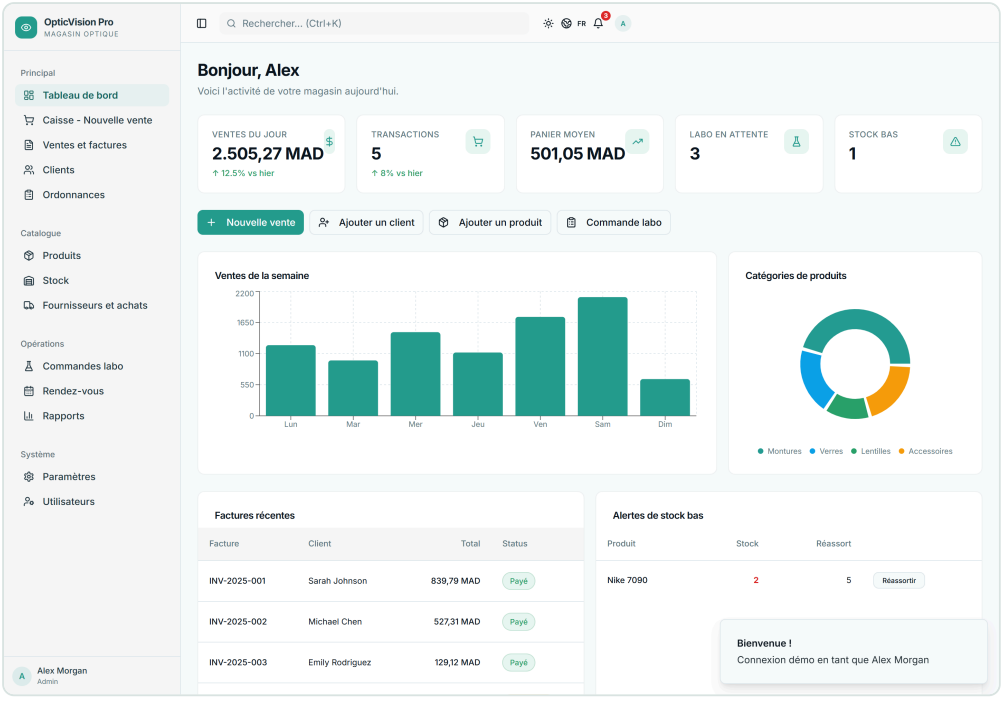
OpticPOS

OPTICAL STORE MANAGEMENT

Complete Manual — Installation, Accounts, Daily Use, Administration & Customization



Version 1.0.0 · React 18 + TypeScript + Vite
Interface: French (FR / EN / AR / ES / DE) · Default currency: MAD

Contents

CHAPTER 1

Overview

1.1 What OpticPOS is

OpticPOS is a point-of-sale and operations application for optical stores. It runs entirely in the browser as a single-page React application. There is no backend server and no database to install: all records live in the browser's `localStorage`, seeded with realistic demo data on first run.

That makes it ideal as a ready-to-customize product, a demo/evaluation build, or the frontend half of a system whose API you plan to add. It is not a production retail system as delivered — see section 1.4 and chapter 9.

1.2 Feature list

- Dashboard with sales, transactions, average basket, pending lab orders and low-stock KPIs
- POS checkout: product search, cart, tax, insurance coverage, cash/card payment, change due, receipt
- Sales and invoice history with per-invoice detail and refund status
- Customer records with balance, insurer, purchases, prescriptions and appointments
- Optical prescriptions (OD/OS sphere, cylinder, axis, addition, PD)
- Product catalogue for frames, lenses, contact lenses and accessories
- Inventory with low-stock/out-of-stock states and manual stock adjustment
- Suppliers and purchase orders
- Lab-order workflow in four columns: pending → in progress → ready → delivered
- Appointment list and calendar grouping
- Reports for revenue by payment method, inventory value and best sellers, with CSV export
- Role-based navigation for admin, cashier and technician
- Five interface languages, light/dark theme, MAD currency formatting

1.3 Technology stack

Layer	Technology
Build tool	Vite 8
UI	React 18, TypeScript
Styling	Tailwind CSS 3, shadcn/ui, Radix UI, Lucide icons
Routing	React Router 7 (`BrowserRouter`)
State	Zustand with `persist` middleware
Data	`MockDB` over `localStorage` (no backend)
Charts	Recharts

i18n	i18next + react-i18next, 5 languages
Tests	Vitest + Testing Library
Runtime	Node.js 20.19+ or 22.12+, npm 10+

1.4 What it is not (read first)

> **Before using real customer data**
> Authentication is simulated: the login form only checks that the e-mail contains `@` and the password is at least three characters. No server verifies anything. Role filtering hides sidebar items but does not block direct URLs. Payments are simulated, receipts are not printed, and stock is not decremented at checkout. Treat the current build as a demo/starter, and read chapter 9 in full before storing real patient or payment records.

CHAPTER 2

Installation

2.1 Prerequisites

- Node.js `20.19+` or `22.12+`
- npm `10+` (ships with Node.js)
- A current desktop browser (Chrome, Edge, Firefox or Safari)

Nothing else. No database, no web server, no Docker.

2.2 Install Node.js

Download and install from `https://nodejs.org/` (choose the LTS build). Then open a terminal *inside the OpticPOS folder* — the folder that contains `package.json` — and confirm both tools respond:

```
node --version
npm --version
```

The project also ships `.nvmrc` and `.node-version`, so `nvm use` or `fnm use` selects the right version automatically if you use a version manager.

2.3 One-command setup

Recommended. This checks the environment, installs dependencies, then runs lint, tests and a production build so you know the copy you received is intact:

```
npm run setup
```

Faster, skipping the verification stage:

```
npm run setup -- --skip-verify
```

Install and immediately launch:

```
npm run setup:start
```

2.4 Platform shortcuts

If you would rather not use a terminal, the project includes double-clickable scripts:

Platform	Install only	Install and run
Windows	`install-windows.bat`	`install-and-run-windows.bat`
macOS / Linux	`sh install-mac-linux.sh`	`sh install-and-run-mac-linux.sh`

2.5 Start the app

```
npm run dev
```

Vite prints the local URL. The configured default is:

```
http://localhost:8080/
```

> **If port 8080 is busy**
> Vite automatically picks the next free port (8081, 8082, …) and prints it. Always open the URL that the terminal actually shows — `localhost:8080` and `localhost:8081` keep *separate* browser storage, so opening the wrong one looks like your data disappeared.

2.6 Production build

```
npm run build     # writes optimized static files to dist/
npm run preview   # serves dist/ locally to check it
```

The contents of `dist/` can be hosted on any static host — Vercel, Netlify, Cloudflare Pages, nginx, Apache, an S3 bucket.

> **SPA fallback is required**
> Routing uses browser history. The host must serve `index.html` for unknown paths, otherwise refreshing a deep link such as `/customers/c1` or `/sales/inv1` returns a 404 from the server.

2.7 Verification commands

Command	What it does
`npm run doctor`	Checks Node, npm, `package.json`, lockfile and `node_modules`
`npm run lint`	ESLint over the whole project
`npm test`	Vitest, single run
`npm run test:watch`	Vitest in watch mode
`npm run build`	Production build
`npm run verify`	Lint, then test, then build — the release gate

2.8 Install problems

- **Wrong Node version** — install `20.19+` or `22.12+`; older versions fail on Vite 8.
- **Command not found / wrong folder** — run commands from the folder containing `package.json`.
- **Interrupted install** — delete only this project's `node_modules`, then run `npm run setup` again.
- **Anything else** — run `npm run doctor` first; it names the failing prerequisite.

CHAPTER 3

Accounts and signing in

3.1 The login screen

Opening the app sends you to `/login`. The screen has a credentials form, a role selector, a "remember me" checkbox, three quick demo buttons, and a language switcher in the top-right corner.

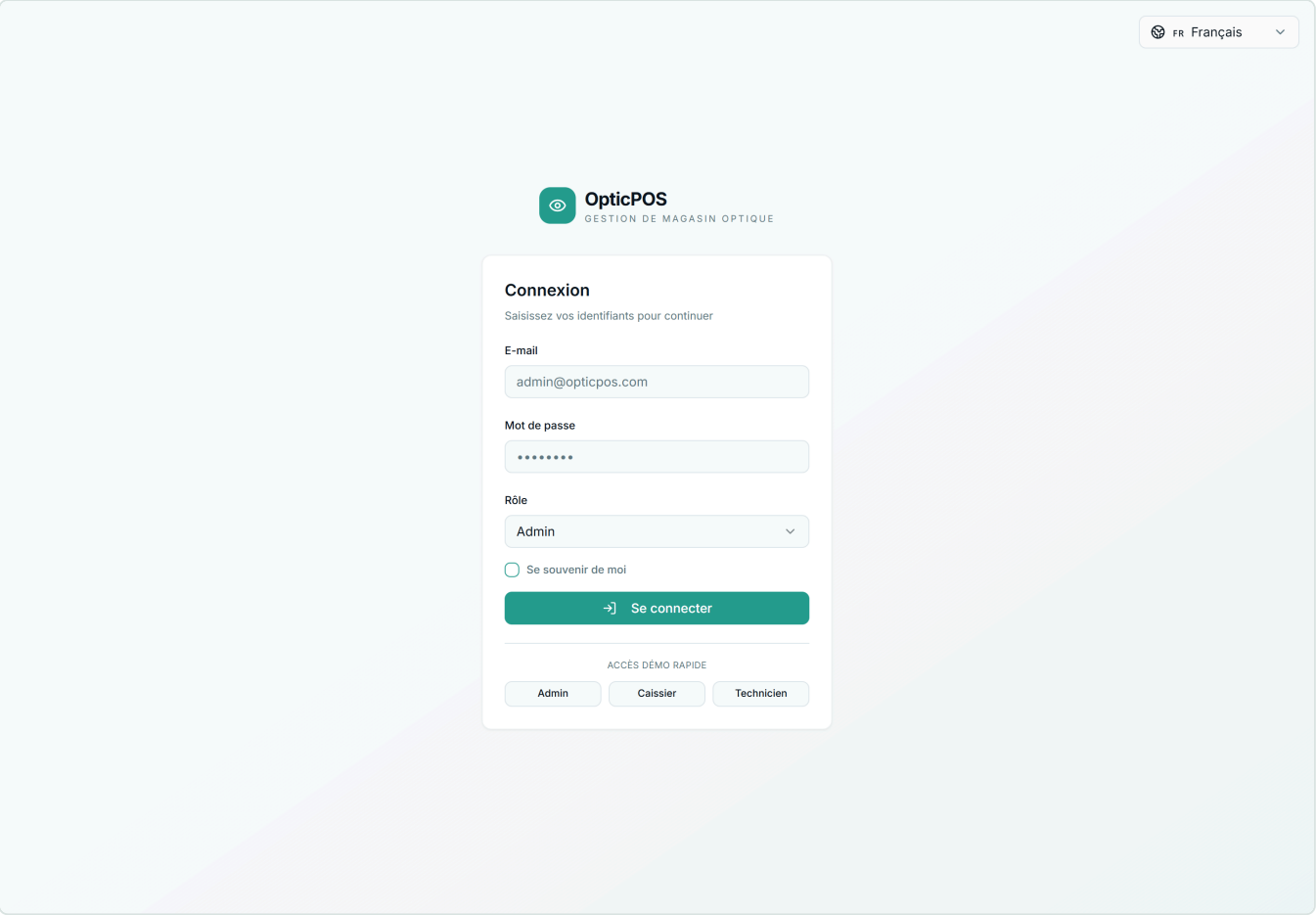


Figure 3.1 — Login screen with quick demo access for the three roles

3.2 Quick demo accounts (fastest way in)

Under **Accès démo rapide**, select one button. You are signed in instantly, with no password:

Button	Name	E-mail	Role
Admin	Alex Morgan	`admin@opticpos.com`	Administrator — everything
Caissier	Jordan Lee	`cashier@opticpos.com`	Cashier — sales floor
Technicien	Casey Wright	`tech@opticpos.com`	Technician — lab work

3.3 Create an account with the form

There is no separate sign-up page: the login form *is* the account-creation path in this build. Submitting it creates a local user session under the identity you choose.

1. Enter any syntactically valid e-mail address (it only needs to contain `@`).
2. Enter a password of at least three characters.
3. Pick the role — **Admin**, **Caissier** or **Technicien**. This decides what the sidebar shows.
4. Optionally tick **Se souvenir de moi**.
5. Select **Se connecter**.

Your e-mail is kept, and the display name is taken from the demo identity for the role you chose. The session is written to the `opticpos-auth` key in `localStorage`, so you stay signed in across reloads until you log out.

> **About "Se souvenir de moi"**
> The checkbox is displayed but changes nothing: the session is already persisted locally either way.

> **No password is ever verified**
> Any e-mail and any 3-character password work, and any visitor can choose the Admin role. This is demo behaviour by design. Section 3.8 lists what to replace before real use.

3.4 Create staff accounts in the app

Sign in as Admin and open **Utilisateurs**. The screen lists staff accounts with name, e-mail, role and status.

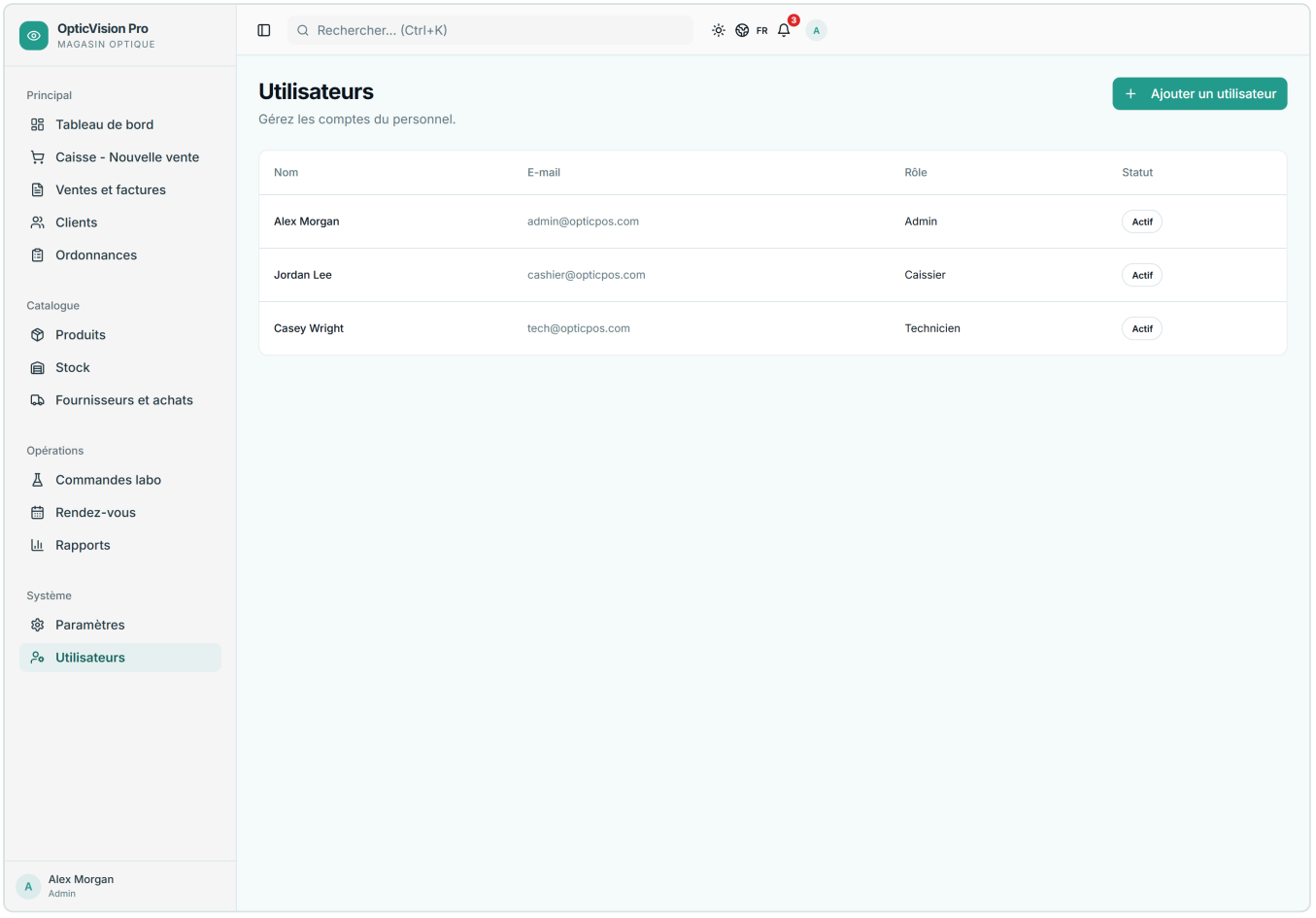


Figure 3.2 — Staff accounts under **Utilisateurs**

1. Select **Ajouter un utilisateur**.
2. Enter the name and e-mail (both required).
3. Choose **Admin**, **Caissier** or **Technicien**.
4. Select **Ajouter l'utilisateur**.

> **These records are display-only**
> Users created here live in React component state. They disappear on reload, they are not login credentials, and there is no password, invite, edit, disable or delete flow. Use this screen to demonstrate staff management, not to provision real access.

3.5 Change the built-in demo identities

To rename the three demo accounts, edit the map at the top of `src/pages/Login.tsx`:

```
const demoUsers: Record<Role, { name: string; email: string }> = {
  admin:      { name: "Alex Morgan",  email: "admin@opticpos.com" },
  cashier:    { name: "Jordan Lee",   email: "cashier@opticpos.com" },
  technician: { name: "Casey Wright", email: "tech@opticpos.com" },
};
```

The same three names are duplicated as the starting rows of the Users screen in `src/pages/Users.tsx` (`initialUsers`) — change both to stay consistent.

3.6 Roles and permissions

The role picked at login filters the sidebar:

Navigation item	Admin	Caissier	Technicien
Tableau de bord	✓	✓	✓
Caisse - Nouvelle vente	✓	✓	—
Ventes et factures	✓	✓	—
Clients	✓	✓	✓
Ordonnances	✓	✓	✓
Produits	✓	✓	—
Stock	✓	—	—
Fournisseurs et achats	✓	—	—
Commandes labo	✓	—	✓
Rendez-vous	✓	✓	✓
Rapports	✓	—	—
Paramètres	✓	—	—

Utilisateurs	✓	—	—

This matrix lives in `src/components/layout/AppSidebar.tsx`, in the `roles` array of each navigation entry. Routes themselves are *not* guarded: a signed-in cashier who types `/reports` still reaches the page.

3.7 Signing out

Open the avatar menu at the top right and select **Déconnexion**. This clears the local authentication state and returns you to `/login`. Cart contents and store settings survive logout; only the session is cleared.

3.8 Adding real authentication

When you connect a backend, the changes are concentrated in a few places:

1. Replace the `handleSubmit` body in `src/pages/Login.tsx` with a call to your auth API; store the returned user and token instead of a fabricated identity.
2. Keep `useAuthStore` (`src/lib/store.ts`) as the session holder, but store a token reference rather than trusting client-side role values.
3. Add a route guard around the `AppLayout` element in `src/App.tsx` that checks both authentication *and* role per route.
4. Enforce every permission again on the server. Client-side role filtering is a convenience, never a security boundary.

CHAPTER 4

The interface

4.1 Application shell

Every signed-in page shares the same shell: a role-filtered sidebar on the left, a top bar across the top, and the page content in the middle.

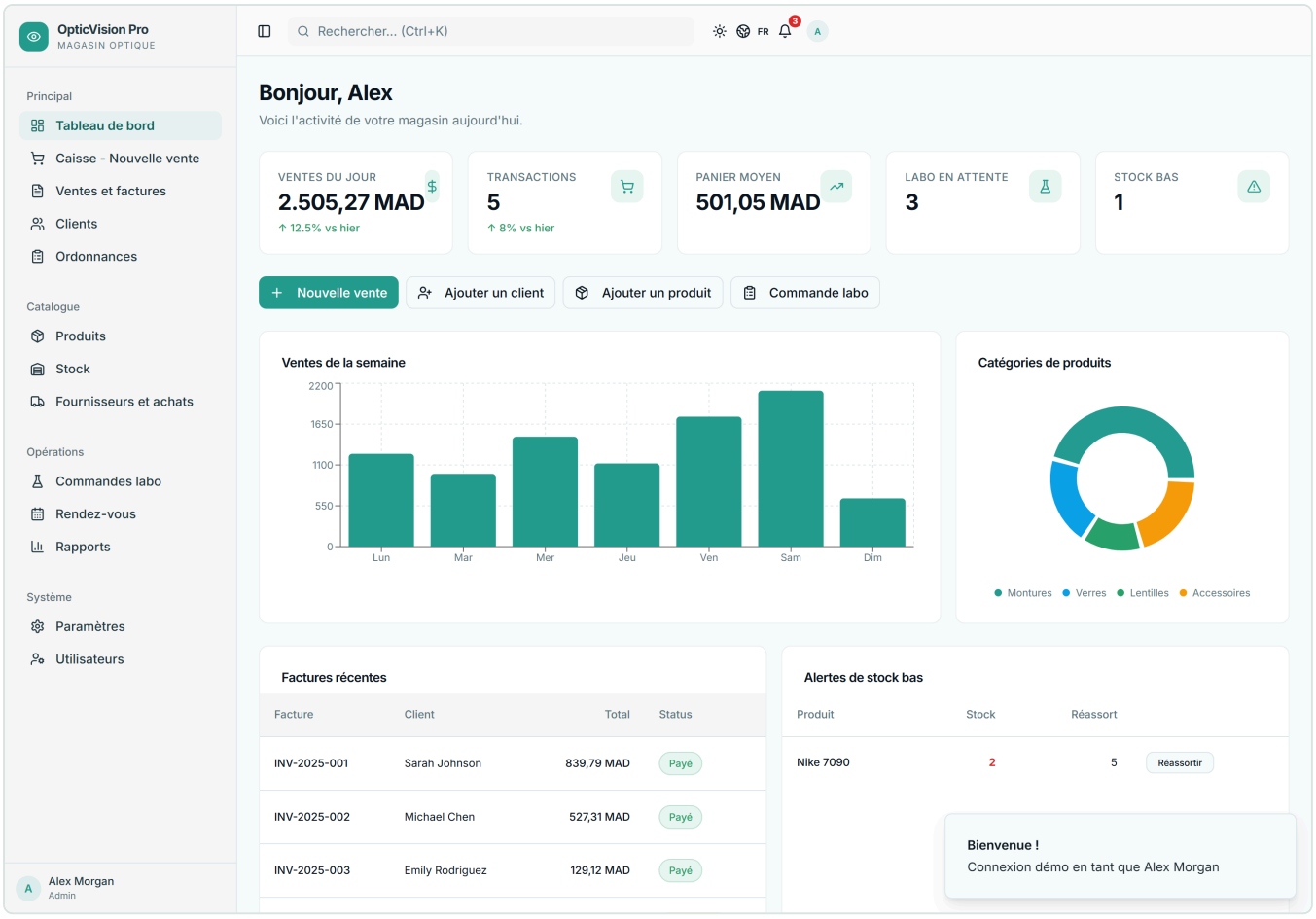


Figure 4.1 — The shell: sidebar, top bar, page content

4.2 Sidebar

Navigation is grouped into four sections — **Principal**, **Catalogue**, **Opérations** and **Système**. Items you do not have the role for are hidden entirely rather than disabled. The footer shows the signed-in name and role.

4.3 Top bar

Control	Behaviour
Panel icon	Collapses and expands the sidebar
Rechercher... (Ctrl+K)	Visual placeholder — global search is not wired up yet
Sun / moon icon	Switches between light and dark theme; the choice persists

Globe + language code	Switches the interface language
Bell	Static demo badge — no notification list opens
Avatar	Menu with **Profil** (opens Settings) and **Déconnexion**

4.4 Language switching

Five languages ship with the app: **Français** (default), **English**, **العربية**, **Español** and **Deutsch**. Switch from the globe control in the top bar, from the switcher on the login screen, or from **Paramètres** → **Langue**. The choice is stored under the `opticpos-language` key and restored on the next visit. Arabic additionally sets the document direction to right-to-left.

4.5 Light and dark theme

The sun/moon button toggles the theme instantly and stores it with the rest of the store settings. Every colour in the app comes from CSS custom properties, so both themes stay consistent across charts, badges and the sidebar.

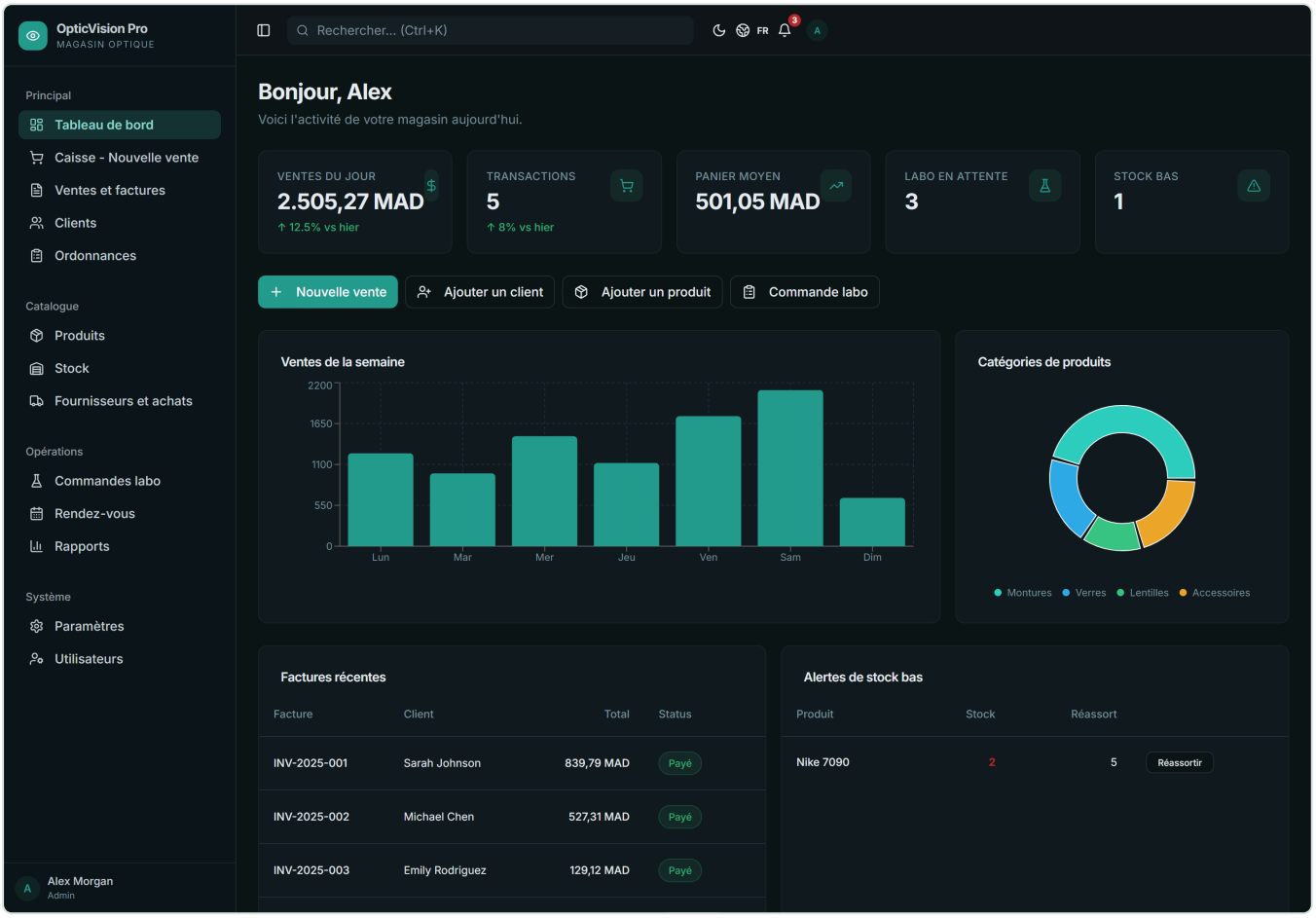


Figure 4.2 — The same dashboard in dark theme

CHAPTER 5

Daily use

5.1 Dashboard

Open **Tableau de bord**. The five KPI cards show sales total, transaction count, average basket, pending lab orders and the number of low-stock products. Below them are four shortcut buttons, a weekly sales chart, a category breakdown, recent invoices and stock alerts.

- **Nouvelle vente** opens the POS
- **Ajouter un client** opens the customer list
- **Ajouter un produit** opens the catalogue
- **Commande labo** opens lab orders
- Clicking a recent invoice opens its detail page
- **Réassortir** on a stock alert opens purchasing

> **Two demo caveats**
> The weekly bar chart uses fixed sample values, and the "today" totals actually sum every stored invoice — there is no date filter yet.

5.2 Complete a sale (POS)

Open **Caisse - Nouvelle vente**. The screen has three columns: product browser, cart, payment panel.

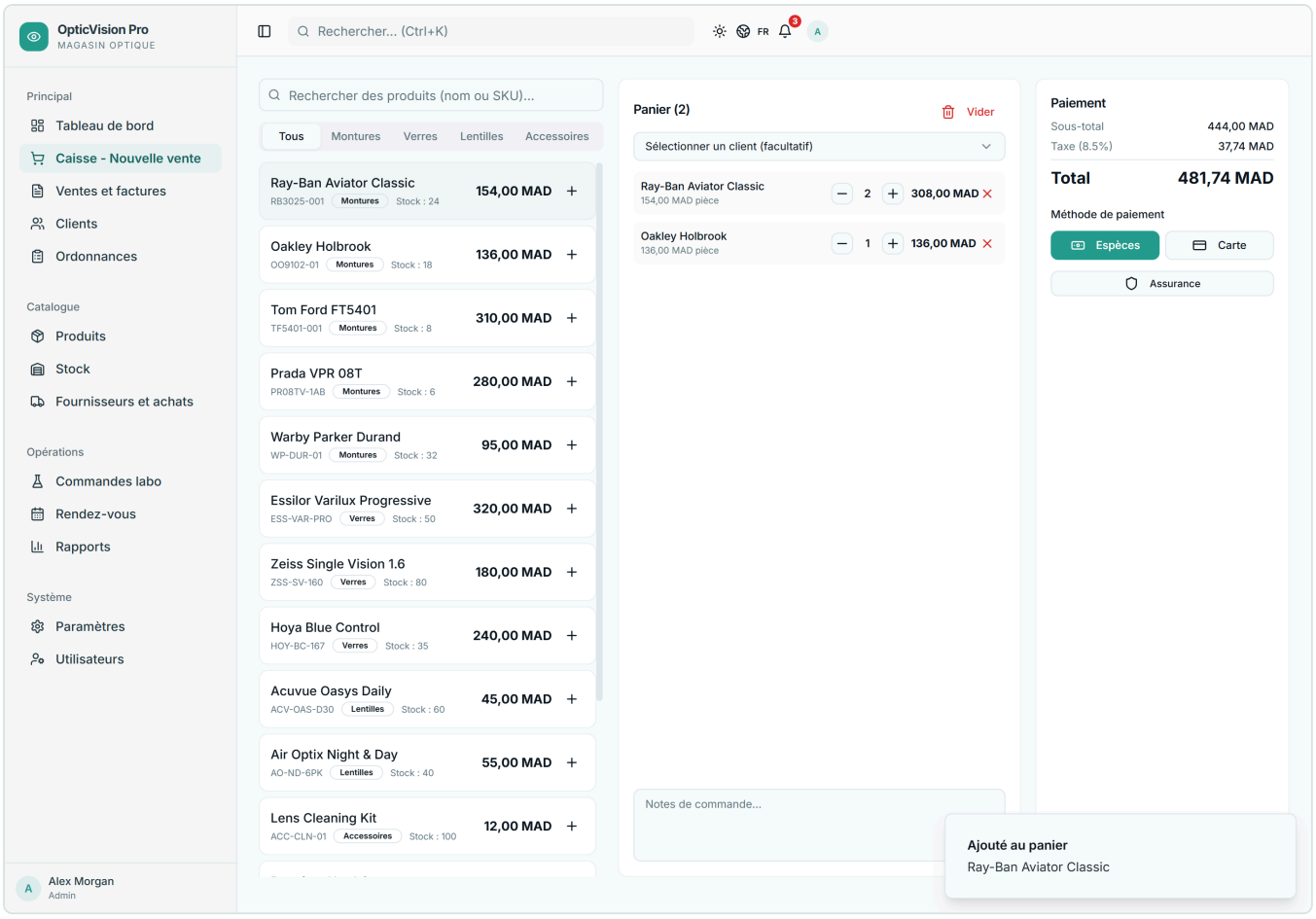


Figure 5.1 — POS: product browser (left), cart (centre), payment panel (right)

ADD PRODUCTS

1. Search by product name or SKU, or use the category tabs: **Tous**, **Montures**, **Verres**, **Lentilles**, **Accessoires**.
2. Select a product card to add it to the cart. Adding it again increases the quantity.
3. Use `+` and `−` to change quantity, the × icon to remove one line, or **Vider** to empty the cart.
4. Optionally select an existing customer and enter order notes.

Products with zero stock are hidden from the browser.

REVIEW TOTALS

```
Subtotal  = sum((unit price × quantity) − line discount)
Tax       = subtotal × configured tax rate
Insurance = subtotal × coverage percentage
Total     = subtotal + tax − insurance
```

TAKE PAYMENT

1. Select **Espèces** or **Carte**.
2. Optionally enable **Assurance** and enter coverage from 0 to 100 percent.
3. Select **Encaisser**.
4. For cash, enter **Espèces reçues**. Completion stays disabled until the amount covers the total; change due is calculated automatically.
5. Select **Terminer la vente**.

A paid invoice is created and stored, the cart is cleared, and the receipt opens.

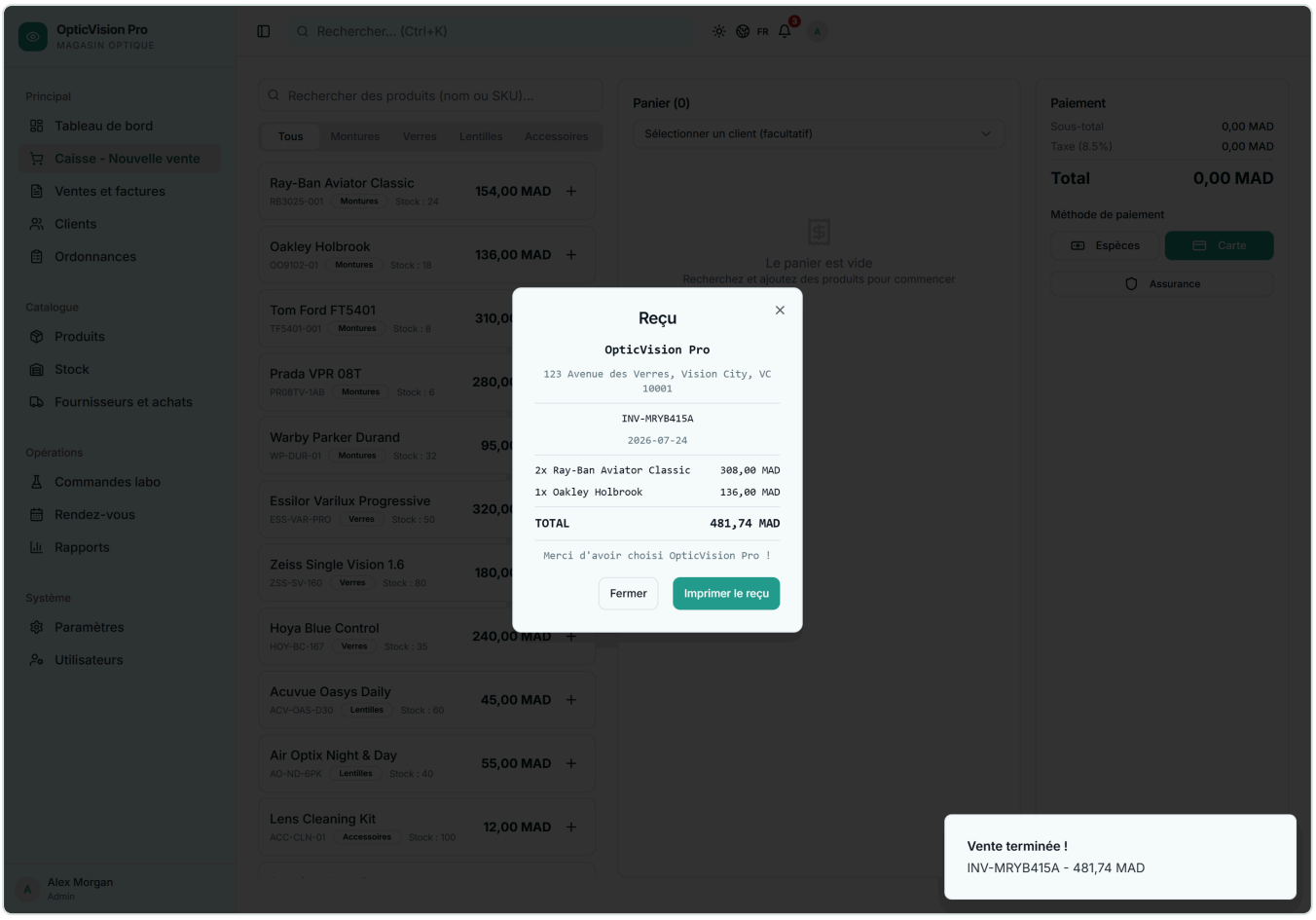


Figure 5.2 — Receipt after a completed sale; the store name, address and footer come from Settings

POS behaviour to know

The cart survives page reloads. Checkout does *not* decrement product stock. Insurance reduces the total but attaches no insurer or policy to the invoice. New invoices always record the cashier as `Admin`. Order notes and the selected prescription are not copied onto the invoice.

Imprimer le reçu

shows a confirmation instead of opening a print dialog.

5.3 Sales and invoices

Open **Ventes et factures**. Search by invoice number or customer name, and filter by **Payé**, **En attente**, **Partiel** or **Remboursé**. Select a row to open the invoice.

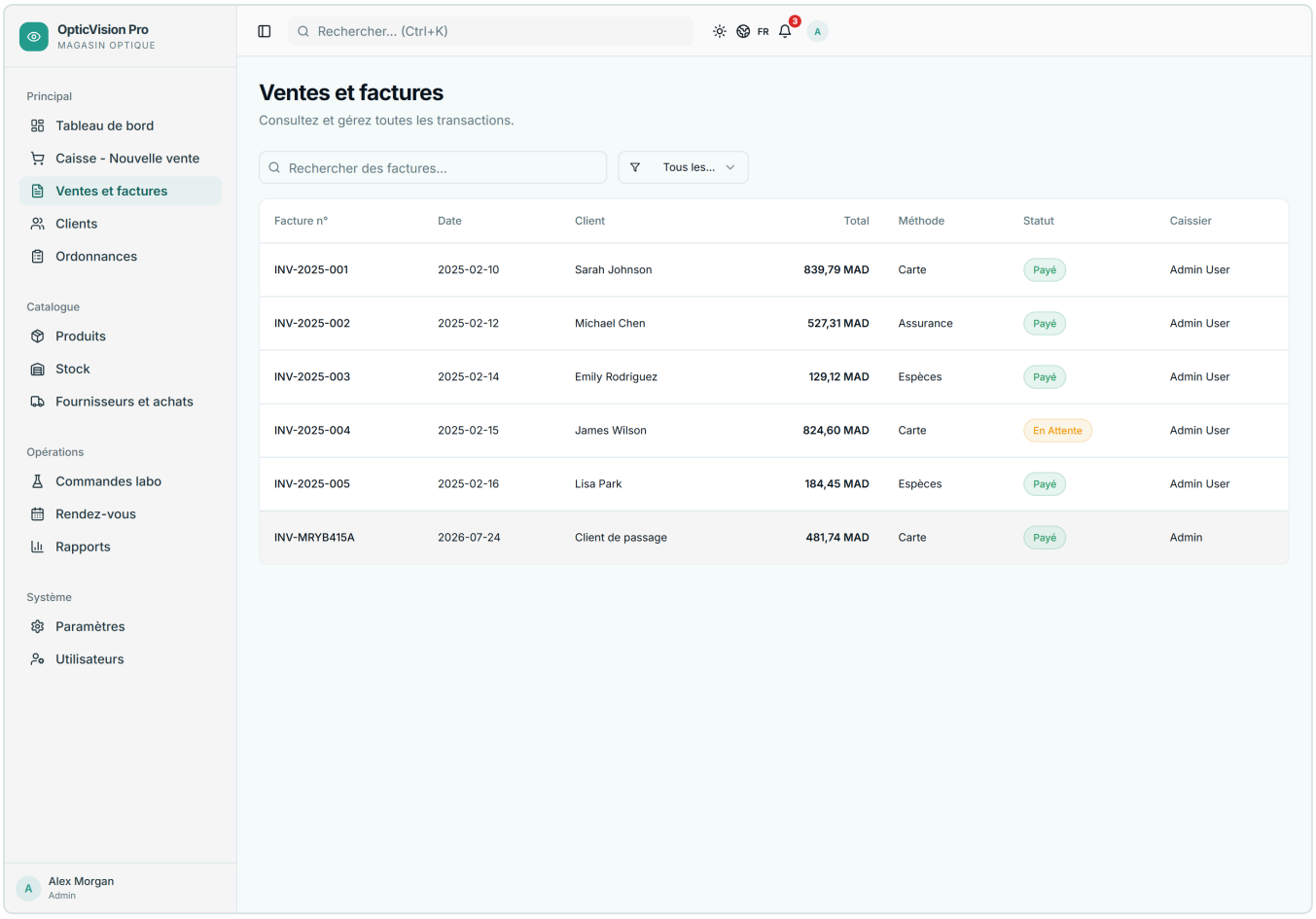


Figure 5.3 — Invoice history with search and status filters

The detail page lists products, quantities, prices, discounts, tax and total, and offers **Réimprimer le reçu**.

5.4 Refunds

On a non-refunded invoice select **Rembourser**, enter a reason and confirm.

> **What a refund does**
> It changes the local invoice status to `refunded` — nothing else. No accounting entry, no stock return, and the reason you typed is not retained.

5.5 Customers

Open **Clients**. Search by name or phone, and select a row to open the profile.

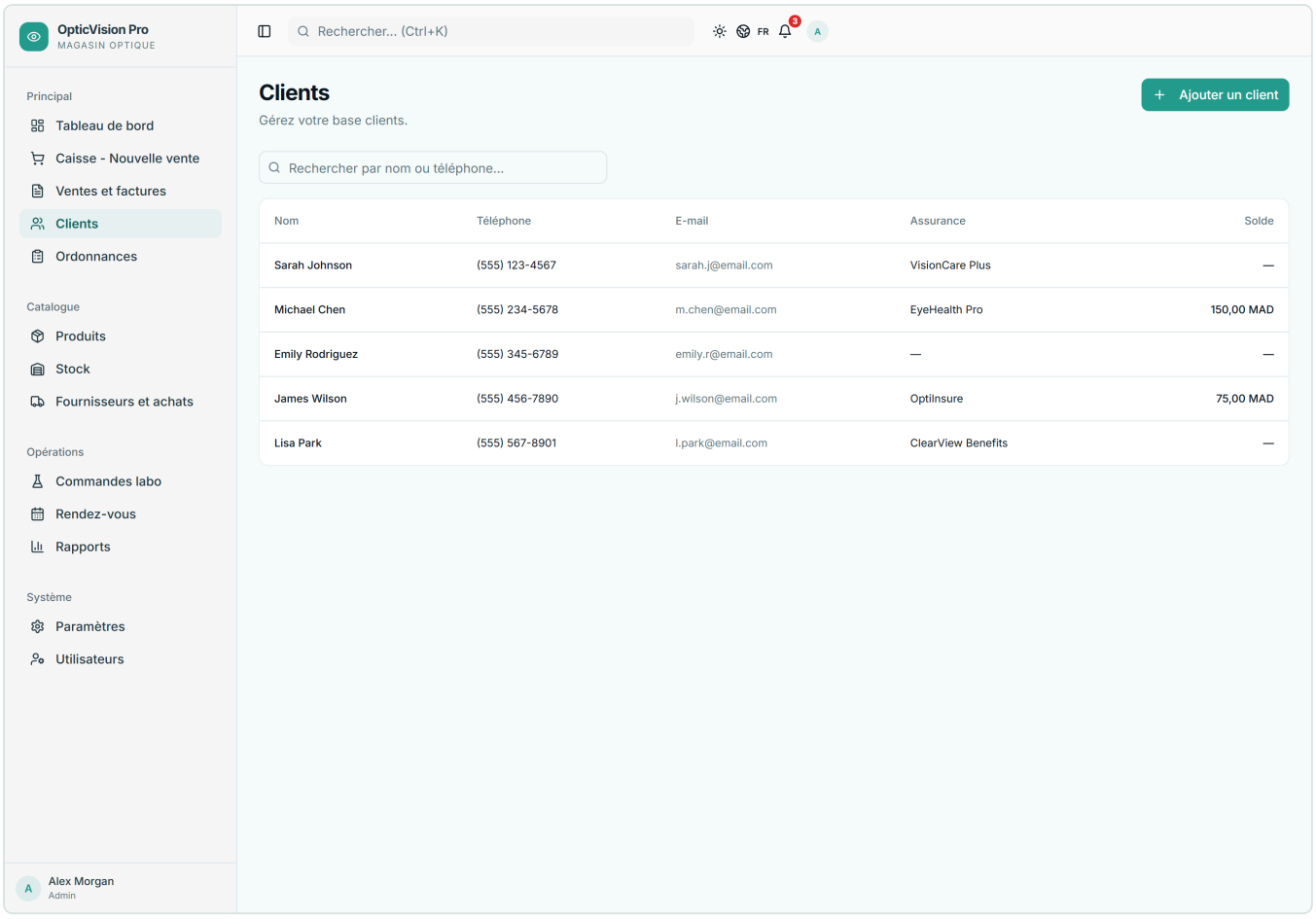


Figure 5.4 — Customer list with search

The profile shows contact details, insurer, outstanding balance and notes, with tabs for the customer's purchases, prescriptions and appointments. Selecting an invoice under **Achats** opens its detail page.

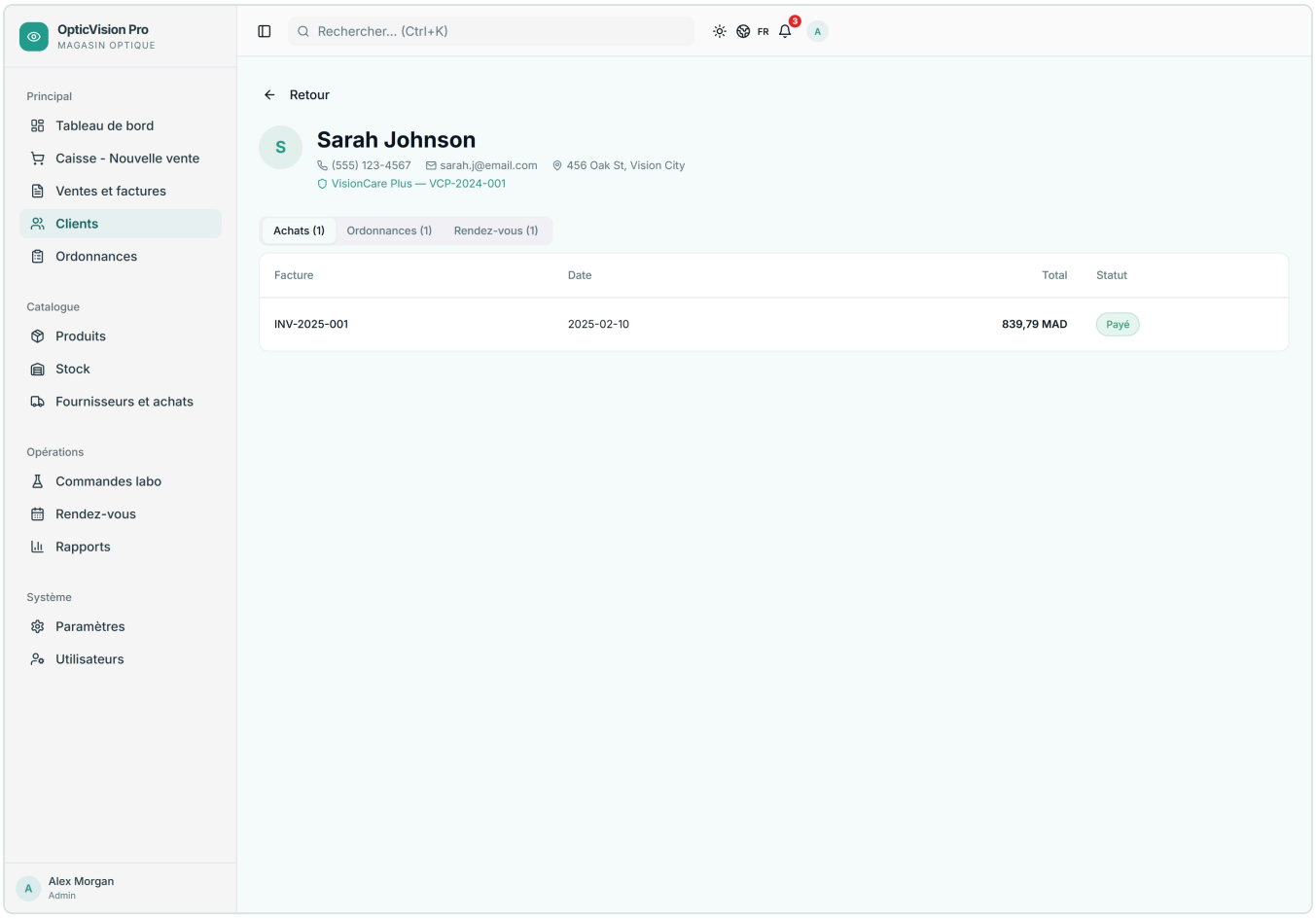


Figure 5.5 — Customer profile with linked records

ADD A CUSTOMER

1. Select **Ajouter un client**.
2. Enter name (required), e-mail, phone, address, insurer and notes.
3. Select **Ajouter le client**.

New customers get a generated ID, today's date, a zero balance and no policy number. Editing and deleting customers are not available in the UI.

5.6 Prescriptions

Open **Ordonnances**. The table shows customer, issue date, doctor, right-eye sphere, left-eye sphere, pupillary distance and expiry.

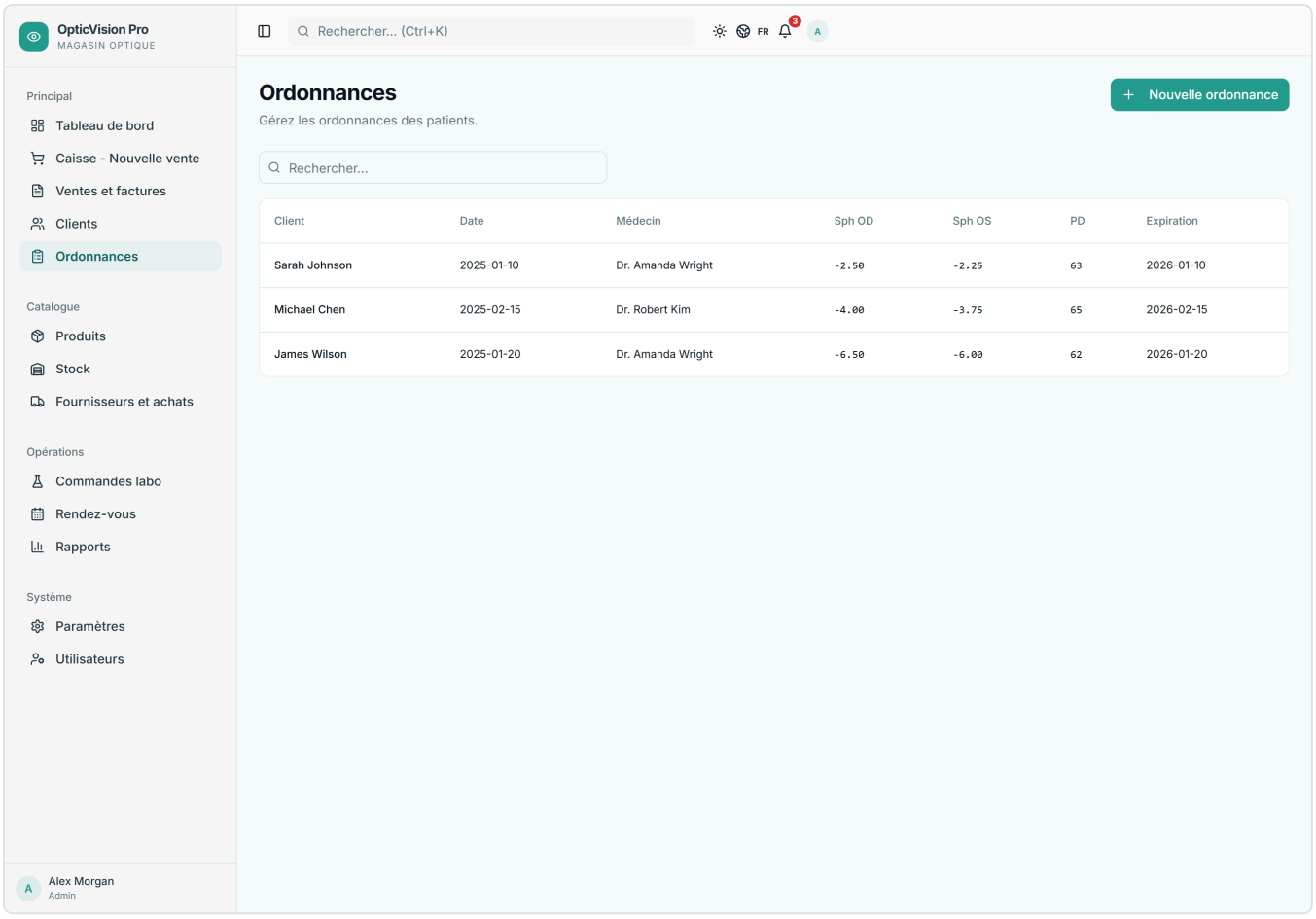


Figure 5.6 — Optical prescriptions

1. Select **Nouvelle ordonnance**.
2. Choose a customer and enter the doctor (both required).
3. Enter issue and expiry dates.
4. Enter OD (right eye) and OS (left eye): sphere, cylinder, axis, addition.
5. Enter PD and notes, then select **Enregistrer l'ordonnance**.

Optical ranges, date order and expiry at checkout are not validated in this build.

5.7 Lab orders

Available to admins and technicians under **Commandes labo**. Orders move through four columns: **En attente** → **En cours** → **Prêt** → **Livré**. Each card shows the order number, customer, lens information, frame, deadline, priority and notes; the action button advances the card one stage.

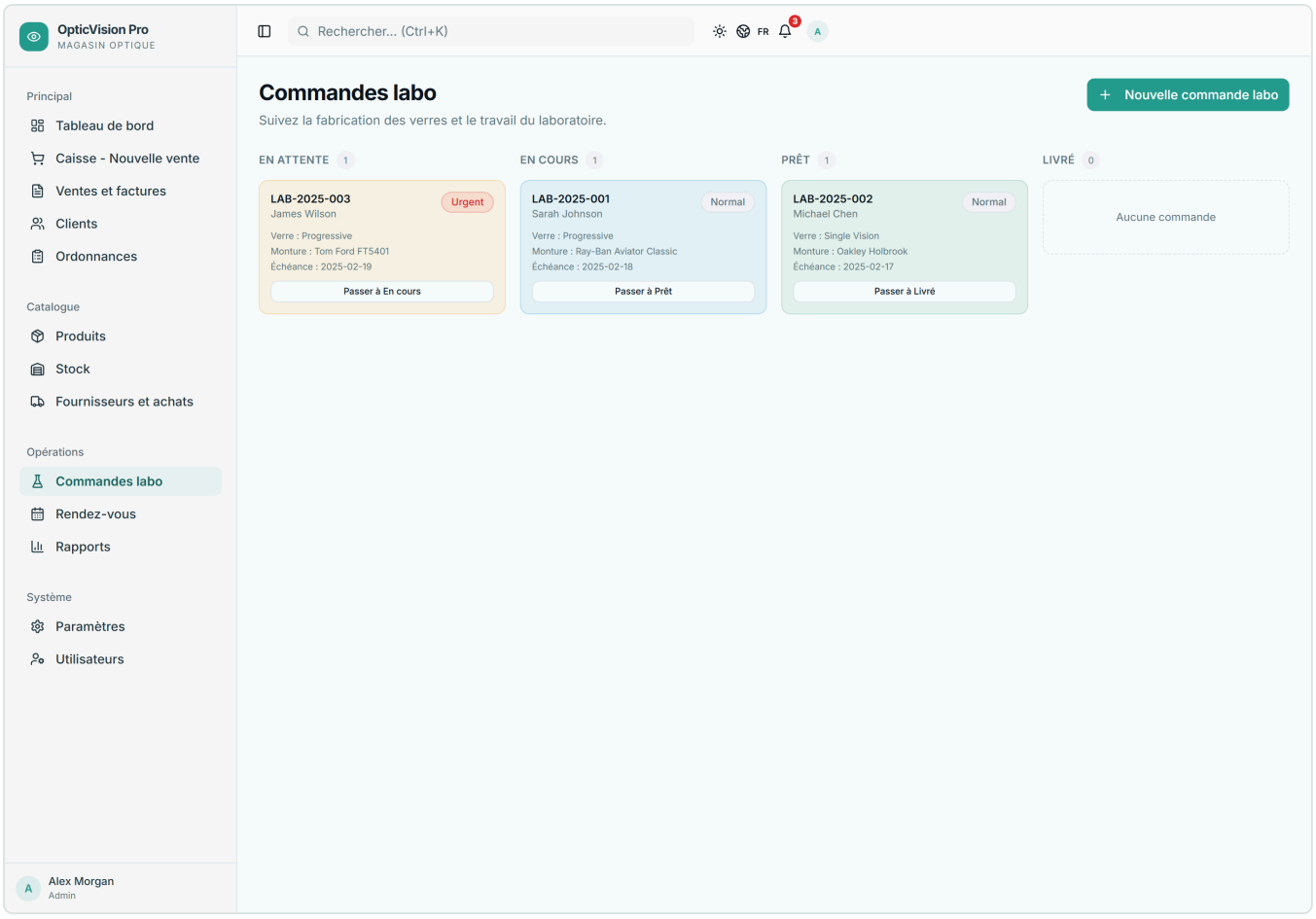


Figure 5.7 — Four-stage lab workflow

1. Select **Nouvelle commande labo**.
2. Choose a customer (required), and optionally one of that customer's prescriptions.
3. Enter lens type, coating and frame.
4. Choose normal or urgent priority, enter the due date and notes.
5. Select **Créer la commande**. The order starts in **En attente**.

5.8 Appointments

Open **Rendez-vous**. The **Liste** tab shows date, time, customer, type, staff and status; the **Calendrier** tab groups appointment cards by date (it is a grouped view, not an interactive month calendar).

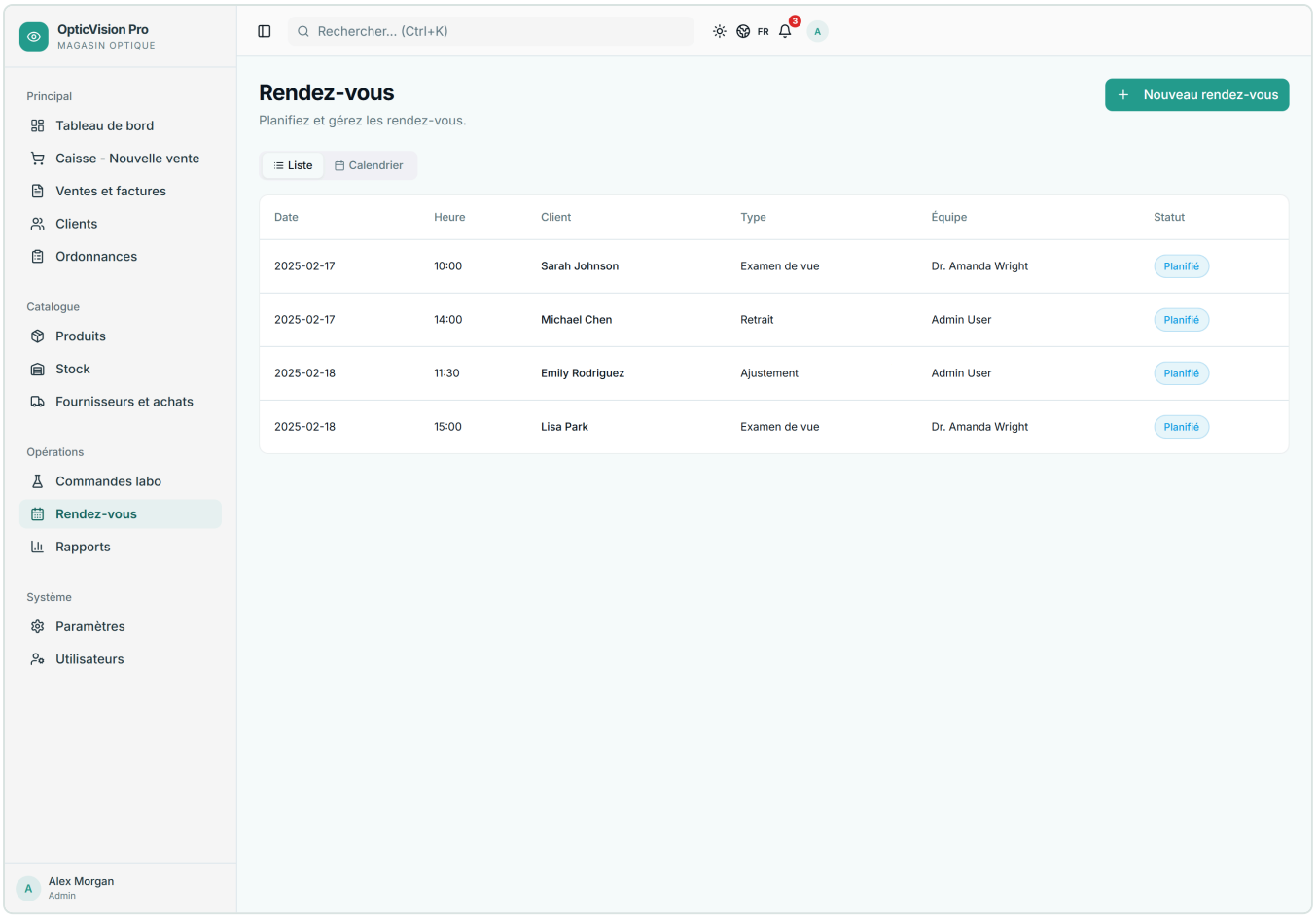


Figure 5.8 — Appointment list and view switcher

1. Select **Nouveau rendez-vous**.
2. Choose a customer.
3. Choose **Examen de vue**, **Retrait**, **Ajustement** or **Suivi**.
4. Enter date, time, staff and notes (customer, date and time are required).
5. Select **Planifier**.

New appointments are created as `scheduled`. Editing, cancelling and completing are not available in the UI.

CHAPTER 6

Administration

Everything in this chapter requires the Admin role.

6.1 Product catalogue

Open **Produits**. Products are split into tabs: **Montures** (frames), **Verres** (lenses), **Lentilles** (contacts) and **Accessoires**. Search matches name or SKU; each table shows name, SKU, brand, price, cost and stock.

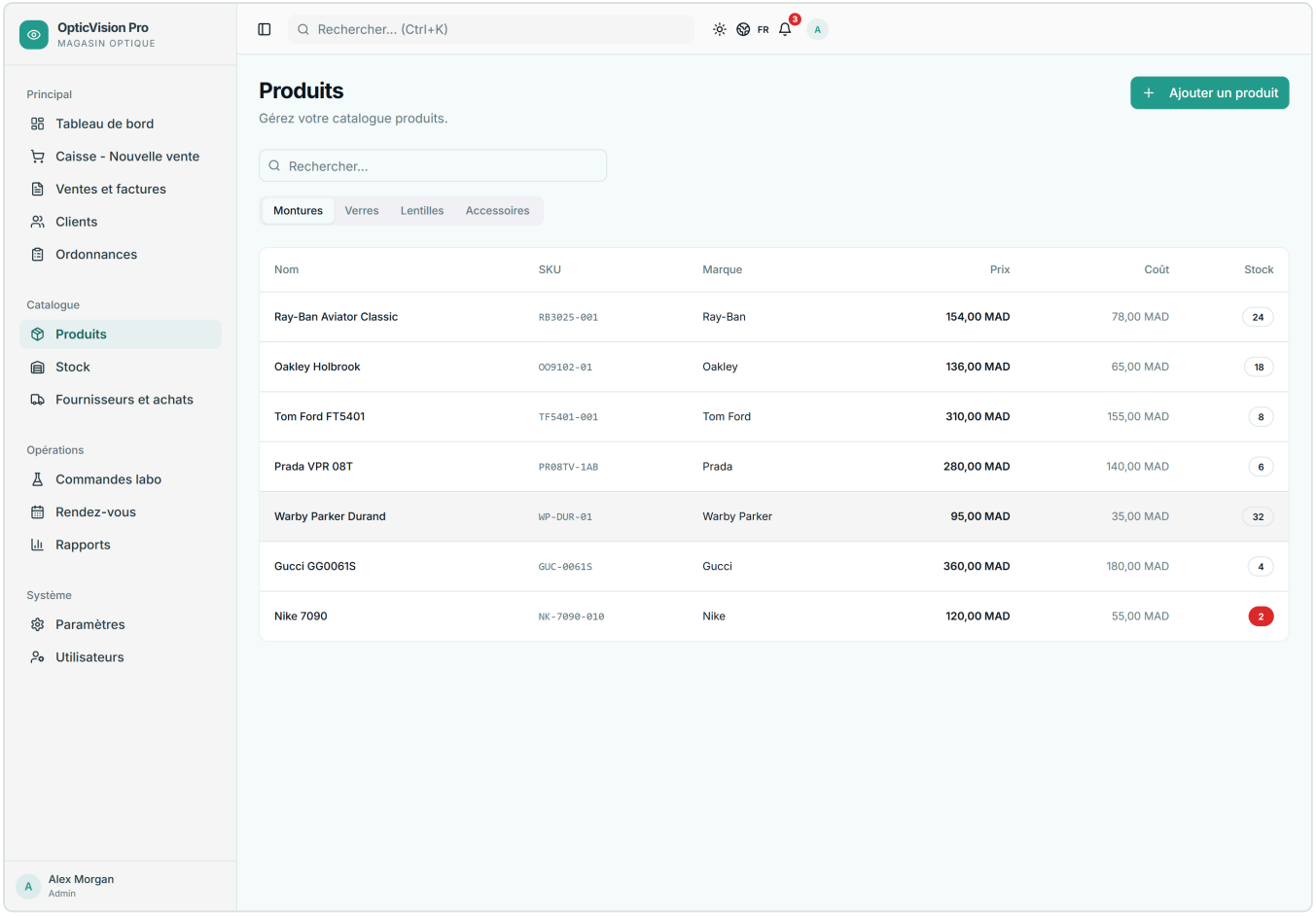


Figure 6.1 — Product catalogue by category

1. Choose the category tab, then **Ajouter un produit**.
2. Confirm the category.
3. Enter name, SKU, brand, description, price, cost, stock and reorder threshold.
4. Select **Ajouter le produit**.

Name, SKU and price are required. Category-specific optical attributes (frame size, lens index, base curve, …) exist in the data model but are not exposed by the create dialog. The catalogue supports create and read only — no edit or delete in the UI.

6.2 Inventory

Open **Stock**. Each row is flagged **Rupture** when stock is zero, **Stock bas** when it is at or below the reorder threshold, and **En stock** otherwise.

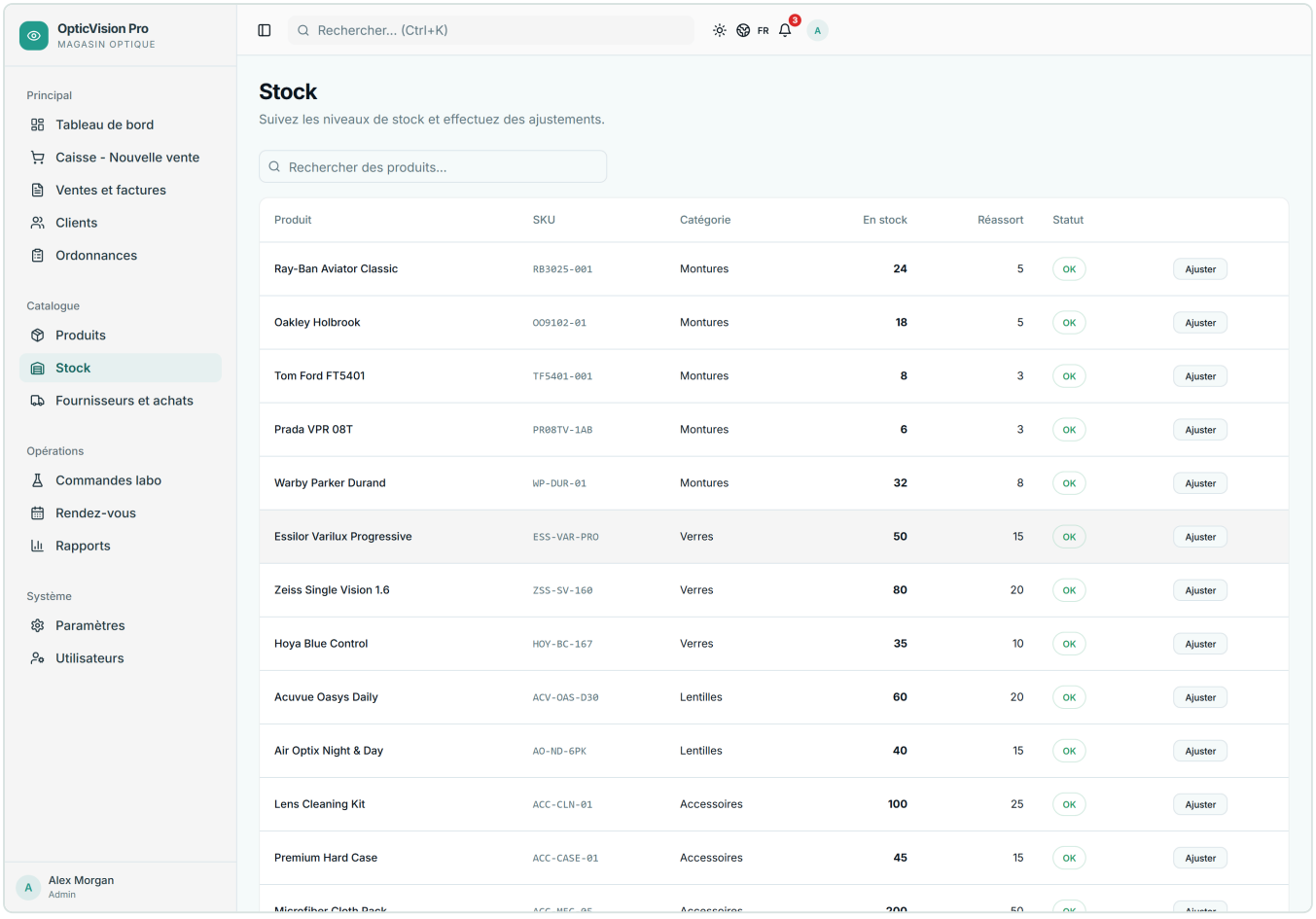


Figure 6.2 — Stock states and the adjustment entry point

1. Select **Ajuster** on a product.
2. Choose addition or removal and enter a quantity.
3. Choose a reason: **Inventaire**, **Dommage**, **Retour** or **Autre**.
4. Select **Appliquer**.

Stock can never go below zero. The reason is collected but no adjustment history is kept.

6.3 Suppliers and purchase orders

Open **Fournisseurs et achats**. The supplier table shows name, contact, e-mail, phone, lead time and payment terms; **Ajouter un fournisseur** creates one (name required).

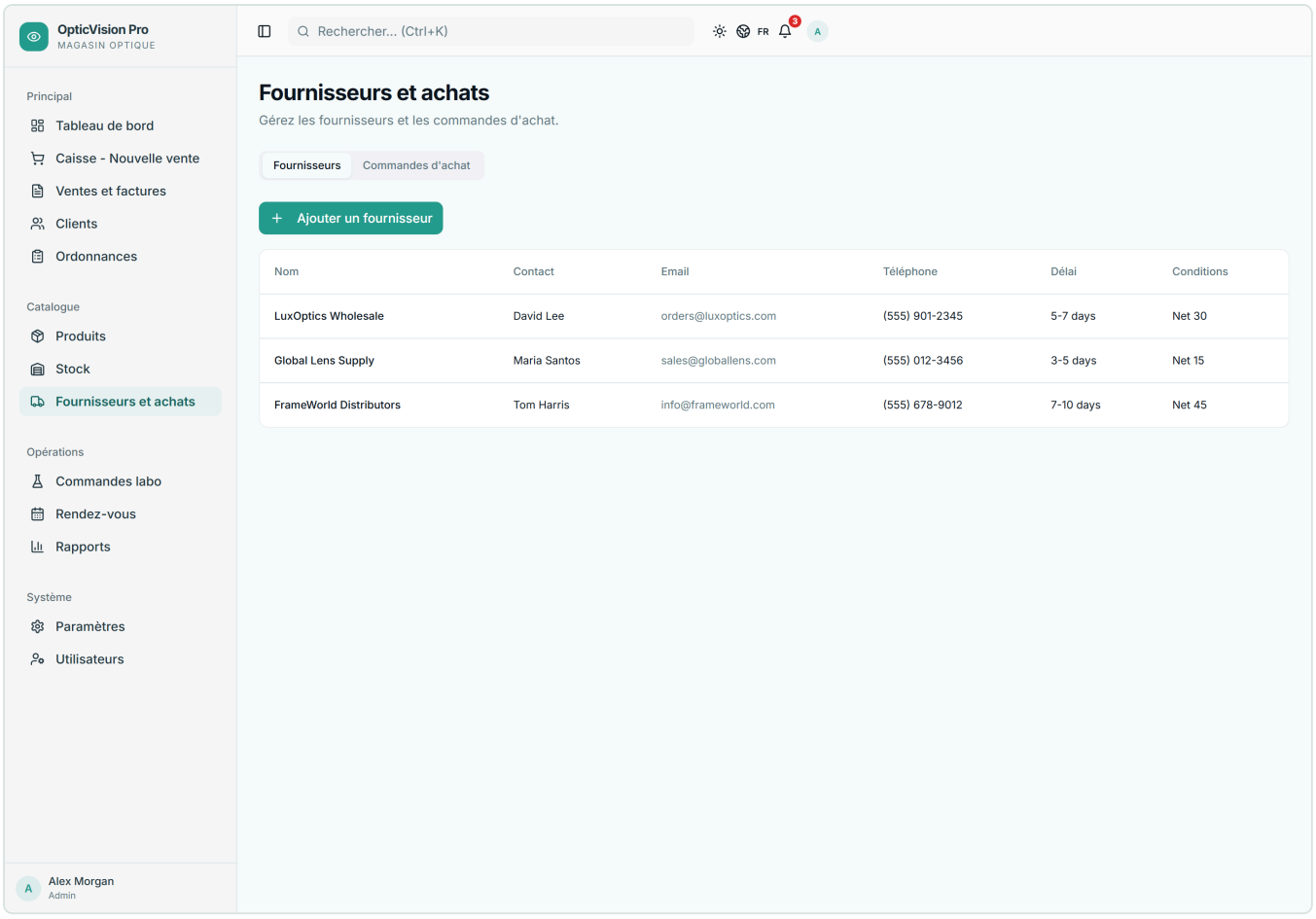


Figure 6.3 — Suppliers and purchase orders

The **Commandes d'achat** tab lists PO number, supplier, expected date, total and status. On an ordered or partially received PO, **Marquer reçu** sets the status to received.

> **Purchasing limits**
> The UI cannot create purchase orders, cannot receive partial lines, and receiving a PO does not add the quantities to inventory.

6.4 Reports and CSV export

Open **Rapports**. Three tabs:

- **Ventes** — revenue by payment method, plus revenue, transaction count and average basket
- **Stock** — total inventory value (`cost × stock`) and a per-product table
- **Meilleures ventes** — product ranking derived from invoice lines

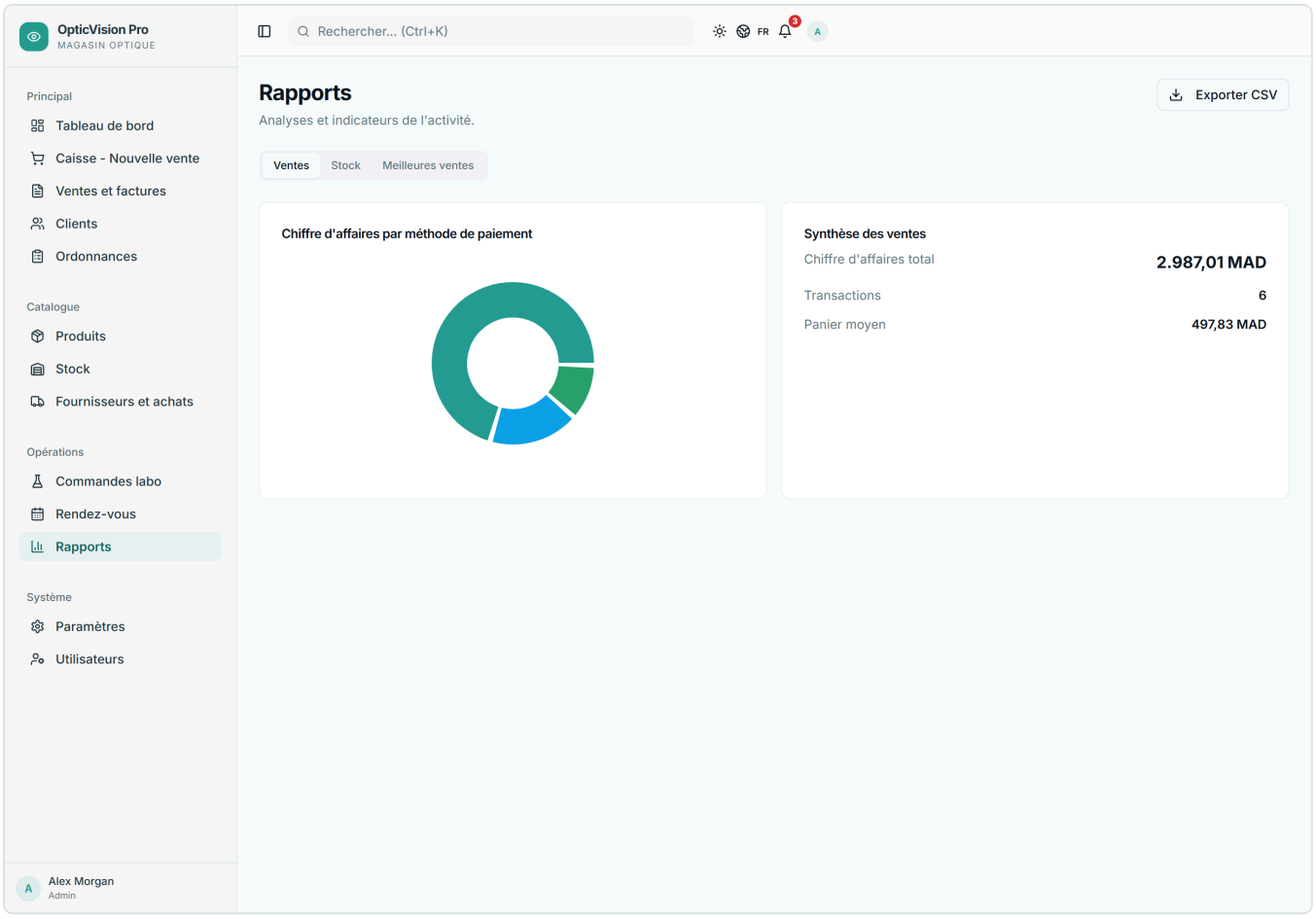


Figure 6.4 — Sales reporting with CSV export

Exporter CSV downloads `opticpos-report.csv` with invoice number, date, customer, total, payment method and status. The export always uses sales data, whichever tab is active. There are no date, store, cashier, tax or margin filters yet.

6.5 Store settings

Open **Paramètres**. Five tabs: **Magasin**, **Taxe**, **Reçu**, **Apparence** and **Langue**.

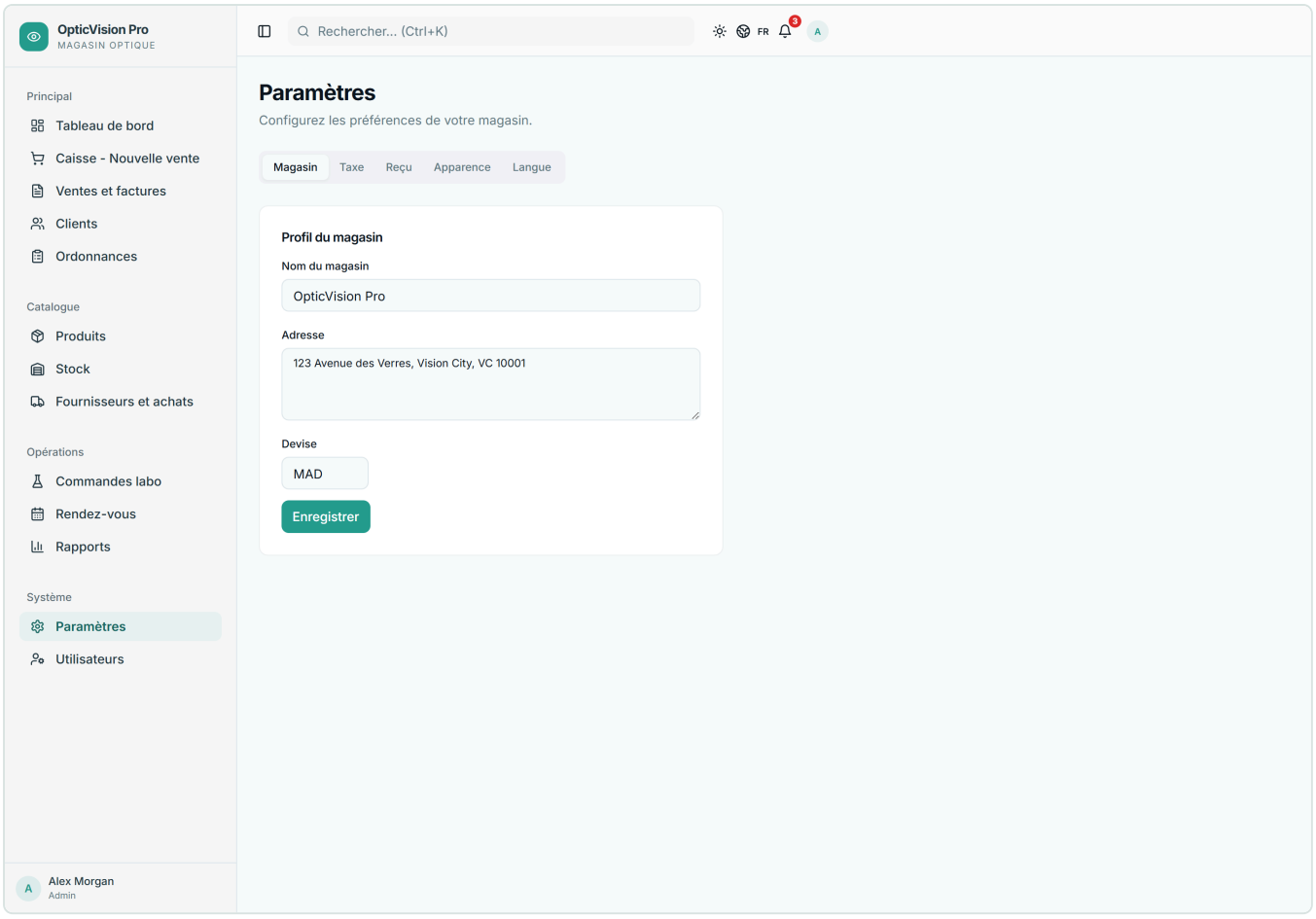


Figure 6.5 — Store profile settings

Tab	Setting	Effect
Magasin	Store name, address, currency code	Name and address appear on the receipt
Taxe	Tax rate (%)	Applied by new POS calculations
Reçu	Footer text	Printed at the bottom of new receipts
Apparence	Light/dark theme, comfortable/compact density	Theme applies instantly
Langue	Interface language	Applies instantly and persists

Each of the first three tabs has its own **Enregistrer** button.

> **Two settings that do not fully apply**
> Density is saved but not yet applied to the layout, and the currency code is saved but most formatting calls use the fixed `MAD` default from `src/lib/i18n.ts`. See section 7.6 for how to change the currency properly.

6.6 Users

Covered in chapter 3, sections 3.4 and 3.5.

CHAPTER 7

Customization

This chapter goes from the changes any store owner can make in the running app, down to the code edits a developer makes to rebrand and extend the product.

7.1 Store identity — no code needed

Sign in as Admin, open **Paramètres**, and set the store name, address, currency code, tax rate and receipt footer. These are stored per browser under `opticpos-settings`.

To change the shipped defaults for everyone instead, edit the initial state of the settings store in `src/lib/store.ts`:

```
storeName:     "OpticVision Pro",
storeAddress:  "123 Avenue des Verres, Vision City, VC 10001",
currency:      "MAD",
taxRate:       8.5,
theme:         "light",
density:       "comfortable",
receiptFooter: "Merci d'avoir choisi OpticVision Pro !",
paymentMethods: ["cash", "card", "insurance"],
insurers: ["VisionCare Plus", "EyeHealth Pro", "OptiInsure", "ClearView Benefits"],
```

Bump the persistence version

The settings store uses `persist` with `version: 2`. Browsers that already ran the app keep their saved copy, so new defaults will not appear until you raise the version number and extend the `migrate` function in the same file.

7.2 Brand name, logo and browser title

What	Where
App name in the sidebar and login	`app.name` in `src/lib/translations.ts` (per language)
Sidebar subtitle / login subtitle	`app.subtitle` and `login.subtitle`, same file
Logo mark	The `Eye` Lucide icon in `src/components/layout/AppSidebar.tsx` and `src/pages/Login.tsx`
Browser tab title, description, Open Graph tags	`index.html`
Favicon and static assets	`public/`

To use a real logo image, drop the file into `public/` and replace the icon element:

```
{/* before */}
<div className="h-10 w-10 rounded-xl bg-primary flex items-center justify-center">
```

```
  <Eye className="h-5 w-5 text-primary-foreground" />
</div>

{/* after */}
<img src="/logo.svg" alt="Your store" className="h-10 w-10 rounded-xl" />
```

7.3 Colour theme

All colours are HSL CSS custom properties defined twice in `src/index.css` — once under `:root` for light theme and once under `.dark`. Tailwind maps them to utility classes in `tailwind.config.ts`, so changing one variable restyles every component, badge and chart that uses it.

```
:root {
  --primary:    174 62% 38%;   /* teal — buttons, active nav, logo */
  --accent:     199 89% 48%;
  --background: 200 20% 98%;
  --foreground: 200 50% 8%;
  --success:    152 60% 40%;
  --warning:     38 92% 50%;
  --destructive:  0 72% 51%;
  --radius:     0.625rem;
  --chart-1 … --chart-5         /* chart series colours */
  --sidebar-background, --sidebar-primary, --sidebar-accent, …
}
```

Format matters
Values are bare HSL channels — `174 62% 38%`, not `hsl(174, 62%, 38%)` — because Tailwind wraps them as `hsl(var(--primary))`. Keep the format or the colour silently breaks.

To rebrand to, say, a deep blue: set `--primary: 221 70% 45%` in `:root`, set a lighter variant such as `221 70% 55%` in `.dark` (dark mode needs more luminance), and update `--ring`, `--sidebar-primary` and `--chart-1` to match. Nothing else needs touching.

7.4 Typography

The app uses Inter, loaded from Google Fonts in `index.html` and declared in two places: `fontFamily.sans` in `tailwind.config.ts`, and the `body` rule in `src/index.css`. To change the typeface, update the font link and both declarations. To drop the external request entirely, self-host the font files in `public/` and replace the link with an `@font-face` rule.

7.5 Corner radius and density

`--radius` in `src/index.css` drives every rounded corner; Tailwind derives `lg`, `md` and `sm` from it. Set it to `0.25rem` for a sharper, more clinical look or `1rem` for a softer one.

The **density** setting is persisted but not yet wired to the layout. To implement it, read `useSettingsStore(s => s.density)` in `src/App.tsx` next to the existing `ThemeInit` component and toggle a `compact` class on `document.documentElement`, then add compact spacing overrides in `src/index.css`.

7.6 Currency and locale

Formatting is centralized in `src/lib/i18n.ts`. The locale follows the selected language through `languageLocales`, while the currency code defaults to `MAD`:

```
export const languageLocales: Record<SupportedLanguage, string> = {
  fr: "fr-MA", en: "en-US", ar: "ar-MA", es: "es-ES", de: "de-DE",
};
```

To ship a different currency, change the default currency code in `src/lib/i18n.ts` and the `currency` default in `src/lib/store.ts`. To make the Settings currency field genuinely take effect, have the formatting helper read `useSettingsStore.getState().currency` instead of the hard-coded constant.

7.7 Interface language and translations

Every visible string lives in `src/lib/translations.ts`, one nested object per language, keyed as `app.*`, `common.*`, `nav.*`, `pages.*`, `login.*`, `settings.*`, `labels.*`. To reword the app — for example to call customers "patients" — edit the values there; no component changes are needed.

```
fr: {
  translation: {
    app: { name: "OpticPOS", subtitle: "Magasin optique" },
    nav: { dashboard: "Tableau de bord", pos: "Caisse – Nouvelle vente", … },
  },
},
```

Some screens still hold French literals

A few pages (POS, Users) contain hard-coded French strings rather than translation keys. Search for the literal text and move it into `translations.ts` if you need those screens fully localized.

7.8 Add a new language

1. Add the code to `supportedLanguages` in `src/lib/i18n.ts`.
2. Add matching entries to `languageLocales` and `languageFlags`.
3. Add a full translation object under the new key in `src/lib/translations.ts`.
4. Add the language name to the `languages` block of every existing language.

The switcher, the detector and the persisted preference all read from `supportedLanguages`, so nothing else needs editing. Right-to-left is handled automatically for Arabic; extend the check in `syncDocumentLanguage` for other RTL languages.

7.9 Tax rate

Admins change it live under **Paramètres** → **Taxe**. The shipped default (`8.5`) is the `taxRate` value in `src/lib/store.ts`. The rate applies to new POS calculations only; existing invoices keep the tax recorded at the time of sale.

7.10 Seed / demo data

All demo records — products, customers, prescriptions, invoices, lab orders, suppliers, purchase orders, appointments — are defined in `src/lib/mock-data.ts` and written to `localStorage` the first time each collection is read.

- **To load your own catalogue**, replace the seed arrays in that file.
- **To ship an empty app**, set the seed arrays to `[]`.
- **To see new seed data in a browser that already ran the app**, clear site data — old collections are not migrated.

7.11 Navigation and roles

The sidebar is data-driven. In `src/components/layout/AppSidebar.tsx`, four arrays — `mainNav`, `inventoryNav`, `operationsNav`, `systemNav` — hold entries of this shape:

```
{ titleKey: "nav.reports", url: "/reports", icon: BarChart3, roles: ["admin"] }
```

Change `roles` to change who sees an item, reorder the arrays to reorder the menu, and swap `icon` for any other Lucide icon. Group labels come from `nav.groups.*` in the translations.

7.12 Add a new page or module

1. Add the entity type and data access to `src/lib/mock-data.ts` — or, better, behind a service interface you can later point at a real API.
2. Create the page component in `src/pages/`.
3. Add a lazy import and a `<Route>` inside the `AppLayout` route in `src/App.tsx`.
4. Add a navigation entry with the right `roles` in `AppSidebar.tsx`.
5. Add the labels to every language in `src/lib/translations.ts`.
6. Reuse the shared components — `PageHeader`, `StatCard`, `StatusBadge`, `EmptyState` — so the new screen matches the rest.
7. Add validation and a test, then run `npm run verify`.

7.13 Deployment

Build with `npm run build` and host the `dist/` folder on any static host.

Setting	Value
Install command	`npm install`
Build command	`npm run build`
Output directory	`dist`
Required host rule	Rewrite unknown paths to `/index.html`

To change the dev server port, edit `server.port` in `vite.config.ts`.

CHAPTER 8

Technical reference

8.1 Architecture

```
Browser
├── React pages and shared components
├── React Router route tree
├── Zustand persisted stores
│   ├── authentication   (opticpos-auth)
│   ├── cart             (opticpos-cart)
│   └── settings         (opticpos-settings)
└── MockDB collections in localStorage
    ├── products          ├── lab orders
    ├── customers         ├── suppliers
    ├── prescriptions     ├── purchase orders
    └── invoices          └── appointments
```

TanStack Query is installed and its provider wraps the app, but pages currently read directly from Zustand and `MockDB` — it is there for when you add a real API.

8.2 Route map

Route	Page	Auth	Visible to
`/login`	Login	No	All
`/`	Redirect to dashboard	No	All
`/dashboard`	Dashboard	Yes	Admin, cashier, technician
`/pos`	POS	Yes	Admin, cashier
`/sales`	Sales	Yes	Admin, cashier
`/sales/:id`	Sale detail	Yes	Linked from sales
`/customers`	Customers	Yes	All roles
`/customers/:id`	Customer detail	Yes	Linked from customers
`/prescriptions`	Prescriptions	Yes	All roles
`/products`	Products	Yes	Admin, cashier
`/inventory`	Inventory	Yes	Admin
`/lab`	Lab orders	Yes	Admin, technician
`/appointments`	Appointments	Yes	All roles
`/purchases`	Suppliers & purchases	Yes	Admin

`/reports`	Reports	Yes	Admin
`/settings`	Settings	Yes	Admin
`/users`	Users	Yes	Admin
`*`	Not found	No	All

`AppLayout` performs one authentication check for all nested routes. Role filtering happens in `AppSidebar`, not in route guards.

8.3 Source layout

```
src/
├── components/
│   ├── layout/      App shell, top bar, sidebar
│   ├── shared/      PageHeader, StatCard, StatusBadge, EmptyState, LanguageSwitcher
│   └── ui/          shadcn/ui and Radix primitives
├── hooks/           Shared UI hooks
├── lib/
│   ├── i18n.ts          Languages, locales, currency and label helpers
│   ├── translations.ts  All UI strings, per language
│   ├── mock-data.ts     Types, seed records, local MockDB
│   ├── store.ts         Persisted Zustand stores
│   └── utils.ts         Styling helpers
├── pages/           Lazy-loaded route components
├── test/            Vitest setup and tests
├── App.tsx          Providers and route tree
└── main.tsx         React entry point
```

8.4 Storage keys

Key	Content
`opticpos-auth`	Current user and authentication flag
`opticpos-cart`	Cart lines, customer, prescription ID, notes
`opticpos-settings`	Store, currency, tax, theme, density, receipt (version 2)
`opticpos-language`	Selected interface language
`opticpos-products`	Products and stock totals
`opticpos-customers`	Customers
`opticpos-prescriptions`	Prescriptions
`opticpos-invoices`	Sales invoices
`opticpos-lab-orders`	Lab orders
`opticpos-suppliers`	Suppliers

`opticpos-purchase-orders`	Purchase orders
`opticpos-appointments`	Appointments

MockDB collections have no schema versioning or migration. Only the settings store migrates (version 2 normalizes currency, address and receipt text).

8.5 Data model

Entity	Core fields
Product	ID, name, SKU, category, brand, price, cost, stock, reorder level, description; optional frame/lens/contact attributes
Customer	ID, identity and contact details, insurer, policy number, notes, created date, balance
Prescription	Customer, issue/expiry dates, doctor, OD and OS sphere/cylinder/axis/addition, PD, notes
Invoice	Number, customer, date, line items, subtotal, tax, total, payment method/status, cashier
Lab order	Customer, prescription, lens/coating/frame, status, priority, due date, created date, notes
Supplier / PO / appointment	Defined alongside the others in `src/lib/mock-data.ts`

8.6 Calculations

```
Inventory value    = sum(product.cost × product.stock)
Invoice line total = (price × quantity) − discount
POS tax            = subtotal × taxRate / 100
Insurance discount = subtotal × coverage / 100
POS total          = subtotal + tax − insurance discount
Average basket     = invoice revenue / invoice count
```

Invoice numbers are generated in the browser from the current timestamp. The dashboard's "today" label is aspirational: the totals sum every stored invoice with no date filter.

CHAPTER 9

Data, security and limitations

9.1 Where data lives

Operational data, authentication state, settings and the active cart are stored in the current browser profile using `localStorage`. That means:

- Data exists only on that browser profile, on that device.
- Private/incognito windows may discard it when closed.
- Clearing site data deletes everything.
- Different browsers — and `localhost` vs `127.0.0.1` — do not share data.
- Two staff members never see each other's changes.
- There is no automatic backup or restore.
- Anyone with access to the browser profile can read or modify the records.

9.2 Authentication reality

The login form checks only that the e-mail contains `@` and the password is at least three characters. Nothing is sent to a server. Quick demo login creates a local user record directly. Sidebar items are hidden by role, but typing a URL reaches any route.

> **Do not treat the current authentication or role behaviour as a security boundary.**

9.3 Before using real customer data

The data model holds contact details, insurance information, prescriptions and transaction history — sensitive personal and health-related records. Implement all of the following first:

- Privacy/legal review for the country of deployment
- Strong authentication and least-privilege authorization, enforced server-side
- Encryption in transit and at rest
- Server-side audit logs, with sensitive payloads excluded from logs
- Retention and deletion policies; consent and access-request procedures
- Backups and disaster recovery
- Secure secret and key management

9.4 Functional limitations

SALES

- Checkout does not reduce stock; no payment gateway or cash-drawer integration; no real receipt printing.
- Refunds only change invoice status — no returns, exchanges, partial refunds or voids.

- Invoice numbers are browser-generated from a timestamp; insurance is a percentage discount, not a claim.

STOCK AND PURCHASING

- Stock adjustments retain no history or reason; receiving a PO does not add stock; the UI cannot create POs.
- No transfers, reservations, batches, serials or locations.

CUSTOMERS AND CLINICAL RECORDS

- No edit or delete flows, no duplicate detection, no optical-range validation.
- Prescription expiry is not enforced; no document attachments or doctor registry.

APPOINTMENTS AND LAB

- No conflict detection or reminders; appointment status cannot be changed in the UI.
- Lab status changes have no audit trail; dates are not validated against the current time.

REPORTING AND INTERFACE

- Dashboard "today" totals include all invoices; the weekly chart is fixed sample data.
- CSV export always exports invoice rows; no reporting filters.
- Top-bar search is not wired; the notification count is static; density is not applied; currency changes are not fully reflected.

CHAPTER 10

Troubleshooting and maintenance

10.1 Environment check

```
npm run doctor
```

Checks Node.js, npm, `package.json`, the lockfile and the dependency folder. Run this before anything else when something will not start.

10.2 Common problems

Symptom	Fix
Install fails	Confirm Node `20.19+` / `22.12+` and npm `10+`; run from the folder with `package.json`; retry `npm run setup`
Install was interrupted	Delete only this project's `node_modules`, then reinstall
Port 8080 in use	Vite picks the next free port — use the exact URL it prints
404 on a deep link after deploying	Configure the host to serve `index.html` for unknown routes
Data looks stale or missing	Same browser profile? Same hostname *and* port? Site data cleared? Private window?
Theme looks wrong	Toggle the theme, reload, then inspect or remove the `opticpos-settings` item
Build or lint fails	Run `npm run lint`, `npm test`, `npm run build` separately to isolate the stage

To find what is holding a port on Windows:

```
Get-NetTCPConnection -LocalPort 8080 -ErrorAction SilentlyContinue
```

Do not terminate an unknown process until you know what it is.

10.3 Back up demo data

In the browser developer console, on the OpticPOS origin:

```
copy(JSON.stringify({ ...localStorage }, null, 2));
```

Save the copied JSON somewhere safe — it may contain customer, prescription and sales details. This is a development convenience, not a supported backup format.

10.4 Reset the demo

> **This permanently deletes all locally entered data**
> Records, settings, login state and cart contents are all removed and cannot be recovered. Back up first (section 10.3) if any of it matters.

```
localStorage.clear();
location.reload();
```

Seed data is recreated as each collection is read again. To fix only a corrupted theme without losing records, remove the single settings key instead:

```
localStorage.removeItem("opticpos-settings");
location.reload();
```

10.5 Release checklist

1. Run `npm run doctor`.
2. Install from the lockfile in a clean environment.
3. Run `npm run verify`.
4. Test login for all three roles.
5. Test POS cash and card checkout.
6. Test create flows: customer, prescription, lab order, appointment, product, supplier.
7. Test stock addition and removal.
8. Test sales filtering, refund status, reports and CSV export.
9. Test light and dark theme, and each interface language.
10. Test direct-route refreshes on the target web server.
11. Re-read chapter 9 before any real data is entered.

10.6 Command cheat sheet

Command	Purpose
`npm run setup`	Check environment, install, verify
`npm run setup:start`	Install and launch in one step
`npm run dev`	Start the development server
`npm run build`	Production build into `dist/`
`npm run preview`	Serve the production build locally
`npm run lint`	ESLint
`npm test`	Vitest, single run
`npm run verify`	Lint + test + build

`npm run doctor`	Prerequisite check

10.7 Reporting a problem

Include the operating system, Node and npm versions, the command that failed, the complete terminal error, the browser and version, the route where it happened, whether local data can be reset, and the steps to reproduce.

> **Never include real passwords, payment data, insurance identifiers or patient details in an issue report.**

APPENDIX

Screenshot index

All captures in this manual were taken from a running build at 1440×1000, French interface, seed data only. Source files live in `docs/screenshots/`.

File	Route / state	Figure
`01-login.png`	`/login`	3.1
`02-dashboard.png`	`/dashboard`, admin	Cover, 4.1
`03-pos.png`	`/pos`, populated cart	5.1
`04-receipt.png`	POS receipt dialog	5.2
`05-customers.png`	`/customers`	5.4
`06-customer-detail.png`	`/customers/c1`	5.5
`07-prescriptions.png`	`/prescriptions`	5.6
`08-products.png`	`/products`	6.1
`09-inventory.png`	`/inventory`	6.2
`10-lab-orders.png`	`/lab`	5.7
`11-appointments.png`	`/appointments`	5.8
`12-purchases.png`	`/purchases`	6.3
`13-reports.png`	`/reports`	6.4
`14-settings.png`	`/settings`	6.5
`15-users.png`	`/users`	3.2
`16-sales.png`	`/sales`	5.3
`17-dashboard-dark.png`	`/dashboard`, dark theme	4.2