Yournotify Internal Product Handbook

This handbook explains the platform the way internal teams, implementation partners, and technical stakeholders actually need it explained: what it is, how it is structured, how data moves through it, what technical surfaces exist, how commercial control works, and how to deploy it without hand-waving. It is meant to be used as an internal source of truth, not as a short-form brochure.

Use this description consistently across teams: **Yournotify is a growth marketing platform for customer acquisition, engagement, retention, and brand visibility.** In practical terms, businesses use it to acquire, engage, and retain customers through email campaigns, SMS messages, WhatsApp marketing, lead generation tools, and reward incentives.

The current document is also available as an updated PDF export and a Word-friendly DOC export. Product teams usually move between the handbook and the live product surfaces, especially the email dashboard, contacts workspace, integration center, and developer area.

The purpose of this handbook is not only to describe screens. It is to help internal teams explain the operating model behind the screens. That means it intentionally covers data preparation, attribute design, sync behavior, webhook direction, wallet implications, and rollout ownership. Those are the topics that usually determine whether a customer sees the product as dependable and expandable or as a collection of disconnected tools.

It is also useful to say what this handbook is not. It is not a replacement for API reference material, provider-specific protocol documentation, or low-level migration notes. Those references still matter, especially for engineering teams. What this handbook does is bridge the gap between deep technical artifacts and customer-facing operational understanding so that sales, onboarding, support, product, and engineering can use one coherent story.

Updated 20 April 2026. Audience: sales, onboarding, support, product ops, engineering, implementation, and partners.

01. What the Product Is

Yournotify is a growth marketing platform for customer acquisition, engagement, retention, and brand visibility. That definition matters because the product is larger than a campaign sender and more structured than a generic notification service. It combines audience data, reusable content, channel execution, workflow control, lifecycle logic, analytics, pricing logic, and integration surfaces in one operating environment.

The product should not be positioned as only a marketing email tool, only an SMS gateway, or only a developer API. Each of those is a valid entry point, but none of them fully describes the product. The stronger internal story is that Yournotify helps a business decide who to reach, what to say, when to send it, through which channel, with what cost control, and how to measure the result from one workspace.

The most important outcome phrase to keep stable across decks, onboarding, documentation, and demos is this: **acquire, engage, and retain customers through email campaigns, SMS messages, WhatsApp marketing, lead generation tools, and reward incentives.** That phrasing connects the visible front-of-house use cases with the deeper product layers such as lifecycle triggers, imports, attributes, syncs, and wallet-backed execution.

Why this matters

- Customers usually arrive with one immediate pain point, but expand into adjacent capabilities quickly.
- Internal teams need a common language so sales, onboarding, support, and engineering are not describing different products.
- The platform is strongest when positioned as an operating layer for growth and customer communication, not a single isolated feature.

Consistent positioning language

- Use “growth marketing platform” for category-level description.
- Use “acquire, engage, and retain customers” for the business outcome.
- Use “email campaigns, SMS messages, WhatsApp marketing, lead generation tools, and reward incentives” for the visible product levers.

---

02. Users and Teams

Yournotify is broad by design. Different teams enter from different surfaces, but they are all operating on the same shared data model. That is why contacts, lists, segments, campaign metadata, sender configuration, wallet state, and integration mappings have to be documented together rather than in isolation.

Team	What they usually need	What they care about most	Typical starting surface
Marketing	Campaigns, templates, forms, segmentation, analytics, and audience growth.	Speed of campaign creation, content reuse, list quality, and visible performance.	/email, /templates/email, /forms
Operations and support	Transactional alerts, issue-driven messages, customer updates, delivery verification.	Reliability, speed, status clarity, and fallback handling.	/sms, /whatsapp
Sales and CRM	Lead capture, contact enrichment, imports, lists, segments, lifecycle follow-up.	Audience quality, sync freshness, and timely follow-up.	/contacts, /lists, /segments
Engineering and product	REST APIs, webhooks, SMTP, SMPP, integration callbacks, embedded editor flows.	Predictable contracts, field mapping, callback reliability, and operational visibility.	/developer, /integration
Admin and finance	Plans, entitlements, wallet, permissions, pricing controls, transaction history.	Governance, usage visibility, commercial predictability, and policy enforcement.	/wallet, /team

These personas overlap in practice. A lifecycle rollout may start with CRM or growth teams, require engineering for sync and webhook setup, then rely on support teams to verify delivery evidence after launch. The platform is therefore most successful when internal handoff points are documented and when each team understands the data they inherit from the previous step.

A useful internal operating habit is to map every customer initiative across four questions. Who owns the audience data. Who owns message content. Who owns execution readiness. Who owns post-send verification. In smaller accounts one team may answer all four. In larger accounts those answers split across marketing, support, engineering, and finance. That split is exactly where weak implementations go wrong, because ownership gaps appear between import and campaign launch, or between webhook setup and delivery troubleshooting.

For that reason, every implementation conversation should identify the lead operating team and the dependent teams early. A marketing-led rollout without engineering involvement may work if the account only uses forms and local templates. The same approach fails if the account expects CRM sync, lifecycle stage derivation, transactional callbacks, and wallet-sensitive multi-channel sending. The handbook should help internal teams spot that difference quickly.

03. Core Product Modules

The product is best understood as a connected set of modules rather than as a flat navigation list. Each area solves a specific job, but the value compounds because audience data, template logic, execution state, and commercial controls can all be reused.

Module	What it does	Why it exists	Unique value proposition
Campaigns	Create, schedule, update, test, and monitor messages across channels.	Acts as the central execution layer for outbound engagement.	Unifies promotional and transactional delivery under one operating model.
Templates	Stores reusable local templates and resolves approved WhatsApp templates.	Reduces duplication and speeds repeated execution.	Lets teams standardize message structure while respecting channel-specific rules.
Audience	Manages contacts, lists, segments, lifecycle context, imports, and profile history.	Keeps recipient context structured and reusable.	Turns disconnected customer records into a usable targeting layer.
Automation	Handles event-based, schedule-based, and condition-based workflow execution.	Moves the product beyond one-off campaigns.	Converts messaging into repeatable journeys instead of manual repetition.
Lifecycle	Maps customer timing and state from imported or synced data.	Supports stage-aware engagement rather than flat list blasting.	Allows teams to act on business meaning, not just on contact existence.
Analytics	Tracks campaign outcomes, recipient-level evidence, and operational signals.	Shows what happened and what needs correction.	Connects communication activity to iteration and accountability.
Setup	Configures senders, channels, teams, domains, and integrations.	Makes production readiness explicit.	Builds operational correctness into the product instead of leaving it outside the product.
Commercial	Handles plans, pricing, entitlements, wallet state, and usage control.	Formalizes monetization and execution limits.	Aligns platform usage with controlled scale and real channel cost.

The modules are not independent silos. A contact import can create or update audience records, which can then be grouped into a list, filtered by a segment, referenced by a campaign, charged through wallet pricing, and confirmed back to another system through webhooks. That full chain is the real product.

Another useful way to explain the module structure is by customer maturity. Early-stage accounts often begin with forms, contacts, templates, and one or two campaign channels. Growth-stage accounts then add segments, analytics, rewards, and more deliberate branding. Mature operational accounts add integration syncs, developer webhooks, lifecycle logic, and stronger commercial governance. The product is designed to support that progression without forcing a full rebuild each time the customer expands the scope.

This is why internal teams should avoid treating the modules as a list of isolated menu items. The better explanation is that Yournotify is a layered operating system. Audience readiness sits below execution. Execution sits below

measurement. Measurement and commercial state feed back into future targeting and workflow decisions. Once teams understand that loop, the product stops looking like a disconnected collection of screens.

---

04. Major Use Cases

Internal teams should know the main customer jobs the platform can solve. These use cases are not just marketing examples; they are also implementation frames. They determine which channel is primary, whether audience data must be imported or synced, whether approval-bound templates are needed, and whether the account needs wallet-backed transactional capacity or broader growth workflows.

Promotional communication

Promotional use cases usually begin in the campaign and template layers. Teams define message strategy, create reusable assets, select segments or lists, and then monitor campaign performance. In this mode, the platform behaves like a growth engine: acquisition campaigns from forms, re-engagement journeys from audience history, and branded communication across email, SMS, WhatsApp, and push where policy permits.

- Email newsletters, launch campaigns, and seasonal promotions.
- Promotional SMS for short offers and urgent reminders.
- WhatsApp marketing templates where approved under channel policy.
- Push or in-app prompts for product reactivation and lightweight visibility.

Transactional communication

Transactional use cases behave differently. They usually start from an event outside the dashboard: an order change, billing event, authentication step, support action, or operational system trigger. The platform needs to resolve recipient identity quickly, enforce channel-specific rules, and preserve clear delivery evidence. That is why APIs, webhook callbacks, sender readiness, and template correctness matter as much as the campaign editor.

- Order confirmations, delivery updates, and service notifications.
- Billing reminders, payment issues, renewal notices, and account alerts.
- Password reset, authentication, and one-time operational messaging.
- Customer support or service desk notifications originating from another system.

Lifecycle and retention

Lifecycle-led use cases sit between growth and operations. They are neither random blasts nor purely transactional messages. They rely on data refresh, stage mapping, contact state, and timing windows. A renewal journey, inactive-user reactivation program, or milestone sequence all depend on data quality and sync behavior before they depend on message copy.

- Welcome journeys and onboarding sequences.
- Renewal nudges, subscription expiry prompts, and account health reminders.
- Win-back sequences for inactive or at-risk users.
- Milestone communication tied to customer stage or profile changes.

Lead generation and audience building

Lead generation is an explicit part of the product story and should be said directly. Forms, imports, sheets, CRM syncs, and list-building are not pre-work outside the platform. They are part of the platform’s acquisition motion. This matters especially when teams use the forms area to collect new leads or the contact import flow to operationalize offline audience sources.

Industry examples

Industry	Typical use case	Best-fit product mix	Why the fit is strong
E-commerce and retail	Acquire new buyers, recover abandoned interest, drive repeat purchases, send order updates.	Email, SMS, WhatsApp, rewards, wallet-backed usage control.	Combines acquisition, branded promotion, and operational order communication in one lifecycle.
Financial services	Send alerts, onboarding, billing reminders, lifecycle updates, and trust messaging.	SMS, email, WhatsApp, automation, lifecycle, API and webhook flows.	Balances speed, auditability, and repeatable state-based messaging.
Education	Drive applications, reminders, admission follow-up, student engagement, and deadline campaigns.	Forms, contacts, email, SMS, lifecycle segments, rewards.	Supports both acquisition funnels and deadline-driven retention.
Healthcare	Appointment reminders, care-plan nudges, onboarding, and service updates.	SMS, WhatsApp, email, automation, transactional flows.	Strong when timing and direct reach matter more than long-form promotion.
Media and events	Audience acquisition, registration, announcements, sponsor visibility, and post-event follow-up.	Lead generation forms, email, SMS, push, analytics.	Connects audience growth, event communication, and sponsor-facing brand campaigns.
Telecom and utilities	Usage notices, billing, service alerts, outage communication, and retention messaging.	SMS, email, WhatsApp, SMPP or API, lifecycle timing.	Operational systems can trigger immediate communication while growth teams still run retention flows.

How value compounds over time

The first successful use case often changes what the customer believes the product is for. An account that begins with email campaigns may later realize that the same contact model can support WhatsApp utility messages and renewal flows. An account that begins with SMS reminders may later discover that forms, segments, and rewards can be used to build a broader growth program. This is why internal teams should always ask what the second and third likely use cases are, not just the first one that closed the sale.

A simple pattern appears across industries. Phase one is immediate communication pain: launches, reminders, confirmations, or audience upload. Phase two is repeatability: templates, lists, and cleaner targeting. Phase three is systemization: syncs, lifecycle stages, webhooks, automation, and commercial controls. Yournotify is strongest when internal teams guide customers through those phases rather than freezing the account at phase one.

Industry operating notes

Retail and commerce implementations usually need aggressive audience refresh, offer segmentation, order-status communication, and repeat-purchase retention. Financial services customers often care more about trust, correctness, and event-driven operational messaging. Education and healthcare deployments depend heavily on timing, reminders, and long-running nurture or guidance sequences. Utilities and telecom scenarios care about service events, structured operational notice, and sometimes protocol compatibility. Media and events use cases emphasize audience growth, branded visibility, sponsor communication, and time-bound campaign peaks.

These differences affect configuration choices. Commerce teams often care about coupon or reward linkage. Financial teams care about template correctness and callback evidence. Education teams care about form capture and lead-to-

contact conversion. Healthcare teams care about timing-sensitive reminders and profile accuracy. Enterprise operational teams care about imports, syncs, and API contracts. The same product surface can support all of them, but the implementation pattern is not identical.

---

05. Templates, Contacts, Lists, and Segments

Templates deserve to appear before contacts in this handbook because many workflows begin by deciding the content object and only then selecting the audience object. That is especially true for WhatsApp, where approval-bound templates shape what can be sent, and for operational programs where a message pattern is reused across many recipient groups.

Local templates

Local templates are reusable content objects stored by the platform for email, SMS, push, and in-app messaging. They reduce duplication, help teams standardize copy and structure, and make it possible for operational flows to reuse pre-approved patterns instead of rebuilding each message. This is the logic behind the templates dashboard: it is not just a content library, it is a productivity and consistency layer.

```
GET /templates?channel=email|sms|push|inapp
```


```
{
  "status": "success",
  "data": {
    "results": [
      {
        "id": 901,
        "name": "Welcome Series v2",
        "channel": "email",
        "visibility": "private",
        "is_owner": true,
        "updated_at": "2026-04-20T07:48:13.000Z"
      }
    ],
    "total": 1,
    "per_page": 10,
    "page": 1
  }
}
```


WhatsApp templates

WhatsApp templates are not stored in the same way as local templates. They are resolved from approved Meta template definitions and carry channel-specific constraints such as category, language, component structure, and approval status. This distinction matters because implementation teams often assume “template” means the same thing everywhere. In Yournotify, local templates and approved WhatsApp templates are related but not identical concepts.

```
GET /campaigns/whatsapp/templates
```


```
{
  "status": "success",
  "data": {
    "rows": [
      {
        "id": "order_update:en_US",
        "name": "order_update",
        "language": "en_US",
        "category": "UTILITY",
        "status": "APPROVED",
```

```
        "text": "Hello {{1}}, your order {{2}} is now {{3}}.",
        "components": [
          {
            "type": "BODY",
            "text": "Hello {{1}}, your order {{2}} is now {{3}}.",
            "example": {
              "body_text": [["Amina", "#19361", "Delivered"]]
            }
          }
        ]
      }
    ],
    "total": 1
  }
}
```

Use `MARKETING` for promotional template context and `UTILITY` or `AUTHENTICATION` for transactional context.

Audience objects

Object	Meaning	Practical role	What teams should remember
Contact	A single audience record.	Stores reachability, profile attributes, lifecycle context, and status.	It is the base unit for targeting and delivery eligibility.
List	A reusable collection of contacts.	Supports grouped targeting, imports, sync destinations, and stable audience buckets.	Lists are often operational destinations for imports and syncs.
Segment	A dynamic subset of the audience.	Evaluates rules and returns matching contacts at execution time.	Segments rely on data quality more than on manual maintenance.
Direct recipient object	A one-off inline target.	Used for transactional or developer-driven sends without full audience preparation.	Useful for immediate operational flows, but less reusable than saved audience objects.

The practical relationship is simple. Templates define what is intended to be sent. Contacts define who can be reached. Lists and segments define how the target set is assembled. Direct recipient payloads exist for fast operational or developer-triggered sending, but they do not replace audience modeling when the use case grows beyond one-off dispatch.

Contact fields and sample attribs

Attribs should be understood here as one flexible field on the contact record, not as a separate entity in the product. A contact still has core fields such as name, email, telephone, channel status, and lifecycle context. The `attribs` field simply holds extra structured values that the business wants to preserve for targeting, personalization, sync matching, or reporting slices.

```
{
  "name": "Amina Yusuf",
  "email": "amina@example.com",
  "telephone": "+2348012345678",
  "attribs": {
    "phone": "+2348012345678",
    "customer_tier": "gold",
```

```
    "branch": "Ikeja",
    "crm_owner": "growth-team",
    "plan_family": "premium",
    "renewal_date": "2026-05-14",
    "external_customer_id": "CUS-20491"
  }
}
```

That sample shows the practical split. Core identity remains explicit. The `attribs` field carries business-specific detail that may vary by account. A retail business may store loyalty tier, preferred store, and last cart class. An education account may store intake term, faculty, or application stage. A financial-services account may store product type, risk band, or branch ownership. The field is flexible, but the keys still need discipline if the account wants reliable segmentation and personalization.

Lists and segments are not interchangeable

This distinction is worth stating clearly because teams often confuse it in live projects. A list is a stable container. It is useful when the business wants a known collection of contacts, often tied to an import, a sync target, or a deliberate audience bucket. A segment is an evaluation rule. It is useful when the business wants the current contacts who match a condition right now. Lists are often operational destinations. Segments are decision layers.

That difference becomes more important as the account matures. A team may import contacts into a “March Event Leads” list, then create a segment that filters that list down to contacts who have not yet responded, or to contacts whose attributes show a particular source, region, or customer stage. Internal teams should teach customers to avoid using static lists for logic that should really be expressed as a segment rule.

Content and audience governance

Templates and audience objects should be governed together. A reusable template without a stable targeting method produces manual work. A well-structured segment without a governed template library still produces inconsistent messaging. The best accounts establish naming rules, visibility rules, and ownership rules for both content and audience entities. This reduces confusion when campaigns move from one operator to another or when multiple teams share the same workspace.

The basic principle is simple. If something is meant to be reused, give it a stable name, a clear owner, and a reliable input shape. That applies equally to templates, lists, segments, import mappings, and sync configurations.

---

06. Contact Fields and Attribute Mapping

This is one of the most important areas to explain properly because a large share of product behavior depends on fields that are not always visible in the main editor. The product uses a mixture of first-class columns and a flexible JSON field for additional attributes. That field is commonly referred to in the codebase as `attribs`. It should be treated as a field on contact or campaign records, not as a separate product entity.

Why attribs matter

Attributes allow implementation teams to keep business-specific fields without waiting for a schema change for every customer variation. In practical terms, the `attribs` field lets the product hold custom recipient context, import-linked metadata, and channel-specific identifiers while keeping high-value operational fields explicit. Done well, this makes the platform adaptable. Done poorly, it creates hidden coupling and makes debugging harder. That is why the handbook needs to explain the field with more depth than a one-line gloss.

Campaign attribs

Campaign records store JSONB attributes alongside first-class fields such as channel and campaign type. In the current codebase, campaign attribs are used for operational linkage and import provenance. Common examples referenced by the routes include `imported_list_id`, `pending_import_job_id`, `pasted_import_job_id`, and `pasted_import_result`. These values matter because they tell support and implementation teams how the campaign was assembled and what import activity fed it.

In plain language, campaign attribs often answer questions such as: Did this campaign come from a prepared list or from pasted input? Was there an import job involved? If the audience was built just before launch, what job result should an operator inspect? When teams say “the send logic is fine but the audience seems wrong,” campaign attribs are often part of the trail.

Contact attribs

Contacts also store JSONB attributes, but the role is broader. Contacts need flexible space for customer-specific fields that are neither universal nor disposable. A business may care about branch, plan family, education cohort, merchant tier, policy type, service zone, or an external system identifier. Rather than creating a bespoke column for each of those, the platform keeps custom detail in contact `attribs` while retaining important reachability and status data as explicit fields.

This distinction matters operationally. Contact records are not just passive storage. They are read during segmentation, lifecycle evaluation, import upserts, and channel resolution. That means any field stored in `attribs` should be treated as live operational input if teams plan to target on it or derive message content from it.

Reachability and channel identity

The codebase explicitly resolves phone identity from more than one potential location. For SMS and WhatsApp, the send flow checks values in `attribs.phone`, `attribs.msisdn`, and `attribs.mobile`. For push, it can resolve `attribs.push_subscriber_id` or `attribs.push_token`. This is a good example of why field-level documentation cannot be shallow. If onboarding teams import a phone number under a custom attribute key that the send flow does not recognize, the contact may exist but still not be reachable.

Operationally, the best practice is to normalize as early as possible and only use alternative attribute keys when necessary for compatibility with source systems. The more the platform has to guess at runtime, the more room there is for ambiguity. The attribute layer is flexible, but it should still be governed.

Lifecycle data is broader than attribs

Not every meaningful field lives in `attribs`. Lifecycle stage, lifecycle definition key, status key, completion dates, and engagement-related profile metrics are stored in dedicated lifecycle and profile tables as well. That means “contact state” in Yournotify is not a single flat object. It is a combination of explicit contact fields, flexible attributes, lifecycle records, and profile-level activity signals such as engagement score and last activity timestamps.

Data layer	What it typically holds	Why it is separate	Operational implication
Core contact fields	Name, email, telephone, reachability status, basic identity.	These are foundational to targeting and delivery.	If these are wrong, sending and dedupe become unreliable.
Contact attribs field	Custom business fields, alternate identifiers, source-specific metadata.	It varies by customer and use case.	Useful for segmentation and personalization, but requires naming discipline.
Campaign attribs field	Import linkage, campaign assembly context, job metadata.	It captures execution provenance and dynamic campaign context.	Useful for debugging launch preparation and audience build history.
Lifecycle tables	Stage, status keys, lifecycle definition linkage, completion dates.	Lifecycle state needs structured evaluation and transitions.	Supports timing-aware journeys and stage-based targeting.
Subscriber profile metrics	Engagement score, last activity time, derived profile indicators.	These are calculated or behavior-driven rather than simple attributes.	Useful for retention and quality scoring.

Practical governance rules for attribs

Rule	What it means in practice	Why it matters
Keep core identifiers explicit	Name, email, telephone, and channel status should stay in obvious fields where the product expects them.	Prevents reachability and dedupe behavior from becoming ambiguous.
Use attribs for customer-specific context	Store business fields that vary by account or vertical inside attributes.	Maintains schema flexibility without losing important detail.
Normalize alternate phone keys	If source systems use mobile, msisdn, or phone differently, map them deliberately.	SMS and WhatsApp delivery depend on recognizable field names or clean normalization.
Document import and sync mappings	Keep a known rule for how source columns become contact or campaign attributes.	Reduces silent data drift across repeated imports and sync runs.
Avoid storing business logic only in free-form notes	If a field will drive segmentation, personalization, or lifecycle logic, it needs a stable key.	Prevents valuable data from becoming operationally unusable.

Personalization and attribute readiness

The `attribs` field is not just for storage. It directly influences message relevance when teams personalize templates or derive audience subsets from business context. If a commerce team stores customer tier or abandoned cart class in a

stable attribute key, that field can support both segmentation and message wording. If an education team stores intake term, campus, or application stage cleanly, that data can drive highly specific journeys. If those same values are inconsistently mapped across imports, the result is not only bad targeting but also misleading personalization.

That is why attribute governance is really communication governance. Poorly structured attributes create broken audience logic, weaker analytics slices, and more support ambiguity. Cleanly structured attributes create better campaigns, better retention workflows, and easier debugging. In many implementations, the quality of the attribute layer determines whether the account can evolve beyond simple blasts.

Campaign attribution and provenance

Campaign attribs deserve special emphasis because they help reconstruct how a sendable object was assembled. For example, when a campaign links back to an imported list or a pending import job, support teams can tell whether the audience was manually prepared, pasted inline, or generated through an import workflow. That provenance is often what separates “the message content looked correct” from “the right people were actually targeted.”

Internal teams should get used to thinking of campaign attribs as execution context rather than decorative metadata. They answer operational questions such as where the audience came from, what preparatory job created it, and whether a launch depended on dynamic audience assembly. Those questions matter in audits, escalations, and post-incident reviews.

How to explain attribs to a customer or internal team

- Attribs are flexible custom fields, not a dumping ground for everything.
- Reachability fields should be normalized carefully because sending depends on them.
- Lifecycle and engagement state are not only attributes; they also live in dedicated data structures.
- If a team wants to target or personalize using a field, that field needs a clear naming and mapping rule.

07. Contact Import and Audience Operations

Audience operations are a major part of the product and should be treated as first-class product behavior, not as setup work outside the main story. Imports are how many accounts become usable for the first time. They are also how recurring campaigns, lifecycle rollouts, and reactivation programs maintain data freshness before any message is sent.

Primary import entry point

The main API surface is `POST /contacts/import`. The route accepts source content such as CSV or spreadsheet-derived input and parameters such as `lists`, `target_mode`, `auto_list_name`, and `defer_start`. The UI around this lives in the contact import route, where files are parsed, headers are normalized, and mappings are prepared before the API receives the final payload.

Synchronous versus queued imports

The import service supports more than one execution mode. Smaller imports can be processed directly and returned quickly. Larger jobs are queued and tracked in background processing infrastructure, with job records, notifications, and completion events. This is important because implementation teams often ask whether imports are “instant.” The right answer is that imports are optimized for responsiveness, but the platform also has a background path so large datasets do not lock the request path or create poor operator experience.

That design also explains why an account may see a list destination or contact count update after the initial request returns. The import was accepted, but the actual row processing continued in background. Teams need that distinction when explaining status to customers.

How rows are prepared

The import service normalizes rows, deduplicates by email and telephone combinations, validates contact shapes, and prepares custom attribute payloads. This means the import layer is not just a bulk insert. It is an audience preparation workflow. It decides whether a row maps cleanly to a contact, whether it updates an existing audience record, and whether the custom data should become contact attributes. It is one of the most important trust points in the entire product.

Import concern	What the platform does	Why it matters
Header normalization	Standardizes incoming column names from CSV or spreadsheet sources.	Reduces operator error and makes repeated imports more predictable.
Dedupe	Matches on email and telephone patterns where appropriate.	Prevents needless duplication and keeps audience counts trustworthy.
Attribute capture	Maps extra columns into contact attribs.	Preserves source-system richness for segmentation and personalization.
Target routing	Can link imported contacts to existing lists or auto-create a list.	Keeps imports immediately useful for campaign and workflow selection.
Async fallback	Queues larger imports into background jobs.	Protects the interactive path while preserving auditability.

Import results and operator evidence

Import result reporting includes counts such as created, updated, and skipped rows. Those numbers are essential because audience issues rarely show up as code exceptions alone. A job can succeed from an infrastructure point of view while still producing a poor business outcome if too many records were skipped or if a column mapping caused important fields to land in the wrong place. The best internal explanation is that import completion is not just about “done”; it is about the shape of the result.

The platform also emits import-related notifications and developer webhook events such as `contact.import.completed` and `contact.import.failed`. That means external systems can react when audience preparation finishes, which is especially useful for downstream automation or for implementation dashboards that monitor sync health.

Import modes and when to use them

There are at least three common audience-loading patterns in the product. The first is a fast one-time upload to get an account moving. The second is a structured recurring import operated by a human team on a schedule. The third is a true systemized sync where the source system becomes the durable upstream owner. Internal teams should identify which of those patterns the customer is really using, because each one implies a different operating burden and risk profile.

A one-time upload is fast but fragile if it becomes the long-term habit. A recurring manual import can work for smaller organizations, but it needs mapping discipline and ownership. A true sync is the most durable, but it requires stronger agreement on identity fields, lifecycle implications, and monitoring. The product supports all three patterns, but the handbook should make clear that they are not interchangeable maturity levels.

Operational example

Consider a customer uploading a spreadsheet of renewal candidates. The import flow parses headers, normalizes core columns, maps extra columns into attribs, deduplicates against existing contacts, attaches or creates the target list, and reports created, updated, and skipped counts. If the next campaign underperforms, the right investigation path is not only the message copy. Teams should also ask how many rows were skipped, whether the renewal date became a usable attribute, and whether the target list actually represents the intended cohort. That is the difference between treating import as clerical work and treating it as audience engineering.

Common failure patterns

- Important columns land in free-form attributes with inconsistent key names across separate imports.
- Phone numbers are present but not normalized into keys the send flow resolves correctly.
- The operator assumes a successful job means the audience is correct without checking created, updated, and skipped totals.
- Manual recurring uploads continue long after the account should have moved to sync.
- Lists are auto-created repeatedly without naming discipline, making later segmentation harder.

Audience operations beyond import

Audience operations continue after the first load. Contacts can be updated individually, enriched through sync, moved into lists, evaluated by segments, and profiled by lifecycle and engagement state. The contacts area, lists area, and segments area should be described internally as one operational cluster rather than three unrelated screens.

That cluster is where audience quality is either protected or diluted over time. Contacts need consistent identifiers. Lists need stable purpose. Segments need business logic that reflects current reality. Lifecycle state needs ongoing refresh. If

any of those drift, message quality and reporting trust usually drift with them.

08. Channels and Delivery

Yournotify supports multiple channels under a common execution model, but each channel still has specific strengths, validation rules, and routing assumptions. Teams should describe the channels as coordinated, not identical.

Channel	Strength	Typical use	Key controls
Email	Rich content, flexible formatting, broad reporting.	Newsletters, onboarding, invoices, lifecycle sequences, branded communication.	Domains, senders, reply-to, attachments, templates, deliverability readiness.
SMS	Immediate, concise, high attention.	Alerts, reminders, short promotions, operational updates.	Sender IDs, routing, transactional versus promotional usage, phone normalization.
WhatsApp	Structured templates and strong service messaging continuity.	Order updates, reminders, support flows, utility messaging, approved marketing templates.	Template approval, number identity, component compatibility, policy category.
Push	Low-cost immediacy inside app ecosystems.	Reactivation, release prompts, urgent engagement, habit nudges.	Push token or subscriber identity, target URL, app context.
In-app	Product-native context inside the customer experience.	Guidance, announcements, prompts, onboarding, contextual nudges.	Placement, audience targeting, product-state logic.

How delivery should be explained internally

Direct sends optimize for speed and narrow recipient sets. Scheduled and bulk campaigns optimize for orchestration, traceability, and reporting continuity. The platform validates sender readiness, template structure, media compatibility, wallet charging logic, and channel-specific requirements before or during execution depending on the path. Delivery is therefore not just a final handoff to a provider. It is a layered process that includes input validation, audience resolution, cost control, provider submission, and post-send evidence.

Channel value by growth objective

Growth objective	Most natural channels	Reason
Customer acquisition	Email, SMS, forms, rewards	Strong for opt-in capture, follow-up, and activation sequences.
Customer engagement	Email, WhatsApp, push, in-app	Supports repeated interaction and richer brand or product touchpoints.
Customer retention	SMS, WhatsApp, email, lifecycle, rewards	Combines timing-sensitive reminders with incentive and state-based messaging.
Brand visibility	Email, WhatsApp marketing, forms, media-rich content	Supports repeated exposure and more expressive campaign storytelling.

Channel selection heuristics

Internal teams should also be able to explain why one channel is a better first choice than another. Email is usually the most expressive and reusable content channel. SMS is the most concise and time-sensitive. WhatsApp is strongest

where approved templates, rich service updates, or conversation continuity matter. Push works best when the customer already controls an app relationship and wants low-friction reactivation. In-app messaging is strongest when the product itself is the place where action happens. These are not hard rules, but they are strong operational defaults.

The right channel mix also depends on data quality. If an account has strong email coverage but weak phone normalization, email may be the safer early channel even if leadership prefers SMS. If an account already has a trusted WhatsApp identity and structured template library, WhatsApp may be the fastest path to reliable service messaging. If an account has product usage telemetry but weaker external identifiers, push or in-app may outperform more expensive channels. Channel planning is therefore part messaging strategy and part data reality check.

When teams ask where to start in the live product, the answer depends on channel scope. Campaign creation usually begins in /email, /sms, or /whatsapp, but delivery reliability still depends on sender and data readiness set elsewhere in the product.

---

09. Key Flows

These are the main flows internal teams should be able to explain and support clearly. Each flow crosses more than one module, which is why shallow feature-by-feature explanations tend to fail.

Standard campaign launch

1. Bring in audience through import, forms, sync, or direct API input.
2. Choose a reusable template or create fresh content.
3. Set sender and channel details.
4. Choose lists, segments, or direct recipients.
5. Run test send if the channel and use case call for it.
6. Launch immediately, schedule, or save draft.
7. Review analytics, recipient-level outcomes, and any downstream callbacks.

Transactional operational message

1. An external application, workflow, or support action triggers the event.
2. The platform resolves recipient identity and sender context.
3. The flow selects a direct message or approved template path.
4. Validation checks wallet state, template compatibility, channel rules, and required identifiers.
5. The send executes immediately and preserves evidence for later inspection.
6. Provider callbacks and webhooks update the downstream view of what happened.

Lifecycle-led engagement

1. Operational or CRM data arrives through import or sync.
2. Lifecycle state is mapped from that incoming data.
3. Segments or automations target the relevant stage, score, or inactivity pattern.
4. Reusable content is selected for each stage.
5. Messages send according to state and timing rather than static membership alone.

What commonly breaks these flows

Most failures are not caused by a single broken button. They usually come from mismatches between field mapping, sender readiness, template assumptions, or wallet state. A contact can exist but still not be reachable. A message can be valid content-wise but invalid for a channel. A sync can finish but still leave lifecycle state incomplete. A webhook can fire but still not give the downstream system enough context to act. The product works best when internal teams explain these dependencies early instead of presenting execution as a one-click black box.

10. Automation, Lifecycle, and Rewards

Automation is the workflow layer. Lifecycle is the customer-state layer. Rewards add incentive-driven behavior to the communication system. Together they move the product from isolated sends to compounding programs.

Automation

Automation coordinates event-driven or schedule-driven steps such as delays, branches, conditions, and action nodes. A useful internal explanation is that campaigns define a message event, while automation defines the logic that decides when message events should happen repeatedly. It is the difference between launching a send and designing a communication system.

Lifecycle

Lifecycle modeling lets teams act on business meaning instead of raw list membership. Imported or synced signals can place a contact into a stage, set a lifecycle status key, or update a profile indicator. From that point on, retention, renewal, and milestone messaging become more explainable because the platform has state, not just contact presence.

Rewards

Rewards turn incentives into part of the engagement model. That can mean loyalty-style programs, promotion-linked reward actions, submission-based participation, or structured encouragement to take the next action. Rewards matter to the product story because they provide another way to acquire, engage, and retain customers beyond raw message frequency.

In the live product, teams often navigate between /automations, /lifecycle, and /rewards when building recurring programs. That path should be referenced naturally in implementation discussions rather than isolated in a separate “links” appendix.

11. API, Webhooks, and SDK

The platform guide should make it unmistakable that Yournotify is not UI-only. It exposes programmable interfaces for application teams, implementation partners, and operational systems. Those interfaces are not side features. They are central to transactional messaging, sync-driven lifecycle work, and automated downstream processing.

Surface	What it does	When to use it	What to explain internally
REST API	Creates campaigns, manages contacts, fetches templates, triggers syncs, and operates the workspace programmatically.	Use for application integration, internal tooling, and system-to-system control.	APIs are how another system tells Yournotify what to do.
Developer webhooks	Sends event payloads to a configured destination URL for selected modules.	Use for asynchronous follow-up and downstream workflow reactions.	Webhooks are how Yournotify tells another system what just happened.
Provider and channel callbacks	Receives delivery and event feedback from email, SMS, WhatsApp, and push providers.	Use for status reconciliation and reporting evidence.	These keep product state aligned with provider-side reality.
SDK or hosted editor surfaces	Embeds content-building and export workflows inside another product experience.	Use when Yournotify functionality must live inside a partner or internal application.	The editor path is about embedability, not just standalone authoring.

Developer webhook modules

The current webhook service supports modules such as `contact`, `email`, `sms`, and `whatsapp`. The developer-facing configuration flow in the developer webhook page is where accounts set a destination URL and choose which modules they want to receive. That should be documented as a business-operational tool as much as a technical tool, because support and product teams often rely on webhook outcomes to explain customer-visible behavior.

Webhook event patterns worth understanding

There are several webhook classes in the platform. Developer webhooks notify external systems about internal events. Public or provider-specific webhook routes ingest callback events from email, WhatsApp, push, and other channel providers. Import completion can also dispatch webhook events. This matters because “webhooks” is often used as a single vague term, but the direction of travel is different. Some webhooks leave Yournotify. Some webhooks enter Yournotify. Teams should say which one they mean.

Examples of internal developer-facing events include contact import completion or failure. Examples of inbound provider callbacks include delivery updates, open or click tracking, bounce data, and channel-specific event feeds. If a customer wants system-to-system reliability, the right answer is not merely “we have webhooks.” The right answer is which webhook path they need and what event family it covers.

Common implementation scenarios

Scenario	Primary interface	What usually matters most
An application wants to trigger operational messages directly.	REST API plus provider callbacks.	Payload stability, template correctness, fast status evidence.

Scenario	Primary interface	What usually matters most
A CRM or back-office system wants to know when audience processing finishes.	Developer webhook events.	Reliable destination URL, event filtering, and result object parsing.
A partner wants to embed content authoring in another product.	SDK or hosted editor surfaces.	Embed behavior, export options, and content ownership boundaries.
An operations team wants two-way accountability between source systems and communication outcomes.	Combination of API, webhooks, analytics, and integration sync runs.	Traceability across system boundaries rather than isolated success signals.

These scenarios matter because the same customer may need more than one at once. A retail app might create campaigns through the API, receive delivery updates through callbacks, and still rely on manual dashboard users for template management. A finance platform might sync contacts from a source system but also expect webhook confirmation when an import job completes. Internal teams should resist the temptation to force every implementation into one “recommended” pattern. The better approach is to identify what the upstream system owns, what the Yournotify workspace owns, and how evidence moves between them.

Another important point is that programmability does not replace operational UI, and the UI does not replace programmability. Mature accounts usually use both. The dashboard is where operators review state, adjust templates, inspect analytics, and manage lists or segments. APIs and webhooks are where system-to-system timing and reliability live. The strongest implementations choose the right surface for each responsibility instead of trying to do everything in one place.

---

12. SMTP and SMPP

Protocol-level references matter because not every customer or partner integrates through modern REST patterns first. Some environments already know how to submit email through SMTP or SMS through SMPP, and they want the platform to fit those realities without forcing a full re-platforming effort.

Reference	What it does	Best-fit scenario	Why it matters in the handbook
SMTP documentation	Explains email submission using standard mail protocol behavior.	Use when a customer already has software that sends via SMTP.	Shows that the platform can fit existing enterprise mail workflows.
SMPP documentation	Explains telecom-style SMS submission patterns.	Use when a partner or business system already works with SMPP-based messaging.	Signals compatibility with telecom-oriented environments and gateways.
Webhook documentation	Explains event callbacks and asynchronous notifications.	Use when another system must react after sends, imports, or status changes.	Keeps implementation planning grounded in real event flow.
API documentation	Explains authenticated product operations over REST.	Use when application-driven integration is preferred.	Provides the highest-control modern integration path.

Including SMTP and SMPP in the internal handbook prevents the product story from leaning too far toward web-dashboard-only thinking. It gives sales and implementation teams a better answer for customers who already have operational infrastructure and simply need the platform to integrate into it. In those cases, compatibility is part of the product's value proposition, not a secondary note.

13. Analytics and Admin

Yournotify includes both measurement and governance. Analytics show what happened. Admin and operations controls help teams keep the workspace reliable, auditable, and commercially correct. Internal teams should avoid presenting analytics as a vanity layer. In practice, it is part of issue handling, campaign iteration, and stakeholder trust.

Analytics scope

- Campaign-level outcome tracking across supported channels.
- Recipient-level delivery and status details where channel support exists.
- Audience and engagement visibility that can feed retention or segmentation logic.
- Operational usage signals that help support teams trace what happened.

Admin scope

- Inspect users, subscriptions, sender readiness, pricing configuration, and wallet state.
- Control team access and permission boundaries.
- Troubleshoot delivery, execution, and account-level configuration issues.
- Review whether operational signals line up with the commercial model and entitlement state.

One of the most useful internal habits is to separate “execution accepted” from “business outcome confirmed.” A campaign may be accepted into the send path, a provider may acknowledge receipt, and a callback may later refine the final status. Analytics and operations views are what allow those distinctions to remain visible instead of collapsing into a misleading success-or-failure simplification.

Operational investigation framework

When something looks wrong, the investigation should follow a structured path instead of jumping straight to provider blame or UI blame. First confirm the audience input. Was the contact or list actually correct. Did the import or sync deliver the expected data. Next confirm the execution object. Was the campaign configured for the intended channel, sender, and template mode. Then confirm the commercial and readiness layer. Was wallet state sufficient. Were senders valid. Were template or media rules satisfied. After that, inspect provider submission and callback evidence. Only then is it reasonable to conclude that the issue lives purely at the provider edge.

This framework matters because many support escalations are really multi-layer issues. A campaign may appear failed when the underlying send path actually succeeded but a later status update, timeout ambiguity, or callback reconciliation path produced a misleading terminal view. Likewise, a campaign may appear sent but still target the wrong audience if import or segment logic was wrong earlier in the chain. The analytics and admin story should therefore be taught as evidence assembly, not just as dashboard observation.

Another useful support habit is to classify issues by layer: data preparation, execution setup, commercial readiness, provider submission, or callback reconciliation. Once issues are framed that way, internal teams know which surface to inspect first and which team should be involved. That reduces the time lost when everyone is looking at the wrong layer.

14. Wallets, Pricing, Plans, and Entitlements

The platform uses both subscription packaging and usage charging. These need to be explained separately because they answer different questions. Plans and entitlements answer what the account is allowed to do. Wallets and channel pricing answer what execution costs in practice.

Plans and entitlements

Plans define what the workspace can access and how much capacity it has across forms, templates, senders, segments, integrations, rewards, and lifecycle functionality. Entitlements are the more precise allowances inside that plan model. When internal teams discuss packaging, they should speak in terms of access, limits, and controlled availability rather than only in terms of monthly subscription labels.

Wallet and channel pricing

Wallets and ledgers track available spend and channel usage. Pricing answers what a specific communication action costs. This is especially important for SMS, WhatsApp, and other usage-driven channels where delivery is not merely a feature toggle but an expenditure event. The wallet area should be described as part of execution control, not just as a finance screen.

Concept	What it answers	Why teams should care
Plan	What features and limits does this account get?	Shapes the overall product surface available to the customer.
Entitlement	What exact resource or capability is allowed?	Clarifies fine-grained limits and reduces packaging ambiguity.
Wallet	What available value can be spent on communication activity?	Directly affects send eligibility for usage-based channels.
Channel pricing	What does each communication action cost?	Keeps go-live planning realistic and helps explain spend behavior.

A useful internal framing is that commercial logic sits inside the operating path rather than after it. If a send depends on available balance, pricing, or entitlement state, commercial setup is part of delivery readiness. This prevents late-stage surprises where a campaign is fully configured but not economically executable.

Packaging and pricing conversations

Commercial conversations usually break down when teams mix feature access and usage cost into one vague promise. A customer may have access to a channel in principle but still need wallet funding to use it at scale. Another customer may have balance available but lack the entitlement or configuration to activate the workflow they want. That is why plans, entitlements, pricing, and wallet state should always be described as related but separate layers of readiness.

There is also an internal sales and onboarding implication. If the customer expects a high-volume operational program, teams should discuss wallet behavior and channel pricing early rather than after technical setup is complete. If the customer expects complex automation, integrations, or rewards, teams should confirm that the plan and entitlement model support that scope before configuration begins. Commercial clarity protects implementation speed because it removes ambiguity from what the account is actually allowed to do once the build is finished.

15. Integrations and Syncing

Integrations are how customers connect source systems to communication workflows. In a mature implementation, imports may initialize the audience once, but syncs are what keep the data current. This is why the integration story deserves more depth than a list of provider logos.

Supported provider pattern

The current integration layer includes providers such as Google Sheets, Zoho, HubSpot, and Xero, while the broader codebase also reflects support patterns for additional systems such as Odoo, PostgreSQL, MySQL, MongoDB, Airtable, and Pipedrive. The right internal story is not “we connect to everything.” It is that the platform has a structured integration model for SaaS apps, file-like sources, databases, and operational systems.

Integration category	Examples	What typically syncs	Why it matters
CRM	Zoho, HubSpot, Pipedrive, Odoo	Leads, contacts, owner context, stage data.	Provides sales and lifecycle signal for ongoing targeting.
Sheets and tabular tools	Google Sheets, Airtable	Lists, campaign-ready rows, lightweight operational data.	Supports non-technical teams and low-friction audience refresh.
Accounting and finance	Xero and related finance sources	Billing state, invoice context, finance-linked customer status.	Useful for reminders, collections, and trust-sensitive communication.
Databases	PostgreSQL, MySQL, MongoDB	Customer records, events, and internal business context.	Provides high-trust source data for lifecycle and transactional messaging.

OAuth, configuration, and mapping

Several integrations use OAuth start and callback flows. Others rely on structured configuration and field mapping. The platform also exposes capabilities information so operators can understand provider support and mapping expectations before launching a sync. This is critical because integrations fail less often from authentication alone than from poor field mapping or unrealistic assumptions about source structure.

Mappings typically focus on identity fields such as name, email, and telephone, then attach remaining fields to attributes. Targets can point to existing lists or to new list creation behavior, and some sync paths can also feed lifecycle-source behavior. The implication is that a sync is not just a data pull. It is a contract between source fields, contact identity rules, and downstream audience operations.

Sync run behavior

The sync flow creates tracked run records with status, total rows, created rows, updated rows, skipped rows, error rows, and metadata. That is exactly the level of detail implementation teams need. A sync should never be described as just “connected.” A synced system is only operationally useful when teams can answer whether the last run completed, what changed, what was skipped, and whether any field mapping needs correction.

The integration center is therefore one of the most operationally important surfaces in the product. It is where accounts connect source systems, review sync readiness, and understand whether lifecycle or campaign programs are consuming fresh data or stale data.

How integrations relate to imports

Imports and syncs solve related but different problems. Imports are excellent for bootstrapping, one-time migration, or ad hoc audience upload. Syncs are better for continuing alignment with a living source system. Many mature accounts use both: an import to get started quickly, then a sync to stop relying on manual refresh. Internal teams should explain that evolution clearly so customers do not mistake a successful first import for a durable data strategy.

Choosing between import, sync, webhook, and API

Pattern	Best when	Main strength	Main limitation
Manual import	The customer needs speed and has a static dataset.	Fastest path to initial audience readiness.	Becomes brittle if used as the permanent system of record.
Scheduled or recurring sync	The customer has an upstream system with changing audience data.	Keeps contact state and lifecycle context fresh over time.	Requires mapping discipline and monitoring.
API-driven writes	An application wants direct control over contact or campaign creation.	High control and low manual overhead.	Requires engineering effort and contract stability.
Webhook-driven follow-up	Another system needs to react after a Yournotify event occurs.	Asynchronous, event-based coordination.	Not a replacement for initial data loading.

Sync maturity and lifecycle impact

Teams often underestimate how much sync quality affects lifecycle messaging. If lifecycle stages, scores, or business-state indicators are derived from synced data, then stale syncs create stale journeys. A retention campaign may look correct in the editor but still target the wrong people if the upstream source stopped refreshing or a field mapping changed quietly. This is why sync monitoring belongs in the operational playbook, not only in engineering notes.

A mature implementation therefore asks four questions about every integration. Is the connection healthy. Is the mapping still correct. Are the results producing the expected created, updated, skipped, and error patterns. And are downstream campaigns or automations consuming the synced fields the way the business expects. When those questions are visible, the integration layer becomes a reliable foundation instead of a hidden risk.

16. Rollout Checklist

This section is meant for onboarding, implementation, support, and product operations teams. It should function as a real readiness checklist rather than a ceremonial appendix.

Initial rollout checklist

1. Confirm the primary use case: promotional, transactional, lifecycle, lead generation, or integration-driven.
2. Confirm the channel mix and sender readiness for each channel in scope.
3. Confirm whether templates are local, approved WhatsApp templates, or direct-message driven.
4. Confirm audience source: import, sync, form capture, or direct API payload.
5. Confirm how custom fields will map into contact attribs and whether any reachability data needs normalization.
6. Confirm commercial setup: plan, entitlements, wallet, and expected usage pricing.
7. Confirm reporting needs, webhook needs, and ownership for issue handling.
8. Confirm whether APIs, SMTP, SMPP, or sync callbacks are part of scope.

Go-live checklist

1. Run test sends for every channel in scope.
2. Validate sender identity, provider readiness, and domain or number setup.
3. Validate template variables, approved WhatsApp components, and rendered message output.
4. Validate import or sync freshness and confirm created, updated, and skipped counts are understood.
5. Validate analytics visibility, callback receipt, and delivery evidence paths.
6. Validate wallet or pricing assumptions for usage-based channels before scaling volume.
7. Validate escalation steps for failed delivery, blocked spend, sync errors, or mapping mistakes.

What not to skip

Do not skip field mapping review. Do not skip sender verification. Do not skip wallet checks for usage-based channels. Do not skip the distinction between import success and business-quality audience readiness. Those are the most common causes of rollout pain because they look secondary until the first live send or sync goes wrong.

First 30 days operating cadence

The first month after launch usually determines whether the account becomes deeply adopted or remains a shallow single-use deployment. Week one should focus on confirming sender readiness, audience shape, and the first campaign or transactional path. Week two should focus on validating result quality, not only send success. Week three should focus on reducing manual steps through better templates, list discipline, or initial automation. Week four should focus on identifying which import, sync, webhook, or lifecycle improvements turn the early win into a repeatable operating model.

This cadence matters because many accounts are declared “live” before the system is actually stable. A send may have gone out, but the team may still lack mapping discipline, monitoring, or a shared operating model. The purpose of the handbook is to help internal teams see go-live as the start of a quality loop rather than the end of setup.

Maturity progression after go-live

After the first month, most healthy accounts move through a recognizable progression. First they stabilize the initial channel and workflow. Then they reduce repetitive work through reusable templates, clearer audience ownership, and better naming. After that they start trusting the product with more timing-sensitive or higher-volume work, which usually forces better attention to wallet policy, sender configuration, and sync freshness. Finally, the more mature accounts begin to treat the platform as a coordinated operating layer with lifecycle state, automation logic, webhook feedback, and cross-team ownership.

Internal teams should actively look for signs that the account is ready for the next maturity step. Repeated manual imports may mean it is time for sync. Frequent copy duplication may mean it is time for a stronger template library. Generic list sends may mean the account is ready for segments and lifecycle-driven targeting. A flood of support questions about status may mean the account needs clearer callback handling or better operator training. Those are not merely support issues; they are product-maturity signals.

The handbook should therefore be used not only during initial onboarding but also during account expansion reviews. It helps teams identify where the customer is operating today and what the next operational improvement should be. That turns the handbook into a working tool rather than a static orientation document.

---

17. Glossary

Term	Meaning in Yournotify
Campaign	An execution object that combines content, audience, sender, timing, and channel settings.
Template	A reusable content asset used to prefill campaign or direct-send content.
Contact	A single audience record with identifying and channel reachability data.
List	A reusable grouping of contacts, often used as an import or sync destination.
Segment	A dynamic audience subset based on conditions or rules.
Lifecycle	A model of customer state and timing derived from source data and stage logic.
Entitlement	A formal feature or capacity allowance tied to plan logic.
Wallet	A value store used to fund usage-based communication activity.
Webhook	An event notification that either leaves the platform to another system or enters it from a provider.
SMTP	Mail submission protocol used by email-oriented systems.
SMPP	Telecom-oriented protocol for SMS submission.
Sync run	A tracked integration execution with counts and status metadata.

18. FAQ and Internal Notes

Frequently asked questions

- **Is Yournotify only for marketing?** No. It supports marketing, transactional, operational, lifecycle, lead generation, and developer-driven communication.
- **Why include attribs in the handbook?** Because many real implementations depend on custom data mapping, alternate identifiers, and operational linkage that live in attribute fields.
- **Why explain imports and syncs in so much detail?** Because audience quality determines campaign quality, lifecycle correctness, and the reliability of personalized or transactional delivery.
- **Why include protocol references like SMTP and SMPP?** Because technical buyers and enterprise partners need to know the product can fit different integration maturities.
- **Why is there so much emphasis on data movement?** Because imports, syncs, and attribute mapping are what make targeting, personalization, lifecycle logic, and reporting trustworthy.

Internal positioning notes

- Do not position the product as only a campaign tool when the conversation is about operational or transactional messaging.
- Do not position the product as only a developer API when the customer needs non-technical team workflows too.
- Do not treat imports as setup-only and syncs as optional; for many accounts, data movement is central to the value story.
- The strongest product story is the combination of audience, reusable content, multi-channel delivery, workflow control, flexible attributes, and commercial structure.

The most useful internal test is simple. If a new team member reads this guide and can explain how a contact enters the system, how data gets normalized, how templates and audience are combined, how commercial state affects execution, how delivery evidence returns, and how the account should mature after go-live, then the handbook is doing its job. If the reader can only repeat navigation labels, the guide is still too shallow. The product story should always connect business outcome, data model, execution flow, and operating ownership in one explanation. That standard should hold during onboarding, support escalation, renewal planning, and every implementation review.