1v0 Ship v1.0 Faster

# Using Laravel Passport for Advanced API Authentication

## Using Laravel Passport for Advanced API Authentication

**Laravel Passport** brings a full OAuth2 server to your Laravel app—great when you need first-class features beyond simple token auth: password grants, client credentials (machine-to-machine), refresh tokens, and *scopes* for fine-grained permissions. In this guide you'll install Passport, wire up guards, issue tokens via multiple grants, enforce scopes, and add a tiny UI to test flows quickly.

## 1 – Install Passport & Bootstrap Keys

Add Passport to your project and generate encryption keys and default clients (password & personal access).

```
composer require laravel/passport

php artisan migrate
php artisan passport:installCode language: Bash (bash)
```

`passport:install` creates RSA keys and seeds OAuth clients into `oauth_clients`. You'll get a *Password Grant Client* (for username/password token exchange), and a *Personal Access Client* for user-issued long-lived tokens (developer tooling / first-party use).

## 2 – Configure Auth Guard for APIs

Use Passport as the driver for the `api` guard so `auth:api` middleware validates OAuth2 access tokens.

```
// config/auth.php (snippet)
'guards' => [
    'web' => [
        'driver' => 'session',
        'provider' => 'users',
    ],
    'api' => [
        'driver' => 'passport',
        'provider' => 'users',
    ],
],Code language: PHP (php)
```

Switching the `api` guard to the `passport` driver makes `auth:api` and Passport's scope middleware work out of the box for routes under `routes/api.php`.

```
// app/Providers/AuthServiceProvider.php (snippet)
use Laravel\Passport\Passport;

public function boot(): void
{
    $this->registerPolicies();

    Passport::routes(); // registers /oauth/* routes

    // Optional: token expiry & scopes
    Passport::tokensExpireIn(now()->addHours(2));
    Passport::refreshTokensExpireIn(now()->addDays(30));
```

```
    Passport::tokensCan([
        'read-posts'  => 'Read posts data',
        'write-posts' => 'Create or update posts',
        'admin'       => 'Full administrative access',
    ]);
}Code language: PHP (php)
```

`Passport::routes()` exposes the OAuth endpoints (`/oauth/token`, `/oauth/authorize`, etc.). You can tune lifetimes and define named *scopes* describing which capabilities a token grants.

## 3 – Issuing Tokens via Password Grant

The **Password Grant** lets first-party apps exchange a user's email+password for an access + refresh token pair. Store the client ID/secret in env.

OAUTH_PASSWORD_CLIENT_ID=3 OAUTH_PASSWORD_CLIENT_SECRET=your-password-client-secret

Find these values from the output of `passport:install` (or in the `oauth_clients` table where `password_client = 1`). Keep the secret confidential.

```
// app/Http/Controllers/OAuth/PasswordGrantController.php
namespace App\Http\Controllers\OAuth;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Http;

class PasswordGrantController
{
    public function token(Request $request)
    {
        $request->validate([
```

```
            'username' => ['required','email'],
            'password' => ['required'],
            'scope'    => ['nullable','string'], // e.g. "read-posts
write-posts"
        ]);

        $payload = [
            'grant_type'    => 'password',
            'client_id'     =>
config('services.passport.password_client_id'),
            'client_secret' =>
config('services.passport.password_client_secret'),
            'username'      => $request->username,
            'password'      => $request->password,
            'scope'         => $request->input('scope',''),
        ];

        $response = Http::asForm()->post(url('/oauth/token'),
$payload);

        return response()->json($response->json(),
$response->status());
    }
}Code language: PHP (php)
```

This endpoint proxies your request to Passport's `/oauth/token` and returns the OAuth2 response (access token, refresh token, expires_in). Limit this to trusted first-party clients; never expose the password client secret to browsers you don't control.

```
// config/services.php (snippet)
'passport' => [
    'password_client_id'     => env('OAUTH_PASSWORD_CLIENT_ID'),
    'password_client_secret' => env('OAUTH_PASSWORD_CLIENT_SECRET'),
],

// routes/api.php (snippet)
use App\Http\Controllers\OAuth\PasswordGrantController;
Route::post('/oauth/password/token', [PasswordGrantController::class,
```

```
'token']);Code language: PHP (php)
```

Config values centralize secrets and keep controllers clean. The route exposes a first-party-only token exchange endpoint your SPA/mobile client can call via your backend.

## 4 – Client Credentials (M2M) Tokens

For server-to-server integrations (no user), use the **Client Credentials** grant. Create a client and use its ID/secret to fetch tokens.

```
php artisan passport:client --client
# copy the client_id and client_secretCode language: Bash (bash)
```

This creates a confidential client for machines. Those tokens won't carry a user, so your API should check capabilities accordingly (e.g., scoped to service roles only).

```
// Example: validating a client-credentials call in a controller
// $request->user() will be null; rely on token scopes/claims
public function m2mEndpoint(Request $request)
{
    // Gate access by scope via middleware or manual check
    return ['status' => 'ok'];
}Code language: PHP (php)
```

Because there's no user context, don't assume `$request->user()` exists. Protect these routes with scope middleware to limit what the machine can do.

## 5 – Protecting Routes & Enforcing Scopes

Use Passport's middleware to require a valid token and specific scopes. You can combine multiple scopes for least-privilege access.

```
// routes/api.php
use App\Http\Controllers\PostApiController;

Route::middleware(['auth:api', 'scopes:read-posts'])->get('/posts',
[PostApiController::class, 'index']);

Route::middleware(['auth:api', 'scopes:write-posts'])->post('/posts',
[PostApiController::class, 'store']);

Route::middleware(['auth:api',
'scopes:admin'])->delete('/posts/{post}', [PostApiController::class,
'destroy']);Code language: PHP (php)
```

`auth:api` verifies the token signature and expiration. The `scopes:` middleware ensures the token contains the required scope(s). Tokens without the scope will receive `403` responses.

## 6 – Refresh Tokens & Rotation

Clients should refresh before access tokens expire. Passport supports refresh tokens out of the box via the `refresh_token` grant.

```
# Example POST to /oauth/token (form-encoded)
grant_type=refresh_token
client_id=...
client_secret=...
refresh_token=... (from previous token response)Code language: Bash (bash)
```

On success, Passport returns a new access token (and often a new refresh token). Store tokens securely (never in localStorage for sensitive apps—prefer HTTP-only cookies or native secure storage on mobile).

## 7 – Personal Access Tokens (Developer UX)

Users can manually create long-lived tokens to use in tools like Postman. Add simple endpoints to mint / revoke them with chosen scopes.

```
// app/Http/Controllers/PersonalTokenController.php
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PersonalTokenController extends Controller
{
    public function create(Request $request)
    {
        $request->validate([
            'name'  => ['required','string','max:60'],
            'scopes' => ['array'],
            'scopes.*' => ['string'],
        ]);

        $token = $request->user()->createToken(
            $request->name,
            $request->input('scopes', []) // e.g. ['read-posts']
        );

        return ['token' => $token->accessToken];
    }

    public function revoke(Request $request)
```

```
    {
        $request->user()->token()->revoke();
        return ['status' => 'revoked'];
    }
}Code language: PHP (php)
```

`createToken()` issues a personal access token for the authenticated user with the requested scopes. This is convenient for CLI scripts and power users; still apply least-privilege scopes.

## 8 – UI: OAuth Token Playground (Password Grant)

Here's a tiny Blade UI that exchanges email/password for an access token, then calls a protected endpoint with scopes.

```
<!-- resources/views/oauth/playground.blade.php -->
@extends('layouts.app')

@section('content')
<div class="container">
  <h1>OAuth Token Playground (Passport)</h1>

  <div class="row g-3">
    <div class="col-md-4"><input id="email" class="form-control"
placeholder="Email"></div>
    <div class="col-md-4"><input id="password" class="form-control"
type="password" placeholder="Password"></div>
    <div class="col-md-4"><input id="scope" class="form-control"
placeholder="Scopes (space-separated), e.g. read-posts"></div>
  </div>

  <button class="btn btn-theme mt-3" onclick="getToken()">Get
Token</button>
```

```
  <button class="btn btn-secondary mt-3 ms-2" onclick="callApi()">Call
/api/posts</button>

  <pre id="out" class="mt-3"></pre>
</div>

<script
src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
<script>
let token = null;

function getToken() {
  axios.post('/api/oauth/password/token', {
    username: document.getElementById('email').value,
    password: document.getElementById('password').value,
    scope: document.getElementById('scope').value
  }).then(res => {
    token = res.data.access_token;
    document.getElementById('out').textContent =
JSON.stringify(res.data, null, 2);
  }).catch(err => {
    document.getElementById('out').textContent = err.response ?
JSON.stringify(err.response.data, null, 2) : err;
  });
}

function callApi() {
  if (!token) return alert('Get a token first');
  axios.get('/api/posts', { headers: { Authorization: `Bearer
${token}` }})
    .then(res => document.getElementById('out').textContent =
JSON.stringify(res.data, null, 2))
    .catch(err => document.getElementById('out').textContent =
err.response ? JSON.stringify(err.response.data, null, 2) : err);
}
</script>
@endsectionCode language: PHP (php)
```

The page first hits your password-grant proxy to fetch an access token (optionally with scopes), then uses it to call a scoped, protected API. This is perfect for quick verification without Postman.

## Wrapping Up

Passport equips Laravel with a full OAuth2 server: password and client-credentials grants, refresh tokens, personal access tokens, and scope-based authorization. You set up guard integration, issued tokens securely, enforced scopes on routes, and built a tiny UI to test flows. Choose Passport when you need interoperable OAuth2 with multiple client types and granular permissions; for simpler first-party apps, Sanctum may suffice.

## What’s Next

- How to Add JWT Authentication to Laravel APIs
- How to Build a Multi-Auth API with Laravel & Sanctum
- Integrating Laravel with Third-Party APIs (Mail, SMS, Payment)