**Laravel-10 and Vue.js 3 Restfull API Project Task Management**

cÖ\_‡g Laravel Install Ki‡ev| Zvi Rb¨ Avgv‡`i Command PvjvB‡Z n‡et

composer create-project Laravel/Laravel “Project Name”

Gici Database Connection Kivi Rb¨ Avg‡`i .env File G wM‡q MySql Gi Configaretion Ki‡Z n‡e| Database G Project ‡Uwej ˆZwi Kivi Rb¨ Avgiv Migration Folder G †Uwej ˆZwi K‡i Avgv‡`i Command e¨envi Ki‡Z n‡e|Gi Rb¨ Avgviv

**# php artisan make:model “Model Name”** -all

**# php artisan make:model “Task”** -all

GB Command e¨envi Ki‡ev GZ K‡i Avgvi Model,Controller, factory,seed,migration,request Policies, GB File ¸‡jvI ˆZwi n‡q hv‡e|

Gevi Avgiv ‡mB Model Gi migration File G wM‡q GKUv †Uwej ‰Zwi Ki‡ev|

database\migrations\2023\_09\_17\_061408\_create\_tasks\_table.php

public function up(): void  
{  
 Schema::create('tasks', function (Blueprint $table) {  
 $table->id();  
 $table->string('name');  
 $table->boolean('is\_completed')->default(false);  
 $table->string('status\_active')->default(1);  
 $table->string('is\_delete')->default(0);  
 $table->timestamps();  
 });  
}

Gevi Avgiv GB Table G Development Gi Rb¨ wKQz Fake Data insert Kivi Rb¨ Factory File G wb‡¤œi Code

database\factories\Task\TaskFactory.php

public function definition(): array  
{  
 return [  
 'name' => $this->faker->sentence(),  
 'is\_completed' => rand(0, 1),  
 'status\_active' => 1,  
 'is\_delete' => 0,  
 ];  
}

Gici Avgv‡`i Seeder File hv‡ev †mLv‡b Run Method G wM‡qt

database\seeders\TaskSeeder.

public function run(): void  
{  
 Task::factory()->count(10)->create();

}

G KvR Kivi ci Avgiv GB Command Pvjv‡ev|

**# php artisan migrate –seed =”TaskSeeder”**

Gici Avgiv Avg‡`i Controller File hv‡ev, Index method Database †\_‡K Data me¸‡jv wb‡q Avmvi Rb¨ Avgiv Query wjL‡ev|

app/Http/Controllers/Task/TaskController.php

public function index()  
 {  
// return response()->json(['data' => TaskResource::collection(Task::orderBy('id')->cursorPaginate(5))]);  
 try {  
 $tasks = Task::orderBy('id')->cursorPaginate(5);  
  
 if ($tasks->isEmpty()) {  
 return response()->json(['message' => 'No tasks found.']);  
 }  
  
// return response()->json(['data' => TaskResource::collection($tasks)]);  
 return response()->json($tasks);  
 } catch (\Exception $e) {  
 Log::error('Error fetching tasks: ' . $e->getMessage());  
 return response()->json(['error' => 'Failed to fetch tasks.'], 500);  
 }  
  
 }

GLv‡b †h Query Gi †h KvRUv Kiv n‡q‡Q Zvn‡jv OOP Class Gi gva¨‡g Method Øviv KvR Kiv n‡q‡Q| TaskResource n‡”Q Class Ges Collection n‡”Q Zvi Method GLb cÖkœ n‡”Q Avgiv †Kb GB TaskResource e¨envi Ki‡ev? KviYt Laravel m¤ú` n‡jv Laravel Eloquent Model Gi Dci Avi GKwU ¯Íi ˆZwi Kiv|GwU GB Eloquent Model Gi Rb¨ wd‡i Av‡m hv WvUvi Awfbœ web¨vm ev AvKvi ˆZwi Kivi AbymwZ †`q|

app/Http/Resources/Task/TaskResource.php

<?php  
  
namespace App\Http\Resources;  
  
use Illuminate\Http\Request;  
use Illuminate\Http\Resources\Json\JsonResource;  
  
class TaskResource extends JsonResource  
{  
 /\*\*  
 \*  
 \* @return array<string, mixed>  
 \*/  
 public function toArray(Request $request): array  
 {  
 //return parent::toArray($request);  
 return [  
 'id' => $this->id,  
 'name' => $this->name,  
 'is\_completed' => (boolean) $this->is\_completed,  
 'status\_active' => $this->status\_active,  
 'is\_delete' => $this->is\_delete,  
 'created\_at' => $this->created\_at,  
 'updated\_at' => $this->updated\_at,  
 ];  
 }  
}

GLv‡b †h KvRUv Kiv n‡q‡Q Avgvi Model Gi Colmun Gi bvgUv Key Avi Avgvi Request ‡\_‡K †h WvUv Avm‡Z‡Q Zv †fjy‡Z †mU Kiv n‡q‡Q| Collection Method GB toArray wb‡”Q| Ges Task Class n‡”Q Avgvi Model Avi all Method & GLv‡b WvUv‡m‡i †Uwe‡ji mKj Column Gi WvUv Return K‡i| model Uv Avgiv †`‡Lwb GLb|

protected $fillable = [  
 'name',  
 'is\_completed',  
 'status\_active',  
 'is\_delete',  
];

Gevi Avgiv wVKgZ WvUv‡em †\_‡K wb‡q Avmvi WvUv ¸wj †`L‡Z cviwQ wKbv Zv †`L‡ev| Gi Rb¨ Avgv‡`i &GKUv Route ‰Zwi Ki‡ev| †h‡nZz Avgiv Api wb‡q KvR Ki‡ev †mRb¨ API Route ‰Zwi Ki‡ev| Gi Rb¨ Avgv‡`i Route Folder G api.php File G hv‡ev|

routes/api.php

use Illuminate\Http\Request;  
use Illuminate\Support\Facades\Route;  
use App\Http\Controllers\Task\TaskController;  
  
Route::prefix('v1')->group(function ()   
{  
 Route::apiResource('tasks', TaskController::class);  
});

Avgv‡`i GB Route wVK Av‡Q wKbv ev Route Gi wK wK Method Av‡Q Zv †PK Ki‡ev wb‡¤œi Command Gi gva¨‡g|

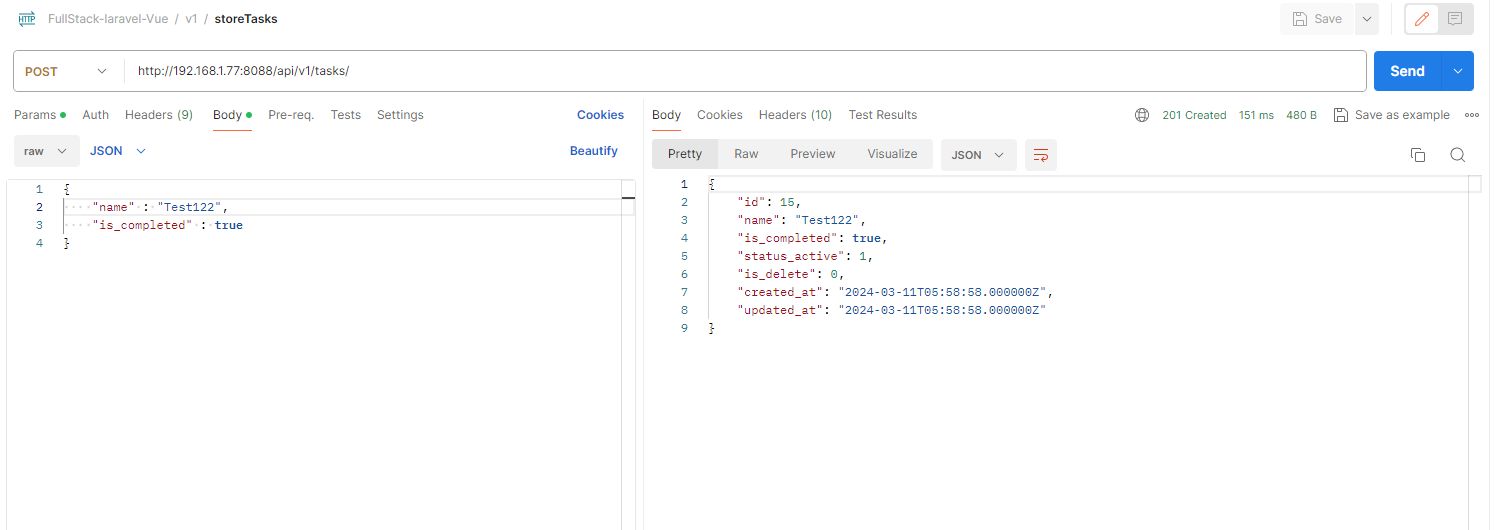
Php artisan route:list

Data Store Kivi Rb¨ Avgiv Post Method Ø¦viv Data Store Ki‡ev Gi Rb¨ Avgiv StoreTaskRequest File **G** Authorize true Ki‡ev| Ges

<?php  
  
namespace App\Http\Requests;  
  
use Illuminate\Foundation\Http\FormRequest;  
  
class StoreTaskRequest extends FormRequest  
{  
   
 public function authorize(): bool  
 {  
 return true;  
 }  
 public function rules(): array  
 {  
 return [  
 'name' => 'required|string|max:255',  
 'is\_completed' => 'required|boolean',  
 'status\_active' => 'integer',  
 'is\_delete' => 'integer',  
 ];  
 }  
}

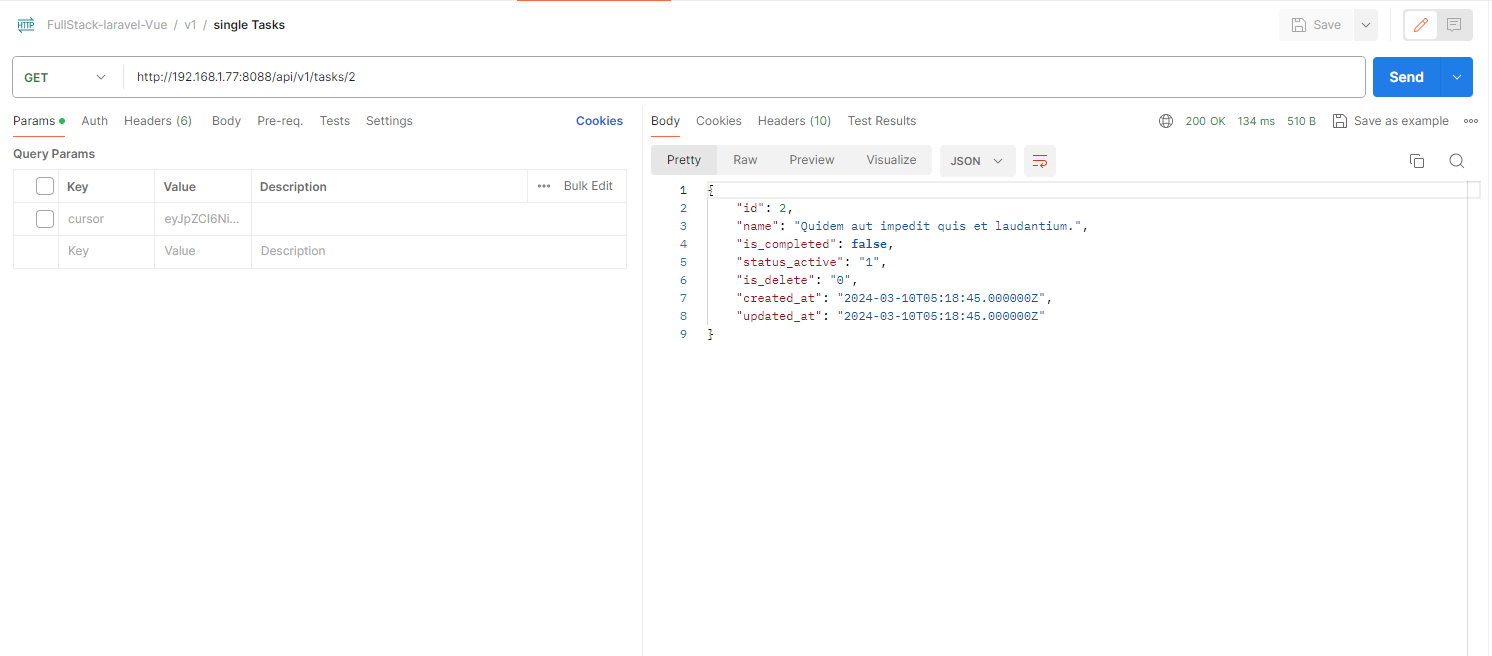
Gici Avgiv Data Store Kivi Rb¨ Store Method Code wjL‡evt

public function store(StoreTaskRequest $request)  
{  
 try {  
 $validatedData = $request->validated();  
 $validatedData['status\_active'] = $validatedData['status\_active'] ?? 1;  
 $validatedData['is\_delete'] = $validatedData['is\_delete'] ?? 0;  
 $task = Task::create($validatedData);  
 return response()->json(TaskResource::make($task), 201);  
 } catch (\Exception $e)  
 {  
 Log::error('Error creating task: ' . $e->getMessage());  
 return response()->json(['error' => 'Failed to create task.'], 500);  
 }  
}



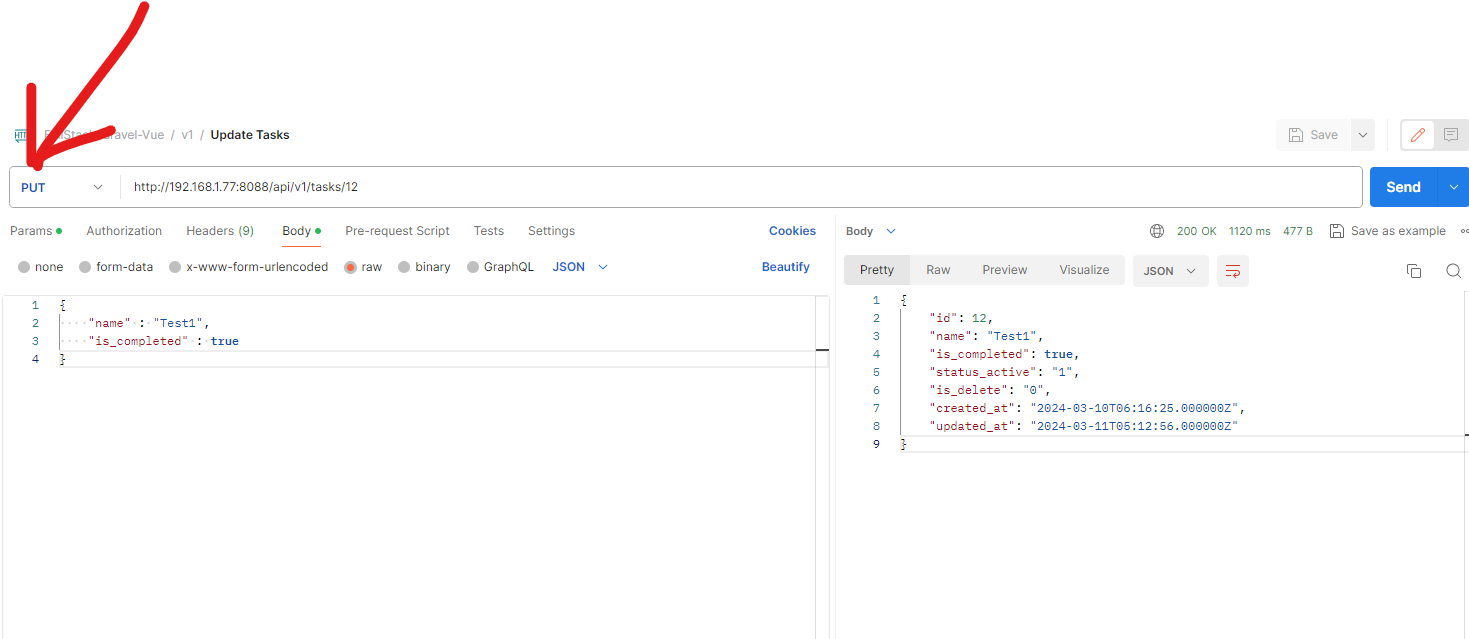
Gici Avgiv PostMan G GB fv‡e Data Send Ki‡ev| Single Data Show Kivi Rb¨ Avgiv Show Method G GBfv‡e wjL‡ev

public function show(Task $task)  
{  
 try {  
 return response()->json(TaskResource::make($task));  
 } catch (\Exception $e) {  
 Log::error('Error fetching task: ' . $e->getMessage());  
 return response()->json(['error' => 'Failed to fetch task.'], 500);  
 }  
}



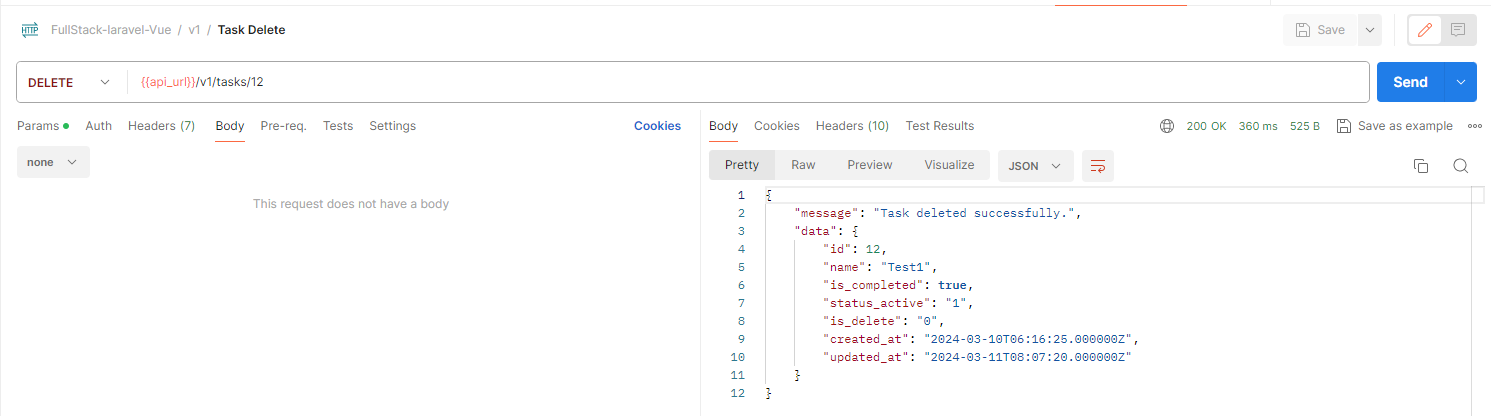
Data Update Gi Rb¨ Avgiv wb‡¤œi KvRUv Ki‡ev GB PostMan wb¤œ³ Dcv‡q Request Ki‡ev|

public function update(UpdateTaskRequest $request, Task $task)  
{  
 try {  
 $validatedData = $request->validated();  
 $task->update($validatedData);  
 return response()->json(TaskResource::make($task));  
 } catch (\Exception $e) {  
 Log::error('Error updating task: ' . $e->getMessage());  
 return response()->json(['error' => 'Failed to update task.'], 500);  
 }  
}



Gevi Avgiv †`L‡ev wKfv‡e Data Delete Ki‡ev|

public function destroy(Task $task)  
{  
 try {  
 $task->delete();  
 return response()->json(['message' => 'Task deleted successfully.', 'data' => TaskResource::make($task)]);  
 } catch (\Exception $e) {  
 Log::error('Error deleting task: ' . $e->getMessage());  
 return response()->json(['error' => 'Failed to delete task.'], 500);  
 }  
}



Avgiv †`Ljvg Lye mn‡R wKfv‡e CRUD Operation (Create, Read, Update and Delete) Kiv hvq API Gi gva¨‡g|

Gevi Avgiv †`L‡ev wKfv‡e Sanctum Token Use Kiv hvq| Sanctum n‡”Qt Laravel Sanctum SPAs (একক পৃষ্ঠার অ্যাপ্লিকেশন), মোবাইল অ্যাপ্লিকেশন এবং সহজ, টোকেন ভিত্তিক API-এর জন্য একটি পালক ওজনের প্রমাণীকরণ সিস্টেম সরবরাহ করে। Sanctum আপনার অ্যাপ্লিকেশনের প্রতিটি ব্যবহারকারীকে তাদের অ্যাকাউন্টের জন্য একাধিক API টোকেন তৈরি করতে দেয়। এই টোকেনগুলিকে ক্ষমতা / সুযোগ দেওয়া হতে পারে যা নির্দিষ্ট করে যে টোকেনগুলিকে কোন কাজগুলি সম্পাদন করার অনুমতি দেওয়া হয়েছে৷

কিভাবে এটা কাজ করে Laravel Sanctum দুটি পৃথক সমস্যা সমাধানের জন্য বিদ্যমান। লাইব্রেরির গভীরে খনন করার আগে আসুন প্রতিটি নিয়ে আলোচনা করি।

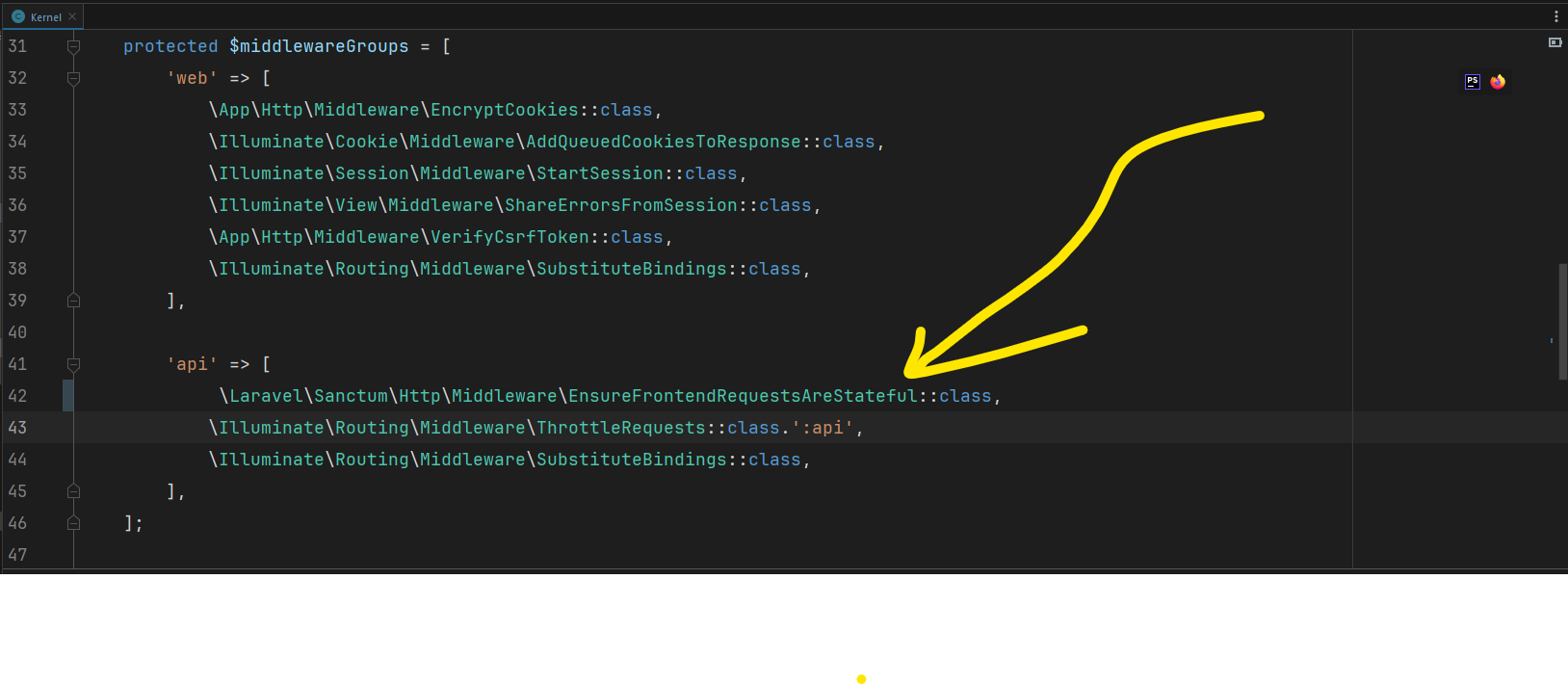
API টোকেন প্রথমত, Sanctum হল একটি সাধারণ প্যাকেজ যা আপনি OAuth-এর জটিলতা ছাড়াই আপনার ব্যবহারকারীদের API টোকেন ইস্যু করতে ব্যবহার করতে পারেন। এই বৈশিষ্ট্যটি GitHub এবং অন্যান্য অ্যাপ্লিকেশন দ্বারা অনুপ্রাণিত যা "ব্যক্তিগত অ্যাক্সেস টোকেন" ইস্যু করে। উদাহরণস্বরূপ, কল্পনা করুন আপনার অ্যাপ্লিকেশনের "অ্যাকাউন্ট সেটিংস" এর একটি স্ক্রীন রয়েছে যেখানে একজন ব্যবহারকারী তাদের অ্যাকাউন্টের জন্য একটি API টোকেন তৈরি করতে পারে৷ আপনি সেই টোকেনগুলি তৈরি এবং পরিচালনা করতে Sanctum ব্যবহার করতে পারেন। এই টোকেনগুলির সাধারণত খুব দীর্ঘ মেয়াদ শেষ হওয়ার সময় থাকে (বছর), তবে ব্যবহারকারী যে কোনও সময় ম্যানুয়ালি প্রত্যাহার করতে পারে। Laravel Sanctum একটি একক ডাটাবেস টেবিলে ব্যবহারকারী API টোকেন সংরক্ষণ করে এবং অনুমোদন হেডারের মাধ্যমে ইনকামিং HTTP অনুরোধগুলিকে প্রমাণীকরণ করে এই বৈশিষ্ট্যটি অফার করে যাতে একটি বৈধ API টোকেন থাকা উচিত।

SPA Authentication

দ্বিতীয়ত, একটি লারাভেল চালিত API-এর সাথে যোগাযোগ করতে প্রয়োজন এমন একক পৃষ্ঠা অ্যাপ্লিকেশন (এসপিএ) প্রমাণীকরণের একটি সহজ উপায় অফার করার জন্য Sanctum বিদ্যমান। এই এসপিএগুলি আপনার লারাভেল অ্যাপ্লিকেশনের মতো একই সংগ্রহস্থলে থাকতে পারে বা একটি সম্পূর্ণ পৃথক সংগ্রহস্থল হতে পারে, যেমন Vue CLI বা একটি Next.js অ্যাপ্লিকেশন ব্যবহার করে তৈরি একটি SPA। এই বৈশিষ্ট্যের জন্য, Sanctum কোনো ধরনের টোকেন ব্যবহার করে না। পরিবর্তে, Sanctum Laravel-এর অন্তর্নির্মিত কুকি ভিত্তিক সেশন প্রমাণীকরণ পরিষেবাগুলি ব্যবহার করে৷ সাধারণত, Sanctum এটি সম্পন্ন করতে Laravel এর ওয়েব প্রমাণীকরণ গার্ড ব্যবহার করে। এটি CSRF সুরক্ষা, সেশন প্রমাণীকরণের সুবিধা প্রদান করে, সেইসাথে XSS এর মাধ্যমে প্রমাণীকরণের প্রমাণপত্র ফাঁস থেকে রক্ষা করে। Sanctum শুধুমাত্র কুকি ব্যবহার করে প্রমাণীকরণ করার চেষ্টা করবে যখন ইনকামিং অনুরোধটি আপনার নিজস্ব SPA ফ্রন্টএন্ড থেকে আসে। যখন Sanctum একটি ইনকামিং HTTP অনুরোধ পরীক্ষা করে, এটি প্রথমে একটি প্রমাণীকরণ কুকি পরীক্ষা করবে এবং, যদি কোনটি উপস্থিত না থাকে, Sanctum তারপর একটি বৈধ API টোকেনের জন্য অনুমোদন শিরোনাম পরীক্ষা করবে।

You may install Laravel Sanctum via the Composer package manager:

composer require laravel/sanctum

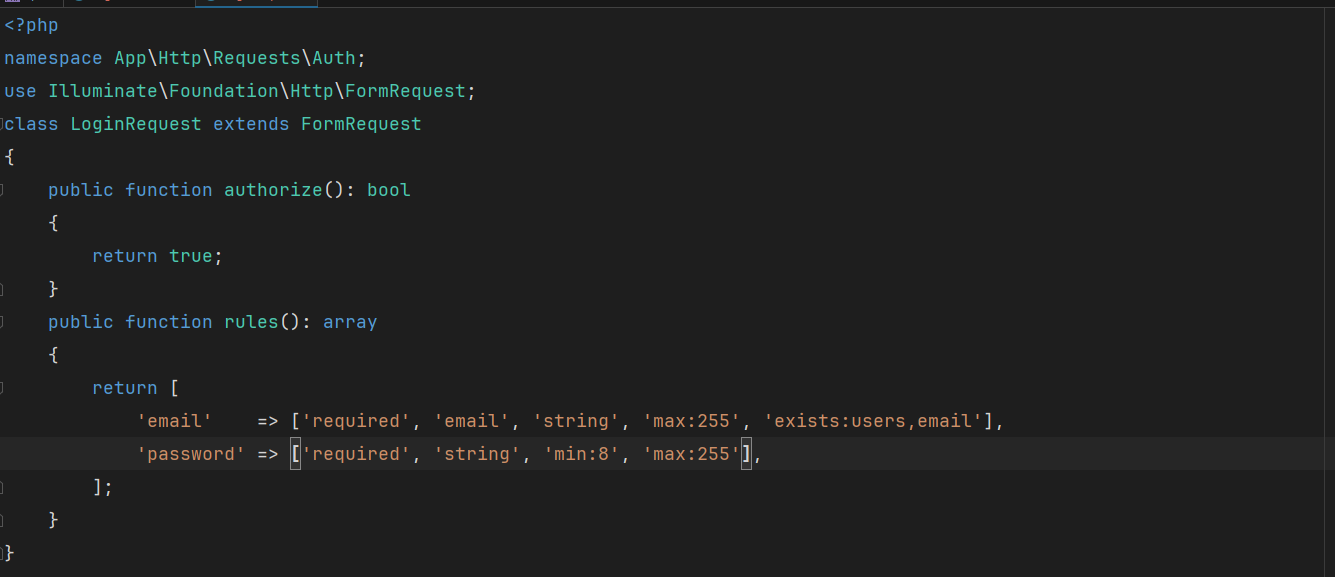




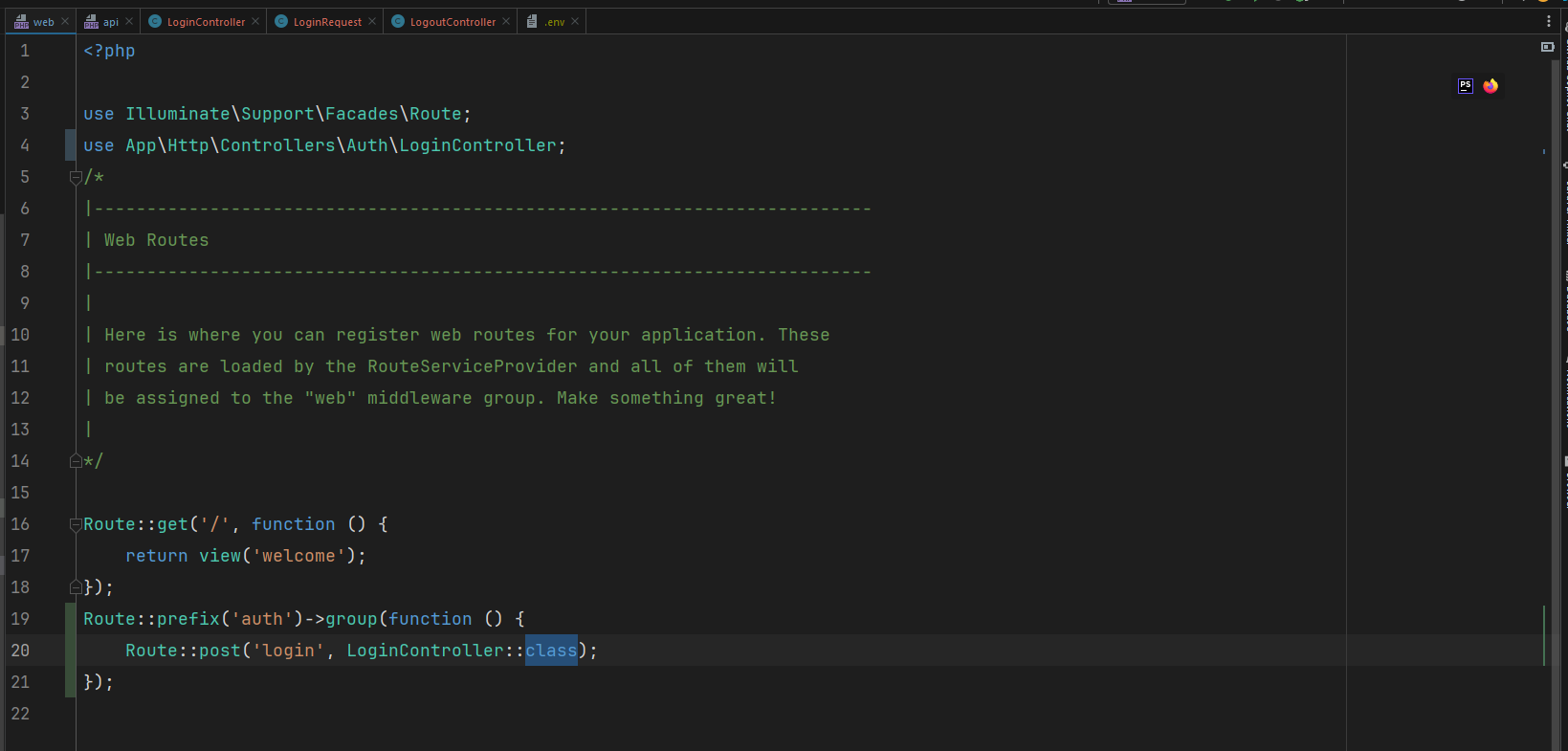
Gici Avgiv Login Gi Rb¨ GKUv LoginController ‰Zwi Ki‡ev Zvi Rb¨ Command w`‡evt

php artisan make:controller Auth/LoginController -i

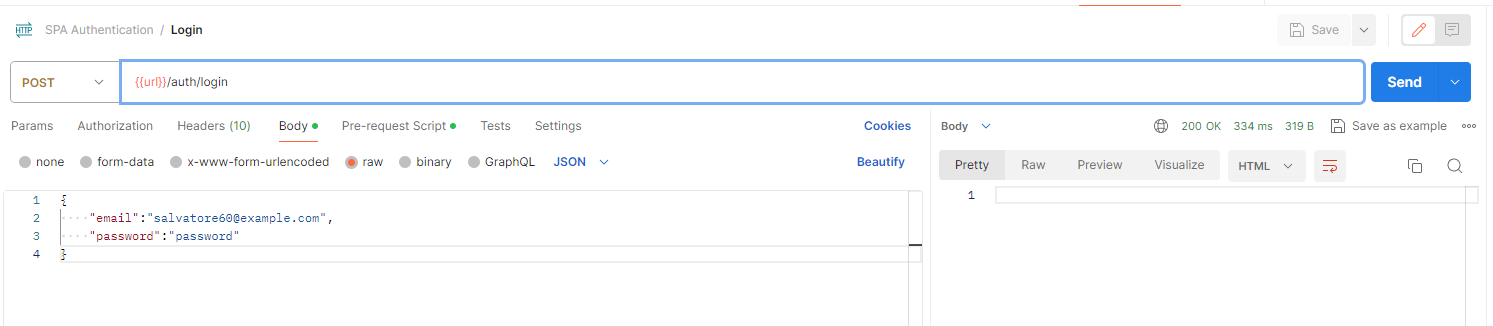
php artisan make:request Auth/LoginRequest



<?php  
namespace App\Http\Controllers\Auth;  
use App\Models\User;  
use App\Http\Controllers\Controller;  
use Illuminate\Http\Request;  
use App\Http\Requests\Auth\LoginRequest;  
use Illuminate\Support\Facades\Hash;  
  
class LoginController extends Controller  
{  
 public function \_\_invoke(LoginRequest $request)  
 {  
 try {  
 // Attempt to authenticate the user  
 if (!auth()->attempt($request->only('email', 'password'))) {  
 // Return error response if authentication fails  
 return response()->json(['message' => 'Invalid login details'], 401);  
 }  
 // Retrieve the authenticated user  
 $user = auth()->user();  
 // Create a new token for the authenticated user  
 $token = $user->createToken('auth\_token')->plainTextToken;  
 // Return success response with user details and token  
 return response()->json([  
 'message' => 'User logged in successfully',  
 'user' => $user,  
 'token' => $token  
 ], 200);  
 } catch (\Exception $e) {  
 // Return error response if an exception occurs  
 return response()->json(['message' => $e->getMessage()], 500);  
 }  
 }  
  
}

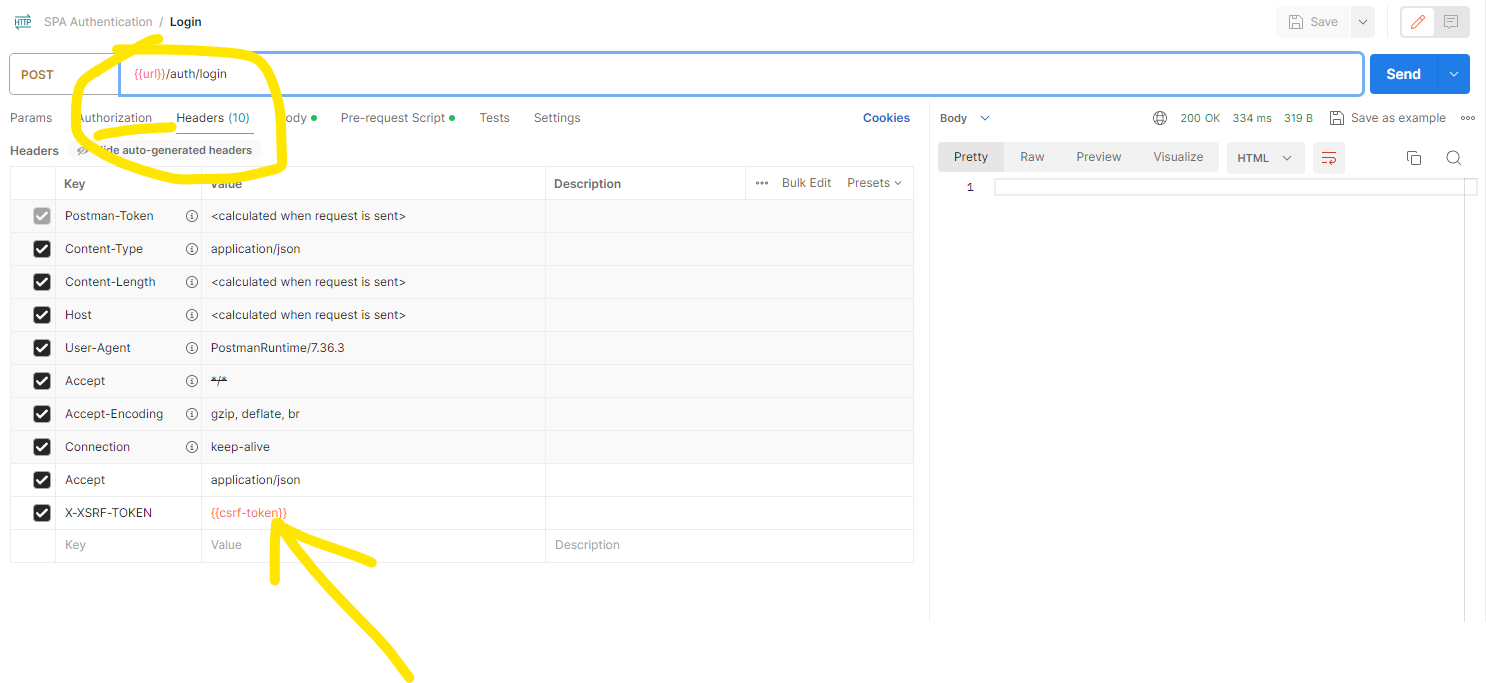


Gici Avgiv PostMan G Avgiv Key Avi Value Set Ki‡ev



Avi Avgiv menually CSRF-Toke Gi Rb¨ Pre-request Script G wb‡¤œi Code wjL‡evt

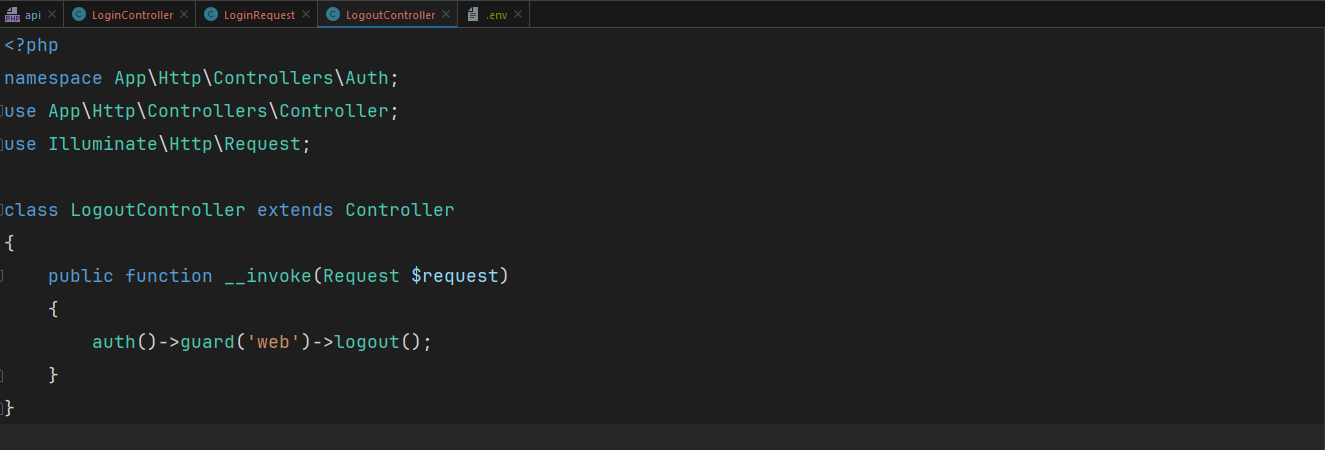
pm.sendRequest({  
 url: "http://192.168.1.77:8088/sanctum/csrf-cookie",  
 method: "GET"  
},function(err,res, {cookies}){  
 if(!err)  
 {  
 pm.globals.set('csrf-token', cookies.get('XSRF-TOKEN'))  
 }  
})



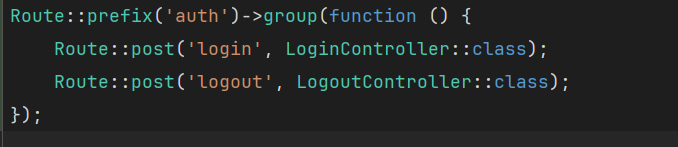
Gevi Avgiv †`L‡ev wKfv‡e logout Ki‡ev| Gi Rb¨ Avgiv

php artisan make:controller Auth/LogoutController -i

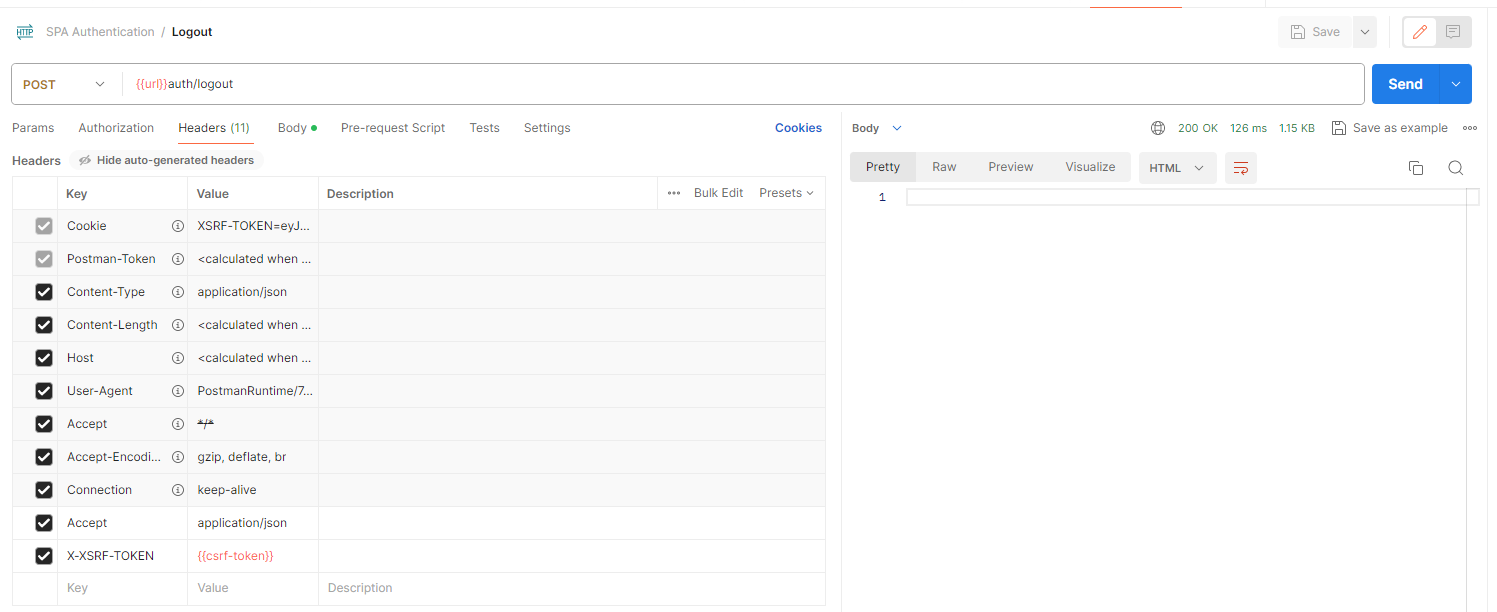
GB Command Uv Pvjv‡ev| Zvici Controller G wM‡q Avgiv GB fv‡e wjL‡evt



Avi Web.php ‡Z wjL‡ev



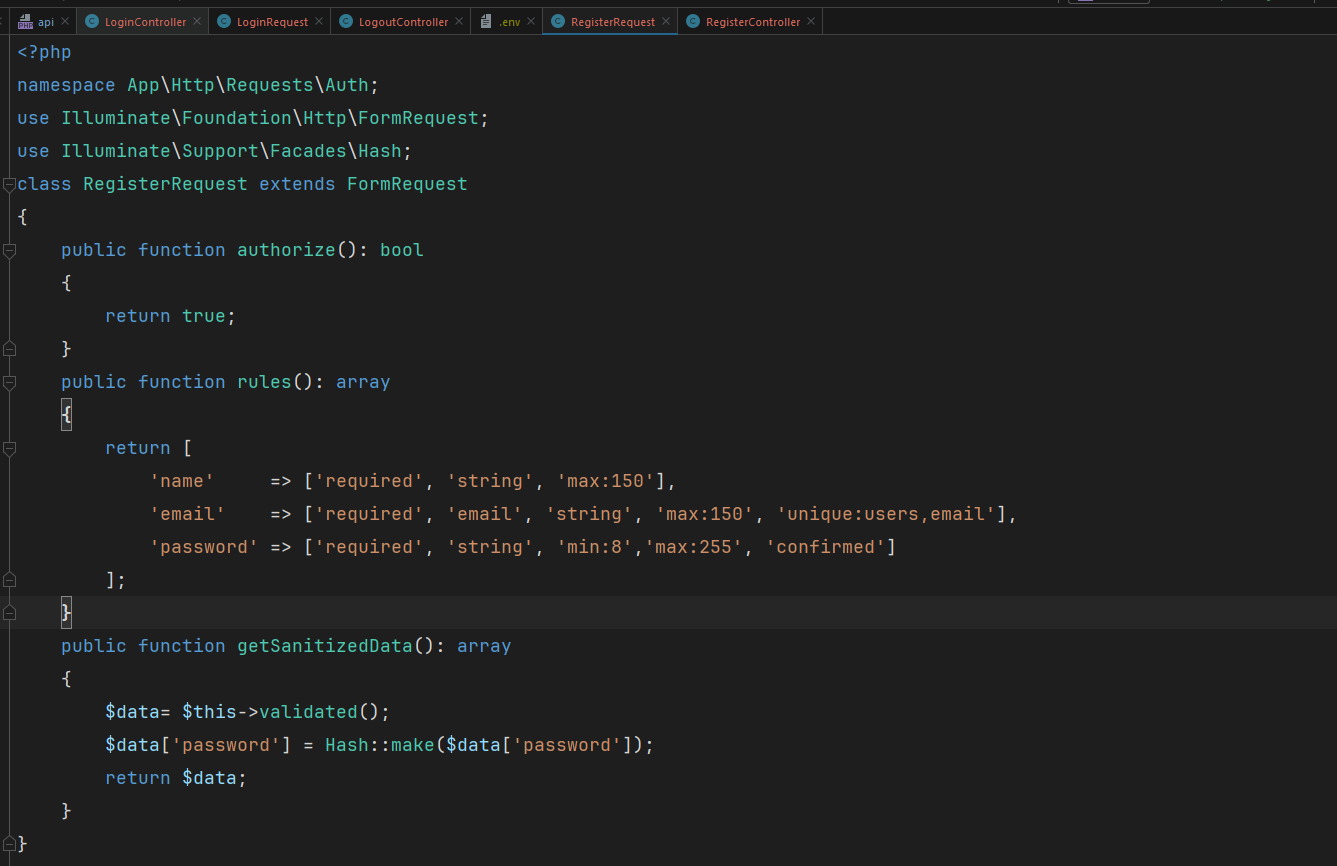
Gici Avgiv PostMan Open K‡i Pre-request Script Gi me wKQz Removed Ki‡ev



Gevi Avgiv †`L‡ev wKfv‡e Register Ki‡ev Zv †`L‡ev| Gi Rb¨ Avgiv IB Command wU Pvjv‡ev|

php artisan make:controller Auth/RegisterController -i

php artisan make:request Auth/RegisterRequest



GB Request G Avgiv HTTPS & Gi Data Validation Check K‡i getSanitizedata. Gici Avgiv register Controller G P‡j hv‡ev| †mLv‡b Avgiv Avgv‡`i Register Gi Rb¨ cÖ‡qvRbxq jwRK wjL‡ev| Ges Database G Data ‡mf Ki‡ev|

<?php  
namespace App\Http\Controllers\Auth;  
use App\Http\Controllers\Controller;  
use App\Http\Requests\Auth\RegisterRequest;  
use App\Models\User;  
use Illuminate\Http\Request;  
class RegisterController extends Controller  
{  
   
 public function \_\_invoke(RegisterRequest $request)  
 {  
 try {  
 $data = User::create($request->getSanitizedData());  
 //$token = $data->createToken('auth\_token')->plainTextToken;  
 return response()->json([  
 'message' => 'User created successfully',  
 'status' => 'success',  
 //'token' => $token,  
 'data' => $data  
 ], 201);  
 } catch (\Exception $e)  
 {  
 return response()->json([  
 'message' => $e->getMessage(),  
 'status' => 'error'  
 ],  
 500);  
 }  
 }  
}

