INTEGRATION GUIDE

Sayl CRM

Configure, deploy and connect every channel —
website, Shopify, WordPress, WhatsApp, Telegram, SMS, email and more.

Version 1.0 · For platform operators, IT administrators and integrators

Table of contents

1. Before you start
2. How channels work in Sayl CRM
3. Server-side configuration (env vars & feature flags)
4. Workspace setup & inboxes
5. WhatsApp (Meta Cloud API)
6. Web Chat — embed on any website
7. Shopify integration
8. WordPress / WooCommerce integration
9. Telegram
10. Facebook Messenger
11. Instagram (DMs)
12. Email (Mailgun inbound + SMTP outbound)
13. SMS (Twilio)
14. Outgoing webhooks — push events to your systems
15. Bot flows, AI & broadcasts (quick tour)
16. Going live: pre-flight checklist
17. Troubleshooting

1. Before you start

This guide assumes you already have a Sayl CRM instance running. If you don't, the platform is a Spring Boot application that requires:

- **Java 21** and **Maven 3.8+** to build the JAR.
- **MongoDB** 6.0+ (single node for dev, replica set for production).
- An **HTTPS-terminating reverse proxy** in front of the app (Nginx, Caddy, AWS ALB, Cloudflare). Every channel webhook must reach a public HTTPS URL — most providers refuse plain HTTP.
- Outbound internet access from the app server to the channel APIs (Meta Graph, Telegram, Twilio, Mailgun, etc.).

You'll need **three URLs** handy throughout this guide. Replace them everywhere you see the placeholders below:

Placeholder	What it is	Example
`{HOST}`	Public HTTPS host of your Sayl CRM instance	`https://crm.example.com`
`{LINE_ID}`	The MongoDB `_id` of a messaging line, shown in the URL bar when you open the line in the dashboard	`652f1c9a4e2b8d0001abcdef`
`{LINE_KEY}`	The public, opaque key for a Web Chat line — visible in the channel settings page	`wc_8a3f...e91`

> **Webhooks need to be reachable.**
>
> If you're testing locally, expose your dev port (`8081`) using a tunnel like `ngrok` , `cloudflared` or `tailscale funnel` . Channel providers will not deliver to `localhost` .

2. How channels work in Sayl CRM

Sayl CRM uses a **compile-time adapter pattern**. Every channel — WhatsApp, Telegram, SMS, Email, etc. — plugs into the same internal pipeline:

1. **Inbound webhook** — the channel provider POSTs an event (a new message, a delivery receipt, etc.) to a Sayl URL. Each channel has its own URL but the same shape.
2. **Channel-agnostic event** — the webhook controller validates the payload (HMAC signature or shared secret), then translates it into a generic `InboundEvent` object.
3. **Inbound message processor** — one shared service routes the event to the correct workspace, conversation and agent. Tenant isolation, brute-force protection, audit logging and SLA tracking apply uniformly across channels.
4. **Outbound adapter** — when an agent replies in the inbox, the registry dispatches the message back through the channel-specific outbound client.

Practically, this means: **once one channel works, the rest of the product (inbox, AI suggestions, broadcasts, reports, exports) works identically for every other channel.** You configure each channel once, then forget about

it.

The two-layer kill-switch

Each channel is gated by two independent toggles. Both must be ON for the channel to be usable:

Layer	Where	What it does
Platform feature flag	`application.yml` or env var (`SAYL_CHANNELS_TELEGRAM=false`)	Disables the channel for **all workspaces**. Use this for staged rollouts or to quickly take a misbehaving channel offline.
Per-workspace toggle	Super-admin: `/admin/workspaces/{id}/channels`	Decides which channels each customer can use, regardless of plan.

3. Server-side configuration

Set these as environment variables (preferred) or in `application.yml`. Any secret should come from a secret manager (AWS Secrets Manager, GCP Secret Manager, HashiCorp Vault), never from a file checked into git.

Mandatory before going live

Variable	What it's for
`SAYL_ENCRYPTION_KEY`	32-byte AES-256-GCM key. Encrypts every channel token at rest. **Generate a fresh one** with `openssl rand -base64 32` and rotate it via the platform's key-rotation runbook.
`SPRING_DATA_MONGODB_URI`	MongoDB connection string. Use a replica set with auth in production.
`META_APP_SECRET`	App secret from your Meta App. Used to verify HMAC-SHA256 signatures on every inbound WhatsApp / Messenger / Instagram event. Without this, webhooks are silently rejected.
`META_WEBHOOK_VERIFY_TOKEN`	Random string you pick yourself. Meta echoes it during the webhook subscription handshake.
`LEMONSQUEEZY_WEBHOOK_SECRET`	HMAC secret for billing webhooks. Required if you take card payments via LemonSqueezy.
`SAYL_INITIAL_ADMIN_PASSWORD`	Bootstrap password for the platform super admin. If unset a random one is printed once on first boot — copy it from stdout.

Channel feature flags (kill-switches)

All default to `true`. Set any to `false` to disable a channel platform-wide:

```
SAYL_CHANNELS_WHATSAPP=true
SAYL_CHANNELS_WEB_CHAT=true
SAYL_CHANNELS_TELEGRAM=true
SAYL_CHANNELS_FACEBOOK_MESSENGER=true
SAYL_CHANNELS_INSTAGRAM=true
SAYL_CHANNELS_EMAIL=true
SAYL_CHANNELS_SMS=true
```

Optional, per-feature

Variable	Purpose
`OPENAI_API_KEY` · `ANTHROPIC_API_KEY` · `AZURE_OPENAI_API_KEY`	Platform-fallback AI keys. Workspaces using *Bring-Your-Own-AI* override these per workspace.
`MAIL_HOST` · `MAIL_PORT` · `MAIL_USERNAME` · `MAIL_PASSWORD`	Outgoing SMTP for invitations, OTPs, trial-warning emails. Required for the email channel's outbound adapter as well.

Variable	Purpose
`SAYL_OTP_PRINT_TO_CONSOLE`	Set to `true` only in dev to print OTPs to stdout. Default is `false` and that's correct for production.

Cookie security.

In `application.yml`, `cookie.secure` defaults to `false` for local development. Behind HTTPS in production, set it to `true` — otherwise session cookies leak over HTTP if a user ever lands on an HTTP URL.

4. Workspace setup & inboxes

Sayl is multi-tenant. Every channel belongs to a **workspace** via an **inbox** via a **line**:

```
Workspace "Acme"
  └── Inbox "Sales" (Shared / multi-channel)
        ├── Line: WhatsApp · +20 100 555 0001
        ├── Line: Telegram · @acme_sales_bot
        └── Line: Web Chat · acme.com widget
  └── Inbox "Support" (Dedicated / English-only)
        └── Line: Email · support@acme.com
```

Concepts:

- **Workspace** — your tenant. Members, billing, subscription, audit log all live at this scope.
- **Inbox** — a virtual mailroom. Choose a routing strategy (*Dedicated* = manually assigned, *Shared* = round-robin with language match) and a list of agents.
- **Line** — a single connected channel: one WhatsApp number, one Telegram bot, one widget, one email mailbox. Multiple lines per inbox are fine.

Setup order (do this once per workspace):

1. Sign in to `{HOST}/login` as the workspace owner.
2. Open **Settings** → **Inboxes** and create at least one inbox.
3. Open **Settings** → **Team** and invite agents. Assign them to the inbox(es) they should service.
4. Open **Settings** → **Channels** → **⟨channel of choice⟩** and follow that channel's section in this guide.

5. WhatsApp (Meta Cloud API)

Inbound webhook `POST {HOST}/webhook`
Outbound API Meta Graph v17+
Settings page `/settings/channels/whatsapp`
Auth HMAC-SHA256 via `X-Hub-Signature-256`

> **Two onboarding paths.**
>
> Sayl CRM supports two ways to bring a WhatsApp number online:
>
> - **Dedicated**
>
> — you own the Meta App and WhatsApp Business Account; you provide the Phone Number ID + permanent access token. Maximum control. Sections 5.1–5.3 below.
> - **Shared**
>
> — the platform operator adds your number to *Sayl's* WABA. You don't need a Meta account and never see credentials, but daily caps and stricter compliance gates apply. Section 5.4.

5.1 What you need from Meta (Dedicated)

Sign in to business.facebook.com and complete WhatsApp Business Platform onboarding. You will end up with:

- A **WhatsApp Business Account ID (WABA)**.
- A **Phone Number ID** per phone you add (this is the routing key — *not* the phone number itself).
- A **System User Access Token** with `whatsapp_business_messaging` + `whatsapp_business_management` permissions. Make it permanent, not a 60-day token.
- The Meta App's **App Secret** — this is the platform-level `META_APP_SECRET` you set in section 3, shared across all WhatsApp lines on this Sayl instance.

5.2 Connect the line in Sayl (Dedicated)

1. In Sayl, open **Settings** → **Channels** → **WhatsApp** and click *Add Line*. Keep the default *Use my own Meta App* tab selected.
2. Fill in *Display name*, *Phone number*, *Phone Number ID* and *Permanent Access Token*. Pick the inbox to connect this line to.
3. Sayl encrypts the token using `SAYL_ENCRYPTION_KEY` and stores it. The token is never written to logs.
4. Click *Test send* to confirm outbound works — Sayl will send a small `hello_world` template to a number you specify.

5.3 Configure the Meta webhook (Dedicated)

1. In Meta App Dashboard → **WhatsApp** → **Configuration** → **Webhooks**, set:
   - **Callback URL:** `{HOST}/webhook`
   - **Verify token:** the value you set as `META_WEBHOOK_VERIFY_TOKEN`
2. Subscribe the WABA to the following fields: `messages`, `message_status`, `message_template_status_update`.
3. Click *Verify and Save*. Meta will issue a GET to your URL with a challenge — Sayl echoes it if the token matches.
4. Send a real WhatsApp message from your phone to the connected number. It should land in the Sayl inbox within a couple of seconds.

> **Templates.**
>
> Outbound messages outside the 24-hour customer service window must use a Meta-approved template. Sayl does *not* auto-submit templates — you create them in WhatsApp Manager, wait for approval, then they appear in Sayl's *Broadcasts* → *Templates* picker.

5.4 Shared mode — request a number on Sayl's WABA

If you don't have your own Meta App and don't want to manage WABA / system-user tokens, request a Shared-mode line instead. The customer-facing flow is short:

1. In Sayl, open **Settings** → **Channels** → **WhatsApp** → **Add Line** and switch to the *Use Sayl's shared infrastructure* tab.
2. Provide the phone number you want to use (in E.164, e.g. `+201001234567`), your business name, the inbox, and an optional note for the platform operator. Submit.
3. The line is created immediately with provisioning status **PENDING_REVIEW**. It shows up in your Lines list with a *Pending review* badge. All sends from this line are blocked by the compliance guard until approval.
4. Your platform operator reviews the request under `/admin/whatsapp-lines`, adds your number to Sayl's WABA in Meta Business Manager, then approves with the resulting Phone Number ID + system-user token. The line flips to **APPROVED + ACTIVE** and is sendable.
5. If the operator rejects the request, the rejection note is shown on the line card. Address the issue (most commonly: ownership proof, business name match), delete the rejected request, and resubmit.

> **Compliance gates that apply only to Shared lines.**
>
> Because shared traffic flows through the platform's single Meta App, one bad customer's behaviour drags every other customer's quality rating down. The platform enforces the following extra rules on shared-mode lines (configurable per-deployment via `sayl.compliance.*` properties):
>
> - **Graduated daily broadcast cap** — 50 recipients/day in week 1, 250 in week 2, 1000 from day 21 onwards.
> - **Per-recipient marketing-template velocity cap** — at most 3 marketing templates to the same recipient per rolling 7-day window, platform-wide.
> - **KYB business verification** required before sending — guided flow under *Settings* → *Compliance*.
> - **AUP acceptance** required from the workspace owner — also under *Settings* → *Compliance*.
> - **Quality-rating freeze** — broadcasts blocked when Meta drops the line to RED until quality recovers.
> - **Auto-pause on Meta failure codes** — line is paused on codes 131045 (template paused), 131031 (account locked), or ≥5×{131048, 131049, 368} in 1 hour.

> **For platform operators.**
>
> The shared-line review queue lives at `/admin/whatsapp-lines` with a pending-count badge in the admin sidebar. Approval is a 2-field form: *Phone Number ID* + *Access Token* (both obtained from Meta after adding the customer's number to Sayl's WABA). Rejection requires a note that's surfaced verbatim to the customer. Audit actions emitted: `SHARED_LINE_REQUESTED`, `SHARED_LINE_APPROVED`, `SHARED_LINE_REJECTED`, `SHARED_LINE_REQUEST_CANCELLED`.

6. Web Chat — embed on any website

Inbound `POST {HOST}/chat/{LINE_KEY}/message` **Streaming** Server-Sent Events

Settings page `/settings/channels/web-chat` **Auth** Origin allowlist + signed visitor cookie

The web-chat widget is a small JavaScript file served by your Sayl instance. It opens a bubble on any page, persists visitor identity in a signed cookie and streams agent replies live over SSE.

6.1 Create the line

1. Open **Settings** → **Channels** → **Web Chat** and click *New widget*.
2. Fill in:
   - *Widget name* — internal label.
   - *Primary color* — hex code, used for the bubble and header.
   - *Welcome message* — first thing visitors see.
   - *Allowed origins* — one origin per line, exactly as the browser will send it: `https://acme.com`, `https://www.acme.com`. **No trailing slash, no path.** Requests from any other origin return 403.
3. Save. Sayl shows a generated `{LINE_KEY}` and a ready-made `<script>` snippet.

6.2 Embed the snippet

Paste this just before `</body>` on every page where you want the widget:

```
<script src="{HOST}/js/sayl-chat-widget.js"></script>
<script>
  SaylChat.init({
    lineKey: "{LINE_KEY}",
    primaryColor: "#0D9488"
  });
</script>
```

That's it. The widget will:

- Drop a signed `sayl_visitor` cookie so returning visitors keep their conversation history.
- Create a `Contact` in the workspace with `phone = "webchat:<token>"` on the first message.
- Stream agent replies in real time via SSE, no polling.
- Rate-limit visitors to 60 messages/minute using in-process Bucket4j to stop abuse.

6.3 Customizing

Need	Where
Change colors / welcome text after launch	Edit the line in **Settings → Channels → Web Chat**. Changes take effect on the visitor's next page load — no need to update the snippet.
Hide the widget on certain pages	Don't include the snippet on those pages, or wrap `SaylChat.init` in a conditional check.
Pass logged-in user identity	Currently the widget is anonymous-by-design. Visitor → contact mapping is a roadmap item.

7. Shopify integration

Sayl does **not** ship a native Shopify app today. You integrate by embedding the Web Chat widget into your Shopify theme, which gives you a chat bubble on the storefront with conversations flowing back into Sayl.

7.1 Add the widget to your theme

1. Create a Web Chat line as described in section 6.1. In *Allowed origins*, list both:
   - `https://your-store.myshopify.com`
   - `https://your-custom-domain.com` (if you've connected one)
2. In Shopify admin, go to **Online Store → Themes → ⋯ → Edit code**.
3. Open `layout/theme.liquid` and find the `</body>` tag.
4. Insert the snippet just before `</body>`:

   ```
   {% if template != 'checkout' %}
     <script src="{HOST}/js/sayl-chat-widget.js"></script>
     <script>
       SaylChat.init({ lineKey: "{LINE_KEY}", primaryColor: "#0D9488" });
     </script>
   {% endif %}
   ```

   The `template != 'checkout'` guard prevents Shopify from rejecting the script during checkout, where third-party scripts are restricted.
5. Save. Open the storefront in an incognito window, click the bubble, send a test message — it should appear in your Sayl inbox.

7.2 Pre-fill customer context (optional)

If a customer is logged into Shopify, you can pass their order number into the first message. Add this *before* calling `SaylChat.init`:

```
<script>
  {% if customer %}
    window.__sayl_customer_meta = {
      name: {{ customer.name | json }},
      email: {{ customer.email | json }},
      lastOrder: {{ customer.last_order.name | json }}
    };
  {% endif %}
</script>
```

Agents can then ask about `__sayl_customer_meta` in the message body — full structured pass-through is on the roadmap.

7.3 Order & abandoned cart notifications via outgoing webhook

Use Shopify's **Notifications** → **Webhooks** to forward order events to a small connector you operate. From there, send a follow-up WhatsApp message via Sayl's outgoing webhook (section 14) or via the Broadcasts UI. Direct Shopify → Sayl ingestion is on the roadmap; for now this two-hop pattern works reliably.

8. WordPress / WooCommerce integration

As with Shopify, the WordPress integration is the Web Chat widget snippet. There is no Sayl plugin in the WordPress directory yet.

8.1 The simplest path: WPCode (or any header/footer plugin)

1. Create a Web Chat line (section 6.1). In *Allowed origins*, list the WordPress site's exact origin — `https://example.com` and the `www` variant if you use both.
2. Install the free WPCode — Insert Headers and Footers plugin (or any equivalent).
3. Open **Code Snippets → Header & Footer**.
4. Paste this in the *Footer* box:

```
<script src="{HOST}/js/sayl-chat-widget.js"></script>
<script>
  SaylChat.init({ lineKey: "{LINE_KEY}", primaryColor: "#0D9488" });
</script>
```

5. Save. Visit the site, click the bubble, send a test message. Done.

8.2 Theme-level alternative

If you'd rather not install a plugin, edit your active theme's `footer.php` and paste the snippet just before `</body>`. Use a **child theme** so updates don't overwrite it.

8.3 WooCommerce store hints

- Customer email and order ID can be passed via the same `__sayl_customer_meta` trick from the Shopify section, populated from the WooCommerce `get_current_user` object.
- For order-status follow-ups, use WP Webhooks or *Code Snippets* to call your Sayl-side outgoing-webhook endpoint when an order changes state.
- Disable the widget on the checkout page if your conversion rate suffers — many themes already do this for third-party scripts.

9. Telegram

Inbound webhook `POST {HOST}/webhook/telegram/{LINE_ID}`
Outbound API Telegram Bot API **Settings page** `/settings/channels/telegram`
Auth `X-Telegram-Bot-Api-Secret-Token` header

9.1 Create a bot with BotFather

1. Open Telegram and message `@BotFather`.
2. Send `/newbot`. Pick a display name, then a username ending in `_bot`.
3. BotFather replies with an **HTTP API token** like `1234567:ABC-DEF...`. Keep it secret.
4. (Optional) Use `/setdescription`, `/setuserpic`, and `/setcommands` to make the bot look professional.

9.2 Connect the line in Sayl

1. Open **Settings → Channels → Telegram → Add Line**.
2. Paste the bot token. Sayl will:
   - Validate the token against `getMe`.
   - Encrypt and store it.
   - Generate a random secret and call `setWebhook` on Telegram, registering `{HOST}/webhook/telegram/{LINE_ID}` with the secret in `X-Telegram-Bot-Api-Secret-Token`.
3. Send `/start` to your bot from Telegram. The conversation should land in the Sayl inbox immediately.

> **One bot, one Sayl instance.**
> Telegram only supports a single webhook URL per bot. If you previously pointed the bot at a different system, calling `setWebhook` from Sayl overwrites it.

10. Facebook Messenger

Inbound webhook `POST {HOST}/webhook` (object: `page`) **Outbound API** Meta Graph v17+ (Page Send API)
Settings page `/settings/channels/meta-page` **Auth** HMAC-SHA256 (shared with WhatsApp)

Messenger reuses the same Meta App, the same `META_APP_SECRET`, and the same `{HOST}/webhook` URL as WhatsApp. The webhook router demuxes by the `object` field in the payload.

1. In Meta Business Suite, ensure the Page is connected to your Meta App.

2. Generate a long-lived **Page Access Token** for the Page (Graph API Explorer or Business Settings → System Users).
3. In your Meta App → **Messenger** → **Settings** → **Webhooks**, subscribe the Page to: `messages`, `messaging_postbacks`, `message_deliveries`, `message_reads`.
4. The callback URL and verify token are the same as WhatsApp — set them once.
5. In Sayl, **Settings** → **Channels** → **Meta Page** → **Add Line**: paste the Page ID and Page Access Token. Choose the inbox.
6. Have someone DM your Page from Facebook — it should appear in Sayl.

11. Instagram (DMs)

Inbound webhook `POST {HOST}/webhook` (object: `instagram`)

Outbound API Meta Graph (Instagram Messaging) **Settings page** `/settings/channels/meta-page`

Auth HMAC-SHA256 (shared with WhatsApp)

Instagram Messaging is wired to a Facebook Page that owns the connected Instagram Business account. The Sayl line creation flow is identical to Messenger — same access token shape, same webhook endpoint.

1. Ensure the Instagram account is set to *Business* or *Creator* and is connected to a Facebook Page in **Settings → Accounts Center**.
2. In your Meta App, enable the **Instagram** product. Subscribe the page to: `messages`, `messaging_postbacks`.
3. Toggle *Connected Tools → Allow access to messages* in the Instagram app, otherwise webhooks won't fire.
4. In Sayl, **Settings → Channels → Meta Page → Add Line → Channel = Instagram**. Paste the IG Business account ID and the Page Access Token.
5. Send a DM to the IG account from a different Instagram account — it should appear in Sayl.

> **Limits to be aware of.**
>
> Instagram only delivers messages from accounts that haven't been blocked, and stories/comments are *not* currently routed — only direct messages. Story replies do come through.

12. Email (Mailgun inbound + SMTP outbound)

Inbound webhook `POST {HOST}/webhook/email/mailgun/{LINE_ID}`

Outbound transport SMTP (your existing mail server or a relay) **Settings page** `/settings/channels/email`

Auth Mailgun signature (timestamp + token + HMAC)

Sayl's email channel uses Mailgun's *Routes* for inbound (because parsing MIME at scale is a problem you don't want to solve yourself), and your platform's SMTP credentials for outbound.

12.1 Configure outbound (one-time, platform-level)

Set `MAIL_HOST`, `MAIL_PORT`, `MAIL_USERNAME`, `MAIL_PASSWORD` environment variables (section 3). The same SMTP is used for transactional email and for outbound on the email channel.

12.2 Configure inbound (per line)

1. In Sayl, **Settings** → **Channels** → **Email** → **Add Line**: pick a mailbox address (e.g. `support@acme.com`), pick the inbox, save. Sayl gives you a Mailgun signing key.
2. In Mailgun, add and verify your domain (DNS: SPF + DKIM + MX). Test with `dig MX`.
3. In Mailgun → **Receiving** → **Routes**, create a route:
   - Expression: `match_recipient("support@acme.com")`
   - Action: `forward("{HOST}/webhook/email/mailgun/{LINE_ID}")`
   - Action: `stop()` (so subsequent routes don't double-fire)
4. In Mailgun → **Settings** → **Webhooks**, set the signing secret to the value Sayl generated.
5. Email `support@acme.com` from any external account. The thread should appear in Sayl as a conversation, with attachments preserved.

13. SMS (Twilio)

Inbound webhook `POST {HOST}/webhook/sms/twilio/{LINE_ID}`

Outbound API Twilio Messages API **Settings page** `/settings/channels/sms`

Auth Twilio signature in `X-Twilio-Signature`

13.1 What you need from Twilio

- An **Account SID** and **Auth Token** (top of the Twilio console).
- A purchased phone number — or a Messaging Service SID, which is preferable for high-volume sending because it pools numbers and handles A2P 10DLC compliance for you.

13.2 Connect the line

1. In Sayl, **Settings → Channels → SMS → Add Line**: paste Account SID, Auth Token, and either the From-number or Messaging Service SID. Pick the inbox. Save.
2. Sayl encrypts the Auth Token, validates the credentials with a Twilio `GET /Accounts/{SID}` probe, and shows you the inbound webhook URL.
3. In the Twilio console, open your phone number's settings (or the Messaging Service's *Inbound Settings*):
   - **A message comes in** → Webhook → `{HOST}/webhook/sms/twilio/{LINE_ID}` (HTTP POST)
4. Save. Send an SMS to the number from your phone — it should arrive in Sayl within a few seconds.

> **US/Canada A2P 10DLC.**
>
> Outbound throughput is heavily restricted on plain long-codes in the US/Canada unless your brand and campaign are registered with The Campaign Registry. Use a Messaging Service and complete A2P registration before launching a broadcast.

14. Outgoing webhooks — push events to your systems

Sayl pushes platform events (new conversation, message sent, payment approved, escalation flagged, etc.) to your endpoints over HTTPS, signed with HMAC-SHA256.

1. Open **Settings → Webhooks → Add**.
2. Fill in:
   - *Target URL* — your endpoint, must be HTTPS.
   - *Events* — pick from the catalog (`conversation.created` , `message.delivered` , `broadcast.completed` , etc.).
   - *Secret* — Sayl generates a random one if you don't provide it. Save it.

3 Verify each request server-side with this Node example:

```
import crypto from "node:crypto";

function isValidSaylWebhook(headers, rawBody, secret) {
  const signature = headers["x-sayl-signature"];
  const expected  = crypto.createHmac("sha256", secret)
                          .update(rawBody)
                          .digest("hex");
  return crypto.timingSafeEqual(
    Buffer.from(signature),
    Buffer.from(expected)
  );
}
```

4 Sayl retries failed deliveries with exponential backoff (1s, 5s, 25s, 125s, 625s) and surfaces failures in the webhook detail page.

15. Bot flows, AI & broadcasts (quick tour)

Once any channel is connected, the rest of the platform applies uniformly. The high-level controls:

Capability	Where	Notes
Bot flows	`Settings → Bot Flows`	Decision-tree builder. Triggers: *FIRST_MESSAGE*, *KEYWORD_MATCH*, *OUTSIDE_HOURS*, *NO_AGENT_AVAILABLE*. Works on every channel — channel-aware media (e.g. WhatsApp templates) is selected automatically.
AI assist	`Settings → AI`	Reply suggestions (3 in conversation language), translation, voice transcription, sentiment, summarization. *Bring-your-own-AI* per workspace (OpenAI / Anthropic / Gemini / Azure OpenAI) with monthly token budget; falls back to platform keys when budget is exceeded.
Broadcasts	`Broadcasts → New campaign`	Pre-approved templates → tag/pipeline-stage filtered targets. Bucket4j-rate-limited at 80 msg/s globally to stay under WhatsApp's send limits. Live SSE stats per campaign.
SLA tracking	`Settings → SLA`	Per-inbox first-response and resolution targets. Breaches push an SSE alert to managers and increment a Prometheus counter you can scrape.
Reports & exports	`Reports`	Per-agent CSAT, SLA compliance, conversation metrics. Export as CSV, JSON or PDF (iText 7 + Noto Naskh Arabic for bidirectional PDFs).

16. Going live: pre-flight checklist

Run through this list before pointing real customer traffic at the platform. None of these checks are optional in production.

1. **Set `SAYL_ENCRYPTION_KEY` to a real 32-byte key** from a secret manager. Without this every channel token at rest is reversible.
2. **Set `META_APP_SECRET` and `META_WEBHOOK_VERIFY_TOKEN`**. Without these, every Meta inbound is silently rejected.
3. **Rotate the auto-generated initial admin password**. It's printed once on first boot — change it immediately.
4. **Set `cookie.secure: true`** in `application.yml` for the production profile.
5. **Enable the OWASP Dependency-Check** in CI by running `mvn verify` *without* `-Dowasp.skip=true`.
6. **HTTPS termination + HSTS preload**. The HSTS header is already set by the app — back it with a real TLS terminator (Nginx / ALB / Cloudflare).
7. **MongoDB replica set + backups + auth**. The default `docker-compose.yml` is a single-node, no-auth dev setup. Use Atlas or a 3-node replica set with rotated credentials in production.
8. **Log shipping**. Pair the built-in `RequestLoggingAspect` with Loki / CloudWatch / similar. Surface correlation IDs on error pages.
9. **Test a real message on every channel you've enabled.** The 11 mandatory integration tests pass in CI but they don't exercise *your* credentials.
10. **Force a 6th failed login** on a test account. Confirm the 15-minute lockout fires and an audit-log entry is written.

17. Troubleshooting

Webhooks aren't arriving

Symptom	Likely cause	What to check
Provider says *Delivered*, Sayl shows nothing	HMAC mismatch — signature header doesn't match `META_APP_SECRET` / Twilio Auth Token / Mailgun signing key	App logs at `WARN` level for `WebhookSignatureVerifier`. Re-paste the secret.
Provider returns *Could not verify webhook*	Verify token mismatch on the GET handshake	Make sure `META_WEBHOOK_VERIFY_TOKEN` in env matches the value pasted in Meta's webhook config.
Telegram says *setWebhook returned 401*	Bot token wrong, or your reverse proxy stripped the secret header	Telegram requires the full `X-Telegram-Bot-Api-Secret-Token` to reach the app.
Mailgun route fires but body is empty	Reverse proxy buffer too small for MIME parts	Increase Nginx `client_max_body_size` to at least `30m`.

Outbound messages fail

Symptom	Likely cause	What to check
WhatsApp returns `(#131047) Re-engagement message`	You're trying to send a free-form message outside the 24-hour window	Use a pre-approved template instead.
Twilio returns `21610`	Recipient has opted out by replying *STOP*	Either ask them to reply *START*, or stop sending. This is required by carrier compliance.
SMTP `421 Try again later`	Rate limit on your relay	Throttle outbound or upgrade the SMTP plan. Sayl's email outbound has no built-in retry queue today.
Web Chat widget loads but agent reply never arrives	EventSource (SSE) blocked by intermediate proxy	Set `X-Accel-Buffering: no` and disable response buffering in your reverse proxy for `/chat/*/stream`.

The widget doesn't appear at all on Shopify / WordPress

- Check the browser console — most likely **403 Forbidden on `/chat/{LINE_KEY}/init`**. The origin you're embedding from isn't in the line's *Allowed origins*. Add it (exact host + scheme, no path) and reload.
- Some WordPress themes minify and defer scripts in a way that breaks `SaylChat.init` being called before the `<script src=...>` finishes loading. Wrap the init in `window.addEventListener("load", ...)` if so.
- On Shopify checkout pages, third-party scripts are blocked — the `{% if template != 'checkout' %}` guard from section 7.1 is required.

Need help? Open `/admin/diagnostics` as a super admin to see live platform health.
For everything else, the source spec lives at `specs/001-sayl-crm-platform/` in your Sayl repo.