Text Notifications — Technical Details

This document covers server-side configuration, authentication, API security, cron scheduling, and Strapi integration for the Timebars notification system. It is intended for system administrators and developers.

> **End-user configuration and testing:** See Text Notifications Guide for setting up Pushover, creating notification configurations, and managing thresholds.

Table of Contents

1. Environment Variables
2. Data Sources — How Projects Are Fetched
3. Authentication Flow
4. Route Protection
5. API Route Security
6. Strapi Collections
7. Strapi User Fields Required
8. Automated Cron Job
9. On-Publish Endpoint
10. Manual Test Button — What It Actually Does
11. Low-Level Connectivity Test

Environment Variables

The following env vars must be set on the server:

```
PUSHOVER_APP_TOKEN=     # From pushover.net → Your Applications → API Token
PUSHOVER_USER_KEY=      # From pushover.net → Home dashboard, top section
NOTIFICATION_STRAPI_KEY # Used by the automated endpoint to fetch configs
and project data from Strapi (no user session needed — designed for cron
use)
SYSTEM_ADMIN_EMAIL=     # The email whose active pubset is used to fetch
project data for automated runs
```

> **Note:** `PUSHOVER_API_URL` and `ORTHISURL` env vars are not used. The code hardcodes `https://api.pushover.net/1/messages.json` directly in `pushover.js` and `pushover/route.js`.

Data Sources — How Projects Are Fetched

The automated notification route always fetches from the **`timebars` Strapi collection** (the pubset), filtering to the `SYSTEM_ADMIN_EMAIL`'s active pubset. It reads `tbmdjoined` rows and filters down to:

- `tbType === "Project"`
- `tbMDStatus === "In progress"`

Only those rows enter the evaluation engine. Projects that are not of type Project, or whose status is not exactly `"In progress"`, are silently skipped.

The notification configurations are fetched from the **`notifications`** Strapi collection, each linked to an **order** in the **`orders`** collection. The order carries the product licence (e.g., Agilebars Tier 2, Costbars Tier 1), and the system reads the `product_code` from that order to decide which evaluation logic to apply.

Product code prefix convention

Prefix	Product	Scenarios evaluated
`CB*`	Costbars	5 scenarios
`TB*`	Timebars	3-condition check
`AB*`	Agilebars	2-condition check

Authentication Flow

When a user signs in, NextAuth runs a JWT callback (`app/auth/auth.js`) that performs two things beyond standard login:

1. Stores the Strapi JWT in the session (`session.jwt`) — used for all subsequent Strapi API calls.
2. Makes an extra call to Strapi to fetch two RBAC fields from the user record:
    - `primary_role` — determines admin access
    - `customer_id` — organisation identifier for data scoping

These are then available on `session.user` throughout the app.

Route Protection

Route	Guard
`/admin`	Must be authenticated **AND** `primary_role === 'admin'` or email is the designated admin email
`/admin/notifications`	Must be authenticated only — **no role check**

`/admin/notifications` is reachable by any logged-in user. The main `/admin` page is the harder gate.

API Route Security

Route	Protection
`POST /api/notifications/automated`	System-only — uses `NOTIFICATION_STRAPI_KEY` env var. No user session required. Designed for cron.
`GET/POST /api/notifications/pushover`	Requires `Authorization: Bearer <token>` header
`GET/POST /api/notifications/twilio`	**No auth check** — open if reachable
`POST /api/notifications/on-publish`	Validated via Strapi JWT token

> **Note:** The Twilio route validates input but does not verify the caller's identity. As long as it is only called server-side from `AdminNotificationSender` this is acceptable in practice, but it could be called directly by anyone who knows the URL. Consider adding a bearer token check if the route becomes externally reachable.

Strapi Collections

`orders` — stores purchases, each linked to a product and a user. Filtered by the `owner` field (email address) to scope to each user's licences.

When `/admin/notifications` loads, it calls `getMyOrders(email, jwt)` which hits:

```
GET /api/orders?populate[0]=product&populate[1]=user
    &filters[owner][$eq]={your-email}
```

Fields displayed from each order:

UI label	Strapi field
Order name	`order.name`
Product name / code	`order.product.name` / `order.product.product_code`
Cost	`order.total`
Expiration	`order.expires_on`
Active status	`order.active_status`

`notifications` — stores per-manager notification configurations, each linked back to an order by `order.id`. Multiple notification configs can exist per order (one per manager). Fields cover all thresholds,

channels, quiet hours, cooldown, and daily limit settings.

Strapi User Fields Required

Each user record in the Strapi `users` collection needs two custom fields:

Field	Type	Purpose
`primary_role`	String	Set to `'admin'` to grant `/admin` access
`customer_id`	String / Number	Organisation identifier for data scoping

If `primary_role` is null or missing, the user passes the `/admin/notifications` check (authenticated only) but is redirected away from `/admin` itself.

Automated Cron Job

The automated notification chain runs as a bash cron job on the production server.

Chain of execution

```
cron job (server)
  └─ runs automate_pushover_twillio.sh
       └─ internal guard: exits if weekend or outside 09:00–18:00 server
time
       └─ POST https://www.timebars.com/api/notifications/automated
            └─ fetches all active notification configs (via
NOTIFICATION_STRAPI_KEY)
            └─ fetches all in-progress projects (via
fetchProjectDataFromPubset)
            └─ calls evaluateInProgressProjects() ← scenarios run here
            └─ for each flagged manager:
                 checks timezone / quiet hours / cooldown / daily limit
                 sends Pushover (+ SMS if escalation conditions met)
                 updates last_notification_sent + notification_count_today
in Strapi
```

Script location and schedule

The script lives at `~/docker/tbwwwp/automate_pushover_twillio.sh` and is registered in root's crontab via `sudo crontab -e`.

Recommended schedule — every 4 hours, any day:

```
0 */4 * * * /home/jcox/docker/tbwwwp/automate_pushover_twillio.sh >>
/var/log/ppm-notifications.log 2>&1
```

Previous schedule — hourly, weekdays only:

```
0 6-23 * * 1-5 /home/jcox/docker/tbwwwp/automate_pushover_twillio.sh
```

> **Absolute path required:** `sudo crontab -e` registers in root's crontab, so `~/` expands to `/root/` not `/home/jcox/`. Always use the full path.

Setup checklist

```
# Verify script exists and is executable
ls -la /home/jcox/docker/tbwwwp/automate_pushover_twillio.sh
chmod +x /home/jcox/docker/tbwwwp/automate_pushover_twillio.sh

# Create log file writable by root
sudo touch /var/log/ppm-notifications.log
sudo chmod 644 /var/log/ppm-notifications.log

# Verify cron registration
sudo crontab -l
```

On-Publish Endpoint

`POST /api/notifications/on-publish` triggers an immediate notification check when users publish a dataset from any of the three client apps.

Behaviour differences from automated runs

Behaviour	Automated cron	On-publish
Business hours check	Applied	**Bypassed**
Quiet hours check	Applied	**Bypassed**
Cooldown	Applied	Applied (prevents duplicate sends on rapid re-publish)
Daily limit	Applied	Applied
Authentication	`NOTIFICATION_STRAPI_KEY`	Strapi JWT token

Deduplication

Two-layer deduplication prevents duplicate sends:

1. **In-memory cache** (per user per day) — fast, survives normal runtime restarts
2. **Strapi `last_notification_sent` field** — persistent, survives full server restarts

Client integration

A reference integration snippet is at `app/admin/notifications/tbrunp-integration-snippet.js`. The design is fire-and-forget: notification failures never block or break the publish operation. Missing notification configs or no in-progress projects result in silent skips.

Manual Test Button — What It Actually Does

The **Run Manual Test** button on `/admin/notifications` calls `evaluateInProgressProjects()` **directly in the browser** using project data and notification configs that were loaded server-side when the page rendered. It does not make any HTTP request and does not write to the server.

- Results and debug logs appear in the UI (Show Debug Logs button) and in the browser console (F12 → Console).
- The server log at `/var/log/ppm-notifications.log` is only written when the cron job fires the real API call.
- The **Send to [Manager]** buttons in `AdminNotificationSender` call the Pushover/Twilio routes directly and bypass the cron, timing gates, and cooldown/daily-limit counters entirely. They are intended for verifying that keys and phone setup work.

Low-Level Connectivity Test

Before configuring anything in the admin UI, confirm Pushover is reachable from the server:

```
curl -s \
  --form-string "token=YOUR_PUSHOVER_APP_TOKEN" \
  --form-string "user=YOUR_PUSHOVER_USER_KEY" \
  --form-string "message=Test from Timebars!" \
  --form-string "title=PPM Test" \
  https://api.pushover.net/1/messages.json
```

A successful response: `{"status":1,"request":"..."}` — push arrives on the phone within seconds.

You can also test the app's own API route once the dev server is running:

```
curl -X POST http://localhost:3001/api/notifications/pushover \
  -H "Content-Type: application/json" \
  -d '{"message":"Hello from Timebars","title":"PPM Test","priority":1}'
```

Document version: 1.0 — 2026-05-07