1v0 Ship v1.0 Faster

# How to Build a REST API with Laravel 12 & Sanctum

## How to Build a REST API with Laravel 12 & Sanctum

Building a REST API in Laravel requires a secure way to authenticate requests. **Laravel Sanctum** is the recommended package for issuing and managing API tokens. In this guide, you'll configure Sanctum, protect routes, issue tokens, and build endpoints for authentication and data access. We'll also build a small test UI to interact with the API.

## 1 – Install and Configure Sanctum

First, install Sanctum and publish its configuration and migrations.

```
composer require laravel/sanctum

php artisan vendor:publish --
provider="Laravel\Sanctum\SanctumServiceProvider"

php artisan migrateCode language: Bash (bash)
```

The package creates a `personal_access_tokens` table where issued tokens are stored. Publishing the config file gives you full control over expiration and storage options.

## 2 – Enable Sanctum Middleware

Add Sanctum's middleware to the `api` guard so that token authentication works automatically on protected routes.

```
// app/Http/Kernel.php (snippet)
protected $middlewareGroups = [
    'api' => [
\Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::cl
ass,
        'throttle:api',
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],
];Code language: PHP (php)
```

Including `EnsureFrontendRequestsAreStateful` allows SPAs and mobile apps to use cookies or tokens seamlessly. Throttling protects APIs against brute force attacks.

## 3 – Updating the User Model

Add the `HasApiTokens` trait to the User model so it can issue and manage tokens.

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

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

class User extends Authenticatable
{
    use HasApiTokens;
```

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

Now you can call `$user->createToken()` to generate tokens tied to the user. Each token can have its own abilities (scopes).

## 4 – Authentication Routes and Controllers

Create endpoints for registering, logging in, and logging out. Tokens are returned after login and should be used in the `Authorization` header.

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

Route::post('/register', [AuthApiController::class, 'register']);
Route::post('/login', [AuthApiController::class, 'login']);
Route::middleware('auth:sanctum')->post('/logout',
[AuthApiController::class, 'logout']);Code language: PHP (php)
```

These routes provide a minimal authentication API. The `auth:sanctum` middleware ensures only valid token holders can call logout.

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

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

class AuthApiController extends Controller
{
    public function register(Request $request)
    {
```

```
        $data = $request->validate([
            'name' => 'required|string|max:255',
            'email' => 'required|email|unique:users',
            'password' => 'required|string|min:6',
        ]);

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

        return response()->json(['user' => $user], 201);
    }

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

        $user = User::where('email', $request->email)->first();

        if (!$user || !Hash::check($request->password,
$user->password)) {
            return response()->json(['message' => 'Invalid
credentials'], 401);
        }

        $token = $user->createToken('api-token',
['*'])->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 `login` method issues a token on success. The token must be sent as `Authorization: Bearer <token>` in subsequent requests. `logout()` revokes the current token by deleting it.

## 5 – Protecting API Routes

To secure your API endpoints, wrap them with `auth:sanctum`. Example: protecting a posts resource.

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

Route::middleware('auth:sanctum')->group(function () {
    Route::get('/posts', [PostApiController::class, 'index']);
    Route::post('/posts', [PostApiController::class, 'store']);
});Code language: PHP (php)
```

Now only authenticated users with valid tokens can fetch or create posts. If the token is missing or invalid, Laravel returns a 401 Unauthorized response automatically.

## 6 – Example Resource Controller

Here's a simple API controller for posts. It validates input and ties posts to the authenticated user.

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

use App\Models\Post;
use Illuminate\Http\Request;

class PostApiController extends Controller
{
    public function index()
    {
        return Post::with('user:id,name')->latest()->paginate(10);
    }

    public function store(Request $request)
    {
        $data = $request->validate([
            'title' => 'required|string|max:255',
            'body'  => 'required|string',
        ]);

        $post = $request->user()->posts()->create($data);

        return response()->json($post, 201);
    }
}Code language: PHP (php)
```

The `index` endpoint paginates posts and includes the author's name. The `store` method creates a post linked to the authenticated user, ensuring accountability and traceability.

## 7 – Quick UI: API Tester Blade

For convenience, here's a small Blade UI to test your API directly in the browser. It uses Axios to make requests with the Bearer token.

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

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

  <div class="mb-3">
    <label class="form-label">Bearer Token</label>
    <input id="token" type="text" class="form-control"
placeholder="Paste your token here">
  </div>

  <button class="btn btn-theme mb-3" onclick="getPosts()">Fetch
Posts</button>
  <pre id="result"></pre>
</div>

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

```
}
</script>
@endsectionCode language: PHP (php)
```

This UI lets you test API endpoints quickly—just paste a token and click “Fetch Posts.” Useful for developers or QA before using Postman or mobile clients.

## Wrapping Up

With Sanctum, building a secure REST API in Laravel is straightforward. You installed and configured Sanctum, updated the User model, built authentication routes, protected endpoints, and added a small UI tester. This approach is lightweight, token-based, and ideal for SPAs and mobile apps. You now have the foundation of a production-ready API.

## What’s Next

- Using Laravel with GraphQL: A Beginner’s Guide
- How to Build a Multi-Auth API with Laravel & Sanctum
- How to Add JWT Authentication to Laravel APIs