

---

# Créer une API Laravel de zéro avec authentification

Guide complet pour créer une API Laravel RESTful de zéro avec authentification, incluant la gestion des véhicules, trajets, maintenance et supervision.

**Programmation** **Laravel** **30 min de lecture** **Niveau Intermédiaire**

---

Document généré le 27/06/2026 à 21h25 · [nouv.fr/wiki/creer-api-laravel-authentification](https://nouv.fr/wiki/creer-api-laravel-authentification)

# Sommaire

60 section(s) · 30 min de lecture

## ▢ Vue d'ensemble

## ▢ Étape 1 : Installation des prérequis

- ↳ Vérifier PHP
- ↳ Installer Composer
- ↳ Installer MySQL/MariaDB

## ▢ Étape 2 : Créer le projet Laravel

- ↳ Créer un nouveau projet
- ↳ Vérifier l'installation

## ▢ Étape 3 : Configuration de la base de données

- ↳ Créer la base de données
- ↳ Configurer le fichier .env
- ↳ Générer la clé d'application

## ▢ Étape 4 : Créer les modèles et migrations

- ↳ Créer le modèle Vehicule
- ↳ Créer le modèle Trajet
- ↳ Créer le modèle Maintenance
- ↳ Éditer les migrations
- ↳ Exécuter les migrations

## ▢ Étape 5 : Créer les contrôleurs

- ↳ Créer le contrôleur VehiculeController
- ↳ Créer le contrôleur TrajetController
- ↳ Créer le contrôleur MaintenanceController
- ↳ Créer le contrôleur HealthController

## ▢ Étape 6 : Définir les routes API (sans authentification)

- ↳ Installer le support API (Laravel 11+)

## ▢ Étape 7 : Implémenter les méthodes des contrôleurs

- ↳ VehiculeController
- ↳ TrajetController
- ↳ MaintenanceController

- ↳ HealthController

## □ **Étape 8 : Ajouter la validation**

- ↳ Créer les Form Requests (recommandé)

## □ **Étape 9 : Tester l'API (sans authentification)**

- ↳ Démarrer le serveur de développement

- ↳ Tester avec Postman

- ↳ Vérifier que tout fonctionne

## □ **Étape 10 : Ajouter l'authentification avec Laravel Sanctum**

- ↳ Installer Laravel Sanctum

- ↳ Publier la configuration

- ↳ Exécuter les migrations Sanctum

- ↳ Configurer Sanctum

- ↳ Ajouter Sanctum au middleware

## □ **Étape 11 : Créer le système d'authentification**

- ↳ Vérifier la table users

- ↳ Exécuter les migrations

- ↳ Configurer le modèle User

- ↳ Créer le contrôleur d'authentification

## □ **Étape 12 : Protéger les routes avec l'authentification**

## □ **Étape 13 : Tester l'API avec authentification**

- ↳ Tester avec Postman

## □ **Résumé des routes**

- ↳ Routes publiques

- ↳ Routes d'authentification

- ↳ Routes protégées (nécessitent un token)

## □ **Points importants**

- ↳ Sécurité

- ↳ Performance

- ↳ Bonnes pratiques

## □ **Prochaines étapes**

## □ **Ressources supplémentaires**

Ce guide vous explique comment créer une API Laravel RESTful complète de zéro. Nous allons d'abord créer toute l'API fonctionnelle, puis ajouter l'authentification avec Laravel Sanctum.

**Prérequis** : Avoir PHP 8.1+ et Composer installés sur votre système.

---

## ▣ Vue d'ensemble

---

Ce guide vous permettra de créer :

- ▣ **API RESTful** avec routes pour véhicules, trajets et maintenance
- ▣ **Base de données** avec migrations et modèles
- ▣ **Contrôleurs** pour gérer les ressources
- ▣ **Validation** des données
- ▣ **Route de santé** pour la supervision
- ▣ **Authentification** avec Laravel Sanctum (ajoutée après)

**Approche** : Nous créons d'abord l'API complète et fonctionnelle, puis nous ajoutons la sécurité avec l'authentification.

---

## ▣ Étape 1 : Installation des prérequis

---

### Vérifier PHP

```
php -v
```

📋 Copier

Vous devez avoir PHP 8.1 ou supérieur. Si ce n'est pas le cas, installez-le :

### Sur macOS (avec Homebrew) :

```
brew install php@8.2
```

📋 Copier

### Sur Linux :

```
/bin/bash -c "$(curl -fsSL https://php.new/install/linux/8.4)"
```

📋 Copier

Cette commande installe automatiquement PHP 8.4, Composer et le Laravel installer.

### Sur Windows :

Ouvrez PowerShell en tant qu'administrateur et exécutez :

```
Set-ExecutionPolicy Bypass -Scope Process -Force;  
[System.Net.ServicePointManager]::SecurityProtocol =  
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object  
System.Net.WebClient).DownloadString('https://php.new/install/windows/8.4'))
```

📋 Copier

Cette commande installe automatiquement PHP 8.4, Composer et le Laravel installer.

## Installer Composer

**Sur Linux :** Composer est déjà installé par la commande ci-dessus. Vérifiez l'installation :

```
composer --version
```

📋 Copier

**Sur macOS :**

```
curl -sS https://getcomposer.org/installer | php  
sudo mv composer.phar /usr/local/bin/composer  
composer --version
```

📋 Copier

**Sur Windows :** Composer est déjà installé par la commande ci-dessus. Vérifiez l'installation :

```
composer --version
```

📋 Copier

Vérifiez l'installation de PHP et Composer :

```
php -v  
composer --version
```

📋 Copier

## Installer MySQL/MariaDB

**Sur macOS (avec Homebrew) :**

```
brew install mysql  
brew services start mysql
```

📋 Copier

**Sur Ubuntu/Debian :**

```
sudo apt install mysql-server
sudo systemctl start mysql
sudo systemctl enable mysql
```

✂ Copier

**Sur Windows :** Téléchargez MySQL depuis [mysql.com](https://www.mysql.com)

---

## □ Étape 2 : Créer le projet Laravel

---

### Créer un nouveau projet

```
composer create-project laravel/laravel api-vehicules
cd api-vehicules
```

✂ Copier

### Vérifier l'installation

```
php artisan --version
```

✂ Copier

---

## □ Étape 3 : Configuration de la base de données

---

### Créer la base de données

Connectez-vous à MySQL :

```
mysql -u root -p
```

✂ Copier

Créez la base de données :

```
CREATE DATABASE api_vehicules CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
EXIT;
```

✂ Copier

### Configurer le fichier .env

Éditez le fichier `.env` à la racine du projet :

```
nano .env
```

✂ Copier

Modifiez les lignes suivantes :

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=api_vehicules
DB_USERNAME=root
DB_PASSWORD=votre_mot_de_passe
```

✂ Copier

## Générer la clé d'application

```
php artisan key:generate
```

✂ Copier

---

## □ Étape 4 : Créer les modèles et migrations

---

### Créer le modèle Vehicule

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

✂ Copier

### Créer le modèle Trajet

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

✂ Copier

### Créer le modèle Maintenance

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

✂ Copier

## Éditer les migrations

**database/migrations/xxxx\_create\_vehicules\_table.php :**

```
Schema::create('vehicules', function (Blueprint $table) {
    $table->id();
    $table->string('marque');
    $table->string('modele');
    $table->string('immatriculation')->unique();
    $table->year('annee');
    $table->integer('kilometrage')->default(0);
    $table->timestamps();
});
```

📄 Copier

#### database/migrations/xxxx\_create\_trajets\_table.php :

```
Schema::create('trajets', function (Blueprint $table) {
    $table->id();
    $table->foreignId('vehicule_id')->constrained()->onDelete('cascade');
    $table->date('date_depart');
    $table->time('heure_depart');
    $table->string('lieu_depart');
    $table->date('date_arrivee')->nullable();
    $table->time('heure_arrivee')->nullable();
    $table->string('lieu_arrivee')->nullable();
    $table->integer('kilometres_parcourus')->default(0);
    $table->text('notes')->nullable();
    $table->timestamps();
});
```

📄 Copier

#### database/migrations/xxxx\_create\_maintenances\_table.php :

```
Schema::create('maintenances', function (Blueprint $table) {
    $table->id();
    $table->foreignId('vehicule_id')->constrained()->onDelete('cascade');
    $table->date('date_maintenance');
    $table->string('type_maintenance');
    $table->text('description');
    $table->decimal('cout', 10, 2)->nullable();
    $table->integer('kilometrage');
    $table->timestamps();
});
```

📄 Copier

## Exécuter les migrations

```
php artisan migrate
```

📄 Copier

---

## □ Étape 5 : Créer les contrôleurs

---

## Créer le contrôleur VehiculeController

```
php artisan make:controller VehiculeController --api
```

📄 Copier

## Créer le contrôleur TrajetController

```
php artisan make:controller TrajetController --api
```

📄 Copier

## Créer le contrôleur MaintenanceController

```
php artisan make:controller MaintenanceController --api
```

📄 Copier

## Créer le contrôleur HealthController

```
php artisan make:controller HealthController
```

📄 Copier

Dans `app/Http/Controllers/HealthController.php`, ajoutez une méthode `index()` qui retourne un statut de santé.

---

## 📄 Étape 6 : Définir les routes API (sans authentification)

---

### Installer le support API (Laravel 11+)

Dans les versions récentes de Laravel (11+), le fichier `routes/api.php` n'est plus créé par défaut. Vous devez installer le support API :

```
php artisan install:api
```

📄 Copier

Cette commande va :

- Créer le fichier `routes/api.php`
- Configurer les middlewares nécessaires dans `bootstrap/app.php`

**Note** : Si vous utilisez Laravel 10 ou antérieur, le fichier `routes/api.php` existe déjà par défaut.

Dans `routes/api.php`, ajoutez toutes les routes **sans authentification pour l'instant** :

```
// Route de santé (publique)
Route::get('health', [HealthController::class, 'index']);

// Gestion des véhicules (sans protection pour l'instant)
Route::get('vehicules', [VehiculeController::class, 'index']);
Route::post('vehicules', [VehiculeController::class, 'store']);
Route::get('vehicules/{id}', [VehiculeController::class, 'show']);
Route::put('vehicules/{id}', [VehiculeController::class, 'update']);
Route::delete('vehicules/{id}', [VehiculeController::class, 'destroy']);

// Gestion des trajets
Route::get('vehicules/{vehicule_id}/trajets', [TrajetController::class, 'index']);
Route::post('vehicules/{vehicule_id}/trajets', [TrajetController::class, 'store']);
Route::get('trajets/{id}', [TrajetController::class, 'show']);

// Gestion de la maintenance
Route::get('vehicules/{vehicule_id}/maintenances', [MaintenanceController::class, 'index']);
Route::post('vehicules/{vehicule_id}/maintenances', [MaintenanceController::class, 'store']);
Route::put('maintenances/{id}', [MaintenanceController::class, 'update']);
```

📄 Copier

**Note** : Pour l'instant, toutes les routes sont publiques. Nous ajouterons l'authentification à l'étape suivante.

---

## 📄 Étape 7 : Implémenter les méthodes des contrôleurs

---

### VehiculeController

Vous devrez implémenter :

- `index()` - Liste tous les véhicules
- `store()` - Crée un nouveau véhicule
- `show($id)` - Affiche un véhicule spécifique
- `update($id)` - Met à jour un véhicule
- `destroy($id)` - Supprime un véhicule

### TrajetController

Vous devrez implémenter :

- `index($vehicule_id)` - Liste tous les trajets d'un véhicule
- `store($vehicule_id)` - Crée un nouveau trajet pour un véhicule
- `show($id)` - Affiche un trajet spécifique

### MaintenanceController

Vous devrez implémenter :

- `index($vehicule_id)` - Liste toutes les maintenances d'un véhicule
- `store($vehicule_id)` - Crée une nouvelle maintenance pour un véhicule

- `update($id)` - Met à jour une maintenance

## HealthController

Implémentez :

- `index()` - Retourne un statut de santé (ex: `{"status": "ok"}`)

---

## □ Étape 8 : Ajouter la validation

---

### Créer les Form Requests (recommandé)

```
php artisan make:request StoreVehiculeRequest
php artisan make:request UpdateVehiculeRequest
php artisan make:request StoreTrajetRequest
php artisan make:request StoreMaintenanceRequest
php artisan make:request UpdateMaintenanceRequest
```

📋 Copier

Dans chaque Form Request, définissez les règles de validation dans la méthode `rules()`.

---

## □ Étape 9 : Tester l'API (sans authentification)

---

### Démarrer le serveur de développement

```
php artisan serve
```

📋 Copier

L'API sera accessible sur `http://localhost:8000/api`

### Tester avec Postman

Ouvrez Postman et créez une nouvelle collection pour votre API.

#### Route de santé (publique) :

- Méthode : GET
- URL : `http://localhost:8000/api/health`
- Headers : Aucun
- Body : Aucun

#### Créer un véhicule (sans authentification pour l'instant) :

- Méthode : POST
- URL : `http://localhost:8000/api/vehicules`
- Headers : Content-Type: `application/json`

- Body (raw JSON) :

```
{
  "marque": "Toyota",
  "modele": "Corolla",
  "immatriculation": "AB-123-CD",
  "annee": 2020,
  "kilometrage": 50000
}
```

📋 Copier

### Lister les véhicules :

- Méthode : GET
- URL : `http://localhost:8000/api/vehicules`
- Headers : Aucun
- Body : Aucun

### Récupérer un véhicule spécifique :

- Méthode : GET
- URL : `http://localhost:8000/api/vehicules/1`
- Headers : Aucun
- Body : Aucun

### Mettre à jour un véhicule :

- Méthode : PUT
- URL : `http://localhost:8000/api/vehicules/1`
- Headers : Content-Type: `application/json`
- Body (raw JSON) :

```
{
  "marque": "Toyota",
  "modele": "Corolla",
  "immatriculation": "AB-123-CD",
  "annee": 2021,
  "kilometrage": 55000
}
```

📋 Copier

### Créer un trajet pour un véhicule :

- Méthode : POST
- URL : `http://localhost:8000/api/vehicules/1/trajets`
- Headers : Content-Type: `application/json`
- Body (raw JSON) :

```
{
  "date_depart": "2024-01-15",
  "heure_depart": "08:00",
  "lieu_depart": "Paris",
  "date_arrivee": "2024-01-15",
  "heure_arrivee": "12:00",
  "lieu_arrivee": "Lyon",
  "kilometres_parcourus": 450
}
```

📋 Copier

### Lister les trajets d'un véhicule :

- Méthode : GET
- URL : `http://localhost:8000/api/vehicules/1/trajets`
- Headers : Aucun
- Body : Aucun

### Récupérer un trajet spécifique :

- Méthode : GET
- URL : `http://localhost:8000/api/trajets/1`
- Headers : Aucun
- Body : Aucun

### Créer une maintenance pour un véhicule :

- Méthode : POST
- URL : `http://localhost:8000/api/vehicules/1/maintenances`
- Headers : Content-Type: `application/json`
- Body (raw JSON) :

```
{
  "date_maintenance": "2024-01-10",
  "type_maintenance": "Vidange",
  "description": "Vidange complète avec changement de filtre",
  "cout": 150.00,
  "kilometrage": 50000
}
```

📋 Copier

### Lister les maintenances d'un véhicule :

- Méthode : GET
- URL : `http://localhost:8000/api/vehicules/1/maintenances`
- Headers : Aucun
- Body : Aucun

### Mettre à jour une maintenance :

- Méthode : PUT
- URL : `http://localhost:8000/api/maintenances/1`
- Headers : Content-Type: `application/json`
- Body (raw JSON) :

```
{
  "date_maintenance": "2024-01-10",
  "type_maintenance": "Vidange",
  "description": "Vidange complète avec changement de filtre et huile",
  "cout": 180.00,
  "kilometrage": 50000
}
```

📋 Copier

### Supprimer un véhicule :

- Méthode : `DELETE`
- URL : `http://localhost:8000/api/vehicules/1`
- Headers : Aucun
- Body : Aucun

### Vérifier que tout fonctionne

Testez toutes les routes pour vous assurer que l'API fonctionne correctement avant d'ajouter l'authentification. Une fois que tout fonctionne, passez à l'étape suivante pour sécuriser l'API.

---

## 📌 Étape 10 : Ajouter l'authentification avec Laravel Sanctum

---

Maintenant que l'API fonctionne, nous allons ajouter l'authentification pour sécuriser les routes.

### Installer Laravel Sanctum

Laravel Sanctum est le package recommandé pour l'authentification d'API :

```
composer require laravel/sanctum
```

📋 Copier

### Publier la configuration

```
php artisan vendor:publish --provider="LaravelSanctumSanctumServiceProvider"
```

📋 Copier

### Exécuter les migrations Sanctum

```
php artisan migrate
```

📋 Copier

Cette commande créera la table `personal_access_tokens` nécessaire pour gérer les tokens d'authentification.

## Configurer Sanctum

Vérifiez le fichier `config/sanctum.php`. Les valeurs par défaut conviennent généralement, mais vous pouvez les ajuster si nécessaire.

## Ajouter Sanctum au middleware

### Pour Laravel 11+ (bootstrap/app.php) :

Éditez `bootstrap/app.php` et ajoutez le middleware dans la section API :

```
->withMiddleware(function (Middleware $middleware) {
    $middleware->api(prepend: [
        LaravelSanctumHttpMiddlewareEnsureFrontendRequestsAreStateful::class,
    ]);
})
```

📋 Copier

### Pour Laravel 10 et antérieur (app/Http/Kernel.php) :

Éditez `app/Http/Kernel.php` et ajoutez dans le groupe `api` :

```
protected $middlewareGroups = [
    'api' => [
        // ... autres middlewares
        LaravelSanctumHttpMiddlewareEnsureFrontendRequestsAreStateful::class,
    ],
];
```

📋 Copier

---

## 📌 Étape 11 : Créer le système d'authentification

---

### Vérifier la table users

Laravel inclut déjà une migration pour la table `users`. Vérifiez qu'elle existe dans `database/migrations/`. Si elle n'existe pas, créez-la :

```
php artisan make:migration create_users_table
```

📋 Copier

Dans la migration, ajoutez les colonnes nécessaires :

```
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->timestamp('email_verified_at')->nullable();
    $table->string('password');
    $table->rememberToken();
    $table->timestamps();
});
```

📄 Copier

## Exécuter les migrations

```
php artisan migrate
```

📄 Copier

## Configurer le modèle User

Éditez `app/Models/User.php` et ajoutez le trait `HasApiTokens` :

```
use LaravelSanctumHasApiTokens;

class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;
    // ... reste du code
}
```

📄 Copier

## Créer le contrôleur d'authentification

```
php artisan make:controller AuthController
```

📄 Copier

Dans `app/Http/Controllers/AuthController.php`, ajoutez les méthodes suivantes :

**Méthode** `register()` :

```

public function register(Request $request)
{
    $validated = $request->validate([
        'name' => 'required|string|max:255',
        'email' => 'required|string|email|max:255|unique:users',
        'password' => 'required|string|min:8|confirmed',
    ]);

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

    $token = $user->createToken('auth_token')->plainTextToken;

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

```

✂ Copier

### Méthode login() :

```

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

    if (!Auth::attempt($validated)) {
        return response()->json([
            'message' => 'Identifiants invalides'
        ], 401);
    }

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

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

```

✂ Copier

### Méthode logout() :

```
public function logout(Request $request)
{
    $request->user()->currentAccessToken()->delete();

    return response()->json([
        'message' => 'Déconnexion réussie'
    ]);
}
```

📋 Copier

### Méthode user() :

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

📋 Copier

N'oubliez pas d'ajouter les imports en haut du fichier :

```
use IlluminateHttpRequest;
use AppModelsUser;
use IlluminateSupportFacadesAuth;
use IlluminateSupportFacadesHash;
```

📋 Copier

---

## 📌 Étape 12 : Protéger les routes avec l'authentification

---

Maintenant, modifiez `routes/api.php` pour protéger les routes avec l'authentification :

```
// Route de santé (reste publique)
Route::get('health', [HealthController::class, 'index']);

// Routes d'authentification (publiques)
Route::post('register', [AuthController::class, 'register']);
Route::post('login', [AuthController::class, 'login']);

// Routes protégées par authentification
Route::middleware('auth:sanctum')->group(function () {
    // Authentification
    Route::post('logout', [AuthController::class, 'logout']);
    Route::get('user', [AuthController::class, 'user']);
    // Gestion des véhicules
    Route::get('vehicules', [VehiculeController::class, 'index']);
    Route::post('vehicules', [VehiculeController::class, 'store']);
    Route::get('vehicules/{id}', [VehiculeController::class, 'show']);
    Route::put('vehicules/{id}', [VehiculeController::class, 'update']);
    Route::delete('vehicules/{id}', [VehiculeController::class, 'destroy']);
    // Gestion des trajets
    Route::get('vehicules/{vehicule_id}/trajets', [TrajetController::class, 'index']);
    Route::post('vehicules/{vehicule_id}/trajets', [TrajetController::class, 'store']);
    Route::get('trajets/{id}', [TrajetController::class, 'show']);
    // Gestion de la maintenance
    Route::get('vehicules/{vehicule_id}/maintenances', [MaintenanceController::class, 'index']);
    Route::post('vehicules/{vehicule_id}/maintenances', [MaintenanceController::class, 'store']);
    Route::put('maintenances/{id}', [MaintenanceController::class, 'update']);
});
```

📋 Copier

---

## 📌 Étape 13 : Tester l'API avec authentification

---

### Tester avec Postman

#### 1. Inscription :

- Méthode : `POST`
- URL : `http://localhost:8000/api/register`
- Headers : `Content-Type: application/json`
- Body (raw JSON) :

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "password123",
  "password_confirmation": "password123"
}
```

📋 Copier

La réponse contiendra un token. Copiez ce token pour les requêtes suivantes.

#### 2. Connexion :

- Méthode : `POST`
- URL : `http://localhost:8000/api/login`
- Headers : `Content-Type: application/json`
- Body (raw JSON) :

```
{
  "email": "john@example.com",
  "password": "password123"
}
```

📋 Copier

La réponse contiendra un `token`. Copiez ce token.

### 3. Configurer l'authentification dans Postman :

Pour toutes les routes protégées, vous devez ajouter le token :

- Allez dans l'onglet **Authorization**
- Sélectionnez le type **Bearer Token**
- Collez votre token dans le champ **Token**

### 4. Créer un véhicule (avec token) :

- Méthode : `POST`
- URL : `http://localhost:8000/api/vehicules`
- Headers : `Content-Type: application/json`
- Authorization : `Bearer Token` (collez votre token)
- Body (raw JSON) :

```
{
  "marque": "Toyota",
  "modele": "Corolla",
  "immatriculation": "AB-123-CD",
  "annee": 2020,
  "kilometrage": 50000
}
```

📋 Copier

### 5. Lister les véhicules (avec token) :

- Méthode : `GET`
- URL : `http://localhost:8000/api/vehicules`
- Authorization : `Bearer Token` (collez votre token)
- Headers : Aucun autre header nécessaire
- Body : Aucun

### 6. Déconnexion :

- Méthode : `POST`
- URL : `http://localhost:8000/api/logout`
- Authorization : `Bearer Token` (collez votre token)
- Headers : Aucun autre header nécessaire

- Body : Aucun

## 7. Route de santé (toujours publique) :

- Méthode : GET
- URL : `http://localhost:8000/api/health`
- Headers : Aucun
- Body : Aucun
- Authorization : Aucune nécessaire (route publique)

**Astuce Postman :** Vous pouvez créer une variable d'environnement pour stocker votre token et l'utiliser automatiquement dans toutes vos requêtes. Allez dans l'onglet "Environments" et créez une variable `token`, puis utilisez `{{token}}` dans le champ Authorization.

---

## ☐ Résumé des routes

---

### Routes publiques

- GET `/api/health` - Statut de santé

### Routes d'authentification

- POST `/api/register` - Inscription
- POST `/api/login` - Connexion

### Routes protégées (nécessitent un token)

#### Gestion des véhicules :

- GET `/api/vehicules` - Liste tous les véhicules
- POST `/api/vehicules` - Crée un véhicule
- GET `/api/vehicules/{id}` - Affiche un véhicule
- PUT `/api/vehicules/{id}` - Met à jour un véhicule
- DELETE `/api/vehicules/{id}` - Supprime un véhicule

#### Gestion des trajets :

- GET `/api/vehicules/{id}/trajets` - Liste les trajets d'un véhicule
- POST `/api/vehicules/{id}/trajets` - Crée un trajet pour un véhicule
- GET `/api/trajets/{id}` - Affiche un trajet

#### Gestion de la maintenance :

- GET `/api/vehicules/{id}/maintenances` - Liste les maintenances d'un véhicule
- POST `/api/vehicules/{id}/maintenances` - Crée une maintenance pour un véhicule
- PUT `/api/maintenances/{id}` - Met à jour une maintenance

#### Authentification :

- POST `/api/logout` - Déconnexion

- `GET /api/user` - Informations de l'utilisateur connecté
- 

## ▣ Points importants

---

### Sécurité

- Toutes les routes (sauf `/health`, `/register`, `/login`) nécessitent un token d'authentification
- Utilisez HTTPS en production
- Validez toutes les données d'entrée
- Protégez contre les attaques CSRF (Sanctum le fait automatiquement)

### Performance

- Utilisez la pagination pour les listes (`paginate()`)
- Ajoutez des index sur les colonnes fréquemment recherchées
- Utilisez le cache pour les données statiques

### Bonnes pratiques

- Respectez les conventions REST
  - Utilisez les codes HTTP appropriés (200, 201, 404, 422, etc.)
  - Retournez des réponses JSON cohérentes
  - Documentez votre API (considérez Swagger/OpenAPI)
- 

## ▣ Prochaines étapes

---

Une fois l'API de base créée, vous pouvez :

1. **Ajouter des tests** avec PHPUnit
  2. **Documenter l'API** avec Swagger/OpenAPI
  3. **Ajouter la pagination** sur les listes
  4. **Implémenter la recherche et le filtrage**
  5. **Ajouter des relations** entre les modèles
  6. **Mettre en place des événements** (Events/Listeners)
  7. **Configurer les queues** pour les tâches asynchrones
  8. **Déployer l'API** sur un serveur de production
- 

## ▣ Ressources supplémentaires

---

- [Documentation Laravel](#)
- [Laravel Sanctum](#)

- [RESTful API Design](#)
  - [HTTP Status Codes](#)
- 

**Note** : Ce guide vous donne les commandes et la structure. Vous devrez implémenter la logique métier dans chaque contrôleur selon vos besoins spécifiques.