Réussir l'intégration du paiement mobile via webview

Guide d'implémentation

Version du document 1.4

Sommaire

1. HISTORIQUE DU DOCUMENT

Version	Auteur	Date	Commentaire
1.4	Natixis Payment Solutions	16/10/2019	Version initiale

2. PRÉSENTATION

Systempay vous propose une solution unique pour l'intégration du paiement mobile à vos applications.

Notre solution couvre les applications natives iOS et Android. Elle est basée sur l'utilisation du composant **webview**.

Une webview permet d'afficher du contenu déjà disponible sur le web au sein de l'application.

Ainsi, la solution Systempay de paiement mobile via webview offre plusieurs avantages au marchand :

- Une configuration unique pour le web et le mobile.

  Vous pouvez réutiliser à l'identique la configuration des paiements de votre site web.

  Les moyens de paiement activés, les règles anti-fraudes, etc. sont repris dans l'application mobile.

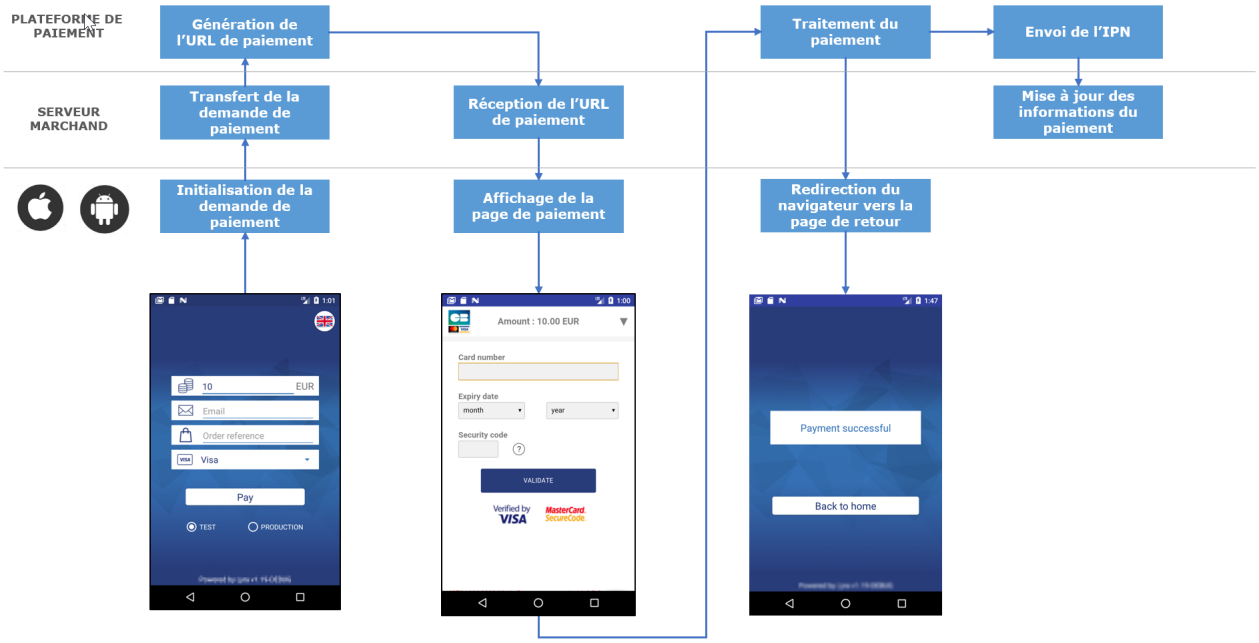
- Une cohérence dans l'affichage des informations du parcours acheteur.

  Nos pages de paiement sont responsives et donc, capables de s'adapter aux différents terminaux de vos clients (mobile, tablette ou ordinateur de bureau).

- Une haute sécurité grâce, d'une part, à notre certification PCI DSS et d'autre part à la gestion du 3DS intégrée dans le parcours de paiement.

PCI DSS (= Payment Card Industry Data Security Standard) est la norme de sécurité de l'industrie des cartes de paiement. C'est un standard de sécurité des données pour les principaux groupes de cartes de paiement tels que Visa, MasterCard, American Express, Discover et JCB.

3. CINÉMATIQUE DU PAIEMENT



L'acheteur valide son panier.

1. L'application mobile initialise une demande de paiement auprès du serveur marchand.
2. Le serveur marchand envoie une demande de paiement à la plateforme.
3. La plateforme génère une URL de paiement et la transmet en retour à l'application mobile.
4. Le serveur marchand envoie l'URL de paiement à l'application mobile.
5. L'application mobile ouvre la page de paiement dans une webview.
6. L'acheteur saisit les données de sa carte puis clique sur **Valider**.
7. La plateforme procède au paiement, puis transmet la notification de paiement au serveur marchand.
8. Le serveur marchand analyse le résultat du paiement.
9. L'acheteur est redirigé automatiquement vers l'application du marchand.

4. INTÉGRATION DU PAIEMENT

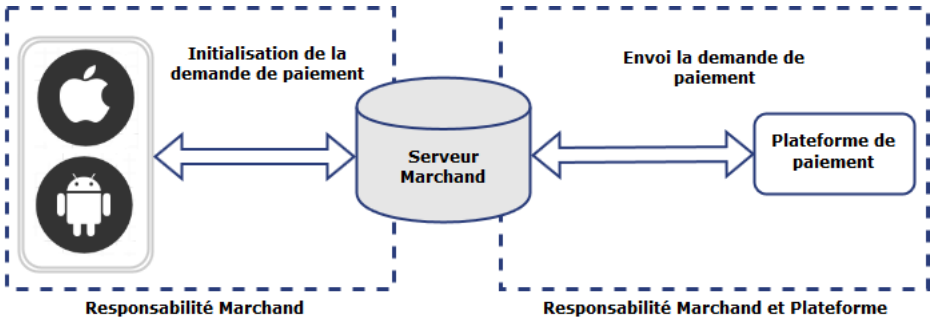
Des exemples de codes sont mis à disposition pour faciliter l'intégration :

Serveur marchand *https://github.com/lyra/webview-payment-sparkjava-integration-sample*
iOS *https://github.com/lyra/webview-payment-ios-integration-sample*
Android *https://github.com/lyra/webview-payment-android-integration-sample*

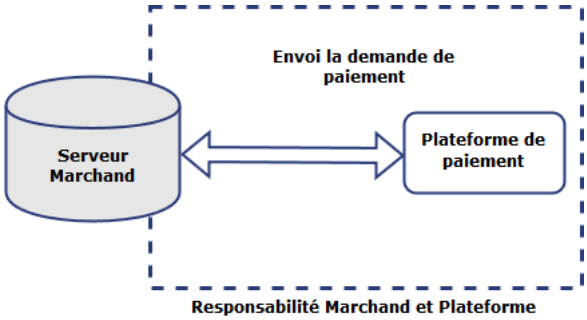
IMPORTANT
Assurez-vous d'avoir pris connaissances des commentaires présents dans les fichiers readme avant de lancer l'application. Les fichiers `MainActivity.kt` et `app-configuration.properties` doivent etre modifiés suivants les instructions fournies dans les commentaires.

L'intégration se déroule en deux phases:

- intégration des échanges entre le serveur marchand et la plateforme de paiement
- intégration des échanges entre l'application mobile et le serveur marchand.



5. PHASE 1: LE SERVEUR MARCHAND



5.1. Transfert de la demande de paiement

Le serveur marchand reçoit une demande de paiement de la part de l'application mobile et doit la transmettre à la plateforme de paiement.

Pour cela, le site marchand va générer un formulaire de paiement HTML qu'il postera à la plateforme de paiement.

L'intégrité des données échangées est garantie par un échange de signatures alphanumériques entre la plateforme de paiement et le serveur marchand.

Le serveur marchand transmettra la signature alphanumérique dans le formulaire de paiement (voir chapitre **Calculer la signature** du **Guide d'implémentation API Formulaire** disponible sur notre site documentaire).

IMPORTANT
Toutes les données de votre formulaire doivent être encodées en UTF-8.
Les caractères spéciaux (accents, ponctuations, etc.) seront ainsi correctement interprétés par la plateforme de paiement.
Dans le cas contraire, le calcul de signature sera erroné et le formulaire sera rejeté.

1. Création du formulaire de paiement

Pour créer le formulaire de paiement :

1. Utilisez l'ensemble des champs présents dans ce tableau pour construire la demande de paiement.

Nom du champ	Description	Format	Valeur
vads_site_id	Identifiant de la boutique	n8	Ex : 12345678
vads_currency	Code numérique de la monnaie à utiliser pour le paiement, selon la norme ISO 4217 (code numérique)	n3	Ex : 978 pour l'euro (EUR)
vads_amount	Montant du paiement dans sa plus petite unité monétaire (le centime pour l'euro)	n..12	Ex : 3000 pour 30,00 EUR
vads_cust_email	Adresse e-mail de l'acheteur	ans..150	Ex: abc@example.com
vads_payment_cards	Type de carte.	String	Ex: **VISA** (Consultez le Guide d'implémentation API Formulaire pour connaître la liste des valeurs possibles).
vads_order_id	Numéro de commande	ans..64	Ex : 2-XQ001
vads_version	Version du protocole d'échange avec la plateforme de paiement	enum	**V2**

Nom du champ	Description	Format	Valeur
vads_theme_config	Permet de gagner en performances en désactivant des éléments de la page de paiement comme le sélecteur de langue, les logos du bas de page, etc.	map	**SIMPLIFIED_DISPLAY=true**
vads_trans_date	Date et heure du formulaire de paiement dans le fuseau horaire UTC	n14	Respectez le format AAAAMMJJhhmmss Ex : 20170701130025
vads_trans_id	Numéro de la transaction	n6	Ex : 123456
vads_payment_config	Type de paiement	enum	**SINGLE** pour un paiement en 1 fois **MULTI** pour un paiement en plusieurs fois
vads_page_action	Action à réaliser	enum	**PAYMENT**
vads_ctx_mode	Mode de communication avec la plateforme de paiement	enum	**TEST** ou **PRODUCTION**
vads_action_mode	Mode d'acquisition des données de la carte	enum	**INTERACTIVE**
signature	Signature garantissant l'intégrité des requêtes échangées entre le site marchand et la plateforme de paiement.	ans44	Ex: NrHSHyBBBc +TtcauudspNHQ5cYcy4tS4IjvdC0ztFe8=

2. Utilisez les champs ci-dessous pour gérer le retour vers l'application mobile à la fin du paiement.

   Un paiement peut aboutir sur 4 états différents :

   - Paiement accepté
   - Paiement refusé
   - Paiement en erreur
   - Paiement abandonné par l'acheteur.

   Pour chaque état vous devez associer une URL:

Nom du champ	Description	Format	Valeur
vads_url_success	URL où sera redirigé l'acheteur, **en cas de succès**.	ans..1024	Ex: **http://webview.success**
vads_url_refused	URL où sera redirigé l'acheteur, **en cas de refus**.	ans..1024	Ex: **http://webview.refused**
vads_url_cancel	URL où sera redirigé l'acheteur **en cas d'abandon** ou d'expiration (timeout).	ans..1024	Ex: **http://webview.cancel**
vads_url_error	URL où sera redirigé l'acheteur **en cas d'erreur**.	ans..1024	Ex: **http://webview.error**

3. Utilisez les champs ci-dessous pour configurer les temps de redirection vers l'application mobile à la fin du paiement:

Nom du champ	Description	Format
vads_redirect_success_timeout	Définit le délai d'attente avant redirection après un paiement réussi. Ce délai est exprimé en seconde et doit être compris entre 0 et 300 secondes. **Valorisez ce champ à "0" pour ne pas afficher le ticket de paiement et rediriger automatiquement l'acheteur vers l'application mobile.**	n..3
vads_redirect_error_timeout	Définit le délai d'attente avant redirection après un paiement refusé. Ce délai est exprimé en seconde et doit être compris entre 0 et 300 secondes.	n..3

Nom du champ	Description	Format
	Valorisez ce champ à "0" pour ne pas afficher la page de refus de paiement et rediriger automatiquement l'acheteur vers l'application mobile.	

Tableau 1 : Liste des champs facultatifs disponibles.

4. Ajoutez les autres champs optionnels en fonction de vos besoins (voir chapitre **Utiliser des fonctions complémentaires** du **Guide d'implémentation API Formulaire** disponible sur notre site documentaire).

5. Consulter le chapitre **Calculer la signature** du **Guide d'implémentation API Formulaire** et calculez la valeur du champ **signature**.

2. Envoi de la demande de paiement

L'API de création de paiement est disponible en mode POST à cette adresse :

https://paiement.systempay.fr/vads-payment/entry.silentInit.a

IMPORTANT
L'URL de l'API de création de paiement est différente de l'URL de la page de paiement, telle que décrite dans le Guide d'implémentation API Formulaire.

Extrait du code d'exemple:

```
List<NameValuePair> formParameters = new ArrayList<>();

formParameters.add(new BasicNameValuePair("vads_action_mode", "INTERACTIVE"));
formParameters.add(new BasicNameValuePair("vads_amount", amount));
formParameters.add(new BasicNameValuePair("vads_ctx_mode", mode));
formParameters.add(new BasicNameValuePair("vads_currency", currency));
if (StringUtils.isNotEmpty(email)) {
formParameters.add(new BasicNameValuePair("vads_cust_email", email));
}

formParameters.add(new BasicNameValuePair("vads_language", language));
if (StringUtils.isNotEmpty(orderId)) {
 formParameters.add(new BasicNameValuePair("vads_order_id", orderId));
}
formParameters.add(new BasicNameValuePair("vads_page_action", "PAYMENT"));

//Set the card type if provided
if (StringUtils.isNotEmpty(cardType)) {
 formParameters.add(new BasicNameValuePair("vads_payment_cards", cardType.toUpperCase()));
}
formParameters.add(new BasicNameValuePair("vads_payment_config", "SINGLE"));
formParameters.add(new BasicNameValuePair("vads_site_id", merchantSiteId));
formParameters.add(new BasicNameValuePair("vads_theme_config", "SIMPLIFIED_DISPLAY=true"));
formParameters.add(new BasicNameValuePair("vads_trans_date",
 calculateDateFormatInUTC("yyyyMMddHHmmss")));
formParameters.add(new BasicNameValuePair("vads_trans_id", String.format("%06d",
 transactionId)));
formParameters.add(new BasicNameValuePair("vads_url_cancel", "http://webview_" +
 merchantSiteId + ".cancel"));
formParameters.add(new BasicNameValuePair("vads_url_error", "http://webview_" +
 merchantSiteId + ".error"));
formParameters.add(new BasicNameValuePair("vads_url_refused", "http://webview_" +
 merchantSiteId + ".refused"));
formParameters.add(new BasicNameValuePair("vads_url_return", "http://webview_" +
 merchantSiteId + ".return"));
formParameters.add(new BasicNameValuePair("vads_url_success", "http://webview_" +
 merchantSiteId + ".success"));
formParameters.add(new BasicNameValuePair("vads_version", "V2"));

//Create the string to sign
String concatenateMapParams = "";
for (NameValuePair pair : formParameters) {
 concatenateMapParams += pair.getValue() + "+";
}
//Add private key in signature
concatenateMapParams += usedMerchantKey;
```



```
//Add signature to form parameters
formParameters.add(new BasicNameValuePair("signature", hmacSha256(concatenateMapParams,
 usedMerchantKey)));
```

5.2. Réception de l'URL de paiement

La plateforme de paiement retourne une réponse au format JSON contenant un code d'état HTTP de succès ou d'erreur.

Succès

En cas de succès, la plateforme de paiement renvoie un code d'état HTTP `200 OK`.

La réponse contient l'URL de paiement vers laquelle l'application mobile doit rediriger l'acheteur.

```
{
 "status":"INITIALIZED",
 "redirect_url":"https://paiement.systempay.fr:443/vads-payment/
exec.refresh.a;jsessionid=CE2Cb9daEDe7f6dBF31FE65e.vadpayment01bdx"
}
```

Erreur

En cas d'erreur, la plateforme de paiement renvoie un code d'état HTTP `400 Bad Request` ou `500 Internal Server Error`.

La réponse contiendra le détail de l'erreur.

```
{
 "status":"ERROR",
 "error":" {"code": "09", "value": "Missing or invalid parameter value"}"
}
```

Pour plus de détails, consultez la liste des codes d'erreur de l'API Formulaire:

https://paiement.systempay.fr/doc/fr-FR/error-code/sitemap.html

Extrait du code d'exemple:

```
//If the HTTP return code is 200 (OK) we prepare the generated URL
if (httpResponseCode == 200) {
 if ("INITIALIZED".equals(responseData.get("status"))) {
  redirectionUrl = responseData.get("redirect_url");
 } else {
  //Payment could  not be created. Maybe a missing parameter, an invalid value or signature?
  //Use logs here in order to detect and fix the real cause
  throw new RuntimeException("Error in payment initialization. Returned error: " +
 responseData.get("error"));
 }
} else {
 throw new RuntimeException("Error in payment initialization. HTTP errorCode: " +
 httpResponse);
}
```

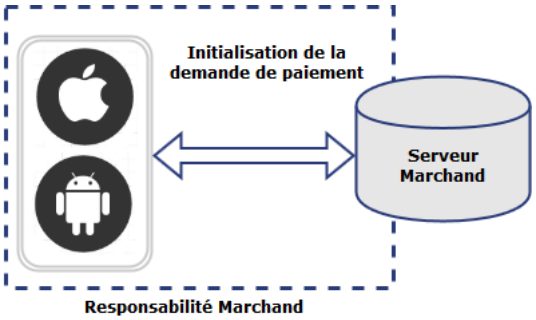
5.3. Traitement de la notification de fin de paiement (IPN)

Une fois le paiement effectué, la plateforme de paiement notifie le serveur marchand du résultat de la transaction.

Les données seront envoyées sur l'URL de notification définie dans le .

Consultez le **Guide d'implémentation API Formulaire**, disponible sur notre site documentaire, pour le paramétrage des règles de notification et l'analyse des données transmises.

6. PHASE 2: L'APPLICATION MOBILE



6.1. Initialisation de la demande de paiement

Lorsque l'acheteur valide sa commande l'application génère une "payload" contenant les données du panier, les coordonnées de l'acheteur, les informations de livraison etc.

L'exemple mis à disposition exploite les données suivantes:

```
{
 "email": "example@email.com",
 "orderId": "myOrderId-1",
 "amount": "200",
 "currency": "978",
 "mode": "TEST",
 "language": "fr",
 "cardType": ""
}
```

L'application mobile transmet la demande de paiement au serveur marchand via une requête POST.

Extrait du code d'exemple pour Android:

```
val conn = URL(serverUrl).openConnection() as HttpURLConnection
conn.requestMethod = "POST"
conn.setRequestProperty("Content-type", "application/json")
conn.setRequestProperty("Accept", "*/*")
conn.doInput = true
conn.doOutput = true
conn.connectTimeout = 15000

val os = conn.outputStream
val writer = BufferedWriter(OutputStreamWriter(os, "UTF-8"))
writer.write(payload.toString())
writer.flush()
writer.close()
os.close()

conn.connect()

val out = OutputStreamWriter(conn.outputStream)
```


Extrait du code d'exemple pour iOS:

```
/// Build an URLRequest according required payment information : server url, email, amount,
 mode, lang
///
/// - Returns: URLRequest object
func buildRequest() -> URLRequest? {
 let serverUrl: NSURL = NSURL(string: PaymentProvider.SERVER_URL)!
 var urlRequest = URLRequest(url:serverUrl as URL)
 urlRequest.httpMethod = "POST"
 var params: [String: String] = ["amount": paymentInfo.amount, "currency":
 paymentInfo.currency, "mode": paymentInfo.mode, "language": paymentInfo.lang]
 if !paymentInfo.email.isEmpty{
  params["email"] = paymentInfo.email
 }
 if !paymentInfo.cardType.isEmpty{
  params["cardType"] = paymentInfo.cardType
 }
 do{
  let jsonParam = try JSONSerialization.data(withJSONObject: params, options: [])
  urlRequest.httpBody = jsonParam
 }
 catch{
  return nil
 }

 return urlRequest
}
```

6.2. Affichage de la page de paiement dans une web view

Une fois la demande traitée, la plateforme de paiement retourne l'URL de paiement à l 'application mobile.

L'application initialise une webview et affiche la page de paiement.

Extrait de l'exemple de code pour Android (**Kotlin**)

```
val webView = WebView(this)

// Url loading
webView.loadUrl(url)

// Enable javascript
webView.settings.javaScriptEnabled = true

// To allow debug WebView from Chrome Dev Tools
WebView.setWebContentsDebuggingEnabled(false)

// Define new web view client by overriding shouldOverrideUrlLoading method in order to check
 urls
webView.webViewClient = object: WebViewClient() {
 override fun onPageFinished(view: WebView, url: String) {
  progressBar.visibility = View.GONE
  super.onPageFinished(view, url)
 }

 @Suppress("OverridingDeprecatedMember")
 override fun shouldOverrideUrlLoading(view: WebView, url: String): Boolean {
  return checkUrl(webView, url)
 }

 @TargetApi(Build.VERSION_CODES.LOLLIPOP)
 override fun shouldOverrideUrlLoading(view: WebView, webResourceRequest: WebResourceRequest):
 Boolean {
  return checkUrl(webView, webResourceRequest.url.toString())
 }
}
webView.canGoForward()
```


Extrait de l'exemple de code pour iOS (**Swift**)

```
/// Call server to get payment url, supply a block completion (callback)
///
/// - Returns: status boolean, payment url
func getPaymentContext(completion: @escaping (Bool, String, NSError?) -> ()){
 // Build request
 let urlRequest = buildRequest()
 // Call server to obtain a payment Url
 // Completion is a callback, giving call status, and payment url if success
 if let request = urlRequest{
  let task = URLSession.shared.dataTask(with: request) { (data: Data?, response: URLResponse?,
 error: Error?) in
   if error != nil{
    completion(false, "", NSError.init(domain:PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_NO_CONNECTION.errorCode, userInfo: [NSLocalizedFailureReasonErrorKey:
 PaymentProvider.ERROR_NO_CONNECTION.errorMsg]))
   }
   if let httpResponse = response as? HTTPURLResponse {
    let json = try? JSONSerialization.jsonObject(with: data!, options: .mutableContainers) as?
 NSDictionary
    var redirectionUrl = ""
    var errorMsg = ""
    if let jsonResponse = json {
     redirectionUrl = (jsonResponse!["redirectionUrl"] as? String)!
     errorMsg = (jsonResponse!["errorMessage"] as? String)!
    }
    switch(httpResponse.statusCode){
     case 200:
      completion(true, redirectionUrl, nil)
     case 400, 500:
      completion(false, "", NSError.init(domain:PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_SERVER.errorCode, userInfo: [NSLocalizedFailureReasonErrorKey:
 PaymentProvider.ERROR_SERVER.errorMsg + errorMsg]) )
     default:
      completion(false, "", NSError.init(domain:PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_UNKNOW.errorCode, userInfo: [NSLocalizedFailureReasonErrorKey:
 PaymentProvider.ERROR_UNKNOW.errorMsg]))
    }
   }
   else{
    completion(false, "", NSError.init(domain:PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_TIMEOUT.errorCode, userInfo: [NSLocalizedFailureReasonErrorKey:
 PaymentProvider.ERROR_TIMEOUT.errorMsg]))
   }
 }
 task.resume()
 } else{
  completion(false, "", NSError.init(domain:PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_UNKNOW.errorCode, userInfo: [NSLocalizedFailureReasonErrorKey:
 PaymentProvider.ERROR_UNKNOW.errorMsg]))
 }
}
```

6.3. Détection de la fin de paiement

Afin de détecter la fin du paiement, il est nécessaire d'analyser les différentes URLs qui passent par la webview.

En fonction de l'URL, l'application mobile pourra:

- accepter la propagation de l'URL et afficher la page dans la webview

  (par exemple, lors d'une demande de paiement, pour permettre à l'acheteur d'effectuer son paiement)

- refuser la propagation de l'URL et reprendre la main.

  (par exemple, en cas de succès et à la fin du paiement, pour permettre à l'acheteur de retourner à l'application native)

Grâce à un mécanisme d'écoute des URLs, vous avez le contrôle sur la cinématique de paiement et pouvez décider à quel moment basculer sur votre application native.

Extrait de l'exemple de code pour Android (Kotlin)

```
private fun checkUrl(view: WebView, url: String): Boolean {
 val isCallBack = isCallbackUrl(url)

 when {
  // payment is finish
  isCallBack -> {
   view.stopLoading()
   goToFinalActivity(url)
  }
  else -> view.loadUrl(url)
 }
 return (!isCallBack)
}
```

Extrait de l'exemple de code pour iOS (**Swift**) :

```
func notifyPaymentFinish(navigationAction: WKNavigationAction){
 let webViewUrlResponse = self.buildWebviewUrlResponse(navigationAction: navigationAction)
 var error: NSError?
 switch webViewUrlResponse.paymentStatus {
  case "success":
   error = nil
  case "cancel":
   error = NSError.init(domain: PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_PAYMENT_CANCELATION.errorCode, userInfo:
 [NSLocalizedFailureReasonErrorKey: PaymentProvider.ERROR_PAYMENT_CANCELATION.errorMsg])
  case "refused":
   error = NSError.init(domain:PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_PAYMENT_REFUSED.errorCode, userInfo: [NSLocalizedFailureReasonErrorKey:
 PaymentProvider.ERROR_PAYMENT_REFUSED.errorMsg])
  default:
   error = NSError.init(domain:PaymentProvider.ERROR_DOMAIN, code:
 PaymentProvider.ERROR_UNKNOW.errorCode, userInfo: [NSLocalizedFailureReasonErrorKey:
 PaymentProvider.ERROR_UNKNOW.errorMsg])

 }
 self.dismiss(animated: true) {
  self.paymentDelegate?.didPaymentProcessFinish(error: error)
 }
}
```