

UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ

Aplicaciones Web Interactivas

FACULTAD DE INGENIERÍA

ÁREA EN CIENCIAS DE LA COMPUTACIÓN

Manual de programador: Biblioteca

Galván Ojeda Montserrat Guadalupe

GRUPO: 13:00-14:00

FECHA DE ENTREGA: Diciembre del 2022

Introducción:

Este proyecto resuelve la problemática de tener un mal manejo de la biblioteca escolar en una escuela primaria donde mi mamá es maestra, esto debido a que siempre se ha optado por realizar dicho manejo con registros físicos en un libro de registros, lo que hace mas complicado el seguimiento de préstamos vencidos, registros erróneos o perdida de información que puede causar la perdida de estos libros.

Con este proyecto se espera ayudar a tener un mejor control en esta biblioteca y realizar estos registros de una manera automática para una mayor confiabilidad y rapidez.

Desarrollo:

Me enfoque en realizar un diseño llamativo y que pudiera ser atractivo siguiendo la estética de las escuelas primarias. Para la programación utilice el framework Laravel combinado con Bootstrap 5 para el diseño.

Para su instalación es necesario crear la carpeta vendor colocando desde la terminal los comandos para instalación de los paquetes de Laravel y como extra la instrucción composer require barryvdh/laravel-dompdf para tener la biblioteca de manejo de pdfs.

Se espera que los administradores sean los únicos que pueden realizar acciones significativas en el sistema, por lo que las cuentas de administrador son previamente definidas y seria con el objetivo que solo el profesor encargado de la biblioteca escolar pueda realizar estas modificaciones y brindar los debidos préstamos.

Los alumnos entraran como usuarios normales, donde podrán ver la disponibilidad de los libros, así como una breve descripción de cada libro que se encuentre en la biblioteca.

- **Modelos:**

Inicialmente declare los modelos para tener una idea de cómo estaría estructurada la base de datos.

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class alumnos extends Model
{
    use HasFactory;

    protected $table = 'alumnos';
    protected $fillable = [
        'Nombres',
        'ApellidosPat',
        'ApellidosMat',
        'Edad',
        'Grado',
        'Grupo'
    ];
}
```

Inicie con el modelo de alumnos y profesores que serian los usuarios que podrían solicitar el préstamo de los libros, en el caso de los alumnos se registraría su Nombre, Apellido Paterno y Materno, Edad, Grado y Grupo. Los profesores solo tendrían los primeros tres datos, ya que al ser menos seria más fácil identificarlos. Estos datos se guardarían principalmente para dos acciones, identificar las personas que solicitan los libros y poder contactarlas después de que su préstamo se agote.

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class profesores extends Model
{
    use HasFactory;

    protected $table = 'profesores';
    protected $fillable = [
        'Nombres',
        'ApellidosPat',
        'ApellidosMat'
    ];
}
```

Pasando con el modelo de los libros, el cual seria el principal en el sistema. Se solicita el guardado de el Titulo, el name que tendrá el nombre de la imagen de portada del libro, la cual será almacenada en la carpeta pública del storage, dirección que se indicará en la variable Path, continuando con los datos se solicitará Autor, Editorial, Categoría y Disponibilidad, esta última solo podrá ser 1 que representa que el libro esta en biblioteca o 2 que representa que el libro está actualmente en préstamo.

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class libros extends Model
{
    use SoftDeletes;
    use HasFactory;

    protected $table = 'libros';
    protected $fillable = [
        'Titulo',
        'name',
        'Path',
        'Autor',
        'Editorial',
        'Categoría',
        'Disponibilidad'
    ];
}
```

```
use HasFactory;

public function nombreAlumno()
{
    $alumno = Alumnos::find($this->ClaveUsuarioA);
    return $alumno->Nombres;
}

public function nombreProfesor()
{
    $profesor = Profesores::find($this->ClaveUsuarioP);
    return $profesor->Nombres;
}

public function nombreLibro()
{
    $libro = Libros::find($this->ClaveLib);
    return $libro->Titulo;
}

protected $table = 'prestamos';
protected $fillable = [
    'ClaveUsuario',
    'ClaveLib',
    'FechaInicio',
    'FechaFinal',
    'NomRenov'
];
}
```

Para los prestamos se utilizará una clave de usuario que podrá ser tanto el id de alumno como el id de profesor, seguido del id del libro solicitado, la fecha de inicio del préstamo, así como la fecha donde se espera que sea regresado el libro, por ultimo se guarda el numero de renovaciones que se han solicitado.

En este mismo archivo declare las funciones para poder acceder a las tablas que se conectan con las llaves foráneas antes mencionadas y así poder obtener el nombre del alumno, del profesor y del libro.

Por ultimo el modelo de devoluciones tiene el id de los prestamos para conectar los pedidos, el estado del préstamo, donde 1 representa activo y 2 representa entregado, la fecha en que se realizó la devolución y una descripción del estado en que se entregó el libro. Al igual que el modelo anterior se declararon las funciones para acceder a los nombres de los alumnos, los profesores y el libro, los cuales acceden por medio de la clave del préstamo a las funciones del modelo anterior.

```

use HasFactory;

public function nombreAlumnoD()
{
    $alumno = Prestamos::find($this->ClaveDev);
    return $alumno->ClaveUsuarioA;
}

public function nombreProfesorD()
{
    $profesor = Prestamos::find($this->ClaveDev);
    return $profesor->ClaveUsuarioF;
}

public function nombreLibroD()
{
    $libro = Prestamos::find($this->ClaveDev);
    return $libro->nombreLibro();
}

protected $table = 'devoluciones';
protected $fillable = [
    'ClaveDev',
    'EstadoPrestamo',
    'Fecha',
    'EstadoLibro'
];

```

• Migraciones:

Siguiendo la estructura de los modelos cree las migraciones en el mismo nombre y siguiendo las mismas variables, pero con más especificaciones.

```

public function up()
{
    Schema::create('alumnos',
    function (Blueprint $table) {
        $table->id();
        $table->string('Nombres');
        $table->string('ApellidoPat');
        $table->string('ApellidoMat');
        $table->integer('Edad');
        $table->integer('Grado');
        $table->string('Grupo');
        $table->timestamps();
    });
}

```

Para los alumnos primero definí su id que sería consecutivo cada que se cree un objeto de este tipo, sin embargo, el sistema no podrá realizar la acción de registrar debido a que la escuela ya cuenta previamente con un registro de todos los alumnos inscritos, por lo que se optó por ingresar esos datos en un archivo externo que contuviera todos los campos requeridos y a partir de eso importarlos en el sistema. De esta manera sería más sencillo el registro de estos

usuarios.

Tanto el nombre, Apellido paterno y materno serán strings, seguido de la edad y grado que serían enteros, el grupo sería un solo carácter.

Para los profesores sería algo muy parecido, ya que los datos se guardarían en el sistema a través de una exportación de Excel con los datos. Donde todos los datos (Nombre, Apellido Paterno y Apellido Materno) serían strings.

```

public function up()
{
    Schema::create('profesores',
    function (Blueprint $table) {
        $table->id();
        $table->string('Nombres');
        $table->string('ApellidoPat');
        $table->string('ApellidoMat');
        $table->timestamps();
    });
}

```

```

public function up()
{
    Schema::create('libros',
    function (Blueprint $table) {
        $table->id();
        $table->string('Titulo');
        $table->string('name');
        $table->string('path');
        $table->softDeletes();
        $table->string('Autor');
        $table->string('Editorial');
        $table->string('Categoria');
        $table->integer('Disponibilidad');
        $table->timestamps();
    });
}

```

Los libros sí podrán ser registrados desde el sistema por lo que su id sería consecutivo, el título, la variable que tendrá el nombre de la imagen y su dirección serían strings. Se declara un softDeletes para permitir la incorporación de imágenes en la base de datos, seguido del Autor, Editorial y Categoría que serían strings y por último la disponibilidad que sería un entero entre 1 y 2.

Para los préstamos se hacen los primeros usos de llaves foranes que apuntan a los id de las tablas alumnos y profesores respectivamente, los cuales serán almacenados como Clave de Usuarios y permitirán hacer el registro de la persona que solicita el libro, con la misma lógica se usa una llave foránea para la tabla libros que representara el libro que se está solicitado.

Se continuará con el guardado de la fecha en que se realizo el pedido, el cual se generará de manera automática, así como la fecha final que será un mes después de la solicitud. También iniciará el contador de las veces que se solicita la renovación de la fecha de regreso, esta solo permitirá aumentar un mes más el préstamo y se tiene un límite de 3 por persona.

```
public function up()
{
    Schema::create('prestamos',
        function (Blueprint $table) {
            $table->id();
            $table->foreignId(
                'ClaveUsuarioA')->nullable()
                ->constrained('alumnos');
            $table->foreignId(
                'ClaveUsuarioP')->nullable()
                ->constrained('profesores');
            $table->foreignId(
                'ClaveLib')->constrained(
                'libros');
            $table->date('FechaInicio');
            $table->date('FechaFinal');
            $table->integer('NumRenov');

            $table->timestamps();
        });
}
```

```
public function up()
{
    Schema::create('devoluciones',
        function (Blueprint $table) {
            $table->id();
            $table->foreignId(
                'ClaveDev')->constrained(
                'prestamos');
            $table->integer(
                'EstadoPrestamo');
            $table->date('Fecha')
                ->nullable();
            $table->string('EstadoLibro')
                ->nullable();
            $table->timestamps();
        });
}
```

Por último, se realiza la migración de las devoluciones que contendrá la llave foránea que conecta al id de la tabla préstamos, esto con el objetivo de tener los datos contenidos por cada pedido, seguido de un entero que representa el estado actual del préstamo, si se registra un valor de 1 significa el préstamo esta activo, de lo contrario el libro ya ha sido regresado y por ende ya se tiene una fecha y una descripción de como se regreso el libro. De lo contrario todos estos datos permanecerán en null. Cada que se crea un objeto de préstamo, se crea su registro de devoluciones que colocara automáticamente el estado y los demás campos se colocaran con null hasta que se registre la devolución correspondiente.

En todos se agregan los timestamps para registrar cuando se modifican o crea cada uno de los registros.

• Seeders:

Para crear las pruebas del código genere seeders por cada tabla, sin embargo, para la versión final solo utilice usuarios (alumnos y profesores) y libros previamente registrados con el objetivo de evitar los errores en los registros de los elementos que utilizan tablas foráneas.

Así como uno para definir los roles que tendrá el sistema y colocar las credenciales que permitirían el acceso a cada uno de estos roles.

```
public function run()
{
    $role2 = Role::create(['name' =>
        'usuario']);
    $role3 = Role::create(['name' =>
        'Admin']);

    DB::table('users')->insert([
        ['name' => 'admin',
            'email' => 'admin@gmail.com',
            'password' => Hash::make(
                'admin'),
        ]);

    $user = User::find(1);
    $user->assignRole('Admin');

    DB::table('users')->insert([
        ['name' => '123',
            'email' => '123@gmail.com',
            'password' => Hash::make(
                '123'),
        ]);

    $user = User::find(1);
    $user->assignRole('usuario');
}
```

• Base de Datos:

The following table represents the data shown in the screenshots, organized by the 'Id' column.

Id	Nombre	Tipo de libro	Cantidad	Estado	Fecha de compra	Autor	Género	Editorial	Precio	Disponible
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

• Vistas:

Primero se tiene la vista pagina.Blade que sería la principal, ya que presenta los datos de la escuela y las redes sociales oficiales para contacto, así como los diferentes servicios que podría ofrecer la escuela dentro de la pagina si se sigue trabajando en este proyecto, los cuales permitirían modificar las actividades y horarios, pero el que actualmente funciona es solo el de Biblioteca.

El botón de biblioteca, así como el de iniciar sesión en la navbar conducirán a el login propio de laravel, aquí el usuario deberá ingresar sus credenciales y el sistema le asignará un rol para ser enviado a la vista de la biblioteca.

Dentro de la biblioteca cualquier tipo de usuario podrá ver todos los libros registrados en la base de datos, cada uno con su título, autor e imagen de portada. Si el nombre del libro se encuentra en rojo significa que el libro actualmente está en préstamo, de lo contrario se encuentra disponible y esto es tomado del campo disponibilidad de la tabla libros.

Si se entró como administrador podrá visualizar mas opciones, en la navbar se mostrarán otras opciones que llevan a los datos de otras tablas. Así como un botón para agregar un nuevo libro y otro que generara un pdf con los datos de la tabla libros. Dentro de cada card que muestra los libros se tiene la opción de editar la información de un libro y eliminar el registro de un libro.

Al oprimir en editar se dirigirá a la vista de form el cual tiene un formulario que permite registrar cada campo de la tabla libros a excepción de la disponibilidad que se colocara automáticamente como 1, que significa que el libro está actualmente disponible. Una vez ingresados todos los datos y la imagen de portada se guardará el registro en la bd y se regresará la vista a la biblioteca donde aparecerá automáticamente el nuevo registro.

El diseño esta especificado para que cada imagen ingresada se adapte al tamaño general de la card, esto con la finalidad de que cada una tenga un diseño similar.

Si se presiona en el botón de generar pdf, se enviara a la vista download que contiene el diseño de lo que se imprimirá en el archivo pdf que internamente generara dicho archivo y regresara a la vista de la biblioteca.

Si se desea editar un registro se presiona en el botón y se dirigirá a la vista edicionB, donde se mostrará un formulario con los datos que actualmente se tienen registrados, para modificarlos solo se edita el deseado y los otros se dejan como están, es importante considerar que, al igual que agregar uno nuevo, el registro de disponibilidad no puede ser alterado. Una vez que tenemos los datos nuevos presionamos en editar y se nos regresara a la vista de la biblioteca donde se mostraran los nuevos datos.

Por último, en esta vista, se podrá eliminar, e internamente solo se enviará al controlador, el cual regresará a la vista de la biblioteca cuando haya completado la función indicada, por lo que en la vista no se llega a percibir.

Como siguiente página, se puede entrara a ver los actuales registros de préstamo desde la navbar en la opción prestamos, esto nos dirigirá a la vista prestamos.blade donde se tendrán opciones muy parecidas a la página anterior, se tendrá un botón de registrar, el cual tiene la función de agregar el préstamo nuevo, presionándolo podemos dirigirnos a la vista formP donde se nos presentara un formulario de opciones, donde podemos elegir de la lista desplegable quien de los alumnos o profesores actualmente registrados en el sistema son los que realizaran un prestamos, así como el libro, de los disponibles actualmente, que solicitan llevarse. Cuando presionamos en guardar se nos regresará a la vista de prestamos con el primer registro, el cual contendrá también la fecha actual, que es la de inicio de prestamos y la de devolución esperada que será un mes después de la actual con un contador de renovación a 0. Internamente se inicializará un registro de la tabla devolución, donde se pasara el id del préstamo generado y los demás campos se establecerán en null.

Las renovaciones se podrán realizar solo el día que se vaya a vencer el préstamo, donde aparecerá un botón dentro del registro que, si se presiona, alargara la fecha limite de préstamo con un mes más, esto solo puede ser realizado tres veces por cada registro, por lo que el tiempo máximo que tendrá una persona para alargar su préstamo será de 4 meses en total.

Al igual que la vista anterior se podrá generar el archivo pdf con los datos de cada préstamo registrado, incluso los que ya no estén activos.

Para el diseño de esta página solo se mostrarán los registros que tengan estado activo y así evitar que se acumulen muchos registros.

Por ultimo se tiene la vista de devoluciones la cual mostrara los registros de devolución que se generan automáticamente cuando se realiza un préstamo, para que cada registro de préstamo sea marcado como entregado es necesario oprimir el botón de devolución en el campo que tenga los datos deseados. Esto nos mandara a la vista edicionD donde se mostrará un formulario donde podremos ingresar las condiciones en las que se regresó el libro e internamente se registrara la fecha y se modificara el campo de disponibilidad del libro en cuestión. El botón que contiene una grafica envía a la vista chart que muestra una grafica de pastel generada por GoogleChart.

Todos los diseños de las paginas son responsivos y siguen el mismo modelo de la principal que es la biblioteca a excepción de los formularios de edición o creación que cambian un poco el estilo, pero respetando en gran parte la estructura de las otras páginas.

• Controlador:

El controlador contiene todas las funciones que se utilizan en las vistas, tomando en cuenta las rutas definidas en web.php, cada una de estas funciones permite controlar los movimientos en la base de datos, así como los cambios de vistas.

Funciones de la vista biblioteca:

```
public function biblioteca()
{
    $d = Libros::all();
    //dd($d);
    return view('biblioteca')
        ->with('libros', $d);
}

public function agregar(Request $request)
{
    $d = new Libros();

    $d -> Titulo = $request -> Titulo;

    $file = $request -> file('archivo');
    $d -> path = $request -> file('archivo') -> storeAs(
        'public', $file -> getClientOriginalName());
    $d -> name = $file -> getClientOriginalName();

    $d -> Autor = $request -> Autor;
    $d -> Editorial = $request -> Editorial;
    $d -> Categoria = $request -> Categoria;
    $d -> Disponibilidad = '1';
    $d -> save();

    return redirect("/biblioteca");
}

public function formulario()
{
    return view('form');
}
```

```
public function eliminar2(Request $request)
{
    $d = Libros::find($request -> libro) -> delete();

    return redirect("/biblioteca");
}

public function muestraedicion($id)
{
    $d = Libros::find($id);

    return view('edicionB') -> with('libro', $d);
}

public function update(Request $request)
{
    $d = Libros::find($request -> id);

    $d -> Titulo = $request -> Titulo;
    $d -> Autor = $request -> Autor;
    $d -> Editorial = $request -> Editorial;
    $d -> Categoria = $request -> Categoria;
    $d -> save();

    return redirect("/biblioteca");
}
```

Funciones de la vista prestamos:

```
public function prestamos()
{
    $d = Prestamos::all();
    $a = Alumnos::all();
    $p = Profesores::all();

    return view('prestamos')
        ->with('prestamos', $d)
        ->with('alumnos', $a)
        ->with('profesores', $p);
}

public function downloadPDF()
{
    $d = Prestamos::all();

    view()->share('download', $d);

    $pdf = PDF::loadView('download', ['prestamos' =>
        $d]);

    return $pdf->download('Prestamos.pdf');
}
```

```
public function formularioP()
{
    $d = Libros::all();
    $a = Alumnos::all();
    $p = Profesores::all();

    return view('formP')
        ->with('libros', $d)
        ->with('alumnos', $a)
        ->with('profesores', $p);
}

public function agregarP(Request $request)
{
    $d = new Prestamos();
    $v = new Devoluciones();

    $d -> ClaveUsuarioA = $request -> ClaveUsuarioA;
    $d -> ClaveUsuarioP = $request -> ClaveUsuarioP;
    $d -> ClaveLib = $request -> ClaveLib;
    $l = Libros::find($d -> ClaveLib);
    $l -> Disponibilidad = '2';

    $d -> FechaInicio = $request -> FechaInicio;
    $d -> FechaFinal = $request -> FechaFinal;
    $d -> NomRenov = '0';
    $d -> save();
    $l -> save();
    $v -> ClaveDev = $d -> id;
    $v -> EstadoPrestamo = '0';
    $v -> save();

    return redirect("/prestamos");
}
```

```
public function muestrarenov($id)
{
    $d = Prestamos::find($id);

    $d -> FechaFinal = date('Y-m-d', strtotime('+ 1
        month'));
    $d -> NomRenov += '1';
    $d -> save();

    return redirect("/prestamos");
}
```

Funciones de la vista devoluciones:

```

public function devoluciones()
{
    $d = Devoluciones::all();
    $t = Prestamos::all();
    $a = Alumnos::all();
    $p = Profesores::all();

    return view('devoluciones')
        ->with('devoluciones', $d)
        ->with('prestamos', $t)
        ->with('alumnos', $a)
        ->with('profesores', $p);
}

public function muestradev($id)
{
    $d = Devoluciones::find($id);

    return view('edicionD') -> with('devolucion', $d);
}

```

```

public function updateD(Request $request)
{
    $d = Devoluciones::find($request -> id);

    $d -> EstadoPrestamo = '1';
    $d -> Fecha = date('Y/m/d');
    $d -> EstadoLibro = $request -> EstadoLibro;

    $p = Prestamos::find($d -> ClaveDev);
    $l = Libros::find($p -> ClaveLib);
    $l -> Disponibilidad = '1';

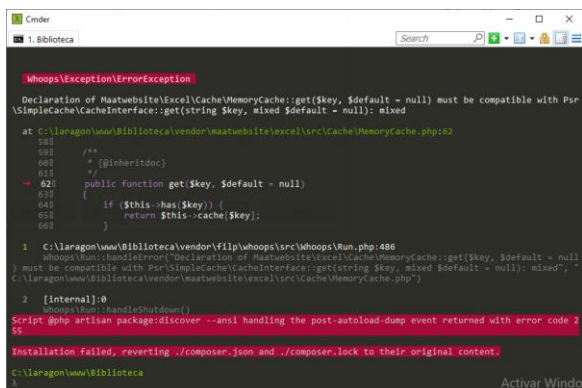
    $d -> save();
    $l -> save();

    return redirect("/devoluciones");
}

```

Conflictos:

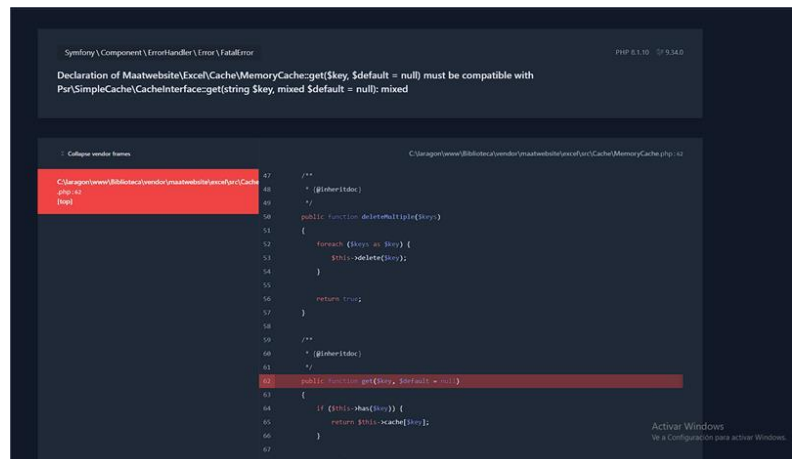
No pude implementar la exportación/importación en Excel debido a que al descargar la librería maatwebsite/Excel me generaba un error en los archivos generados dentro de la carpeta vendor, que me impedía usar cualquier modulo del proyecto e investigando no pude encontrar la solución a este problema.



```

C:\laragon\www\Biblioteca>
Whoops\Exception\ErrorException
Declaration of Maatwebsite\Excel\Cache\MemoryCache::get($key, $default = null) must be compatible with Psr\SimpleCache\CacheInterface::get(string $key, mixed $default = null): mixed
at C:\laragon\www\Biblioteca\vendor\maatwebsite\excel\src\Cache\MemoryCache.php:62
60:         /**
61:          * @inherited
62:          */
63:         public function get($key, $default = null)
64:         {
65:             if ($this->has($key)) {
66:                 return $this->cache[$key];
67:             }
68:         }
69:     }
70: }
1 C:\laragon\www\Biblioteca\vendor\filp\whoops\src\Whoops\Run.php:486
Whoops\Run::handleError("Declaration of Maatwebsite\Excel\Cache\MemoryCache::get($key, $default = null) must be compatible with Psr\SimpleCache\CacheInterface::get(string $key, mixed $default = null): mixed", "C:\laragon\www\Biblioteca\vendor\maatwebsite\excel\src\Cache\MemoryCache.php")
2 [Internal] @
Whoops\Run::handleShutdown()
Script @php artisan package:discover --ansi handling the post-autoload-dump event returned with error code 255
Installation failed, reverting ./composer.json and ./composer.lock to their original content
C:\laragon\www\Biblioteca>

```



```

Symfony\Component\ErrorHandler\Error\FatalError
PHP 8.1.10 - 17.9.34.0

Declaration of Maatwebsite\Excel\Cache\MemoryCache::get($key, $default = null) must be compatible with
Psr\SimpleCache\CacheInterface::get(string $key, mixed $default = null): mixed

C:\laragon\www\Biblioteca\vendor\maatwebsite\excel\src\Cache\MemoryCache.php:62
62:         /**
63:          * @inherited
64:          */
65:         public function get($key, $default = null)
66:         {
67:             if ($this->has($key)) {
68:                 return $this->cache[$key];
69:             }
70:         }
71:     }
72: }

```