TECHNICAL LEAD DOCUMENTATION

Omnisend Promo Automation

Newsletter + Abandoned Checkout Unique Promo Codes

Version:	1.0
Date:	May 2026
Status:	Ready for Implementation
Audience:	Technical Lead / Backend Dev

1. Executive Summary

This document describes the full technical implementation plan for Omnisend-integrated promotional code automation on the Laravel-based storefront. Two automated campaigns are in scope:

Campaign	Discount	Validity	Trigger
Newsletter Welcome	10% off	30 days	User subscribes to newsletter
Abandoned Checkout	15% off	48 hours	User starts checkout but does not complete

Promo codes are generated exclusively in the Laravel database (source of truth) and passed to Omnisend via custom events. Redemption validation reuses existing PromoValidator, user_promo, and expiration_date logic. No Omnisend-side coupon engine is used.

> *Critical design constraint: Omnisend cannot create or validate discount codes in a custom Laravel checkout. Codes must originate in your database and be delivered to Omnisend as event properties (merge tags).*

2. System Architecture

2.1 Component Overview

The following components interact to deliver the promo automation flow:

Component	Technology	Responsibility
OmnisendPromoCodeService	Laravel Service Class	Generate unique Promo rows with correct discount/validity/targeting
OmnisendPromoEventService	Laravel Service / extends OmniSend.php	POST custom events to Omnisend Events API
AuthController::subscribeNewsletter	Laravel Controller	Wire newsletter subscription to promo issuance
IssueAbandonedCartPromoJob	Laravel Queue Job	Detect abandoned carts and trigger promo issuance
Omnisend Automation A	Omnisend Dashboard	Email newsletter welcome code on newsletter_subscribed event
Omnisend Automation B	Omnisend Dashboard	Email abandoned cart code on checkout_abandoned event
PromoValidator	Existing Laravel Model	Enforce single-use, expiry, and customer targeting at checkout

Component	Technology	Responsibility
OrdersService	Existing Laravel Service	Attach user_promo on successful order (enforces one-time use)

2.2 Data Flow

The end-to-end sequence for both campaign types follows the same pattern:

1. Storefront or API emits a trigger (newsletter subscribe OR cart abandon signal).
2. Laravel backend creates a unique Promo row in the promos table with expiration_date, recurrence=1, and customer targeting.
3. Laravel POSTs a custom event to the Omnisend Events API, carrying promoCode and promoExpiresAt as properties.
4. Omnisend automation is triggered; it emails the customer using merge tags from the event.
5. Customer applies the code at checkout; PromoValidator checks expiry and single-use status.
6. On order success, OrdersService attaches user_promo, permanently blocking re-use.

3. Implementation Tasks

Six tasks must be completed, in the recommended order below. Each task maps to an entry in the project plan.

#	Task	Description	Status
1	`promo-service`	OmnisendPromoCodeService — generate unique Promo rows (10%/30d, 15%/48h, recurrence=1, targets)	Pending
2	`omnisend-events`	Emit newsletter_subscribed and checkout_abandoned custom events with promoCode / promoExpiresAt properties	Pending
3	`newsletter-hook`	Wire subscribeNewsletter: issue promo + Omnisend contact sync + trigger automation	Pending
4	`abandon-trigger`	Implement abandon detection via delayed job or Omnisend native cart + checkout_abandoned event	Pending
5	`omnisend-ui`	Document Omnisend dashboard steps: custom events, two automations, merge tags, suppression on placed order	Pending

#	Task	Description	Status
6	`qa-validation`	Test one-time use, expiry, and idempotent re-triggers on staging	Pending

3.1 Task 1 — OmnisendPromoCodeService

Create App\Services\Omnisend\OmnisendPromoCodeService with two public methods:

Method Signature	Campaign	Discount	Expiry
issueNewsletterPromo(string $email, ?User $user): Promo	Newsletter Welcome	10% (PERCENT)	now() + 30 days
issueAbandonedCartPromo(string $email, ?User $user, ?string $cartId): Promo	Abandoned Checkout	15% (PERCENT)	now() + 48 hours

Key implementation requirements:

- Code prefix: NL10- for newsletter, AC15- for abandoned cart, followed by a cryptographically random suffix.
- Set hidden = true on every generated Promo row (codes must not appear in public promo listings).
- Set recurrence = 1 to enforce one-time use per customer via the user_promo pivot.
- Attach the customer in promo_targets using TARGET_TYPE_SELECTED_CUSTOMER when user_id is available. For guests, store source email in metadata or use phone/email targeting.
- Set description to omnisend_newsletter_10 or omnisend_abandon_15 for reporting and idempotency queries.
- Idempotency: before creating, query promo_targets (or the description prefix) to check whether an active, non-expired promo for this email + campaign already exists. If it does, return the existing promo and do not fire a duplicate event.

> *Follow the pattern from HandleOrderCashbackPromo::createPromocode — this is the existing precedent for auto-generated hidden promos.*

Configuration keys to add to config/integrations.php or config/variables/services.php:

Config Key	Default	Description
OMNISEND_NEWSLETTER_DISCOUNT_PERCENT	10	Percent discount for newsletter welcome promo
OMNISEND_NEWSLETTER_VALIDITY_DAYS	30	Days until newsletter promo expires
OMNISEND_ABANDON_DISCOUNT_PERCENT	15	Percent discount for abandoned cart promo

Config Key	Default	Description
OMNISEND_ABANDON_VALIDITY_HOURS	48	Hours until abandoned cart promo expires
OMNISEND_ABANDON_DELAY_MINUTES	120	Minutes of cart inactivity before abandon job fires

3.2 Task 2 — Omnisend Custom Event Emitter

Extend app/Services/Conversion/OmniSend.php or create OmnisendPromoEventService. The payload structure for both events is:

Property	newsletter_subscribed value	checkout_abandoned value
eventName	newsletter_subscribed	checkout_abandoned
eventID	UUID (unique per call)	UUID (unique per call)
origin	api	api
eventVersion	(empty string)	(empty string)
eventTime	ISO 8601 timestamp	ISO 8601 timestamp
contact.email	Customer email	Customer email
properties.promoCode	$promo->name	$promo->name
properties.promoExpiresAt	$promo->expiration_date	$promo->expiration_date
properties.discountPercent	10	15
properties.cartValue	N/A	Optional — cart total

> *The eventVersion field must be an empty string for custom events per Omnisend API documentation. Do not omit it.*

3.3 Task 3 — Newsletter Subscription Hook

Modify AuthController::subscribeNewsletter. The current flow only syncs to Mailchimp via Spatie Newsletter. The updated flow adds:

7. Issue promo via OmnisendPromoCodeService::issueNewsletterPromo() (idempotent).
8. Sync or create contact in Omnisend (use OmniSendSyncContacts PATCH pattern). Optionally set contact custom fields: lastPromoCode, lastPromoExpiresAt, lastPromoCampaign = omnisend_newsletter.
9. POST newsletter_subscribed custom event to Omnisend via the event emitter from Task 2.

Mailchimp can remain active in parallel during the transition period. If Mailchimp is retired later, remove only the Spatie Newsletter call.

3.4 Task 4 — Abandoned Cart Detection

Two implementation options are available. Option B is recommended for alignment with the existing backend:

	Laravel Job (Recommended)
How it works	Persist started checkout events in DB; schedule IssueAbandonedCartPromoJob at +OMNISEND_ABANDON_DELAY_MINUTES.
Requires storefront JS	No — uses existing backend events
Complexity	Moderate — new job + table/queue entry
Merge tags for promo	Full control — codes generated in Laravel before event fire
Recommended?	Yes — consistent with existing pattern

the implementation steps are:

10. Store started checkout payload (persist to a new carts table or a queue record with customer email, cart contents, and timestamp).
11. Schedule IssueAbandonedCartPromoJob to run after OMNISEND_ABANDON_DELAY_MINUTES (default 120).
12. In the job, check whether an order has been placed since the started checkout timestamp. If yes, discard. If no, call OmnisendPromoCodeService::issueAbandonedCartPromo() and fire the checkout_abandoned event.
13. Cancel or no-op the job if a placed order event is received before it fires (set a flag on the cart record).

3.5 Task 5 — Omnisend Dashboard Configuration

Step 1: Register Custom Events

Go to Settings > Store > Custom Events in the Omnisend dashboard. Create both events below (names must match the API values exactly, including case):

Event Name	Property	Type
newsletter_subscribed	promoCode	Text
newsletter_subscribed	promoExpiresAt	Date / Text

Event Name	Property	Type
newsletter_subscribed	discountPercent	Number
checkout_abandoned	promoCode	Text
checkout_abandoned	promoExpiresAt	Date / Text
checkout_abandoned	discountPercent	Number
checkout_abandoned	cartValue	Number (optional)

Step 2: Automation A — Newsletter Welcome (10%)

- Trigger: Custom event → newsletter_subscribed
- Entry filter: Contact does not have tag newsletter_promo_sent
- Optional suppression: Contact placed an order in the last 24h (prevents post-purchase abuse)
- Action 1: Send email — include merge tags {{ promoCode }} and {{ promoExpiresAt }}
- Action 2: Add tag newsletter_promo_sent
- Frequency: Run once per contact (do not re-enter if idempotency is handled on the API side)

Step 3: Automation B — Abandoned Cart (15%)

- Trigger: Custom event → checkout_abandoned
- Entry filter: Contact has not placed an order since the event
- Timing: First email at +1 hour, optional reminder at +24 hours (before 48h expiry)
- Action: Email with {{ promoCode }} and {{ promoExpiresAt }}, add tag abandon_promo_sent
- Exit condition: Contact places order or pays — cancel workflow (standard Omnisend placed order suppression)

Step 4: Optional Contact Custom Fields

Mirror promo data on the contact record for segmentation and re-targeting:

- lastPromoCode
- lastPromoExpiresAt
- lastPromoCampaign

Update via the Contacts API using the existing OmniSendSyncContacts PATCH pattern.

3.6 Task 6 — QA & Validation

Newsletter Test Cases

14. Subscribe with a test email. Verify: unique NL10- code created in DB, newsletter_subscribed event fired, email received with correct code and expiry.

15. Apply the code at checkout. Verify: discount applied correctly (10%), user_promo row attached on order completion.
16. Attempt to reapply the same code after order. Verify: PromoValidator rejects with recurrence error.
17. Subscribe with the same email again. Verify: idempotency — no duplicate promo created, no duplicate event fired.

Abandoned Cart Test Cases

18. Start checkout (fire started checkout event) and do not complete. Wait OMNISEND_ABANDON_DELAY_MINUTES. Verify: unique AC15- code created, checkout_abandoned event fired, email received.
19. Apply the code within 48 hours. Verify: 15% discount applied, code consumed.
20. Attempt reuse after order. Verify: blocked by PromoValidator.
21. Start checkout, then complete an order before the delay fires. Verify: abandon job is cancelled, no promo issued.
22. Start checkout twice with the same email. Verify: only one promo issued (idempotency).

4. Validation & Business Rules

All business constraints are enforced by existing Laravel mechanisms — no custom Omnisend-side enforcement is needed or reliable.

Requirement	Enforced By	Key Fields
10% discount for newsletter	Promo model: type=PERCENT, amount=10	promos.type, promos.amount
15% discount for abandoned cart	Promo model: type=PERCENT, amount=15	promos.type, promos.amount
30-day validity	expiration_date set at creation; PromoValidator rejects expired	promos.expiration_date
48-hour validity	expiration_date set at creation; PromoValidator rejects expired	promos.expiration_date
One-time use per customer	recurrence=1 + user_promo attach on order in OrdersService	promos.recurrence, user_promo
Customer targeting	promo_targets with TARGET_TYPE_SELECTED_CUSTOMER	promo_targets
No duplicate issuance	Idempotency check in OmnisendPromoCodeService before creation	promos.description prefix
Guest checkout support	Verify PromoValid rule covers email-match targeting for guests	PromoValid.php