EBook Gratis

APRENDIZAJE PayPal

Free unaffiliated eBook created from
Stack Overflow contributors.

#paypal

Tabla de contenido

Acerca de

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.

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

Capítulo 1: Empezando con PayPal

Observaciones

Estas guías llevarán al usuario a través de los procedimientos de configuración de la cuenta para aplicaciones, cuentas, etc. Contendrá todo lo necesario para trabajar con las API de PayPal.

Versiones

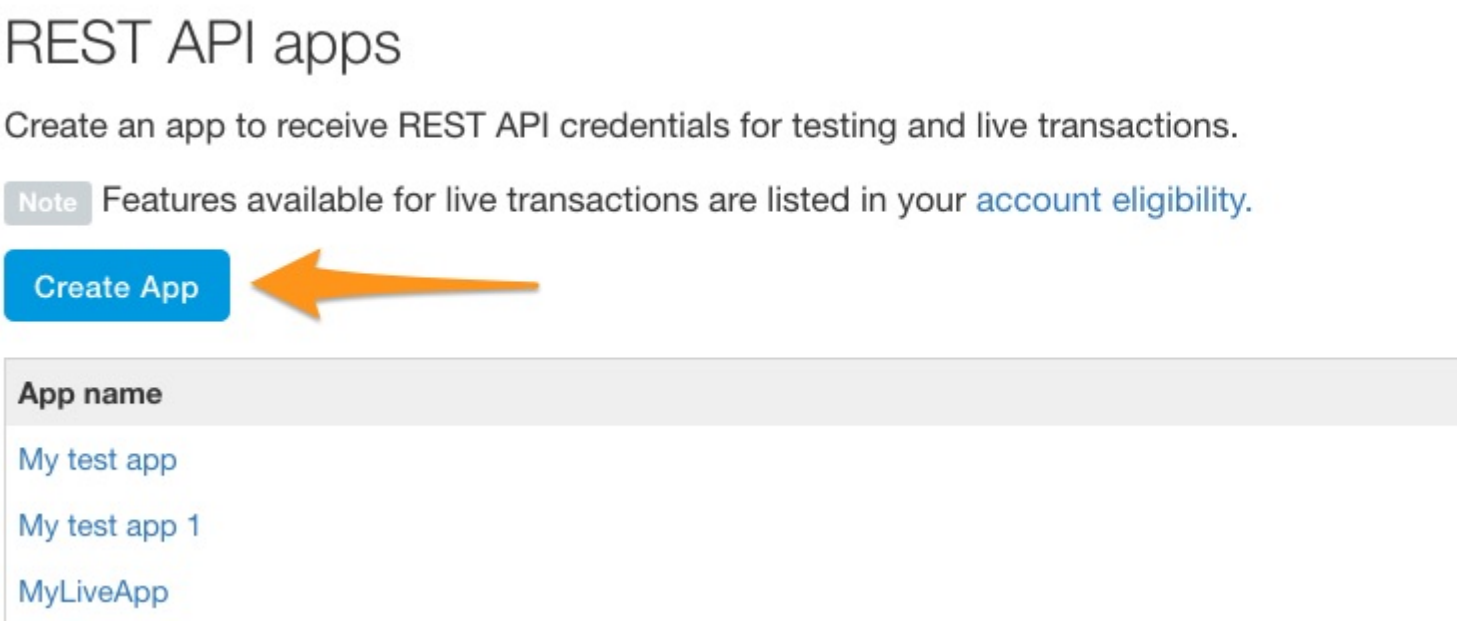
Versión	Fecha de lanzamiento
1.0.0	2016-04-11

Examples

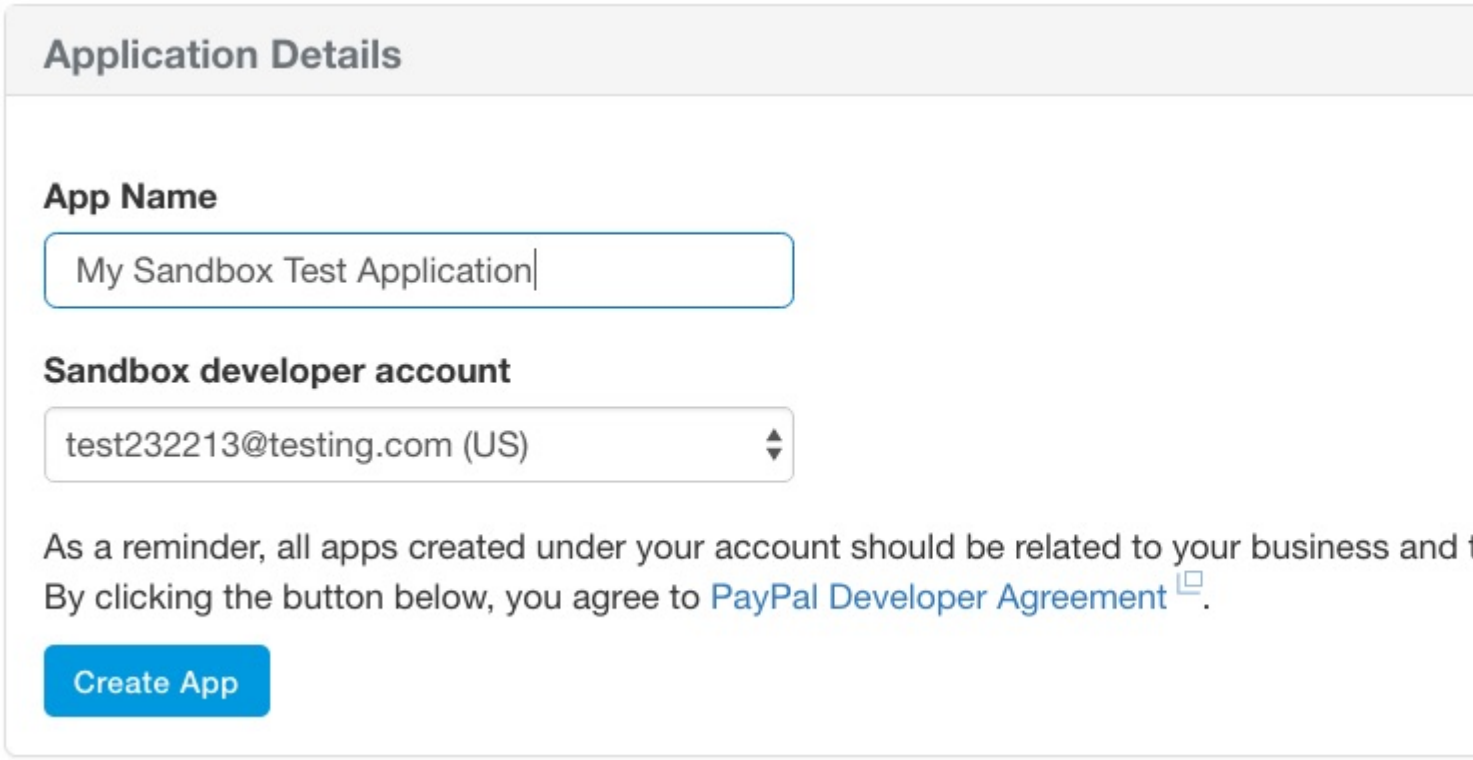
Creando una aplicación y obteniendo claves de identificación / secreto de cliente

Para comenzar a construir con las API de PayPal, debe crear una aplicación para obtener un ID de cliente y un secreto.

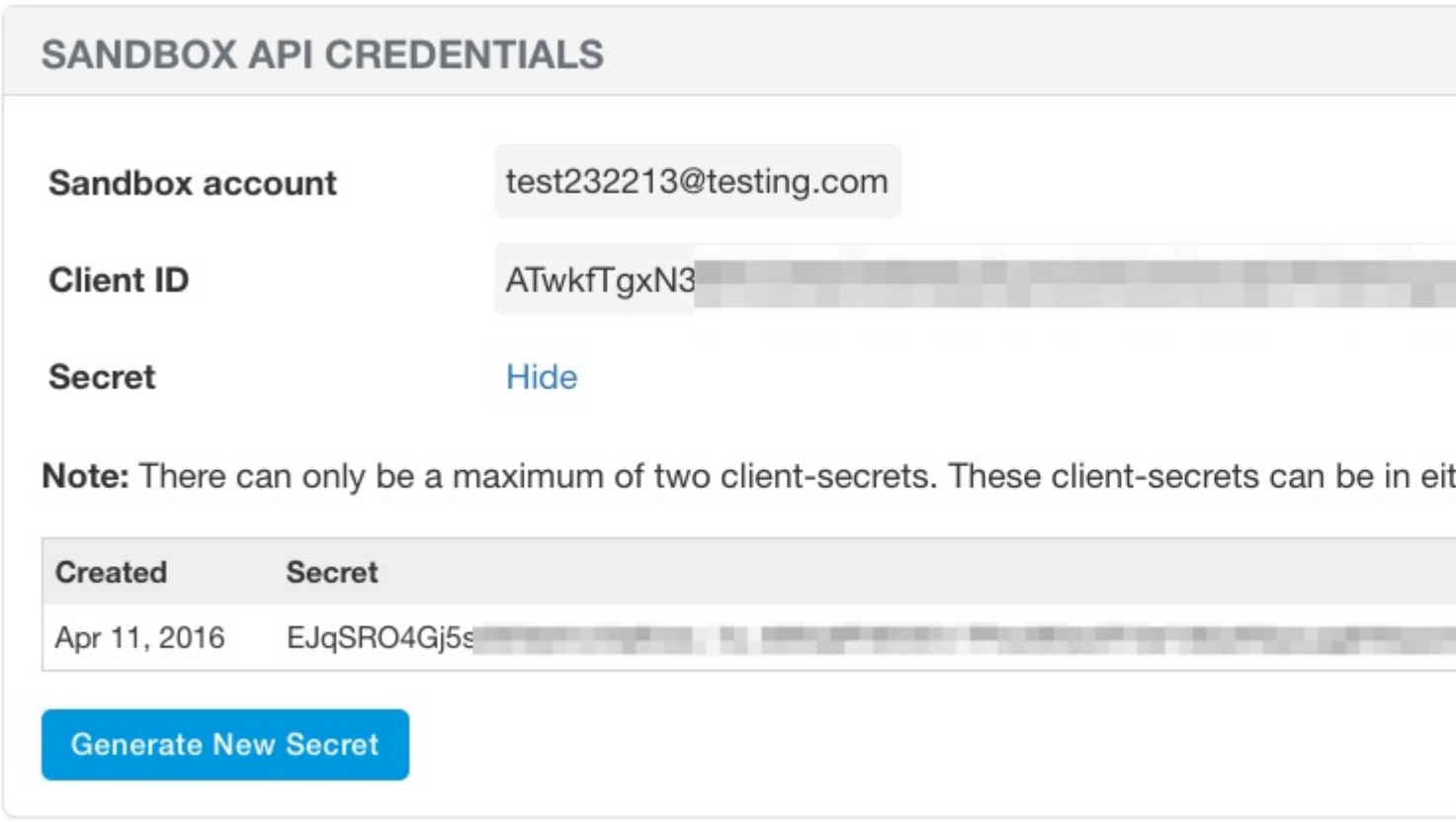
Vaya a https://developer.paypal.com/developer/applications/ , inicie sesión y haga clic en "Crear aplicación", como se muestra a continuación:



A continuación, ingrese el nombre de la aplicación, seleccione la cuenta de prueba de sandbox que desea usar (si es una cuenta nueva, deje el valor predeterminado) y haga clic en "Crear aplicación".



Una vez que se haya creado la aplicación, se le proporcionará su ID de sandbox y su ID de cliente en vivo y su secreto, que tendrá un aspecto similar al siguiente:

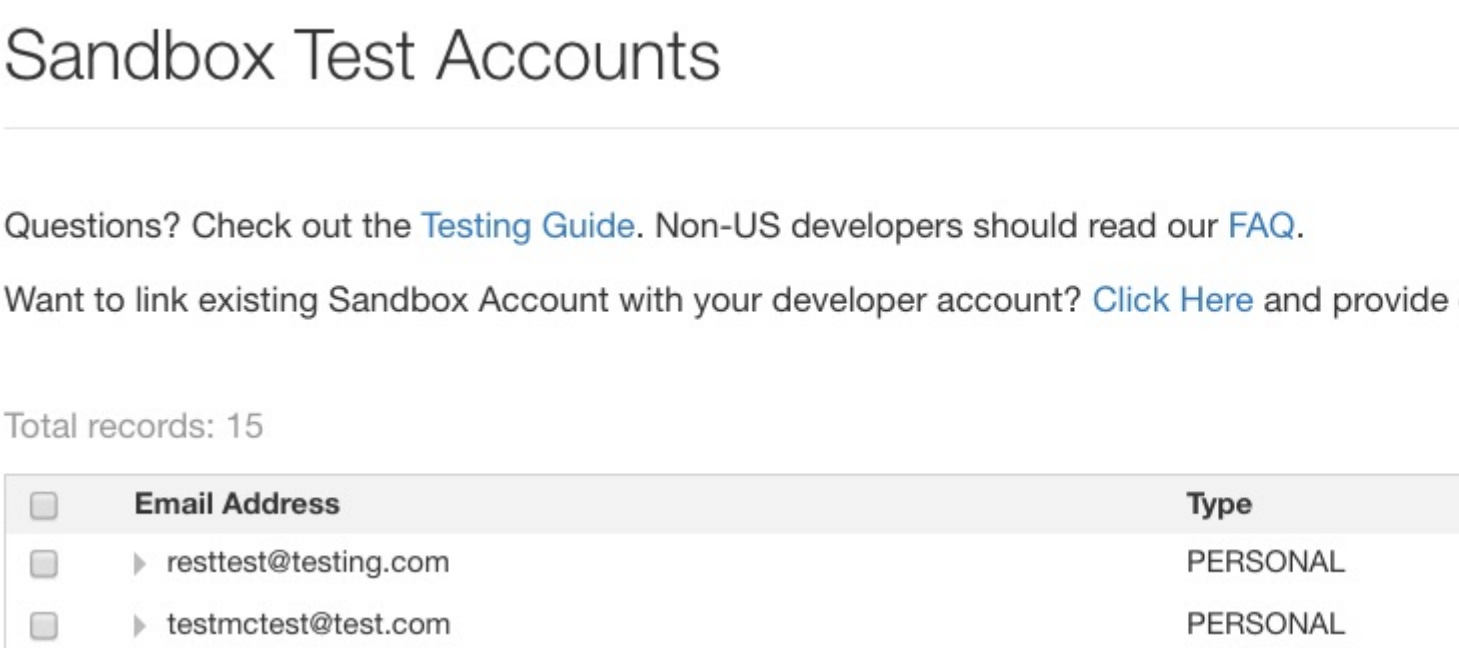


Estas credenciales son las que utilizará cuando realice solicitudes a las API de PayPal para autenticar su aplicación y realizar solicitudes.

Configuración de cuentas de prueba de usuario de sandbox

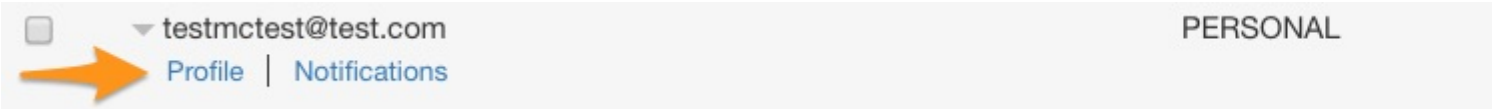
Al probar su integración de PayPal en el sandbox, deberá tener configuradas las cuentas de usuario de sandbox para utilizarlas en el flujo de pagos.

Vaya a https://developer.paypal.com/developer/accounts/ , inicie sesión con su cuenta de PayPal y haga clic en "Crear cuenta", como se muestra a continuación:



Ingrese los detalles de las cuentas para el nuevo usuario de prueba, incluido un correo electrónico único, información de la cuenta, método de pago, saldo, etc., y haga clic en "Crear cuenta" en la parte inferior de la página una vez que haya terminado. Esto creará la nueva cuenta para que comiences a usar.

Para ver los detalles de la cuenta de este nuevo usuario, expanda la entrada en la página de cuentas y haga clic en "Perfil".



Una vez que se cargue la información del perfil, al hacer clic en la pestaña "Financiamiento" se le proporcionará la información de pago de esa cuenta, incluida la información de la tarjeta de crédito que se puede usar para el procesamiento directo de la tarjeta de crédito contra el sandbox.

NOTA: Al usar los puntos finales de la API de sandbox, debe usar la cuenta de prueba de sandbox para iniciar sesión y pagar los bienes de prueba, ya que la información de su cuenta en vivo no funcionará.

Lea Empezando con PayPal en línea: https://riptutorial.com/es/paypal/topic/406/empezando-con-paypal

Capítulo 2: Creación de suscripciones / pagos periódicos

Parámetros

Parámetro	Detalles
billingAgreementAttributes	Objeto de configuración para crear el acuerdo de facturación.
plan de facturación	ID de plan de facturación de la cadena de consulta
billingPlanAttribs	Objeto de configuración para crear el plan de facturación.
billingPlanUpdateAttributes	Objeto de configuración para cambiar un plan de facturación a un estado activo
Identificación del cliente	Su ID de cliente de aplicación (claves OAuth)
http	Referencia al paquete http para configurar nuestro servidor simple.
isoDate	Fecha ISO para configurar la fecha de inicio de la suscripción.
campo de golf	Objeto de enlace HATEOAS para extraer la URL de redireccionamiento a PayPal
params	Parámetros de cadena de consulta
paypal	Referencia al SDK de PayPal
secreto	Su aplicación secreta (claves OAuth)
simbólico	El token de aprobación del acuerdo de facturación proporcionado después de que PayPal redirige para ejecutar el acuerdo de facturación

Observaciones

Estos ejemplos pasan por el proceso de creación de un sistema de suscripción / pago recurrente mediante PayPal.

El proceso para crear una suscripción es:

1. Crear un plan de facturación. Este es un modelo reutilizable que describe los detalles de la suscripción.

2. Activar el plan de facturación.
3. Cuando desea crear una suscripción para un usuario, crea un acuerdo de facturación utilizando el ID del plan de facturación al que deben suscribirse.
4. Una vez creado, redirige al usuario a PayPal para confirmar la suscripción. Una vez confirmado, el usuario es redirigido al sitio web del comerciante.
5. Por último, ejecute el acuerdo de facturación para comenzar la suscripción.

Examples

Paso 2: creación de una suscripción para un usuario mediante un acuerdo de facturación (ejemplo de nodo)

El segundo paso para crear una suscripción para un usuario es crear y ejecutar un acuerdo de facturación, basado en un plan de facturación activado existente. Este ejemplo asume que ya ha pasado y activado un plan de facturación en el ejemplo anterior, y tiene una identificación para que el plan de facturación haga referencia en el ejemplo.

Cuando esté configurando un acuerdo de facturación para crear una suscripción para un usuario, seguirá tres pasos, que pueden ser una reminiscencia de procesar un pago de PayPal:

1. Usted crea un acuerdo de facturación, haciendo referencia a un plan de facturación subyacente a través de la identificación.
2. Una vez creado, redirige al usuario a PayPal (si paga mediante PayPal) para confirmar la suscripción. Una vez confirmado, PayPal redirige al usuario a su sitio utilizando la redirección provista en el plan de facturación subyacente.
3. A continuación, ejecuta el acuerdo de facturación utilizando un token proporcionado a través de la redirección de PayPal.

Este ejemplo está configurando un servidor HTTP basado en Express para mostrar el proceso de acuerdo de facturación.

Para comenzar el ejemplo, primero necesitamos configurar nuestra configuración. Agregamos cuatro requisitos, el SDK de PayPal, `body-parser` de `body-parser` para manejar cuerpos codificados JSON, `http` para nuestra integración de servidor simple y `express` para el marco Express. Luego definimos nuestro ID de cliente y el secreto para crear una aplicación, configuramos el SDK para el recinto de seguridad y luego configuramos bodyParser para manejar cuerpos 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());
```

Nuestro primer paso en el acuerdo de facturación es crear una ruta para manejar la creación de un acuerdo de facturación, y redirigir al usuario a PayPal para confirmar esa suscripción. Suponemos que se pasa un ID de plan de facturación como un parámetro de cadena de consulta, como al cargar la siguiente URL con un ID de plan del ejemplo anterior:

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

Ahora necesitamos usar esa información para crear el acuerdo de facturación.

```
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');
            }
        }
    });
});
```

Comenzamos extrayendo el ID del plan de facturación de la cadena de consulta y creamos la fecha en que debe comenzar el plan.

La siguiente definición de objeto, `billingAgreementAttributes` , consiste en información para la suscripción. Contiene información legible sobre el plan, una referencia a la identificación del plan de facturación, el método de pago y los detalles de envío (si es necesario para la suscripción).

A continuación, se realiza una llamada a `billingAgreement.create(...)` , pasando el objeto `billingAgreementAttributes` que acabamos de crear. Si todo tiene éxito, deberíamos tener un objeto de acuerdo de facturación que nos sea devuelto con detalles sobre nuestra suscripción recién creada. Ese objeto también contiene una serie de enlaces de HATEOAS que nos proporcionan los próximos pasos que se pueden tomar en este acuerdo recién creado. El que nos importa aquí está etiquetado como `approval_url` .

Recorremos todos los enlaces provistos para colocarlos en un objeto de fácil referencia. Si `approval_url` es uno de esos enlaces, redirigimos al usuario a ese enlace, que es PayPal.

En este punto, el usuario confirma la suscripción en PayPal y se redirige de nuevo a la URL proporcionada en el plan de facturación subyacente. Junto con esa URL, PayPal también pasará un token a lo largo de la cadena de consulta. Ese token es lo que usaremos para ejecutar (o iniciar) la suscripción.

Vamos a configurar esa funcionalidad en la siguiente ruta.

```
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');
        }
    });
});
```

Extraemos el token de la cadena de consulta, luego hacemos una llamada a `billingAgreement.execute` , pasando ese token. Si todo tiene éxito, ahora tenemos una suscripción válida para el usuario. El objeto de devolución contiene información sobre el acuerdo de facturación activo.

Por último, configuramos nuestro servidor HTTP para escuchar el tráfico a nuestras rutas.

```
//create server
```

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

Paso 1: crear un modelo de suscripción utilizando un plan de facturación (ejemplo de nodo)

Al crear una suscripción para un usuario, primero debe crear y activar un plan de facturación al que se suscriba un usuario mediante un acuerdo de facturación. El proceso completo para crear una suscripción se detalla en las observaciones de este tema.

En este ejemplo, vamos a utilizar el SDK de nodo de PayPal . Puedes obtenerlo de NPM usando el siguiente comando:

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

Dentro de nuestro archivo .js, primero configuramos nuestra configuración de SDK, que incluye agregar un requisito para el SDK, definir nuestro ID de cliente y el secreto para crear nuestra aplicación , y luego configurar el SDK para el entorno de 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
});
```

A continuación, necesitamos configurar dos objetos JSON. El objeto `billingPlanAttribs` contiene la información y el desglose de pagos del plan de facturación al que podemos suscribir a los usuarios, y el objeto `billingPlanUpdateAttributes` contiene valores para configurar el plan de facturación en un estado activo, lo que permite su uso.

```
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"
    }
}];
```

Dentro del objeto `billingPlanAttribs` , hay algunos datos relevantes:

- **nombre / descripción / tipo** : información visual básica para describir el plan y el tipo de plan.
- **Payment_definitions** : Información sobre cómo debería funcionar y facturarse el plan. Más detalles sobre los campos aquí .
- **merchant_preferences** : estructuras de tarifas adicionales, URL de redireccionamiento y configuraciones para el plan de suscripción. Más detalles sobre los campos aquí .

Con esos objetos en su lugar, ahora podemos crear y activar el plan de facturación.

```
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);
            }
        });
    }
});
```

Llamamos a `billingPlan.create(...)` , pasando el objeto `billingPlanAttribs` que acabamos de crear. Si eso tiene éxito, el objeto devuelto contendrá información sobre el plan de facturación. Por el bien del ejemplo, solo necesitamos usar la ID del plan de facturación para activar el plan para su uso.

A continuación, llamamos a `billingPlan.update(...)` y `billingPlanUpdateAttributes` la ID del plan de facturación y el objeto `billingPlanUpdateAttributes` que creamos anteriormente. Si eso tiene éxito,

nuestro plan de facturación ahora está activo y listo para usar.

Para crear una suscripción para un usuario (o varios usuarios) en este plan, necesitaremos hacer referencia al ID del plan de facturación (`billingPlan.id` anterior), así que guárdelo en un lugar al que se pueda hacer referencia fácilmente.

En el segundo paso de suscripción, debemos crear un acuerdo de facturación basado en el plan que acabamos de crear y ejecutarlo para comenzar a procesar las suscripciones para un usuario.

Lea Creación de suscripciones / pagos periódicos en línea:
https://riptutorial.com/es/paypal/topic/467/creacion-de-suscripciones---pagos-periodicos

Capítulo 3: Hacer un pago con tarjeta de crédito (nodo)

Parámetros

Parámetro	Detalles
card_data	Objeto JSON que contiene información de pago por transacción
detalles de la tarjeta de crédito	Objeto JSON que contiene datos de tarjetas de crédito que se envían a PayPal para ser guardados
Identificación del cliente	Su ID de cliente de aplicación de PayPal (credenciales de OAuth 2)
paypal	Referencia del SDK del nodo de PayPal
secreto	Su solicitud de PayPal secreta (credenciales OAuth 2)
uuid	Referencia al paquete node-uuid

Observaciones

Esta muestra lleva al usuario a acreditar una transacción de tarjeta de crédito simple utilizando los SDK de PayPal.

Examples

Muestra de nodo

Comience por instalar el módulo Nodo de PayPal desde NPM

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

En su archivo de aplicación, agregue la información de configuración para el 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
```

```
});
```

Agregamos el requisito para el SDK, luego configuramos las variables para el ID de cliente y el secreto que se obtuvieron al crear una aplicación . A continuación, configuramos nuestra aplicación con estos detalles y especificamos el entorno en el que estamos trabajando (en vivo o en una zona de pruebas).

A continuación, configuramos el objeto JSON que contiene la información de pago para el pagador.

```
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."
}]};
```

Añadir una `intent` de `sale` , y una `payment_method` de `credit_card` . A continuación, agregue los detalles de la tarjeta y la dirección de la tarjeta de crédito en `funding_instruments` , y el monto a cobrar en las `transactions` . Aquí se pueden colocar múltiples objetos de transacción.

Por último, hacemos una solicitud a `payment.create(...)` , pasando nuestro objeto `card_data` , para procesar el pago.

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

Si la transacción fue exitosa, deberíamos ver un objeto de respuesta similar al siguiente:

```
{
  "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
}
```


En este objeto de devolución, tener un `state` de `approved` nos dice que la transacción se realizó correctamente. Bajo el objeto de `links` hay una cantidad de enlaces de HATEOAS que proporcionan los siguientes pasos potenciales que se pueden tomar en la acción que se acaba de realizar. En este caso, podemos recuperar información sobre el pago realizando una solicitud GET al punto final `self` proporcionado.

Realizar un pago con una tarjeta de crédito abovedada (nodo)

En este ejemplo, veremos cómo almacenar una tarjeta de crédito utilizando la bóveda de PayPal, luego haremos referencia a esa tarjeta de crédito almacenada para procesar una transacción de tarjeta de crédito para un usuario.

La razón por la que querríamos usar la bóveda es para no tener que almacenar información confidencial de tarjetas de crédito en nuestros propios servidores. Simplemente hacemos referencia al método de pago a través de un ID de almacenamiento proporcionado, lo que significa que no tenemos que lidiar con muchas regulaciones de cumplimiento de PCI con el almacenamiento de las tarjetas de crédito por nosotros mismos.

Al igual que con las muestras anteriores, comenzamos con la configuración de nuestro entorno.

```
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 única diferencia con respecto a las muestras anteriores aquí es que requerimos un nuevo paquete, `node-uuid` , que se utilizará para generar UUID únicos para los pagadores cuando se almacena la tarjeta. Puede instalar ese paquete a través de:

```
npm install node-uuid
```

A continuación, definimos el objeto JSON de la tarjeta de crédito que se enviará a la bóveda de PayPal para su almacenamiento. Contiene información de la tarjeta, así como un ID de pagador único que generamos mediante el uso de `node-uuid` . Debe almacenar este `payer_id` único en su propia base de datos, ya que se utilizará al crear un pago con la tarjeta de almacenamiento.

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

Por último, debemos almacenar la tarjeta de crédito y procesar el pago utilizando esa tarjeta. Para `credit_card.create(...)` una tarjeta de crédito, llamamos `credit_card.create(...)` , pasando el objeto `credit_card_details` que acabamos de crear. Si todo va bien, deberíamos devolver un objeto con detalles sobre la tarjeta abovedada. Por el bien de un pago con esa tarjeta, solo necesitamos realmente dos datos: el payer_id que ya almacenamos y el Id. De la bóveda, que también debe almacenarse como referencia en nuestra propia base de datos.

```
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));
            }
        });
    }
});
```

En la sección que sigue a la bóveda exitosa de la tarjeta de crédito, simplemente estamos definiendo los detalles de la tarjeta y procesando el pago, como hicimos con el ejemplo anterior del procesamiento de la tarjeta de crédito. La principal diferencia en la estructura del objeto `card_data` es la sección `funding_instruments` , que definimos bajo `payer` . En lugar de definir la información de la tarjeta de crédito, en su lugar usamos el siguiente objeto que contiene la referencia de la ID de la bóveda y la ID del pagador:

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

Así es como utilizamos una tarjeta abovedada para procesar un pago.

Lea Hacer un pago con tarjeta de crédito (nodo) en línea:
https://riptutorial.com/es/paypal/topic/444/hacer-un-pago-con-tarjeta-de-credito--nodo-

Capítulo 4: Hacer un pago de PayPal

Parámetros

Parámetro	Detalles
Identificación del cliente	Su ID de cliente de aplicación de PayPal (credenciales de OAuth 2)
campo de golf	Objeto de referencia simple para todos los enlaces HATEOAS devueltos de PayPal
pagoId	El ID del pago devuelto por PayPal para completar el pago.
payerId	El ID del pagador devuelto de PayPal para completar el pago
paypal	Referencia del SDK del nodo de PayPal
payReq	Objeto JSON que contiene información de pago por transacción
req	El objeto de solicitud de la solicitud del servidor.
res	El objeto de respuesta de la solicitud del servidor.
secreto	Su solicitud de PayPal secreta (credenciales OAuth 2)

Observaciones

Estas muestras cubren cómo procesar un pago a través de PayPal, utilizando los SDK de PayPal. Estos son ejemplos de solicitudes simples que describen el proceso de varios pasos para permitir esta opción de pago.

Examples

Ejemplo de Node Express Server

En este ejemplo, vamos a configurar una integración de servidor Express para mostrar cómo procesar un pago con PayPal, utilizando el SDK de nodo de PayPal. Usaremos una estructura JSON estática para los detalles de pago en aras de la brevedad.

Hay tres pasos generales que seguiremos cuando construyamos las funciones para manejar el pago de PayPal:

1. Creamos un objeto JSON que contiene el pago que pretendemos procesar a través de PayPal. Luego lo enviamos a PayPal para obtener un enlace para redirigir al usuario para

confirmar el pago.
2. A continuación, redirigimos al usuario a PayPal para confirmar el pago. Una vez confirmado, PayPal redirige al usuario a nuestra aplicación.
3. Una vez devuelto a la aplicación, completamos el pago en nombre del usuario.

Desglosando esto como una aplicación de Nodo simple, comenzamos por obtener el SDK de Nodo de PayPal de NPM:

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

A continuación, configuramos la configuración de la aplicación y los paquetes.

```
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
});
```

Requerimos cuatro requisitos para esta aplicación:

1. El paquete HTTP para nuestro servidor.
2. El paquete SDK del nodo de PayPal.
3. El paquete bodyParser para trabajar con cuerpos codificados JSON.
4. El framework Express para nuestro servidor.

Las siguientes líneas configuran variables para el ID de cliente y el secreto que se obtuvieron al crear una aplicación . Luego configuramos `bodyParser` para permitir cuerpos codificados en JSON, luego configuramos nuestra aplicación usando los detalles de la aplicación y especificamos el entorno en el que estamos trabajando (en vivo para producción o en una zona de pruebas para prueba).

Ahora vamos a crear la ruta para crear una solicitud de pago con 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');
            }
        }
    });
});
```

Lo primero que hacemos es configurar el objeto JSON de solicitud de pago, que contiene la información que debemos proporcionar a PayPal para crear el pago. Establecemos la `intent` de la `sale` , especificamos las URL de redireccionamiento (donde PayPal debe reenviar al usuario después de que confirmen / cancelen el pago), agregamos un `payment_method` de pago de `paypal` para indicar que haremos un pago de PayPal, luego especificamos la información de la transacción para El pagador para confirmar.

Luego llamamos a `payment.create(...)` , pasando nuestro objeto `payReq` . Esto enviará la solicitud de creación de pago a PayPal. Una vez que se devuelve, y tiene éxito, podemos recorrer los enlaces de HATEOAS proporcionados en el objeto de retorno para extraer la URL a la que necesitamos redireccionar al usuario, que está etiquetada bajo `approval_url` .

El formato para los enlaces HATEOAS puede causar un código de referencia frágil si se usa directamente, por lo que hacemos un bucle a través de todos los enlaces proporcionados y los colocamos en un objeto de referencia mejor para futuras pruebas contra los cambios. Si se encuentra la `approval_url` en ese objeto, redirigimos al usuario.

En este punto, el usuario es redirigido a PayPal para confirmar el pago. Una vez que lo hacen, se redirigen de nuevo al `return_url` que especificamos en la función `createPayment(...)` .

Ahora tenemos que proporcionar una ruta para manejar esa devolución, a fin de completar el pago.

```
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');
            }
        }
    });
});
```

Cuando el usuario vuelva a su aplicación, habrá tres parámetros de cadena de consulta que también se enviarán: `paymentId` , `PayerID` y `token` . Solo tenemos que tratar con los dos primeros.

Extraemos los parámetros y colocamos el PayerID en un objeto simple para la necesidad del paso de ejecución de pago. A continuación, se realiza una llamada a `payment.execute(...)` , pasando esos dos parámetros, para completar el pago.

Una vez que se realiza esa solicitud, vemos si el pago se completó con éxito al verificar si el estado de `payment.state` está configurado como `approved` . Si es así, podemos almacenar lo que necesitamos del objeto de pago que se devuelve.

Nuestro último paso es inicializar nuestro servidor y escuchar el tráfico que llega a las rutas que especificamos.

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

Una vez que se inicializa el servidor, vaya a `http://localhost:3000/create` inicializa el proceso de pago.

Lea Hacer un pago de PayPal en línea: https://riptutorial.com/es/paypal/topic/449/hacer-un-pago-de-paypal

Capítulo 5: Pagos futuros móviles (aplicación de extremo a extremo)

Observaciones

Este ejemplo muestra un ejemplo práctico de principio a fin de crear un pago futuro de PayPal desde un dispositivo Android, utilizando un servidor Node.

Examples

Paso 1 de Android: diseño, inicialización y manejo de la respuesta del servidor

El código de muestra completo para esta aplicación (servidor Android + Node) está disponible en el repositorio de Github para desarrolladores de PayPal .

La primera etapa de crear la parte de Android de nuestra aplicación es configurar un diseño básico y manejar las respuestas que provienen del servidor que configuraremos en Node.

Comience por crear un nuevo objeto PayPalConfiguration para alojar la información de su aplicación.

```
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"));
```

A continuación, agregamos un botón simple a `onCreate(...)` para que actúe como nuestra iniciación de pago. Esto es simplemente para desencadenar la acción, y debe colocarse como el proceso de inicio para crear un pago futuro para un usuario (por ejemplo, cuando acuerdan una suscripción).

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

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

En `res > layout > activity_main.xml` agregamos la definición del botón con su acción asociada, cuando se hace clic, se llama `beginFuturePayment(...)` , que definiremos en un minuto.

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

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

Bajo `res > values > strings.xml` , luego agregamos una referencia de cadena para el botón.

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

Ahora agregamos el controlador de botones, para iniciar la llamada para comenzar el proceso de pago futuro cuando el usuario haga clic en el botón. Lo que estamos haciendo aquí es iniciar el servicio de pago con el objeto de configuración que configuramos en la parte superior de este ejemplo.

```
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);
}
```

Cuando se inicie la llamada para realizar un pago en el futuro, se nos proporcionará cierta información que deberá enviarse a nuestro servidor. `authCode` esta información de la solicitud de pago futuro válida (`authCode` y `metadataId`), luego ejecutamos la solicitud asíncrona al servidor para completar el pago futuro (detallado en el paso 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");
    }
}
```

Por último, definimos nuestro `onDestroy()` .

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

Android Paso 2: Async Server Request

El código de muestra completo para esta aplicación (servidor Android + Node) está disponible en el repositorio de Github para desarrolladores de PayPal .

En este punto, se hizo clic en el botón de pagos futuros de PayPal, tenemos un código de autenticación y un ID de metadatos del SDK de PayPal, y debemos pasarlos a nuestro servidor para completar el proceso de pago futuro.

En el proceso de fondo a continuación, estamos haciendo algunas cosas:

- Configuramos el URI para que nuestro servidor sea `http://10.0.2.2:3000/fpstore` , que está llegando al `/fpstore` final `/fpstore` de nuestro servidor que se ejecuta en localhost.
- Luego se configura el objeto JSON que se enviará a través, que contiene el código de autenticación y el ID de metadatos.
- Luego se realiza la conexión. En el caso de una solicitud exitosa (rango 200/201) podemos esperar una respuesta del servidor. Leemos esa respuesta y luego la devolvemos.
- Por último, tenemos un `onPostExecute(...)` configurado para manejar esa cadena de servidor devuelta. En el caso de este ejemplo, simplemente se registra.

```
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);
    }
}
```

Paso 3 de Android: Servidor de nodo para obtener el token de acceso y procesar el pago

El código de muestra completo para esta aplicación (servidor Android + Node) está disponible en el repositorio de Github para desarrolladores de PayPal .

Desde el paso 2, se realizó una solicitud asíncrona a nuestro servidor en el `/fpstore` final `/fpstore`

, pasando el código de autenticación y el ID de metadatos. Ahora necesitamos cambiarlos por un token para completar la solicitud y procesar el pago futuro.

Primero configuramos nuestras variables de configuración y objeto.

```
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());
```

Ahora configuramos una ruta Express que escuchará las solicitudes POST enviadas al punto final `/fpstore` desde nuestro código de Android.

Estamos haciendo una serie de cosas en esta ruta:

- Capturamos el código de autenticación y el ID de metadatos del cuerpo POST.
- Luego realizamos una solicitud para `generateToken()` , pasando por el objeto de código. Si tiene éxito, obtenemos un token que se puede utilizar para crear el pago.
- A continuación, los objetos de configuración se crean para el pago futuro que se realizará, y se realiza una solicitud a `payment.create(...)` , que pasa a través de los pagos futuros y los objetos de configuración de pago. Esto crea el pago futuro.

```
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));
                }
            });
        }
    });
});
```

Por último, creamos el servidor para escuchar en el puerto 3000.

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

Lea Pagos futuros móviles (aplicación de extremo a extremo) en línea: https://riptutorial.com/es/paypal/topic/4537/pagos-futuros-moviles--aplicacion-de-extremo-a-extremo-

Capítulo 6: PayPal móvil / pagos con tarjeta de crédito

Parámetros

Parámetro	Detalles
botón	Botón de pago simple
configuración	Objeto de configuración de PayPal que contiene nuestro ID de cliente (desde la creación de la aplicación) y el entorno que queremos usar (sandbox o en vivo)
pago	Detalles de pago de PayPal
pagoConfig	Intención de configuración para la información de pago y la configuración
serviceConfig	Intención de configuración para los datos de parámetros de configuración

Observaciones

Muestras relacionadas con el procesamiento de pagos en dispositivos móviles.

Examples

Android: aceptar un pago con tarjeta de crédito / PayPal

En este tutorial, aprenderemos cómo configurar el SDK de Android de PayPal para procesar un pago simple mediante un pago de PayPal o una compra con tarjeta de crédito. Al final de este ejemplo, debe tener un botón simple en una aplicación que, al hacer clic, reenviará al usuario a PayPal para confirmar un pago establecido, luego devolverá al usuario a la aplicación y registrará la confirmación del pago.

El código de aplicación completo para este ejemplo está disponible en el repositorio Github para desarrolladores de PayPal .

Empecemos.

El primer paso es obtener y agregar el SDK a su proyecto . Añadimos la referencia a nuestras dependencias de build.gradle así:

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

```
}
```

Ahora nos dirigimos a nuestro archivo MainActivity.java (o donde quiera que desee agregar la integración del botón de PayPal), y agregamos un objeto de `config` para la ID de nuestro cliente y el entorno (recinto de seguridad) que usaremos.

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

Ahora vamos a crear un botón en nuestro `onCreate(...)` , que nos permitirá procesar un pago a través de PayPal una vez que se haga clic.

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

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

Ahora necesitamos definir la funcionalidad de ese botón. En su archivo XML> res> diseño> puede agregar la siguiente definición para el botón, que definirá el texto y el controlador de clic para el botón con el 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" />
```

Cuando se hace clic, el botón llamará al `beginPayment(...)` . Luego podemos agregar el texto del botón a nuestro archivo strings.xml, de esta manera:

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

Con el botón en su lugar, ahora tenemos que manejar el clic del botón para comenzar a procesar el pago. Agregue el siguiente `beginPayment(...)` debajo de nuestro método `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);
}
```

Lo que estamos haciendo aquí es configurar primero la intención del servicio (`serviceConfig`), utilizando la `config` que habíamos definido previamente para nuestra ID de cliente y el entorno de sandbox. Luego especificamos el objeto de pago que queremos procesar. Por el bien de este ejemplo, estamos estableciendo un precio, una moneda y una descripción estáticos. En su aplicación final, estos valores deben obtenerse de lo que el usuario está tratando de comprar en la aplicación. Por último, configuramos `paymentConfig` , agregando tanto la `config` como `payment` objetos de `payment` que habíamos definido previamente, y comenzamos la actividad.

En este punto, al usuario se le presentarán las pantallas de inicio de sesión y pago de PayPal, lo que les permitirá seleccionar si desea pagar con PayPal o con una tarjeta de crédito (mediante entrada manual o card.io si la cámara está disponible). Esa pantalla se verá algo así:

Una vez hecho esto, debemos tener un controlador listo para cuando PayPal reenvíe al usuario a la aplicación después de la confirmación del pago o la cancelación. Vamos a anular `onActivityResult(...)` para ese propósito.

```
@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.toJSONObject().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");
    }
}
```

Dentro del `onActivityResult(...)` , estamos comprobando si el `resultCode` que se devuelve es RESULT_OK (pago confirmado por el usuario), RESULT_CANCELED (pago cancelado por el usuario) o RESULT_EXTRAS_INVALID (hubo un problema de configuración). En el caso de una confirmación válida, obtenemos el objeto que se devuelve del pago y, en este ejemplo, lo registramos. Lo que nos será devuelto debería ser similar al siguiente:

```
{
    "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 observamos el objeto de `response` , podemos ver que tenemos un `state` de `approved` , lo que significa que el pago se confirmó. En este punto, ese objeto debe enviarse a su servidor para confirmar que realmente se realizó un pago. Para obtener más información sobre esos pasos, consulte estos documentos .

Nuestro último paso es limpiar nuestro `onDestroy(...)` .

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

Eso es todo al respecto. En este ejemplo, hemos creado un botón simple para procesar un pago con PayPal o con una tarjeta de crédito. A partir de este punto, hay algunos pasos siguientes para que pueda ampliar esta muestra:

- Recopilación de información de pago de forma dinámica según la selección de productos del usuario en el `beginPayment(...)` .
- Enviar la confirmación de pago a su servidor y verificar que el pago realmente se realizó.
- Manejo de los casos de error y cancelación de usuarios dentro de la aplicación.

Lea PayPal móvil / pagos con tarjeta de crédito en línea:
https://riptutorial.com/es/paypal/topic/608/paypal-movil---pagos-con-tarjeta-de-credito

Capítulo 7: Webhooks

Parámetros

Parámetro	Detalles
aplicación	Nuestra referencia de aplicación Express
bodyParser	La referencia del paquete body-parser para trabajar con cuerpos codificados JSON
Identificación del cliente	El ID del cliente de la aplicación (credenciales OAuth 2)
http	El paquete http para ejecutar el servidor
paypal	El objeto de referencia SDK del nodo de PayPal
secreto	La aplicación secreta (credenciales OAuth 2)
webhookId	ID del webhook a modificar.
webhookUpdate	Objeto JSON que contiene los detalles del webhook a actualizar

Observaciones

Estos ejemplos cubren ejemplos prácticos de cómo usar los webhooks de PayPal para proporcionar monitoreo de eventos para su aplicación y pagos.

Examples

Pruebas de Sandbox Webhooks con ngrok y Express (Nodo)

En este ejemplo, veremos cómo probar las notificaciones de webhook en sandbox, utilizando ngrok para proporcionar un túnel para nuestro oyente HTTP Node, que se ejecuta en localhost, a Internet. Para este ejemplo, vamos a utilizar Node para configurar notificaciones web para eventos de pago (como un pago), y luego configurar el servidor para escuchar los mensajes POST HTTP entrantes de los eventos webhook.

Hay algunos pasos que vamos a seguir aquí para que esto suceda:

1. Configure un servidor simple para escuchar el tráfico POST entrante de los webhooks, que será la notificación de PayPal, y comience a escuchar en localhost.
2. Luego use ngrok para proporcionar un túnel desde localhost a Internet para que PayPal pueda publicar una notificación.

3. Por último, suscriba nuestra aplicación (basada en las credenciales proporcionadas) a los eventos de webhook que deseamos rastrear, proporcionando el URI público de ngrok desde el paso 2.

Creando un Oyente de Webhooks

Lo primero que tenemos que hacer es crear el oyente. La razón por la que estamos empezando con el oyente es porque necesitamos la URL ngrok live para proporcionar a los webhooks cuando los creamos o actualizamos.

```
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');
});
```

Nuestro oyente es una ruta simple usando Express. Escuchamos cualquier tráfico POST entrante, luego escupimos el cuerpo POST a la consola. Podemos usar esto para hacer lo que queramos con el oyente cuando se presente.

Cuando creamos el servidor HTTP al final, lo configuramos para escuchar en el puerto 3001 de localhost. Ejecute ese script ahora para comenzar a escuchar el tráfico.

Usando ngrok para exponer el Oyente a Internet

Con el servicio de escucha configurado en localhost: 3001, nuestro siguiente trabajo es exponer ese script a Internet, para que se pueda enviar el tráfico, que es el trabajo de ngrok.

Ejecute el siguiente comando desde una ventana de terminal:

```
ngrok http 3001
```

Eso iniciará el proceso de proporcionar un túnel en vivo para localhost en el puerto 3001, y proporcionará la siguiente información una vez que se ejecute:

Como podemos ver, la dirección en vivo que podemos usar para dirigir el webhook de PayPal a nuestro oyente en ejecución en localhost es `http(s)://055b3480.ngrok.io` . Eso es todo lo que necesitamos saber para configurar al oyente.

Suscripción a notificaciones

Nuestro último paso es crear webhooks para nuestra aplicación, que creará notificaciones cuando ocurran ciertos eventos con pagos, reembolsos, etc. en nuestra aplicación. Solo necesitamos crear estos webhooks una vez para vincularlos a la aplicación, por lo que no es necesario que se ejecuten cada vez que desee usarlos.

Primero configuramos el entorno de PayPal agregando el requisito para el SDK del nodo de PayPal, nuestro ID de cliente / secreto para crear una aplicación, y luego configuramos el entorno para el entorno de pruebas.

```
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
});
```

A continuación, configuramos la estructura JSON para nuestros webhooks. `webhooks` contiene dos elementos de información, la `url` que deben enviarse todos los eventos de webhook y los `event_types` que queremos suscribirnos.

En el caso de este ejemplo, la `url` se establece en nuestra URL en vivo de ngrok, y los eventos que estamos escuchando son casos en los que los pagos se completan o se rechazan.

Para obtener una lista completa de posibles eventos, consulte https://developer.paypal.com/docs/integration/direct/rest-webhooks-overview/#event-type-support .

Por último, pasamos el objeto de `webhooks` a la llamada para crear los webhooks, `notification.webhook.create` . Si tiene éxito, PayPal enviará notificaciones al punto final que especificamos, que se está ejecutando en 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);
    }
});
```

Una vez que emitimos un pago utilizando esas credenciales de aplicación, la información sobre el estado de pago se enviará al punto final que configuramos.

Un ejemplo del cuerpo de POST que PayPal envía como la notificación podría ser similar al siguiente, que se envió después de un pago exitoso de PayPal:

```
{
  "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"
    }
  ]
}
```

Actualización de un webhook con una nueva URL (ejemplo de nodo)

Este ejemplo le mostrará cómo actualizar una URL de reenvío de webhook existente (donde las notificaciones deben POSTARSE). Para ejecutar esto, debe tener la ID proporcionada por PayPal cuando creó sus webhooks por primera vez.

En primer lugar, agregue el SDK de PayPal y configure el entorno (en el siguiente cuadro de arena).

```
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
});
```

A continuación, configure la estructura JSON y los detalles de webhook. Asigne la ID de su webhook a `webhookId` primero. A continuación, en el `webhookUpdate` , especifique una operación de reemplazo, establezca la `path` a `/url` para especificar una actualización de ese recurso y proporcione la nueva URL para reemplazarla con un `value` inferior.

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

Por último, llame a `notification.webhook.replace(...)` , pasando en `webhookId` y `webhookUpdate` .

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

Si todo tiene éxito, un objeto similar al siguiente debe ser devuelto por PayPal y, en el caso de esta muestra, debe mostrarse en el terminal con la información recién actualizada.

```
{
    "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
}
```

Lea Webhooks en línea: https://riptutorial.com/es/paypal/topic/575/webhooks

Creditos

S. No	Capítulos	Contributors
1	Empezando con PayPal	Community, Jonathan LeBlanc, Nathan Arthur
2	Creación de suscripciones / pagos periódicos	Jonathan LeBlanc
3	Hacer un pago con tarjeta de crédito (nodo)	Jonathan LeBlanc
4	Hacer un pago de PayPal	Jonathan LeBlanc
5	Pagos futuros móviles (aplicación de extremo a extremo)	Jonathan LeBlanc
6	PayPal móvil / pagos con tarjeta de crédito	Jonathan LeBlanc
7	Webhooks	Jonathan LeBlanc