Acerca del proyecto final

Este proyecto final contiene todo lo visto durante la capacitación de Laravel nivel intermedio-preavanzado

En los siguientes enlaces se encuentran los repositorios de la capacitación de php y laravel:

- PHP.
- Laravel capacitación.

Levantar nuestro proyecto base y configurarlo

Se genera un nuevo proyecto laravel con el siguiente comando:

Laravel new laravelfp

Una vez instalado los componentes y el proyecto como tal, se procede a crear una base de datos con el nombre:

laravelfp

Configurando vistas

Procedemos a realizar las migraciones base

php artisan migrate

Procedemos a utilizar el componente de autorización

php artisan make:auth

Ahora creamos nuestra área de administrador:

- **Crear nueva carpeta admin**
  - **index.blade.php**
- **Crear nueva carpeta posts**
  - **create.blade.php**
  - **edit.blade.php**
  - **index.blade.php**
- **Crear nueva carpeta users**
  - **create.blade.php**
  - **edit.blade.php**
  - **index.blade.php**
- **Crear nueva carpeta categories**

- **create.blade.php**
- **index.blade.php**
- **edit.blade.php**

Migración de la tabla **users**

php atisan make:migration add_role_id_to_users_table --table=users

Up

Schema::table('users', function (Blueprint $table) { // $table->integer("role_id")->index()->unsigned()->nullable(); $table->integer('is_active')->default(0); });

Down

Schema::table('users', function (Blueprint $table) { // $table->dropColumn('role_id'); $table->dropColumn('is_active'); });

Se procede a generar la migracion de la tabla de roles y su modelo

php artisan make:model Role -m

Up

Schema::create('roles', function (Blueprint $table) { $table->increments('id'); $table->string('name'); $table->timestamps(); });

Configuración de relación y entrada de datos

Se configura la relación en el model de User

Up

public function role(){ return $this->belongsTo('App\Role'); }

Se verifica su funcionamiento en /home registrando un nuevo usuario

Probando relacion con Tinker

Activamos la interfaz de tinker con el siguiente comando:

php artisan tinker

Generamos un nuevo objeto

$user=App\User::find(1)

Podemos acceder ahora a la relación con Role

$user->role

Nos debe generar como salida la relación

App\User::create(['name'=>'Consultant','email'=>'amartinezb@advanced-consulting.biz','password'=>'Progangelo1'])

Cerramos Tinker

Controlador administrativo y rutas

Generamos la ruta prefabricada de Laravel

Route::resource('admin/users', 'UserController');

Generamos el controlador para nuestra nueva ruta con artisan

php artisan make:controller --resource AdminUsersController

Instalación de NodeJs

Descargamos el binario para la arquitectura que nos corresponde y se instala como cualquier binario de windows

Se procede a verificar si se instalo correctamente con el siguiente comando:

node -v

Intalación de gulp y assets

Se ejecuta el siguiente comando:

npm install --global gulp npm install --save-dev gulp

npm install

npm install -g gulp

npm install laravel-elixir --save-dev

npm install laravel-elixir-vue-2 --save-dev

npm install --save laravel-elixir-webpack-official

Si existe algún problema seguimos con la guía oficial de Gulp

Se procede a crear un archivo gulpfile.js

Si se crea un contenido por defecto se comenta y se pone el siguiente:

**var elixir=require('laravel-elixir');

elixir(function(mix){ mix.sass('app.scss') });**

Se procede a descargar los assets correspondientes

- Assets.

Dentro traen 3 carpetas:

Se colocan en la carpeta resources/assets

- **css**
- **js**

Se coloca dentro de la carpeta public

- **fonts**

Generamos dos métodos dentro de nuestro archivo gulpfile.js

```
.styles([
        'libs/blog-post.css',
        'libs/bootstrap.css',
        'libs/font-awesome.css',
        'libs/metisMenu.css',
        'libs/sb-admin-2.css'
    ], './public/css/libs.css')

    .scripts([
        'libs/jquery.js',
        'libs/bootstrap.js',
        'libs/metisMenu.js',
        'libs/scripts.js',
        'libs/sb-admin-2.js'
    ], './public/js/libs.js')
```

Ejecutamos gulp para revisar que todo este correcto

gulp

Creando el master page del area Administrable

Descargamos la plantilla base y la colocamos en el index del area admin

- Layout Administrador

Generamos una nueva ruta

```
Route::get('/admin', function($id) {
    //
    return view('admin.index');
})->name('admins');
```

Modificando masterpage administrable para corregir el body

Se modifica el archivo /resources/assets/sass/app.scss

```
#admin-page{
    padding-top:0px;
}
```

Se ejecuta **gulp** para actualizar app.css

Listando usuarios

modificamos el controlador de User

```
use App\User;
 public function index()
    {
        //
        // Obtenemos a los usuarios
         $users = User::all();
        return view('admin.users.index',compact('users'));
    }
```

modificamos la vista

```
 @if($users)
    @foreach($users as $user)
     <tr>
        <td>{{$user->id}}</td>
        <td>{{$user->name}}</td>
        <td>{{$user->email}}</td>
        {{-- <td>{{$user->created_at}}</td>
        <td>{{$user->updated_at}}</td> --}}
        {{-- Con formato para humanos --}}
        <td>{{$user->created_at->diffForHumans()}}</td>
        <td>{{$user->updated_at->diffForHumans()}}</td>
      </tr>
    @endforeach
    @endif
```

Modificando el index a un mejor formato

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

@section('content')
<h1>User</h1>
```

```
<table class="table">
    <thead>
      <tr>
        <th>Firstname</th>
        <th>Lastname</th>
        <th>Email</th>
        <th>Role</th>
        <th>Active</th>
        <th>Created</th>
        <th>Updated</th>
      </tr>
    </thead>
    <tbody>
    @if($users)
    @foreach($users as $user)
     <tr>
        <td>{{$user->id}}</td>
        <td>{{$user->name}}</td>
        <td>{{$user->email}}</td>
        {{-- Nota:Si esta nulo el id de referencia manda una excepción --}}
        <td>{{$user->role->name}}</td>
        {{-- Nota:Ejecutando operador ternario para condición --}}
        <td>{{$user->is_active ==1 ? 'Active' : 'Inactive'}}</td>
        {{-- <td>{{$user->created_at}}</td>
        <td>{{$user->updated_at}}</td> --}}
        {{-- Con formato para humanos --}}
        <td>{{$user->created_at->diffForHumans()}}</td>
        <td>{{$user->updated_at->diffForHumans()}}</td>
      </tr>
    @endforeach
    @endif
    </tbody>
  </table>

@endsection()
```

Creando usuarios e integrando el motor de vistas

modificamos el controlador de User en la acción create

```
public function create()
    {
        //
        return view('admin.users.create');
    }
```

modificamos la vista admin.users.create

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

@section('content')
<h1>Create users</h1>
@endsection()
```

descargamos con composer el empaquetado colectivo de html

```
composer require laravelcollective/html
```

Realizamos la configuración correspondiente en congfig/app.php

en los providers se añade el collective

```
Collective\Html\HtmlServiceProvider::class
```

Procedemos a generar un formulario base para crear usuarios

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

@section('content')


    <h1>Create Users</h1>


     {!! Form::open(['method'=>'POST', 'action'=>
'AdminUsersController@store']) !!}


      <div class="form-group">
            {!! Form::label('name', 'Name:') !!}
            {!! Form::text('name', null, ['class'=>'form-control'])!!}
       </div>

         <div class="form-group">
            {!! Form::submit('Create User', ['class'=>'btn btn-primary']) !!}
         </div>

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


    {{-- @include('includes.form_error') --}}
```

```
@stop
```

Creando campos y probando el formulario

En el controlador permitimos obtener todo el request y presentarlo

```
public function store(Request $request)
    {
        //
        return $request->all();
    }
```

Acción create

```
 public function create()
    {
        //Obtenemos todos los roles y enviamos en un viewbag
        $roles = Role::pluck('name','id')->all();
        // $roles = array_pluck(, 'developer.name', 'developer.id');
        return view('admin.users.create',compact('roles'));
    }
```

Vista admin/users/create.blade.php

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



@section('content')


    <h1>Create Users</h1>


     {!! Form::open(['method'=>'POST', 'action'=>
'AdminUsersController@store','files'=>true]) !!}


      <div class="form-group">
            {!! Form::label('name', 'Name:') !!}
            {!! Form::text('name', null, ['class'=>'form-control'])!!}
       </div>
```

```
          <div class="form-group">
         {!! Form::label('email', 'Email:') !!}
         {!! Form::email('email', null, ['class'=>'form-control'])!!}
        </div>

         <div class="form-group">
             {!! Form::label('role_id', 'Role:') !!}
             {!! Form::select('role_id', [''=>'Choose Options'] + $roles ,
null, ['class'=>'form-control'])!!}
         </div>


         <div class="form-group">
             {!! Form::label('is_active', 'Status:') !!}
             {!! Form::select('is_active', array(1 => 'Active', 0=> 'Not
Active'), 0 , ['class'=>'form-control'])!!}
          </div>


         <div class="form-group">
             {!! Form::label('photo_id', 'Photo:') !!}
             {!! Form::file('photo_id', null, ['class'=>'form-control'])!!}
          </div>



         <div class="form-group">
             {!! Form::label('password', 'Password:') !!}
             {!! Form::password('password', ['class'=>'form-control'])!!}
          </div>


          <div class="form-group">
             {!! Form::submit('Create User', ['class'=>'btn btn-primary']) !!}
          </div>


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



  @stop
```

Campo de validacion y generar un request customizable

Generamos un request para mediante artisan

```
php artisan make:request UsersCreateRequest
```

modificamos el valor de retorno de authorize a true

```
 public function authorize()
    {
        return true;
    }
```

ponemos las reglas, para que todos los campos sean requeridos

```
public function rules()
    {
        return [
            //
            'name'=>'required',
            'email'=>'required',
            'role_id'=>'required',
            'is_active'=>'required',
            'password'=>'required'
        ];
    }
```

en el controlador AdminUsersController.php cambiamos el request por default al nuestro

```
public function store(UsersRequest $request)
    {
        //
        return $request->all();
    }
```

Se genera un nuevo usuario y observamos como recarga la página y no recupera los datos. Esto se debe a que activamos la validación.

Mostrando errores e incluyendolo con blade

Verificamos que exista errores en el request, si los hay lo despliega como una alerta.

```
  @if(count($errors)>0)
    <div class="alert alert-danger">
    <ul>
    @foreach($errors->all() as $error)
    <li>
    {{$error}}
    </li>
    @endforeach
```

```
    </ul>
    </div>
    @endif
```

Para mejorar el performance se genera un nuevo archivo como vista parcial de errores

includes/form_error.blade.php

dentro de la vista...

```
@include('includes.form_error')
```

dentro del form_error

```
 @if(count($errors)>0)
    <div class="alert alert-danger">
    <ul>
    @foreach($errors->all() as $error)
    <li>
    {{$error}}
    </li>
    @endforeach
    </ul>
    </div>
    @endif
```