# WebHook


- Structure du Webhook
- Structure du Webhook
- Valider un WebHook
- Valider un WebHook
- Gestion des erreurs
- Gestion des erreurs

# Structure du Webhook

Le WebHook est envoyé à l'adresse que vous avez enregistrée et contient l'objet Json suivant :

```
{ Guid: '', MessageType: '', DateTime: '' }
```

Guid correspond au Guid sous lequel vous pouvez récupérer les Données dans l'API en fonction du MessageType.

DateTime est le moment où l'objet a été créé dans DeDeSales.

## Définition de MessageType

| Texte | Valeur |
|---|---|
| Order | 0 |
| ReturnOrder | 1 |
| NewCustomer | 2 |
| CustomerChangeAdress | 3 |

# Structure du Webhook

Le WebHook est envoyé à l'adresse que vous avez enregistrée et contient l'objet Json suivant :

```
{ Guid: '', MessageType: '', DateTime: '' }
```

Guid correspond au Guid sous lequel vous pouvez récupérer les Données dans l'API en fonction du MessageType.

DateTime est le moment où l'objet a été créé dans DeDeSales.

## Définition de MessageType

| Texte | Valeur |
|---|---|
| Order | 0 |
| ReturnOrder | 1 |
| NewCustomer | 2 |
| CustomerChangeAdress | 3 |

# Valider un WebHook

Le WebHook envoie l'en-tête suivant :

`HTTP_X_HUB_SIGNATURE_256`

Si vous récupérez les données du courrier et calculez le hash avec votre secret, vous pouvez vous assurer que l'appel provient bien de DeDeSales.

## Tester la validation du WebHook

Vous pouvez utiliser les valeurs suivantes de `secret` et `payload` pour vérifier que votre implémentation est correcte :

- `secret` : `It's a Secret to Everybody`
- `payload` : `Hello, World!`

Si votre implémentation est correcte, les signatures que vous créez devraient correspondre aux valeurs de signature suivantes :

- Signature : `757107ea0eb2509fc211221cce984b8a37570b6d7586c22c46f4379c8b043e17`
- X-Hub-Signature-256 : `sha256=757107ea0eb2509fc211221cce984b8a37570b6d7586c22c46f4379c8b043e17`

## Exemples

Vous pouvez utiliser le langage de programmation de votre choix pour implémenter la validation HMAC dans votre code. Voici quelques exemples montrant comment une implémentation pourrait se présenter dans différents langages.

### Exemple Ruby

Par exemple, vous pouvez définir la fonction `verify_signature` suivante :

```
def verify_signature(payload_body) signature = 'sha256=' +
OpenSSL::HMAC.hexdigest(OpenSSL::Digest.new('sha256'), ENV['SECRET_TOKEN'], payload_body)
return halt 500, "Signatures didn't match!" unless Rack::Utils.secure_compare(signature,
request.env['HTTP_X_HUB_SIGNATURE_256']) end
```

Ensuite, vous pouvez appeler cette fonction lorsque vous recevez un WebHook :

```
post '/payload' do request.body.rewind payload_body = request.body.read
verify_signature(payload_body) push = JSON.parse(payload_body) "I got some JSON:
#{push.inspect}" end
```

## Exemple Python

Par exemple, vous pouvez définir et appeler la fonction `verify_signature` suivante lorsque vous recevez un WebHook :

```
import hashlib import hmac def verify_signature(payload_body, secret_token, signature_header):
"""Verify that the payload was sent from GitHub by validating SHA256. Raise and return 403 if
not authorized. Args: payload_body: original request body to verify (request.body())
secret_token: GitHub app webhook token (WEBHOOK_SECRET) signature_header: header received from
GitHub (x-hub-signature-256) """ if not signature_header: raise HTTPException(status_code=403,
detail="x-hub-signature-256 header is missing!") hash_object =
hmac.new(secret_token.encode('utf-8'), msg=payload_body, digestmod=hashlib.sha256)
expected_signature = "sha256=" + hash_object.hexdigest() if not
hmac.compare_digest(expected_signature, signature_header): raise
HTTPException(status_code=403, detail="Request signatures didn't match!")
```

## Exemple JavaScript

Par exemple, vous pouvez définir la fonction `verifySignature` suivante et l'appeler dans n'importe quel environnement JavaScript lorsque vous recevez un WebHook :

```
let encoder = new TextEncoder(); async function verifySignature(secret, header, payload) { let
parts = header.split("="); let sigHex = parts[1]; let algorithm = { name: "HMAC", hash: {
name: 'SHA-256' } }; let keyBytes = encoder.encode(secret); let extractable = false; let key =
await crypto.subtle.importKey( "raw", keyBytes, algorithm, extractable, [ "sign", "verify" ],
); let sigBytes = hexToBytes(sigHex); let dataBytes = encoder.encode(payload); let equal =
await crypto.subtle.verify( algorithm.name, key, sigBytes, dataBytes, ); return equal; }
function hexToBytes(hex) { let len = hex.length / 2; let bytes = new Uint8Array(len); let
index = 0; for (let i = 0; i < hex.length; i += 2) { let c = hex.slice(i, i + 2); let b =
parseInt(c, 16); bytes[index] = b; index += 1; } return bytes; }
```

## Exemple Typescript

Par exemple, vous pouvez définir et appeler la fonction `verify_signature` suivante lorsque vous recevez un WebHook :

JavaScript

```
import { Webhooks } from "@octokit/webhooks"; const webhooks = new Webhooks({ secret: process.env.WEBHOOK_SECRET, }); const handleWebhook = async (req, res) => { const signature = req.headers["x-hub-signature-256"]; const body = await req.text(); if (!(await webhooks.verify(body, signature))) { res.status(401).send("Unauthorized"); return; } // The rest of your logic here };
```

## Exemple C#

Par exemple, vous pouvez définir la fonction `verify_signature` suivante :

```
using System; using System.Text; using System.Security.Cryptography; public bool verify_signature( string header_HTTP_X_HUB_SIGNATURE_256, string postData, string secret ) { HMACSHA256 mySHA256 = new HMACSHA256(Encoding.ASCII.GetBytes(secret)); byte[] hashValue = mySHA256.ComputeHash(Encoding.ASCII.GetBytes(postData)); return header_HTTP_X_HUB_SIGNATURE_256 == PrintByteArray(hashValue); } private String PrintByteArray(byte[] array) { String result = String.Empty; for (int i = 0; i < array.Length; i++) result += $"{array[i]:X2}"; return result; }
```

# Valider un WebHook

Le WebHook envoie l'en-tête suivant :

`HTTP_X_HUB_SIGNATURE_256`

Si vous récupérez les données du courrier et calculez le hash avec votre secret, vous pouvez vous assurer que l'appel provient bien de DeDeSales.

## Tester la validation du WebHook

Vous pouvez utiliser les valeurs suivantes de `secret` et `payload` pour vérifier que votre implémentation est correcte :

- `secret` : `It's a Secret to Everybody`
- `payload` : `Hello, World!`

Si votre implémentation est correcte, les signatures que vous créez devraient correspondre aux valeurs de signature suivantes :

- Signature : `757107ea0eb2509fc211221cce984b8a37570b6d7586c22c46f4379c8b043e17`
- X-Hub-Signature-256 : `sha256=757107ea0eb2509fc211221cce984b8a37570b6d7586c22c46f4379c8b043e17`

## Exemples

Vous pouvez utiliser le langage de programmation de votre choix pour implémenter la validation HMAC dans votre code. Voici quelques exemples montrant comment une implémentation pourrait se présenter dans différents langages.

### Exemple Ruby

Par exemple, vous pouvez définir la fonction `verify_signature` suivante :

```
def verify_signature(payload_body) signature = 'sha256=' +
OpenSSL::HMAC.hexdigest(OpenSSL::Digest.new('sha256'), ENV['SECRET_TOKEN'], payload_body)
return halt 500, "Signatures didn't match!" unless Rack::Utils.secure_compare(signature,
request.env['HTTP_X_HUB_SIGNATURE_256']) end
```

Ensuite, vous pouvez appeler cette fonction lorsque vous recevez un WebHook :

```
post '/payload' do request.body.rewind payload_body = request.body.read verify_signature(payload_body) push = JSON.parse(payload_body) "I got some JSON: #{push.inspect}" end
```

## Exemple Python

Par exemple, vous pouvez définir et appeler la fonction `verify_signature` suivante lorsque vous recevez un WebHook :

```
import hashlib import hmac def verify_signature(payload_body, secret_token, signature_header): """Vérifie que la charge utile a été envoyée par GitHub en validant SHA256. Lève et retourne 403 si non autorisé. Args: payload_body: corps original de la requête à vérifier (request.body()) secret_token: token webhook de l'application GitHub (WEBHOOK_SECRET) signature_header: en-tête reçu de GitHub (x-hub-signature-256) """ if not signature_header: raise HTTPException(status_code=403, detail="l'en-tête x-hub-signature-256 est manquant !") hash_object = hmac.new(secret_token.encode('utf-8'), msg=payload_body, digestmod=hashlib.sha256) expected_signature = "sha256=" + hash_object.hexdigest() if not hmac.compare_digest(expected_signature, signature_header): raise HTTPException(status_code=403, detail="Les signatures de la requête ne correspondent pas !")
```

## Exemple JavaScript

Vous pouvez par exemple définir la fonction `verifySignature` suivante et l'appeler dans n'importe quel environnement JavaScript lorsque vous recevez un WebHook :

```
let encoder = new TextEncoder(); async function verifySignature(secret, header, payload) { let parts = header.split("="); let sigHex = parts[1]; let algorithm = { name: "HMAC", hash: { name: 'SHA-256' } }; let keyBytes = encoder.encode(secret); let extractable = false; let key = await crypto.subtle.importKey( "raw", keyBytes, algorithm, extractable, [ "sign", "verify" ], ); let sigBytes = hexToBytes(sigHex); let dataBytes = encoder.encode(payload); let equal = await crypto.subtle.verify( algorithm.name, key, sigBytes, dataBytes, ); return equal; } function hexToBytes(hex) { let len = hex.length / 2; let bytes = new Uint8Array(len); let index = 0; for (let i = 0; i < hex.length; i += 2) { let c = hex.slice(i, i + 2); let b = parseInt(c, 16); bytes[index] = b; index += 1; } return bytes; }
```

## Exemple Typescript

Par exemple, vous pouvez définir et appeler la fonction `verify_signature` suivante lorsque vous recevez un WebHook :

JavaScript

```
import { Webhooks } from "@octokit/webhooks"; const webhooks = new Webhooks({ secret: process.env.WEBHOOK_SECRET, }); const handleWebhook = async (req, res) => { const signature = req.headers["x-hub-signature-256"]; const body = await req.text(); if (!(await webhooks.verify(body, signature))) { res.status(401).send("Unauthorized"); return; } // The rest of your logic here };
```

## Exemple C#

Par exemple, vous pouvez définir la fonction `verify_signature` suivante :

```
using System; using System.Text; using System.Security.Cryptography; public bool verify_signature( string header_HTTP_X_HUB_SIGNATURE_256, string postData, string secret ) { HMACSHA256 mySHA256 = new HMACSHA256(Encoding.ASCII.GetBytes(secret)); byte[] hashValue = mySHA256.ComputeHash(Encoding.ASCII.GetBytes(postData)); return header_HTTP_X_HUB_SIGNATURE_256 == PrintByteArray(hashValue); } private String PrintByteArray(byte[] array) { String result = String.Empty; for (int i = 0; i < array.Length; i++) result += $"{array[i]:X2}"; return result; }
```

# Gestion des erreurs

Si un WebHook ne peut pas être livré à l'URL que vous avez spécifiée, le WebHook configuré est automatiquement supprimé si les deux événements suivants se produisent :

1. 1 000 tentatives échouées consécutives ont été effectuées.
2. La première tentative date de plus de 24 heures.

Lors de la suppression, vous recevez une notification par e-mail à l'adresse e-mail que vous avez enregistrée lors de la création du WebHook.

# Gestion des erreurs

Si un WebHook ne peut pas être livré à l'URL que vous avez spécifiée, le WebHook configuré sera automatiquement supprimé si les deux événements suivants se produisent :

1. 1 000 tentatives consécutives ont échoué.
2. La première tentative date de plus de 24 heures.

Lors de la suppression, vous recevrez une notification par e-mail à l'adresse e-mail que vous avez fournie lors de la création du WebHook.