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
```

Revision #6
Created 2026-07-05 11:33:05 UTC by New Admin
Updated 2026-07-05 11:55:50 UTC by New Admin