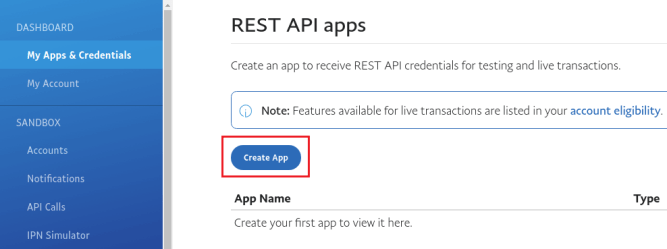
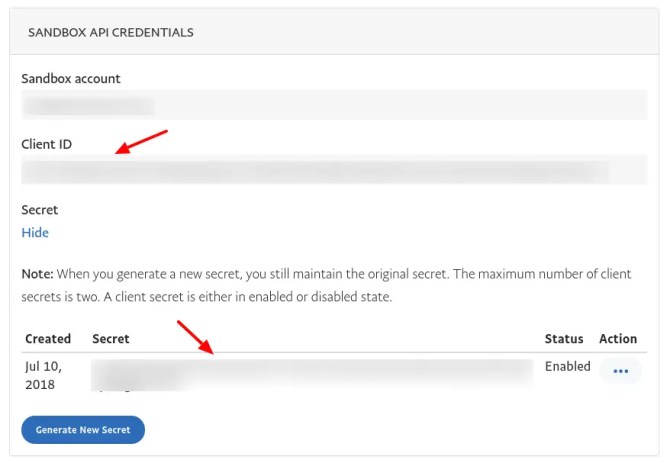
Head over to your [PayPal developer](https://developer.paypal.com) account and login into it. In the developer dashboard, click on the ‘My Apps & Credentials’ menu. Then click on the ‘Create App’ button under the REST API apps section.



Follow the steps as prompted and you will get your sandbox and live credentials. For now, copy the client id and client secret of sandbox mode. We should first test payments in sandbox mode and if it works then switch to live mode.



**Basic Setup in Laravel to Accept Payment using PayPal**

To integrate the PayPal system in Laravel, you need to perform certain steps.

* Set up the environment required – install packages, store API credentials, etc.
* Build a payment flow.
* Store successful transactions in the database.

I am going to create a database table first to hold the transaction details. I’ll name the table as ‘payments’. Create a migration for the ‘payments’ table using the command:

php artisan make:migration create\_payments\_table

Open the migration file and specify the columns in the up() method.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <?php  ...  ...  public function up()  {      Schema::create('payments', function (Blueprint $table) {          $table->bigIncrements('id');          $table->string('payment\_id');          $table->string('payer\_id');          $table->string('payer\_email');          $table->float('amount', 10, 2);          $table->string('currency');          $table->string('payment\_status');          $table->timestamps();      });  } |

Execute this migration using the command:

php artisan migrate

This command will create a ‘payments’ table in your database. Next, create a model corresponding to the ‘payments’ table.

php artisan make:model Payment

After this, add the PayPal credentials in your .env file.

|  |  |
| --- | --- |
| 1  2  3 | PAYPAL\_CLIENT\_ID=PASTE\_HERE\_CLIENT\_ID  PAYPAL\_CLIENT\_SECRET=PASTE\_HERE\_CLIENT\_SECRET  PAYPAL\_CURRENCY=USD |

Clear the configuration cache using the command:

php artisan config:cache

For the sake of the tutorial, I have passed the ‘USD’ currency. The user can change the currency as per their requirements.

Now to build a PayPal payment flow, the system requires to have few routes. Let’s define them in the routes/web.php file.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php  ...  ...  Route::get('payment', 'PaymentController@index');  Route::post('charge', 'PaymentController@charge');  Route::get('success', 'PaymentController@success');  Route::get('error', 'PaymentController@error'); |

We will create PaymentController in the next steps.

**PayPal Payment Gateway Integration in Laravel**

Integrating payment gateways API in the application is quite complex stuff. Fortunately, Omnipay library made developer’s lives easy. [Omnipay](https://github.com/thephpleague/omnipay) is the most popular payment processing library for PHP. It gives an easy and clean code for integrating different payment gateways. You don’t need to read payment gateway documentation thoroughly. All you need to do is use the code/system provided by Omnipay and you are done.

That being said, install Omnipay library using the command:

composer require league/omnipay omnipay/paypal

After this, create a controller PaymentController and define the methods mentioned in the route file.

php artisan make:controller PaymentController

The controller is responsible to initiate and complete the PayPal flow for your users.

* Call the view.
* Take the user inputs.
* Process the amount towards PayPal.
* Authorize the payment and charge the customer.
* Insert transaction details into the database.

I am adding a code in the PaymentController which will perform all these steps.

**PaymentController.php**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106 | <?php    namespace App\Http\Controllers;    use Illuminate\Http\Request;  use Omnipay\Omnipay;  use App\Models\Payment;    class PaymentController extends Controller  {        private $gateway;        public function \_\_construct()      {          $this->gateway = Omnipay::create('PayPal\_Rest');          $this->gateway->setClientId(env('PAYPAL\_CLIENT\_ID'));          $this->gateway->setSecret(env('PAYPAL\_CLIENT\_SECRET'));          $this->gateway->setTestMode(true); //set it to 'false' when go live      }        /\*\*       \* Call a view.       \*/      public function index()      {          return view('payment');      }        /\*\*       \* Initiate a payment on PayPal.       \*       \* @param  \Illuminate\Http\Request  $request       \*/      public function charge(Request $request)      {          if($request->input('submit'))          {              try {                  $response = $this->gateway->purchase(array(                      'amount' => $request->input('amount'),                      'currency' => env('PAYPAL\_CURRENCY'),                      'returnUrl' => url('success'),                      'cancelUrl' => url('error'),                  ))->send();                    if ($response->isRedirect()) {                      $response->redirect(); // this will automatically forward the customer                  } else {                      // not successful                      return $response->getMessage();                  }              } catch(Exception $e) {                  return $e->getMessage();              }          }      }        /\*\*       \* Charge a payment and store the transaction.       \*       \* @param  \Illuminate\Http\Request  $request       \*/      public function success(Request $request)      {          // Once the transaction has been approved, we need to complete it.          if ($request->input('paymentId') && $request->input('PayerID'))          {              $transaction = $this->gateway->completePurchase(array(                  'payer\_id'             => $request->input('PayerID'),                  'transactionReference' => $request->input('paymentId'),              ));              $response = $transaction->send();                if ($response->isSuccessful())              {                  // The customer has successfully paid.                  $arr\_body = $response->getData();                    // Insert transaction data into the database                  $payment = new Payment;                  $payment->payment\_id = $arr\_body['id'];                  $payment->payer\_id = $arr\_body['payer']['payer\_info']['payer\_id'];                  $payment->payer\_email = $arr\_body['payer']['payer\_info']['email'];                  $payment->amount = $arr\_body['transactions'][0]['amount']['total'];                  $payment->currency = env('PAYPAL\_CURRENCY');                  $payment->payment\_status = $arr\_body['state'];                  $payment->save();                    return "Payment is successful. Your transaction id is: ". $arr\_body['id'];              } else {                  return $response->getMessage();              }          } else {              return 'Transaction is declined';          }      }        /\*\*       \* Error Handling.       \*/      public function error()      {          return 'User cancelled the payment.';      }  } |

We have called a blade file in the index method. So, create a blade file called payment.blade.php and add code below in it.

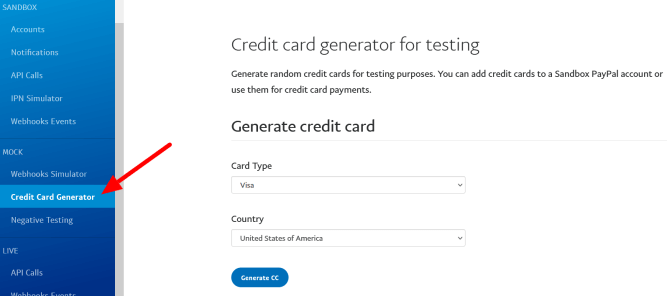
|  |  |
| --- | --- |
| 1  2  3  4  5 | <form action="{{ url('charge') }}" method="post">      <input type="text" name="amount" />      {{ csrf\_field() }}      <input type="submit" name="submit" value="Pay Now">  </form> |

When we submit this form, control goes to the charge function in the PaymentController and controller processes the rest payment flow.

**Test Payment in Sandbox Mode**

While testing a payment, you require to add dummy card details to your PayPal sandbox account. You can get these dummy card details from your [PayPal Developer Dashboard](https://developer.paypal.com/developer/accounts).

Upon login to the dashboard, head over to the Credit Card Generator from the left-side menu and generate dummy card details.



**Make Payment Live**

Once you are done with testing the transactions in the sandbox mode, you are ready to switch to production mode. To make your payment system live, update your live PayPal client id and client secret in the environment file.

Next, in the constructor method of a PaymentController, set the **false** value to the setTestMode() method.

|  |  |
| --- | --- |
| 1 | $this->gateway->setTestMode(false); |

**Send Product Information to PayPal**

In the previous steps, we are sending the amount to pay on PayPal. You may also want to send product information. The user can see these product details on the payment page before making a payment.

To send the product information, you need to pass the ‘items’ array to the purchase method as follows.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | $response = $this->gateway->purchase(array(      'amount' => $request->input('amount'),      'items' => array(          array(              'name' => 'Course Subscription',              'price' => $request->input('amount'),              'description' => 'Get access to premium courses.',              'quantity' => 1          ),      ),      'currency' => env('PAYPAL\_CURRENCY'),      'returnUrl' => url('paymentsuccess'),      'cancelUrl' => url('paymenterror'),  ))->send(); |

Here I am passing the product details statically. You should make it dynamic depending on your flow.

I hope you got to know about PayPal payment gateway integration in Laravel. This tutorial should help you to add the PayPal flow to your Laravel application. It will allow you to receive online payments through PayPal.