

 eBook Gratuit

APPRENEZ

PayPal

eBook gratuit non affilié créé à partir des
contributeurs de Stack Overflow.

#paypal

Table des matières

| | |
|---|-----------|
| À propos..... | 1 |
| Chapitre 1: Démarrer avec PayPal..... | 2 |
| Remarques..... | 2 |
| Versions..... | 2 |
| Exemples..... | 2 |
| Création d'une application et obtention d'un identifiant client / de clés secrètes..... | 2 |
| Configuration des comptes de test utilisateur sandbox..... | 4 |
| Chapitre 2: Création d'abonnements / paiements récurrents..... | 5 |
| Paramètres..... | 5 |
| Remarques..... | 5 |
| Exemples..... | 6 |
| Étape 2: Création d'un abonnement pour un utilisateur à l'aide d'un accord de facturation | 6 |
| Étape 1: Création d'un modèle d'abonnement à l'aide d'un plan de facturation (exemple de n..... | 9 |
| Chapitre 3: Effectuer un paiement par carte de crédit (noeud)..... | 12 |
| Paramètres..... | 12 |
| Remarques..... | 12 |
| Exemples..... | 12 |
| Exemple de noeud..... | 12 |
| Effectuer un paiement avec une carte de crédit voûtée (noeud)..... | 15 |
| Chapitre 4: Effectuer un paiement PayPal..... | 18 |
| Paramètres..... | 18 |
| Remarques..... | 18 |
| Exemples..... | 18 |
| Exemple de serveur Node Express..... | 18 |
| Chapitre 5: Paiements futurs mobiles (application de bout en bout)..... | 22 |
| Remarques..... | 22 |
| Exemples..... | 22 |
| Etape 1: mise en page, initialisation et gestion des réponses du serveur..... | 22 |
| Android Étape 2: Demande de serveur asynchrone..... | 24 |
| Etape 3 d'Android: Serveur de noeud pour obtenir le paiement de jeton et de processus d'ac..... | 25 |

| | |
|--|-----------|
| Chapitre 6: Paypal Mobile / Paiements par carte de crédit | 28 |
| Paramètres..... | 28 |
| Remarques..... | 28 |
| Exemples..... | 28 |
| Android: accepter un paiement PayPal / carte de crédit..... | 28 |
| Chapitre 7: Webhooks | 32 |
| Paramètres..... | 32 |
| Remarques..... | 32 |
| Exemples..... | 32 |
| Test des Webhooks Sandbox avec ngrok et Express (Node)..... | 32 |
| Mise à jour d'un Webhook avec une nouvelle URL (exemple de noeud)..... | 36 |
| Crédits | 38 |

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [paypal](#)

It is an unofficial and free PayPal ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official PayPal.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec PayPal

Remarques

Ces guides guideront l'utilisateur dans les procédures de configuration de compte pour les applications, les comptes, etc. Il contiendra tout ce qui est nécessaire pour travailler avec les API de PayPal.

Versions

| Version | Date de sortie |
|---------|----------------|
| 1.0.0 | 2016-04-11 |

Exemples

Création d'une application et obtention d'un identifiant client / de clés secrètes

Pour commencer à créer avec les API PayPal, vous devez créer une application pour obtenir un identifiant client et un secret.

Accédez à <https://developer.paypal.com/developer/applications/> , connectez-vous et cliquez sur "Créer une application", comme indiqué ci-dessous:

REST API apps

Create an app to receive REST API credentials for testing and live transactions.

Note Features available for live transactions are listed in your [account eligibility](#).

Create App



App name

My test app

My test app 1

MyLiveApp

Ensuite, entrez un nom d'application, sélectionnez le compte de test sandbox que vous souhaitez utiliser (s'il s'agit d'un nouveau compte, laissez la valeur par défaut), puis cliquez sur "Créer une application".

Application Details

App Name

Sandbox developer account

As a reminder, all apps created under your account should be related to your business and t
By clicking the button below, you agree to [PayPal Developer Agreement](#).

Create App

Une fois l'application créée, vous recevrez votre sandbox et votre identifiant client et secret, qui ressembleront à ceci:

SANDBOX API CREDENTIALS

Sandbox account

test232213@testing.com

Client ID

ATwkJTgxN3

Secret

Hide

Note: There can only be a maximum of two client-secrets. These client-secrets can be in eit

| Created | Secret |
|---------|--------|
|---------|--------|

| | |
|--------------|-------------|
| Apr 11, 2016 | EJqSRO4Gj5s |
|--------------|-------------|

Generate New Secret

Ces informations d'identification sont celles que vous utiliserez lors de demandes adressées aux API PayPal afin d'authentifier votre application et de faire des demandes.

Configuration des comptes de test utilisateur sandbox

Lorsque vous testez votre intégration PayPal sur sandbox, vous devez configurer des comptes utilisateur sandbox pour utiliser le flux de paiement.

Rendez-vous sur <https://developer.paypal.com/developer/accounts/>, connectez-vous avec votre compte PayPal et cliquez sur "Créer un compte", comme ci-dessous:

Sandbox Test Accounts

Questions? Check out the [Testing Guide](#). Non-US developers should read our [FAQ](#).


Want to link existing Sandbox Account with your developer account? [Click Here](#) and provide

Total records: 15

| <input type="checkbox"/> | Email Address | Type |
|--------------------------|------------------------|----------|
| <input type="checkbox"/> | ▶ resttest@testing.com | PERSONAL |
| <input type="checkbox"/> | ▶ testmctest@test.com | PERSONAL |

Entrez dans les détails du compte pour le nouvel utilisateur du test, y compris un e-mail unique, les informations de compte, le mode de paiement, le solde, etc., puis cliquez sur "Créer un compte" en bas de la page. Cela créera le nouveau compte pour vous de commencer à utiliser.

Pour voir les détails du compte de ce nouvel utilisateur, développez l'entrée sur la page des comptes et cliquez sur "Profil".

| | | |
|---|---|----------|
| <input type="checkbox"/> | testmctest@test.com | PERSONAL |
|  | Profile Notifications | |

Une fois ces informations de profil chargées, cliquez sur l'onglet «Financement» pour obtenir les informations de paiement relatives à ce compte, y compris les informations de carte de crédit pouvant être utilisées pour le traitement direct des cartes de crédit.

REMARQUE: Lorsque vous utilisez les points de terminaison API sandbox, vous devez utiliser un compte de test sandbox pour vous connecter et payer les produits test, car les informations de votre compte live ne fonctionneront pas.

Lire Démarrer avec PayPal en ligne: <https://riptutorial.com/fr/paypal/topic/406/demarrer-avec-paypal>

Chapitre 2: Création d'abonnements / paiements récurrents

Paramètres

| Paramètre | Détails |
|-----------------------------|---|
| billingAgreementAttributes | Objet de configuration pour créer l'accord de facturation |
| facturationPlan | ID de plan de facturation de la chaîne de requête |
| billingPlanAttribs | Objet de configuration pour créer le plan de facturation |
| billingPlanUpdateAttributes | Objet de configuration pour modifier un plan de facturation en un état actif |
| identité du client | Votre ID client d'application (clés OAuth) |
| http | Référence au package http pour configurer notre serveur simple |
| isoDate | Date ISO pour définir la date de début de l'abonnement |
| liens | Objet lien HATEOAS pour extraire l'URL de redirection vers PayPal |
| params | Paramètres de chaîne de requête |
| Pay Pal | Référence au SDK PayPal |
| secret | Votre secret d'application (clés OAuth) |
| jeton | Le jeton d'approbation de l'accord de facturation fourni après la redirection de PayPal pour exécuter l'accord de facturation |

Remarques

Ces exemples passent par le processus de création d'un système de paiement par abonnement / paiement récurrent via PayPal.

Le processus de création d'un abonnement consiste à:

1. Créez un plan de facturation. Ceci est un modèle réutilisable qui décrit les détails de l'abonnement.
2. Activer le calendrier de facturation.
3. Lorsque vous souhaitez créer un abonnement pour un utilisateur, vous créez un contrat de

facturation à l'aide de l'ID du plan de facturation auquel ils doivent être abonnés.

4. Une fois créé, vous redirigez l'utilisateur vers PayPal pour confirmer l'abonnement. Une fois confirmé, l'utilisateur est redirigé vers le site Web du marchand.
5. Enfin, vous exécutez l'accord de facturation pour commencer l'abonnement.

Exemples

Étape 2: Création d'un abonnement pour un utilisateur à l'aide d'un accord de facturation (exemple de noeud)

La deuxième étape de la création d'un abonnement pour un utilisateur consiste à créer et à exécuter un accord de facturation, basé sur un plan de facturation activé existant. Cet exemple suppose que vous avez déjà parcouru et activé un plan de facturation dans l'exemple précédent et que vous avez un ID pour ce plan de facturation à référencer dans l'exemple.

Lorsque vous configurez un accord de facturation pour créer un abonnement pour un utilisateur, vous suivrez trois étapes, qui peuvent vous rappeler de traiter un paiement PayPal:

1. Vous créez un accord de facturation en référençant un plan de facturation sous-jacent via l'ID.
2. Une fois créé, vous redirigez l'utilisateur vers PayPal (si vous payez via PayPal) pour confirmer l'abonnement. Une fois confirmé, PayPal redirige l'utilisateur vers votre site en utilisant la redirection fournie dans le plan de facturation sous-jacent.
3. Vous exécutez ensuite l'accord de facturation en utilisant un jeton fourni via la redirection PayPal.

Cet exemple consiste à configurer un serveur HTTP Express pour présenter le processus de contrat de facturation.

Pour démarrer l'exemple, nous devons d'abord configurer notre configuration. Nous avons ajouté quatre exigences, le kit de développement PayPal, l' `body-parser` de `body-parser` pour la gestion des corps encodés JSON, l' `http` pour notre intégration de serveur simple et l' `express` pour le framework Express. Nous définissons ensuite notre identifiant client et le secret de création d'une application, configurons le SDK pour le bac à sable, puis configurons `bodyParser` pour gérer les corps JSON.

```
var paypal = require('paypal-rest-sdk'),
    bodyParser = require('body-parser'),
    http = require('http'),
    app = require('express')();

var clientId = 'YOUR APPLICATION CLIENT ID';
var secret = 'YOUR APPLICATION SECRET';

paypal.configure({
  'mode': 'sandbox', //sandbox or live
  'client_id': clientId,
  'client_secret': secret
});
```

```
app.use(bodyParser.json());
```

Notre première étape dans l'accord de facturation consiste à créer un itinéraire pour gérer la création d'un accord de facturation et à rediriger l'utilisateur vers PayPal pour confirmer cet abonnement. Nous supposons qu'un ID de plan de facturation est transmis en tant que paramètre de chaîne de requête, par exemple en chargeant l'URL suivante avec un ID de plan de l'exemple précédent:

```
http://localhost:3000/createagreement?plan=P-3N543779E9831025ECYGDNVQ
```

Nous devons maintenant utiliser ces informations pour créer l'accord de facturation.

```
app.get('/createagreement', function(req, res){
  var billingPlan = req.query.plan;

  var isoDate = new Date();
  isoDate.setSeconds(isoDate.getSeconds() + 4);
  isoDate.toISOString().slice(0, 19) + 'Z';

  var billingAgreementAttributes = {
    "name": "Standard Membership",
    "description": "Food of the World Club Standard Membership",
    "start_date": isoDate,
    "plan": {
      "id": billingPlan
    },
    "payer": {
      "payment_method": "paypal"
    },
    "shipping_address": {
      "line1": "W 34th St",
      "city": "New York",
      "state": "NY",
      "postal_code": "10001",
      "country_code": "US"
    }
  };

  // Use activated billing plan to create agreement
  paypal.billingAgreement.create(billingAgreementAttributes, function (error,
  billingAgreement){
    if (error) {
      console.error(error);
      throw error;
    } else {
      //capture HATEOAS links
      var links = {};
      billingAgreement.links.forEach(function(linkObj) {
        links[linkObj.rel] = {
          'href': linkObj.href,
          'method': linkObj.method
        };
      });
    }

    //if redirect url present, redirect user
    if (links.hasOwnProperty('approval_url')){
      res.redirect(links['approval_url'].href);
    }
  });
});
```

```

        } else {
            console.error('no redirect URI present');
        }
    }
});
});

```

Nous commençons par extraire l'identifiant du plan de facturation de la chaîne de requête et créons la date à laquelle le plan doit démarrer.

La définition d'objet suivante, `billingAgreementAttributes`, consiste en informations pour l'abonnement. Il contient des informations lisibles sur le plan, une référence à l'ID du plan de facturation, le mode de paiement et les détails d'expédition (si nécessaire pour l'abonnement).

Ensuite, un appel à `billingAgreement.create(...)` est effectué en transmettant l'objet `billingAgreementAttributes` nous venons de créer. Si tout se passe bien, nous devrions nous faire parvenir un objet d'accord de facturation contenant des détails sur notre abonnement nouvellement créé. Cet objet contient également un certain nombre de liens HATEOAS nous fournissant les prochaines étapes à suivre pour cet accord nouvellement créé. Celui qui nous intéresse ici est appelé `approval_url`.

Nous parcourons tous les liens fournis pour les placer dans un objet facilement référencé. Si `approval_url` est l'un de ces liens, nous redirigeons l'utilisateur vers ce lien, qui est PayPal.

À ce stade, l'utilisateur confirme l'abonnement sur PayPal et est redirigé vers l'URL fournie dans le plan de facturation sous-jacent. Parallèlement à cette URL, PayPal transmettra également un jeton le long de la chaîne de requête. Ce jeton est ce que nous allons utiliser pour exécuter (ou démarrer) l'abonnement.

Configurez cette fonctionnalité dans l'itinéraire suivant.

```

app.get('/processagreement', function(req, res){
    var token = req.query.token;

    paypal.billingAgreement.execute(token, {}, function (error, billingAgreement) {
        if (error) {
            console.error(error);
            throw error;
        } else {
            console.log(JSON.stringify(billingAgreement));
            res.send('Billing Agreement Created Successfully');
        }
    });
});
});

```

Nous extrayons le jeton de la chaîne de requête, puis nous appelons `billingAgreement.execute`, en transmettant ce jeton. Si tout est réussi, nous avons maintenant un abonnement valide pour l'utilisateur. L'objet de retour contient des informations sur l'accord de facturation actif.

Enfin, nous avons configuré notre serveur HTTP pour écouter le trafic sur nos routes.

```
//create server
```

```
http.createServer(app).listen(3000, function () {
  console.log('Server started: Listening on port 3000');
});
```

Étape 1: Création d'un modèle d'abonnement à l'aide d'un plan de facturation (exemple de noeud)

Lors de la création d'un abonnement pour un utilisateur, vous devez d'abord créer et activer un calendrier de facturation auquel un utilisateur est ensuite abonné à l'aide d'un accord de facturation. Le processus complet de création d'un abonnement est détaillé dans les remarques de ce sujet.

Dans cet exemple, nous allons utiliser le [SDK PayPal Node](#) . Vous pouvez l'obtenir à partir de NPM à l'aide de la commande suivante:

```
npm install paypal-rest-sdk
```

Dans notre fichier .js, nous avons d'abord configuré notre configuration SDK, qui inclut l'ajout d'une exigence pour le SDK, la définition de notre ID client et le secret de [création de notre application](#) , puis la configuration du SDK pour l'environnement sandbox.

```
var paypal = require('paypal-rest-sdk');

var clientId = 'YOUR CLIENT ID';
var secret = 'YOUR SECRET';

paypal.configure({
  'mode': 'sandbox', //sandbox or live
  'client_id': clientId,
  'client_secret': secret
});
```

Ensuite, nous devons configurer deux objets JSON. L'objet `billingPlanAttribs` contient les informations et la répartition des paiements pour le `billingPlanAttribs` facturation auquel nous pouvons abonner les utilisateurs, et l'objet `billingPlanUpdateAttributes` contient des valeurs permettant de définir le `billingPlanUpdateAttributes` facturation à un état actif, ce qui lui permet d'être utilisé.

```
var billingPlanAttribs = {
  "name": "Food of the World Club Membership: Standard",
  "description": "Monthly plan for getting the t-shirt of the month.",
  "type": "fixed",
  "payment_definitions": [{
    "name": "Standard Plan",
    "type": "REGULAR",
    "frequency_interval": "1",
    "frequency": "MONTH",
    "cycles": "11",
    "amount": {
      "currency": "USD",
      "value": "19.99"
    }
  ]
}
```

```

    }],
    "merchant_preferences": {
      "setup_fee": {
        "currency": "USD",
        "value": "1"
      },
      "cancel_url": "http://localhost:3000/cancel",
      "return_url": "http://localhost:3000/processagreement",
      "max_fail_attempts": "0",
      "auto_bill_amount": "YES",
      "initial_fail_amount_action": "CONTINUE"
    }
  };

  var billingPlanUpdateAttributes = [{
    "op": "replace",
    "path": "/",
    "value": {
      "state": "ACTIVE"
    }
  }];
}];

```

Dans l'objet `billingPlanAttribs` , il existe des éléments d'information pertinents:

- **nom / description / type** : informations visuelles de base pour décrire le plan et le type de plan.
- **payment_definitions** : informations sur le fonctionnement et la facturation du plan. Plus de détails sur les champs [ici](#) .
- **merchant_preferences** : structures de frais supplémentaires, URL de redirection et paramètres du plan d'abonnement. Plus de détails sur les champs [ici](#) .

Avec ces objets en place, nous pouvons maintenant créer et activer le plan de facturation.

```

paypal.billingPlan.create(billingPlanAttribs, function (error, billingPlan){
  if (error){
    console.log(error);
    throw error;
  } else {
    // Activate the plan by changing status to Active
    paypal.billingPlan.update(billingPlan.id, billingPlanUpdateAttributes, function(error,
response){
      if (error) {
        console.log(error);
        throw error;
      } else {
        console.log(billingPlan.id);
      }
    });
  }
});

```

Nous appelons `billingPlan.create(...)` , en passant l'objet `billingPlanAttribs` que nous venons de créer. Si cela réussit, l'objet de retour contiendra des informations sur le plan de facturation. À titre d'exemple, il suffit d'utiliser l'ID du plan de facturation pour activer le plan à utiliser.

Ensuite, nous appelons `billingPlan.update(...)` , en transmettant l'ID du plan de facturation et

l'objet `billingPlanUpdateAttributes` nous avons créé précédemment. Si cela réussit, notre plan de facturation est maintenant actif et prêt à être utilisé.

Afin de créer un abonnement pour un utilisateur (ou plusieurs utilisateurs) sur ce plan, nous devons référencer l'identifiant du plan de facturation (`billingPlan.id` ci-dessus), afin de le stocker dans un endroit facilement référencable.

Lors de la deuxième étape de la souscription, nous devons créer un accord de facturation basé sur le plan que nous venons de créer et l'exécuter pour commencer à traiter les abonnements d'un utilisateur.

Lire [Création d'abonnements / paiements récurrents en ligne](#):

<https://riptutorial.com/fr/paypal/topic/467/creation-d-abonnements---paiements-recurrents>

Chapitre 3: Effectuer un paiement par carte de crédit (nœud)

Paramètres

| Paramètre | Détails |
|---------------------|---|
| card_data | Objet JSON contenant des informations de paiement pour la transaction |
| credit_card_details | Objet JSON contenant des données de carte de crédit envoyées à PayPal |
| identité du client | Votre identifiant de client d'application PayPal (identifiants OAuth 2) |
| Pay Pal | Référence du SDK du nœud PayPal |
| secret | Votre secret d'application PayPal (identifiants OAuth 2) |
| uuid | Référence au package node-uuid |

Remarques

Cet exemple amène l'utilisateur à créditer une simple transaction par carte de crédit à l'aide des kits de développement PayPal.

Exemples

Exemple de nœud

Commencez par installer le module PayPal Node à partir de NPM

```
npm install paypal-rest-sdk
```

Dans votre fichier d'application, ajoutez les informations de configuration du SDK

```
var paypal = require('paypal-rest-sdk');

var client_id = 'YOUR CLIENT ID';
var secret = 'YOUR SECRET';

paypal.configure({
  'mode': 'sandbox', //sandbox or live
  'client_id': client_id,
  'client_secret': secret
});
```

Nous ajoutons l'exigence pour le SDK, puis nous configurons des variables pour l'ID client et le secret obtenus lors de la [création d'une application](#) . Nous configurons ensuite notre application en utilisant ces détails et spécifions l'environnement dans lequel nous travaillons (live ou sandbox).

Ensuite, nous configurons l'objet JSON contenant les informations de paiement pour le payeur.

```
var card_data = {
  "intent": "sale",
  "payer": {
    "payment_method": "credit_card",
    "funding_instruments": [{
      "credit_card": {
        "type": "visa",
        "number": "4417119669820331",
        "expire_month": "11",
        "expire_year": "2018",
        "cvv2": "874",
        "first_name": "Joe",
        "last_name": "Shopper",
        "billing_address": {
          "line1": "52 N Main ST",
          "city": "Johnstown",
          "state": "OH",
          "postal_code": "43210",
          "country_code": "US" }}}}],
  "transactions": [{
    "amount": {
      "total": "7.47",
      "currency": "USD",
      "details": {
        "subtotal": "7.41",
        "tax": "0.03",
        "shipping": "0.03"}},
    "description": "This is the payment transaction description."
  }
  ]};
```

Ajouter une `intent` de `sale` et une `payment_method` de `credit_card` de `credit_card` . Ensuite, ajoutez les détails de la carte et de l'adresse de la carte de crédit dans `funding_instruments` , ainsi que le montant à facturer pour les `transactions` . Plusieurs objets de transaction peuvent être placés ici.

Enfin, nous faisons une demande à `payment.create(...)` , en passant notre objet `card_data` , afin de traiter le paiement.

```
paypal.payment.create(card_data, function(error, payment){
  if(error){
    console.error(error);
  } else {
    console.log(payment);
  }
});
```

Si la transaction a réussi, nous devrions voir un objet de réponse similaire à celui-ci:

```
{
```



```

"id": "PAY-9BS08892W3794812YK4HKFQY",
"create_time": "2016-04-13T19:49:23Z",
"update_time": "2016-04-13T19:50:07Z",
"state": "approved",
"intent": "sale",
"payer": {
  "payment_method": "credit_card",
  "funding_instruments": [
    {
      "credit_card": {
        "type": "visa",
        "number": "xxxxxxxxxxxx0331",
        "expire_month": "11",
        "expire_year": "2018",
        "first_name": "Joe",
        "last_name": "Shopper",
        "billing_address": {
          "line1": "52 N Main ST",
          "city": "Johnstown",
          "state": "OH",
          "postal_code": "43210",
          "country_code": "US"
        }
      }
    }
  ]
},
"transactions": [
  {
    "amount": {
      "total": "7.47",
      "currency": "USD",
      "details": {
        "subtotal": "7.41",
        "tax": "0.03",
        "shipping": "0.03"
      }
    },
    "description": "This is the payment transaction description.",
    "related_resources": [
      {
        "sale": {
          "id": "0LB81696PP288253D",
          "create_time": "2016-04-13T19:49:23Z",
          "update_time": "2016-04-13T19:50:07Z",
          "amount": {
            "total": "7.47",
            "currency": "USD"
          },
          "state": "completed",
          "parent_payment": "PAY-9BS08892W3794812YK4HKFQY",
          "links": [
            {
              "href":
"https://api.sandbox.paypal.com/v1/payments/sale/0LB81696PP288253D",
              "rel": "self",
              "method": "GET"
            },
            {
              "href":
"https://api.sandbox.paypal.com/v1/payments/sale/0LB81696PP288253D/refund",

```

```

        "rel": "refund",
        "method": "POST"
    },
    {
        "href": "https://api.sandbox.paypal.com/v1/payments/payment/PAY-9BS08892W3794812YK4HKFQY",
        "rel": "parent_payment",
        "method": "GET"
    }
],
"fmf_details": {
},
"processor_response": {
    "avs_code": "X",
    "cvv_code": "M"
}
}
]
}
],
"links": [
    {
        "href": "https://api.sandbox.paypal.com/v1/payments/payment/PAY-9BS08892W3794812YK4HKFQY",
        "rel": "self",
        "method": "GET"
    }
],
"httpStatusCode": 201
}

```

Dans cet objet de retour, le fait d'avoir un `state approved` indique que la transaction a abouti. Sous l'objet `links`, il y a un certain nombre de liens **HATEOAS** qui fournissent les prochaines étapes potentielles pouvant être effectuées sur l'action qui vient d'être effectuée. Dans ce cas, nous pouvons récupérer des informations sur le paiement en faisant une requête GET à l' `self` point final fourni.

Effectuer un paiement avec une carte de crédit voûtée (nœud)

Dans cet exemple, nous examinerons comment stocker une carte de crédit en utilisant le coffre PayPal, puis référencer cette carte de crédit stockée pour traiter une transaction par carte de crédit pour un utilisateur.

La raison pour laquelle nous souhaiterions utiliser le coffre-fort est que nous n'avons pas besoin de stocker des informations de carte de crédit sensibles sur nos propres serveurs. Nous référençons simplement la méthode de paiement via un identifiant de coffre fourni, ce qui signifie que nous n'avons pas à traiter de nombreuses réglementations de conformité PCI pour stocker les cartes de crédit nous-mêmes.

Comme pour les échantillons précédents, nous commençons par configurer notre environnement.

```

var paypal = require('paypal-rest-sdk'),
    uuid = require('node-uuid');

```

```
var client_id = 'YOUR CLIENT ID';
var secret = 'YOUR SECRET';

paypal.configure({
  'mode': 'sandbox', //sandbox or live
  'client_id': client_id,
  'client_secret': secret
});
```

La seule différence avec les exemples précédents est que nous avons besoin d'un nouveau package, `node-uuid`, qui doit être utilisé pour générer des UUID uniques pour les payeurs lors du stockage de la carte. Vous pouvez installer ce package via:

```
npm install node-uuid
```

Ensuite, nous définissons l'objet JSON de carte de crédit qui sera envoyé au coffre PayPal pour le stockage. Il contient des informations provenant de la carte, ainsi qu'un identifiant de payeur unique que nous générons en utilisant `node-uuid`. Vous devriez stocker ce `payer_id` unique dans votre propre base de données car il sera utilisé lors de la création d'un paiement avec la carte voûtée.

```
var create_card_details = {
  "type": "visa",
  "number": "4417119669820331",
  "expire_month": "11",
  "expire_year": "2018",
  "first_name": "John",
  "last_name": "Doe",
  "payer_id": uuid.v4()
};
```

Enfin, nous devons stocker la carte de crédit et traiter le paiement en utilisant cette carte. Pour `credit_card.create(...)` une carte de crédit, nous appelons `credit_card.create(...)`, en transmettant l'objet `credit_card_details` que nous venons de créer. Si tout se passe bien, un objet devrait nous être retourné avec des détails sur la carte voûtée. Par souci de paiement avec cette carte, nous n'avons besoin que de deux éléments d'information: l'id de payeur que nous avons déjà enregistré et l'ID de coffre, qui devrait également être stocké comme référence dans notre propre base de données.

```
paypal.credit_card.create(create_card_details, function(error, credit_card){
  if(error){
    console.error(error);
  } else {
    var card_data = {
      "intent": "sale",
      "payer": {
        "payment_method": "credit_card",
        "funding_instruments": [{
          "credit_card_token": {
            "credit_card_id": credit_card.id,
            "payer_id": credit_card.payer_id
          }
        }]
      }
    }
  }
});
```

```

    }}
  },
  "transactions": [{
    "amount": {
      "total": "7.47",
      "currency": "USD",
      "details": {
        "subtotal": "7.41",
        "tax": "0.03",
        "shipping": "0.03"
      }
    },
    "description": "This is the payment transaction description."
  }]
};

paypal.payment.create(card_data, function(error, payment){
  if(error){
    console.error(error);
  } else {
    console.log(JSON.stringify(payment));
  }
});
});

```

Dans la section qui suit la mise en coffre de la carte de crédit, nous définissons simplement les détails de la carte et traitons le paiement, comme nous l'avons fait avec l'exemple de traitement de carte de crédit précédent. La principale différence dans la structure de l'objet `card_data` est la section `funding_instruments`, que nous définissons sous `payer`. Au lieu de définir les informations de carte de crédit, nous utilisons plutôt l'objet suivant qui contient la référence d'ID de coffre-fort et l'ID de payeur:

```

"credit_card_token": {
  "credit_card_id": credit_card.id,
  "payer_id": credit_card.payer_id
}

```

C'est ainsi que nous utilisons une carte avec coffre-fort pour traiter un paiement.

Lire Effectuer un paiement par carte de crédit (nœud) en ligne:

<https://riptutorial.com/fr/paypal/topic/444/effectuer-un-paiement-par-carte-de-credit--noud->

Chapitre 4: Effectuer un paiement PayPal

Paramètres

| Paramètre | Détails |
|--------------------|---|
| identité du client | Votre identifiant de client d'application PayPal (identifiants OAuth 2) |
| liens | Objet de référence simple pour tous les liens de retour HATEOAS de PayPal |
| paymentId | L'identifiant du paiement retourné par PayPal afin de compléter le paiement |
| payerId | L'identifiant du payeur retourné de PayPal afin de compléter le paiement |
| Pay Pal | Référence du SDK du nœud PayPal |
| payReq | Objet JSON contenant des informations de paiement pour la transaction |
| req | L'objet Request de la requête du serveur |
| res | L'objet réponse de la requête du serveur |
| secret | Votre secret d'application PayPal (identifiants OAuth 2) |

Remarques

Ces exemples expliquent comment traiter un paiement via PayPal en utilisant les kits de développement PayPal. Ce sont des exemples de demandes simples qui décrivent le processus en plusieurs étapes pour autoriser cette option de paiement.

Exemples

Exemple de serveur Node Express

Dans cet exemple, nous allons configurer une intégration de serveur Express pour afficher comment traiter un paiement avec PayPal à l'aide du SDK Node de PayPal. Nous utiliserons une structure JSON statique pour les détails du paiement, par souci de concision.

Nous suivrons trois étapes générales lors de la création des fonctions pour gérer le paiement PayPal:

1. Nous créons un objet JSON contenant le paiement que nous avons l'intention de traiter via

PayPal. Nous l'envoyons ensuite à PayPal pour obtenir un lien vers lequel l'utilisateur sera redirigé afin de confirmer le paiement.

2. Ensuite, nous redirigeons l'utilisateur vers PayPal pour confirmer le paiement. Une fois confirmé, PayPal redirige l'utilisateur vers notre application.
3. Une fois retourné à l'application, nous complétons le paiement au nom de l'utilisateur.

En décomposant cette application en une simple application de noeud, nous commençons par obtenir le kit de développement PayPal Node:

```
npm install paypal-rest-sdk
```

Ensuite, nous configurons la configuration de l'application et les packages.

```
var http = require('http'),
    paypal = require('paypal-rest-sdk'),
    bodyParser = require('body-parser'),
    app = require('express')();

var client_id = 'YOUR APPLICATION CLIENT ID';
var secret = 'YOUR APPLICATION SECRET';

//allow parsing of JSON bodies
app.use(bodyParser.json());

//configure for sandbox environment
paypal.configure({
  'mode': 'sandbox', //sandbox or live
  'client_id': client_id,
  'client_secret': secret
});
```

Nous avons besoin de quatre exigences pour cette application:

1. Le package HTTP pour notre serveur.
2. Le package SDK PayPal Node.
3. Le package bodyParser pour travailler avec des corps encodés en JSON.
4. Le framework Express pour notre serveur.

Les lignes suivantes définissent des variables pour l'ID client et le secret obtenus lors de la [création d'une application](#) . Nous avons ensuite configuré `bodyParser` pour autoriser les corps codés JSON, puis configurer notre application à l'aide des détails de l'application et spécifier l'environnement dans lequel nous travaillons (en direct pour la production ou le sandbox pour les tests).

Maintenant, créons la route pour créer une demande de paiement avec PayPal.

```
app.get('/create', function(req, res){
  //build PayPal payment request
  var payReq = JSON.stringify({
    'intent':'sale',
    'redirect_urls':{
      'return_url':'http://localhost:3000/process',
      'cancel_url':'http://localhost:3000/cancel'
```

```

    },
    'payer':{
      'payment_method':'paypal'
    },
    'transactions':[{
      'amount':{
        'total':'7.47',
        'currency':'USD'
      },
      'description':'This is the payment transaction description.'
    }]
  });

paypal.payment.create(payReq, function(error, payment){
  if(error){
    console.error(error);
  } else {
    //capture HATEOAS links
    var links = {};
    payment.links.forEach(function(linkObj){
      links[linkObj.rel] = {
        'href': linkObj.href,
        'method': linkObj.method
      };
    });

    //if redirect url present, redirect user
    if (links.hasOwnProperty('approval_url')){
      res.redirect(links['approval_url'].href);
    } else {
      console.error('no redirect URI present');
    }
  }
});
});
});

```

La première chose à faire est de configurer l'objet JSON de demande de paiement, qui contient les informations nécessaires à PayPal pour créer le paiement. Nous définissons l' `intent` de `sale`, spécifiez les URL de redirection (où PayPal doit renvoyer l'utilisateur après avoir confirmé / annulé le paiement), ajoutez une `payment_method` de `paypal` pour signaler que nous effectuerons un paiement PayPal, puis spécifiez les informations de transaction pour le payeur à confirmer.

Nous appelons alors `payment.create(...)`, en passant notre objet `payReq`. Cela enverra la demande de paiement à PayPal. Une fois que retourne, et réussit, nous pouvons boucle à travers les fournis [HATEOAS](#) liens dans l'objet de retour pour extraire l'URL que nous devons rediriger l'utilisateur vers, qui est étiqueté sous `approval_url`.

Le format des liens HATEOAS peut entraîner un code de référence fragile s'il est utilisé directement. Nous parcourons donc tous les liens fournis et les plaçons dans un objet de référence mieux adapté aux modifications futures. Si l' `approval_url` se trouve alors dans cet objet, nous redirigeons l'utilisateur.

À ce stade, l'utilisateur est redirigé vers PayPal pour confirmer le paiement. Une fois qu'ils le font, ils sont redirigés vers le `return_url` que nous avons spécifié dans la fonction `createPayment(...)`.

Nous devons maintenant fournir un itinéraire pour gérer ce retour, afin de compléter le paiement.

```
app.get('/process', function(req, res){
  var paymentId = req.query.paymentId;
  var payerId = { 'payer_id': req.query.PayerID };

  paypal.payment.execute(paymentId, payerId, function(error, payment){
    if(error){
      console.error(error);
    } else {
      if (payment.state == 'approved'){
        res.send('payment completed successfully');
      } else {
        res.send('payment not successful');
      }
    }
  });
});
```

Lorsque l'utilisateur est renvoyé à votre application, trois paramètres de chaîne de requête seront envoyés avec le `paymentId`, le `PayerID` et le `token`. Il suffit de traiter les deux premiers.

Nous extrayons les paramètres et plaçons le `PayerID` dans un objet simple pour le besoin de l'étape d'exécution du paiement. Ensuite, un appel est effectué à `payment.execute(...)`, en transmettant ces deux paramètres, afin de compléter le paiement.

Une fois cette demande effectuée, nous vérifions `payment.state` si le paiement a été effectué avec succès en vérifiant si `payment.state` est défini sur `approved`. Si c'est le cas, nous pouvons stocker ce dont nous avons besoin à partir de l'objet de paiement renvoyé.

Notre dernière étape consiste à initialiser notre serveur et à écouter le trafic entrant sur les routes que nous avons spécifiées.

```
//create server
http.createServer(app).listen(3000, function () {
  console.log('Server started: Listening on port 3000');
});
```

Une fois le serveur initialisé, aller à `http://localhost:3000/create` initialise le processus de paiement.

Lire Effectuer un paiement PayPal en ligne: <https://riptutorial.com/fr/paypal/topic/449/effectuer-un-paiement-paypal>

Chapitre 5: Paiements futurs mobiles (application de bout en bout)

Remarques

Cet exemple montre un exemple pratique de création d'un [futur paiement PayPal](#) à partir d'un périphérique Android, à l'aide d'un serveur de nœuds.

Exemples

Etape 1: mise en page, initialisation et gestion des réponses du serveur

L'exemple de code complet pour cette application (serveur Android + Node) est disponible dans le [référentiel Github de PayPal Developer](#).

La première étape de la création de la partie Android de notre application consiste à configurer une disposition de base et à gérer les réponses qui reviennent du serveur que nous allons configurer dans Node.

Commencez par créer un nouvel objet `PayPalConfiguration` pour héberger les informations de votre application.

```
private static PayPalConfiguration config = new PayPalConfiguration()
    .environment(PayPalConfiguration.ENVIRONMENT_SANDBOX)
    .clientId("YOUR APPLICATION CLIENT ID")
    .merchantName("My Store")
    .merchantPrivacyPolicyUri(Uri.parse("https://www.example.com/privacy"))
    .merchantUserAgreementUri(Uri.parse("https://www.example.com/legal"));
```

Ensuite, nous ajoutons un simple bouton à `onCreate(...)` pour faire office d'initiation de paiement. Ceci est simplement pour déclencher l'action, et devrait être placé en tant que processus d'initiation pour créer un paiement futur pour un utilisateur (par exemple, quand ils s'accordent sur un abonnement).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    final Button button = (Button) findViewById(R.id.paypal_button);
}
```

Sous `res > layout > activity_main.xml` on ajoute la définition du bouton avec son action associée, quand on clique dessus, il appelle `beginFuturePayment(...)`, que nous définirons dans une minute.

```
<Button android:id="@+id/paypal_button"
```

```
android:layout_height="wrap_content"
android:layout_width="wrap_content"
android:text="@string/paypal_button"
android:onClick="beginFuturePayment" />
```

Sous `res > values > strings.xml` nous ajoutons alors une référence de chaîne pour le bouton.

```
<string name="paypal_button">Process Future Payment</string>
```

Maintenant, nous ajoutons le gestionnaire de bouton, pour lancer l'appel pour commencer le processus de paiement futur lorsque l'utilisateur clique sur le bouton. Ce que nous faisons ici est de démarrer le service de paiement avec l'objet de configuration que nous avons configuré en haut de cet exemple.

```
public void beginFuturePayment(View view){
    Intent serviceConfig = new Intent(this, PayPalService.class);
    serviceConfig.putExtra(PayPalService.EXTRA_PAYPAL_CONFIGURATION, config);
    startService(serviceConfig);

    Intent intent = new Intent(this, PayPalFuturePaymentActivity.class);
    intent.putExtra(PayPalService.EXTRA_PAYPAL_CONFIGURATION, config);
    startActivityForResult(intent, 0);
}
```

Lorsque cet appel pour effectuer un paiement ultérieur est lancé, nous recevons des informations qui devront être envoyées à notre serveur. Nous extrayons ces informations de la demande de paiement future valide (`authCode` et `metadataId`), puis `authCode` la demande asynchrone sur le serveur pour effectuer le paiement futur (détaillé à l'étape 2).

```
@Override
protected void onActivityResult (int requestCode, int resultCode, Intent data){
    if (resultCode == Activity.RESULT_OK){
        PayPalAuthorization auth =
data.getParcelableExtra(PayPalFuturePaymentActivity.EXTRA_RESULT_AUTHORIZATION);
        if (auth != null){
            try{
                //prepare params to be sent to server
                String authCode = auth.getAuthorizationCode();
                String metadataId = PayPalConfiguration.getClientMetadataId(this);
                String [] params = {authCode, metadataId};

                //process async server request for token + payment
                ServerRequest req = new ServerRequest();
                req.execute(params);

            } catch (JSONException e) {
                Log.e("FPSample", "JSON Exception: ", e);
            }
        }
    } else if (resultCode == Activity.RESULT_CANCELED) {
        Log.i("FPSample", "User canceled.");
    } else if (resultCode == PayPalFuturePaymentActivity.RESULT_EXTRAS_INVALID) {
        Log.i("FPSample", "Invalid configuration");
    }
}
```

Enfin, nous définissons notre `onDestroy()` .

```
@Override
public void onDestroy(){
    stopService(new Intent(this, PayPalService.class));
    super.onDestroy();
}
```

Android Étape 2: Demande de serveur asynchrone

L'exemple de code complet pour cette application (serveur Android + Node) est disponible dans le [référentiel Github de PayPal Developer](#) .

À ce stade, le bouton PayPal des paiements futurs a été cliqué, nous avons un code d'authentification et un identifiant de métadonnées du kit de développement PayPal, et nous devons les transmettre à notre serveur pour terminer le processus de paiement futur.

En arrière-plan, nous faisons quelques choses:

- Nous avons configuré l'URI pour que notre serveur soit `http://10.0.2.2:3000/fpstore` , qui `/fpstore` noeud final `/fpstore` de notre serveur s'exécutant sur localhost.
- L'objet JSON qui sera envoyé via est alors configuré, qui contient le code d'authentification et l'ID de métadonnées.
- La connexion est alors faite. En cas de demande réussie (plage 200/201), nous pouvons attendre une réponse du serveur. Nous lisons cette réponse puis la renvoyons.
- Enfin, nous avons une `onPostExecute(...)` configurée pour gérer cette chaîne de serveur renvoyée. Dans le cas de cet exemple, il est simplement connecté.

```
public class ServerRequest extends AsyncTask<String, Void, String> {
    protected String doInBackground(String[] params){
        HttpURLConnection connection = null;
        try{
            //set connection to connect to /fpstore on localhost
            URL u = new URL("http://10.0.2.2:3000/fpstore");
            connection = (HttpURLConnection) u.openConnection();
            connection.setRequestMethod("POST");

            //set configuration details
            connection.setRequestProperty("Content-Type", "application/json");
            connection.setRequestProperty("Accept", "application/json");
            connection.setAllowUserInteraction(false);
            connection.setConnectTimeout(10000);
            connection.setReadTimeout(10000);

            //set server post data needed for obtaining access token
            String json = "{\"code\": \"" + params[0] + "\", \"metadataId\": \"" + params[1] +
            "\"}";

            Log.i("JSON string", json);

            //set content length and config details
            connection.setRequestProperty("Content-length", json.getBytes().length + "");
            connection.setDoInput(true);
            connection.setDoOutput(true);
            connection.setUseCaches(false);
```

```

//send json as request body
OutputStream outputStream = connection.getOutputStream();
outputStream.write(json.getBytes("UTF-8"));
outputStream.close();

//connect to server
connection.connect();

//look for 200/201 status code for received data from server
int status = connection.getResponseCode();
switch (status){
    case 200:
    case 201:
        //read in results sent from the server
        BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
        StringBuilder sb = new StringBuilder();
        String line;
        while ((line = bufferedReader.readLine()) != null){
            sb.append(line + "\n");
        }
        bufferedReader.close();

        //return received string
        return sb.toString();
    }

} catch (MalformedURLException ex) {
    Log.e("HTTP Client Error", ex.toString());
} catch (IOException ex) {
    Log.e("HTTP Client Error", ex.toString());
} catch (Exception ex) {
    Log.e("HTTP Client Error", ex.toString());
} finally {
    if (connection != null) {
        try{
            connection.disconnect();
        } catch (Exception ex) {
            Log.e("HTTP Client Error", ex.toString());
        }
    }
}
return null;
}

protected void onPostExecute(String message) {
    //log values sent from the server - processed payment
    Log.i("HTTP Client", "Received Return: " + message);
}
}

```

Etape 3 d'Android: Serveur de noeud pour obtenir le paiement de jeton et de processus d'accès

L'exemple de code complet pour cette application (serveur Android + Node) est disponible dans le [référentiel Github de PayPal Developer](#) .

A partir de l'étape 2, une requête asynchrone a été effectuée sur notre serveur au niveau du

`/fpstore` **final** `/fpstore` , en transmettant le code d'authentification et l'ID de métadonnées. Nous devons maintenant les échanger contre un jeton pour compléter la demande et traiter le paiement futur.

Tout d'abord, nous configurons nos variables de configuration et notre objet.

```
var bodyParser = require('body-parser'),
    http = require('http'),
    paypal = require('paypal-rest-sdk'),
    app = require('express')();

var client_id = 'YOUR APPLICATION CLIENT ID';
var secret = 'YOUR APPLICATION SECRET';

paypal.configure({
  'mode': 'sandbox',
  'client_id': client_id,
  'client_secret': secret
});

app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
```

Nous établissons maintenant un itinéraire Express qui écoutera les requêtes POST envoyées au point de terminaison `/fpstore` partir de notre code Android.

Nous faisons un certain nombre de choses dans cette voie:

- Nous capturons le code d'authentification et l'ID de métadonnées du corps POST.
- Nous faisons ensuite une demande à `generateToken()` , en passant par l'objet code. En cas de succès, nous obtenons un jeton pouvant être utilisé pour créer le paiement.
- Ensuite, les objets de configuration sont créés pour le paiement futur à effectuer et une demande de `payment.create(...)` est transmise, transmettant les futurs objets de configuration de paiement et de paiement. Cela crée le paiement futur.

```
app.post('/fpstore', function(req, res){
  var code = {'authorization_code': req.body.code};
  var metadata_id = req.body.metadataId;

  //generate token from provided code
  paypal.generateToken(code, function (error, refresh_token) {
    if (error) {
      console.log(error);
      console.log(error.response);
    } else {
      //create future payments config
      var fp_config = {'client_metadata_id': metadata_id, 'refresh_token':
refresh_token};

      //payment details
      var payment_config = {
        "intent": "sale",
        "payer": {
          "payment_method": "paypal"
        },
        "transactions": [{
```

```

        "amount": {
            "currency": "USD",
            "total": "3.50"
        },
        "description": "Mesozoic era monster toy"
    }
}
};

//process future payment
paypal.payment.create(payment_config, fp_config, function (error, payment) {
    if (error) {
        console.log(error.response);
        throw error;
    } else {
        console.log("Create Payment Response");
        console.log(payment);

        //send payment object back to mobile
        res.send(JSON.stringify(payment));
    }
});
}
});
});
});

```

Enfin, nous créons le serveur pour écouter sur le port 3000.

```

//create server
http.createServer(app).listen(3000, function () {
    console.log('Server started: Listening on port 3000');
});

```

Lire Paiements futurs mobiles (application de bout en bout) en ligne:

<https://riptutorial.com/fr/paypal/topic/4537/paiements-futurs-mobiles--application-de-bout-en-bout->

Chapitre 6: Paypal Mobile / Paiements par carte de crédit

Paramètres

| Paramètre | Détails |
|----------------|--|
| bouton | Bouton de paiement simple |
| config | Objet de configuration PayPal contenant notre identifiant client (issu de la création d'application) et l'environnement que nous souhaitons utiliser (sandbox ou live) |
| Paiement | Détails de paiement PayPal |
| paiementConfig | Intention de configuration pour les informations et paramètres de paiement |
| serviceConfig | Intention de configuration pour les données de paramètre de configuration |

Remarques

Exemples liés au traitement des paiements sur des appareils mobiles

Exemples

Android: accepter un paiement PayPal / carte de crédit

Dans ce tutoriel, nous allons apprendre à configurer le [SDK Android de PayPal](#) pour traiter un paiement simple via un paiement PayPal ou un achat par carte de crédit. À la fin de cet exemple, vous devriez avoir un bouton simple dans une application qui, une fois cliquée, transmettra l'utilisateur à PayPal pour confirmer un paiement défini, puis le renverra à l'application et enregistrera la confirmation du paiement.

Le code d'application complet de cet exemple est disponible dans le [référentiel Github des développeurs PayPal](#).

Commençons.

La première étape consiste à [obtenir et à ajouter le SDK à votre projet](#). Nous ajoutons la référence à nos dépendances build.gradle comme ceci:

```
dependencies {  
    compile 'com.paypal.sdk:paypal-android-sdk:2.14.1'  
    ...  
}
```

```
}
```

Nous allons maintenant nous rendre sur notre fichier `MainActivity.java` (ou sur l'endroit où vous souhaitez ajouter l'intégration du bouton PayPal) et ajouter un objet de `config` pour notre ID client et l'environnement (sandbox) que nous allons utiliser.

```
private static PayPalConfiguration config = new PayPalConfiguration()  
    .environment(PayPalConfiguration.ENVIRONMENT_SANDBOX)  
    .clientId("YOUR CLIENT ID");
```

Nous allons maintenant créer un bouton dans notre `onCreate(...)`, ce qui nous permettra de traiter un paiement via PayPal une fois cliqué.

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    final Button button = (Button) findViewById(R.id.paypal_button);  
}
```

Nous devons maintenant définir les fonctionnalités de ce bouton. Dans votre fichier XML principal `res> layout>`, vous pouvez ajouter la définition suivante pour le bouton, qui définira le gestionnaire de texte et `onClick` pour le bouton avec l'ID `paypal_button`.

```
<Button android:id="@+id/paypal_button"  
    android:layout_height="wrap_content"  
    android:layout_width="wrap_content"  
    android:text="@string/paypal_button"  
    android:onClick="beginPayment" />
```

Lorsque vous cliquez dessus, le bouton appelle la `beginPayment(...)`. Nous pouvons alors ajouter le texte du bouton à notre fichier `strings.xml`, comme ceci:

```
<string name="paypal_button">Pay with PayPal</string>
```

Avec le bouton en place, nous devons maintenant gérer le clic du bouton pour commencer le traitement du paiement. Ajoutez la `beginPayment(...)` ci-dessous sous notre précédente méthode `onCreate(...)`.

```
public void beginPayment(View view) {  
    Intent serviceConfig = new Intent(this, PayPalService.class);  
    serviceConfig.putExtra(PayPalService.EXTRA_PAYPAL_CONFIGURATION, config);  
    startService(serviceConfig);  
  
    PayPalPayment payment = new PayPalPayment(new BigDecimal("5.65"),  
        "USD", "My Awesome Item", PayPalPayment.PAYMENT_INTENT_SALE);  
  
    Intent paymentConfig = new Intent(this, PaymentActivity.class);  
    paymentConfig.putExtra(PayPalService.EXTRA_PAYPAL_CONFIGURATION, config);  
    paymentConfig.putExtra(PaymentActivity.EXTRA_PAYMENT, payment);  
    startActivityForResult(paymentConfig, 0);  
}
```



```
}
```

Ce que nous faisons ici est d'abord de définir l'intention de service (`serviceConfig`), en utilisant la `config` que nous avons définie précédemment pour notre ID client et l'environnement sandbox. Nous spécifions ensuite l'objet de paiement que nous souhaitons traiter. Dans cet exemple, nous définissons un prix, une devise et une description statiques. Dans votre application finale, ces valeurs doivent être obtenues à partir de ce que l'utilisateur essaie d'acheter dans l'application. Enfin, nous avons configuré le `paymentConfig`, en ajoutant les objets de `config` et de `payment` précédemment définis et en démarrant l'activité.

À ce stade, l'utilisateur se verra présenter les écrans de connexion et de paiement PayPal, lui permettant de choisir entre payer avec PayPal ou une carte de crédit (via une saisie manuelle ou card.io si la caméra est disponible). Cet écran ressemblera à ceci:

Une fois cela fait, nous devons avoir un gestionnaire prêt pour que PayPal renvoie l'utilisateur à l'application après confirmation du paiement ou de l'annulation. `onActivityResult(...)` à cette fin.

```
@Override
protected void onActivityResult (int requestCode, int resultCode, Intent data){
    if (resultCode == Activity.RESULT_OK){
        PaymentConfirmation confirm = data.getParcelableExtra(
            PaymentActivity.EXTRA_RESULT_CONFIRMATION);
        if (confirm != null){
            try {
                Log.i("sampleapp", confirm.toJSONString().toString(4));

                // TODO: send 'confirm' to your server for verification

            } catch (JSONException e) {
                Log.e("sampleapp", "no confirmation data: ", e);
            }
        }
    } else if (resultCode == Activity.RESULT_CANCELED) {
        Log.i("sampleapp", "The user canceled.");
    } else if (resultCode == PaymentActivity.RESULT_EXTRAS_INVALID) {
        Log.i("sampleapp", "Invalid payment / config set");
    }
}
```

Dans la `onActivityResult(...)`, nous `resultCode` si le `resultCode` qui revient est `RESULT_OK` (paiement confirmé par l'utilisateur), `RESULT_CANCELED` (paiement annulé par l'utilisateur) ou `RESULT_EXTRAS_INVALID` (problème de configuration). Dans le cas d'une confirmation valide, nous récupérons l'objet renvoyé par le paiement et, dans cet exemple, le consignons. Ce qui nous sera retourné devrait ressembler à ceci:

```
{
  "client": {
    "environment": "sandbox",
    "paypal_sdk_version": "2.14.1",
    "platform": "Android",
    "product_name": "PayPal-Android-SDK"
  },
  "response": {
    "create_time": "2016-05-02T15:33:43Z",
```

```
        "id": "PAY-0PG63447RB821630KK1TXGTY",
        "intent": "sale",
        "state": "approved"
    },
    "response_type": "payment"
}
```

Si nous regardons sous l'objet de `response`, nous pouvons voir que nous avons un `state approved`, ce qui signifie que le paiement a été confirmé. À ce stade, cet objet doit être envoyé à votre serveur pour confirmer qu'un paiement a bien été effectué. Pour plus d'informations sur ces étapes, [consultez ces documents](#).

Notre dernière étape est de `onDestroy(...)` notre `onDestroy(...)`.

```
@Override
public void onDestroy() {
    stopService(new Intent(this, PayPalService.class));
    super.onDestroy();
}
```

C'est tout ce qu'on peut en dire. Dans cet exemple, nous avons créé un bouton simple pour traiter un paiement avec PayPal ou une carte de crédit. A partir de là, il y a quelques étapes à suivre pour développer cet exemple:

- Extraction dynamique des informations de paiement en fonction de la sélection du produit utilisateur dans la `beginPayment(...)`.
- Envoi de la confirmation de paiement à votre serveur et vérification du paiement effectif.
- Gestion des erreurs et des cas d'utilisateurs d'annulation dans l'application.

Lire Paypal Mobile / Paiements par carte de crédit en ligne:

<https://riptutorial.com/fr/paypal/topic/608/paypal-mobile---paiements-par-carte-de-credit>

Chapitre 7: Webhooks

Paramètres

| Paramètre | Détails |
|--------------------|---|
| app | Notre référence d'application express |
| bodyparser | La référence du package body-parser pour travailler avec des corps codés JSON |
| identité du client | L'ID du client d'application (informations d'identification OAuth 2) |
| http | Le package http pour exécuter le serveur |
| Pay Pal | Objet de référence PayPal Node SDK |
| secret | Le secret de l'application (informations d'identification OAuth 2) |
| webhookId | ID du webhook à modifier |
| webhookUpdate | Objet JSON contenant les détails du webhook à mettre à jour |

Remarques

Ces exemples couvrent des exemples pratiques d'utilisation des Webhooks de PayPal pour assurer la surveillance des événements de votre application et de vos paiements.

Exemples

Test des Webhooks Sandbox avec ngrok et Express (Node)

Dans cet exemple, nous allons tester les notifications Webhook dans Sandbox, en utilisant [ngrok](#) pour fournir un tunnel pour notre écouteur HTTP Node, exécuté sur localhost, sur Internet. Pour cet exemple, nous allons utiliser Node pour configurer les webhooks de notification pour les événements de paiement (comme un paiement en cours), puis configurer le serveur pour qu'il écoute les messages HTTP POST entrants provenant des événements Webhook.

Il y a quelques étapes que nous allons suivre ici pour que cela se produise:

1. Configurez un serveur simple pour écouter le trafic POST entrant depuis les webhooks, qui sera la notification de PayPal, et commencez à écouter localhost.
2. Utilisez ensuite ngrok pour fournir un tunnel à partir de localhost vers Internet afin que PayPal puisse envoyer une notification via.
3. Enfin, abonnez-vous à notre application (basée sur les informations d'identification fournies)

pour les événements Webhook que nous voulons suivre, en fournissant l'URI public ngrok à partir de l'étape 2.

Création d'un écouteur Webhooks

La première chose à faire est de créer l'auditeur. La raison pour laquelle nous commençons avec l'auditeur est que nous avons besoin de l'URL en direct ngrok à fournir aux webhooks lorsque nous les créons ou les mettons à jour.

```
var bodyParser = require('body-parser'),
    http = require('http'),
    app = require('express')();

app.use(bodyParser.json());

app.post('/', function(req, res){
  console.log(JSON.stringify(req.body));
});

//create server
http.createServer(app).listen(3001, function () {
  console.log('Server started: Listening on port 3001');
});
```

Notre auditeur est un itinéraire simple utilisant Express. Nous écoutons tout trafic POST entrant, puis crachons le corps POST sur la console. Nous pouvons l'utiliser pour faire ce que nous voulons avec l'auditeur quand il entre en jeu.

Lorsque nous créons le serveur HTTP à la fin, nous le configurons pour écouter sur le port 300 de l'hôte local. Exécutez ce script maintenant pour commencer à écouter le trafic.

Utilisation de ngrok pour exposer le récepteur à Internet

Avec l'écouteur configuré sur localhost: 3001, notre tâche suivante consiste à exposer ce script à Internet, afin que le trafic puisse lui être envoyé, ce qui est le travail de ngrok.

Exécutez la commande suivante depuis une fenêtre de terminal:

```
ngrok http 3001
```

Cela lancera le processus de fourniture d'un tunnel en direct pour localhost sur le port 3001 et fournira les informations suivantes une fois exécutées:

ngrok by @inconshreveable

```
Tunnel Status      online
Version            2.0.25/2.0.
Region             United Stat
Web Interface      http://127.
Forwarding         http://055b
Forwarding         https://055

Connections        ttl      opn
                   0        0
```

Comme nous pouvons le constater, l'adresse en direct que nous pouvons utiliser pour pointer l'accès payant à notre écouteur en cours sur localhost est `http(s)://055b3480.ngrok.io`. C'est tout ce que nous devons savoir pour configurer l'auditeur.

S'abonner aux notifications

Notre dernière étape consiste à créer des webhooks pour notre application, qui créeront des notifications lorsque certains événements se produiront avec des paiements, des remboursements, etc. sur notre application. Nous avons seulement besoin de créer ces webhooks une fois pour les lier à l'application, ils n'ont donc pas besoin d'être exécutés chaque fois que vous souhaitez les utiliser.

Tout d'abord, nous avons configuré l'environnement PayPal en ajoutant l'exigence pour le SDK Node PayPal, notre ID / secret client de créer une application, puis en configurant l'environnement pour Sandbox.

```
var paypal = require('paypal-rest-sdk');

var clientId = 'YOUR APPLICATION CLIENT ID';
var secret = 'YOUR APPLICATION SECRET';

paypal.configure({
  'mode': 'sandbox', //sandbox or live
  'client_id': clientId,
  'client_secret': secret
});
```

Ensuite, nous avons configuré la structure JSON pour nos webhooks. `webhooks` contient deux informations, l' `url` laquelle tous les événements webhook doivent être envoyés et les `event_types`

auxquels nous voulons nous abonner.

Dans le cas de cet exemple, l' `url` est définie sur notre URL ngrok live, et les événements que nous écoutons sont des cas où les paiements sont terminés ou refusés.

Pour une liste complète des événements potentiels, voir

<https://developer.paypal.com/docs/integration/direct/rest-webhooks-overview/#event-type-support>

Enfin, nous passons l'objet `webhooks` dans l'appel pour créer les webhooks, `notification.webhook.create`. En cas de succès, PayPal enverra des notifications au terminal que nous avons spécifié, qui s'exécute sur localhost.

```
var webhooks = {
  "url": "https://436e4d13.ngrok.io",
  "event_types": [{
    "name": "PAYMENT.SALE.COMPLETED"
  }, {
    "name": "PAYMENT.SALE.DENIED"
  }
]
};

paypal.notification.webhook.create(webhooks, function (err, webhook) {
  if (err) {
    console.log(err.response);
    throw error;
  } else {
    console.log("Create webhook Response");
    console.log(webhook);
  }
});
```

Une fois que nous avons émis un paiement à l'aide de ces informations d'identification, les informations sur l'état du paiement seront envoyées au terminal que nous avons configuré.

Un exemple du corps POST que PayPal envoie en tant que notification pourrait ressembler à ceci, qui a été envoyé après un paiement PayPal réussi:

```
{
  "id": "WH-9FE9644311463722U-6TR22899JY792883B",
  "create_time": "2016-04-20T16:51:12Z",
  "resource_type": "sale",
  "event_type": "PAYMENT.SALE.COMPLETED",
  "summary": "Payment completed for $ 7.47 USD",
  "resource": {
    "id": "18169707V5310210W",
    "state": "completed",
    "amount": {
      "total": "7.47",
      "currency": "USD",
      "details": {
        "subtotal": "7.47"
      }
    }
  },
  "payment_mode": "INSTANT_TRANSFER",
  "protection_eligibility": "ELIGIBLE",
```

```

"protection_eligibility_type": "ITEM_NOT_RECEIVED_ELIGIBLE,UNAUTHORIZED_PAYMENT_ELIGIBLE",
"transaction_fee": {
  "value": "0.52",
  "currency": "USD"
},
"invoice_number": "",
"custom": "",
"parent_payment": "PAY-809936371M327284GK4L3FHA",
"create_time": "2016-04-20T16:47:36Z",
"update_time": "2016-04-20T16:50:07Z",
"links": [
  {
    "href": "https://api.sandbox.paypal.com/v1/payments/sale/18169707V5310210W",
    "rel": "self",
    "method": "GET"
  },
  {
    "href":
"https://api.sandbox.paypal.com/v1/payments/sale/18169707V5310210W/refund",
    "rel": "refund",
    "method": "POST"
  },
  {
    "href": "https://api.sandbox.paypal.com/v1/payments/payment/PAY-
809936371M327284GK4L3FHA",
    "rel": "parent_payment",
    "method": "GET"
  }
]
},
"links": [
  {
    "href": "https://api.sandbox.paypal.com/v1/notifications/webhooks-events/WH-
9FE9644311463722U-6TR22899JY792883B",
    "rel": "self",
    "method": "GET"
  },
  {
    "href": "https://api.sandbox.paypal.com/v1/notifications/webhooks-events/WH-
9FE9644311463722U-6TR22899JY792883B/resend",
    "rel": "resend",
    "method": "POST"
  }
]
}

```

Mise à jour d'un Webhook avec une nouvelle URL (exemple de noeud)

Cet exemple va vous montrer comment mettre à jour une URL de redirection Webhook existante (où les notifications doivent être postées). Pour ce faire, vous devez avoir l'ID fourni par PayPal lors de la création de vos webhooks.

Tout d'abord, ajoutez le SDK PayPal et configurez l'environnement (sandbox ci-dessous).

```

var paypal = require('paypal-rest-sdk');

var clientId = 'YOUR APPLICATION CLIENT ID';
var secret = 'YOUR APPLICATION SECRET';

```

```
paypal.configure({
  'mode': 'sandbox', //sandbox or live
  'client_id': clientId,
  'client_secret': secret
});
```

Ensuite, configurez la structure JSON et les détails du Webhook. Attribuez d'abord l'identifiant de votre webhook à `webhookId` . Ensuite, dans `webhookUpdate` , spécifiez une opération `replace`, définissez le `path` vers `/url` pour spécifier une mise à jour de cette ressource et indiquez la nouvelle URL pour la remplacer par une `value` .

```
var webhookId = "YOUR WEBHOOK ID";
var webhookUpdate = [{
  "op": "replace",
  "path": "/url",
  "value": "https://64fb54a2.ngrok.io"
}];
```

Enfin, appelez `notification.webhook.replace(...)` en transmettant `webhookId` et `webhookUpdate` .

```
paypal.notification.webhook.replace (webhookId, webhookUpdate, function (err, res) {if (err)
{console.log (err); jeter err;} else {console.log (JSON.stringify (res));} })
```

Si tout se passe bien, un objet similaire à celui-ci doit être fourni par PayPal et, dans le cas de cet exemple, affiché dans le terminal avec les informations nouvellement mises à jour.

```
{
  "id": "4U496984902512511",
  "url": "https://64fb54a2.ngrok.io",
  "event_types": [{
    "name": "PAYMENT.SALE.DENIED",
    "description": "A sale payment was denied"
  }],
  "links": [{
    "href": "https://api.sandbox.paypal.com/v1/notifications/webhooks/4U496984902512511",
    "rel": "self",
    "method": "GET"
  }, {
    "href": "https://api.sandbox.paypal.com/v1/notifications/webhooks/4U496984902512511",
    "rel": "update",
    "method": "PATCH"
  }, {
    "href": "https://api.sandbox.paypal.com/v1/notifications/webhooks/4U496984902512511",
    "rel": "delete",
    "method": "DELETE"
  }],
  "httpStatusCode": 200
}
```

Lire Webhooks en ligne: <https://riptutorial.com/fr/paypal/topic/575/webhooks>

Crédits

| S. No | Chapitres | Contributeurs |
|-------|--|--|
| 1 | Démarrer avec PayPal | Community , Jonathan LeBlanc , Nathan Arthur |
| 2 | Création d'abonnements / paiements récurrents | Jonathan LeBlanc |
| 3 | Effectuer un paiement par carte de crédit (nœud) | Jonathan LeBlanc |
| 4 | Effectuer un paiement PayPal | Jonathan LeBlanc |
| 5 | Paiements futurs mobiles (application de bout en bout) | Jonathan LeBlanc |
| 6 | Paypal Mobile / Paiements par carte de crédit | Jonathan LeBlanc |
| 7 | Webhooks | Jonathan LeBlanc |