Documentation Technique

Interface de communication Joomla - Livestorm

P&P Forum Project - Geppia

Table des matières

1 Introduction

Le but de ce projet est de réaliser une interface faisant communiquer le système d'un site web fonctionnant sous Joomla (CMS et Framework PHP) avec Livestorm, un outil de communication vidéo pour les entreprises. Ce projet a été réalisé dans le cadre du Process & Packaging Forum, une journée de webconférences organisée par le GEPPIA.
Plus concrètement, cette interface permettra d'enregistrer un même participant à différentes webconférences à l'aide d'un unique formulaire externalisé.

1.1 Contraintes techniques

Dans le cas pratique, le site fonctionne avec le CMS Joomla et le formulaire est généré par le module RS Form Pro. En réalité, le programme utilisé par cette interface de communication peut-être adapté à de nombreuses installations différentes. En effet, la partie serveur n'est autre qu'un script PHP ne nécessitant pas particulièrement de fonctionner avec Joomla. Cpependant, la partie cliente du programme devra être reprogrammée partiellement ou complètement en fonction des choix des technologies utilisées. Il s'agit d'un script JS récupérant les données d'un formulaire et les envoyant au serveur via AJAX.

1.2 Ressources - API Livestorm

Livestorm propose depuis Décembre 2020 une API en mode Bêta. Certaines fonctions sont donc limitées en usage (maximum 10.000 appels par mois et 5 requêtes par seconde) et le manuel est encore incomplet. Cependant, cela demeure une ressource indispensable pour interconnecter notre formulaire et les évènements Livestorm.

2 Réalisation et Intégration

2.1 Dialoguer avec l'API via CURL

Pour inscrire des participants à une réunion, il faut commencer par dialoguer avec l'API fournie par Livestorm. Pour cela il faut déjà générer une clé d'API (ou jeton d'identification) auprès de Livestorm. La méthode est expliquée sur cette page:
`https://developers.livestorm.co/docs/authorization`

Voici une requête CURL de test afin de vérifier nos paramètres. La structure présentée ci-dessous sera réutilisée pour toutes les autres requêtes.

```php
<?php

$cr = curl_init();

$token = "your_token";
$url = "https://api.livestorm.co/v1/ping";

// Creating signature
$timestamp = time();
$signature = hash_hmac('sha1', $timestamp, $token);

// Header parameters
$headers = [
        "HTTP/2",
        "authorization: ".$token,
        "Accept: application/vnd.api+json",
        "Content-Type: application/json",
        "sig: ".$signature,
];

// Request Options
curl_setopt($cr, CURLOPT_URL, $url);
curl_setopt($cr, CURLOPT_RETURNTRANSFER, true);
curl_setopt($cr, CURLOPT_SSL_VERIFYPEER, false);
curl_setopt($cr, CURLOPT_HTTPHEADER, $headers);
curl_setopt($cr, CURLINFO_HEADER_OUT, true);

?>
```

Il est important de respecter les options et l'entête de la requête, sans quoi vous risquez d'obtenir un code d'erreur comme réponse de la part de l'API.

Il faut ensuite éxecuter cette requête et récupérer la réponse de l'API afin de traiter cette dernière:

```php
<?php
$response = curl_exec($cr);
$error = curl_error($cr);

if ($error)
{
        // Exit with error message
}
else
{
        $response = json_decode($response, true);
}

curl_close($cr);
?>
```

Attention, il est primordial de savoir dissocier les erreurs de la requête et les erreurs de retour de l'API. L'erreur testée ligne 6 est une erreur qui pourrait subvenir lors de l'envoi ou de la réception de la requête par PHP. Les erreurs que pourraient renvoyer l'API (mauvais paramètres, données non conformes...) doivent être traitées après décodage de la trame JSON. Un exemple non exhaustif est donné ci-dessous:

```php
<?php
if (!empty($response["errors"]))
{
        if ($response["errors"][0]["code"] == "401")
        {
                // Authetication failed
        }
        if ($response["errors"][0]["code"] == "429")
        {
                // Too much request (rate limit)
        }
}
else
{
        // Authetication success
}
?>
```

2.2 Récupérer les données depuis un formulaire

Les données sont saisies par un client depuis un formulaire. Dans notre cas nous utiliserons RS Form Pro. Ces données doivent être envoyées au script PHP qui les traitera, les formalisera et effectuera une requête à l'API Livestorm.

2.2.1 Envoi des données via AJAX

Pour une meilleure expérience utilisateur, la norme actuelle est de réaliser l'interaction client-serveur en arrière-plan, sans chargement de page à l'aide de la technologie AJAX

```
<script type="text/javascript" >

function sendMyForm(){
    // Annule l'envoi par défaut du formulaire
    event.preventDefault();

    //AJAX Request
    fetch('https://www.geppia.com/test/script.php',
    {
        method: 'POST',
        body: new FormData(document.querySelector('form#userForm')),
    }).then(function (response)
    {
        if (response.ok) {
            return response.json();
        }
        return Promise.reject(response);
    }).then(function (data)
    {
        // Response
        if (data['status'] == true){
            // Request Success
        }
        else {
            // Request Failed
        }
    }).catch(function (error)
    {
        alert(error);
    });

    return false;
};
```

2.2.2 Paramétrage RSForm Pro

Afin de faire s'éxecuter notre code Javascript au moment de valider le formulaire RSForm Pro, nous devons paramétrer ce dernier. Choisir le formulaire à éditer puis Propriétés > Formulaire > Attributs du formulaire. Trois champs doivent être remplis:

- Action: mettre le chemin absolu de votre script PHP
- Attributs HTML supplémentaires: `onsubmit="return sendMyForm();"`
- ID CSS: `userForm`

Quelque soit le mode d'utilisation du formulaire (module CMS ou formulaire natif), il est important de faire concorder le nom de la fonction d'appel, l'identifiant du formulaire, ceux des champs et le nom du script entre les différents code Javascript, PHP et HTML.

2.2.3 Affichage avancé

Il est important que l'utilisateur puisse avoir connaissance de l'état courant de son action. Pour cela, la méthode classique est d'afficher de courts messages informatifs à proximité du bouton de validation du formulaire. Dans le cas de RSForm Pro, on peut ajouter une structure de type span et la gérer directement depuis notre Javascript avec quelques fonctions de bases, en supposant que du code CSS pour une meilleure mise en page a déjà été inséré selon les class .succes ou .fail:

```
1  if (document.getElementById("label-status-text") == null)
2  {
3      // Add element if it doesn't exist already
4      document.querySelector("div.rsform-block-validation").insertAdjacentHTML(
5          'beforebegin',
6          '<span id="label-status-text"></span>'
7      );
8  }
9
10 // Set a message
11 document.querySelector("#label-status-text").innerText = "Error";
12
13 // Set or remove a style
14 document.querySelector("#label-status-text").classList.remove("success");
15 document.querySelector("#label-status-text").classList.add("fail");
```

2.3 Remarques

Le code présenté dans ce document est trop insuffisant pour être mis en production. Il ne présente pas non plus les mesures à adopter pour garantir la sécuité des données utilisateurs. Le restant non fourni sera à écrire en fonction de chaque installation et des attentes spécifiques au projet. Cependant, il est important de garder en mémoire quelques points:

2.3.1 Gestion des erreurs

La documentation de l'API Livestorm fournit toutes les valeurs de codes retour possibles. Il est important de considérer chaque possibilité.

2.3.2 Gestion des quotas

Si vous êtes ammené à générer de nombreuses requêtes simultanées vers l'API Livestorm, vous risquerez d'être confronté à des problèmes de quotas (voir plus haut: erreur 429). Afin de gérer ces quotas, il peut être intéressant de retenter une requête après une seconde de pause et dans une limite d'essais ratés. Voici un exemple de code partiel pouvant répondre à cette problématique:

```php
<?php

// $session is an array of events sessions id
$i = 0;
$try = 0;
$nberrors = 0;
$cut_while = false;

while ($i < count($sessions) && !$cut_while)
{
        $retour = registerToSession($sessions[$i]);// +others params

        if ($retour["code"] == 429 && $try < 3) {
                sleep(1);
                $try++;
        }
        else if ($retour["code"] == 429 && $try >= 3) {
                $nberrors++;
                $i++;
        }
        else {
               $i++;
        }
}

?>
```

3 Exemple: Enregistrer des participants à une réunion

En fonction des requêtes, une liste de données specifiques peut-être éxigée par l'API. Voici une exemple reprenant la structure précédente et l'adaptant pour inscrire des particpant à une session d'un évènement. Les données spécifiques dépendent également des champs que vous avez ajoutés dans la session en créant l'évènement depuis votre compte Livestorm.

```php
<?php

function registerToSession($idsession, $email, $firstname, $lastname,
    $company, $ciefun, $country)
{
    $cr = curl_init();

    $token = "replace_with_your_token";
    $url = "https://api.livestorm.co/v1/sessions";

    $data = '{
        "data": {
            "type": "people",
            "attributes": {
                "fields":[
                    {"id":"email","value": "'.$email.'"},
                    {"id":"first_name","value":"'.$firstname.'"},
                    {"id":"last_name","value":"'.$lastname.'"},
                    {"id": "company", "value":"'.$company.' ('.$ciefun.')"},
                    {"id": "job", "value":"'.$ciefun.'"},
                    {"id": "country", "value":"'.$country.'"}
                ]
            }
        }
    }';

    $timestamp = time();
    $signature = hash_hmac('sha1', $timestamp, $token);

    $headers = [
        "HTTP/2",
        "authorization: ".$token,
        "Accept: application/vnd.api+json",
        "Content-Type: application/json",
        "sig: ".$signature,
    ];

```

```php
curl_setopt($cr, CURLOPT_URL, $url."/".$idsession."/people");
curl_setopt($cr, CURLOPT_RETURNTRANSFER, true);
curl_setopt($cr, CURLOPT_SSL_VERIFYPEER, false);
curl_setopt($cr, CURLOPT_HTTPHEADER, $headers);
curl_setopt($cr, CURLINFO_HEADER_OUT, true);
curl_setopt($cr, CURLOPT_POST, true);
curl_setopt($cr, CURLOPT_POSTFIELDS, $data);

$response = curl_exec($cr);
$error = curl_error($cr);

if($error)
{
    $retour = array("code" => 2, "msg" => "Request Error");
}
else{
    $response = json_decode($response, true);

    if (!empty($response["errors"]))
    {
        if ($response["errors"][0]["code"] == "422")
        {
            $retour = array(
                "code" => 3,
                "msg" => "You are already register for this session"
            );
        }
        else if ($response["errors"][0]["code"] == "400")
        {
            $retour = array(
                "code" => 1,
                "msg" => "Error in the form"
            );
        }
        .
        .
        .
        else if ($response["errors"][0]["code"] == "429")
        {
            $retour = array(
                "code" => 429,
                "msg" => "Too much request ! Please try later"
            );
        }
    }

```

```
84          else
85          {
86              $retour = array(
87                  "code" => 0,
88                  "msg" => "Registration success"
89              );
90          }
91      }
92
93      curl_close($cr);
94      return $retour;
95
96  }
97  ?>
```

4 Informations

Réalisé par Nathan Chevalier: nathan.chevalier@epita.fr
En réponse aux problématiques techniques posées par le projet P&P Forum