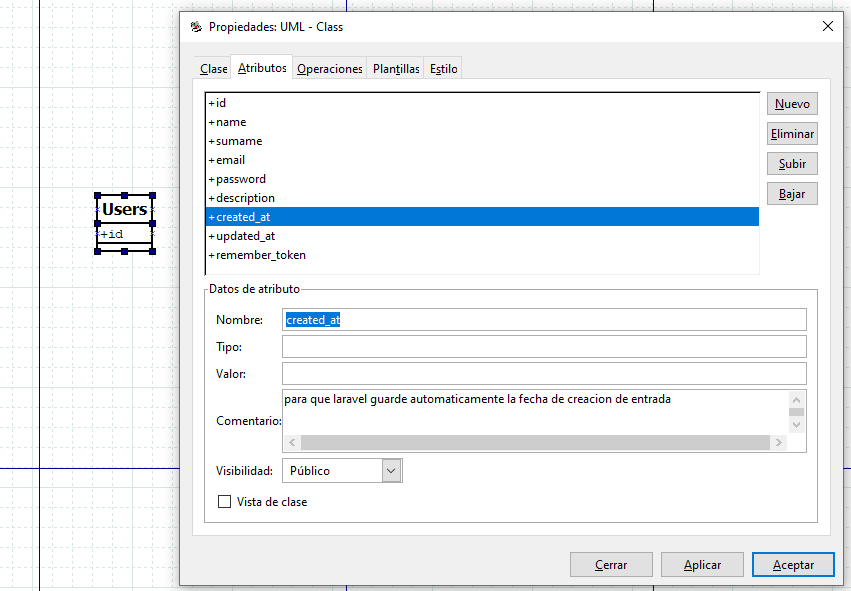
**CAPITULO 2**

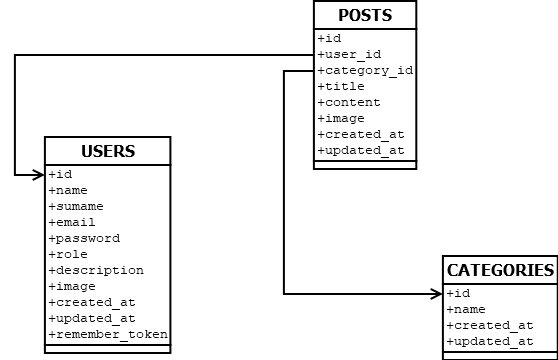
**DISEÑANDO LA BASE DE DATOS**

Lo primero que tendremos que hacer antes de nada es diseñar la base de datos. Que relaciones va a tener que elementos va a guardar…

Podemos utilizar la ayuda del programa que nos permitirá hacer diagramas UML, de flujo, de relaciones… Nosotros utilizaremos día program, que puede ser descargado en: <http://dia-installer.de/index.html.es> .



Aquí hemos creado la clase Users con todos los atributos que podréis ver en la imagen. Los últimos tres se utilizan en proyecto de laravel, donde el primero guarda automáticamente la fecha de entrada del registro automáticamente, el segundo actualiza el registro automáticamente y el tercero, pero no menos importante, guarda un token por cada inicio de sesión para que laravel pueda recordar los usuarios.



Una vez tenemos el esquema, vamos a pasar este diseño a código SQL y crearemos la base de datos.



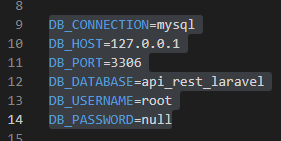
Una vez desarrollado el código abrimos phpmyadmin y le damos a la pestañita SQL pegamos este código que hemos desarrollado y si no se ha generado ningún error nos creara las tres tablas que queríamos desarrollar.

Como vemos en el código cada una tiene los atributos que pusimos en el diagrama UML y además restricciones 🡪 CONTRAINT, las dos primera tienen una restricción de clave primaria para su atributo id, y la ultima tiene además 2 claves foráneas que apuntan al id de las otras dos tablas relacionando estas dos con la tabla post.

A continuación, rellenaremos la base de datos con datos de pruebas, ponemos insertar en cada una de las tablas que creamos y rellenamos los campos que hicimos.

Vamos a pasar a conectar esta base de datos que hemos creado ahora ya con datos que hemos introducido manualmente, con nuestro proyecto de laravel, para ello haremos lo siguiente:

1. Vamos al archivo .env, y relleneramos los siguientes campos dejandolos de la siguiente manera. Obviamente esto cambia dependiendo de cómo hayamos desarrollado la base de datos, pero para este caso en concreto:

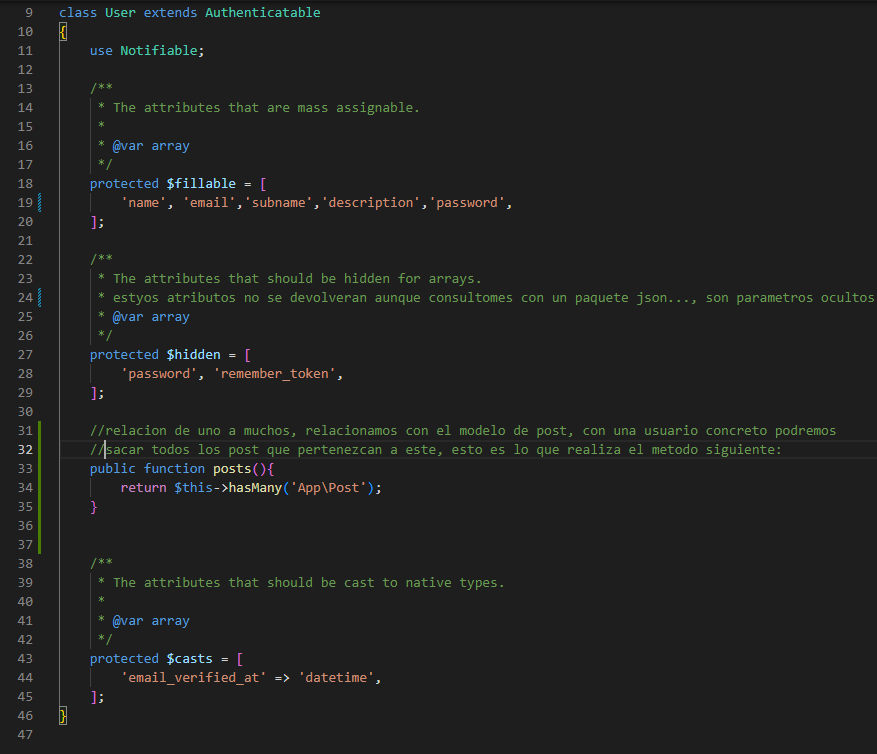


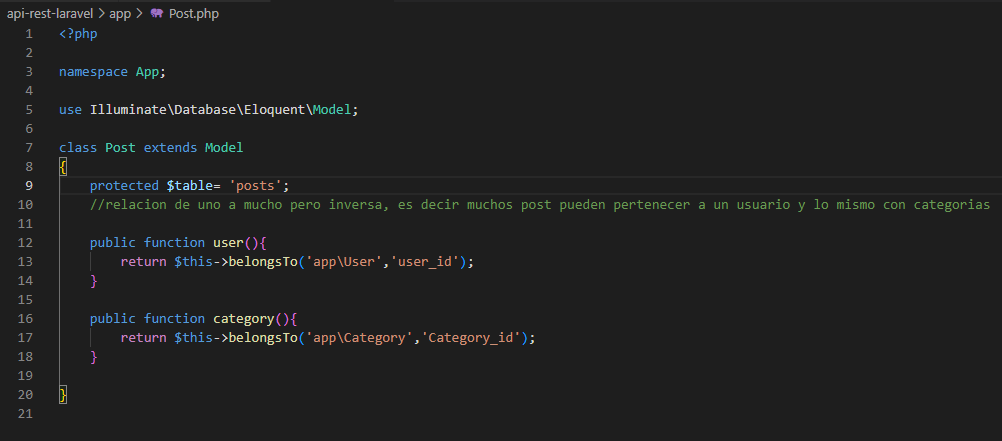
1. Vamos a crear el modelo de usuario el de post y el de categorías. Si escalamos y ampliamos la aplicación tendríamos que crear mas modelos. Nos dirigimos a la carpeta app, y veremos que ya hay uno de nuestros modelos creados, el modelo User pues Laravel ya viene por defecto con esta clase creada por el hecho de que tiene la función de autentificar incorporada, además de que viene con unos atributos ya predefinidos.
2. Crearemos los demás modelos desde el cmd de Windows, tendremos que ir a la carpeta donde tenemos creado el proyecto, en nuestro caso api\_rest\_laravel y desde ahí ejecutaremos los comandos: php artisan make:model Category. Haremos lo mismo con el modelo post.

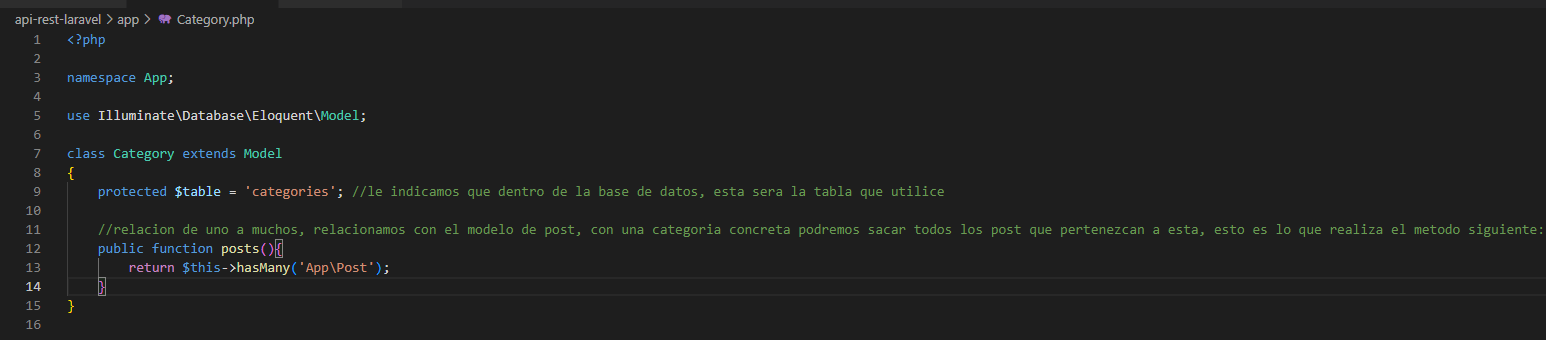
Podemos observar diferencias, el modelo User extiende de authentificable mientras que los otros modelos que hemos creado incorporan la librería model. Esto es porque

authentificable ya se considera como un modelo en sí, y la otras dos clases no tendrían mucho sentido que extendieran esta clase.

1. Se usa Eloquent como librería de model pues nos permite trabajar de una manera correcta, separando capas y trayendo de la base de datos en forma de objetos los datos que necesitemos.
2. Los modelos quedarían:







Ahora para probar que todo funciona correctamente utilizaremos el controlador de pruebas que creamos anteriormente:

Para utilizar estos modelos obviamente dentro del controlador tendremos que importar de esta manera, **Use App\Post** en el caso del modelo post. (aunque la carpeta sea app, se debe escribir App en mayúsculas pues es como declara el namespace laravel).

Crearemos una función para gastar y además una nueva ruta para comprobar en el navegador que todo funciona correctamente, var\_dump muestra el contenido de la variable con todos los datos.

public function testOrm(){

  $posts = Post::all();//esto es como hacer un select \* de la tabla posts

  var\_dump($posts);

  die(); //esta funcion es un alias de la funcion exit, sale del script y ---------deja de ejecutar

}

El resultado de esto es un objeto de tipo colección de la clase Eloquent, no se ve correctamente la información que se esta mostrando, pero podemos hacer un for each que recorra todos los posts y crea una variable llamada post por cada uno para ir mostrándolos uno a uno. El código quedaría:

public function testOrm(){

  $posts = Post::all();//esto es como hacer un select \* de la tabla posts

  var\_dump($posts);

foreach($posts as $post){

            echo "<h1>".$post->title."</h1>";

echo "<p>".$post->content."</p>";

            echo '<hr>'; //linea horizontal que separa cada uno

        }

  die(); //esta funcion es un alias de la funcion exit, sale del script y ---------deja de ejecutar

}

Si ahora quisiéramos mostrar datos de las tablas relacionados con post como lo son la de users o la de categories lo haremos así:

 echo "<span style='color:gray;'>{$post->user->name} - {$post->category->name}</span>";

como vemos la sintaxis es extraña pues estamos llamando a un método como si fuera una atributo, esto lo permite la clase eloquent o ORM, lo que hacemos aquí es del objeto post llamar a su función user y del array que nos devuelve cojemos el atributo name. Lo mismo con el método category.

Podemos hacer eso mismo con el objeto category:

 $categories = Category::all();

     foreach($categories as $category){

        echo "<h1>{$category->name}</h1>";

        foreach($category->posts as $post){ //llamamos a la función post --------del modelo category, eloquent en orm lo convierte en una propied-------- y por eso podemos usar esta sintaxis

              echo "<h3>".$post->title."</h3>";

              echo "<span style='color:gray;'>{$post->user->name} - **------------------------** {$post->category->name}</span>";

              echo "<p>".$post->content."</p>";

        }

   echo '<hr>';

}