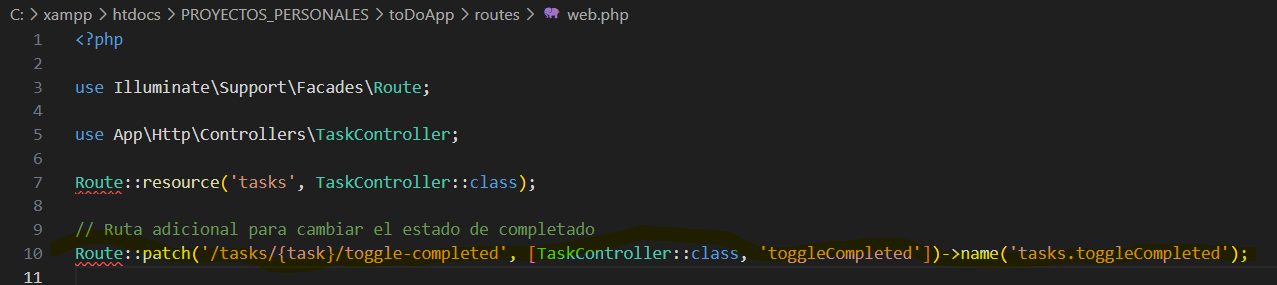
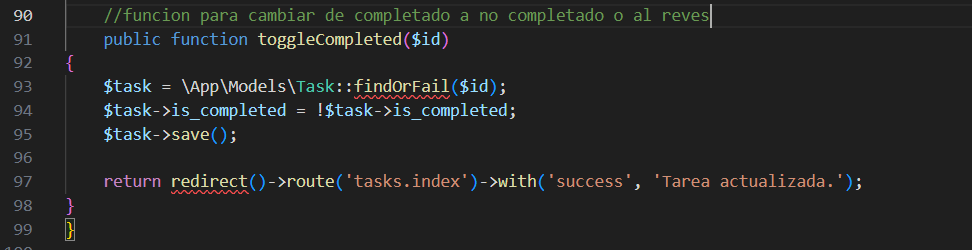
1º Crear un boton para cambiar de completada a no completada en la vista principal, sin tener que pasar por la vista de editar tarea

Necesito una ruta adicional (la ruta que habíamos puesto era automática para los métodos crud, pero para esto voy a tener que crear un nuevo método en el controlador, por lo que necesito una ruta que apunte a ese método, y que cuando pulse el botón que voy a poner en la tabla, me lleve a ese método y lo ejecute con el id de la tarea para la que esoty pulsando el botón)

Por lo tanto, en routes/web.php añadimos la siguiente ruta (pongo lo que había y lo que añadimos)



Como se ve arriba, esta ruta coge un id y vamos a la funcion toggleCompleted del controlador de tasks. Ahora vamos a TaskController, a poner el método toggleCompleted

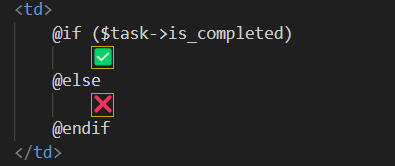


Coge la task del id que le pasamos con un findOrFail

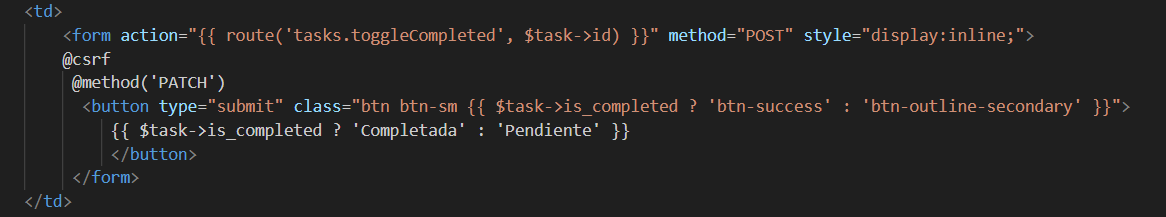
Para esa task, en is completed, pongo lo contrario de lo que habia

**Y con esa tarea llamo a save, que será un método de los que vienen con el modelo, que cambia la task (antes cuando hicimos el modificar tarea, llamábamos al método update, le he preguntado y me ha dicho que update esta preparado para recibir un array con todos los valores nuevos, y en cambio save, es solo para cuando hemos cambiado solamente uno o dos parámetros, en la documentación debe salir la diferencia, pero eso que el modelo tiene varios métodos, preparados para recibir los datos de una forma u otra)**

Ahora, en index.blade, cambiamos el simbolito de completada o no,



por este código

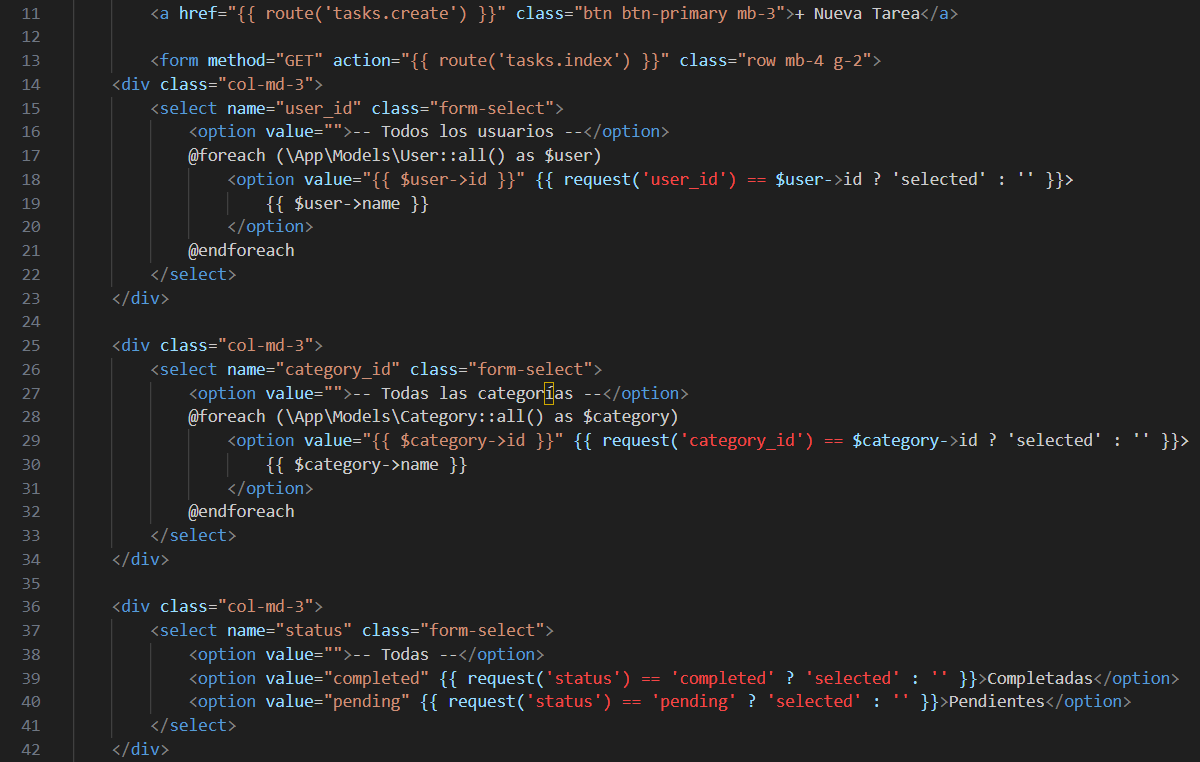


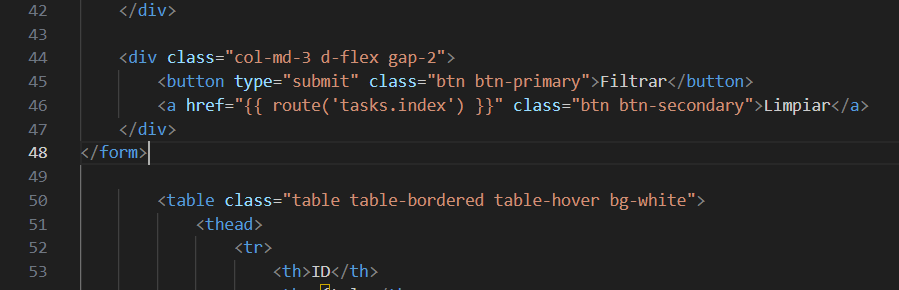
Que por la ruta llega al método del controlador del ToggleCompleted con task id del botón donde estoy pulsando

Y cuando se muestran tareas depende de si esta completada o no muestra una cosa u otra

SIGUIENTE MEJORA, PONER FILTROS PARA ORDENAR POR COMPLETADA, POR USUARIO O POR CATEGORIA

Para esto, ponemos un filtro encima de la tabla para seleccionar por que parámetro ordenamos. Esto lo hacemos poniendo un formulario encima de la tabla (en index.blade.php)

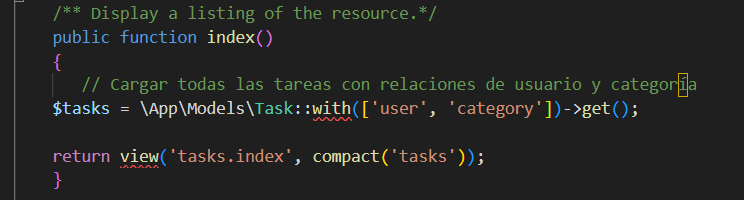
****

****

Este formulario, nos va a redirigir a tasks.index, es decir, a volver a mostrar la tabla, pero ahora vamos a cambiar el método index para que ordene según lo que marcamos en el formulario, y en el formulario tiene 3 selects, uno que carga todos los usuarios, selecciono uno por el nombre y en el id pone el value, otro que carga todas las categorías y lo mismo, selecciono una por nombre y me selecciona en el value su id, y luego otro que selección completado o no completado, por ultimo un botón con submit o con limpiar (que llama a tasks.index, pero sin pasarle nada, mientras que submit llama a tasks.index, pero pasándole un value)

Ahora en taskController, vamos a cambiar el método index, para que filtre,

Antes era asi



Y ahora es asi



Lo mismo pero en vez de hacer tasks = get

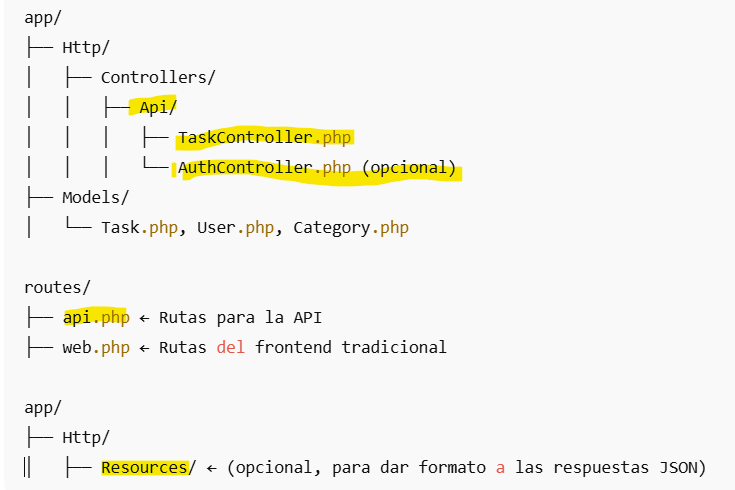
Hacemos una query, con 3 if de posibles cosas que me vengan en el request, para hacer la query de una forma u otra (where)

Y por ultimo aquí si como antes, hago tasks=query->get

Y ya tasks, lo paso a tasks.index.blade, pero filtrado con el where

**AHORA VAMOS A CONECTAR TODO ESTO A UNA API REST, PARA QUE LUEGO UN FRONTEND CON ESTA API, ME PUEDA HACER PETICIONES**

Estructura del proyecto con una api, marco lo que es nuevo que hay que añadir



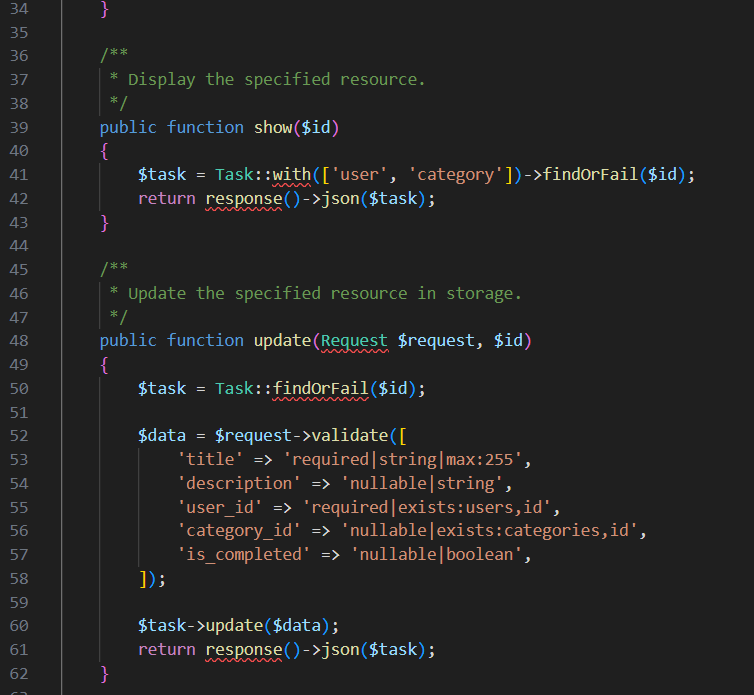
en controllers estaría TaskController, pero ahora vamos a crear en Api otro TaskController. Uno será el controlador de las peticiones clásicas que vengan por el frontend, pulsando botones, y otro será el controlador de las peticiones que vienen por la api.

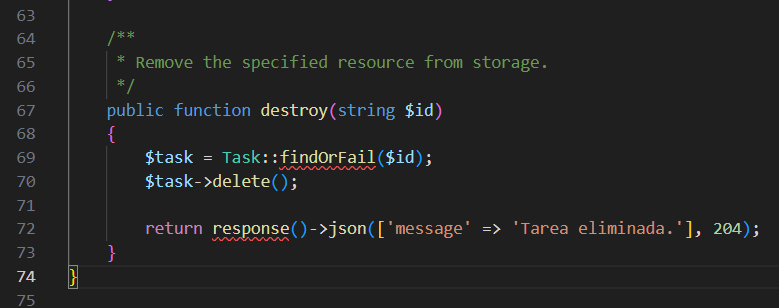


Para ello en consola

**php artisan make:controller Api/TaskController –api**

****

****

****

Son funciones iguales que las del otro controlador pero en vez de dirigirnos con la respuesta a una blade preparada para mostrarla, devuelven un json, y ya la aplicación que haya hecho la consulta, se apañara para filtrar este json.

La funcion index, se ve que como devuelve una collection esta vuelve en json, sin que yo se lo tenga que decir.... por eso el return no pone json, y en las demás si

Luego, hay que modificar **routes/api.php**



Como antes, automáticamente poner las url por las que se hace consulta a los métodos del controlador

Entonces al hacer en el navegador

<http://localhost:8000/api/tasks>

me devuelve esto



Un json con todas las tareas

Ahora, quiero hacer que no todo el mundo con el enlace de la api me pueda acceder, modificar datos, etc, asi que voy a poner un sistema de autenticación con **Laravel Sanctum**

1º Lo instalamos, poniendo en consola

composer require laravel/sanctum

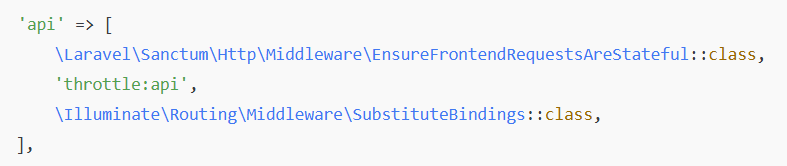
luego hago esto, que copia y crea carpetas en migraciones y en otros sitios

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

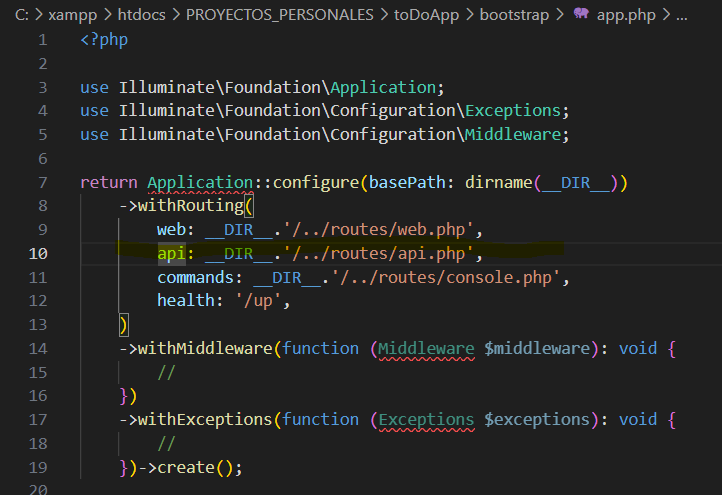
Y luego tengo que ejecutar las migraciones otra vez

php artisan migrate

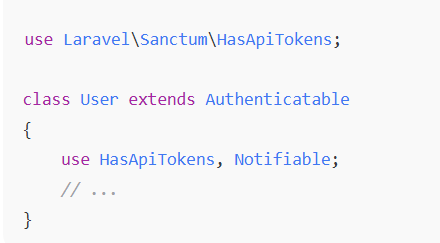
entonces se habran creado carpetas y archivos nuevos. En **app/Http/Kernel.php**, añadimos este array



(en ultimas versiones de laravel nos hemos encontrado que app/Http/Kernel.php ya no existe, y que nos tenemos que ir a bootstrap/app.php, que tiene las rutas de todo el proyecto, y allí añadir estas, para que la api funcione



Y luego en el modelo de user, que es a quien va a afectar esto, porque va a tener que estar el usuario logueado y con un token valido para poder hacer consultas, añadiremos

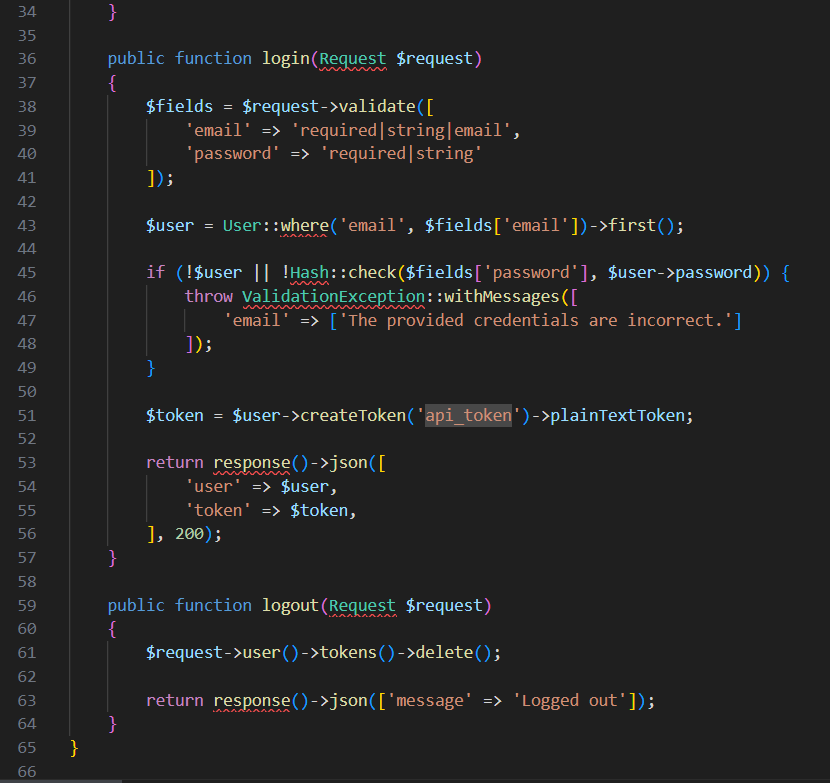


Luego hay que crear un controlador, para los métodos login, logout y registro de usuarios, todo esto via api

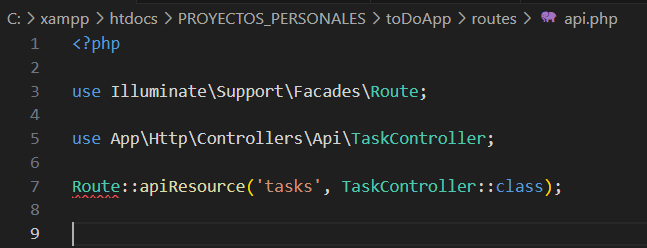
php artisan make:controller Api/AuthController

Dentro de AuthController ponemos

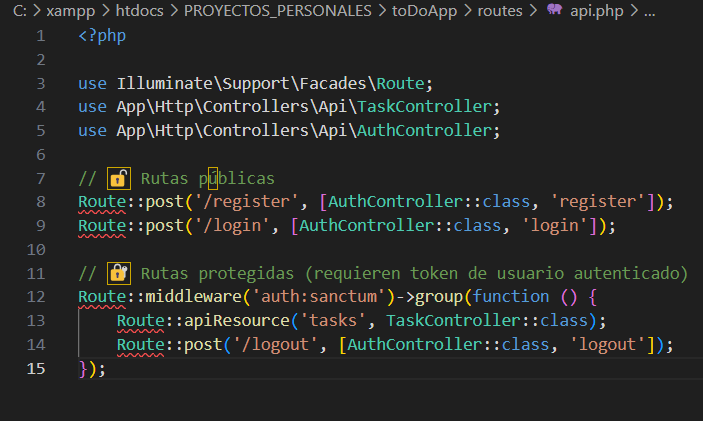




Y ya por ultimo, en **routes/api,php,** hay que añadir esto, para separar las rutas de libre acceso con las que solo se puede acceder si estamos autenticados y tenemos un token valido (antes routes/api.php era



Y después quedara asi

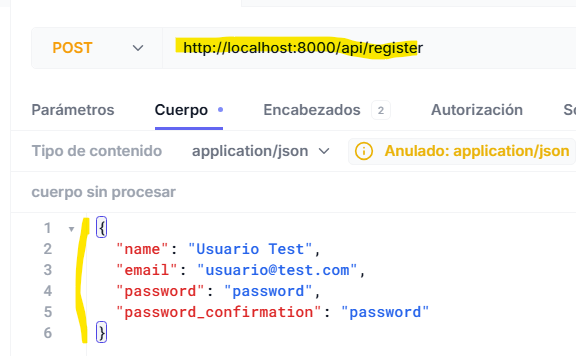


Por entender las rutas, /register, me lleva a el método register de AuthController

/login me lleva al método login de Authcontroller

Luego de las protegidas, esta logout de authcontroller, y esta tasks de taskcontroller, que engloba todos los métodos del taskcontroller que hemos definido antes

**Ahora, para probar que todo esto funciona, hemos ido a hoppostch.io, hemos enviado esta solicitud**



Ver arriba que esta url nos debería de llevar al método register, que lo que hace es registrar al usuario, y devolvernos un token (al método register se podía acceder sin estar autenticado)

(esta por arriba pero era esto)

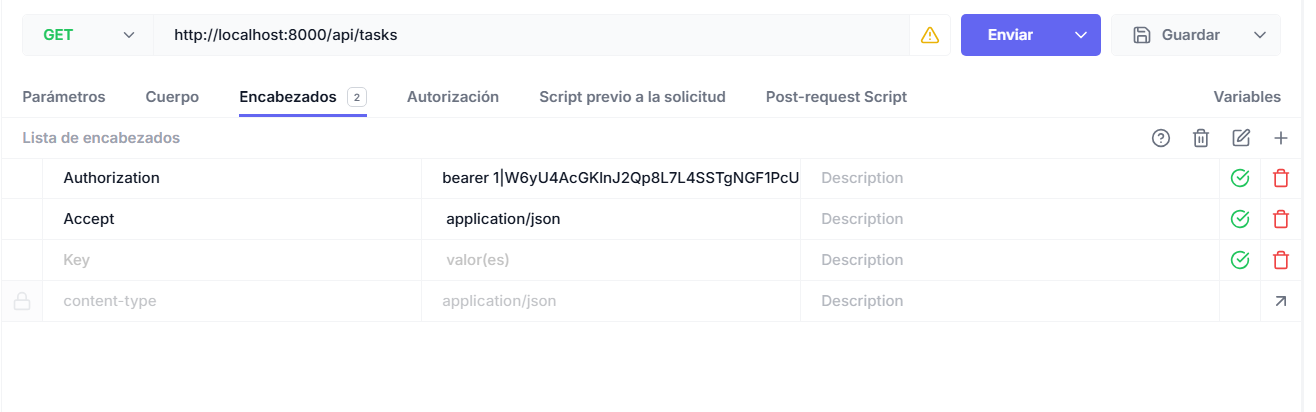


Y la respuesta que nos ha devuelto es esta

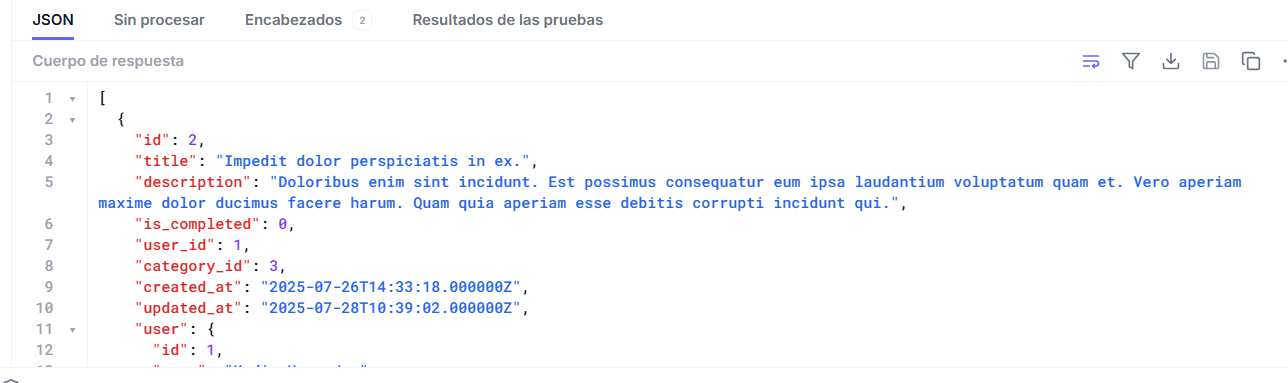


Con este token, ya puedo hacer solicitudes, a los métodos que si que estaban protegidos, porque esta asociado a mi usuario

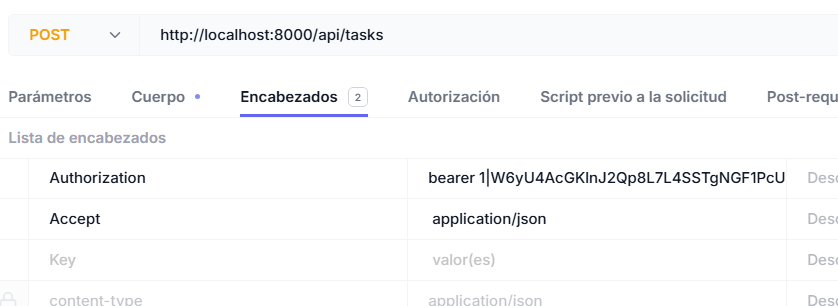
Vamos a probar alguno de estos métodos



Para que me devuelva todas las tareas en un json, la solicitud que llama al método que devolvía todas las tareas en json, y en los encabezados, ponemos bearer token, y me devuelve esto



Otra prueba, para probar el método de crear tareas desde la api

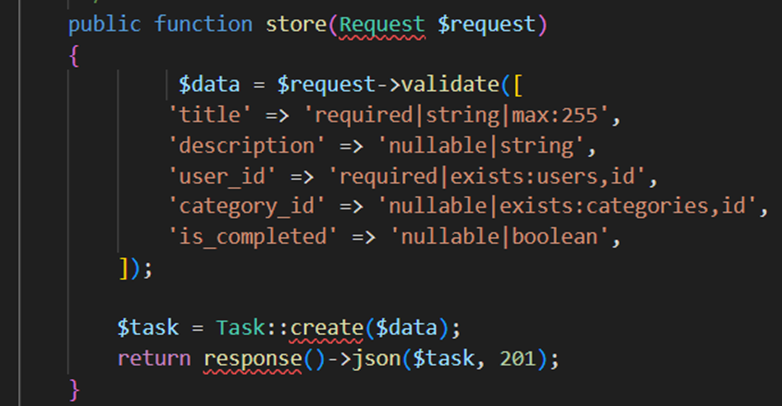


Esta solicitud, con estos encabezados (el token para que me deje acceder al método)

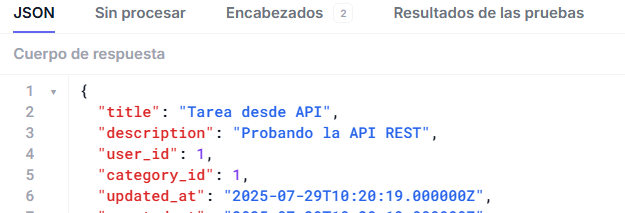
Y en el cuerpo ponemos esto



Y entonces esta info se ha pasado a esta funcion (estaba por arriba)



Que ha recibido el body que le hemos pasado como request, lo ha validado, y ha creado una task con estos datos, y me ha devuelto en json la task que ha creado, que queda asi



Pues ya estaría, comprobado que funciona lanzando peticiones a la api, con el body en json como los datos que se pasan y recibiendo json. Ahora vamos a hacer una aplicación en react o en vue, que trabaje con esta api, para no tener que estar con postman o similar lanzando peticiones a la api y cambiando los body y poniendo el token en el encabezado, sino que lo haga la aplicación, y tengamos un frontend para interactuar, mostrar el json que devuelven estas peticiones, y lanzar nuevas peticiones a la api