Envoyer des SMS avec l’API SMS

Table des matières

1 Présentation Quoi ?

Le serveur DMZ met à disposition de ses utilisateurs, dans le cadre de leurs projets, une API pour effectuer des notifications par SMS. Derrière cet intitulé énigmatique se cache un premier pas dans le monde des API Rest. Cette API offre la possibilité d’envoyer des SMS sur un téléphone portable depuis n’importe quel appareil (arduino, esp32, raspberry, PC , serveur virtuel…) disposant d’une connexion Internet.

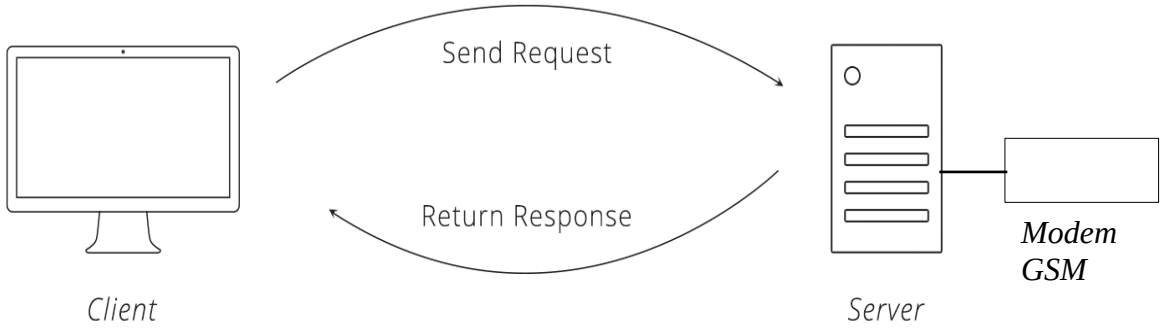
Il est possible de programmer simplement toute application pour que cette dernière demande l’envoi d’un SMS si un événement particulier se produit. Des programmes basiques en langage C, Javascript PHP, java et python sont proposés en exemple dans ce document pour vous aider à développer votre application.

Bien que les opérateurs téléphoniques parlent d’envoi de SMS en illimité, la notion d'illimité a ses limites. Prenons l'exemple de SFR. Comme indiqué dans leurs conditions générales d'abonnement, les SMS illimités sont en fait limités à **200 destinataires dans le mois**. Au-delà de ce nombre, vos SMS n'arriveront pas à destination. De plus l’opérateur limite la fréquence à laquelle vous pouvez envoyer des SMS.

Le service offert par la DMZ a un plafond de 140 SMS par jour et par utilisateur sauf dérogation particulière dûment justifiée. Chaque envoi de SMS doit être espacé de 15s.

2 Conditions d’utilisation Comment ?

Pour envoyer un SMS il suffit d’envoyer une requête http méthode GET ou POST à l’api du serveur.



Le endpoint ***Ruche/api/sendSMS*** sert à créer un nouveau SMS. Chaque SMS créé est conservé dans la base de donnée. La sécurité est assuré par l’utilisation de clé propre à chaque utilisateur et le contrôle du quota.

1 Requête pour créer et envoyer un SMS

la requête est composé de trois partie :

1. **Url du serveur** : http://touchardinforeseau.servehttp.com/
2. **Endpoint** : Ruche/api/sendSMS
3. **Querry String** : ?key=123456ABCDEF&number=06XXXXXXXX&message=test

   La Querry String est composé des champs suivants

   - **key** Votre clé d’identification
   - **number** le numéro de téléphone du destinataire
   - **message** le contenu du SMS encodé sous forme d’url (Percent-encoding)

Vous pouvez également, si vous le préférez, envoyer les paramètres en POST. Dans ce cas, le contenu du message n’a pas besoin d’être encodé.

En résumé voici une requête complète en GET

http://touchardinforeseau.servehttp.com/Ruche/api/sendSMS? key=YYYYYYYYY&number=XXXXXXXXXX&message=test

Pour vérifier votre commande, coller votre requête URL dans la barre d’adresse de votre navigateur web, vous devriez recevoir un SMS contenant le message test. Remplacer les Y par votre clé api et les X par le numéro de téléphone du destinataire.

2 Réponse :

Le code de retour HTTP indique le succès ou non de l’opération :

Si le code de retour est 202

La **requête a été reçue et acceptée** mais n'a peut être pas encore été traitée. En effet le serveur n’a aucun moyen d’affirmer que le sms est bien reçu sur le téléphone du destinataire. il n'y a aucun moyen en HTTP d'envoyer une réponse asynchrone ultérieure indiquant le résultat issu du traitement de la requête.

le corps de la réponse en retour contient un objet json contenant le status, le numéro du destinataire, le login de l’utilisateur créateur, le message et l’encodage des caractères utilisés.

```
{"status":"202 Accepted",
"numero":"0689744235",
"creator":"alecren",
"message":"test",
"encodage":"Default_No_Compression"}
```

En cas de problème les codes de retour peuvent être :

Une réponse d'erreur dans la classe **400,** elle indique les **erreurs du client** , l'API REST refuse d 'honorer la requête, c'est-à-dire que le client n'a pas fourni les données nécessaires pour effectuer l’envoi, ou dépasse les quotas imposés.

403, "Bad Request", "La demande ne peut pas être satisfaite en raison d'un mauvais numéro de téléphone"

403, "Request Entity Too Large", "Votre message est trop long"

Cette erreur intervient lorsque la longueur du message SMS est supérieur à 160 octets. Ce qui correspond à 160 caractères ASCII ou 80 caractères UNICODE. L’api choisit automatiquement l’encodage nécessaire en fonction de la présence de caractères unicode dans le message.

405, "Authorization Required", "Veuillez fournir les détails d'authentification appropriés."

Cette erreur intervient lorsque la clé API est absente ou erronée.

406, "**daily quota exceeded**", "You have exceeded your daily quota"

Cette erreur intervient lorsque vous essayez d’envoyer plus de 140 SMS par jour.

429, "**Too Many Requests**", "Wait before making another request."

L'utilisateur a émis trop de requêtes dans un laps temps donné. Cette erreur intervient lorsque vous ne dépassez pas la limite d'intervalle d'envoi de 15 secondes.

Les réponses d’erreur dans la classe 500 indique les erreurs du côté serveur. Par exemple le serveur rencontre des problèmes de connexion avec la base de données ou avec le modem GSM.

500, "Internal Server Error", "Internal Server Error"

Cette erreur indique que le serveur a rencontré un problème inattendu qui l'empêche d’effectuer la requête. Cela peut être la conséquence d’un problème de connexion avec la base de données.

503, **Service Unavailable**, The server was unavailable or unable to process your request. Try your request later.

Le serveur rencontre des problèmes avec le modem GSM ou avec le réseau de l’opérateur GSM.

3 Exemples de code client en C

langage C La bibliothèque curl est certainement la plus efficace et la plus utilisée.

```c
// Compilation : gcc -Wall -std=c11 sendSMS.c -lcurl -o sendSMS

#include <curl/curl.h>
#include <curl/easy.h>
#include <stdio.h>

int main()
{
    CURL *hnd = curl_easy_init();
// Méthode POST
    curl_easy_setopt(hnd, CURLOPT_CUSTOMREQUEST, "POST");
// URL du serveur
    curl_easy_setopt(hnd, CURLOPT_URL,
"http://touchardinforeseau.servehttp.com/Ruche/api/sendSMS");

// Header
    struct curl_slist *headers = NULL;
    headers = curl_slist_append(headers, "cache-control: no-cache");
    headers = curl_slist_append(headers, "Content-Type: application/x-www-form-
urlencoded");

    curl_easy_setopt(hnd, CURLOPT_HTTPHEADER, headers);

// Body
    curl_easy_setopt(hnd, CURLOPT_POSTFIELDS,
"key=O7VZJ5LOABU&number=0689744235&message=Test avec langage C");

// Execution de la requète
    CURLcode ret = curl_easy_perform(hnd);

/* si erreurs */
    if (ret != CURLE_OK) {
        printf("Failed : %s", curl_easy_strerror(ret));
    }
    return 0;
}
```

```
pi@raspberrypi3:~/test $ ./sendSMS
{"status":"202 Accepted","numero":"0689744235","creator":"toto","message":"Test avec
langage C","encodage":"Default_No_Compression"}
```

4 Exemples de code client en php

En langage PHP

La bibliothèque cURL de PHP est certainement la plus efficace et la plus utilisée des méthodes pour se connecter à un API.

```
<?php

	$curl = curl_init();
	$numero = '0689744235';
	$key = 'O7VZJ5LOABX';
	$message = 'un test avec Php Curl';
	$postfields = 'key='.$key.'&number='.$numero.'&message='.$message;
	$options = array(
		CURLOPT_URL => "http://touchardinforeseau.servehttp.com/Ruche/api/sendSMS",
		CURLOPT_RETURNTRANSFER => true,
		CURLOPT_ENCODING => "",
		CURLOPT_MAXREDIRS => 10,
		CURLOPT_TIMEOUT => 0,
		CURLOPT_FOLLOWLOCATION => true,
		CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
		CURLOPT_CUSTOMREQUEST => "POST",
		CURLOPT_POSTFIELDS => $postfields,
		CURLOPT_HTTPHEADER => array(
			"Content-Type: application/x-www-form-urlencoded"
		));
	curl_setopt_array($curl, $options);

	$response = curl_exec($curl);

	curl_close($curl);
	echo $response;
?>
```

5 Exemple de code client en Java

Unirest est une bibliothèque client HTTP légère en Java de mashape .

Pour installer une bibliothèque externe sous netbeans suivre le tuto https://www.youtube.com/watch?v=OQOpYHwA1A0. Vous devez télécharger la bibliothèque (fichier .jar)

Unirest facilite la création de requêtes en Java, voici une requête POST de base qui expliquera tout: La demande est faite quand `as[Type]()` est invoquée, les types possibles comprennent `Json`, `String`…

```
package sendsms2;

import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;

public class SendSMS2 {

    public static void main(String[] args) throws UnirestException {
        Unirest.setTimeouts(0, 0);
        HttpResponse<JsonNode> response =
Unirest.post("http://touchardinforeseau.servehttp.com/Ruche/api/sendSMS")
                .header("Content-Type", "application/x-www-form-urlencoded")
                .field("key", "O7VZJ5LOABU")
                .field("number", "0689744235")
                .field("message", "Un test en Java")
                .asJson();
        System.out.print("Réponse du serveur : ");
          // Une fois la réponse reçue, vérifions le code d'état
        System.out.println( response.getStatus() );
          // Affichons le corps de la réponse
        System.out.println( response.getBody() );
    }
}
```

le programme exemple sur GitHub

6 Exemple de code client en python

Ce code client en python est proposé pour les enseignants en NSI. (attention à bien utiliser python 3). Ce code a été testé sur ma raspberry pi4 version debian buster.

J'utilise la bibliothèque standard **requests.** c'est un module python permettant d'utiliser le protocole http de façon ultra simple! **Lien vers la documentation python en français**

Pour l'installer utiliser pip**:**

```
pip install requests
```

```
#!/usr/bin/python3
import requests
url = "http://touchardinforeseau.servehttp.com/Ruche/api/sendSMS"
body = 'key=O7VZJ5LOABU&number=0689744235&message=un test avec Python3'
headers = {
  'Content-Type': 'application/x-www-form-urlencoded'
}

reponse = requests.request("POST", url, headers=headers, data = body)
print(reponse.text.encode('utf8'))
```

Le code utilise la méthode **request()** de l'objet **requests**. Cette méthode retourne la réponse obtenue.

Exécution :

```
pi@raspberrypi:~/python $ ./sendSMS.py
b'{"status":"202 Accepted","numero":"0689744235","creator":"toto","message":"un test avec Python3","encodage":"Default_No_Compression"}\n'
```

Test avec un clé API erronée

```
pi@raspberrypi:~/python $ ./sendSMS.py
b'{"status":405,"message":"Authorization Required","detail":"Please provide proper authentication details."}'
```

7 Exemple de code client en Javascript

Cet exemple montre comment envoyer un SMS à partir des données issues d'un formulaire. Le formulaire possède un champ hidden qui contiendra la clé API. Attention la page web doit être protégée. Autrement dit l'accès a ce formulaire n'est possible que lorsque l'utilisateur est authentifié.

Cette page doit être construite dynamiquement en PHP.

La page html

```
<!DOCTYPE html>
<html>
  <head>
    <title>test</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="//ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
    <script src="main.js" type="text/javascript"></script>
  </head>

  <body>
    <div>Ecrire un SMS</div>
    <form action="" class="formName">
      <div class="form-group">
        <label class="font-weight-bold">Number : </label><br />
        <input type="text" id="number" name="number" size="12" placeholder="Number"><br>
      </div>
      <div class="form-group">
        <label class="font-weight-bold">Message : </label><br />
        <textarea rows="5" id="message" name="message" maxlength="160" class="form-control">
        </textarea>
      </div>
      <input type="hidden" id="key" name="key" value="RDIK9LVVYEYZYZFX">
    </form>
    <button type="button" class="btn btn-blue" id="envoyer">Envoyer</button>

  </body>
</html>
```

1 Le fichier main.js (avec jquery et $.ajax)

```
function envoyerSMS(){
        console.log("envoi demandé");
        let message = $('#message').val();
        let number = $('#number').val();
        let key = $('#key').val();

        var settings = {
          "url": "http://touchardinforeseau.servehttp.com/Ruche/api/sendSMS",
          "method": "POST",
          "timeout": 0,
          "headers": {
                "Content-Type": "application/x-www-form-urlencoded"
          },
          "data": {
                "key": key,
                "number": number,
                "message": message
          }
        };


        $.ajax(settings).done(function (response) {
          console.log(response);
          if (response.status === "202 Accepted")
                alert("SMS envoyé");
        });
}

$(function () {
   $('#envoyer').click(envoyerSMS);
});
```

2 Autre possibilité avec jquery $.getJSON

```
function EnvoyerSMS() {
    console.log("clique sur envoyer");
    let message = $('#message').val();
    let number = $('#number').val();
    let form_data = $('.formName').serialize();
    let apiSms = 'http://touchardinforeseau.servehttp.com/Ruche/api/sendSMS';


    if (!message || isNaN(number)) {
        alert('fournir un numero valide et un message');
        return false;
    }


    $.getJSON(apiSms, form_data, function (response, status, error) {
        if (response.status === "202 Accepted") {
            alert("message Accepted");
        } else {
            console.log("Probleme !!!");
        }
    }).fail(function (response, status, error) {
            alert("Error : " + error);
    });
}

$(function () {
    $('#envoyer').click(EnvoyerSMS);
});
```

8 Procédure d’installation côté Serveur

1 Gammu-smsd

Gammu-smsd est un processus daemon qui se charge de recevoir et d'envoyer les SMS. Pour ce faire il analyse périodiquement le modem GSM pour lire les SMS reçus et les stocker dans la table **inbox** de la base de données **smsd** . Il envoie les SMS placés dans la table **outbox** puis archive les SMS envoyé dans la table **sentitems**

Attention vous ne pouvez pas exécuter Gammu et Gammu-SMSD en même temps sur le même modem GSM, cependant vous pouvez contourner cette limitation en suspendant temporairement SMSD en utilisant les signaux SIGUSR1 et SIGUSR2.

- **SIGUSR1** Suspend SMSD, en fermant la connexion au modem GSM
- **SIGUSR2** Reprend SMSD (après la suspension précédente).

2 Installation de gammu-smsd

```
pi@raspberrypi3:~ $ sudo apt-get install gammu-smsd
pi@raspberrypi3:~ $ gammu-smsd -v
Gammu-smsd version 1.33.0
```

3 Installation de la base de données

Avant d'exécuter gammu-smsd, vous devez créer la base de données **smsd** et les tables nécessaires , qui sont décrites ci-dessous.

L'utilisateur smsd@localhost accédant à la base de données n'a pas besoin de beaucoup de privilèges, les privilèges suivants devraient être suffisants:

```
GRANT USAGE ON *.* TO 'smsd'@'localhost' IDENTIFIED BY 'passwordSMSD';

GRANT SELECT, INSERT, UPDATE, DELETE ON `smsd`.* TO 'smsd'@'localhost';

CREATE DATABASE smsd;
```

Pour la création des tables Vous pouvez trouver le script dans le répertoire docs/sql/mysql.sql.

4 Configuration de Gammu-smsd

Gammu-smsd lit sa configuration à partir d'un fichier de configuration. Son emplacement peut être spécifié sur la ligne de commande (option -c), sinon le fichier par défaut **/etc/gammu-smsdrc** est utilisé.

La section **[gammu]** est la configuration d'une connexion au modem GSM la section **[smsd]** configure le démon SMS lui-même. Le paramètre **pin** est facultatif, mais vous devez le définir si votre carte SIM après la mise sous tension nécessite un code PIN.

5 Le fichier de configuration

```
# Configuration file for Gammu SMS Daemon
# Gammu library configuration, see gammurc(5)

[gammu]

port = /dev/ttyUSB0
connection = at
GammuLoc = fr_FR.UTF8
gammucoding = utf8

[smsd]
service = sql
driver = native_mysql
host = localhost
user = smsd
password = passwordSMSD
database = smsd

logfile = syslog
# Increase for debugging information
debuglevel = 0

pin = 0000
```

Dans la section smsd permet de paramétrer le service sur sql, l’ensemble des paramètres pour ce connecter à la base doit être renseigné.

9 Envoyer un SMS côté serveur avec le service SQL

Vous avez deux possibilités pour envoyer un SMS. La première en ligne de commande

gammu-smsd-inject est un programme qui met en file d'attente le message, qui sera ensuite envoyé par le démon à l'aide du modem GSM connecté.

```
root@raspberrypi3:/# gammu-smsd-inject TEXT 0689744235 -text "Nous sommes le
`date`"
gammu-smsd-inject[14265]: Connected to Database: smsd on localhost
gammu-smsd-inject[14265]: Connected to Database native_mysql: smsd on localhost
gammu-smsd-inject[14265]: Written message with ID 1
```

La deuxième en utilisant une requète SQL INSERT sur la table outbox

```
INSERT INTO outbox (DestinationNumber, TextDecoded, CreatorID, Coding) VALUES
( '0601020304' , 'Un test', 'PhilippeS', 'Default_No_Compression' )
```

Lorsque le sms est envoyé une copie est archivé dans la table **sentitems.**

10 Le code php côté serveur

https://github.com/PhilippeSimier/Aggregator/blob/master/html/api/sendSMS.php

La bibliothèque des fonctions utilisées

https://github.com/PhilippeSimier/Aggregator/blob/master/html/api/biblio.php