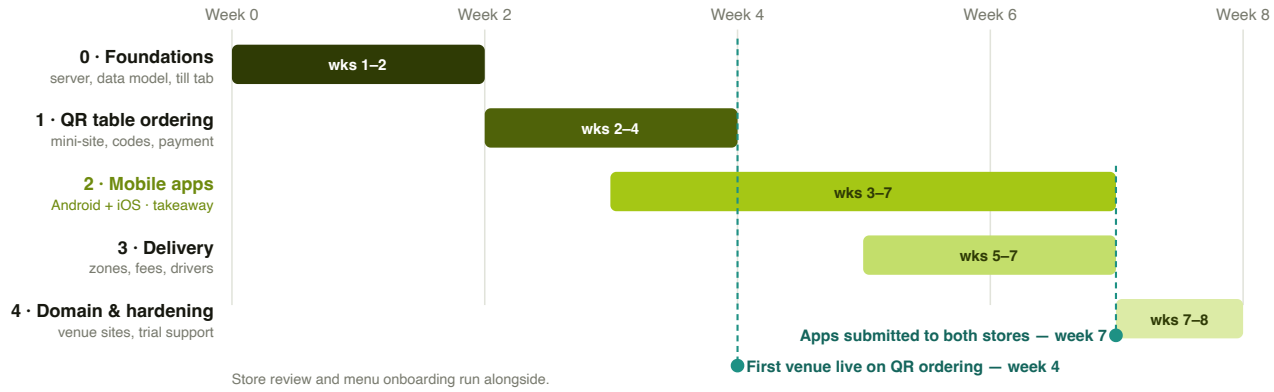
12 What will actually go wrong

Every one of these is survivable if it is planned for and expensive if it is discovered. They are ordered by how much damage they do, not how likely they are.

RISK	IMPACT	WHAT HAPPENS	HOW WE HANDLE IT
Venue misses an order	SEVERE	A customer pays, nobody accepts, they arrive to nothing. One incident destroys a venue's confidence in the whole product permanently.	Repeating audible alert, three-minute escalation, automatic customer notification on delay, one-tap pause. Trial venue runs with us watching for two weeks.
App Store account & rejection	SEVERE	D-U-N-S verification can take three weeks — inside an eight-week plan that is nearly half the schedule. A 4.2.6 rejection on top of it would end the date, and a per-venue app strategy risks the account itself.	**Open the developer account in week 1, before any code.** Single-binary picker model per section 5 — the model Apple names as acceptable. Real venues and menus at first submission; budget two review cycles.
Internet drops at the venue	SEVERE	The till keeps selling in the room but cannot receive online orders. Customers pay for food nobody will cook.	Heartbeat from the till, automatic suspension of online ordering when a venue goes dark, queue and replay with stale orders flagged for a human. Kitchen tablet on 4G as a second path.
Online and till prices differ	SEVERE	Two implementations of the money rules drift. The customer is charged the wrong amount and we cannot say which total was correct.	Server canonical for online, till tests ported as a contract suite that fails the build on divergence. Section 11.2.
Wrong VAT by channel	HIGH	Eat-in and takeaway VAT differ under UK rules. One product, one rate today. Gets the venue's VAT return wrong.	Per-channel VAT in the catalogue from phase 0. Confirmed with the trial venue's accountant before any second venue.
Menu data is not consumer-ready	HIGH	Till menus are terse codes with no photos or descriptions. Onboarding stalls at every venue and the app looks empty.	Treat menu enrichment as a funded onboarding service with proper tooling — bulk edit, image upload, copy-from-template. Budget it per venue.
Payment partner gap	HIGH	Our card relationship is card-present. If the online product lacks mobile SDKs, wallets or server-side refunds, phase 1 stalls.	**Confirm now**, before phase 0 ends. Fallback is a dedicated online processor alongside — commercially worse, not a blocker.
Kitchen overwhelmed	HIGH	Online orders arrive alongside a full dining room. Service collapses and staff blame the tablet.	Capacity caps per time slot, honest prep times, pause button, and staff training that this is a tool they control rather than one imposed on them.
Nobody installs the app	HIGH	The apps are the biggest single investment and consumer acquisition is genuinely hard.	QR ordering ships first and needs no install; receipt QR codes and table tents drive the app to people already in the venue. **We never depend on cold app-store discovery.**
Single-process server ceiling	MEDIUM	Live order push requires one process. Fine now; a limit once orders are business-critical across many venues.	Documented now, shared pub/sub when volume warrants. Not phase 0 work.
Chargebacks and fraud	MEDIUM	New exposure for venues that have only ever taken card-present payments.	3-D Secure on every transaction, dispute view in the back office, documented policy before first go-live.
Scope creep	MEDIUM	Table booking, loyalty campaigns, marketing emails, a live driver map — each reasonable, together a year.	Section 6 lists what is explicitly out. Phases 1 and 2 ship independently; anything new waits until after launch.

13 Roadmap, and what I need decided

FIGURE 6 — EIGHT-WEEK DELIVERY PLAN



The overlap is the plan. Phase 2 starts as soon as phase 0 has settled the API it consumes, so the app and web tracks run in parallel rather than end to end — that is what fits eight weeks, and it is the assumption the date rests on. QR ordering still lands first because it needs no app store and no customer acquisition: it proves the pipeline on a real venue and populates the real menus Apple will want to see at submission.

Decisions I need from you

Everything else I can proceed on. On a two-month run there is no slack to rediscover these later, so I would like all four settled in **week 1**.

1 App Store publishing model

One Vesopa app containing every venue, or a branded app per venue? **Recommendation: one Vesopa app with a venue picker — the model Apple explicitly names as acceptable under 4.2.6 — with per-venue branded apps sold later as an add-on, published from the venue's own developer account.** This is the decision with the least room to change later.

2 Do we get into delivery logistics?

Recommendation: no. Support venues with their own drivers, and integrate a third-party courier later as an option. Building driver supply and taking on delivery liability is a different business. I want this ruled out explicitly so it cannot drift into scope.

3 The trial venue

Which venue goes first, and are they briefed that they are a trial? I need one cooperative venue that will take real orders, tell us when it breaks, and let us sit in during service — **confirmed by week 3**, because the week-4 go-live depends on it. **Ideally one with an existing aggregator bill, so we can measure what we actually saved them.**

4 How the parallel tracks are resourced

Not a scope question — a date question. Eight weeks works because the app and web tracks overlap from week 3 (figure 6). **Run end to end by one person, the same work is closer to twenty weeks.** I need to know what I can count on from week 3. If it stays one developer, the honest move is to hold the two-month date for *QR ordering and the apps* and let delivery follow, rather than ship all three half-finished.

TWO THINGS THAT MUST START IN WEEK 1

Neither needs a line of code, both have external lead times, and on an eight-week plan both are on the critical path: **(1)** open the Apple Developer organisation account — D-U-N-S verification is regularly a three-week wait, which is nearly half this schedule and the likeliest single reason we would miss the date; **(2)** confirm with Dojo / Paymentsense what their online, card-not-present product supports. If their offering is thin we need to know in week 1, not week 5, because every channel depends on taking money.

Commission figures in section 3 are published ranges for UK independents as of July 2026 and vary by contract; indicative, not quotations. The eight-week schedule assumes the figure-6 overlap is genuinely resourced, the week-1 items start on time, and the trial venue is confirmed by week 3.