Бесплатная электронная книга

# УЧУСЬ PayPal

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

#paypal

# Около

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

# глава 1: Начало работы с PayPal

## замечания

Эти руководства будут принимать пользователя через процедуры настройки учетной записи для приложений, учетных записей и т. Д. Он будет содержать все, что необходимо для работы с API-интерфейсами PayPal.

## Версии

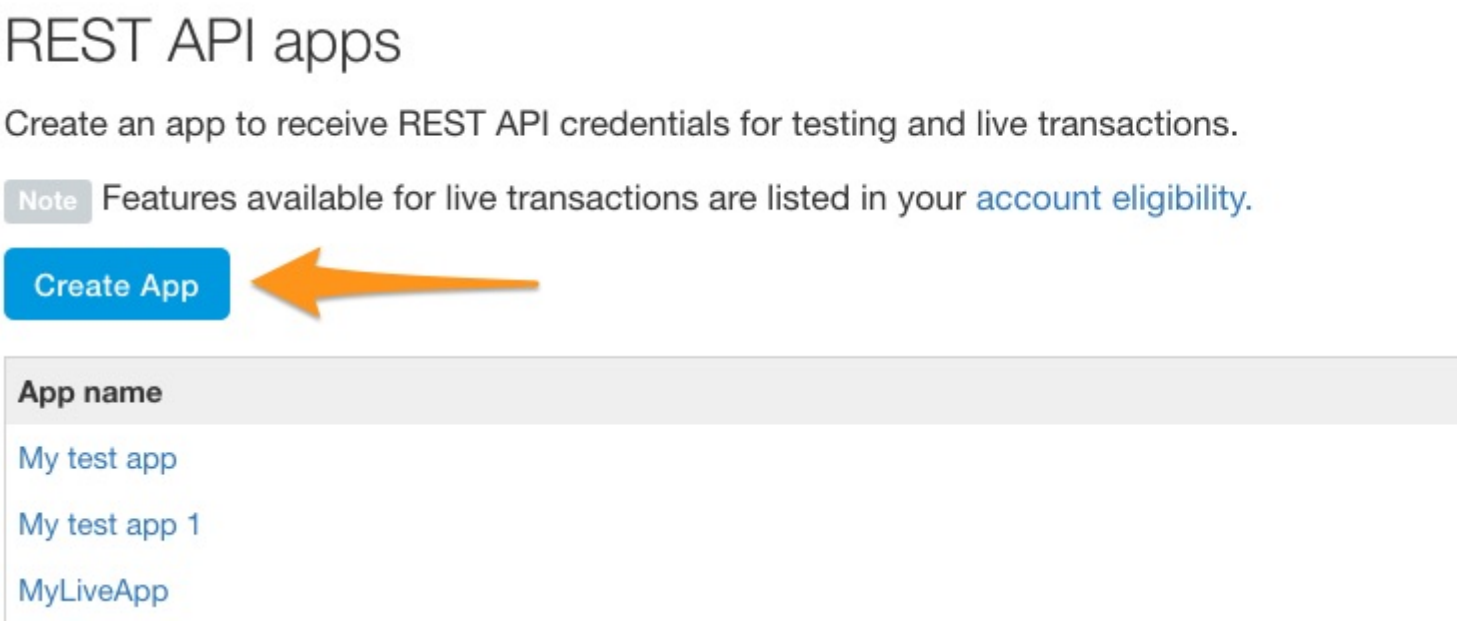
| Версия | Дата выхода |
| --- | --- |
| 1.0.0 | 2016-04-11 |

## Examples

### Создание приложения и получение идентификационных / секретных ключей клиента

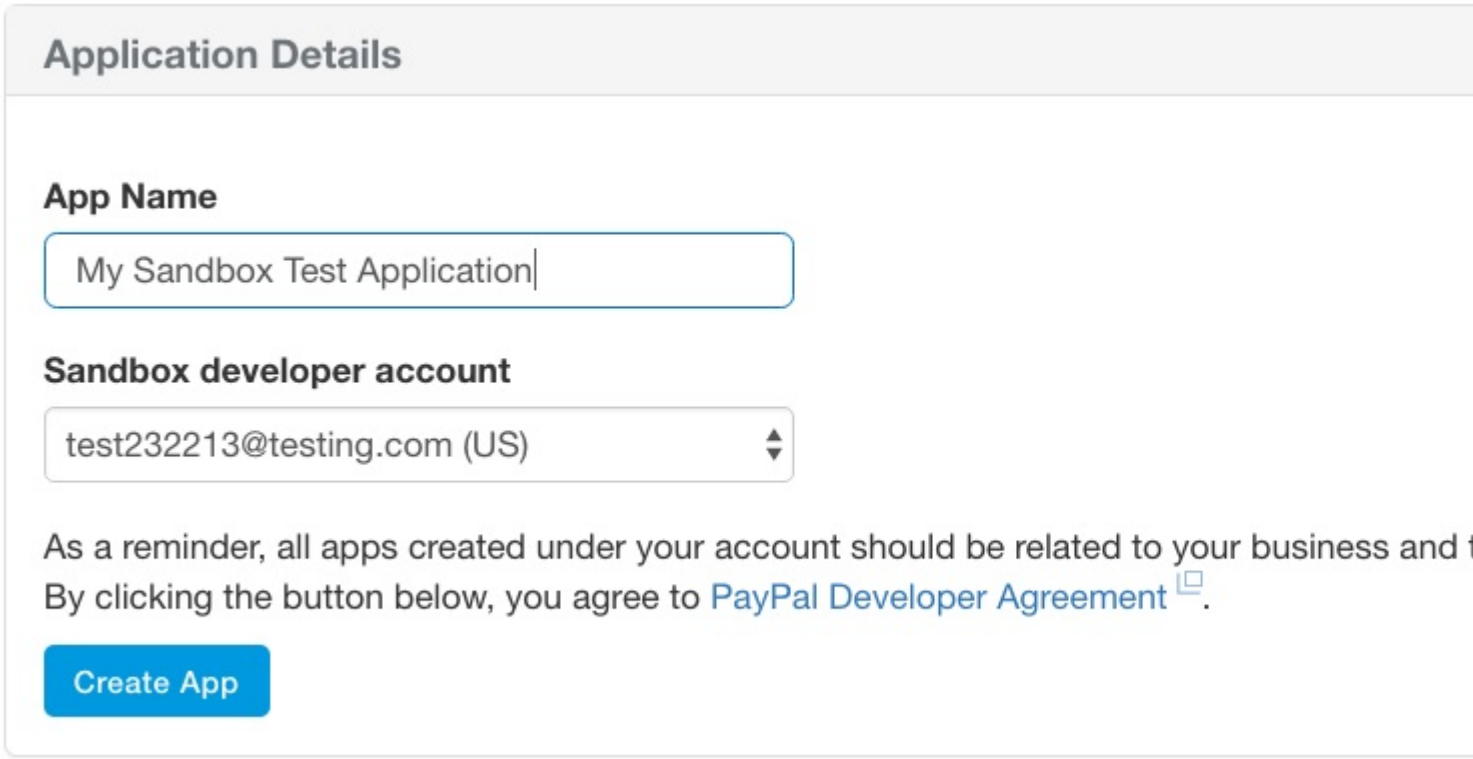
Чтобы начать работу с API-интерфейсами PayPal, вам необходимо создать приложение для получения идентификатора клиента и его секретности.

Перейдите на страницу https://developer.paypal.com/developer/applications/ , войдите в систему и нажмите «Создать приложение», как показано ниже:

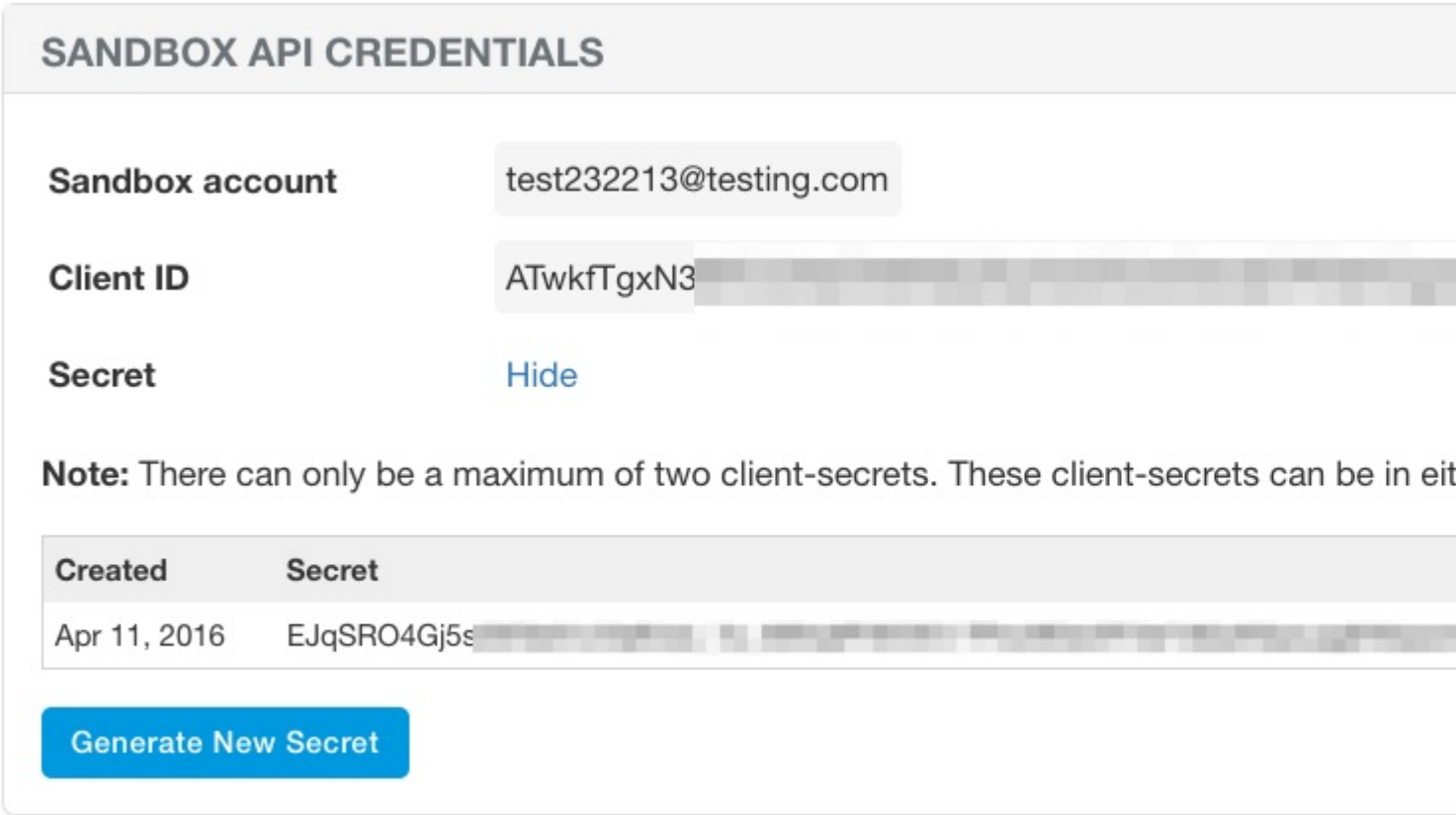



Затем введите имя приложения, выберите тестовую учетную запись для песочницы,

которую вы хотите использовать (если это новая учетная запись, оставьте значение по умолчанию) и нажмите «Создать приложение».




После создания приложения вам будет предоставлена ваша песочница и идентификатор клиента и секретный код, который будет выглядеть примерно так:

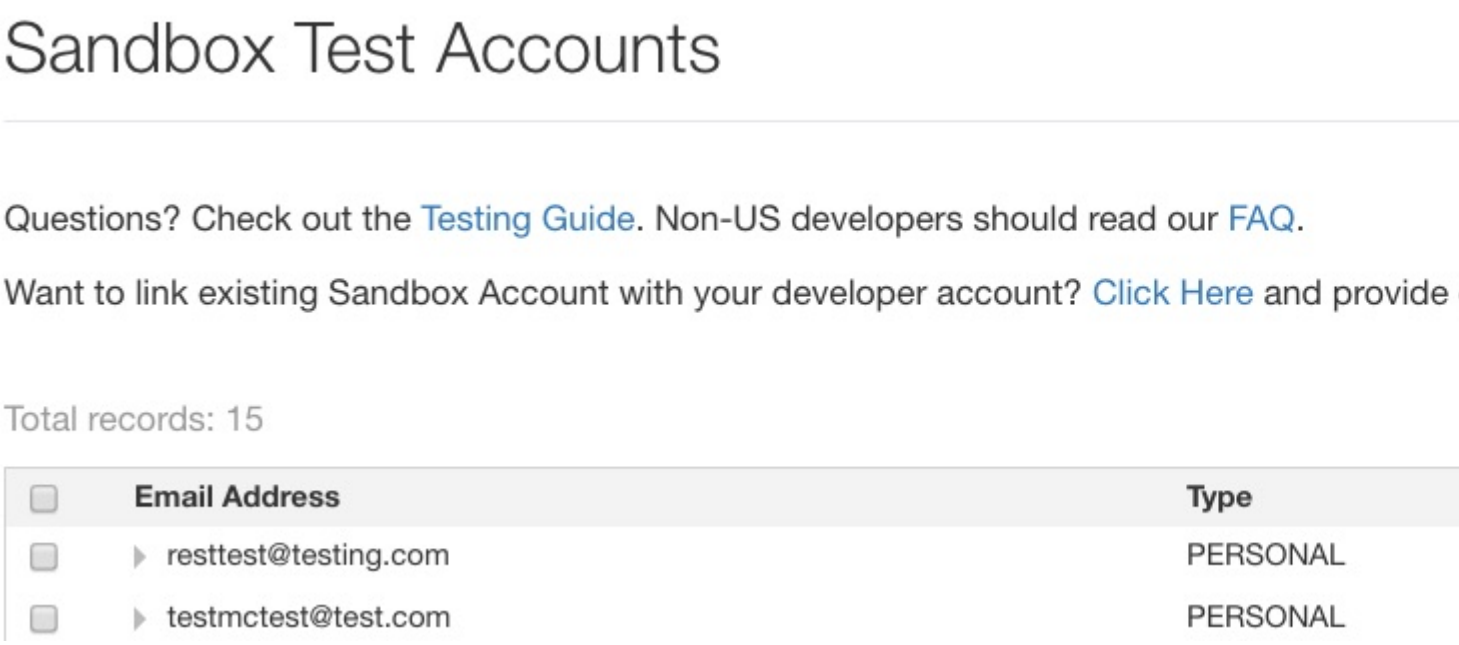


Эти учетные данные - это то, что вы будете использовать при обращении к API-интерфейсам PayPal, чтобы аутентифицировать ваше приложение и делать запросы.

### Настройка тестовых учетных записей для песочницы

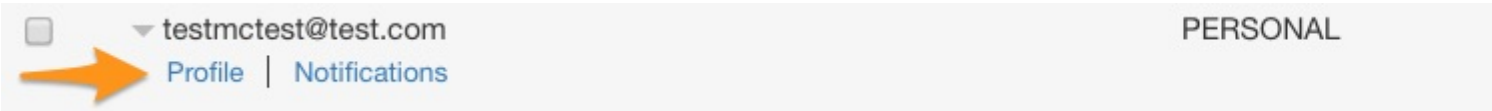
При тестировании вашей интеграции PayPal на песочнице вам понадобятся учетные записи пользователей песочницы, которые будут использоваться для прохождения потока платежей.

Перейдите на страницу https://developer.paypal.com/developer/accounts/ , войдите в систему, используя свою учетную запись PayPal, и нажмите «Создать учетную запись», как показано ниже:




Введите данные учетных записей для нового пользователя теста, включая уникальный адрес электронной почты, информацию об учетной записи, способ оплаты, баланс и т. Д. И нажмите «Создать учетную запись» внизу страницы после ее завершения. Это создаст новую учетную запись для вас, чтобы начать использовать.

Чтобы просмотреть сведения об учетной записи для этого нового пользователя, разверните запись на странице учетных записей и нажмите «Профиль».




Как только эта информация профиля загрузится, нажатие на вкладке «Финансирование» даст вам информацию о платежах для этой учетной записи, включая информацию о кредитной карте, которая может быть использована для прямой обработки кредитных карт в песочнице.

ПРИМЕЧАНИЕ. При использовании конечных точек API песочницы вам необходимо использовать тестовую учетную запись «песочница» для входа в систему и оплаты тестовых товаров, так как информация о вашем реальном счете не будет работать.

Прочитайте Начало работы с PayPal онлайн: https://riptutorial.com/ru/paypal/topic/406/начало-работы-с-paypal

# глава 2: Webhooks

## параметры

| параметр | подробности |
|---|---|
| приложение | Наша экспресс-ссылка на приложение |
| bodyParser | Ссылка на пакет body-parser для работы с закодированными органами JSON |
| ID клиента | Идентификатор клиента приложения (учетные данные OAuth 2) |
| HTTP | Пакет http для запуска сервера |
| PayPal | Опорный объект SDK SDK |
| секрет | Секрет приложения (учетные данные OAuth 2) |
| webhookId | ID веб-камеры, подлежащей изменению |
| webhookUpdate | Объект JSON, содержащий сведения о веб-узле, которые необходимо обновить |

## замечания

Эти примеры охватывают рабочие примеры того, как использовать веб-камеры PayPal для мониторинга событий для вашего приложения и платежей.

## Examples

### Тестирование веб-кнопок Sandbox с помощью ngrok и Express (Node)

В этом примере мы рассмотрим тестирование уведомлений webhook в песочнице, используя ngrok, чтобы предоставить туннель для нашего HTTP-прослушивателя Node, работающего на localhost, в Интернете. В этом примере мы собираемся использовать Node для создания веб-узлов уведомлений о событиях оплаты (например, сделанного платежа), а затем настроить сервер для прослушивания входящих HTTP-сообщений POST из событий webhook.

Мы предпримем несколько шагов, чтобы это произошло:

1. Настройте простой сервер, чтобы прослушивать входящий трафик POST с веб-узлов,

который будет являться уведомлением от PayPal, и начать прослушивание на localhost.
2. Затем используйте ngrok для предоставления туннеля из локального хоста в Интернет, чтобы PayPal мог отправлять уведомления через.
3. Наконец, подпишитесь на наше приложение (на основе предоставленных учетных данных) на события веб-хоста, которые мы хотим отслеживать, предоставив публичный URI ngrok с шага 2.

**Создание прослушивателя Webhooks**

Первое, что нам нужно сделать, это создать слушателя. Причина, по которой мы начинаем с слушателя, заключается в том, что нам нужен прямой URL-адрес ngrok, чтобы предоставлять веб-узлы, когда мы создаем или обновляем их.

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

Наш слушатель - это простой маршрут, использующий Express. Мы слушаем любой входящий трафик POST, а затем выставляем тело POST на консоль. Мы можем использовать это, чтобы делать все, что захочет, с слушателем, когда оно появится.

Когда мы создаем HTTP-сервер в конце, мы настроили его на прослушивание на localhost port 3001. Запустите этот скрипт, чтобы начать прослушивание трафика.

**Использование ngrok для размещения слушателя в Интернете**

Когда слушатель настроен на localhost: 3001, наша следующая работа заключается в том, чтобы открыть этот скрипт в Интернете, чтобы трафик мог быть отправлен на него, что является задачей ngrok.

Выполните следующую команду из окна терминала:

```
ngrok http 3001
```

Это запустит процесс предоставления живого туннеля для локального хоста на порту 3001 и предоставит следующую информацию после запуска:


Как мы видим, живым адресом, который мы можем использовать, чтобы указать веб-хост PayPal на наш текущий прослушиватель на localhost, является `http(s)://055b3480.ngrok.io` . Это все, что нам нужно знать, чтобы настроить слушателя.

**Подписка на уведомления**

Нашим последним шагом является создание веб-узлов для нашего приложения, которое будет создавать уведомления, когда определенные события происходят с платежами, возмещениями и т. Д. В нашем приложении. Нам нужно только создать эти webhooks один раз, чтобы связать их с приложением, поэтому их не нужно запускать каждый раз, когда вы хотите их использовать.

Сначала мы настраиваем среду PayPal, добавляя требование к SDK PayPal Node, нашему идентификатору клиента / секретарю от создания приложения, а затем настраиваем среду для песочницы.

```
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
});
```

Затем мы создали структуру JSON для наших веб-узлов. `webhooks` содержит две части

информации, `url` который должны быть отправлены все события веб-хоста, и `event_types` которые мы хотим подписаться.

В случае с этим образцом `url` установлен на наш прямой URL-адрес ngrok, а события, которые мы слушаем, - это случаи, когда платежи завершены или отклонены.

Полный список потенциальных событий см. На странице https://developer.paypal.com/docs/integration/direct/rest-webhooks-overview/#event-type-support .

Наконец, мы передаем объект `webhooks` в вызов для создания webhooks, `notification.webhook.create` . В случае успеха PayPal отправит уведомления конечной точке, которую мы указали, которая работает на 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);
    }
});
```

После того, как мы произведем платеж с использованием этих учетных данных, информация о состоянии платежа будет отправлена в конечную точку, которую мы установили.

Пример органа POST, который PayPal отправляет в качестве уведомления, может выглядеть следующим образом, которое было отправлено после успешного платежа PayPal:

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

```
      "details": {
        "subtotal": "7.47"
      }
    },
    "payment_mode": "INSTANT_TRANSFER",
    "protection_eligibility": "ELIGIBLE",
    "protection_eligibility_type": "ITEM_NOT_RECEIVED_ELIGIBLE,UNAUTHORIZED_PAYMENT_ELIGIBLE",
    "transaction_fee": {
      "value": "0.52",
      "currency": "USD"
    },
    "invoice_number": "",
    "custom": "",
    "parent_payment": "PAY-809936371M327284GK4L3FHA",
    "create_time": "2016-04-20T16:47:36Z",
    "update_time": "2016-04-20T16:50:07Z",
    "links": [
      {
        "href": "https:\/\/api.sandbox.paypal.com\/v1\/payments\/sale\/18169707V5310210W",
        "rel": "self",
        "method": "GET"
      },
      {
        "href":
"https:\/\/api.sandbox.paypal.com\/v1\/payments\/sale\/18169707V5310210W\/refund",
        "rel": "refund",
        "method": "POST"
      },
      {
        "href": "https:\/\/api.sandbox.paypal.com\/v1\/payments\/payment\/PAY-
809936371M327284GK4L3FHA",
        "rel": "parent_payment",
        "method": "GET"
      }
    ]
  },
  "links": [
    {
      "href": "https:\/\/api.sandbox.paypal.com\/v1\/notifications\/webhooks-events\/WH-
9FE9644311463722U-6TR22899JY792883B",
      "rel": "self",
      "method": "GET"
    },
    {
      "href": "https:\/\/api.sandbox.paypal.com\/v1\/notifications\/webhooks-events\/WH-
9FE9644311463722U-6TR22899JY792883B\/resend",
      "rel": "resend",
      "method": "POST"
    }
  ]
}
```


## Обновление веб-узла с новым URL-адресом (пример узла)

В этом примере вам будет показано, как обновить существующий URL-адрес пересылки webhook (где уведомления должны быть отправлены POSTED). Чтобы запустить это, вы должны иметь ID, предоставленный PayPal, когда вы впервые создали свои веб-камеры.

Сначала добавьте SDK PayPal и настройте среду (песочница ниже).

```
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
});
```

Затем настройте структуру JSON и данные веб-узла. Сначала назначьте идентификатор для своего веб-чек в `webhookId` . Затем в `webhookUpdate` укажите операцию замены, укажите `path /url` для указания обновления этого ресурса и укажите новый URL-адрес, чтобы заменить его `value` .

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

Наконец, вызовите `notification.webhook.replace(...)` , передав в `webhookId` и `webhookUpdate` .

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

Если все удастся, то из PayPal должен быть возвращен объект, аналогичный приведенному ниже, и, в случае этого примера, отображаться в терминале с обновленной информацией.

```
{
    "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
}
```

Прочитайте Webhooks онлайн: https://riptutorial.com/ru/paypal/topic/575/webhooks

# глава 3: Мобильные будущие платежи (от конца до конца)

## замечания

В этом примере показан практический пример конца будущего платежа PayPal с Android-устройства с использованием сервера Node.

## Examples

### Android Шаг 1: Макет, инициализация и обработка ответов сервера

Полный образец кода для этого приложения (сервер Android + Node) доступен в репозитории PayPal Developer Github .

Первым этапом создания части приложения Android является настройка базового макета и обработка ответов, возвращаемых с сервера, который мы установим в узле.

Начните с создания нового объекта PayPalConfiguration для размещения информации о вашем приложении.

```
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"));
```

Затем добавим простую кнопку `onCreate(...)` чтобы действовать как начало платежа. Это просто инициирует действие и должно быть помещено в качестве процесса инициирования для создания будущего платежа для пользователя (например, когда они согласны с подпиской).

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

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

В разделе `res > layout > activity_main.xml` мы добавим определение для кнопки со связанным с ней действием, при нажатии на нее называется `beginFuturePayment(...)` , который мы определим через минуту.

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

В разделе `res > values > strings.xml` мы добавим строчную ссылку для кнопки.

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

Теперь мы добавляем обработчик кнопки, чтобы инициировать вызов, чтобы начать процесс будущих платежей, когда пользователь нажимает кнопку. Что мы делаем здесь, это запуск платежного сервиса с помощью объекта конфигурации, который мы установили в верхней части этого примера.

```
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);
}
```

Когда будет инициирован вызов для будущего платежа, нам будет предоставлена некоторая информация, которая должна быть отправлена на наш сервер. Мы извлекаем эту информацию из действительного будущего платежного запроса ( `authCode` и `metadataId` ), затем выполняем запрос асинхронного запроса на сервер для завершения будущей оплаты (подробно описанный в шаге 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");
    }
}
```

Наконец, мы определяем наш `onDestroy()` .

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

## Android Шаг 2: Запрос сервера Async

Полный образец кода для этого приложения (сервер Android + Node) доступен в репозитории PayPal Developer Github .

На этом этапе нажата кнопка будущих платежей PayPal, у нас есть код аутентификации и идентификатор метаданных из SDK PayPal, и нам нужно передать их на наш сервер, чтобы завершить процесс будущих платежей.

В фоновом процессе ниже мы делаем несколько вещей:

- Мы настроили URI, чтобы для нашего сервера был `http://10.0.2.2:3000/fpstore` , который попадает в `/fpstore` точку `/fpstore` нашего сервера, работающего на localhost.
- Затем будет создан объект JSON, который будет отправлен через него, который содержит код аутентификации и идентификатор метаданных.
- Затем соединение будет выполнено. В случае успешного запроса (диапазон 200/201) мы можем ожидать ответа с сервера. Мы читаем этот ответ, а затем возвращаем его.
- Наконец, у нас есть `onPostExecute(...)` настроенный для обработки возвращенной строки сервера. В случае этого примера он просто регистрируется.

```
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 Шаг 3: Сервер узла для получения доступа к токенам и

### процессинговым платежам

Полный образец кода для этого приложения (сервер Android + Node) доступен в репозитории PayPal Developer Github .

Начиная с шага 2, асинхронный запрос был `/fpstore` нашему серверу в `/fpstore` точке `/fpstore` , передавая код аутентификации и идентификатор метаданных. Теперь нам нужно обменять их на токен, чтобы выполнить запрос и обработать будущий платеж.

Сначала мы настраиваем наши переменные конфигурации и объект.

```
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());
```

Теперь мы настроили экспресс-маршрут, который будет прослушивать запросы POST, отправленные в `/fpstore` точку `/fpstore` из нашего кода Android.

На этом пути мы делаем несколько вещей:

- Мы фиксируем код аутентификации и идентификатор метаданных из тела POST.
- Затем мы делаем запрос на `generateToken()` , проходя через объект кода. В случае успеха мы получаем токен, который можно использовать для создания платежа.
- Затем создаются объекты конфигурации для будущей оплаты, которая должна быть сделана, и делается запрос на `payment.create(...)` , проходящий через будущие объекты конфигурации оплаты и оплаты. Это создает будущий платеж.

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

Наконец, мы создаем сервер для прослушивания на порту 3000.

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

Прочитайте Мобильные будущие платежи (от конца до конца) онлайн:
https://riptutorial.com/ru/paypal/topic/4537/мобильные-будущие-платежи--от-конца-до-конца-

# глава 4: Мобильные платежи PayPal / кредитной картой

## параметры

| параметр | подробности |
|---|---|
| кнопка | Простая кнопка оплаты |
| конфиг | Объект конфигурации PayPal, содержащий наш идентификатор клиента (от создания приложения) и среду, которую мы хотим использовать (песочница или живая музыка) |
| оплата | Детали оплаты PayPal |
| paymentConfig | Конфигурация Намерение для информации и настроек платежа |
| serviceConfig | Конфигурация для данных параметров конфигурации |

## замечания

Образцы, связанные с обработкой платежей на мобильных устройствах

## Examples

### Android: принятие платежа по PayPal / кредитной карте

В этом уроке мы узнаем, как настроить SDK PayPal для обработки простой оплаты посредством оплаты PayPal или покупки кредитной карты. В конце этого примера вы должны иметь простую кнопку в приложении, которое при нажатии будет перенаправлять пользователя в PayPal, чтобы подтвердить установленный платеж, затем вернуть пользователя обратно в приложение и зарегистрировать подтверждение платежа.

Полный код приложения для этого примера доступен в репозитарии Gigub для разработчиков PayPal Developer .

Давайте начнем.

Первым шагом является получение и добавление SDK в ваш проект . Мы добавляем ссылку на наши зависимости build.gradle, например:

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

Теперь мы перейдем к нашему файлу MainActivity.java (или там, где вы хотите добавить интеграцию с кнопкой PayPal), и добавьте в объект `config` наш идентификатор клиента и среду (песочницу), которую мы будем использовать.

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

Теперь мы создадим кнопку в нашем `onCreate(...)` , который позволит нам обрабатывать платеж через PayPal после щелчка.

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

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

Теперь нам нужно определить функциональность этой кнопки. В вашем файле res> layout> main XML вы можете добавить следующее определение для кнопки, которое будет определять текст и обработчик onClick для кнопки с идентификатором 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" />
```

При нажатии кнопки будет вызываться метод `beginPayment(...)` . Затем мы можем добавить текст для кнопки в наш файл strings.xml, например:

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

С помощью кнопки на месте мы теперь должны обрабатывать нажатие кнопки, чтобы начать обработку платежа. Добавьте следующий `beginPayment(...)` ниже нашего предыдущего `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);
}
```

Мы здесь делаем настройку сервиса ( `serviceConfig` ), используя `config` которую мы определили ранее для нашего идентификатора клиента и среды песочницы. Затем мы указываем объект платежа, который мы хотим обработать. Для этого примера мы устанавливаем статическую цену, валюту и описание. В конечном приложении эти значения должны быть получены из того, что пользователь пытается купить в приложении. Наконец, мы установили `paymentConfig` , добавив в него как `config` и `payment` объекты, которые мы ранее определили, и запустите `paymentConfig` .

На этом этапе пользователю будут представлены экраны входа и оплаты PayPal, позволяющие им выбирать, оплачивать ли PayPal или кредитную карту (с помощью ручного ввода или card.io, если камера доступна). Этот экран будет выглядеть примерно так:

После этого мы должны иметь обработчик, готовый к тому, когда PayPal перенаправляет пользователя обратно в приложение после подтверждения оплаты или отмены. Давайте переопределим `onActivityResult(...)` для этой цели.

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

В `onActivityResult(...)` мы проверяем, возвращается ли `resultCode` , RESULT_OK ( пользовательская оплата), RESULT_CANCELED (аннулированный пользовательский платеж) или RESULT_EXTRAS_INVALID (возникла проблема с конфигурацией). В случае действительного подтверждения мы получаем объект, который возвращается из платежа, и в этом примере зарегистрируйте его. То, что будет возвращено нам, должно выглядеть примерно так:

```
{
    "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"
}
```

Если мы посмотрим на объект `response` , мы увидим, что у нас есть `state approved` , что означает, что платеж был подтвержден. На этом этапе этот объект должен быть отправлен на ваш сервер, чтобы подтвердить, что платеж действительно прошел. Дополнительные сведения об этих шагах см. В этих документах .

Наш последний шаг - очистка в нашем `onDestroy(...)` .

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

Вот и все. В этом примере мы создали простую кнопку для обработки платежа с помощью PayPal или кредитной карты. С этого момента вам предстоит несколько следующих шагов по расширению этого примера:

- Динамическое `beginPayment(...)` информации о платежах на основе выбора пользовательского продукта в `beginPayment(...)` .
- Отправка подтверждения платежа на ваш сервер и подтверждение того, что платеж действительно прошел.
- Обработка ошибок и случаев аннулирования пользователя в приложении.

Прочитайте Мобильные платежи PayPal / кредитной картой онлайн:
https://riptutorial.com/ru/paypal/topic/608/мобильные-платежи-paypal---кредитной-картой

# глава 5: Оплата PayPal

## параметры

| параметр | подробности |
|---|---|
| ID клиента | Ваш идентификатор клиента приложения PayPal (учетные данные OAuth 2) |
| связи | Простой ссылочный объект для всех возвратных ссылок HATEOAS из PayPal |
| PaymentID | Идентификатор платежа, возвращенного из PayPal, для завершения платежа |
| payerId | Идентификатор плательщика, возвращенный с PayPal, для завершения платежа |
| PayPal | Ссылка на SDK SDK PayPal |
| payReq | Объект JSON, содержащий информацию о платежах для транзакции |
| REQ | Объект запроса из запроса сервера |
| Рез | Объект ответа из запроса сервера |
| секрет | Ваш секретный код приложения PayPal (учетные данные OAuth 2) |

## замечания

Эти образцы охватывают процесс обработки платежа через PayPal с использованием SDK PayPal. Это простые образцы запросов, которые описывают многоэтапный процесс для разрешения этой опции платежа.

## Examples

### Пример сервера узла

В этом примере мы собираемся настроить интеграцию с Express Server, чтобы отобразить, как обрабатывать платеж с помощью PayPal, используя SDK PayPal Node. Для краткости мы будем использовать статическую структуру JSON для деталей платежа.

Есть три общих шага, которые мы последуем при построении функций для обработки

платежа PayPal:

1. Мы создаем объект JSON, содержащий платеж, который мы намерены обрабатывать через PayPal. Затем мы отправляем это в PayPal, чтобы получить ссылку для перенаправления пользователя для подтверждения платежа.
2. Затем мы перенаправляем пользователя в PayPal для подтверждения платежа. После подтверждения PayPal перенаправляет пользователя обратно в наше приложение.
3. После возврата в приложение мы завершим платеж от имени пользователя.

Разрушая это как простое приложение Node, мы начинаем с получения SDK SDK PayPal из NPM:

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

Затем мы настраиваем конфигурацию и пакеты приложений.

```
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
});
```

Для этого приложения требуется четыре требования:

1. HTTP-пакет для нашего сервера.
2. Пакет SDK для узла PayPal.
3. Пакет bodyParser для работы с закодированными органами JSON.
4. Рамки Express для нашего сервера.

Следующие несколько строк устанавливают переменные для идентификатора клиента и секретности, которые были получены при создании приложения . Затем мы настраиваем `bodyParser` чтобы разрешать закодированные JSON тела, затем настраиваем наше приложение с использованием сведений о приложении и определяем среду, в которой мы работаем (для производства или для тестовой среды).

Теперь давайте создадим маршрут для создания платежного запроса с помощью 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');
            }
        }
    });
});
```

Первое, что мы делаем, - это настроить объект JSON для запроса платежа, который содержит информацию, которую мы должны предоставить PayPal для создания платежа. Мы устанавливаем `intent sale` , указываем URL-адреса переадресации (где PayPal должен перенаправлять пользователя после подтверждения / отмены платежа), добавьте `payment_method paypal` чтобы сообщить, что мы сделаем платеж PayPal, а затем укажите информацию о транзакции для плательщика для подтверждения.

Затем мы вызываем `payment.create(...)` , проходящую в нашем объекте `payReq` . Это отправит запрос на создание платежа в PayPal. После того, как это будет возвращено и будет успешным, мы сможем пропустить предоставленные ссылки HATEOAS в возвращаемом объекте, чтобы извлечь URL-адрес, который нам нужен, чтобы перенаправить пользователя, который помечен как `approval_url` .

Формат ссылок HATEOAS может вызвать хрупкий ссылочный код, если он используется напрямую, поэтому мы перебираем все предоставленные ссылки и помещаем их в лучший ссылочный объект для будущего доказательства против изменений. Если `approval_url` затем найден в этом объекте, мы перенаправить пользователя.

На этом этапе пользователь перенаправляется в PayPal для подтверждения платежа. Как только они это сделают, они перенаправляются обратно в `return_url` который мы указали в функции `createPayment(...)` .

Теперь нам нужно предоставить маршрут для обработки этого возврата, чтобы завершить платеж.

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

Когда пользователь будет возвращен обратно в ваше приложение, будут также три параметра строки запроса, которые будут отправлены вместе, `paymentId` , `PayerID` и `token` . Нам нужно иметь дело только с первыми двумя.

Мы извлекаем параметры и размещаем PayerID в простом объекте для необходимости этапа выполнения платежа. Затем выполняется вызов `payment.execute(...)` , передавая эти два параметра для завершения платежа.

После того, как этот запрос будет сделан, мы увидим, успешно завершил ли платеж, проверив, `approved` ли платежный `payment.state` . Если это так, мы можем сохранить то, что нам нужно, из возвращаемого объекта платежа.

Нашим последним шагом является инициализация нашего сервера и прослушивание трафика, приближающегося к указанным нами маршрутам.

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

После инициализации сервера перейдите к `http://localhost:3000/create` инициализирует

процесс оплаты.

Прочитайте Оплата PayPal онлайн: https://riptutorial.com/ru/paypal/topic/449/оплата-paypal

# глава 6: Оплата кредитной карты (Узел)

## параметры

| параметр | подробности |
|---|---|
| card_data | Объект JSON, содержащий информацию о платежах для транзакции |
| данные кредитной карты | Объект JSON, содержащий данные кредитной карты, которые отправляются в PayPal, чтобы быть сводчатыми |
| ID клиента | Ваш идентификатор клиента приложения PayPal (учетные данные OAuth 2) |
| PayPal | Ссылка на SDK SDK PayPal |
| секрет | Ваш секретный код приложения PayPal (учетные данные OAuth 2) |
| UUID | Ссылка на пакет node-uuid |

## замечания

Этот пример берет пользователя посредством кредитования простой транзакции с использованием PayPal SDK.

## Examples

### Образец узла

Начните с установки модуля узла PayPal из NPM

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

В файле приложения добавьте информацию о конфигурации для 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
});
```

Мы добавляем требование для SDK, затем настраиваем переменные для идентификатора клиента и секретности, которые были получены при создании приложения . Затем мы настраиваем наше приложение, используя эти данные, и указываем среду, в которой мы работаем (в прямом эфире или в песочнице).

Затем мы настраиваем объект JSON, который содержит информацию о платежах для плательщика.

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

Добавьте `intent sale` и `payment_method credit_card` . Затем добавьте в карточку и адресную информацию для кредитной карты в разделе `funding_instruments` и сумму, подлежащую `funding_instruments` по `transactions` . Здесь можно разместить несколько объектов транзакции.

Наконец, мы делаем запрос на `payment.create(...)` , передавая наш объект `card_data` , чтобы обработать платеж.

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

```
  }
});
```

Если транзакция прошла успешно, мы должны увидеть объект ответа, аналогичный следующему:

```
{
  "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
}
```

В этом возвращенном объекте, имеющем `state approved` говорится, что транзакция прошла успешно. Под объектом `links` есть ряд ссылок HATEOAS, которые предоставляют потенциальные последующие шаги, которые могут быть предприняты в действии, которое только что было выполнено. В этом случае мы можем получить информацию об оплате, сделав запрос GET на предоставленную `self` конечную точку.

## Выполнение платежа со сводной кредитной картой (узлом)

В этом примере мы рассмотрим, как хранить кредитную карту с помощью хранилища PayPal, а затем ссылаться на эту сохраненную кредитную карту для обработки транзакции кредитной карты для пользователя.

Причина, по которой мы хотим использовать хранилище, заключается в том, что нам не нужно хранить конфиденциальную информацию о кредитных картах на наших

собственных серверах. Мы просто ссылаемся на способ оплаты с помощью предоставленного идентификатора хранилища, а это значит, что нам не нужно иметь дело со многими нормами соответствия PCI с сохранением самих кредитных карт.

Как и в предыдущих примерах, мы начинаем с создания нашей среды.

```
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
});
```

Единственное отличие от предыдущих образцов заключается в том, что нам требуется новый пакет `node-uuid` , который должен использоваться для создания уникальных UUID для плательщиков при хранении карты. Вы можете установить этот пакет через:

```
npm install node-uuid
```

Затем мы определяем объект JSON кредитной карты, который будет отправлен в хранилище PayPal для хранения. Он содержит информацию с карты, а также уникальный идентификатор плательщика, который мы генерируем с помощью `node-uuid` . Вы должны сохранить этот уникальный `payer_id` в своей собственной базе данных, поскольку он будет использоваться при создании платежа со сводчатой картой.

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

Наконец, нам необходимо сохранить кредитную карту и обработать платеж с использованием этой карты. Чтобы `credit_card.create(...)` кредитную карту, мы вызываем `credit_card.create(...)` , передавая объект `credit_card_details` который мы только что создали. Если все пойдет хорошо, мы должны вернуть объект нам с подробностями о сводчатой карте. Для оплаты этой карты нам понадобятся только две части информации: плательщик, который мы уже сохранили, и идентификатор хранилища, который также должен храниться как ссылка в нашей собственной базе данных.

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

В разделе, посвященном успешному сводке кредитной карты, мы просто определяем данные карты и обрабатываем платеж, как это было сделано с предыдущим примером обработки кредитных карт. Основное различие в структуре `card_data` объекта является `funding_instruments` раздел, который мы определяем под `payer` . Вместо определения информации о кредитной карте вместо этого мы используем следующий объект, который содержит ссылку идентификатора хранилища, и идентификатор плательщика:

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

Так мы используем сводчатую карточку для обработки платежа.

Прочитайте Оплата кредитной карты (Узел) онлайн: https://riptutorial.com/ru/paypal/topic/444/оплата-кредитной-карты--узел-

# глава 7: Создание подписки / периодические платежи

## параметры

| параметр | подробности |
| --- | --- |
| billingAgreementAttributes | Объект конфигурации для создания биллингового соглашения |
| billingPlan | Идентификатор плана платежей из строки запроса |
| billingPlanAttribs | Объект конфигурации для создания платежного плана |
| billingPlanUpdateAttributes | Объект конфигурации для изменения плана фактурирования в активное состояние |
| ID клиента | Идентификатор клиента вашего приложения (ключи OAuth) |
| HTTP | Ссылка на пакет http для настройки нашего простого сервера |
| ISODate | Дата ISO для установки даты начала подписки |
| связи | Объект ссылки HATEOAS для извлечения URL-адреса переадресации в PayPal |
| Титулы | Параметры строки запроса |
| PayPal | Ссылка на SDK PayPal |
| секрет | Секрет вашего приложения (ключи OAuth) |
| знак | Акт одобрения биллингового соглашения, предоставленный после перенаправления PayPal для выполнения биллингового соглашения |

## замечания

Эти примеры проходят процесс создания системы подписки / повторной оплаты с использованием PayPal.

Процесс создания подписки:

1. Создайте тарифный план. Это многоразовая модель, которая описывает детали подписки.
2. Активируйте тарифный план.
3. Когда вы хотите создать подписку для пользователя, вы создаете соглашение о выставлении счетов, используя идентификатор плана фактурирования, на который они должны быть подписаны.
4. После создания вы перенаправляете пользователя в PayPal для подтверждения подписки. После подтверждения пользователь перенаправляется обратно на веб-сайт продавца.
5. Наконец, вы выполняете соглашение о выставлении счетов, чтобы начать подписку.

# Examples

### Шаг 2. Создание подписки для пользователя с использованием соглашения о выставлении счетов (образец узла)

Второй шаг к созданию подписки для пользователя - это создание и выполнение соглашения о выставлении счетов на основе существующего активированного плана фактурирования. В этом примере предполагается, что вы уже прошли и активировали тарифный план в предыдущем примере и имеете идентификатор для этого плана фактурирования для ссылки в примере.

Когда вы настраиваете соглашение о выставлении счетов для создания подписки для пользователя, вы будете следовать 3 шагам, которые могут напоминать обработку платежа PayPal:

1. Вы создаете соглашение о выставлении счетов, ссылаясь на базовый тарифный план с помощью идентификатора.
2. После создания вы перенаправляете пользователя в PayPal (при оплате через PayPal) для подтверждения подписки. После подтверждения PayPal перенаправляет пользователя обратно на ваш сайт, используя перенаправление, указанное в базовом тарифном плане.
3. Затем вы выполняете соглашение о выставлении счетов, используя маркер, предоставленный обратно через перенаправление PayPal.

В этом примере создается HTTP-сервер с поддержкой Express, чтобы продемонстрировать процесс соглашения о выставлении счетов.

Чтобы начать пример, сначала нам нужно настроить нашу конфигурацию. Мы добавляем четыре требования: SDK PayPal, `body-parser` для обработки закодированных JSON тел, `http` для нашей простой интеграции с сервером и `express` для платформы Express. Затем мы определяем наш идентификатор клиента и секрет, создавая приложение, настраиваем

SDK для песочницы, а затем настраиваем bodyParser для обработки тел 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());
```

Наш первый шаг в биллинговом соглашении заключается в создании маршрута для обработки заключения соглашения о выставлении счетов и перенаправления пользователя в PayPal для подтверждения этой подписки. Мы предполагаем, что идентификатор плана фактурирования передается как параметр строки запроса, например, путем загрузки следующего URL с идентификатором плана из предыдущего примера:

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

Теперь нам нужно использовать эту информацию для создания соглашения о выставлении счетов.

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

Мы начинаем с извлечения идентификатора платежного плана из строки запроса и создания даты начала плана.

Следующее определение объекта, `billingAgreementAttributes` , состоит из информации для подписки. Он содержит читаемую информацию о плане, ссылку на идентификатор платежного плана, способ оплаты и данные о доставке (при необходимости для подписки).

Затем выполняется вызов для `billingAgreement.create(...)` , передавая объект `billingAgreementAttributes` мы только что создали. Если все будет успешным, мы должны вернуть объект соглашения об оплате, содержащий сведения о нашей недавно созданной подписке. Этот объект также содержит несколько ссылок HATEOAS, которые дают нам следующие шаги, которые можно предпринять в этом недавно созданном соглашении. Тот, о котором мы заботимся здесь, обозначается как `approval_url` .

Мы перебираем все предоставленные ссылки, чтобы поместить их в объект, который легко ссылается. Если `approval_url` является одной из этих ссылок, мы перенаправляем пользователя на эту ссылку, которая является PayPal.

На этом этапе пользователь подтверждает подписку на PayPal и перенаправляется обратно на URL-адрес, указанный в базовом тарифном плане. Наряду с этим URL-адресом PayPal также передает токен вдоль строки запроса. Этот токен - это то, что мы собираемся использовать для выполнения (или начала) подписки.

Давайте создадим эту функциональность в следующем маршруте.

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

Мы извлекаем токен из строки запроса, а затем вызываем вызов `billingAgreement.execute` , проходя через этот токен. Если все успешно, у нас теперь есть действительная подписка для пользователя. Объект возврата содержит информацию об активном биллинговом соглашении.

Наконец, мы настроили наш HTTP-сервер для прослушивания трафика на наши маршруты.

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

## Шаг 1. Создание модели подписки с использованием плана фактурирования (пример узла)

При создании подписки для пользователя сначала необходимо создать и активировать план фактурирования, который затем подписчивается на использование биллингового соглашения. Полный процесс создания подписки подробно описан в этом разделе.

В этом примере мы будем использовать SDK PayPal Node . Вы можете получить его из NPM, используя следующую команду:

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

В нашем .js-файле мы сначала настроили нашу конфигурацию SDK, которая включает добавление требования к SDK, определение нашего идентификатора клиента и секретности от создания нашего приложения , а затем настройку SDK для среды песочницы.

```
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
});
```

Затем нам нужно настроить два объекта JSON. Объект `billingPlanAttribs` содержит информацию и `billingPlanAttribs` оплаты для плана фактурирования, на который мы можем подписаться, и объект `billingPlanUpdateAttributes` содержит значения для установки плана фактурирования в активное состояние, что позволяет использовать его.

```
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"
    }
}];
```

Внутри объекта `billingPlanAttribs` есть некоторые релевантные фрагменты информации:

- **имя** / **описание** / **тип** : Основная визуальная информация для описания плана и тип плана.
- **payment_definitions** : информация о том, как план должен функционировать и быть выставлен счет. Подробнее о полях здесь .
- **merchant_preferences** : дополнительные структуры вознаграждения, URL-адреса переадресации и настройки для плана подписки. Подробнее о полях здесь .

С помощью этих объектов мы теперь можем создавать и активировать тарифный план.

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

Мы называем `billingPlan.create(...)` , передавая объект `billingPlanAttribs` который мы только что создали. Если это будет успешным, объект возврата будет содержать информацию о плане фактурирования. Для примера нам просто нужно использовать идентификатор платежного плана, чтобы активировать план для использования.

Затем мы вызываем `billingPlan.update(...)` , передавая идентификатор платежного плана и объект `billingPlanUpdateAttributes` мы создали ранее. Если это будет успешным, наш тарифный план теперь будет активным и готовым к использованию.

Чтобы создать подписку для пользователя (или нескольких пользователей) в этом плане, нам нужно будет ссылаться на идентификатор платежного плана ( `billingPlan.id` выше), поэтому сохраните его в месте, на которое можно легко ссылаться.

На втором этапе подписки нам необходимо создать соглашение о выставлении счетов на основе только что созданного плана и выполнить его, чтобы начать обработку подписки для пользователя.

Прочитайте Создание подписки / периодические платежи онлайн: https://riptutorial.com/ru/paypal/topic/467/создание-подписки---периодические-платежи

# кредиты

| S. No | Главы | Contributors |
|---|---|---|
| 1 | Начало работы с PayPal | Community, Jonathan LeBlanc, Nathan Arthur |
| 2 | Webhooks | Jonathan LeBlanc |
| 3 | Мобильные будущие платежи (от конца до конца) | Jonathan LeBlanc |
| 4 | Мобильные платежи PayPal / кредитной картой | Jonathan LeBlanc |
| 5 | Оплата PayPal | Jonathan LeBlanc |
| 6 | Оплата кредитной карты (Узел) | Jonathan LeBlanc |
| 7 | Создание подписки / периодические платежи | Jonathan LeBlanc |