Intégration de l'iFrame

Bénéfices de l'iFrame

Cette option permet à l'acheteur de rester sur un environnement connu et de confiance.

Le tunnel de paiement est fluide et les abandons paniers plus rares.

Il est possible d'utiliser un Template personnalisé pour le formulaire de carte qui sera intégré au sein de la page de paiement. Pour plus de détails : Mise en place d'un Template Personnalisé

L'iframe est web responsive et peut être intégré dans une application mobile.

Table des matières

Synthèse de la mise en place de l'iframe

- Mise en place de l'iframe dans la *page de paiement du marchand*
- Configuration des URLs passée dans l'objet "urls" de la requète de paiement
  - Préparation des paramètres `urls.return` et `urls.cancel` avec les URL de retour en paiement abouti ou abandonné.
  - Placement de ces 2 paramètres dans le hash du `paramètre Data`
  - Passage de `Data` en paramètre de l'URL fournie à l'attribut `src` de l'iframe
  - Ajout du paramètre `CustomField14=iframe` pour avoir uniquement l'affichage du bloc de paiement
- Création des pages 'Succès' et 'Échec'
  - Préparation du code HTML de `redirect-success` et `redirect-failure`
- Mise en place de la redirection en JavaScript
  - Intégration de la méthode `postMessage` dans `redirect-success` et `redirect-failure`
  - Intégration de l'écouteur d'évenement (Event Listener) dans la *page de paiement du marchand*

Mise en place de l'iframe et configuration de l'URL passée à l'atttribut src

Balise iframe & customField14

Une iframe est une balise HTML qui permet d'intégrer une page HTML dans une autre.
Il faut pour cela utiliser la **balise <iframe>** et **spécifier l'URL** de la page qu'on souhaite afficher dans **l'attribut src.** L'URL de la page de paiement doit contenir la valeur suivante : **customField14=iframe**.

Exemple de selection de l'iFrame dans une requete de paiement en utilisant le customField14

```
        ...
    "template" :  {
          "customFields": {
                "customField14": "iframe"
            }
    },
        ...
```

Exemple d'implémentation du formulaire au sein d'une page HTML

Exemple d'intégration du formulaire carte au sein d'une page HTML

```
<!DOCTYPE html>
<HTML lang="en">
  <HEAD>
    <META charset="UTF-8" />
    <META http-equiv="X-UA-Compatible" content="IE=edge" />
    <META name="viewport" content="width=device-width, initial-scale=1.0" />
    <TITLE>Site marchand - commande</TITLE>
  </HEAD>
  <BODY>
    ...
    <IFRAME
      src="https://paymentpage.axepta.bnpparibas/payssl.aspx?token=YOUR_PAYMENT_TOKEN"
    />
    ...
  </BODY>
</HTML>
```

Réglages : Hauteur & Largeur

En pixel

```
<IFRAME
  src="https://paymentpage.axepta.bnpparibas/payssl.aspx?token=YOUR_PAYMENT_TOKEN"
  height="600"
  width="800"
/>
```

En CSS

```
#payment-iframe {
  height: 600px;
  width: 800px;
}

// ou pour un rendu fullscreen
#payment-iframe {
  height: 100vh;
  width: 100vw;
}

<IFRAME
  id="payment-iframe"
  src="https://paymentpage.axepta.bnpparibas/payssl.aspx?token=YOUR_PAYMENT_TOKEN"
/>
```

Création des pages 'Return'

Quand le porteur soumet le formulaire de paiement, une requête est envoyée au serveur Axepta.

Suivant si l'utilisateur poursuit le paiement ou l'abandonne, la réponse est renvoyé sur l'`url return` ou l'`url cancel`.

Si le paiement va à son terme (avec succès ou pas), une réponse est envoyée à l'`url return`.
Si le titulaire de la carte annule le paiement, une réponse est envoyée à l'`url cancel`.

L'objectif est d'afficher une page de succès ou d'échec à la place de l'iFrame en fonction du résultat du paiement.

Dans le cas normal où le paiement va à son terme, la réponse sur l'url return n'est pas suffisante pour déterminer si le paiement est un succès ou si il a été refusée.

Il faut demander au serveur Axepta le détail de la transaction, en utilisant **l'api getByPayId**
(plus de détail sur cette page: Gestion des callbacks et notification de paiement)

> Remarque : le serveur renvoie une réponse HTTP avec la méthode POST. On doit donc créer une route sur le serveur du site marchand pour traiter cette réponse.

Si on souhaite rediriger l'utilisateur sur une page du site marchand affichée à la place de l'iframe, il faudra prévoir une page de redirection.

En fonction du résultat du détail de la transaction, on aura:

url de retour Axepta	urls.return		urls.cancel
getByPayId	OK	ERROR	n/a
redirection site marchand	Paiement succès	Paiement échec	Paiement échec

Le marchand créé ces pages qui seront affichées lors de la redirection du porteur.

Par exemple,

- Paiement succès : https://mon-site.com/redirect-success
- Paiement en échec : https://mon-site.com/redirect-failure

Chacune exécutera un script qui redirigera l'utilisateur vers la page en succès ou en erreur de paiement.

La mise en place de ce script est indiqué dans la section suivante.

Mise en place de la redirection en JavaScript

Intégration de la méthode `postMessage` dans `redirect-success` et `redirect-failure`

La solution `postMessage` "*permet une communication inter-domaine en toute sécurité*".

L'objectif est de "communiquer" avec la page parente. Cette communication se fera grâce à un événement de type `MessageEvent`

Url return (success - failed) intégration backend integration

```
const express = require('express');

const app = express();
const PORT = 3000;
```

```
const API_BASE_URL = 'https://paymentpage.axepta.bnpparibas';
const BEARER_TOKEN = 'YOUR_BEARER_TOKEN_HERE';

// Template function to generate HTML response based on payment status
function renderPaymentPage(payId, status, isSuccess) {
  const title = isSuccess ? 'Payment Successful' : 'Payment Failed';
  const message = isSuccess ? 'Payment Successful!' : 'Payment Failed';
  const eventType = isSuccess ? 'PAYMENT_SUCCESS' : 'PAYMENT_FAILED';
  const redirectUrl = isSuccess
    ? `/order/confirmation?payId=${payId}`
    : `/payment/retry?payId=${payId}`;

  return `
    <!DOCTYPE html>
    <html>
    <head>
      <title>${title}</title>
    </head>
    <body>
      <h1>${message}</h1>
      <p>Transaction ID: ${payId}</p>
      <script>
        // If page is loaded inside an iframe, notify the parent window
        if (window.parent !== window) {
          window.parent.postMessage({
            type: '${eventType}',
            payId: '${payId}',
            status: '${status}'
          }, '*');
        }

        // Redirect user after 3 seconds
        setTimeout(() => {
          window.location.href = '${redirectUrl}';
        }, 3000);
      </script>
    </body>
    </html>
  `;
}

// Return URL endpoint - This is where Axepta redirects customers after payment
app.get('/payment/return', async (req, res) => {
  try {
    // Step 1: Extract PayID from URL query parameters
    // Example: /payment/return?PayID=abc123
    const payId = req.query.PayID;

    // Validate that PayID exists
    if (!payId) {
      return res.status(400).send('PayID is missing');
    }

    // Step 2: Call Axepta API to retrieve payment details
    const response = await fetch(
      `${API_BASE_URL}/api/v2/payments/getByPayId/${payId}`,
      {
        method: 'GET',
        headers: {
          'Authorization': `Bearer ${BEARER_TOKEN}`,
          'Content-Type': 'application/json'
        }
      }
    );

    // Check if API request was successful
    if (!response.ok) {
      throw new Error(`API returned status ${response.status}`);
    }
```

```
    // Parse JSON response from API
    const paymentData = await response.json();
    const status = paymentData.status;

    // Step 3: Determine if payment was successful
    const isSuccess = status === 'SUCCESS' || status === 'AUTHORIZED';

    // Step 4: Send HTML response to customer
    res.send(renderPaymentPage(payId, status, isSuccess));

  } catch (error) {
    // Log error for debugging
    console.error('Error retrieving payment details:', error.message);

    // Return simple error message to customer
    res.status(500).send('Server error');
  }
});

// Start the server
app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

La page d'échec est également à créer.

Intégration de l'écouteur d'évenement (Event Listener) dans la page de paiement

Dans la page qui contient l'iframe, on va mettre en place un écouteur d'événement. Celui-ci sera chargé d'exécuter une fonction quand il recevra un événement de type `message`.

Modification du code de la page qui intégre l'iframe - Ajout EventListener

```
<!DOCTYPE html>
<HTML lang="en">

<HEAD>
   <META charset="UTF-8" />
   <META http-equiv="X-UA-Compatible" content="IE=edge" />
   <META name="viewport" content="width=device-width, initial-scale=1.0" />
   <TITLE>Site marchand - commande</TITLE>
</HEAD>

<BODY>
   <SCRIPT>
      // sur la page courante
      // on place un écouteur avec la méthode addEventListener
      window.addEventListener(
         // type d'évenement à écouter
         "message",
         // fonction de retour (callback) qui sera exécutée
         // quand un évenement de type "message" sera émis
         (event) => {
            // Si l'origin n'est pas la même que celle placée dans le message
            // on arrête l'exécution de la fonction
            if (event.origin !== "https://mon-site.com") return;

            const data = event.data;
            // on redirige l'utilisateur sur la page souhaitée
            if (data.type === "PAYMENT_SUCCESS") {
               window.location.href = "https://mon-site.com/payment-success";
            } else {
               window.location.href = "https://mon-site.com/payment-failed";
            }
         }
      );
   </SCRIPT>
   ...
   <IFRAME src="https://paymentpage.axepta.bnpparibas/payssl.aspx?token=YOUR_PAYMENT_TOKEN" />
   ...
</BODY>

</HTML>
```

Customisation dynamique

D'autres champs personnalisés peuvent être ajoutés à la requête afin de changer dynamiquement l'iframe.

Veuillez trouver ci-dessous les champs disponibles :

- CustomField100 : couleur du background
- CustomField101 : couleur du texte des boutons lorsqu'ils est actif
- CustomField102 : couleur du background des boutons lorsqu'il est actif
- CustomField103 : couleur au hover du texte des boutons lorsque le bouton est actif
- CustomField104 : couleur au hover du background des boutons lorsque le bouton est actif
- CustomField105 : couleur du texte des bouton lorsqu’ils sont inactifs
- CustomField106 : couleur du fond des boutons lorsqu’ils sont inactifs

Les valeurs **hexadécimales** des couleurs doivent être utilisées dans les champs CustomField et le symbole **'#' doit être encodé**, il sera remplacé par la valeur '%23'.

Ainsi pour afficher un bouton rouge (quand celui-ci est activé), il faudra valoriser le CustomField102 de la manière suivante : CustomField102=%23FF0000.