# WattSun — Single Source of Truth

## Quick Navigation

1. Project Overview & Structure  
2. Repository Structure & Ops Guide  
3. Updated Structure & Ops (Latest Changes + Current Phase Plan)  
 - Immediate Task: Phase 6.4 — Customer Reflection  
 - Next Phase: 6.5 Hardening  
4. API Contracts & Domain Ownership  
5. Migration Ledger & Backend Refactor Plan  
6. Objectives, Achievements & Roadmap  
 - Pending Tasks  
 - Future Features

## Project Overview & Structure

# WattSun Solar – Project README

## Overview

WattSun Solar is an e-commerce platform for selling solar energy kits and related products.

The system integrates \*\*frontend\*\* (HTML/CSS/JS) and \*\*backend\*\* (Node.js/Express) to handle:

- Product display from database

- Shopping cart and checkout

- Order tracking for customers

- Admin panel for managing orders, users, and inventory

- Financing calculators

- User authentication (signup/login/password reset)

## Technology Stack

- \*\*Frontend:\*\* HTML5, CSS3, JavaScript (Vanilla)

- \*\*Backend:\*\* Node.js, Express.js

- \*\*Database:\*\* SQLite3 (with `inventory.db` for products, `users.db` for users)

- \*\*Hosting:\*\* Synology NAS (HTTP/HTTPS), Cloudflare Tunnel

- \*\*Version Control:\*\* Git (GitHub with SSH authentication)

- \*\*Deployment:\*\* Manual + Git pull to NAS, with auto-start backend script

... (truncated for brevity in this code snippet) ...

## Repository Structure & Ops Guide

# WattSun Monorepo — Structure & Ops Guide (Aug 2025)

This document captures the unified \*\*wattsun\*\* repository layout and the day‑to‑day operations we set up today (NAS runtime, Cloudflared, update flow, logs, and backups). It’s meant to be copy‑paste friendly for future you.

---

## 1) High‑level overview

- \*\*Single repo:\*\* `WattSunK/wattsun` (PC and NAS both sync to this).

- \*\*Paths:\*\*

- \*\*PC (Windows):\*\* `C:\Users\Steve\Documents\wattsun`

- \*\*NAS (Synology):\*\* `/volume1/web/wattsun`

- \*\*Runtime model:\*\* Node app on NAS (port \*\*3001\*\*), front‑end static pages served by the same Node server under `/public/`.

- \*\*Cloudflared:\*\* exposes the NAS service publicly as `https://api.wattsun.co.ke` → `127.0.0.1:3001` on the NAS.

- \*\*Pull‑only NAS:\*\* NAS can \*\*pull\*\* from GitHub but cannot push (push URL set to `DISABLED`). All changes flow \*\*PC → GitHub → NAS\*\*.

---

## 2) Repository layout

```text

wattsun/

├─ server.js

├─ package.json

├─ package-lock.json

├─ .gitignore

├─ .gitattributes

├─ .backend\_checksum

├─ README.md # general project readme (this doc is a companion)

├─ COPY\_INSTRUCTIONS.md

├─ STRUCTURE\_CHECKLIST.md

├─ docs/

│ ├─ roadmap.md

│ ├─ migration-ledger.md

│ ├─ adr/

│ └─ runbooks/

├─ infra/

│ └─ cloudflared/

│ ├─ README.md

│ └─ config.sample.yml # example tunnel config

├─ public/ # \*\*frontend\*\* (site + admin)

│ ├─ index.html

│ ├─ status.html # status page with CF + API checks

│ ├─ admin/

│ ├─ images/ # consolidated images

│ ├─ js/

│ ├─ css/

│ └─ …

├─ routes/ # backend API endpoints (Express)

│ ├─ login.js

│ ├─ signup.js

│ ├─ reset.js

│ ├─ items.js

│ ├─ categories.js

│ ├─ checkout.js

│ ├─ myorders.js

│ ├─ orders.js

│ └─ track.js

├─ scripts/ # \*\*ops helpers\*\* (NAS + maintenance)

│ ├─ start\_nas.sh

│ ├─ stop\_nas.sh

│ ├─ restart\_nas.sh

│ ├─ git\_pull\_update.sh # safe auto-update + restart + health-check

│ ├─ standardize\_dbs.sh # one-off: migrate legacy DB files into data/dev

│ ├─ backup\_sqlite.sh # snapshot data/dev/\*.dev.db + orders.dev.json

│ ├─ start\_cloudflared.sh

│ ├─ db\_password\_pid\_alert.js

│ └─ watch-password-alerts.js

├─ data/ # \*\*standardized data root\*\*

│ └─ dev/

│ ├─ wattsun.dev.db # users DB (canonical file)

│ ├─ inventory.dev.db # inventory DB (items/categories/messages)

│ ├─ users.dev.db # (migrated legacy)

│ └─ orders.dev.json # orders store (transitional)

├─ user-setup/

│ └─ users.db -> ../data/dev/wattsun.dev.db # \*\*symlink\*\* for compatibility

├─ inventory.db -> data/dev/inventory.dev.db # \*\*symlink\*\* (compatibility)

├─ data/users.db -> data/dev/users.dev.db # \*\*symlink\*\* (compatibility)

├─ orders.json -> data/dev/orders.dev.json # \*\*symlink\*\* (compatibility)

├─ logs/

│ ├─ app.out

│ ├─ app.err

│ ├─ update.log

│ ├─ cloudflared.out

│ └─ cloudflared.err

└─ run/

├─ app.pid

├─ cloudflared.pid

└─ last\_sha # last deployed commit

```

### Notes

- Those \*\*symlinks\*\* keep older code working while we gradually move all paths to env‑driven configs. You can safely keep them for now.

- The \*\*status page\*\* (`public/status.html`) shows green/yellow/red for API and Cloudflare tunnel and auto-refreshes every 30s.

---

## 3) Environment (.env)

Create/maintain on NAS at `/volume1/web/wattsun/.env`:

```ini

PORT=3001

# Standardized paths (optional for now – code still works with symlinks)

# DB\_PATH=./data/dev/wattsun.dev.db

# DB\_PATH\_USERS=./data/dev/wattsun.dev.db

# Logging

LOG\_LEVEL=info

# Email (optional; required to actually send contact emails)

# SMTP\_HOST=

# SMTP\_PORT=587

# SMTP\_USER=

# SMTP\_PASS=

# EMAIL\_FROM="WattSun <no-reply@wattsun.co.ke>"

```

> If SMTP is not set, the contact route should still \*\*save\*\* messages; we can enable graceful email‑send fallback.

---

## 4) Start/stop & updates (NAS)

From `/volume1/web/wattsun`:

```bash

# Start/stop/restart the backend cleanly

scripts/start\_nas.sh

scripts/stop\_nas.sh

scripts/restart\_nas.sh

# Pull latest from GitHub, install deps if needed, restart, health-check, log

scripts/git\_pull\_update.sh

# One-off: migrate legacy \*.db to data/dev + create symlinks (already done)

scripts/standardize\_dbs.sh

```

### Health checks

```bash

curl -s -o /dev/null -w "%{http\_code}\n" http://127.0.0.1:3001/api/health # 200 expected

# via Cloudflare (public)

curl -s -o /dev/null -w "%{http\_code}\n" https://api.wattsun.co.ke/api/health

```

### Logs

```bash

tail -n 80 logs/app.out

tail -n 80 logs/app.err

tail -n 120 logs/update.log

```

---

## 5) Cloudflared

- Sample config: `infra/cloudflared/config.sample.yml`

- Runtime via helper:

```bash

scripts/start\_cloudflared.sh # writes logs/cloudflared.\* and run/cloudflared.pid

```

- Verify Cloudflare edge:

```bash

curl -I https://api.wattsun.co.ke/api/health | egrep -i 'HTTP/|server|cf-ray|cf-cache-status'

curl -s https://api.wattsun.co.ke/cdn-cgi/trace | egrep 'h=|colo=|warp='

```

- Status page (`public/status.html`) checks both API and Cloudflare; you can open `https://api.wattsun.co.ke/status.html`.

---

## 6) Git workflow (safe)

- \*\*PC → GitHub → NAS\*\* only.

- NAS \*\*cannot push\*\*:

```bash

cd /volume1/web/wattsun

git remote -v

# origin git@github.com:WattSunK/wattsun.git (fetch)

# origin DISABLED (push)

```

- Typical flow:

1. Develop on PC → `git commit` → `git push` to `main`.

2. On NAS: run `scripts/git\_pull\_update.sh` (manually or via scheduled task).

3. Check `logs/update.log` and `run/last\_sha` to confirm deployed commit.

- Tags: we created \*\*`v0.2.0`\*\* after the initial unification and scripts.

---

## 7) Backups

- Snapshot all important runtime files:

```bash

scripts/backup\_sqlite.sh

# Saves to: /volume1/backups/wattsun/YYYY-MM-DD\_HHMMSS/

# Keeps last 30 snapshots

```

---

## 8) Common ops recipes

```bash

# Kill the currently running app by PID file (if necessary)

[ -f run/app.pid ] && kill "$(cat run/app.pid)" || true

# Verify which process owns port 3001

netstat -tlnp | grep :3001

# DB quick look

sqlite3 data/dev/wattsun.dev.db ".tables"

sqlite3 data/dev/inventory.dev.db ".tables"

# Orders debug (JSON store)

node -e "const a=require('fs').readFileSync('data/dev/orders.dev.json','utf8'); const j=JSON.parse(a); console.log('orders:', j.length)"

```

---

## 9) Known improvements (next steps)

- \*\*Orders → SQLite\*\*: migrate from `orders.dev.json` to tables (`orders`, `order\_items`, `status\_history`) with a migration script.

- \*\*Env‑driven DB paths\*\* everywhere (phase‑out symlinks): add `DB\_PATH\_USERS`, `DB\_PATH\_INVENTORY`, etc., in `.env` and use them in routes.

- \*\*Auth/UI parity through Cloudflare\*\*: ensure login UX handles API responses correctly (we verified server returns `success: true`).

- \*\*SMTP graceful fallback\*\* in contact route (already outlined) and `/api/admin/smtp-test` endpoint for one‑click verification.

- \*\*Staging\*\*: later, add `staging.wattsun.co.ke` mapping to a second PORT, if needed.

---

## 10) Quick rollback

```bash

cd /volume1/web/wattsun

scripts/stop\_nas.sh

git checkout <KNOWN\_GOOD\_SHA\_OR\_TAG>

scripts/start\_nas.sh

```

> `run/last\_sha` shows the last deployed commit SHA written by the update script.

---

## 11) Support crib notes

- \*\*Logs:\*\* `/volume1/web/wattsun/logs/{app.out,app.err,update.log,cloudflared.\*}`

- \*\*Status page:\*\* `/public/status.html`

- \*\*Health endpoint:\*\* `/api/health`

- \*\*NAS cannot push:\*\* ensures production changes only come from GitHub.

---

\_This README reflects the repo at the time of the reorg and the scripts/config we deployed together on \*\*Aug 11, 2025 (Europe/Paris)\*\*.\_

## 12) Admin Dashboard — Strategy & Final Aim

This section defines how the Admin area fits into the monorepo, what we’re building now, and the target end-state.

### 12.1 Goals (what “done” looks like)

- A clean, stable Admin dashboard with:

- \*\*Profile, Home, Logout, Search\*\* (core shell)

- \*\*Orders\*\* (list, search, status filter, pagination, view, edit status/driver/notes)

- \*\*Users, Items, Dispatch, Settings\*\* (read-only first, then targeted edits)

- Changes saved by Admin \*\*reflect in customer views\*\* immediately (focus or lightweight ping).

- Keep \*\*plain HTML + modular JS\*\*; no SPA framework right now.

- Be minimally invasive to existing pages/routes; changes are additive and reversible.

### 12.2 Scope boundaries (now vs later)

- \*\*Now:\*\* Admin shell restore; Orders list; Orders edit (status, driver, notes); driver directory.

- \*\*Soon:\*\* Merge admin updates into customer views (`GET /api/orders`), add date filter, validation, and smoke tests.

- \*\*Later:\*\* Broaden edits (Users/Items/Dispatch), full visual pass (optional `admin.v2.css` toggle).

### 12.3 Architecture & data model

- \*\*DB single entry-point:\*\* `inventory.db` is the canonical SQLite file for Admin too.

- \*\*Overlay table for Admin changes:\*\* `admin\_order\_meta`

```sql

CREATE TABLE IF NOT EXISTS admin\_order\_meta (

order\_id TEXT PRIMARY KEY,

status TEXT NOT NULL,

driver\_id INTEGER,

notes TEXT,

updated\_at TEXT NOT NULL DEFAULT (datetime('now'))

);

```

Rationale: we can persist Admin changes \*\*without rewriting\*\* any legacy orders store; we merge on read later.

### 12.4 API surface (admin)

- `GET /api/admin/users?type=Driver` → list Drivers for the dropdown.

- `PATCH /api/admin/orders/:id` → update `status`, `driverId`, `notes`; writes to `admin\_order\_meta`.

- Existing read endpoints (e.g., `GET /api/orders`) remain unchanged initially; later we merge overlay so customers see updated status/driver automatically.

### 12.5 Frontend structure (admin)

- Location: `/public` (served by the same Node app).

- Files:

- `public/dashboard.html` — admin shell (sidebar + content slot)

- `public/partials/orders.html` — Orders list partial (IDs: `#ordersTable`, `#ordersTbody`, `#ordersSearch`, `#ordersStatus`, `#ordersPager`)

- `public/partials/orders-modal.html` — Edit modal (status/driver/notes)

- `public/admin/js/data-adapter.js` — fetch + normalization layer

- `public/admin/js/dashboard.js` — tiny partial loader (relative paths)

- `public/admin/js/orders-controller.js` — Orders list (search/filter/pager/view)

- `public/admin/js/orders-edit.js` — Edit modal + PATCH + inline row refresh

- `public/admin/admin.css` (current look) and optional `admin.v2.css` (style pass, toggled with `?css=v2`)

\*\*Script order\*\* in `dashboard.html` (use cache-busters consistently):

```html

<script src="./admin/js/data-adapter.js?v=3"></script>

<script src="./admin/js/orders-controller.js?v=3" defer></script>

<script src="./admin/js/orders-edit.js?v=3" defer></script>

<script src="./admin/js/dashboard.js?v=3" defer></script>

```

### 12.6 Security & permissions

- Gate all `/api/admin/\*` routes behind an Admin check:

```js

function requireAdmin(req, res, next) {

const u = req.session?.user;

if (!u || (u.type !== "Admin" && u.role !== "Admin")) {

return res.status(403).json({ success:false, error:"Forbidden" });

}

next();

}

app.use("/api/admin", requireAdmin);

```

- Backend validation:

- `status ∈ {Pending, Processing, Delivered, Cancelled}`

- `driverId` must reference a `users` row where `type='Driver'`.

### 12.7 Ops: NAS runtime, logs, updates

- Admin shares the same Node process and logging as the rest of the app.

- Use existing restart/update flow; commit → deploy to NAS → restart node service.

- Health checks unchanged; add a simple `GET /api/health` if not present.

### 12.8 Milestones (Phase 6)

- \*\*6.0 Admin Core Restore\*\* — role gate, header/sidebar, Home + Logout, partial loader; no console errors.

- \*\*6.1 Orders List\*\* — table with search, status filter, pagination (10/pg), mutation-safe pager.

- \*\*6.2 View/Edit Order\*\* — modal: status/driver/notes; save updates row inline; localStorage ping.

- \*\*6.3 Backend Wiring\*\* — `GET /api/admin/users?type=Driver`; `PATCH /api/admin/orders/:id`; overlay table created if missing; mounted in `server.js`.

- \*\*6.4 Customer Reflection\*\* — merge `admin\_order\_meta` on read (or re-fetch trigger) so Track/My Orders show updates.

- \*\*6.5 Hardening\*\* — validation messages, currency formatting edge cases, empty states, permission checks, and smoke tests.

### 12.9 Rollback & safety

- All Admin additions are \*\*additive\*\*; rollback by:

- Removing route mounts (`/api/admin/\*`), and

- Removing the three admin files (`orders-edit.js`, `orders-modal.html`, overlay router).

- No schema change to legacy orders; `admin\_order\_meta` can remain (harmless) or be dropped later.

### 12.10 SPA decision

- No SPA for now. Plain HTML + modular JS keeps footprint small, works well on NAS, and aligns with current repo. Re-evaluate only if we outgrow this.

## Updated Structure & Ops (Latest Changes + Current Phase Plan)

# WattSun — Structure & Ops (Updated 2025-08-13)

This document captures how the app is structured on your Synology NAS, the current runtime conventions, database layout, diagnostic endpoints, and the \*\*Phase 6\*\* admin roadmap status. It also prescribes the next task (\*\*6.4 Customer Reflection\*\*) and how we’ll implement and test it.

---

## 1) High‑level layout (repo root)

```

wattsun/

├─ public/ # static site (index.html, assets, admin dashboard)

│ ├─ admin/

│ │ ├─ css/…

│ │ └─ js/

│ │ ├─ dashboard.js

│ │ ├─ data-adapter.js

│ │ ├─ orders-controller.js

│ │ └─ orders-edit.js

│ └─ partials/

│ └─ orders.html

├─ routes/ # Express routes (auth, admin, items, etc)

│ ├─ login.js # POST /api/login (JSON or urlencoded)

│ ├─ signup.js # POST /api/signup (JSON or urlencoded)

│ ├─ reset.js # POST /api/reset-request + /api/reset-confirm (+ legacy /api/reset)

│ ├─ admin-orders.js # PATCH /api/admin/orders/:id

│ ├─ admin-users.js # GET /api/admin/users?type=Driver

│ └─ admin-diagnostics.js # GET /api/admin/\_diag/\*

├─ data/

│ └─ dev/

│ ├─ wattsun.dev.db # \*\*canonical Users DB\*\* (auth/users + admin overlay table)

│ └─ inventory.dev.db # Items/categories/messages (read-mostly)

├─ user-setup/

│ └─ users.db -> ../data/dev/wattsun.dev.db # \*\*symlink\*\* for old code paths

├─ scripts/

│ ├─ start\_nas.sh # exports DB env and launches server.js

│ ├─ stop\_nas.sh # stops the node process safely

│ └─ status\_nas.sh # optional status helper

├─ logs/

│ ├─ app.out # server log (nohup target)

│ └─ app.3010.out # example alt-port run (dev only)

├─ server.js # Express app

└─ docs/

└─ (this file)

```

---

## 2) Runtime & process

- \*\*Node\*\*: v20 (Synology package).

- \*\*Default port\*\*: `3001`. You can override with `PORT`, but we keep `3001` for NAS tasks.

- \*\*Start/Stop\*\* (Task Scheduler uses these):

```bash

# stop

./scripts/stop\_nas.sh

# start (pins DBs via env)

./scripts/start\_nas.sh

# sanity

ps -ef | grep -E "node .\*server\.js|node server\.js" | grep -v grep

curl -fsS http://127.0.0.1:3001/api/admin/\_diag/ping

```

`start\_nas.sh` exports:

- `SQLITE\_DB=$ROOT/data/dev/wattsun.dev.db`

- `DB\_PATH\_USERS=$ROOT/data/dev/wattsun.dev.db`

These make the \*\*Users DB\*\* the single source of truth for auth and the admin overlay.

---

## 3) Databases

### 3.1 Canonical files

- \*\*Users (auth + overlay)\*\* → `data/dev/wattsun.dev.db`

- \*\*Inventory (items/categories/messages)\*\* → `data/dev/inventory.dev.db`

Legacy paths are bridged by a symlink: `user-setup/users.db -> data/dev/wattsun.dev.db`.

### 3.2 Admin overlay table (lives in Users DB)

```

CREATE TABLE IF NOT EXISTS admin\_order\_meta (

order\_id TEXT PRIMARY KEY,

status TEXT NOT NULL,

driver\_id INTEGER,

notes TEXT,

updated\_at TEXT NOT NULL DEFAULT CURRENT\_TIMESTAMP

);

```

> We migrated any previous rows from `inventory.dev.db` and now \*\*write only\*\* to `wattsun.dev.db`.

### 3.3 Quick DB checks

```bash

sqlite3 data/dev/wattsun.dev.db ".tables"

sqlite3 data/dev/wattsun.dev.db "PRAGMA table\_info(users);"

sqlite3 data/dev/wattsun.dev.db "SELECT COUNT(\*) FROM admin\_order\_meta;"

```

---

## 4) Diagnostic endpoints (admin)

```bash

# server liveness

GET /api/admin/\_diag/ping

# → {"success":true,"time":"..."}

# DB wiring

GET /api/admin/\_diag/db

# → { success, sqliteVersion, overlayDbPath, envDbPath, users:{total?,drivers?}, pid }

```

These are already mounted in `server.js` as:

```js

app.use("/api/admin/\_diag", require("./routes/admin-diagnostics"));

```

---

## 5) Auth endpoints (browser + curl)

All accept \*\*JSON\*\* and \*\*URL‑encoded form\*\* payloads.

- \*\*POST\*\* `/api/login`

- body: `{ email, password }`

- 200: `{ ok:true, user:{...} }` • 401: invalid credentials • 500: DB error

- \*\*POST\*\* `/api/signup` (and alias `/api/` when mounted at `/api/signup`)

- body: `{ name, email, phone?, password }`

- 200: `{ ok:true, user:{ id, name, email, phone, type:"User", status:"Active" } }`

- 409: `Email already registered`

- 500: DB error (see logs)

- \*\*POST\*\* `/api/reset-request` (aliases: `/api/reset/request`, legacy `/api/reset` without token)

- body: `{ email }`

- 200: `{ ok:true, token, expires }` \*(token included for dev)\*

- \*\*POST\*\* `/api/reset-confirm` (aliases: `/api/reset/confirm`, legacy `/api/reset` with token)

- body: `{ token, password }`

- 200: `{ ok:true }`

\*\*Troubleshooting signup 500 (browser-only):\*\*

1) Tail logs to see `[signup]` details:

```bash

tail -n 120 logs/app.out | grep -i '\[signup\]\|\[login\]\|\[reset\]'

```

2) If it’s a duplicate email you’ll get \*\*409\*\*, not 500.

3) Ensure `server.js` mounts the router once and before any catch‑all:

```js

app.use("/api/signup", require("./routes/signup"));

```

4) Body parser is applied by the route (JSON + urlencoded), so you don’t need a global change.

---

## 6) Phase 6 — Admin Dashboard

### 6.0 Core Restore ✅

- Role gate, header/sidebar, Home/Logout, partial loader; no console errors.

### 6.1 Orders List ✅ (baseline)

- Table rendering stable with \*\*search\*\*, \*\*status filter\*\*, and \*\*pagination (10/pg)\*\*.

- Network uses `/api/orders?page=&per=&q=&status=`; client filters keep UI responsive.

### 6.2 View / Edit Order ✅ (baseline)

- \*\*View\*\* modal shows canonical order data + overlay (status/driver/notes).

- \*\*Edit\*\* drawer opens; \*\*status + notes\*\* persist through PATCH; \*\*driver\*\* requires a user with `type='Driver'`.

- Create a driver quickly:

```sql

INSERT INTO users (name,email,phone,type,status,created\_at)

VALUES ('Test Driver','driver1@example.com','+254700000001','Driver','Active',CURRENT\_TIMESTAMP);

```

### 6.3 Backend wiring ✅

- \*\*GET\*\* `/api/admin/users?type=Driver` → list of drivers.

- \*\*PATCH\*\* `/api/admin/orders/:id` → writes overlay fields to `admin\_order\_meta` in \*\*wattsun.dev.db\*\*.

- `server.js` pins the DB via env (`DB\_PATH\_USERS` / `SQLITE\_DB`) and exposes `\_diag` endpoints.

### 6.4 Customer Reflection 🔜 (next task)

When admin saves an order, the \*\*customer view\*\* (Track / My Orders) should refresh \*\*immediately\*\* without reloading.

\*\*Plan\*\*

1) \*\*Emit signal on save\*\* (admin UI, in `orders-edit.js` after successful PATCH):

```js

// flag for other tabs

localStorage.setItem("ordersUpdatedAt", String(Date.now()));

// message for same-tab iframes (defensive)

window.postMessage({ type:"orders-updated" }, "\*");

```

2) \*\*Listen & re-fetch\*\* (customer pages, e.g. `public/js/myorders.js` / `public/js/track.js`):

- on `window.focus` → refetch

- on `window.storage` for key `ordersUpdatedAt` → refetch

- on `window.message` with `{type:"orders-updated"}` → refetch

- optional: fallback polling (30–60s) behind a feature flag

3) \*\*Refetch mechanics\*\*

- Call the \*\*existing\*\* orders endpoint (same shape as manual refresh today).

- Preserve pagination and filter state across refreshes.

4) \*\*Smoke tests\*\*

- Open Customer tab A (My Orders). Open Admin tab B (Orders).

- In B: change status of order X → \*\*A updates within 1s\*\*.

- Repeat with driver & notes; ensure UI merges overlay correctly.

### 6.5 Hardening (after 6.4)

- Input validation (server + client), currency formatting, empty states.

- Permission checks (admin-only routes), rate limiting on auth/reset.

- Add minimal tests for: list paging, edit, reflection flow.

---

## 7) Known gaps / To‑watch

- \*\*Orders totals\*\* show “KES 0” for historical data; server aggregation pending.

- \*\*Favicon\*\* 404 in console (cosmetic).

- \*\*Signup\*\* shows “Database error” in some browser flows while curl succeeds → use the log hint above; we now translate duplicate emails to \*\*409\*\* and accept both JSON and urlencoded bodies.

---

## 8) Quick commands

```bash

# Diagnostics

curl -fsS http://127.0.0.1:3001/api/admin/\_diag/ping

curl -fsS http://127.0.0.1:3001/api/admin/\_diag/db | jq .

# Auth (replace values)

curl -fsS -X POST http://127.0.0.1:3001/api/signup \

-H 'Content-Type: application/json' \

-d '{"name":"demo","email":"demo@example.com","phone":"+254700000777","password":"Passw0rd!"}'

curl -fsS -X POST http://127.0.0.1:3001/api/login \

-H 'Content-Type: application/json' \

-d '{"email":"demo@example.com","password":"Passw0rd!"}'

```

---

## 9) How to run locally on a different port (dev only)

```bash

PORT=3010 SQLITE\_DB=$PWD/data/dev/wattsun.dev.db \

DB\_PATH\_USERS=$PWD/data/dev/wattsun.dev.db \

node server.js

```

---

## 10) Next thread (Step 6.4 kickoff)

\*\*Title:\*\* Phase 6.4 — Customer Reflection (Orders Sync)

\*\*Kickoff message (paste in new chat):\*\*

> Implement Step 6.4. When admin saves an order, customer My Orders / Track Order should refresh immediately without reload.

> \*\*Deliver:\*\*

> 1) After successful PATCH in admin `orders-edit.js`, write `localStorage.ordersUpdatedAt=Date.now()` and `postMessage({type:'orders-updated'})`.

> 2) In customer pages (`myorders.js`, `track.js`), add listeners for `focus`, `storage('ordersUpdatedAt')` and `message('orders-updated')` to refetch orders and re-render, preserving filters/pagination.

> 3) Include minimal smoke tests and copy‑paste code blocks for each file.

## API Contracts & Domain Ownership

# ADR-001: API Contracts & Domain Ownership

\*\*Status:\*\* Proposed • \*\*Date:\*\* 2025-08-11

\*\*Decision:\*\* Standardize endpoint shapes and ownership per domain to reduce regressions and enable gradual refactor without changing URLs.

---

## Principles

- \*\*No breaking changes now.\*\* Existing routes continue to work; this ADR documents the contract.

- \*\*Same-origin frontends.\*\* Customer and Admin call relative `/api/...` endpoints.

- \*\*Consistent shapes.\*\* Monetary values in `priceCents/totalCents/depositCents` (integers).

- \*\*Errors:\*\* `{ success:false, error:{ code, message } }`

- \*\*Success:\*\* `{ success:true, ...payload }`

---

## Auth (Domain: `auth`)

- `POST /api/auth/signup`

\*\*Body:\*\* `{ fullName, email, phone, password }`

\*\*Returns:\*\* `{ success, user:{ id, fullName, email, phone, role }, message }`

\*\*Notes:\*\* Frontend stores session object as `wattsunUser` in localStorage (source of truth for UI).

- `POST /api/auth/login`

\*\*Body:\*\* `{ emailOrPhone, password }`

\*\*Returns:\*\* `{ success, user:{...}, message }`

- `POST /api/auth/reset` (request link/code)

\*\*Body:\*\* `{ emailOrPhone }` → `{ success, message }`

- `POST /api/auth/reset/confirm`

\*\*Body:\*\* `{ tokenOrCode, newPassword }` → `{ success, message }`

---

## Users (Domain: `users`)

- `GET /api/users/me` → `{ success, user:{ id, fullName, email, phone, role, createdAt } }`

- `PUT /api/users/me`

\*\*Body:\*\* editable profile fields (email/phone typically non-editable by user)

→ `{ success, user:{...} }`

- (Later) Audit: password/profile changes appended to `users\_audit`.

---

## Catalog (Domain: `catalog`)

- `GET /api/items`

\*\*Query:\*\* optional filters later

\*\*Returns:\*\* `{ success, items:[ { id, sku, name, description, priceCents, categoryId, image, active } ] }`

- `GET /api/categories`

→ `{ success, categories:[ { id, name, image, active } ] }`

---

## Cart & Checkout (Domain: `cart`)

- `POST /api/cart/checkout`

\*\*Body (example minimal):\*\*

```

{

"items": [ { "productId": "p1", "qty": 2 } ],

"customer": { "fullName", "email", "phone" },

"notes": "optional"

}

```

\*\*Returns:\*\*

```

{

"success": true,

"order": {

"id": "...",

"orderNumber": "...",

"totalCents": 123456,

"depositCents": 12000,

"createdAt": "..."

},

"message": "Order created"

}

```

---

## Orders (Customer scope) (Domain: `orders`)

- `GET /api/orders`

\*\*Query:\*\* `phone`, `page=1`, `per=5`

\*\*Returns:\*\* `{ success, page, per, total, orders:[ { id, orderNumber, status, totalCents, createdAt } ] }`

\*(Note: Customer view overlaps with tracking but may include more details when logged in.)\*

---

## Tracking (Public) (Domain: `tracking`)

- `GET /api/track`

\*\*Query:\*\* `phone` (required), `status` (optional), `page=1`, `per=5`

\*\*Returns:\*\*

```

{

"success": true,

"page": 1,

"per": 5,

"total": 12,

"orders": [

{

"orderNumber": "...",

"status": "Pending",

"createdAt": "2025-07-20T12:00:00Z",

"totalCents": 76500

}

]

}

```

---

## Admin Orders (Admin scope) (Domain: `orders`)

- `GET /api/admin/orders`

\*\*Query (optional):\*\* `q`, `status`, `page=1`, `per=10`, `from`, `to`

\*\*Returns:\*\* `{ success, page, per, total, orders:[ { id, orderNumber, fullName, phone, email, status, totalCents, createdAt } ] }`

- `PUT /api/admin/orders/:id/status`

\*\*Body:\*\* `{ status, note }`

\*\*Returns:\*\* `{ success, order:{ id, orderNumber, status }, history:{ id, status, changedBy, changedAt, note } }`

\*\*Rule:\*\* Append to `order\_status\_history` on every change.

- `PUT /api/admin/orders/:id/assign-driver`

\*\*Body:\*\* `{ driverUserId }` → `{ success, order:{ id, orderNumber, driverUserId } }`

\*\*Note:\*\* `driverUserId` must exist and have role `Driver`.

---

## Notifications (Domain: `notifications`)

- (Internal) `POST /api/internal/notify`

\*\*Body:\*\* `{ channel:"email", template:"order\_status\_changed", to, payload }`

\*\*Returns:\*\* `{ success }`

\*\*Later:\*\* Store send attempts in `notifications` with status.

---

## Status & Enumerations (shared)

- \*\*Order statuses:\*\* `Pending | Confirmed | Dispatched | Delivered | Closed | Cancelled`

- \*\*Roles:\*\* `Customer | Admin | Driver | Installer | Manufacturer` (extendable)

- \*\*Pagination defaults:\*\* Customer/Tracking `per=5`; Admin `per=10`.

---

## Security & AuthZ (summary)

- Admin endpoints require role `Admin`.

- Customer endpoints read-only where unauthenticated (Tracking), richer data when logged in.

- Current frontends use `wattsunUser` in localStorage to drive UI; backend to enforce roles via middleware (implementation timing TBD).

---

## Consequences

- Clear “owners” per domain reduce accidental regressions.

- Numeric money types end price formatting issues.

- A stable contract lets us refactor code behind the same URLs.

---

## Migration Ledger & Backend Refactor Plan

# WattSun Modular Monolith — Migration Ledger

\*\*Status:\*\* Draft • \*\*Date:\*\* 2025-08-11 • \*\*Author:\*\* MK & team

\*\*Goal:\*\* Reorganize backend internally (domains) with \*zero\* disruption to current Cloudflared, NAS, and Git setup.

\*\*Non-goals now:\*\* No staging/prod cut-over yet; no endpoint changes; no code moves for frontends.

---

## Ground Truth (do not change)

- Frontends stay where they are:

- Customer site: `/web/wattsun/public/`

- Admin UI: `/web/wattsun/public/admin/` (path-based under same origin)

- Backend root stays: `/web/wattsun/`

- Cloudflared hostnames/ports (current):

- `api.wattsun.co.ke` → `http://127.0.0.1:3001` (DEV API sandbox)

- \*(reserved)\* `staging.wattsun.co.ke` → `http://127.0.0.1:3000`

- \*(reserved)\* `www.wattsun.co.ke` → `http://127.0.0.1:3000`

- Golden baselines: keep existing zips/tags as rollback points.

---

## Legend

- [ ] Not started • [~] In progress • [x] Done • ⏭️ Planned / blocked by a later phase

---

## Phase A — Create the skeleton (no behavior change)

- [ ] Add backend structure under `/web/wattsun/`:

- [ ] `src/core/` (bootstrap/middleware placeholders)

- [ ] `src/domains/{auth,users,catalog,cart,orders,tracking,notifications}/` (empty READMEs)

- [ ] `src/db/{migrations,seeds}/` (placeholders)

- [ ] `scripts/` (placeholders: `import-orders.md`, `backup.md`, `health.md`)

- [ ] `test/` (placeholder)

- [ ] `docs/adr/` (see ADR-001 below)

- [ ] Record current stable tags/zips in \*\*Appendix A\*\* of this file.

\*\*Owner:\*\* \_\_\_\_ • \*\*Target date:\*\* \_\_\_\_

---

## Phase B — Paper mapping & lifecycles (design only)

- [ ] Map legacy → target (no code yet):

- `server.js` → remains; later imports from `src/core/\*`

- `routes/\*.js` → `src/domains/\*` (see table below)

- `orders.json` → \*\*DB tables\*\* (import later)

- [ ] Define \*\*Order Status Lifecycle\*\* (single source of truth):

`Pending → Confirmed → Dispatched → Delivered → Closed/Cancelled`

- [ ] Approve ADR-001 (API contracts & owners).

\*\*Owner:\*\* \_\_\_\_ • \*\*Target date:\*\* \_\_\_\_

---

## Phase C — Backend internal tidy (no endpoint change)

- [ ] Create empty domain routers/services (no wiring yet).

- [ ] Keep all existing routes live under `routes/`.

- [ ] Document “facade router” approach for a later seamless mount.

\*\*Owner:\*\* \_\_\_\_ • \*\*Target date:\*\* \_\_\_\_

---

## Phase D — Frontend guardrails (no visual change)

- [ ] Confirm all browser calls use \*\*relative\*\* `/api/...` (same-origin ready).

- [ ] Confirm image strategy is followed:

- Site visuals → `/public/images/`

- Product photos → `/public/images/products/`

\*\*Owner:\*\* \_\_\_\_ • \*\*Target date:\*\* \_\_\_\_

---

## Phase E — Data model prep (execute later with staging)

\*(Write only; don’t run yet)\*

- [ ] Schema doc for:

- `orders`, `order\_items`, `order\_status\_history`, `notifications`

- [ ] Import plan \*\*orders.json → DB\*\*:

- [ ] Field mapping table

- [ ] Verification checklist (counts, spot checks)

- [ ] Rollback plan (keep read-only copy of `orders.json`)

- [ ] Decide per-env DB file locations (dev/staging/prod folders).

\*\*Owner:\*\* \_\_\_\_ • \*\*Target date:\*\* \_\_\_\_

---

## Phase F — First real move = \*\*Orders domain\*\* (when ready)

- [ ] Copy logic into `src/domains/orders/` (controller/service/dao structure).

- [ ] Re-export via legacy route so URLs remain unchanged.

- [ ] Append-only `order\_status\_history` on every admin status update.

- [ ] Document manual test steps (admin update → visible in customer Track/My Orders).

\*\*Owner:\*\* \_\_\_\_ • \*\*Target date:\*\* \_\_\_\_

---

## Phase G — Quality gates & ops hygiene (lightweight)

- [ ] Smoke checklist in `docs/`:

- `/api/health` green

- Auth loop (signup→login→/users/me)

- Checkout creates order (dev DB)

- Admin can update status; history row appended

- Track lists up to 5 with pagination

- [ ] Rollback note: `git reset --hard <stable-tag>` + restart service

- [ ] Daily DB backup note (where, how long retained)

\*\*Owner:\*\* \_\_\_\_ • \*\*Target date:\*\* \_\_\_\_

---

## Legacy → Target Mapping Table

| Legacy path | Target domain folder | Notes |

|---|---|---|

| `routes/login.js`, `routes/auth.js` | `src/domains/auth/` | Keep response shape stable (see ADR-001) |

| `routes/users.js` | `src/domains/users/` | `/users/me` profile read/update |

| `routes/items.js` / `routes/catalog.js` | `src/domains/catalog/` | Products/categories; price in cents |

| `routes/cart.js` / checkout bits | `src/domains/cart/` | Server-side totals & deposit |

| `routes/orders.js` | `src/domains/orders/` | Admin ops, status updates, items |

| `routes/track.js` | `src/domains/tracking/` | Public tracking (phone + optional status) |

| (mailer utils) | `src/domains/notifications/` | Email now; SMS later |

---

## Appendix A — Baselines & Rollback

- Frontend stable tag: `stable-frontend` (commit: \_\_\_\_ ; date: \_\_\_\_)

- Backend stable tag: `stable-backend` (commit: \_\_\_\_ ; date: \_\_\_\_)

- Golden zip(s) location: \_\_\_\_

- Quick rollback: `git reset --hard <tag>` → restart → verify smoke checklist.

---

## Objectives, Achievements & Roadmap

# WattSun Solar – Objectives & Roadmap

## Project Objectives

1. \*\*E-commerce for Solar Products\*\*

2. \*\*Customer Order Tracking\*\*

3. \*\*Admin Management\*\*

4. \*\*User Authentication\*\*

5. \*\*Scalable Infrastructure\*\*

6. \*\*Inventory Control\*\*

## Achievements To Date

- Backend & frontend deployed live on NAS

- GitHub SSH setup for both backend and frontend repos

- Dynamic product rendering from `/api/items`

- Cart and checkout integrated with backend

- Orders saved to DB and email confirmations sent

- Fully functional authentication loop (signup, login, reset)

- Admin dashboard restored with working sidebar and partials

- `/api/track` integrated with pagination

- Financing calculators integrated with backend

- Image directory structure standardized

... (rest of roadmap as drafted) ...

# Step 6.8 — Dashboard Completion

Complete the remaining dashboard tabs without regressions. Dependencies: Orders (6.4) frozen as truth. Scope: Dispatch, Settings, MyOrders, Profile. Includes RBAC, API stubs, DB deltas, and test plan.

## Scope & Acceptance

- Dispatch: assign/unassign driver; set status; note; audit trail. Accept: changes persist and reflect in Orders immediately.

- Settings: admin-only key–value configs. Accept: persistence across reloads, RBAC enforced.

- MyOrders: paginated customer orders with filters. Accept: only own orders, consistent currency.

- Profile: edit own fields with validation; password/email changes require confirmation. Accept: persisted.

## Redirect Policy

Single source = URL `?login=1&next=/path` with tab-scoped sessionStorage. Validate `next` as site-relative; remove legacy `checkoutRedirect`.