DOTCLOCK

A–Z Site Documentation

The complete reference for the DotClock marketing & commerce platform — architecture, every page and API, the admin suite, the creator studio, and the design system.

Last revised **2026-06-04** · Build target **Node 22 · Astro 6 · MongoDB 6 · Stripe**
Includes the latest release: Admin Payments & Expenses, Gift a DotClock, the upgraded Studio, the site-wide digital-rain fix, and the 3D + living-glow CTA buttons.

Table of contents

1. Product overview

DotClock is a hand-finished European-oak desk clock with a **64×32 RGB LED matrix** display. Beyond showing the time it scrolls live data — weather, sunrise, UV, news headlines, earthquake alerts — pulled automatically from the owner's location. No phone app, no account, no subscription: the device is configured from any browser on the same Wi-Fi network at `dotclock.local`.

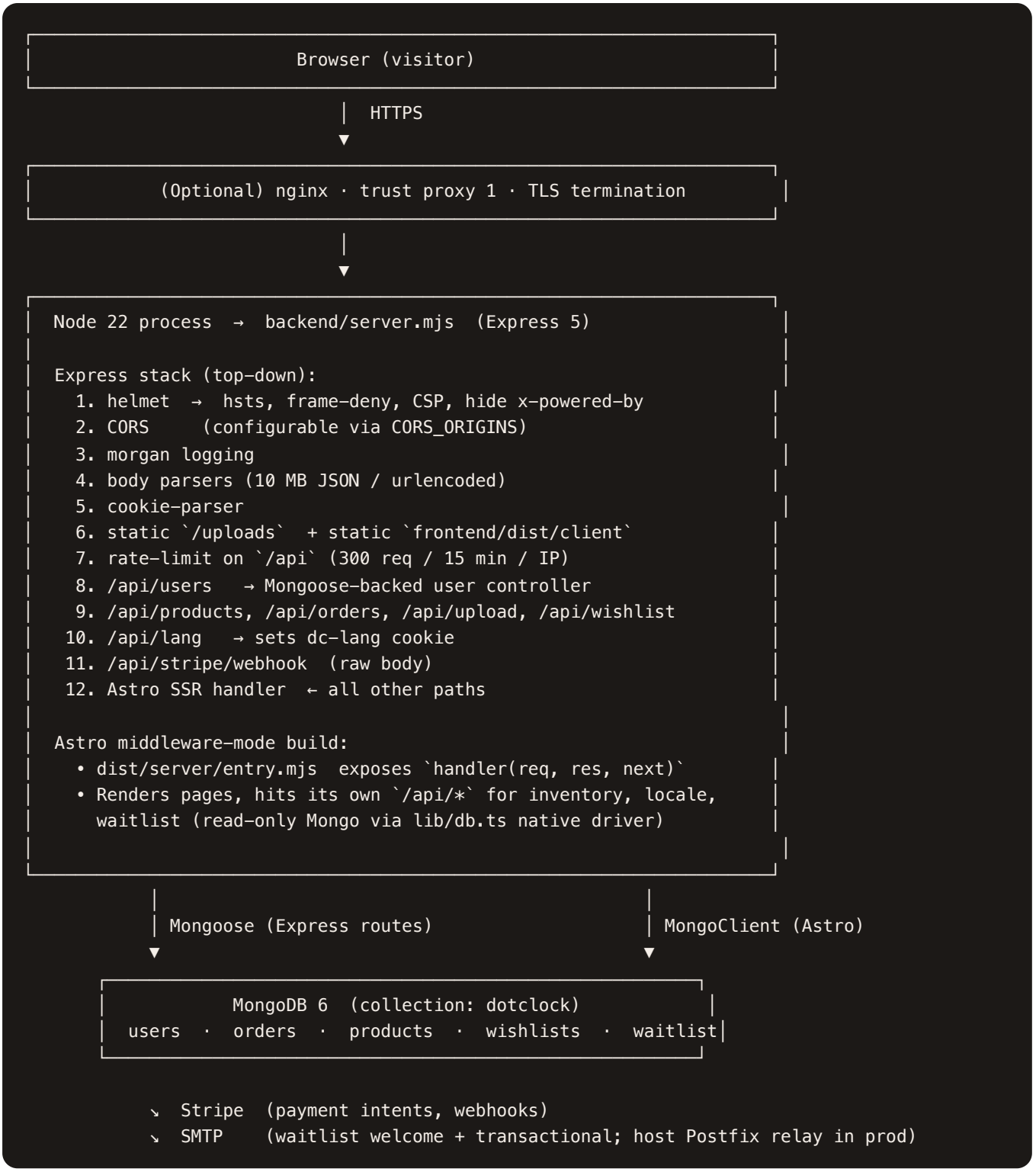
This repository is the **marketing + commerce site** for that product. It serves:

- A long-form landing page that tells the product's story.
- A live, interactive banner-builder so visitors can prototype their setup.
- A waitlist + back-in-stock notification system backed by MongoDB.
- A real shopping cart and Stripe checkout for direct sales.
- A lightweight admin area for the founder to view orders.
- A public **roadmap & changelog surface** — `/compare` for "DotClock vs X" buyers, `/firmware` for release notes with an RSS feed.
- **17 localised home pages** in addition to English: Italian, Spanish, French, German, Portuguese, Polish, Russian, Turkish, Hebrew, Arabic, Berber, Hindi, Malayalam, Vietnamese, Mandarin, Korean, Japanese. RTL handled for `ar` and `he`.

Single-product commerce. The repo is intentionally a "one SKU done well" site, not a generic store: `DOTCLOCK-V3-OAK` is the only product, and the Buy section is the only purchase surface. Every system (inventory, currency, share, restock) is designed around that single SKU.

---

2. Architecture at a glance



Why two HTTP layers? Astro's Node adapter in *middleware mode* exports a `handler(req, res, next)` that Express mounts as its **final** middleware. Everything before it (security headers, REST API, static, rate limit) is plain Express. This keeps:

- **Static landing-page traffic** off the Astro renderer (served by `express.static` with `max-age=31536000, immutable` for hashed assets).
- **Authenticated/REST work** in a familiar Express/Mongoose codebase.

- **SSR for dynamic Astro pages** (every `/api/*` Astro route, plus pages with `export const prerender = false`).

Why two Mongo clients? Express routes use Mongoose (schemas + middleware). Astro's read-only API routes use the official `mongodb` driver via `frontend/src/lib/db.ts` — lighter, no schema overhead, fine for counters/aggregations. Both point at the same database.

---

3. Tech stack

Layer	Choice	Notes
Runtime	**Node 22**	ESM throughout; pinned in `.nvmrc`, Docker uses Node 22, host deploy uses `nvm`.
Web framework (UI)	**Astro 6** (`output: "server"`, `adapter: node middleware`)	Page-first; component islands only if needed.
Web framework (API)	**Express 5**	Wraps Astro SSR; owns `/api/*` routes, security, static, webhooks.
Database	**MongoDB 6**	Mongo 6 chosen to match on-disk format of legacy volume.
ODM	**Mongoose 9** (backend) + **mongodb 7 native** (frontend)	Two clients, same DB — see "Why two Mongo clients?" above.
Auth	**jsonwebtoken** (HS256), **bcryptjs** (cost 12)	JWT in `dc_token` HTTP-only cookie.
Payments	**Stripe** Payment Intents + PayPal (optional)	Webhook-driven order paid state.
Email	**nodemailer** SMTP	Mailtrap for dev, any SMTP for prod.
Image processing	**sharp** (Astro Image)	AVIF preferred, JPEG/PNG fallbacks.
Sitemap	**@astrojs/sitemap**	Outputs `sitemap-index.xml` + locale alternates.
Security headers	**helmet 8**	Custom CSP; HSTS preload-eligible; `Server: DotClock`.
Rate limit	**express-rate-limit 8**	Global `/api`: 300/15 min; login: 30/10 min skipSuccess; password: 3/hr.
CSS	Hand-rolled with cascade layers + scroll-driven animations	`animation-timeline: scroll() / view()` with JS fallback.
Font	**ProFontWindows.woff** self-hosted	`font-display: swap`, `local()` first.
TypeScript	Astro `astro/check` for type-checking	Source is `.ts` / `.tsx` / `.astro`; build emits JS.
Process	**systemd** (`dotclock-site`) + host `mongod`	Node app on port 4321 behind nginx; MongoDB stays local to the host.

4. Repository layout

```
dotclock-site/
├── Dockerfile                · Optional container build for local/dev use
├── docker-compose.yml        · Optional local stack: web + mongo
├── deploy/                   · Sample nginx + systemd production configs
├── DEPLOYMENT_NOTES.md       · One-off prod notes
├── README.md                 · User-facing quick-start
├── DOCUMENTATION.md          · ← you are here
├── package.json              · Root: backend deps + scripts
├── .env.example              · Template — copy to .env
├── .dockerignore             · Keeps node_modules + dist out of build context
├── .gitignore
│
├── backend/                  · Express + Mongoose
│   ├── server.mjs            · Production server (Express + Astro middleware)
│   ├── server.api.mjs        · API-only dev server (no Astro, for `npm run dev:full`)
│   ├── config/
│   │   └── db.js             · mongoose.connect(MONGO_URI)
│   ├── middleware/
│   │   ├── authMiddleware.js   · `protect` + `admin` guards (JWT cookie)
│   │   ├── rateLimiter.js      · Factory for per-route rate limits
│   │   └── errorMiddileware.js · `notFound` + generic `errorHandler`
│   ├── models/
│   │   ├── userModel.js        · User schema, bcrypt pre-save, matchPassword
│   │   ├── orderModel.js       · Order, orderItems, addressSchema, Stripe linkage
│   │   ├── productModel.js     · Single seeded product (DOTCLOCK-V3-OAK)
│   │   └── wishListModel.js    · Per-user product list
│   ├── controllers/          · All business logic for the routes below
│   │   ├── userController.js
│   │   ├── orderController.js
│   │   ├── productController.js
│   │   ├── wishListController.js
│   │   ├── uploadController.js  · Multer config + handler
│   │   └── paypalController.js
│   ├── routes/
│   │   ├── userRoutes.js
│   │   ├── orderRoutes.js
│   │   ├── productRoutes.js
│   │   ├── wishListRoutes.js
│   │   └── uploadRoutes.js
│   ├── scripts/
│   │   └── makeAdmin.js      · One-off: promote a user to isAdmin
│   └── utils/
│       ├── generateToken.js · Signs JWT + sets cookie
│       └── sendEmail.js      · Light nodemailer wrapper
│
├── uploads/                  · User-uploaded avatars (bind-mounted in compose)
│
└── frontend/                 · Astro project root
    ├── astro.config.mjs      · Sitemap, Node adapter (middleware mode), i18n, prefetch
    ├── tsconfig.json
    ├── public/
    │   ├── favicon.svg
    │   ├── og-image.png
    │   ├── manifest.webmanifest
```

```
│   ├── robots.txt       · Allow / + Disallow private routes + Sitemap URL
│   ├── guide.pdf        · DotClock V3 user-guide PDF
│   ├── fonts/
│   │   └── ProFontWindows.woff
│   └── images/
│       ├── hero.png
│       └── setup.png
└── src/
    ├── env.d.ts
    ├── middleware.ts        · Locale redirect heuristics
    ├── i18n/
    │   ├── ui.ts            · All translation strings, per-locale objects
    │   └── utils.ts         · `useTranslations(lang)` accessor
    ├── data/
    │   └── site.ts          · SINGLE SOURCE OF TRUTH for copy, pricing,
    │                          batch config, currency table
    ├── lib/
    │   ├── db.ts            · Cached MongoClient + getCollection helper
    │   ├── mailer.ts        · nodemailer wrapper (used by API routes)
    │   ├── setup-db.ts      · One-off: create waitlist indexes
    │   └── emails/
    │       └── waitlist-welcome.ts   · HTML+text welcome template
    ├── layouts/
    │   ├── Layout.astro     · <head>, JSON-LD, view-transition CSS,
    │   │                      delegated tilt/magnetic/ripple/reveal JS
    │   └── ContentPage.astro · Wrapper for legal/info pages
    ├── styles/
    │   └── global.css       · Tokens, base, components, animations,
    │                          ::view-transition rules
    ├── components/          · All reusable UI islands (see §6)
    ├── assets/product/      · Source images — pipelined through Astro Image
    └── pages/               · Every route (see §5)
        ├── index.astro      · Home (English)
        ├── {it,es,fr,de,ja,zh,ml,ar}/index.astro
        ├── about.astro, press.astro, privacy.astro, terms.astro,
        │   shipping.astro, returns.astro, warranty.astro
        ├── login.astro, register.astro, account.astro,
        │   reset-password/[token].astro
        ├── cart.astro, checkout.astro, success.astro, orders.astro
        ├── admin/{index,orders}.astro
        └── api/
            ├── inventory.ts          · NEW · batch counter + ship-by
            ├── locale.ts             · NEW · IP → currency
            └── waitlist/{index,count}.ts
```

5. Pages — every route, end to end

Pages live in `frontend/src/pages/`. Astro maps file paths to URLs 1:1. Pages with `export const prerender = false` are server-rendered on every request; the rest are static-generated at build time.

Public marketing pages

Route	File	Purpose
`/`	`pages/index.astro`	Home. Loads every section component (Hero → Buy) in order. SSR.
`/it/`, `/es/`, `/fr/`, `/de/`, `/pt/`, `/pl/`, `/ru/`, `/tr/`, `/he/`, `/ar/`, `/ber/`, `/hi/`, `/ml/`, `/vi/`, `/zh/`, `/ko/`, `/ja/`	`pages/<locale>/index.astro`	Localised home for each non-default locale (17 total). Same component tree; locale picked up via `Astro.currentLocale` and read by `useTranslations`. `dir="rtl"` is set on `<html>` for `ar` and `he`.
`/compare`	`pages/compare.astro`	"DotClock vs LaMetric vs Vestaboard vs Tidbyt" comparison page (13 honest rows). Sticky table header, horizontal scroll-snap on mobile, accent CTA back to `/#buy`. SSR.
`/firmware`	`pages/firmware.astro`	Reverse-chronological firmware release notes. Latest entry gets an accent border + "Latest" pill. Colour-coded change pills (feature/fix/perf/security). Anchor links per version.
`/firmware.xml`	`pages/firmware.xml.ts`	RSS 2.0 feed of firmware releases. `application/rss+xml`, `atom:self` link, CDATA-wrapped HTML body. Cached 5 min + SWR 1 hr. Auto-discovered via `<link rel="alternate">` in `<head>`.
`/about`	`pages/about.astro`	Founder/brand story (ContentPage layout).
`/press`	`pages/press.astro`	Press kit + contact: `press@dotclock.com`.
`/privacy`	`pages/privacy.astro`	Privacy policy. Contact `privacy@dotclock.com`.
`/terms`	`pages/terms.astro`	Terms of service. Contact `legal@dotclock.com`.
`/shipping`	`pages/shipping.astro`	Shipping times & rates.
`/returns`	`pages/returns.astro`	30-day return policy + RMA via `support@dotclock.com`.
`/warranty`	`pages/warranty.astro`	2-year warranty terms.

Account, cart, checkout (`noindex` — Disallow in `robots.txt`)

Route	File	Purpose
`/login`	`pages/login.astro`	Login form, talks to `POST /api/users/login`. Shows Loader3D overlay while authenticating.
`/register`	`pages/register.astro`	Signup form, `POST /api/users/register`. Sends verification email.
`/reset-password/<token>`	`pages/reset-password/[token].astro`	Lands from password-reset email, posts to `POST /api/users/reset-password/:token`.
`/account`	`pages/account.astro`	Profile editor + avatar upload (multipart to `POST /api/users/profile/avatar`).
`/cart`	`pages/cart.astro`	Local-storage cart UI (`dotclock-cart` key). Read-only of localStorage; nothing server-side.
`/checkout`	`pages/checkout.astro`	Stripe Elements form; creates an Order then a PaymentIntent (`POST /api/orders/:id/pay/intent`).
`/success`	`pages/success.astro`	Post-payment landing with order ID, polled via `GET /api/orders/:id`.
`/orders`	`pages/orders.astro`	List my orders (`GET /api/orders/mine`).

Admin (`noindex`)

Route	File	Purpose
`/admin`	`pages/admin/index.astro`	Admin landing — gated by client check against `dc-user-admin` cookie. Server-side guard is the protected API.
`/admin/orders`	`pages/admin/orders.astro`	Orders list + status changes (`GET /api/orders`, `PUT /api/orders/:id/status`).

API routes (Astro server endpoints)

Method	Path	File	Purpose
POST	`/api/waitlist`	`api/waitlist/index.ts`	Add email to waitlist; sends welcome on first add. Also accepts `source:"restock"` for the back-in-stock form on Buy.
GET	`/api/waitlist/count`	`api/waitlist/count.ts`	Return current waitlist size (cached 30s).
GET	`/api/waitlist/recent`	`api/waitlist/recent.ts`	Last N signups anonymised to region only (no name / email leaks). Powers the live activity ticker. Cached 60s + SWR 5min.
GET	`/api/inventory`	`api/inventory.ts`	Live batch counter + business-day ship-by date.
GET	`/api/locale`	`api/locale.ts`	IP/Accept-Language → currency + display price.
GET	`/firmware.xml`	`pages/firmware.xml.ts`	RSS 2.0 feed of firmware releases (one source of truth with `/firmware`). Cached 5min + SWR 1hr.

All Express routes (`/api/users` , `/api/orders` , …) are handled **before** the Astro middleware in `backend/server.mjs` , so they never touch the Astro renderer.

---

6. Components — every reusable block

All Astro components live in `frontend/src/components/`. They are **template-first**: most have a small inline `<script>` for behaviour and a scoped `<style>` for visuals. Where a script needs to survive Astro's view transitions, it either:

- Listens for `document.addEventListener("astro:after-swap", …)` to re-bind, **or**
- Uses delegated listeners on `document` (see `Layout.astro` for tilt / magnetic / ripple).

Layout-level (always rendered)

Component	Mounted by	Role
`Layout.astro`	every page	The HTML shell. Sets `<title>`, description, OG/Twitter, hreflang (9 locales), canonical, robots, JSON-LD (Product, FAQPage, Organization, WebSite, BreadcrumbList), preloads the font, ships theme-init + delegated event listeners. Hosts `#scroll-progress`, persisted modals & cursor.
`ContentPage.astro`	legal/info pages	Wraps `Layout` with a centred article container + `<Nav>` / `<Footer>`. Used by about/press/privacy/terms/shipping/returns/warranty.
`Nav.astro`	every page	Top navigation. Auth-aware (reads `dc-user-*` cookies). Locale switcher posts to `/api/lang`. Hosts the account dropdown.
`Footer.astro`	every page	Three link groups + social + guide PDF download. `transition:persist` so it isn't repainted on navigation.
`Cursor.astro`	every page	**Spotlight-glow cursor.** Single accent-warm radial-gradient blob that follows the pointer with exponential lerp; never hides the native OS cursor underneath. Brightens + grows when over interactive elements. Re-asserts on `astro:after-swap` and `visibilitychange`. Hidden on touch/coarse pointer; static (no lerp) for `prefers-reduced-motion`. Tones down via `mix-blend-mode: multiply` in light theme.
`Loader3D.astro`	every page	Full-viewport overlay loader (`position: fixed; inset: 0;`). Exposes `window.showLoader(text?)` / `window.hideLoader()`.
`LoginModal.astro`	every page	In-page login/forgot-password modal. Used by `Nav` and standalone login page.
`ProfileModal.astro`	every page	Avatar + profile quick-edit overlay.

Home-page sections (top to bottom)

Component	Section ID	What it does
`Hero.astro`	`#top`	Headline, sub-headline, two CTAs (Get yours / How it works), live banner-preview strip, hero image carousel (2 AVIFs, `loading="eager"`, `fetchpriority="high"` on the LCP).
`StatBar.astro`	—	4 hero stats ("0 API keys", "3 min setup"…) under the hero.
`Craft.astro`	—	Materials + joinery storytelling section.
`DayNight.astro`	—	Side-by-side day/night image with a toggle that switches the visible image via opacity (or View Transitions API on supporting browsers).
`BannerBuilder.astro`	`#builder`	Live LED preview + module checklist. **Now URL/localStorage backed** (see §15.1) with a "Share this setup" button.
`Features.astro`	`#features`	8-card feature grid (auto-located weather, adaptive brightness, news ticker, alerts, browser config, fonts, languages, persistence).
`Setup.astro`	`#how`	3-step setup walkthrough with an inline mock device illustration.
`Specs.astro`	`#specs`	10-row specification table (display, frame, dimensions, weight, power, connectivity, sensors, configuration, OTA, warranty).
`Reviews.astro`	—	5-star summary + customer quotes (placeholder until real reviews land).
`FAQ.astro`	`#faq`	8 questions in `<details>` accordions with `interpolate-size` for smooth height transitions.
`Waitlist.astro`	—	Email capture form → `POST /api/waitlist`. Live counter from `GET /api/waitlist/count`.
`Buy.astro`	`#buy`	Cream-on-dark Buy card with price + live stock badge + Add-to-cart + restock form + trust badges + guide-PDF download. Now powers currency switch, batch counter and back-in-stock signup (see §15.2-§15.4).

Shared/utility

Component	Notes
(none beyond above)	The site intentionally keeps the component count small — most "components" are sections, and shared bits live in the layout or as global CSS classes.

7. API surface

7.1 Astro API routes

> These live in `frontend/src/pages/api/`. They run inside the Astro SSR handler, which Express mounts as its last middleware. They use the **mongodb native driver** via `frontend/src/lib/db.ts` (a cached `MongoClient` shared across hot reloads).

`POST /api/waitlist`

Body — JSON or form-encoded:

```
{ "email": "alice@example.com", "source": "footer" }
```

Behaviour

- Validates email shape, length ≤ 254.
- In-memory IP rate limit: **5 requests / 60 s / IP** (per Node process).
- Upserts `{ email }` into `waitlist`. `$setOnInsert` means re-submits don't overwrite the original.
- On first insert: sends `waitlist-welcome` email via `lib/mailer` (fire-and-forget — SMTP failures don't fail the request).

Responses

Status	Body
200	`{ ok: true, total: number, emailSent: boolean }`
400	`{ ok: false, error: "invalid_email" | "invalid_body" }`
429	`{ ok: false, error: "rate_limited" }`
500	`{ ok: false, error: "server_error" }`
503	`{ ok: false, error: "db_unavailable" }` (no `MONGODB_URI`)

`GET /api/waitlist/count`

Returns `{ total: number, configured: boolean }`. **Cached 30 s** via `Cache-Control: public, max-age=30`. Uses `estimatedDocumentCount()` (no full scan).

`GET /api/inventory` (new)

Live batch counter for the Buy section. See §15.2 for the full spec.

Response:

```
{
  "configured": true,
  "target": 100,
  "sold": 0,
  "remaining": 100,
  "outOfStock": false,
  "shipBy": "2026-05-28"
}
```

Cached `public, max-age=30, stale-while-revalidate=120`.

`GET /api/locale` (new)

IP → currency display. See §15.4.

Response:

```
{
  "country": "US",
  "currency": "USD",
  "symbol": "$",
  "displayPrice": 249,
  "source": "cloudflare"
}
```

Cached `public, max-age=300, stale-while-revalidate=3600`, `Vary: cf-ipcountry, x-vercel-ip-country, accept-language`.

7.2 Express REST routes

> Mounted by `backend/server.mjs` under `/api/*`. All use **Mongoose**. JWT cookie auth via `protect` / `admin` middleware in `backend/middleware/authMiddleware.js`. Global rate limit: **300 req / 15 min / IP**.

Users — `/api/users`

Method	Path	Auth	Description
POST	`/register`	public	Create account; sends verification email.
POST	`/login`	rate-limited (30 / 10 min, skipSuccess)	Authenticate; sets `dc_token`, `dc_user_id`, `dc_user_email`, `dc_user_admin` cookies.
POST	`/logout`	public	Clears auth cookies.
GET	`/verify/:token`	public	Email-verification landing.
POST	`/forgot-password`	rate-limited (3 / hr)	Send password-reset email.
POST	`/reset-password/:token`	rate-limited (3 / hr)	Set new password.
GET	`/profile`	protect	Read own profile.
PUT	`/profile`	protect	Update name/email/password.
POST	`/profile/avatar`	protect, multer	Upload avatar to `uploads/`; stores public URL on user doc.
DELETE	`/profile`	protect	Delete own account.
GET	`/`	admin	List users.
GET	`/:id`	admin	Read user.
PUT	`/:id`	admin	Update user (incl. `isAdmin` flag).
DELETE	`/:id`	admin	Delete user.

Orders — `/api/orders`

Method	Path	Auth	Description
POST	`/`	protect	Create order from cart payload.
GET	`/`	admin	List all orders (admin dashboard).
GET	`/mine`	protect	List signed-in user's orders.
GET	`/:id`	protect	Read order (owner or admin).
POST	`/:id/pay/intent`	protect	Create Stripe PaymentIntent for this order.
POST	`/:id/paypal/create`	protect	Create PayPal order (optional second processor).
POST	`/:id/paypal/capture`	protect	Capture PayPal payment after approval.
PUT	`/:id/pay`	protect	Mark paid (called by client after Stripe confirm).
PUT	`/:id/deliver`	admin	Mark shipped/delivered.
PUT	`/:id/status`	admin	Set status (pending / processing / shipped / delivered / cancelled).

Products — `/api/products`

Method	Path	Auth	Description
GET	`/`	public	List products. (DotClock auto-seeded as the only one.)
POST	`/`	admin	Create product.
GET	`/top`	public	Top-rated products (used by review widget).
GET	`/:id`	public	Read product.
PUT	`/:id`	admin	Update product.
DELETE	`/:id`	admin	Delete product.
POST	`/:id/reviews`	protect	Add a review.

Wishlist — `/api/wishlist`

Method	Path	Auth	Description
GET	`/`	protect	Read own wishlist.
POST	`/`	protect	Add product.
DELETE	`/`	protect	Clear all.
DELETE	`/:productId`	protect	Remove single product.

Uploads — `/api/upload`

Method	Path	Auth	Description
POST	`/`	admin, multer (single `image`)	Upload product image to `uploads/`.

Misc

Method	Path	Notes
POST	`/api/lang`	Sets `dc-lang` cookie (`en` or `it`). JS-readable for nav UI.
GET	`/api/verify/:token`	Shortcut to email verification (mirrors `/api/users/verify/:token`).
POST	`/api/stripe/webhook`	Raw-body endpoint; verifies signature with `STRIPE_WEBHOOK_SECRET`.

---

8. Data model — MongoDB collections

`users` (Mongoose)

```
{
  _id, name, email (unique, lowercased),
  password,                  // bcrypt cost 12
  isAdmin: false,
  isVerified: false,
  verificationToken, resetToken, resetTokenExpiry,
  avatar: "",                // path under /uploads
  createdAt, updatedAt
}
```

`orders` (Mongoose)

```
{
  _id, user (ObjectId → users),
  orderItems: [{ name, qty, image, price, product (→ products) }],
  shippingAddress: { firstName, lastName, address, city, postalCode, country, phone },
  paymentMethod: "stripe",
  paymentResult: { id, status, updateTime, emailAddress },
  itemsPrice, shippingPrice, taxPrice, totalPrice,
  currency: "eur",
  isPaid: false, paidAt,
  isDelivered: false, deliveredAt,
  stripePaymentIntentId,
  status: "pending" | "processing" | "shipped" | "delivered" | "cancelled",
  createdAt, updatedAt
}
```

> The new `/api/inventory` route reads this collection via the native driver and counts non-cancelled paid orders where `paidAt >= batch.startedAt` (with `createdAt` fallback for older records).

`products` (Mongoose)

```
{
  name, slug, description, shortDescription,
  price: 229, sku: "DOTCLOCK-V3-OAK",
  images: ["/images/hero.png"],
  category: "clocks", brand: "DotClock",
  countInStock: 100, isFeatured: true
}
```

Auto-seeded by `server.mjs` at boot if the SKU isn't present.

`wishlists` (Mongoose)

Per-user product list — used by the wishlist endpoints. Same shape as the typical wishlist model: `{ user, products: [ObjectId] }`.

`waitlist` (native driver — `lib/db.ts`)

```
{
  email,                    // unique, lower-cased
  source,                   // free-text segment: "footer", "hero", "restock"…
  ip, userAgent, locale,
  confirmed: false,         // reserved for future double-opt-in
  createdAt: Date
}
```

Indexes (run `npx tsx frontend/src/lib/setup-db.ts`):

- `{ email: 1 }` unique → prevents duplicate signups.
- `{ createdAt: -1 }` → sorted feed for an admin export.

Note — the new "back-in-stock" form (§15.3) reuses this collection with `source: "restock"`, so the same export script gives both lists segmented by source.

---

9. Authentication & sessions

Sign-up / login flow

1. Client posts `{ email, password }` to `POST /api/users/register` (or `/login`).
2. Server hashes password (bcrypt, cost 12), creates user, emits a JWT via `utils/generateToken.js`.
3. JWT is set as `dc-token` HTTP-only, `SameSite=Lax`, `Secure` in production, 30-day TTL (`JWT_EXPIRES_IN`).
4. Non-sensitive identity is mirrored to JS-readable cookies so the nav can render without an API round-trip:
   - `dc-user-id`, `dc-user-email`, `dc-user-admin`.
5. Subsequent API calls send `dc-token` automatically; the `protect` middleware verifies it with `JWT_SECRET` and loads `req.user`.

Email verification

- `POST /api/users/register` generates `verificationToken`, emails a link to `/api/verify/:token`.
- Hitting that link sets `isVerified: true`.

Password reset

- `POST /api/users/forgot-password` generates `resetToken` + `resetTokenExpiry: +1h`, emails a reset link.
- The reset page (`/reset-password/[token]`) posts new password to `POST /api/users/reset-password/:token`.

Admin gating

- Express: `admin` middleware checks `req.user.isAdmin === true`.
- UI: the `dc-user-admin` cookie controls whether the admin nav entry + cursor variant show. **The cookie is not the security check** — every admin endpoint re-checks the JWT.

---

10. Cart, checkout & payments (Stripe)

Cart — `frontend/src/pages/cart.astro`

- Cart is stored entirely in `localStorage` under key `dotclock-cart`:

```
{
  items: [{ key, id, name, quantity, prices: { price, currency_code }, images }],
  items_count: number
}
```

- `Buy.astro`'s Add-to-cart button writes this shape and dispatches `dc:cart-updated` so `Nav.astro` can refresh its badge.
- Cart page renders directly from localStorage — no server call.

Checkout — `frontend/src/pages/checkout.astro`

1. Collects shipping address.
2. `POST /api/orders` → creates an order with `isPaid: false`, returns the order ID.
3. `POST /api/orders/:id/pay/intent` → returns a Stripe `client_secret`.
4. Stripe Elements confirms payment client-side with that secret.
5. On success → `PUT /api/orders/:id/pay` marks the order paid; client redirects to `/success?orderId=…`.
6. **Webhook safety net** — `POST /api/stripe/webhook` listens for `payment_intent.succeeded` and is the authoritative source for marking paid (the client call is best-effort).

PayPal is wired in as an optional second processor (`paypalController.js`) but is off by default unless you light up its frontend buttons.

---

11. Email pipeline

Two distinct wrappers:

Used by	Where	Notes
Astro `/api/waitlist`	`frontend/src/lib/mailer.ts`	Reads `SMTP_*` env. Returns `{ success }` (caller fire-and-forgets).
Express controllers	`backend/utils/sendEmail.js`	Used for auth: verify, password reset, welcome-after-verify.

Both speak SMTP via nodemailer and pick one of **two delivery modes** based on `SMTP_USER`:

1. **Authenticated SMTP** — when `SMTP_USER` holds a real value (not `REPLACE_ME`), mail goes out through `SMTP_HOST:SMTP_PORT` with auth. The `.env.example` defaults point at Mailtrap, so a fresh `.env` copy "just works" for dev (mail lands in the Mailtrap inbox, not real inboxes).
2. **Host Postfix relay** — when `SMTP_USER` is unset or left as `REPLACE_ME` (the production setup on `abiot.it`), both wrappers relay through the host's Postfix on port 25 via `SMTP_RELAY_HOST` (default `127.0.0.1`). In Docker you can still override that variable if the relay lives outside the container.

Production deliverability (host-relay mode). The host's Postfix + OpenDKIM are configured for the `abiot.it` domain — SPF lists the host IP and OpenDKIM signs `*@abiot.it` with the `dotclock` selector. Therefore **`SMTP_FROM` must be an `@abiot.it` address** (e.g. `DotClock <no-reply@abiot.it>`). Any other domain (e.g. the old `dev@dotclock.local`) goes out unsigned from a non-resolvable domain and Gmail rejects it with `550-5.7.26 … SPF/DKIM did not pass`. Verify a send in `/var/log/mail.log`: a healthy one logs `DKIM-Signature field added (s=dotclock, d=abiot.it)` then `status=sent (250 …)`.

> **Gotcha — "no email" usually means "no database".** If MongoDB is unreachable, registration silently takes a mock path (returns a mock user and sends **no** verify email) and the waitlist takes a mock fallback (sends the welcome mail but never persists). Before chasing SMTP, confirm the DB is up: `GET /api/waitlist/count` returns `{"configured":true,"total":<n>}` when healthy, and the mock `total:999` when the DB is down. See §16 for the deploy gotcha that leaves a container pointed at a dead Mongo.

Waitlist welcome template — `frontend/src/lib/emails/waitlist-welcome.ts`:

- Hand-written HTML + plain-text alternative, no MJML/React Email.
- Mentions the `hello@dotclock.com` reply-to and a one-click `mailto:?subject=unsubscribe`.

12. Internationalisation

Astro config (`astro.config.mjs`):

```
i18n: {
  defaultLocale: "en",
  locales: [
    "en", "it", "es", "fr", "de", "ja", "zh", "ml", "ar",
    "pt", "ko", "hi", "vi", "ru", "pl", "ber", "he", "tr",
  ],
  routing: { prefixDefaultLocale: false }
}
```

Translations (`frontend/src/i18n/ui.ts`) — one big object keyed by locale code; every component reads via:

```
import { useTranslations } from "../i18n/utils";
const t = useTranslations(Astro.currentLocale as any);
t("hero.headline")          // returns the string in the active locale
```

Pages — one file per locale at `pages/<locale>/index.astro` (18 files total — English at the root + 17 locale subfolders). Each imports the same component tree as the English home (they're thin wrappers — Astro doesn't dedupe component imports when the path differs).

`hreflang` — `Layout.astro` emits a complete `<link rel="alternate" hreflang="…">` set for all 17 locales plus `x-default` .

OG locale — `Layout.astro` maps the active locale to an OG locale (`en_GB` , `it_IT` , `es_ES` , `fr_FR` , `de_DE` , `ja_JP` , `zh_CN` , `ml_IN` , `ar_AE` , `pt_PT` , `ko_KR` , `hi_IN` , `vi_VN` , `ru_RU` , `pl_PL` , `ber_DZ` , `he_IL` , `tr_TR`) and emits `og:locale:alternate` for every other one.

RTL — `<html dir>` is set to `rtl` when the active locale is `ar` or `he` , otherwise `ltr` . All other locales render LTR (Berber uses Tifinagh which is technically RTL/bidirectional, but written as a Tamazight transliteration here it reads LTR).

Language switching — `Nav.astro` lists all 17 locales in its dropdown with native-script labels (`Português` , `한국어` , `Tamaziɣt` , `Türkçe` , `Tiếng Việt` , `Русский` , `हिन्दी` , `עברית` , etc.). Selecting one navigates to that locale's home via `data-astro-reload` . A `dc-lang` cookie is also set so the Express layer can honour the choice on subsequent server-rendered requests.

13. SEO & structured data

`Layout.astro` is the SEO heart. Every page goes through it. It emits:

- `<title>` (from prop `title`, default `<site.name> — <site.tagline>`).
- `<meta name="description">`, `keywords`, `author`, `theme-color`, `color-scheme`.
- `<link rel="canonical">` computed from `Astro.url.pathname` + the configured site URL.
- `<meta name="robots">` and `<meta name="googlebot">` — flipped to `noindex, nofollow` when `noindex` prop is true (used by cart, checkout, success, login, register, reset-password, account, orders, admin).
- Full **hreflang** matrix (9 locales + `x-default`).
- **Open Graph** — `og:type=product`, title/description/image, locale + alternates.
- **Twitter Card** — `summary_large_image`, `@dotclock` site/creator.
- **Product OG** — `product:price:amount`, `:currency`, `:availability`, `:condition`, `:brand`, `:retailer_item_id`.
- Self-hosted favicon + `apple-touch-icon` + webmanifest.
- 5 JSON-LD blocks: `Product` (with `Offer`, shipping, returns), `FAQPage`, `Organization`, `WebSite` (with sitelinks search box), `BreadcrumbList`.

Sitemap — `@astrojs/sitemap` integration auto-emits `sitemap-index.xml` + per-locale sitemaps.

`robots.txt` — allows `/`, disallows `/api/`, `/uploads/`, `/account`, `/orders`, `/cart`, `/checkout`, `/success`, `/reset-password/`, `/login`, `/register`, `/admin`. Sitemap URL points to `https://dotclock.com/sitemap-index.xml`.

Lighthouse SEO scores **100/100** on `/` against the local production build.

---

14. Theming, design tokens & animation system

Tokens — `frontend/src/styles/global.css` defines all colours, sizes, easings, durations as CSS custom properties on `:root`. The light theme is the same token set re-declared under `[data-theme="light"]`. The theme is initialised by an inline `<script is:inline>` in `Layout.astro` so there's no FOUC.

Token group	Examples
Surfaces	`--bg`, `--bg-elev-1`, `--bg-elev-2`, `--line`
Text	`--text`, `--text-muted`, `--text-dim` (recently bumped for WCAG AA)
Brand	`--accent` (LED red), `--accent-warm` (LED amber), `--oak`, `--oak-light`
Type sizes	`--fs-xs` … `--fs-4xl`
Spacing	`--space-1` … `--space-20`
Radii	`--radius-sm`, `--radius`, `--radius-lg`, `--radius-xl`
Motion	`--ease-out`, `--ease-spring`, `--ease-spring-bouncy`, `--dur-fast`, `--dur-slow`, `--dur-reveal`

Cascade layers — base/components/utilities to keep specificity predictable.

Scroll-driven animations — uses modern CSS:

- `animation-timeline: scroll(root)` drives the `#scroll-progress` bar.
- `animation-timeline: view()` reveals `.reveal` sections as they cross the viewport.
- JS fallback in `Layout.astro` covers Firefox and older Safari via `IntersectionObserver`.

View transitions — Astro's `ClientRouter` is mounted in `Layout.astro`. `global.css` adds polished `::view-transition-old(root)` / `::view-transition-new(root)` keyframes (200 ms fade-out, 260 ms fade-in with subtle Y translation). The `#scroll-progress` bar is given its own no-op transition group so it's not caught in the page fade. `prefers-reduced-motion` reduces durations to 1 ms.

Interactive effects — `Layout.astro` ships **delegated** handlers on `document` for:

- `[data-tilt]` cards (cursor → 3D rotation),
- `[data-magnetic]` CTAs (cursor pull),
- `[data-ripple]` click ripple.

Delegated handlers survive view-transition DOM swaps automatically, so no re-binding is needed. The reveal observer and word-split re-run on `astro:after-swap`.

Cursor (`Cursor.astro`) — single accent-warm spotlight glow that follows the pointer; native OS cursor stays visible underneath. See §15.5.

Below-fold perf — `content-visibility: auto` + `contain-intrinsic-size` applied to `#features`, `#specs`, `#faq`, `.reviews`, `.builder`, `.buy`, and `.footer`. Sections skip painting until they're near the viewport — measurable LCP win on mid-tier mobile, no layout shift thanks to fixed intrinsic sizes.

Section dividers — `<div class="section-divider"><span/><span/><span/></div>` drops three pulsing LED beads between major homepage sections (Craft↔DayNight, BannerBuilder↔Features, Setup↔Specs, FAQ↔Waitlist). 2.4s pulse, animation off for reduced motion.

Conic-gradient Buy border — animated rotating highlight around the Buy card. Uses `@property --buy-angle` so the angle actually tweens (CSS would otherwise refuse to interpolate between `conic-gradient(from Xdeg, …)` values). 9s rotation, ~0.6 opacity. Sits behind `.buy-content` (z-index 0 inside an isolated stacking context) so it never paints over the Buy button.

LED particle burst — 42 short-lived pixel sprites in `--accent` / `--accent-warm` / white explode from the Buy button on successful add-to-cart. Gravity + friction physics, ~0.9s lifetime, lazy-mounted shared canvas. Skipped on `prefers-reduced-motion`.

Cursor-aware hero glow — 220 px accent-warm radial highlight follows the pointer across the hero LED banner. CSS-only render; JS just writes `--gx` / `--gy` percentages on `pointermove`, rAF-throttled. Settles to centre on touch / reduced-motion.

Universal button spinner — `<button data-loading="true">` works on ANY button class: dims content, paints a centred ring. See §15.6.

Universal focus ring — `:focus-visible` paints a 2 px accent outline + 12 px LED-glow `box-shadow`. Keyboard-only; transition removed for reduced motion.

---

15. Recently shipped features

15.1 Banner-builder URL sharing

Files: `frontend/src/components/BannerBuilder.astro`

The interactive "pick what scrolls" module checklist now persists user selections so they can be shared.

State model

- The selection is a sorted, comma-joined list of module IDs (excluding locked modules — those are always on).
- Encoded into the URL hash as `#banner=temp,city,sunrise`.

Precedence order at load

1. **URL hash** (`location.hash`) — if present, wins.
2. `localStorage["dc-banner-modules"]` — fallback for returning visitors.
3. **Defaults** — the `default: true` modules in `data/site.ts`.

Behaviour

- Every checkbox change writes both `localStorage` and `history.replaceState()` so the URL bar always reflects the current setup (no extra back-stack entry).
- A **Share this setup** button copies the share link via the Clipboard API. On mobile (`navigator.share` available), it offers the native share sheet. Falls back to `window.prompt()`.
- A **Reset** button restores the defaults and clears the hash.

Why a hash and not a query param? The hash never hits the server — keeps the SSR cache key stable and avoids a redundant render per unique selection.

15.2 Live batch inventory + ship-by date

Files: `frontend/src/pages/api/inventory.ts`, `frontend/src/components/Buy.astro`, `frontend/src/data/site.ts`

A new `batch` config in `data/site.ts`:

```
export const batch = {
  target:    Number(process.env.BATCH_TARGET ?? 100),
  startedAt: process.env.BATCH_STARTED_AT ?? "2026-05-01T00:00:00Z",
  shipDays:  Number(process.env.BATCH_SHIP_DAYS ?? 3),
};
```

`GET /api/inventory`:

1. Reads `orders` via the native driver.
2. Counts non-cancelled paid orders where `paidAt >= batch.startedAt` (fallback to `createdAt`).
3. `remaining = max(target - sold, 0)`; `outOfStock` when `remaining <= 0`.
4. `shipBy` = today + `shipDays` **business days** (skips Sat/Sun). Returns `null` when sold out.
5. Cached `public, max-age=30, stale-while-revalidate=120` so a viral burst doesn't hammer Mongo.

Buy section UI

- SSR renders a sensible fallback badge so the page is never blank if the fetch fails.
- On `astro:page-load`, `Buy.astro` calls `/api/inventory` and rewrites the badge to:
  - `In stock · ships by Fri 28 May` (normal)
  - `Only 7 left in this batch · ships by Mon 1 Jun` (when ≤ 10 % remaining — yellow `data-state="low"`)
  - `Batch sold out — join the next batch` (red, reveals the restock form, disables Buy button)

15.3 Back-in-stock notify

Files: `frontend/src/components/Buy.astro` (reuses existing `POST /api/waitlist`)

When `/api/inventory` returns `outOfStock: true`, the Buy section hides the Buy button and reveals a `<form id="restock-form">`. Submitting posts to `POST /api/waitlist` with `source: "restock"` — so the existing waitlist endpoint becomes the segmented source of truth (export `source = "restock"` to message everyone waiting on the next batch).

No backend change required.

Validation, error mapping, spinner and "✓ You're on the list" success state are all handled client-side in the same script block that does the inventory fetch.

15.4 Currency auto-switch by IP

Files: `frontend/src/pages/api/locale.ts`, `frontend/src/components/Buy.astro`, `frontend/src/data/site.ts`

A new `currencies` table in `data/site.ts`:

```
EUR: { symbol: "€",   display: 229,   countries: ["DEFAULT"] }
USD: { symbol: "$",   display: 249,   countries: ["US"] }
GBP: { symbol: "£",   display: 199,   countries: ["GB"] }
CAD: { symbol: "CA$", display: 339,   countries: ["CA"] }
AUD: { symbol: "A$",  display: 379,   countries: ["AU", "NZ"] }
INR: { symbol: "₹",   display: 19999, countries: ["IN"] }
JPY: { symbol: "¥",   display: 35900, countries: ["JP"] }
CHF: { symbol: "CHF", display: 229,   countries: ["CH", "LI"] }
```

`GET /api/locale` resolves the visitor's country in this order:

1. `?country=US` **query string** (QA override — works without a VPN).
2. `CF-IPCountry` header — Cloudflare's free geolocation.
3. `X-Vercel-IP-Country` — Vercel's equivalent.
4. `Accept-Language` heuristic (e.g. `en-IN` → IN).
5. Defaults to `DEFAULT` → EUR.

Response is cached `public, max-age=300, stale-while-revalidate=3600`, with `Vary` set on the country headers so each visitor sees their own currency at the CDN edge.

Buy section UI

On `astro:page-load`, `Buy.astro` calls `/api/locale`, and if the resolved currency isn't EUR it swaps:

- `#buy-price-symbol` text (e.g. `€` → `$`),
- `#buy-price-display` text (e.g. `229` → `249`),
- Reveals the small "· charged in EUR" disclaimer next to the tax label.

Stripe still charges in EUR. This is purely a display affordance — most international visitors mentally convert "€229" with friction, and seeing "$249" removes it without complicating the payment processor.

15.5 Spotlight glow cursor

Files: `frontend/src/components/Cursor.astro` (mounted in `Layout.astro`)

A single radial-gradient blob in `--accent-warm` follows the pointer with exponential lerp (`(target - current) * 0.28` per frame). Two `<html>` class names drive the visuals:

- `.spotlight-on` — visible while the pointer is inside the document.
- `.spotlight-hot` — blob brightens + grows to 480×480 when over `a, button, input, select, textarea, label, summary, [data-ripple], [data-magnetic], [role="button"]`.

Crucially: the native OS cursor stays visible underneath at all times — no `cursor: none`. This kills the entire class of "where did my cursor go?!" bugs that haunted earlier dot+ring implementations.

Robustness:

- Element is `transition:persist`, so coords + classes survive view transitions.
- Re-asserts state on `astro:after-swap` (re-adds `.spotlight-on`, restarts rAF).
- `visibilitychange` listener restarts rAF when the tab is foregrounded.
- Hidden entirely on touch / coarse pointer via `@media (hover: none) and (pointer: coarse)`.
- `prefers-reduced-motion` skips the lerp loop (snap to target each frame) and uses a shorter opacity transition.
- Light theme: `mix-blend-mode: multiply` with reduced opacity so the warm glow doesn't blow out cream surfaces.

15.6 Loader system

Three coordinated patterns, all in `frontend/src/styles/global.css` + `frontend/src/components/Loader3D.astro`.

1. **`.dc-skeleton`** — drop-in placeholder utility class with a 1.6 s `dc-shimmer` animation. Variants: `--text`, `--line`, `--avatar`, `--card`. 4× slower animation under `prefers-reduced-motion`.

   ```
   <span class="dc-skeleton dc-skeleton--text" style="width: 60%"></span>
   ```

2. **`<button data-loading="true">`** — universal centred spinner. Hides the button's content via `visibility: hidden` on children, paints a `::after` ring spinning at 0.7 s. Works on **any** button class — drop the attribute on, take it off when done.

```
btn.setAttribute("data-loading", "true");
// …await something
btn.removeAttribute("data-loading");
```

3. `<Loader3D />` — full-viewport overlay (`position: fixed; inset: 0;`) for blocking operations. Glassier backdrop than before (`backdrop-filter: blur(14px) saturate(140%)`), spring scale-in on enter, scale-down on exit, soft pulsing LED core at the centre of the 3 rotating rings. Stretched animation durations for reduced motion.

```
window.showLoader("Authenticating…");
await fetch(…);
window.hideLoader();
```

15.7 Live activity ticker + animated count

Files: `frontend/src/pages/api/waitlist/recent.ts` , `frontend/src/components/Waitlist.astro`

Activity ticker — `GET /api/waitlist/recent?limit=6` returns the last N waitlist signups anonymised to a country/region derived from the `locale` field already stored on signup. No names, no emails leak — only `{ region, ts }` . 60s cache + 5min SWR.

```
{
  "items": [
    { "region": "Italy",   "ts": "2026-05-25T13:42:11Z" },
    { "region": "Germany", "ts": "2026-05-25T13:30:04Z" }
  ],
  "configured": true
}
```

The Waitlist section renders a small ticker under the count badge that rotates one item every 4.2 s with a 320 ms opacity+Y fade: `"Someone from Italy joined · 2 min ago"` .

Animated count — the `@property --num` system that was already declared in `global.css` is now actually wired up. On load and after each successful signup:

```
countEl.setAttribute("data-counting", "true");
countEl.style.setProperty("--num", "0");
requestAnimationFrame(() => countEl.style.setProperty("--num", String(total)));
// after 1800ms, hand off to plain textContent for screen readers:
setTimeout(() => {
  countEl.removeAttribute("data-counting");
  countEl.textContent = new Intl.NumberFormat().format(total);
}, 1800);
```

`font-variant-numeric: tabular-nums` keeps the digits from jittering layout while they tween.

15.8 `/compare` page

File: `frontend/src/pages/compare.astro`

Honest side-by-side: DotClock vs LaMetric Time vs Vestaboard vs Tidbyt, across 13 rows (display, materials, app-required, fees, alerts, languages, warranty, power, price). DotClock column visually featured with an accent border + accent header + accent price; competitor "no" answers use a neutral em-dash, not a red ×. Sticky table header, horizontal scroll-snap on mobile so each column lands cleanly when swiping. Bottom CTA links back to `/#buy` with the live price and magnetic + ripple affordances. Honest disclaimer: prices/feature claims for competitors are point-in-time, with `support@dotclock.com` listed for corrections.

Linked from Footer "Product" column.

15.9 `/firmware` changelog + RSS feed

Files: `frontend/src/data/firmware.ts` (source of truth), `frontend/src/pages/firmware.astro`, `frontend/src/pages/firmware.xml.ts`

Single data source feeds both the HTML page and the RSS feed. Each entry: `version` + `date` (ISO) + `summary` + `changes[]`, where each change has `kind: "feature" | "fix" | "perf" | "security"` and `text`.

`/firmware` — reverse-chronological release notes. Latest entry gets an accent border + soft accent shadow + "Latest" pill. Each change row gets a colour-coded pill (feature = accent, fix = warm, perf = oak, security = red). Anchor links per version (`#v3.4.0`) with `scroll-margin-top` so the sticky nav doesn't cover the heading on jump.

`/firmware.xml` — RSS 2.0 feed served via an Astro endpoint. Real `Content-Type: application/rss+xml`, `atom:self` link, CDATA-wrapped HTML body so Reeder / Feedly / GitHub Watch consume it cleanly. Cached 5 min + SWR 1 hr.

Auto-discovered via `<link rel="alternate" type="application/rss+xml" title="DotClock firmware releases" href="/firmware.xml" />` in `Layout.astro`'s `<head>`.

JSON-LD: `SoftwareApplication` schema on `/firmware` so the latest version + date is eligible for Google rich-result rendering.

15.10 17 locales

The site now ships in **17 languages** (English + 16 localised home pages):

Region	Locales
Western Europe	English (en), Italian (it), Spanish (es), French (fr), German (de), Portuguese (pt)
Eastern Europe	Polish (pl), Russian (ru)
MENA	Turkish (tr), Hebrew (he), Arabic (ar), Berber (ber)
South Asia	Hindi (hi), Malayalam (ml)
Southeast Asia	Vietnamese (vi)
East Asia	Mandarin (zh), Korean (ko), Japanese (ja)

Everything that's locale-aware was updated in lockstep:

- `astro.config.mjs` `i18n.locales` lists all 17.
- `Layout.astro` emits hreflang + OG locale alternates for all 17 plus `x-default`.
- `<html dir>` is set to `rtl` for `ar` and `he`.
- `Nav.astro` lists all 17 in its dropdown with native-script labels in a regional order.
- `src/i18n/ui.ts` has full translation objects per locale.
- `pages/<locale>/index.astro` exists for each.

15.11 Motion + look polish

Several smaller upgrades that landed together as the "10× better" pass:

- **Staggered Features grid reveal** — each of the 8 feature cards eases in 60 ms after the previous via the global `.reveal--stagger` rule.
- `content-visibility: auto` on Features / Specs / FAQ / Reviews / Builder / Buy / Footer with explicit `contain-intrinsic-size` — measurable LCP win, no layout shift.
- **Section-divider LED beads** between major sections.
- **Magnetic CTAs** extended to Footer guide-PDF download, Buy restock submit, BannerBuilder Reset.
- **Custom focus ring** — 2 px accent outline + 12 px LED glow on `:focus-visible`.
- **Astro view transitions** — snappier 200 ms fade-out / 260 ms fade-in keyframes with subtle Y translation, with `prefers-reduced-motion` opt-out and a no-op transition group for the persisted scroll-progress bar so it doesn't get caught in the page fade.
- **Animated conic-gradient border** on the Buy card (see §14).
- **LED-particle burst** on add-to-cart (see §14).
- **Cursor-aware glow** on the hero LED banner (see §14).

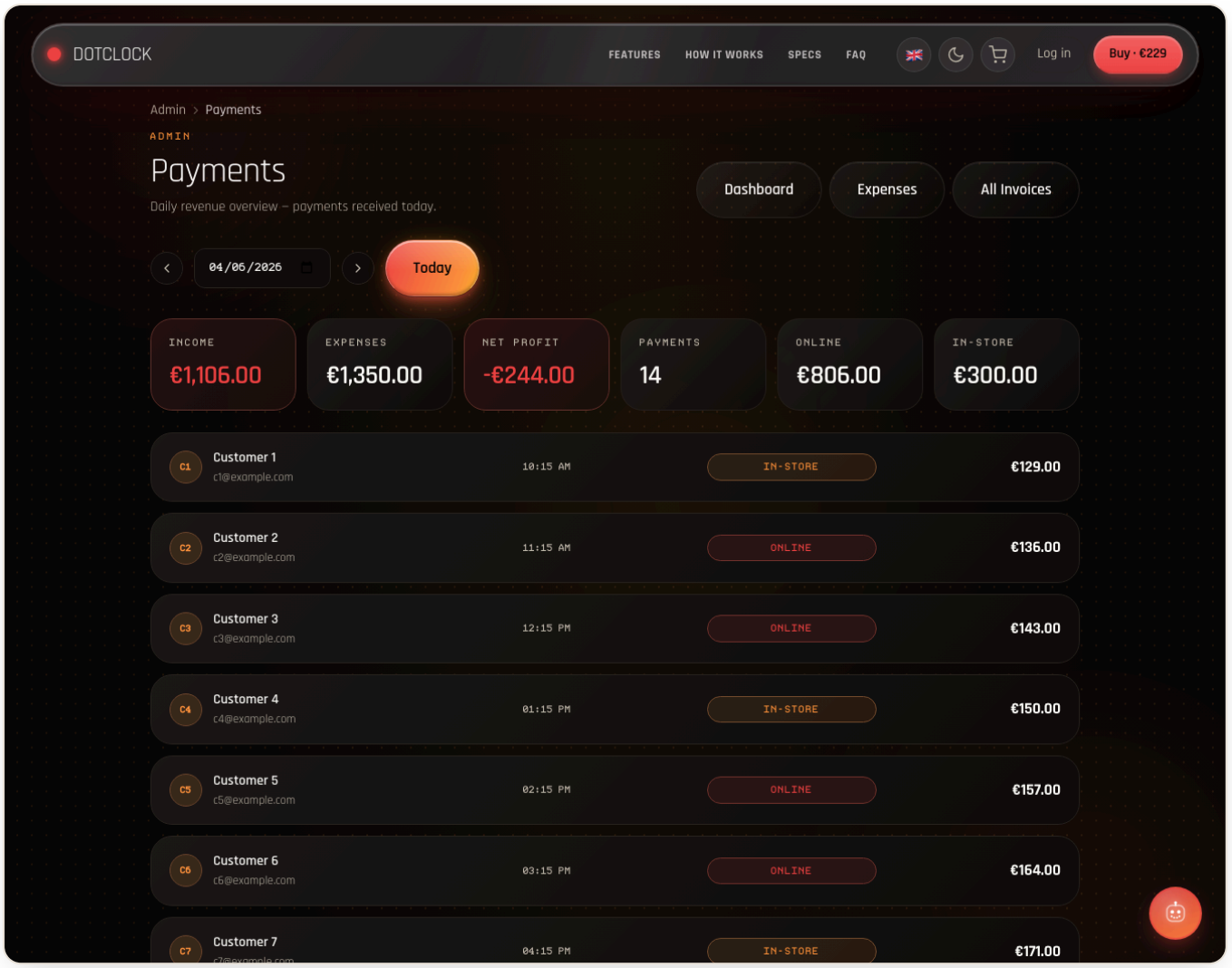
15.12 Bug fixes (Buy button + Carousel arrows)

- **Buy button** was visually layered under the rotating conic `::after` overlay because `.buy-card` has `isolation: isolate` and the `::after` sat at `z-index: 1` while `.buy-content` (containing the button) was at `auto`. Even with `pointer-events: none` on the overlay, animating a layer on top of a CTA is fragile and could mask the button. **Fix:** moved `::after` to `z-index: 0` and lifted `.buy-content` + `.buy-decoration` to `z-index: 1` with `position: relative`. The conic border now paints behind all interactive surfaces.
- **Carousel arrows** were invisible on touch devices because the CSS only set `opacity: 1` under `.features-image-wrap:hover`, and touch devices have no hover. **Fix:** baseline `opacity: 0.55` (subtle desktop hint), `1` on parent hover or focus-visible, and a `@media (hover: none) and (pointer: coarse)` block forcing `opacity: 1` with a denser background. The carousel JS also now re-binds on `astro:page-load` (the original captured DOM refs at module load, breaking after any client-side navigation) and uses an `__dcCarousel` flag to avoid double-binding.

15.13 Admin Payments dashboard — daily revenue

Route: `/admin/payments` · **API:** `GET /api/orders/billing/daily?date=YYYY-MM-DD` · **Files:** `frontend/src/pages/admin/payments.astro`, `backend/routes/orderRoutes.js`

A manager-facing "what came in today" view. Until now the only money screen was the owner-only invoice list; this adds a focused daily dashboard and **opens the billing routes to all admin roles** (manager, admin, owner) so floor managers can see takings without owner access.



Each functionality in depth:

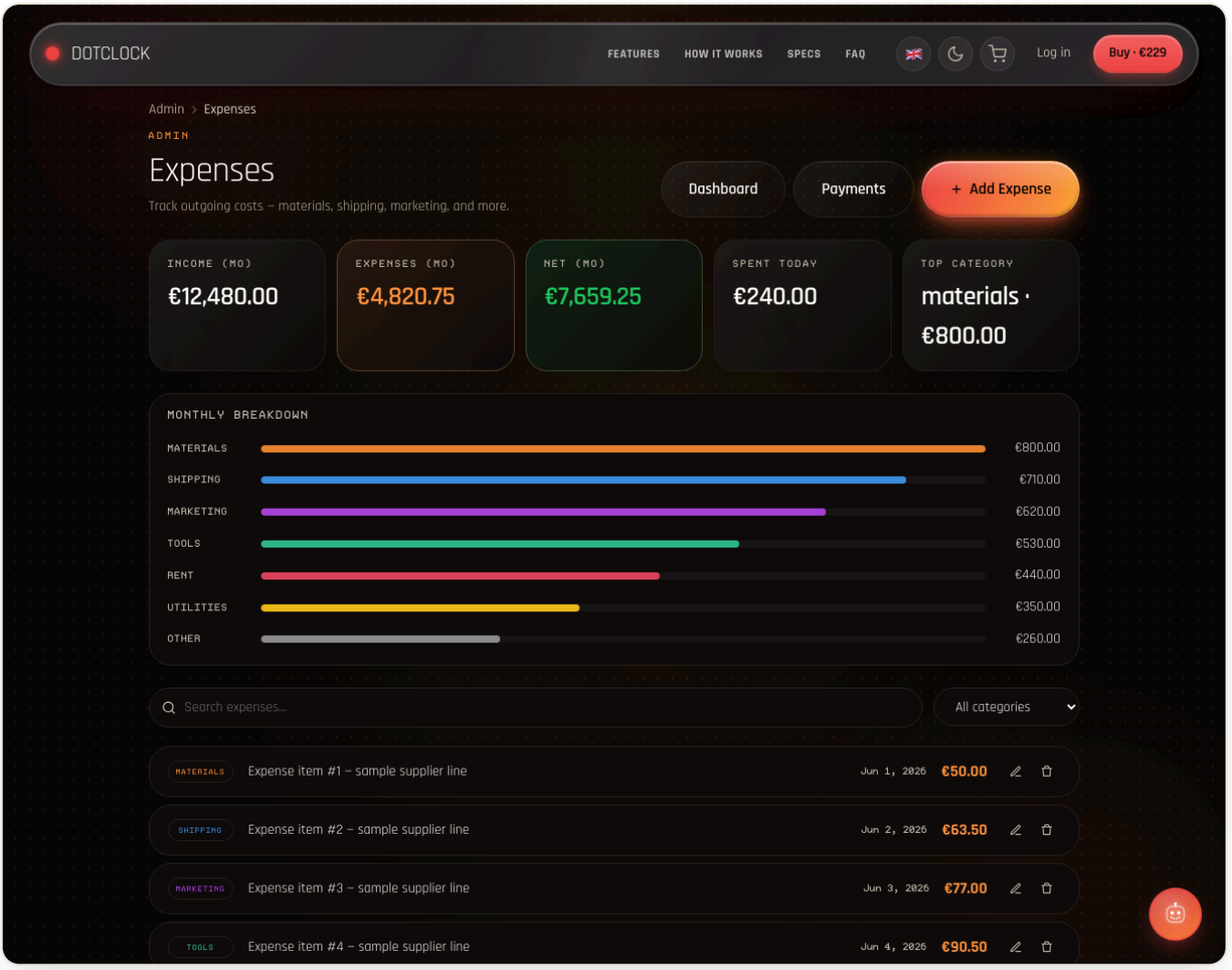
- **Date navigation** — `◀` / `▶` step a day at a time, a native date picker jumps to any date, and a **Today** shortcut snaps back. Every change refetches `/api/orders/billing/daily?date=…`.
- **Stat strip** — four cards: **Total Revenue** (highlighted), **Payments** (count), **Online** total and **In-store** total. Values are formatted with `Intl.NumberFormat` in each order's currency.
- **Per-payment list** — one card per paid order: an initials avatar, customer name + email (falls back to the shipping address for guest / in-store sales), the paid-at time, a colour-coded **channel badge** (`online` = accent, `in-store` = amber), and the amount in right-aligned tabular figures. Cards stagger in with a rise animation.
- **Connected financials (income → expenses → net)** — the endpoint also aggregates the **same day's expenses** and returns `income`, `expenses`, and `net` (= income − expenses). The stat strip therefore shows **Income**, **Expenses** and a **Net Profit** card that turns **green in profit / red at a loss**, so a manager reads the day's P&L at a glance. A header link crosses to **Expenses**.
- **Pagination** — the per-payment list paginates at **10 / page** (prev · numbered · next), matching every other admin list; the pager hides when there's a single page.

- **Access change** — `GET /api/orders/billing` and the new `GET /api/orders/billing/daily` both moved from `owner` to `admin` middleware. The page additionally does a defence-in-depth gate (`/api/users/profile` → redirect non-admins to `/`).
- **The aggregation** — the endpoint matches `{ isPaid: true, paidAt: { $gte: dayStart, $lt: dayEnd } }`, populates the buyer (`name email avatar`), sorts newest-first, tallies `online` vs `inStore` from each order's `saleChannel`, sums the day's expenses, and returns rounded income/expenses/net plus the raw orders.

15.14 Expenses management

Route: `/admin/expenses` · **API:** `/api/expenses` (CRUD) + `/api/expenses/summary` · **Files:** `frontend/src/pages/admin/expenses.astro`, `backend/routes/expenseRoutes.js`, `backend/models/expenseModel.js`

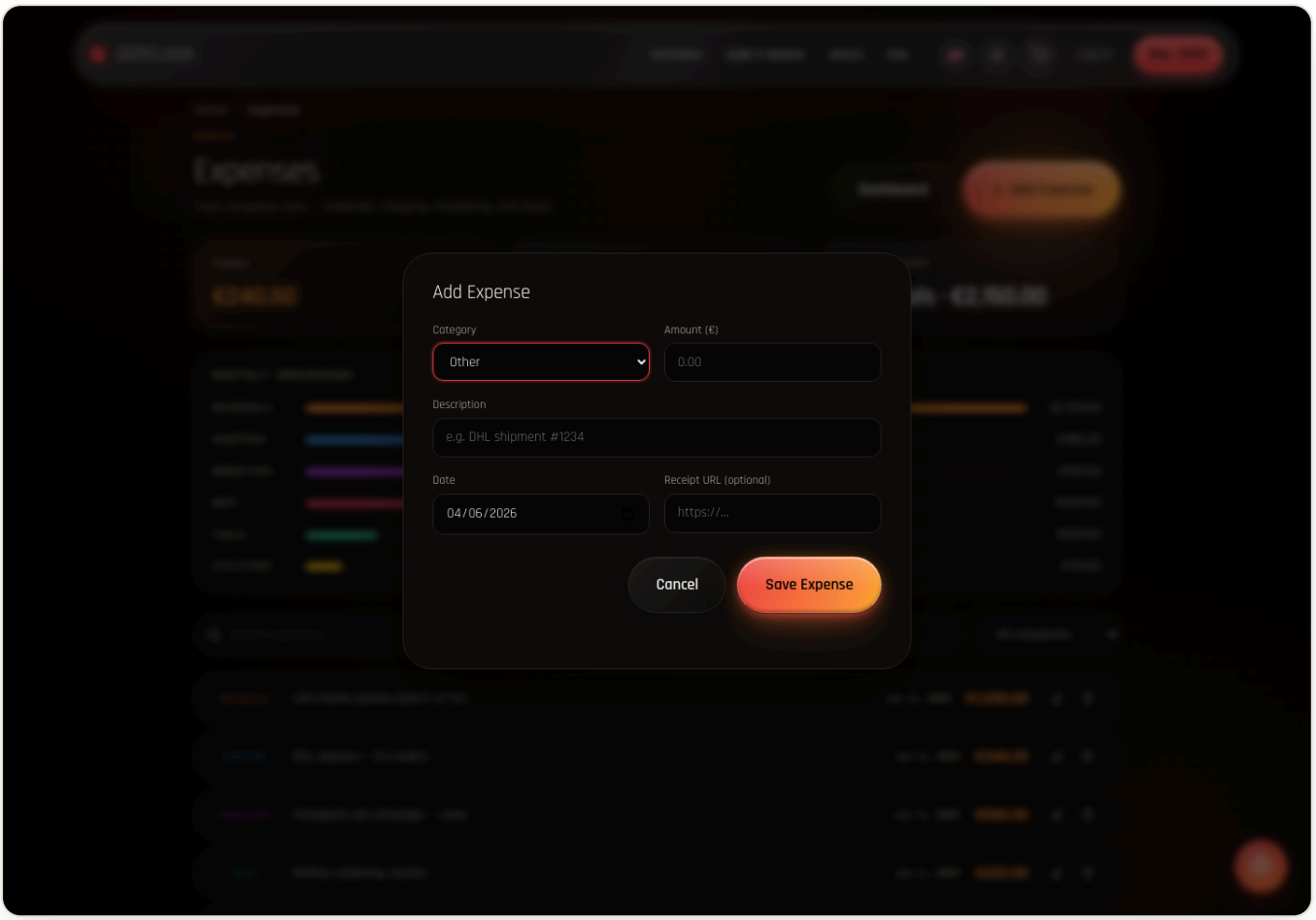
Bookkeeping for outgoing costs — the counterpart to the revenue dashboard, so a manager can read profit at a glance.



- **Data model** (`Expense`): `category` (enum — shipping, materials, tools, marketing, rent, utilities, other), `description` (≤ 500), `amount` (≥ 0), `currency` (default `eur`), `date`, optional `receiptUrl`, `createdBy` (User ref), and timestamps. Indexed on `date` and `{ category, date }`.
- **Connected financials (income → expenses → net)** — `GET /api/expenses/summary` also joins **income from paid orders** for today and the month, returning `income` and `net` alongside the expense

totals. The stat strip shows **Income (mo)**, **Expenses (mo)** and a colour-coded **Net (mo)** card (green/red), with a header link across to **Payments** — so revenue and costs read as one connected picture.

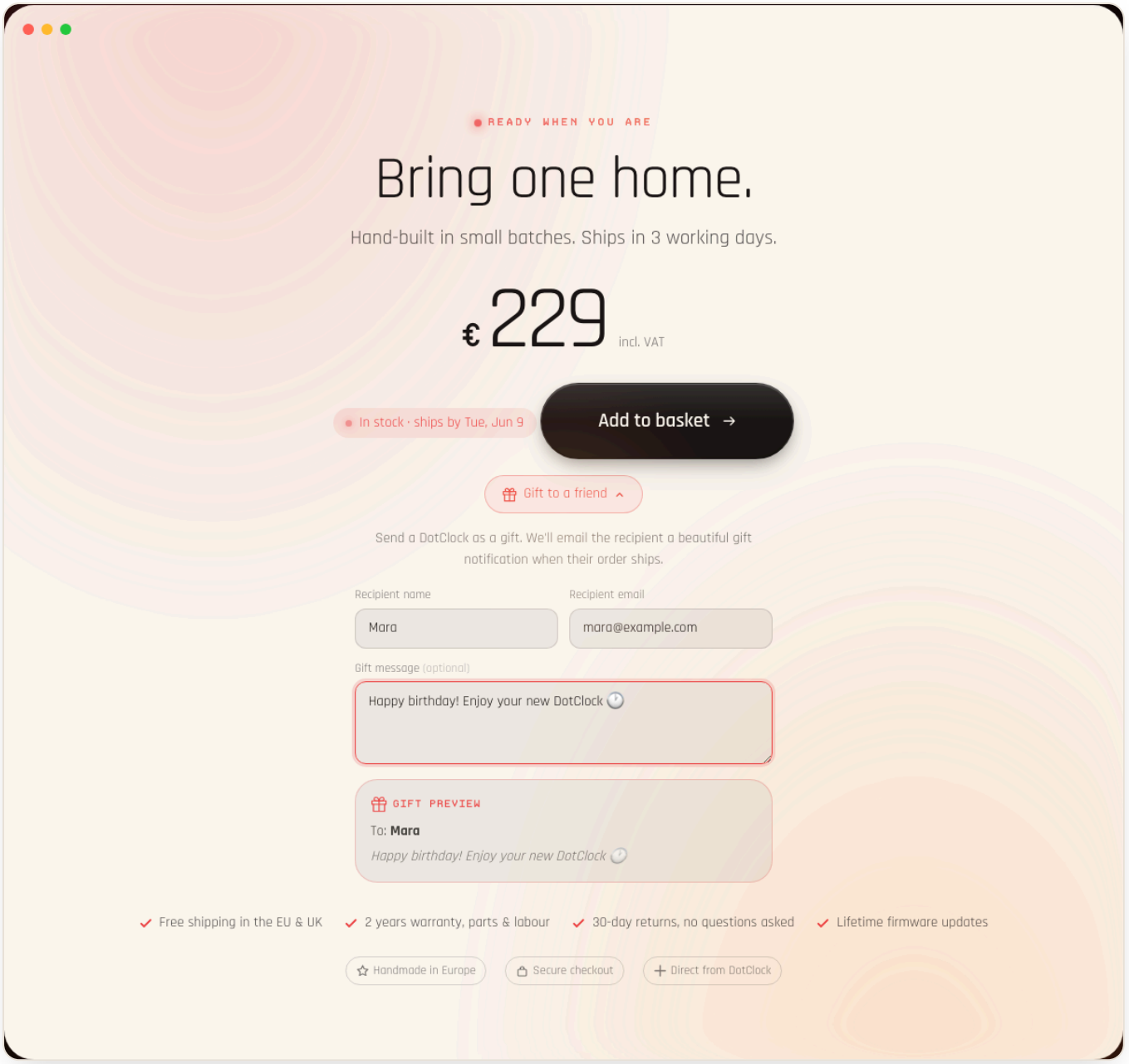
- **Summary cards** — **Income (mo)**, **Expenses (mo)**, **Net (mo)**, **Spent today**, and **Top Category** (largest monthly spend). Driven by `GET /api/expenses/summary`, which aggregates today's and the month's spend grouped by category, plus monthly/today income from orders.
- **Pagination** — the expense list paginates at **12 / page** (prev · numbered · next); changing the search or category filter resets to page 1.
- **Monthly category breakdown** — one horizontal bar per category, each given a fixed colour and scaled to the largest category; widths animate on load.
- **Search + filter** — live text search over description/category plus a category dropdown, applied client-side to the already-fetched list.
- **Add / Edit** — a **centred modal** (`<dialog>`) with category, amount (€), description, date and an optional receipt URL. `POST` creates, `PUT` updates.
- **Delete** — only **elevated** roles (admin/owner) see the delete control; managers can add/edit but not delete. Enforced in the UI (`isElevated`) **and** server-side (`DELETE /api/expenses/:id` returns 403 for non-elevated).
- **Validation** — the server requires a non-empty description and a finite amount ≥ 0, and rounds amounts to 2 dp.



> All newly added modals open **dead-centre** of the viewport (`position: fixed; inset: 0; margin: auto`) regardless of scroll position, over a blurred backdrop, with a spring scale-in (disabled under `prefers-reduced-motion`).

15.15 Gift a DotClock

Where: the Buy card (`frontend/src/components/Buy.astro`), carried through to the cart.

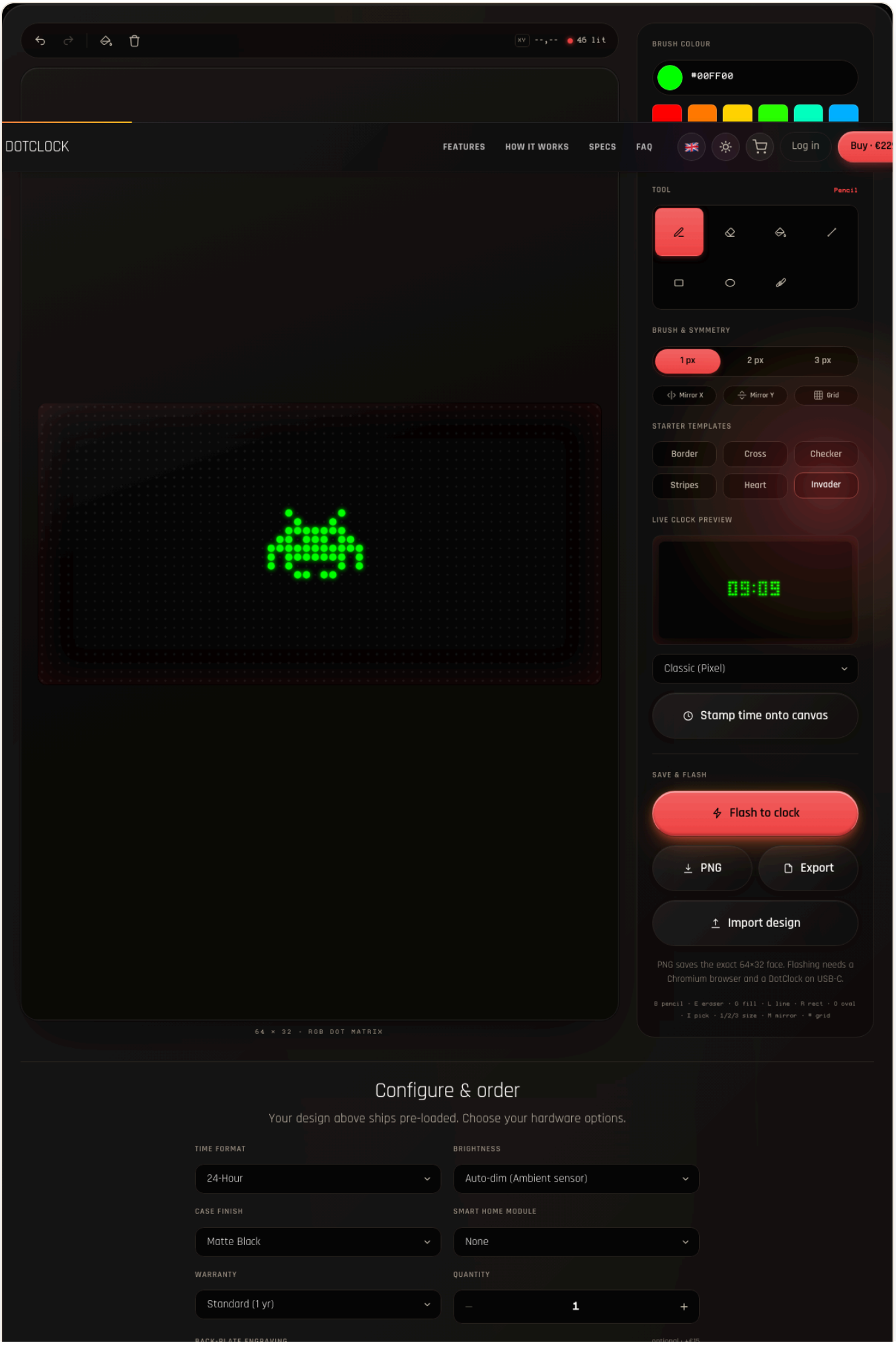


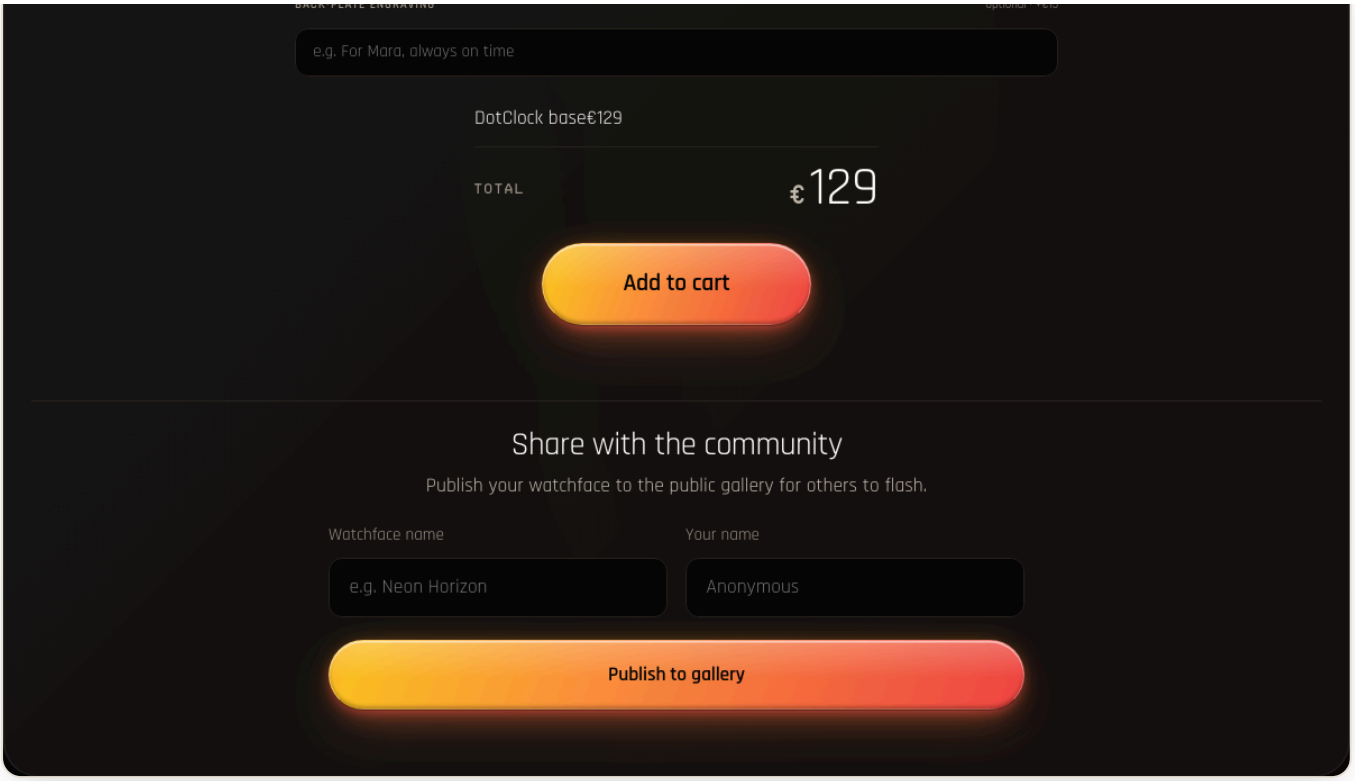
- **Gift toggle** — a "Gift to a friend" disclosure under the buy button expands a panel (animated slide-in) with recipient **name**, **email**, and an optional **message** (≤ 300 chars).
- **Live preview** — as the buyer types a name + email, a "Gift preview" card shows `To: <name>` and the message (or a friendly default), so they see exactly what the recipient will receive.
- **Cart integration** — a filled gift panel makes Add-to-cart create a **separate gift line item** (unique key `…-gift-<timestamp>`) labelled "🎁 … (Gift)" carrying `gift: true` and `giftRecipient { name, email, message }` . Ordinary purchases still stack. The form resets after adding and the feedback line confirms "🎁 Gift for `<name>` added to cart."

15.16 DotClock Studio — pro drawing tools & deep configurator

Route: `/studio` · **File:** `frontend/src/pages/studio.astro`

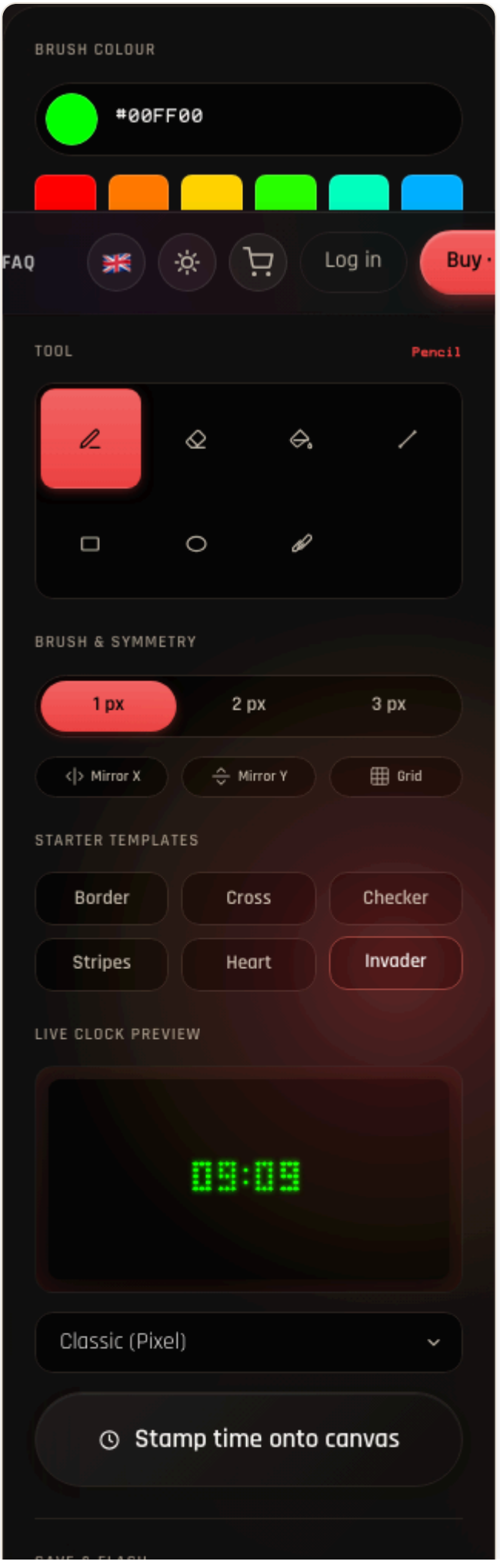
The studio combines the 64×32 watchface pixel editor, a hardware configurator, and community publishing in one workspace. This release turns the editor from a basic pencil/eraser into a real pixel-art tool and deepens the order flow.

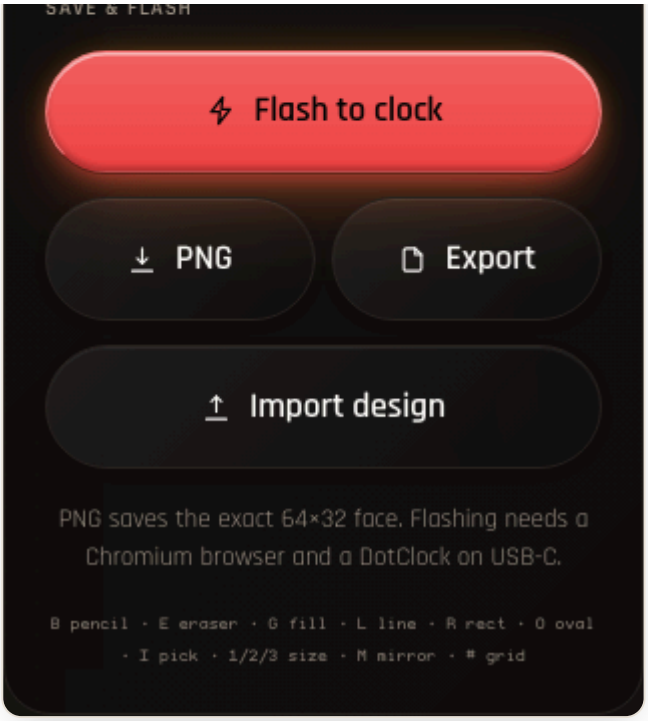

DOTCLOCK
FEATURES
HOW IT WORKS
SPECS
FAQ
Log in
Buy · €22
BRUSH COLOUR
#00FF00
TOOL
Pencil
BRUSH & SYMMETRY
1 px
2 px
3 px
Mirror X
Mirror Y
Grid
STARTER TEMPLATES
Border
Cross
Checker
Stripes
Heart
Invader
LIVE CLOCK PREVIEW
09:09
Classic (Pixel)
Stamp time onto canvas
SAVE & FLASH
Flash to clock
PNG
Export
Import design
PNG saves the exact 64×32 face. Flashing needs a Chromium browser and a DotClock on USB-C.
64 × 32 · RGB DOT MATRIX
Configure & order
Your design above ships pre-loaded. Choose your hardware options.
TIME FORMAT
24-Hour
BRIGHTNESS
Auto-dim (Ambient sensor)
CASE FINISH
Matte Black
SMART HOME MODULE
None
WARRANTY
Standard (1 yr)
QUANTITY
1



Drawing engine

- **Seven tools** — Pencil, Eraser, **Fill bucket** (contiguous flood-fill), **Line**, **Rectangle**, **Ellipse**, and **Eyedropper**. Shape tools render a live translucent preview while dragging and commit on release; freehand strokes are **interpolated** (Bresenham) so fast drags never leave gaps.
- **Brush size** — 1 / 2 / 3 px square brush, applied to the pencil, eraser and shape strokes.
- **Mirror / symmetry** — independent **Mirror X** and **Mirror Y** toggles that combine into 4-way symmetry — ideal for clock faces.
- **Grid overlay** — a toggleable per-cell guide drawn on the canvas.
- **History & helpers** — 80-deep **undo/redo**, flood-fill the whole canvas, clear, and a live **XY + lit-LED** readout. Everything persists to `localStorage` and survives reloads.


BRUSH COLOUR
#00FF00
FAQ
Log in
Buy
TOOL
Pencil
BRUSH & SYMMETRY
1 px
2 px
3 px
Mirror X
Mirror Y
Grid
STARTER TEMPLATES
Border
Cross
Checker
Stripes
Heart
Invader
LIVE CLOCK PREVIEW
09:09
Classic (Pixel)
Stamp time onto canvas



Live clock preview (non-destructive) — a mini LED panel ticks the **current time every second** in the chosen font (Classic 3×5 / Modern 4×6 / Retro 5×7). Previously, changing the font *wiped your artwork*; the preview is now a separate surface that never touches the canvas. A **"Stamp time onto canvas"** button paints the time in your brush colour on demand (snapshot-protected so it's undoable).

Starter templates — one click loads Border, Cross, Checker, Stripes, Heart or Invader, rendered in the current brush colour.

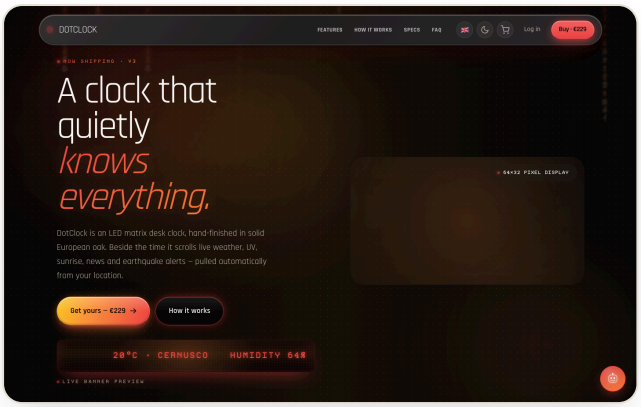
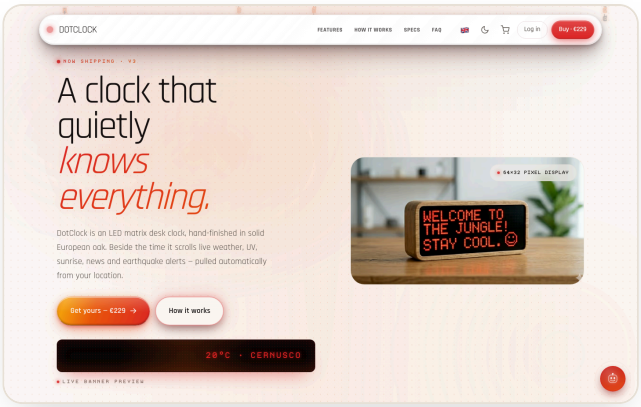
Save / share / flash — **Flash to clock** over **Web Serial** (Chromium + USB-C), **PNG** export of the exact 64×32 face, re-editable **JSON** export/import, and **Publish to gallery** (`POST /api/gallery`).

Deep configurator (Configure & order) — time format, brightness, case finish, smart-home module, **extended warranty**, a **quantity stepper**, and optional **back-plate engraving** (+€15). An **itemised price breakdown** lists every add-on plus the per-unit × quantity maths and updates live. Add-to-cart writes a custom line item carrying the full spec (font, format, brightness, case, module, warranty, engraving, custom-grid flag) and the pixel `gridData`; identical configurations **stack** via a deterministic key instead of duplicating.

Keyboard shortcuts — `B` pencil · `E` eraser · `G` fill · `L` line · `R` rect · `O` oval · `I` pick · `1` / `2` / `3` brush size · `M` mirror · `#` grid · `⌘Z` / `⌘⇧Z` undo/redo.

15.17 Digital rain — site-wide & light-theme-correct

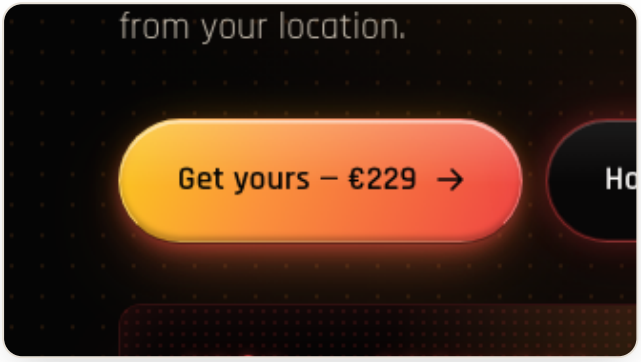
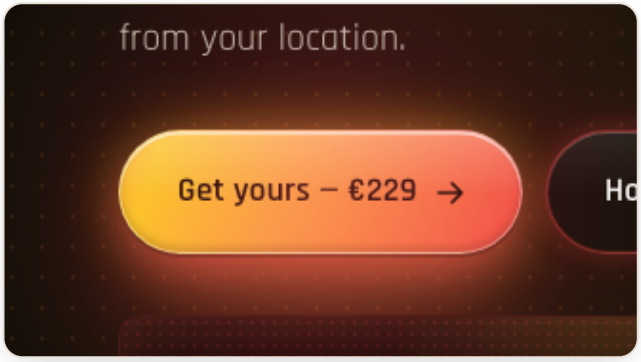
File: `frontend/src/components/DigitalRain.astro`, rendered in `Layout.astro` on **every** page.

Dark theme	Light theme
	

- A full-viewport Matrix-style canvas (katakana + hex + symbols) sits behind all content (`z-index: -2`) with per-column drop speeds, fading trails, an accent-coloured trail, a bright head, and a glow. It runs at 60 fps, **pauses while the tab is hidden**, defers start to `requestIdleCallback` , disables itself under `prefers-reduced-motion` , and persists across client navigations (`transition:persist`).
- **Light-theme fix.** The `[data-theme="light"]` override had never actually applied: Astro scoped the `[data-theme="light"]` ancestor with this component's own scope id (which lives on `<html>` , not the canvas), so the selector could never match and light mode silently fell back to the base `mix-blend-mode: screen` — which renders to white on a near-white background, i.e. invisible. Switching to `:global([data-theme="light"])` (the pattern the other theme-aware components already use) restores the override — `multiply` blend at `opacity: 0.25` , no centre mask — so the rain is visible on **every** page in light mode (verified on `/` , `/about` , `/firmware` , `/studio` , `/compare`).

15.18 CTA button system — 3D depth + living glow

File: `frontend/src/styles/global.css` (`.btn`).

Resting	Hover (lift + glow)
	

- **Living brand gradient** — red → amber → glow drifts slowly side-to-side (7 s) for a warm LED shimmer.
- **3D pill** — a bright rounded top highlight plus an inner bottom shade give the cap its curvature, and a crisp 1px ledge sits it on the surface.
- **Glow halo** — warm accent / amber / glow shadow layers make the button float and emit light; on **hover** it lifts 3 px and the halo intensifies; on **press** the curvature inverts so the cap dips toward the surface.
- **Accessible focus ring** — a keyboard-only `:focus-visible` brand halo with a gap so it reads on any surface.

- Ghost / secondary buttons and `prefers-reduced-motion` opt out of the shimmer.

---

16. Build, Docker & deployment

Multi-stage Dockerfile (`Dockerfile`):

```
node:22-alpine ── base ─┬─ deps    : npm ci
                        ├─ builder : COPY . . && npm run build
                        └─ runner  : COPY dist + node_modules + backend
                                     CMD ["node", "backend/server.mjs"]
```

The runner image contains **only** the built `frontend/dist` , `node_modules` , `package.json` and the `backend/` Express code — no source, no caches.

docker-compose (`docker-compose.yml`):

```
services:
  web   : build → image dotclock-web:latest
          port 4321:4321
          mounts ./uploads → /app/uploads   (live user avatars — never wiped on deploy)
          tmpfs /tmp:64M
          security_opt: no-new-privileges
          init: true
          extra_hosts: host.docker.internal:host-gateway   (for the Postfix mail relay)
          healthcheck: GET /  (every 30s)
          depends_on: mongo (condition: service_healthy)
  mongo : image mongo:6
          NOT published to the host — reachable only on the `dotclock`
          network at mongo:27017 (internal-only by design; nothing on the
          host or internet can connect to it)
          volume mongodb_data   (data persists across container recreates)
          healthcheck: mongosh admin.ping
```

Both services pull common hardening from a YAML anchor `&secure` : `init: true` , `no-new-privileges` , `restart: unless-stopped` . The app reaches Mongo over the compose network (`MONGO_URI` / `MONGODB_URI = mongodb://mongo:27017/dotclock`) — **not** a host IP. (A previous revision removed the `mongo` service and pointed the app at a host Mongo that didn't exist, which silently dropped the whole stack into mock mode; see §11.)

Starting the stack:

```
docker compose up -d
open http://localhost:4321
```

Deploying to the VPS (`root@abiot.it:/opt/dotclock-site` , fronted by nginx at `https://website.abiot.it` — see `DEPLOYMENT_NOTES.md`):

- The production app runs directly on the host under `systemd` as `dotclock-site` ; nginx proxies to `127.0.0.1:4321` .
- The rsync deploy still **excludes** `.env` **and** `uploads/` . `.env` is managed by hand on the server; `uploads/` contains live user files.
- After any `.env` change or rebuild, restart the service with `systemctl restart dotclock-site` .

Hosting recommendations:

- **Self-hosted VPS** (current setup): nginx terminates TLS, forwards to `:4321`. `app.set("trust proxy", 1)` is on so `req.ip` is the real client.
- **Railway / Render / Fly.io** — works as-is; just set env vars.
- **Vercel / Netlify** — swap `@astrojs/node` for the matching Astro adapter and detach the Express layer (move `/api/users`, `/api/orders` etc. to serverless functions).

---

17. Environment variables

See `.env.example` for the canonical list. Summary:

Var	Required for	Example / default
`PORT`	server	`4321`
`HOST`	server	`0.0.0.0`
`NODE_ENV`	server	`production` (enables CSP)
`SITE_URL`	emails (links)	`https://dotclock.com`
`PUBLIC_SITE_URL`	Astro build (sitemap, canonicals)	same as above
`CORS_ORIGINS`	Express CORS	comma-separated origins
`MONGO_URI`	Express (Mongoose)	`mongodb://127.0.0.1:27017/dotclock`
`MONGODB_URI`	Astro API (native)	typically same as `MONGO_URI`
`MONGODB_DB`	Astro API	`dotclock`
`JWT_SECRET`	auth	64+ random bytes
`JWT_EXPIRES_IN`	auth	`30d`
`STRIPE_SECRET_KEY`	payments	`sk_live_…` / `sk_test_…`
`STRIPE_WEBHOOK_SECRET`	payments	`whsec_…`
`SMTP_HOST/PORT/SECURE/USER/PASS/FROM`	email	Mailtrap for dev
`BATCH_TARGET` **(new)**	inventory	`100`
`BATCH_STARTED_AT` **(new)**	inventory	ISO date — anything paid after this counts in the batch
`BATCH_SHIP_DAYS` **(new)**	inventory	`3`

18. Local development workflow

```
# Install deps (root package.json holds everything)
npm install

# Run MongoDB (Docker is easiest)
docker run -d --name dotclock-mongo -p 27017:27017 mongo:6

# Copy + edit env
cp .env.example .env
# Set: MONGO_URI, MONGODB_URI, JWT_SECRET, SMTP_*

# Create the waitlist indexes (one time)
npx tsx frontend/src/lib/setup-db.ts

# Run Astro dev server (no Express; uses Vite proxies for /api/users etc.)
npm run dev               # http://localhost:4321

# Run Astro + the API server together for full-stack iteration
npm run dev:full          # ASTRO + API in parallel
```

Type-check: `npm run typecheck` (runs `astro check`).

Production locally: `npm run preview` builds + starts the Express server, mirroring Docker.

19. Security posture

Cookies

- `dc-token` (JWT) — HTTP-only, `SameSite=Lax`, `Secure` in prod.
- `dc-user-*` identity mirrors — JS-readable (UI only), not trusted by the API.

HTTP headers (helmet)

- HSTS: `max-age=63072000; includeSubDomains; preload`.
- `Frame-ancestors: 'none'`, `X-Frame-Options: DENY`.
- `X-Content-Type-Options: nosniff`.
- `Referrer-Policy: strict-origin-when-cross-origin`.
- `Permissions-Policy`: blocks camera, mic, geo, USB, sensors; allows `payment=(self)` for Stripe.
- Custom `Server: DotClock` overrides the Express default.
- CSP (prod only) — `default-src 'self'`, allows inline scripts/styles (Astro inlines small ones), Stripe + cdnfonts in `frame-src` / `style-src`.

Rate limits

- Global `/api`: 300 req / 15 min / IP.
- `POST /api/users/login`: 30 / 10 min / IP, `skipSuccessfulRequests: true`.
- `POST /api/users/forgot-password|reset-password`: 3 / hr / IP.
- `POST /api/waitlist`: 5 / 60 s / IP (in-memory per Node process).

Auth

- `bcrypt` cost **12** for password hashing.
- JWT signed `HS256` with `JWT_SECRET`.
- All write endpoints require `protect`; admin endpoints additionally require `admin`.

Trust proxy — `app.set("trust proxy", 1)` because the prod deployment sits behind one reverse-proxy hop. Without this, `express-rate-limit` would rate-limit on the proxy's loopback address instead of the real client IP.

Body limits — 10 MB for JSON / URL-encoded (allows avatar upload metadata; Multer caps the actual file separately).

20. Operations playbook

Exporting the waitlist

Mongo isn't published to the host, so run the tools **inside** the container:

```
docker compose exec mongo mongoexport \
            --uri="mongodb://localhost:27017/dotclock" \
            --collection=waitlist \
            --type=csv --fields=email,createdAt,source \
            --out=/tmp/waitlist.csv
docker cp dotclock-site-mongo-1:/tmp/waitlist.csv ./waitlist.csv
```

Filter to back-in-stock subscribers only: add `--query='{"source":"restock"}'`.

Promoting a user to admin

```
node backend/scripts/makeAdmin.js alice@example.com
```

Cutting a new product batch

When you start a fresh batch, set:

```
BATCH_TARGET=120
BATCH_STARTED_AT=2026-06-01T00:00:00Z
```

…and restart the container. The `/api/inventory` counter resets because it only counts paid orders newer than `BATCH_STARTED_AT`.

Looking up a customer order

Admin nav → `/admin/orders` → search by order ID or filter by status.

Rotating Stripe keys

Update `STRIPE_SECRET_KEY` + `STRIPE_WEBHOOK_SECRET` in env and restart. Stripe's old keys remain valid until revoked in dashboard.

Backing up MongoDB

The compose volume is named `mongodb_data` (under the `dotclock-site` project). Standard `mongodump`:

```
docker exec dotclock-site-mongo-1 mongodump --archive=/data/db/backup.archive
docker cp dotclock-site-mongo-1:/data/db/backup.archive ./backups/$(date +%F).archive
```

Watching logs

```
journalctl -u dotclock-site -f
journalctl -u mongod -f
```

Diagnosing "verification / waitlist email not received"

1. **Is the DB up?** `curl -s localhost:4321/api/waitlist/count` → `{"configured":true,…}`. A `total:999` (or registration returning a `mock-user-id`) means Mongo is unreachable, so registration sends **no** mail at all (see §11). Check `systemctl status mongod` and `journalctl -u mongod -n 100 --no-pager`.
2. **Does the running service have the current env?** `systemctl show dotclock-site --property=Environment --no-pager`. If the process still has stale config, restart it with `systemctl restart dotclock-site` (see §16).
3. **What did the host SMTP do?** `tail -f /var/log/mail.log`. Healthy: `DKIM-Signature field added (s=dotclock, d=abiot.it)` + `status=sent (250 …)`. A `from=<…@dotclock.local>` sender or a `550-5.7.26` bounce means `SMTP_FROM` is not an `@abiot.it` address (see §11).

---

21. Known limitations / future work

- **PayPal capture path is wired but not surfaced** in the checkout UI. Enable by adding the button to `checkout.astro` and calling `paypal/create` + `paypal/capture`.
- **Waitlist double-opt-in** — the `confirmed` field exists but no verification email is sent for waitlist signups (only product accounts).
- **Inventory is per-instance for rate limit + per-DB for counts.** Multi-instance deployments should move the in-memory rate-limit map to Redis or Mongo TTL.
- **Currency rates are hand-tuned** (memorable price points, not live FX). Switch to a daily-refreshed table if you need accuracy.
- **No /journal or content collection** yet — Astro content collections are ready to use when long-form content lands.
- **Reviews are static** — the `productController` accepts reviews, but the homepage `Reviews.astro` shows hand-curated copy.
- **Sentry/Plausible/etc. analytics** — not wired. Add in `Layout.astro` head when needed.

22. Glossary

- **LCP** — Largest Contentful Paint, Core Web Vitals metric.
- **CLS** — Cumulative Layout Shift, Core Web Vitals metric.
- **CSP** — Content Security Policy.
- **HSTS** — HTTP Strict Transport Security.
- **ODM** — Object Document Mapper (Mongoose).
- **PWA** — Progressive Web App; `manifest.webmanifest` makes the site installable.
- **SSR** — Server-Side Rendering.
- **SWR** — `stale-while-revalidate` Cache-Control directive: serve cached, refresh in background.
- **JSON-LD** — JSON Linked Data, the structured-data format Google prefers.
- **hreflang** — HTML `<link rel="alternate" hreflang="…">` telling search engines about locale variants.
- **OG** — Open Graph protocol (Facebook's social-share meta).

End of documentation.