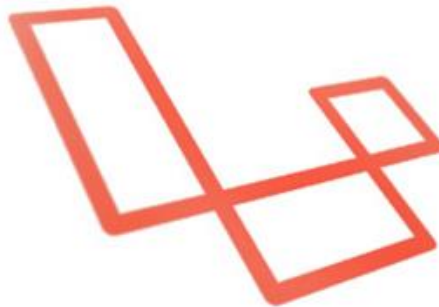


Desarrollo de Aplicaciones Empresariales

LABORATORIO N°

Laravel – Seeders, Migraciones, Excepciones y Validación de Usuario

CODIGO DEL CURSO:



laravel

Alumno(s)		Nota
Grupo		
Ciclo		
Fecha de entrega		

I.- OBJETIVOS:

- Crear Seeders en Laravel.
- Lograr la validación de Usuario.
- Habilitar la funcionalidad de recuperar contraseña por parte del usuario.

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 MIGRACIÓN DE MODELOS A LA BASE DE DATOS

Para migrar con éxito nuestros modelos a la base de datos, primero debemos crear archivos de migración.

1. Ingrese a la ventana de comandos con ruta en el directorio de nuestro proyecto.

```
C:\wamp64\www\GestorImagenes>
```

2. Ingrese el siguiente comando para crear el archivo "CrearTablaAlbumes"

```
php artisan make:migration --create="albumes" CrearTablaAlbumes
```

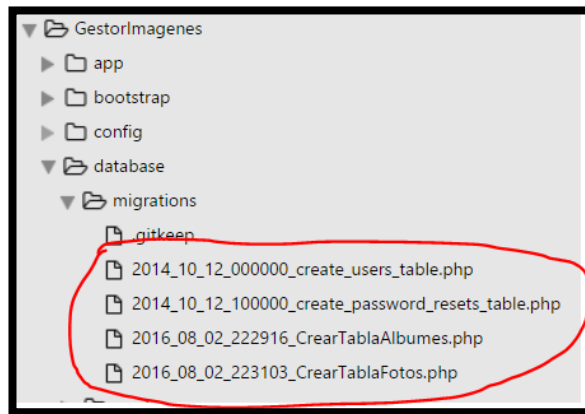
```
C:\wamp64\www\GestorImagenes>php artisan make:migration --create="albumes" CrearTablaAlbumes
Created Migration: 2016_08_02_222916_CrearTablaAlbumes
```

3. Ingrese el siguiente comando para crear el archivo "CrearTablaFotos"

```
php artisan make:migration --create="fotos" CrearTablaFotos
```

```
C:\wamp64\www\GestorImagenes>php artisan make:migration --create="fotos" CrearTablaFotos
Created Migration: 2016_08_02_223103_CrearTablaFotos
```

4. Asegúrese de que los archivos se visualicen en la carpeta **Database**



5. Borre el archivo “**create_password_resets_table.php**”, ya que nuestra modalidad de recuperado de contraseña, será diferente.

Ciertamente la tabla usuarios debe ser creada y está contemplada por Laravel, sin embargo, los campos no son los mismos del todo.

1. Abra el archivo “**create_users_table.php**”, modifíquelo de modo tal que quede como en la imagen mostrada.

```
public function up()
{
    Schema::create('usuarios', function(Blueprint $table)
    {
        $table->increments('id');
        $table->string('nombre');
        $table->string('email')->unique();
        $table->string('password', 60);
        $table->string('pregunta');
        $table->string('respuesta');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

Ahora modificaremos el archivo de migraciones para “album”

2. Abra el archivo “**CrearTablaAlbumes**” y modifíquelo para que quede de la siguiente manera:

```
public function up()
{
    Schema::create('albumes', function(Blueprint $table)
    {
        $table->increments('id');
        $table->string('nombre');
        $table->string('descripcion');
        $table->integer('usuario_id')->unsigned();
        $table->foreign('usuario_id')->references('id')->on('usuarios');
        $table->timestamps();
    });
}
```

3. De manera análoga modifique el archivo “**CrearTablaFotos**” y modifíquelo para que quede de la siguiente manera:

```

public function up()
{
    Schema::create('fotos', function(Blueprint $table)
    {
        $table->increments('id');
        $table->string('nombre');
        $table->string('descripcion');
        $table->string('ruta');
        $table->integer('album_id')->unsigned();
        $table->foreign('album_id')->references('id')->on('albumes');
        $table->timestamps();
    });
}

```

4. Grabe los archivos de migración.
5. Ingrese a la ventana de comandos con ruta en nuestro proyecto.
6. Ejecute el siguiente comando para proceder con la migración:

php artisan migrate

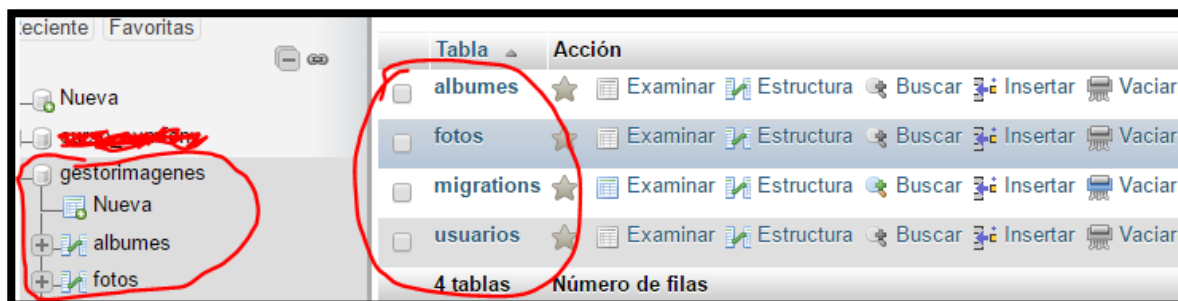
```

C:\wamp64\www\GestorImagenes>php artisan migrate
Migration table created successfully.
Migrated: 2014_10_12_000000_create_users_table
Migrated: 2016_08_02_222916_CrearTablaAlbumes
Migrated: 2016_08_02_223103_CrearTablaFotos

C:\wamp64\www\GestorImagenes>

```

7. Verifique mediante **phpmyadmin**, que las tablas se hayan creado correctamente en la base de datos.



CREANDO SEEDERS

1. Abra el archivo **GestorImagenes/database/seeds/DatabaseSeeder.php**.
2. Modifique el archivo para que quede de la siguiente manera:

Es importante el orden de creación de los Seeders, primero van las tablas menos dependientes de otras.

```

3  use Illuminate\Database\Seeder;
4  use Illuminate\Database\Eloquent\Model;
5
6  use GestorImagenes\Album;
7  use GestorImagenes\Foto;
8  use GestorImagenes\Usuario;
9
10 class DatabaseSeeder extends Seeder {
11
12     /**
13      * Run the database seeds.
14      *
15      * @return void
16      */
17     public function run()
18     {
19         DB::statement('SET FOREIGN_KEY_CHECKS = 0');
20
21         Foto::truncate();
22         Album::truncate();
23         Usuario::truncate();
24
25         $this->call('UsuariosSeeder');
26         $this->call('AlbumesSeeder');
27         $this->call('FotosSeeder');
28     }
29 }
30
31

```

3. A continuación, haga 3 copias del archivo **DatabaseSeeder** con los nombres **UsuariosSeeder**, **FotosSeeder** y **AlbumesSeeder**.
4. Modifique el archivo **UsuariosSeeder** para que quede de la siguiente manera:

```

10 class UsuariosSeeder extends Seeder {
11
12     /**
13      * Run the database seeds.
14      *
15      * @return void
16      */
17     public function run()
18     {
19         for ($i=0; $i < 50; $i++) {
20             Usuario::create(
21                 [
22                     'nombre' => "usuario$i",
23                     'email' => "email$i@test.com",
24                     'password' => bcrypt("pass$i"),
25                     'pregunta' => "preg$i",
26                     'respuesta' => "resp$i"
27                 ]
28             );
29         }
30     }
31 }
32

```

EXPLICACIÓN:

1. Creamos un ciclo **for** para crear 50 usuarios al azar en la base de datos. No consideramos su ID porque es autogenerado.
 2. La función **bcrypt()**, sirve para encriptar el parámetro entregado
5. Modifique el archivo **AlbumesSeeder** para que quede de la siguiente manera:

```
public function run()
{
    $usuarios=Usuario::all();//eloquent
    $contador=0;
    foreach ($usuarios as $usuario)
    {
        $cantidad=mt_rand(0,15);
        for ($i=0; $i < $cantidad; $i++){
            $contador++;
            Album::create(
                [
                    'nombre' => "Nombre Album$contador",
                    'descripcion' => "Descripción álbum $contador",
                    'usuario_id' => $usuario->id
                ]
            );
        }
    }
}
```

6. Modifique el archivo **FotosSeeder** para que quede de la siguiente manera:

```
public function run()
{
    $albumes=Album::all();//eloquent
    $contador=0;
    foreach ($albumes as $album) {
        $cantidad=rand(0,5);
        for ($i=0; $i < $cantidad; $i++){
            $contador++;
            Foto::create(
                [
                    'nombre' => "Nombre Foto$contador",
                    'descripcion' => "Descripción foto$contador",
                    'ruta' => '/img/text.png',
                    'album_id' => $album->id
                ]
            );
        }
    }
}
```

7. **PASO RECOMENDADO:** Es recomendable ejecutar el siguiente comando antes del punto Nro. 8

composer dumpautoload -> Este comando sirve para refrescar o actualizar un archivo en el que figuran los seeders, sus nombres y demás características; Y ya que hemos creado nuevos seeds, es recomendable hacer uso del comando en mención.

8. Abra la ventana de comandos con ruta en el proyecto y ejecute el siguiente comando:

php artisan db:seed

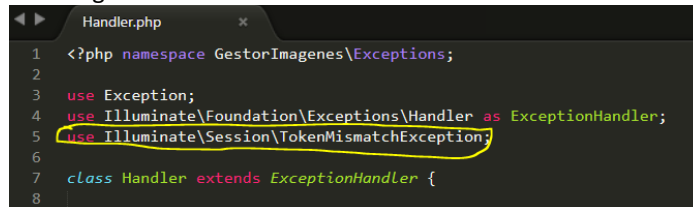
El resultado debería ser similar al siguiente:

```
C:\wamp64\www\GestorImagenes>php artisan db:seed
Seeded: UsuariosSeeder
Seeded: AlbumesSeeder
Seeded: FotosSeeder
```

Explore la base de datos y responda:
¿Qué sucedió?

CREANDO MENSAJES Y PÁGINAS DE EXCEPCIÓN PERSONALIZADAS

1. Abra el archivo **app/exception/Handler.php**, y modifique las líneas resaltadas en la función render
 - a. Primero importe la librería siguiente:

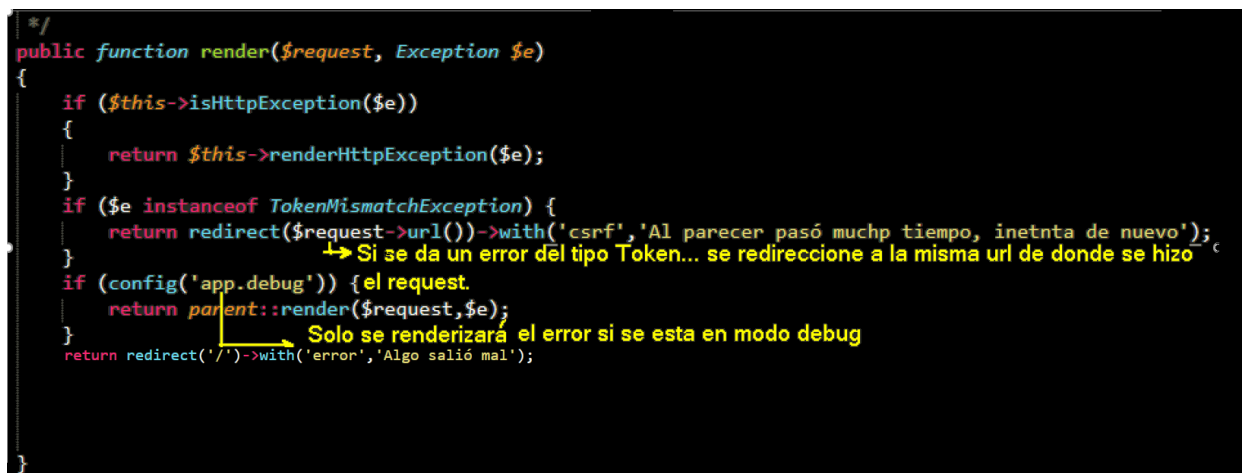


```

1 <?php namespace GestorImagenes\Exceptions;
2
3 use Exception;
4 use Illuminate\Foundation\Exceptions\Handler as ExceptionHandler;
5 use Illuminate\Session\TokenMismatchException;
6
7 class Handler extends ExceptionHandler {
8

```

- b. Luego modificaremos la función render para que quede de la siguiente manera.



```

*/
public function render($request, Exception $e)
{
    if ($this->isHttpException($e))
    {
        return $this->renderHttpException($e);
    }
    if ($e instanceof TokenMismatchException) {
        return redirect($request->url())->with('csrf','Al parecer pasó mucho tiempo, intenta de nuevo');
    }
    if (config('app.debug')) {el request.
        return parent::render($request,$e);
    }
    return redirect('/')->with('error','Algo salió mal');
}

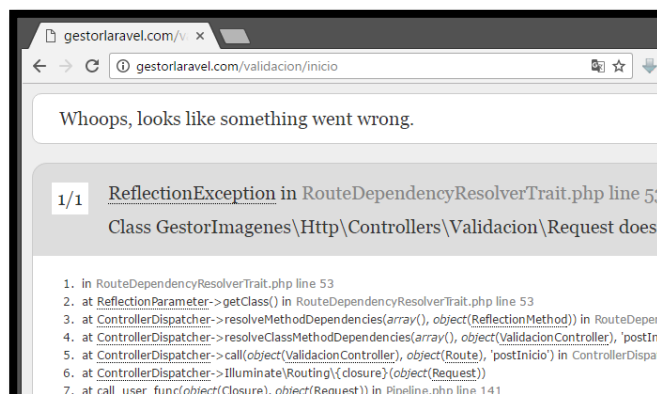
```

Aclaración: Se ha puesto un mensaje de exceso de tiempo sin actividad en el tipo de error Token... por que el token de un formulario cambia si pasa mucho tiempo.

Para probar que el error se está dando haremos lo siguiente:

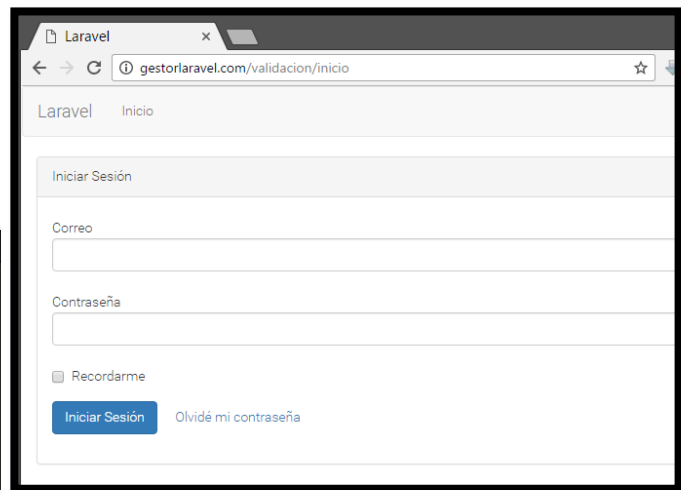
1. Vamos a probar que el error general (para que se renderice cuando estamos en modo debug), vamos a ocasionar un error de cualquier tipo en modo debug.

Asumiendo que ya estoy en modo debug (**TRUE en el archivo .env**).



Ahora generaremos el mismo error, solo que antes cambiaremos en el archivo `.env`, el modo **debug false**. Lo que debería suceder es que no se haga un **debug** del error y se re-direccione a la página a la página de inicio.

```
Handler.php .env
1 APP_ENV=local
2 APP_DEBUG=false
3 APP_KEY=MndsK41j0X5xPCqHyYNFDZtQiLYkan1z
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
```

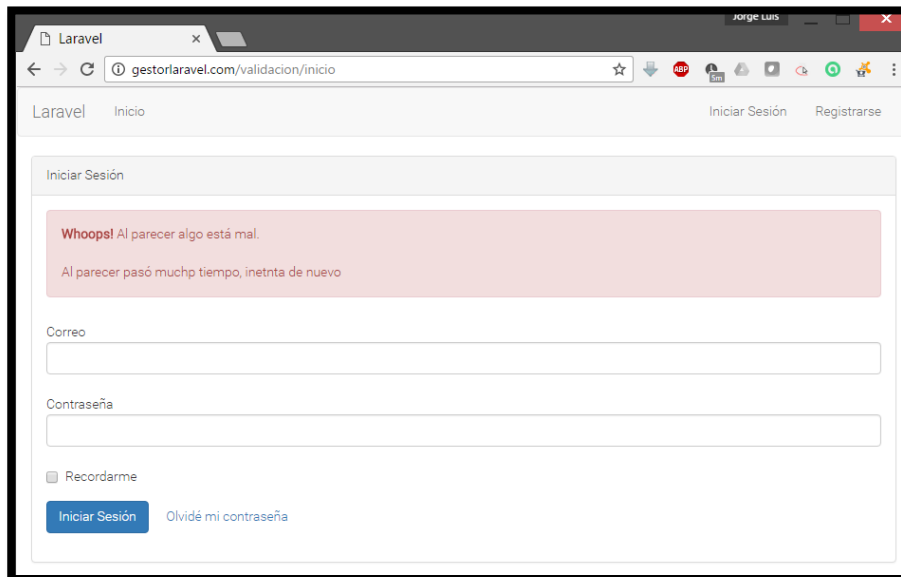


2. Vamos a probar que el error se dé por la excepción de Token...
 - a. Primero nos dirigimos a la vista de **inicio**, modificamos su contenido para agregar el siguiente código:

```
inicio.blade.php Handler.php .env
1 @extends('app')
2
3 @section('content')
4 <div class="container-fluid">
5   <div class="row">
6     <div class="col-md-8 col-md-offset-2">
7       <div class="panel panel-default">
8         <div class="panel-heading">Iniciar Sesión</div>
9         <div class="panel-body">
10           @if (count($errors) > 0)
11             <div class="alert alert-danger">
12               <strong>Whoops!</strong> Al parecer algo está mal.<br><br>
13               <ul>
14                 @foreach ($errors->all() as $error)
15                   <li>{{ $error }}</li>
16                 @endforeach
17               </ul>
18             </div>
19           @endif
20           @if (Session::has('csrf'))
21             <div class="alert alert-danger">
22               <strong>Whoops!</strong> Al parecer algo está mal.<br><br>
23               {{Session::get('csrf')}}
24             </div>
25           @endif
26
```

NOTA: Como podemos observar, estamos analizando que cuando un error de tipo Token... se dé, la función `render` que captura ese error renderiza la misma URL pero en la sesión crea un variable de nombre `"csrf"`, esta variable contiene un mensaje que será mostrado justo en el body de la página de lanzarse el error.

- b. Ahora solo queda probar el formulario, pero eliminando el token con la función "inspeccionar elemento" de nuestro navegador, y hacer submit.



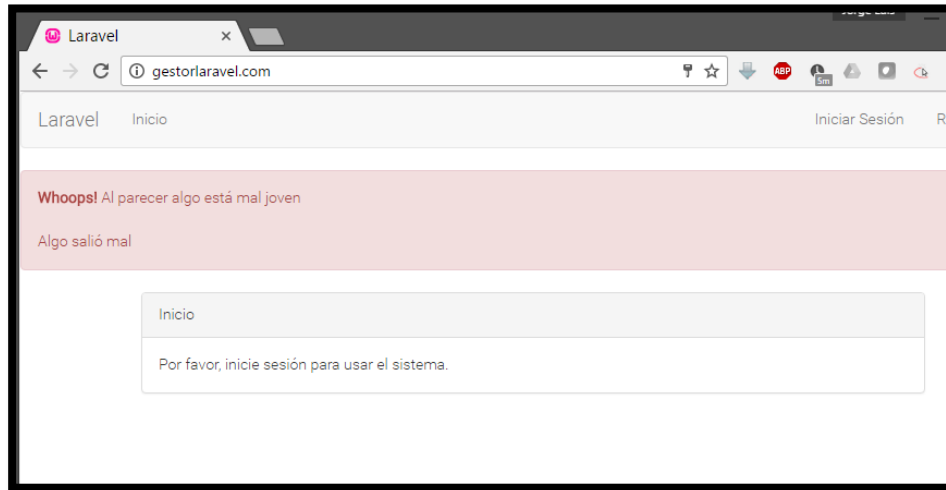
3. Vamos a probar una excepción en el inicio al momento de tratar de acceder a la aplicación.
- El primer caso es que el usuario no esté validado, de modo que modificaremos el archivo **bienvenida.blade.php**, y le añadiremos el siguiente código:

```
1 @extends('app')
2
3 @section('content')
4
5 @if(Session::has('error'))
6     <div class="alert alert-danger">
7         <strong>Whoops!</strong> Al parecer algo está mal joven <br><br>
8         {{Session::get('error')}}
9     </div>
10 @endif
11
12 <div class="container">
13     <div class="row">
14         <div class="col-md-10 col-md-offset-1">
15             <div class="panel panel-default">
16                 <div class="panel-heading">Inicio</div>
17
```

- La otra forma sería cuando el usuario sí esté validado, entonces el error se generaría en inicio, no en bienvenida. Modifique la vista inicio.blade.php, para que quede de la siguiente manera:

```
1 @extends('app')
2
3 @section('content')
4
5 @if(Session::has('error'))
6     <div class="alert alert-danger">
7         <strong>Whoops!</strong> Al parecer algo está mal joven <br><br>
8         {{Session::get('error')}}
9     </div>
10 @endif
11
12 <div class="container">
13     <div class="row">
14         <div class="col-md-10 col-md-offset-1">
```

Por último, para probar el error intentemos iniciar sesión con credenciales falsas, el resultado debería ser el siguiente:

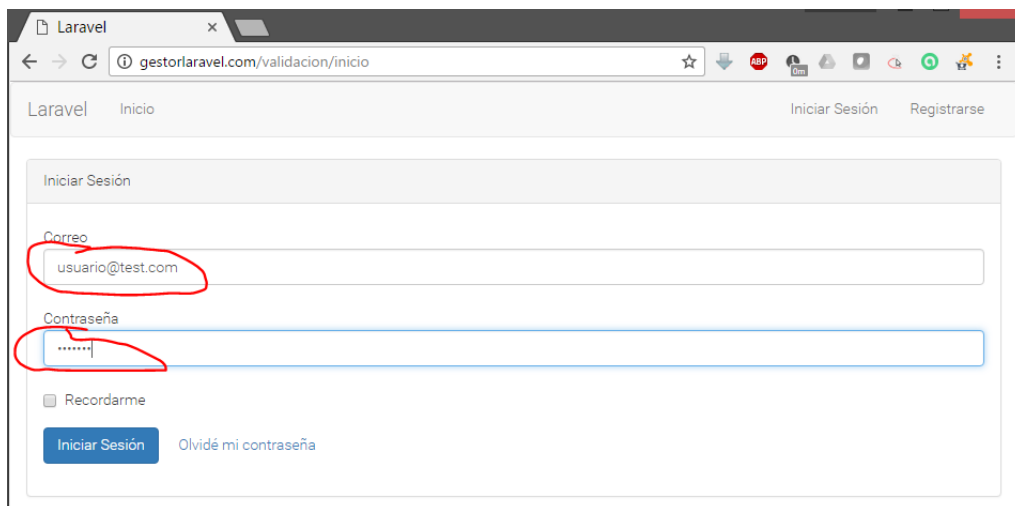


En ambos casos tendremos un error similar, claro que como no tenemos una credencial aún creada, no podemos probar el error cuando el usuario está loggeado.

*** Para finalizar la sección, cambie el archivo .env a true el modo de debug, para identificar fácilmente los errores.

ACCESO Y VALIDACIÓN DE USUARIOS

1. Identifiquemos el error que se produce al momento tratar de iniciar sesión con una cuenta errónea.



2. Clic en “Iniciar Sesión”.



NOTA: El error presentado indica que no existe la clase Request.

Este error se debe a que no lo estamos importando, por ende, no estamos haciendo uso de ella.

3. Diríjase a la clase ValidaciónController y agregue el “USE” que se indica a continuación:

```
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 use Illuminate\Http\Request;
8
9 class ValidacionController extends Controller {
10
```

Ahora modificaremos otro posible error.

Cuando el usuario intente cerrar sesión hará uso de la función a continuación:

```
134 public function getSalida()
135 {
136     $this->auth->logout();
137
138     return redirect(property_exists($this, 'redirectAfterLogout') ? $this->redirectAfterLogout : '/');
139 }
140
```

La función getSalida() se ejecutará cuando el usuario desee cerrar sesión, y la redirección se hará hacia “/” , sin embargo si observamos el constructor de esta clase:

```
public function __construct(Guard $auth, Registrar $registrar)
{
    $this->auth = $auth;
    $this->registrar = $registrar;

    $this->middleware('guest', ['except' => 'getLogout']);
}
```

Tenemos la ejecución del Middleware “guest”, Como se vio en teoría, este Middleware se encarga de ver si es que existe un usuario de tipo digamos invitado, se tomarán acciones especificadas en su función, sin embargo se hace excepción al

momento de ejecutar una función “getLogout”, debemos cambiar esta función por “getSalida”. Realice el cambio para que quede de la siguiente manera:

```
public function __construct(Guard $auth, Registrar $registrar)
{
    $this->auth = $auth;
    $this->registrar = $registrar;

    $this->middleware('guest', ['except' => 'getSalida']);
}
```

Con este cambio ya podremos iniciar y cerrar sesión de una mejor manera.

Para registrar a un usuario el constructor hace uso de una variable de tipo de dato Registrar con el nombre \$registrar.

```
public function __construct(Guard $auth, Registrar $registrar)
{
    $this->auth = $auth;
    $this->registrar = $registrar;
}
```

Para registrarse, se hace uso de la función postRegistro()

```
public function postRegistro(Request $request)
{
    $validator = $this->registrar->validator($request->all());

    if ($validator->fails())
    {
        $this->throwValidationException(
            $request, $validator
        );
    }

    $this->auth->login($this->registrar->create($request->all()));

    return redirect($this->redirectPath());
}
```

¿Dónde se encuentra la clase Registrar?

4. Ubique el archivo **app/Services/Registrar.php**.

```

1  <?php namespace GestorImagenes\Services;
2
3  use GestorImagenes\User;
4  use Validator;
5  use Illuminate\Contracts\Auth\Registrar as RegistrarContract;
6
7  class Registrar implements RegistrarContract {
8
9      /**
10       * Get a validator for an incoming registration request.
11       *
12       * @param array $data
13       * @return \Illuminate\Contracts\Validation\Validator
14       */
15     public function validator(array $data)
16     {
17         return Validator::make($data, [
18             'name' => 'required|max:255',
19             'email' => 'required|email|max:255|unique:users',
20             'password' => 'required|confirmed|min:6',
21         ]);
22     }
23
24     /**
25      * Create a new user instance after a valid registration.
26      *
27      * @param array $data
28      * @return User
29      */
30     public function create(array $data)
31     {
32         return User::create([
33             'name' => $data['name'],
34             'email' => $data['email'],
35             'password' => bcrypt($data['password']),
36         ]);
37     }
38
39 }
40

```

En este archivo, la función `validator()`, se encarga de verificar que, al momento del registro de un usuario, se reciban los campos correspondientes y de hecho, cumplan con ciertas características, sin embargo, los campos se han modificado.

5. Modifique el archivo para que, los campos del registro coincidan, de la siguiente manera:

```

15     public function validator(array $data)
16     {
17         return Validator::make($data, [
18             'nombre' => 'required|max:255',
19             'email' => 'required|email|max:255|unique:usuarios',
20             'password' => 'required|confirmed|min:6',
21             'pregunta' => 'required|max:255',
22             'respuesta' => 'required|max:255',
23         ]);
24     }
25

```

Hasta aquí, tenemos lista la forma de validación de datos en el formulario de registro.

Regresando a la función **postRegistro()**:

```

public function postRegistro(Request $request)
{
    $validator = $this->registrar->validator($request->all());
    if ($validator->fails())
    {
        $this->throwValidationException(
            $request, $validator
        );
    }
    $this->auth->login($this->registrar->create($request->all()));
    return redirect($this->redirectPath());
}

```

Como podemos observar en la función (ACTION) **postRegistro**, se hace uso de la función `validator()` con la finalidad de validar los campos al momento de registrar a un usuario, sin embargo luego se hace uso del ciclo IF, en caso de que los campos no hayan sido validados correctamente.

Si no se entra al ciclo IF, la siguiente función a utilizar será **create()**; función encargada de crear el registro para el usuario en el archivo `Registrar.php`; Sin embargo este archivo no se encuentra correctamente modificado ya que los campos del usuario han cambiado.

TAREA: INVESTIGUE INYECCIÓN POR DEPENDENCIA EN LARAVEL.

6. Modifique el archivo Registrar.php para que coincida con el siguiente código:

```
31  */
32  public function create(array $data)
33  {
34      return Usuario::create([
35          'nombre' => $data['nombre'],
36          'email' => $data['email'],
37          'password' => bcrypt($data['password']),
38          'pregunta' => $data['pregunta'],
39          'respuesta' => $data['respuesta'],
40      ]);
41  }
42
43  }
```

Esta modificación se realizó para que al momento de crear un usuario.

Laravel utiliza una tabla por defecto para la creación y validación de una cuenta de usuarios. Esto se hace por medio de la carpeta **CONFIG**, en su archivo **AUTH**.

7. Modifique el archivo app/config/auth.php. Asignemos el verdadero modelo.

```
28  |
29  */
30
31  'model' => 'GestorImagenes\Usuarid',
32
33  /*
34  |-----
```

Sin más modificaciones pruebe que puede ingresar al sistema con una cuenta real. El resultado debería ser el inicio de sesión exitoso.

¿Cuáles fueron las credenciales utilizadas para acceder al sistema?

El resultado debería ser el siguiente:



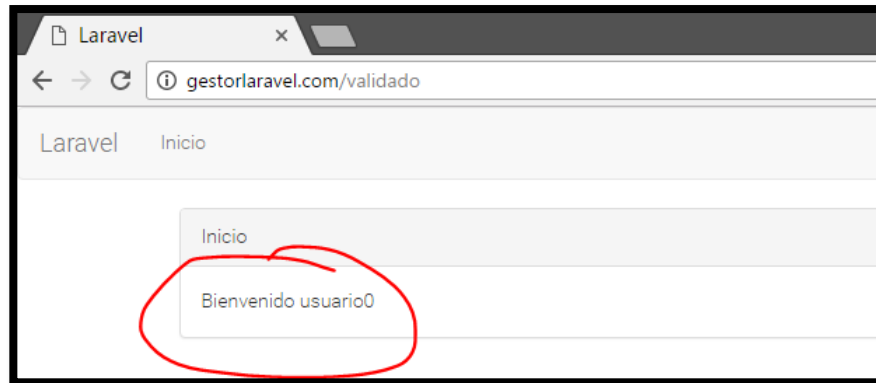
Para darle un poco más de estilo, vamos a modificar la palabra usuario por el nombre del usuario que está ingresando al sistema.

8. Abra el archivo **inicio.blade.php** y modifique la impresión de la siguiente manera:

```
<div class= panel panel-default >
  <div class="panel-heading">Inicio</div>

  <div class="panel-body">
    Bienvenido {{Auth::user()->nombre}}
  </div>
</div>
</div>
```

9. Actualice la página y observe los cambios. El elemento Auth, es quien tiene las credenciales del usuario registrado.



10. Cierre Sesión.
11. Ingrese al menú para registrarse y ponga sus datos.

A screenshot of a web browser window. The address bar shows 'gestorlaravel.com/validacion/registro'. The page has a header with 'Laravel' and 'Inicio'. Below the header, there is a panel with a heading 'Registrarse'. The form contains the following fields: 'Nombre' with the value 'Jorge', 'Correo Electrónico' with the value 'jorge@test.com', 'Contraseña' with masked characters '*****', 'Confirm Password' with masked characters '*****', 'Pregunta' with the value 'preg', and 'Respuesta' with the value 'resp'. At the bottom of the form is a blue button labeled 'Registrarse'.

Resultado:



12. Verifique la creación del usuario en la tabla correspondiente en MySQL.

50	usuario49	email49@test.com	\$2y\$10\$QjHzqrp6oqnomJuzR8pgTuVa76q6nXX...	preg49	resp49	NULL
51	Jorge	jorge@test.com	\$2y\$10\$SfJui2xHng6nTlAs1.FOOFqbFZcywe...	preg	resp	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL

MODIFICANDO EL REQUEST DEL INICIO DE SESION

Hasta ahora, hemos terminado de corregir el request que envía la sección de REGISTRO (en resumen, una serie de reglas para validar los campos de registro, etc.), de manera análoga, trabajaremos con el inicio de sesión.

1. Abra el archivo ValidacionController y ubique el action postInicio (action con método post para el inicio de sesión).

```

96 public function postInicio(Request $request)
97 {
98     $this->validate($request, [
99         'email' => 'required|email', 'password' => 'required',
100     ]);
101     $credentials = $request->only('email', 'password');
102     if ($this->auth->attempt($credentials, $request->has('remember')))
103     {
104         return redirect()->intended($this->redirectPath());
105     }
106     return redirect($this->loginPath())
107         ->withInput($request->only('email', 'remember'))
108         ->withErrors([
109             'email' => $this->getFailedLoginMessage(),
110         ]);
111 }
112
113
114
115

```

Como se puede observar, en este caso la función **validate**, está creada dentro del **action**, por el tema de **INYECCIÓN DE DEPENDENCIA**, debería estar en un archivo distinto (análogo a **postRegistro()** con el archivo **Request** en la carpeta **http/requests**).

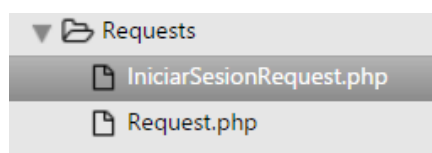
2. Abra la ventana de comandos y escriba el siguiente código:

```

C:\wamp64\www\GestorImagenes>php artisan make:request IniciarSesionRequest
Request created successfully.
C:\wamp64\www\GestorImagenes>

```

3. Verifique que el archivo haya sido creado en la carpeta de requests.



Tenemos por ejemplo a la función `authorize()`:

Puede devolver verdadero o falso, cuando retorne falso, se entiende que el usuario que hizo un tipo de petición no tiene el permiso suficiente para realizar la acción, sin embargo, como siempre se va a tratar de una operación de petición, el retorno será siempre TRUE.

- Seguidamente corte el código de las reglas detalladas en el action `postInicio()`, a la función `rules()` de nuestro Request creado.

```

19  *
20  * @return array
21  */
22  public function rules()
23  {
24      return [
25          'email' => 'required|email',
26          'password' => 'required',
27      ];
28  }
29
30  }
31

```

```

1  <?php namespace GestorImagenes\Http\Requests;
2
3  use GestorImagenes\Http\Requests\Request;
4
5  class IniciarSesionRequest extends Request {
6
7      /**
8       * Determine if the user is authorized to make this request.
9       *
10      * @return bool
11      */
12      public function authorize()
13      {
14          return true;
15      }
16
17      /**
18       * Get the validation rules that apply to the request.
19       *
20      * @return array
21      */
22      public function rules()
23      {
24          return [
25              //
26          ];
27      }
28  }
29

```

- En el action `postInicio` de nuestro controlador de Validación, modifique el tipo de dato que se recibe como parámetro, la idea es que ahora recibamos un tipo request => `IniciarSesionRequest()`.

```

96  public function postInicio(IniciarSesionRequest $request)
97  {
98
99      $credentials = $request->only('email', 'password');
100
101      if ($this->auth->attempt($credentials, $request->has('remember')))
102      {
103          return redirect()->intended($this->redirectPath());
104      }
105
106      return redirect($this->loginPath());

```

- Adicionalmente, en el mismo archivo (`ValidacionController.php`), haga un "use" del archivo request que acabamos de crear.

```

7  use Illuminate\Http\Request;
8  use GestorImagenes\Http\Requests\IniciarSesionRequest;
9

```

CREANDO LA FUNCIONALIDAD PARA RECUPERAR LA CONTRASEÑA POR PARTE DEL USUARIO

Ahora crearemos la funcionalidad para que el usuario pueda recuperar su contraseña.

- Ubiquemos el action `postRecuperar()` en `ValidacionController`.

```

162  }
163  public function postRecuperar(){
164      return 'recuperando contraseña';
165  }

```

Vamos a crear un **Request** personalizado para este proceso.

¿Cuál sería el proceso de crear un Request de nombre: "RecuperarContraseñaRequest"?

Créelo y coloque el resultado de la ejecución del comando adecuado.

- En el nuevo archivo creado (`RecuperarContraseñaRequest.php`), haremos una pequeña copia del Request para Iniciar Sesión en su sección de reglas.
- Copie parte del código (reglas), del request Iniciar sesión para que ahora el Request de Recuperar Contraseña que de la siguiente manera:

```

21  */
22  public function rules()
23  {
24      return
25      [
26          'email' => 'required|email|exists:usuarios,email',
27          'password' => 'required|min:6|confirmed',
28          'pregunta' => 'required',
29          'respuesta' => 'required',
30      ];
31  }
32
33  }
34

```

Al usar las **reglas** de este **Request**, estamos indicando que el campo **email** será requerido, y que además exista en la tabla **“usuarios”** en su columna **email**. Por otro lado, que el **password** sea **requerido** y tenga como **mínimo** 6 caracteres.

- En el Request recientemente creado coloque en “true”, el valor de retorno en su función authorize:

```

12  public function authorize()
13  {
14      return true;
15  }
16

```

- Ya que vamos a usar el nuevo Request en nuestro controlador de Validación, importe o haga referencia al archivo con un “use”. Además usaremos el modelo de usuario, de modo que también incluiremos la referencia al mismo:

```

8  use GestorImagenes\Http\Requests\IniciarSesionRequest;
9  use GestorImagenes\Http\Requests\RecuperarContraseñaRequest;
10 use GestorImagenes\Usuario;
11 class ValidacionController extends Controller {
12

```

- Ahora modifique la función postRecuperar() para que quede de la siguiente manera:

```

164 public function postRecuperar(RecuperarContraseñaRequest $request){
165     $pregunta=$request->get('pregunta');
166     $respuesta=$request->get('respuesta');
167     $email=$request->get('email');
168     $usuario=Usuario::where('email','=',$email)->first();
169
170     if($pregunta==$usuario->pregunta && $respuesta==$usuario->respuesta){//tanto en tipo como en valor
171         $contrasena=$request->get('password');
172         $usuario->password=bcrypt($contrasena);
173         $usuario->save();
174         return redirect('/validacion/inicio')->with(['recuperada'=>'La contraseña se cambió. Inicia Sesión']);
175     }
176
177     return redirect('/validacion/recuperar')->withInput($request->only('email','pregunta'))
178     ->withErrors(['pregunta'=>'La pregunta y/o respuesta ingresada no coinciden.']);
179 }

```

EXPLICACIÓN:

Ya que postRecuperar(), es una función que se ejecutará mediante el verbo POST, éste action servirá para guardar los cambios en caso de querer modificar una contraseña.

Si observamos detenidamente, lo primero que debería hacer esta función, es corroborar que todos los campos en el registro de Recuperar Contraseña, hayan sido llenados correctamente. Sin embargo, quien está encargado de hacer ese trabajo es el Request que hemos creado en su función rules; Aquí ya no nos preocupamos por eso. Este mecanismo se hace llamar INYECCIÓN DE DEPENDENCIA =).

Entonces, como tenemos claro que todos los campos están ya debidamente verificados, primero creamos variables que serán recuperadas a través de la variable request.

En la línea 168: creamos un objeto del modelo USUARIO con una consulta ELOQUENT donde se especifica que el usuario buscado será el que en su campo EMAIL tenga el email que hemos recibido del Request. La función first(), indica que se obtendrá un único registro con ese email (Aunque está claro que los correos deberían ser únicos.). [LA FUNCIÓN FIND DE ELOQUENT, ES SIMILAR.].

En la línea 170: La estructura condicional será usada con la finalidad de verificar que la pregunta ingresada sea igual a la pregunta del usuario recuperado de la base de datos no solo en valor sino también en tipo (===); lo mismo sucede con la respuesta.

En caso de que la estructura condicional mencionada en el párrafo anterior, esté correcta; Del Request recuperamos la nueva contraseña ingresada por el usuario. Luego al objeto \$usuario, asignamos el valor de su nueva contraseña(haciendo uso de la función bcrypt para que la guarde encriptada y no como tal.).

Antes de terminar se hace uso de la función SAVE en **usuario** para que sus cambios sean guardados.

Finalmente se hace una redirección al action INICIO del controlador VALIDACION para que se muestre la vista del inicio de sesión; Además enviamos un array asociativo con la variable **RECUPERADA** con el contenido: **LA CONTRASEÑA SE CAM....**

En caso de haber fallado el proceso de actualizar la contraseña del usuario, se hará una redirección al action RECUPERAR del action VALIDACIÓN, Es decir a la misma vista de donde intentó cambiar la contraseña. WithInput nos indica que cuando la página sea recargada, los inputs de email y pregunta, deberían escribirse de manera automática. WithErrors, envía un array asociativo con la variable **pregunta** y el contenido: **La pregunta y/o respu....**

Probando el código:

1. Intente recuperar la contraseña con un correo válido, dos contraseñas correctas (con más de 5 caracteres), pero con pregunta y/o respuesta equivocada. Anote los resultados

2. Intente recuperar la contraseña con todos los datos correctos. ¿Qué sucede?

Como ha notado, cuando comentemos cualquier tipo de error en el formulario de recuperación de contraseña, los errores son impresos en la pantalla debido al código que tenemos en la vista **recuperar**:

```
<div class="panel-heading">Recuperar Contraseña</div>
<div class="panel-body">
    @if (count($errors) > 0)
        <div class="alert alert-danger">
            <strong>Whoops!</strong> Al parecer algo está mal.<br><br>
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
    @endif

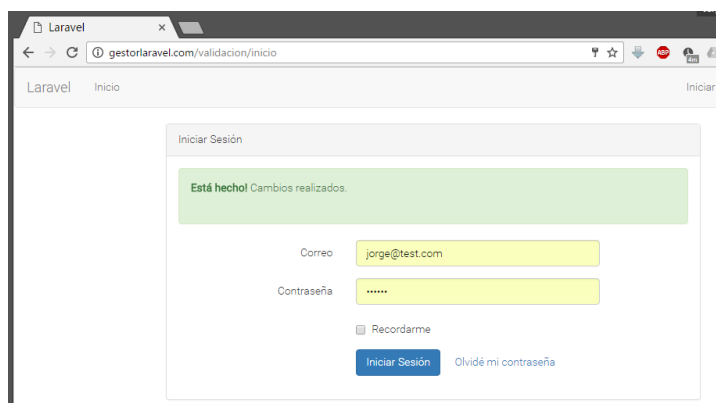
    <form class="form-horizontal" role="form" method="POST" action="/validacion/recuperar">
        <input type="hidden" name="_token" value="{{ csrf_token() }}">
    </form>
```

Sin embargo, cuando hacemos todo correctamente (es decir colocamos los campos como debieron ser y en efecto se cambia la contraseña), no obtenemos el mensaje personalizado que pusimos de contenido: **“La contraseña se cambió....”**. Esto se debe a que en la vista de inicio no se ha considerado poner este tipo de mensajes.

7. Ingrese a la vista de Inicio para modificar el siguiente código:

```
19         @endif
20         @if (Session::has('csrf'))
21             <div class="alert alert-danger">
22                 <strong>Whoops!</strong> Al parecer algo está mal.<br><br>
23                 {{Session::get('csrf')}}
24             </div>
25         @endif
26
27         @if (Session::has('recuperada'))
28             <div class="alert alert-success">
29                 <strong>Está hecho!</strong> Cambios realizados.<br><br>
30                 {{Session::get('recuperada')}}
31             </div>
32         @endif
33
34         <form class="form-horizontal" role="form" method="POST" action="/validacion/inicio">
35             <input type="hidden" name="_token" value="{{ csrf_token() }}">
36
37             <div class="form-group">
```

8. Intente recuperar una contraseña nuevamente, con todos los datos correctos y valide que los resultados son parecidos a la imagen a continuación:



The screenshot shows a web browser window with the address bar displaying "gestorlaravel.com/validacion/inicio". The page title is "Laravel Inicio". The main content area is titled "Iniciar Sesión" and features a green success message: "Está hechol Cambios realizados." Below this, there are two input fields: "Correo" with the value "jorge@test.com" and "Contraseña" with masked characters "*****". A checkbox labeled "Recordarme" is present. At the bottom, there is a blue "Iniciar Sesión" button and a link "Olvidé mi contraseña".