

Taqnix POS Terminal

A guide to Taqnix POS Terminal: installing and running the app, preparing Odoo, connecting a terminal, selling, payments and receipts, working offline, orders and sync, restaurant tables and kitchen tickets, the Pro screens, closing a session, settings, themes, the tax engine, architecture, security, accuracy and limits.

1. Start here

In plain words: **Taqnix POS Terminal** is a point of sale for **Odoo 19** that runs in the browser as an installable app (a PWA). It keeps the catalogue, the customers and the open orders on the device, keeps selling when the network is gone, and sends every order to Odoo with the call Odoo's own point of sale uses, `pos.order.sync_from_ui`. There is no Odoo module to install: the terminal is a separate web application that you run on your own server and that talks to your Odoo. It comes in English and Arabic, ready to run (you only need Node.js) and as complete source code, for one company and an unlimited number of terminals.

It is made for cafés, shops and restaurants that run their sales on Odoo 19, and for integrators who set it up for them (one purchase for each client company). Chapters 1 to 7 get it running, 8 to 21 are daily use, 22 to 24 are for developers, and 25 to 30 cover security, what was and was not verified, troubleshooting, updates and support.

The pictures come from a demo shop, **Harbor Café & Market** (US dollars, Dearborn MI) with two points of sale. **Main Shop** is retail and café: prices shown with tax, cash rounded to 0.05, the loyalty programme Harbor Rewards and the promo code SUMMER10. **Harbor Restaurant** has the floors Ground floor and Terrace, a Kitchen printer and a Bar printer, and shows prices without tax. The cashier is Sara Haddad, a plain Point of Sale user. The Odoo address on the pictures is a local test server: you type the address of your own Odoo.

Odoo is a trademark of Odoo S.A. This product is not made or endorsed by Odoo S.A.



The terminal in four moments: selling (also offline), orders that wait and then sync, tables and kitchen, and a split payment with change.

2. What you get

In plain words: Everything the terminal does, with the chapter that explains it.

Feature	What it does	Chapter
Setup and connection test	Server, database, login and API key; a report on the version, the user's rights and the apps found.	6
Point of sale and opening cash	Choose the point of sale, count the opening cash, open the session.	7
Sell screen, variants, combos	Catalogue, search, barcode scanner; order panel with numpad, discount, note, pricelist; options and combos.	8, 9
Customers	Search, create (also offline when the Odoo user may); pricelist and fiscal position follow the customer.	10
Payment and receipts	Several payments, change, cash rounding, customer account, invoice flag; 58 or 80 mm receipts with a barcode and a QR code.	11
Offline work, Orders, Sync	Selling goes on without a network; orders wait on the device and are sent when Odoo answers. Reprint, refund, retry, export.	12 to 14
Restaurant	Floor plan, table orders, kitchen tickets, split, transfer, merge, bill with tip. Only with Odoo's Restaurant app and a restaurant point of sale.	15
Pro screens	Dashboard, Stock lookup, Layaway, Loyalty and promo codes. Included in the price; each has a switch in Settings.	16 to 19
Close the session, Settings	Count the cash, close, print the report. Themes, brand colour, language, PIN lock, manager PIN.	20, 21
Ready-to-run web app	The terminal already built, with a launcher for Windows, macOS and Linux: unzip it, start it, open the address. Only Node.js is needed.	4
Source code and tools	The complete Next.js source, the tests (3,336 in the standard run), a Dockerfile and an nginx example.	22 to 24

Built with Next.js 16, React 19, TypeScript, Tailwind CSS 4, IndexedDB through Dexie, Zustand and Zod.

3. What you need

In plain words: An Odoo 19, a small server for the terminal, HTTPS, and a Chrome or Edge browser on each till.

Item	Details
Odoo	Odoo 19 Community with Point of Sale . Enterprise was not tested. Setup warns about an older server and does not refuse it; other versions were not tested.
Optional Odoo apps	Restaurant (<code>pos_restaurant</code>), Loyalty (<code>pos_loyalty</code>), Discount (<code>pos_discount</code> , whole-order discount), Inventory (<code>stock</code> , stock lookup). A part whose app is missing does not appear.
The server of the terminal	Node.js 20.9 or newer (all the ready-to-run app needs), or Docker (the image uses Node 24 and is built from the source). It serves the app and forwards calls to Odoo.
HTTPS	The terminal installs as an app and works offline only from an <code>https://</code> address or <code>http://localhost</code> (chapter 4).
The till	A tablet or PC with Chrome or Edge (service workers, IndexedDB, Web Locks). Tablet layouts (1024 × 768, 768 × 1024) are shown in this guide.
An Odoo user	One for each terminal or cashier: Internal User plus Point of Sale / User , with an API key (chapter 5).
Printer, scanner	Optional. Printing goes through the browser. A scanner must act like a keyboard. Neither was verified on real hardware (chapter 27).

4. Install and run the app

In plain words: The terminal is a Node.js application. The package holds it twice: ready to run (unzip it and start it, only Node.js is needed) and as source code (build it yourself). Either way

you start it on a computer or server, put HTTPS in front of it when tills on other devices use it, then open its address on each till and install it from the browser. Docker works too.

Option 1: the ready-to-run web app. Nothing to install or build.

- 1 Install Node.js 20.9 or newer from nodejs.org (`node -v` shows the version you have).
- 2 Unzip the package, then unzip `taqnix-pos-terminal-webapp.zip` in it. The folder `taqnix-pos-terminal-webapp` is the finished application.
- 3 Open a terminal (Command Prompt on Windows) in that folder and start it with the address of your Odoo: `sh start.sh https://erp.example.com` on macOS and Linux, `start.bat https://erp.example.com` on Windows.
- 4 Open `http://localhost:3000` in Chrome or Edge and connect the terminal to Odoo (chapter 6). Stop it with Ctrl+C.

```
# macOS and Linux, in the folder taqnix-pos-terminal-webapp
sh start.sh https://erp.example.com
# Windows (Command Prompt, or double-click start.bat)
start.bat https://erp.example.com
# another port; every option
sh start.sh https://erp.example.com --port 3100
sh start.sh --help
# the same without the launcher script
node run.cjs https://erp.example.com
```

The launcher checks the Node.js version, adds the Odoo address you type to

`POS_ALLOWED_ORIGINS`, reads `.env` (copy `.env.example` to `.env` to keep your settings) and says so when the port is taken. The folder has its own `README.txt` too.

It listens only on this computer (127.0.0.1) unless you say otherwise. That is enough to try it, and for a till on the same computer: `http://localhost` counts as a secure address, so the app installs and works offline. For tills on other devices start it with `--host 0.0.0.0` (or `POS_HOST=0.0.0.0`) and put HTTPS in front of it (below). To have it start with the computer and restart after a crash, run `node run.cjs` with a process manager you already use (systemd, pm2, Windows Task Scheduler).

Option 2: build it from the source code. Use it when you change the app or want to build it on your own server.

- 1 Unzip the package, then unzip `taqnix-pos-terminal.zip` and open a terminal in `src/pos-terminal` (the folder with `package.json`).
- 2 Run `npm ci` , then copy `.env.example` to `.env` and set `POS_ALLOWED_ORIGINS` to the address of your Odoo.
- 3 Run `npm run build` (it also writes the service worker and the list of licences), then `npm start` . The server listens on port 3000; use `npm start -- -p 3100` to change it.
- 4 Put a TLS front in front of the port (below), then open the HTTPS address in Chrome or Edge on the till.

```
npm ci
npm run dev          # development server on http://localhost:3000 (permissive proxy de
npm run build        # production build + service worker (scripts/make-sw.mjs) + licenc
npm start            # production server (NODE_ENV=production), port 3000 (-p to change
POS_STANDALONE=1 npm run build  # .next/standalone, used by the Dockerfile
npm run pack:webapp    # after that build: a ready-to-run folder in webapp-dist/, like th
docker compose up -d    # build + run with the settings of .env (copy .env.example)
```

The server reads its settings from environment variables, in `.env` or in the environment of the process:

```
POS_ALLOWED_ORIGINS=https://erp.example.com
# POS_ALLOW_ANY_ORIGIN=1
# POS_ALLOW_PRIVATE=1
# POS_CONNECT_SRC=https://erp.example.com
# POS_TRUST_PROXY=1
# POS_DISABLE_JSONRPC=1
# POS_HOST=127.0.0.1
# PORT=3000
```

Variable	Meaning
<code>POS_ALLOWED_ORIGINS</code>	Comma list of the Odoo origins the <code>/api/odoo</code> proxy may reach. In production nothing else is allowed.
<code>POS_ALLOW_ANY_ORIGIN=1</code>	Accept any Odoo address typed in Setup. Only for a trusted LAN or development: on a reachable server it makes the proxy a relay to any host.
<code>POS_ALLOW_PRIVATE=1</code>	Allow private addresses (10.x, 192.168.x, 127.0.0.1) for origins not listed. Link-local and cloud-metadata addresses are always refused.
<code>POS_TRUST_PROXY=1</code>	A reverse proxy sets <code>X-Forwarded-For</code> , <code>-Host</code> and <code>-Proto</code> . Without it these headers are ignored.
<code>POS_DISABLE_JSONRPC=1</code>	Refuse <code>/jsonrpc</code> : only API keys (JSON-2), never Odoo passwords. Recommended on Odoo 19.
<code>POS_CONNECT_SRC</code>	Extra origins in the Content-Security-Policy, only for the direct connection mode. It is read when the headers are built: set it for <code>npm run build</code> and for the start.
<code>POS_PROXY_LOG=1</code>	Log the method, path and status of proxied calls, never bodies or keys.
<code>POS_HOST</code> , <code>PORT</code>	Only for the ready-to-run app: the address (default <code>127.0.0.1</code> , this computer only) and the port (default 3000) it listens on. <code>0.0.0.0</code> opens it to the network: only behind HTTPS.

Development mode (`NODE_ENV` not `production`) allows any origin: never use it on a reachable server.

Docker. Copy `.env.example` to `.env` and run `docker compose up -d`. The Dockerfile (Node 24, Next.js standalone output, non-root user, health check) and the compose file are provided; the image was not built and tested here.

```
# docker-compose.yml
services:
  pos:
    build: .
    restart: unless-stopped
    ports:
      - "3000:3000"
    env_file: .env

# without compose (header of the Dockerfile)
docker build -t taqnix-pos . && docker run -p 3000:3000 --env-file .env taqnix-pos
```

HTTPS. `deploy/nginx.conf.example` is a TLS front; run the app with `POS_TRUST_PROXY=1` behind it. It sets `client_max_body_size 26m` (the proxy accepts 25 MB), a 70 second read timeout (the proxy waits 60 seconds for Odoo) and never lets `/sw.js` be cached, or updates would never arrive. On the internet add a VPN, single sign-on or an IP allow-list (chapter 25).

```

# TLS front for the terminal (a PWA can only be installed from https:// or http://localhost)
# Run the app with POS_TRUST_PROXY=1 behind this proxy (it writes the X-Forwarded-* headers)
# Internet-facing? /api/odoo has no login of its own: restrict who reaches it (VPN, SSO /
# proxy, or an IP allow-list like the commented `allow`/`deny` lines below).
server {
    listen 443 ssl http2;
    server_name pos.example.com;
    ssl_certificate      /etc/letsencrypt/live/pos.example.com/fullchain.pem;
    ssl_certificate_key  /etc/letsencrypt/live/pos.example.com/privkey.pem;

    client_max_body_size 26m;          # the proxy accepts up to 25 MB (orders with many li

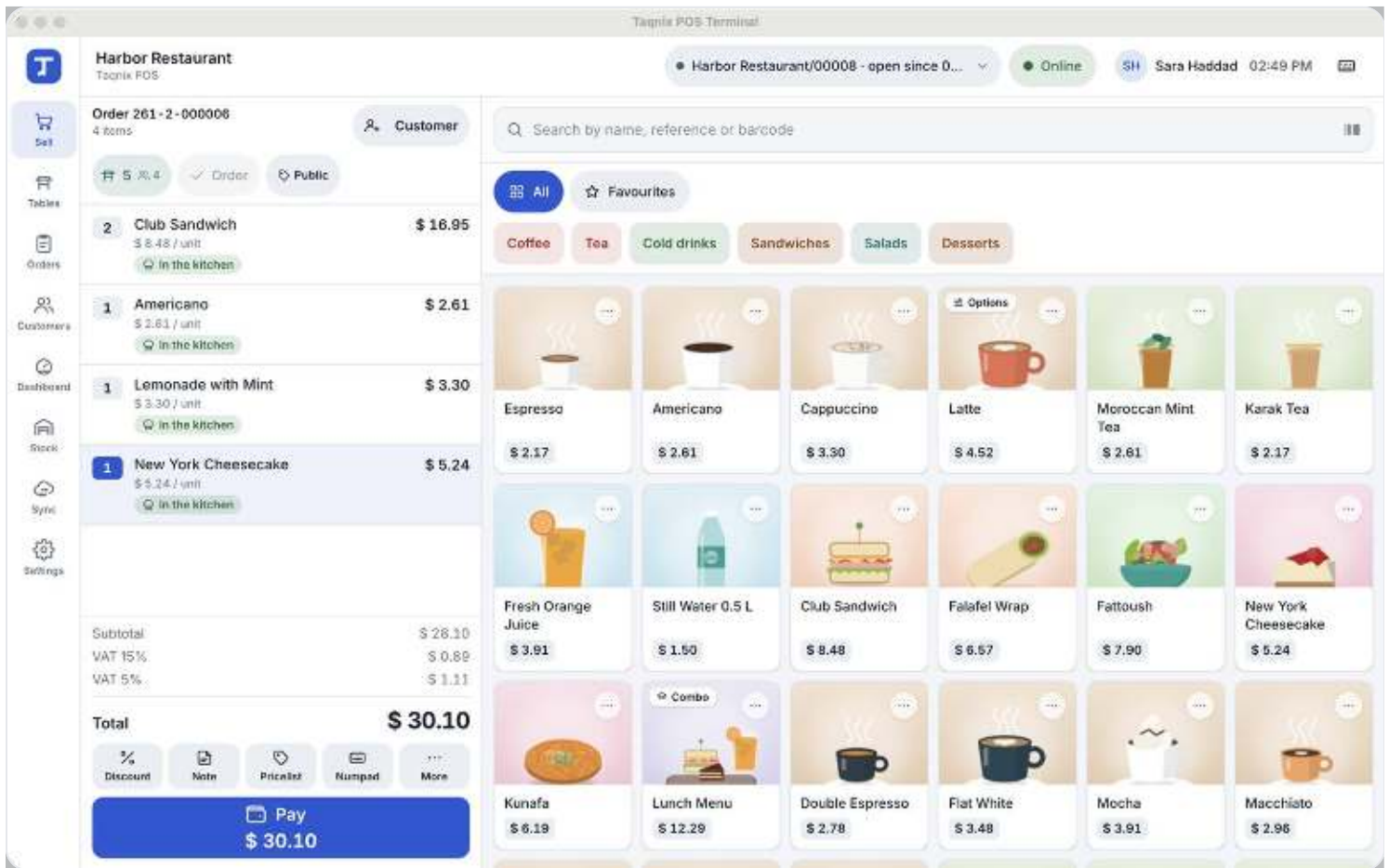
    # allow 203.0.113.0/24;          # the shop's public address(es)
    # deny all;

    location / {
        proxy_pass http://127.0.0.1:3000;
        proxy_set_header Host $host;
        # overwrite (never append) what the client sent: the app trusts these only with POS_T
        proxy_set_header X-Forwarded-For $remote_addr;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_read_timeout 70s;      # the proxy gives Odoo 60 s
    }

    # The service worker must never be cached by a CDN or by nginx
    location = /sw.js {
        proxy_pass http://127.0.0.1:3000;
        add_header Cache-Control "no-cache, no-store, must-revalidate" always;
    }
}

```

On each till. Open the HTTPS address in Chrome or Edge once while online: the browser stores the whole app (a service worker), and from then on the terminal starts without a network. Install it with the install button of the browser. The terminal also offers it in a banner (**Install**, **Dismiss**), and **Settings** → **About** has an **Install app** button.



The restaurant point of sale on table 5 in an installed app window titled Taqnia POS Terminal, with no tabs and no address bar.

5. Prepare Odoo

In plain words: The terminal logs in to Odoo as an ordinary user. Prepare one user for each terminal or cashier, give it an API key, and check the point of sale. Nothing is installed in Odoo.

- 1 Create the user (**Settings → Users**): **Internal User** with the access right **Point of Sale / User**, and nothing more unless you need a right below.
- 2 Log in as that user, open the preferences, **Account Security**, press **New API Key**, name it after the terminal and copy the key at once: Odoo shows it only one time.
- 3 Check the point of sale: its payment methods (one cash method), cash control if you want the cash counted, and the invoice journal if cashiers will use the **Invoice** switch.
- 4 Check the taxes and fiscal positions (below), then connect the terminal (chapter 6).

Important: Use one API key for each terminal, from a user with only the Point of Sale group. When a device is lost, revoke its key in Odoo.

A plain Point of Sale user cannot do a few things in Odoo 19. The Setup test lists what the user lacks, and the terminal works without these rights.

Right in Odoo	What it unlocks
Contacts / Creation	Create and edit customers. Without it the New customer form says so and Odoo refuses the customer.
Products / Create	Save a free-text option value (Other (specify)). Without it that order waits in the Sync screen.
Accounting / Invoicing	Record cash in and out. This terminal has no button for it: do it in Odoo.
Accounting read rights	Odoo's full Z report (Point of Sale → Reporting → Sales Details). A manager prints it in Odoo.
Inventory / User	Sell products tracked by lot or serial number. Without it those orders wait in the Sync screen with "missing group: Inventory/User".
Point of Sale / Administrator	Close a session stuck in closing control (Point of Sale → Sessions → Close Session & Post Entries). Not needed daily.

Important: Fiscal positions must map each tax explicitly to a tax, also to a 0 % tax. A position without tax lines is inconsistent in Odoo 19 itself: Odoo's point of sale keeps the taxes, its server removes them, some orders are saved as unpaid orders named "/", and the closing can need a forced close. Taxes computed by Python code cannot be computed offline (chapter 23).

Restaurant. Floors, tables, the kitchen and bar printers with their product categories, the tip product and the service presets (Dine In, Takeout, Delivery) are set in Odoo (**Point of Sale → Configuration**). The terminal shows them but does not edit the floor plan.

6. Connect the terminal

In plain words: The first time you open the terminal it shows **Connect this terminal to Odoo**. Choose the language and the theme, type the address, database, login and API key, run a test, and save. The key stays on this device.

Connect this terminal to Odoo
Point of Sale for Odoo 19 that keeps saving when the network is gone.

Language: English العربية

Theme: Light Dark Ocean Forest Sunset

Server URL*
https://erp.example.com
The address you open Odoo with, for example https://erp.example.com

Database*
Find databases

Login*

Password or API key*
An API key of a user with only the Point of Sale group is recommended

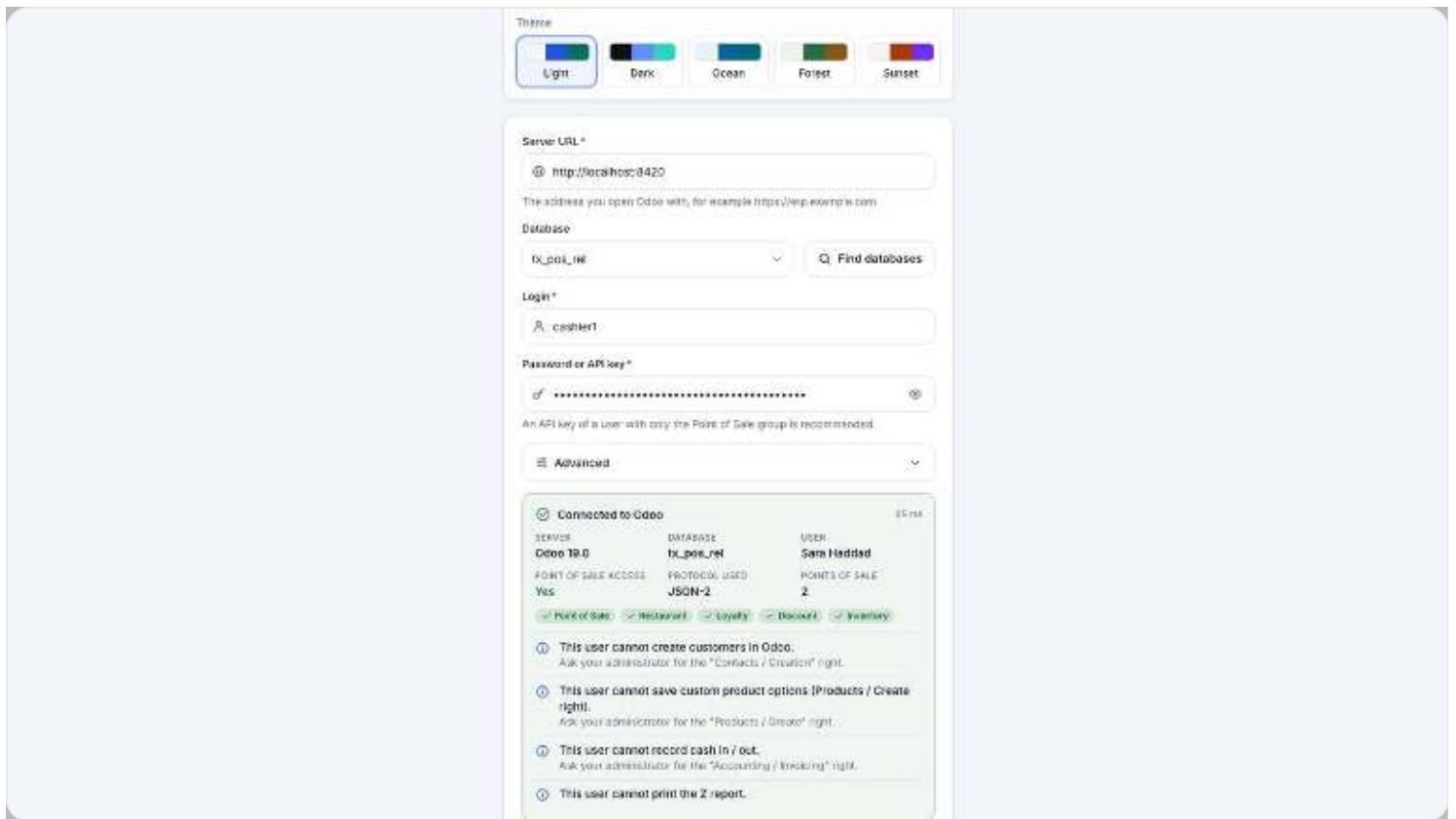
Advanced

Test connection Save and continue ->

The Setup screen: language and theme chips, Server URL, Database, Login, Password or API key, Advanced, Test connection, Save and continue.

Field	What to type
Server URL	The address you open Odoo with, for example <code>https://erp.example.com</code> , no page path. Without a scheme the terminal tries https, and for a server in your own network (localhost, private address, single name, <code>.local</code> , <code>.lan</code>) then http, and says so. An address typed with <code>https://</code> is never changed to http.
Database	The name. Find databases lists them when the server publishes the list.
Login, Password or API key	An API key of a user with only the Point of Sale group is recommended. An API key pasted with a space or line break around it is trimmed.
Advanced → Connection	Through this app (proxy) is recommended and works with any Odoo. Direct to Odoo only when your Odoo's reverse proxy allows this app (CORS); then set <code>POS_CONNECT_SRC</code> .
Advanced → Protocol	Automatic uses JSON-2 when the server offers it, otherwise JSON-RPC. JSON-2 needs an API key; with a password the terminal uses JSON-RPC, deprecated by Odoo 19 and announced for removal in Odoo 20.
Request timeout (seconds)	How long to wait for Odoo before working offline: 5 to 120, default 30.
Lock with a PIN	6 to 12 digits. The key is then stored encrypted and the PIN is asked at start (chapter 25).

Test connection changes nothing in Odoo. It checks the server, database, login and the right to use the Point of Sale, counts the points of sale, reads the installed apps and asks what the user may do. **Save and continue** works only after a good test (it runs the test for you if you did not).



The report: Connected to Odoo (85 ms), Odoo 19.0, user Sara Haddad, JSON-2, 2 points of sale, five apps found, four notes about missing rights.

The four notes are information, not errors: they list rights the user lacks (chapter 5).

Message	Cause and fix
Cannot connect to {host}.	Wrong host or port, Odoo stopped, no HTTPS certificate or, in direct mode, no CORS. Check host and port; the report offers the http address when there is no certificate.
This terminal cannot reach its own server (the app that forwards to Odoo). Odoo itself may be fine.	The Node.js app is stopped or unreachable. Start it and test again.
This address does not look like an Odoo server. / This server is not Odoo 19.	Another site answers: use the address of Odoo itself. The second is a warning only; other versions were not tested.
This database does not exist on the server. / The login or the password / API key is not accepted.	Press Find databases ; check the login and paste the key again (Preferences → Account Security).
Too many failed logins from this network. Odoo waits a minute ...	Wait one minute and test again.
This user cannot use the Point of Sale. / There is no point of sale this user can open.	Give Point of Sale / User , or create a point of sale.
This app refused to forward to that server.	Add the server to <code>POS_ALLOWED_ORIGINS</code> and restart.
This terminal only accepts API keys: password logins are turned off on this server.	<code>POS_DISABLE_JSONRPC=1</code> is set. Use an API key.

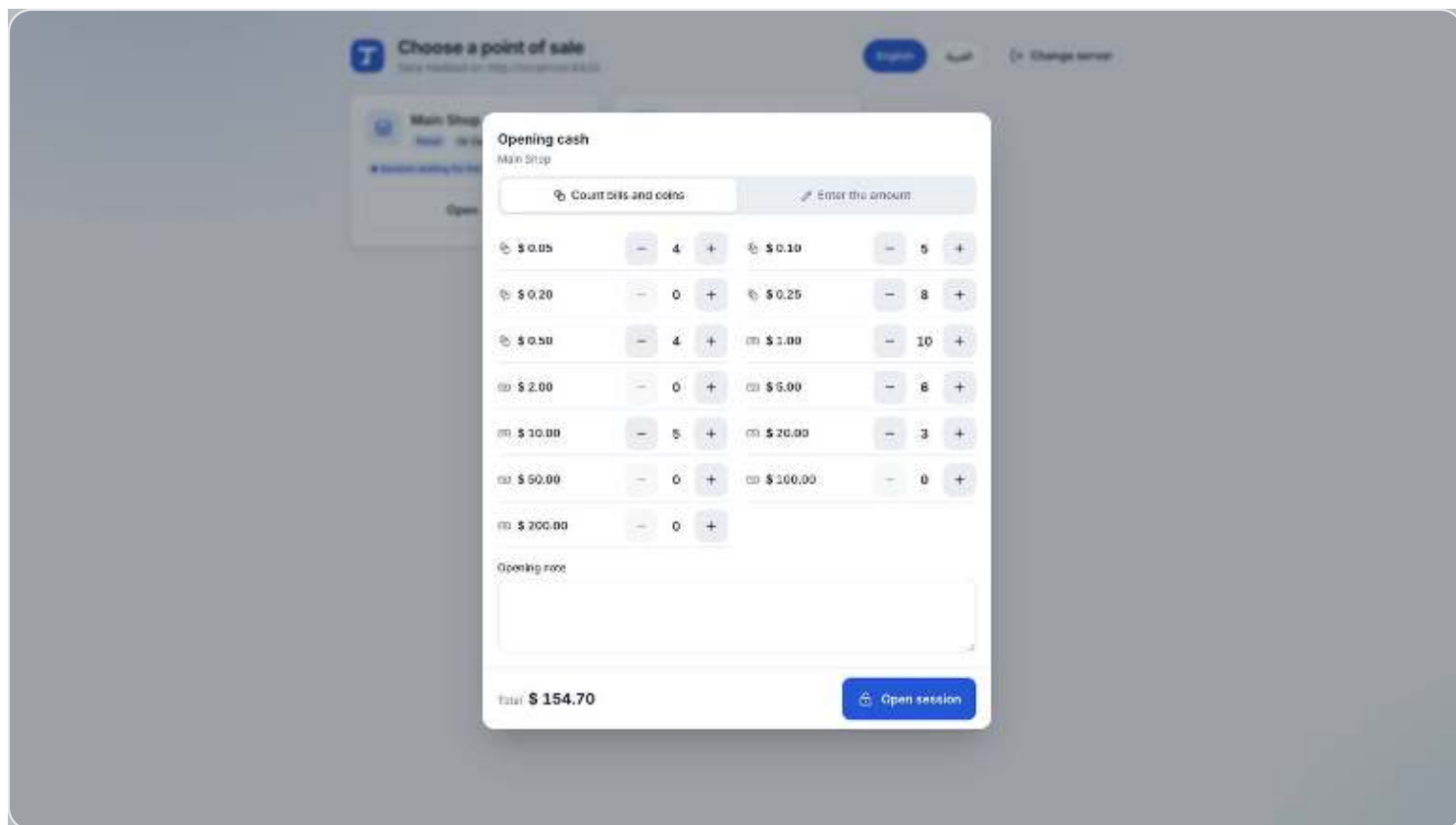
7. Choose the point of sale and open the session

In plain words: After the connection is saved, the terminal lists the points of sale the user may open. You open one, count the opening cash and start selling. Next time it goes straight back to the session you left.



Choose a point of sale: Main Shop (Retail) and Harbor Restaurant (Restaurant), both Closed, each with Open.

Each card shows the name, **Retail** or **Restaurant**, **Cash control**, and the state of the session. The button says **Open**, or **Continue** when a session is already open; it is disabled while a session is being closed. **Change server** signs out of this server, and the terminal refuses to switch server while orders are unsent.



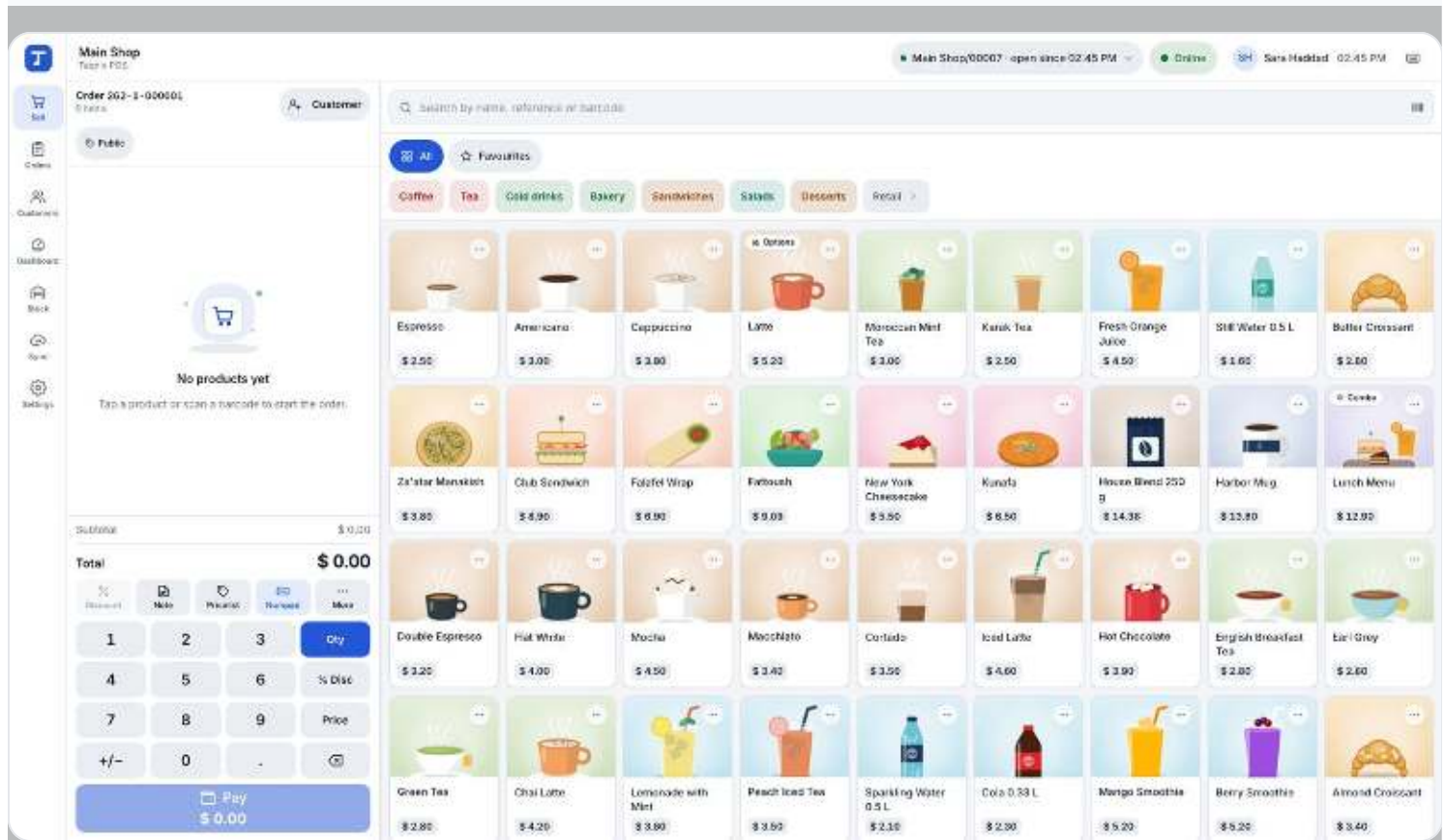
Opening cash for Main Shop: bills and coins with steppers, total 154.70, Opening note, Open session.

With cash control you use **Count bills and coins** (the values come from Odoo; on the picture 4×0.05 , 5×0.10 , 8×0.25 , 4×0.50 , 10×1.00 , 6×5.00 , 5×10.00 and 3×20.00 make 154.70) or **Enter the amount**, add an **Opening note** and press **Open session**. Without cash control only the note is asked.

Situation	What happens
First opening on a device	Odoo gives the device an identifier (Settings → Terminal identity). Open the first session while online.
The terminal starts offline	It continues the session on the device, with no dialog, and checks Odoo in the background.
The session was closed in the back office	A red banner: "This session was closed on another device. Open a new one." with Open a new session . The terminal notices within about a minute; waiting orders go to the new session.
The session is in closing control	A red banner: a manager must finish the closing in Odoo (chapter 20).

8. The Sell screen

In plain words: The catalogue is on the right and the current order on the left. One tap on a product adds it. Every change is saved on the device at once, so a reload brings you back to the same order.

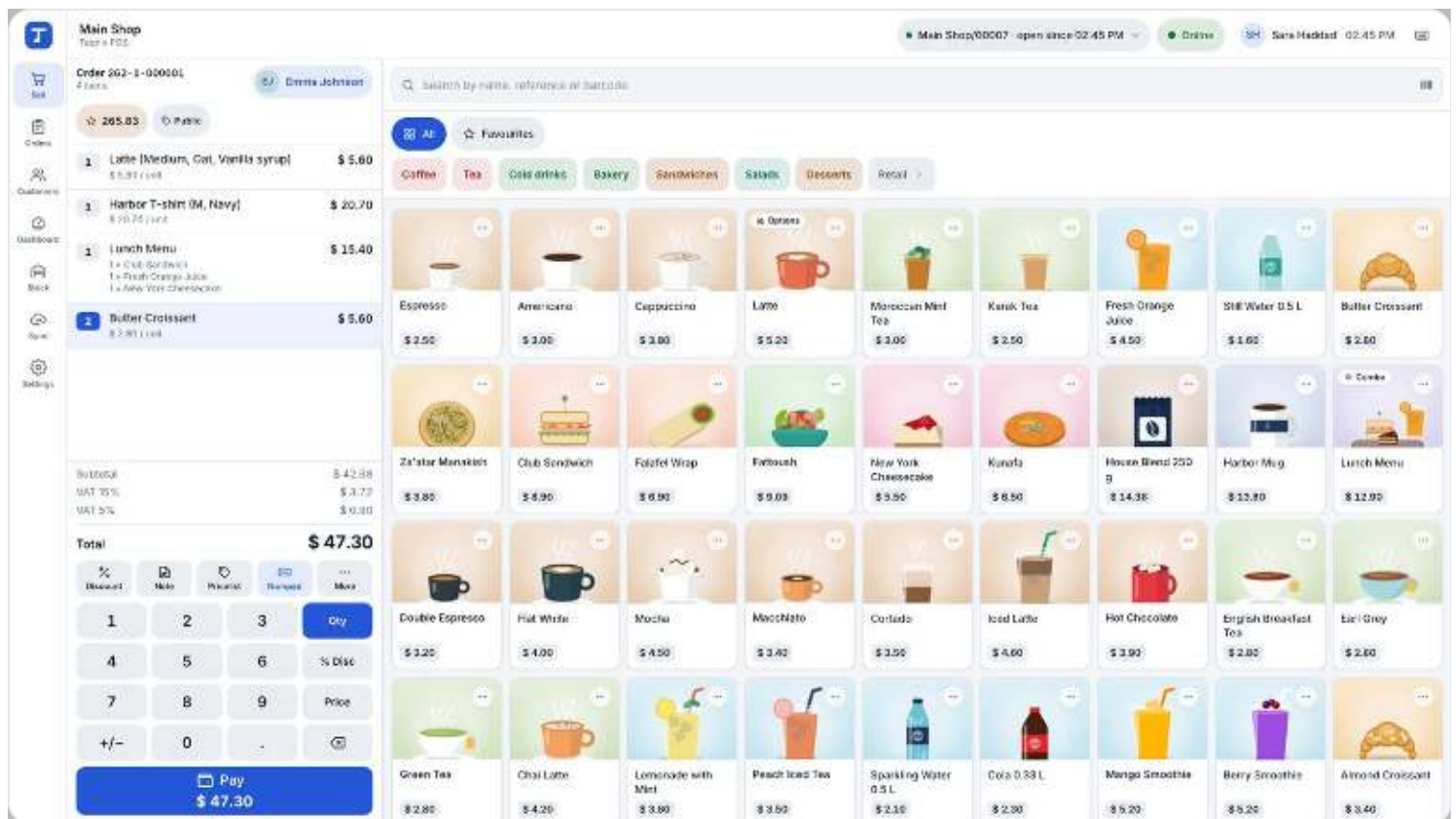


Main Shop with an empty order: rail, top bar, search, category chips and tiles; Latte has an Options badge, Lunch Menu a Combo badge.

The frame. The rail has **Sell**, **Tables** (restaurant only), **Orders**, **Customers**, **Dashboard**, **Stock**, **Sync** and **Settings**. The top bar shows the session chip (a menu with **Sync** and **Close session**), the connection chip (**Online**, **Syncing n orders**, **n sync errors**, **Offline**, **Offline · n orders waiting**; it opens the Sync screen), the cashier, the keyboard-shortcuts button and, with a lock PIN, **Lock the terminal**.

The catalogue. The search box (**Search by name, reference or barcode**) ignores capitals and accents. **All**, **Favourites** and the category chips filter the tiles. **Options** marks a product with attributes, **Combo** a combo. One tap adds the product; the same product again adds to the quantity of its line, as in Odoo. The ... button on a tile (or a long press, or a right click) opens its choices. The grid shows at most 1,200 tiles; then it asks you to search or pick a category.

Button	What it does
Customer (F4)	Choose or create the customer; their pricelist and fiscal position apply (chapter 10).
Discount	A percentage on the selected line, or on the whole order when Odoo's Discount app is installed and a discount product is set. Quick values 5, 10, 15, 20 % or any value from 0 to 100.
Note	A note for the customer (printed on the receipt) and an internal note.
Pricelist	Appears only when the point of sale has more than one pricelist.
Numpad	Show or hide it; Settings → Terminal can force it always on or hidden. The modes Qty , % Disc and Price decide what the keys change on the selected line.
More	Hold order , New order , Refund , Cancel order , Orders on hold , and the actions of optional parts: Layaways , Rewards & promo code , and in a restaurant Split bill , Bill , Ticket preview , Transfer .
Pay	Shows the total and opens the payment screen (chapter 11).



An order for Emma Johnson: four lines (the Lunch Menu with its three items), VAT 15% 3.72, VAT 5% 0.90, total 47.30.

The terminal numbers orders itself (for example 262-1-000001); Odoo gives its own name to the order when it receives it (Main Shop - 000001). The chip 265.83 is the loyalty balance of the customer, **Public** the pricelist.

Scanner and keyboard. A scanner that works like a keyboard (it types the code fast and ends with Enter) can be used anywhere on the Sell screen, not only in the search box. The terminal reads codes with Odoo's default barcode rules: product barcodes add the product, EAN-13 codes starting with 23 carry a price, with 21 a weight, codes starting with 22 a discount for the selected line, and 042 selects a customer. **Settings → Terminal** switches the scanner off and sets the maximum time between characters (35 ms by default) and the shortest barcode (6 characters). Scanners were not verified on real hardware.

Key	Action
F2 or /	Search products
F4	Choose a customer
F8 or Ctrl+Enter	Go to payment
F9	New order
Q, D, P	Numpad mode: quantity, discount, price
↑ ↓ and Del	Select a line; remove the selected line
Enter, Esc, ?	Validate the payment; close or go back; show this list

When a manager PIN is set, the terminal asks for it before the actions you switched on in **Settings → Security**: discounts, price changes, refunds, removing a line, cancelling an order (chapter 25).

Tablets. In landscape (1024 × 768) the two panels stay. In portrait (768 × 1024) the order collapses into a bar, **Review the order**, with the item count, the total and the **Pay** button; tap the bar to open the order.

Main Shop

Taqnix POS

Main Shop/00007 · open since 02:45 PM

Online

02:45 PM

Sell

Orders

Customers

Dashboard

Stock

Sync

Settings

Order 262-1-000001

4 items

EJ Emma Johnson

265.83

Public

1 Latte (Medium, Oat, Vanilla syrup)

\$ 5.60 / unit

\$ 5.60

1 Harbor T-shirt (M, Navy)

\$ 20.70 / unit

\$ 20.70

1 Lunch Menu

1 × Club Sandwich

1 × Fresh Orange Juice

1 × New York Cheesecake

\$ 15.40

2 Butter Croissant

\$ 2.80 / unit

\$ 5.60

Subtotal

\$ 42.68

VAT 15%

\$ 3.72

VAT 5%

\$ 0.90

Total

\$ 47.30

Discount

Note

Pricelist

Numpad

More

Pay

\$ 47.30

Search by name, reference or barcode

All

Favourites

Coffee

Tea

Cold drinks

Bakery

Sandwiches

Salads

Espresso

\$ 2.50

Americano

\$ 3.00

Cappuccino

\$ 3.80

Options

Latte

\$ 5.20

Moroccan Mint Tea

\$ 3.00

Karak Tea

\$ 2.50

The same order on a tablet in landscape (1024 × 768).



Main Shop

Taqnix POS

Session

Online

02:45 PM



Sell



Orders



Customers



Dashboard



Stock



Sync



Settings

Search by name, reference or barcode



All

Favourites

Coffee

Tea

Cold drinks

Bakery

Sandwiches

Salads

Desserts

Rel



Espresso

\$ 2.50



Americano

\$ 3.00



Cappuccino

\$ 3.80



Latte

\$ 5.20



Moroccan Mint Tea

\$ 3.00



Karak Tea

\$ 2.50



Fresh Orange Juice

\$ 4.50



Still Water 0.5 L

\$ 1.60



Butter Croissant

\$ 2.80



Za'atar Manakish

\$ 3.80



Club Sandwich

\$ 8.90



Falafel Wrap

\$ 6.90



Review the order
4 items

\$ 47.30

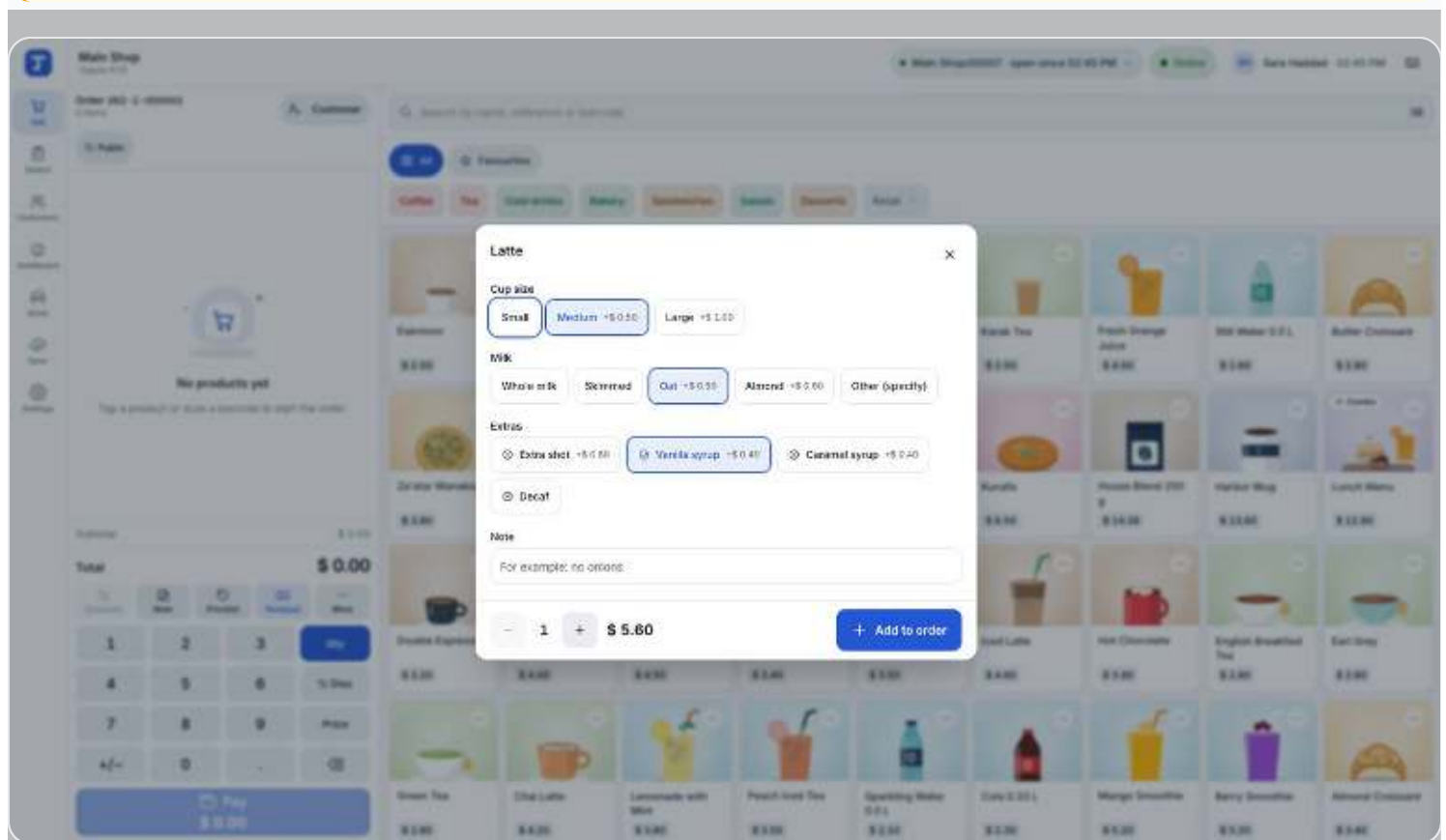


Pay

A tablet in portrait (768 × 1024): the bar “Review the order”, 4 items, 47.30, and Pay.

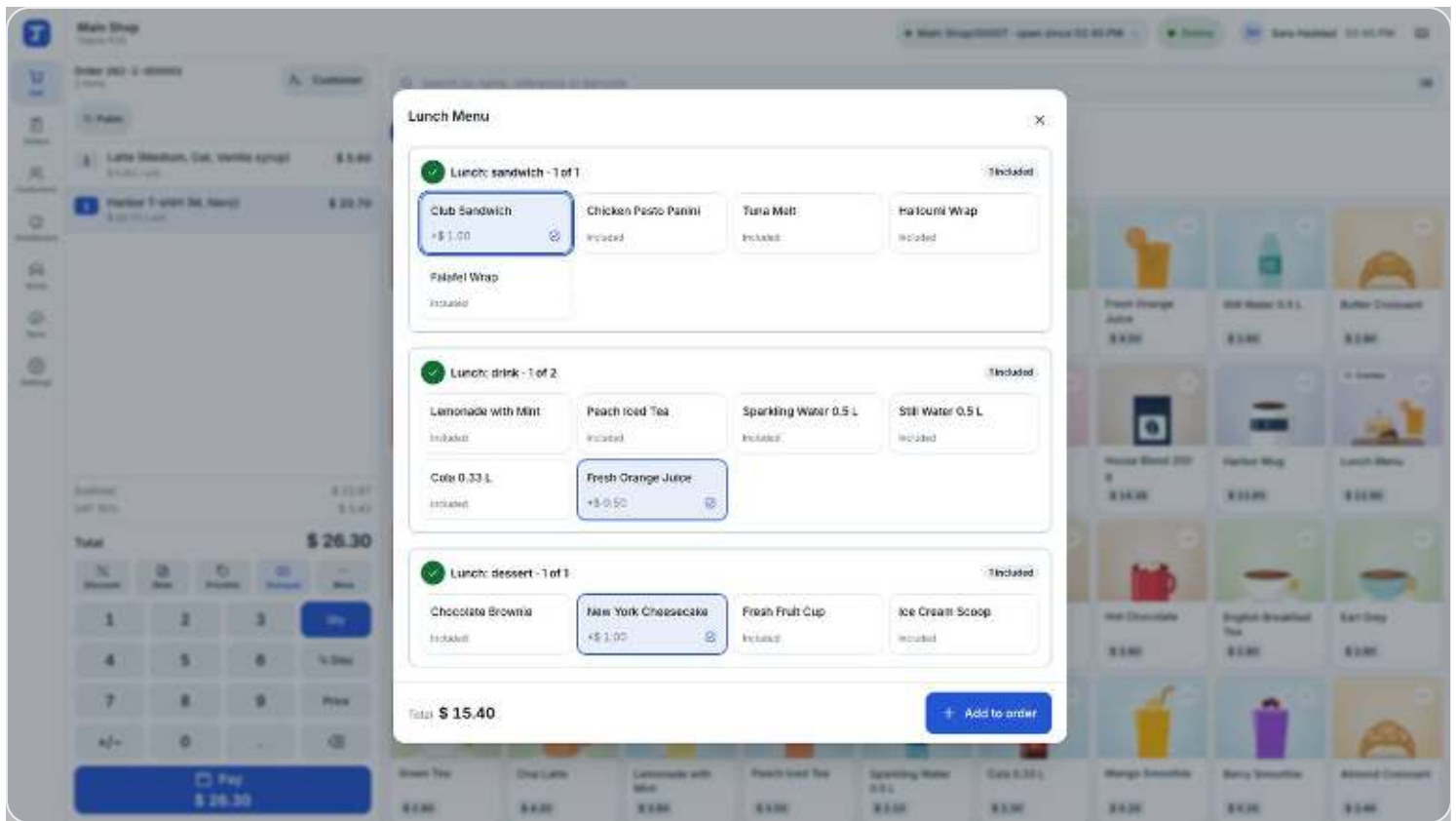
9. Variants and combos

In plain words: A product with an **Options** badge or a **Combo** badge opens a window where you choose; the price at the bottom changes as you choose. The result goes to the order as one line, a combo with its items under it.



The Latte window: Cup size, Milk, Extras (Medium, Oat and Vanilla syrup chosen), Note, quantity, price 5.60, Add to order.

Variants. Each attribute is a group: pills, a drop-down, colour swatches, or extras you can tick several of, as Odoo displays them. The extra price is written on the value. Choices that cannot be combined are greyed and struck through, and a missing combination says “This combination is not available.” A custom value such as **Other (specify)** asks for a text; Odoo accepts it only from a user with **Products / Create** (chapter 5).

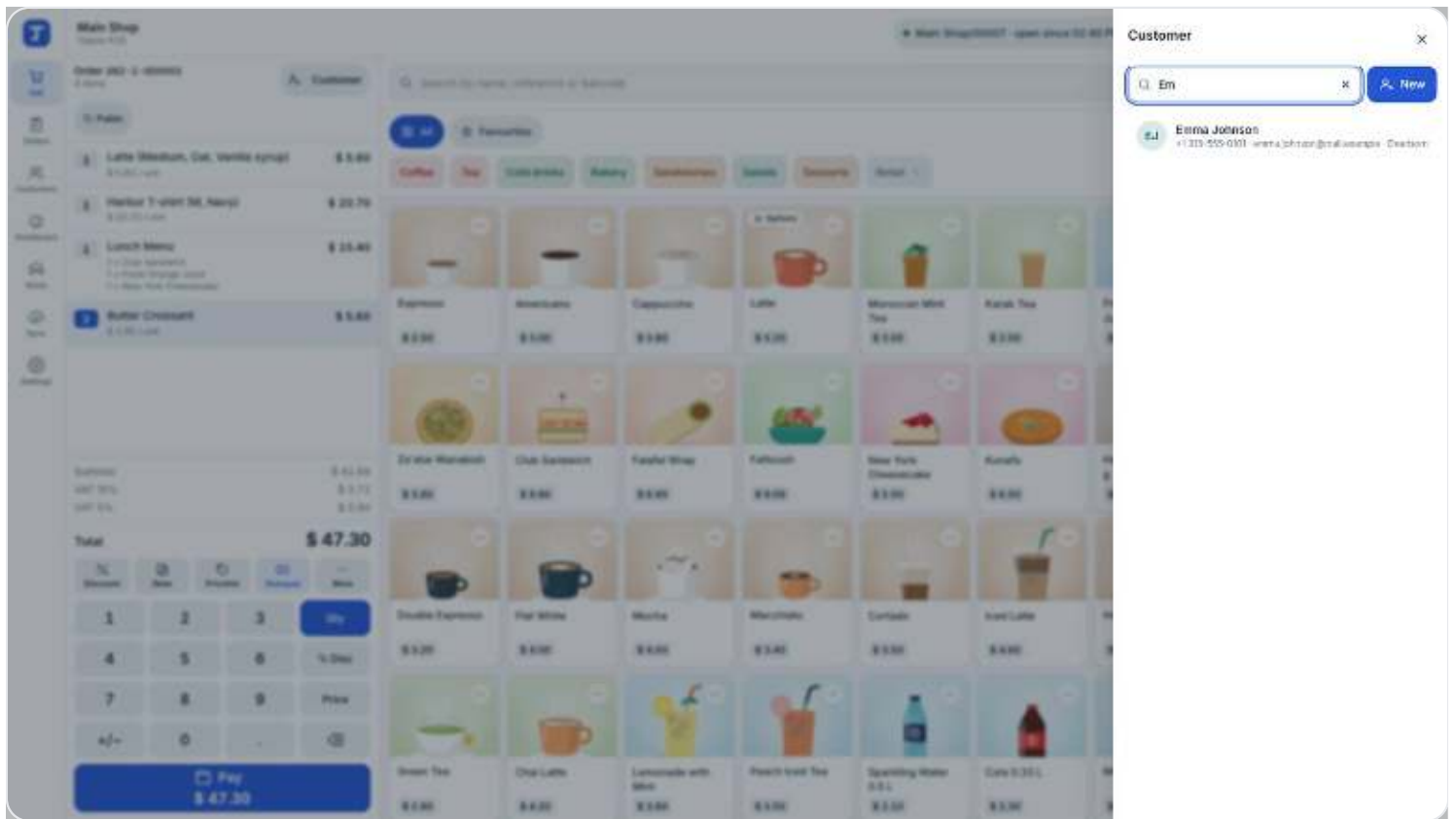


The Lunch Menu window: three groups ("1 of 1", "1 of 2", "1 of 1"), cards Included or with an extra price; total 15.40.

Combos. A combo is made of groups. A group title shows how many items you picked out of the maximum ("1 of 2"), and a badge how many are included in the combo price. A group with one item is chosen for you. **Add to order** works only when every group is complete. The price of a combo item cannot be changed alone. In a restaurant each item goes to the printer of its own category.

10. Customers

In plain words: The customers are kept on the device, so you can search and choose one offline. Choosing a customer applies their pricelist and fiscal position, as Odoo does.



The customer panel (F4) with "Em" typed: Emma Johnson, and the New button.

Press **Customer** on the Sell screen. The search looks at the name, the phone (any format), the email and the barcode; **New** creates a customer without leaving the order.

Main Shop
Test a POS

Customers
34 customers on this device

Q Name, phone, email or barcode

Customers

Administrator
+1 313-555-0107 - admin@example

Ava Davis
+1 313-555-0107 - ava.davis@example

Chris Dubois
+1 313-555-0107 - chris.dubois@example

Emma Johnson
+1 313-555-0101 - emma.johnson@example

Ellen Thomas
+1 313-555-0110 - ellen.thomas@example

Grace Lee
+1 734-555-0123 - grace.lee@example

Greenleaf Grocers
+1 734-555-0105 - orders@greenleaf.org

Harbor Cafe & Market
+1 313-555-0102 - hello@harbor-cafe.mkt...

Hassan Farouk
+1 313-555-0110 - hassan.farouk@example

James Miller
+1 313-555-0106 - james.miller@example

Emma Johnson

Use for the current order Edit

The Close user of this terminal may not change customers (needs "Contacts / Creation").

PHONE
+1 313-555-0101

EMAIL
emma.johnson@example

ADDRESS
585 Michigan Ave, 48126 Dearborn, United States

TAX ID
-

PRICELIST
Public (USD)

RECENT ORDERS ON THIS DEVICE
Main Shop - 000001

15/10/2026 \$ 47.30

Customers: 34 on this device, the A to Z list, the card of Emma Johnson with Use for the current order and a greyed Edit.

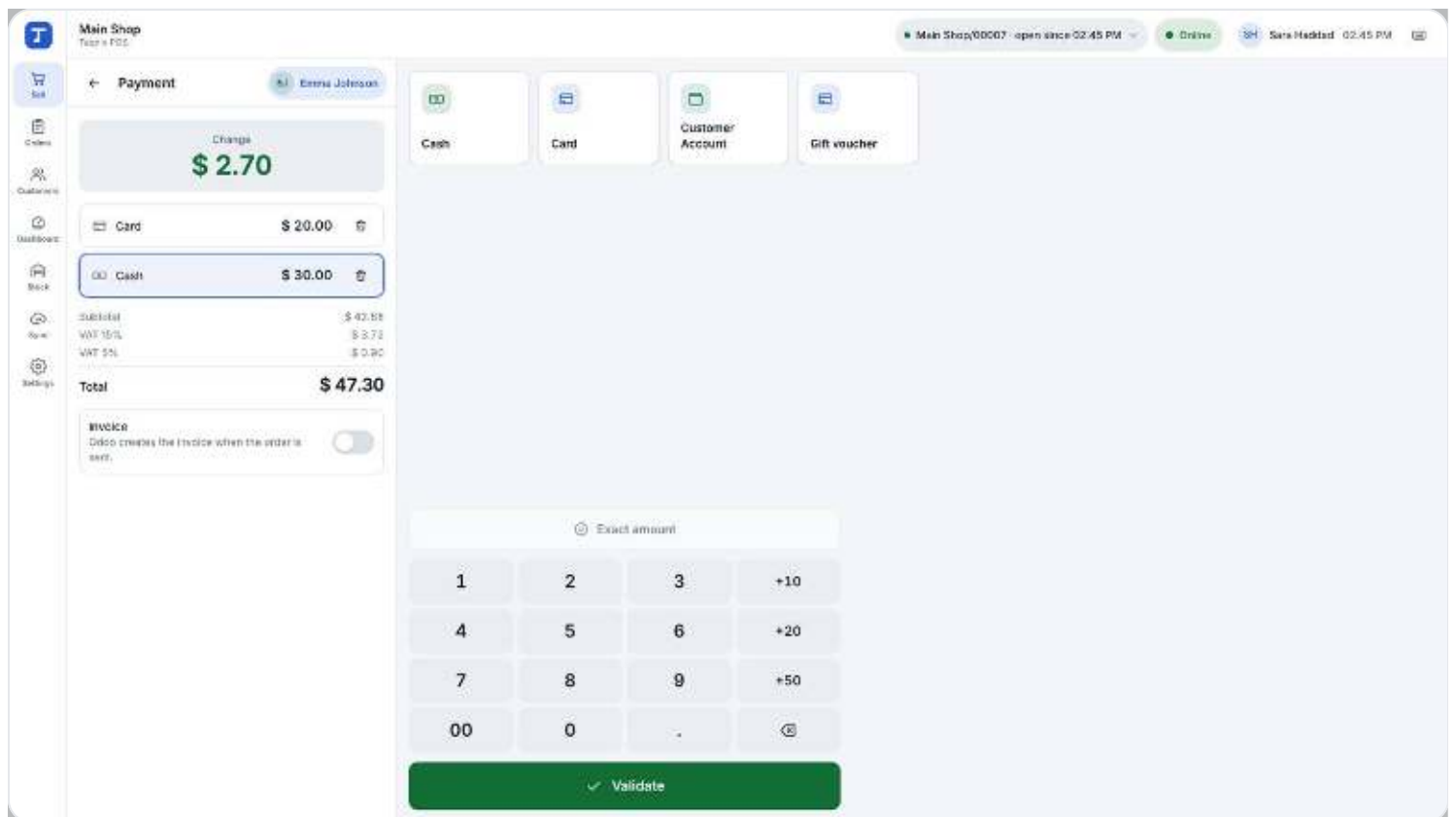
The card shows the contact data, the pricelist, the fiscal position and the recent orders on this device. **Use for the current order** sets the customer on the open order. **Edit** needs the connection and the **Contacts / Creation** right; here it is greyed and the card says why.

The New customer form, with a red box: this user cannot create customers in Odoo.

New customer works offline. The customer is saved on the device with the tag **Will sync**, can be used in orders at once, and is sent to Odoo before the orders that use it. Odoo 19 lets only users with **Contacts / Creation** create customers: without it the form shows the red box, Odoo refuses the customer, and the orders waiting for it stay in the Sync screen until the right is granted and you press **Retry**. When not every customer is on the device, **Search in Odoo** looks on the server (it needs the connection) and keeps what it finds.

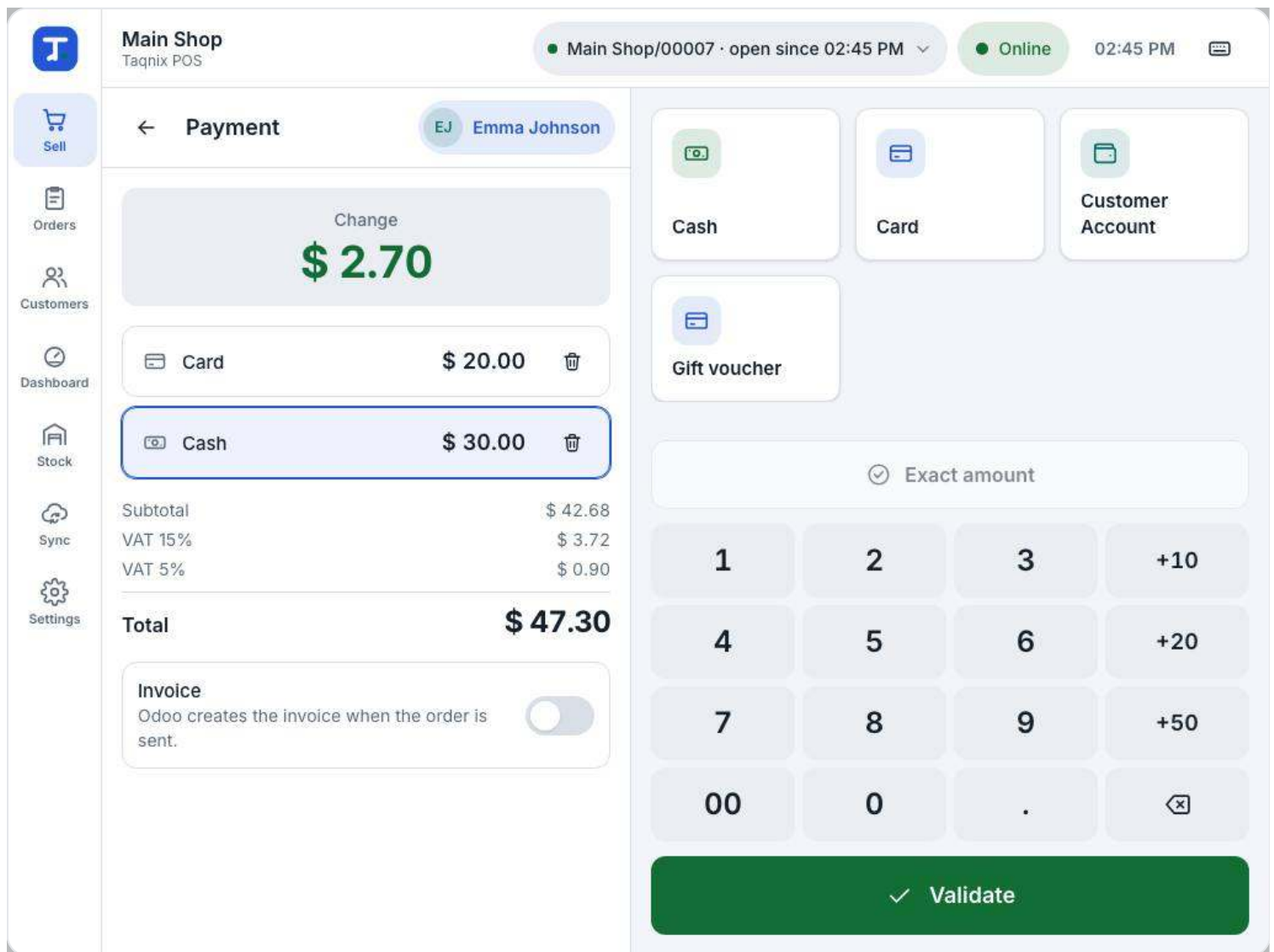
11. Payment, receipt and printing

In plain words: Press **Pay**, choose one or more payment methods, enter the amounts and press **Validate**. The terminal saves the order on the device first, then shows the receipt, then sends the order to Odoo.



Payment of 47.30: Card 20.00 and Cash 30.00, Change 2.70, the Invoice switch, method tiles, Exact amount, the keypad and Validate.

Part	How it works
Method tiles	One for each payment method of the point of sale. A tap adds a line with the amount due; tap a line to select it and type its amount; the trash button removes it.
Exact amount, +10, +20, +50	Exact amount sets the selected line to what is due. The +keys add that amount to the selected line.
Remaining, Change	The grey box shows what is due in amber, then the change in green. Change can only be given back in cash.
Cash rounding	When the point of sale rounds cash (Main Shop: 0.05, cash only), a cash amount must be a multiple of it; the screen shows the rounding.
Customer Account, Invoice	Both need a customer. Odoo creates the invoice when it receives the order, also if the sale was made offline.
Lot and serial numbers	For a tracked product the terminal asks for the numbers when you add it. The cashier needs Inventory / User in Odoo (chapter 5).
Keyboard	Digits edit the selected payment, Enter validates, Esc goes back.



The same screen on a tablet in landscape (1024 × 768).

The receipt shows what will print: the company block, receipt and order numbers, the customer, the lines with a tax letter (A, C), the subtotal and the taxes by group with their base ("VAT 15% (A) on 24.79"), the payments and the change, the footer text of the point of sale, a Code-128 barcode of the receipt number, and a QR code "Scan to get your invoice" with the address of Odoo's portal and a code (when Odoo's ticket QR option is on). The right side has a badge (**On this device, Waiting to sync, Sending to Odoo..., In Odoo**), the switch **58 mm / 80 mm, Print** and **New order**.

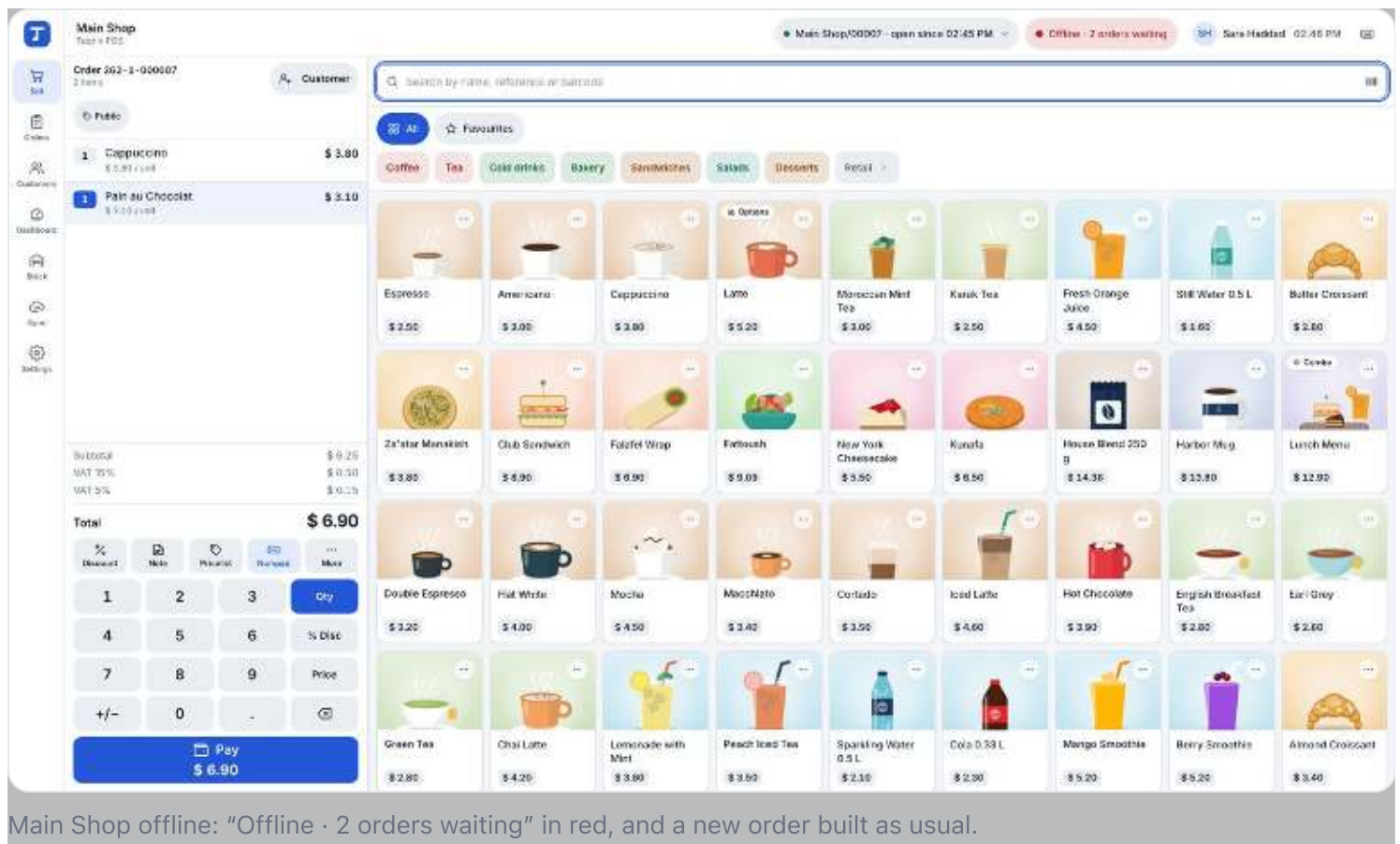


Receipt 262-1-000001, 80 mm: tax letters A and C, VAT 15% (A) on 24.79 = 3.72, VAT 5% (C) on 17.89 = 0.90, total 47.30, change 2.70, barcode, QR; on the right In Odoo.

Printing. Print uses the print function of the browser and prints only the receipt. Choose the printer once in the dialog, margins None, paper size the width of the roll. Chrome prints without the dialog only when started with `--kiosk-printing`. **Settings → Terminal** sets the paper width and **Print the receipt automatically**. There is no direct ESC/POS printing (chapter 27). Old receipts are reprinted from Orders (chapter 13).

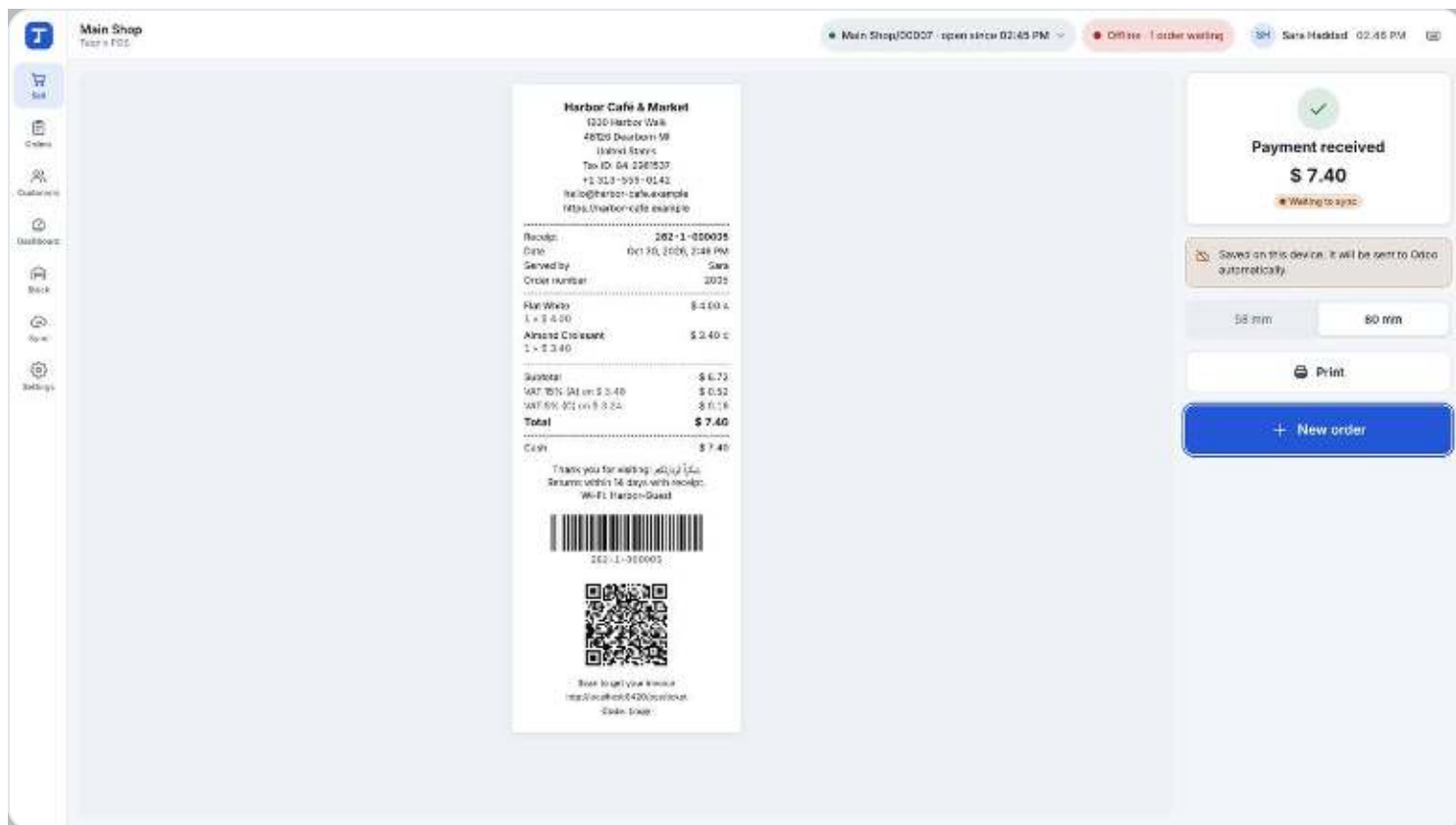
12. Working offline

In plain words: When the network or Odoo is gone you keep selling. The red chip says **Offline • n orders waiting**. Orders wait on the device and are sent, one at a time, when Odoo answers again. Nothing is lost on the way.



Main Shop offline: "Offline · 2 orders waiting" in red, and a new order built as usual.

What is on the device. A copy of what the terminal needs, in the browser database (IndexedDB): products, variants and combos, customers, taxes and fiscal positions, pricelists, loyalty programmes, the configuration of the point of sale and your open orders. After the first full download the terminal starts from this copy.



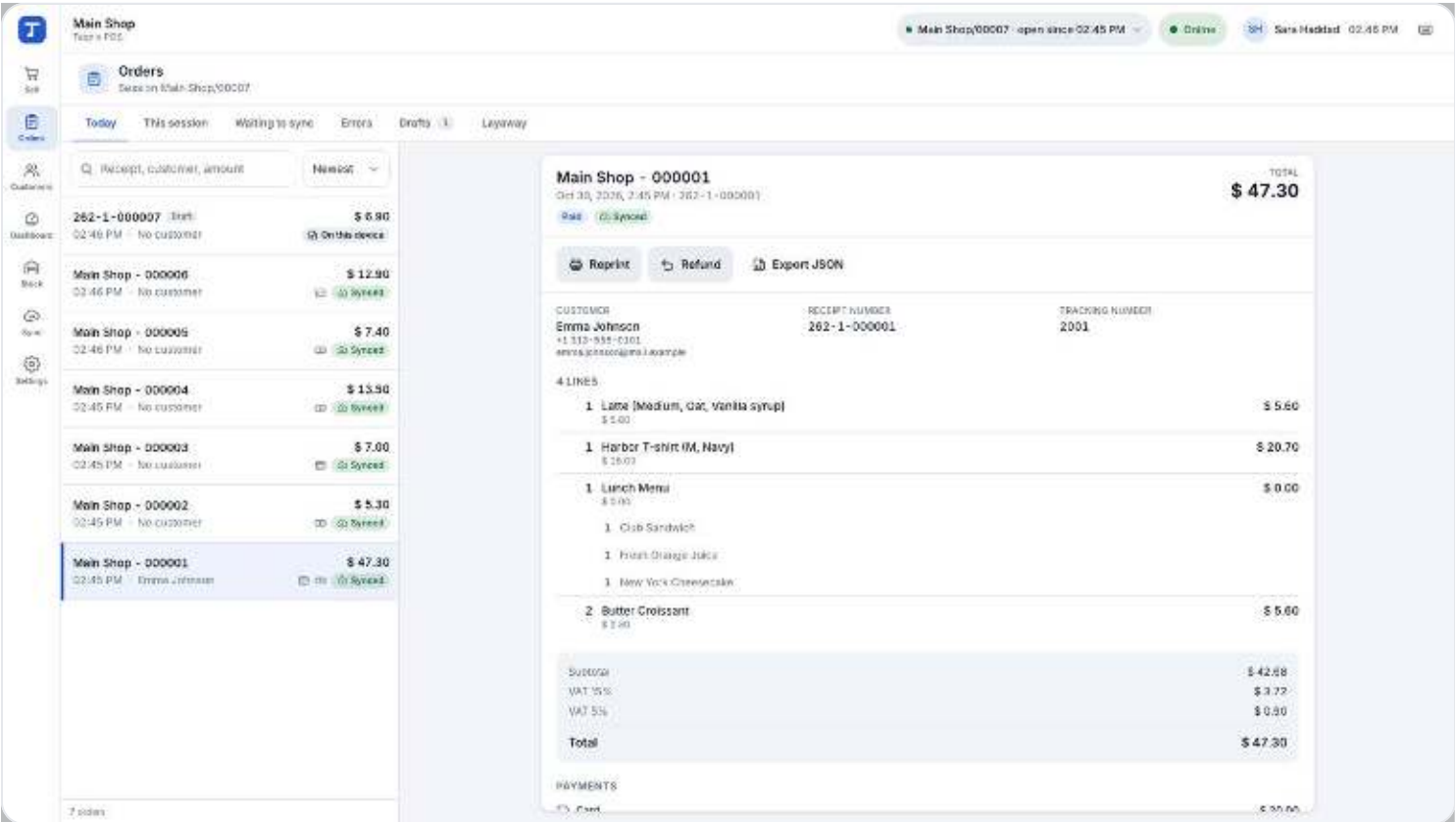
An order made offline (cash 7.40): badge Waiting to sync and "Saved on this device. It will be sent to Odoo automatically."

Step	What the terminal does
Validate	The paid order is written to the device before the receipt shows, then queued.
The queue	One order at a time, oldest first. Each order has a unique id that Odoo recognises, so an answer lost on the way is no problem: the order is sent again and not created twice.
Network errors	Retried after 5 s, 15 s, 45 s, 2 min, 5 min, then every 15 minutes. A new session or a reconnection resets the waiting.
Refused by Odoo	A business error (a missing right, a validation) is not retried by itself. The order stays in Errors with the reason until you fix the cause and press Retry sync .
Waking up	While offline the terminal asks Odoo with a small request (no password) about every 30 seconds and sends as soon as Odoo answers. Chromium also wakes the open page through Background Sync.
Catalogue changes	Small updates every 5 minutes (a setting) and on Pull data now ; a full download when a point of sale is chosen. A tax changed after a sale does not alter it: the order is sent as recorded and the event is logged. A customer created offline is sent first.
Starting offline	The first visit stores the whole app in the browser, so the terminal starts with the server stopped. Two tabs share one database; only one sends.

Important: What can go wrong. (1) If the browser data is cleared, or the browser deletes it to free space, the orders not yet sent are gone; the Sync screen warns and offers **Keep data on this device**. (2) If the session was closed in Odoo meanwhile, Odoo puts the orders in the open session of the same point of sale, or they wait until one is opened. (3) A missing right keeps an order in **Errors**. Chapter 28 lists the messages. Measured speed: chapter 26.

13. Orders

In plain words: The Orders screen lists the sales of this terminal, also those Odoo has not received. Reprint a receipt, refund an order, retry a refused order, export it.

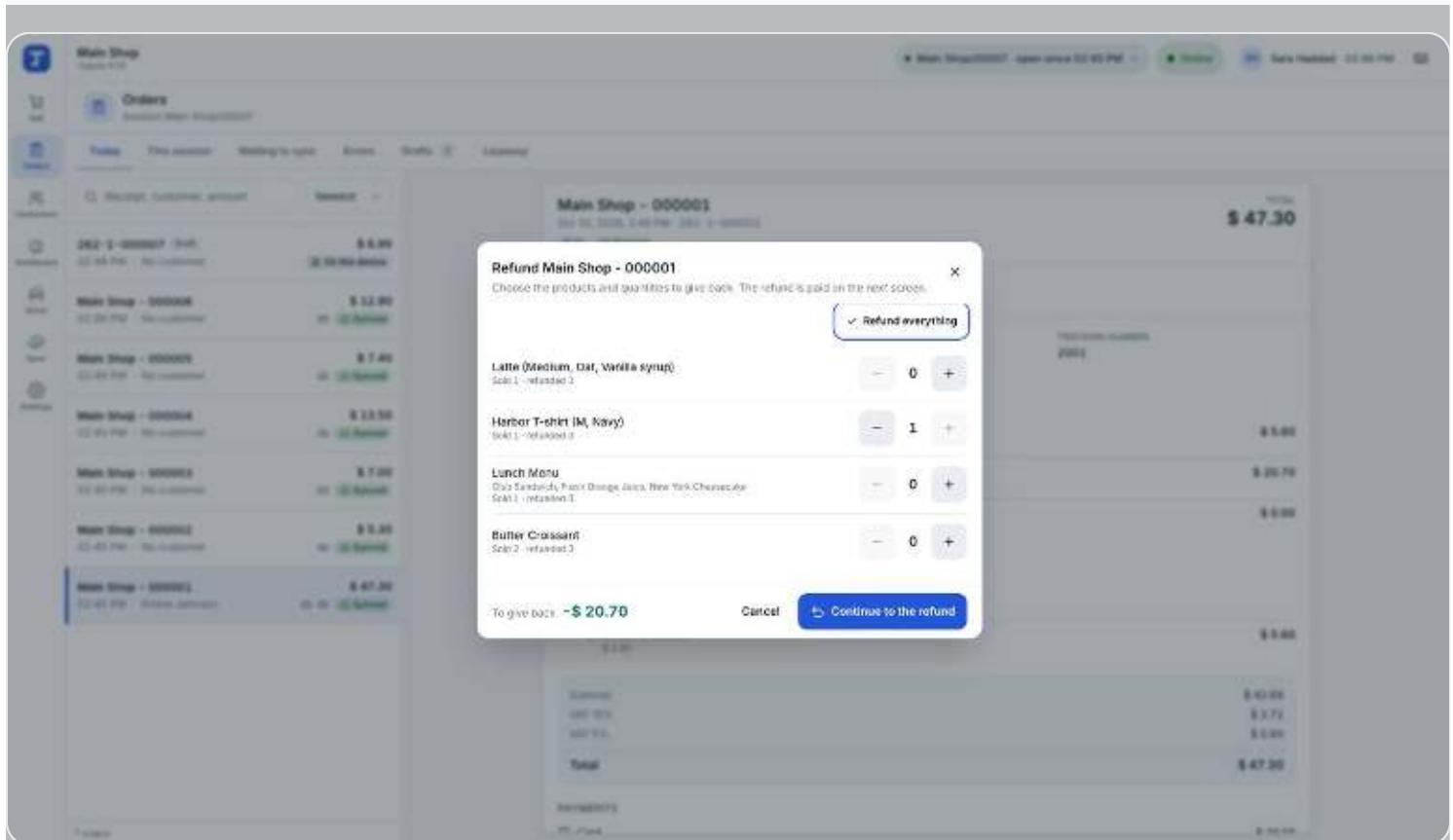


Orders of Main Shop: six tabs, 7 orders, the detail of Main Shop - 000001 (47.30, Paid, Synced) with Reprint, Refund and Export JSON.

Tab	Contents
Today, This session	The sales of the day on this terminal; the orders of the open session.
Waiting to sync, Errors	Orders Odoo has not confirmed; orders Odoo refused, with the reason.
Drafts, Layaway	Held orders and open table orders; orders kept open with a deposit (chapter 18).

Search by receipt number, customer or amount; sort by newest, oldest, largest or smallest. The detail shows the state badges, lines, totals, payments and the **Sync history**.

Button	What it does
Reprint	Opens the receipt again, marked as a copy, and prints it through the browser.
Refund	Only for an order Odoo has confirmed; an unsent order can be refunded once it is sent.
Retry sync	Sends an order of the Errors tab again after you fixed the cause.
Export JSON	Saves the order as a file, for support or to enter it in Odoo by hand.
Discard	Only for an order Odoo refused. The terminal downloads its JSON first, then asks you to type DISCARD and a reason. If the customer paid, settle the money outside the terminal.

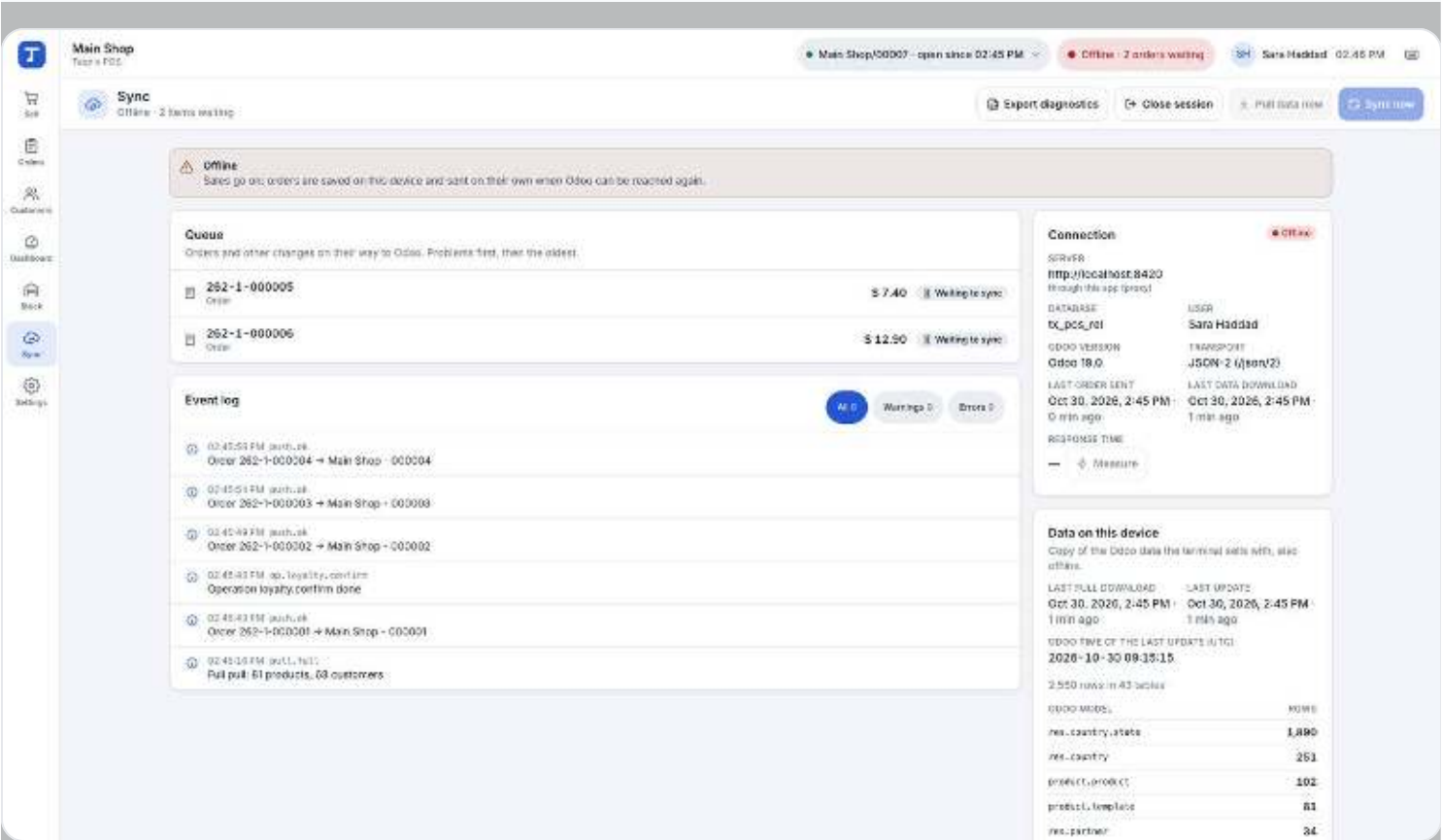


The Refund window for Main Shop - 000001: one T-shirt selected, "To give back -20.70", Continue to the refund.

Refund. Choose the lines and quantities (the window shows what was sold and already refunded) or **Refund everything**, then **Continue to the refund**. The payment screen opens with negative amounts (**Refund to customer**); pay it out and validate. The terminal counts the refunds already made on this device, so you cannot refund more than what is left. A refund is an order of its own, sent like any other. A manager PIN is asked first if it protects refunds.

14. The Sync screen

In plain words: The Sync screen shows what is on its way to Odoo, what the terminal knows about the connection and what is stored on the device. Open it from the rail or by tapping the connection chip.



Sync offline ("Offline - 2 items waiting"): Queue with 262-1-000005 (7.40) and 262-1-000006 (12.90), Event log, Connection, Data on this device (2,560 rows in 43 tables).

Main Shop
Test a POS

Main Shop/00007 - open since 02:45 PM
Online
Sara Haddad 02:45 PM

Sync
Everything is in Odoo

Export diagnostics
Close session
Pull data now
Sync now

Queue

Orders and other changes are their way to Odoo. Problems first, then the oldest.

Everything is synced

No order or change is waiting for Odoo.

Event log

All

Warnings

Errors

02:40:20 PM push,ok

Order 262-1-000008 → Main Shop - 000008

02:40:19 PM push,ok

Order 262-1-000005 → Main Shop - 000005

02:40:09 PM push,ok

Order 262-1-000004 → Main Shop - 000004

02:45:44 PM push,ok

Order 262-1-000003 → Main Shop - 000003

02:45:42 PM push,ok

Order 262-1-000002 → Main Shop - 000002

02:45:43 PM op,logically,confirm

Operation logically confirm done

02:45:43 PM push,ok

Order 262-1-000001 → Main Shop - 000001

02:45:39 PM pull,full

Full pull: 61 products, 63 customers.

Connection

Online

SERVER:

http://localhost:8420

through this app (proof)

DATABASE:

cx_pos_fci

USER:

Sara Haddad

ODOO VERSION:

Odoo 18.0

TRANSPORT:

JSON-2.0 (xml2)

LAST ORDER SENT:

Oct 30, 2026, 2:46 PM

0 min ago

LAST DATA DOWNLOAD:

Oct 30, 2026, 2:45 PM

1 min ago

RESPONSE TIME:

Measure

Data on this device

Copy of the Odoo data the terminal sells with, also offline.

LAST PULL DOWNLOAD:

Oct 30, 2026, 2:45 PM

1 min ago

LAST UPDATE:

Oct 30, 2026, 2:45 PM

1 min ago

ODOO TIME OF THE LAST UPDATE (UTC)

2026-10-30 09:33:15

2,500 rows in 43 tables

ODOO MODELS

ROWS

res.country.state

1,890

res.country

251

product.product

102

product.template

61

res.partner

34

product.template

99

product.template.attribute.value

29

The same screen online: "Everything is in Odoo", an empty queue and enabled Pull data now and Sync now.

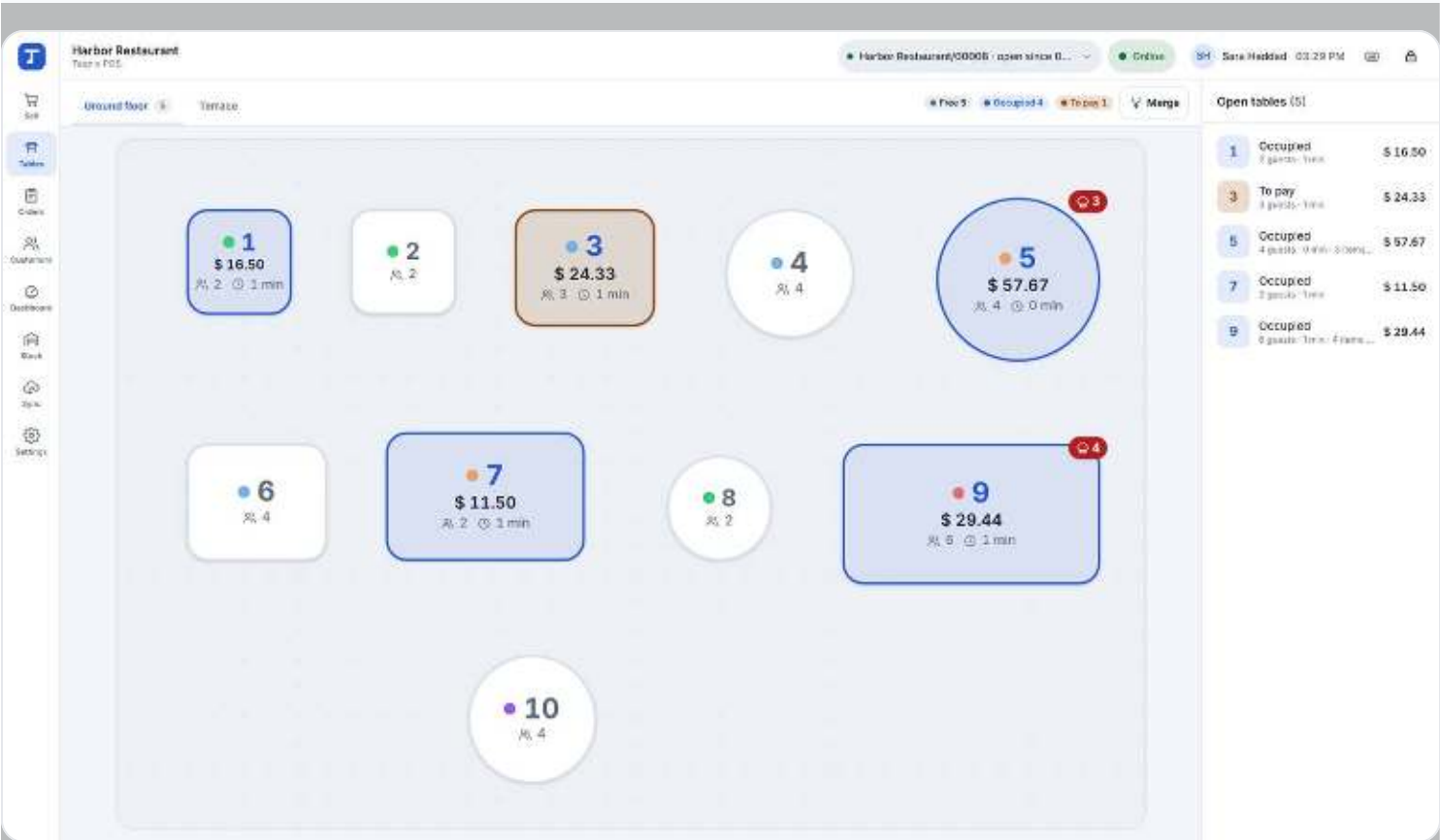
Card or button	What it shows or does
Queue	Orders and other changes (new customers, loyalty points, table orders to cancel) with state, attempts, next try and last error; problems first. Retry now, Retry all.
Event log	The last 500 events; filters All, Warnings, Errors.
Connection	Online or offline, server and how it is reached, database, user, Odoo version, transport, last order sent, last download. Measure times a small request to Odoo.
Data on this device	Last full download and last update, and the rows of each Odoo model. Re-download everything replaces products, customers, taxes and prices; unsent orders, waiting changes, the connection, the PIN and the settings stay.
Storage	Space used, and whether the browser promised to keep it. Keep data on this device asks for that promise.
Sync now, Pull data now	Send the queue; download catalogue changes. Both need the connection.
Export diagnostics	A JSON file for support: no passwords, keys, PINs, customer data or order lines.
Close session	Opens the closing flow (chapter 20).

15. Restaurant: tables, kitchen, split, tips

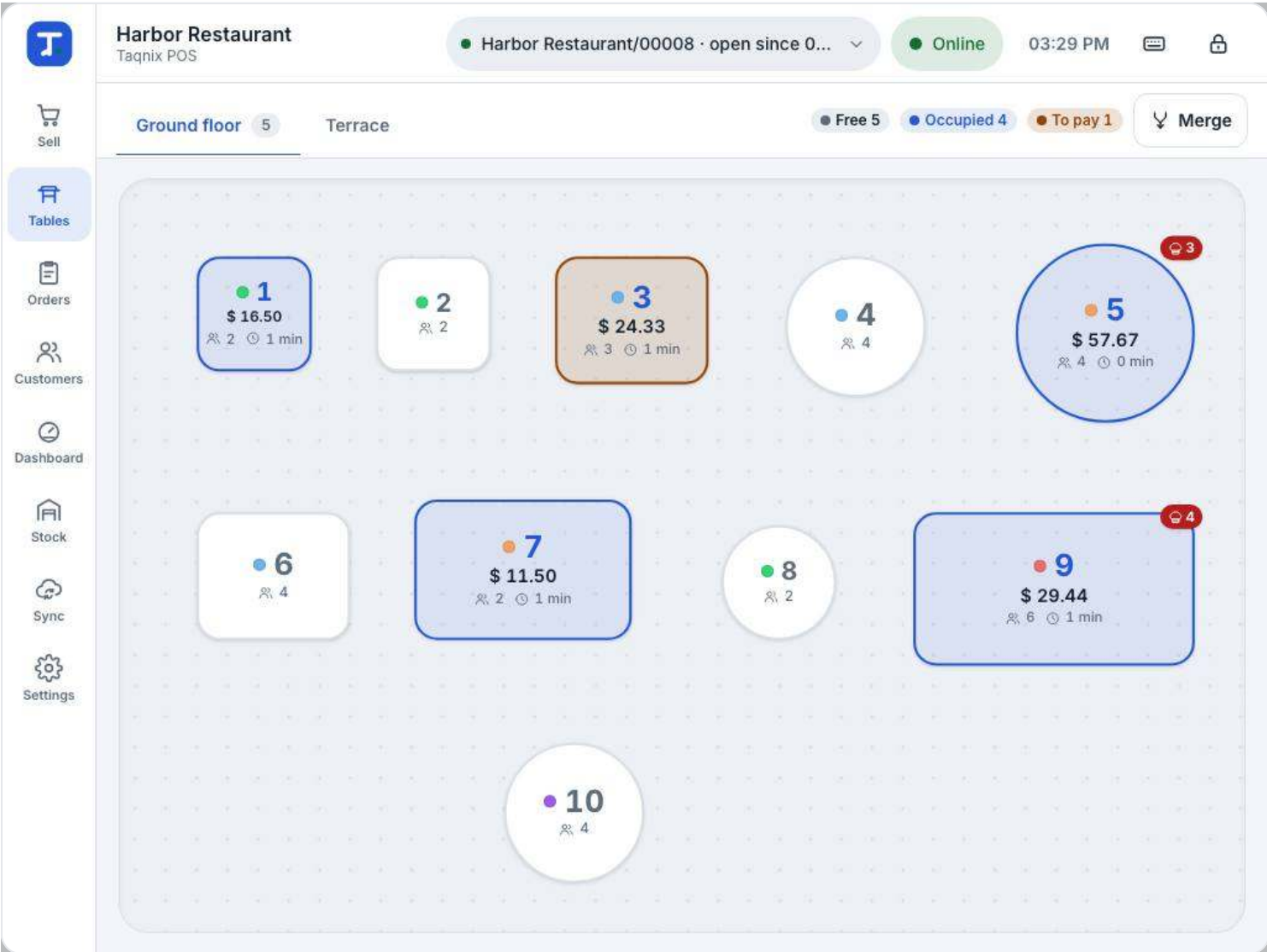
In plain words: These screens appear only when Odoo's **Restaurant** app is installed *and* the point of sale is a restaurant. Main Shop is retail: it shows no Tables entry, no table chip and no kitchen button. Harbor Restaurant shows all of it.

The floor plan. Tables opens it. Floors are tabs (**Ground floor, Terrace**). Tables sit at the position, shape and size they have in Odoo, scaled to the screen; the plan is never mirrored in Arabic. A table shows its number, its Odoo colour dot and, when busy, total, guests and minutes. White is free, blue

Occupied, amber **To pay** (bill printed, nothing changed since). A red chef-hat badge counts items not yet sent to the kitchen. **Merge** starts a merge; a wide screen also lists **Open tables**.



Harbor Restaurant, Ground floor: 5 free, 4 occupied, 1 to pay; busy tables carry a red badge for items not yet sent; Open tables (5) on the right.



The floor plan on a tablet in landscape (1024 × 768).



Sell



Tables



Orders



Customers



Dashboard



Stock



Sync



Settings

Ground floor 5

Terrace

Free 5

Occupied 4

To pay 1



Merge

1
\$ 16.50
2 1 min

2
2

3
\$ 24.33
3 1 min

4
4

5
\$ 57.67
4 0 min

6
4

7
\$ 11.50
2 1 min

8
2

9
\$ 29.44
6 1 min

10
4

The floor plan on a tablet in portrait (768 × 1024).

Open a table. Tap it: a free table starts an order with one guest, a busy one opens its order. The Sell screen shows a **table chip** and the kitchen button **Order** with the number of changes to send. The chip opens the table sheet: guests (1 to 99), the service (**Dine In, Takeout, Delivery**; it sets the pricelist and fiscal position), **Split bill, Bill, Transfer, Floor plan** (or **Unmerge**) and the kitchen section. Leaving a table sends its changes to Odoo.

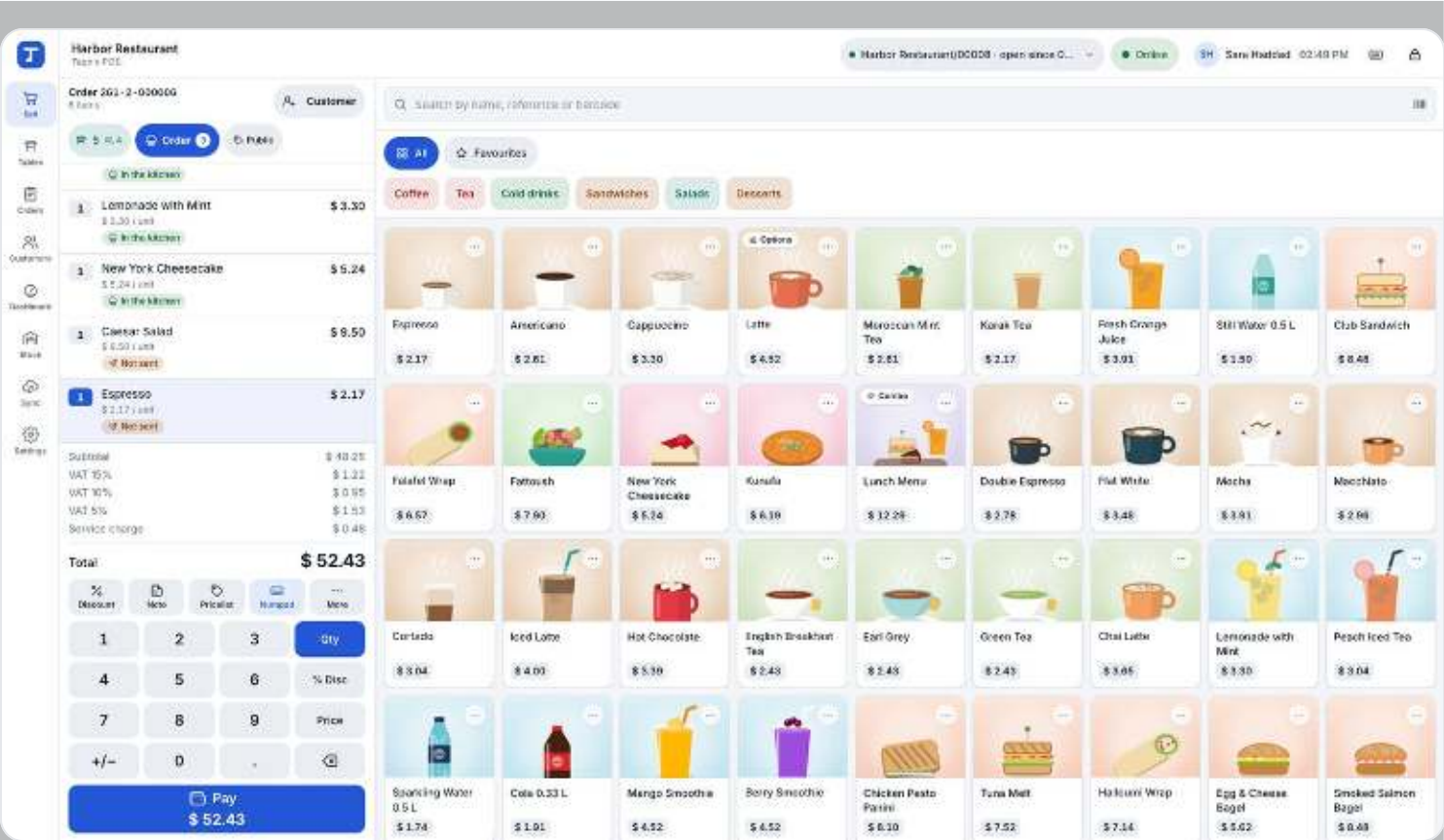
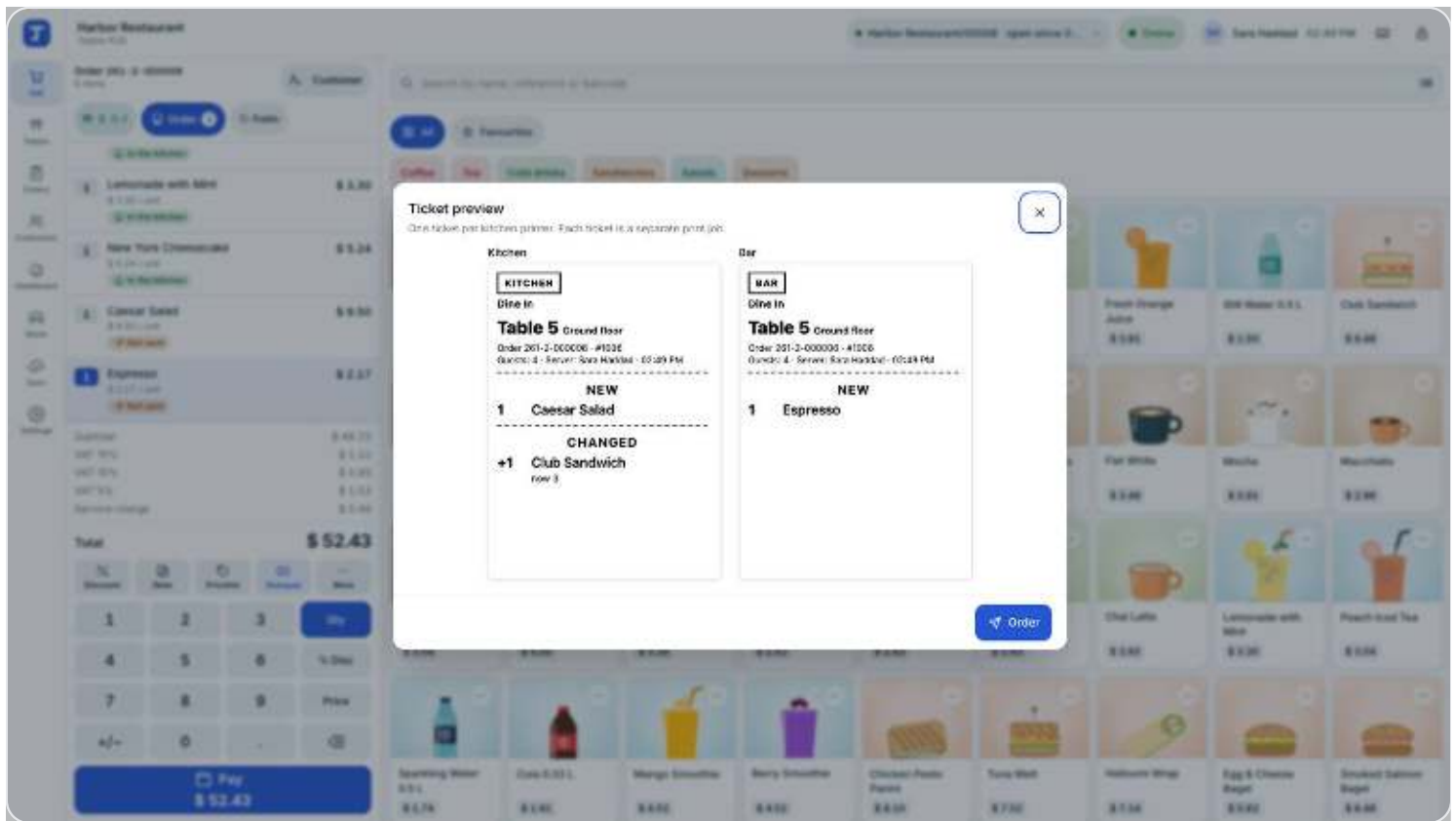


Table 5 (4 guests), order 261-2-000006: two lines In the kitchen, two Not sent; total 52.43 with a service charge; the Order button counts the changes to send.

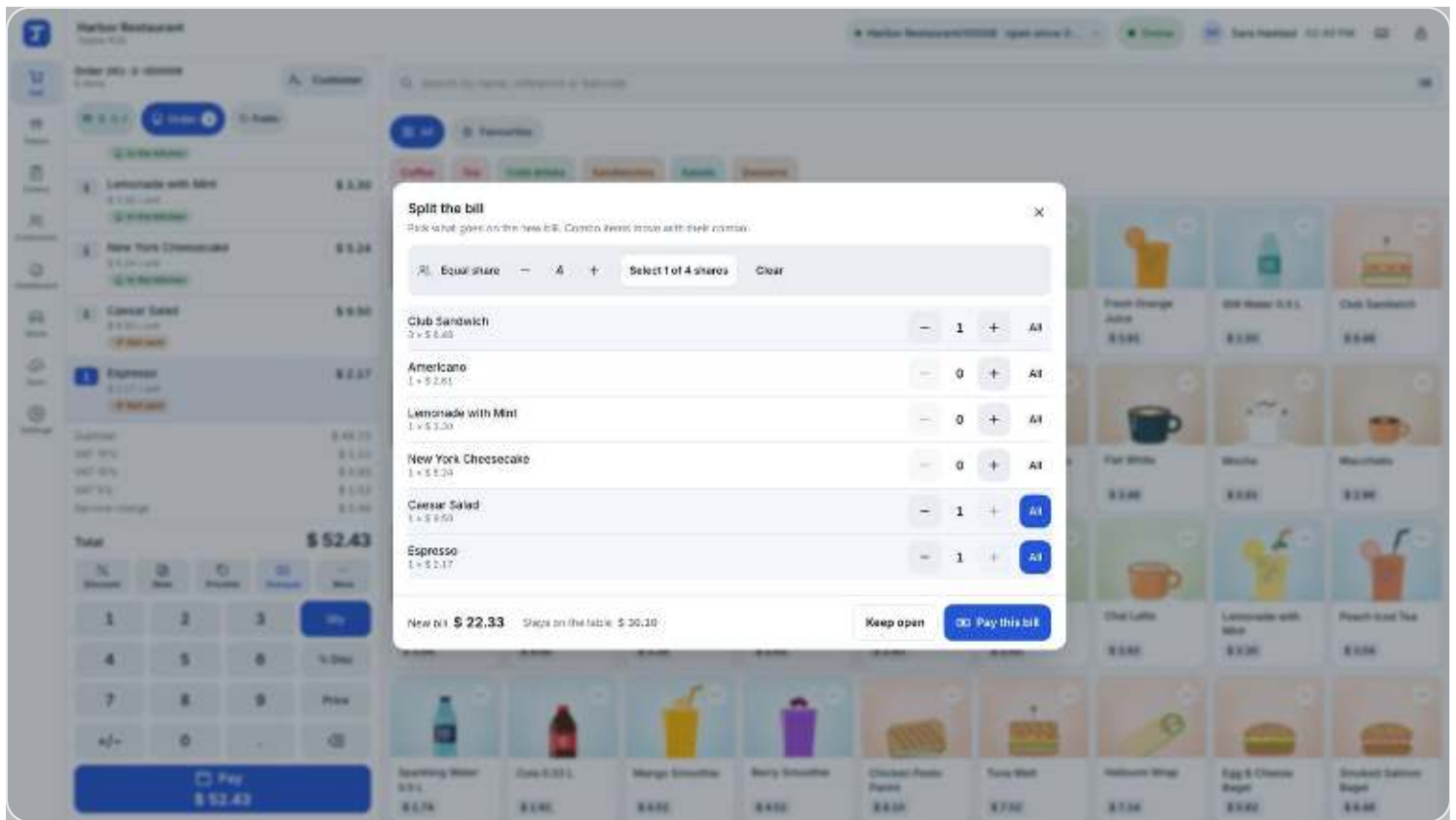
The kitchen. Each line shows **In the kitchen** or **Not sent**. **Order** sends only what changed since the last time, as Odoo's own point of sale does. A ticket has the sections **NEW**, **CHANGED** (with the new quantity) and **CANCELLED**, notes and options, and combo items under the combo name. There is one ticket for each kitchen printer of Odoo: a line goes to every printer whose product categories contain one of its categories or their parents. Items no printer takes print on the default station **Other items**. Tips, discounts and rewards never go to the kitchen. **Ticket preview** shows the tickets first; **Reprint** prints the last ones marked DUPLICATE.



Ticket preview of table 5: the Kitchen ticket (NEW 1 Caesar Salad, CHANGED +1 Club Sandwich now 3) and the Bar ticket (NEW 1 Espresso).

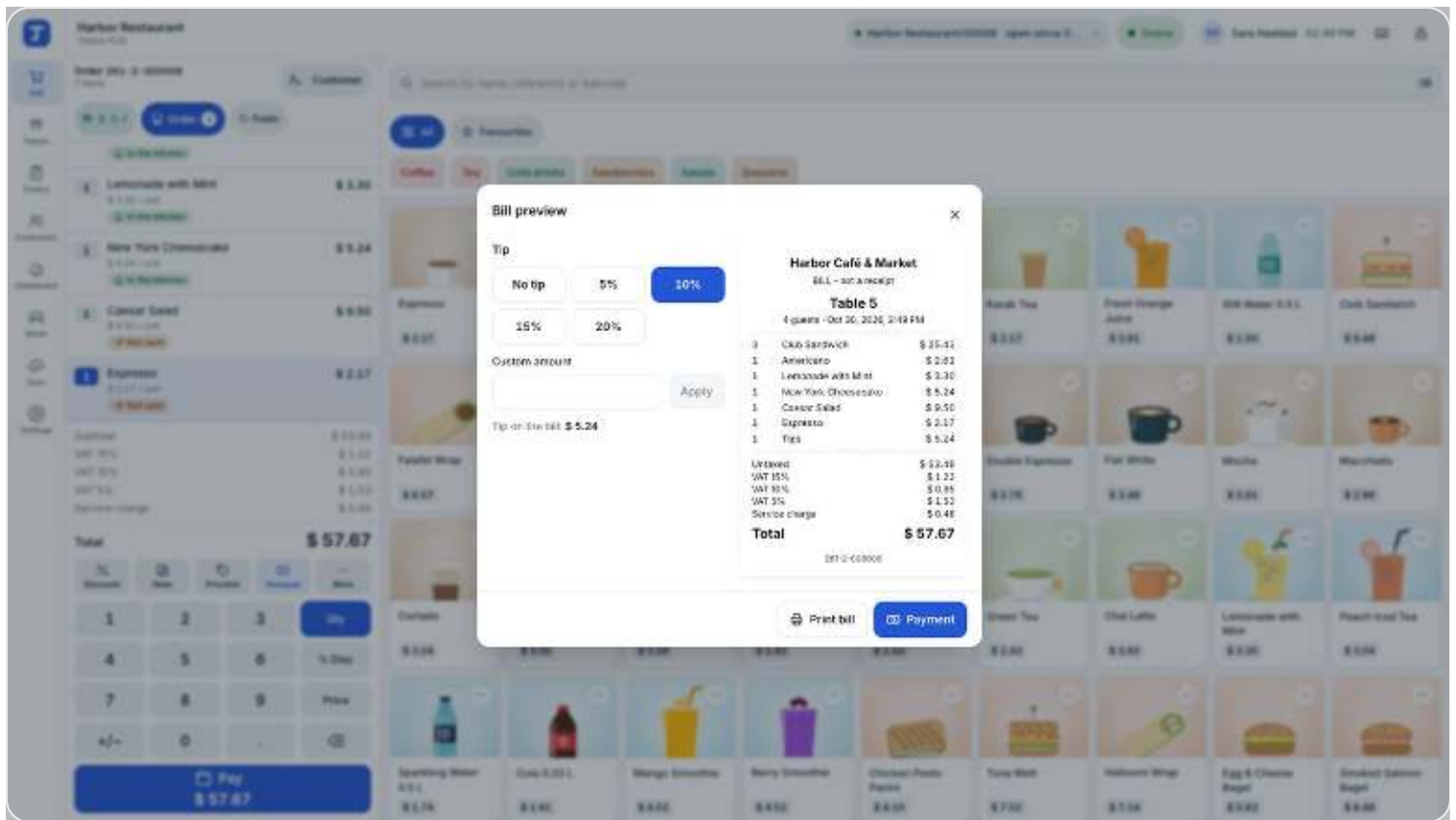
Important: Printing is browser printing. Each ticket is one print job for a printer the operating system can print to; Chrome shows its dialog for every job unless started with `--kiosk-printing`. The terminal does not drive an Odoo IoT Box or Epson ePOS network printers, and has no raw ESC/POS output: each kitchen station needs such a printer, or its own browser running the terminal. Courses ("fire course") are not implemented (chapter 27).

Split bill. Choose the lines and quantities for the new bill (combo items move with their combo), or use **Equal share** to select one share of the number of guests. The window shows the **New bill** and what **Stays on the table**. **Pay this bill** opens its payment; **Keep open** only creates it. The new bill is a separate order named like 5B, 5C, up to 26 parts. Odoo merges every open order of one table into one, so the new bill has no table link on the server; the link is kept on the device and the bill shows on its table in the plan. An order with payments cannot be split. A line cut in two can move one cent between the bills (tax is rounded line by line, as in Odoo).



Split the bill: Equal share with 4 guests, three lines moved; New bill 22.33, Stays on the table 30.10.

Bill and tip. Bill opens the **Bill preview**. Choose **No tip**, 5, 10, 15 or 20 % of the bill, or a **Custom amount** and **Apply**. The tip is a line with Odoo's tip product (set with the Tips option in Odoo); a new choice replaces it, 0 removes it. **Print bill** prints it, titled "BILL - not a receipt", and the table turns **To pay**; **Payment** goes to the payment screen. Tips after a card payment (a capture) are not supported.



Bill preview of table 5: tips No tip to 20%, 10% chosen, "Tip on the bill: 5.24"; untaxed 53.49, total 57.67.

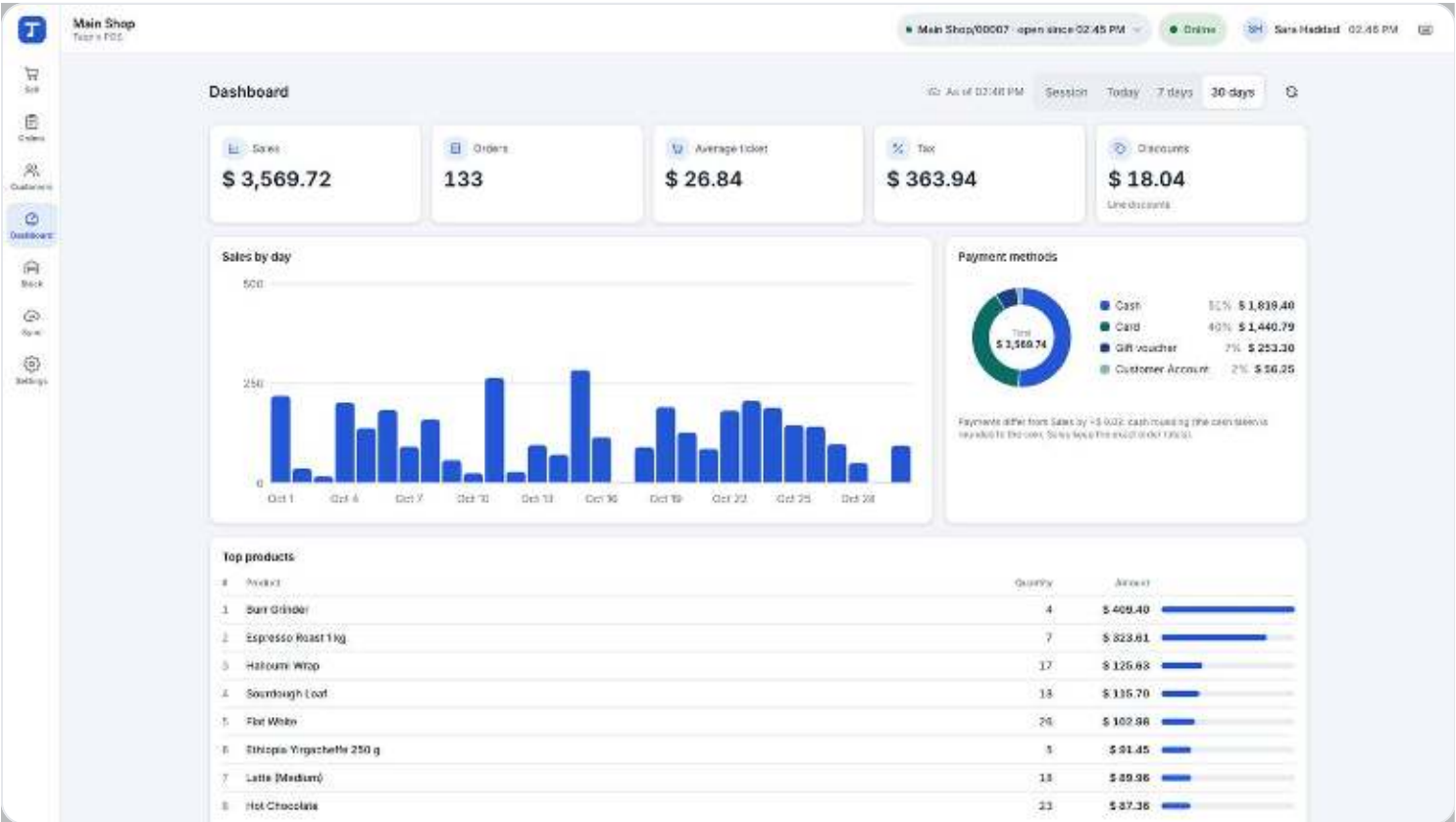
Transfer and merge. **Transfer** asks for the destination table ("Moving 5: tap the destination table", with **Cancel**). To a free table only the table changes; to a busy table the items join its order (equal lines add up) and the emptied order is cancelled, also offline. **Merge** on the plan does the same: first the table to move, then the destination. **Unmerge** puts the items of a merged table back on a new order of that table, which must be free.



Transfer mode: the banner "Moving 5: tap the destination table" with Cancel.

16. Dashboard

In plain words: The Dashboard answers "how are we doing?" for the session, the day, 7 days or 30 days. It is a Pro screen with a switch in Settings → Features.



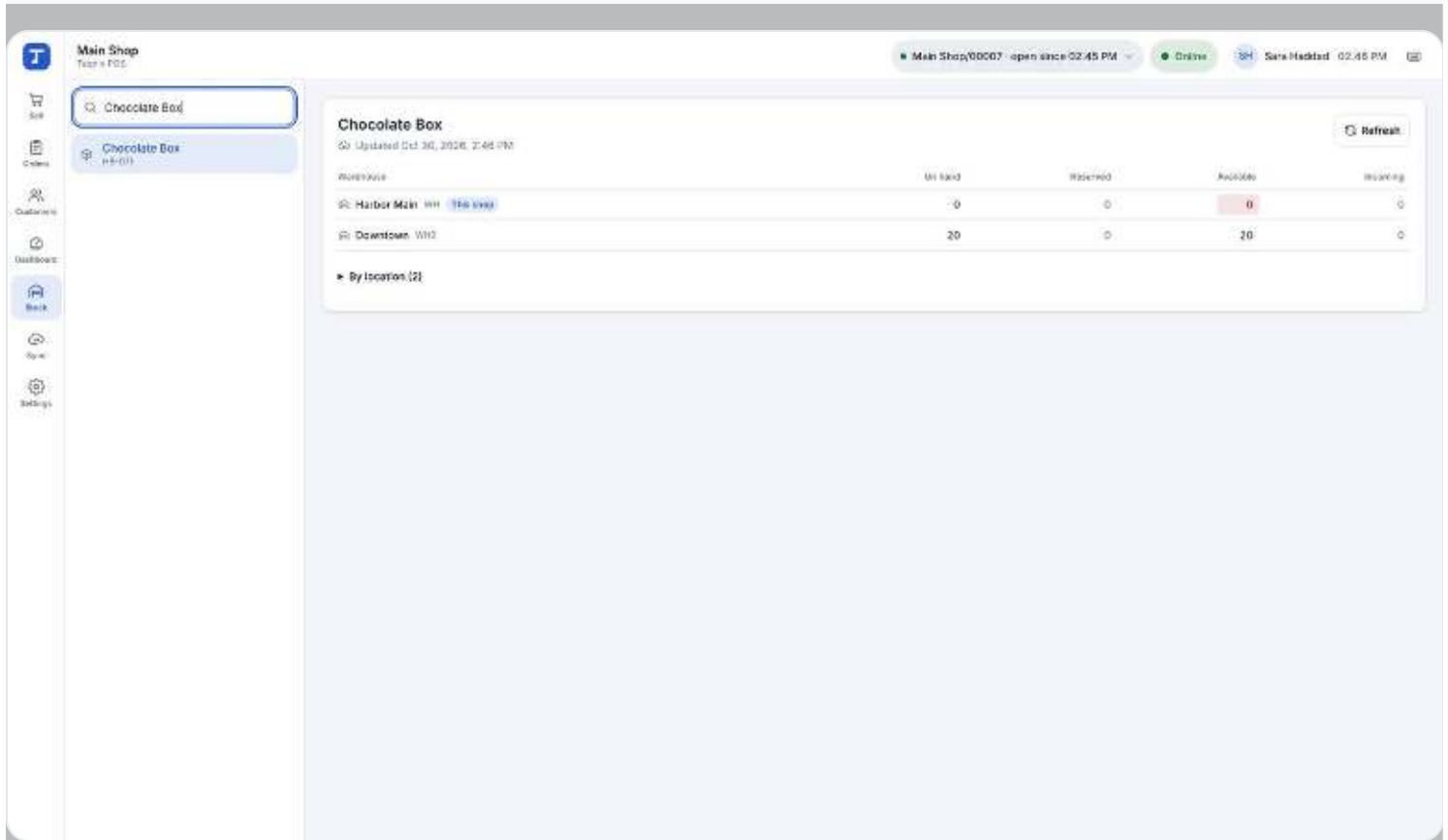
Dashboard of Main Shop, 30 days: Sales 3,569.72, Orders 133, Average ticket 26.84, Tax 363.94, Discounts 18.04; Sales by day; payment methods; Top products.

Buttons: **Session**, **Today**, **7 days**, **30 days**. Cards: **Sales**, **Orders**, **Average ticket**, **Tax**, **Discounts**. The chart is by hour for the session and today, by day for 7 and 30 days. **Top products** lists the eight products with the highest amount, without tips, discounts and rewards.

Detail	How it works
Source	Odoo's figures for the period plus the orders of this terminal that Odoo has not received yet, each counted once. The label reads "As of 02:46 PM" and "includes n orders waiting to sync".
Offline	The last answer for the same period ("Saved answer of ... (offline)") plus this terminal's orders; if never loaded, "This terminal only (offline)".
Discounts	Line discounts only. Discount lines of a whole-order discount are not counted.
Payments and Sales	Odoo keeps exact order totals but cash is rounded to the coin, so payments can differ from Sales by the cash rounding; the screen says so (here +0.02).

17. Stock lookup

In plain words: Search a product and see how many are in each warehouse, from Odoo, while the customer waits. It needs Odoo's Inventory app and the connection, and has a switch in Settings → Features.



The screenshot shows the Odoo POS interface. On the left is a sidebar with navigation icons for Shop, Orders, Customers, Dashboard, Back, Receipts, and Settings. The main area displays a search for 'Chocolate Box' with a dropdown menu showing the selected item. Below the search bar is a table titled 'Chocolate Box' with columns: Warehouse, On hand, Reserved, Available, and Incoming. The table shows two warehouses: Harbor Main (This shop) with 0 available (highlighted in red) and Downtown W10 with 20 available. A 'Refresh' button is in the top right corner of the table. Below the table is a link 'By location (2)'.

Warehouse	On hand	Reserved	Available	Incoming
Harbor Main This shop	0	0	0	0
Downtown W10	20	0	20	0

Chocolate Box (HB-071): Harbor Main (This shop) 0 available, highlighted; Downtown 20 available.

Type a name, reference or barcode. The table shows **On hand**, **Reserved**, **Available** and **Incoming** for each warehouse, the warehouse of this point of sale first with the tag **This shop**. Available below 5 is shown as low, 0 or less as out of stock. **By location** opens the locations; a product with variants lists each variant. Offline, the screen shows the last answer for each product you looked up, with its time: "Offline: this is the last answer of Odoo, not the current stock." The tiles of the Sell screen show no stock.

18. Layaway (deposits)

In plain words: A layaway keeps an order open with a deposit; the customer pays the balance later. In Odoo the order stays a draft with its deposits until the balance is collected.

The screenshot shows the Odoo POS interface for a payment. At the top, it says 'Main Shop' and 'Test POS'. The customer name 'Liam Garcia' is displayed. The payment method is 'Card', and the amount paid is '\$30.00'. The total amount is '\$110.40'. The remaining balance is '\$80.40'. A 'Layaway' box is visible with the option 'Keep open with deposit' selected. The interface includes a numeric keypad and a 'Validate' button.

Payment for Liam Garcia: total 110.40, Card 30.00 paid, Remaining 80.40; the Layaway box with Keep open with deposit.

- 1 Choose a customer on the order: a layaway needs someone who owes the balance.
- 2 On the payment screen enter the deposit as a normal payment.
- 3 In the **Layaway** box press **Keep open with deposit**. The terminal shows a deposit receipt, **LAYAWAY - DEPOSIT**, to hand to the customer.
- 4 Later open **More** → **Layaways** and press **Collect payment** for the customer. Pay the balance and **Validate**.

Rule	Why
A customer is required; the deposit is more than 0 and less than the total; no change, no customer-account deposit, no refund, no table order.	A deposit is real money taken for a balance somebody owes.
A deposit is locked: it cannot be edited or removed here, and the order cannot be cancelled here.	So the cashier cannot delete the deposit and collect the full price. Collect the balance, or refund the deposit in Odoo.
Odoo cannot close a session that has an open layaway (a draft order).	Collect the balance, or cancel it in Odoo, before closing (chapter 20).
A layaway made on another device must be downloaded first (Pull data now).	The list says so.

The screenshot shows the 'Layaways' screen in the Odoo POS interface. At the top, a warning message reads: 'Odoo cannot close a session while it has open layaways (draft orders): collect them or cancel them in Odo before closing.' Below this, there are two layaway entries:

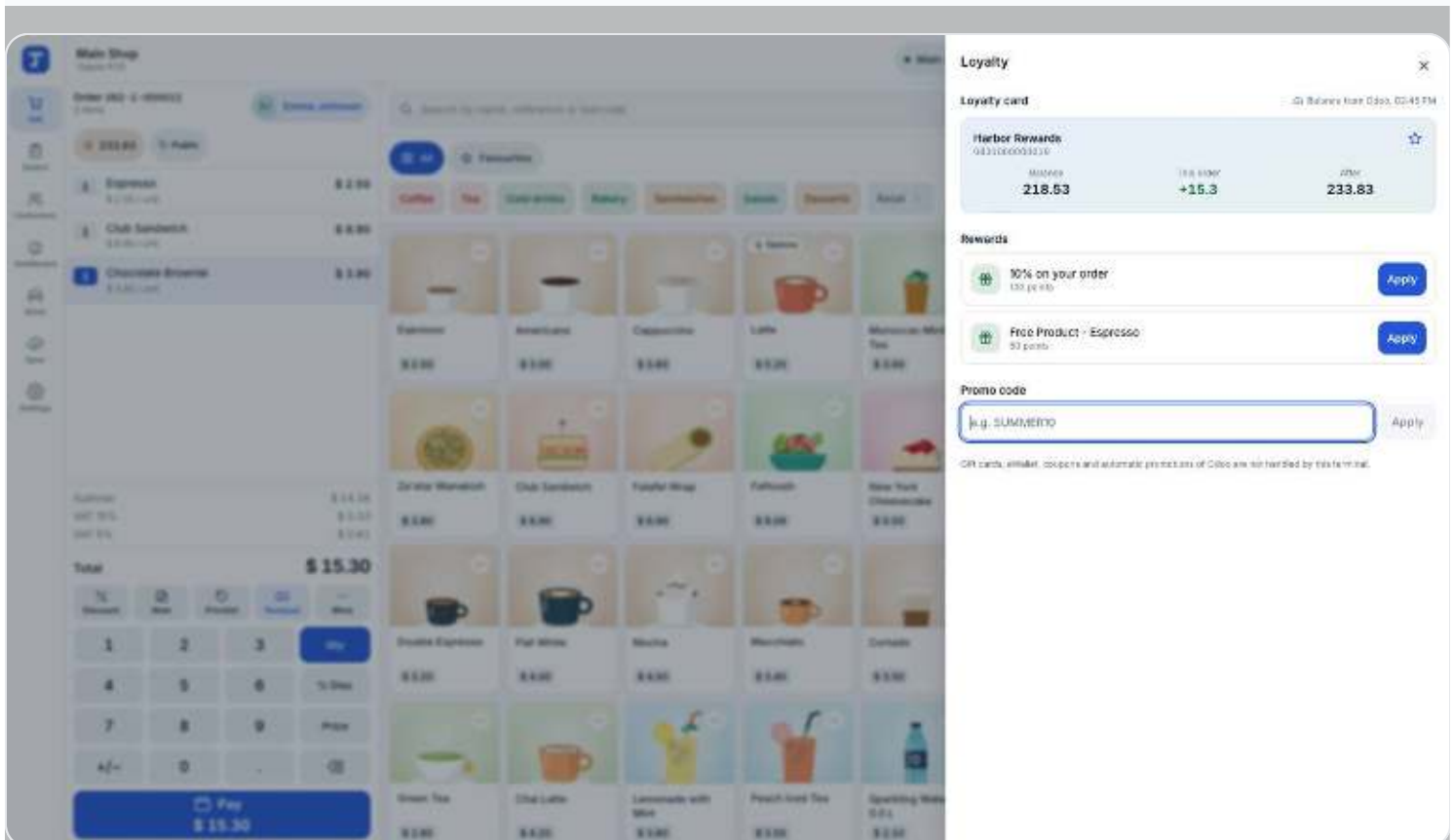
Customer	Total	Paid	Balance due	Action
Olivia Brown (OB)	\$56.35	\$20.00	\$36.35	Collect payment
Liam Garcia (LG)	\$110.40	\$30.00	\$80.40	Collect payment

The total due for all layaways is \$116.75.

Layaways of Main Shop: total due 116.75; Olivia Brown (balance 36.35) and Liam Garcia (balance 80.40), each with Collect payment.

19. Loyalty and promo codes

In plain words: When Odoo's **Loyalty** app is installed, the terminal shows the customer's points, the rewards they can use and a promo code field. The points of a sale are computed on the terminal with the rules of Odoo's loyalty programmes and confirmed to Odoo after the order is sent. It has a switch in Settings → Features.



Loyalty panel for Emma Johnson: Harbor Rewards balance 218.53, this order +15.3, after 233.83; two rewards with Apply; a Promo code field.

Choose a customer, then tap the points chip next to the customer or use **More → Rewards & promo code**. The card shows the **Balance** from Odoo (with its time), **This order** and **After**. **Apply** adds a reward to the order as a line; the trash button removes it. A reward that no longer fits after a change is removed, with a message. A promo code such as SUMMER10 is typed in **Promo code** and works offline.

Supported	Details
Programmes	Loyalty cards (one for each customer) and promo codes (rules with a code).
Earning	Points per order, per amount or per unit; minimum amount and quantity, product filters, dates, pricelists, usage limits, nominative programmes.
Rewards	A discount (percent, fixed for the order, or per point) on the order, the cheapest line or specific products, with a maximum; a free product already in the order.

Not supported	What the terminal does
Gift cards and eWallet	Not handled. Products that trigger an active gift card or eWallet programme ("Gift Card", "Top-up eWallet") are hidden from the catalogue and the barcode lookup, because the terminal cannot create or top up a card. Without an active programme such a product is ordinary, as in Odoo.
Coupons, next-order coupons, buy X get Y, automatic promotions, card codes, split points, multi-product rewards, free shipping	Not handled and not shown.

Using points needs the connection: the terminal asks Odoo first, and takes off the balance the points spent by orders Odoo has not confirmed yet. After the order reaches Odoo the terminal confirms the points once (the queue shows **Loyalty points**) and the card is credited or debited then. If the loyalty switch is turned off with an order open, its reward lines are removed.

20. Close the session

In plain words: At the end of the shift you count the cash and close the session; Odoo posts it with its accounting entries. Closing needs the connection: offline, keep selling and close later. Nothing is lost.

Open **Close session** from the session chip, the Sync screen or **Settings → Terminal identity**. The box **Before closing** checks:

Check	If it is not met
Connected to Odoo	Closing needs the connection.
Everything sent to Odoo	Press Sync now . Odoo must have every order and waiting change first.
No open draft orders	Finish or cancel held and table orders (Resume , Cancel). Drafts left in Odoo by other terminals can be cancelled with the closing, except those with a deposit (layaways): collect them first.
Customer account sales have a customer	Odoo refuses otherwise. Set the customer in Odoo (Point of Sale → Orders), then Check again .

Main Shop

Tier 1 POS

Main Shop/00007

open since 02:45 PM

Online

SM - Sales Hacked - 02:48 PM

Close session

Main Shop/00007

Back to setting

Before closing

Connected to Odoo

Everything sent to Odoo

No open draft orders

Customer account sales have a customer

Count the cash and confirm

Session

Opened

Oct 30, 2026, 2:45 PM

Orders

10

Sales

\$ 286.85

Cash in the drawer

Count the cash in the drawer, Odoo compares it with what it expects.

Count bills and coins

Enter the total

Counted cash

254.40

Expected

\$ 254.90

Counted

\$ 254.40

Difference

- \$ 0.50

Other payment methods

Enter the card terminal total, when it differs from Odoo, the difference is posted at closing.

Card

Expected \$ 188.65

\$188.65

Customer Account

Expected \$ 0.00

Waiting to cash

Gift voucher

Expected \$ 0.00

0

Session summary

From Odoo (orders, cash, payment methods) and from the orders made on this device (taxis, products).

Orders

10

Sales

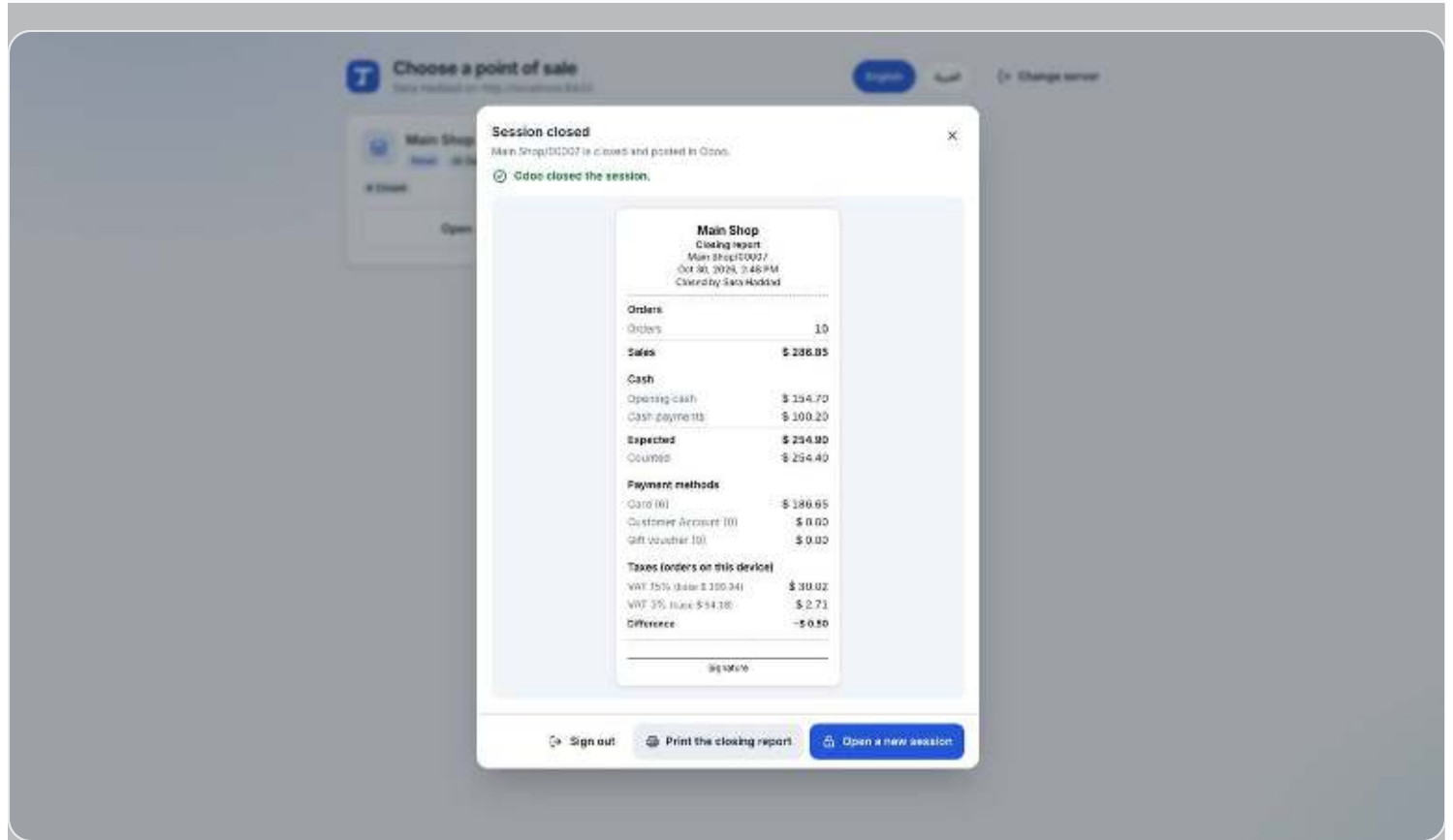
\$ 286.85

After this step, this session accepts no more orders.

Close the session

Close session of Main Shop/00007: four checks green; 10 orders, sales 286.85; cash expected 254.90, counted 254.40, difference -0.50; Close the session.

Count the cash by bills and coins or by total. **Expected** is the opening cash plus cash payments and cash moves. In **Other payment methods** enter the card terminal's total when it differs from Odoo; the difference is posted at closing. A **Closing note** stays on the session for the manager. **Session summary** lists orders, sales, cash, payment methods and, from the orders made on this device, taxes and the products sold most. If the difference is above the threshold (**Cash difference that needs a confirmation**, 5 by default) you must type the session name. Then press **Close the session** and confirm.



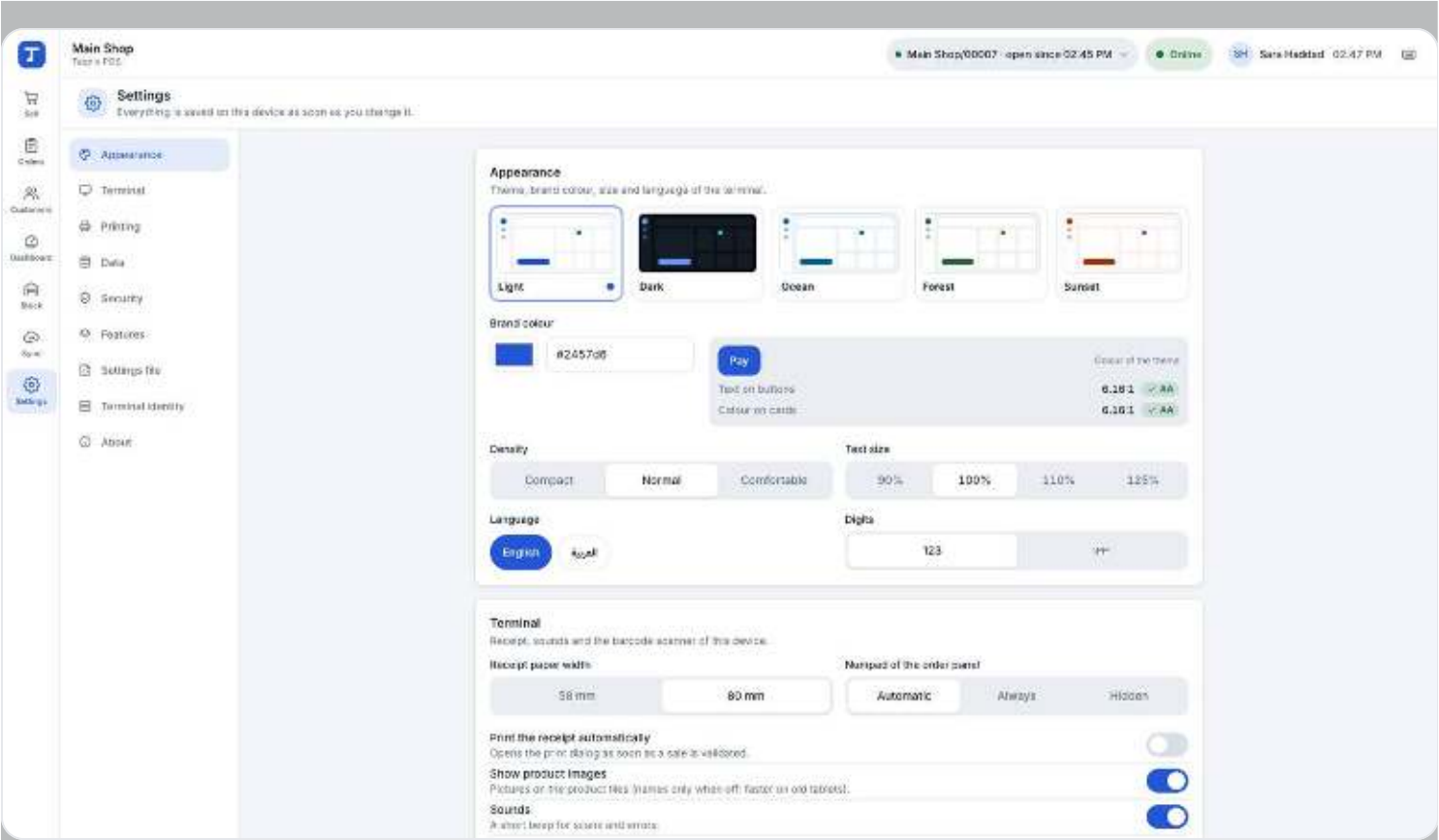
After closing: opening cash 154.70 + cash payments 100.20 = expected 254.90, counted 254.40, difference -0.50; taxes of this device's orders; Sign out, Print, Open a new session.

Important: If Odoo refuses at the last step, the session may stay in closing control: it takes no orders and no new session can open until it is closed. Sent orders are safe. A user with **Point of Sale / Administrator** opens **Point of Sale → Sessions** in Odoo, selects the session, chooses **Close Session & Post Entries** and, if asked to force the closing, checks the difference and confirms. Then open a new session from the picker.

Print the report through the browser, then **Open a new session** or **Sign out** (unsent orders stay on the device). The terminal does not print Odoo's full Z report (it needs accounting rights): a manager prints it in **Point of Sale → Reporting → Sales Details**.

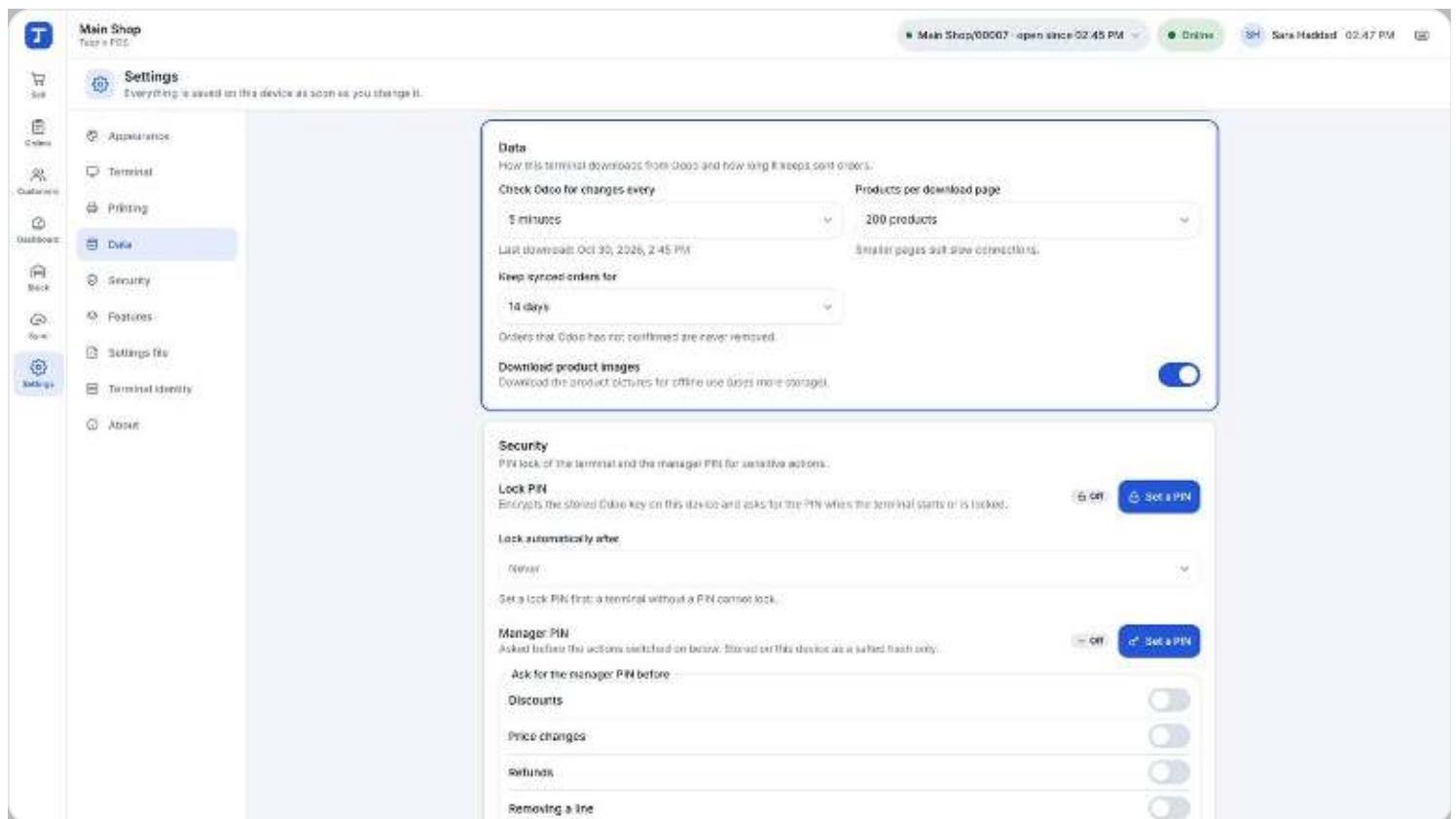
21. Settings

In plain words: Settings are saved on this device as soon as you change them, not in Odoo. To copy them to another terminal, use the settings file.

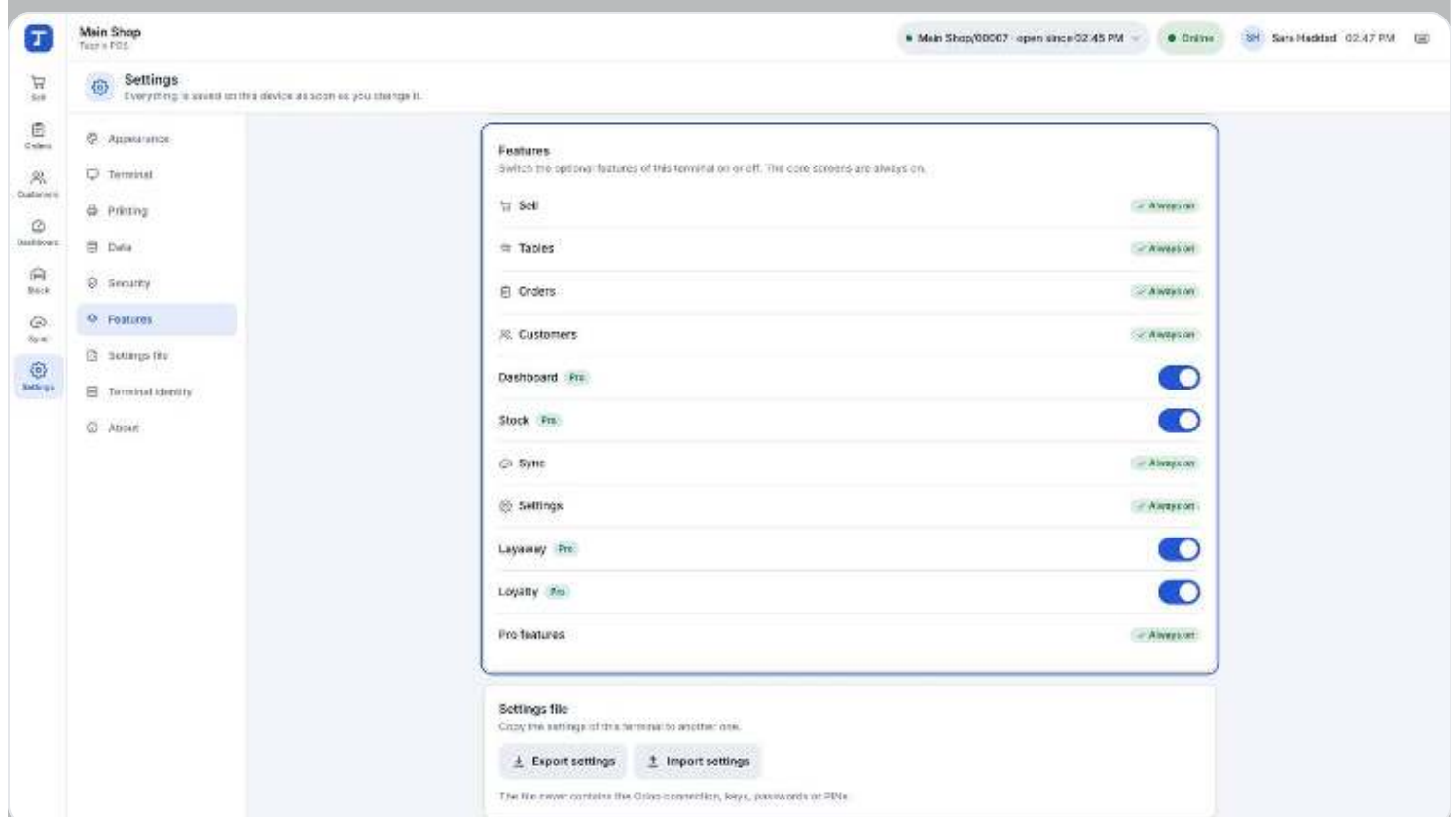


Settings, Appearance: five themes, brand colour #2457d6 with contrast 6.16:1 (AA), Density, Text size, Language, Digits; below, the Terminal card.

Section	What you can set
Appearance	Theme (five). Brand colour (picker, hex field, two contrast readings: AAA, AA, "Large text only", "Too low"); a colour too pale or dark for the theme is adjusted; Use the theme colour resets it. Density , Text size (90 to 125 %), Language (English, Arabic: also the direction), Digits (123 or ١٢٣).
Terminal	Receipt paper width , Numpad of the order panel (Automatic, Always, Hidden), Print the receipt automatically , Show product images , Sounds , and the Barcode scanner (on or off, maximum time between characters 20 to 120 ms, shortest barcode 3 to 12).
Printing	How printing works, Print a test page , and the kitchen and bar printers configured in Odoo (read only).
Data	Check Odoo for changes every 1 to 60 minutes; Products per download page 50 to 1,000; Keep synced orders for 3 to 365 days (14 by default; unconfirmed orders are never removed); Download product images .
Security	Lock PIN, automatic lock, manager PIN, cash difference threshold (chapter 25).
Features	Switches for Dashboard, Stock, Layaway and Loyalty; the core screens are always on.
Settings file	Export settings ; Import settings shows Setting , Now , After import before Apply . It never contains the connection, keys, passwords or PINs; a file from a newer version is refused.
Terminal identity	Device identifier, server, database, login, user, connection mode, point of sale, session and Close session .

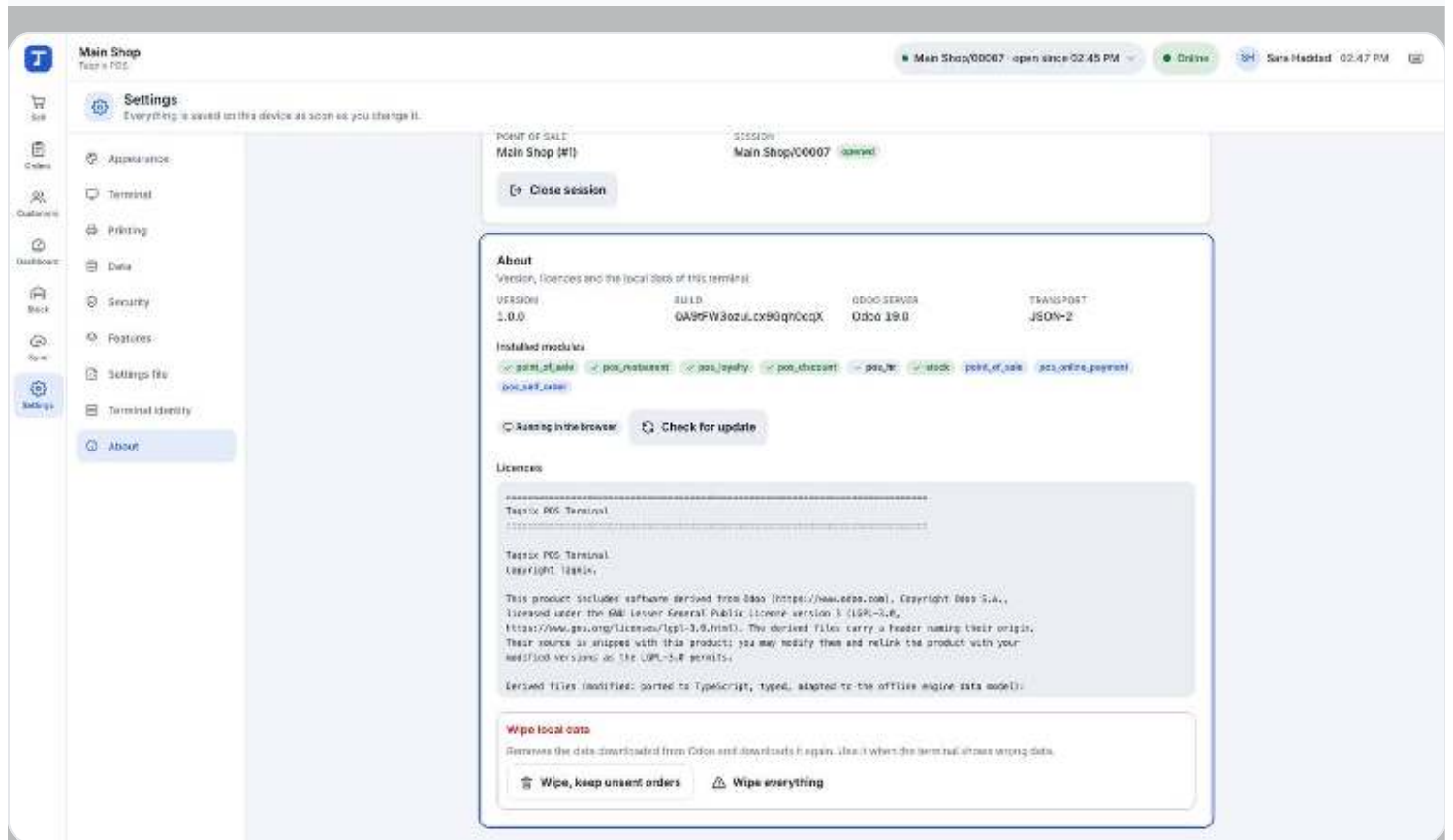


Settings, Data: every 5 minutes, 200 products per page, orders kept 14 days; below, the Security card.



Settings, Features: core screens Always on; Dashboard, Stock, Layaway and Loyalty with a Pro tag and a switch; the Settings file card.

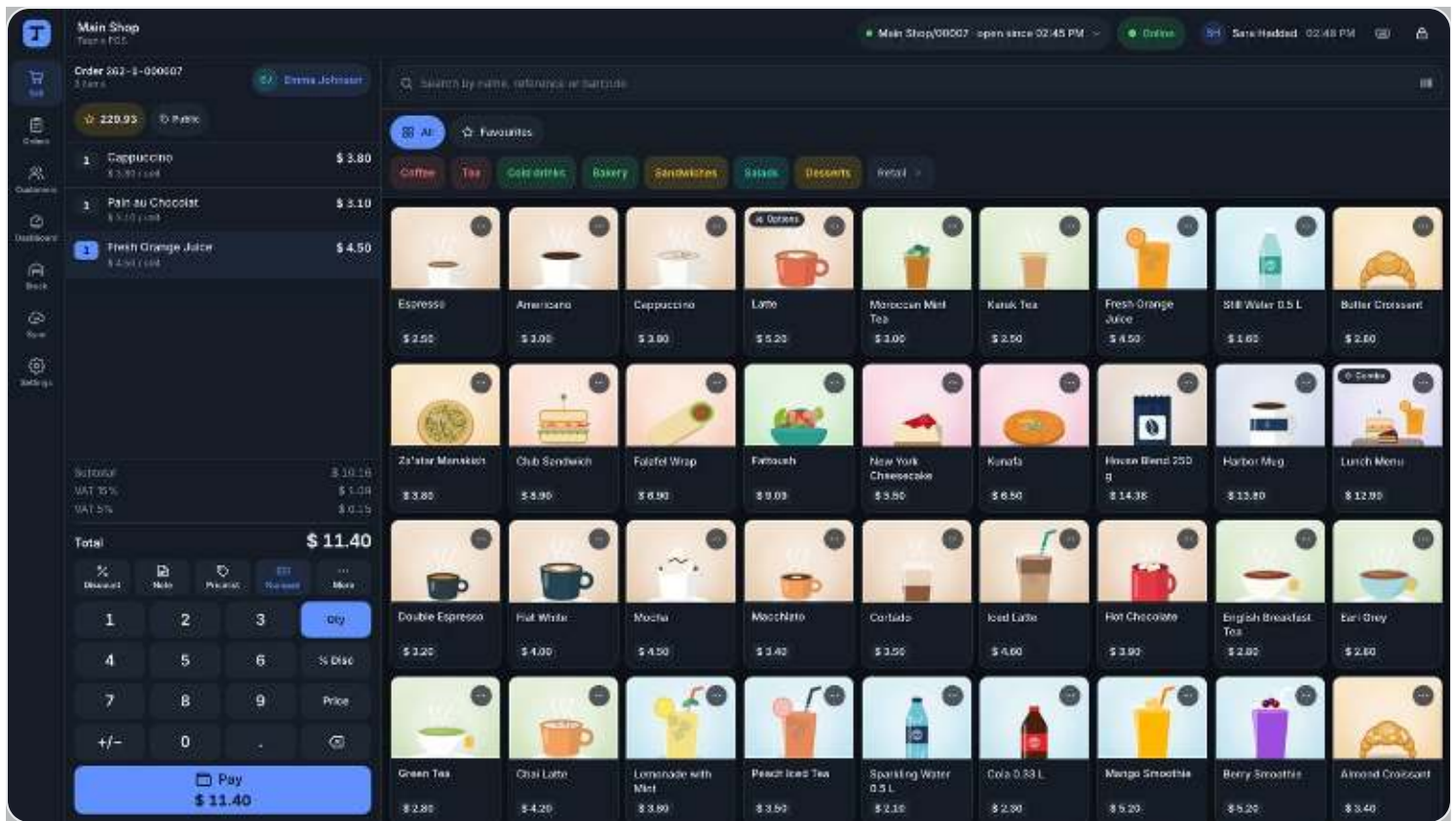
About shows the version (1.0.0), the build, the Odoo server, the transport and the apps found (ticked, or a dash when absent), **Check for update** (chapter 29), **Install app** when the browser offers it, and the **Licences**. **Wipe local data** removes the data downloaded from Odoo and downloads it again. **Wipe, keep unsent orders** keeps unconfirmed orders and changes, the connection and the settings. **Wipe everything** also deletes the unsent orders: it asks twice, and the second time you type the number of orders that will be lost.



Settings, About: version 1.0.0, Odoo 19.0, JSON-2, installed modules, Check for update, licences, Wipe local data.

22. Themes, brand colour and languages (for developers)

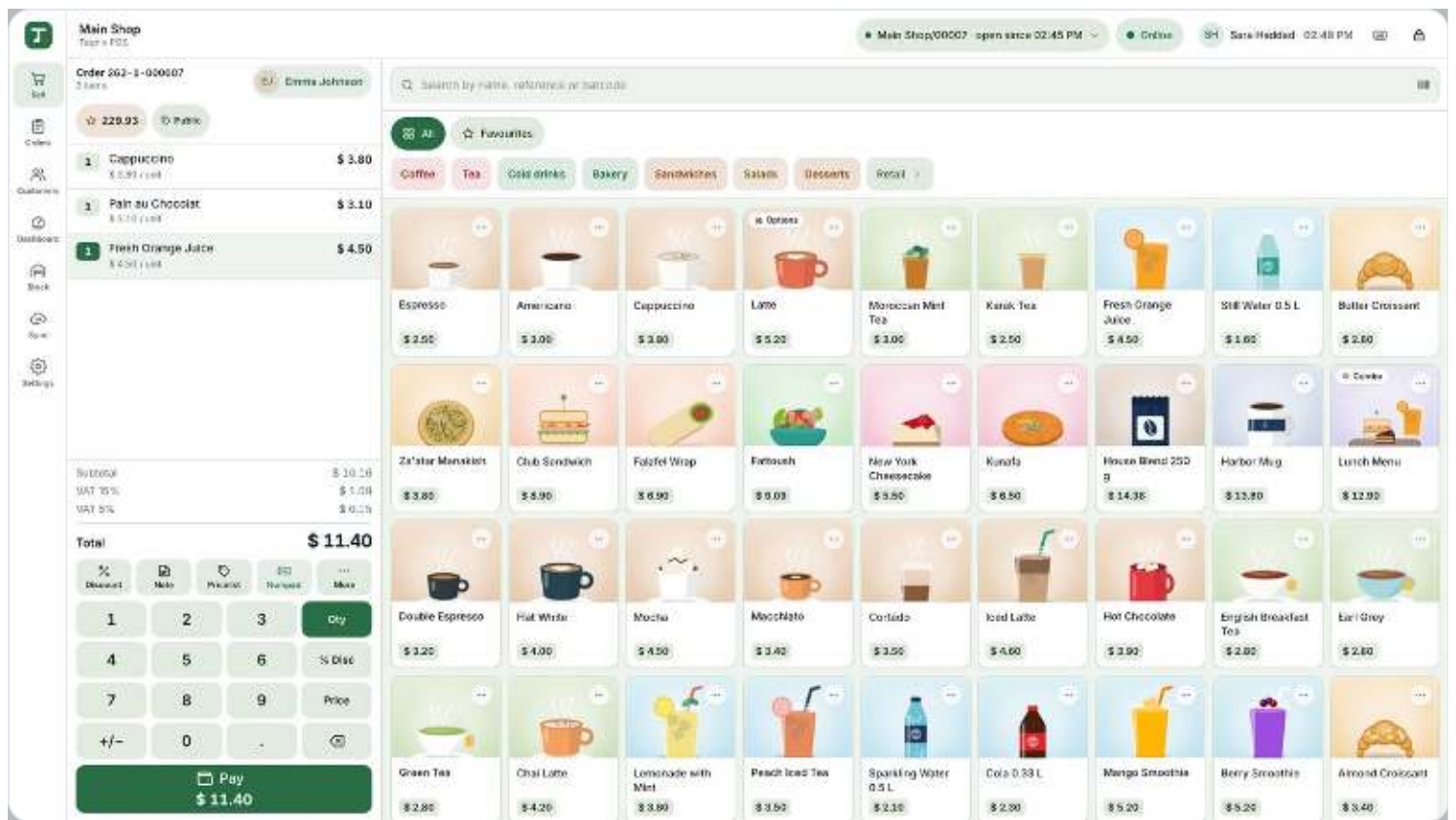
In plain words: Every colour, radius and size of the interface is a CSS variable. A theme is one block of variables, and no component hard-codes a colour.



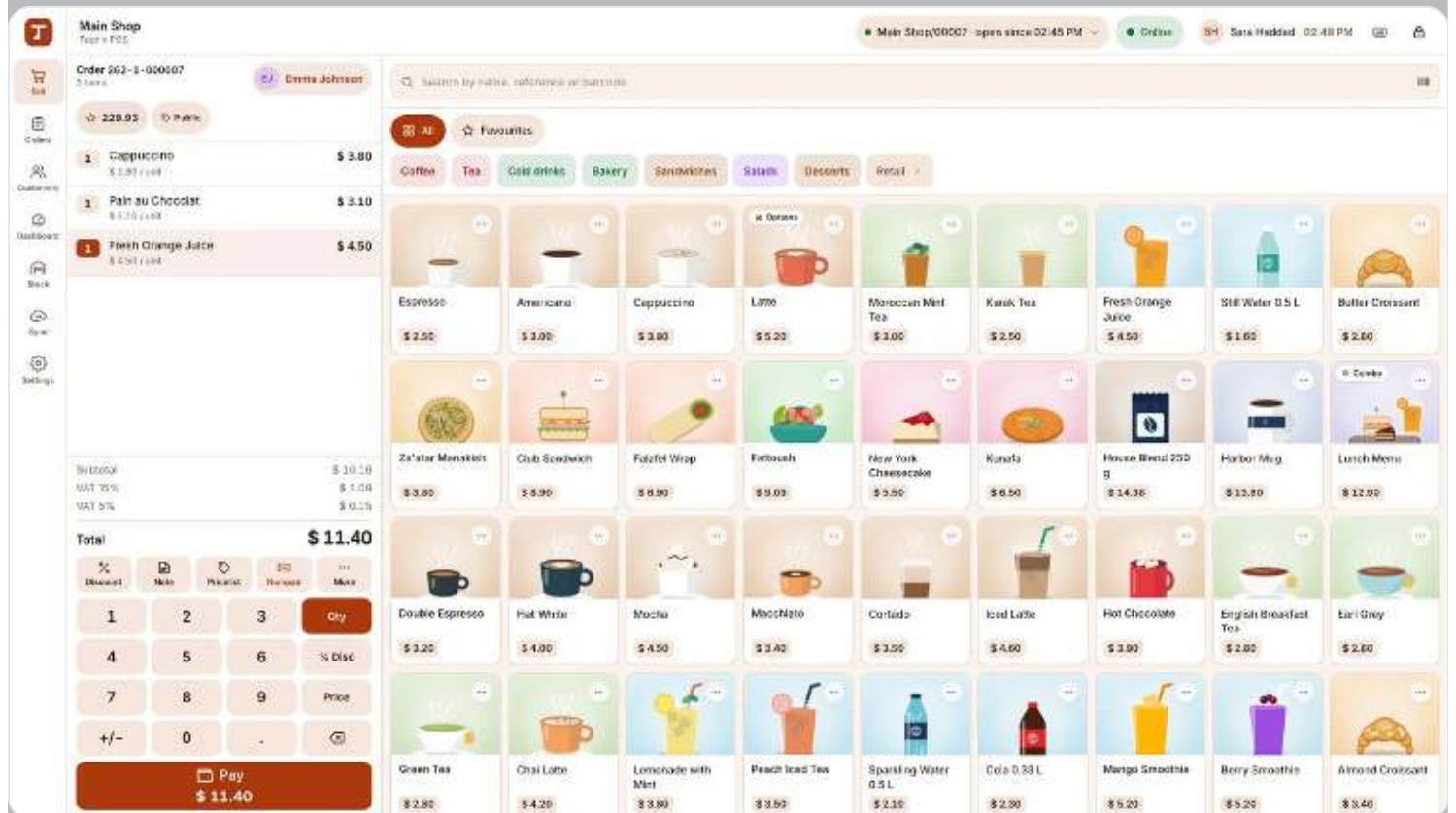
The Sell screen in the Dark theme (an order of three items, total 11.40).



The same screen in the Ocean theme.



The same screen in the Forest theme.



The same screen in the Sunset theme.

The variables. The light theme is on `:root` in `src/app/globals.css`. The other themes override what they change on `:root[data-theme="..."]`; the rest comes from light. The tokens are `--pos-bg`, `-surface`, `-surface-2`, `-border`, `-fg`, `-muted`, `-primary`, `-primary-fg`, `-accent`, `-success`, `-warning`, `-danger`, then radius, shadow, font and density. An `@theme inline` block maps Tailwind's utilities to them (`bg-surface`, `text-fg`, `bg-primary`), so components never name a colour.

Add a theme. A theme lives in five places; the example is the real Ocean theme:

```
/* 1. src/app/globals.css: one block of variables per theme (the tokens a theme does not
:root[data-theme="ocean"] {
  --pos-bg: #eaf3f9;
  --pos-surface: #ffffff;
  --pos-surface-2: #dcebf4;
  --pos-border: #c4dbe9;
  --pos-fg: #0e2433;
  --pos-muted: #516c7e;
  --pos-primary: #0a6595;
  --pos-primary-fg: #ffffff;
  --pos-accent: #06687f;
}

/* 2. src/lib/theme/prefs.ts: the list of names */
export const THEMES = ['light', 'dark', 'ocean', 'forest', 'sunset'] as const

/* 3. src/lib/theme/themes.ts: the colours of the small theme chips (a test keeps them eq
ocean: { bg: '#eaf3f9', surface: '#ffffff', primary: '#0a6595', accent: '#06687f' },

/* 4. src/app/layout.tsx: the boot script, so there is no flash of the wrong theme */
var T=['light','dark','ocean','forest','sunset'];

/* 5. src/lib/i18n/en/core.ts and ar/core.ts: the name shown on the chip */
'theme.ocean': 'Ocean',
```

Then check the contrast: `src/lib/theme/contrast-themes.test.ts` reads the stylesheet and requires 4.5:1 (WCAG AA) for every text and background pair the components use, in every theme, and checks that the chip colours equal the stylesheet. Add the new name to the list in that test. Setup and Settings build their theme chips from `THEMES`.

Brand colour. The picker writes `--pos-primary` and `--pos-primary-fg` on `<html>`.

`src/lib/theme/contrast.ts` picks a readable text colour, and adjusts a colour that is too pale or

dark for the cards (below 3:1). Preferences live in `localStorage` (`pos.prefs`), and a small script in `layout.tsx` applies theme, direction, colour and density before the first paint.

Right to left. The language sets `dir` on `<html>` . The CSS uses logical utilities (`ms-*` , `ps-*` , `text-start`), amounts are always left to right (`.amount`), codes like 262-1-000001 keep their order inside Arabic text, and the floor plan is never mirrored. Fonts: Inter and IBM Plex Sans Arabic, bundled.

Languages. English and Arabic are built in. Texts are flat dictionaries (`'sell.pay': 'Pay'`), one file for each area in `src/lib/i18n/en/` and `ar/` , read with `t('key', {vars})` ; plural messages have the forms of the language (English two, Arabic six); `GLOSSARY.md` fixes the Arabic terms. Tests fail when Arabic lacks a key or placeholder of English, when a key used in the code does not exist, and when visible text in `src/components` bypasses `t()` .

Important: A third language is a code change in several places, not a drop-in folder: the type `Lang` and the list `LANGS` (`src/lib/theme/prefs.ts`), the direction, the plural rules (`plural.ts`), the number and date locale (`format.ts`), the boot script of `layout.tsx` , the language chips and the registration of the dictionaries all name the two languages.

23. How totals are computed

In plain words: The terminal must compute totals before Odoo can be reached, and Odoo trusts the totals a point of sale sends. So they must be exact. The terminal uses a TypeScript port of Odoo 19's own tax helper and of the pricing rules of Odoo's point of sale, and that port was checked against Odoo itself.

What it covers. Taxes (percentage, fixed, division, group; price included or excluded; negative rates; reverse charge), fiscal positions, pricelists (fixed, percentage, formula, based on another pricelist, dates, quantity tiers), variant extras, combo price split, cash rounding, the whole-order discount, rounding per line or globally, currencies with 0, 2 and 3 decimals. It is plain TypeScript with no browser, database or network (`src/lib/engine/` , documented in its README).

Check	Result
667 reference scenarios built in Odoo 19 (399 orders, 216 unit prices, 36 combos, 16 whole-order discounts), expected numbers from Odoo 19's own Python tax helpers.	Terminal = Odoo's Python on 667 of 667.
447 of those orders were also sent to Odoo with <code>sync_from_ui</code> and read back.	All accepted and paid.
The same scenarios through Odoo's own point-of-sale JavaScript.	Terminal = Odoo's JavaScript on 664 of 667. The 3 documented exceptions are cases where Odoo's JavaScript and Python disagree; the terminal follows the Python, which Odoo's server recomputes.
An independent replay: 1,149 orders built through the real order flow and sent to Odoo 19 with <code>sync_from_ui</code> as a cashier (several roundings, currencies, sales, refunds).	1,085 stored; 64 refused by Odoo with an access error on lot or serial products, which a Point of Sale-only cashier cannot read. Every order recomputed with Odoo's Python and JavaScript: terminal = Python on every order.

Where Odoo's own client and server disagree. The terminal computes what Odoo's point-of-sale client computes and sends, because that is what `sync_from_ui` stores. In a few cases Odoo's server then stores something slightly different:

Case	What happens
Whole-order cash rounding	The client sends the unrounded total and pays the rounded amount; the server computes the rounded total. The stored total is the client's, except when change is given: then Odoo adds a negative cash payment and recomputes, so the stored total is the rounded one and the amount paid is net of change (with a card payment plus cash change Odoo can even refuse the payment and save the order as "/"). Not a sync conflict.
A fiscal position without tax mapping	Odoo's client keeps the taxes, its server removes them. Orders can end as "/" and the closing can be blocked. Map every tax, also to a 0 % tax.
Pricing	The server never re-prices POS lines. A negative quantity matches no pricelist rule in the POS, while the backend applies rules with minimum quantity 0.
Line totals rounded by the backend	The backend form rounds a line total differently (by 0.01 on some lines in the check); order totals agree.
Rounded globally, negative-price line	The invoice tax can differ by 0.01; a sale with a negative total is recomputed by the server on flipped quantities (also 0.01).

Deliberate differences. Pricelist items are also filtered by date at the moment of the sale (Odoo's POS filters only at loading, so a terminal running for days would keep an expired promotion). Line totals are rounded to the currency. The tax ids inside each tax group are sorted.

Important: Not supported: taxes computed by Python code (the terminal says "A tax of this product cannot be computed offline. Ask your manager to check the tax setup."); pricelists in another currency and unit conversions, ignored as in Odoo's POS; whole-order discount lines when the point of sale's currency differs from the company's (the company-currency amounts are left out, as Odoo's POS does). The Portuguese fiscal rule was checked with Odoo's Python helpers only. Loyalty rewards are plain lines for the engine.

Licence. `account_tax.ts` , `float.ts` , `pricing.ts` , `fiscal.ts` , `order.ts` , `combo.ts` , `attributes.ts` and `discount.ts` are ported from Odoo 19. They carry a header naming their origin, are listed in `NOTICE` and stay under the LGPL-3.0 (`LICENSE-LGPL-3.0.txt`). The test tools that run Odoo's own JavaScript read it from a local Odoo installation; no Odoo file is copied into the package.

24. Architecture (for developers)

In plain words: The terminal is one client-rendered page, a small proxy to Odoo, a manifest and a service worker. The work happens in the browser, on data kept in IndexedDB. This is a map for a developer who changes the code.

The whole interface is one route, `/` ; screens switch inside it, so a full reload works offline from one cached HTML file. The server side has `/api/odoo` (the proxy), `/manifest.webmanifest` and `/sw.js` (written into `public/` by `scripts/make-sw.mjs` after `next build`). The browser calls the proxy, which forwards to the allowed Odoo origins (unless you choose the direct mode).

Folder	What it holds
<code>src/app/</code>	Layout and boot script, page, manifest, <code>globals.css</code> , <code>api/odoo/route.ts</code> (the proxy).
<code>src/lib/odoo/</code>	One <code>OdooClient</code> over JSON-2 (<code>/json/2/<model>/<method></code> , bearer key) and JSON-RPC; the connection test; the proxy and its address rules; typed errors; zod schemas.
<code>src/lib/db/</code> , <code>sync/</code>	The Dexie schema, queries and PIN encryption; push, pull, scheduler, order payload, closing calls, event log.
<code>src/lib/engine/</code> , <code>pos/</code>	Prices, taxes, totals (pure TypeScript); the open-order store, receipt, barcode rules, scanner decoder, Code-128 and QR.
<code>src/lib/restaurant/</code> , <code>pro/</code> , <code>features/</code> , <code>i18n/</code> , <code>theme/</code> , <code>pwa/</code> , <code>screens/</code>	Restaurant and Pro logic; the feature registry; dictionaries; themes; service worker registration; the logic behind the Orders, Customers, Sync, Settings, Close and Lock screens.
<code>src/lib/terminal/</code>	The facade: the only door from the interface to the connection, the database and the sync.
<code>src/components/</code>	Screens and widgets, one folder for each area, and <code>extensions/</code> where optional parts register.
<code>deploy/</code> , <code>scripts/</code>	The nginx examples; <code>deploy/webapp/</code> , the launcher and README of the ready-to-run app; the generators of the service worker and the licences file; <code>scripts/pack-webapp.mjs</code> , which assembles the ready-to-run folder from the standalone build (<code>npm run pack:webapp</code>).

The local database. One Dexie table for each mirrored Odoo model, named like the model with the dot turned into an underscore, keyed by `id` (`product_template` , `res_partner` ,
`account_tax` ...), plus `meta` (connection, session, installed modules, last pull, preferences, caches),
`orders` (by uuid), `ops` (operations that are not orders), `images` and `log` . Zustand stores hold
screen state; the open order is written to Dexie shortly after each change.

Data in and out. Settings, taxes, pricelists and so on load with Odoo's own point-of-sale loader
`pos.session.load_data` ; the catalogue is paged with

`product.template.load_product_from_pos` ; later pulls ask for what changed since the server time of the last pull and check deletions on the server. Orders go out with `pos.order.sync_from_ui` , one for each call, idempotent by uuid; drafts (tables, layaways) go as drafts. Other operations (new customers, loyalty confirmations, cancelling a table order, closing) are queued in `ops` and run in order before the orders that depend on them.

The feature registry. `registerFeature` declares which Odoo apps a feature needs (`requires.modules`), a condition on the point of sale (`requires.config`), whether it needs the connection (`requires.online`), its place in the rail (`nav`), its screen and the slots it fills (`cartSlot` , `paymentSlot` , `orderActions` , `lineBadge` , `settingsPanel`). A feature whose app is missing, or that is switched off, is not mounted and its code is never loaded.

Add a Pro feature (the steps of `src/lib/pro/README.md` , for an example called `giftcards`):

- 1 Logic in `src/lib/pro/giftcards/` as pure functions with tests. Network only through `getOdooCaller()` (null offline); check every answer with zod.
- 2 Cache what must stay readable offline with `readCache` / `writeCache` (`meta` keys `pro.giftcards.*`).
- 3 The screen in `src/components/pro/giftcards/` (default export); wrap a slot with `lazy()` in `src/components/pro/slots.tsx` .
- 4 Messages `pro.giftcards.*` in `src/lib/i18n/en/pro.ts` and `ar/pro.ts` (the test fails if Arabic lacks a key).
- 5 Register the feature in the registration file in `src/components/extensions/` .
- 6 Add the id to `PRO_FEATURE_IDS` (`src/lib/pro/features.ts`) and the texts `pro.feature.giftcards` and `_desc` : the feature gets its switch in Settings → Features, on by default.
- 7 Never import feature code statically in the registration file: a feature that is off or missing must load nothing.

```
// the registration file of the Pro features (src/components/extensions/): one call per f
registerFeature({
  id: 'giftcards',
  title: 'Gift cards',
  tier: 'pro',
  requires: { modules: ['pos_loyalty'] },
  nav: { icon: 'gift', labelKey: 'pro.giftcards.title', order: 65 }, // place in the le
  screen: () => import('@components/pro/giftcards/giftcards-screen'), // loaded only whe
  // optional: cartSlot, paymentSlot, orderActions, settingsPanel
})
```

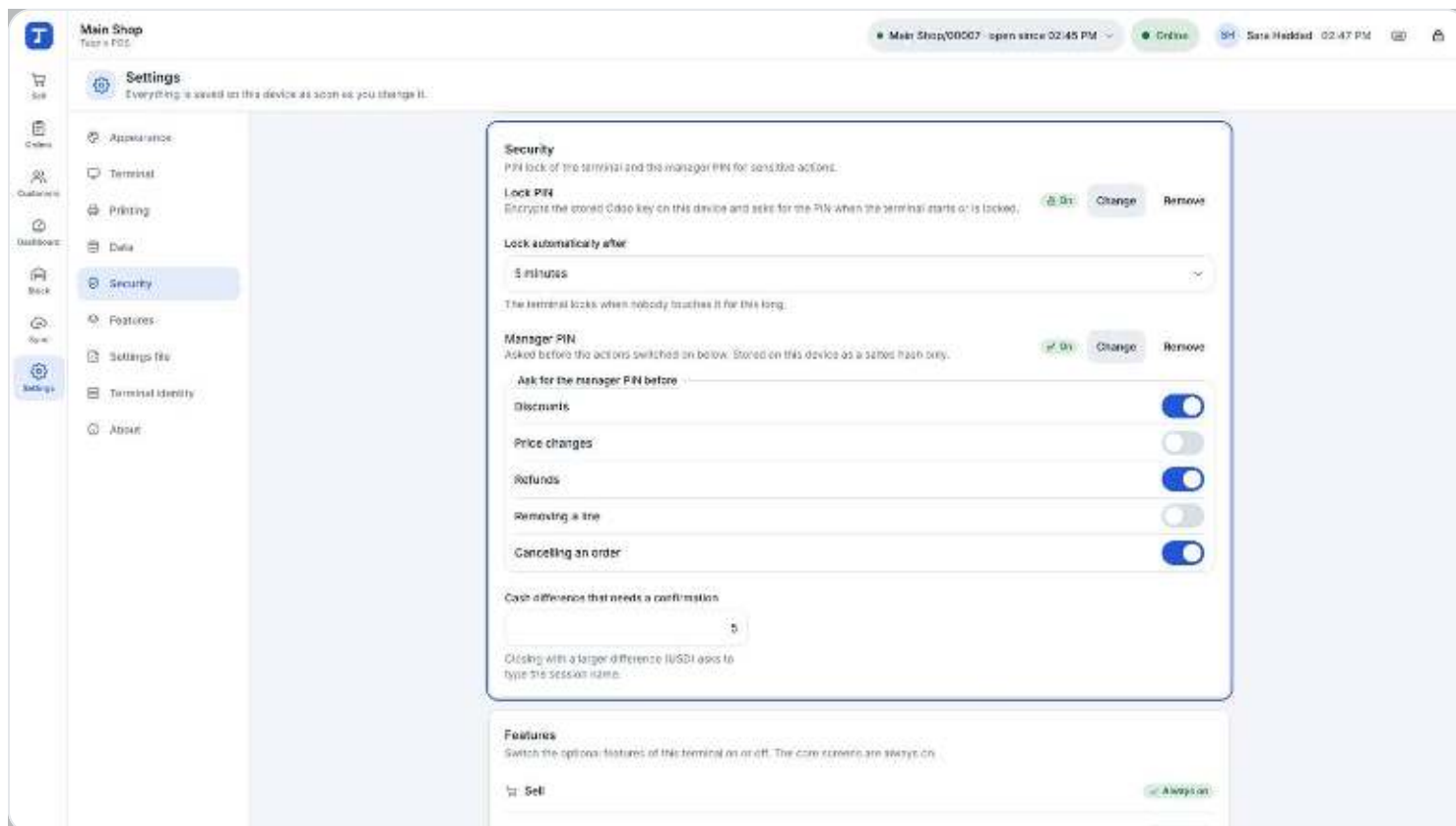
Tests. `npm test` runs Vitest (jsdom, an in-memory IndexedDB, a fake Odoo in the test process): 3,336 pass and 66 live tests that need a running Odoo 19 are skipped. `npm run typecheck` runs strict TypeScript and `npm run lint` ESLint. To run the live tests on your own Odoo 19 test database set `POS_TEST_URL` , `POS_TEST_DB` , `POS_TEST_LOGIN` , `POS_TEST_SECRET` (an API key) and `POS_TEST_ADMIN_SECRET` (an administrator's key) and run `npx vitest run tests/live` ; some also need `psql` access (`PGPASSWORD`). They open and close sessions: never run them on a production database.

25. Security

In plain words: The terminal keeps an Odoo key on the device and passes your calls through a small proxy. This chapter says what is protected and how, and what is not.

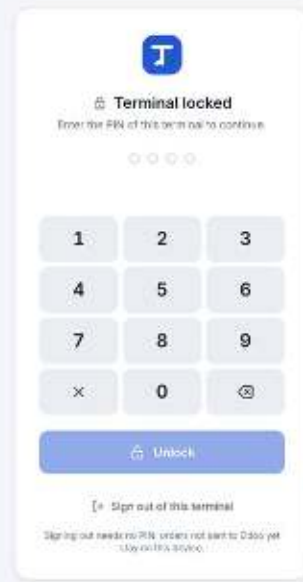
The key on the device. The browser keeps the Odoo key in IndexedDB. Without a lock PIN it is stored as it is. With a lock PIN it is encrypted (the PIN goes through PBKDF2-SHA-256 to an AES-GCM key) and the decrypted copy lives only in memory. The PIN is asked at start and after an automatic lock.

Important: The honest limit. Someone who copies the browser profile can try PINs offline, and the counter of wrong tries lives in the same profile. The PIN protects against casual access to a till, not against a thief with the disk. Use one API key for each terminal from a user with only the Point of Sale group, and revoke it in Odoo when a device is lost.



Settings, Security: Lock PIN On, lock after 5 minutes, Manager PIN On with three switches on, difference threshold 5.

Protection	How it works
Lock PIN	6 to 12 digits, set in Setup or Settings → Security . Lock automatically after Never, 1, 2, 5, 10, 15 or 30 minutes, and the lock button in the top bar. After 5 wrong PINs the terminal makes you wait, 30 seconds first, doubling up to 15 minutes.
Manager PIN	4 to 12 digits, stored only as a salted PBKDF2-SHA-256 hash. Asked before the actions you switch on: Discounts, Price changes, Refunds, Removing a line, Cancelling an order ; nothing is asked until a PIN is set. An approval holds for 20 seconds for the same action on the same line or order. Wrong PINs bring a growing wait (up to 5 minutes). Switching a protection off, or importing a settings file that does, asks for the manager PIN.
Sign out of this terminal	Needs no PIN, so a locked till can always be released. It removes the login and the downloaded data; unsent orders stay on the device.



The lock screen: "Terminal locked", keypad, Unlock, and Sign out of this terminal.

The proxy. Every call of the browser goes to `/api/odoo` , built as a narrow door:

Rule	Detail
Allow-listed routes	<code>/jsonrpc</code> , <code>/json/2/<model>/<method></code> , <code>/web/webclient/version_info</code> , <code>/web/version</code> , <code>/web/database/list</code> , and product images (<code>/web/image/...</code> with a <code>unique</code> query only). The path is matched as it arrives, never decoded first.
Servers and addresses	Only the origins of <code>POS_ALLOWED_ORIGINS</code> (any origin only with <code>POS_ALLOW_ANY_ORIGIN=1</code>). Every address the host resolves to is checked and the connection is pinned to them (safe against DNS rebinding). Link-local and cloud-metadata addresses are always refused; private networks only with <code>POS_ALLOW_PRIVATE=1</code> or a listed origin.
Requests	Browser cookies are never forwarded, <code>Set-Cookie</code> never returns, redirects are not followed. The request must come from the app's own page (<code>Origin</code> against <code>Host</code> ; forwarded headers only with <code>POS_TRUST_PROXY=1</code>), as JSON, up to 25 MB, 60 second limit.
Failed logins	Counted for each client and Odoo server: 10 in 10 minutes make the proxy answer 429 for 10 minutes. A missing-rights error does not count. Clients are told apart only with <code>POS_TRUST_PROXY=1</code> ; otherwise all share one count, and 10 failed attempts by anyone pause the logins of every terminal for 10 minutes.
Logs	With <code>POS_PROXY_LOG=1</code> only method, path and status; never a body or a key.

Headers. Every page carries a Content-Security-Policy (`default-src 'self'` : everything from the app itself plus `POS_CONNECT_SRC` ; inline scripts and styles stay allowed because the framework needs them), `nosniff` , `Referrer-Policy: same-origin` , `X-Frame-Options: SAMEORIGIN` and a Permissions-Policy that gives camera, USB, serial and Bluetooth to the app only.

Important: `/api/odoo` has no login of its own. Anyone who can reach it can try credentials against your Odoo (within the limit above). On an internet-facing server use TLS and put the terminal behind a VPN, single sign-on or an IP allow-list.

Deployment checklist	Do this
TLS and access	HTTPS (chapter 4); internet-facing: VPN, single sign-on or IP allow-list in front of <code>/api/odoo</code> .
Origins	<code>POS_ALLOWED_ORIGINS</code> set to your Odoo; no <code>POS_ALLOW_ANY_ORIGIN</code> on a reachable server.
Proxy	<code>POS_TRUST_PROXY=1</code> only behind a proxy that overwrites <code>X-Forwarded-*</code> , as the nginx example does.
Keys	<code>POS_DISABLE_JSONRPC=1</code> ; one API key for each terminal, Point of Sale group only; revoke a lost device's key.
On the till	Lock PIN with automatic lock; manager PIN for discounts, price changes and refunds.
Exports	An exported order (JSON) holds sale data: keep it private. The diagnostics and settings files hold no keys, passwords or PINs.

26. How accurate is it

In plain words: This chapter says what was checked, how and where, and what was not checked. Read it with the next one, Limits.

Totals follow Odoo's point of sale. The terminal matched Odoo 19's own Python tax helpers on 667 of 667 reference scenarios, Odoo's own JavaScript on 664 of 667 (3 documented exceptions), and, in an independent replay of 1,149 orders sent to Odoo 19 as a cashier, Odoo's Python on every order (chapter 23 has the details). Where Odoo's own client and server disagree, the terminal does what Odoo's client does. It does not compute Python-code taxes or pricelists in another currency.

Offline guarantees. A paid order is written to the device before the receipt shows. It is sent once: Odoo matches it by its unique id, so a lost answer cannot create a second order. A synced order is removed from the device only after Odoo confirmed it and after the retention period (14 days by

default); an order Odoo has not confirmed is never removed by the terminal. This assumes the browser keeps its data: if the browser data is cleared, unsent orders are lost (chapter 12).

Performance, and where it was measured. All numbers come from one Apple Silicon Mac with Odoo, the app and the browser on the same computer and a window of 1366 × 768 (not a tablet). On a tablet and a real network the times will be longer.

What was measured	Result on that Mac
First full download of 20,000 products and 20,000 customers	24.5 seconds; IndexedDB 21 MB
Warm reload to a usable product grid	0.78 to 0.87 seconds
Product search	About 80 ms, including the 60 ms pause after typing
Scrolling the product grid	60 frames per second
Adding a product to the order	1.7 ms (median)
1,000 orders queued offline, then sent	169 seconds, in order, no duplicate
JavaScript of the app	1.75 MB raw, 511 KB gzip, in total
Starting offline from the first visit	Confirmed with the server stopped

The tests. `npm test` passes 3,336 tests; 66 more, which need a running Odoo 19, are skipped by default.

Area	Tests passed
Tax and price engine (1,783 comparison scenarios against Odoo 19 Python, JavaScript and server; 55 unit and property tests; 28 money regressions and fuzzed orders)	1,866
Connection, proxy, local database and sync (fake Odoo in the test process)	393
Themes and colour contrast	357
Screen logic (orders, customers, sync, settings, close, lock, diagnostics)	257
Order flow, receipt, barcodes, scanner	158
Interface components (sell, payment, settings, orders, close, sync, setup, lock, shell, no hard-coded text)	89
Restaurant (floor, kitchen changes, split, merge, tips)	85
Pro features (dashboard, stock, layaway, loyalty)	66
Languages (English and Arabic keys, plurals, formats)	54
Feature registry, PWA, others	11
Total	3,336

Important: Not verified. Receipt and kitchen printers, barcode scanners, cash drawers and payment terminals were not tried on real hardware. Only Chromium-class browsers (Chrome, Edge) were used; Safari and Firefox were not. Odoo Enterprise and Odoo versions other than 19 were not tested. The Docker image was not built and tested, and nothing was measured on a tablet. You are responsible for checking that receipts, taxes and reports meet the laws that apply to you, and for testing the terminal against your own Odoo before using it in a shop (licence point 5).

27. Limits

In plain words: This is what the terminal does not do or what was not verified, and what to do instead. Read it before you use the terminal in a shop.

Limit	What it means	What to do instead
Hardware not verified	Receipt and kitchen printers, barcode scanners, cash drawers and payment terminals were not tried on real hardware.	Test your own devices before opening. Use Print a test page .
Browser printing only	No raw ESC/POS, no Odoo IoT Box, no Epson ePOS network printing. Chrome prints without a dialog only with <code>--kiosk-printing</code> . Each kitchen station needs a printer the operating system can print to, or its own browser running the terminal.	Install the printers in the operating system and start Chrome with <code>--kiosk-printing</code> .
Browsers	Chrome and Edge were used. Safari and Firefox were not verified. Background Sync is a Chromium extra: orders are sent while the terminal page is open.	Use Chrome or Edge and keep the terminal open.
Rights of a plain cashier	Cannot create customers (Contacts / Creation), save free-text option values (Products / Create), record cash in and out (Accounting / Invoicing) or print Odoo's full Z report. Lot or serial products need Inventory / User .	Grant the right, or do it in Odoo (chapter 5).
No cash in / out button	The terminal has no screen for cash moves.	Record them in Odoo.
Restaurant	No courses ("fire course"). The floor plan is edited in Odoo. No tips after a card payment (capture). Cancelling a layaway with a refund of the deposit is not done here.	Do these in Odoo; add the tip before payment.
Loyalty	Supported: loyalty cards and promo codes. Not supported: gift cards, eWallet, coupons, next-order coupons, buy X get Y, automatic promotions, card codes.	Sell those in Odoo's own point of sale.

Products of an active gift card or eWallet programme are hidden.

Fiscal positions, taxes, pricelists	A fiscal position without tax lines is inconsistent in Odoo 19. Python-code taxes cannot be computed. Pricelists in another currency are ignored, as in Odoo's POS.	Map every tax, also to 0 %; use percentage, fixed, division or group taxes.
Not in this version	Several companies in one terminal, pricelists in several currencies, a login or PIN for each employee (<code>pos_hr</code>), payment-terminal integration, self-order and kiosk, online payment.	One terminal for each company; one Odoo user for each cashier.
Password logins	Odoo 19 deprecated <code>/jsonrpc</code> , used by password logins, and announced its removal in Odoo 20.	Use API keys (JSON-2) and <code>POS_DISABLE_JSONRPC=1</code> .
The lock PIN	A convenience lock; it does not stop someone who copies the browser profile (chapter 25).	One key for each terminal, Point of Sale group only; revoke it when a device is lost.
Docker, Odoo versions	The Dockerfile and compose file are provided, the image was not built and tested here. Odoo 19 Community was tested; Enterprise and other versions were not.	Build and test the image, and test on a copy of your own Odoo.
Ready-to-run app	Started on macOS with Node.js 24. In Chrome this build installed, cached all its files, started offline with the server stopped and connected to Odoo; the sale and sync checks ran on the build from the source (the same code). <code>start.bat</code> on Windows, Linux, and Node.js 20 and 22 were not tried.	Try it on your own computer first; if <code>start.bat</code> does not start, run <code>node run.cjs</code> in a Command Prompt.
Closing needs the	Odoo posts the session. A session with an open layaway or other open drafts cannot	Keep selling; close when online.

connection	be closed.	
Event log language	The lines of the Event log on the Sync screen are written in English, also when the terminal is in Arabic.	Read the order states and messages on the Orders screen; send the English lines to support when you ask for help.
Tablets, tiles	Performance was measured on a Mac, not on a tablet. Tiles show no stock, and the grid shows at most 1,200 tiles.	Measure on your tablets; switch off Show product images on slow ones; use the Stock screen and search.

28. Troubleshooting

In plain words: Find the message you see, read the cause and do what the last column says. For the Setup screen see the table in chapter 6.

Message or symptom	Cause	What to do
"This session was closed on another device. Open a new one." (red banner)	The session was closed in the back office or on another device.	Press Open a new session ; waiting orders go to it.
"This session is being closed: a manager must finish closing it in Odoo."	The session is in closing control.	A manager closes it in Odoo (chapter 20).
"Odoo refused the login ..." (red banner)	The key was revoked, the password changed or the user archived.	Press Enter new credentials ; unsent orders stay.
"Odoo cannot be reached. The terminal keeps working offline."	Network or Odoo is down.	Keep selling; orders are sent when Odoo answers.
"This terminal's own server does not answer."	The Node.js app of the terminal is stopped or unreachable; Odoo may be fine.	Check that the app runs and is reachable from the till.
"Node.js 20.9 or newer is required ..." when you start the ready-to-run app	Node.js is older than 20.9, or not installed.	Install the current LTS version from nodejs.org , open a new terminal and check <code>node -v</code> .
"Port 3000 is already in use ..."	The terminal is already running, or another program uses that port.	Stop it, or start with another port: <code>sh start.sh --port 3100</code> .
Another device cannot open the address of the ready-to-run app	It listens on this computer only (127.0.0.1) unless told otherwise.	Start it with <code>--host 0.0.0.0</code> behind HTTPS (chapter 4) and allow the port in the firewall of the computer.
An order in Errors with "missing right: Inventory/User" or another group	The Odoo user lacks a right.	An administrator grants it (chapter 5); press Retry sync .

"This order waits until the new customer exists in Odoo."	The customer was not created yet, or Odoo refused it for lack of Contacts / Creation .	Grant the right and retry the customer change in Sync.
"Odoo refused this record. Retrying will not help until the cause is fixed."	A rule of Odoo refuses the data.	Export JSON , fix the cause in Odoo, retry; or send the file to support.
"The order failed the safety check before it was sent ..."	Totals or payments do not add up; the terminal did not send it.	The order stays on the device. Export its JSON for support.
"Odoo saved the order as paid but could not validate its payment (it is named "/")."	Odoo could not validate the payment.	It is never sent again. Check its payments in the Odoo back office.
"Another device had this table open: Odoo merged both orders ..."	Two devices worked on one table; one paid while the other merged.	Both orders are kept, nothing was sent twice. Compare them in Odoo and discard the wrong one.
"No session is open in Odoo for this point of sale ..."	The session was closed or is stuck in closing control.	Open a new session or have a manager finish the closing.
"Too many failed logins ..."	Odoo waits a minute; the proxy refuses (429) for 10 minutes after 10 failed logins.	Wait. Without <code>POS_TRUST_PROXY=1</code> all tills share the count.
"The browser may delete the terminal data (unsent orders too) ..." / "The storage is almost full."	The browser did not promise to keep the data, or the device is nearly full.	Press Keep data on this device ; install the terminal as an app; sync waiting orders and free space.
The print dialog opens for every print	Chrome asks for each job.	Start Chrome with <code>--kiosk-printing</code> .
No install button; offline start fails	The address is not HTTPS (and not localhost).	Serve the terminal over HTTPS (chapter 4).

"No products here"	No product is "Available in POS", or the download is not finished.	Mark the products in Odoo, then Pull data now .
"A tax of this product cannot be computed offline ..."	A tax is computed by Python code.	Replace it with a supported tax (chapter 23).
"Some data is missing on the terminal. Download the data again ..."; wrong data	The local copy is incomplete or damaged.	Re-download everything in Sync, or Settings → About → Wipe, keep unsent orders .
Refund is not available	The order has not reached Odoo, or this device does not know its lines.	Send the order first; an order from another terminal is refunded there or in Odoo.
"Wrong PIN" and a wait	Too many wrong PINs.	Wait; or Sign out of this terminal (no PIN) and connect again.

29. Update and backup

In plain words: You update the terminal on the server and the tills pick the new version up by themselves. A till never changes version in the middle of a sale: the cashier decides when.

- 1 Put the new files in place of the old ones and keep your `.env`.
- 2 Ready-to-run app: unzip the new `taqnix-pos-terminal-webapp.zip` into a new folder, copy your `.env` into it, stop the old terminal and start the new one. From the source: run `npm ci` and `npm run build`, then restart the app (or rebuild the Docker image and start it again).
- 3 Each build has its own build id inside the service worker. Open terminals look for a new version every 10 minutes and when you press **Check for update** in **Settings → About**.
- 4 The new version downloads in the background and a banner appears: **A new version of the terminal is ready**. with **Later** and **Update**.

5

When the cashier presses **Update**, the terminal saves the open order, reloads into the new version, and the order is back.

Orders not yet sent, the connection, the lock PIN and the settings stay in the browser through an update. Updates of the 1.x versions are included in the price.

Important: Before you wipe the browser data of a till, or move the terminal to another Odoo server, send every order: open Sync and wait for "Everything is synced". The terminal refuses to connect to another server while orders are unsent. Wiping the browser data deletes the orders Odoo has not received.

Backup. The master data is in Odoo: back it up as usual. A till holds only a copy and the orders Odoo has not received. On the server, keep your `.env` and any change you made to the source. To set up a till like another one, **Export settings** on the old one and **Import settings** on the new one; the file has no connection, so run Setup on the new till.

30. Support

In plain words: Installation help and bug fixes go through the marketplace page where you bought the terminal, or by email to **support@taqnix.com**. Updates of the 1.x versions are included.

Write with your Odoo version, the browser and its version, the device and screen size, a screenshot or a short description of what you did, and the file from **Sync → Export diagnostics**. That file has no passwords, keys or PINs, no customer data and no order lines.

Licence (summary)	Detail
Grant	One purchase gives one company (the legal entity that bought it, with its branches and an unlimited number of terminals) the right to install, run and modify the source code for its own business, on its own or a hosting provider's servers. An agency or integrator deploys it for a client only if that client company has its own purchase.
Updates	The version you bought and all 1.x updates published for it.
Not allowed	Reselling, sub-licensing, publishing the source code, or offering the application (modified or not) as a hosted multi-tenant service or as a competing product or template.
Third-party parts	The engine files in <code>src/lib/engine/</code> (listed in <code>NOTICE</code>) are adapted from Odoo 19 and stay under the LGPL-3.0 (<code>LICENSE-LGPL-3.0.txt</code>). The fonts Inter and IBM Plex Sans Arabic are under the SIL Open Font License 1.1. The npm packages keep their own licences (MIT or Apache-2.0).
No warranty	The software is provided as it is. You must check that receipts, taxes and reports meet the laws of your business and country, and test the terminal against your own Odoo before using it in a shop.

The full text is `licensing/license.txt` . Odoo is a trademark of Odoo S.A.; this product is not made or endorsed by Odoo S.A.

Changelog

1.0.0 – Initial release

- Offline-first point of sale for Odoo 19 as an installable progressive web app (Next.js 16, React 19, TypeScript): no Odoo module to install.
- Setup with a connection test (JSON-2 with API key or JSON-RPC, proxy or direct mode), point-of-sale picker, opening cash count and session closing with cash difference and closing report.

- Sell screen: search, categories, favourites, barcode scanner, variants, combos, notes, discounts, pricelists, customers (also created offline when the user may), held orders.
- Payment with several methods, change, cash rounding, customer account and invoice flag; receipts 58 / 80 mm with tax letters, barcode and QR; browser printing.
- Offline data in IndexedDB with delta downloads; orders sent once, in order, with retries; Orders and Sync screens with reprint, refund, retry, export and diagnostics.
- Tax and price engine ported from Odoo 19, checked against 667 scenarios computed by Odoo and 1,149 orders sent to Odoo.
- Restaurant (when Odoo's Restaurant is installed): floor plan, table orders, kitchen tickets per printer, split bill, transfer, merge, tips.
- Pro screens: dashboard, stock lookup across warehouses, layaway with deposits, loyalty points and promo codes.
- Settings: 5 themes and brand colour, density, text size, English and Arabic (right to left), lock PIN, manager PIN.
- Security: allow-listed proxy with address checks and a failed-login limit, Content-Security-Policy, encrypted key with a lock PIN.
- Ready-to-run web app with launchers for Windows, macOS and Linux (only Node.js needed), and the complete source code.
- 3,336 automated tests; Dockerfile and nginx examples; documentation in English and Arabic.

© Taqnix. Licensed to one company for an unlimited number of terminals (licensing/license.txt); the tax engine files are LGPL-3.0.