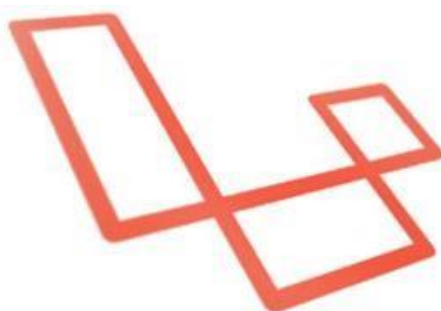


Desarrollo de Aplicaciones Empresariales

LABORATORIO N°

Laravel – Definición de Proyecto y Creación de Modelos

CODIGO DEL CURSO:



laravel

<i>Alumno(s)</i>		<i>Nota</i>
Victor Alejandro Calvo Guzmán		
<i>Grupo</i>	<i>C-15-A</i>	
<i>Ciclo</i>	<i>III</i>	
<i>Fecha de entrega</i>		

	Desarrollo de Aplicaciones Empresariales – Laravel	Nro. DD-106 Página 1 de 10
---	---	-------------------------------

I.- OBJETIVOS:

II.- SEGURIDAD:



Advertencia:

En este laboratorio está prohibida la manipulación del hardware, conexiones eléctricas o de red; así como la ingestión de alimentos o bebidas.

III.- FUNDAMENTO TEÓRICO:

Revise sus diapositivas del tema antes del desarrollo del laboratorio.

IV.- NORMAS EMPLEADAS:

No aplica V.-

RECURSOS:

- En este laboratorio cada alumno trabajará con un equipo con Windows 8.

VI.- METODOLOGÍA PARA EL DESARROLLO DE LA TAREA:

- El desarrollo del laboratorio es individual.

VII.- PROCEDIMIENTO:

Nota:

Las secciones en cursivas son demostrativas, pero sirven para que usted pueda instalar las herramientas de desarrollo en un equipo externo.

CREANDO UN DOMINIO LOCAL

1. Ingresamos a **C:\Windows\System32\drivers\etc** y abrimos el archivo **hosts** con el **block de notas**.
2. Agregamos un nombre de dominio con resolución de nuestro localhost:
3. Probemos el link creado en nuestro browser, debería abrirse la página de xamp o wamp.

```
127.0.0.1      localhost
127.0.0.1      curso_symfony.dev
127.0.0.1      gestorlaravel.com|
```

```
# localhost name resolution is handled within
#      127.0.0.1      localhost
#      ::1           localhost|
#      127.0.0.1      gestorlaravel.com
```

Sin embargo, la página a la que hemos sido redireccionados, es la página principal de xamp o wamp,

4. Vamos a modificar el archivo:
 - a. En WAMPSERVER: **C:\wamp64\bin\apache\apache2.4.17\conf\extra\httpd-vhosts.conf**
 - b. En XAMP: **C:\xamp\apache\conf\extra\httpd-vhosts.conf**

```
<VirtualHost *:80>
    DocumentRoot "C:/xampp/htdocs/GestorImagenes/public"
    ServerName gestorlaravel.com
</VirtualHost>
```

```
<VirtualHost *:80>
    DocumentRoot "C:/xampp/htdocs/GestorImagenes/public"
    ServerName gestorlaravel.com
</VirtualHost>
```

CREANDO LA BASE DE DATOS PARA EL PROYECTO

En esta sección vamos a crear la base de datos. Para ello deberíamos tener instalado MySQL.

1. Ingresar a la página de **PhpMyAdmin**.
2. Crear una nueva base de datos de nombre: gestorimagenes.



Crear base de datos ⓘ

gestorimagenes Cotejamiento ▼ Crear

3. Las tablas serán creadas de manera análoga a Symfony2.

CONFIGURANDO EL ENTORNO Y EL PROYECTO DE LARAVEL

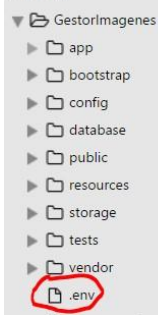
1. Abra **Sublime Text**.
2. Elija la opción **File->OpenFolder** y abra la carpeta del proyecto creado con Composer.
3. Abra la ventana de comandos de Windows y ubíquese en la carpeta del proyecto.
4. Escriba **"php artisan list"**. Visualizará la lista de comandos disponibles que están disponibles en **artisan**
5. Configurar el nombre de la aplicación:

- a. Escriba **"php artisan app:name GestorImagenes"**.

```
C:\wamp64\www\GestorImagenes>php artisan app:name GestorImagenes
Application namespace set!
```

- b. Para corroborar el cambio abra el archivo **app/Http/Controllers/Auth/HomeController.php**, Notará que el nuevo espacio de nombres es el asignado.

FOLDERS



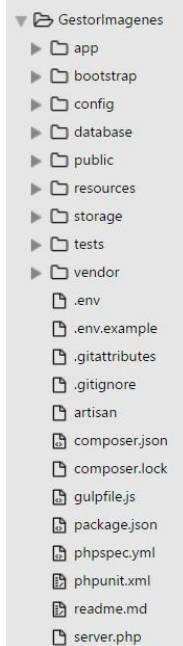
```
1 <?php namespace GestorImagenes\Http\Controllers;
2
3 class HomeController extends Controller {
4
```

- a. Configure el código para que quede de la siguiente manera:

```
w Goto Tools Project Preferences Help
.env
1 APP_ENV=local
2 APP_DEBUG=true
3 APP_KEY=MnDsK4lj0X5xPCqHyYnFDZtQiLYkan1z
4
5 DB_HOST=localhost
6 DB_DATABASE=gestorimagenes
7 DB_USERNAME=root
8 DB_PASSWORD=root
9
10 CACHE_DRIVER=file
11 SESSION_DRIVER=file
12
```

6. Ubique el archivo **".env"**.

FOLDERS



DEFINIENDO EL PROYECTO

El proyecto que crearemos en este curso será un gestor de imágenes y álbumes que un usuario registrado podrá realizar.

Algunos casos de Uso:

- El usuario podrá crear, modificar y eliminar fotografías.
- El usuario podrá crear, modificar y eliminar álbumes.
- Un álbum tiene varias fotografías.
- Una fotografía pertenece a un solo álbum.
- Un usuario tiene varios álbumes.
- La fotografía tiene un identificador, nombre, descripción, una ruta en el servidor, y el álbum al que pertenece.
- El álbum tiene un identificador, nombre, descripción y el usuario al que pertenece.
- El usuario tiene un identificador, nombre, **email y password**. **Email y password**: Estos campos, laravel los considera por defecto, es decir que tiene una programación ya dada incluso para enviar un correo y confirmar a un usuario.

CREANDO LOS MODELOS PARA EL PROYECTO

1. Ubique el archivo **User.php** que se encuentra dentro de la carpeta **app**.

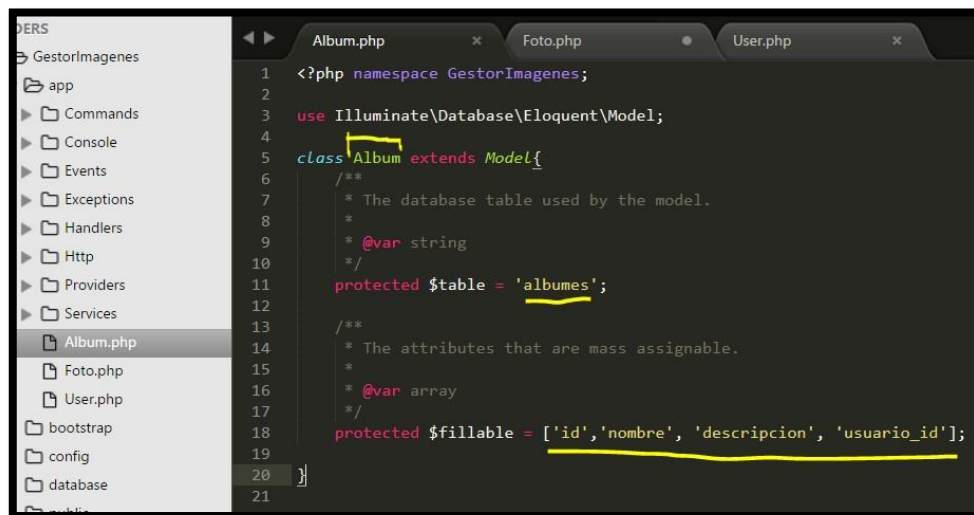

```
.env User.php
1 <?php namespace GestorImagenes;
2
3 use Illuminate\Auth\Authenticatable;
4 use Illuminate\Database\Eloquent\Model;
5 use Illuminate\Auth\Passwords\CanResetPassword;
6 use Illuminate\Contracts\Auth\Authenticatable as AuthenticatableContract;
7 use Illuminate\Contracts\Auth\CanResetPassword as CanResetPasswordContract;
8
9 class User extends Model implements AuthenticatableContract, CanResetPasswordContract {
10
11     use Authenticatable, CanResetPassword;
12
13     /**
14      * The database table used by the model.
15      *
16      * @var string
17      */
18     protected $table = 'users';
19
20     /**
21      * The attributes that are mass assignable.
22      *
23      * @var array
24      */
25     protected $fillable = ['name', 'email', 'password'];
26
27     /**
28      * The attributes excluded from the model's JSON form.
29      *
30      * @var array
31      */
32     protected $hidden = ['password', 'remember_token'];
33 }
34
35
```

2. Renombre el archivo por: **Usuario.php**.
3. En la línea 9, modifique el nombre de la clase por **Usuario**.
4. En la línea 18, modifique el nombre de la tabla por **usuarios**.
5. En la línea 25, modifique los campos que el usuario tendrá, para que quede de la siguiente manera:

```
    * @var array
    */
    protected $fillable = ['id', 'nombre', 'email', 'password'];
    /**
```

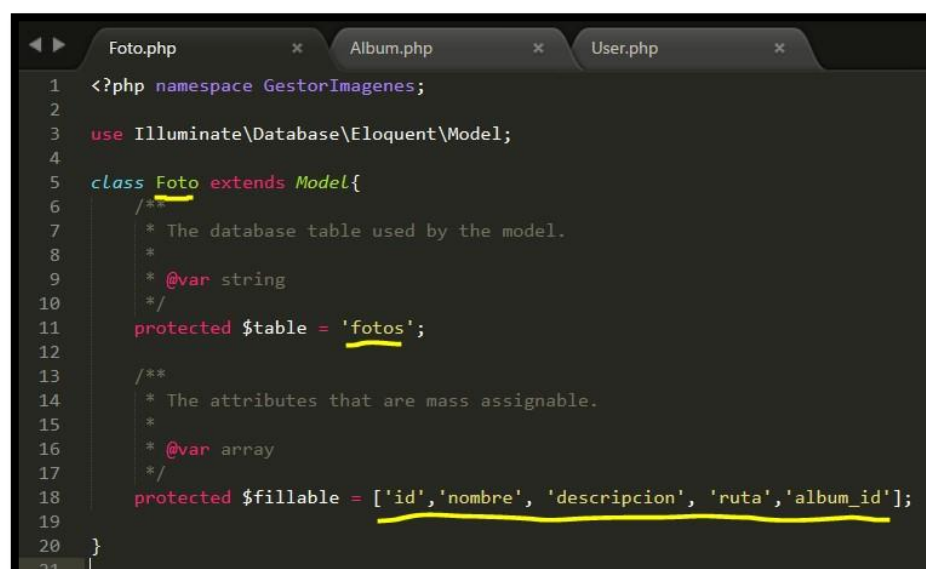
Crearemos el siguiente modelo.

6. Cree un nuevo archivo de nombre **Album.php** en la carpeta en la que se encuentra **Usuario.php**.
7. Copie el código de usuario.php, y modifíquelo para que quede de la siguiente manera:



```
1 <?php namespace GestorImágenes;
2
3 use Illuminate\Database\Eloquent\Model;
4
5 class Album extends Model{
6     /**
7      * The database table used by the model.
8      *
9      * @var string
10     */
11     protected $table = 'albumes';
12
13     /**
14      * The attributes that are mass assignable.
15      *
16      * @var array
17      */
18     protected $fillable = ['id', 'nombre', 'descripcion', 'usuario_id'];
19
20 }
21
```

8. Repita los pasos 6 y 7, solo que ahora para crear el modelo a la entidad **Foto.php**. El código debería quedar de la siguiente manera:



```
1 <?php namespace GestorImágenes;
2
3 use Illuminate\Database\Eloquent\Model;
4
5 class Foto extends Model{
6     /**
7      * The database table used by the model.
8      *
9      * @var string
10     */
11     protected $table = 'fotos';
12
13     /**
14      * The attributes that are mass assignable.
15      *
16      * @var array
17      */
18     protected $fillable = ['id', 'nombre', 'descripcion', 'ruta', 'album_id'];
19
20 }
21
```

DEFINIENDO RELACIONES DE CARDINALIDAD ENTRE MODELOS

1. Crearemos la relación de **ÁLBUM -> FOTO**: Un álbum tiene **varias fotos**.
2. Crearemos la relación de **ÁLBUM -> USUARIO**: Un álbum pertenece a un **único usuario**.
3. Modificamos el modelo álbum para que quede de la siguiente manera:



```
18     protected $fillable = ['id', 'nombre', 'descripcion', 'usuario_id'];
19
20     public function fotos(){
21         return $this->hasMany('GestorImágenes\Foto');
22     }
23     public function propietario(){
24         return $this->belongsTo('GestorImágenes\Usuario');
25     }
26 }
27
```

4. Crearemos la relación de **FOTO -> ÁLBUM**: Una foto pertenece a un **álbum**.
5. Modificaremos el modelo Foto para que quede de la siguiente manera:


```

18     protected $fillable = ['id', 'nombre', 'descripcion', 'ruta', 'album_id'];
19
20     public function album(){
21         return $this->belongsTo('GestorImagenes\Album');
22     }
23 }
24

```

6. Crearemos la relación USUARIO -> ALBUM: Un usuario posee varios álbumes.
7. Modificaremos el modelo Usuario para que quede de la siguiente manera:

```

32     protected $hidden = ['password', 'remember_token'];
33
34     public function albums(){
35         return $this->hasMany('GestorImagenes\Album');
36     }
37 }
38

```

A este punto, tenemos listos los modelos. Posteriormente realizaremos la migración a la base de datos.

EJERCICIO:

- Modifique el modelo USUARIO para que ahora, adicionalmente tenga 2 campos “pregunta” “respuesta”, estos campos servirán para una posible recuperación de contraseña del mismo.

```

/*
protected $fillable = ['id', 'nombre', 'email', '
password', 'pregunta', 'respuesta'];

```

CREANDO LOS CONTROLADORES

1. Ubique la carpeta **app** -> **http** -> **controllers**.
2. Dentro de ella encontrará una carpeta que agrupa a los controladores de la autenticación: **Auth**.
3. Renombre la carpeta “Auth” por “validacion”.
4. Dentro de la carpeta “validacion”, tendremos 2 archivos, elimine el archivo “Password”. Borraremos el archivo porque nosotros haremos la recuperación de contraseña por medio de la pregunta y respuesta secreta.
5. En la carpeta “validacion”, queda un archivo “AuthController.php”. Renómbrelo a “ValidacionController.php”.
6. Este archivo se encargará de controlar funciones como iniciar sesión, crear cuentas, validar al usuario que va a iniciar sesión, etc.
7. Modifique el namespace de **ValidacionController.php**. la última ruta debería ser: **Validacion** (en lugar de “Auth”, ya que modificamos el nombre de la carpeta)

```

ValidationController.php
1 <?php namespace GestorImagenes\Http\Controllers\Validacion;
2
3 use GestorImagenes\Http\Controllers\Controller;
4 use Illuminate\Contracts\Auth\Guard;
5 use Illuminate\Contracts\Auth\Registrar;
6 use Illuminate\Foundation\Auth\AuthenticatesAndRegistersUsers;
7
8 class AuthController extends Controller {
9
10     /*
11

```

```

Usuario.php  ValidationController.php x  Album.php x  php x
<?php namespace GestorImagenes\Http\Controllers\
Validacion;

```

Si usamos la combinación de teclas **Alt + clic** Sobre “AuthenticatesAndRegistersUsers”, iremos a la definición del Trait.


```

1 <?php namespace GestorImagenes\Http\Controllers\Auth;
2
3 use GestorImagenes\Http\Controllers\Controller;
4 use Illuminate\Contracts\Auth\Guard;
5 use Illuminate\Contracts\Auth\Registrar;
6 use Illuminate\Foundation\Auth\AuthenticatesAndRegistersUsers;
7
8 class AuthController extends Controller {
9
10     /**
11      * Registration & Login Controller
12      *
13      * This controller handles the registration of new users, as well as the
14      * authentication of existing users. By default, this controller uses
15      * a simple trait to add these behaviors. Why don't you explore it?
16      */
17
18     /**
19      * Create a new authentication controller instance.
20      *
21      * @param \Illuminate\Contracts\Auth\Guard $auth
22      * @param \Illuminate\Contracts\Auth\Registrar $registrar
23      * @return void
24      */
25     public function __construct(Guard $auth, Registrar $registrar)
26     {
27         $this->auth = $auth;
28         $this->registrar = $registrar;
29
30         $this->middleware('guest', ['except' => 'getLogout']);
31     }
32 }
33
34
35
36
37
38
39

```

EJERCICIO

Investigue y averigüe: ¿Cuál es la utilidad, características y funcionalidad de un TRAIT en PHP?

Son un mecanismo de reutilización de código en lenguajes de herencia simple, como PHP. El objetivo de un rasgo es el de reducir las limitaciones propias de la herencia simple permitiendo que los desarrolladores reutilicen a voluntad conjuntos de métodos sobre varias clases independientes y pertenecientes a clases jerárquicas distintas. La semántica a la hora combinar Traits y clases se define de tal manera que reduzca su complejidad y se eviten los problemas típicos asociados a la herencia múltiple y a los Mixins.

Un Trait es similar a una clase, pero con el único objetivo de agrupar funcionalidades muy específicas y de una manera coherente. No se puede instanciar directamente un Trait. Es por tanto un añadido a la herencia tradicional, y habilita la composición horizontal de comportamientos; es decir, permite combinar miembros de clases sin tener que usar herencia.

8. Lo que tiene la definición del Trait es la obtención del registro de un usuario, login del mismo, etc., Aunque todas las funciones ya están declaradas en inglés, algunas no son lo suficientemente óptimas y adecuadas para la aplicación que deseamos desarrollar. Entonces lo que haremos será crear una copia del código del trait, de modo que irá incluido en el controlador de validación y obviaremos la sentencia "USE", para omitir el uso del archivo.
9. **Copie** el código de la declaración del **Trait** y **péguelo** en el controlador de **validación**, (Todo el código copiado irá en vez del "use" que se encuentra en la **fila 21**).
10. Para ordenar el código, ubique el **constructor** del **controlador** y **muévelo** inmediatamente después de la declaración de la variable protegida **\$registrar**.
El resultado debería tener cierta similitud respecto de la imagen a continuación.


```

24  * @var \Illuminate\Contracts\Auth\Guard
25  */
26  protected $auth;
27
28  /**
29   * The registrar implementation.
30   *
31   * @var \Illuminate\Contracts\Auth\Registrar
32   */
33  protected $registrar;
34
35  /**
36   * Create a new authentication controller instance.
37   *
38   * @param \Illuminate\Contracts\Auth\Guard $auth
39   * @param \Illuminate\Contracts\Auth\Registrar $registrar
40   * @return void
41   */
42  public function __construct(Guard $auth, Registrar $registrar)
43  {
44      $this->auth = $auth;
45      $this->registrar = $registrar;
46
47      $this->middleware('guest', ['except' => 'getLogout']);
48  }
49
50  /**
51   * Show the application registration form.
52   *
53   * @return \Illuminate\Http\Response
54   */
55  public function getRegister()
56  {
57      return view('auth.register');
58  }
59
60  /**
61   * Handle a registration request for the application.
62   *
63   * @param \Illuminate\Http\Request $request
64   * @return \Illuminate\Http\Response
65   */
66  public function postRegister(Request $request)
67  {
68      $validator = $this->registrar->validator($request->all());

```

11. Ubique la función “**getRegister**” y modifique el nombre a “**getRegistro**”.
12. Dentro de la misma función, **modifique** los valores retornados, y retorne el texto: “**formulario creación cuenta**”.

```

53  * @return \Illuminate\Http\Response
54  */
55  public function getRegistro()
56  {
57      return 'formulario creación cuenta';
58  }
59

```

13. Ubique la función “**postRegister**” y modifique el nombre a “**postRegistro**”.
14. Ubique la función “**getLogin**” y modifique el nombre a “**getInicio**”
15. Ubique la función “**postLogin**” y modifique el nombre a “**postInicio**”
16. Ubique la función “**getLogout**” y modifique el nombre a “**getSalida**”
17. Ubique la función “**loginPath**”. En la ruta, al final de la sentencia **return**, modifique “**/auth/Login**” por “**/validacion/inicio**”
18. Ubique la función **getInicio** y modifique el valor de retorno de la misma por el texto: “**mostrando formulario inicio sesión**”
19. Ubique la función **getFailedLoginMessage**, es la función que se ejecuta en caso de algún dato incorrecto al momento de realizar la validación. Modifique su valor de retorno por el texto: “**email o contraseña incorrectos.**”
20. Ubique la función **redirectPath**, modifique la ruta en el valor de retorno: **/home** por **/inicio**.
Estamos realizando cambios de lenguaje únicamente, más adelante realizaremos cambios funcionales.
21. Crearemos una función para recuperar la cuenta en caso el usuario no recuerde sus credenciales. Agregue una función al final con el siguiente código:


```

public function loginPath()
{
    return property_exists($this, 'loginPath') ? $this->loginPath : '/auth/login';
}
public function getRecuperar(){
    return 'recuperar contraseña';
}
public function postRecuperar(){
    return 'recuperando contraseña';
}

```

```

151     }
152     public function getRecuperar(){
153         return 'recuperar contraseña';
154     }
155     public function postRecuperar(){
156         return 'recuperando contraseña';
157     }

```

22. Ahora abra el archivo **HomeController.php**.
23. Modifique el nombre del archivo a: **"InicioController.php"**
24. Modifique el nombre de la clase de **"HomerController"** a **"InicioController"**
25. Ubique la función **Index**, modifique el nombre de la función por **"getIndex"**. Modifique su valor de retorno por el texto: **"página de inicio validado"**.

```

21     public function __construct()
22     {
23         $this->middleware('auth');
24     }
25
26     /**
27      * Show the application dashboard to the user.
28      *
29      * @return Response
30      */
31     public function getIndex()
32     {
33         return 'página de inicio validado';
34     }
35

```

26. Abra el archivo **WelcomeController.php**. //Esta es una página que se les muestra a los invitados, vale decir a los que aún no han iniciado sesión.
27. Modifique el nombre del archivo por: **"BienvenidaController.php"**
28. Así mismo, modifique el nombre de la clase **"WelcomeController"** por **"BienvenidaController"**
29. Ubique la función **Index**, modifique el nombre de la función por **"getIndex"**. Modifique su valor de retorno por el texto: **"bienvenida a la aplicación"**.

```

}
    *
    * @return Response
    */
    public function getIndex()
    {
        return 'página de inicio validado';
    }
}

```

```
5
6 ▼
7     /**
8      * Show the application welcome screen to the user.
9      *
10     * @return Response
11     */
12     public function getIndex()
13     {
14         return 'bienvenida a la aplicacion';
15     }
16 }
17
```


OBSERVACIONES

- Al primer paso, agregar datos a la carpeta del wamp en el blog de notas debemos tener en cuenta que primero vamos a hacer click derecho en la imagen del blog de notas a través de la búsqueda por Windows y seleccionar por administrador para que los cambios den efecto.
- El usuario que vaya a ingresar va a tener que colocar su id, nombre, email y password necesariamente y adicionalmente deberá colocar su pregunta con respuesta para la verificación de su contraseña.
- Al hacer pasar el mouse sobre un nombre resaltado como Authentica... podemos ver que nos resalta a una decisión de trait, la cual al hacer click sobre esta nos llevara hacia ella.
- El trait es un mecanismo de reutilización de código
- Debemos colocar siempre el cambio de la clase cuando renombramos a una de estas.

CONCLUSIONES

- Los álbumes como las fotos van a poseer id, nombre y descripción, así como los datos para llenar de cada una que los diferenciara.
- En la función fotos vamos a utilizar hasMany para llamarla
- En la función propietario vamos a utilizar belong to para llamarlo
- En álbum también vamos a utilizar estas dos herramientas mencionadas.
- Al cambiar de nombre al español a varias funciones tenemos que cambiar también a donde se van a realizar sus operaciones o si van a comunicarse a otro file también su nombre se debe de ser llamado en ambos.

