

Laravel 10 Integrating with PayPal, OAuth and SSO

Example 1: PayPal Integration

Step 1: Set Up a PayPal Developer Account

- ✓ Create a PayPal Developer Account: Go to PayPal Developer and log in with your PayPal account. If you don't have one, create an account.
- ✓ Create a Sandbox Account: In the dashboard, navigate to "Sandbox" and create a business account. This will be used for testing purposes.
- ✓ Create an App: Go to "My Apps & Credentials" and create a new app under the Sandbox section. You'll receive a Client ID and Client Secret, which you'll use to configure your Laravel application.

Step 2: Install PayPal SDK

You'll need to install the PayPal SDK for PHP. You can do this using Composer.

```
composer require paypal/rest-api-sdk-php
```

Step 3: Configure PayPal in Laravel

Add PayPal Credentials: In your .env file, add the following lines with your PayPal sandbox credentials:

```
PAYPAL_CLIENT_ID=your_client_id  
PAYPAL_CLIENT_SECRET=your_client_secret  
PAYPAL_MODE=sandbox # or 'live' for production
```

Update config/services.php: Add the PayPal configuration to your config/services.php file.

```
'paypal' => [  
    'client_id' => env('PAYPAL_CLIENT_ID'),  
    'secret' => env('PAYPAL_CLIENT_SECRET'),  
    'settings' => [  
        'mode' => env('PAYPAL_MODE', 'sandbox'),  
        'http.ConnectionTimeout' => 30,  
        'log.LogEnabled' => true,  
        'log.FileName' => storage_path() . '/logs/paypal.log',  
        'log.LogLevel' => 'ERROR'  
    ],  
],
```

Step 4: Create Routes and Controllers

Create Routes: Define the routes in your routes/web.php file.

```
use App\Http\Controllers\PayPalController;  
  
Route::get('paypal', [PayPalController::class, 'index']);  
Route::post('paypal', [PayPalController::class, 'payWithPayPal'])->name('paypal.pay');  
Route::get('paypal/success', [PayPalController::class, 'success'])->name('paypal.success');  
Route::get('paypal/cancel', [PayPalController::class, 'cancel'])->name('paypal.cancel');
```

Create Controller: Generate a controller to handle PayPal logic.

```
php artisan make:controller PayPalController
```

PayPalController.php:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use PayPal\Api\Amount;
use PayPal\Api\Payer;
use PayPal\Api\Payment;
use PayPal\Api\PaymentExecution;
use PayPal\Api\RedirectUrls;
use PayPal\Api\Transaction;
use PayPal\Rest\ApiContext;
use PayPal\Auth\OAuthTokenCredential;
use Illuminate\Support\Facades\Redirect;

class PayPalController extends Controller
{
    private $apiContext;

    public function __construct()
    {
        $paypalConfig = config('services.paypal');

        $this->apiContext = new ApiContext(
            new OAuthTokenCredential(
                $paypalConfig['client_id'],
                $paypalConfig['secret']
            )
        );

        $this->apiContext->setConfig($paypalConfig['settings']);
    }

    public function index()
    {
        return view('paypal.index');
    }

    public function payWithPayPal(Request $request)
    {
        $payer = new Payer();
        $payer->setPaymentMethod('paypal');

        $amount = new Amount();
        $amount->setTotal($request->amount);
        $amount->setCurrency('USD');

        $transaction = new Transaction();
```

```

$transaction->setAmount($amount);
$transaction->setDescription('Payment description');

$redirectUrls = new RedirectUrls();
$redirectUrls->setReturnUrl(route('paypal.success'))
    ->setCancelUrl(route('paypal.cancel'));

$payment = new Payment();
$payment->setIntent('sale')
    ->setPayer($payer)
    ->setTransactions([$transaction])
    ->setRedirectUrls($redirectUrls);

try {
    $payment->create($this->apiContext);

    return Redirect::away($payment->getApprovalLink());
} catch (\PayPal\Exception\PayPalConnectionException $ex) {
    return redirect()->route('paypal.cancel');
}
}

public function success(Request $request)
{
    if (empty($request->PayerID) || empty($request->token)) {
        return redirect()->route('paypal.cancel');
    }

    $paymentId = $request->paymentId;
    $payment = Payment::get($paymentId, $this->apiContext);

    $execution = new PaymentExecution();
    $execution->setPayerId($request->PayerID);

    try {
        $result = $payment->execute($execution, $this->apiContext);

        if ($result->getState() == 'approved') {
            return 'Payment success';
        }
    } catch (\PayPal\Exception\PayPalConnectionException $ex) {
        return redirect()->route('paypal.cancel');
    }

    return redirect()->route('paypal.cancel');
}

public function cancel()
{
    return 'Payment canceled';
}

```

```
}  
}
```

Step 5: Create Views

Create a View for Payment: Create a view file resources/views/paypal/index.blade.php.

```
<!DOCTYPE html>  
<html>  
<head>  
    <title>PayPal Payment</title>  
</head>  
<body>  
    <h1>Pay with PayPal</h1>  
    <form action="{{ route('paypal.pay') }}" method="POST">  
        @csrf  
        <input type="text" name="amount" placeholder="Enter Amount">  
        <button type="submit">Pay with PayPal</button>  
    </form>  
</body>  
</html>
```

Step 6: Testing the Integration

Run Your Application: Start your Laravel application.

```
php artisan serve
```

Test the Payment Flow: Navigate to the payment page (e.g., <http://localhost:8000/paypal>), enter an amount, and click the button to pay with PayPal. You'll be redirected to the PayPal sandbox environment for testing.

Note: (PHP 7.2+)

Fix warning

Warning: sizeof(): Parameter must be an array or an object that implements Countable in lib/PayPal/Common/PayPalModel.php on line 178

```
lib/PayPal/Common/PayPalModel.php  
@@ -175,7 +175,7 @@ private function _convertToArray($param)  
175 175     foreach ($param as $k => $v) {  
176 176         if ($v instanceof PayPalModel) {  
177 177             $ret[$k] = $v->toArray();  
178      -     } elseif (sizeof($v) <= 0 && is_array($v)) {  
178      +     } elseif (is_array($v) && sizeof($v) <= 0) {  
179 179         $ret[$k] = array();  
180 180     } elseif (is_array($v)) {  
181 181         $ret[$k] = $this->_convertToArray($v);
```

lib/PayPal/Common/PayPalModel.php on line 178

[old]

```
} elseif (sizeof($v) <= 0 && is_array($v)) {
```

[change to this]

```
} elseif (is_array($v) && sizeof($v) <= 0) {
```

Additional Considerations

- ✓ Error Handling: Implement proper error handling and logging for production environments.
- ✓ Security: Ensure secure handling of sensitive data and compliance with PayPal's security guidelines.
- ✓ Production Configuration: For live payments, update the PAYPAL_MODE in the .env file to live and use live credentials.

Example 2: Laravel Socialite

Laravel SSO using Socialite

1. Create project

```
\>composer create-project --prefer-dist --ignore-platform-reqs laravel/laravel:8.2.0 socialitedemo  
\>php -S localhost:8003 -t .\public
```

2. Create Laravel authentication

```
\>composer require laravel/ui:3.0.0 --dev --ignore-platform-reqs
```

Install UI Bootstrap, compile and run

```
\>php artisan ui bootstrap --auth  
\>npm install  
\>npm run dev
```

If you are getting this message when I run npm run dev:

DisabledForUser Please make sure that the app id is set correctly.

Go to your windows settings Windows settings → Notifications & actions and enable notifications then try **npm run dev** again.

Try accessing login page

| | |
|-------------------------------|-------------------------------|
| http://localhost/public/login | //if hosted in xampp |
| http://localhost:8000/login | //if using project dev server |

Laravel Login Register

Login

E-Mail Address

Password

☐ Remember Me

Login

[Forgot Your Password?](#)

3. Edit design and add a button

./resources/auth/login.blade.php

```
@extends('layouts.app')

@section('content')
<div class="container">
  <div class="row justify-content-center">
    <div class="col-md-8">
      <div class="card">
        <div class="card-header">{{ __('Login') }}</div>

        <div class="card-body">
          <form method="POST" action="{{ route('login') }}">
            @csrf

            <div class="form-group row">
              <div class="col-md-6 offset-md-3">
                <a href="#" class="btn btn-danger btn-block">Login with Google</a>
                <a href="#" class="btn btn-primary btn-block">Login with Facebook</a>
                <a href="#" class="btn btn-dark btn-block">Login with Github</a>
              </div>
            </div>

            <p style="text-align: center;">OR</p>

            <div class="form-group row">
              <div class="col-md-6 offset-md-3">
                <input id="email" type="email" class="form-control @error('email') is-invalid @enderror"
name="email" value="{{ old('email') }}" required autocomplete="email" autofocus placeholder="Email
Required">

                @error('email')
                  <span class="invalid-feedback" role="alert">
                    <strong>{{ $message }}</strong>
                  </span>
                @enderror
              </div>
            </div>
          </form>
        </div>
      </div>
    </div>
  </div>
</div>
```

```

        </div>
    </div>

    <div class="form-group row">

        <div class="col-md-6 offset-md-3">
            <input id="password" type="password" class="form-control @error('password') is-invalid
@enderror" name="password" required autocomplete="current-password" placeholder="Password Required">

            @error('password')
                <span class="invalid-feedback" role="alert">
                    <strong>{{ $message }}</strong>
                </span>
            @enderror
        </div>
    </div>

    <div class="form-group row">
        <div class="col-md-6 offset-md-4">
            <div class="form-check">
                <input class="form-check-input" type="checkbox" name="remember" id="remember" {{
old('remember') ? 'checked' : '' }}>

                <label class="form-check-label" for="remember">
                    {{ __('Remember Me') }}
                </label>
            </div>
        </div>
    </div>

    <div class="form-group row mb-0">
        <div class="col-md-8 offset-md-4">
            <button type="submit" class="btn btn-primary">
                {{ __('Login') }}
            </button>

            @if (Route::has('password.request'))
                <a class="btn btn-link" href="{{ route('password.request') }}">
                    {{ __('Forgot Your Password?') }}
                </a>
            @endif
        </div>
    </div>
</form>
</div>
</div>
</div>
</div>
</div>
</div>
@endsection

```

4. Install Laravel Socialite

```
composer require laravel/socialite:5.0.3 --ignore-platform-reqs
```

5. We will login with facebook, google and github

./config/services.php

```
...
'github' => [
    'client_id' => env('GITHUB_CLIENT_ID'),
    'client_secret' => env('GITHUB_CLIENT_SECRET'),
    'redirect' => 'http://example.com/callback-url',
],

'facebook' => [
    'client_id' => env('FACEBOOK_CLIENT_ID'),
    'client_secret' => env('FACEBOOK_CLIENT_SECRET'),
    'redirect' => 'http://example.com/callback-url',
],

'google' => [
    'client_id' => env('GOOGLE_CLIENT_ID'),
    'client_secret' => env('GOOGLE_CLIENT_SECRET'),
    'redirect' => 'http://example.com/callback-url',
],
...
```

./routes/web.php

```
...
Route::get('login/google', 'App\Http\Controllers\Auth\LoginController@redirectToGoogle')->name('login.google');
Route::get('login/google/callback', 'App\Http\Controllers\Auth\LoginController@handleGoogleCallback');

Route::get('login/facebook', 'App\Http\Controllers\Auth\LoginController@redirectToFacebook')->name('login.facebook');
Route::get('login/facebook/callback', 'App\Http\Controllers\Auth\LoginController@handleFacebookCallback');

Route::get('login/github', 'App\Http\Controllers\Auth\LoginController@redirectToGithub')->name('login.github');
Route::get('login/github/callback', 'App\Http\Controllers\Auth\LoginController@handleGithubCallback');
...
```

6. Put URL to login buttons

```
...
<div class="form-group row">
    <div class="col-md-6 offset-md-3">
        <a href="{{ route('login.google') }}" class="btn btn-danger btn-block">Login with Google</a>
        <a href="{{ route('login.facebook') }}" class="btn btn-primary btn-block">Login with Facebook</a>
        <a href="{{ route('login.github') }}" class="btn btn-dark btn-block">Login with Github</a>
    </div>
</div>
```



```
</div>
```

```
...
```

7. Put ID and Secret Variable for each app (config/services.php<->.env)

.env

```
...
GOOGLE_CLIENT_ID=""
GOOGLE_CLIENT_SECRET=""

FACEBOOK_CLIENT_ID=""
FACEBOOK_CLIENT_SECRET=""

GITHUB_CLIENT_ID=""
GITHUB_CLIENT_SECRET=""
```

8. Change redirect for each app in ./config/services.php from the other routes in ./routes/web.php

./config/services.php

```
...
'github' => [
    'client_id' => env('GITHUB_CLIENT_ID'),
    'client_secret' => env('GITHUB_CLIENT_SECRET'),
    'redirect' => 'http://localhost:8003/login/github/callback',
],

'facebook' => [
    'client_id' => env('FACEBOOK_CLIENT_ID'),
    'client_secret' => env('FACEBOOK_CLIENT_SECRET'),
    'redirect' => 'http://localhost:8003/login/facebook/callback',
],

'google' => [
    'client_id' => env('GOOGLE_CLIENT_ID'),
    'client_secret' => env('GOOGLE_CLIENT_SECRET'),
    'redirect' => 'http://localhost:8003/login/google/callback',
],
...
```

./app/Http/Controllers/LoginController.php

```
...
// Google login redirect and callback
public function redirectToGoogle()
{
    return Socialite::driver('google')->redirect();
}
```

```

public function handleGoogleCallback()
{
    $user = Socialite::driver('google')->user();
    // $user->token;
}

// Facebook login redirect and callback
public function redirectToFacebook()
{
    return Socialite::driver('facebook')->redirect();
}

public function handleFacebookCallback()
{
    $user = Socialite::driver('facebook')->user();
    // $user->token;
}

// Github login redirect and callback
public function redirectToGithub()
{
    return Socialite::driver('github')->redirect();
}

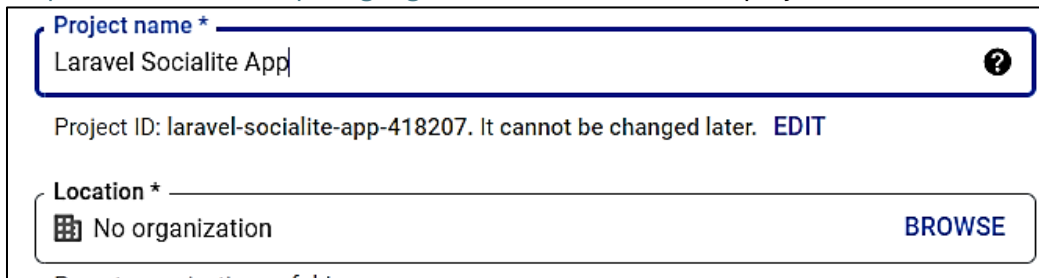
public function handleGithubCallback()
{
    $user = Socialite::driver('github')->user();
    // $user->token;
}
...

```

9. Create app for Google, Facebook and Github

For Google:

- Login to <https://console.developers.google.com/> and create a new project

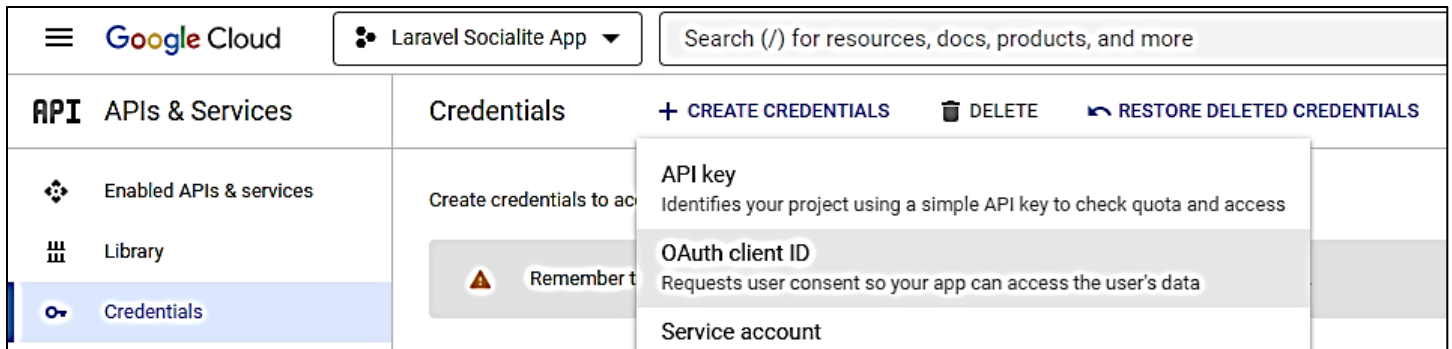


Project name * ?

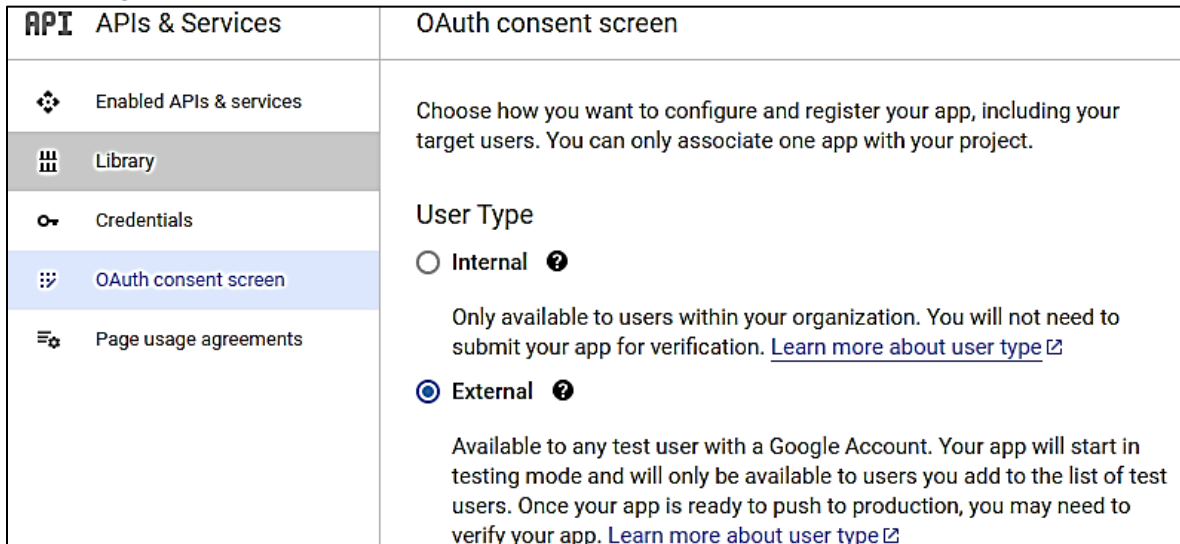
Project ID: **laravel-socialite-app-418207**. It cannot be changed later. [EDIT](#)

Location * [BROWSE](#)

- Once created, navigate to that project then go into “Credentials” (click API Overview->Credentials)
- Click “create credentials” → OAuth Client ID



- d. Click “configure consent screen” then select “External” and click create.



- e. Fill in App Name and Support Email then click **Save and Continue**

App information

This shows in the consent screen, and helps end users know who you are and contact you

App name *

Laravel Socialite App

The name of the app asking for consent

- f. Click back to credentials→create credentials→oauth client id→select application type and add redirect urls then click Create

←

Create OAuth client ID

A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, each will need its own client ID. See [Setting up OAuth 2.0](#) for more information. [Learn more](#) about OAuth client types.

Application type *

Web application

Name *

Web client 1

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

Authorized redirect URIs

For use with requests from a web server

URIs 1 *

http://localhost:8003/login/google/callback

g. Take note of Client ID and Secret

ⓘ

OAuth access is restricted to the [test users](#) listed on [OAuth consent screen](#)

| | |
|---------------|--|
| Client ID | 652999623378- ajb5e9ntgo0vrsvfcnomum5i s.googleusercontent.com |
| Client secret | GOCSPX-QPuF6V3Aw-GftYh JXcJD2 |

h. You may set these in the .env file

```
GOOGLE_CLIENT_ID="652999623378-ajb5e9ntaf0mhnn.apps.googleusercontent.com"
GOOGLE_CLIENT_SECRET="GOCSPX-QPuF3Aw-GftYh4oUJ9a-JXcJD2"
```

For Facebook

a. Navigate to <https://developers.facebook.com/apps> and click 'create app' → and select facebook login


Create an app

Add use case

App details


What do you want your app to do?

These are the most common use cases developers have used on Meta for Developers. Each use case has more functionality. Customize use cases once your app is created.

☒


Authenticate and request data from users with Facebook Login

Our most common use case. A secure, fast way for users to log into your app or website to ask for permissions to access their data to personalize their experience.

☐


Launch a game on Facebook

Launch a game that players can find and play directly in their Feed or messages on both desktop and mobile devices. [Learn more about Instant Games.](#)

- b. Add an app name then click 'Create App' (facebook login will popup)

Add an app name

This is the app name that will show on your My Apps page and in the Facebook app.

- c. Look for setting up facebook login as follows:

1. Customize this app

Add and customize the use cases you need for this app. You can add more even if this step is complete.

☒

Customize adding a Facebook Login button

☒

Explore and add more use cases





- d. Look for login using WWW

Use cases > Customize > Quickstart

Facebook Login quickstart

Customize your use case so your app works the way you need it to.

Use the Quickstart to add Facebook Login to your app. To get started, select the platform for this app.

iOS

Android

Web

Other

e. Enter site URL and click Save then click the Continue button

iOS

Android

Web

Other

1. Tell Us about Your Website

Tell us what the URL of your site is.

Site URL

Save

f. From there, you can navigate back to the Facebook App Dashboard→Use Case

Laravel Socialite App

App ID: 918225216668120

Dashboard

Required actions


Use cases

Review

Publish Unpublished

Use cases

Add and customize the use cases you want to add to your app. Some options are required for your app to work, and others are up to you. You can at any time, but keep in mind that adding more after your app has gone through App Review, or gone live, will require new reviews.



Authenticate and request data from users with Facebook Login
Our most common use case. A secure, fast way for users to log into your app or game and for the app to ask for permissions to access their data to personalize their experience.

Authentication and account creation
Added: Facebook Login, public_profile

Customize

Facebook Login settings

Customize your use case so your app works the way you need it to.



Easily add Facebook Login to your app with our Quickstart

Client OAuth settings

- ☒ Yes

Client OAuth login
Enables the standard OAuth client token flow. Secure your application and prevent abuse by locking down which token redirect URIs are allowed with the options below. Disable globally if not used. [?]
- ☒ Yes

Web OAuth login
Enables web-based Client OAuth Login. [?]
- ☐ No

Force Web OAuth reauthentication
When on, prompts people to enter their Facebook password in order to log in on the web. [?]
- ☒ Yes

Use Strict Mode for redirect URIs
Only allow redirects that exactly match the Valid OAuth Redirect URIs. Strongly recommended. [?]
- ☒ Yes

Enforce HTTPS
Enforce the use of HTTPS and the Java SDK recommendation. [?]
- ☐ No

Embedded Web OAuth Login
Enable web-based OAuth Login. [?]

Valid OAuth Redirect URIs

A manually specified redirect_uri used with Login on the web must exactly match one of the URIs listed here. This list is also used by the JavaScript SDK for in-app browsers that suppress popups. [?]

A manually specified redirect_uri used with Login on the web must exactly match one of the URIs listed here. This list is also used by the JavaScript SDK for in-app browsers that suppress popups. [?]

<http://localhost:8003/login/facebook/callback> ✕

[Copy to clipboard](#)

☐ **No** **Login from Devices**
Enables the OAuth client login flow for devices like a smart TV [?]

☐ **No** **Login with the JavaScript SDK**
Enables Login and signed-in with the JavaScript SDK. [?]

Allowed Domains for the JavaScript SDK
Login and signed-in functionality of the JavaScript SDK will only be available on these domains. [?]

Deauthorize

Deauthorize callback URL

Redirect URI Validator

Redirect URI to Check

g. Go back to facebook app dashboard → app settings → basic → get app id and app secret

Laravel Socialite App App ID: 918225216668120 ?

[Dashboard](#) [Required actions](#) [Use cases](#) [Review](#) [Publish](#) Unpublished

App ID

App secret
 [Show](#)

Display name

Namespace

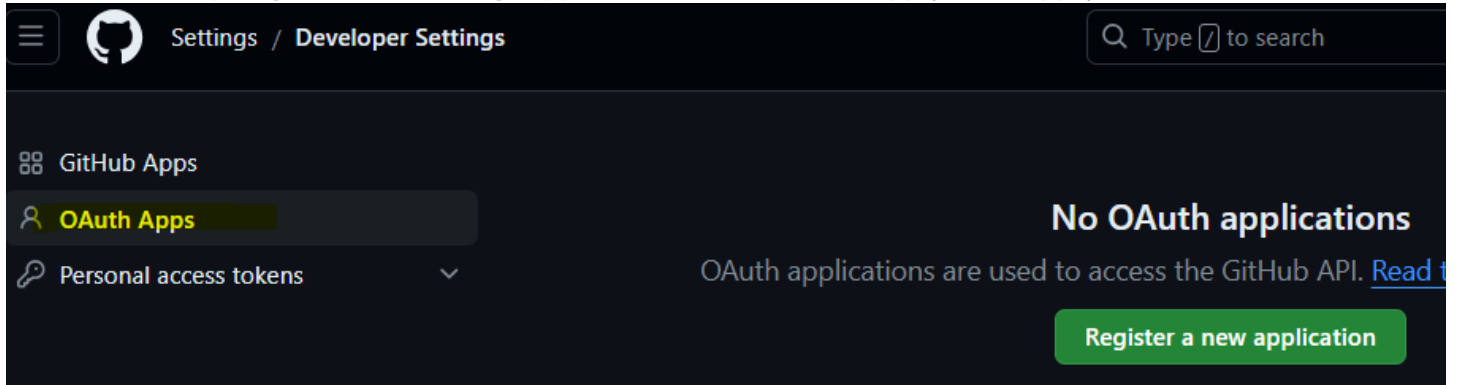
App domains

Contact email ⓘ

h. You may then set the ID and Secret in you .env file

```
FACEBOOK_CLIENT_ID="91822xxxxx8120"
FACEBOOK_CLIENT_SECRET="1e857b59xxxxxxxxxx17c30f80fde2783e0"
```


- a. Go to <https://github.com/settings/apps> and click New Github App (OAuth Apps)



- b. Set the following configurations (then click Register App:

Register a new OAuth application

Application name *

Something users will recognize and trust.

Homepage URL *

The full URL to your application homepage.

Application description

This is displayed to all users of your application.

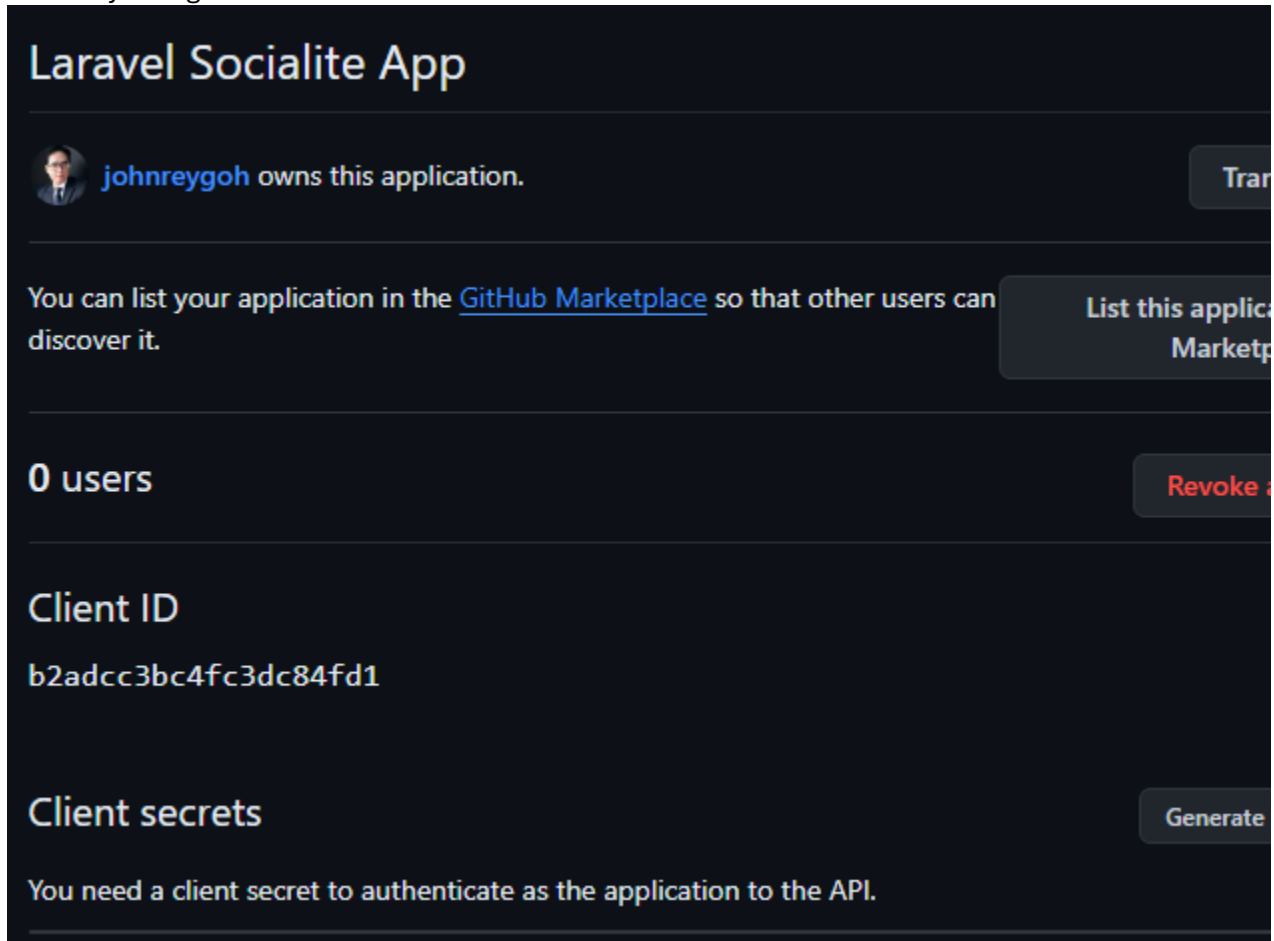
Authorization callback URL *

Your application's callback URL. Read our [OAuth documentation](#) for more information.

☐ **Enable Device Flow**

Allow this OAuth App to authorize users via the Device Flow.

- c. You may now get Client ID and Secret



- d. You may now set the client id and secret in your .env file

```
...
GITHUB_CLIENT_ID="b2adcc3bc4fc3dc84fd1"
GITHUB_CLIENT_SECRET="4b6ae95axxxxxxxxxxxxxx9bb05c143632296"
```

10. Back to your Laravel app, configure caching

```
\> php artisan config:cache
```

11. Add some extra fields to Users Table

./database/migrations/create_users_table

```
...
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->string('provider_id')->nullable();
    $table->string('avatar')->nullable();
    $table->timestamp('email_verified_at')->nullable();
    $table->string('password')->nullable;
    $table->rememberToken();
});
```

```
$table->timestamps();
});
...
```

12. Run the migration

```
\> php artisan migrate:fresh
```

*setup your mysql for this

*don't forget to revisit your .env for database connection

13. Update your User model

./app/Models/User.php

```
...
protected $fillable = [
    'name',
    'email',
    'provider_id',
    'avatar',
    'password',
];
...
```

14. Add a method that creates new user or login existing user

./app/Http/Controllers/auth/LoginController.php

```
...
<?php
...

class LoginController extends Controller
{

    // Google login redirect and callback
    public function redirectToGoogle()
    {
        return Socialite::driver('google')->redirect();
    }

    public function handleGoogleCallback()
    {
        $user = Socialite::driver('google')->user();
        $this->_registerOrLoginUser($user);
        return redirect()->route('home');
    }

    // Facebook login redirect and callback
    public function redirectToFacebook()
```

```

{
    return Socialite::driver('facebook')->redirect();
}

public function handleFacebookCallback()
{
    $user = Socialite::driver('facebook')->user();
    $this->_registerOrLoginUser($user);
    return redirect()->route('home');
}

// Github login redirect and callback
public function redirectToGithub()
{
    return Socialite::driver('github')->redirect();
}

public function handleGithubCallback()
{
    $user = Socialite::driver('github')->user();
    $this->_registerOrLoginUser($user);
    return redirect()->route('home');
}

protected function _registerOrLoginUser($data){
    $user = User::where('email','=',$data->email)->first();
    if(!$user){
        $user = new User();
        $user->name = $data->name;
        $user->email = $data->email;
        $user->provider_id = $data->id;
        $user->avatar = $data->avatar;
        $user->save();
    }

    Auth::login($user);
}

}
...

```

15. Add user avatar beside name

./resources/views/layouts/app.blade.php

```

...
<li class="nav-item dropdown">
    name }}"
        style="border:1px solid #cccccc;

```

```

        border-radius:5px;
        width:39px;
        height:auto;
        float:left;
        margin-right:7px">
        <a id="navbarDropdown" class="nav-link dropdown-toggle" href="#" role="button" data-
toggle="dropdown" aria-haspopup="true" aria-expanded="false" v-pre>
        {{ Auth::user()->name }}
        </a>
...

```

16. Test SSO <http://localhost:8003/login>

Extra Topic: Whitelisting IP Address using a custom middleware in Laravel 10

Creating a Whitelist Middleware for IP Addresses in Laravel 10

App\Http\Middleware\CheckIP.php

```

<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;

class CheckIP
{
    /**
     * Handle an incoming request.
     *
     * @param \Illuminate\Http\Request
     $request
     * @param \Closure $next
     * @return mixed
     */
    public function handle(Request $request, Closure $next)
    {
        // Define allowed IP addresses (adjust as needed)
        $allowedIps = [
            '192.168.1.100',
            '10.0.0.200',
        ];

        // Get the client's IP address
        $clientIp = $request->ip();

        // Check if the IP address is allowed
        if (!in_array($clientIp, $allowedIps)) {

```

```

        // Return a response indicating unauthorized access (customize as needed)
        return response('Unauthorized access', 401);
    }

    return $next($request);
}
}

```

Explanation:

- Import necessary classes: We import Closure and Illuminate\Http\Request for handling requests and responses.
- Define allowed IP addresses: Create an array of allowed IP addresses. Replace these with your desired IP addresses.
- Get client's IP address: Use \$request->ip() to retrieve the client's IP address.
- Check IP against whitelist: Iterate over the allowed IP addresses and check if the client's IP matches any of them.
- Handle unauthorized access: If the IP is not allowed, return an unauthorized response. Customize this response as needed (e.g., redirect to a specific page).
- Allow access: If the IP is allowed, proceed to the next middleware or the route handler using \$next(\$request).

Registering the Middleware:

To apply this middleware to specific routes or globally, you can use the kernel.http middleware group:

```

// app/Http/Kernel.php
protected $middleware = [
    // ... other middleware
    \App\Http\Middleware\CheckIP::class,
];

```

Or apply it to specific routes in your routes/web.php file:

```

Route::middleware(['check_ip'])->group(function () {
    // Routes that require IP whitelisting
    Route::get('/dashboard', function () {
        // ...
    });
});

```

Additional Considerations:

- ✓ For more complex scenarios, you might want to store the allowed IP addresses in a database or configuration file.
- ✓ To handle different environments (development, staging, production), you can use environment variables to store IP addresses.
- ✓ Consider using Laravel's rate limiting features for additional security measures.