



**EBook Gratuito**

# APPENDIMENTO

# PayPal

Free unaffiliated eBook created from  
**Stack Overflow contributors.**

**#paypal**

# Sommario

Di.....	1
<b>Capitolo 1: Iniziare con PayPal.....</b>	<b>2</b>
Osservazioni.....	2
Versioni.....	2
Examples.....	2
Creazione di un'applicazione e ottenimento di ID client / chiavi segrete.....	2
Impostazione degli account di test utente sandbox.....	4
<b>Capitolo 2: Creazione di abbonamenti / pagamenti ricorrenti.....</b>	<b>5</b>
Parametri.....	5
Osservazioni.....	5
Examples.....	6
Passaggio 2: creazione di una sottoscrizione per un utente che utilizza un contratto di fa.....	6
Passaggio 1: creazione di un modello di abbonamento utilizzando un piano di fatturazione (.....	9
<b>Capitolo 3: Effettuare un pagamento con carta di credito (nodo).....</b>	<b>12</b>
Parametri.....	12
Osservazioni.....	12
Examples.....	12
Esempio di nodo.....	12
Effettuare un pagamento con una carta di credito a volta (nodo).....	15
<b>Capitolo 4: Fare un pagamento con PayPal.....</b>	<b>18</b>
Parametri.....	18
Osservazioni.....	18
Examples.....	18
Esempio di server Node Express.....	18
<b>Capitolo 5: Mobile Future Payments (End-End App).....</b>	<b>22</b>
Osservazioni.....	22
Examples.....	22
Passaggio 1 di Android: configurazione, inizializzazione e gestione della risposta del ser.....	22
Passaggio 2 di Android: richiesta server asincrono.....	24
Passaggio 3 di Android: Node Server per ottenere token di accesso e pagamento di processo.....	25

<b>Capitolo 6: Pagamenti con carta di credito / carta di credito</b> .....	<b>28</b>
Parametri.....	28
Osservazioni.....	28
Examples.....	28
Android: accettazione di un pagamento con carta di credito / PayPal.....	28
<b>Capitolo 7: Webhooks</b> .....	<b>32</b>
Parametri.....	32
Osservazioni.....	32
Examples.....	32
Test di Sandbox Webhooks con ngrok ed Express (Node).....	32
Aggiornamento di un webhook con un nuovo URL (esempio di nodo).....	36
<b>Titoli di coda</b> .....	<b>38</b>

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

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

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

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

# Capitolo 1: Iniziare con PayPal

## Osservazioni

Queste guide porteranno l'utente attraverso le procedure di configurazione dell'account per applicazioni, account, ecc. Conterrà tutto ciò che è necessario per lavorare con le API di PayPal.

## Versioni

Versione	Data di rilascio
1.0.0	2016/04/11

## Examples

### Creazione di un'applicazione e ottenimento di ID client / chiavi segrete

Per iniziare a costruire con le API PayPal, devi creare un'applicazione per ottenere un ID client e un segreto.

Vai a <https://developer.paypal.com/developer/applications/>, accedi e fai clic su "Crea app", come mostrato di seguito:

### REST API apps

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

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

Create App



#### App name

My test app

My test app 1

MyLiveApp

Quindi, inserire il nome di un'applicazione, selezionare l'account di test sandbox che si desidera utilizzare (se si tratta di un nuovo account, lasciare il valore predefinito) e fare clic su "Crea app".

## Application Details

### App Name

My Sandbox Test Application

### Sandbox developer account

test232213@testing.com (US)

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

Create App

Una volta che l'applicazione è stata creata, ti verrà fornita la sandbox e l'ID cliente e segreto in tempo reale, che sarà simile al seguente:

## SANDBOX API CREDENTIALS

**Sandbox account** test232213@testing.com

**Client ID** ATwkfTgxN3

**Secret** [Hide](#)

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

Created	Secret
Apr 11, 2016	EJqSRO4Gj5s

Generate New Secret

Queste credenziali sono quelle che utilizzerai quando effettui richieste alle API di PayPal per autenticare la tua richiesta ed effettuare richieste.

## Impostazione degli account di test utente sandbox

Quando si verifica l'integrazione di PayPal su sandbox, è necessario impostare gli account utente di sandbox da utilizzare per passare attraverso il flusso di pagamento.

Vai a <https://developer.paypal.com/developer/accounts/>, accedi usando il tuo account PayPal e clicca su "Crea Account", come di seguito:

# Sandbox Test Accounts

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

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

Total records: 15

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

Inserisci i dettagli dell'account per il nuovo utente di test, inclusa un'e-mail univoca, informazioni sull'account, metodo di pagamento, saldo, ecc. E fai clic su "Crea account" nella parte inferiore della pagina una volta eseguita. Questo creerà il nuovo account che inizierai a utilizzare.

Per visualizzare i dettagli dell'account per questo nuovo utente, espandere la voce nella pagina degli account e fare clic su "Profilo".

<input type="checkbox"/>	▼ testmctest@test.com	PERSONAL
	<a href="#">Profile</a>   <a href="#">Notifications</a>	

Una volta caricate le informazioni sul profilo, facendo clic sulla scheda "Finanziamenti" verranno fornite le informazioni di pagamento per quell'account, compresi i dati della carta di credito che potrebbero essere utilizzati per l'elaborazione diretta della carta di credito contro sandbox.

NOTA: quando si utilizzano gli endpoint API sandbox, è necessario utilizzare l'account di prova sandbox per accedere e pagare i beni di prova, poiché le informazioni sull'account live non funzioneranno.

Leggi [Iniziare con PayPal online](https://riptutorial.com/it/paypal/topic/406/iniziare-con-paypal): <https://riptutorial.com/it/paypal/topic/406/iniziare-con-paypal>

# Capitolo 2: Creazione di abbonamenti / pagamenti ricorrenti

## Parametri

Parametro	Dettagli
billingAgreementAttributes	Oggetto di configurazione per creare il contratto di fatturazione
billingPlan	ID piano di fatturazione dalla stringa di query
billingPlanAttribs	Oggetto di configurazione per creare il piano di fatturazione
billingPlanUpdateAttributes	Oggetto di configurazione per la modifica di un piano di fatturazione in uno stato attivo
Identificativo cliente	L'ID del client dell'applicazione (chiavi OAuth)
http	Riferimento al pacchetto http per configurare il nostro server semplice
isodate	Data ISO per l'impostazione della data di inizio dell'abbonamento
link	Oggetto link HATEOAS per l'estrazione dell'URL di reindirizzamento su PayPal
params	Parametri stringa di query
paypal	Riferimento all'SDK di PayPal
segreto	La tua domanda segreta (chiavi OAuth)
gettone	Il token di approvazione del contratto di fatturazione fornito dopo che PayPal ha effettuato il reindirizzamento per l'esecuzione del contratto di fatturazione

## Osservazioni

Questi esempi passano attraverso il processo di creazione di un sistema di pagamento in abbonamento / ricorrente utilizzando PayPal.

Il processo per creare un abbonamento è:

1. Crea un piano di fatturazione. Questo è un modello riutilizzabile che delinea i dettagli

dell'abbonamento.

2. Attiva il piano di fatturazione.
3. Quando si desidera creare un abbonamento per un utente, si crea un contratto di fatturazione utilizzando l'ID del piano di fatturazione a cui devono essere abbonati.
4. Una volta creato, reindirizza l'utente a PayPal per confermare l'abbonamento. Una volta confermato, l'utente viene reindirizzato al sito web del commerciante.
5. Infine, si esegue il contratto di fatturazione per iniziare l'abbonamento.

## Examples

### Passaggio 2: creazione di una sottoscrizione per un utente che utilizza un contratto di fatturazione (esempio di nodo)

Il secondo passaggio per creare un abbonamento per un utente è creare ed eseguire un contratto di fatturazione, in base a un piano di fatturazione attivato esistente. Questo esempio presuppone che tu abbia già eseguito e attivato un piano di fatturazione nell'esempio precedente e che un ID per tale piano di fatturazione faccia riferimento nell'esempio.

Quando imposti un contratto di fatturazione per creare un abbonamento per un utente, seguirai 3 passaggi, che potrebbero ricordare l'elaborazione di un pagamento PayPal:

1. Si crea un contratto di fatturazione, facendo riferimento a un piano di fatturazione sottostante tramite l'ID.
2. Una volta creato, reindirizza l'utente a PayPal (se paga tramite PayPal) per confermare l'abbonamento. Una volta confermato, PayPal reindirizza l'utente al tuo sito utilizzando il reindirizzamento fornito nel piano di fatturazione sottostante.
3. Esegui quindi il contratto di fatturazione utilizzando un token restituito tramite il reindirizzamento PayPal.

Questo esempio sta configurando un server HTTP basato su Express per mostrare il processo del contratto di fatturazione.

Per iniziare l'esempio, dobbiamo prima impostare la nostra configurazione. Aggiungiamo quattro requisiti, l'SDK PayPal, `body-parser` per la gestione dei corpi codificati JSON, `http` per la nostra semplice integrazione con i server e `express` per il framework Express. Quindi definiamo il nostro ID client e segreto dalla creazione di un'applicazione, configuriamo l'SDK per la sandbox, quindi configuriamo `bodyParser` per la gestione dei corpi 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());
```

Il nostro primo passo nel contratto di fatturazione è creare un percorso per gestire la creazione di un contratto di fatturazione e reindirizzare l'utente su PayPal per confermare tale iscrizione. Supponiamo che un ID del piano di fatturazione venga passato come parametro della stringa di query, ad esempio caricando il seguente URL con un ID del piano dell'esempio precedente:

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

Ora dobbiamo usare queste informazioni per creare il contratto di fatturazione.

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

```

Iniziamo estraendo l'ID del piano di fatturazione dalla stringa di query e creando la data in cui il piano dovrebbe iniziare.

La successiva definizione dell'oggetto, `billingAgreementAttributes`, consiste in informazioni per la sottoscrizione. Contiene informazioni leggibili sul piano, un riferimento all'ID del piano di fatturazione, il metodo di pagamento e i dettagli di spedizione (se necessario per l'abbonamento).

Successivamente, viene effettuata una chiamata a `billingAgreement.create(...)`, passando nell'oggetto `billingAgreementAttributes` appena creato. Se tutto è andato a buon fine, dovremmo ricevere da noi un oggetto di contratto di fatturazione contenente i dettagli sul nostro abbonamento appena creato. Tale oggetto contiene anche una serie di collegamenti HATEOAS che ci forniscono i prossimi passi che possono essere intrapresi su questo accordo appena creato. Quello che ci interessa qui è etichettato come `approval_url`.

Passiamo in rassegna tutti i collegamenti forniti per metterli in un oggetto facilmente referenziato. Se `approval_url` è uno di quei link, reindirizziamo l'utente a quel link, che è PayPal.

A questo punto l'utente conferma l'iscrizione su PayPal e viene reindirizzato all'URL fornito nel piano di fatturazione sottostante. Insieme a tale URL, PayPal passerà anche un token lungo la stringa di query. Quel token è ciò che useremo per eseguire (o avviare) l'abbonamento.

Impostiamo questa funzionalità nel seguente percorso.

```

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

```

Estraiamo il token dalla stringa di query, quindi effettuiamo una chiamata a `billingAgreement.execute`, passando per quel token. Se tutto ha successo, ora abbiamo un abbonamento valido per l'utente. L'oggetto di reso contiene informazioni sul contratto di fatturazione attivo.

Infine, impostiamo il nostro server HTTP per ascoltare il traffico verso i nostri percorsi.

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

## Passaggio 1: creazione di un modello di abbonamento utilizzando un piano di fatturazione (esempio di nodo)

Quando si crea un abbonamento per un utente, è prima necessario creare e attivare un piano di fatturazione a cui un utente è quindi iscritto utilizzando un contratto di fatturazione. Il processo completo per la creazione di un abbonamento è dettagliato nelle osservazioni di questo argomento.

All'interno di questo esempio, utilizzeremo l' [SDK del nodo PayPal](#) . Puoi ottenerlo da NPM usando il seguente comando:

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

All'interno del nostro file .js, per prima cosa configuriamo la nostra configurazione dell'SDK, che include l'aggiunta di un requisito per l'SDK, la definizione dell'ID client e del segreto dalla [creazione della nostra applicazione](#) e quindi la configurazione dell'SDK per l'ambiente 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
});
```

Quindi, dobbiamo impostare due oggetti JSON. L'oggetto `billingPlanAttribs` contiene le informazioni e la ripartizione dei pagamenti per il piano di fatturazione a cui possiamo iscrivere gli utenti e l'oggetto di `billingPlanUpdateAttributes` contiene i valori per impostare il piano di fatturazione su uno stato attivo, consentendone l'utilizzo.

```
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"
  }
}
]];

```

`billingPlanAttribs` , ci sono alcune informazioni rilevanti:

- **nome / descrizione / tipo** : informazioni visive di base per descrivere il piano e il tipo di piano.
- **payment\_definitions** : informazioni su come il piano dovrebbe funzionare e essere fatturato. Maggiori dettagli sui campi [qui](#) .
- **merchant\_preferences** : strutture tariffarie aggiuntive, URL di reindirizzamento e impostazioni per il piano di abbonamento. Maggiori dettagli sui campi [qui](#) .

Con questi oggetti, ora possiamo creare e attivare il piano di fatturazione.

```

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);
      }
    });
  }
});

```

Chiamiamo `billingPlan.create(...)` , passando l'oggetto `billingPlanAttribs` che abbiamo appena creato. Se ciò ha esito positivo, l'oggetto di ritorno conterrà informazioni sul piano di fatturazione. Per fare un esempio, è sufficiente utilizzare l'ID del piano di fatturazione per attivare il piano per l'uso.

Successivamente, chiamiamo `billingPlan.update(...)` , passando l'ID del piano di fatturazione e

l'oggetto di `billingPlanUpdateAttributes` che abbiamo creato in precedenza. Se ciò è successo, il nostro piano di fatturazione è ora attivo e pronto per l'uso.

Per creare un abbonamento per un utente (o più utenti) su questo piano, dovremo fare riferimento `billingPlan.id` piano di fatturazione ( `billingPlan.id` sopra), quindi memorizzarlo in un luogo a cui è possibile fare riferimento facilmente.

Nella seconda fase della sottoscrizione, dobbiamo creare un contratto di fatturazione basato sul piano appena creato ed eseguirlo per iniziare l'elaborazione delle sottoscrizioni per un utente.

Leggi [Creazione di abbonamenti / pagamenti ricorrenti online](#):

<https://riptutorial.com/it/paypal/topic/467/creazione-di-abbonamenti---pagamenti-ricorrenti>

# Capitolo 3: Effettuare un pagamento con carta di credito (nodo)

## Parametri

Parametro	Dettagli
card_data	Oggetto JSON contenente le informazioni di pagamento per la transazione
dettagli della carta di credito	Oggetto JSON contenente dati della carta di credito che viene inviato a PayPal per essere classificato
Identificativo cliente	Il tuo ID client dell'applicazione PayPal (credenziali OAuth 2)
paypal	Riferimento dell'SDK del nodo PayPal
segreto	La tua applicazione PayPal segreta (credenziali OAuth 2)
uuid	Riferimento al pacchetto node-uuid

## Osservazioni

Questo esempio porta l'utente a riaccreditare una semplice transazione con carta di credito utilizzando gli SDK PayPal.

## Examples

### Esempio di nodo

Inizia installando il modulo Node PayPal da NPM

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

Nel file dell'applicazione, aggiungere le informazioni di configurazione per l'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
```

```
});
```

Aggiungiamo il requisito per l'SDK, quindi impostiamo le variabili per l'ID client e il segreto che sono stati ottenuti durante la [creazione di un'applicazione](#) . Configuriamo quindi la nostra applicazione utilizzando questi dettagli e specificando l'ambiente in cui operiamo (live o sandbox).

Quindi, impostiamo l'oggetto JSON che contiene le informazioni di pagamento per il pagatore.

```
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."
  }
];
```

Aggiungere un `intent` di `sale` , ed una `payment_method` di `credit_card` . Successivamente, aggiungere i dettagli della carta e dell'indirizzo per la carta di credito sotto `funding_instruments` e l'importo da addebitare nelle `transactions` . Qui possono essere posizionati più oggetti di transazione.

Infine, facciamo una richiesta a `payment.create(...)` , passando il nostro oggetto `card_data` , al fine di elaborare il pagamento.

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

Se la transazione ha avuto successo, dovremmo vedere un oggetto risposta simile al seguente:

```

{
  "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-9BS08892W3794812YK4HKFY",
  "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-9BS08892W3794812YK4HKFY",
    "rel": "self",
    "method": "GET"
  }
],
"httpStatusCode": 201
}

```

In questo oggetto di reso, lo `state` di `approved` ci dice che la transazione ha avuto successo. Sotto l'oggetto `links` sono una serie di link [HATEOAS](#) che forniscono potenziali passi successivi che possono essere intrapresi sull'azione appena eseguita. In questo caso, siamo in grado di recuperare informazioni sul pagamento effettuando una richiesta GET al `self` endpoint fornito.

## Effettuare un pagamento con una carta di credito a volta (nodo)

In questo esempio, vedremo come memorizzare una carta di credito utilizzando il vault di PayPal, quindi fare riferimento a quella carta di credito memorizzata per elaborare una transazione con carta di credito per un utente.

Il motivo per cui vorremmo utilizzare il vault è che non dobbiamo archiviare informazioni sensibili sulle carte di credito sui nostri server. Facciamo semplicemente riferimento al metodo di pagamento tramite un ID del vault fornito, il che significa che non dobbiamo occuparci di molte norme di conformità PCI con l'archiviazione delle carte di credito da noi.

Come con i campioni precedenti, iniziamo con l'impostazione del nostro ambiente.

```

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
});

```

L'unica differenza rispetto ai campioni precedenti è che stiamo richiedendo un nuovo pacchetto, `node-uuid`, che deve essere usato per generare UUID univoci per i pagatori quando si memorizza la carta. Puoi installare quel pacchetto tramite:

```
npm install node-uuid
```

Successivamente, definiamo l'oggetto JSON della carta di credito che verrà inviato al vault di PayPal per l'archiviazione. Contiene informazioni dalla carta e un ID di pagamento unico che generiamo usando `node-uuid`. È necessario memorizzare questo `payer_id` univoco nel proprio database poiché verrà utilizzato durante la creazione di un pagamento con la carta `payer_id`.

```

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

```

Infine, dobbiamo conservare la carta di credito ed elaborare il pagamento utilizzando quella carta. Per classificare una carta di credito, chiamiamo `credit_card.create(...)`, passando l'oggetto `credit_card_details` che abbiamo appena creato. Se tutto va bene, dovremmo avere un oggetto restituito con dettagli sulla carta coperta. Per il pagamento con quella carta, abbiamo solo bisogno di due informazioni: il `payer_id` che abbiamo già memorizzato e l'ID del vault, che dovrebbe anche essere memorizzato come riferimento nel nostro database.

```

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));
    }
});
}
});

```

Nella sezione successiva al vaulting riuscito della carta di credito, stiamo semplicemente definendo i dettagli della carta ed elaborando il pagamento, come abbiamo fatto con il precedente esempio di elaborazione della carta di credito. La differenza principale nella struttura del `card_data` oggetto è la `funding_instruments` sezione, che definiamo sotto `payer`. Invece di definire le informazioni sulla carta di credito, utilizziamo invece il seguente oggetto che contiene il riferimento dell'ID del vault e l'ID del beneficiario:

```

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

```

È così che utilizziamo una carta blindata per elaborare un pagamento.

**Leggi Effettuare un pagamento con carta di credito (nodo) online:**

<https://riptutorial.com/it/paypal/topic/444/effettuare-un-pagamento-con-carta-di-credito--nodo->

# Capitolo 4: Fare un pagamento con PayPal

## Parametri

Parametro	Dettagli
Identificativo cliente	Il tuo ID client dell'applicazione PayPal (credenziali OAuth 2)
link	Oggetto di riferimento semplice per tutti i link HATEOAS di ritorno da PayPal
IDPagamento	L'ID del pagamento restituito da PayPal per completare il pagamento
PayerID	L'ID del pagatore restituito da PayPal per completare il pagamento
paypal	Riferimento dell'SDK del nodo PayPal
payReq	Oggetto JSON contenente le informazioni di pagamento per la transazione
req	L'oggetto richiesta dalla richiesta del server
res	L'oggetto risposta dalla richiesta del server
segreto	La tua applicazione PayPal segreta (credenziali OAuth 2)

## Osservazioni

Questi esempi illustrano come elaborare un pagamento tramite PayPal, utilizzando gli SDK PayPal. Questi sono esempi di richieste semplici che delineano il processo multi-step per consentire questa opzione di pagamento.

## Examples

### Esempio di server Node Express

In questo esempio, configureremo un'integrazione del server Express per visualizzare come elaborare un pagamento con PayPal, utilizzando l'SDK del nodo PayPal. Utilizzeremo una struttura JSON statica per i dettagli di pagamento per brevità.

Ci sono tre passaggi generali che seguiremo quando costruiremo le funzioni per gestire il pagamento PayPal:

1. Creiamo un oggetto JSON contenente il pagamento che intendiamo elaborare tramite

PayPal. Lo inviamo quindi a PayPal per ottenere un link per reindirizzare l'utente al fine di confermare il pagamento.

2. Successivamente, reindirizziamo l'utente a PayPal per confermare il pagamento. Una volta confermato, PayPal reindirizza l'utente alla nostra applicazione.
3. Una volta tornati all'app, completiamo il pagamento per conto dell'utente.

Interrompendo il tutto come una semplice app Node, iniziamo ottenendo l'SDK del Nodo PayPal da NPM:

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

Successivamente, impostiamo la configurazione dell'app e i pacchetti.

```
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
});
```

Richiediamo quattro requisiti per questa app:

1. Il pacchetto HTTP per il nostro server.
2. Il pacchetto SDK del nodo PayPal.
3. Il pacchetto bodyParser per lavorare con i corpi codificati JSON.
4. Il framework Express per il nostro server.

Le righe successive impostano le variabili per l'ID client e il segreto che sono stati ottenuti durante la [creazione di un'applicazione](#). Quindi impostiamo `bodyParser` per consentire i corpi codificati JSON, quindi configura la nostra applicazione utilizzando i dettagli dell'applicazione e specifica l'ambiente in cui stiamo lavorando (live per la produzione o sandbox per i test).

Ora creiamo il percorso per la creazione di una richiesta di pagamento 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');
    }
  }
});
});
});

```

La prima cosa che facciamo è impostare l'oggetto JSON della richiesta di pagamento, che contiene le informazioni che dobbiamo fornire a PayPal per creare il pagamento. Definiamo l'`intent` di `sale`, specificare gli URL di reindirizzamento (dove PayPal deve inoltrare l'utente a dopo aver confermato / annullato il pagamento), aggiungere un `payment_method` di pagamento di `paypal` per segnalare che effettueremo un pagamento PayPal, quindi specificare le informazioni di transazione per il pagatore per confermare.

Chiamiamo quindi `payment.create(...)`, passando nel nostro oggetto `payReq`. Questo invierà la richiesta di pagamento di creazione a PayPal. Una volta che ritorna, e ha successo, possiamo scorrere i collegamenti **HATEOAS** forniti nell'oggetto `return` per estrarre l'URL a cui dobbiamo reindirizzare l'utente, che è etichettato sotto `approval_url`.

Il formato per i link HATEOAS può causare un codice di riferimento fragile se utilizzato direttamente, quindi eseguiamo un ciclo di tutti i collegamenti forniti e li inseriamo in un oggetto di riferimento migliore per la prova futura contro le modifiche. Se in questo oggetto viene quindi trovato `approval_url`, reindirizziamo l'utente.

A questo punto l'utente viene reindirizzato su PayPal per confermare il pagamento. Una volta che lo fanno, vengono reindirizzati al `return_url` che abbiamo specificato nella funzione `createPayment(...)`.

Ora dobbiamo fornire un percorso per gestire tale reso, al fine di completare il pagamento.

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

Quando l'utente viene restituito alla tua app, ci saranno tre parametri della stringa di query che verranno inviati insieme, il `paymentId`, il `PayerID` e il `token`. Dobbiamo solo occuparci dei primi due.

Estraiamo i parametri e posizioniamo il `PayerID` in un oggetto semplice per la necessità del passaggio di esecuzione del pagamento. Successivamente, viene effettuata una chiamata a `payment.execute(...)`, passando questi due parametri, per completare il pagamento.

Una volta effettuata la richiesta, vediamo se il pagamento è stato completato correttamente controllando se `payment.state` è impostato su `approved`. In tal caso, possiamo memorizzare ciò di cui abbiamo bisogno dall'oggetto di pagamento che viene restituito.

Il nostro ultimo passo è inizializzare il nostro server e ascoltare il traffico che arriva ai percorsi che abbiamo specificato.

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

Una volta che il server è stato inizializzato, andare a `http://localhost:3000/create` inizializza la procedura di pagamento.

Leggi [Fare un pagamento con PayPal online](https://riptutorial.com/it/paypal/topic/449/fare-un-pagamento-con-paypal): <https://riptutorial.com/it/paypal/topic/449/fare-un-pagamento-con-paypal>

---

# Capitolo 5: Mobile Future Payments (End-End App)

## Osservazioni

Questo esempio mostra un pratico esempio end-to-end di creazione di un [pagamento futuro PayPal](#) da un dispositivo Android, utilizzando un server Node.

## Examples

### Passaggio 1 di Android: configurazione, inizializzazione e gestione della risposta del server

Il codice di esempio completo per questa applicazione (server Android + Node) è disponibile nel [repository Github per sviluppatori PayPal](#).

La prima fase della creazione della parte Android della nostra applicazione consiste nell'impostare un layout di base e gestire le risposte provenienti dal server che configureremo nel nodo.

Inizia creando un nuovo oggetto `PayPalConfiguration` per ospitare le informazioni dell'applicazione.

```
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"));
```

Successivamente, aggiungiamo un semplice pulsante a `onCreate(...)` per fungere da nostro avvio di pagamento. Questo è semplicemente per innescare l'azione, e dovrebbe essere inserito come processo di avvio per la creazione di un pagamento futuro per un utente (ad esempio quando concordano su un abbonamento).

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

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

Sotto `res > layout > activity_main.xml` aggiungiamo la definizione per il pulsante con la sua azione associata, quando viene cliccato chiama `beginFuturePayment(...)`, che definiremo tra 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" />
```

Sotto `res > values > strings.xml` aggiungiamo quindi una stringa di riferimento per il pulsante.

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

Ora aggiungiamo il gestore di pulsanti, per avviare la chiamata per iniziare il processo di pagamento futuro quando l'utente fa clic sul pulsante. Quello che stiamo facendo qui è avviare il servizio di pagamento con l'oggetto di configurazione che abbiamo impostato all'inizio di questo esempio.

```
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);
}
```

Quando viene avviata la chiamata per effettuare un pagamento futuro, verranno fornite alcune informazioni che dovranno essere inviate al nostro server. `authCode` queste informazioni dalla richiesta di pagamento valida in futuro ( `authCode` e `metadataId` ), quindi `authCode` la richiesta async al server per completare il pagamento futuro (dettagliato nel passaggio 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");
    }
}
```

```
}
```

Infine, definiamo il nostro `onDestroy()` .

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

## Passaggio 2 di Android: richiesta server asincrono

Il codice di esempio completo per questa applicazione (server Android + Node) è disponibile nel [repository Github per sviluppatori PayPal](#) .

A questo punto è stato fatto clic sul pulsante di pagamento futuro di PayPal, abbiamo un codice di autenticazione e ID di metadati dall'SDK di PayPal e dobbiamo trasferirli sul nostro server per completare la procedura di pagamento futura.

Nel processo in background di seguito, stiamo facendo alcune cose:

- `/fpstore` l'URI che per il nostro server sia `http://10.0.2.2:3000/fpstore` , che sta colpendo l'endpoint `/fpstore` del nostro server in esecuzione su localhost.
- Viene quindi impostato l'oggetto JSON che verrà inviato, che contiene il codice di autenticazione e l'ID dei metadati.
- La connessione viene quindi effettuata. Nel caso di una richiesta riuscita (intervallo 200/201), possiamo aspettarci una risposta dal server. Leggiamo questa risposta e poi la restituiamo.
- Infine, abbiamo un `onPostExecute(...)` impostato per gestire la stringa del server restituito. Nel caso di questo esempio, è semplicemente registrato.

```
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);
}
}

```

## Passaggio 3 di Android: Node Server per ottenere token di accesso e pagamento di processo

Il codice di esempio completo per questa applicazione (server Android + Node) è disponibile nel

[repository Github per sviluppatori PayPal](#) .

Dal passaggio 2, è stata `/fpstore` una richiesta asincrona al nostro server `/fpstore` , passando il codice di autenticazione e l'ID dei metadati. Ora dobbiamo scambiarli con un token per completare la richiesta ed elaborare il pagamento futuro.

Per prima cosa impostiamo le nostre variabili di configurazione e l'oggetto.

```
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());
```

Ora impostiamo una rotta Express che ascolterà le richieste POST inviate `/fpstore` dal nostro codice Android.

Stiamo facendo una serie di cose in questo percorso:

- Acquisiamo il codice di autenticazione e l'ID dei metadati dal corpo del POST.
- Quindi facciamo una richiesta a `generateToken()` , passando attraverso l'oggetto codice. In caso di successo, otteniamo un token che può essere utilizzato per creare il pagamento.
- Successivamente, gli oggetti config vengono creati per il pagamento futuro che deve essere fatto e viene `payment.create(...)` una richiesta a `payment.create(...)` , passando lungo i futuri oggetti di pagamento e di configurazione del pagamento. Questo crea il pagamento 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": {
```



# Capitolo 6: Pagamenti con carta di credito / carta di credito

## Parametri

Parametro	Dettagli
pulsante	Pulsante di pagamento semplice
config	Oggetto di configurazione PayPal che ospita il nostro ID cliente (dalla creazione dell'applicazione) e l'ambiente che vogliamo utilizzare (sandbox o live)
pagamento	Dettagli di pagamento PayPal
paymentConfig	Intento di configurazione per le informazioni e le impostazioni di pagamento
ServiceConfig	Intento di configurazione per i dati dei parametri di configurazione

## Osservazioni

Esempi relativi all'elaborazione dei pagamenti su dispositivi mobili

## Examples

### Android: accettazione di un pagamento con carta di credito / PayPal

In questo tutorial impareremo come configurare l' [SDK Android PayPal](#) per elaborare un semplice pagamento tramite un pagamento PayPal o un acquisto con carta di credito. Alla fine di questo esempio, si dovrebbe avere un semplice pulsante in un'applicazione che, quando si fa clic, inoltrerà l'utente a PayPal per confermare un pagamento impostato, quindi restituirà l'utente all'applicazione e registrerà la conferma del pagamento.

Il codice completo dell'applicazione per questo esempio è disponibile nel [repository Github per sviluppatori PayPal](#) .

Iniziamo.

Il primo passo è [ottenere e aggiungere l'SDK al tuo progetto](#) . Aggiungiamo il riferimento alle nostre dipendenze build.gradle in questo modo:

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

```
}
```

Ora ci dirigiamo verso il nostro file `MainActivity.java` (o ovunque desideri aggiungere l'integrazione del pulsante PayPal) e aggiungiamo un oggetto di `config` per il nostro ID client e l'ambiente (sandbox) che useremo.

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

Ora creeremo un pulsante nel nostro `onCreate(...)`, che ci consentirà di elaborare un pagamento tramite PayPal una volta cliccato.

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

Ora dobbiamo definire la funzionalità per quel pulsante. Nel file di ricerca > layout > file XML principale è possibile aggiungere la seguente definizione per il pulsante, che definirà il testo e il gestore `onClick` per il pulsante con l'ID `paypal_button`.

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

Quando si fa clic, il pulsante chiamerà il `beginPayment(...)`. Possiamo quindi aggiungere il testo per il pulsante al nostro file `strings.xml`, in questo modo:

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

Con il pulsante in atto, ora dobbiamo gestire il clic del pulsante per iniziare l'elaborazione dei pagamenti. Aggiungiamo il seguente `beginPayment(...)` sotto il nostro precedente metodo `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);  
}
```

```
}
```

Quello che stiamo facendo qui è la prima impostazione dell'intento di servizio ( `serviceConfig` ), utilizzando la `config` che avevamo precedentemente definito per il nostro ID client e l'ambiente sandbox. Specifichiamo quindi l'oggetto di pagamento che vogliamo elaborare. Per il gusto di questo esempio, stiamo impostando un prezzo, una valuta e una descrizione statici. Nella tua applicazione finale, questi valori dovrebbero essere ottenuti da ciò che l'utente sta cercando di acquistare nell'applicazione. Infine, impostiamo il `paymentConfig` , aggiungendo sia gli oggetti `config` e `payment` che avevamo precedentemente definito, sia `paymentConfig` l'attività.

A questo punto l'utente verrà presentato con le schermate di accesso e pagamento PayPal, consentendo loro di scegliere se pagare con PayPal o con carta di credito (tramite inserimento manuale o card.io se la fotocamera è disponibile). Quella schermata sarà simile a questa:

Una volta terminato, è necessario disporre di un gestore pronto per quando PayPal inoltrerà l'utente indietro all'applicazione dopo la conferma del pagamento o della cancellazione.

`onActivityResult(...)` per questo scopo.

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

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

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

All'interno del `onActivityResult(...)` , stiamo verificando se il `resultCode` che viene restituito è `RESULT_OK` (pagamento confermato dall'utente), `RESULT_CANCELED` (pagamento annullato dall'utente) o `RESULT_EXTRAS_INVALID` (c'era un problema di configurazione). Nel caso di una conferma valida, otteniamo l'oggetto che viene restituito dal pagamento e, in questo esempio, registrarlo. Quello che ci verrà restituito dovrebbe apparire come il seguente:

```
{
  "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"
}
```

Se guardiamo sotto l'oggetto `response`, possiamo vedere che abbiamo uno `state` di `approved`, il che significa che il pagamento è stato confermato. A questo punto, quell'oggetto dovrebbe essere inviato al tuo server per confermare che il pagamento è stato effettivamente effettuato. Per ulteriori informazioni su questi passaggi, [consultare questi documenti](#).

Il nostro ultimo passo è ripulire il nostro `onDestroy(...)`.

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

Questo è tutto ciò che c'è da fare. In questo esempio abbiamo creato un semplice pulsante per elaborare un pagamento con PayPal o una carta di credito. Da questo punto, ci sono alcuni prossimi passi per espandere questo esempio:

- `beginPayment(...)` dinamicamente le informazioni di pagamento in base alla selezione del prodotto dell'utente nel metodo `beginPayment(...)`.
- Inviando la conferma di pagamento al tuo server e verificando che il pagamento sia effettivamente andato a buon fine.
- Gestione dei casi di errore e cancellazione dell'utente all'interno dell'app.

**Leggi Pagamenti con carta di credito / carta di credito online:**

<https://riptutorial.com/it/paypal/topic/608/pagamenti-con-carta-di-credito---carta-di-credito>

# Capitolo 7: Webhooks

## Parametri

Parametro	Dettagli
App	Il nostro riferimento per le applicazioni Express
bodyParser	Il riferimento del pacchetto body-parser per lavorare con i corpi codificati JSON
Identificativo cliente	L'ID del client dell'applicazione (credenziali OAuth 2)
http	Il pacchetto http per l'esecuzione del server
paypal	L'oggetto di riferimento SDK del nodo PayPal
segreto	Il segreto dell'applicazione (credenziali OAuth 2)
webhookId	ID del webhook da modificare
webhookUpdate	Oggetto JSON contenente i dettagli del webhook da aggiornare

## Osservazioni

Questi esempi illustrano esempi di come utilizzare i webhook di PayPal per fornire il monitoraggio degli eventi per l'applicazione e i pagamenti.

## Examples

### Test di Sandbox Webhooks con ngrok ed Express (Node)

In questo esempio esamineremo le notifiche di webhook in sandbox, utilizzando [ngrok](#) per fornire un tunnel per il listener HTTP Node, in esecuzione su localhost, a Internet. Per questo esempio, useremo il Nodo per configurare i webhook di notifica per gli eventi di pagamento (come ad esempio un pagamento effettuato), quindi configurare il server per ascoltare i messaggi POST HTTP in arrivo dagli eventi webhook.

Ci sono alcuni passi che seguiremo qui per far sì che ciò accada:

1. Imposta un semplice server per ascoltare il traffico POST in entrata dai webhooks, che sarà la notifica da PayPal, e iniziare ad ascoltare su localhost.
2. Quindi utilizzare ngrok per fornire un tunnel da localhost a Internet in modo che PayPal possa inviare notifiche tramite.

3. Infine, iscriviti alla nostra applicazione (in base alle credenziali fornite) agli eventi webhook che vogliamo monitorare, fornendo l'URI ngrok pubblico dal passaggio 2.

## Creare un listener Webhooks

La prima cosa che dobbiamo fare è creare l'ascoltatore. Il motivo per cui iniziamo con l'ascoltatore è perché abbiamo bisogno dell'URL live di ngrok da fornire ai webhook quando li creiamo o li aggiorniamo.

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

Il nostro ascoltatore è un percorso semplice che utilizza Express. Ascoltiamo qualsiasi traffico POST in entrata, quindi sputiamo il corpo del POST alla console. Possiamo usarlo per fare tutto ciò che vorremmo con l'ascoltatore quando entrerà in gioco.

Quando creiamo il server HTTP alla fine, lo impostiamo per ascoltare sulla porta localhost 3001. Esegui ora lo script per iniziare ad ascoltare il traffico.

## Usando ngrok per esporre l'ascoltatore a Internet

Con il listener impostato su localhost: 3001, il nostro prossimo lavoro è quello di esporre questo script a Internet, in modo che possa essere inviato al traffico, che è il lavoro di ngrok.

Esegui il seguente comando da una finestra di terminale:

```
ngrok http 3001
```

Ciò avvierà il processo di fornitura di un tunnel live per localhost sulla porta 3001 e fornirà le seguenti informazioni una volta eseguito:

## ngrok by @inconshreveable

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

Connections        ttl      opn
                   0        0
```

Come possiamo vedere, l'indirizzo live che possiamo utilizzare per indirizzare il webhook di PayPal al nostro listener in esecuzione su localhost è `http(s)://055b3480.ngrok.io`. Questo è tutto ciò che dobbiamo sapere per impostare l'ascoltatore.

### Sottoscrizione alle notifiche

Il nostro ultimo passo è creare webhook per la nostra applicazione, che creerà notifiche quando determinati eventi si verificano con pagamenti, rimborsi, ecc. Sulla nostra app. Abbiamo solo bisogno di creare questi webhook una volta per associarli all'applicazione, in modo che non debbano essere eseguiti ogni volta che vuoi usarli.

Per prima cosa impostiamo l'ambiente PayPal aggiungendo il requisito per l'SDK del Nodo PayPal, il nostro ID cliente / segreto dalla creazione di un'applicazione, quindi configurando l'ambiente per sandbox.

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

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

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

Successivamente, impostiamo la struttura JSON per i nostri webhook. `webhooks` contiene due informazioni, l' `url` cui devono essere inviati tutti gli eventi webhook e gli `event_types` cui vogliamo iscriversi.

Nel caso di questo esempio, l' `url` è impostato sul nostro URL live ngrok e gli eventi che stiamo ascoltando sono casi in cui i pagamenti sono completati o negati.

Per un elenco completo dei potenziali eventi, consultare

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

Infine, passiamo l'oggetto `webhooks` alla chiamata per creare i webhooks, `notification.webhook.create`. In caso di esito positivo, PayPal invierà notifiche all'endpoint specificato, che è in esecuzione su 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 volta emesso un pagamento utilizzando tali credenziali dell'applicazione, le informazioni sullo stato del pagamento verranno inviate all'endpoint che configuriamo.

Un esempio del corpo POST che PayPal invia come notifica potrebbe essere simile al seguente, che è stato inviato dopo un pagamento PayPal andato a buon fine:

```
{
  "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"
  }
]
}

```

## Aggiornamento di un webhook con un nuovo URL (esempio di nodo)

Questo esempio ti mostrerà come aggiornare un URL di inoltro webhook esistente (dove le notifiche devono essere POSTATE a). Per eseguirlo, dovresti avere l'ID restituito da PayPal quando hai creato per la prima volta i tuoi webhook.

Innanzitutto, aggiungi l'SDK PayPal e configura l'ambiente (sandbox in basso).

```

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
});
```

Quindi, imposta la struttura JSON e i dettagli del webhook. Assegna prima l'ID del tuo webhook a `webhookId`. Successivamente, nel `webhookUpdate`, specificare un'operazione di sostituzione, impostare il `path` su `/url` per specificare un aggiornamento di tale risorsa e fornire il nuovo URL per sostituirlo con il `value` inferiore.

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

Infine, chiama `notification.webhook.replace(...)`, passando in `webhookId` e `webhookUpdate`.

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

Se tutto riesce, un oggetto simile al seguente dovrebbe essere restituito da PayPal e, nel caso di questo campione, visualizzato nel terminale con le informazioni appena aggiornate.

```
{
  "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
}
```

Leggi Webhooks online: <https://riptutorial.com/it/paypal/topic/575/webhooks>

## Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con PayPal	<a href="#">Community</a> , <a href="#">Jonathan LeBlanc</a> , <a href="#">Nathan Arthur</a>
2	Creazione di abbonamenti / pagamenti ricorrenti	<a href="#">Jonathan LeBlanc</a>
3	Effettuare un pagamento con carta di credito (nodo)	<a href="#">Jonathan LeBlanc</a>
4	Fare un pagamento con PayPal	<a href="#">Jonathan LeBlanc</a>
5	Mobile Future Payments (End-End App)	<a href="#">Jonathan LeBlanc</a>
6	Pagamenti con carta di credito / carta di credito	<a href="#">Jonathan LeBlanc</a>
7	Webhooks	<a href="#">Jonathan LeBlanc</a>