1 LARAVEL Framework - Tutorial 03 - Creación de Sistema de Registros y Login en Laravel

Vamos a crear una aplicación en Laravel desde cero para permitir registro de usuarios y login en dicha aplicación.

1.1 Sumario

2 Configuración Inicial

- **ATENCION: Damos por supuesto que ya tenemos instalado un servidor web, php, mysql, compose, usuario MySQL y Base de Datos.**

```
# Crear una carpeta para la aplicación.
mkdir gestion
cd gestion

# Instalamos Laravel
composer create-project laravel/laravel . dev-master
```

3 Configuración de los Recursos

3.1 Configuración del fichero de entorno

*** Configuramos primero la conexión a la base de datos en el fichero .env:**

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=aRNXU9ArUhxqwIczPhxFMwVNPrhNCnDz

DB_HOST=localhost
DB_DATABASE=c2base2
DB_USERNAME=c2base2
DB_PASSWORD=xxxxxxxx

CACHE_DRIVER=file
SESSION_DRIVER=file
QUEUE_DRIVER=sync

MAIL_DRIVER=smtp
MAIL_HOST=mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
```

3.2 Modelos

- Ahora definimos el modelo que nos permitirá interactuar con la base de datos a través de Eloquent ORM.
- Para ello editamos el modelo app/User.php que ya está en la instalación por defecto.
- Si no existiera podríamos crearlo con:

```
# Con esta instrucción se creará una plantilla de Modelo y una Migration.
php artisan make:model User
```

- Editamos el fichero **app**/**User.php** y lo adaptamos a los siguiente campos:

```
<?php namespace App;

use Illuminate\Auth\Authenticatable;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Auth\Passwords\CanResetPassword;
use Illuminate\Contracts\Auth\Authenticatable as AuthenticatableContract;
use Illuminate\Contracts\Auth\CanResetPassword as CanResetPasswordContract;

class User extends Model implements AuthenticatableContract, CanResetPasswordContract {

use Authenticatable, CanResetPassword;

/**
         * The database table used by the model.
         *
         * @var string
         */
protected $table = 'users';

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

/**
         * The attributes excluded from the model's JSON form.
         *
         * @var array
         */
protected $hidden = ['password', 'remember_token'];
}
```

3.3 Migraciones

- Creamos las migraciones para la tabla **users**:

```
# Si no tuviéramos una plantilla de Migration para Users podríamos hacerla con:
php artisan make:migration create_users_table --create="users"
```

- Editamos el fichero **database/migrations/2015_04_21_150127_create_users_table.php** e indicamos los tipos de datos de la tabla users.

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration {

/**
         * Run the migrations.
         *
         * @return void
         */
public function up()
{
Schema::create('users', function(Blueprint $table)
{
$table->increments('id');
$table->string('email')->unique();
$table->string('password');
$table->string('username')->unique();
$table->text('bio')->nullable();
$table->rememberToken();
$table->timestamps();
});
}


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

}
```

- Ejecutamos la migration:

```
php artisan migrate
#Migration table created successfully.
#Migrated: 2014_10_12_000000_create_users_table
#Migrated: 2014_10_12_100000_create_password_resets_table

# Si queremos deshacer la creación de la tabla:
php artisan migrate:rollback

# Si nos diese un error
composer dump-autoload
```

3.4 Rutas

- Editamos el fichero app/routes.php

```
<?php

/*
|--------------------------------------------------------------------------
```

```
| Application Routes
|--------------------------------------------------------------------------
|
| Here is where you can register all of the routes for an application.
| It's a breeze. Simply tell Laravel the URIs it should respond to
| and give it the controller to call when that URI is requested.
|
*/

// Cuando usamos un controlador resource tenemos que implementar todos los métodos
// index, store, etc.. Aunque se pueden indicar en la ruta cuales no queremos con except
Route::resource('users','UsersController');
```

3.5 Controladores

- Creamos el controlador **app/Http/Controllers/UsersController.php** de la ruta definida anteriormente.

```
php artisan make:controller UsersController
```

- Contenido por defecto de **app/Http/Controllers/UsersController.php**:

```
<?php namespace App\Http\Controllers;

use App\Http\Requests;
use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

// Indicamos que trabajamos con Vistas
use View;

// Indicamos que usamos el Modelo User.
use App\User;

class UsersController extends Controller {

/**
         * Display a listing of the resource.
         *
         * @return Response
         */
public function index()
{
// Prueba a ver si funciona la ruta /users
// return 'Lista de todos los usuarios';

// Devolvemos una Vista con toda la lista de usuarios.
// Usamos el método Mágico withUsers que lo que envía es una
// variable $users que contiene todos los usuarios.
return view('users')->withUsers(User::all());
}

/**
         * Show the form for creating a new resource.
         *
         * @return Response
         */
public function create()
{
//
}

/**
         * Store a newly created resource in storage.
         *
         * @return Response
         */
public function store()
{
//
}
```

```
/**
         * Display the specified resource.
         *
         * @param  int  $id
         * @return Response
         */
public function show($id)
{
//
}

/**
         * Show the form for editing the specified resource.
         *
         * @param  int  $id
         * @return Response
         */
public function edit($id)
{
//
}

/**
         * Update the specified resource in storage.
         *
         * @param  int  $id
         * @return Response
         */
public function update($id)
{
//
}

/**
         * Remove the specified resource from storage.
         *
         * @param  int  $id
         * @return Response
         */
public function destroy($id)
{
//
}

}
```

3.6 FormBuilder para crear formularios

- En las **versiones anteriores de Laravel 5** se incluía un paquete llamado **FormBuilder** y **HTMLBuilder**.
- Con ese paquete era muy cómodo generar formularios y código HTML sin tener que escribir todo el código.
- Si queremos instalarlo podemos hacerlo siguiendo las instrucciones en: **http://laravelcollective.com/**
    - Editar el fichero **composer.json** y añadir en "require" la siguiente línea: **"laravelcollective/html": "~5.0"**
    - Actualizar el proyecto con: **composer update**
    - Editar el fichero **config/app.php** para añadir un nuevo **provider** y dos nuevos **alias**:

```
// config/app.php

'providers' => [
'Collective\Html\HtmlServiceProvider',
],

'aliases' => [
'Form'      => 'Collective\Html\FormFacade',
'Html'      => 'Collective\Html\HtmlFacade',
],
```

- En caso de fallos u problemas aquí va otra dirección con información para la instalación de HTMLBuilder y FormBuilder en: **http://josephralph.co.uk/laravel-5-html-and-form-builder-packages/**

- Para usar el formulario recordar que **los tags** a usar cambian a:

```
{!! Form::open(...) !!}
{!! Form::text(...) !!}
```

- Al crear los formularios con **FormBuilder** automáticamente añade el campo **hidden "_token"** para el control de CSRF y si usamos métodos **PUT, PATCH o DELETE** se añade un nuevo campo **hidden "_method"** que contiene la acción a realizar. Los datos se enviarán usando el método POST para las 3 acciones citadas anteriormente.

- **Formulario de actualización** con envío por PUT con **FormBuilder** dentro de una Vista que recibe una **variable $usuario**:

```
{!! Form::open(array('url'=>'users/'.$id,'method'=>'PUT')) !!}
<p>
{!! Form::label('username','Nuevo Usuario') !!}
{!! Form::text('username',$usuario->username) !!}
</p>
<p>
{!! Form::label('bio','Nueva Biografía') !!}
{!! Form::textarea('bio',$usuario->bio) !!}
</p>
<p>
{!! Form::label('password','Nueva Contraseña') !!}
{!! Form::text('password') !!}
</p>
<p>
{!! Form::submit('Actualizar Datos') !!}
</p>
{!! Form::close() !!}
```

- **Formulario clásico en HTML equivalente al anterior** con envío por PUT y que recibe una **variable $usuario**:

```
<form method="POST" action="/users/{{$id}}">
<p><label for="username">Nuevo Usuario: <input type="text" name="username" value="{{ $usuario->username }}"/></label></p>
<p><label for="bio">Nueva Biografía: <textarea name="bio">{{ $usuario->bio }}</textarea></p>
<p><label for="password">Nueva Contraseña:<input type="password" name="password" value=""/></label></p>
<input type="hidden" name="_token" value="{{ csrf_token() }}"/>
<input type="hidden" name="_method" value="PUT"/>
<button>Actualizar Datos</button>
</form>
```

3.7 Vistas

- Para poder trabajar con el controlador y mostrar resultados procedentes de rutas, necesitamos crear vistas.
- Nos creamos una carpeta **resources/views/layouts** para nuestras plantillas de la aplicación.
- Creamos un fichero **resources/views/layouts/master.blade.php** que contendrá el fichero master o esqueleto de la aplicación.

- Contenido de **resources/views/layouts/master.blade.php**:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<title>@yield('titulo')</title>
</head>
<body>
@yield('contenido')
</body>
</html>
```

- Creamos el fichero **resources/views/users.blade.php** con el siguiente contenido:

```
@extends('layouts/master')

@section ('titulo')Usuarios @stop

@section('contenido')
```

```
<h1>Listado de todos los usuarios</h1>
@foreach($users as $user)
<p>{{ $user->username }}</p>
@endforeach
@stop
```

4 Configuración del Registro de Usuarios

4.1 Programación de método create() en el controlador

- Necesitamos programar en el controlador de Users el método **create()** para mostrar un formulario de creación de usuarios.
- Modificación realizada en **app**/**Http**/**Controllers**/**UsersController.php**:

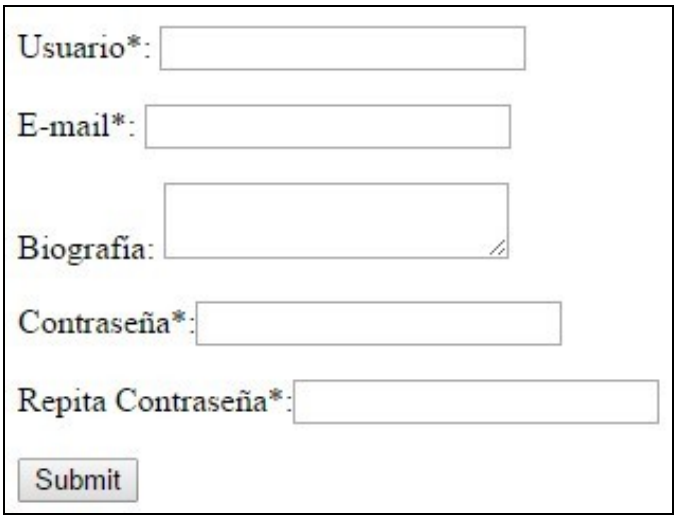
```
/**
         * Show the form for creating a new resource.
         *
         * @return Response
         */
public function create()
{
return view('create');
}
```

4.2 Creación de la vista con el formulario de alta

- Creamos la Vista **resources**/**views**/**create.blade.php**:

```
@extends('layouts.master')

@section('titulo')Regístrate @stop

@section('contenido')

@foreach($errors->all() as $error)
<p>{{$error}}</p>
@endforeach

<form method="POST" action="/users">
<p><label for="username">Usuario*: <input type="text" name="username" value="{{ Input::old('username') }}"/></label></p>
<p><label for="email">E-mail*: <input type="email" name="email" value="{{ Input::old('email') }}"/></label></p>
<p><label for="bio">Biografía: <textarea name="bio">{{ Input::old('bio') }}</textarea></p>
<p><label for="password">Contraseña*:<input type="password" name="password" value="{{ Input::old('password') }}"/></label></p>
<p><label for="password-repeat">Repita Contraseña*:<input type="password" name="password-repeat" value="{{ Input::old('password-repea
<input type="hidden" name="_token" value="{{ csrf_token() }}"/>
<button>Submit</button>
</form>
@stop
```

- Si vamos con el navegador a la URL /**uses**/**create** se mostrará el siguiente formulario:



4.3 Almacenamiento de datos

- Para gestionar la **recepción de datos del formulario** lo haremos en el método **store()** del controlador de Users
- Además realizaremos la **validación de datos del formulario**.
- Modificación de **app/Http/Controllers/UsersController.php** método **store()** para gestionar los datos recibidos del formulario:

```
<?php namespace App\Http\Controllers;

use App\Http\Requests;
use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

// Indicamos que trabajamos con Vistas
use View;

// Indicamos que usamos el Modelo User.
use App\User;

// Validación de formularios.
use Validator;

// Hash de contraseñas.
use Hash;

// Redireccionamientos.
use Redirect;

class UsersController extends Controller {

/**
         * Display a listing of the resource.
         *
         * @return Response
         */
public function index()
{
// Prueba a ver si funciona la ruta /users
// return 'Lista de todos los usuarios';

// Devolvemos una Vista con toda la lista de usuarios.
// Usamos el método Mágico withUsers que lo que envía es una
// variable $users que contiene todos los usuarios.
return view('users')->withUsers(User::all());
}

/**
         * Show the form for creating a new resource.
         *
         * @return Response
         */
```

```
public function create()
{
return view('create');
}

/**
         * Store a newly created resource in storage.
         *
         * @return Response
         */
public function store(Request $request)
{
// Realizamos la validación de datos recibidos del formulario.
$rules=array(
'username'=>'required|unique:users', // Username es único en la tabla users
'email'=>'required|email|unique:users', // Username es único en la tabla users
'password'=>'required|min:6',
'password-repeat'=>'required|same:password'
);

// Llamamos a Validator pasándole las reglas de validación.
$validator=Validator::make($request->all(),$rules);

// Si falla la validación redireccionamos de nuevo al formulario
// enviando la variable Input (que contendrá todos los Input recibidos)
// y la variable errors que contendrá los mensajes de error de validator.
if ($validator->fails())
{
return Redirect::to('users/create')
->withInput()
->withErrors($validator->messages());
}

// Si la validación es OK, estamos listos para almacenar en la base de datos los datos.
User::create(array(
'username'=>$request->input('username'),
'email'=>$request->input('email'),
'password'=>Hash::make($request->input('password')),
'bio'=>$request->input('bio')
));

// Redireccionamos a users
return Redirect::to('users');
}

/**
         * Display the specified resource.
         *
         * @param  int  $id
         * @return Response
         */
public function show($id)
{
//
}

/**
         * Show the form for editing the specified resource.
         *
         * @param  int  $id
         * @return Response
         */
public function edit($id)
{
//
}

/**
         * Update the specified resource in storage.
         *
         * @param  int  $id
         * @return Response
         */
```

```
public function update($id)
{
//
}

/**
         * Remove the specified resource from storage.
         *
         * @param  int  $id
         * @return Response
         */
public function destroy($id)
{
//
}

}
```

5 Configuración de páginas de perfiles de usuarios

- La idea es que cuando se muestren todos los usuarios, apunten mediante un hiperenlace al perfil con la información de ese usuario.

5.1 Modificación Vista Users

- Modificación de plantilla **resources/views/users.blade.php**:

```
@extends('layouts/master')

@section ('titulo')Usuarios @stop

@section('contenido')
<h1>Listado de todos los usuarios</h1>
@foreach($users as $user)
<p>{{ $user->username }} (<a href='users/{{$user->id}}'>ver perfil</a>)</p>
@endforeach
@stop
```

5.2 Programación método show($id) en controlador UsersController.php

```
<?php namespace App\Http\Controllers;

use App\Http\Requests;
use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

// Indicamos que trabajamos con Vistas
use View;

// Indicamos que usamos el Modelo User.
use App\User;

// Validación de formularios.
use Validator;

// Hash de contraseñas.
use Hash;

// Redireccionamientos.
use Redirect;

class UsersController extends Controller {

/**
         * Display a listing of the resource.
         *
         * @return Response
         */
public function index()
```

```
{
// Prueba a ver si funciona la ruta /users
// return 'Lista de todos los usuarios';

// Devolvemos una Vista con toda la lista de usuarios.
// Usamos el método Mágico withUsers que lo que envía es una
// variable $users que contiene todos los usuarios.
return view('users')->withUsers(User::all());
}

/**
         * Show the form for creating a new resource.
         *
         * @return Response
         */
public function create()
{
return view('create');
}

/**
         * Store a newly created resource in storage.
         *
         * @return Response
         */
public function store(Request $request)
{
// Realizamos la validación de datos recibidos del formulario.
$rules=array(
'username'=>'required|unique:users', // Username es único en la tabla users
'email'=>'required|email|unique:users', // Username es único en la tabla users
'password'=>'required|min:6',
'password-repeat'=>'required|same:password'
);

// Llamamos a Validator pasándole las reglas de validación.
$validator=Validator::make($request->all(),$rules);

// Si falla la validación redireccionamos de nuevo al formulario
// enviando la variable Input (que contendrá todos los Input recibidos)
// y la variable errors que contendrá los mensajes de error de validator.
if ($validator->fails())
{
return Redirect::to('users/create')
->withInput()
->withErrors($validator->messages());
}

// Si la validación es OK, estamos listos para almacenar en la base de datos los datos.
User::create(array(
'username'=>$request->input('username'),
'email'=>$request->input('email'),
'password'=>Hash::make($request->input('password')),
'bio'=>$request->input('bio')
));

// Redireccionamos a users
return Redirect::to('users');
}

/**
         * Display the specified resource.
         *
         * @param  int  $id
         * @return Response
         */
public function show($id)
{
// Se muestra la información de un usuario.
// Comprobamos si el $id existe en la base de datos.
$usuario=User::find($id);

if ($usuario== null)
```

```
return Redirect::to('users');

return view('perfil')->withElusuario($usuario);
}

/**
         * Show the form for editing the specified resource.
         *
         * @param  int  $id
         * @return Response
         */
public function edit($id)
{
//
}

/**
         * Update the specified resource in storage.
         *
         * @param  int  $id
         * @return Response
         */
public function update($id)
{
//
}

/**
         * Remove the specified resource from storage.
         *
         * @param  int  $id
         * @return Response
         */
public function destroy($id)
{
//
}

}
```

5.3 Creación de vista resources/views/perfil.blade.php

```
@extends('layouts/master')

@section ('titulo')Usuario {{$elusuario->username}} @stop

@section('contenido')
<h2>Usuario: {{$elusuario->username}}</h2>
<p>E-mail: {{$elusuario->email}}</p>
<p>Biografía: {{$elusuario->bio}}</p>
@stop
```

6 Permitir actualizaciones y borrados

6.1 Actualizaciones

- Para actualizar un usuario vamos a añadir un **enlace a Edit** en la vista que apunte a la edición de ese usuario.
- Contenido de **app/resources/views/perfil.blade.php**:

```
@extends('layouts/master')

@section ('titulo')Usuario {{$elusuario->username}} @stop

@section('contenido')
<h2>Usuario: {{$elusuario->username}}</h2>
<p>E-mail: {{$elusuario->email}}</p>
<p>Biografía: {{$elusuario->bio}}</p>
<a href="/users/{{$elusuario->id}}/edit">Editar mi perfil</a> | <a href="/users">Volver</a>
@stop
```

- Ahora programaremos en el controlador **app/Http/Controllers/UsersController.php** el método **edit()** que renderizará el formulario de Edición:

```
<?php namespace App\Http\Controllers;

use App\Http\Requests;
use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

// Indicamos que trabajamos con Vistas
use View;

// Indicamos que usamos el Modelo User.
use App\User;

// Validación de formularios.
use Validator;

// Hash de contraseñas.
use Hash;

// Redireccionamientos.
use Redirect;

class UsersController extends Controller {

/**
         * Display a listing of the resource.
         *
         * @return Response
         */
public function index()
{
// Prueba a ver si funciona la ruta /users
// return 'Lista de todos los usuarios';

// Devolvemos una Vista con toda la lista de usuarios.
// Usamos el método Mágico withUsers que lo que envía es una
// variable $users que contiene todos los usuarios.
return view('users')->withUsers(User::all());
}

/**
         * Show the form for creating a new resource.
         *
         * @return Response
         */
public function create()
{
return view('create');
}

/**
         * Store a newly created resource in storage.
         *
         * @return Response
         */
public function store(Request $request)
{
// Realizamos la validación de datos recibidos del formulario.
$rules=array(
'username'=>'required|unique:users', // Username es único en la tabla users
'email'=>'required|email|unique:users', // Username es único en la tabla users
'password'=>'required|min:6',
'password-repeat'=>'required|same:password'
);

// Llamamos a Validator pasándole las reglas de validación.
$validator=Validator::make($request->all(),$rules);

// Si falla la validación redireccionamos de nuevo al formulario
// enviando la variable Input (que contendrá todos los Input recibidos)
```

```
// y la variable errors que contendrá los mensajes de error de validator.
if ($validator->fails())
{
return Redirect::to('users/create')
->withInput()
->withErrors($validator->messages());
}

// Si la validación es OK, estamos listos para almacenar en la base de datos los datos.
User::create(array(
'username'=>$request->input('username'),
'email'=>$request->input('email'),
'password'=>Hash::make($request->input('password')),
'bio'=>$request->input('bio')
));

// Redireccionamos a users
return Redirect::to('users');
}

/**
         * Display the specified resource.
         *
         * @param  int  $id
         * @return Response
         */
public function show($id)
{
// Se muestra la información de un usuario.
// Comprobamos si el $id existe en la base de datos.
$usuario=User::find($id);

if ($usuario== null)
return Redirect::to('users');

return view('perfil')->withElusuario($usuario);
}

/**
         * Show the form for editing the specified resource.
         *
         * @param  int  $id
         * @return Response
         */
public function edit($id)
{
// Se muestra la información de un usuario.
// Comprobamos si el $id existe en la base de datos.
$usuario=User::find($id);

if ($usuario== null)
return Redirect::to('users');

return view('editar')->with('id',$id);
}

/**
         * Update the specified resource in storage.
         *
         * @param  int  $id
         * @return Response
         */
public function update($id)
{
//
}

/**
         * Remove the specified resource from storage.
         *
         * @param  int  $id
         * @return Response
         */
```

```
public function destroy($id)
{
//
}

}
```

- Contenido de **app/resources/views/editar.blade.php** dónde todos los campos son opcionales:

```
@extends('layouts.master')

@section('titulo')Editar mi Perfil @stop

@section('contenido')

@foreach($errors->all() as $error)
<p>{{$error}}</p>
@endforeach

<form method="POST" action="/users/{{$id}}">
<p><label for="username">Nuevo Usuario: <input type="text" name="username" value=""/></label></p>
<p><label for="email">Nuevo Email: <input type="email" name="email" value=""/></label></p>
<p><label for="bio">Nueva Biografía: <textarea name="bio"></textarea></p>
<p><label for="password">Nueva Contraseña:<input type="password" name="password" value=""/></label></p>
<input type="hidden" name="_token" value="{{ csrf_token() }}"/>
<input type="hidden" name="_method" value="PUT"/>
<button>Actualizar Datos</button>
</form>

<br/><a href="/users/{{$id}}">Volver</a>
@stop
```

- Contenido de **app/Http/Controllers/UsersController.php** el método **update()** que recibirá los datos a actualizar:

```
<?php namespace App\Http\Controllers;

use App\Http\Requests;
use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

// Indicamos que trabajamos con Vistas
use View;

// Indicamos que usamos el Modelo User.
use App\User;

// Validación de formularios.
use Validator;

// Hash de contraseñas.
use Hash;

// Redireccionamientos.
use Redirect;

class UsersController extends Controller {

/**
         * Display a listing of the resource.
         *
         * @return Response
         */
public function index()
{
// Prueba a ver si funciona la ruta /users
// return 'Lista de todos los usuarios';

// Devolvemos una Vista con toda la lista de usuarios.
// Usamos el método Mágico withUsers que lo que envía es una
// variable $users que contiene todos los usuarios.
return view('users')->withUsers(User::all());
```

```
}

/**
         * Show the form for creating a new resource.
         *
         * @return Response
         */
public function create()
{
return view('create');
}

/**
         * Store a newly created resource in storage.
         *
         * @return Response
         */
public function store(Request $request)
{
// Realizamos la validación de datos recibidos del formulario.
$rules=array(
'username'=>'required|unique:users', // Username es único en la tabla users
'email'=>'required|email|unique:users', // Username es único en la tabla users
'password'=>'required|min:6',
'password-repeat'=>'required|same:password'
);

// Llamamos a Validator pasándole las reglas de validación.
$validator=Validator::make($request->all(),$rules);

// Si falla la validación redireccionamos de nuevo al formulario
// enviando la variable Input (que contendrá todos los Input recibidos)
// y la variable errors que contendrá los mensajes de error de validator.
if ($validator->fails())
{
return Redirect::to('users/create')
->withInput()
->withErrors($validator->messages());
}

// Si la validación es OK, estamos listos para almacenar en la base de datos los datos.
User::create(array(
'username'=>$request->input('username'),
'email'=>$request->input('email'),
'password'=>Hash::make($request->input('password')),
'bio'=>$request->input('bio')
));

// Redireccionamos a users
return Redirect::to('users');
}

/**
         * Display the specified resource.
         *
         * @param  int  $id
         * @return Response
         */
public function show($id)
{
// Se muestra la información de un usuario.
// Comprobamos si el $id existe en la base de datos.
$usuario=User::find($id);

if ($usuario== null)
return Redirect::to('users');

return view('perfil')->withElusuario($usuario);
}

/**
         * Show the form for editing the specified resource.
         *
```

```
         * @param  int  $id
         * @return Response
         */
public function edit($id)
{
// Se muestra la información de un usuario.
// Comprobamos si el $id existe en la base de datos.
$usuario=User::find($id);

if ($usuario== null)
return Redirect::to('users');

return view('editar')->with('id',$id);
}

/**
         * Update the specified resource in storage.
         *
         * @param  int  $id
         * @return Response
         */
public function update($id,Request $request)
{
// Reglas de validación
$reglas = array(
'username' =>'unique:users',  // Deberá ser único en la tabla users
'email' =>'email|unique:users',  // Deberá ser único en la tabla users
'password' => 'min:6'
);

$validator= Validator::make($request->all(),$reglas);

if ($validator->fails())
{
return Redirect::to('users/'.$id.'/edit')
->withInput()
->withErrors($validator->messages());
}

$usuario = User::find($id);

if ($request->input('username'))
$usuario->username=$request->input('username');
if ($request->input('email'))
$usuario->email=$request->input('email');
if ($request->input('bio'))
$usuario->bio=$request->input('bio');
if ($request->input('password'))
$usuario->password=Hash::make($request->input('password'));

// Grabamos el usuario en la tabla.
$usuario->save();

// Redireccionamos a la página personal del usuario.
return Redirect::to('users/'.$id);
}

/**
         * Remove the specified resource from storage.
         *
         * @param  int  $id
         * @return Response
         */
public function destroy($id)
{
//
}

}
```

6.2 Borrados

- Para el borrado añadiremos en la página de edición **resources/views/editar.blade.php** un botón para borrar el usuario. Para ello generaremos un **formulario con el botón submit y action DELETE** (en este caso lo hacemos con **FormBuilder**):

```
@extends('layouts.master')

@section('titulo')Editar mi Perfil @stop

@section('contenido')

@foreach($errors->all() as $error)
<p>{{$error}}</p>
@endforeach

<form method="POST" action="/users/{{$id}}">
<p><label for="username">Nuevo Usuario: <input type="text" name="username" value=""/></label></p>
<p><label for="email">Nuevo Email: <input type="email" name="email" value=""/></label></p>
<p><label for="bio">Nueva Biografía: <textarea name="bio"></textarea></p>
<p><label for="password">Nueva Contraseña:<input type="password" name="password" value=""/></label></p>
<input type="hidden" name="_token" value="{{ csrf_token() }}"/>
<input type="hidden" name="_method" value="PUT"/>
<button>Actualizar Datos</button>
</form>

<hr/>

{!! Form::open(array('url'=>'users/'.$id,'method'=>'DELETE')) !!}
{!! Form::submit('Borrar mi Perfil') !!}
{!! Form::close() !!}

<br/><a href="/users/{{$id}}">Volver</a>

@stop
```

- Tendremos que programar el método **destroy()** de borrado en el controlador de Usuarios **app/Http/Controllers/UsersController.php**:

```
<?php namespace App\Http\Controllers;

use App\Http\Requests;
use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

// Indicamos que trabajamos con Vistas
use View;

// Indicamos que usamos el Modelo User.
use App\User;

// Validación de formularios.
use Validator;

// Hash de contraseñas.
use Hash;

// Redireccionamientos.
use Redirect;

class UsersController extends Controller {

/**
         * Display a listing of the resource.
         *
         * @return Response
         */
public function index()
{
// Prueba a ver si funciona la ruta /users
// return 'Lista de todos los usuarios';
```

```
// Devolvemos una Vista con toda la lista de usuarios.
// Usamos el método Mágico withUsers que lo que envía es una
// variable $users que contiene todos los usuarios.
return view('users')->withUsers(User::all());
}

/**
         * Show the form for creating a new resource.
         *
         * @return Response
         */
public function create()
{
return view('create');
}

/**
         * Store a newly created resource in storage.
         *
         * @return Response
         */
public function store(Request $request)
{
// Realizamos la validación de datos recibidos del formulario.
$rules=array(
'username'=>'required|unique:users', // Username es único en la tabla users
'email'=>'required|email|unique:users', // Username es único en la tabla users
'password'=>'required|min:6',
'password-repeat'=>'required|same:password'
);

// Llamamos a Validator pasándole las reglas de validación.
$validator=Validator::make($request->all(),$rules);

// Si falla la validación redireccionamos de nuevo al formulario
// enviando la variable Input (que contendrá todos los Input recibidos)
// y la variable errors que contendrá los mensajes de error de validator.
if ($validator->fails())
{
return Redirect::to('users/create')
->withInput()
->withErrors($validator->messages());
}

// Si la validación es OK, estamos listos para almacenar en la base de datos los datos.
User::create(array(
'username'=>$request->input('username'),
'email'=>$request->input('email'),
'password'=>Hash::make($request->input('password')),
'bio'=>$request->input('bio')
));

// Redireccionamos a users
return Redirect::to('users');
}

/**
         * Display the specified resource.
         *
         * @param  int  $id
         * @return Response
         */
public function show($id)
{
// Se muestra la información de un usuario.
// Comprobamos si el $id existe en la base de datos.
$usuario=User::find($id);

if ($usuario== null)
return Redirect::to('users');

return view('perfil')->withElusuario($usuario);
}
```

```
/**
         * Show the form for editing the specified resource.
         *
         * @param  int  $id
         * @return Response
         */
public function edit($id)
{
// Se muestra la información de un usuario.
// Comprobamos si el $id existe en la base de datos.
$usuario=User::find($id);

if ($usuario== null)
return Redirect::to('users');

return view('editar')->with('id',$id);
}

/**
         * Update the specified resource in storage.
         *
         * @param  int  $id
         * @return Response
         */
public function update($id,Request $request)
{
// Reglas de validación
$reglas = array(
'username' =>'unique:users',  // Deberá ser único en la tabla users
'email' =>'email|unique:users',  // Deberá ser único en la tabla users
'password' => 'min:6'
);

$validator= Validator::make($request->all(),$reglas);

if ($validator->fails())
{
return Redirect::to('users/'.$id.'/edit')
->withInput()
->withErrors($validator->messages());
}

$usuario = User::find($id);

if ($request->input('username'))
$usuario->username=$request->input('username');
if ($request->input('email'))
$usuario->email=$request->input('email');
if ($request->input('bio'))
$usuario->bio=$request->input('bio');
if ($request->input('password'))
$usuario->password=Hash::make($request->input('password'));

// Grabamos el usuario en la tabla.
$usuario->save();

// Redireccionamos a la página personal del usuario.
return Redirect::to('users/'.$id);
}


/**
         * Remove the specified resource from storage.
         *
         * @param  int  $id
         * @return Response
         */
public function destroy($id)
{
$usuario = User::find($id);
$usuario->delete();
```

```
return Redirect::to('users');
}

}
```

7 Gestión de la Autenticación

7.1 Creación de Controlador para gestión de Login/Logout

- Cuando instalamos Laravel por defecto instala un **HomeController**. En ese controlador es dónde se recomienda programar las rutas de la aplicación para usuarios autenticados.

- Editamos **app/Http/routes.php** para crear una nueva ruta que gestionará ese HomeController:

```
<?php
/*
|--------------------------------------------------------------------------
| Application Routes
|--------------------------------------------------------------------------
|
| Here is where you can register all of the routes for an application.
| It's a breeze. Simply tell Laravel the URIs it should respond to
| and give it the controller to call when that URI is requested.
|
*/

// Cuando usamos un controlador resource tenemos que implementar todos los métodos
// index, store, etc.. Aunque se pueden indicar en la ruta cuales no queremos con except
Route::resource('users','UsersController');

// Creamos un Controlador para gestionar la autenticación en HomeController.
Route::controller('/','HomeController');
```

7.2 Modificación Controlador HomeController para gestionar usuarios autenticados

- **Información sobre Autenticación en Laravel: http://laravel.com/docs/5.0/authentication**.
- Al instalar Laravel ya tenemos un HomeController que es dónde por defecto Laravel gestiona la web para usuarios autenticados, aunque ésto se podría modificar.
- Vamos a modificar el HomeController por defecto borrando sus métodos para adaptarlo a nuestra autenticación

- Contenido de **app/Http/Controllers/HomeController.php**:

```
<?php namespace App\Http\Controllers;
use Auth;
use Redirect;

class HomeController extends Controller {

/*
        |--------------------------------------------------------------------------
        | Home Controller
        |--------------------------------------------------------------------------
        |
        | This controller renders your application's "dashboard" for users that
        | are authenticated. Of course, you are free to change or remove the
        | controller as you wish. It is just here to get your app started!
        |
        */

public function getIndex()
{
return Redirect::to('users');
}


// Al acceder a la página /login validará las credenciales.
// http://laravel.com/docs/5.0/validation#controller-validation
public function getLogin()
```

```
{
// Creamos un array de credenciales a validar.
// Metemos un usuario de ejemplo para probar si funcionan las credenciales de validación.
$credenciales=array('username'=>'veiga','password'=>'abc123');

// Ejemplo de prueba de validación pero que NO loguea al usuario.
// return (Auth::validate($credenciales) ? 'Match': 'No match...');

// Ejemplo de prueba de validación pero que SI loguea al usuario creando una sesión para ese usuario.
// return (Auth::attempt($credenciales) ? 'Match': 'No match...');
// El parámetro true es para que deje al usuario logueado indefinidamente hasta que se desloguee manualmente.

if (Auth::attempt($credenciales,true))
{
return Redirect::to('users');
}
else
return "Acceso denegado, datos incorrectos";

}

// Página de logout.
public function getLogout()
{
Auth::logout();
return Redirect::to('users');
}

}
```

- Para probar si funciona la autenticación vamos a /login.

7.3 Modificación de una vista para mostrar si un usuario está o no logueado en el sistema

- Contenido de **resources/views/users.blade.php**:

```
@extends('layouts/master')

@section ('titulo')Usuarios @stop

@section('contenido')


@if (Auth::check())
Usuario Actual: {{ Auth::user()->username }}. {!! Html::link('logout','Desconectar') !!}
<hr/>
@endif
<h1>Listado de todos los usuarios</h1>
@foreach($users as $user)
<p>{{ $user->username }} (<a href='users/{{$user->id}}'>ver perfil</a>)</p>
@endforeach
@stop
```

7.4 Creación de Formulario de Login y modificación de HomeController

Vamos a completar la aplicación realizando un formulario de Login y gestionando los datos de ese formulario de acceso.

7.4.1 Creación de formulario de Login

- Contenido de **resources/views/login.blade.php** :

```
@extends('layouts.master')

@section ('titulo')Login @stop

@section('contenido')

@if (Input::old())
Error: datos de Acceso Incorrectos.
```

```
@endif

{!! Form::open(array('url'=>'login')) !!}

<p>
{!! Form::label('username','Usuario') !!}
{!! Form::text('username') !!}
</p>
<p>
{!! Form::label('password','Contraseña') !!}
{!! Form::password('password') !!}
</p>
<p>{!! Form::submit('Acceder') !!}</p>

{!! Form::close() !!}

@stop
```

7.4.2 Edición de HomeController para gestionar el formulario de Login

- Contenido de **app/Http/HomeController.php** :

```
<?php namespace App\Http\Controllers;
use Auth;
use Redirect;
use Request;

class HomeController extends Controller {

/*
        |--------------------------------------------------------------------------
        | Home Controller
        |--------------------------------------------------------------------------
        |
        | This controller renders your application's "dashboard" for users that
        | are authenticated. Of course, you are free to change or remove the
        | controller as you wish. It is just here to get your app started!
        |
        */

public function getIndex()
{
return Redirect::to('users');
}


// Al acceder a login mostramos un formulario de Login.
public function getLogin()
{
return view('login');
}

// Al recibir los datos del formulario de Login.
public function postLogin()
{
$credenciales=array(
'username'=>Request::input('username'),
'password'=>Request::input('password')
);

if (Auth::attempt($credenciales))
{
// En  lugar de redireccionarlo a una página en cuestión lo redireccionamos
// a la página a la cuál el usuario quería ir antes de estar autenticado en la aplicación.
// Esa página debería estar identificada en una variable de sesión.

// En el caso de que no ésté en la sesión especificada la URL a la que quería ir
// se le puede indicar por defecto la URL por defecto a la que ir: ('users') en este caso.
return Redirect::intended('users');
}
else
return Redirect::to('login')->withInput();
```

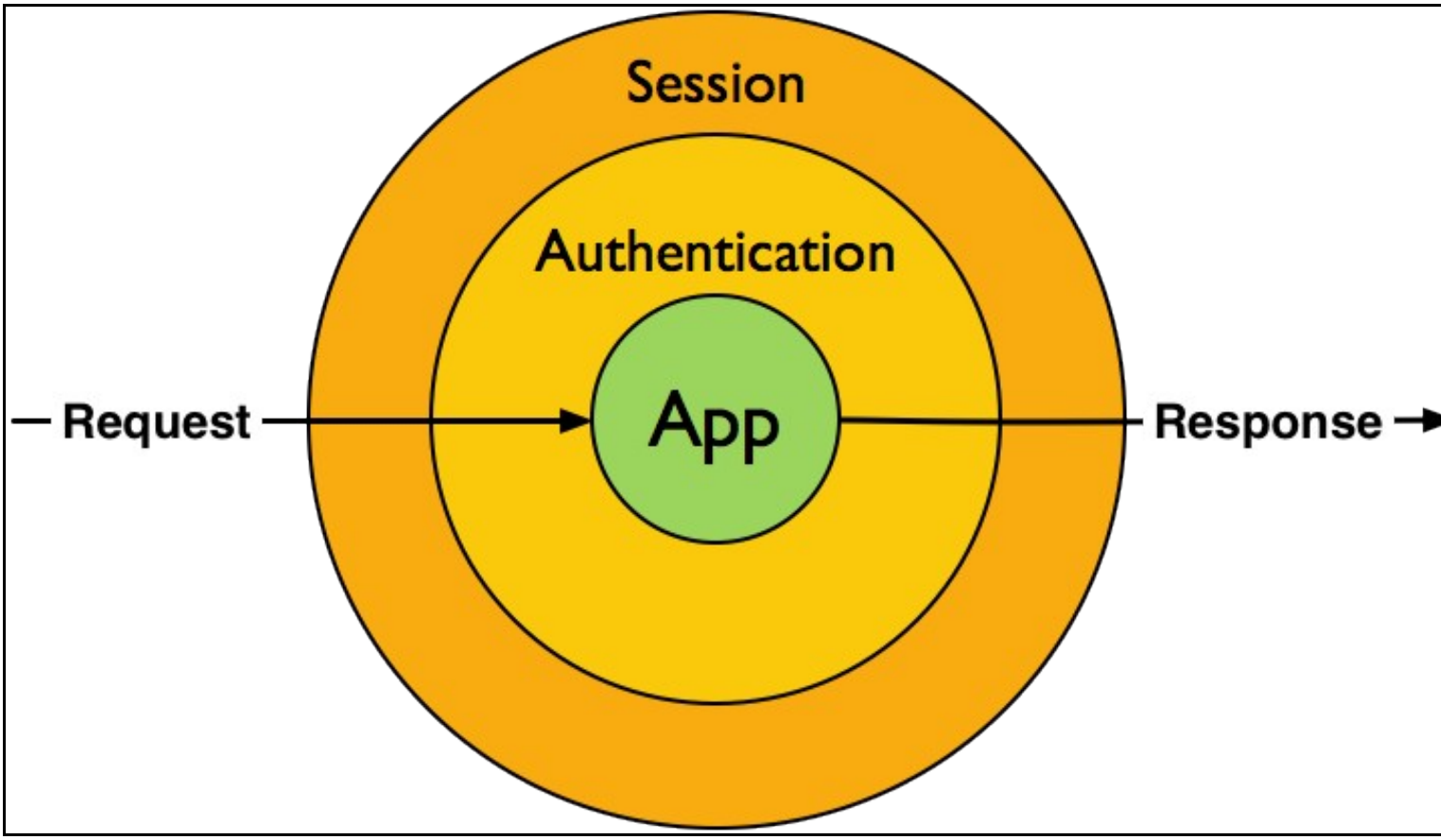
```
}


// Página de logout.
public function getLogout()
{
Auth::logout();
return Redirect::to('users');
}

}
```

7.5 Proteger la edición de perfiles por otros usuarios

- Si nos fijamos en **app/Http/Controllers/UsersController.php** podemos ver que en varios métodos tenemos el siguiente código, dónde comprobamos si un usuario existe y si no existe lo redireccionamos a Users:

```
if ($usuario== null)
        return Redirect::to('users');
```

- Podemos centralizar eso en un **middleware (Existe)** que se ejecute para los métodos **'show','edit','update' y 'destroy'**.
- También podemos crear otro **middleware (Propietario)** para comprobar si el usuario logueado coincide con el id del usuario que queremos gestionar. Lo haremos para los métodos **'edit','update' y 'destroy'**.



- Imagen extraída de http://StackPHP.com -

7.5.1 Creación de middlewares

- Los middleware se pueden generar desde:

```
php artisan make:middleware NombreMiddleware
```

- Generamos un nuevo **middleware app/Http/Middleware/Existe.php**:

```
php artisan make:middleware Existe
```

- Contenido del fichero **app/Http/Middleware/Existe.php**:

```
<?php namespace App\Http\Middleware;

use Illuminate\Contracts\Routing\Middleware;
use Closure;
use Redirect;
use App\User;

class Existe{
/**
         * Handle an incoming request.
         *
         * @param  \Illuminate\Http\Request  $request
         * @param  \Closure  $next
         * @return mixed
         */
public function handle($request, Closure $next)
{
// Parte del middleware dónde se modifica la solicitud entrante

// Aquí comprobaremos si el usuario que estamos comprobando existe.
$id=$request->segment(2);
$user= User::find($id);

// Si no existe el usuario que se pone como parámetro se manda a la página de users.
if ($user==null)
return Redirect::to('users');

// Si el usuario existe se envía a $next($request)
// que sería la parte de la aplicación a la que tendría que ir.
$respuesta=$next($request);


// Se devuelve la respuesta.
return ($respuesta);
}
}
```

- Generamos un nuevo **middleware app/Http/Middleware/Propietario.php**:

```
php artisan make:middleware Propietario
```

- Contenido del fichero **app/Http/Middleware/Propietario.php**:

```
<?php namespace App\Http\Middleware;

use Illuminate\Contracts\Routing\Middleware;
use Closure;
use Redirect;
use App\User;
use Auth;

class Propietario{
/**
         * Handle an incoming request.
         *
         * @param  \Illuminate\Http\Request  $request
         * @param  \Closure  $next
         * @return mixed
         */
public function handle($request, Closure $next)
{
// Parte del middleware dónde se modifica la solicitud entrante

$id=$request->segment(2);

// Comprobamos si el usuario logueado coincide con el users/id que tenemos en la ruta.
// Si no es así lo redireccionamos a users
if (Auth::user()->id !== (int) $id)
return Redirect::to('users');
```

```
// Si coincide se envía a $next($request)
// que sería la parte de la aplicación a la que tendría que ir.
$respuesta=$next($request);

// Se devuelve la respuesta.
return ($respuesta);
}
}
```

7.5.2 Registro de los middlewares

- Los middleware se pueden registrar **a nivel global** (se ejecutarían en cada petición de la aplicación) o sólo **a nivel de routing**.
- Para registrarlos a **nivel global** editar el fichero: **app/Http/Kernel.php** y añadir el middleware a **protected $middleware**.
- Para registrarlos a **nivel de routing** editar el fichero: **app/Http/Kernel.php** y añadir el middleware a **protected $routeMiddleware**.

- Nosotros los registraremos a **nivel de Routing**:

```
protected $routeMiddleware = [
'auth' => 'App\Http\Middleware\Authenticate',
'auth.basic' => 'Illuminate\Auth\Middleware\AuthenticateWithBasicAuth',
'guest' => 'App\Http\Middleware\RedirectIfAuthenticated',
'existe'=>'App\Http\Middleware\Existe',
'propietario'=>'App\Http\Middleware\Propietario',
];
```

7.5.3 Configuración de controlador para usar los middlewares anteriores

- A continuación configuraremos nuestro controlador para que use los middleware anteriores y realizaremos los cambios en los métodos correspondientes comentando el código que sobra.
- Hay que añadir use Auth; al fichero. Véase el código final del controlador.

- Contenido del fichero **app/Http/Controllers/UsersController.php**:

```
<?php namespace App\Http\Controllers;

use App\Http\Requests;
use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

// Indicamos que trabajamos con Vistas
use View;

// Indicamos que usamos el Modelo User.
use App\User;

// Validación de formularios.
use Validator;

// Hash de contraseñas.
use Hash;

// Redireccionamientos.
use Redirect;

// Auth.
use Auth;

class UsersController extends Controller {

public function __construct()
{
// Le indicamos que use los siguientes middleware en este controlador.
// Primero para chequear si un usuario existe /users/id
$this->middleware('existe',['only'=>['show','edit','update','destroy']]);

// Segundo para comprobar si el usuario conectado es el propietario
$this->middleware('propietario',['only'=>['edit','update','destroy']]);
}
```

```
/**
         * Display a listing of the resource.
         *
         * @return Response
         */
public function index()
{
// Prueba a ver si funciona la ruta /users
// return 'Lista de todos los usuarios';

// Devolvemos una Vista con toda la lista de usuarios.
// Usamos el método Mágico withUsers que lo que envía es una
// variable $users que contiene todos los usuarios.
return view('users')->withUsers(User::all());
}

/**
         * Show the form for creating a new resource.
         *
         * @return Response
         */
public function create()
{
return view('create');
}

/**
         * Store a newly created resource in storage.
         *
         * @return Response
         */
public function store(Request $request)
{
// Realizamos la validación de datos recibidos del formulario.
$rules=array(
'username'=>'required|unique:users', // Username es único en la tabla users
'email'=>'required|email|unique:users', // Username es único en la tabla users
'password'=>'required|min:6',
'password-repeat'=>'required|same:password'
);

// Llamamos a Validator pasándole las reglas de validación.
$validator=Validator::make($request->all(),$rules);

// Si falla la validación redireccionamos de nuevo al formulario
// enviando la variable Input (que contendrá todos los Input recibidos)
// y la variable errors que contendrá los mensajes de error de validator.
if ($validator->fails())
{
return Redirect::to('users/create')
->withInput()
->withErrors($validator->messages());
}

// Si la validación es OK, estamos listos para almacenar en la base de datos los datos.
User::create(array(
'username'=>$request->input('username'),
'email'=>$request->input('email'),
'password'=>Hash::make($request->input('password')),
'bio'=>$request->input('bio')
));

// Redireccionamos a users
return Redirect::to('users');
}

/**
         * Display the specified resource.
         *
         * @param  int  $id
         * @return Response
```

```
        */
public function show($id)
{
// Se muestra la información de un usuario.
// Comprobamos si el $id existe en la base de datos.
$usuario=User::find($id);

// Creamos una variable para pasarle a la vista 'perfil' e indicarle si nuestro id
// de usuarios logueados coincide con el de la URL.
// Devuelve true o false si el ID de la URL coincide con el id de la persona logueada.
// Auth::id() equivale a Auth::user()->id
$propietario= (Auth::id() === (int) $id);

// Con el middleware 'existe' activado ésto no hace falta:
/*
                if ($usuario== null)
                        return Redirect::to('users');
                */

return view('perfil')->withElusuario($usuario)->withPropietario($propietario);
}

/**
         * Show the form for editing the specified resource.
         *
         * @param  int  $id
         * @return Response
         */
public function edit($id)
{
// Se muestra la información de un usuario.
// Comprobamos si el $id existe en la base de datos.
$usuario=User::find($id);

// Con el middleware 'existe' activado ésto no hace falta:
/*
                if ($usuario== null)
                        return Redirect::to('users');
                */

return view('editar')->with('id',$id);
}

/**
         * Update the specified resource in storage.
         *
         * @param  int  $id
         * @return Response
         */
public function update($id,Request $request)
{
// Reglas de validación
$reglas = array(
'username' =>'unique:users',  // Deberá ser único en la tabla users
'email' =>'email|unique:users',  // Deberá ser único en la tabla users
'password' => 'min:6'
);

$validator= Validator::make($request->all(),$reglas);

if ($validator->fails())
{
return Redirect::to('users/'.$id.'/edit')
->withInput()
->withErrors($validator->messages());
}

$usuario = User::find($id);

if ($request->input('username'))
$usuario->username=$request->input('username');
if ($request->input('email'))
$usuario->email=$request->input('email');
```

```
if ($request->input('bio'))
$usuario->bio=$request->input('bio');
if ($request->input('password'))
$usuario->password=Hash::make($request->input('password'));

// Grabamos el usuario en la tabla.
$usuario->save();

// Redireccionamos a la página personal del usuario.
return Redirect::to('users/'.$id);
}


/**
         * Remove the specified resource from storage.
         *
         * @param  int  $id
         * @return Response
         */
public function destroy($id)
{
$usuario = User::find($id);
$usuario->delete();

return Redirect::to('users');
}

}
```

7.5.4 Modificación de plantilla perfil.blade.php

- Vamos a modificar la plantilla perfil.blade.php para que se muestre el hiperenlace Editar mi perfil, solamente si somos los usuarios logueados y propietarios de nuestro perfil.

- Contenido del fichero **resources/views/perfil.blade.php**:

```
@extends('layouts/master')

@section ('titulo')Usuario {{$elusuario->username}} @stop

@section('contenido')
<h2>Usuario: {{$elusuario->username}}</h2>
<p>E-mail: {{$elusuario->email}}</p>
<p>Biografía: {{$elusuario->bio}}</p>

@if ($propietario)
<a href="/users/{{$elusuario->id}}/edit">Editar mi perfil</a>
@endif

| <a href="/users">Volver</a>
@stop
```

7.6 Creación de una página adicional con información para miembros registrados

- **Para terminar** vamos a crear una página para usuarios registrados que muestre información específica y que necesite que el usuario se loguee previamente.
- Creamos una nueva ruta **getMiembros** en **app/Http/Controllers/HomeController.php**.
- En el **constructor de HomeController** tenemos que indicarle que use el **middleware auth** (instalado por defecto) para dicha página.

- Contenido de **app/Http/Controllers/HomeController.php**:

```
<?php namespace App\Http\Controllers;
use Auth;
use Redirect;
use Request;

class HomeController extends Controller {

/*
```

```
        |--------------------------------------------------------------------------
        | Home Controller
        |--------------------------------------------------------------------------
        |
        | This controller renders your application's "dashboard" for users that
        | are authenticated. Of course, you are free to change or remove the
        | controller as you wish. It is just here to get your app started!
        |
        */

public function __construct()
{
// Este middleware se llamará solamente para la página miembros.
$this->middleware('auth',array('only'=>'getMiembros'));
}

public function getIndex()
{
return Redirect::to('users');
}


// Al acceder a login mostramos un formulario de Login.
public function getLogin()
{
return view('login');
}

// Al recibir los datos del formulario de Login.
public function postLogin()
{
$credenciales=array(
'username'=>Request::input('username'),
'password'=>Request::input('password')
);

if (Auth::attempt($credenciales))
{
// En  lugar de redireccionarlo a una página en cuestión lo redireccionamos
// a la página a la cuál el usuario quería ir antes de estar autenticado en la aplicación.
// Esa página debería estar identificada en una variable de sesión.

// En el caso de que no ésté en la sesión especificada la URL a la que quería ir
// se le puede indicar por defecto la URL por defecto a la que ir: ('users') en este caso.
return Redirect::intended('users');
}
else
return Redirect::to('login')->withInput();
}


// Página de logout.
public function getLogout()
{
Auth::logout();
return Redirect::to('users');
}

// Página para "sólo miembros" de la aplicación.
public function getMiembros()
{
// Podemos devolver una vista.
// En este caso devolvemos un texto.
return "<h1>Noticias para miembros de la Aplicación.</h1><p>Se les comunica que las cuotas de Noviembre ya están disponibles
para su pago en la cuenta 1234 8558 7887 6789.</p><p>Sin otro particular, reciban un cordial saludo.</p><p>La Dirección.</p>";
}
}
```

- Cuando alguien quiera entrar en la página /**miembros** si no está logueado le pedirá autenticarse, pero para que lo haga a través de nuestra página y no utilizando la **vista auth/login de Laravel**, editaremos el fichero **app/Http/Middleware/Authenticate.php** para que redireccione a la vista **login.blade.php** y no **auth/login.blade.php**:

```
<?php namespace App\Http\Middleware;

use Closure;
use Illuminate\Contracts\Auth\Guard;

class Authenticate {

/**
         * The Guard implementation.
         *
         * @var Guard
         */
protected $auth;

/**
         * Create a new filter instance.
         *
         * @param  Guard  $auth
         * @return void
         */
public function __construct(Guard $auth)
{
$this->auth = $auth;
}

/**
         * Handle an incoming request.
         *
         * @param  \Illuminate\Http\Request  $request
         * @param  \Closure  $next
         * @return mixed
         */
public function handle($request, Closure $next)
{
if ($this->auth->guest())
{
if ($request->ajax())
{
return response('Unauthorized.', 401);
}
else
{
return redirect()->guest('login');
}
}

return $next($request);
}

}
```

- **A partir de este momento cuando vayamos a** /miembros **se solicitará autenticación (si no estamos logueados) para ver el contenido de dicha página.**

--Veiga (discusión) 00:08 7 may 2015 (CEST)