Mozout

Generated on 13/11/2026 19:46

KNOWLEDGE BASE · PUBLIC DOCUMENT

Adding SMS verification to your site: PHP, Node.js and Python examples

WhatsApp & APIs

CODE	VERSION	COUNTRY · LANGUAGE
RLX-MZ-KB-2661	v2026.11.13	Mozambique · English

MozOut Mozambique
RLX Holdings, S.A. · Group RLX Holdings
Tax ID (NUIT) 401 191 607 · Legal entity no. (NUEL) 105 045 974 · Economic activity code (CAE) 62022

https://www.meu.mozout.co.mz/knowledgebase/2661

Adding SMS verification to your site: PHP, Node.js and Python examples

WhatsApp & APIs Created on **November 13, 2026** 3 min read



The short answer: it takes two `POST` requests with JSON and the key in the `Authorization: Bearer` header: one to `/v1/sms/otp` with the number, the other to `/v1/sms/otp/{id}/verify` with the code the customer typed. In any language the logic is the same: keep the `id` in the server session, check, and decide on the `valid` field. The examples below follow those on the Verification SMS page, with the key read from the environment and an `Idempotency-Key` on the send.

Before you copy

1. **A Sandbox key** with `sms:write`, created on the API page in Meu MozOut, kept in an environment variable (`API_KEY`), never in the code or the repository.

2. **The requests go out from your server.** Never call the API from the browser or a mobile app: the key would be visible to everyone.

3. **The `id` stays in the server session**, tied to the user and to what they are doing.

PHP

```
<?php
// Keep the key on the server, e.g. in an environment variable
function api_call(string $path, array $body, ?string $idem = null): array {
$h = ["Authorization: Bearer " . getenv("API_KEY"), "Content-Type: application/json"];
if ($idem) { $h[] = "Idempotency-Key: " . $idem; }
$ch = curl_init("https://api.mozout.co.mz" . $path);
curl_setopt_array($ch, [CURLOPT_RETURNTRANSFER => true, CURLOPT_POST => true,
CURLOPT_HTTPHEADER => $h, CURLOPT_POSTFIELDS => json_encode($body), CURLOPT_TIMEOUT =>
30]);
```

```
$r = json_decode(curl_exec($ch), true);
return $r["data"] ?? ["error" => $r["error"] ?? null];
}
// 1. Send the code
$otp = api_call("/v1/sms/otp", ["number" => "841234567"], bin2hex(random_bytes(16)));
$_SESSION["otp_id"] = $otp["id"];
// 2. Check the code the user typed
$check = api_call("/v1/sms/otp/" . $_SESSION["otp_id"] . "/verify", ["code" => $_POST["code"]]);
if (!empty($check["valid"])) { /* number confirmed */ }
```

Node.js

With the `fetch` and `crypto.randomUUID()` built into Node.js 20 or later, inside an `async` function:

```
const API = "https://api.mozout.co.mz/v1/sms/otp";
const headers = { "Authorization": "Bearer " + process.env.API_KEY,
"Content-Type": "application/json" };
// 1. Send the code
const sent = await (await fetch(API, { method: "POST",
headers: { ...headers, "Idempotency-Key": crypto.randomUUID() },
body: JSON.stringify({ number: "841234567" }) })).json();
// 2. Check the code the user typed
const check = await (await fetch(`${API}/${sent.data.id}/verify`, {
method: "POST", headers, body: JSON.stringify({ code: userCode }) })).json();
if (check.data && check.data.valid) { /* number confirmed */ }
```

Python

With the `requests` library:

```
import os, uuid, requests
API = "https://api.mozout.co.mz/v1/sms/otp"
H = {"Authorization": "Bearer " + os.environ["API_KEY"]}
# 1. Send the code
otp = requests.post(API, headers={**H, "Idempotency-Key": str(uuid.uuid4())},
json={"number": "841234567"}, timeout=30).json()["data"]
# 2. Check the code the user typed
check = requests.post(f"{API}/{otp['id']}/verify", headers=H,
json={"code": user_code}, timeout=30).json().get("data", {})
if check.get("valid"):
pass # number confirmed
```

What the examples leave to you

Missing	Why
Handling HTTP errors	A `422` (invalid number) or a `429` (limit) has no `data`. See **reading the verification API's answers**.
Messages for each status	`valid: false` can mean a typo, expired, failed or replaced: read `status`.
Brakes on the form	A captcha and per-IP limits before calling the API. See **fraud through your code form**.
Resending	`POST /v1/sms/otp/{id}/resend`, with the same `id`.

If a check fails on the network, do not repeat it blindly.
Every check with a wrong code uses an attempt. Look at the status first with `GET /v1/sms/otp/{id}` (needs `sms:read`) and then decide.

Test everything in the Sandbox:
The send response includes the `code`, so your automated test can check it without a phone. In Production, change only the key. See **testing verification in the Sandbox**.

RECOMMENDED

Want the full examples and your own key? See the API page.

See the API page →

SEE ALSO

- **SMS verification API: we generate, send and check the code, and your system keeps only the id** →
- **Testing SMS verification in the Sandbox: the code comes back in the response** →
- **Reading the verification API's answers: valid false, 409, 422 and 429** →
- **Verification code statuses: pending, verified, expired, failed and cancelled** →
- **Sending SMS through the API: key, permissions, Sandbox and your first request** →

An API key or token has leaked: what to do, in the right order

MozOut Mozambique

RLX Holdings, S.A. · Group RLX Holdings

TAX ID (NUIT)	401 191 607
LEGAL ENTITY NO. (NUEL)	105 045 974
ECONOMIC ACTIVITY CODE (CAE)	62022
REGISTERED OFFICE	Bairro 7 de Setembro Vilankulo 1304 · CEP 0411-02, Inhambane, Mozambique
CONTACT	info@mozout.co.mz · +258 84 635 1111 · +258 86 635 1111 · www.mozout.co.mz
OPENING HOURS	Monday to Friday - 08 AM to 04 PM Saturday & Sunday - Closed

Online version of this document

RLX-MZ-KB-2661 · v2026.11.13 · Generated on 13/11/2026 19:46