Building a Mobile App Backend with Laravel 12 API

Building a Mobile App Backend with Laravel 12 API

A mobile app needs a reliable backend to handle authentication, serve JSON data, sync user content, and enforce security. Laravel 12 provides everything you need to build a **RESTful API** that mobile apps (iOS/Android) can consume easily. In this tutorial you'll create authentication endpoints, version your API, format JSON consistently, apply rate limiting, and prepare for push notifications.

1 – Set Up Sanctum for Mobile Token Auth

Mobile apps typically use **token-based auth**. Sanctum is lighter than Passport and perfect for mobile-first APIs.

```
composer require laravel/sanctum

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

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

Publishing Sanctum migrations adds `personal_access_tokens`. Tokens issued here are tied to users, making mobile logins stateless and secure.

2 – Auth Controller for Mobile

Create endpoints for login, logout, and fetching the current user. These return JSON only (no views).

```
// app/Http/Controllers/Api/MobileAuthController.php
namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

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

        if (! $token = Auth::attempt($credentials)) {
            return response()->json(['message' => 'Invalid
credentials'], 401);
        }

        $user = Auth::user();
        $token = $user->createToken('mobile')->plainTextToken;

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

    public function me(Request $request)
    {
        return response()->json($request->user());
```

```
    }

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

`login` validates credentials and issues a personal access token. The mobile client stores this token securely (Keychain/Keystore). `me` lets apps fetch the logged-in user, while `logout` revokes the token.

3 – API Versioning

Versioning avoids breaking old mobile clients. Use URI prefixes like `/api/v1/` and separate controllers by namespace.

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

Route::prefix('v1')->group(function () {
    Route::post('/login',  [MobileAuthController::class, 'login']);
    Route::post('/logout', [MobileAuthController::class,
'logout'])->middleware('auth:sanctum');
    Route::get('/me',      [MobileAuthController::class,
'me'])->middleware('auth:sanctum');
});Code language: PHP (php)
```

Prefixing routes with `v1` ensures older apps keep working when you release `v2` with changes. Each version can have its own controllers/resources.

4 – Consistent JSON Responses

Always return consistent shapes, so mobile devs can code against stable schemas. Wrap responses in a common format.

```
// app/Http/Resources/UserResource.php
namespace App\Http\Resources;

use Illuminate\Http\Resources\Json\JsonResource;

class UserResource extends JsonResource
{
    public function toArray($request)
    {
        return [
            'id'    => $this->id,
            'name'  => $this->name,
            'email' => $this->email,
        ];
    }
}Code language: PHP (php)
```

`UserResource` ensures consistent user JSON regardless of DB schema changes. Use resources for all entities returned to mobile clients.

```
// Example in controller
return response()->json([
    'status' => 'ok',
    'data'   => new UserResource($request->user())
]);Code language: PHP (php)
```

Wrapping every response with `status` + `data` (and optionally `error`) gives mobile developers a predictable structure across endpoints.

5 – Rate Limiting & Security

Mobile APIs are public-facing. Apply strict `throttle` limits to prevent brute force and abuse.

```
// routes/api.php (snippet)
Route::middleware('throttle:60,1')->prefix('v1')->group(function () {
    // all endpoints here are limited to 60 per minute per IP
});Code language: PHP (php)
```

Use the `throttle` middleware globally or per route group. Adjust per your scale. Combine with `auth:sanctum` so only authenticated clients can access sensitive endpoints.

6 – Push Notifications Hook (Optional)

Mobile apps often expect notifications. While Laravel doesn't send push directly, you can store device tokens and call FCM (Firebase) or APNs (Apple) APIs.

```
// database/migrations/...create_device_tokens.php
Schema::create('device_tokens', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->constrained()->cascadeOnDelete();
    $table->string('token')->unique(); // FCM/APNs token
    $table->string('platform'); // ios / android
    $table->timestamps();
});Code language: PHP (php)
```

Saving device tokens per user allows you to target them with push via FCM/APNs when certain backend events fire (new message, order shipped, etc.).

7 – Example Mobile Response

Here's a sample JSON payload mobile clients would consume from a protected endpoint:

```
{
  "status": "ok",
  "data": {
    "user": {
      "id": 1,
      "name": "Alice",
      "email": "alice@example.com"
    },
    "posts": [
      {
        "id": 101,
        "title": "Hello World",
        "body": "This is the first post..."
      }
    ]
  }
}Code language: JSON / JSON with Comments (json)
```

This structure is predictable, easy to parse, and extendable. Even when you add new fields, older mobile clients won't break if they ignore them.

Wrapping Up

You built a mobile-ready backend: token auth with Sanctum, versioned endpoints, resource-based JSON, rate limiting, and optional push integration. This setup ensures your iOS/Android apps can consume data securely and reliably. Stick to consistent JSON schemas and evolve with versioning when breaking changes are unavoidable.

What’s Next

- Integrating Laravel with Third-Party APIs (Mail, SMS, Payment)
- How to Build a Secure File Upload API in Laravel
- Using Laravel Passport for Advanced API Authentication