✅ 1. Use Laravel as an API that Connects to MySQL

If you're **building an API** in Laravel that **reads/writes data from a MySQL database**, here's how you do it.

🔧 Step-by-step:

➤ 1. Configure .env to connect MySQL

Edit your .env file:

Env

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1        # Or your database IP
DB_PORT=3306
DB_DATABASE=your_db_name
DB_USERNAME=your_username
DB_PASSWORD=your_password
```

➤ 2. Migrate or connect to an existing MySQL table

Bash

```
php artisan migrate
```

Or create a model for your table:

bash

```
php artisan make:model Product -m
```

➤ 3. Create API Routes (routes/api.php)

php

```
use App\Http\Controllers\ProductController;

Route::get('/products', [ProductController::class, 'index']);
Route::post('/products', [ProductController::class, 'store']);
```

➤ 4. Create the Controller

bash

```
php artisan make:controller ProductController
```

Example ProductController.php:

```
php

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

class ProductController extends Controller
{
   public function index()
   {
      return Product::all();  // GET all products
   }

   public function store(Request $request)
   {
      $product = Product::create($request->all());  // Create product
      return response()->json($product, 201);
   }
}
```

✅ 2. Laravel App Connects to a Remote API Instead of Local DB (Optional Case)

If your **Laravel app needs to connect to a remote API that holds the data**, you won't use MySQL directly. Instead:

- Use **HTTP client** in Laravel (`Http::get`, `Http::post`) to fetch data.
- Example:

```
php
CopyEdit
use Illuminate\Support\Facades\Http;

$response = Http::get('https://api.example.com/products');
$data = $response->json();
```

□ Summary

Use Case	Approach
Laravel connects directly to MySQL and serves API	Configure `.env`, use models/controllers to handle data
Laravel connects to another API (not directly MySQL)	Use Laravel HTTP Client to fetch API data

1️⃣ Prepare Laravel as an API backend

Laravel already makes API creation pretty straightforward.

a) Set up routes

In routes/api.php:

```
php
CopyEdit
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Api\ProductController;

Route::get('/products', [ProductController::class, 'index']);
Route::post('/products', [ProductController::class, 'store']);
```

b) Create a controller

```
bash
CopyEdit
php artisan make:controller Api/ProductController
```

Example controller:

```
php
CopyEdit
namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Models\Product;

class ProductController extends Controller
{
   public function index()
   {
      return response()->json(Product::all());
   }

   public function store(Request $request)
   {
      $product = Product::create($request->all());
      return response()->json($product, 201);
   }
}
```

c) Enable CORS (so Vercel app can call Laravel API)

Install CORS package:

```
bash
CopyEdit
composer require fruitcake/laravel-cors
```

Then in app/Http/Middleware/HandleCors.php, configure allowed origins in config/cors.php:

php
CopyEdit
'paths' => ['api/*'],
'allowed_methods' => ['*'],
'allowed_origins' => ['https://your-vercel-app.vercel.app'],

---

2 Deploy Laravel API to a server

Vercel **can't directly run PHP/Laravel** — it's for front-end JS frameworks.
So, you'll need to **host Laravel separately**. Options:

- **Free/cheap hosting** → Render, Railway, Heroku (with custom MySQL), Hostinger, etc.
- **Paid hosting** → VPS (DigitalOcean, Linode, AWS Lightsail).

Example (Render deployment):

1. Push Laravel to GitHub.
2. Create new **Web Service** on Render.com.
3. Set the start command:

   bash
   CopyEdit
   php artisan migrate --force
   php artisan serve --host 0.0.0.0 --port $PORT

4. Add your .env variables in Render's dashboard:

   env
   CopyEdit
   APP_KEY=base64:...
   APP_ENV=production
   DB_CONNECTION=mysql
   DB_HOST=your-db-host
   DB_PORT=3306
   DB_DATABASE=your-db-name
   DB_USERNAME=your-db-user
   DB_PASSWORD=your-db-password

---

3 Connect Vercel App (frontend) to Laravel API

In your Vercel-hosted Next.js app, call the Laravel API endpoint:

javascript
CopyEdit
export default async function getProducts() {

```
  const res = await fetch('https://your-laravel-api.onrender.com/api/products');
  const data = await res.json();
  return data;
}
```

If your Laravel app is public, it will work directly — just ensure CORS is set correctly.

✅ **In summary**

- **Laravel API** → Hosted on a PHP-compatible server (NOT Vercel).
- **MySQL DB** → Either same host or an external service (PlanetScale, Railway MySQL).
- **Vercel Frontend** → Fetches data from Laravel API via HTTPS.