WhatsApp for Partners

- Partner API and Embed Integration Guide
- WhatsApp API Quick Start for Developers

Partner API and Embed Integration Guide

Agent instruction	Copy

```
Implement ChatArchitect WhatsApp Partner embedded
registration in this codebase following the official guide:
https://support.chatarchitect.com/books/whatsapp-for-
partners/page/partner-api-and-embed-integration-guide.
First inspect the existing frontend/backend/env/storage
patterns, then add the backend session creation endpoint,
frontend iframe flow with postMessage handling and signed
webhook verification/storage.
Keep all Partner API calls server-side, never expose
partner token to browser code.
```

This guide explains how a partner backend creates a WhatsApp registration session, embeds the registration UI in an iframe, receives completion events, and retrieves final app credentials.

Overview

The integration is backend-first:

1. Your backend calls the Partner API to create a registration session.
2. The API returns a `registration_id` and an `iframe_url`.
3. Your frontend renders the `iframe_url` in an iframe.

4. The user completes phone verification, the form, and Meta Embedded Signup inside the iframe.
5. On completion, ChatArchitect sends a signed webhook to your backend.
6. Your backend stores the returned `app_id` and `app_secret`.
7. If webhook delivery fails or you need a fallback, your backend can call the result endpoint.

The iframe never exposes `app_secret` to the browser, the DOM, URLs, or `postMessage`. Final credentials are delivered only to your backend through the webhook or the Partner Result API.

Base URL

Use the public base URL provided by ChatArchitect for your environment:

```
https://<chatarchitect-registration-domain>
```

All Partner API endpoints in this guide are relative to that base URL.

Authentication

All Partner API requests require a Bearer token:

```
Authorization: Bearer <partner_token>
```

ChatArchitect provides this token to you. Treat it as a secret. The same value is also used as the HMAC secret for webhook signature verification.

Create Embed Session

Create a new registration session from your backend.

```
POST /partner?action=create_session
Authorization: Bearer <partner_token>
Content-Type: application/json
```

Request body:

```
{
  "parent_origin": "https://partner.example",
```

```
  "webhook_url": "https://partner.example/api/chatarchitect/whatsapp-registration-webhook",
  "partner_user_id": "user_456",
  "partner_state": "opaque-state-value",
  "integration": "My CRM",
  "lang": "en"
}
```

Fields:

Field	Required	Description
`parent_origin`	Yes	Origin of the parent page that will host the iframe, for example `https://partner.example`. Use scheme, host, and optional port only. Must start with `http://` or `https://`. Used as the target origin for iframe `postMessage` events.
`webhook_url`	Yes	HTTPS URL that receives the completion webhook. Must start with `https://`.
`partner_user_id`	No	Your user/account/customer id. Returned unchanged in the webhook.
`partner_state`	No	Opaque value for your own correlation or CSRF/session checks. Returned unchanged in the webhook.
`integration`	No	Optional custom integration name. It can be any string. If omitted or blank, the registration uses `Partner`.
`lang`	No	Initial language code. Defaults to `en`. Supported values: `de`, `en`, `es`, `pt`, `ru`.

Successful response:

```
{
  "registration_id": "abc123def456ghi789jkl0",
  "iframe_url": "https://<chatarchitect-registration-
domain>/?s=abc123def456ghi789jkl0&mode=embed",
  "expires_at": "2026-07-05T10:00:00+00:00",
  "status": "started"
}
```

Notes:

- Store `registration_id` in your backend. You need it for result polling and webhook retry.
- `iframe_url` is safe to send to your frontend.
- The stored session mode is authoritative. Adding `mode=embed` to another URL does not convert a standalone session into an embed session.

Example with `curl`:

```
curl -X POST "https://<chatarchitect-registration-domain>/partner?action=create_session" \
  -H "Authorization: Bearer <partner_token>" \
  -H "Content-Type: application/json" \
  -d '{
    "parent_origin": "https://partner.example",
    "webhook_url": "https://partner.example/api/chatarchitect/whatsapp-registration-webhook",
    "partner_user_id": "user_456",
    "partner_state": "opaque-state-value",
    "integration": "My CRM",
    "lang": "en"
  }'
```

Embed the Iframe

Render the returned `iframe_url` in your frontend.

```
<iframe
  id="ca-whatsapp-registration"
  src="https://<chatarchitect-registration-domain>/?s=abc123def456ghi789jkl0&mode=embed"
  style="width: 100%; height: 720px; border: 0;"
  allow="clipboard-write; encrypted-media; fullscreen"
></iframe>
```

Avoid a fixed `min-height` if the iframe sits inside a modal or another constrained container. The iframe sends `height` events as its content changes, so the parent page can shrink or grow the iframe instead of forcing an extra scrollbar.

If you use the `sandbox` attribute, include enough permissions for the registration flow and Meta Embedded Signup:

```
<iframe
  src="https://<chatarchitect-registration-domain>/?s=abc123def456ghi789jkl0&mode=embed"
  sandbox="allow-scripts allow-forms allow-same-origin allow-popups allow-popups-to-escape-
```

```
sandbox"
></iframe>
```

Avoid placing secrets in the iframe URL. The only required iframe state is already contained in `iframe_url`.

Iframe Events

The iframe sends status-only events to the parent window using `postMessage`.

Message shape:

```
{
  "type": "ca.whatsapp.registration",
  "event": "ready",
  "registration_id": "abc123def456ghi789jkl0"
}
```

Events:

Event	Meaning
`ready`	Iframe loaded and initialized.
`height`	Iframe height changed. Payload can include `height`.
`phone_verified`	User verified the WhatsApp phone number.
`facebook_started`	User clicked Connect WhatsApp and Meta Embedded Signup started.
`completed`	Registration completed. Fetch credentials from your backend storage or Partner Result API.
`failed`	Registration failed. Payload can include `message`.
`cancelled`	User cancelled Meta Embedded Signup.

Parent-page listener example:

```
<script>
  const iframe = document.getElementById('ca-whatsapp-registration');
  const expectedOrigin = 'https://<chatarchitect-registration-domain>';
  const minHeight = 480;
  const maxModalHeight = () => Math.max(minHeight, window.innerHeight - 220);
```

```
  window.addEventListener('message', (event) => {
    if (event.origin !== expectedOrigin) return;

    const data = event.data || {};
    if (data.type !== 'ca.whatsapp.registration') return;

    if (data.event === 'height' && data.height) {
      const contentHeight = Math.max(minHeight, Number(data.height));
      iframe.style.height = `${Math.min(contentHeight, maxModalHeight())}px`;
    }

    if (data.event === 'completed') {
      // Your backend should receive the signed webhook.
      // Optionally call your own backend to refresh registration status.
    }

    if (data.event === 'failed') {
      // Show a retry/support state in your UI.
    }
  });
</script>
```

`postMessage` never contains `app_secret`.

Completion Webhook

When an embed registration completes, ChatArchitect sends a `POST` request to the `webhook_url` from the session creation request.

Payload:

```
{
  "event": "whatsapp.registration.completed",
  "registration_id": "abc123def456ghi789jkl0",
  "partner_user_id": "user_456",
  "partner_state": "opaque-state-value",
  "app_id": "ca_app_123",
  "app_secret": "secret_value",
  "phone": "421233221242",
```

```
  "integration": "My CRM",
  "completed_at": "2026-07-04T10:00:00+00:00"
}
```

Headers:

```
Content-Type: application/json
X-CA-Event-Id: evt_<registration_id>_<attempt>
X-CA-Timestamp: <unix_timestamp>
X-CA-Signature: sha256=<hmac_sha256>
```

Signature formula:

```
sha256=<HMAC_SHA256(X-CA-Timestamp + "." + raw_request_body, partner_token)>
```

Verification requirements:

- Read the raw request body before JSON parsing.
- Reject stale timestamps according to your replay window, for example 5 minutes.
- Compute HMAC SHA-256 using your Partner API Bearer token as the secret.
- Compare the computed signature with `X-CA-Signature` using constant-time comparison.
- Process events idempotently by `X-CA-Event-Id` or `registration_id`.

Node.js verification example:

```
import crypto from 'node:crypto';

function verifyChatArchitectWebhook({ rawBody, timestamp, signature, partnerToken }) {
  const expected = 'sha256=' + crypto
    .createHmac('sha256', partnerToken)
    .update(`${timestamp}.${rawBody}`)
    .digest('hex');

  const expectedBuffer = Buffer.from(expected);
  const signatureBuffer = Buffer.from(signature || '');
  if (expectedBuffer.length !== signatureBuffer.length) {
    return false;
  }

  return crypto.timingSafeEqual(
    expectedBuffer,
    signatureBuffer
```

```
  );
}
```

Respond with any `2xx` status when the webhook is accepted. Non-`2xx` responses are treated as failed delivery and can be retried through the Partner API.

Get Registration Result

Use the result endpoint as a backend fallback or status check.

```
GET /partner?action=result&registration_id=<registration_id>
Authorization: Bearer <partner_token>
```

If registration is not completed yet:

```
{
  "status": "phone_verified"
}
```

Possible in-progress statuses include:

```
started
phone_code_sent
phone_verified
form_completed
fb_started
expired
```

If registration is completed:

```
{
  "status": "completed",
  "registration_id": "abc123def456ghi789jkl0",
  "app_id": "ca_app_123",
  "app_secret": "secret_value",
  "phone": "421233221242",
  "integration": "My CRM",
  "completed_at": "2026-07-04T10:00:00+00:00"
}
```

The result endpoint only works for the partner token that created the session.

Retry Webhook

Retry webhook delivery after a completed registration.

```
POST /partner?action=retry_webhook
Authorization: Bearer <partner_token>
Content-Type: application/json
```

Request body:

```
{
  "registration_id": "abc123def456ghi789jkl0"
}
```

Response:

```
{
  "status": "sent",
  "attempts": 2,
  "last_error": ""
}
```

If the registration is not completed, the endpoint returns `409`.

Error Responses

Common error shape:

```
{
  "status": "error",
  "message": "Invalid registration id"
}
```

Common HTTP statuses:

Status	Meaning

`400`	Invalid action, request body, `parent_origin`, `webhook_url`, or registration id.
`401`	Missing or invalid Bearer token.
`404`	Session not found, or it belongs to another partner.
`409`	Retry requested before registration completion.
`500`	Server-side failure.

Security Checklist

- Call Partner API only from your backend, never from browser JavaScript.
- Keep the Partner API token secret. It is also the webhook signing secret.
- Use HTTPS for your parent page and webhook endpoint in production.
- Verify every webhook signature before trusting `app_id` or `app_secret`.
- Store `app_secret` only in backend storage.
- Treat iframe `postMessage` events as status hints only, not as a source of credentials.
- Validate `event.origin` when receiving iframe messages.
- Handle duplicate webhooks idempotently.

Recommended Flow

```
sequenceDiagram
    participant PartnerBackend as Partner backend
    participant PartnerFrontend as Partner frontend
    participant CA as ChatArchitect registration
    participant User as User

    PartnerBackend->>CA: POST /partner?action=create_session
    CA-->>PartnerBackend: registration_id, iframe_url
    PartnerBackend-->>PartnerFrontend: iframe_url
    PartnerFrontend->>CA: Load iframe_url
    CA-->>PartnerFrontend: postMessage ready/height
    User->>CA: Verify phone, fill form, complete Meta signup
    CA-->>PartnerFrontend: postMessage completed
    CA->>PartnerBackend: Signed completion webhook
    PartnerBackend-->>CA: 2xx accepted
    PartnerBackend->>PartnerBackend: Store app_id and app_secret
```

WhatsApp API Quick Start for Developers

Use this guide to add ChatArchitect WhatsApp messaging to any third-party service. It covers credentials, outbound messages, templates, delivery statuses, inbound messages, and a basic test flow.

Agent instruction	Copy

```
Implement ChatArchitect WhatsApp API connection for this codebase using the official guide:
https://support.chatarchitect.com/books/whatsapp-for-partners/page/whatsapp-api-quick-start-for-developers
Inspect the app’s existing architecture first, then add server-side credential handling, message sending,
template loading, webhook processing for statuses and inbound messages,
and minimal UI/settings needed for users to connect.
```

What you will build

By the end of this guide, your service can:

- Send WhatsApp text messages and approved template messages.
- Receive asynchronous delivery statuses through a webhook: `submitted`, then `failed`, `sent`, `delivered`, or `read`.
- Receive inbound customer messages through the same webhook.
- Fetch approved WhatsApp templates from the ChatArchitect API.

If a customer writes first, you can reply without a template inside the active customer service window. If your business starts the conversation, use an approved WhatsApp template.

Prerequisites and credentials

The customer must connect WhatsApp Business API through ChatArchitect support and Meta. After setup, ChatArchitect provides:

Credential	Purpose
`APP_ID`	Basic Auth username for ChatArchitect API requests.
`APP_SECRET`	Basic Auth password for ChatArchitect API requests.

Store both values server-side. Do not expose `APP_SECRET` in browser code, public logs, client-side configuration, or URLs.

All examples below use:

```
API base URL: https://api.chatarchitect.com
Auth: Basic Auth with <APP_ID>:<APP_SECRET>
Recipient phone: <recipient_phone>
Webhook URL: <webhook_url>
```

Use phone numbers in international format without `+`, spaces, or brackets, for example `421233221242`.

Quick start: 3 API calls

Step 1 - Register a webhook

Register an HTTPS webhook URL for delivery statuses and inbound messages.

```
POST https://api.chatarchitect.com/webhook
Authorization: Basic base64(<APP_ID>:<APP_SECRET>)
Content-Type: application/json
```

Request body:

```
{
  "channel": "whatsapp",
  "destination": "<recipient_phone>",
  "webhook_separate": "false",
```

```
  "webhook": "<webhook_url>"
}
```

`destination` can be any WhatsApp number connected to the account. `webhook` must be an HTTPS URL reachable from the public internet.

cURL example:

```
curl -X POST "https://api.chatarchitect.com/webhook" \
  -u "<APP_ID>:<APP_SECRET>" \
  -H "Content-Type: application/json" \
  -d '{
    "channel": "whatsapp",
    "destination": "<recipient_phone>",
    "webhook_separate": "false",
    "webhook": "<webhook_url>"
  }'
```

Step 2 - Send a text message

Send an outbound WhatsApp message.

```
POST https://api.chatarchitect.com/whatsappmessage
Authorization: Basic base64(<APP_ID>:<APP_SECRET>)
Content-Type: application/json
```

Request body:

```
{
  "channel": "whatsapp",
  "destination": "<recipient_phone>",
  "payload": {
    "type": "text",
    "message": "Hi John, how are you?"
  }
}
```

The synchronous response confirms that the message was accepted into the queue:

```
{
  "status": "submitted",
  "messageId": "21110c1d-53e1-42b5-9454-1de9a09d4777"
}
```

Final delivery status arrives later through your webhook.

cURL example:

```
curl -X POST "https://api.chatarchitect.com/whatsappmessage" \
  -u "<APP_ID>:<APP_SECRET>" \
  -H "Content-Type: application/json" \
  -d '{
    "channel": "whatsapp",
    "destination": "<recipient_phone>",
    "payload": {
      "type": "text",
      "message": "Hi John, how are you?"
    }
  }'
```

Step 3 - Get approved templates

Fetch approved WhatsApp templates available for the connected account.

```
POST https://api.chatarchitect.com/getHSM
Authorization: Basic base64(<APP_ID>:<APP_SECRET>)
Content-Type: application/json
```

Request body:

```
{
  "channel": "whatsapp",
  "destination": "<recipient_phone>",
  "getHSM": "true"
}
```

Response fragment:

```
{
  "status": "submitted",
  "templates_status": true,
  "templates": [
    {
      "appId": "abcf5776-29e0-4e3a-a8ed-09c51e69d53e",
      "category": "MARKETING",
      "data": "{{26815959-f5b8-49b4-9b0e-7458d34777cc}}\nHello{{1}}"
    }
  ]
}
```


`templates.data` contains the template send syntax. Store or cache this list in your service so users can select approved templates.

cURL example:

```
curl -X POST "https://api.chatarchitect.com/getHSM" \
  -u "<APP_ID>:<APP_SECRET>" \
  -H "Content-Type: application/json" \
  -d '{
    "channel": "whatsapp",
    "destination": "<recipient_phone>",
    "getHSM": "true"
  }'
```

Template rules

Template types

Type	Use case
Text-only	Standard notifications, confirmations, reminders, and updates.
Media	Messages with text plus an image, video, or document.

For most notifications, text-only templates are enough.

Template categories

Category	Typical use
`MARKETING`	Promotions, offers, reactivation, and other non-transactional messages.
`UTILITY`	Transactional or service messages without advertising content.
`OTP`	One-time passwords and verification codes.

Template category affects approval and pricing. Keep promotional content out of `UTILITY` and `OTP` templates.

Template approval

Customers can submit templates through the ChatArchitect application or through ChatArchitect support. If your service needs a deeper integration, you can add template submission through API as a separate feature.

Template send syntax

A template returned by `/getHSM` can look like this:

```
{{26815959-f5b8-49b4-9b0e-7458d34777cc}}\nHello{{1}}
```

Rules:

- The first `{{...}}` block is the required template ID.
- `{{1}}`, `{{2}}`, and other numbered placeholders are template variables.
- Variable values can contain spaces and multiple words.
- Variable values must not contain line breaks.

To send a template message, use the same `/whatsappmessage` endpoint. The difference from a normal text message is the value of `payload.message`.

Example template send body:

```
{
  "channel": "whatsapp",
  "destination": "<recipient_phone>",
  "payload": {
```

```
    "type": "text",
    "message": "{{26815959-f5b8-49b4-9b0e-7458d34777cc}}\nHello John"
  }
}
```

Webhook events

Status events

Delivery errors are asynchronous. A send request can return `submitted`, while final failure details arrive later in the webhook.

Example status event:

```
{
  "app": "aQWPjCAmjav",
  "timestamp": 1580311136040,
  "version": 2,
  "type": "message-event",
  "payload": {
    "id": "ee4a68a0-1203-4c85-8dc3-49d0b3226a35",
    "type": "failed",
    "destination": "<recipient_phone>",
    "payload": {
      "code": 1008,
      "reason": "User is not Opted in and Inactive"
    }
  }
}
```

Important fields:

Field	Meaning
Root `type`	`message-event` for status updates.
`payload.type`	Final status: `failed`, `sent`, `delivered`, or `read`.
`payload.destination`	Customer phone number.

Field	Meaning
`payload.payload.reason`	Failure reason, present for failed messages when available.

Inbound message events

Example inbound text message:

```
{
  "app": "aQWPjCAmjav",
  "timestamp": 1580227766370,
  "version": 2,
  "type": "message",
  "payload": {
    "id": "ABEGkYaYVSEEAhAL3SLAWwHKeKrt6s3FKB0c",
    "source": "<recipient_phone>",
    "type": "text",
    "payload": {
      "text": "Hi"
    }
  }
}
```

Important fields:

Field	Meaning
Root `type`	`message` for inbound customer messages.
`payload.source`	Customer phone number.
`payload.type`	Inbound message type.
`payload.payload.text`	Text content for text messages.

Correlation

Store your outbound request metadata together with the synchronous `messageId`. For webhook processing, also use the customer phone number and webhook `payload.id` to match status events and inbound replies to your internal records.

WhatsApp policy essentials

Follow these practical rules when designing messaging in your service:

- Send campaigns only to customers who expect messages from the business.
- Use approved templates when the business starts the conversation.
- Add an unsubscribe option to marketing templates when relevant.
- Avoid cold lists. High complaint rates can lead to template restrictions or temporary sending limits.
- Keep marketing, utility, and OTP content separated by template category.
- Monitor failed deliveries and complaints before increasing volume.

What to implement in your service

A useful integration usually includes these parts:

Feature	Recommended behavior
Credentials	Store `APP_ID` and `APP_SECRET` server-side per customer account.
Template list	Fetch templates from `/getHSM`, show category and text preview, and let users refresh the list.
Template variables	Detect `{{1}}`, `{{2}}`, and other placeholders and let users map them to service fields.
Audience selection	Let users select recipients and preview personalized messages before sending.
Send history	Store outbound request time, synchronous `messageId`, final webhook statuses, and failure reasons.
Inbound messages	Either route replies to another channel or show them in your own conversation UI.
Webhook processing	Accept status and inbound events on HTTPS, validate data shape, and process duplicate events safely.

Code examples

cURL - send text

```
curl -X POST "https://api.chatarchitect.com/whatsappmessage" \
  -u "<APP_ID>:<APP_SECRET>" \
  -H "Content-Type: application/json" \
  -d '{
    "channel": "whatsapp",
    "destination": "<recipient_phone>",
    "payload": {
      "type": "text",
      "message": "Hi John, how are you?"
    }
  }'
```

Node.js - send text with fetch

```
const APP_ID = process.env.APP_ID;
const APP_SECRET = process.env.APP_SECRET;

const auth = Buffer.from(`${APP_ID}:${APP_SECRET}`).toString('base64');

async function sendText(destination, text) {
  const response = await fetch('https://api.chatarchitect.com/whatsappmessage', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Basic ${auth}`
    },
    body: JSON.stringify({
      channel: 'whatsapp',
      destination,
      payload: {
        type: 'text',
        message: text
      }
    })
  });

  if (!response.ok) {
    throw new Error(`ChatArchitect API error: ${response.status}`);
```

```
  }

  return response.json();
}

sendText('<recipient_phone>', 'Hi John, how are you?')
  .then(console.log)
  .catch(console.error);
```

Python - send text with requests

```
import os
import requests

APP_ID = os.environ["APP_ID"]
APP_SECRET = os.environ["APP_SECRET"]

response = requests.post(
    "https://api.chatarchitect.com/whatsappmessage",
    auth=(APP_ID, APP_SECRET),
    headers={"Content-Type": "application/json"},
    json={
        "channel": "whatsapp",
        "destination": "<recipient_phone>",
        "payload": {
            "type": "text",
            "message": "Hi John, how are you?",
        },
    },
    timeout=30,
)
response.raise_for_status()
print(response.json())
```

Test checklist

Use this checklist before showing the integration to customers:

- Register an HTTPS webhook with `/webhook`.
- Send a normal text message with `/whatsappmessage`.
- Confirm the synchronous response contains `status: submitted`.
- Receive a final `message-event` webhook with `failed`, `sent`, `delivered`, or `read`.
- Fetch templates with `/getHSM`.
- Send one approved template message with a real variable value.
- Confirm failed template sends display the webhook failure reason in your service.
- Send an inbound WhatsApp message from the customer phone and confirm your webhook receives root `type: message`.

Customer message handling options

Option	Behavior	Best for
Minimal	Ignore inbound messages in your service and route operational replies to another channel such as a help desk, email inbox, or team chat.	Fast launch, notification-only use cases.
Advanced	Show inbound messages in your service and let users reply without templates when the customer wrote first.	Full conversation workflows.

You can start with the minimal option and add a conversation UI later.

Partner-facing integration description example

Use a short description like this when presenting the feature to your customers:

```
ChatArchitect.com WhatsApp for <Your Service>

Send and receive WhatsApp messages directly from <Your Service>. Start conversations with approved WhatsApp templates, receive customer replies, and track delivery statuses automatically through webhooks.
```

Recommended feature bullets:

- Incoming and outgoing WhatsApp messages.
- Business-initiated messages with approved templates.
- Delivery status tracking: sent, delivered, read, and failed.
- Template preview and variable mapping.
- Optional bulk messaging from your own audience filters.

Recommended flow

```
sequenceDiagram
    participant Service as Your service
    participant CA as ChatArchitect API
    participant WA as WhatsApp user

    Service->>CA: POST /webhook
    CA-->>Service: Webhook saved
    Service->>CA: POST /getHSM
    CA-->>Service: Approved templates
    Service->>CA: POST /whatsappmessage
    CA-->>Service: status=submitted, messageId
    CA->>Service: message-event sent/delivered/read/failed
    WA->>CA: Customer reply
    CA->>Service: inbound message webhook
```