

Instalaciones

Instalación de composer

Tenemos varias formas de instalar composer en windows una de ellas es descargando directamente el instalado o por consola de comandos, como vemos en la siguiente imagen.

Download Composer Latest: v2.0.13

Windows Installer

The installer - which requires that you have PHP already installed - will download Composer for you and set up your PATH environment variable so you can simply call `composer` from any directory.

Download and run [Composer-Setup.exe](#) - it will install the latest composer version whenever it is executed.

Command-line installation

To quickly install Composer in the current directory, run the following script in your terminal. To automate the installation, use [the guide on installing Composer programmatically](#).

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') === '756890a4488ce9024fc62c56153228907f1545c2285'
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

This installer script will simply check some `php.ini` settings, warn you if they are set incorrectly, and then download the latest

Para linux lo instalamos de la siguiente manera primero nos movemos a la siguiente carpeta `cd ~` lo descargamos `curl -sS https://getcomposer.org/installer -o composer-setup.php` y lo instalamos de manera global para más comunidad a la hora de utilizarlo `sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer`

Explicación de laravel

¿Por qué utilizar laravel?

Laravel es bueno tanto para gente que está empezando a programar como para senior que llevan mucho tiempo en el negocio. En la web nos comentan que para los novatos su gran comunidad y la documentación, guías y videotutoriales ayudan mucho utilizar el sistema.

En cuanto a desarrollador senior, laravel le brinda herramientas sólidas para la inyección de dependencias, pruebas unitarias, colas, eventos en tiempo real y más.

También cuenta con un marco escalable, para ello tiene integrado un sistema de cache distribuido y rápido llamado Redis, también con Laravel vapor permite tener una escala ilimitada.

Instalación de Laravel

Una vez instalado el composer, ya podemos crear nuestro back con el siguiente comando `composer create-project laravel/laravel servidor` esto instalará el proyecto. Cuando acabe la instalación entramos en la carpeta donde esté instalado el servidor y lo lanzamos con el comando `php artisan serve`.

Instalación de la extensión passport

Instalaremos también la extensión que nos permitirá gestionar las autenticación, para apis en laravel llamada passport con el comando `composer require laravel/passport` esperamos a que acabe la instalación, y ya podemos empezar a hacer la configuración esenciales para nuestro proyecto. Lo primero es agregar nuestra base de datos en el proyecto para ello buscamos el archivo `.env` y agregamos el nombre de nuestra base de datos en caso de haber creado un usuario y contraseña también se lo ponemos.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=organizador
DB_USERNAME=root
DB_PASSWORD=
```

Primeros pasos después de la instalación

Ya con nuestra base de datos vinculada crearemos usaremos el siguiente comando para que laravel cree la tablas en la base de datos `php artisan migrate` ya con la base de datos lista podemos hacer `php artisan passport:install` para que nos genere las dos claves de nuestra api.

```
aythami@DESKTOP-PSEBPNM:/mnt/c/Users/friki/Documents/Aythami_Javier_PRW/servidor$ php artisan passport:install
Encryption keys already exist. Use the --force option to overwrite them.
Personal access client created successfully.
Client ID: 11
Client secret: [REDACTED]
Password grant client created successfully.
Client ID: 12
Client secret: [REDACTED]
aythami@DESKTOP-PSEBPNM:/mnt/c/Users/friki/Documents/Aythami_Javier_PRW/servidor$
```

Empezamos con las configuraciones para passport primero nos vamos al nuestro modelo de users y agregamos las siguientes líneas `use Laravel\passport\HasApiTokens;` y `use`

```
HasApiTokens,HasFactory, Notifiable;.
```

```
<?php

namespace App\Models;

use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\passport\HasApiTokens;

class User extends Authenticatable
{
    use HasApiTokens,HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];
}
```

Ahora usamos nos vamos al archivo AuthServiceProvider y ponemos las siguientes líneas `use Laravel\passport\Passport; Passport::routes();`

```
use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;
use Illuminate\Support\Facades\Gate;
use Laravel\passport\Passport;

class AuthServiceProvider extends ServiceProvider
{
    /**
     * The policy mappings for the application.
     *
     * @var array
     */
    protected $policies = [
        // 'App\Models\Model' => 'App\Policies\ModelPolicy',
    ];

    /**
     * Register any authentication / authorization services.
     *
     * @return void
     */
    public function boot()
    {
        $this->registerPolicies();
        Passport::routes();

        //
    }
}
```

También llevaremos un versionado de las distintas apis según vamos trabajando en ello, ya que esta es una buena práctica para la industria, llevaremos el siguiente formato, también se irá actualizando la siguiente tabla.

Version	Funciones	Fecha
V1	Creación de la api y login de usuarios con passport	21/04/2021
V2	Gestion de usuarios y archivos documentos	21/04/2021-02/05/2021
V3	Gestión de categorias y usuarios	02/05/2021-16/05/2021

para llevar este versionado a cambio modificaremos los siguientes archivos del RouterServiceProvider y ponemos la siguiente configuración.

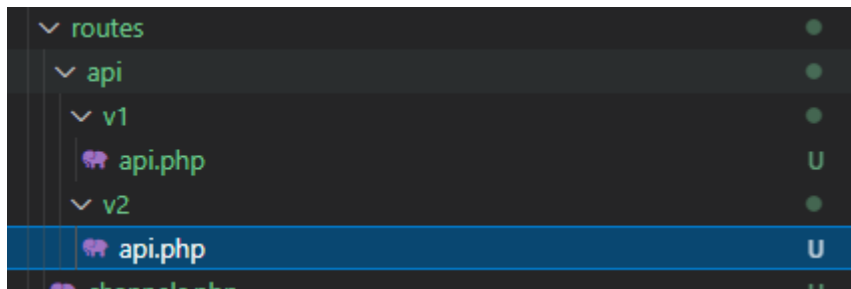
```
public function boot()
{
    $this->configureRateLimiting();

    $this->routes(function () {
        Route::prefix('api/v1')
            ->middleware('api')
            ->namespace($this->namespace)
            ->group(base_path(['routes/api/v1/api.php']));

        Route::middleware('web')
            ->namespace($this->namespace)
            ->group(base_path('routes/web.php'));
    });
}
```

/**

y creamos esa estructura de carpetas en routes.



Generamos las rutas de los usuarios

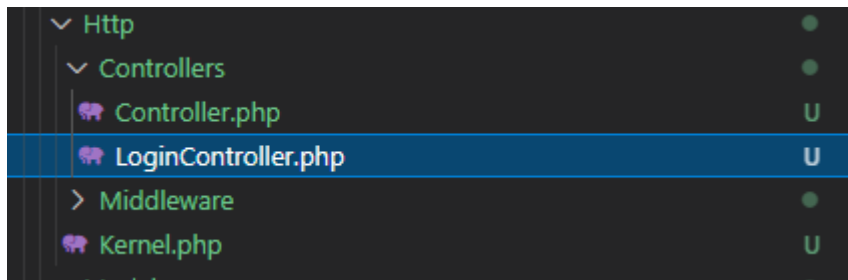
Para ello lo primero que vamos a hacer es ir a la carpeta `v1/api.php` que comentamos antes. y agregamos comentamos la línea que contiene `Route::middleware` y procedemos a crear un prefijo para nuestros comandos de login. Y agregamos nuestra primera ruta a `/login` en nuestro controlador.

```
use Illuminate\Support\Facades\Route;

/*
|-----
| API Routes
|-----
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/
/*
Route::middleware('auth:api')->get('/user', function (Request $request) {
    return $request->user();
});*/

Route::prefix('/user')->group(function(){
    Route::post('/login', 'loginController@login');
});
```

Ahora procedemos a crear nuestro contralor de con el siguiente comando `php artisan make:controller LoginController`.



En ese controlador primero importamos `use Illuminate\Http\Request;` y `use Illuminate\Support\Facades\Auth;` Aprovechamos el Auth que nos proporciona laravel para comprobar si es correcto el login.

```

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class LoginController extends Controller
{
    public function login(Request $request)
    {
        $credentials = $request->only('email', 'password');
        //aprovechamos el Auth que nos proporciona laravel apra comprobar si es correcto el login
        //con 422 o 401 devolvemos que no esta autentificado
        if (!Auth::attempt($credentials)) {
            return response([
                "message" => "Usuario y/o contraseña es invalido."
            ], 422);
        }
        $accessToken= Auth::user()->createToken('authTestToken')->accessToken;
        return response([
            "user" => Auth::user(),
            "access_token" => $accessToken
        ]);
    }
}

```

Debemos devolver un 422 o 401 que nos informa de que no está autenticado.

```

3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\Auth;
7
8 class LoginController extends Controller
9 {
10     public function login(Request $request)
11     {
12         $credentials = $request->only('email', 'password');
13         //con 422 o 401 devolvemos que no esta autentificado
14         if (!Auth::attempt($credentials)) {
15             return response([
16                 "message" => "Usuario y/o contraseña es invalido."
17             ], 422);
18         }
19         $accessToken= Auth::user()->createToken('authTestToken')->accessToken;
20         return response([
21             "user" => Auth::user(),
22             "access_token" => $accessToken
23         ]);
24     }
25 }

```

En caso de que pase el primer if creamos un token de acceso y se lo enviamos a como respuesta. Para ello usamos otra clase de Auth que es user->createToken.

Con esto realizado volveremos a api para agregar un poquito mas de seguridad a nuestro controlador para ello generamos un middleware dentro del grupo de `/user` que use `auth:api` para proteger toda las rutas dentro del controlador que no tengan el token registrado.

```

9 |-----
10 |
11 | Here is where you can register API routes for your application. These
12 | routes are loaded by the RouteServiceProvider within a group which
13 | is assigned the "api" middleware group. Enjoy building your API!
14 |
15 | */
16 | /*
17 | Route::middleware('auth:api')->get('/user', function (Request $request) {
18 |     return $request->user();
19 | });*/
20 |
21 | Route::prefix('/user')->group(function(){
22 |     Route::post('/login', 'App\Http\Controllers\LoginController@login');
23 |     //con este middleware protegemos las rutas dentro de /user
24 |     Route::middleware('auth:api')->get('/all', 'App\Http\Controllers\LoginController@all');
25 | });
26 |
27 |

```

Como medida de seguridad extra vamos a darle un tiempo de vida útil a nuestro token de acceso para ello nos vamos nuevamente a AuthServiceProvider y agregamos la siguiente función de passport `Passport::personalAccessTokenExpireIn(now()->addHours(tiempo en horas));` en mi caso y para probar le he agregado 24 horas de validez al token.

```

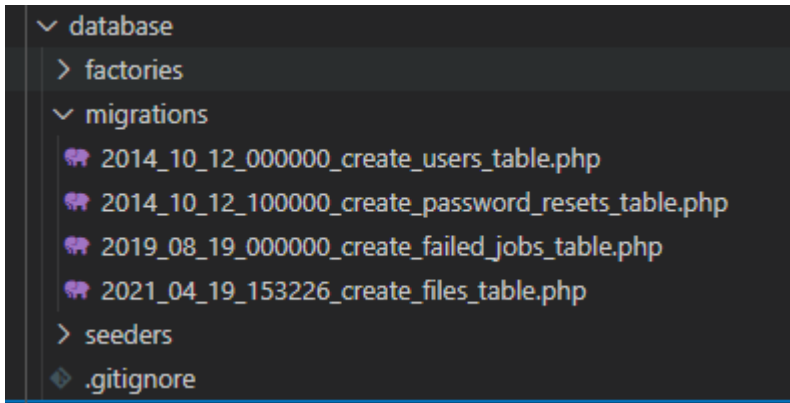
24 | */
25 | public function boot()
26 | {
27 |     $this->registerPolicies();
28 |     Passport::routes();
29 |     Passport::personalAccessTokenExpireIn(now()->addHours(24));
30 |
31 |     //
32 | }
33 | }
34 |

```

Migraciones con Laravel Y Eloquent.

Vamos a ver que el ORM que utiliza eloquent tiene diversas funciones que iremos explicando la primera de ellas es la posibilidad de guardar la base datos y todos sus cambios en el apartado de migraciones para ello procederemos a usar los siguiente comandos.

Para ello usaremos el siguiente comando `php artisan make:migration create_files_table` donde `files` es el nombre en plural de la tabla a utilizar, este siempre tiene que estar en minúscula. Nos generará el siguiente archivo.



Esta nos generará las siguientes 2 funciones dentro de la clase que hemos creado una Up que se encarga de crear las tablas en base de datos y una down que cuando acaba se encarga de borrar.

```
class CreateFilesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('files', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('description');
            $table->date('file_date');
            $table->string('up_date');
            $table->foreignId('user_id')->references('id')->on('users');
        });
    }

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

Nos centraremos en la función up que se encarga de crear las tablas en la base de datos viene ya por defecto con id y con un timestamp. y nosotros le hemos agregado los campos que necesitamos, otro dato interesante es que a la hora de agregar las claves foráneas usaremos foreign o foreignId en caso de usar id.

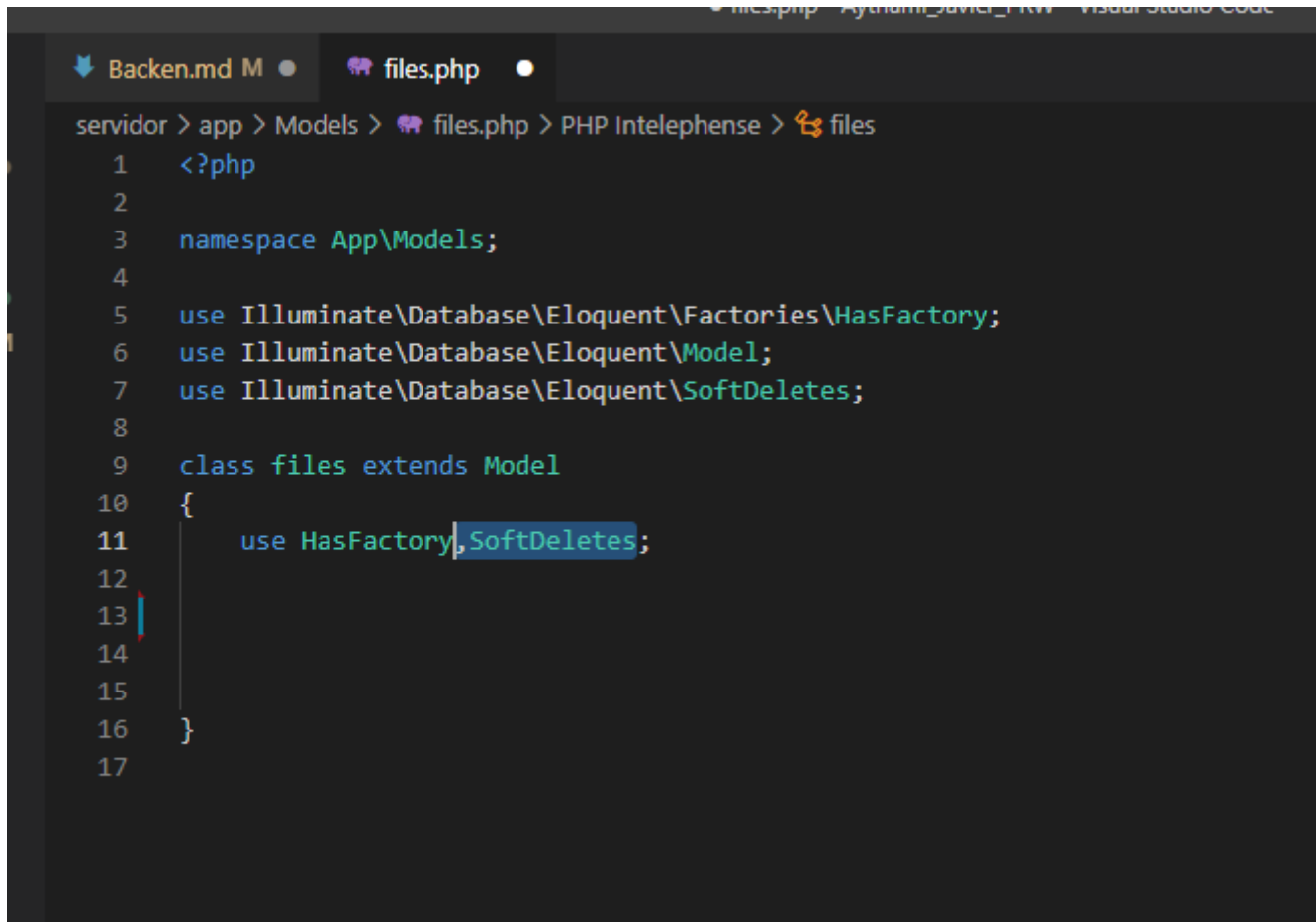
```
*/  
public function up()  
{  
    Schema::create('files', function (Blueprint $table) {  
        $table->id();  
        $table->string('name');  
        $table->string('description');  
        $table->date('file_date');  
        $table->string('up_date');  
        $table->timestamp('created_at')->nullable();  
        $table->foreignId('user_id')->references('id')->on('users');  
    });  
}
```

Modelos con Laravel Y Eloquent.

Vamos a empezar a preparar nuestros modelos para ello usaremos el comando `php artisan make:model Files` donde files es el nombre que tendrá nuestro modelo, esto nos generará la siguiente clase.

```
Backen.md M ● files.php ●  
servidor > app > Models > files.php > PHP Intelephense > files  
1  <?php  
2  
3  namespace App\Models;  
4  
5  use Illuminate\Database\Eloquent\Factories\HasFactory;  
6  use Illuminate\Database\Eloquent\Model;  
7  use Illuminate\Database\Eloquent\SoftDeletes;  
8  
9  class files extends Model  
10 {  
11     use HasFactory;  
12  
13  
14  
15  
16 }  
17
```

Ahora agregamos el `softDelete` que nos proporciona la función de que nuestras clases no se borren sino que se oculten para el usuario manteniendo así los datos en caso de pérdida.

A screenshot of a code editor window. The top bar shows two tabs: 'Backen.md M' and 'files.php'. The breadcrumb navigation at the top reads: 'servidor > app > Models > files.php > PHP Intelephense > files'. The code is written in PHP and is as follows:

```
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7  use Illuminate\Database\Eloquent\SoftDeletes;
8
9  class files extends Model
10 {
11     use HasFactory, SoftDeletes;
12
13
14
15
16 }
17
```

Empezamos a agregar los atributos para nuestras funciones con `$guarded` creamos campos protegidos en nuestro modelo en este caso es `delete_at` que sirve para que laravel guarde si han sido borrados los campos o no. `$fillable` ponemos los campos a los que queremos acceder, y con `$hidden` campos que quieres que estén ocultos cuando trabajas con los modelos como la contraseña.

```
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7  use Illuminate\Database\Eloquent\SoftDeletes;
8
9  class files extends Model
10 {
11     use HasFactory,SoftDeletes;
12
13     protected $guarded = ['deleted_at'];
14
15     protected $fillable = [
16         'name',
17         'description',
18         'file_date',
19         'up_date',
20         'user_id'
21     ];
22 }
23
24
25
26
```

Relaciones en los modelos Eloquent.

Aquí veremos la relación uno muchos que encontramos en nuestro modelo usuario archivos donde un usuario pueden tener muchos archivos. Para ello nos vamos primero a user y ponemos la siguiente función `hasMany(File::class)` y en Files ponemos lo siguiente `$this->belongsTo(User::class)` como se ven en las siguientes imágenes.

```
public function files()
{
    return $this->hasMany(File::class);
}
```

```
public function user()
{
    return $this->belongsTo(User::class);
}
```

En esta ocasión tenemos una relación muchos a muchos entre categorías y archivos donde un archivo tiene muchas categorías y las categorías pertenecen a varios archivos. con `belongsToMany(File::class)` en este caso usa la misma nomenclatura en los dos modelos, también en la documentación encontramos otra

forma de hacerlo que es `return $this->belongsToMany(Role::class, 'role_user', 'user_id', 'role_id');`; esta segunda opcion es si quieres ponerle un nombre diferente a la tabla intermedia.

```
public function categories()
{
    return $this->belongsToMany(Categories::class);
}
```

```
public function files()
{
    return $this->belongsToMany(Files::class);
}
```

Controladores.

Empezamos a usar los controladores para ello vamos a usar el comando `php make:controller nombre --api` en el controlador aparecerán las siguientes funciones index, store, show, update y delete. Index nos suele usar para enviar la información principal del modelo al que pertenece con un `::all` llamando al modelo, más adelante le iremos complicando la lógica.

```
public function index()
{
    $files = Files::all();
    return $files;
}

/**
 * Store a newly created resource in storage.
 */
```

Store nos permite almacenar los archivos que nos llegan de nuestra vista con `::create` podemos ir guardando todo lo que nos llega. Podemos devolver la respuesta y la información que se ha guardado de en la base de datos.

```
public function store(Request $request)
{
    $file = Files::create($request->all);
    return $file;
}
```

Show se usa para buscar un archivo en concreto en este caso usa la función `::find($id)` haciendo la búsqueda por id.

```
*/  
public function show($id)  
{  
    return Files::find($id);  
}
```

En el update primero vamos a validar los datos que nos están entrando por el request para que estos no vengan vacíos y luego procedemos a buscar el archivo que queremos lo modificamos y guardamos.

```
public function update(Request $request)  
{  
    $request->validate([  
        'name' => 'required',  
        'description' => 'required',  
        'file_date' => 'required',  
    ]);  
  
    $data = Files::findOrFail($request->id);  
    $data ->name = $request->name;  
    $data ->description = $request->description;  
    $data ->file_date = $request->file_date;  
    $data->save();  
}
```

Delete es muy parecido al anterior lo que hacemos es buscar en la base de datos si existe en documento y si existe lo eliminamos aquí tienes dos opciones poner delete que va a hacer un soft delete y otra que es destroy que elimina el archivo completamente.

```
public function delete($id)  
{  
    Files::find($id)->delete();  
}
```

Funcionalidades del backend

Aparte de lo básico que se va a usar en el código se han integrado aquí se van explicar cómo se han ido cambiado las cosas para obtener el buen funcionamiento de la aplicación.

Manejando archivos.

Para manejar archivos vamos al store de nuestro controlador files y agregamos unas validaciones para ello usamos el validate de laravel y lo primero que vamos a hacer es que sea requerido, también podemos elegir el tipo de archivo de que puede aceptar con `require|image|max:2048` donde image es el tipo requerido y el max es el tamaño máximo del archivo en kilobytes. De momento voy a limitar las subidas de archivos a 500 megas ya que la aplicación es para guardar documentos, en caso de necesitar más espacio de subida se mirara a posteriori.

```
*/  
public function store(Request $request)  
{  
    $request->validate([  
        'file'=>'required|max:500000',  
        'name' => 'required',  
        'description' => 'required',  
        'file_date' => 'required',  
    ]);  
  
    $file = Files::create($request->all);  
    return $file;  
}  
/**
```

Ahora una vez el archivo validado vamos a ver como lo almacenamos en nuestra carpeta para ello voy a buscar al usuario en cuestión para guardarlo en su carpeta que ha sido previamente creada a la hora de insertar al usuario. Ya con el usuario conseguido guardamos el archivo en la carpeta correspondiente.

```
public function store(Request $request)  
{  
    $request->validate([  
        'token'=> 'required',  
        'file'=>'required|max:500000',  
        'name' => 'required',  
        'description' => 'required',  
        'file_date' => 'required',  
    ]);  
  
    $user = User::where('remember_token', $request->token)->first();  
    $request->file('file')->store($user->user_folder.'/');  
    $file = Files::create($request->all);  
    return $file;  
}
```

Mejorando las respuestas de nuestro servidor

Ahora que ya vemos que responde correctamente nuestro servidor, vamos a mejorar la respuesta que nos da para ello acudimos a la funcionalidad de eloquent resource, para ello crearemos el resource de archivo que nos facilitara la conversiones en json para dar una respuesta más completa por parte de nuestro servidor. Primero usamos el comando `php artisan make:resource Archivo` y se genera la clase encargada de gestionar nuestro json.

```
servidor > app > Http > Resources > ArchivoResource.php > PHP Intelephense > ArchivoResource
1  <?php
2
3  namespace App\Http\Resources;
4
5  use Illuminate\Http\Resources\Json\JsonResource;
6
7  class ArchivoResource extends JsonResource
8  {
9      /**
10       * Transform the resource into an array.
11       *
12       * @param \Illuminate\Http\Request $request
13       * @return array
14       */
15      public function toArray($request)
16      {
17          return parent::toArray($request);
18      }
19  }
20
```

a continuación vamos a mejorar la respuesta de nuestro controlador, `return response(['archivos'=>ArchivoResource::collection($files), 'message'=>'Retrived Successfully'],200);` con esta línea aplicamos el json de nuestros archivos que han sido buscador previamente, junto con un mensaje informando que noto a ido bien y un estatus 200.

```
public function index()
{
    $user = auth()->user();
    $files = Archivo::where('user_id', $user->id)->with('categoria')->get();
    return response(['archivos'=>ArchivoResource::collection($files), 'message'=>'Retrived Successfully'],200);
}

/**
```

También tenemos que poner la carpeta storage accesible para la web para ello usamos el siguiente comando `php artisan storage:link`.

Contraseña olvidada

En este caso vamos a ver como creamos un envío de email con token de para recuperar contraseña que nos proporciona passport.


```
public function forgotPassword(Request $request)
{
    $request->validate([
        'email' => 'required|string|email',
    ]);

    $email = $request->input('email');

    if (User::where('email', $email)->doesntExist()) {
        return response([
            'message' => 'El usuario no existe'
        ], 404);
    }

    $token = Str::random(10);

    DB::table('password_resets')->insert([
        'email' => $email,
        'token' => $token
    ]);

    //Enviamos el email
    Mail::send('Emails.forgot', ['token' => $token], function ($message) use ($email) {
        $message->to($email);
        $message->subject('Reinicia tu contraseña');
    });
    return response([
        'message' => 'Revisa tu email'
    ], 200);
}
```

Para ello primero validamos que el email llegue relleno con la función validate, luego buscamos si existe en la base de datos si es así generamos un contraseña ramdon y la guardamos en la tabla password_resets, directamente ya que no tenemos un modelo para esa tabla.

El mail para ello tenemos que hacer uso de la función Mail de Laravel para ello las escribimos de la siguiente manera.

```
//Enviamos el email
Mail::send('Emails.forgot', ['token' => $token], function ($message) use ($email) {
    $message->to($email);
    $message->subject('Reinicia tu contraseña');
});
return response([
    'message' => 'Revisa tu email'
], 200);
}
```

También tenemos que configurar los datos de nuestro servidor de email en este caso usaremos gmail de manera gratuita que nos permite 100 correos diarios. Para ello nos vamos a nuestro archivo env.

```
MAIL_MAILER=smtp
MAIL_HOST=ssl://smtp.gmail.com
MAIL_PORT=465
MAIL_USERNAME=martinaythami459@gmail.com
MAIL_PASSWORD=
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS=null
MAIL_FROM_NAME="${APP_NAME}"

AWS_ACCESS_KEY_ID=
```

modificar contraseña

Esta función lo que nos permite es con el paso anterior de gestionar una contraseña con el enlace que se manda al correo recuperarla.

```
public function ResetPassword(Request $request)
{
    $request->validate([
        'token' => 'required',
        'password' => 'required',
        'password_confirm' => 'required|same:password'
    ]);
    $token = $request->input('token');

    if (!$passwordResets = DB::table('password_resets')->where('token', $token)->first())
        return response([
            'message' => 'Token invalido'
        ], 404);

    if (!$user = User::where('email', $passwordResets->email)->first()) {
        return response([
            'message' => 'El usuario no existe'
        ], 404);
    }

    $user->password = bcrypt($request->input('password'));
    $user->save();

    return response([
        'message' => 'Success'
    ]);
}
```

Para ello validamos el token que se te ha enviado al correo y las contraseñas introducidas. Vemos si el token generado y el que está guardado en la tabla password_resets coinciden en ese caso pasamos a ver si el usuario existe y ya procedemos a guardar la nueva contraseña en el usuario.

Distintos tipos de búsqueda

Aquí vamos a implementar un par de búsquedas extras aparte de la que vienen por "defecto", ya que no son tan comunes, entre ellas tenemos la búsqueda de archivos por categoría, después de darle una vuelta la forma más sencilla es filtrar el contenido de la siguiente manera .

```

/**
 * @return Illuminate\Http\Response
 */
public function showCategoria($idCategoria)
{
    $archivos = [];
    $user = auth('api')->user();
    $files = Archivo::where('user_id', $user->id)->with('categoria')->get();
    foreach ($files as $file) {
        foreach ($file->categoria as $cat) {
            if ($cat->pivot->categoria_id == $idCategoria) {
                array_push($archivos, $file);
            }
        }
    }
    return response(['archivos' => ArchivoResource::collection($archivos), 'message' => 'Retrived Successfully'], 200);
}

/**
 * Store a newly created resource in storage.
 */

```

Una búsqueda por fecha.

Errores personalizados

Agregamos una respuesta 404 si es usuario ya existe con un mensaje de error personalizado. Como se ve en la siguiente imagen.

```

public function createUser(Request $request)
{
    $request->validate([
        'name' => 'required|string',
        'email' => 'required|string|email',
        'password' => 'required|string',
        'password_confirm' => 'required|same:password'
    ]);
    if (User::where('email', $request->email)->first()) {
        return response([
            'message' => 'El email ya existe'
        ], 404);
    } else {
        $name_folder = Str::random(128);

        User::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => bcrypt($request->password),
            'user_folder' => $name_folder
        ]);

        Storage::disk('local')->put($name_folder . '/prueba.txt', 'Contents');
        return response()->json([
            'message' => 'El usuario se creo correctamente'
        ], 201);
    }
}

```

verificación en dos pasos

Para implementar este método en la api me he regido por la documentación encontrada aqui <https://dev.to/chandreshhere/laravel-email-verification-apis-246c> la cual nos permite enviar un correo para verificar que este existe. Para ello realizaremos los siguiente cambios. Primero implementamos la interfaz `MustVerifyEmail` en nuestro modelo user.

```
9 use Laravel\Passport\HasApiTokens;
10
11 class User extends Authenticatable implements MustVerifyEmail
12 {
13     use HasApiTokens, HasFactory, Notifiable;
14
15     /**
16      * The attributes that are mass assignable.
17      *
18      * @var array
19      */
20     protected $fillable = [
```

Después nos vamos a generar una nuevo controlador para gestionar estas dos páginas nuevas funciones una que recoja el email de verificación y otra que envíe el mail para ello usamos el comando `php artisan make:controller VerificationController`

Problemas

Relación muchos a muchos

Uno de los mayores problemas a la hora de usar laravel ha sido que no me reconocían las relaciones muchos a muchos entre las relaciones porque laravel tiene definidas las relaciones que los modelos tienen que ir singular y las relaciones tablas en plural. Pasando de usar los nombres de los modelos de inglés a español al igual que las tablas para poder comprender el error bien. Ya que laravel coge los modelos y los campos de las tablas en singular y las tablas en plural y no reconoce las relaciones. También me ayuda el `php artisan tinker` que te permite ejecutar la funciones del controlador que quieres y ver los errores en la base de datos.

Corrección de las rutas

Vamos a hacer unas correcciones a las rutas en la v3 por lo que no estaban funcionando bien para ello se le ha puesto un `Route::prefix('/categorias')->group(function ()` que nos permite poner todas las categorías dentro de un grupo, gracias a esto podemos acceder a través de `http://localhost:8000/api/v3/categorias/update` por ejemplo.

```
Route::prefix('/user')->group(function () {
    Route::post('/login', 'App\Http\Controllers\LoginController@login');
    Route::post('/createUser', 'App\Http\Controllers\LoginController@createUser');
    Route::post('/logout', 'App\Http\Controllers\LoginController@logout');
});

Route::prefix('/files')->group(function () {
    Route::get('/index', [ArchivosController::class, 'index']);
    Route::post('/store', [ArchivosController::class, 'store']);
    Route::get('/{id}', [ArchivosController::class, 'show']);
    Route::post('/update', [ArchivosController::class, 'update']);
    Route::post('/delete', [ArchivosController::class, 'delete']);
});
Route::prefix('/categorias')->group(function () {
    Route::get('/index', [CategoriasController::class, 'index']);
    Route::post('/store', 'App\Http\Controllers\CategoriasController@store');
    Route::get('/{id}', [CategoriasController::class, 'show']);
    Route::post('/update', [CategoriasController::class, 'update']);
    Route::post('/delete', [CategoriasController::class, 'delete']);
});
```

Corrección en los controladores de categorías

Se a cambiado la forma de obtener los datos del usuario conectado `$user = auth()->user();` que nos proporciona el usuario cuando se usa la plantilla de laravel a este otro `$user = auth('api')->user();` que es el funciona con las api.

Pruebas

Para probar el funcionamiento de los endpoint del backen he usado postman, una herramienta que nos permite ver la información que estamos introduciendo y la que que recibimos a continuación voy a poner las pruebas que corresponden a cada uno de los endpoint.

```

Route::get('email/verify/{id}', 'App\Http\Controllers\EmailVerificationController@verify');

Route::get('email/resend', 'App\Http\Controllers\EmailVerificationController@resend');

Route::prefix('/user')->group(function() {
    Route::post('/login', 'App\Http\Controllers\UserController@login');
    Route::post('/createUser', 'App\Http\Controllers\UserController@createUser');
    Route::post('/logout', 'App\Http\Controllers\UserController@logout');
    Route::post('/forgot', 'App\Http\Controllers\UserController@forgot');
    Route::post('/resetpassword', 'App\Http\Controllers\UserController@resetpassword');
});

Route::prefix('/archivos')->group(function() {
    Route::get('/index', [ArchivosController@index]);
    Route::post('/store', [ArchivosController@store]);
    Route::get('/{id}', [ArchivosController@show]);
    Route::get('/recuperarArchivo/{id}', [ArchivosController@recuperarArchivo]);
    Route::get('/showCategoria/{id}', [ArchivosController@showCategoria]);
    Route::post('/update', [ArchivosController@update]);
    Route::post('/delete/{id}', [ArchivosController@delete]);
});

Route::prefix('/categorias')->group(function() {
    Route::get('/index', [CategoriasController@index]);
    Route::post('/store', 'App\Http\Controllers\CategoriasController@store');
    Route::get('/{id}', [CategoriasController@show]);
    Route::post('/update', [CategoriasController@update]);
    Route::post('/delete/{id}', [CategoriasController@delete]);
});

```

Como se ve en la imagen contamos con 2 rutas de email puestas por ultimo para el funcionamiento de la validación de correo.

Estas dos solo generan un link de solo uso que una vez usado activan en la base de datos y redirigen al front.

clases / validacion

Save

Send

GET

http://localhost:8000/api/v3/email/verify/11?expires=1621529910&hash=bacb6811b00d8156a93e0d9ab1bc3f74eb3d7d8c&signature=b470efefccf5fb3e37cdfa7f6c39f2e5ee899c8baa050a5ab36bfc888de11c1c

Params

Authorization

Headers (10)

Body

Pre-request Script

Tests

Settings

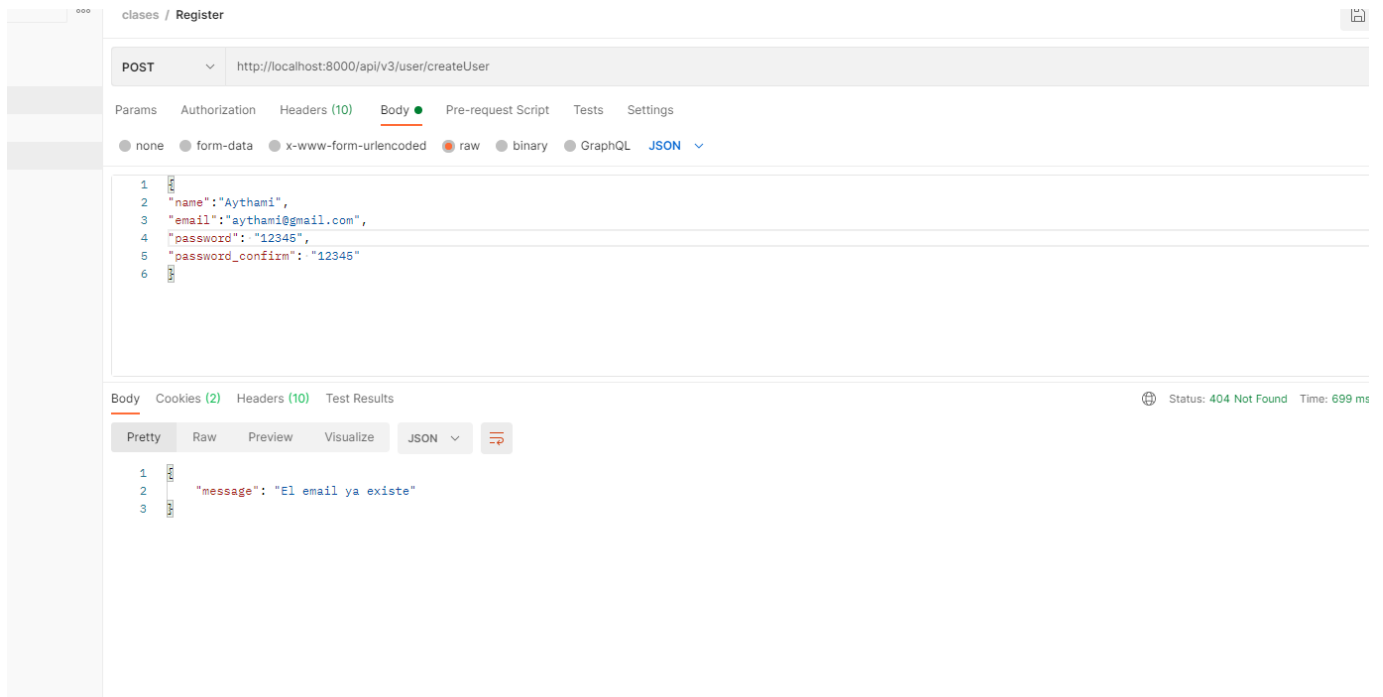
Cookies

Query Params

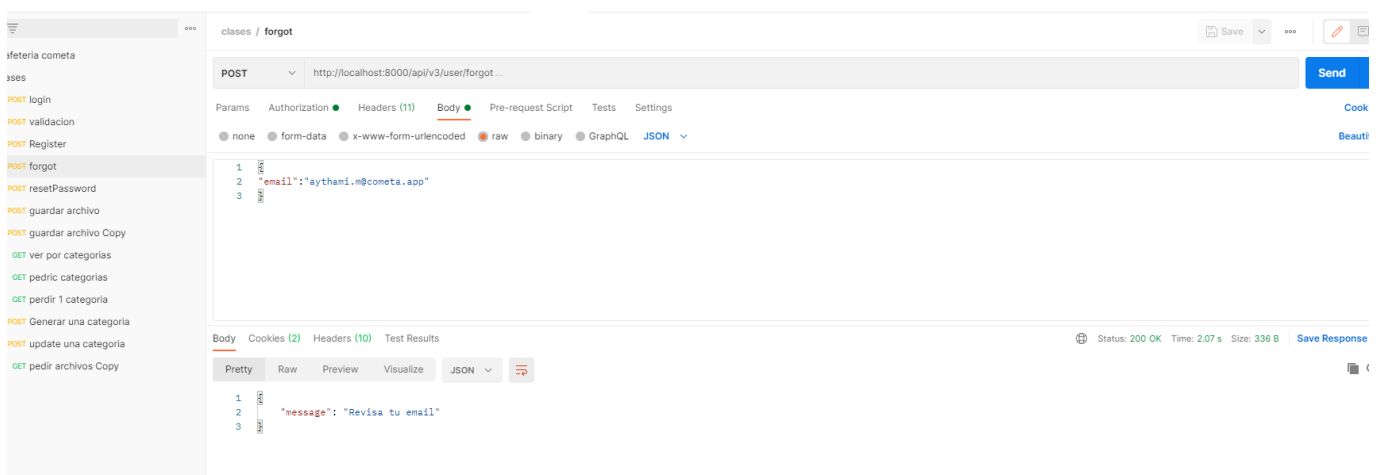
KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> expires	1621529910	
<input checked="" type="checkbox"/> hash	bacb6811b00d8156a93e0d9ab1bc3f74eb3d7d8c	
<input checked="" type="checkbox"/> signature	b470efefccf5fb3e37cdfa7f6c39f2e5ee899c8baa050a5ab36bfc888de11c1c	
Key	Value	Description

Response

- /login que nos permite iniciar sesión en nuestra aplicación como se ven la imagen pide email y contraseña y nos devuelve los datos de usuario necesarios para su funcionamiento. Es de tipo post.



- /forgot de tipo post nos permite enviar un reset password al correo y comprueba si el correo está el sistema y nos responde un revisa tu email.



- /resetpassword nes de tipo post y nos permite enviar nuestro token que hemos enviado por correo y la nueva contraseña cuenta con los campos token, password y password_confirm.

The screenshot displays a REST client interface with a POST request to `http://localhost:8000/api/v3/user/restpassword ...`. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   "token": "GqWLMnVizp",  
3   "password": "123456",  
4   "password_confirm": "123456"  
5 }
```

Below the request, the 'Test Results' tab is active, showing the response body in 'Pretty' format:

```
1 {  
2   "message": "Success"  
3 }
```

En /archivos ya tiene varios middleware uno que nos pedirá el token del usuario y otra que la cuenta esté verificada.

- El / o /index nos trae todos los archivos guardados en la base de datos es una petición de tipo get.

Cafeteria cometa

clases

POST login

POST forgot

POST resetPassword

POST Register

GET pedir archivos

GET pedir categorias

GET pedir 1 categoria

POST Generar una categoria

POST update una categoria

GET pedir archivos por categorias

GET pedir archivos Copy

GET http://localhost:8000/api/v3/archivos/

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Type Bearer Token

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collabo [Learn more about variables](#)

Token eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.e...

Body Cookies (2) Headers (10) Test Results

Pretty Raw Preview Visualize JSON


```

1  {
2    "archivos": [
3      {
4        "id": 1,
5        "name": "Radiografia",
6        "description": "Agregar radiografia",
7        "file_date": "2021-05-18",
8        "file_name": "radiografiapie.jpg",
9        "up_date": "2021-05-18",
10       "user_id": 9,
11       "created_at": null,
12       "updated_at": null,
13       "deleted_at": null,
14       "categoria": [
15         {
16           "id": 2,
17           "name": "Radiografias",
18           "categoria": 1,

```

- Con /showCategoria/{id} podemos traernos todos los archivos que están relacionados con una categoria. Es una petición de tipo get.

clases / pedir archivos por categorias

GET http://localhost:8000/api/v3/archivos/showCategoria/2

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Type Bearer Token

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Token eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.e...

Body Cookies (2) Headers (10) Test Results

Pretty Raw Preview Visualize


```

{"archivos":[{"id":1,"name":"Radiografia","description":"Agregar radiografia","file_date":"2021-05-18","file_name":"radiografiapie.jpg","up_date":"2021-05-18","user_id":9,"created_at":null,"updated_at":null,"deleted_at":null,"categoria":[{"id":2,"name":"Radiografias","categoria":1,"user_id":9,"deleted_at":null,"created_at":"2021-05-18T15:38:28.000000Z","updated_at":"2021-05-18T15:38:28.000000Z","pivot":{"archivo_id":1,"categoria_id":2}}]},{"message":"Retrieved Successfully"}]}

```


Status: 200 OK Time: 793 ms Size: 784 B Save Respo

- /id para traer una de los archivos, es de tipo get .

GET ⌵ http://localhost:8000/api/v3/archivos/2

Params Authorization ● Headers (8) Body Pre-request Script Tests Settings

Type Bearer Token ⌵

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

! Heads up! These parameters hold sensitive data. To keep this data safe, we've masked it. [Learn more about variables](#)

Token eyJ0eXAiOiJKV1Qi

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON ⌵ ≡

```

1  {
2    "id": 2,
3    "name": "Receta medica",
4    "description": "medica",
5    "file_date": "2021-05-18",
6    "file_name": "receta.png",
7    "user_id": 9,
8    "created_at": null,
9    "updated_at": null,
10   "deleted_at": null,
11   "categoria": [
12     {
13       "id": 5,
14       "name": "Recetas",
15       "categoria": 1,
16       "user_id": 11,
17       "deleted_at": null,
18       "created_at": "2021-05-18T15:42:49.000000Z",
19       "updated_at": "2021-05-18T15:57:30.000000Z",
20       "pivot": {
21         "archivo_id": 2,
22         "categoria_id": 5

```

- /store nos permite recoger un archivo file y el resto de información que nos ofrece el usuario, este puede dar dos respuestas el archivo guardado como resultado de que este se ha guardado correctamente o un error de que el archivo ya existe.

new import

classes / pedir archivos

POST http://localhost:8000/api/v3/archivos/store

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> file	product_1.PNG	Archivo de producto
<input checked="" type="checkbox"/> name	Producto	
<input checked="" type="checkbox"/> description	Un fato de un producto	
<input checked="" type="checkbox"/> file_date	2021-05-15	
<input checked="" type="checkbox"/> categories[0]	1	
<input checked="" type="checkbox"/> categories[1]	2	
Key	Value	Description

Body Cookies Headers (10) Test Results

Status: 201 Created Time: 749 ms Size: 529 B Save F

Pretty Raw Preview Visualize

```
{
  "name": "Producto",
  "description": "Un fato de un producto",
  "file_date": "2021-05-15",
  "file_name": "product_1.PNG",
  "user_id": 9,
  "updated_at": "2021-05-18T18:59:27.000000Z",
  "created_at": "2021-05-18T18:59:27.000000Z",
  "id": 48
}
```

classes / guardar archivo

POST http://localhost:8000/api/v3/archivos/store ...

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

KEY	VALUE
<input checked="" type="checkbox"/> file	product_1.PNG
<input checked="" type="checkbox"/> name	Producto
<input checked="" type="checkbox"/> description	Un fato de un producto
<input checked="" type="checkbox"/> file_date	2021-05-15
<input checked="" type="checkbox"/> categories[0]	1
<input checked="" type="checkbox"/> categories[1]	2

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "El archivo ya existe"
3 }
```

- /update nos permite actualizar tanto el archivo como los datos que guardamos de estos. Esta función comprueba el archivo que has subido y si es distinto al que ya tenemos lo reemplaza. La diferencia con el storage es que se pide también el id del archivo

clases / update archivo

POST ⌵ http://localhost:8000/api/v3/archivos/update

Params Authorization ● Headers (11) Body ● Pre-request Script Tests Settings

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

<input checked="" type="checkbox"/>	id	55
<input checked="" type="checkbox"/>	name	Producto upda
<input checked="" type="checkbox"/>	description	Un fato de un producto
<input checked="" type="checkbox"/>	file_date	2021-05-15
<input checked="" type="checkbox"/>	categories[0]	1
<input checked="" type="checkbox"/>	categories[1]	2
	Key	Value

Body Cookies (2) Headers (10) Test Results

Pretty Raw Preview Visualize JSON ⌵ ≡

```
1  {
2    "id": 55,
3    "name": "Producto upda",
4    "description": "Un fato de un producto",
5    "file_date": "2021-05-15",
6    "file_name": "product_1.PNG",
7    "user_id": 9,
8    "created_at": "2021-05-19T15:44:26.000000Z",
9    "updated_at": "2021-05-23T12:28:45.000000Z",
10   "deleted_at": null
11 }
```

- /recuperarArchivo nos permite recuperar la información del archivo. Para ello se introduce un id del archivo. Haciéndolo de esta manera el archivo no tiene url publico.

GET

⌵

http://localhost:8000/api/v3/archivos/recuperarArchivo/55

Params

Authorization ●

Headers (8)

Body

Pre-request Script

Tests

Settings

Type

Bearer Token ⌵

ⓘ Heads up! These param
[Learn more about variak](#)

Token


The authorization header will be automatically generated when you send the request. [Learn more about authorization](#) ➤

Body

Cookies

Headers (12)

Test Results



- /delete nos permite borrar la información del archivo y este último de nuestro servidor. Para ello mandamos la id del archivo.

clases / borrarArchivo


POST ⌵ http://localhost:8000/api/v3/archivos/delete/55

Params Authorization ● Headers (10) Body ● Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON ⌵ 

```
1
2  "message": "El archivo se elimino correctamente"
3
```

En /categorias contamos con acceso al crud de categorías al igual que archivos cuenta con un middleware que obliga al usuario a estar registrado y activo antes de poder acceder a esta situación.

- / o /index que nos proporciona todas las categorías que encuentras en la aplicación. Es de tipo de get.

classes / pedric categorias

GET ⌵ http://localhost:8000/api/v3/categorias/

Params Authorization ● Headers (9) Body Pre-request Script Tests Settings

Type Bearer Token ⌵

! Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend [Learn more about variables](#) ➤

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#) ➤

Token eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.e...

Body Cookies (1) Headers (10) Test Results ⊕ Status: 200 OK

Pretty Raw Preview Visualize JSON ⌵ ↺

```
2  "categorias": [  
3    {  
4      "id": 1,  
5      "name": "Documento Postman 1",  
6      "categoria": null,  
7      "user_id": 9,  
8      "deleted_at": null,  
9      "created_at": "2021-05-18T15:37:55.000000Z",  
10     "updated_at": "2021-05-18T15:37:55.000000Z"  
11   },  
12   {  
13     "id": 2,  
14     "name": "Radiografias",  
15     "categoria": 1,  
16     "user_id": 9,  
17     "deleted_at": null,  
18     "created_at": "2021-05-18T15:38:28.000000Z",  
19     "updated_at": "2021-05-18T15:38:28.000000Z"  
20   }  
21 ]
```

- /store almacenamos las categorías para ello pedimos nombre y categoría padre.

clases / Generar una categoria


POST ⌵ http://localhost:8000/api/v3/categorias/store...

Params Authorization ● Headers (11) **Body ●** Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **JSON** ⌵

```
1 {"name": "Radiografias", "categoria": "1"}
2
```

Body **Cookies (1)** Headers (10) Test Results

Pretty Raw Preview Visualize **JSON** ⌵ 

```
1 {
2   "name": "Radiografias",
3   "categoria": "1",
4   "user_id": 9,
5   "updated_at": "2021-05-18T15:42:49.000000Z",
6   "created_at": "2021-05-18T15:42:49.000000Z",
7   "id": 5
8 }
```

- /update nos permite actualizar la información de las categorías, para ello se le pide la id de la categoría, name y la categoría padre.

The screenshot shows a REST client interface. At the top, a POST request is configured to `http://localhost:8000/api/v3/categorias/update`. The 'Body' tab is selected, showing a JSON payload: `{"id":5,"name":"Documento update","categoria":1}`. Below the request, the 'Test Results' section shows the response body in 'Pretty' format. The response is a JSON object with the following fields: `"id": 5`, `"name": "Documento update"`, `"categoria": null`, `"user_id": 11`, `"deleted_at": null`, `"created_at": "2021-05-18T15:42:49.000000Z"`, and `"updated_at": "2021-05-25T14:48:15.000000Z"`.

```
POST http://localhost:8000/api/v3/categorias/update

{"id":5,"name":"Documento update","categoria":1}

Body Cookies Headers (10) Test Results
Pretty Raw Preview Visualize JSON
1 {
2   "id": 5,
3   "name": "Documento update",
4   "categoria": null,
5   "user_id": 11,
6   "deleted_at": null,
7   "created_at": "2021-05-18T15:42:49.000000Z",
8   "updated_at": "2021-05-25T14:48:15.000000Z"
9 }
```

- `/id` podemos pedir una categoría es de tipo get y nos devuelve la categoría elegida por id.

clases / perdir 1 categoria

GET

▼

http://localhost:8000/api/v3/categorias/2

Params

Authorization ●

Headers (9)

Body

Pre-request Script

Tests

Settings

☒

Cache-Control ⓘ

no-cache

☒

Postman-Token ⓘ

<calculated when request is sent>

☒

Host ⓘ

<calculated when request is sent>

☒

User-Agent ⓘ

PostmanRuntime/7.28.0

☒

Accept ⓘ

/

☒

Accept-Encoding ⓘ

gzip, deflate, br

☒

Connection ⓘ

keep-alive

Key

Value

Body

Cookies (1)

Headers (10)

Test Results

Pretty

Raw

Preview

Visualize

JSON ▼

1

2

3

4

5

6

7

8

9

```
1 {
2   "id": 2,
3   "name": "Radiografias",
4   "categoria": 1,
5   "user_id": 9,
6   "deleted_at": null,
7   "created_at": "2021-05-18T15:38:28.000000Z",
8   "updated_at": "2021-05-18T15:38:28.000000Z"
9 }
```