Connor Leech Follow
I work at Stitch Labs and build https://employbl.com/
Jun 15 · 4 min read

Add Login with Google to your Laravel app

Laravel Socialite handles login with OAuth and is officially maintained by the Laravel team.

In this tutorial we're going to add authentication via Google to a Laravel app. We're going to use Socialite and start from scratch.

1. Create Laravel app

Create a new laravel app with a database and get everything up and running.

2. Install Socialite

Install socialite using composer. Socialite is an official Laravel package documented here.

3. Configure Laravel

Add credentials to **config/services.php.** Socialite supports Facebook,

Twitter, LinkedIn, Google, GitHub and Bitbucket. Other providers require packages from the community, which are all listed here.

These providers follow the OAuth 2.0 spec and therefore require a **client_id**, **client_secret** and **redirect** url. We'll obtain these in the next step! First, add the values to the config file because socialite will be looking for them when we ask it to.

```
'google' => [
    'client_id'     => env('GOOGLE_CLIENT_ID'),
    'client_secret' => env('GOOGLE_CLIENT_SECRET'),
    'redirect'      => env('GOOGLE_REDIRECT')
],
```

Since we added a new package, make sure to add to the service providers array in **config/app.php**:

```
/*
* Package Service Providers...
*/
Laravel\Socialite\SocialiteServiceProvider::class,
```

Service Providers are the central place for application bootstrapping. The above line let's Laravel to know to make the Socialite available for use.

Add an alias to Socialite so it is easier to reference later, also in **config/app.php** file:

```
'aliases' => [
    // ...
    'Socialite' => Laravel\Socialite\Facades
\Socialite::class,
]
```

4. Create the basic authentication scaffold

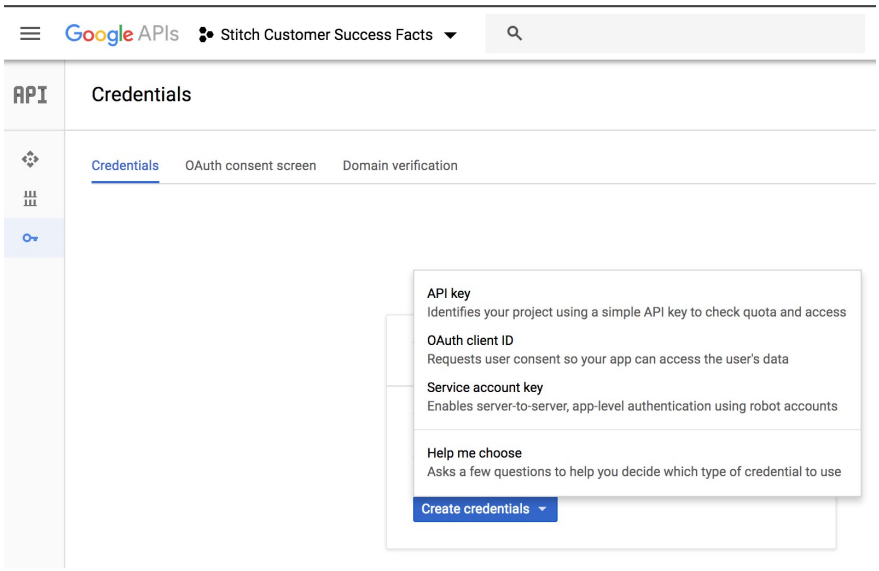
```
php artisan make:auth
```

This will create a user model and some views for register and login pages.

5. Create your app in Google

Create a project: https://console.developers.google.com/projectcreate

Create credentials: https://console.developers.google.com/apis/credentials

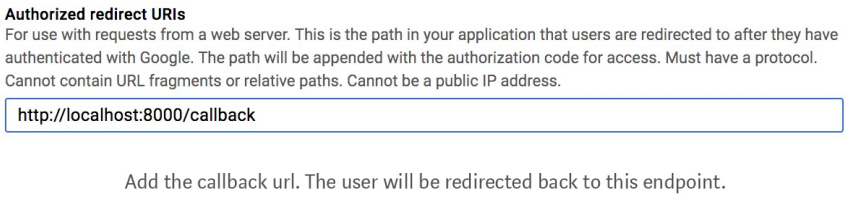


Select OAuth client ID > Consent > Web application

A modal will pop up with your apps client id and client secret. Add these values to your **.env** file.

```
GOOGLE_CLIENT_ID=000000000000-
```

```
XXXXXXXXXXX.apps.googleusercontent.com
GOOGLE_CLIENT_SECRET=XXXXXXXXXXXXX
GOOGLE_REDIRECT=http://localhost:8000/callback
```



Enable the Google+ API: https://console.cloud.google.com /apis/api/plus.googleapis.com/overview (This tells Google what services our application is going to use ie Google+ account login)

6. Handle your routes 🔀

Head into **routes/web.php** and add endpoints for redirect and callback:

```
Route::get('/redirect',
'Auth\LoginController@redirectToProvider');
Route::get('/callback',
'Auth\LoginController@handleProviderCallback');
```

The first method will show the Google authentication page in the same window where the user is viewing your webpage (no annoying popups):

```
/**
  * Redirect the user to the Google authentication page.
  *
  * @return \Illuminate\Http\Response
  */
public function redirectToProvider()
{
    return Socialite::driver('google')->redirect();
}
```

The next method will handle after a successful Google authentication:

```
    /**
     * Obtain the user information from Google.
     *
     * @return \Illuminate\Http\Response
     */
    public function handleProviderCallback()
    {
        try {
            $user = Socialite::driver('google')->user();
        } catch (\Exception $e) {
            return redirect('/login');
        }

        // only allow people with @company.com to login
        if(explode("@", $user->email)[1] !== 'company.com'){
            return redirect()->to('/');
        }

        // check if they're an existing user
        $existingUser = User::where('email',
$user->email)->first();

        if($existingUser){
            // log them in
            auth()->login($existingUser, true);
        } else {
            // create a new user
            $newUser                  = new User;
            $newUser->name            = $user->name;
            $newUser->email           = $user->email;
            $newUser->google_id       = $user->id;
            $newUser->avatar          = $user->avatar;
            $newUser->avatar_original =
$user->avatar_original;
            $newUser->save();

            auth()->login($newUser, true);
        }
        return redirect()->to('/home');
    }
```

Right now this will authenticate the user and dump the info we get back on the screen. If you run this you can see we get a **token**, **refreshToken**, **name**, **email**, **avatar**, **user object**, **avatar** and **avatar_original** field.

7. Update the users table migration

Update your **create_users_table** migration to include these new fields. You could alternatively create a new migration for adding theses columns, which would be good for an existing app:

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table)
{
            $table->increments('id');
            $table->string('name');
            $table->string('google_id');
            $table->string('email')->unique();
            $table->string('password')->nullable();
            $table->string('avatar')->nullable();
            $table->string('avatar_original')->nullable();
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

That's all folks!

. . .