

Építsen biztonságos PHP REST API-t a Laravel 8-ban a Sanctum Auth segítségével

Laravel 8 Sanctum hitelesítési oktatóanyag. Ebben az oktatóanyagban megtanulhatja, hogyan hozhat létre vagy építhet biztonságos PHP RESTful API-t a Laravel 8 alkalmazásban a Laravel 8 sanctum csomag segítségével. Hasonlóképpen lépésről lépésre elmagyarázzuk hogyan tesztelheti a Laravel 8 Sanctum hitelesítési REST API-t a Postman tesztelőeszköz segítségével.

[A Laravel Sanctum](#) kifogástalan, biztonságos, kirívóan gyors, könnyű hitelesítési rendszert kínál egyoldalas alkalmazásokhoz (SPA), mobil alkalmazásokhoz és egyszerű, token alapú API-khoz. A Sanctum egy mélyreható csomag, amely lehetővé teszi minden felhasználó számára, hogy egymástól függetlenül több API tokenet generáljon a fiókjához. Ezek a tokenek különféle szerepeket és hatóköröket biztosítanak, amelyek meghatározzák, hogy a tokenek mely műveletekre jogosultak. Tehát kezdjük el a szentély pihenő api létrehozását a laravel alkalmazásban anélkül, hogy elméletekbe bocsátkoznánk.

Laravel 8 Sanctum Auth REST API példa

Íme az utasítások egy egyszerű, biztonságos API létrehozásához a laravel 8 alkalmazásban.

- 1. lépés:** Hozzon létre Laravel Projectet
- **2. lépés:** Adja hozzá az adatbázis részleteit
- **3. lépés:** Telepítse a Laravel Sanctum Pacakage programot
- **4. lépés:** A Sanctum beállítása
- **5. lépés:** Frissítse a modellt és futtassa az áttelepítést
- **6. lépés:** API-erőforrások létrehozása
- **7. lépés:** Állítsa be a vezérlőket
- **8. lépés:** Hozzon létre REST API-útvonalakat
- **09. lépés:** Tesztelje a REST API-t a Postmanben

Hozzon létre Laravel Projectet

Composer create-project -prefer-dist laravel/laravel blog

A laravel és a mysql adatbázis közötti kommunikációhoz hozzá kell adnia az adatbázis részleteit az .env konfigurációs fájlhoz.

DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=blog

DB_USERNAME=root

DB_PASSWORD=

Telepítse a Laravel Sanctum Pacakage programot

Composer require laravel/sanctum

A sanctum beüzemelése:

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

Ezután engedélyezze a sanctumot a middlewareGroups szekcióban az app/Http/kernel.php állományban.

```
protected $middlewareGroups = [
    'web' => [
        \App\Http\Middleware\EncryptCookies::class,
        \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
        \Illuminate\Session\Middleware\StartSession::class,
        // \Illuminate\Session\Middleware\AuthenticateSession::class,
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,
        \App\Http\Middleware\VerifyCsrfToken::class,
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],

    'api' => [
        \Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::class,
        'throttle:api',
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],
];
```

A migráció futtatásához hajtsa végre a következő parancsot; hasonlóképpen később az adatbázison belül létrehozunk egy sanctum táblát a hitelesítési információk kezelésére.

Php artisan migrate

Ezt követően fel kell készítenünk a sanctum konfigurációt a laravel auth projekthez. Következésképpen adja hozzá a sanctum HasApiTokens osztályt az app/Models/User.php fájlhoz.

```
class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];
}
```

Mentse a modellt és futtassa a migrációt.

Php artisan make:migration create_blogs_table

Néhány adatot hozzá kell adnia a migrációs fájlhoz, ezért nyissa meg, a database/migrations/create_blogs_table.php fájlt.

```
class CreateBlogsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('blogs', function (Blueprint $table) {
            $table->id();
            $table->string( "title" );
            $table->text( "description" );
            $table->timestamps();
        });
    }
}
```

Ezután hozzon létre egy app/Models/Blog.php fájlt, és állítsa be a tábla migrációs tulajdonságait a \$fillable tömbön belül.

```
class Blog extends Model {

    use HasFactory;

    protected $fillable = [
        "title",
        "description"
    ];
}
```

Futtassa a migrációs parancsot az adatbázis tábla létrehozásához.

Php artisan migrate

Építsen API-erőforrásokat

A laravel eloquent api-forrásait vesszük segítségül. Lehetővé teszi a modellhez hasonló válaszok kezelését, például a BlogControllert fogjuk használni, amely megfelel a Blog modell tulajdonságainak.

Parancs végrehajtása a Blog eloquent API-forrásainak generálásához.

Php artisan make:resource Blog

Töltsük fel az app/Http/Resources/Blog.php állományt a következő kóddal:

```
class Blog extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param \Illuminate\Http\Request $request
     * @return array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */
    public function toArray( $request )
    {
        return [
            "id" => $this->id,
            "title" => $this->title,
            "description" => $this->description,
            "created_at" => $this->created_at->format( "m/d/Y" ),
            "updated_at" => $this->updated_at->format( "m/d/Y" )
        ];
    }
}
```

A vezérlők beállítása

Ezután három vezérlőt kell létrehoznunk a hitelesítési folyamat kezelésére, először hozzon létre egy API-könyvtárat a Controllers mappában, ezt követően hozzon létre három fájlt a mappán belül, nevezze el őket AuthController, BaseController és BlogController néven. Ezek a fájlok külön-külön kezelik a bejelentkezési, regisztrációs és blogolási műveleteket. Ezt követően adja hozzá a kódot az app/Http/Controllers/API/BaseController.php fájlhoz:

```
namespace App\Http\Controllers\API;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller as Controller;

class BaseController extends Controller {

    public function sendResponse( $result, $message ) {

        $response = [
            "success" => true,
            "data" => $result,
            "message" => $message,
        ];

        return response()->json( $response, 200 );
    }

    public function sendError( $error, $errorMessages = [], $code = 404 ) {

        $response = [
            "success" => false,
            "message" => $error,
        ];

        if( !empty( $errorMessages ) ) {
            $response[ "data" ] = $errorMessages;
        }

        return response()->json( $response, $code );
    }
}
```


Nyissa meg és helyezze el az alábbi kódot az app/Http/Controllers/API/AuthController.php fájlban, ez felerősíti a felhasználói regisztrációs és bejelentkezési folyamatot:

```
namespace App\Http\Controllers\API;
use Illuminate\Http\Request;
use App\Http\Controllers\API\BaseController as BaseController;
use Illuminate\Support\Facades\Auth;
use Validator;
use App\Models\User;

class AuthController extends BaseController {

    public function signin( Request $request ) {

        if( Auth::attempt([ "email" => $request->email, "password" => $request->password ])) {

            $authUser = Auth::user();
            $success[ "token" ] = $authUser->createToken( "MyAuthApp" )->plainTextToken;
            $success[ "name" ] = $authUser->name;

            return $this->sendResponse( $success, "User signed in" );

        }else {

            return $this->sendError( "Unauthorised.", [ "error" => "Unauthorized" ] );

        }

    }

    public function signup( Request $request ) {

        $validator = Validator::make( $request->all(), [
            "name" => "required",
            "email" => "required",
            "password" => "required",
            "confirm_password" => "required|same:password",
        ] );

        if( $validator->fails() ) {

            return $this->sendError( "Error validation", $validator->errors() );

        }

        $input = $request->all();
        $input[ "password" ] = bcrypt( $input[ "password" ] );
        $user = User::create( $input );
        $succes[ "token" ] = $user->createToken( "MyAuthApp" )->plainTextToken;
        $succes[ "name" ] = $user->name;

        return $this->sendResponse( $succes, "User created successfully" );

    }

}
```

Menjen az app/Http/Controllers/API/BlogController.php fájlhoz, és helyezze be a CRUD műveleti kódot:

```
namespace App\Http\Controllers\API;

use Illuminate\Http\Request;
use App\Http\Controllers\API\BaseController as BaseController;
use Validator;
use App\Models\Blog;
use App\Http\Resources\Blog as BlogResource;

class BlogController extends BaseController {

    public function index() {

        $blogs = Blog::all();
        return $this->sendResponse( BlogResource::collection( $blogs ), "Posts fetched" );
    }

    public function store( Request $request ) {

        $input = $request->all();
        $validator = Validator::make( $input, [
            "title" => "required",
            "description" => "required"
        ]);

        if( $validator->fails() ) {

            return $this->sendError( $validator->errors() );
        }

        $blog = Blog::create( $input );

        return $this->sendResponse( new BlogResource( $blog ), "Post created" );
    }

    public function show( $id ) {

        $blog = Blog::find( $id );

        if( is_null( $blog ) ) {

            return $this->sendError( "Post does not exists" );
        }

        return $this->sendResponse( new BlogResource( $blog ), "Post fetched" );
    }

    public function update( Request $request, Blog $blog ) {

        $input = $request->all();
        $validator = Validator::make( $input, [
            "title" => "required",
            "description" => "required"
        ]);

        if( $validator->fails() ) {

            return $this->sendError( $validator->errors() );
        }

        $blog->title = $input[ "title" ];
        $blog->description = $input[ "description" ];
        $blog->save();

        return $this->sendResponse( new BlogResource( $blog ), "Post updated" );
    }
}
```



```

    public function destroy( Blog $blog ) {
        $blog->delete();

        return $this->sendResponse( [], "Post deleted" );
    }
}

```

Hozzon létre REST API-útvonalakat

A sanctum auth api útvonalak létrehozásához vezérlőket kell használnia, Meghatároztuk a bejelentkezési, bejegyzési módszereket és erőforrásokat az auth api együttes létrehozásához. Nyissa meg és illessze be a kódot a routes/api.php fájlba:

```

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\API\AuthController;
use App\Http\Controllers\API\BlogController;
/*
|-----
| API Routes
|-----
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/
Route::post( "login", [ AuthController::class, "signin" ] );
Route::post( "register", [ AuthController::class, "signup" ] );
Route::get( "blogs", [ BlogController::class, "index" ] );
Route::post( "blogs", [ BlogController::class, "store" ] );
Route::get( "blogs/{blog}", [ BlogController::class, "show" ] );
Route::put( "blogs/{blog}", [ BlogController::class, "update" ] );
Route::delete( "blogs/{blog}", [ BlogController::class, "destroy" ] );

Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
    return $request->user();
});

```

Tesztelje a Sanctum REST API-t a Postmanben

Töltse le az alkalmazást a <https://www.postman.com/downloads/> oldalról.

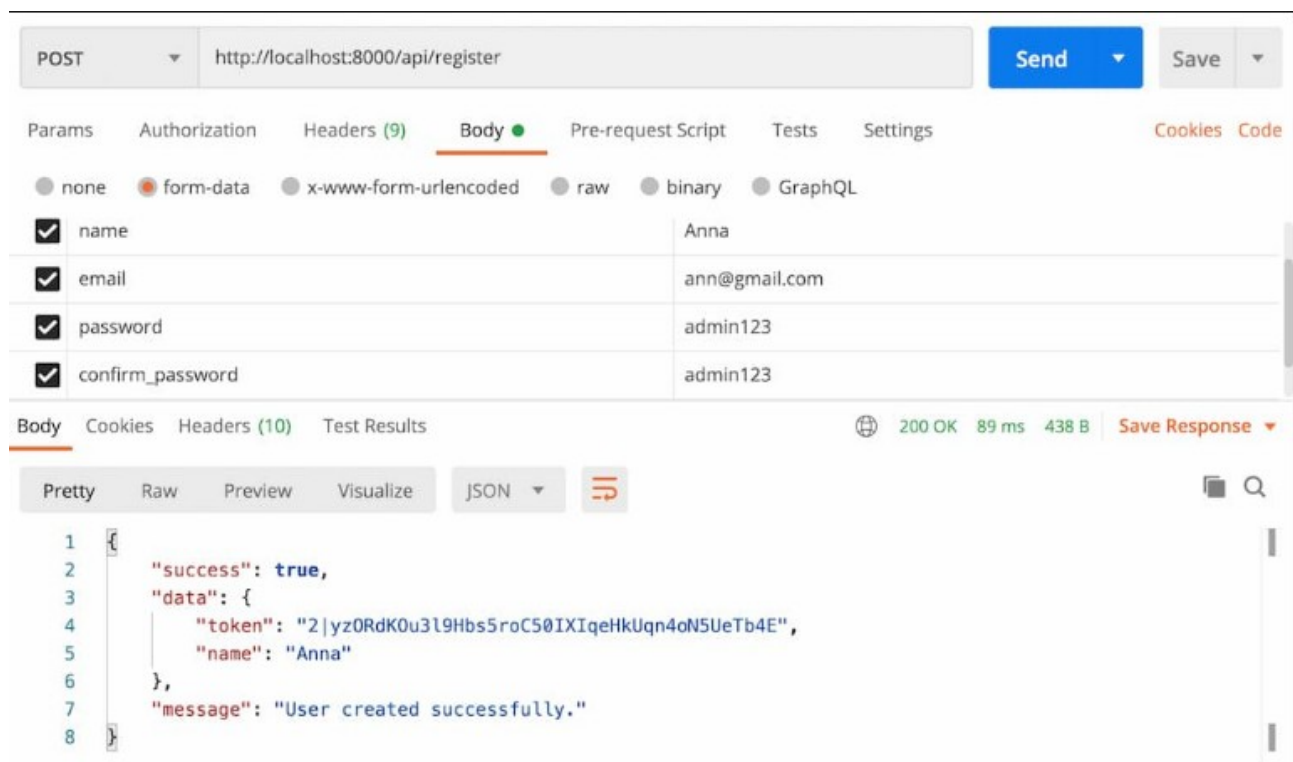
A REST API-t a sanctum auth használatával hoztuk létre, továbbá el kell indítani a laravel fejlesztőkiszolgálót a php artisan paranccsal:

php artisan serve

A Postman segítségével teszteljük a Laravel Sanctum hitelesítési API-kat:

A REST API tesztelése

Nyissa meg a Postman alkalmazást, és a legördülő menüből válassza ki a Post módszert. Írja be az útvonalat vagy az URL-t a címsorba, válassza ki a Törzs részt, írja be a nevet, e-mail címet, jelszót, erősítse meg a jelszót, majd kattintson a Küldés gombra, hogy létrehozzon egy felhasználót a laravel sanctum auth api használatával.



Bejelentkezési API tesztelése

Adja hozzá a következő URL-t a postás címsorához, állítsa a módszert POST-ra, írja be az e-mail címet és a jelszót, majd kattintson a Küldés gombra a felhasználó bejelentkezéséhez.

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8000/api/login
- Body Type:** form-data
- Body Data:**

KEY	VALUE
email	ann@gmail.com
password	admin123
- Response:** 200 OK, 81 ms, 426 B
- Response Body (JSON):**

```
{  "success": true,  "data": {    "token": "3|RHqp6MZmxPGLrLZZyucncLz9RXfFVpnrIFr67fNh",    "name": "Anna"  },  "message": "User signed in"}
```

Állítsa be az engedélyezési token

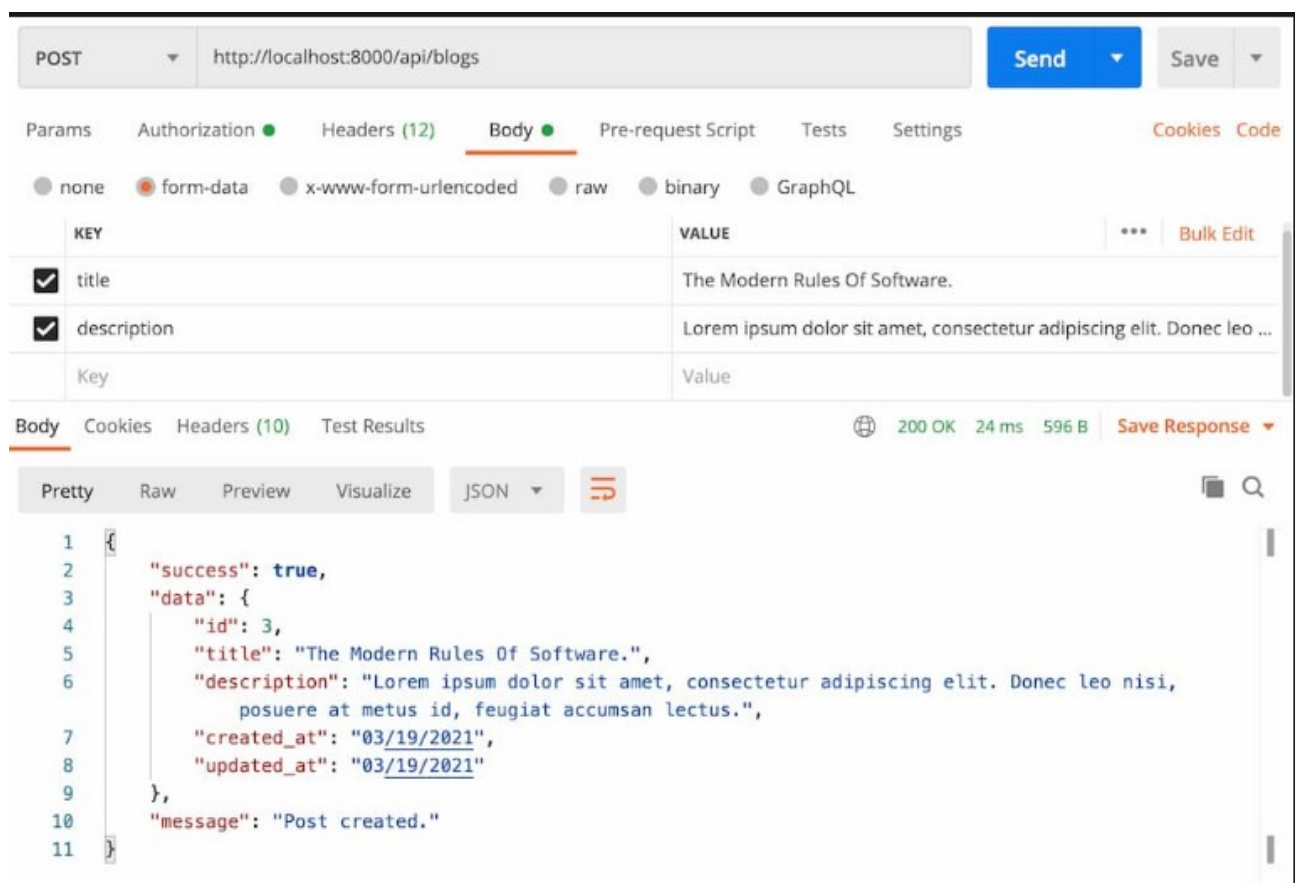
Lépjen be a Postman alkalmazás Engedélyezés lapjára, válassza ki a „Bearer Token” lehetőséget a Típus legördülő menüből, majd adja hozzá az alkalmazásba bejelentkezve kapott hitelesítési token.



The screenshot shows the Postman interface with the 'Authorization' tab selected. Under the 'TYPE' dropdown, 'Bearer Token' is chosen. The 'Token' field contains the value '5|9eSCwLoftXLKyfXKvAnHzVYpV3LNLcRXM9UB4QBC'. A note below states: 'The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)'.

Hozzon létre bejegyzést a Sanctum API-val

Adja hozzá a következő URL-t a postman címsorához, váltsa a módszert POST-ra, írja be a címet és a leírást, majd kattintson a bejegyzés létrehozásához és az adatbázisban való tároláshoz.



The screenshot shows the Postman interface with a POST request to 'http://localhost:8000/api/blogs'. The 'Body' tab is selected, showing a JSON body with the following data:

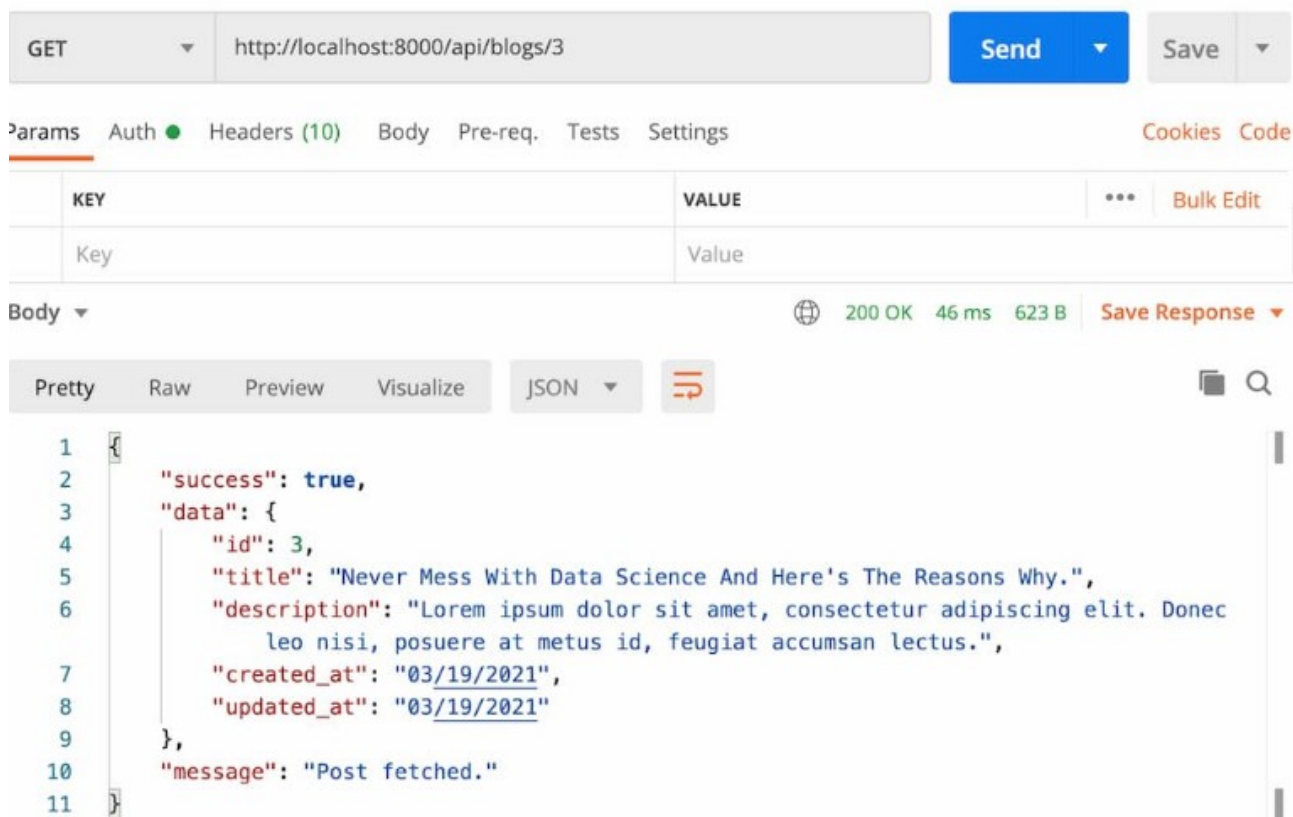
KEY	VALUE
<input checked="" type="checkbox"/> title	The Modern Rules Of Software.
<input checked="" type="checkbox"/> description	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec leo ...
Key	Value

The response is shown in the 'Body' tab, displaying a JSON object:

```
1 {
2   "success": true,
3   "data": {
4     "id": 3,
5     "title": "The Modern Rules Of Software.",
6     "description": "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec leo nisi,
7                   posuere at metus id, feugiat accumsan lectus.",
8     "created_at": "03/19/2021",
9     "updated_at": "03/19/2021"
10  },
11  "message": "Post created."
```

Egyetlen bejegyzés beszerzése

Most már láthatja, hogyan tölthet le egyetlen bejegyzést a post id és a rest api használatával, használja az alábbi URL-t:



GET http://localhost:8000/api/blogs/3 Send Save

Params Auth Headers (10) Body Pre-req. Tests Settings Cookies Code

KEY	VALUE	...	Bulk Edit
Key	Value		

Body 200 OK 46 ms 623 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "success": true,
3   "data": {
4     "id": 3,
5     "title": "Never Mess With Data Science And Here's The Reasons Why.",
6     "description": "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec
7       leo nisi, posuere at metus id, feugiat accumsan lectus.",
8     "created_at": "03/19/2021",
9     "updated_at": "03/19/2021"
10  },
11  "message": "Post fetched."
```


Az összes bejegyzés lekérése

Az összes bejegyzést lekérheti egyszerre, ezért használja a következő REST API-t GET metódussal az összes rekord lekéréséhez.

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:8000/api/blogs
- Buttons:** Send, Save
- Tabs:** Params, Auth, Headers (12), Body (selected), Pre-req., Tests, Settings, Cookies, Code
- Form:** form-data
- Table:** A table with columns KEY and VALUE, and a Bulk Edit button.
- Response:** 200 OK, 28 ms, 1.1 KB, Save Response
- Body:** Pretty, Raw, Preview, Visualize (selected), JSON, and a search icon.
- JSON Response:**

```
1 {
2   "success": true,
3   "data": [
4     {
5       "id": 1,
6       "title": "The Modern Rules Of Software.",
7       "description": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.
8         Donec leo nisi, posuere at metus id, feugiat accumsan lectus.",
9       "created_at": "03/19/2021",
10      "updated_at": "03/19/2021"
11     },
12     {
13       "id": 2,
14       "title": "Seven Ways Laravel Can Improve Your Business.",
15       "description": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.
16         Donec leo nisi, posuere at metus id, feugiat accumsan lectus.",
```

Frissítse a bejegyzést

Frissítsen egy bejegyzést a sanctum auth restful api használatával, adja hozzá a következő útvonalat a Put metódussal egyetlen rekord frissítéséhez az adatbázisban.

The screenshot shows a REST client interface with a PUT request to `http://localhost:8000/api/blogs/2?title=Hello World Rules&description=Lor`. The request is successful, returning a 200 OK status. The response body is a JSON object indicating the post was updated.

Request:

- Method: PUT
- URL: `http://localhost:8000/api/blogs/2?title=Hello World Rules&description=Lor`
- Query Params:

KEY	VALUE
title	Hello World Rules
description	Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Response:

```
{
  "success": true,
  "data": {
    "id": 2,
    "title": "Hello World Rules",
    "description": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
    "created_at": "03/19/2021",
    "updated_at": "03/19/2021"
  },
  "message": "Post updated."
}
```

Rekord törlése

Rekord törléséhez állítsa be a törlés módszerét, és használja a következő URL-t:

The screenshot shows a REST client interface with a DELETE request to `http://localhost:8000/api/blogs/2`. The request is successful, returning a 200 OK status. The response body is a JSON object indicating the post was deleted.

Request:

- Method: DELETE
- URL: `http://localhost:8000/api/blogs/2`

Response:

```
{
  "success": true,
  "data": [],
  "message": "Post deleted."
}
```