

# EXAMTRACKER INDIA

## Notification Delivery Architecture

Technical Reference • February 2026 • v1.0 • MVP: Email via Resend

## Overview

The notification system is the core product loop of ExamTracker. It is what makes users come back. An aspirant completes onboarding, we match them to eligible exams, and then we alert them at every critical moment: when a new exam they qualify for opens, when their application deadline is approaching, when their admit card is available, and when results are declared.

MVP launches with **email only via Resend**. WhatsApp (premium feature) and web push are V2. This keeps complexity low at launch while covering the notification needs of all users. Email works for everyone — even aspirants on slow connections, even those without smartphones.

### MVP SCOPE

Email via Resend is the only notification channel at launch. Three email types: (1) New eligible exam alert, (2) Deadline reminder (7 days before + 1 day before), (3) Weekly digest of all active exams. That is all. Do not build WhatsApp or push until email is working reliably and you have real users.

## The Four Email Notification Types

Every notification ExamTracker sends falls into one of these four types. Each has a different trigger, different content, and different urgency.

| Type                    | Trigger                                | Subject Line  | When Sent  | Priority                                  |
|-------------------------|--|---|--|---|
| New Exam Alert          | Eligible exam added to DB + approved   |  <i>New exam you qualify for: SSC CGL 2025</i> | Within 2 hours of admin approving the exam record  | HIGH — sent immediately                   |
| 7-Day Deadline Reminder | application_end is exactly 7 days away |  <i>7 days left to apply: SSC CGL 2025</i>     | Daily cron at 9:00 AM IST checks for T-7 deadlines | HIGH — time-sensitive                     |
| 1-Day Deadline Reminder | application_end is exactly 1 day away  |  <i>LAST DAY to apply: SSC CGL 2025</i>        | Daily cron at 9:00 AM IST checks for T-1 deadlines | <b>CRITICAL — send twice: 9 AM + 6 PM</b> |
| Weekly Digest           | Every Sunday 10:00 AM IST              | <i>Your 12 open government exams this week</i>  | Weekly scheduled job — every Sunday                | MEDIUM — informational                    |

# The Core Notification Loop

Understanding this loop is the key to building the system correctly. There are two distinct flows: **event-triggered** (new exam approved) and **scheduled/cron** (deadline reminders, weekly digest).

## Flow A: New Exam Alert (Event-Triggered)

1

### Admin approves exam record in dashboard

*Editor clicks "Approve" — notification\_verified = TRUE*

The moment an exam record goes from DRAFT to APPROVED (`notification_verified = TRUE` and `is_active = TRUE`), a Supabase database webhook fires. This webhook triggers a BullMQ job: "`new_exam_notification`" with the `exam_id` as payload.

2

### Eligibility query — find all matching users

*BullMQ worker / PostgreSQL*

The worker runs the eligibility matching query against the `user_profiles` table to find every user who qualifies for this exam. This is the same query the onboarding dashboard uses — reused here.

```
SELECT u.id, u.phone, u.email, up.category
FROM users u
JOIN user_profiles up ON u.id = up.user_id
WHERE up.onboarding_completed = TRUE
    AND u.is_deleted = FALSE
    AND EXTRACT(YEAR FROM AGE(NOW(), up.date_of_birth)) BETWEEN $min_age AND
$max_age_{category}
    AND up.highest_qualification >= $required_qualification
    AND ($required_streams IS NULL OR up.qualification_stream =
ANY ($required_streams))
    AND ($state_code IS NULL OR up.domicile_state = $state_code)
    AND ($gender IS NULL OR up.gender = $gender)
    AND u.email IS NOT NULL -- only email-reachable users
```

#### SCALE NOTE

At 500K MAU, this query could match 50,000–200,000 users for a major SSC notification. Do NOT send 200,000 emails in one shot — Resend will throttle and you could hit rate limits. Batch into chunks of 500 users and process over 2-3 hours. Queue each batch as a BullMQ job. Priority: users whose `application_end` is soonest get their emails first.

3

### Deduplication check — has this user already been notified?

*BullMQ worker / PostgreSQL*

Before sending any email, check the `notification_log` table:

```
SELECT id FROM notification_log
WHERE user_id = $user_id
    AND exam_id = $exam_id
    AND notification_type = 'NEW_EXAM'
```

If a row exists → skip this user (already notified). If no row → proceed to send. This prevents double-notifications if the job accidentally runs twice (e.g., due to a failed retry).

4

## Build personalised email content

BullMQ worker

The email is personalised per user — not a generic blast. Key personalisation fields:

- User's name (from user\_profiles.display\_name)
- User's category — show category-specific vacancy count ("OBC Vacancies: 2,847")
- Whether this is VACANCY\_AWARE mode user — show post-wise breakdown if so
- Application end date formatted clearly: "Apply before 15 March 2026"

5

## Send via Resend API

BullMQ worker → Resend

Each email is sent via **Resend's POST /emails endpoint**. The worker calls Resend with:

- from: "ExamTracker <alerts@examtracker.in>"
- to: user's email address
- subject: personalised subject line
- html: rendered email template (React Email or Handlebars)
- tags: [{ name: "type", value: "new\_exam" }, { name: "exam\_id", value: exam\_id }] — for Resend analytics

On Resend **202 Accepted** response: write a row to **notification\_log** (user\_id, exam\_id, type, channel=EMAIL, status=SENT, sent\_at=NOW(), resend\_email\_id).

On Resend **4xx/5xx error**: BullMQ retries up to 3 times with exponential backoff (1min, 5min, 30min). After 3 failures, move to dead-letter queue and alert you via a Slack webhook or email.

## Flow B: Deadline Reminders (Scheduled — Cron)

Deadline reminders run on a daily cron job at **9:00 AM IST every day**. A second send for 1-day reminders goes at **6:00 PM IST**.

| Time                 | What the cron job does  |
|----------------------|---|
| 9:00 AM IST daily    | Query: SELECT id, name, application_end FROM exams WHERE is_active = TRUE AND notification_verified = TRUE AND application_end = CURRENT_DATE + 7 → send 7-day reminders to all eligible users. Also query for application_end = CURRENT_DATE + 1 → send 1-day reminders. |
| 6:00 PM IST daily    | Second send for 1-day reminders only (application_end = TOMORROW). Re-check notification_log to avoid double-sending to users who already got the 9 AM email today for this exam.   |
| 10:00 AM IST Sundays | Weekly digest: for each user, query all exams they qualify for where is_active = TRUE and application_end > TODAY. Send one email listing all open exams with deadlines. Cap list at 10 exams (most urgent first).  |

### CRON IMPLEMENTATION

Use BullMQ's repeatable jobs to schedule the crons — not external cron services. A BullMQ repeatable job with cron: "0 9 \* \* \*" and tz: "Asia/Kolkata" fires at 9 AM IST daily. The job payload is just the trigger date. The worker then runs the query and fans out individual user emails as child jobs.

## Flow C: Weekly Digest (Every Sunday)

The weekly digest is one email per user showing all exams currently open that they qualify for. It's the one email that users check on their day off to plan the week. Design it to be useful, not spammy.

### Digest content structure:

- Header: "Your government exam updates — Week of [date]"
- Section 1: CLOSING SOON (application\_end within 14 days) — max 3 exams, sorted by urgency
- Section 2: NEWLY OPENED (notification\_date within 7 days) — max 3 exams
- Section 3: ALL OPEN EXAMS — remaining eligible exams, up to 10 total
- Footer: unsubscribe link, link to full dashboard

#### UNSUBSCRIBE RULE

Every email must have a one-click unsubscribe link. This is both a legal requirement and a Resend requirement for good deliverability. The unsubscribe link hits an API route that sets email\_notifications\_enabled = FALSE on the user's profile. Resend also has built-in unsubscribe management — use their suppression list feature to honour unsubscribes automatically.

## The notification\_log Table

Every email sent is recorded in **notification\_log**. This table serves three purposes: deduplication (don't send same notification twice), analytics (open rates, user engagement), and audit trail (prove we sent it if user disputes).

| Column                   | Type                         | Example  | Purpose   |
|--------------------------|------------------------------|--|---|
| <b>id</b>                | <i>UUID</i>                  | <i>gen_random_uuid()</i>                             | Primary key   |
| <b>user_id</b>           | <i>UUID references users</i> | <i>a1b2c3...</i>                                     | Which user received this                            |
| <b>exam_id</b>           | <i>UUID references exams</i> | <i>d4e5f6... (NULL for digests)</i>                  | Which exam this notification is about               |
| <b>notification_type</b> | <i>ENUM</i>                  | <i>NEW_EXAM / DEADLINE_7D / DEADLINE_1D / DIGEST</i> | Type of notification — used for deduplication logic |
| <b>channel</b>           | <i>ENUM</i>                  | <i>EMAIL / WHATSAPP / PUSH</i>                       | Channel used — EMAIL only at MVP                    |
| <b>status</b>            | <i>ENUM</i>                  | <i>SENT / FAILED / BOUNCED</i>                       | Current delivery status                             |
| <b>sent_at</b>           | <i>TIMESTAMPTZ</i>           | <i>2026-03-01 09:00:00+05:30</i>                     | When the API call was made to Resend                |
| <b>resend_email_id</b>   | <i>VARCHAR(100)</i>          | <i>re_abc123...</i>                                  | Resend's message ID — for webhook tracking          |
| <b>opened_at</b>         | <i>TIMESTAMPTZ</i>           | <i>NULL until Resend webhook fires</i>               | When user opened the email (from Resend webhook)    |
| <b>clicked_at</b>        | <i>TIMESTAMPTZ</i>           | <i>NULL until click tracked</i>                      | When user clicked a link in the email               |

### Critical indexes:

- CREATE UNIQUE INDEX ON notification\_log (user\_id, exam\_id, notification\_type) WHERE exam\_id IS NOT NULL — prevents duplicate sends
- CREATE INDEX ON notification\_log (sent\_at) — for analytics queries
- CREATE INDEX ON notification\_log (user\_id, sent\_at DESC) — for user notification history

# Resend Setup & Configuration

## Initial Setup

- Log into Resend (you have an account from Prepleague). Create a new API key specifically for ExamTracker. Label it "examtracker-prod".
- Add domain: examtracker.in. Verify DNS records (Resend provides SPF, DKIM, DMARC records to add). This is mandatory for good deliverability — emails sent from an unverified domain go to spam.
- Set sending email: alerts@examtracker.in (for notifications) and noreply@examtracker.in (for auth magic links).

## Email Templates

Build templates using React Email ([react.email](#)) — the standard for Resend. Templates are React components that compile to HTML. Store them in `/emails` folder in the codebase. Preview locally with the React Email dev server before shipping.

| Template File                      | Used For                           | Key Variables  |
|------------------------------------|------------------------------------|--|
| <code>new-exam-alert.tsx</code>    | New Exam Alert email               | <code>examName, examCategory, totalVacancies, categoryVacancies, applicationEnd, applyUrl, userName, userCategory</code> |
| <code>deadline-reminder.tsx</code> | 7-day and 1-day deadline reminders | <code>examName, applicationEnd, daysLeft, applyUrl, userName — daysLeft drives subject line and urgency color</code>     |
| <code>weekly-digest.tsx</code>     | Sunday weekly digest               | <code>userName, closingSoon[], newlyOpened[], allOpenExams[], totalCount</code>  |
| <code>magic-link.tsx</code>        | Auth magic link email              | <code>magicLinkUrl, userName (if known), expiresAt</code>  |

## Resend Webhooks — Tracking Opens & Clicks

Resend sends webhook events to your backend when email events occur. Register a webhook endpoint in Resend dashboard pointing to: <https://examtracker.in/api/webhooks/resend>

| Resend Event                       | What it means                       | What to do   |
|------------------------------------|-------------------------------------|--|
| <code>email.delivered</code>       | Resend confirmed delivery to inbox  | Update notification_log status = DELIVERED   |
| <code>email.opened</code>          | User opened the email               | Update opened_at timestamp in notification_log   |
| <code>email.clicked</code>         | User clicked a link                 | Update clicked_at timestamp in notification_log  |
| <code>email.bounced</code>         | Email address invalid or inbox full | Update status = BOUNCED. Set user email_notifications_enabled = FALSE. Flag user to add backup contact.                                |
| <code>email.spam_complained</code> | User marked as spam                 | <b>CRITICAL: Update status = SPAM_COMPLAINED. Immediately set email_notifications_enabled = FALSE. Never email this address again.</b> |

### DELIVERABILITY RULE

Spam complaint rate above 0.1% will get your Resend account flagged. Always honour unsubscribes immediately. Never send more than 1 notification per day per user per exam. Weekly digest = 1 email/week. New exam alerts = max 3 per day (if 3 new exams match the

user). If more than 3 new exams on the same day, batch them into one "X new exams found" email.

## User Notification Preferences

Users must be able to control what emails they receive. Store these preferences on **user\_profiles** as boolean columns:

| Preference Column                        | Default           | Controls   |
|--|-------------------|--|
| <code>email_notifications_enabled</code> | <code>TRUE</code> | Master on/off switch. Setting to FALSE stops ALL emails immediately. |
| <code>notify_new_exam_alerts</code>      | <code>TRUE</code> | Whether to receive new eligible exam alerts                          |
| <code>notify_deadline_reminders</code>   | <code>TRUE</code> | 7-day and 1-day deadline reminder emails                             |
| <code>notify_weekly_digest</code>        | <code>TRUE</code> | Sunday weekly digest email   |
| <code>notify_result_alerts</code>        | <code>TRUE</code> | When exam results are declared (future feature)                      |

These preferences are accessible from the user's Settings page in the dashboard. The unsubscribe link in every email deep-links directly to this settings page.

## V2: WhatsApp & Web Push (Not MVP — Documented for Awareness)

These channels are planned but not built at launch. They are noted here so the data model and architecture do not need to be reworked when you add them.

### WhatsApp (V2 — Premium only)

- Provider: AiSensy (Meta WABA reseller, ₹0.13/utility message)
- Gated behind Premium subscription (₹49/month). Upgrade prompt shown when user misses a deadline.
- Message types: Utility only (exam alerts, deadline reminders, admit card notifications). No marketing messages.
- DPDPA 2023 compliance: Explicit WhatsApp consent with timestamp must be recorded before first WhatsApp message. Add `whatsapp_consent_at` and `whatsapp_opt_in` columns to `user_profiles`.
- `notification_log` already has channel = WHATSAPP in the ENUM — no schema change needed.

### Web Push (V2 — All users)

- Provider: OneSignal free tier (up to 10,000 subscribers free)
- Requires user to click "Allow Notifications" in browser — conversion rate ~40-60% of users who see the prompt.
- Use for real-time new exam alerts (no email delay). Push fires immediately when exam approved.
- `notification_log` channel = PUSH already in ENUM.

**WHY  
EMAIL  
FIRST**

Email has 100% reach (every registered user has email or phone, phone → auth → email linked). Push requires browser permission (60% opt-in at best). WhatsApp requires Premium + explicit consent. Email is the only channel with guaranteed reach at launch. Get the email system working perfectly before adding complexity.

