How to Build a Multi-Auth API with Laravel & Sanctum

How to Build a Multi-Auth API with Laravel & Sanctum

Sometimes an app needs different authentication flows for different user types — for example, *customers* and *admins*. With Laravel Sanctum, you can build a **multi-auth API** by defining multiple guards and issuing tokens with different abilities. In this guide, you'll configure Sanctum for multi-auth, create endpoints for both user roles, and secure them appropriately.

1 – Configure Guards in auth.php

By default, Laravel has `web` and `api` guards. You can define extra guards for custom roles like `admin`.

```
// config/auth.php (snippet)
'guards' => [
    'web' => [
        'driver' => 'session',
        'provider' => 'users',
    ],

    'api' => [
        'driver' => 'sanctum',
        'provider' => 'users',
    ],

    'admin' => [
        'driver' => 'sanctum',
        'provider' => 'admins',
    ],
```

```
],

'providers' => [
    'users' => [
        'driver' => 'eloquent',
        'model' => App\Models\User::class,
    ],
    'admins' => [
        'driver' => 'eloquent',
        'model' => App\Models\Admin::class,
    ],
],Code language: PHP (php)
```

This configuration adds an `admin` guard that uses Sanctum for authentication. We'll create an `Admin` model to support it. Each guard maps to its own provider.

2 – Create Admin Model and Migration

Add a separate `admins` table and model. This keeps admin accounts separate from user accounts.

```
// database/migrations/2025_08_27_000000_create_admins_table.php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up(): void {
        Schema::create('admins', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
```

```
            $table->timestamps();
        });
    }
    public function down(): void {
        Schema::dropIfExists('admins');
    }
};Code language: PHP (php)
```

This migration creates a new `admins` table with unique emails. In practice, you might also add `role` or `permissions` fields for fine-grained access control.

```
// app/Models/Admin.php
namespace App\Models;

use Illuminate\Foundation\Auth\User as Authenticatable;
use Laravel\Sanctum\HasApiTokens;

class Admin extends Authenticatable
{
    use HasApiTokens;

    protected $fillable = ['name','email','password'];
    protected $hidden = ['password'];
}Code language: PHP (php)
```

The `Admin` model uses `HasApiTokens` to generate and validate Sanctum tokens, just like the User model.

3 – Authentication Endpoints

Create controllers for login/logout flows for both users and admins. Each will issue tokens tied to the correct guard.

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

Route::post('/user/login', [UserAuthController::class, 'login']);
Route::post('/admin/login', [AdminAuthController::class, 'login']);

Route::middleware('auth:sanctum')->group(function () {
    Route::post('/user/logout', [UserAuthController::class,
'logout']);
    Route::post('/admin/logout', [AdminAuthController::class,
'logout']);
});Code language: PHP (php)
```

This provides separate login endpoints for users and admins. Both use Sanctum, but tokens are tied to their respective models.

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

use App\Models\Admin;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;

class AdminAuthController extends Controller
{
    public function login(Request $request)
    {
        $request->validate([
            'email' => 'required|email',
            'password' => 'required'
        ]);

        $admin = Admin::where('email',$request->email)->first();

        if (!$admin || !Hash::check($request->password,
$admin->password)) {
            return response()->json(['message' => 'Invalid
```

```
credentials'], 401);
        }

        $token = $admin->createToken('admin-
token',['admin'])->plainTextToken;

        return response()->json(['token' => $token]);
    }

    public function logout(Request $request)
    {
        $request->user()->currentAccessToken()->delete();
        return response()->json(['message' => 'Logged out']);
    }
}Code language: PHP (php)
```

The admin login issues a token with ability `admin`. This lets you enforce role-based access later by checking abilities on tokens. Logout deletes the current token.

4 – Securing Routes by Role

You can restrict routes by checking Sanctum token abilities, ensuring only admins can access certain endpoints.

```
// routes/api.php (snippet)
Route::middleware(['auth:sanctum','ability:admin'])->group(function ()
{
    Route::get('/admin/dashboard', function () {
        return ['message' => 'Welcome, Admin!'];
    });
});Code language: PHP (php)
```

Here, only tokens created with the `admin` ability can access `/admin/dashboard`. Normal

user tokens will get a 403 Forbidden response.

5 – UI: Simple API Test Page

Here's a minimal UI to test admin vs user endpoints with tokens.

```
<!-- resources/views/api/multi-auth-test.blade.php -->
@extends('layouts.app')

@section('content')
<div class="container">
  <h1>Multi-Auth API Tester</h1>

  <input type="text" id="token" class="form-control mb-3"
placeholder="Paste Bearer Token">
  <button class="btn btn-theme mb-3" onclick="testUser()">Test User
Endpoint</button>
  <button class="btn btn-danger mb-3" onclick="testAdmin()">Test Admin
Endpoint</button>

  <pre id="result"></pre>
</div>

<script
src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
<script>
function testUser() {
  const token = document.getElementById('token').value;
  axios.get('/api/posts', { headers: { Authorization: `Bearer
${token}` } })
    .then(r => document.getElementById('result').textContent =
JSON.stringify(r.data,null,2))
    .catch(e => document.getElementById('result').textContent = e);
```

```
}
function testAdmin() {
  const token = document.getElementById('token').value;
  axios.get('/api/admin/dashboard', { headers: { Authorization:
`Bearer ${token}` } })
    .then(r => document.getElementById('result').textContent =
JSON.stringify(r.data,null,2))
    .catch(e => document.getElementById('result').textContent = e);
}
</script>
@endsectionCode language: HTML, XML (xml)
```

This page lets you paste a token and call either a user or an admin endpoint. With the wrong token type, you'll see an error — demonstrating multi-auth in action.

Wrapping Up

By configuring multiple guards and issuing tokens with Sanctum, you can support different authentication flows for multiple user types. You built an `Admin` model, set up endpoints for login/logout, secured routes by token abilities, and tested the flow with a simple UI. This approach gives you flexibility while keeping authentication clean and secure.

What's Next

- Using Laravel Passport for Advanced API Authentication
- How to Add JWT Authentication to Laravel APIs
- Integrating Laravel with Third-Party APIs (Mail, SMS, Payment)