Kostenloses eBook

LERNEN PayPal

Free unaffiliated eBook created from
Stack Overflow contributors.

#paypal

Inhaltsverzeichnis

Über

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

Kapitel 1: Erste Schritte mit PayPal

Bemerkungen

Diese Handbücher führen den Benutzer durch die Kontoeinrichtung für Anwendungen, Konten usw.. Sie enthalten alles, was für die Arbeit mit den PayPal-APIs erforderlich ist.

Versionen

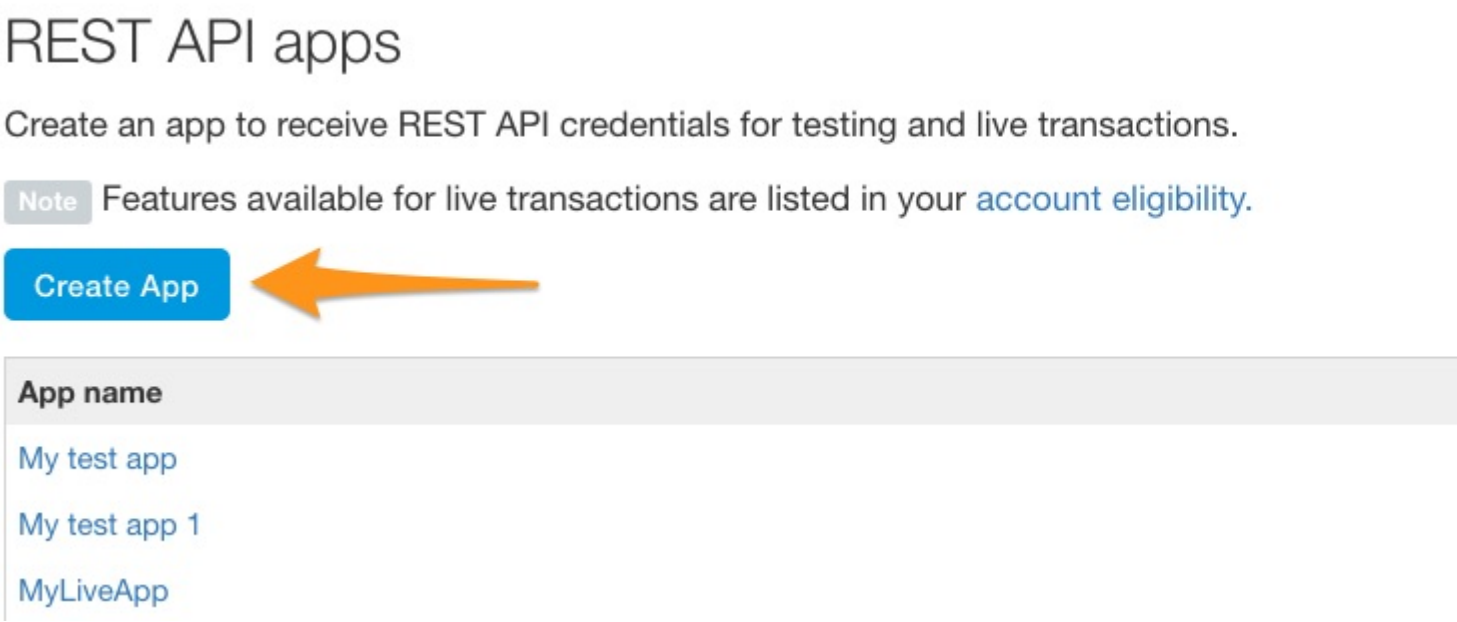
Ausführung	Veröffentlichungsdatum
1.0.0	2016-04-11

Examples

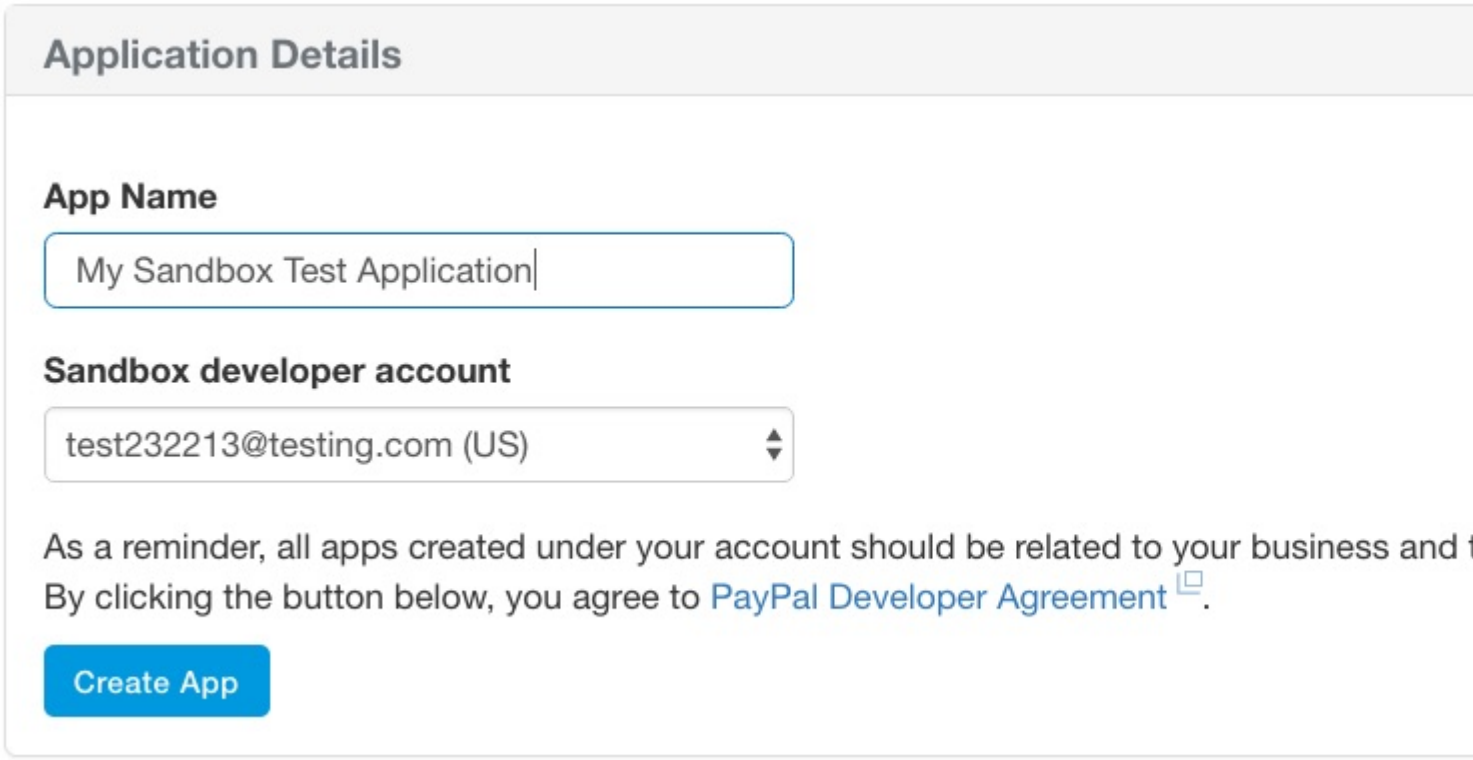
Anwendung erstellen und Client-ID / geheime Schlüssel abrufen

Um mit dem Erstellen von PayPal-APIs beginnen zu können, müssen Sie eine Anwendung erstellen, um eine Client-ID und ein Geheimnis zu erhalten.

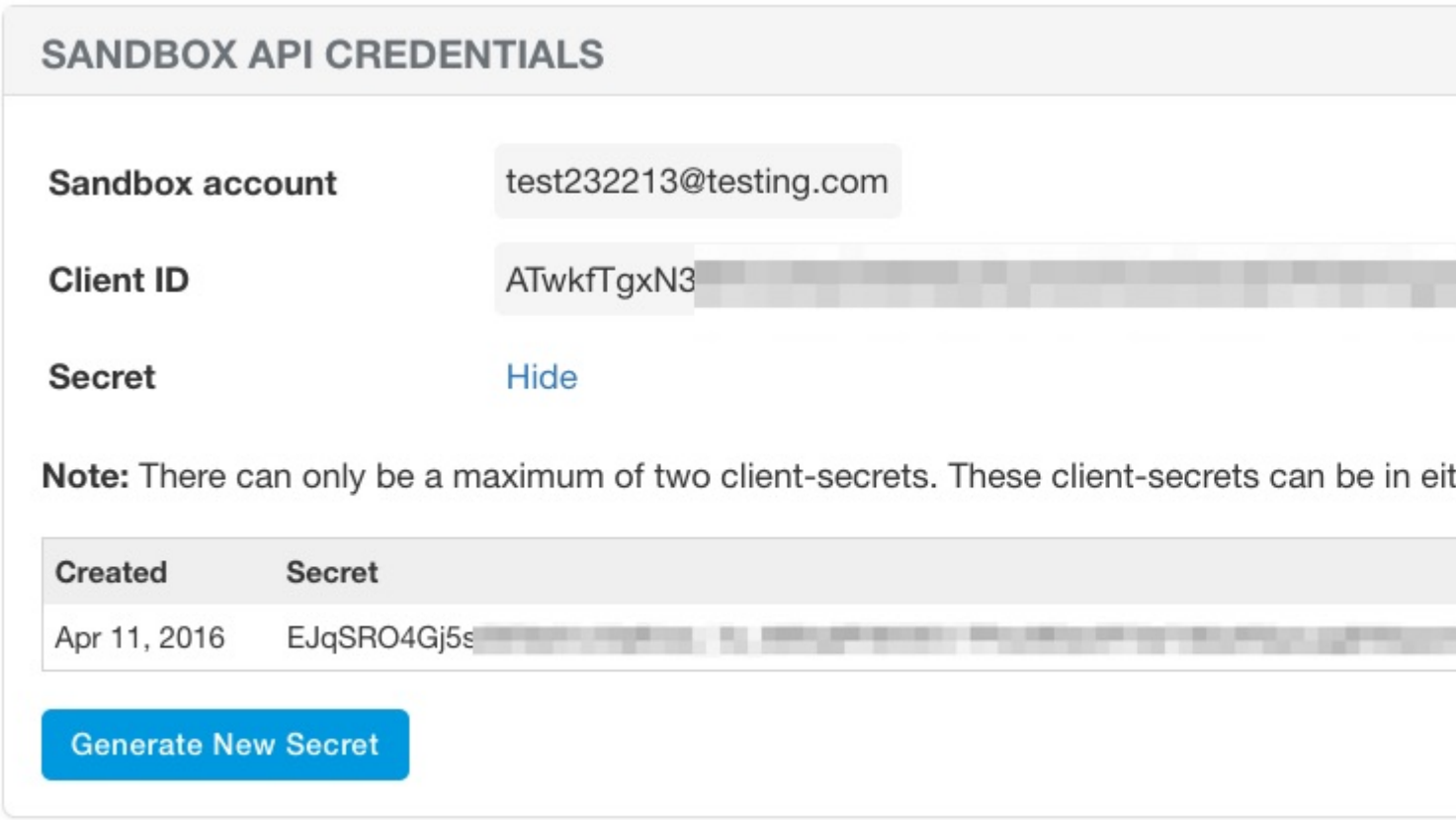
Gehen Sie zu https://developer.paypal.com/developer/applications/ , melden Sie sich an und klicken Sie auf "App erstellen" (siehe unten):



Geben Sie als Nächstes einen Anwendungsnamen ein, wählen Sie das Sandbox-Testkonto aus, das Sie verwenden möchten (wenn es ein neues Konto ist, übernehmen Sie den Standardwert), und klicken Sie auf "App erstellen".



Sobald die Anwendung erstellt wurde, erhalten Sie Ihre Sandbox sowie Ihre Live-Client-ID und Ihr Secret, die ungefähr wie folgt aussehen:

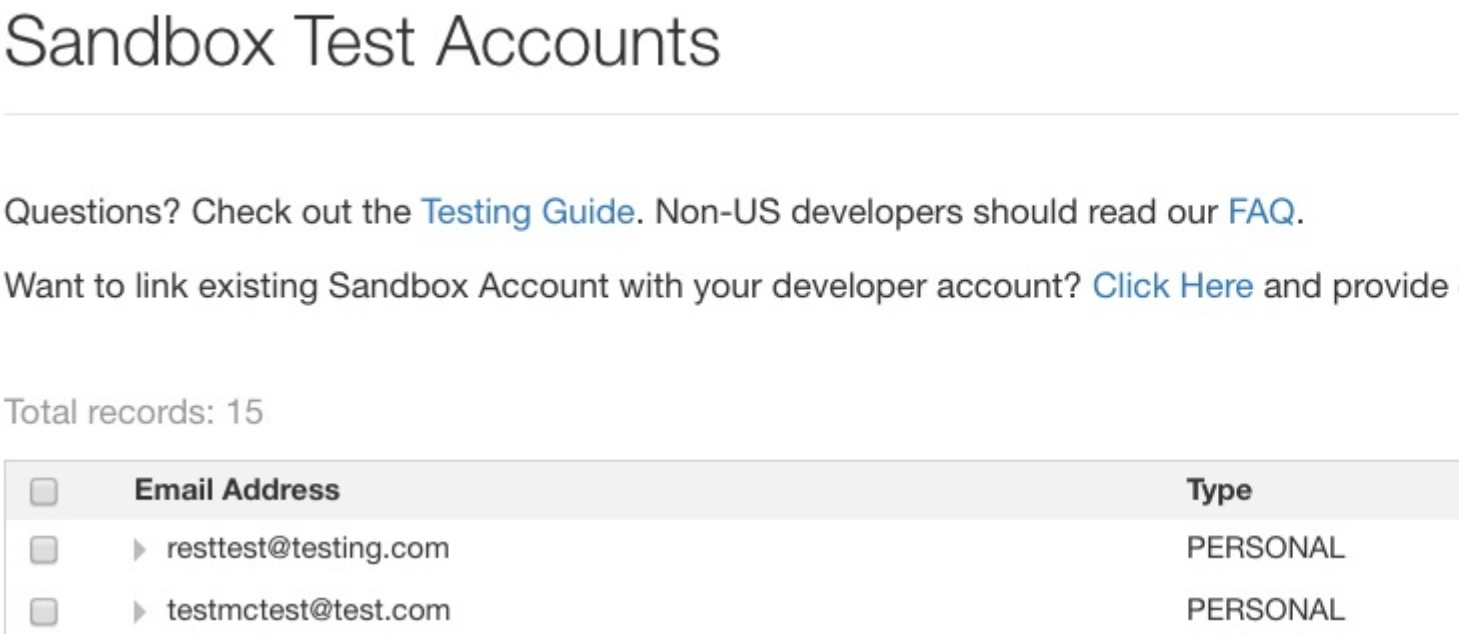


Diese Anmeldeinformationen verwenden Sie, wenn Sie Anfragen an PayPal-APIs stellen, um Ihre Anwendung zu authentifizieren und Anfragen zu stellen.

Einrichten von Sandbox-Benutzertestkonten

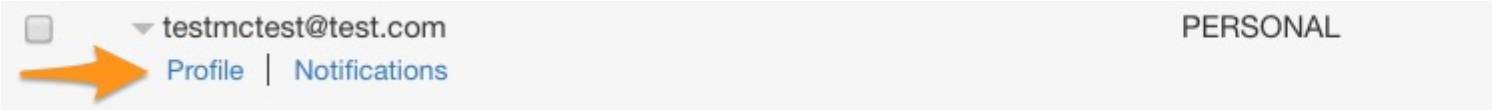
Wenn Sie Ihre PayPal-Integration in Sandbox testen, müssen Sie Sandbox-Benutzerkonten einrichten, um den Zahlungsfluss durchlaufen zu können.

Gehen Sie zu https://developer.paypal.com/developer/accounts/ , melden Sie sich mit Ihrem PayPal-Konto an und klicken Sie wie folgt auf "Konto erstellen":



Geben Sie die Kontodetails für den neuen Testbenutzer ein, einschließlich einer eindeutigen E-Mail-Adresse, Kontoinformationen, Zahlungsmethode, Kontostand usw., und klicken Sie am Ende der Seite auf "Konto erstellen". Dadurch wird das neue Konto erstellt, das Sie verwenden können.

Um die Kontodetails für diesen neuen Benutzer anzuzeigen, erweitern Sie den Eintrag auf der Kontoseite und klicken Sie auf "Profil".



Sobald diese Profilinformationen geladen sind, erhalten Sie durch Klicken auf die Registerkarte "Finanzierung" Zahlungsinformationen für dieses Konto, einschließlich Kreditkarteninformationen, die für die direkte Kreditkartenverarbeitung mit Sandbox verwendet werden können.

HINWEIS: Wenn Sie die Sandbox-API-Endpunkte verwenden, müssen Sie das Sandbox-Testkonto verwenden, um sich anzumelden und die Testware zu bezahlen, da Ihre Live-Kontoinformationen nicht funktionieren.

Erste Schritte mit PayPal online lesen: https://riptutorial.com/de/paypal/topic/406/erste-schritte-mit-paypal

Kapitel 2: Abonnements anlegen / wiederkehrende Zahlungen

Parameter

Parameter	Einzelheiten
billingAgreementAttributes	Konfigurationsobjekt zum Erstellen der Abrechnungsvereinbarung
rechnungsplan	Fakturierungsplan-ID aus der Abfragezeichenfolge
billingPlanAttribs	Konfigurationsobjekt zum Erstellen des Fakturierungsplans
billingPlanUpdateAttributes	Konfigurationsobjekt zum Ändern eines Fakturierungsplans in einen aktiven Status
Kunden ID	Ihre Anwendungsclient-ID (OAuth-Schlüssel)
http	Verweis auf das http-Paket zum Einrichten unseres einfachen Servers
isoDate	ISO-Datum zum Festlegen des Startdatums des Abonnements
Links	HATEOAS-Linkobjekt zum Extrahieren der Weiterleitungs-URL an PayPal
Params	Abfragezeichenfolgeparameter
paypal	Verweis auf das PayPal SDK
Geheimnis	Ihr Anwendungsgeheimnis (OAuth-Schlüssel)
Zeichen	Das Billing Agreement-Genehmigungstoken, das nach der PayPal-Weiterleitung bereitgestellt wird, um den Billing Agreement auszuführen

Bemerkungen

Diese Beispiele durchlaufen den Prozess der Erstellung eines Abonnements / wiederkehrenden Zahlungssystems mit PayPal.

Das Erstellen eines Abonnements umfasst folgende Schritte:

1. Erstellen Sie einen Abrechnungsplan. Dies ist ein wiederverwendbares Modell, das die

Details des Abonnements beschreibt.
2. Aktivieren Sie den Abrechnungsplan.
3. Wenn Sie ein Abonnement für einen Benutzer erstellen möchten, erstellen Sie eine Abrechnungsvereinbarung mit der ID des Abrechnungsplans, den sie abonnieren sollen.
4. Nach dem Erstellen leiten Sie den Benutzer zu PayPal weiter, um das Abonnement zu bestätigen. Nach der Bestätigung wird der Benutzer zur Website des Händlers zurückgeleitet.
5. Zuletzt führen Sie die Abrechnungsvereinbarung aus, um das Abonnement zu beginnen.

Examples

Schritt 2: Erstellen eines Abonnements für einen Benutzer mithilfe einer Abrechnungsvereinbarung (Knotenbeispiel)

Der zweite Schritt zum Erstellen eines Abonnements für einen Benutzer ist das Erstellen und Ausführen einer Abrechnungsvereinbarung auf Grundlage eines vorhandenen aktivierten Abrechnungsplans. In diesem Beispiel wird davon ausgegangen, dass Sie im vorherigen Beispiel bereits einen Fakturierungsplan durchlaufen und aktiviert haben und über eine ID für diesen Fakturierungsplan verfügen, auf die im Beispiel verwiesen werden soll.

Wenn Sie eine Abrechnungsvereinbarung einrichten, um ein Abonnement für einen Benutzer zu erstellen, führen Sie drei Schritte aus, die an die Verarbeitung einer PayPal-Zahlung erinnern können:

1. Sie legen eine Fakturierungsvereinbarung an, die über die ID einen zugrunde liegenden Fakturierungsplan referenziert.
2. Nach dem Erstellen leiten Sie den Benutzer zu PayPal (bei Zahlung per PayPal), um das Abonnement zu bestätigen. Nach der Bestätigung leitet PayPal den Benutzer mithilfe der im zugrunde liegenden Abrechnungsplan angegebenen Weiterleitung auf Ihre Website zurück.
3. Anschließend führen Sie die Abrechnungsvereinbarung mit einem Token aus, das über die PayPal-Weiterleitung zurückgegeben wird.

In diesem Beispiel wird ein Express-basierter HTTP-Server eingerichtet, um den Rechnungslegungsprozess darzustellen.

Um das Beispiel zu starten, müssen wir zuerst unsere Konfiguration einrichten. Wir fügen vier Anforderungen hinzu, das PayPal-SDK, den `body-parser` für die Verarbeitung von JSON-codierten Körpern, `http` für unsere einfache Serverintegration und `express` für das Express-Framework. Wir definieren dann unsere Client-ID und das Geheimnis der Erstellung einer Anwendung, konfigurieren das SDK für die Sandbox und konfigurieren dann bodyParser für die Verarbeitung von JSON-Körpern.

```
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());
```

Der erste Schritt in der Abrechnungsvereinbarung besteht darin, eine Route für die Erstellung einer Abrechnungsvereinbarung zu erstellen und den Benutzer zur Bestätigung dieses Abonnements an PayPal weiterzuleiten. Wir gehen davon aus, dass eine Abrechnungsplan-ID als Parameter für die Abfragezeichenfolge übergeben wird, beispielsweise durch Laden der folgenden URL mit einer Plan-ID aus dem vorherigen Beispiel:

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

Wir müssen diese Informationen nun verwenden, um die Abrechnungsvereinbarung zu erstellen.

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

Wir extrahieren zunächst die Fakturierungsplan-ID aus der Abfragezeichenfolge und erstellen das Datum, an dem der Plan beginnen soll.

Die nächste Objektdefinition, `billingAgreementAttributes` , besteht aus Informationen für das Abonnement. Es enthält lesbare Informationen zum Plan, einen Verweis auf die Fakturierungsplan-ID, die Zahlungsmethode und die Versanddetails (falls für das Abonnement erforderlich).

Als Nächstes wird ein Aufruf von `billingAgreement.create(...)` , der das soeben erstellte `billingAgreementAttributes` Objekt `billingAgreementAttributes` . Wenn alles erfolgreich ist, sollten wir einen Rechnungsvereinbarungsgegenstand mit Details zu unserem neu erstellten Abonnement an uns zurücksenden lassen. Dieses Objekt enthält auch eine Reihe von HATEOAS-Links, die uns die nächsten Schritte ermöglichen, die mit dieser neu erstellten Vereinbarung unternommen werden können. Die, die uns wichtig ist, wird als `approval_url` .

Wir durchlaufen alle bereitgestellten Links, um sie in einem leicht referenzierten Objekt abzulegen. Wenn `approval_url` einer dieser Links ist, leiten wir den Benutzer zu diesem Link weiter, der PayPal ist.

An diesem Punkt bestätigt der Benutzer das Abonnement bei PayPal und wird zu der im zugrunde liegenden Abrechnungsplan angegebenen URL zurückgeleitet. Neben dieser URL wird auch ein Token an die Abfragezeichenfolge übergeben. Dieses Token verwenden wir zum Ausführen (oder Starten) des Abonnements.

Lassen Sie uns diese Funktionalität auf folgende Weise einrichten.

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

Wir extrahieren das Token aus der `billingAgreement.execute` , rufen dann `billingAgreement.execute` und übergeben dieses Token. Wenn alles erfolgreich ist, haben wir jetzt ein gültiges Abonnement für den Benutzer. Das Rückgabeobjekt enthält Informationen zur aktiven Fakturierungsvereinbarung.

Schließlich haben wir unseren HTTP-Server so eingerichtet, dass er den Verkehr auf unseren Routen überwacht.

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

Schritt 1: Erstellen eines Abonnementmodells mithilfe eines Fakturierungsplans (Knotenbeispiel)

Wenn Sie ein Abonnement für einen Benutzer erstellen, müssen Sie zunächst einen Abrechnungsplan erstellen und aktivieren, den ein Benutzer dann mit einer Abrechnungsvereinbarung abonniert. Der vollständige Vorgang zum Erstellen eines Abonnements wird in den Anmerkungen zu diesem Thema beschrieben.

In diesem Beispiel verwenden wir das PayPal-Node-SDK . Sie können es von NPM mit dem folgenden Befehl erhalten:

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

In unserer .js-Datei haben wir zunächst unsere SDK-Konfiguration eingerichtet. Dazu gehört das Hinzufügen einer Anforderung für das SDK, das Definieren unserer Client-ID und des Geheims für das Erstellen unserer Anwendung sowie das Konfigurieren des SDK für die Sandbox-Umgebung.

```
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
});
```

Als Nächstes müssen wir zwei JSON-Objekte einrichten. Das `billingPlanAttribs` Objekt enthält die Informationen und Zahlungsaufschlüsselung für den Abrechnungsplan, die wir Benutzer abonnieren können, und das `billingPlanUpdateAttributes` Objekt enthält Werte für das Setzen des Abrechnungsplans auf einen aktiven Status, sodass es verwendet werden kann.

```
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"
    }
}];
```

Im Objekt `billingPlanAttribs` gibt es einige relevante Informationen:

- **Name / Beschreibung / Typ** : Grundlegende visuelle Informationen zur Beschreibung des Plans und der Art des Plans.
- **payment_definitions** : Informationen darüber, wie der Plan funktionieren soll und in Rechnung gestellt wird. Weitere Details zu den Feldern hier .
- **merchant_preferences** : Zusätzliche Gebührenstrukturen, Weiterleitungs-URLs und Einstellungen für den Abonnementplan. Weitere Details zu den Feldern hier .

Mit diesen Objekten können wir jetzt den Abrechnungsplan erstellen und aktivieren.

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

```
    }
});
```

Wir rufen `billingPlan.create(...)` und übergeben das soeben erstellte `billingPlanAttribs` Objekt. Wenn dies erfolgreich ist, enthält das Rückgabeobjekt Informationen zum Fakturierungsplan. In diesem Beispiel müssen Sie lediglich die Fakturierungsplan-ID verwenden, um den Plan für die Verwendung zu aktivieren.

Als Nächstes rufen wir `billingPlan.update(...)` und übergeben die Fakturierungsplan-ID und das bereits erstellte `billingPlanUpdateAttributes` Objekt. Wenn dies erfolgreich ist, ist unser Abrechnungsplan jetzt aktiv und einsatzbereit.

Um ein Abonnement für einen Benutzer (oder mehrere Benutzer) für diesen Plan zu erstellen, müssen wir auf die Rechnungsplan-ID (`billingPlan.id` oben) `billingPlan.id` , diese also an einem Ort speichern, der leicht referenziert werden kann.

Im zweiten Abonnementschritt müssen wir basierend auf dem soeben erstellten Plan eine Abrechnungsvereinbarung erstellen und ausführen, um Abonnements für einen Benutzer zu verarbeiten.

Abonnements anlegen / wiederkehrende Zahlungen online lesen:
https://riptutorial.com/de/paypal/topic/467/abonnements-anlegen---wiederkehrende-zahlungen

Kapitel 3: Kreditkartenzahlung durchführen (Knoten)

Parameter

Parameter	Einzelheiten
card_data	JSON-Objekt, das Zahlungsinformationen für die Transaktion enthält
Kreditkartendetails	JSON-Objekt, das Kreditkartendaten enthält, die zur Übergabe an PayPal gesendet werden
Kunden ID	Ihre PayPal-Anwendungs-Client-ID (OAuth 2-Berechtigungsnachweis)
paypal	Referenz zum PayPal-Node-SDK
Geheimnis	Ihr PayPal-Antragsgeheimnis (OAuth 2-Berechtigungsnachweise)
uuid	Verweis auf das node-uuid-Paket

Bemerkungen

In diesem Beispiel wird dem Benutzer die Gutschrift einer einfachen Kreditkartentransaktion mit den PayPal-SDKs empfohlen.

Examples

Knotenprobe

Beginnen Sie mit der Installation des PayPal-Node-Moduls von NPM

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

Fügen Sie in Ihrer Anwendungsdatei die Konfigurationsinformationen für das SDK hinzu

```
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
});
```

Wir fügen die Anforderung für das SDK hinzu und richten dann Variablen für die Client-ID und das Geheimnis ein, die beim Erstellen einer Anwendung abgerufen wurden. Wir konfigurieren dann unsere Anwendung anhand dieser Details und geben die Umgebung an, in der wir arbeiten (Live oder Sandbox).

Als Nächstes richten Sie das JSON-Objekt ein, das die Zahlungsinformationen für den Zahlungspflichtigen enthält.

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

Fügen Sie eine `intent` des `sale` , und eine `payment_method` von `credit_card` . `funding_instruments` die Kreditkarten- und Adressdaten für die Kreditkarte `funding_instruments` und den unter `transactions` `funding_instruments` Betrag hinzu. Hier können mehrere Transaktionsobjekte abgelegt werden.

Als letztes `payment.create(...)` wir an `payment.create(...)` und übergeben unser `card_data` Objekt, um die Zahlung zu verarbeiten.

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

Wenn die Transaktion erfolgreich war, sollte ein Antwortobjekt ähnlich dem folgenden angezeigt werden:

```
{
  "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
}
```

In diesem Rückkehrobjekt wird der `state` der `approved` , wenn die Transaktion erfolgreich war. Unter dem `links` Objekt befinden sich eine Reihe von HATEOAS- Links, die potenzielle nächste Schritte für die gerade ausgeführte Aktion darstellen. In diesem Fall können wir Informationen über die Zahlung abrufen, indem Sie eine GET-Anforderung an den angegebenen `self` senden.

Zahlung mit einer Vaulted Credit Card (Node) durchführen

In diesem Beispiel wird erläutert, wie eine Kreditkarte mit dem PayPal-Tresor gespeichert wird. Anschließend wird auf diese gespeicherte Kreditkarte verwiesen, um eine Kreditkartentransaktion für einen Benutzer zu verarbeiten.

Der Grund, warum wir den Tresor verwenden möchten, ist, dass wir keine sensiblen Kreditkarteninformationen auf unseren eigenen Servern speichern müssen. Wir referenzieren die Zahlungsmethode einfach über eine bereitgestellte Tresor-ID. Das bedeutet, dass Sie sich nicht mit vielen PCI-Compliance-Vorschriften befassen müssen, wenn Sie die Kreditkarten selbst speichern.

Wie bei den vorherigen Beispielen beginnen wir mit der Einrichtung unserer Umgebung.

```
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
});
```

Der einzige Unterschied zu den vorherigen Beispielen besteht darin, dass wir ein neues Paket, `node-uuid` , benötigen, das zum Generieren eindeutiger UUIDs für die Zahler beim Speichern der Karte verwendet wird. Sie können dieses Paket über Folgendes installieren:

```
npm install node-uuid
```

Als Nächstes definieren wir das Kreditkarten-JSON-Objekt, das zur Speicherung an den PayPal-Tresor gesendet wird. Es enthält Informationen von der Karte sowie eine eindeutige Kennzahl, die wir mithilfe von `node-uuid` generieren. Sie sollten diese eindeutige `payer_id` in Ihrer eigenen Datenbank speichern, da sie beim Erstellen einer Zahlung mit der Vault-Karte verwendet wird.

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

Schließlich müssen wir die Kreditkarte speichern und die Zahlung mit dieser Karte abwickeln. Um eine Kreditkarte zu `credit_card.create(...)` , rufen wir `credit_card.create(...)` und übergeben das gerade erstellte Objekt `credit_card_details` . Wenn alles gut geht, sollten wir uns einen Gegenstand mit Details über die gewölbte Karte zurücksenden lassen. Für eine Zahlung mit dieser Karte benötigen wir wirklich nur zwei Informationen: die bereits gespeicherte payer_id und die Tresor-ID, die auch als Referenz in unserer eigenen Datenbank gespeichert werden sollte.

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

In dem Abschnitt nach erfolgreicher Übergabe der Kreditkarte definieren wir einfach die Kartendaten und bearbeiten die Zahlung, wie im vorherigen Beispiel der Kreditkartenverarbeitung. Der Hauptunterschied in der Struktur des `card_data` ist der Abschnitt " `funding_instruments` ", den wir unter " `payer` definieren. Anstatt die Kreditkarteninformationen zu definieren, verwenden wir stattdessen das folgende Objekt, das die Tresor-ID-Referenz und die Zahler-ID enthält:

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

So verwenden wir eine gewölbte Karte, um eine Zahlung zu verarbeiten.

Kreditkartenzahlung durchführen (Knoten) online lesen:
https://riptutorial.com/de/paypal/topic/444/kreditkartenzahlung-durchfuhren--knoten-

Kapitel 4: Mobile Future Payments (Ende-zu-Ende-App)

Bemerkungen

Dieses Beispiel zeigt ein praktisches End-to-End-Beispiel für das Erstellen einer zukünftigen PayPal-Zahlung von einem Android-Gerät unter Verwendung eines Knotenservers.

Examples

Android Schritt 1: Layout, Initialisierung und Handhabung der Serverantwort

Den vollständigen Beispielcode für diese Anwendung (Android + Node-Server) finden Sie im PayPal Developer Github-Repository .

Der erste Schritt beim Erstellen des Android-Teils unserer Anwendung besteht darin, ein grundlegendes Layout festzulegen und Antworten zu verarbeiten, die von dem Server stammen, den wir in Node einrichten.

Erstellen Sie zunächst ein neues PayPalConfiguration-Objekt, um Ihre Anwendungsinformationen unterzubringen.

```
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"));
```

Als Nächstes fügen wir eine einfache Schaltfläche für `onCreate(...)` , die als Zahlungsanweisung dient. Dies dient einfach dazu, die Aktion auszulösen, und sollte als Initiierungsprozess für das Erstellen einer zukünftigen Zahlung für einen Benutzer (z. B. wenn sie ein Abonnement vereinbaren) platziert werden.

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

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

Unter `res > layout > activity_main.xml` fügen wir die Definition für die Schaltfläche mit der zugehörigen Aktion hinzu. Wenn Sie darauf `beginFuturePayment(...)` , wird der `beginFuturePayment(...)` , den wir in einer Minute definieren.

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

Unter `res > values > strings.xml` fügen wir dann eine Zeichenfolge für die Schaltfläche hinzu.

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

Jetzt fügen wir den Button-Handler hinzu, um den Anruf einzuleiten, um den zukünftigen Zahlungsvorgang zu starten, wenn der Benutzer auf die Schaltfläche klickt. Wir starten hier den Zahlungsservice mit dem Konfigurationsobjekt, das wir oben in diesem Beispiel eingerichtet haben.

```
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);
}
```

Wenn dieser Anruf für eine zukünftige Zahlung initiiert wird, erhalten wir einige Informationen, die an unseren Server gesendet werden müssen. Wir extrahieren diese Informationen aus der gültigen zukünftigen Zahlungsanforderung (`authCode` und `metadataId`) und führen dann die async-Anforderung an den Server aus, um die zukünftige Zahlung abzuschließen (detailliert in Schritt 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");
    }
```

```
}
```

Zuletzt definieren wir unser `onDestroy()` .

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

Android Schritt 2: Async Server Request

Den vollständigen Beispielcode für diese Anwendung (Android + Node-Server) finden Sie im PayPal Developer Github-Repository .

An diesem Punkt wurde die Schaltfläche "PayPal für zukünftige Zahlungen" angeklickt, wir haben einen Authentifizierungscode und eine Metadaten-ID aus dem PayPal-SDK. Diese Informationen müssen an unseren Server weitergeleitet werden, um den zukünftigen Zahlungsvorgang abzuschließen.

Im Hintergrundprozess unten machen wir ein paar Dinge:

- Wir haben den URI so eingerichtet, dass er für unseren Server `http://10.0.2.2:3000/fpstore` ist `/fpstore` trifft auf den `/fpstore` Endpunkt unseres Servers, der auf localhost ausgeführt wird.
- Das JSON-Objekt, durch das gesendet wird, wird dann eingerichtet, das den Authentifizierungscode und die Metadaten-ID enthält.
- Die Verbindung wird dann hergestellt. Im Falle einer erfolgreichen Anfrage (Bereich 200/201) können wir eine Antwort vom Server erwarten. Wir lesen diese Antwort und geben sie zurück.
- Schließlich haben wir eine `onPostExecute(...)` -Methode eingerichtet, um diese zurückgegebene Serverzeichenfolge zu verarbeiten. In diesem Beispiel wird es einfach protokolliert.

```
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);
    }
}
```

Android Schritt 3: Knotenserver, um Zugriffstoken zu erhalten und Zahlung zu verarbeiten

Den vollständigen Beispielcode für diese Anwendung (Android + Node-Server) finden Sie im PayPal Developer Github-Repository .

Ab Schritt 2 wurde eine asynchrone Anforderung an unseren Server am Endpunkt `/fpstore` , wobei der Authentifizierungscode und die Metadaten-ID übergeben wurden. Wir müssen diese jetzt gegen ein Token eintauschen, um die Anfrage abzuschließen und die zukünftige Zahlung zu bearbeiten.

Zuerst richten wir unsere Konfigurationsvariablen und Objekte ein.

```
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());
```

Jetzt richten wir eine Express-Route ein, die POST-Anforderungen `/fpstore` von unserem Android-Code an den Endpunkt `/fpstore` gesendet werden.

Wir machen eine Reihe von Dingen auf dieser Route:

- Wir erfassen den Authentifizierungscode und die Metadaten-ID aus dem POST-Body.
- Wir stellen dann eine Anforderung an `generateToken()` und übergeben das Code-Objekt. Bei Erfolg erhalten wir ein Token, mit dem die Zahlung erstellt werden kann.
- Als nächstes werden die Konfigurationsobjekte für die zukünftige Zahlung erstellt, die durchgeführt werden soll, und eine Anforderung an `payment.create(...)` wird `payment.create(...)` , wobei die zukünftigen Zahlungs- und Zahlungskonfigurationsobjekte übergeben werden. Dies schafft die zukünftige Zahlung.

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

Als letztes erstellen wir den Server, um Port 3000 zu überwachen.

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

Mobile Future Payments (Ende-zu-Ende-App) online lesen:
https://riptutorial.com/de/paypal/topic/4537/mobile-future-payments--ende-zu-ende-app-

Kapitel 5: Mobile PayPal / Kreditkartenzahlungen

Parameter

Parameter	Einzelheiten
Taste	Einfacher Zahlungsknopf
Konfig	PayPal-Konfigurationsobjekt, das unsere Kunden-ID (von der Anwendungserstellung) und die Umgebung enthält, die wir verwenden möchten (Sandbox oder Live)
Zahlung	PayPal-Zahlungsdetails
paymentConfig	Konfigurationsabsicht für die Zahlungsinformationen und -einstellungen
serviceConfig	Konfigurationsabsicht für die Konfigurationsparameterdaten

Bemerkungen

Beispiele für die Zahlungsabwicklung auf mobilen Geräten

Examples

Android: Akzeptieren einer Zahlung per PayPal / Kreditkarte

In diesem Lernprogramm erfahren Sie, wie Sie das PayPal Android SDK einrichten, um eine einfache Zahlung entweder über eine PayPal-Zahlung oder einen Kreditkartenkauf abzuwickeln. Am Ende dieses Beispiels sollten Sie über eine einfache Schaltfläche in einer Anwendung verfügen, die den Benutzer nach dem Anklicken an PayPal weiterleitet, um eine festgelegte Zahlung zu bestätigen, den Benutzer anschließend an die Anwendung zurückzugeben und die Zahlungsbestätigung zu protokollieren.

Der vollständige Anwendungscode für dieses Beispiel ist im PayPal Developer Github Repository verfügbar.

Lass uns anfangen.

Der erste Schritt besteht darin , das SDK zu erhalten und zu Ihrem Projekt hinzuzufügen . Wir fügen den Bezug zu unseren build.gradle-Abhängigkeiten wie folgt hinzu:

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

```
    ...
}
```

Jetzt gehen wir zu unserer MainActivity.java-Datei (oder wo immer Sie die Integration der PayPal-Schaltfläche hinzufügen möchten) und fügen ein `config` für unsere Client-ID und die Umgebung (Sandbox) hinzu, die wir verwenden werden.

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

Jetzt erstellen wir eine Schaltfläche in unserer `onCreate(...)` -Methode, die es uns ermöglicht, eine Zahlung per PayPal zu `onCreate(...)` , sobald Sie darauf klicken.

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

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

Nun müssen wir die Funktionalität für diese Schaltfläche definieren. In Ihrer res> layout> XML-Hauptdatei können Sie die folgende Definition für die Schaltfläche hinzufügen, die den Text und den onClick-Handler für die Schaltfläche mit der paypal_button-ID definiert.

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

Wenn Sie darauf `beginPayment(...)` ruft die `beginPayment(...)` Methode `beginPayment(...)` . Wir können dann den Text für die Schaltfläche unserer Datei strings.xml hinzufügen, wie folgt:

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

Wenn die Schaltfläche aktiviert ist, müssen wir jetzt mit dem Klicken auf die Schaltfläche klicken, um die Zahlungsabwicklung zu starten. Fügen Sie in der folgenden `beginPayment(...)` -Methode unterhalb unserer vorherigen `onCreate(...)` -Methode hinzu.

```
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);
}
```

Was wir hier tun, ist zuerst das Service-Intent (`serviceConfig`) einzurichten, indem wir die `config` , die wir zuvor für unsere Client-ID und die Sandbox-Umgebung definiert hatten. Wir geben dann das Zahlungsobjekt an, das wir bearbeiten möchten. In diesem Beispiel legen wir einen statischen Preis, eine Währung und eine Beschreibung fest. In Ihrer endgültigen Anwendung sollten diese Werte von dem abgerufen werden, was der Benutzer in der Anwendung zu kaufen versucht. Schließlich setzen wir die oben `paymentConfig` und fügte hinzu , sowohl in der `config` und `payment` Objekte , die wir zuvor definiert hatten, und die Aktivität beginnen.

An diesem Punkt werden dem Benutzer die PayPal-Anmeldungs- und Zahlungsbildschirme angezeigt, in denen er auswählen kann, ob Sie mit PayPal oder einer Kreditkarte bezahlen möchten (über die manuelle Eingabe oder card.io, falls die Kamera verfügbar ist). Dieser Bildschirm sieht ungefähr so aus:

Anschließend benötigen wir einen Handler, der bereit ist, wenn PayPal den Benutzer nach Bestätigung der Zahlung oder Stornierung an die Anwendung weiterleitet. Lassen Sie uns `onActivityResult(...)` für diesen Zweck überschreiben.

```
@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");
    }
}
```

Innerhalb der `onActivityResult(...)` -Methode überprüfen wir, ob der zurückgegebene `resultCode` RESULT_OK (vom Benutzer bestätigte Zahlung), RESULT_CANCELED (vom Benutzer stornierte Zahlung) oder RESULT_EXTRAS_INVALID (es gab ein Konfigurationsproblem) gibt. Im Falle einer gültigen Bestätigung erhalten wir das Objekt, das von der Zahlung zurückgegeben wird, und protokollieren es in diesem Beispiel. Was an uns zurückgegeben wird, sollte in etwa wie folgt aussehen:

```
{
    "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"
}
```

Wenn wir unter dem `response` nachsehen, können wir feststellen, dass wir einen `approved` `state` `approved` , was bedeutet, dass die Zahlung bestätigt wurde. An diesem Punkt sollte dieses Objekt an Ihren Server gesendet werden, um zu bestätigen, dass eine Zahlung tatsächlich ausgeführt wurde. Weitere Informationen zu diesen Schritten finden Sie in diesen Dokumenten .

Unser letzter Schritt ist das Aufräumen in unserem `onDestroy(...)` .

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

Das ist alles dazu. In diesem Beispiel haben wir eine einfache Schaltfläche zum Abwickeln einer Zahlung mit PayPal oder einer Kreditkarte erstellt. Von diesem Punkt aus gibt es einige weitere Schritte, um dieses Beispiel zu erweitern:

- Dynamisches `beginPayment(...)` Zahlungsinformationen basierend auf der Benutzerproduktauswahl in der Methode `beginPayment(...)` .
- Senden Sie die Zahlungsbestätigung an Ihren Server und überprüfen Sie, ob die Zahlung tatsächlich gelaufen ist.
- Behandlung der Fehler- und Stornierungsanwenderfälle innerhalb der App.

Mobile PayPal / Kreditkartenzahlungen online lesen:
https://riptutorial.com/de/paypal/topic/608/mobile-paypal---kreditkartenzahlungen

Kapitel 6: PayPal-Zahlung durchführen

Parameter

Parameter	Einzelheiten
Kunden ID	Ihre PayPal-Anwendungs-Client-ID (OAuth 2-Berechtigungsnachweis)
Links	Einfaches Referenzobjekt für alle zurückgegebenen HATEOAS-Links von PayPal
paymentId	Die ID der Zahlung wird von PayPal zurückgegeben, um die Zahlung abzuschließen
payerId	Die ID des Zahlers wurde von PayPal zurückgegeben, um die Zahlung abzuschließen
paypal	Referenz zum PayPal-Node-SDK
payReq	JSON-Objekt, das Zahlungsinformationen für die Transaktion enthält
req	Das Anforderungsobjekt von der Serveranforderung
res	Das Antwortobjekt aus der Serveranfrage
Geheimnis	Ihr PayPal-Antragsgeheimnis (OAuth 2-Berechtigungsnachweise)

Bemerkungen

In diesen Beispielen wird beschrieben, wie Sie eine Zahlung mit den SDKs von PayPal über PayPal abwickeln. Dies sind einfache Anforderungsbeispiele, die den mehrstufigen Prozess zur Ermöglichung dieser Zahlungsoption beschreiben.

Examples

Beispiel für einen Node Express Server

In diesem Beispiel richten wir eine Express-Server-Integration ein, um anzuzeigen, wie eine Zahlung mit PayPal mithilfe des PayPal Node SDK verarbeitet wird. Der Einfachheit halber verwenden wir eine statische JSON-Struktur für die Zahlungsdetails.

Es gibt drei allgemeine Schritte, die wir beim Ausarbeiten der Funktionen zur Abwicklung der PayPal-Zahlung befolgen werden:

1. Wir erstellen ein JSON-Objekt, das die Zahlung enthält, die wir über PayPal verarbeiten

möchten. Wir senden das dann an PayPal, um einen Link zu erhalten, an den der Benutzer weitergeleitet werden kann, um die Zahlung zu bestätigen.
2. Als Nächstes leiten wir den Benutzer zu PayPal, um die Zahlung zu bestätigen. Nach der Bestätigung leitet PayPal den Benutzer zurück zu unserer Anwendung.
3. Nach der Rückkehr zur App führen wir die Zahlung für den Benutzer durch.

Um es als einfache Node-App aufzuteilen, erhalten wir zunächst das PayPal-Node-SDK von NPM:

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

Als Nächstes richten wir die App-Konfiguration und die Pakete ein.

```
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
});
```

Wir benötigen vier Voraussetzungen für diese App:

1. Das HTTP-Paket für unseren Server.
2. Das PayPal-Node-SDK-Paket.
3. Das bodyParser-Paket für die Arbeit mit JSON-codierten Körpern.
4. Das Express-Framework für unseren Server.

In den nächsten Zeilen werden Variablen für die Client-ID und das Geheimnis eingerichtet, die beim Erstellen einer Anwendung abgerufen wurden. Dann richten wir `bodyParser` , um JSON-codierte Körper zuzulassen, konfigurieren dann unsere Anwendung anhand der Anwendungsdetails und geben die Umgebung an, in der wir arbeiten (live für die Produktion oder Sandbox zum Testen).

Nun erstellen wir die Route zum Erstellen einer Zahlungsanforderung mit 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');
            }
        }
    });
});
```

Als Erstes richten wir das JSON-Objekt für die Zahlungsanforderung ein, das die Informationen enthält, die PayPal zur Erstellung der Zahlung angeben müssen. Wir setzen die `intent sale` , die Redirect URLs angeben (wo PayPal die Benutzer weitergeleitet werden soll , nachdem sie bestätigen / die Zahlung stornieren), fügen Sie in einem `payment_method` von `paypal` , um zu signalisieren , dass wir eine PayPal - Zahlung leisten, dann geben Sie die Transaktionsinformationen für der Zahler zur Bestätigung.

Wir rufen dann `payment.create(...)` und übergeben unser `payReq` Objekt. Dadurch wird die erstellte Zahlungsanforderung an PayPal gesendet. Sobald dies zurückkehrt und erfolgreich ist, können wir die bereitgestellten HATEOAS- Links im Rückgabeobjekt durchlaufen , um die URL zu extrahieren, zu der der Benutzer umgeleitet werden muss. Diese Beschriftung wird unter `approval_url` .

Das Format für die HATEOAS-Links kann bei direkter Verwendung zu fragilem Referenzcode führen. Daher werden alle bereitgestellten Links durchlaufen und in ein besseres Referenzobjekt eingefügt, um die Änderungen zukünftig zu sichern. Wenn dann die `approval_url` URL in diesem Objekt gefunden wird, leiten wir den Benutzer um.

Zu diesem Zeitpunkt wird der Benutzer zu PayPal weitergeleitet, um die Zahlung zu bestätigen. Sobald dies der `return_url` , werden sie zu dem in der Funktion `createPayment(...)` angegebenen

`createPayment(...)` .

Wir müssen jetzt einen Weg angeben, um diese Rückgabe abzuwickeln, um die Zahlung abzuschließen.

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

Wenn der Benutzer an Ihre App zurückgegeben wird, werden drei `paymentId` `PayerID` , die `paymentId` , `PayerID` und das `token` . Wir müssen uns nur mit den ersten beiden beschäftigen.

Wir extrahieren die Parameter und platzieren die PayerID in einem einfachen Objekt für den Zahlungsausführungsschritt. Als Nächstes erfolgt ein Aufruf an `payment.execute(...)` , wobei diese beiden Parameter übergeben werden, um die Zahlung abzuschließen.

Sobald dieser Antrag gestellt wird, sehen wir , wenn die abgeschlossene Zahlung erfolgreich durch Prüfen , ob `payment.state` gesetzt wird `approved` . In diesem Fall können wir speichern, was wir von dem zurückgegebenen Zahlungsobjekt benötigen.

Der letzte Schritt besteht darin, unseren Server zu initialisieren und auf den Verkehr zu achten, der auf die von uns angegebenen Routen kommt.

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

Sobald der Server initialisiert ist, können Sie mit `http://localhost:3000/create` den Zahlungsvorgang starten.

PayPal-Zahlung durchführen online lesen: https://riptutorial.com/de/paypal/topic/449/paypal-zahlung-durchfuhren

Kapitel 7: Webhooks

Parameter

Parameter	Einzelheiten
App	Unsere Express-Anwendungsreferenz
BodyParser	Die Body-Parser-Paketreferenz für die Arbeit mit JSON-codierten Körpern
Kunden ID	Die Anwendungsclient-ID (OAuth 2-Berechtigungsnachweise)
http	Das http-Paket zum Ausführen des Servers
paypal	Das Referenzobjekt des PayPal-Node-SDK
Geheimnis	Das Anwendungsgeheimnis (OAuth 2-Berechtigungsnachweise)
webhookId	ID des zu ändernden Webhooks
webhookUpdate	JSON-Objekt, das die zu aktualisierenden Webhook-Details enthält

Bemerkungen

In diesen Beispielen werden Arbeitsbeispiele für die Verwendung von PayPal-Webhooks für die Ereignisüberwachung für Ihre Anwendung und Zahlungen beschrieben.

Examples

Sandbox-Webhooks mit ngrok und Express (Node) testen

In diesem Beispiel werden Webhook-Benachrichtigungen in Sandbox getestet. Mit ngrok wird ein Tunnel für unseren Node-HTTP-Listener bereitgestellt , der auf localhost auf das Internet ausgeführt wird. In diesem Beispiel werden wir Node verwenden, um Benachrichtigungs-Webhooks für Zahlungsereignisse einzurichten (z. B. eine Zahlung). Dann wird der Server so eingerichtet, dass er auf eingehende HTTP-POST-Nachrichten von den Webhook-Ereignissen wartet.

Es gibt einige Schritte, die wir hier befolgen werden, um dies zu ermöglichen:

1. Richten Sie einen einfachen Server ein, um den eingehenden POST-Verkehr von den Webhooks zu überwachen, der als Benachrichtigung von PayPal angezeigt wird, und beginnen Sie mit dem Abhören von localhost.
2. Verwenden Sie dann ngrok, um einen Tunnel von localhost zum Internet bereitzustellen, damit PayPal eine Benachrichtigung übermitteln kann.

3. Abonnieren Sie schließlich unsere Anwendung (basierend auf den bereitgestellten Anmeldeinformationen) für Webhook-Ereignisse, die wir verfolgen möchten, und stellen Sie die öffentliche ngrok-URI aus Schritt 2 bereit.

Erstellen eines Webhooks Listener

Als erstes müssen wir den Listener erstellen. Der Grund, warum wir mit dem Listener beginnen, ist, dass wir die Webhooks mit der Live-URL von ngrok versehen, wenn wir sie erstellen oder aktualisieren.

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

Unser Zuhörer ist eine einfache Route mit Express. Wir überwachen den eingehenden POST-Verkehr und spucken den POST-Body auf die Konsole aus. Wir können dies nutzen, um mit dem Zuhörer alles zu tun, was wir möchten.

Wenn wir am Ende den HTTP-Server erstellen, richten wir ihn so ein, dass er den localhost-Port 3001 überwacht. Führen Sie dieses Skript jetzt aus, um den Datenverkehr zu überwachen.

Verwenden von ngrok, um den Listener dem Internet zugänglich zu machen

Wenn der Listener auf localhost: 3001 eingerichtet ist, besteht die nächste Aufgabe darin, dieses Skript dem Internet zugänglich zu machen, sodass der Verkehr an ihn gesendet werden kann. Dies ist die Aufgabe von ngrok.

Führen Sie den folgenden Befehl in einem Terminalfenster aus:

```
ngrok http 3001
```

Dadurch wird der Prozess des Bereitstellens eines Live-Tunnels für Localhost auf Port 3001 initiiert und nach dem Ausführen die folgenden Informationen bereitgestellt:

Wie wir sehen können, lautet die Live-Adresse, mit der wir den PayPal-Webhook auf unseren laufenden Listener auf localhost richten können, `http(s)://055b3480.ngrok.io` . Das ist alles, was wir wissen müssen, um den Hörer einzurichten.

Benachrichtigungen abonnieren

Unser letzter Schritt ist das Erstellen von Webhooks für unsere Anwendung, die Benachrichtigungen zu bestimmten Ereignissen wie Zahlungen, Rückerstattungen usw. in unserer App erstellt. Diese Webhooks müssen nur einmal erstellt werden, um sie an die Anwendung zu binden. Sie müssen also nicht jedes Mal ausgeführt werden, wenn Sie sie verwenden möchten.

Zuerst richten wir die PayPal-Umgebung ein, indem wir die Anforderung für das PayPal-Node-SDK hinzufügen, unsere Client-ID / das Geheimnis, eine Anwendung zu erstellen, und dann die Umgebung für Sandbox konfigurieren.

```
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
});
```

Als Nächstes richten wir die JSON-Struktur für unsere Webhooks ein. `webhooks` enthält zwei Informationen, die `url` , an die alle Webhook-Ereignisse gesendet werden sollen, und die `event_types` , die wir abonnieren möchten.

In diesem Beispiel wird die `url` auf unsere Ngrok-Live-URL gesetzt. Bei den Ereignissen, auf die wir hören, handelt es sich um Fälle, in denen Zahlungen abgeschlossen oder abgelehnt werden.

Eine vollständige Liste möglicher Ereignisse finden Sie unter https://developer.paypal.com/docs/integration/direct/rest-webhooks-overview/#event-type-support .

Als letztes übergeben wir das `webhooks` Objekt in den Aufruf zum Erstellen der Webhooks (`notification.webhook.create` . Bei Erfolg sendet PayPal Benachrichtigungen an den angegebenen Endpunkt, der auf localhost ausgeführt wird.

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

Sobald wir eine Zahlung mit diesen Anmeldeinformationen ausgestellt haben, werden Informationen zum Zahlungsstatus an den von uns eingerichteten Endpunkt gesendet.

Ein Beispiel für den POST-Body, den PayPal als Benachrichtigung sendet, könnte wie folgt aussehen, der nach einer erfolgreichen PayPal-Zahlung gesendet wurde:

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

Aktualisieren eines Webhooks mit einer neuen URL (Knotenbeispiel)

In diesem Beispiel erfahren Sie, wie Sie eine vorhandene URL für die Weiterleitung von Webhooks aktualisieren (an die die Benachrichtigungen gepostet werden sollen). Um dies auszuführen, sollten Sie die ID von PayPal bei der Erstellung Ihrer Webhooks zurückgegeben haben.

Fügen Sie zunächst das PayPal-SDK hinzu und konfigurieren Sie die Umgebung (Sandbox unten).

```
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
});
```

Richten Sie als Nächstes die JSON-Struktur und die Webhook-Details ein. Weisen Sie die ID für Ihren `webhookId` zuerst der WebhookId zu. Als Nächstes geben Sie im `webhookUpdate` eine Ersetzungsoperation an, legen Sie den `path` auf `/url` , um eine Aktualisierung dieser Ressource anzugeben, und geben Sie die neue URL an, um sie durch den `value` unter `value` zu ersetzen.

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

Rufen Sie zuletzt `notification.webhook.replace(...)` und übergeben Sie die `webhookId` und das `webhookUpdate` .

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

Wenn alles erfolgreich ist, sollte ein Objekt, das dem folgenden ähnelt, von PayPal zurückgegeben und im Fall dieses Beispiels mit den neu aktualisierten Informationen im Terminal angezeigt werden.

```
{
    "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
}
```

Webhooks online lesen: https://riptutorial.com/de/paypal/topic/575/webhooks

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit PayPal	Community, Jonathan LeBlanc, Nathan Arthur
2	Abonnements anlegen / wiederkehrende Zahlungen	Jonathan LeBlanc
3	Kreditkartenzahlung durchführen (Knoten)	Jonathan LeBlanc
4	Mobile Future Payments (Ende-zu-Ende-App)	Jonathan LeBlanc
5	Mobile PayPal / Kreditkartenzahlungen	Jonathan LeBlanc
6	PayPal-Zahlung durchführen	Jonathan LeBlanc
7	Webhooks	Jonathan LeBlanc