

***Instalación de un servidor Laravel y utilización de Api Rest***

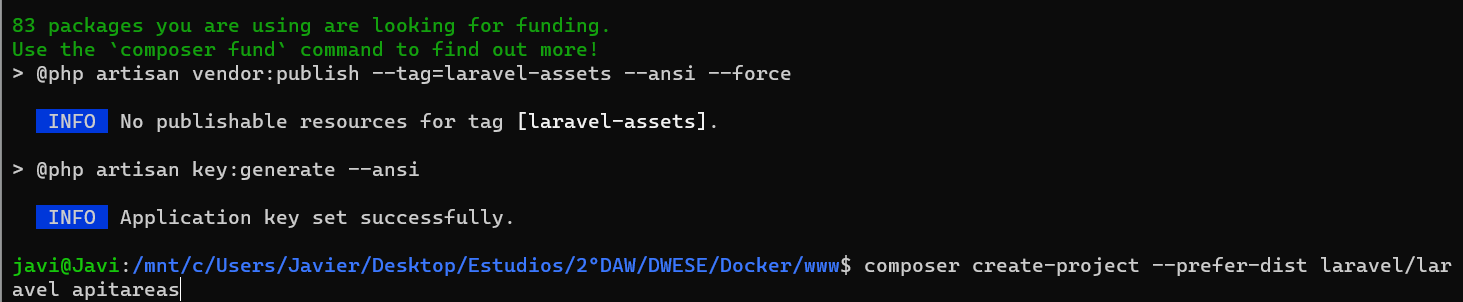
***Fco Javier Gallego Fernández***



***Índice***

Creación del primer proyecto

Hacemos nuestro primer proyecto en laravel, para ello vamos a utilizar el siguiente comando:  
  
**composer create-project --prefer-dist laravel/laravel apitareas**

****

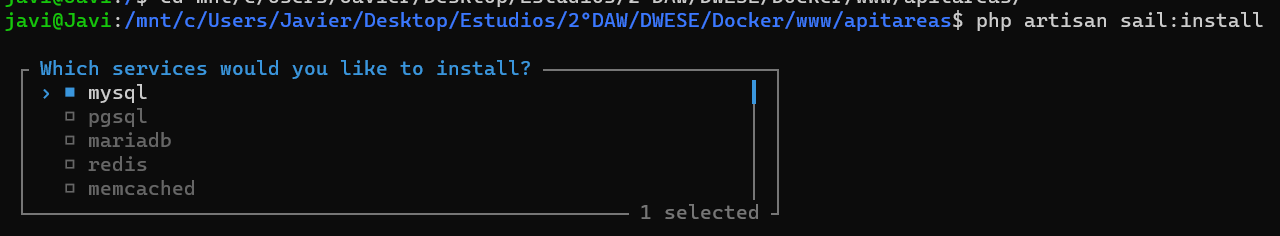
Si no tenemos el composer instalado deberemos ejecutar el comando “**composer install”**

Ahora nos debemos mover a nuestro proyecto con el comando ‘**cd apitareas’**

****

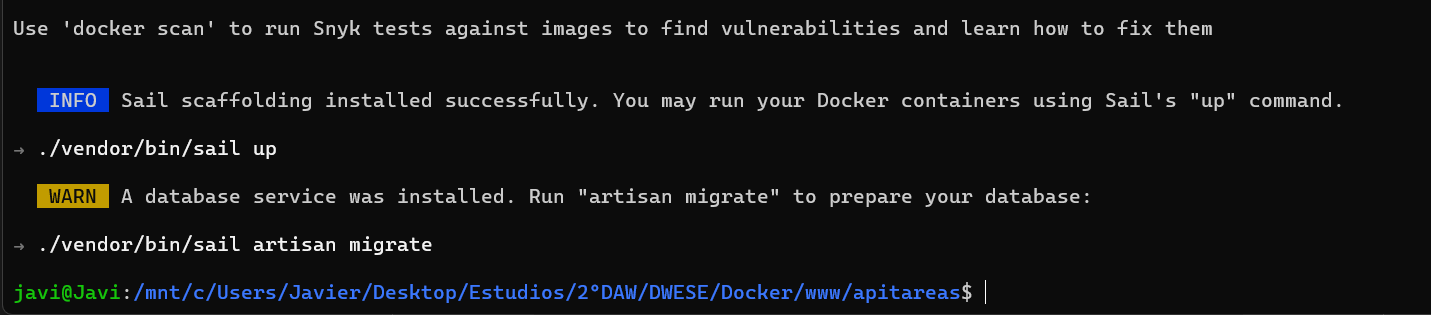
El siguiente paso que tendremos que hacer es instalar Laravel Sail utilizando Artisan con el comando:  
  
‘**php artisan sail:install’**

Al introducir el comando nos preguntara con qué tipo de base de datos queremos trabajar en nuestro proyecto, en nuestro caso utilizaremos mysql.



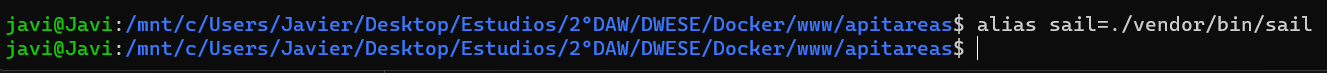
Este proceso entre otras cosas nos va a generar el Docker-compose.yml con el que podremos abrir el contenedor posteriormente.

Cuando obtengamos el siguiente mensaje quiere decir que todo ha ido bien y esta listo para lanzar.



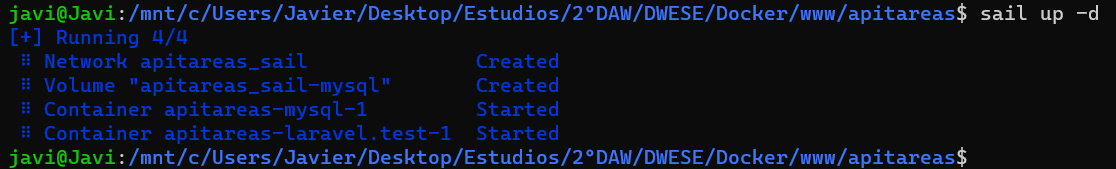
El siguiente paso es levantar el contenedor Docker, para ello tendremos que utilizar el comando '**./vendor/bin/sail up -d'** pero como sail es algo que vamos a tener que estar utilizando le vamos a crear un alias (El cual según distribución puede ser temporal y deberemos de hacer cada vez que arranquemos nuestra máquina Linux) con el siguiente comando:

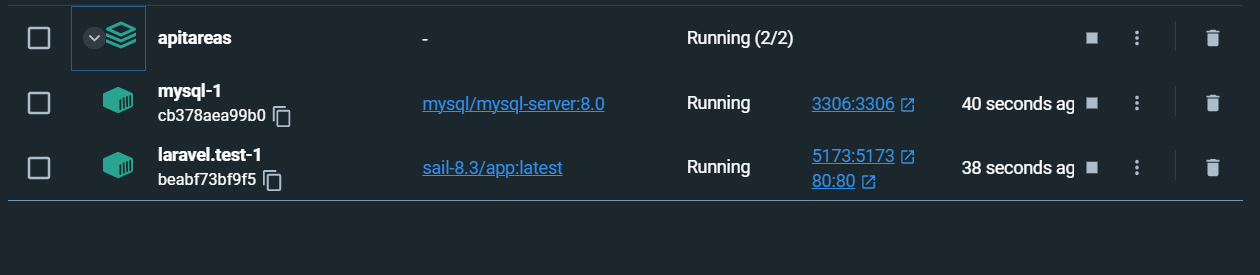
‘**alias sail=./vendor/bil/sail’**

****

Con el alias conseguimos poder escribir ‘**sail’** en lugar de ‘**./vendor/bin/sail’**.

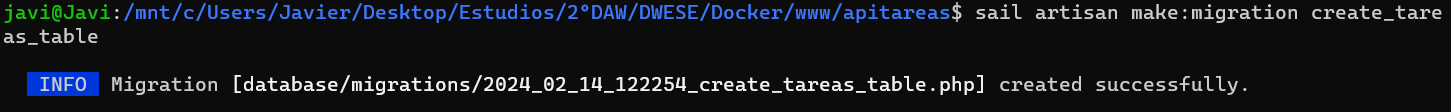
Ahora sí, vamos a levantar el contenedor con el comando ‘**sail up -d’,** recuerda tener antes abierto el Docker desktop abierto:



Si vamos a nuestro Docker podemos comprobar que el contenedor esta funcionando:  


Creación de la tabla tareas

Ahora vamos a crear la tabla tareas, para ello deberemos ejecutar el siguiente comando:  
‘**sail artisan make:migration create\_tareas\_table’**

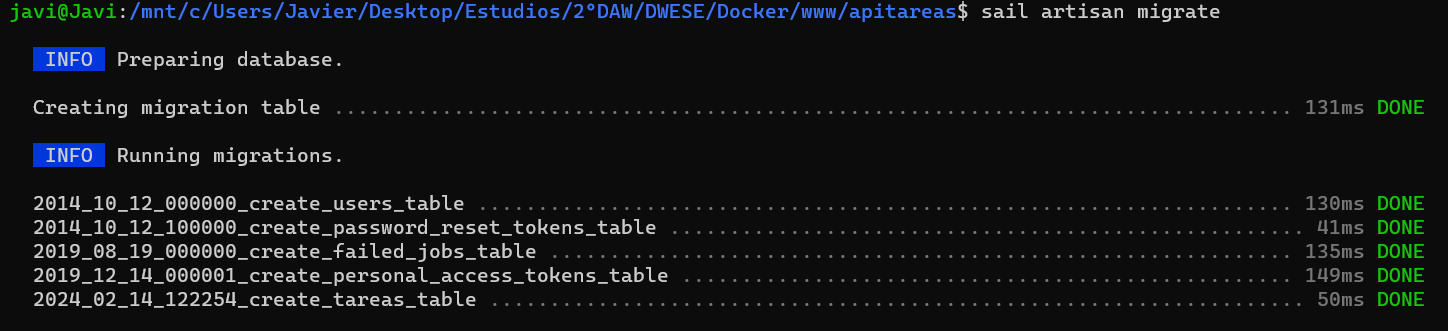


Esto nos crea un archivo en nuestro proyecto **apitareas->database->migrations->%fecha%create\_tareas\_table.php**

Aquí tenemos que definir la estructura de nuestra tabla, en nuestro caso vamos a definir que tenga un id, titulo y descripción.

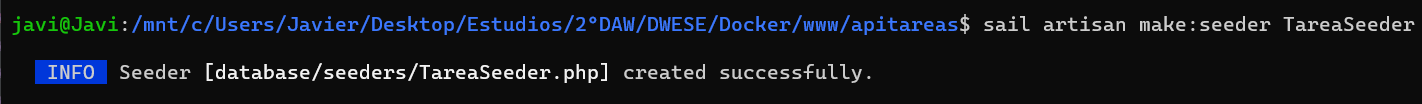


Para crear la tabla deberemos ejecutar el comando ‘**sail artisan migrate’**

****

Introducción de datos en la tabla

Ahora vamos a introducir los datos de nuestra tabla, para ello debemos crear el seeder con el siguiente comando:  
  
‘**sail artisan make:seeder TareaSeeder’**

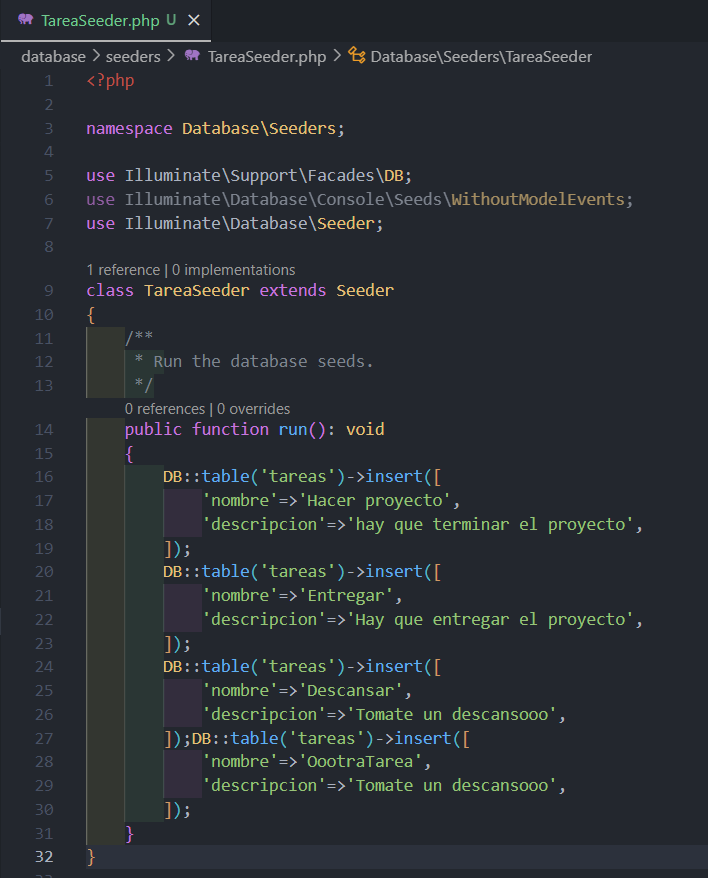


Como bien nos dice el mensaje que devuelte esto nos crea el archivo en nuestro proyecto en **database/seeders/TareasSeeder.php**

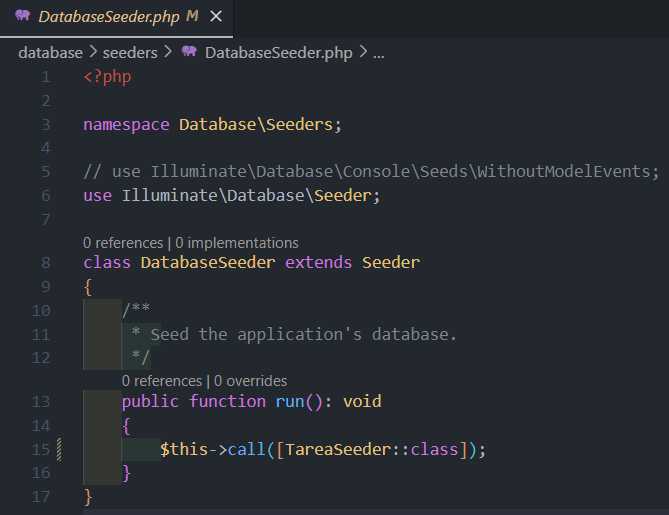
En dicho archivo deberemos importar el archivo DB para ello al inicio incluimos lo siguiente:

‘**use Illuminate\Support\Facades\DB;’**

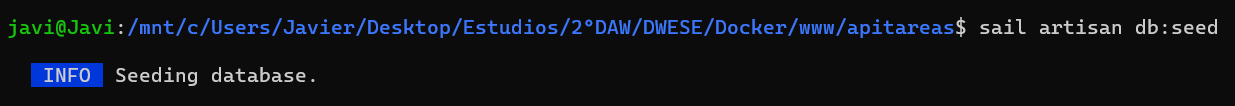
En el archivo vamos a incluir las rows que queramos en nuestra tabla, el archivo debería quedar de la siguiente manera:



Por último en DatabaseSeeder.php debemos añadir ‘**$this->call([TareaSeeder::class]);’**

****

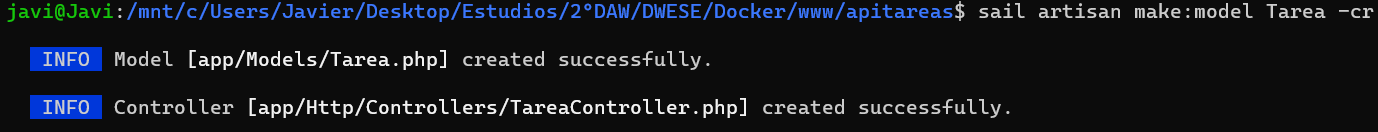
Para hacer los inserts en nuestra DB que hemos incluido en nuestro archivo deberemos ejecutar el comando **‘sail artisan db:seed’**

****

Creación del modelo Tarea

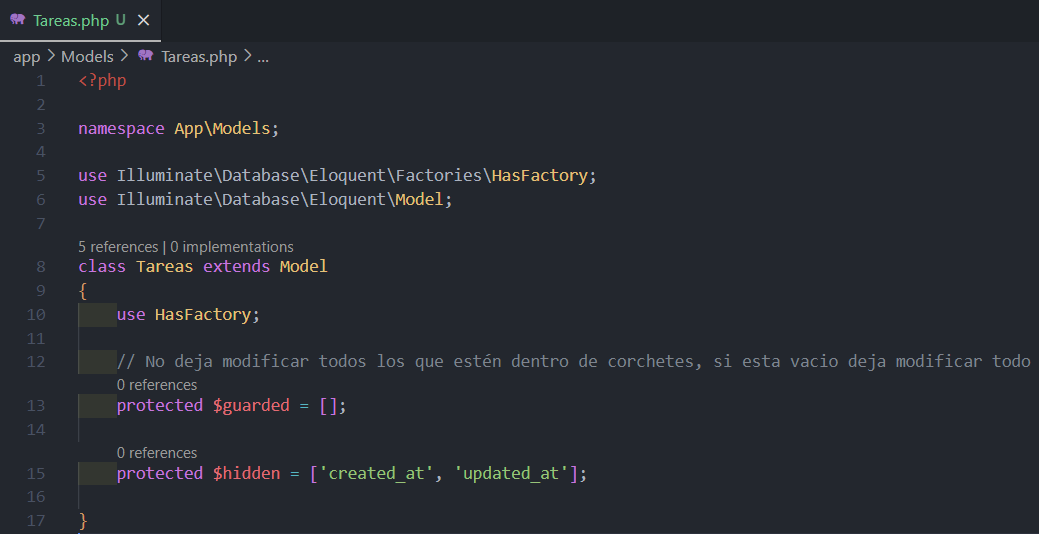
Ahora vamos a crear el modelo de tareas, para ello vamos a utilizar el comando

**‘sail artisan make:model Tarea -cr’**



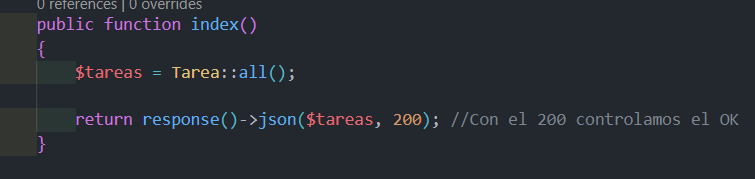
Con esto creamos el modelo Tarea.php y el controller.

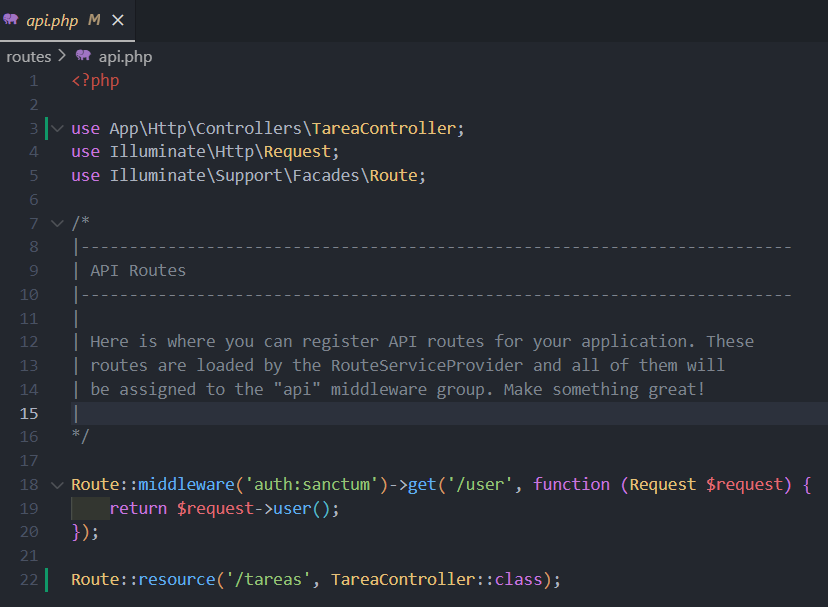
Vamos a empezar a maquetar el modelo Tarea.php. En nuestro caso simplemente le diremos qué puede modificar y los ocultos que se van a rellenar solo.



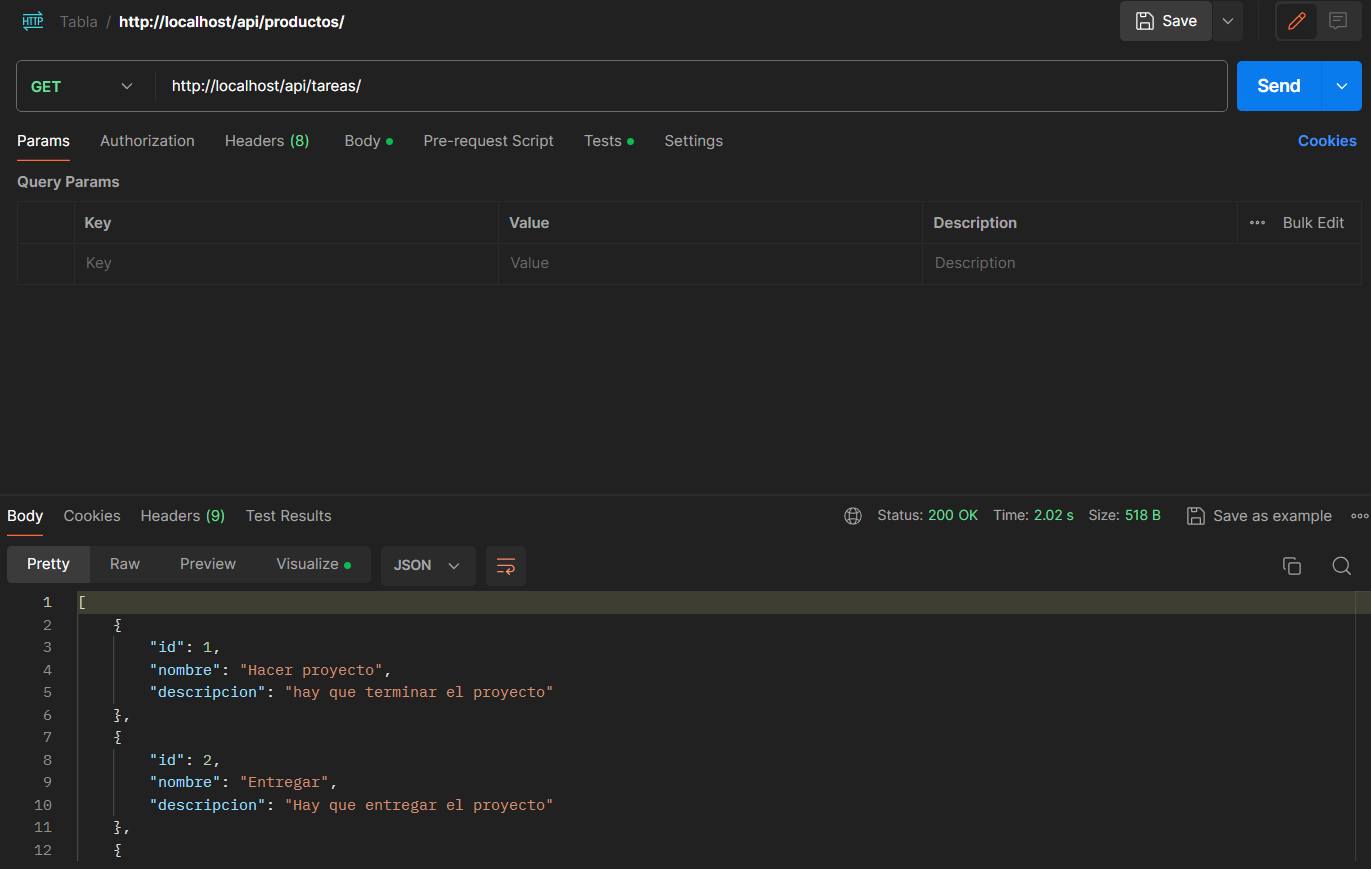
Antes de modificar el controller, vamos a crear el request, el resource y la tabla etiquetas para no tener que volver a modificar el controller.

En el caso que queramos comprobar que todo va correcto, podemos hacerlo añadiendo lo siguiente en TareaController.php



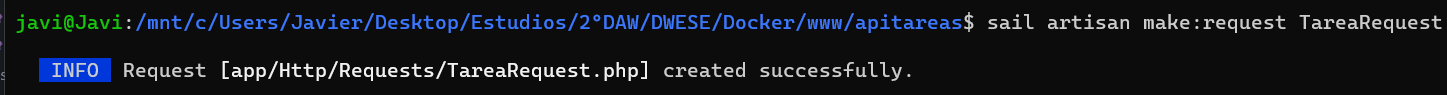
Y completar nuestra api.php añadiendo la siguiente línea:  
 **‘Route::resource('/tareas', TareaController::class);’**  


Si accedemos al postman con el método get en localhost/api/tareas comprobamos que nos devuelve correctamente los datos.



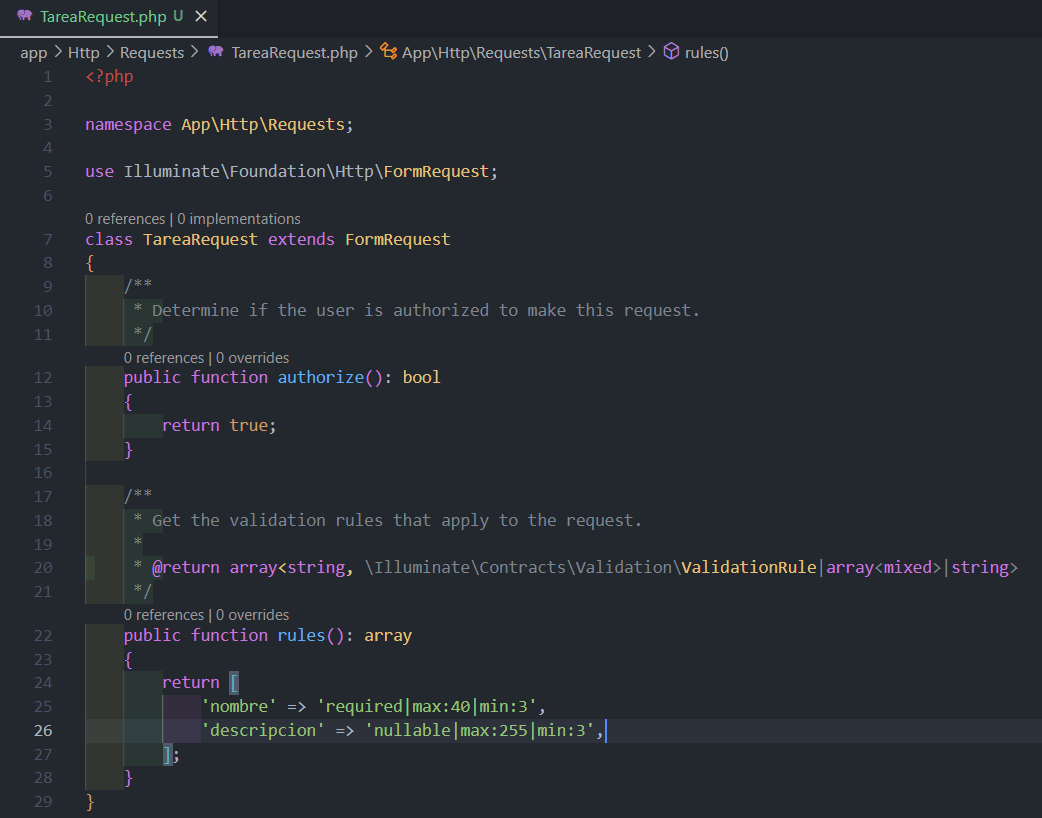
Creación TareaRequest

Vamos a crear nuestro request, para ello vamos a utilizar el comando ‘**sail artisan make:request TareaRequest’**



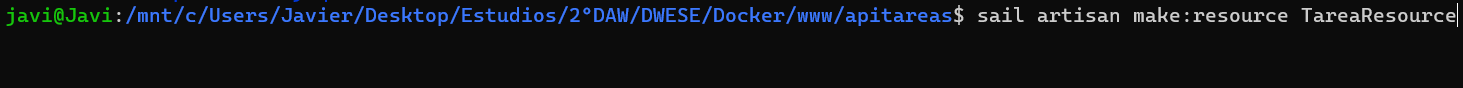
Esto nos crea las requests en **TareaRequest.php.**

En dicho requests le debemos poner en true la función authorize y añadimos las rules que queremos que deba tener los request

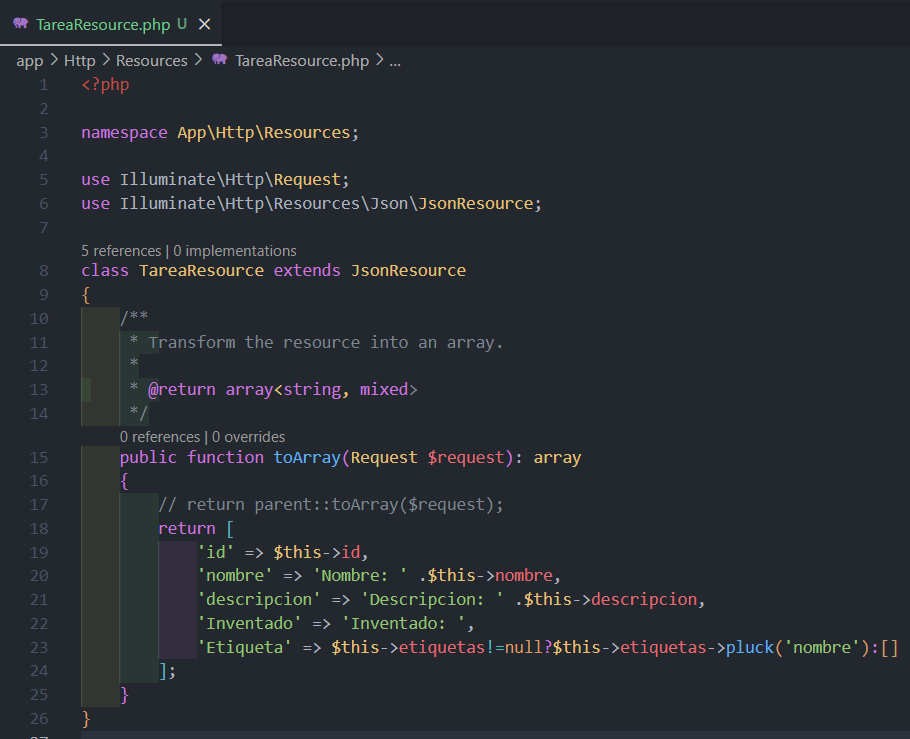


Creación TareaResource

Ahora vamos a crear nuestro resource, para ello utilizamos el comando ‘**sail artisan make:resource TareaResource’**

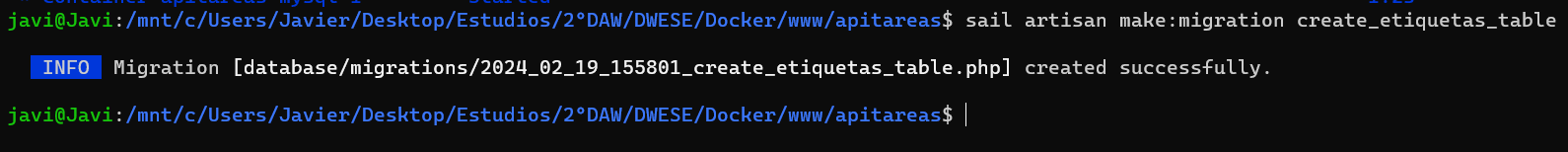
****

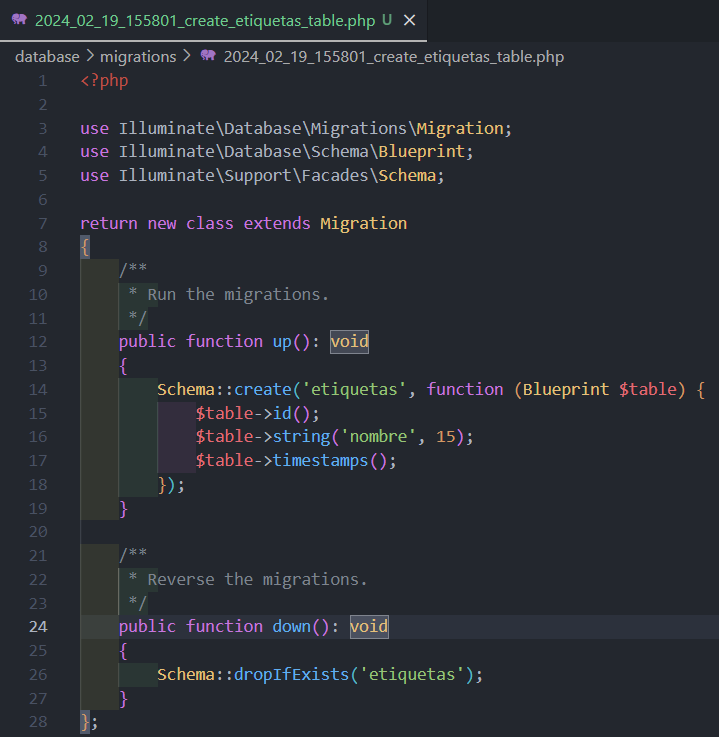
Esto nos crea el archivo TareaResource. En dicho archivo vamos a decirle la estructura que queremos.



Creación de la tabla Etiquetas

Ahora vamos a crear la tabla etiquetas, para ello vamos a utilizar el comando ‘**sail artisan make:migration create\_etiquetas\_table’**



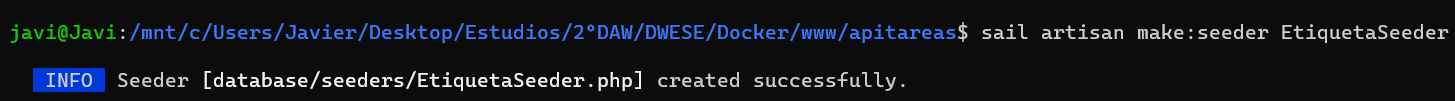
Y ahora maquetamos la estructura de la tabla:  


Y para crear las tablas usaremos el comando ‘**sail artisan migrate’**



Introducción de datos en la tabla Etiquetas

Ahora vamos a introducir los datos de nuestra tabla, para ello debemos crear el seeder con el siguiente comando:  
  
‘**sail artisan make:seeder EtiquetaSeeder’**

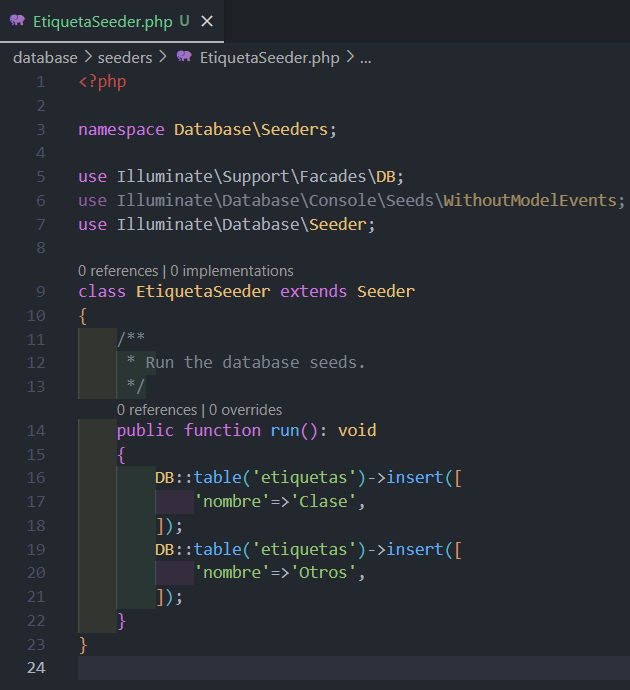


Como bien nos dice el mensaje que devuelte esto nos crea el archivo en nuestro proyecto en **database/seeders/EtiquetaSeeder.php**

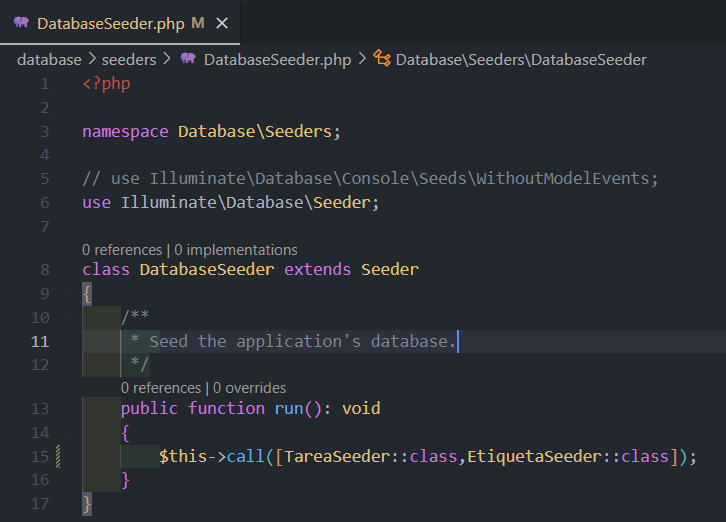
En dicho archivo deberemos importar el archivo DB para ello al inicio incluimos el lo siguiente:

‘**use Illuminate\Support\Facades\DB;’**

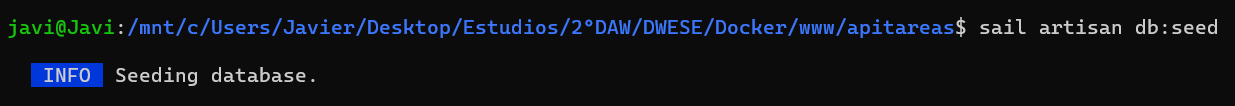
En el archivo vamos a incluir las rows que queramos en nuestra tabla, el archivo debería quedar de la siguiente manera:

****

Por último en DatabaseSeeder.php debemos añadir ‘**EtiquetaSeeder::class**’ de manera que nos quede ‘**$this->call([TareaSeeder::class,EtiquetaSeeder::class]);’**

****

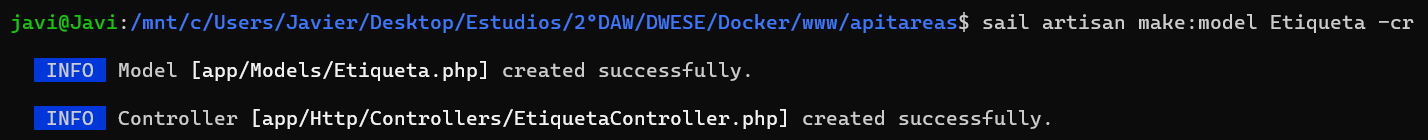
Para hacer los inserts en nuestra DB que hemos incluido en nuestro archivo deberemos ejecutar el comando **‘sail artisan db:seed’**

****

Creación del modelo Etiqueta

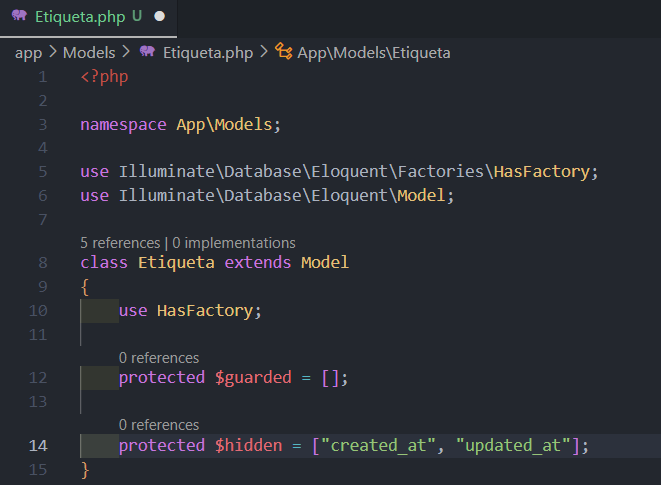
Ahora vamos a crear el modelo de etiquetas, para ello vamos a utilizar el comando

**‘sail artisan make:model Etiqueta -cr’**



Con esto creamos el modelo Etiqueta.php y el controller.

Vamos a empezar a maquetar el modelo Etiqueta.php. En nuestro caso simplemente le diremos qué puede modificar y los ocultos que se van a rellenar solo.

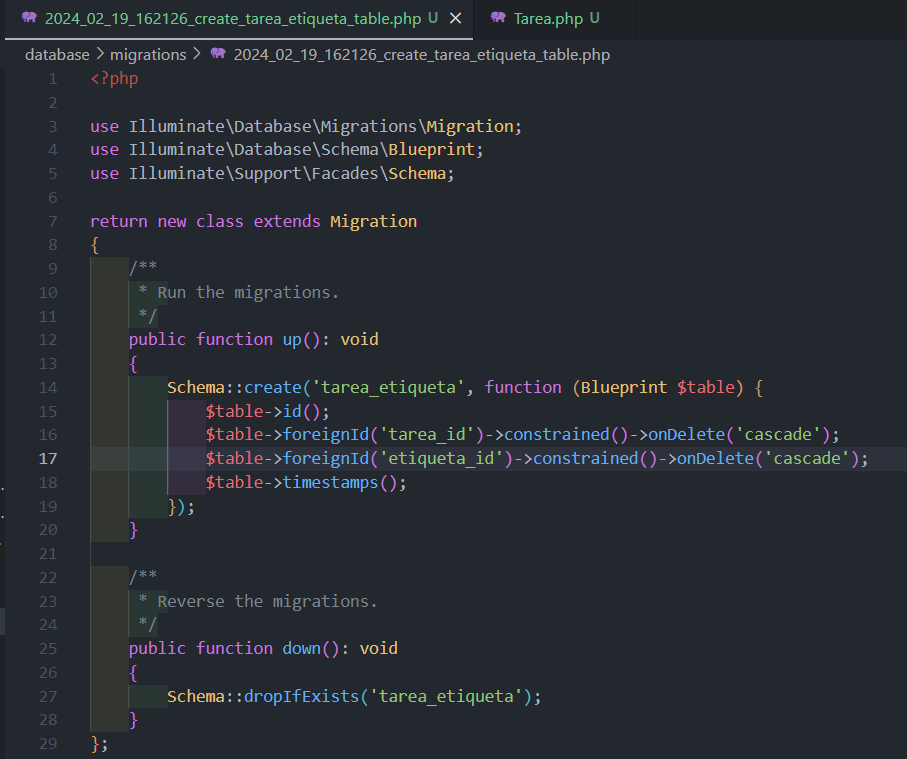


Relacional entre Tareas y Etiquetas

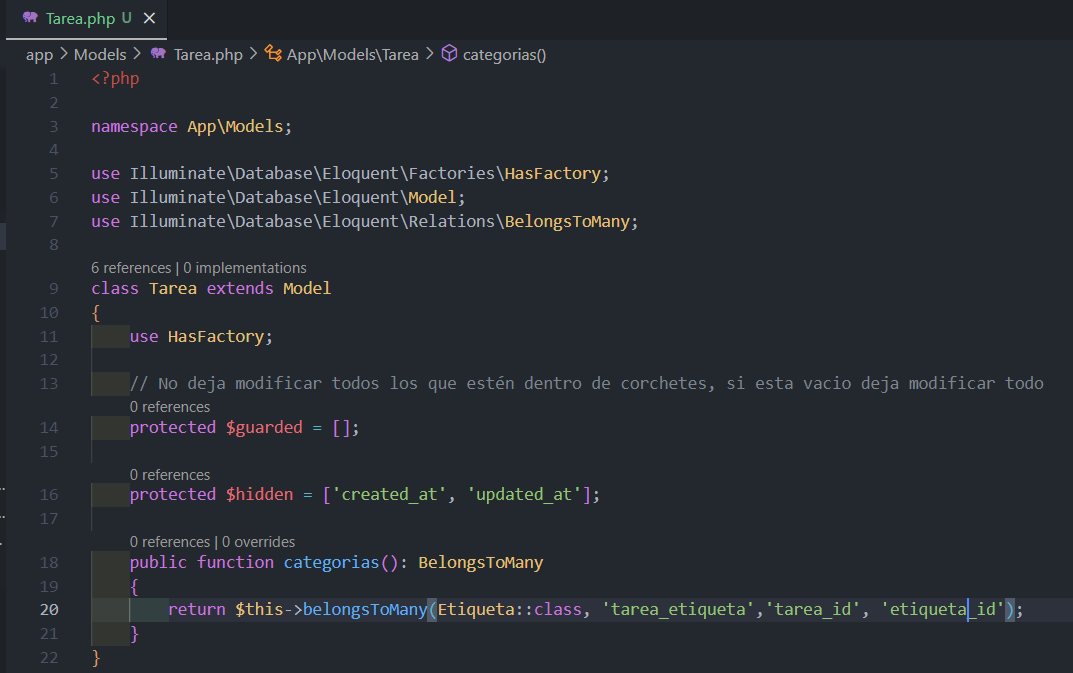
Ahora vamos a crear la tabla referencial, para ello vamos a utilizar el comando ‘**sail artisan make:migration create\_tarea\_etiqueta\_table’**



Ahora maquetamos la estructura de la tabla:

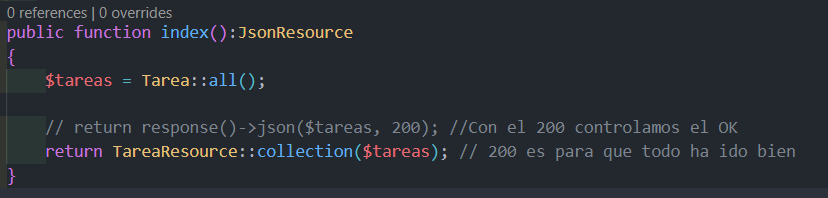


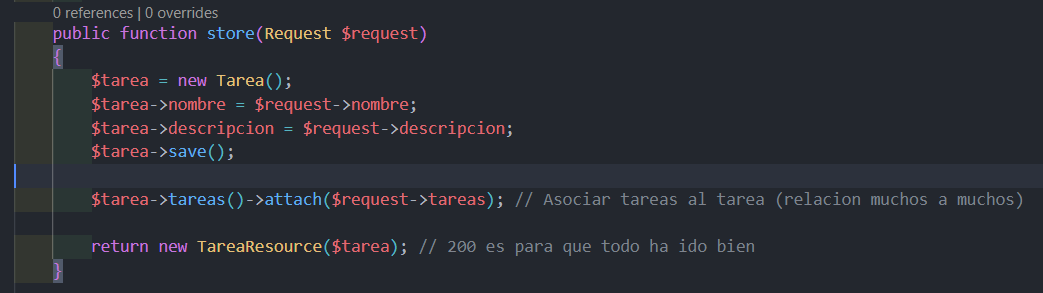
En el modelo Tarea.php debemos crear la relacional, para ello añadimos “**Function categorías(): BelongsToMany**” y especificamos tabla relacional, el id de la primera y el id de la segunda

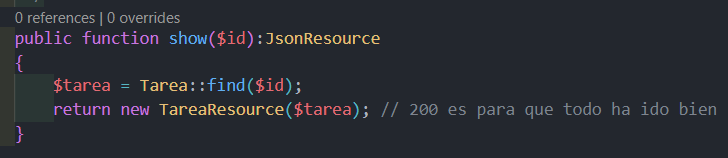


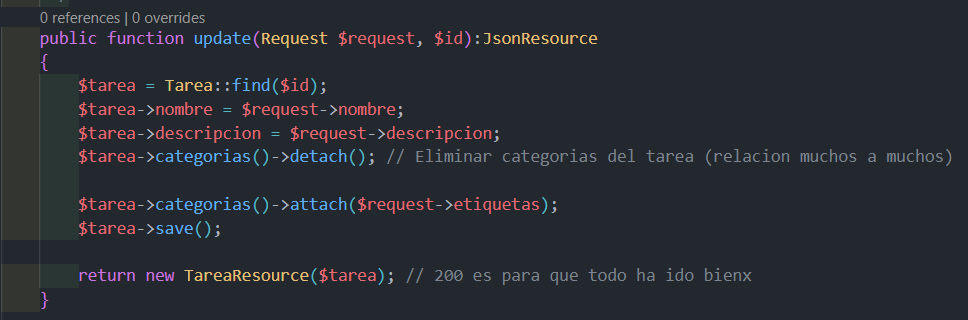
Maquetación TareaController

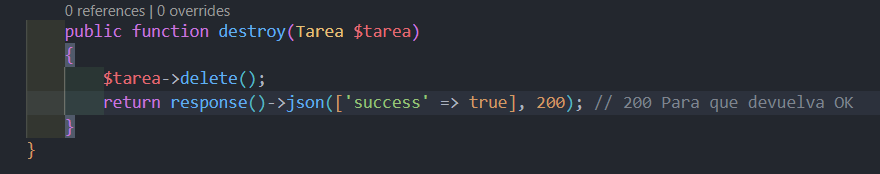
Ahora si podemos completar nuestro TareaController.php:

Function index:  


Function store:  


Function show:  


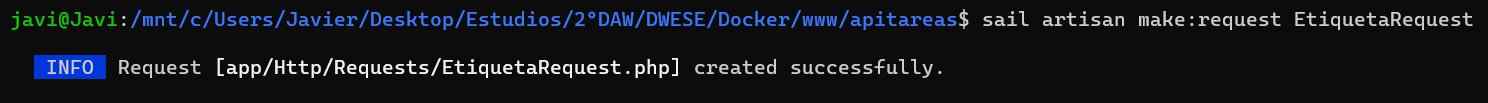
Function update:  


Function destroy:  


Ahora vamos a crear EtiquetaRequest y EtiquetaResource. Para ello seguiremos los pasos igual que antes con Tarea.

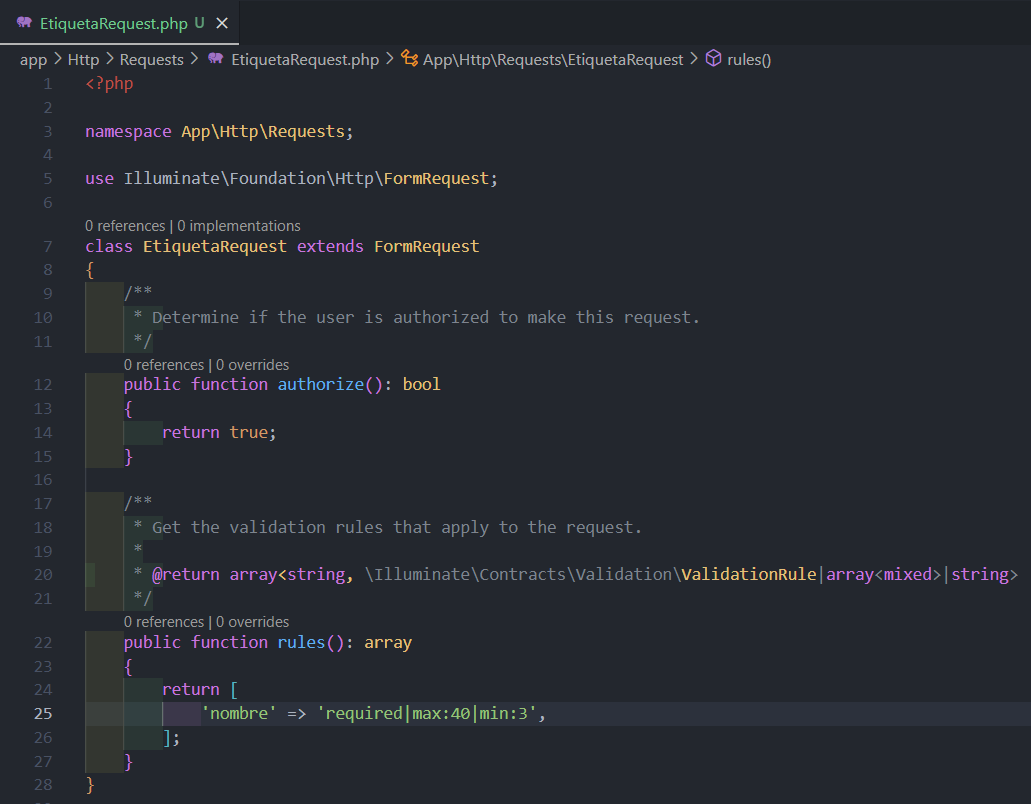
Creación EtiquetaRequest

Para crear nuestro request vamos a utilizar el comando ‘**sail artisan make:request EtiquetaRequest’**



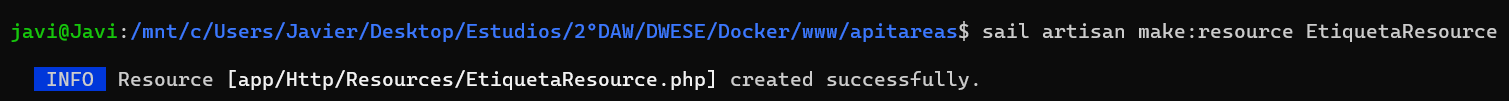
Esto nos crea las requests en EtiquetaRequest.php.

En dicho requests le debemos poner en true la función authorize y añadimos las rules que queremos que deba tener los request

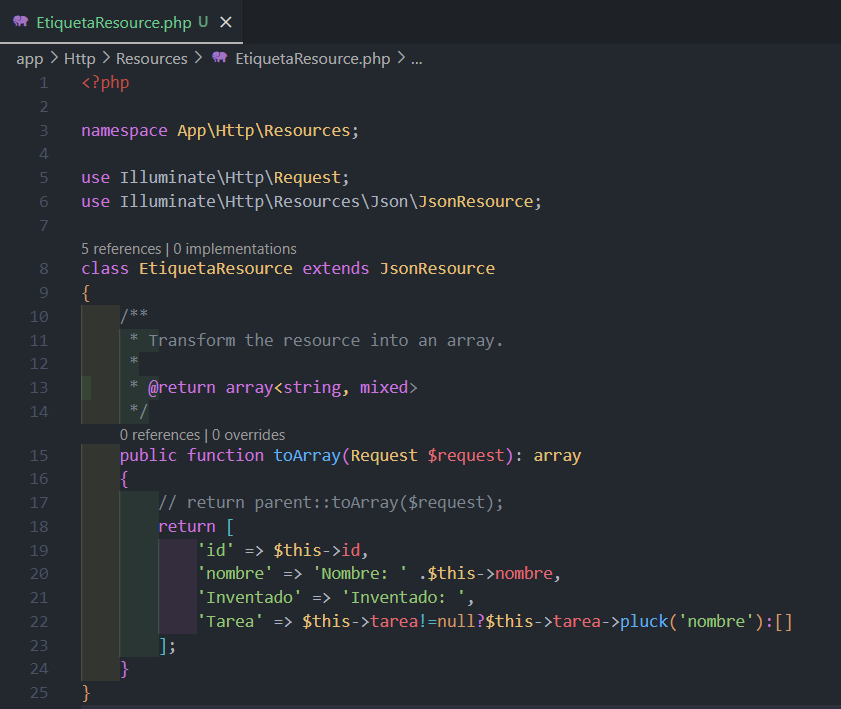


Creación EtiquetaResource

Ahora vamos a crear nuestro resource, para ello utilizamos el comando ‘**sail artisan make:resource EtiquetaResource’**

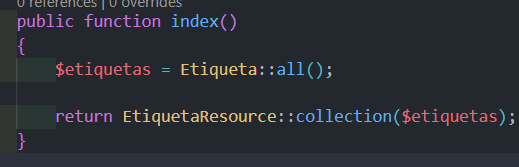
****

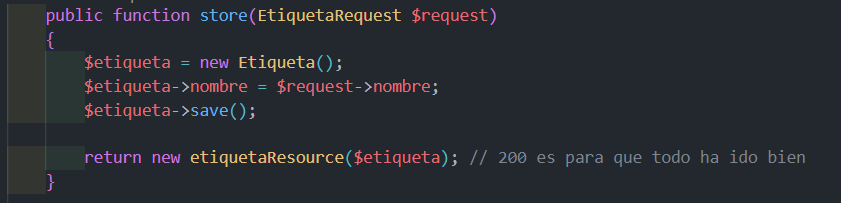
Esto nos crea el archivo TareaResource. En dicho archivo vamos a decirle la estructura que queremos.

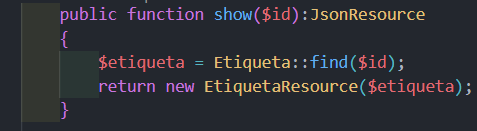


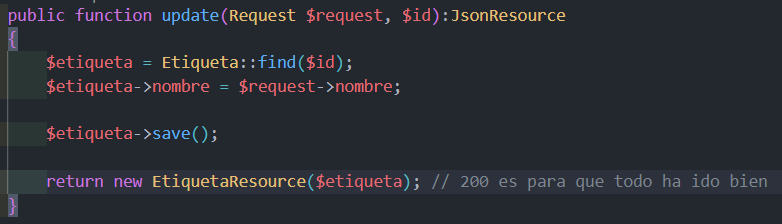
Maquetación EtiquetaController

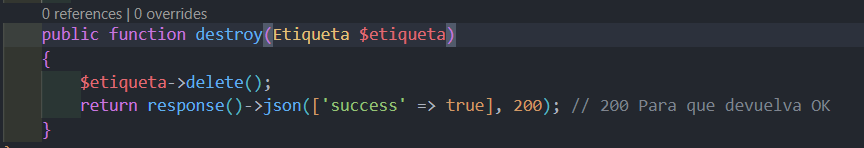
Por último vamos a maquetar EtiquetaController.php al igual que con TareaController.

Function index:  


Function store:  


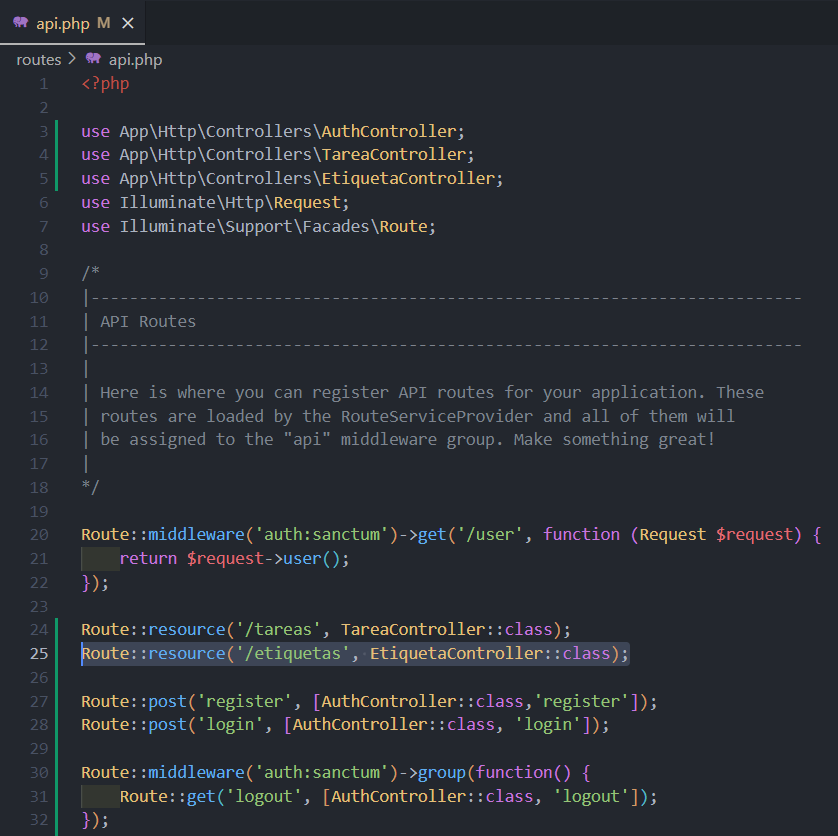
Function show:  


Function update:  


Function destroy:  


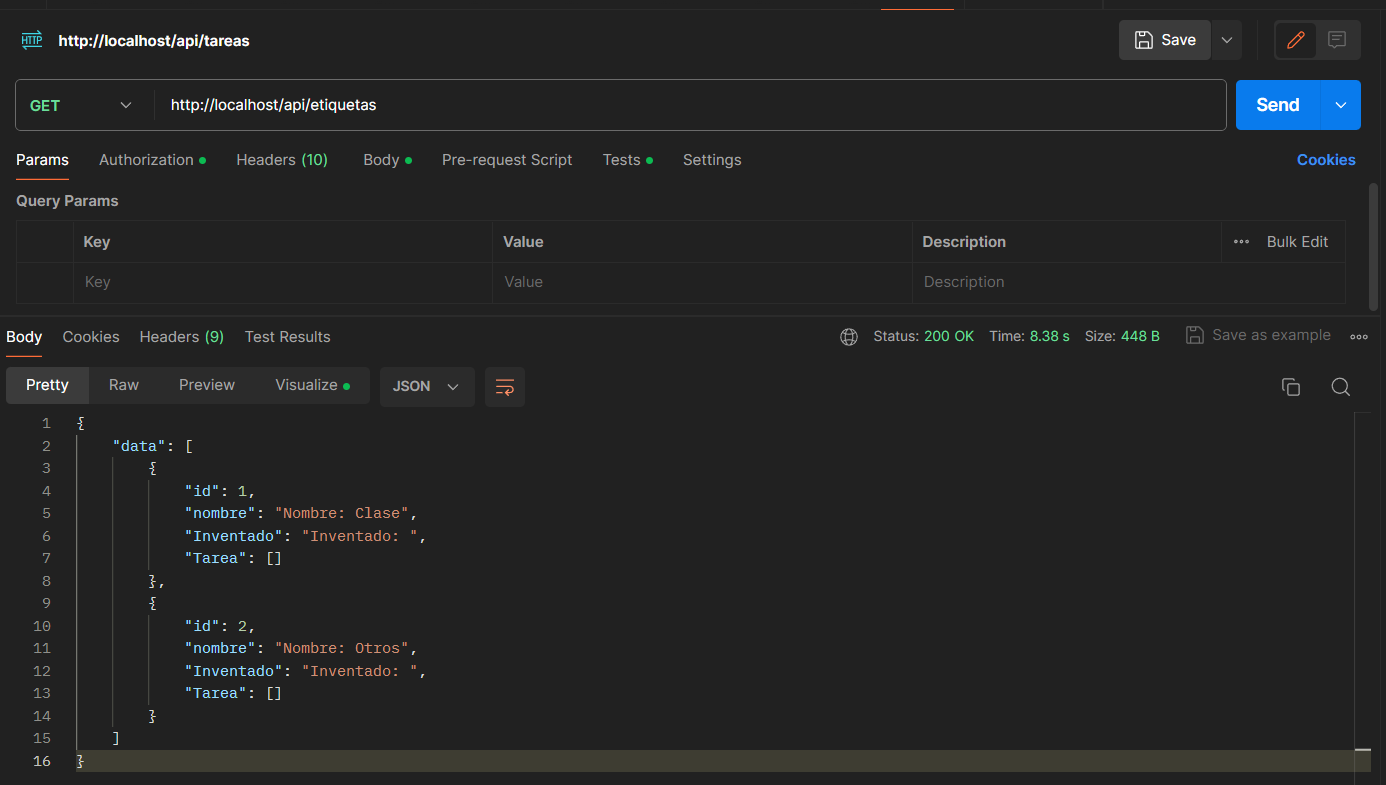
Route Api

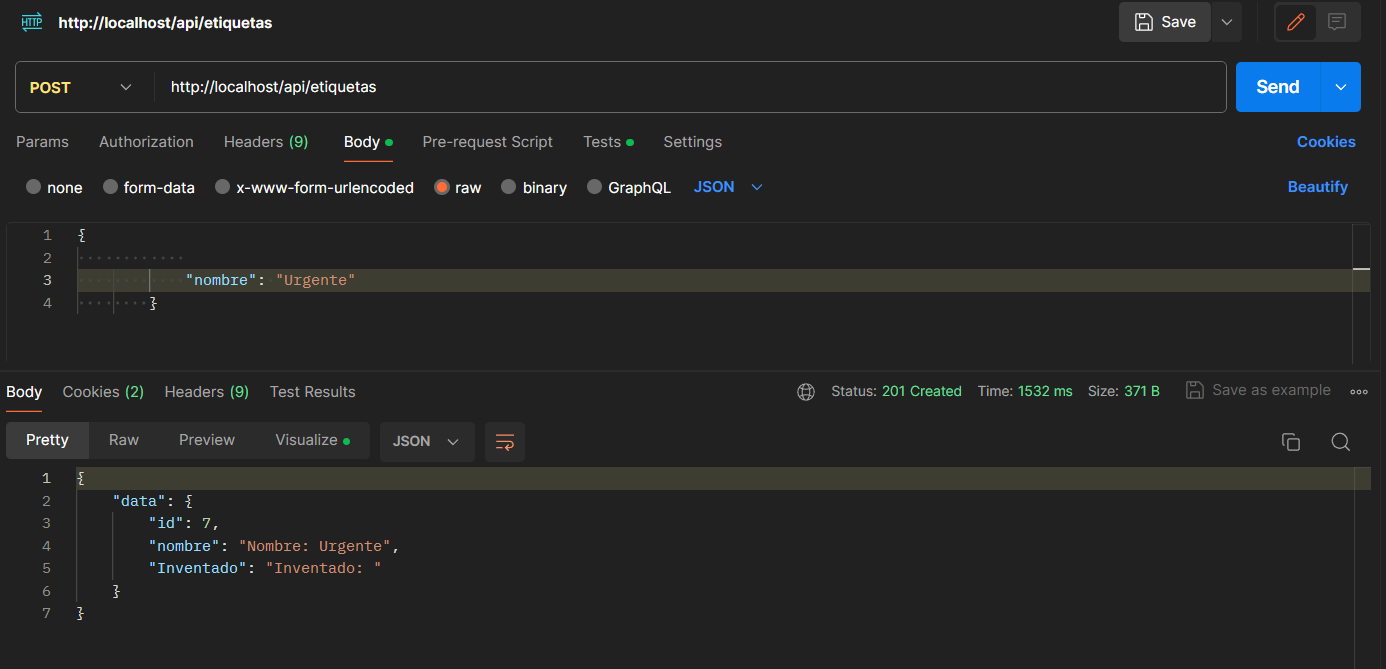
Por último añadimos en api.php el route para etiquetas al igual que con tareas

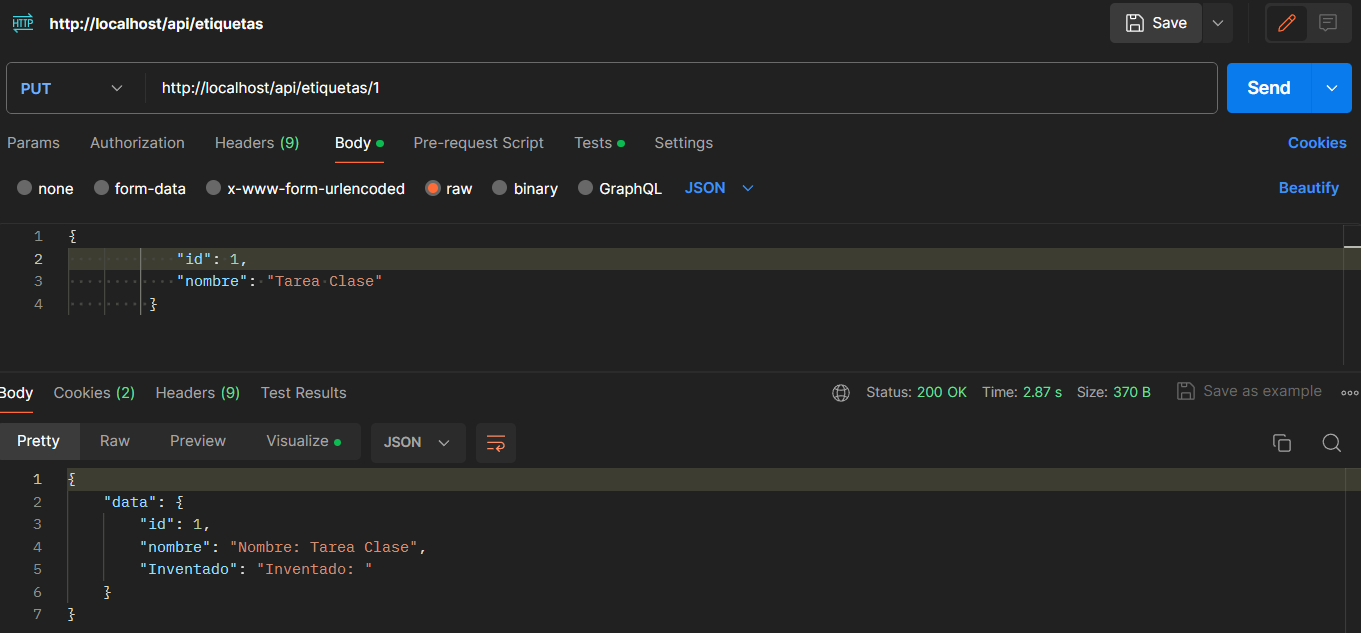


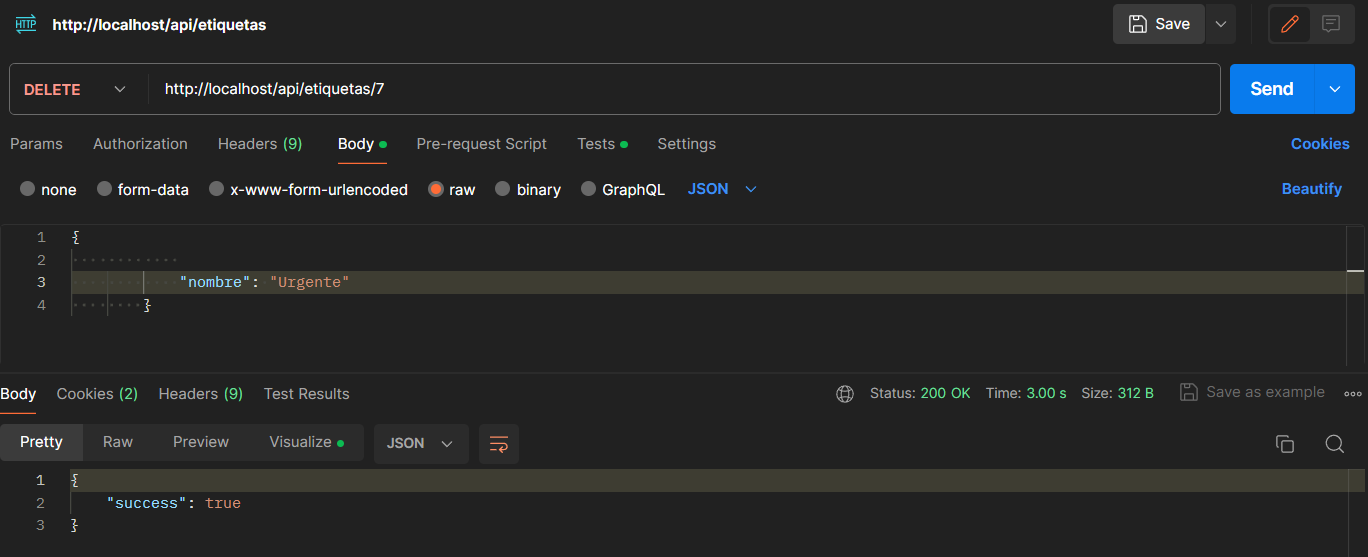
Comprobando con Postman

Vamos a comprobar que todo funcione con el postman.

Si hacemos un get a etiquetas nos debería aparecer los datos en JSON:  


Para añadir etiquetas podemos hacer un post con los datos y se agregará a la base de datos:  


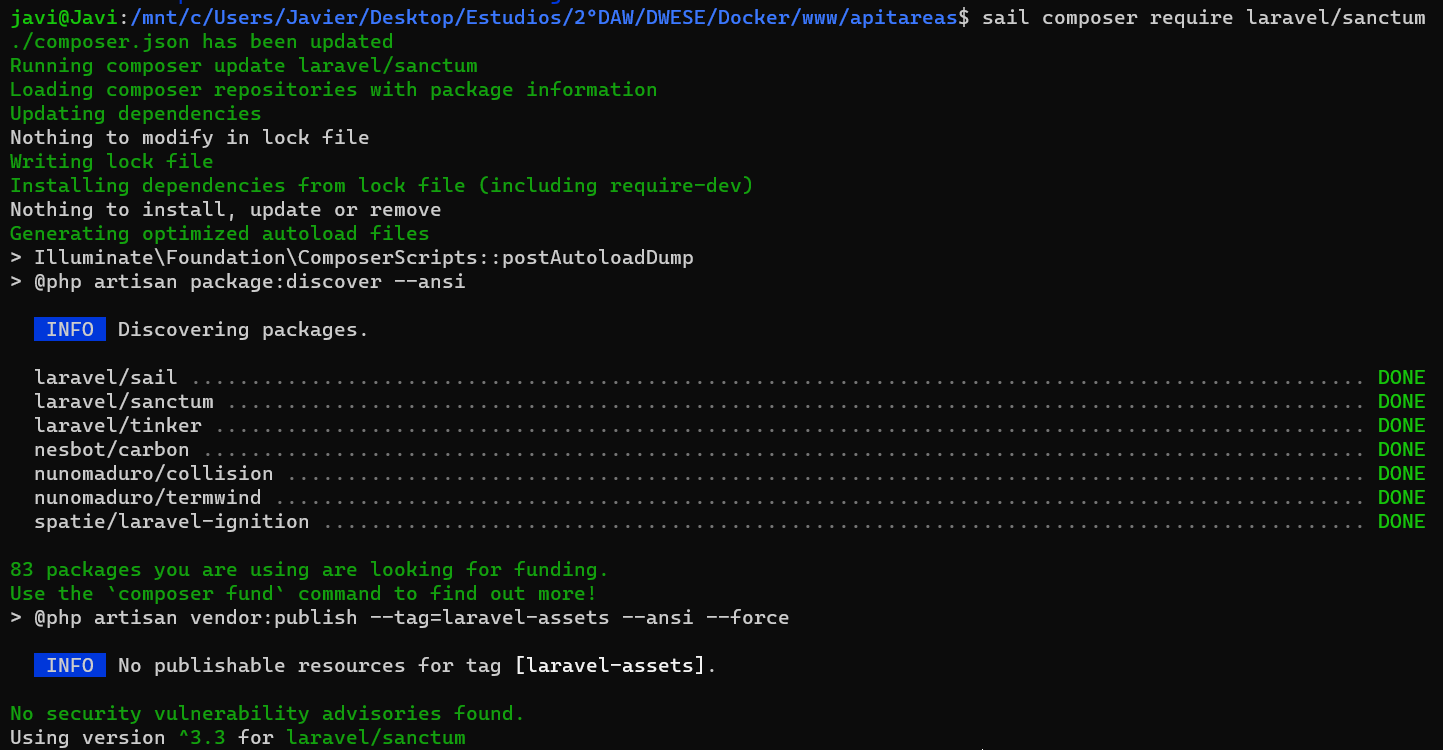
Para hacer un update, en el body con estilo raw le escribimos el update:  


Y si queremos borrar una etiqueta simplemente le pasaremos el id en la url:  


Creación login API REST

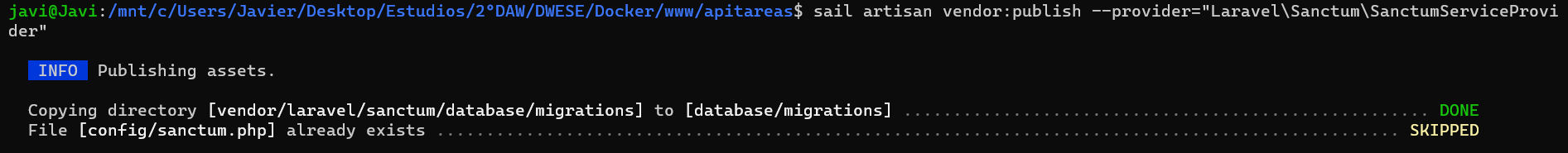
Para crear nuestro login con api rest debemos realizar el laravel/sactum, para ello debemos ejecutar el comando:

**‘sail composer require laravel/sactum’**



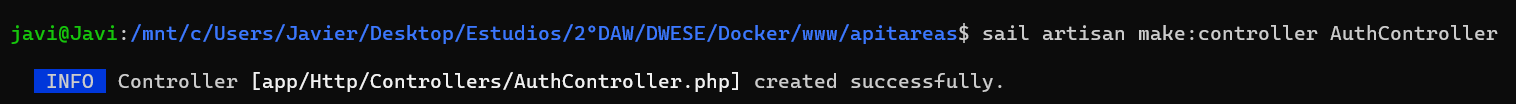
Ahora vamos a tomar la configuración de Laravel Sactum en nuestro proyecto, para ello deberemos ejecutar el siguiente comando  
  
***sail artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvi***

***der"***

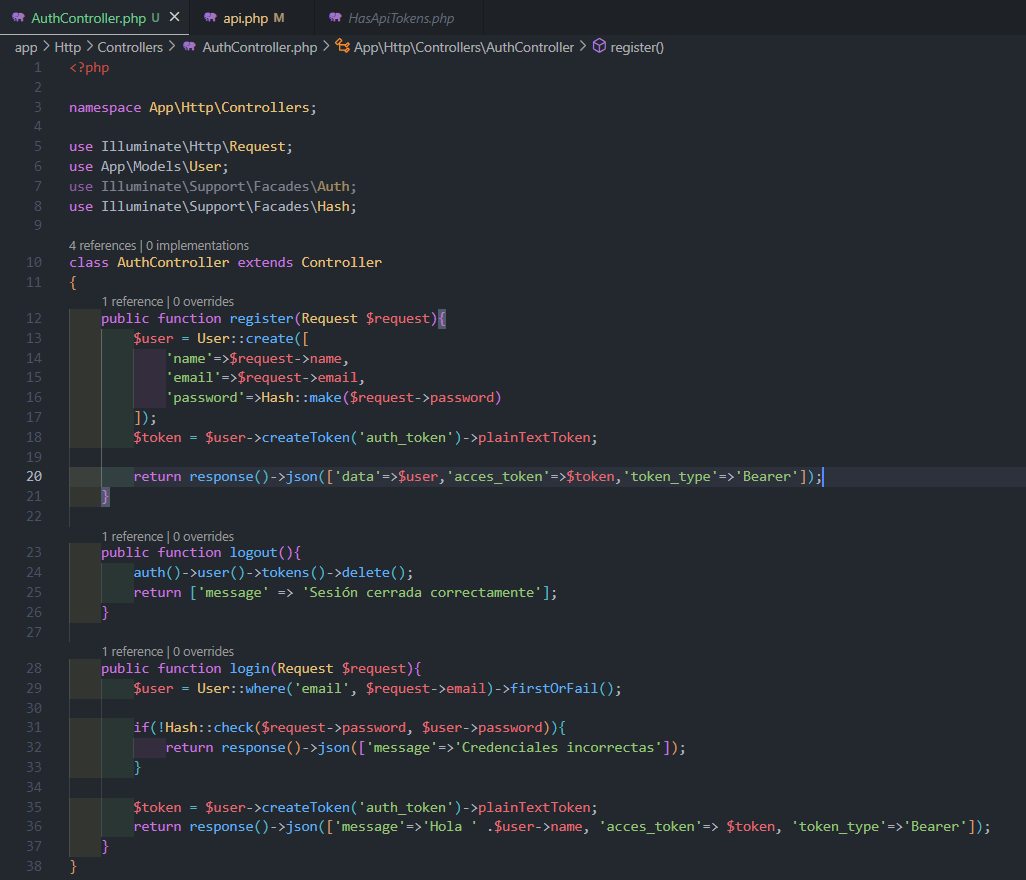
****

Controlador AuthController

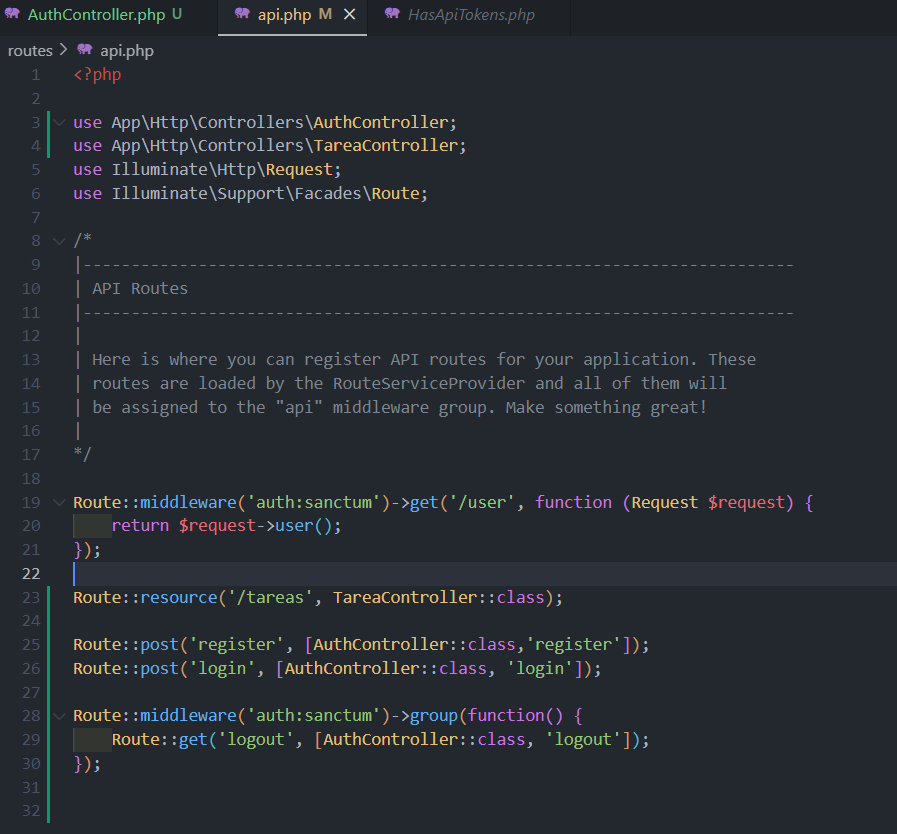
Ahora vamos a hacer un controlador para el login, logout y register, para ello deberemos ejecutar el comando **‘sail artisan make:controller AuthController’**

****

Esto nos crea un archivo un archivo en Controllers llamado **AuthController.php**

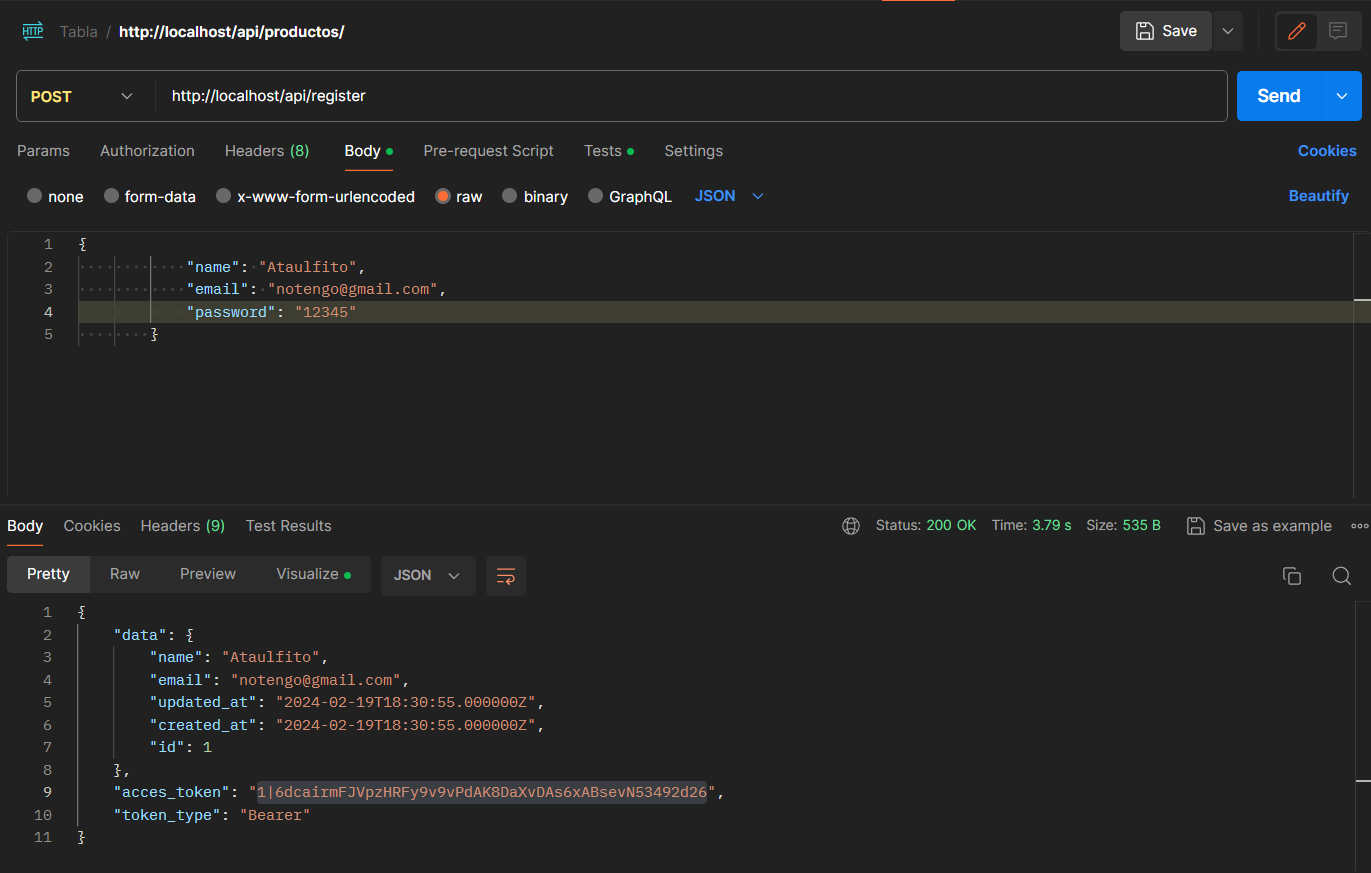
En dicho archivo introduciremos las funciones de login, logout y register. Para ello vamos a completar el archivo de la siguiente manera:  
  


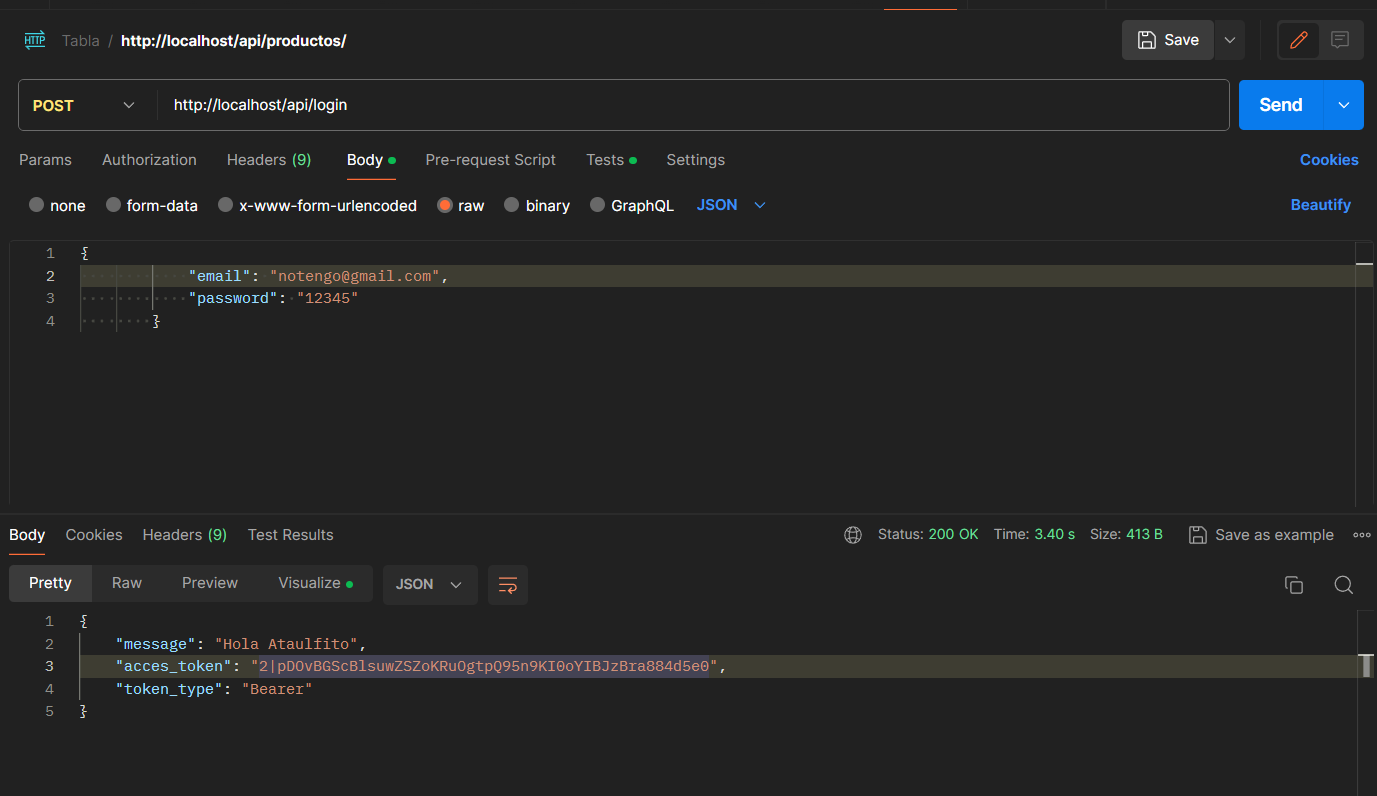
Route api AuthController

Ahora en api.php debemos añadir los route para register, login y logout:  
  


Comprobando Auth con postman

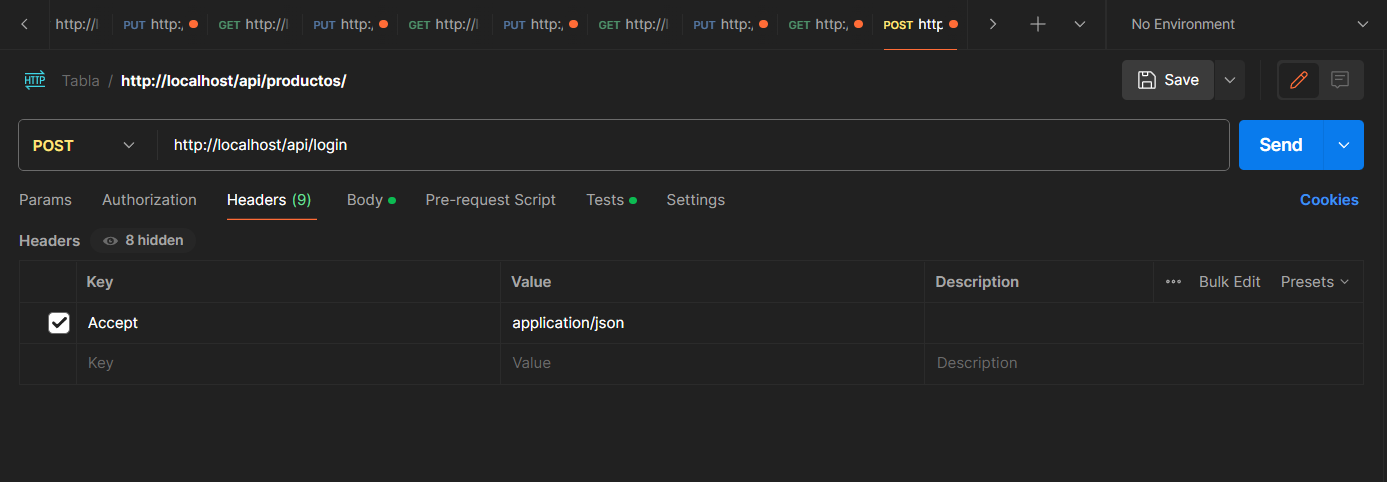
Para comprobar que funciona correctamente vamos al postman, haciendo un post al register de la siguiente manera nos devolvera el acces\_token

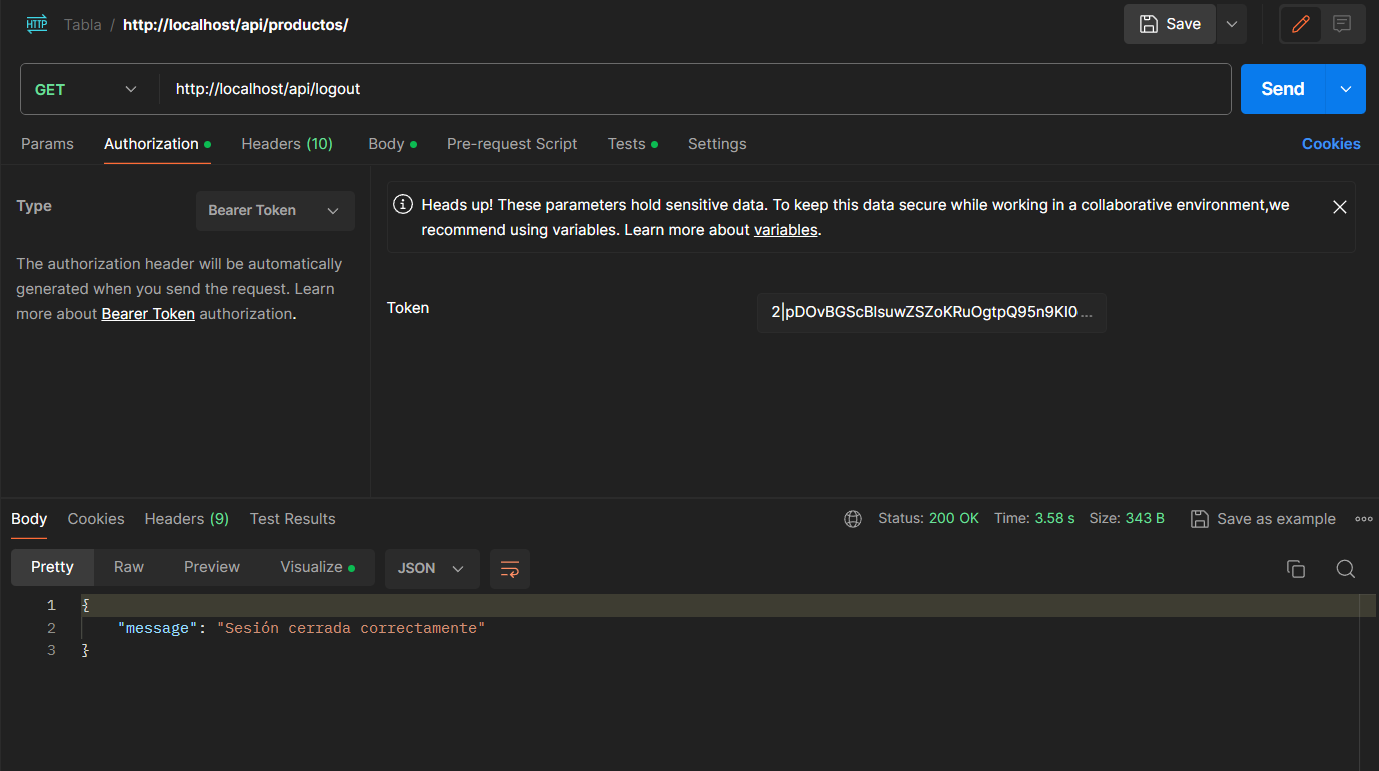


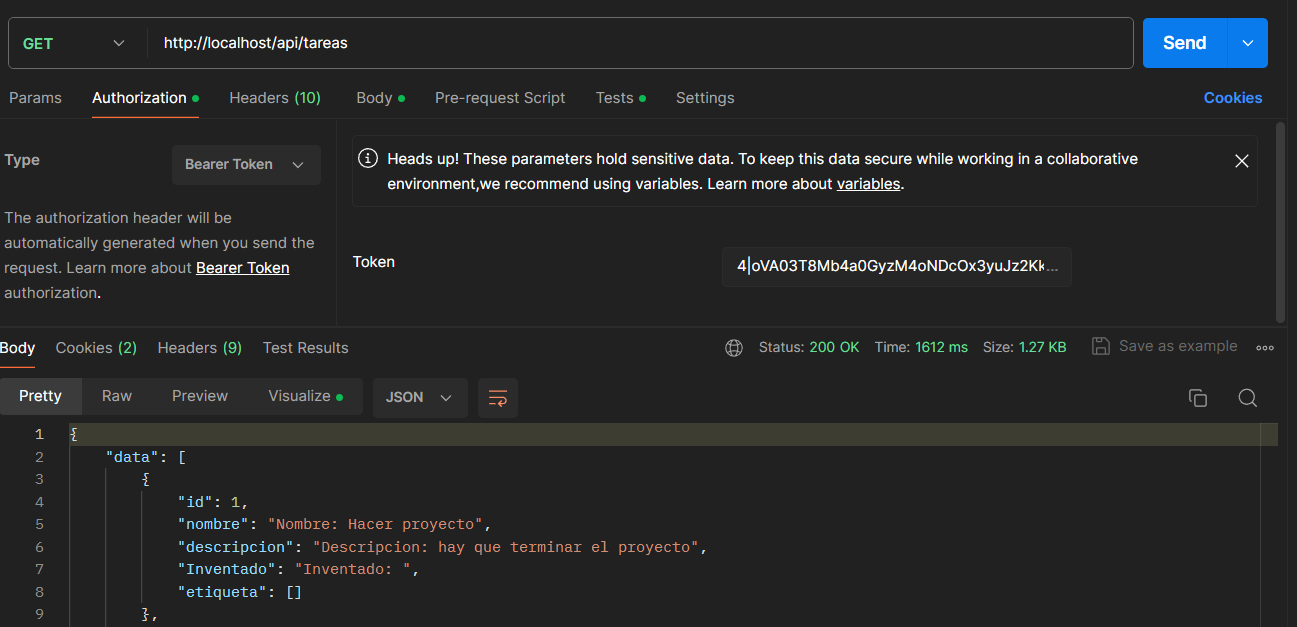
Para probar el login hacemos lo siguiente:  


Y nos debe devolver el token.

Recordar en Headers tener lo siguiente:



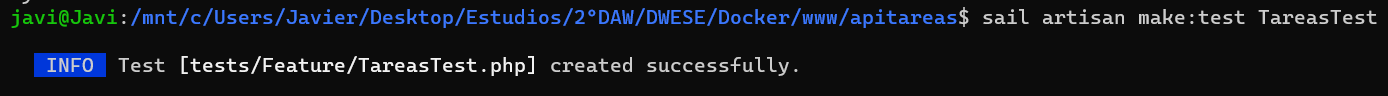
Para cerrar sesion haremos la consulta de la siguiente manera:  


Si implementamos que sea requerido estar logueado para atacar a los endpoints debemos recordar que debemos meter nuestro token en authorization:  


Implementación de test

Vamos a implementar test para comprobar el correcto funcionamiento de nuestro servidor. Para ello vamos a crear nuestro modelo test con sus diferentes funciones. Deberemos ejecutar el sigiuente comando:

‘**sail artisan make:test TareasTest’**

****

Esto nos creara un archivo donde podremos implementar funciones que se ejecutaran cuando hagamos los test en nuestro servidor. En nuestro caso, para ‘TareasTest’ vamos a implementar test para comprobar que las funciones como ‘get’, ‘update’ o ‘delete’ funcionen correctamente.

Antes de nada, saber que si tenemos obligatorio el login y queremos hacer los test, tendremos que añadir las siguientes líneas en nuestras funciones:  
Primero creamos la variable de user.



“**$user = User::factory()->create();**”

Y adaptamos nuestras funciones para que actue como si fuera ese usuario.

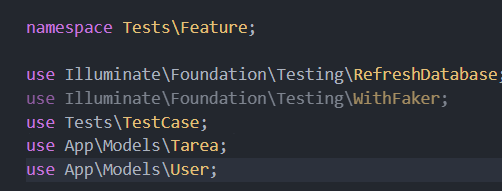


“**$response = $this->actingAs($user)->withSession(['banned' => false])->getJson('api/tareas');**”

E incluir los objetos Tarea y Users:

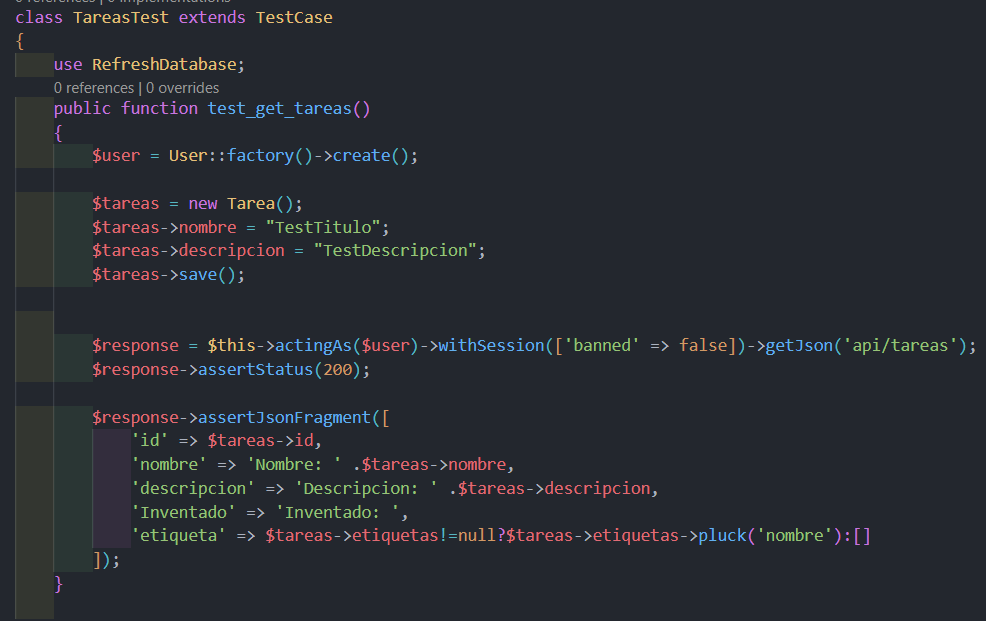
“**use App\Models\Tarea;**”

“**use App\Models\User;**”



Además deberemos usar la función ‘**use RefreshDatabase’** para que pueda hacer las pruebas simulando nuestra base de datos.

Nuestras funciones del test serán las siguientes:

Función test\_get:  


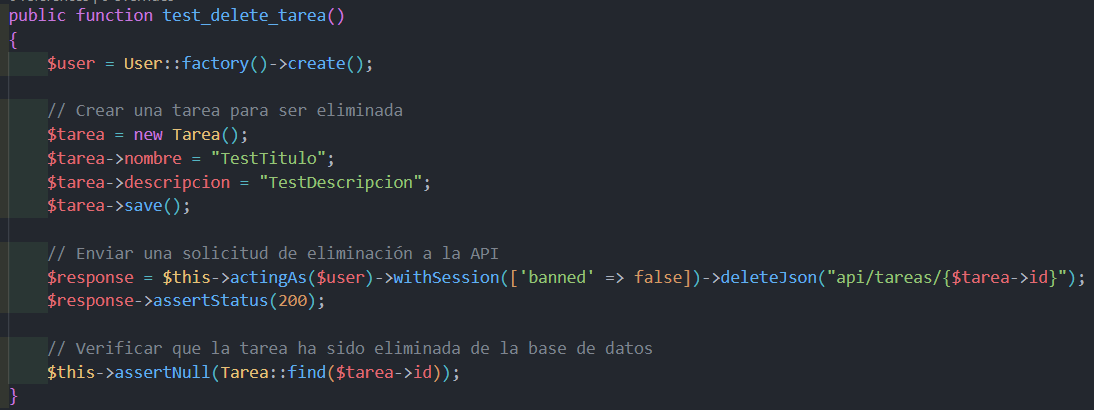
Funcion test\_insert:



Función test\_update:

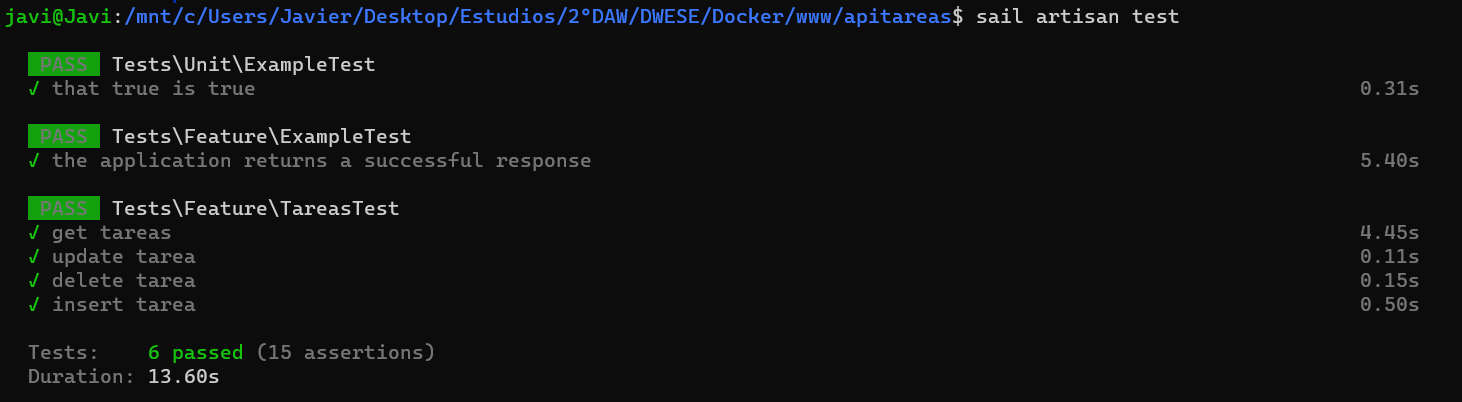


Función test\_delete:



Para realizar las pruebas de test simplemente tendremos que ejecutar el comando

‘**sail artisan test**’:



Y comprobamos que todo ha ido correctamente.

Vamos a crear ahora un test para las etiquetas con la misma estructura de funciones. Para ello repetiremos el proceso que ya hemos realizado con el testareas usando el siguiente comando:

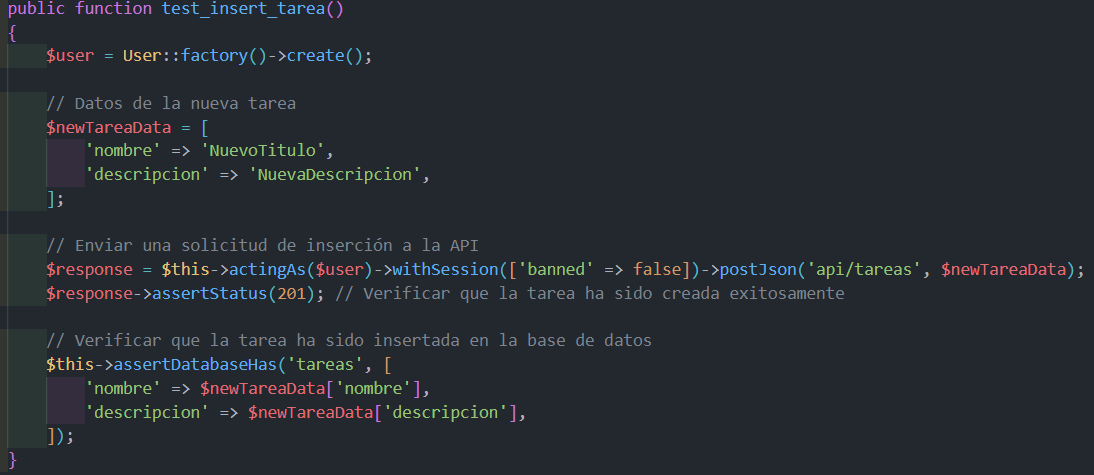
‘**sail artisan make:test EtiquetasTest’**

**‘Recordar usar los import de los modelos y el uso del RefreshDataBase’**

Función get\_etiquetas:



Función insert\_etiquetas:



Función update\_etiquetas:



Función delete\_etiquetas:



Y si volvemos a ejecutar el comando “**sail artisan test**” podemos volver a ejecutar los test, esta vez también para las etiquetas:  
  
