Kampfire API Integration Guide for Event Planning Software

Overview

The Kampfire API enables Event Planning Software (**EPS**) to easily integrate Kampfire's photo uploading and personalized album delivery features into their platform, giving users seamless access to Kampfire's services. The integration assumes that the EPS provides a webpage where Kampfire's iframe can be embedded, allowing users to upload event photos directly. Additionally, it assumes a landing page is available for event guests, where another Kampfire iframe can be embedded, enabling guests to upload their headshots and receive a personalized photo grid.

This API enables EPS to:

1. Set up a dedicated event workspace for each of their user's events, accessible via embedded iframes.
2. Embed an iframe that enables EPS users to upload photo galleries directly to their event's workspace.
3. Embed an iframe that allows users' guests to upload headshots and receive personalized photo grids in return.

Below is the integration guide for the EPS development team to seamlessly integrate Kampfire's limited features into their platform.

Authentication

All API requests to the Kampfire system must be authenticated using HMAC-SHA256 signatures.

How HMAC authentication works:

1. secretKey: each EPS receives a unique secret key, accessible under their profile in Kampfire's web app. This key is required to sign all requests.
2. Signing Data: the HMAC signature is generated by hashing the concatenated values of the HTTP method, URL, and request body, using the secret key.
3. Headers: the signed request must include five essential headers:

Header	Type	Description
X-Signature	String	The HMAC-SHA256 signature of the request
X-Timestamp	String	The UNIX timestamp in milliseconds (e.g., 1739544325876) indicating when the request was generated. It must be within 5 minutes of the current time.
X-Nonce	String	A unique, randomly generated value for each request to prevent replay attacks

X-Client-ID	String	The client id of the EPS
Content-Type	String	Must be application/json for this endpoint.

Endpoints

1. Creating a workspace

Creates a workspace that will serve as a container for event photos and personalized albums for each event hosted by an EPS user.

Endpoint

POST https://prod.eps-api.kampfire.events/api/v1/create-workspace

Headers

- X-Signature
- X-Timestamp
- X-Nonce
- X-Client-ID
- Content-Type: application/json

The client id is accessible under the profile in Kampfire's web app.

Request Body

Json

```
{
    "max_photos": "{{WORKSPACE_MAX_PHOTOS}}",
    "max_guests_albums": "{{WORKSPACE_MAX_GUEST_ALBUMS}}",
    "timestamp": "{{TIMESTAMP}}",
    "nonce": "{{NONCE}}",
}
```

Response Body

Json Stringified

```
{
    "status": "STATUS",
     "workspace_id": "{{WORKSPACE_ID}}",
    "kampaign_id": "{{KAMPAIGN_ID}}",
    "message": "MESSAGE",
}
```

The response returns unique `workspace_id` and 'kampaign_id', which are required for future interactions with Kampfire's API. status is either success/failure, and the message is a string.

For more details see the example below.

2. Photo Uploading via Kampfire iFrame Integration

In the EPS platform's CMS, the user uploads event photos into Kampfire through an iframe that displays the Kampfire speed photo-uploading pad.

iFrame URL

To integrate the Kampfire photo-uploading pad, embed the following iframe in the EPS user's CMS:

```
Html
<iframe
    src="https://eps.kampfire.events/eps/{{WORKSPACE_ID}}/eps-upload"
    allow="camera"
    style="min-height: 490px; width: 100%; border-style: none;">
</iframe>
```

Required Parameters

- WORKSPACE_ID: Obtained in Step 1.

Functionality

- The EPS user can upload thousands of photos via this iframe.
- The uploaded photos will automatically be added to the corresponding workspace in Kampfire.

3. Attendee Selfie Upload & Personalized Album Retrieval

On the event's landing page, the EPS user can embed another iframe that allows event attendees to upload a selfie. This selfie is processed by Kampfire to fetch personalized photos for the attendee.

iFrame URL

```
Html
<iframe

src="https://eps.kampfire.events/eps-headshot/{{WORKSPACE_ID}}/{{KAMPAIGN_ID}}"
    allow="camera"
    style="min-height: 490px; width: 400px; border-style: none;"
    title="">
</iframe>
```

Required Parameters

- WORKSPACE_ID: Obtained in Step 1.
- KAMPAIGN_ID: Obtained in Step 1.

Workflow

1. Attendee Uploads Selfie: The attendee uploads their selfie via the iframe.

2. Kampfire Fetches Personalized Photos: Kampfire processes the selfie and retrieves the personalized photos related to that attendee.
3. Photos Displayed in iFrame: The personalized photos are presented directly within the iframe, allowing attendees to view and download their pictures.

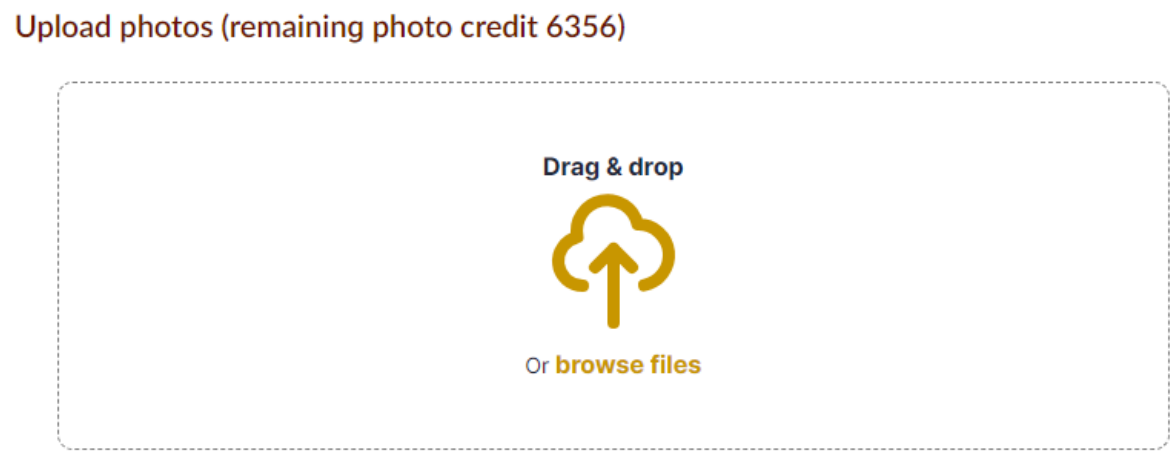
Error Handling

The API returns standard HTTP response codes to indicate the result of an API request:
- 200 OK: Request succeeded.
- 400 Bad Request: Invalid input or parameters.
- 401 Unauthorized: Authentication failed.
- 403 Forbidden: invalid timestamp or nonce.
- 404 Not Found: Resource not found.
- 500 Internal Server Error: Unexpected error on the server.

For additional support or queries, please contact the Kampfire technical team at [support@kampfire.events].

UI

Uploading photos Iframe in EPS' CMS:



Selfiepad in event's landing page:

Example

Example in Node.js for calling create-workspace:

```
const crypto = require('crypto');
const axios = require('axios');

// Sample request details
const method = 'POST';
const urlPath = '/api/v1/create-workspace';
const fullUrl = 'https://prod.eps-api.kampfire.events' + urlPath;
const clientId = 'your-client-id'; // Provided in the profile
const secretKey = 'your-secret-key'; // Provided in the profile

// Request body
const requestBody = {
  max_photos: 500,
  max_guests_albums: 100,
  timestamp: Date.now(),
  nonce: crypto.randomBytes(16).toString('hex'), // Random nonce
};

// Convert body to string for signing
```

```
const bodyString = JSON.stringify(requestBody);

// Concatenate method, URL path, body, and client ID
const dataToSign = method + urlPath + bodyString + clientId;

// Generate the HMAC signature
const signature = crypto.createHmac('sha256',
secretKey).update(dataToSign).digest('hex');

// Set the headers
const headers = {
  'Content-Type': 'application/json',
  'X-Client-ID': clientId,
  'X-Signature': signature,
  'X-Timestamp': requestBody.timestamp,
  'X-Nonce': requestBody.nonce,
};

// Make the API call
axios.post(fullUrl, requestBody, { headers })
  .then(response => {
    console.log('Response:', response.data);
  })
  .catch(error => {
    console.error('Error:', error.response ? error.response.data :
error.message);
  });
```
