PROYECTO LARAVEL

Autor: Guillermo Jáuregui Lahoz
Curso: 2023/2024

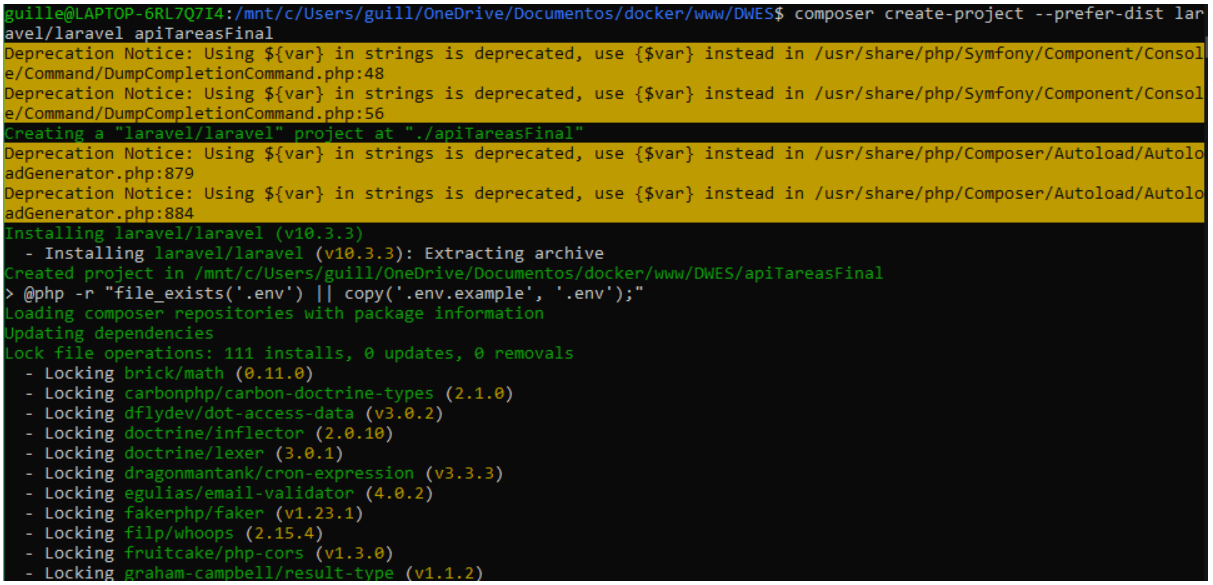
Índice

1. INICIO DEL PROYECTO

Para empezar a trabajar en Laravel necesitamos crear el proyecto en primera instancia. Para eso mediante cd nos introduciremos en la carpeta en la que queremos instalar el proyecto.

Una vez que estemos dentro de la carpeta en la que instalaremos el proyecto necesitaremos instalar composer, mediante el comando “composer install”, seguido de este comando necesitaremos poner el que nos instalará el proyecto el cual es “composer create-project --prefer-dist laravel/laravel apiTareasFinal” en este caso el nombre que se le ha dado es apiTareas, pero el nombre puede ser el que deseemos.

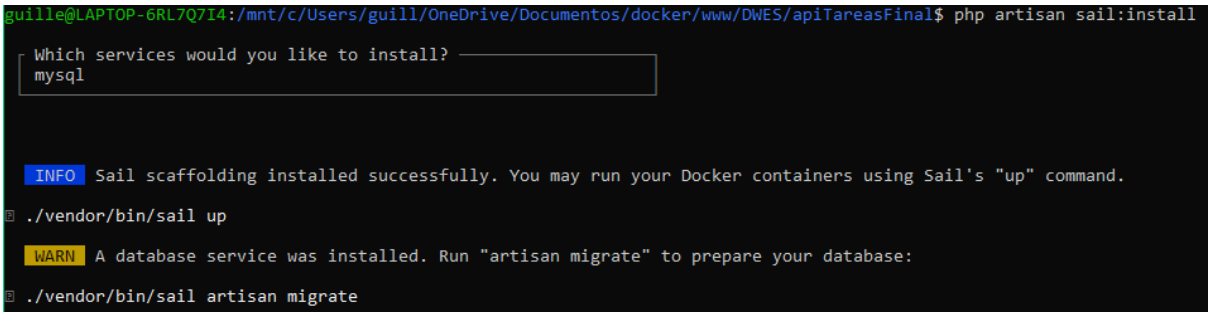
Al introducir este comando nos saldrá lo siguiente:



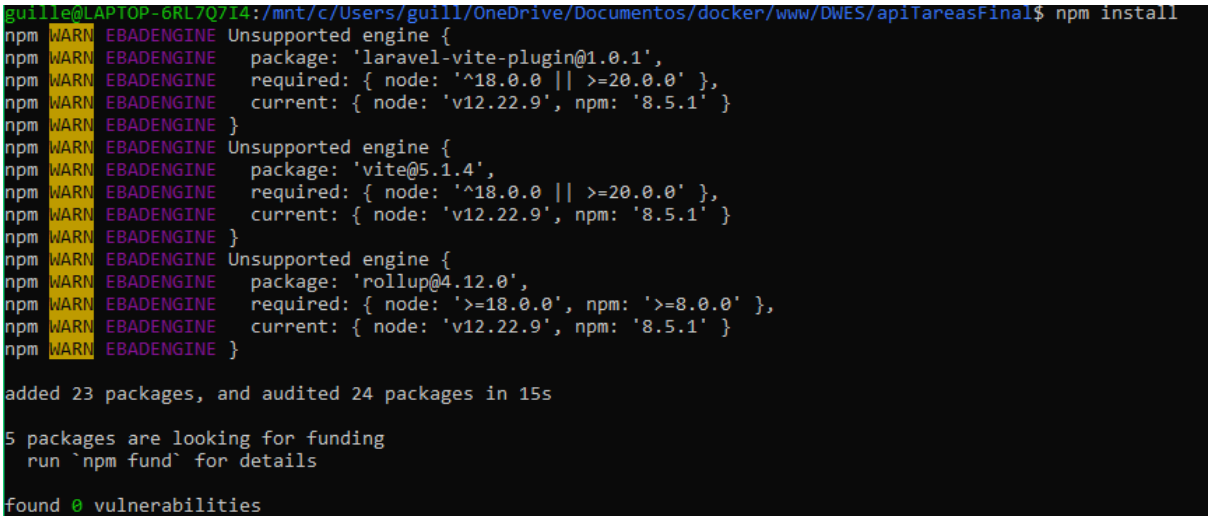
Cuando el proyecto esté instalado tendremos que introducirnos en la carpeta que se nos ha creado, la cual lleva el nombre del proyecto. En este caso será “cd apiTareasFinal”.

Una vez estemos dentro de esa carpeta deberemos poner el comando “php artisan sail:install”, el cual instala y configura Docker dentro del proyecto Laravel.

Cuando lo introducimos saldrá lo siguiente:



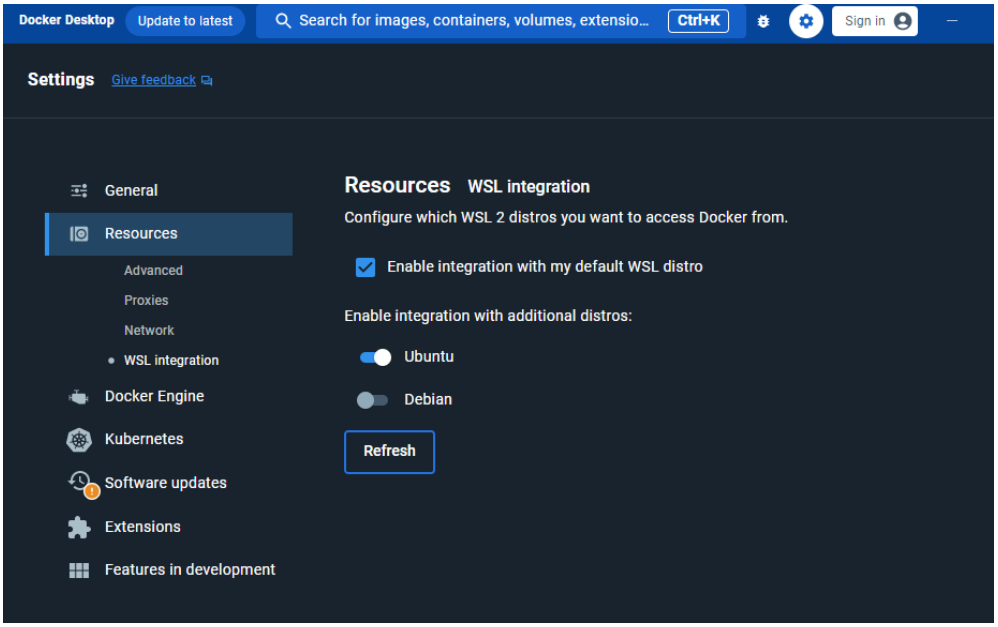
Una vez esté instalado correctamente deberemos poner “npm install” el cual instala todas las dependencias que le hacen falta al proyecto. Al ponerlo saldrá esto:



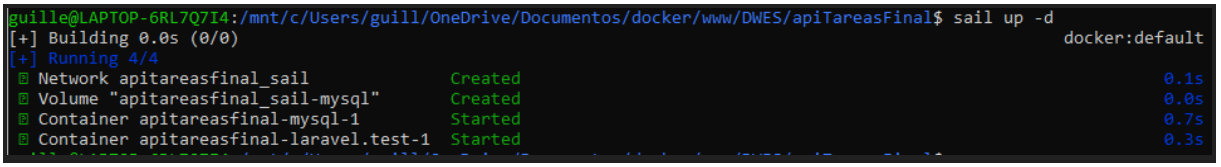
Después de instalar npm deberemos poner “alias sail=./vendor/bin/sail” lo cual hace que es crear un atajo con el nombre “sail” el cual hace referencia al ejecutable ./vendor/bin/sail, para así no tener que poner la ruta del ejecutable cada vez que queremos realizar una acción.

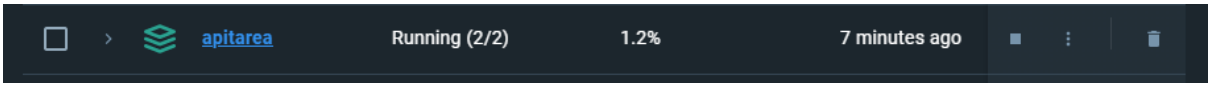
Una vez tengamos el alias definido deberemos poner el comando “sail up -d” el cual es un comando específico de Laravel que se utiliza para levantar los contenedores de Docker.

Pero antes de poner el comando tendremos que tener instalado en el ordenador el DockerDesktop el cual nos permite ver los contenedores de Docker que tenemos creados. Cuando lo tengamos instalado deberemos irnos a los ajustes y la parte de Resources y dentro de esta parte tendremos que acceder donde pone “WSL integration” y allí activar la pestaña de ubuntu ya que es desde donde lo estamos realizando.



Cuando nos aseguremos que esto está activado podremos poner el comando “sail up -d” lo que hará que en la terminal aparezca lo siguiente indicando que la creación del contenedor está realizada, además de poder verlo dentro del DockerDesktop.





2. CREACIÓN DE LAS MIGRACIONES

Ahora que tenemos todo instalado correctamente, empezaremos a crear las tablas del proyecto.

Para empezar pondremos el comando “sail artisan make:migration create_tareas_table”, este comando realiza la migración para crear la tabla tareas en este caso ya que es el nombre que le hemos proporcionado.
Si la tabla se ha creado correctamente tendría que aparecer dentro de la carpeta /database/migrations con la fecha del día en que se esté realizando.

En este archivo que nos de le tendremos que decir los campos que queremos que tenga la tabla, en este caso le daremos id, título y descripción.

```php
public function up(): void
{
    Schema::create('tareas', function (Blueprint $table) {
        $table->id();
        $table->string("titulo", 40);
        $table->string("descripcion", 200)->nullable;
        $table->timestamps();
    });
}
```

Una vez esté creada la tabla tendremos que poner “sail artisan migrate” lo que hará que la tabla realice la migración al contenedor.

Cuando hayamos realizado este tendremos que realizar lo mismo pero con la tabla etiquetas, lo cual se realizará de la misma manera pero con la única variación de cambiar los nombres por etiquetas:

```
guille@LAPTOP-6RL7Q7I4:/mnt/c/Users/guill/OneDrive/Documentos/docker/www/DWES/apiTareasFinal$ sail artisan make:migratio
n create_etiquetas_table

  INFO  Migration [database/migrations/2024_03_03_211403_create_etiquetas_table.php] created successfully.
```

```php
    * Run the migrations.
    */
    public function up(): void
    {
        Schema::create('etiquetas', function (Blueprint $table) {
            $table->id();
            $table->string('nombre',20);
            $table->timestamps();
        });
    }

    /**
    * Reverse the migrations.
    */
    public function down(): void
    {
        Schema::dropIfExists('etiquetas');
    }
};
```

Ahora para finalizar las migraciones del proyecto tendremos que crear la tabla de “tareas_etiquetas”

```
guille@LAPTOP-6RL7Q7I4:/mnt/c/Users/guill/OneDrive/Documentos/docker/www/DWES/apiTareasFinal$ sail artisan make:migratio
n create_tareas_etiquetas_table

  INFO  Migration [database/migrations/2024_03_03_211851_create_tareas_etiquetas_table.php] created successfully.
```

```php
    public function up(): void
    {
        Schema::create('tareas_etiquetas', function (Blueprint $table) {
            $table->id();
            $table->foreignId('tareas_id')->constrained()->onDelete('cascade');
            $table->foreignId('etiquetas_id')->constrained()->onDelete('cascade');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('tareas_etiquetas');
    }
};
```

```
guille@LAPTOP-6RL7Q7I4:/mnt/c/Users/guill/OneDrive/Documentos/docker/www/DWES/apiTareasFinal$ sail artisan migrate

  INFO  Running migrations.

  2024_03_03_211403_create_etiquetas_table ..................................................... 744ms DONE
  2024_03_03_211851_create_tareas_etiquetas_table ............................................. 2,551ms DONE
```

3. CREACIÓN DE LOS MODELOS

Una vez que tengamos creados todas las migraciones del proyecto tendremos que crear los modelos, los cuales se utilizan para insertar información en las tablas.

Empezaremos por el modelo de Tarea el cual se crea ejecutando el comando “sail artisan make:model Tarea -cr”

```
guille@LAPTOP-6RL7Q7I4:/mnt/c/Users/guill/OneDrive/Documentos/docker/www/DWES/apiTareasFinal$ sail artisan make:model Ta
rea -cr

  INFO  Model [app/Models/Tarea.php] created successfully.

  INFO  Controller [app/Http/Controllers/TareaController.php] created successfully.
```

Después de crear el modelo de Tarea crearemos el modelo de Etiqueta, lo cual se realizará de la misma manera que antes, poniendo el comando “sail artisan make:model Etiqueta -cr”

```
guille@LAPTOP-6RL7Q7I4:/mnt/c/Users/guill/OneDrive/Documentos/docker/www/DWES/apiTareasFinal$ sail artisan make:model Et
iqueta -cr

  INFO  Model [app/Models/Etiqueta.php] created successfully.

  INFO  Controller [app/Http/Controllers/EtiquetaController.php] created successfully.
```

Cuando hayamos creado los dos modelos nos habran aparecido los archivos de Tarea.php y Etiqueta.php dentro de la direccion “app/models” dentro de estos archivos tendremos que poner lo siguiente:

Tarea.php:

```php
class Tarea extends Model
{
    use HasFactory;

    0 references
    protected $fillable = ["nombre", "descripcion"];

    0 references
    protected $hidden = ["created_at", "updated_at"];

    0 references | 0 overrides
    public function etiquetas(): BelongsToMany{
        return $this->belongsToMany(Etiqueta::class, 'tareas_etiquetas', "tareas_id", "etiquetas_id");
    }
}
```

Etiqueta.php:

```php
class Etiqueta extends Model
{
    use HasFactory;

    0 references
    protected $fillable = ["nombre"];
    0 references
    protected $hidden = ["created_at", "updated_at"];

    0 references | 0 overrides
    public function tareas(): BelongsToMany{
        return $this->belongsToMany(Tarea::class, "tareas_etiqueta", "etiqueta_id", "tarea_id"  );
    }
}
```

4. CREACIÓN DE LOS SEEDERS

Ahora crearemos los seeders los cuales realizan la inserción de los datos en la tabla.

Tendremos que crear tres seeder debido a que hemos creado tres tablas anteriormente.

El primer seeder que crearemos será el de Tarea, el cual se creará poniendo el comando “sail artisan make seeder TareasSeeder”

Una vez que el seeder esté creado nos encontraremos en la carpeta de “database/seeders” el seeder que acabamos de crear. Dentro de este seeder deberemos poner los datos que le queramos insertar a la tabla:

```php
public function run(): void
{
    DB::table('tareas')->insert([
        'titulo' => 'Hacer de comer',
        'descripcion' => 'Hacer macarrones con tomate'
    ]);

    DB::table('tareas')->insert([
        'titulo' => 'Recoger',
        'descripcion' => 'Recoger la colada'
    ]);

    DB::table('tareas')->insert([
        'titulo' => 'Entrenar',
        'descripcion' => 'Ir al gimnasio'
    ]);
}
```

Ahora crearemos el seeder de etiquetas que se realizará de la misma manera de las tareas pero cambiando los nombres mediante el comando “sail artisan make:seeder EtiquetasSeeder”

```php
public function run(): void
{
    DB::table('etiquetas')->insert([
        'nombre' => 'Cosas del hogar',
    ]);

    DB::table('etiquetas')->insert([
        'nombre' => 'Ayuda',
    ]);

    DB::table('etiquetas')->insert([
        'nombre' => 'Productividad',
    ]);
}
```

Tenemos realizado estos dos seeder ahora tendremos que crear el seeder de tarea_etiqueta. Este archivo será distinto ya que aquí tendremos que realizar la conexión de las otras dos tablas, para realizar esto se hará de la manera siguiente:
Primero tendremos que crear el seeder de la misma manera que hemos creado los otros dos “sail artisan make:seeder TareaEtiquetaSeeder”

Una vez tengamos creado el seeder tendremos que rellenar el archivo de la siguiente manera:

```php
public function run(): void
{
    $tareasComida= Tarea::where('titulo','Hacer de comer')->first();
    $tareasRecoger= Tarea::where('titulo', 'Recoger')->first();
    $tareasEntrenar= Tarea::where('titulo', 'Entrenar')->first();

    $etiquetasHogar= Etiqueta::where('nombre', 'Cosas del hogar')->first();
    $etiquetasAyuda= Etiqueta::where('nombre', 'Ayuda')->first();
    $etiquetasProductividad= Etiqueta::where('nombre', 'Productividad')->first();

    $tareasComida->etiquetas()->attach($etiquetasHogar);
    $tareasRecoger->etiquetas()->attach($etiquetasAyuda);
    $tareasEntrenar->etiquetas()->attach($etiquetasProductividad);
}
```

Con esto creamos varias variables en las que le damos los nombres de los datos que están en las tablas y los atamos a las otras tablas que decimos que hagan referencia.

Ahora tendremos que subir todos los cambios a las base, para eso dentro de DatabaseSeeder.php tendremos que poner lo siguiente:

```php
class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     */
    0 references | 0 overrides
    public function run(): void
    {
        $this->call(TareasSeeder::class);
        $this->call(EtiquetasSeeder::class);
        $this->call(TareaEtiquetaSeeder::class);
    }
```

Una vez pongamos eso tendremos que ejecutar el comando “sail artisan db:seed”

5. CREACIÓN DE LOS REQUEST

Ahora tendremos que crear los request.

Para crear los request tendremos que ejecutar el comando “sail artisan make:request TareaRequest” ya que vamos a crear el request de Tarea.

```
guille@LAPTOP-6RL7Q7I4:/mnt/c/Users/guill/OneDrive/Documentos/docker/www/DWES/proyecto/apiTareasFinal$ sail artisan make
:request TareaRequest

  INFO  Request [app/Http/Requests/TareaRequest.php] created successfully.
```

Una vez ejecutado este comando dentro de “app/https/request” se habrá un creado archivo en el cual tendremos que cambiar y añadir algunos parámetros

```php
public function authorize(): bool
{
    return true;
}
```

```php
public function rules(): array
{
    return [
        'nombre'=>'required|max:40|min:3',
        'descripcion'=>'nullable|max:200|min:3',
    ];
}
```

Con esto le daremos unas reglas a los campos seleccionados, título no podrá tener más de 40 caracteres ni menos de 3 y además siempre tendrá que ir relleno, mientras que la descripción podrá ir vacía, pero si esta rellenar será entre 3 carácteres y 200.

Para crear el request de Etiqueta sería el mismo proceso, poniendo el comando “sail artisan make:request EtiquetaRequest”

```
guille@LAPTOP-6RL7Q7I4:/mnt/c/Users/guill/OneDrive/Documentos/docker/www/DWES/proyecto/apiTareasFinal$ sail artisan make
:request EtiquetaRequest

  INFO  Request [app/Http/Requests/EtiquetaRequest.php] created successfully.
```

Una vez esté ejecutado el comando se creará otro archivo pero con el nombre de etiqueta en la misma ruta que antes, al cual también le tendremos que cambiar y añadir variables.

```php
public function authorize(): bool
{
    return true;
}
```

```php
public function rules(): array
{
    return [
        'nombre'=> 'required|max:40|min:3',
    ];
}
```

6. CREACIÓN DE LOS RESOURCES

Ahora tendremos que crear los resources que harán que los datos de las tablas se pasen a formato json.

Para crear estos resources se tendrá que ejecutar el comando “sail artisan make:resource TareaResource”

```
guille@LAPTOP-6RL7Q7I4:/mnt/c/Users/guill/OneDrive/Documentos/docker/www/DWES/proyecto/apiTareasFinal$ sail artisan make
:resource TareaResource

  INFO  Resource [app/Http/Resources/TareaResource.php] created successfully.
```

Cuando lo hayamos ejecutado se creará un archivo resource en la carpeta “app/http/resource”

En este archivo tendremos que poner los datos como queramos que nos lo devuelva el formato json

```php
public function toArray(Request $request): array
{
    return [
        'id' => $this->id,
        'titulo' => $this->titulo,
        'descripcion' => $this->descripcion,
        'etiquetas' => $this->etiquetas
    ];
}
```

Esto lo hemos realizado para la parte de las tareas, ahora tendremos que realizarlo para la parte de etiquetas.

Lo cual se realizará poniendo el comando “sail artisan make:resource EtiquetaResource”

Aquí también se habrá creado un nuevo archivo el cual tendremos que modificar como antes.

```php
public function toArray(Request $request): array
{
    return[
        'id' => $this->id,
        'nombre' => $this->nombre,
        'tareas' => $this->tareas
    ];
}
```

7. CREACIÓN DE LOS CONTROLLERS

A continuación tendremos que hacer los controllers los cuales sirven para hacer las acciones de get, post, put y delete.

Antes de seguir tendremos que asegurarnos de tener instalado el postman, un programa con el cual podremos comprobar si nuestras modificaciones el los controllers del código funcionan correctamente.

Una vez esté instalado el programa empezaremos a modificar el código.

Primero realizaremos los controllers de tareas, para eso nos introducimos en TareasController.php dentro de “app/http/controller”
Ahora realizaremos la función index la cual devolverá todos los datos de las tareas.

```php
public function index():JsonResource
{
    $tareas= Tarea::all();
    return TareaResource::collection($tareas);
}
```

Ahora realizaremos la función store la cual hará que se cree una nueva tarea.

```php
public function store(Request $request)
{
    $tarea= new Tarea();
    $tarea->titulo=$request->titulo;
    $tarea->descripcion=$request->descripcion;

    $tarea->save();
    $etiquetas= $request->etiquetas;
    $idTarea=$tarea->id;
    $tarea->etiquetas()->attach($etiquetas, ['tareas_id' => $idTarea]);
    return new TareaResource($tarea);       You, hace 1 segundo • Uncommi
}
```

Ahora crearemos la función show la cual muestra los datos de una tarea en concreto que seleccionamos.

```php
public function show($idTarea)
{
    $tarea= Tarea::find($idTarea);
    return new TareaResource($tarea);
}
```

Ahora crearemos la función update la cual actualiza los datos de la tarea que seleccionemos.

```php
public function update(TareaRequest $request, $idTarea)
{
    $tarea= Tarea::find($idTarea);
    $tarea->titulo=$request->titulo;
    $tarea->descripcion=$request->descripcion;
    $tarea->etiquetas()->detach();
    $tarea->etiquetas()->attach($request->etiquetas, ['tareas_id' => $idTarea]);
    $tarea->save();

    return new TareaResource($tarea);       You, hace 1 segundo • Uncommitted chang
}
```

Y por último crearemos la función destroy la cual eliminará la tarea que nosotros decidamos.

```php
public function destroy($idTarea)
{
    $tarea= Tarea::find($idTarea);
    $tarea->etiquetas()->detach();
    $tarea->delete();
    return new TareaResource($tarea);
}
```

Cuando tenemos todas las funciones realizadas debemos ir al archivo api y allí introducir las rutas de los controllers

```php
Route::resource('/tareas', TareaController::class);
```

Ahora crearemos los controllers de Etiqueta, se realizarán las mismas funciones que se han realizado para las tareas.

Función index:

```php
public function index()
{
    $etiquetas= Etiqueta::all();
    return EtiquetaResource::collection($etiquetas);
}
```

Función store:

```php
public function store(EtiquetaRequest $request)
{
    $etiqueta= new Etiqueta();
    $etiqueta->nombre=$request->nombre;

    $etiqueta->save();

    $tareas= $request->tareas;
    $idEtiqueta=$etiqueta->id;
    $etiqueta->tareas()->attach($tareas, ['etiquetas_id' => $idEtiqueta]);
    return new EtiquetaResource($etiqueta);
}
```

Función show:

```php
public function show($idEtiqueta)
{
    $etiqueta= Etiqueta::find($idEtiqueta);
    return new EtiquetaResource($etiqueta);
}
```

Función update:

```php
public function update(EtiquetaRequest $request, $idEtiqueta)
{
    $etiqueta= Etiqueta::find($idEtiqueta);
    $etiqueta->nombre=$request->nombre;

    $etiqueta->tareas()->attach($request->tareas);
    $etiqueta->save();
    return new EtiquetaResource($etiqueta);
}
```

Función destroy:

```php
public function destroy($idEtiqueta)
{
    $etiqueta= Etiqueta::find($idEtiqueta);
    $etiqueta->delete();
    return new EtiquetaResource($etiqueta);
}
```

8. CREACIÓN DE LOS USUARIOS

Ahora tendremos que crear los usuarios, con ellos crearemos un registro, un inicio de sesión y un cierre de sesión.

Para realizar estas funciones tendremos que usar Laravel/sanctum, para poder usarlo tendremos que utilizar el comando ““sail composer require laravel/sanctum”

```
guille@LAPTOP-6RL7Q7I4:/mnt/c/Users/guill/OneDrive/Documentos/docker/www/DWES/proyecto/apiTareasFinal$ sail composer req
uire laravel/sanctum
./composer.json has been updated
Running composer update laravel/sanctum
Loading composer repositories with package information
Updating dependencies
Nothing to modify in lock file
Writing lock file
Installing dependencies from lock file (including require-dev)
Nothing to install, update or remove
Generating optimized autoload files
```

Una vez ejecutado tendremos que poner el comando “sail artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"”.

```
guille@LAPTOP-6RL7Q7I4:/mnt/c/Users/guill/OneDrive/Documentos/docker/www/DWES/proyecto/apiTareasFinal$ sail artisan vend
or:publish --provider="Laravel\Sanctum\SanctumServiceProvider"

  INFO  Publishing assets.

  Copying directory [vendor/laravel/sanctum/database/migrations] to [database/migrations] ..... DONE
  File [config/sanctum.php] already exists ..................................................... SKIPPED
```

Ahora deberemos crear un controller para Auth el cual se crea poniendo el comando “sail artisan make:controller AuthController”

```
guille@LAPTOP-6RL7Q7I4:/mnt/c/Users/guill/OneDrive/Documentos/docker/www/DWES/proyecto/apiTareasFinal$ sail artisan make
:controller AuthController

  INFO  Controller [app/Http/Controllers/AuthController.php] created successfully.
```

Ahora en este controller que hemos creado tendremos que poner las funciones que hemos mencionado anteriormente:

Función de registrar lo que hará que se registre un nuevo usuario:

```php
class AuthController extends controller{
    0 references | 0 overrides
    public function register(Request $request){
        $user = User::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password),
        ]);
        $token = $user->createToken('auth_token')->plainTextToken;
        return response()->json(['data'=>$user, 'acces_token'=>$token, 'token_type'=>'Bearer']);
    }
```

Función de inicio de sesión lo que hará que se inicie la sesión de un usuario ya registrado:

```php
    public function login(Request $request){
        $user= User::where('email', $request->email)->firstOrFail();

        if(!Hash::check($request->password, $user->password)){
            return response()->json(['message'=>'Credenciales invalidas']);
        }

        $token= $user->createToken('auth_token')->plainTextToken;

        return response()->json(['message'=>'Hola'.$user->name,
        'access_token'=> $token,
        'token_type'=>'Bearer']);
    }
```

Función de cierre de sesión lo que hará que se la sesion que esta iniciada:

```php
public  function logout(){
    auth()->user()->tokens()->delete();
    return ['message' => 'Sesion Cerrada'];
}
```

9. REALIZACIÓN DE LOS TEST

Los test sirven para asegurarnos que los endpoints funcionan correctamente.

Para empezar tendremos que crear el archivo factory el cual se crea mediante el comando “sail artisan make:factory TareaFactory”

```
guille@LAPTOP-6RL7Q7I4:/mnt/c/Users/guill/OneDrive/Documentos/docker/www/DWES/proyecto/apiTareasFinal$ sail artisan make
:factory TareaFactory

  INFO  Factory [database/factories/TareaFactory.php] created successfully.
```

Una vez ejecutado se nos creará el archivo dentro de la ruta “database/factories”, en este archivo tendremos que poner lo siguiente:

```php
public function definition(): array
{
    return [
        'titulo' => $this->faker->word,
        'descripcion' => $this->faker->sentence,
    ];
}
```

Cuando lo tengamos realizado se crearemos el archivo etiqueta factory con el comando “sail artisan make:factory EtiquetaFactory”

```
guille@LAPTOP-6RL7Q7I4:/mnt/c/Users/guill/OneDrive/Documentos/docker/www/DWES/proyecto/apiTareasFinal$ sail artisan make
:factory EtiquetaFactory

  INFO  Factory [database/factories/EtiquetaFactory.php] created successfully.
```

Y dentro del archivo que se nos ha creado en la misma ruta que antes tendremos que poner lo siguiente:

```php
public function definition(): array
{
    return [
        'nombre' => $this->faker->word,
    ];
}
```

Una vez tengamos estos archivos Factory ya bien hechos empezaremos a configurar los test.

Tendremos que poner el comando “sail artisan make:test TareaTest” el cual creará una carpeta en la ruta “tests/feature” la cual tendrá dentro el archivo TareaTest.php

Dentro de este archivo introduciremos los test que queremos realizar en este caso los test de mostrar todas las tareas, mostrar una tarea, crear una tarea, actualizar una tarea y borrar una tarea.

Test de mostrar todas las tareas:

```php
public function mostrarTareasTest(){
    $response = $this->get('/tareas');
    $response->assertStatus(200);
}
```

Test de mostrar una tarea:

```php
public function mostrarTareaTest(){
    $response = $this->get('/tareas/1');
    $response->assertStatus(200);
}
```

Test de crear una tarea:

```php
public function crearTareaTest(){
    $response = $this->post('/tareas',[
        'nombre' => 'Tarea 1',
        'descripcion' => 'Descripcion 1',
        'estado' => 'Pendiente',
    ]);
    $response->assertStatus(200);
}
```

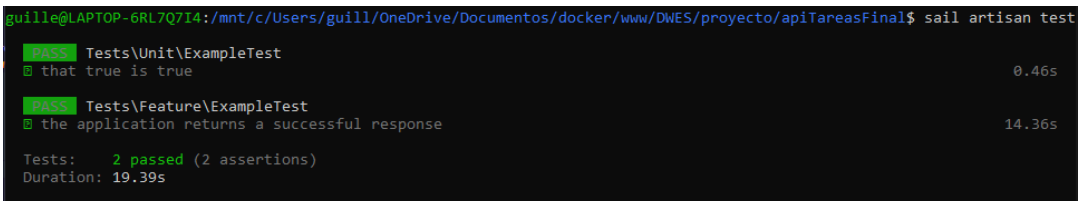
Test de actualizar una tarea:

```php
public function actualizarTareaTest(){
    $response = $this->put('/tareas/1',[
        'nombre' => 'Tarea 1',
        'descripcion' => 'Descripcion 1',
        'estado' => 'Pendiente',
    ]);
    $response->assertStatus(200);
}
```

Test de borrar una tarea:

```php
public function eliminarTareaTest(){
    $response = $this->delete('/tareas/1');
    $response->assertStatus(200);
}
```

Y ahora poniendo el comando “sail artisan test” se tendría que ver algo así si todo ha salido correctamente:


guille@LAPTOP-6RL7Q7I4:/mnt/c/Users/guill/OneDrive/Documentos/docker/www/DWES/proyecto/apiTareasFinal$ sail artisan test
PASS Tests\Unit\ExampleTest
that true is true 0.46s
PASS Tests\Feature\ExampleTest
the application returns a successful response 14.36s
Tests: 2 passed (2 assertions)
Duration: 19.39s