Laravel and JWT

In this article, we will look at using JWT to secure our Laravel APIs.

JSON Web Token (JWT) is an open standard that allows two parties to securely send data and information as JSON objects. This information can be verified and trusted because it is digitally signed.

JWT authentication has aided the wider adoption of stateless API services. It makes it convenient to authorise and verify clients accessing API resources. It is a critical part of the authentication system in javascript powered applications.

Prerequisites

1. Knowledge of PHP
2. Knowledge of Laravel
3. Have composer and Laravel installer installed
4. Knowledge of git
5. Have and know how to use postman

Getting Started

The first thing we are going to do is create a laravel application for testing JWT. If you have the Laravel installer, you can run the following command:

```
$ laravel new laravel-jwt
```

If you do not have the Laravel installer, you can get it by running the following command:

```
$ composer global require "laravel/installer"
```

After creating `laravel-jwt`, navigate into the directory and install the third-party JWT package we will use. Due to an issue with the published version of `tymon/jwt-auth`, we are going to install the dev version of the package. Run the following command:

```
$ composer require tymon/jwt-auth:dev-develop --prefer-source
```

Open `config/app.php` and add the following provider to the providers array:

```
[...]
Tymon\JWTAuth\Providers\LaravelServiceProvider::class,
[...]
```

Add the following facades to the aliases array:

```
[...]
'JWTAuth' => Tymon\JWTAuth\Facades\JWTAuth::class,
'JWTFactory' => Tymon\JWTAuth\Facades\JWTFactory::class,
[...]
```

You need to publish the config file for JWT using the following command:

```
$ php artisan vendor:publish --provider="Tymon\JWTAuth\Providers\LaravelServiceProvider"
```

When that is done, set the `jwt-auth` secret by running the following command:

```
$ php artisan jwt:secret
```

> If you read other articles out there on JWT, you may see configurations for the published config/jwt.php file. Understand that many of the configurations may be for v0.5. This dev version we used will be compatible with the release of stable v1 of tymon/jwt. Namshi/jwt has been deprecated, so make use of Lcobucci/jwt.

We need to make the User model implement JWT. Open `app/User.php` file and replace the content with this:

```
<?php

namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Tymon\JWTAuth\Contracts\JWTSubject;

class User extends Authenticatable implements JWTSubject
{
    use Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'email', 'password',
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password', 'remember_token',
    ];

    public function getJWTIdentifier()
    {
        return $this->getKey();
    }
    public function getJWTCustomClaims()
    {
        return [];
    }
}
```

We have defined the User model to implement `JWTSubject`. We also defined two methods to return the `JWTIdentifier` and `JWTCustomClaims`. Custom claims are used in generating the JWT token.

That concludes the installation of JWT. Let us proceed to set up the rest of our application.

Setup the database

For this guide, we will use an SQLite database. Create the database file as follows:

```
$ touch database/database.sqlite
```

When that is done, open the `.env` file and edit the database settings. Replace:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=homestead
DB_USERNAME=homestead
DB_PASSWORD=secret
```

with

```
DB_CONNECTION=sqlite
DB_DATABASE=/absolute/path/to/database.sqlite
```

Laravel comes with default migration for user's table. We will not need any columns different from what it provides. Run the migrate command to create the table on the database:

```
$ php artisan migrate
```

Our database is ready now.

Create the controllers

We are going to create two controllers for this guide: `UserController` and `DataController`.

The `UserController` will hold all our authentication logic, while the `DataController` will return sample data.

Create the controllers:

```
$ php artisan make:controller UserController
$ php artisan make:controller DataController
```

Open the `UserController` file and edit as follows:

```
<?php

namespace App\Http\Controllers;

use App\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Validator;
use JWTAuth;
use Tymon\JWTAuth\Exceptions\JWTException;

class UserController extends Controller
{
    public function authenticate(Request $request)
    {
        $credentials = $request->only('email', 'password');

        try {
            if (! $token = JWTAuth::attempt($credentials)) {
                return response()->json(['error' => 'invalid_credentials'], 400);
            }
        } catch (JWTException $e) {
            return response()->json(['error' => 'could_not_create_token'], 500);
        }

        return response()->json(compact('token'));
    }

    public function register(Request $request)
    {
            $validator = Validator::make($request->all(), [
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:users',
            'password' => 'required|string|min:6|confirmed',
        ]);

        if($validator->fails()){
                return response()->json($validator->errors()->toJson(), 400);
        }

        $user = User::create([
            'name' => $request->get('name'),
            'email' => $request->get('email'),
            'password' => Hash::make($request->get('password')),
        ]);

        $token = JWTAuth::fromUser($user);

        return response()->json(compact('user','token'),201);
    }

    public function getAuthenticatedUser()
        {
                try {

                        if (! $user = JWTAuth::parseToken()->authenticate()) {
                                return response()->json(['user_not_found'], 404);
                        }

                } catch (Tymon\JWTAuth\Exceptions\TokenExpiredException $e) {

                        return response()->json(['token_expired'], $e->getStatusCode());

                } catch (Tymon\JWTAuth\Exceptions\TokenInvalidException $e) {

                        return response()->json(['token_invalid'], $e->getStatusCode());

                } catch (Tymon\JWTAuth\Exceptions\JWTException $e) {

                        return response()->json(['token_absent'], $e->getStatusCode());

                }

                return response()->json(compact('user'));
```

```
        }
    }
```

The `authenticate` method attempts to log a user in and generates an authorization token if the user is found in the database. It throws an error if the user is not found or if an exception occurred while trying to find the user.

The `register` method validates a user input and creates a user if the user credentials are validated. The user is then passed on to `JWTAuth` to generate an access token for the created user. This way, the user would not need to log in to get it.

We have the `getAuthenticatedUser` method which returns the user object based on the authorization token that is passed.

Now, let us create sample data in the `DataController`:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class DataController extends Controller
{
        public function open()
        {
            $data = "This data is open and can be accessed without the client being authenticated";
            return response()->json(compact('data'),200);

        }

        public function closed()
        {
            $data = "Only authorized users can see this";
            return response()->json(compact('data'),200);
        }
}
```

Next thing is to make the API routes to test the JWT setup.

Creating our routes

Before we define our API routes, we need to create a `JwtM``iddleware` which will protect our routes. Run this command via your terminal.

```
$ php artisan make:middleware JwtMiddleware
```

This will create a new middleware file inside our Middleware directory. This file can be located here `app/Http/Middleware/JwtMiddleware`. Open up the file and replace the content with the following:

```
<?php

namespace App\Http\Middleware;

use Closure;
use JWTAuth;
use Exception;
use Tymon\JWTAuth\Http\Middleware\BaseMiddleware;

class JwtMiddleware extends BaseMiddleware
{

    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        try {
            $user = JWTAuth::parseToken()->authenticate();
        } catch (Exception $e) {
            if ($e instanceof \Tymon\JWTAuth\Exceptions\TokenInvalidException){
                return response()->json(['status' => 'Token is Invalid']);
            }else if ($e instanceof \Tymon\JWTAuth\Exceptions\TokenExpiredException){
                return response()->json(['status' => 'Token is Expired']);
            }else{
                return response()->json(['status' => 'Authorization Token not found']);
            }
        }
        return $next($request);
    }
}
```

This middleware extends `Tymon\JWTAuth\Http\Middleware\BaseMiddleware`, with this, we can catch token errors and return appropriate error codes to our users.

Next, we need to register our middleware. Open `app/http/Kernel.php` and add the following:

```
[...]
protected $routeMiddleware = [
    [...]
    'jwt.verify' => \App\Http\Middleware\JwtMiddleware::class,
];
[...]
```

Next, Open `routes/api.php` and add the content with the following:

```
    [...]
    Route::post('register', 'UserController@register');
    Route::post('login', 'UserController@authenticate');
    Route::get('open', 'DataController@open');

    Route::group(['middleware' => ['jwt.verify']], function() {
        Route::get('user', 'UserController@getAuthenticatedUser');
        Route::get('closed', 'DataController@closed');
    });
```

We defined all the routes we need to test out JWT. Every route we do not wish to secure is kept outside the JWT middleware.

> Use this command to start your server through the terminal: `php artisan serve`

Test on postman

Postman is an application that makes API development easy. It provides the necessary environment required to test APIs as you develop them. If you do not have postman, you can get it from here.

Create a user account for testing

Endpoint : `127.0.0.1:8000/api/register`
Method: `POST`
Payload:

`name`: Test Man
`email`: test@email.com
`password`: secret
`password_confirmation`: secret

User login

Endpoint : `127.0.0.1:8000/api/login`
Method: `POST`
Payload:

`email`: test@email.com
`password`: secret

Accessing an unprotected route

Endpoint : `127.0.0.1:8000/api/open`
Method: `GET`

Access a protected endpoint

Endpoint : `127.0.0.1:8000/api/open`
Method: `GET`
Payload:

`Authorization`: Bearer *insert_user_token_here*

Try to access the data protected by the middleware using the authorization token.

Get the authenticated user data

Endpoint : `127.0.0.1:8000/api/user`
Method: `GET`
Payload:

`Authorization`: Bearer *insert_user_token_here*

Use invalid token to access a users data

Endpoint : `127.0.0.1:8000/api/user`
Method: `GET`
Payload:

`Authorization`: Bearer thistokeniswrong

Accessing a protected route without a token

Endpoint : `127.0.0.1:8000/api/closed`
Method: `GET`

Conclusion

In this guide, we have looked JWT and how to use it for our Laravel application. We set up a controller for user authentication and registration. We also defined a method for getting the authenticated user using the generated token. We saw how it is used to secure our APIs and tested the output data using Postman.

You can use JWT to secure your API endpoints that different clients will access. JWT is a convenient way to authenticate users.

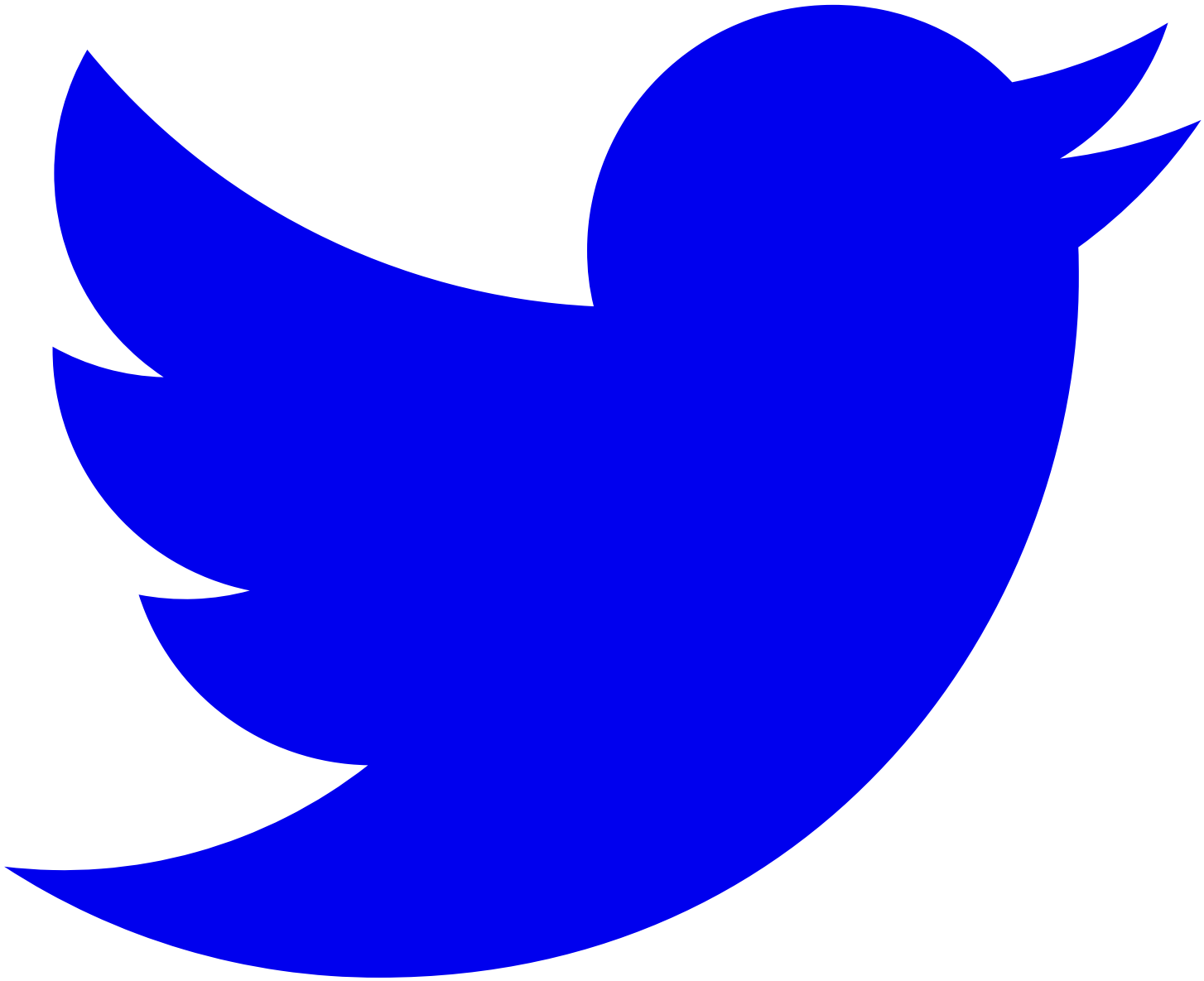
The source code to the application in this article is available on GitHub.

authentication laravel php postman security

June 25, 2018

Fisayo Afolayan

Share article



Ready to begin?

Start building your realtime experience today.

From in-app chat to realtime graphs and location tracking, you can rely on Pusher to scale to million of users and trillions of messages

Sign up for free Contact us

PUSHER

A MessageBird company

Products

- Channels
- Beams

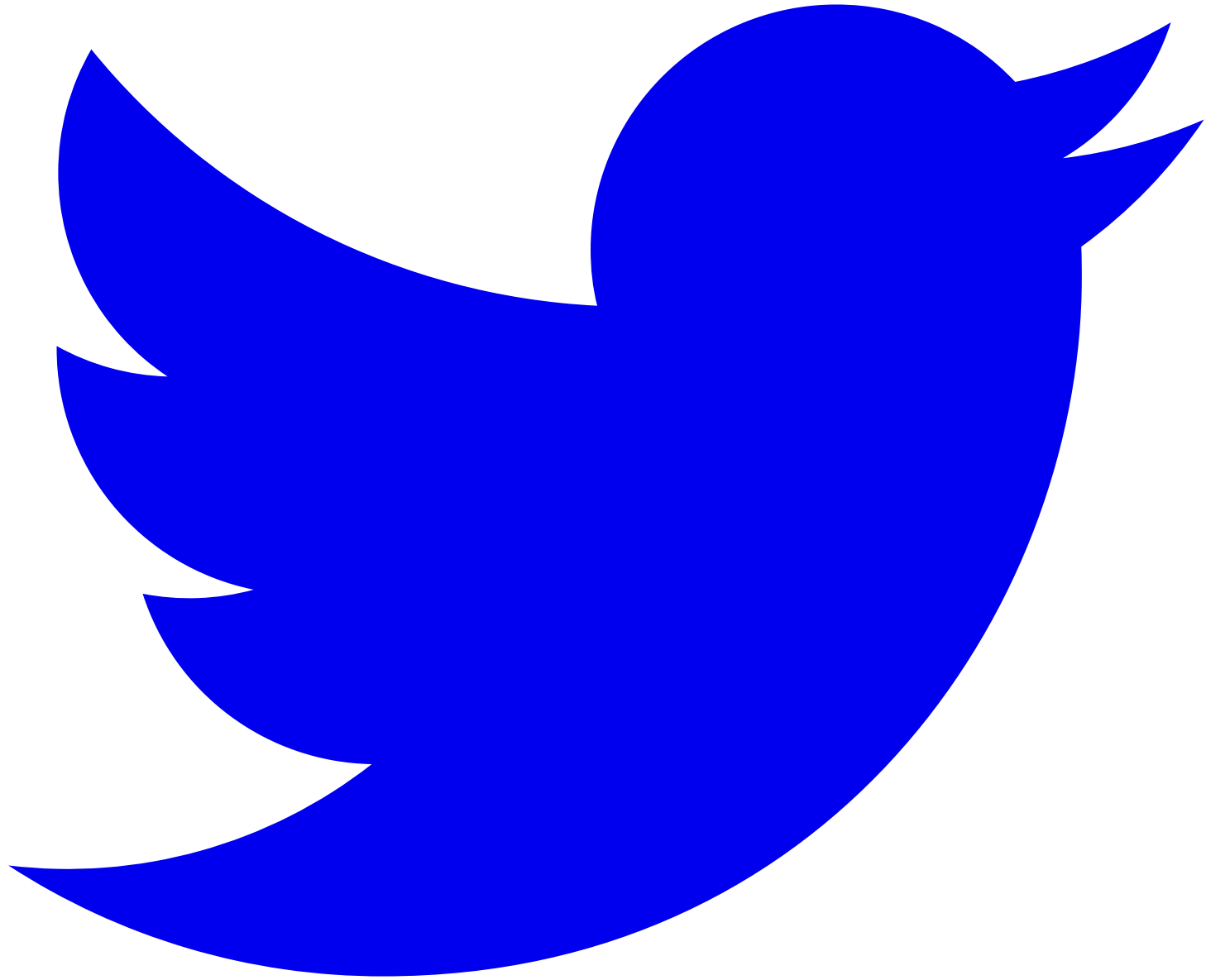
Developers

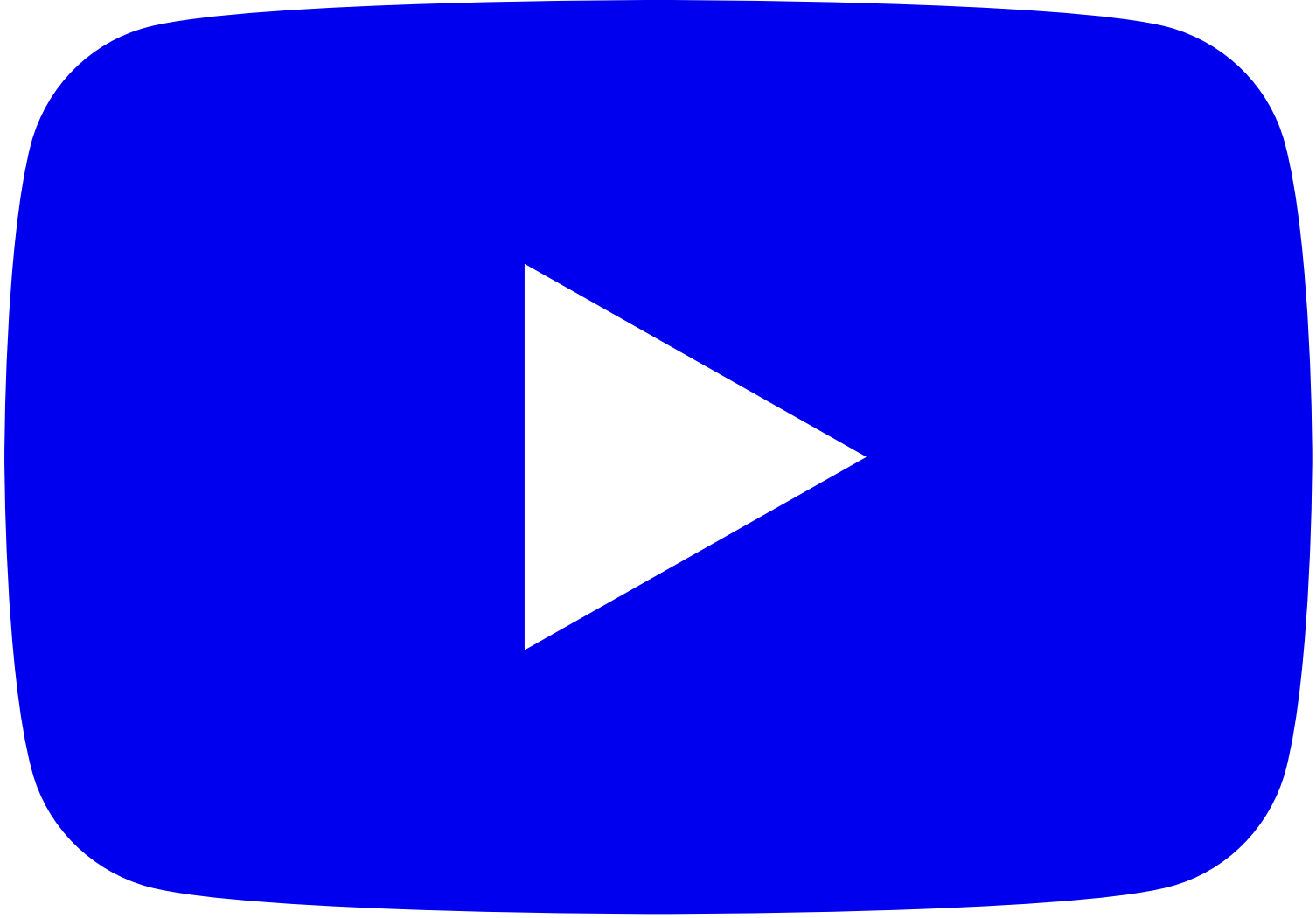
- Docs
- Tutorials
- Status
- Support
- Sessions

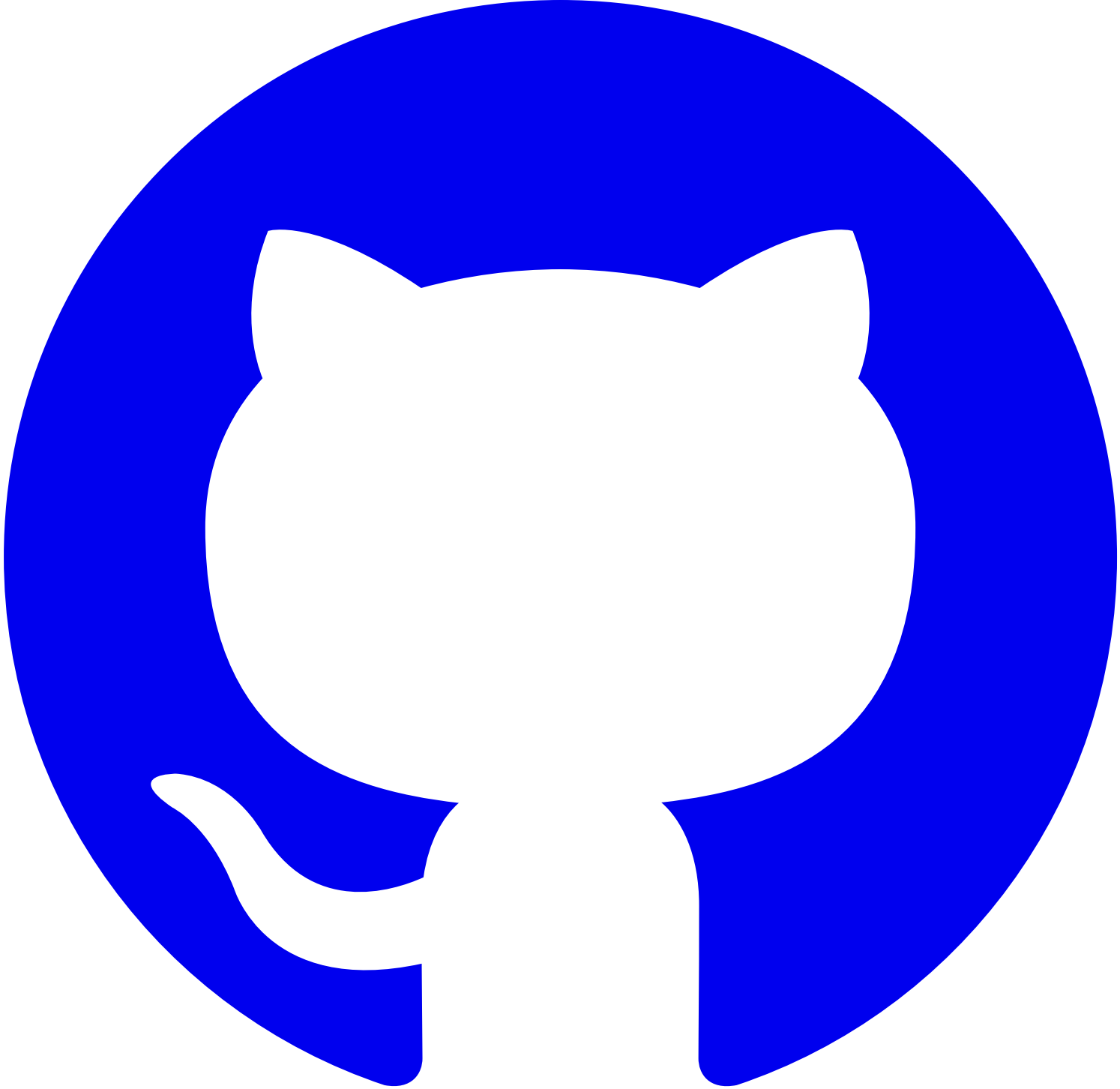
Company

- Contact Sales
- Customer stories
- Terms of Service
- Security
- Careers
- Blog
- Legal

Connect







Pusher Limited is a company registered in England and Wales (No. 07489873) whose registered office is at Eighth Floor 6 New Street Square, New Fetter Lane, London, England, EC4A 3AQ.