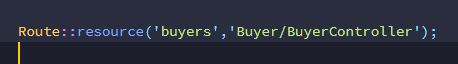
API RES FULL LARAVEL CURSO ESTUDIO

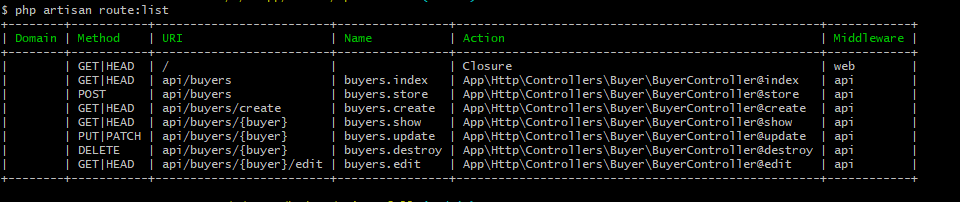
Controladores de recurso



Los mcontroladore sde recursos nos permiter manejar atraves de una solo ruta varios metodod http



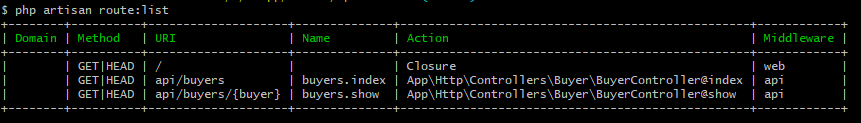
Esta sola ruta tendrá 5 metodos http post get delete update



Como trabajremos con api no utilizaremos el método créate ni edit ya que son para retornar vistas HTML Podemos decirle a nuestra ruta de recursos con que método queremos que trabaje nuestro controlador de la siguiente manera



De esta manera nuestra recurso solo trabajar con estos métodos



También podemos decirle a nuestra rutasa de recurso que trabaje con todo los métodos excepto estos de esta manera



En el modelo producto utilizamos 2 constantes para saber el estado e un producto para no trabajar con el campo del modelo directamente

Esto lo ahremos mediate un función descrita en el modelo el cual validaremos el status del recurso en ese momento diciendo si el modelo utilizado en ese momento en la columna status ess igual a una constante producto disponible retorna true si no mandaría un false lo cual diría que no esta disponible



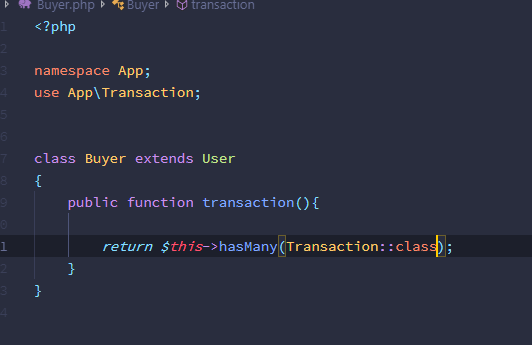
Modelo user para manejar estados utilizamos el mismo proceso de cosntantes y funciones publicas para validar i el usuario loguead en ese momento es esta o no esta verificado



Los atributos que estén dentro de la variable portegida hidden no serán visibles para asignar en masas



RELACIONES ENTRE MODELOS Y TIPO DE RELACIONES uno A muchos( hasMany)



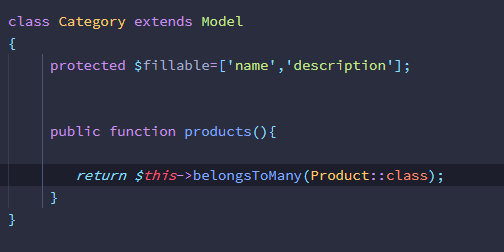
Modelo buyer esta relación hace referencia al relación que hay entre el modelo buyer que extiende de user con el modelo transaction importamos el modelo que vamos a relacionar

En este caso

Se utiliza el hasMany (uno a muchos) modo que el la tabla pasada como parámetro sea que quien estienda del modelo en este caso Transaction

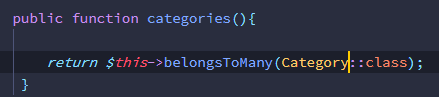
RELACIONES DE MUCHOS A MUCHOS TABLA PIVOTE (CATEGORY AND PRODUCT)

belongsToMany



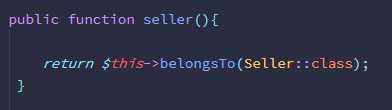
RELACION PRODUCT A CATEGORY SE UTILIZA EL MISMO belongsToMany

//podemos pasar como referencia columnas o tablas pivote en caso tal no trabajemos con la conbenccion

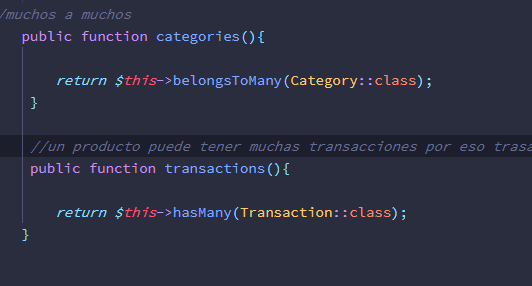


Es importante resaltar que el modelo que pertenece al otro es quien lleva la clase foránea

EJEMPLO producto pertenece a un vendedor entonces el modelo producto es quien lleva la clave foránea de seller (muchos productos pertenecen a un vendedor )



Relaciones de Product



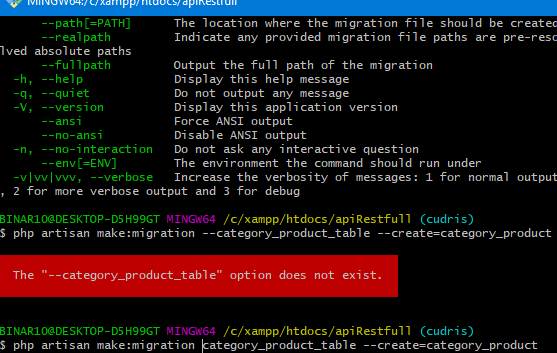
Migraciones de producto



MIGRACION DE L ATABLA PIVOTE PARA LA RELACION QUE HAY ENTRE PRODUCTOS A CATEGORIAS

Para crear una migración utilizamos el make:migration nombremigracion –created=como queremos como se llame la tabla

Si queremos editar una migración lo que haremos es crear otra migración pero enve z de créate le pasaremos table como parámetro la tabla la cual quremos editar

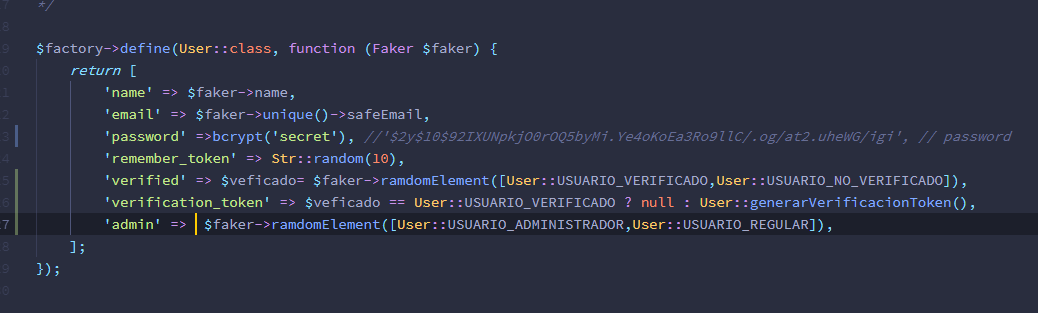


Para esta migración y esta tabla pivote en especifico no habraun id auto incrementable

Ya que por defect accedermos a las referencias de las tablas implicadas



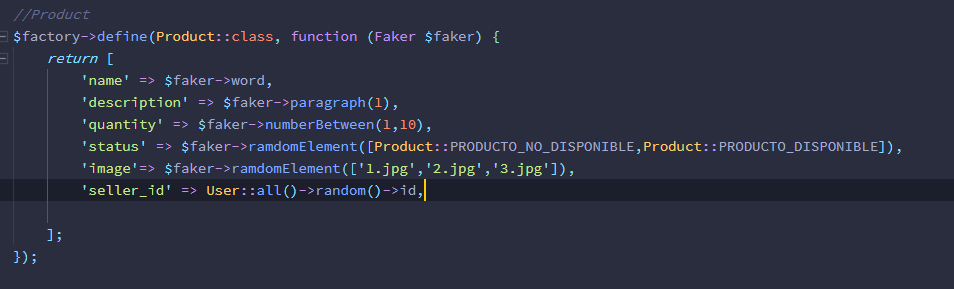
FACTORYS Modelo users



Utilizamos una lógica para los campo verified usando la función ramdomElement ([]) que recibe como paametros un array de valores a asignar al campo

Luego de esto en verification token utilizamos una validación ternaria para generar tokens de verificación en caso tal el campo asignado para el usuario en verified sea diferente de de verificado

Factory porduct



Transactions

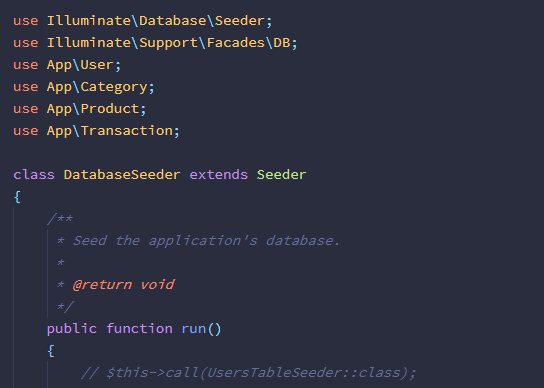
Se crea una lógica para poder asignar un comprador y el prodcut asignado a la transaction no sea el mismo dueño dicho producto

En el modelo Seller obtenemos todos los productos asociados a un vendedor de manera ramdom

Y para el comprador obtenemos usuarios diferentes al del vendedor en ese momento de manera ramdom

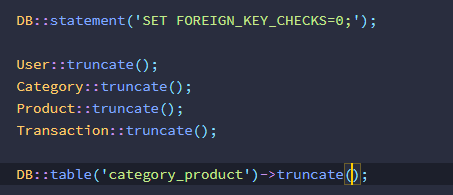


DatabaseSeeder para crear los factorys de los modelos



Aquí instanciamos directamente los factorys que hemos creado en user factory de todo los modelos en el factory de Product crearemos para cada producto una función que le permita asignar entre 1 y 5 categorias por producto ademas como categorías nos devuelve una colección podemos acceder a sus propiedades atarves del método pluck

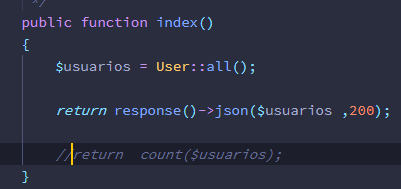
Y con el metodo attach le asignamos la categoría en la relación de la instancia del producto



Ademas como la tabla category\_porudct no existe lo que aremos es utilizar es el helpér DB para poder truncarla

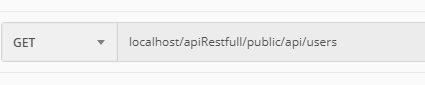
CONTROLADORES DE RECURSOS USERS Metodos(get,show{id},post,put,delete))

PODEMOS UTILIZAR EL ORM DE LARAVEL PARA TRABAJAR con nuestros modelos



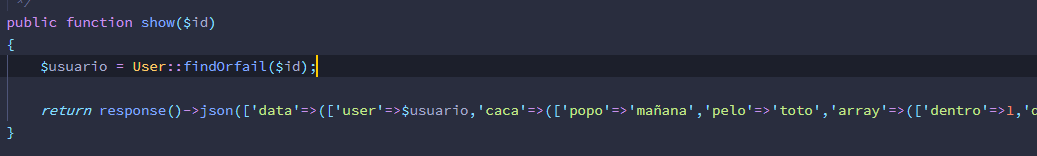
Podemos probar con postman de 2 manera : 1) activando el php artisan serve de nuestro proyecto

2) o mandando directamente la ruta de nuestro proyecto sin activar el php artisan serve



Si l necesidad de tirar el localhost:8000

En ves de utilizar el método find utilizaremos el findOrfail nos permite verificar sin validar que el request de la consulta se true y en caos tal no lo sea nos devuelve un 400



METODO CREATE DE MANERA MASIVA PARA LA FUNCION STORE DE USERCONTROLLER

POR DEFEcto laravel utiliza el método post para crear controladores de recursos

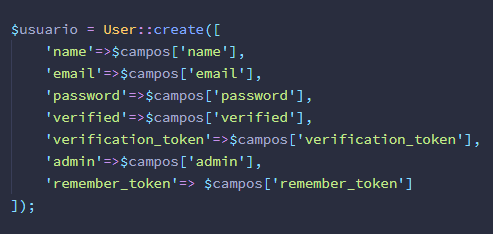
Ademas odemos definir las reglas de validación para los capos al crear un nuevo usuario de varias maneras como lo vemos en la imagen

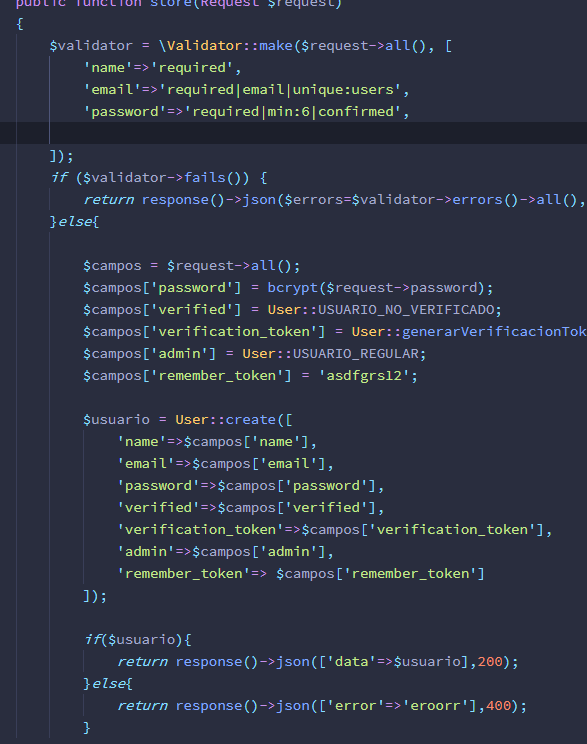
Como estamos creando un usuario de manera masiva con el método créate que recibe un array

Debemos validar dichas columnas para que u usuario no puede darse admin o estar verificado sin hacer el proceso de verificación de manera que validamos con las constantes en user para poder asignarles valores a las columnas que por defecto no pueden ser creadas de manera masiva

ME FUNCIONO LO DE ABAJO

FORMA CORECTA DE UTILIZAR EL METODO CREATE EN MASA





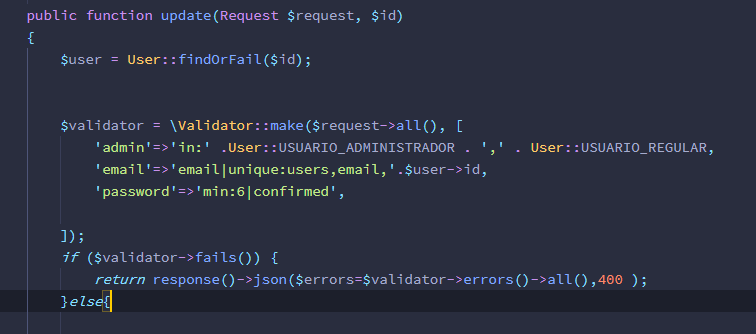
Función update UserController metodo(Put)

En esta función antes que nada obtenemos el user a modificar utilizando el meodo findOrfail

Luego en el validador colocamos campos no requeridos pero con reglas de validaciones muy estrictas por ejemplo en el campo admin utilizamos el vaidador in: para pasarle ciertos parámetros

A los cuales el campo tiene que estar igualado, otro caso particular es el del email hacemos una validación

Para que el campo email sea único pero exceptuando el del usuario al que queremos cambiarle la información y nos valida perfectamente la request en el email no toma en cuenta el email del id del usuario que le pasamos



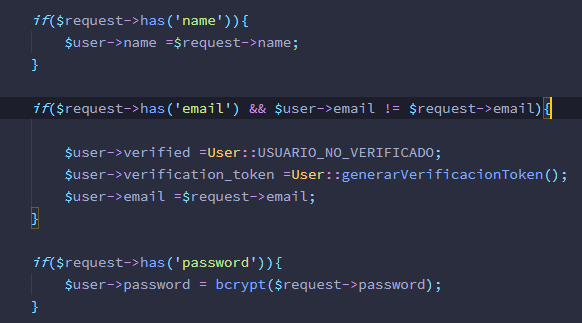
Función update UserController metodo(Put)

En esta misma función ya al pasar las regla de validación

Utilizamos el método has en el request para validar que el parámetro pasado contenga algo y si tiene algo le asignamos al usuario obtenido en ese momento el request con el campo al que queremos modificar

En el caso del email validamos que email obtenido sea diferente del email del usuario en ese momento en caso tal sea diferente colocamos el campo de verifie en false le generamos una verificación de token y le asignamos el nuevo email al usuario

Estamos validando campos en caso tal vengan en el request y si existe se le asignan al campo especifico al usuario al cual queremos modificar



Función update UserController metodo(Put)

Tenemos en cuenta el campo is admin validando ya que es un campo delicado validamos que el usuario este verificado

Para terminar utilizamos el método isDirty() sobre el modelo en ese momento para saber si hubo algún cambio en nuestro usuario esto lo haremos al final de todo el controlador pero lo que haremos es negar el método ¡$user->isDirty() esto loq e hará es si no hemos cambiando ningún dato botara una excepción y pues si no entra aquí

Por ultimo lo que haremos es guardar el usuario con los nuevo datos

