Pla de Recuperació, Transformació i Resiliència
FPCV Formació Professional Comunitat Valenciana
Unió Europea
Fons social europeu
L'FSE inverteix en el teu futur
GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura, Universitats i Ocupació
IES MARIA ENRIQUEZ

Unitat 5. Frameworks. Laravel

2n DAW - IES María Enríquez

Part 2

Resultats d’aprenentatge i criteris d’avaluació

RA 5. Desenvolupa aplicacions Web identificant i aplicant mecanismes per a separar el codi de presentació de la lògica de negoci.

Criteris d'avaluació:

a) S'han identificat els avantatges de separar la **lògica de negoci** dels aspectes de presentació de l'aplicació.

b) S'han analitzat **tecnologies i mecanismes** que permeten realitzar aquesta separació i les seues característiques principals.

c) S'han utilitzat **objectes i controls en el servidor** per a generar l'aspecte visual de l'aplicació web en el client.

d) S'han utilitzat **formularis generats de manera** dinàmica per a respondre als esdeveniments de l'aplicació Web.

e) S'han identificat i aplicat els **paràmetres relatius a la configuració** de l'aplicació Web.

f) S'han escrit aplicacions Web amb **manteniment d'estat i separació de la lògica** de negoci.

g) S'han aplicat els principis de la programació **orientada a objectes**.

h) S'ha provat i **documentat** el codi.

Resultats d'aprenentatge i criteris d'avaluació

RA8. Genera pàgines web dinàmiques analitzant i utilitzant tecnologies i frameworks del servidor web que afigen codi al llenguatge de marques.

Criteris d'avaluació:

a) S'han identificat les diferències entre l'**execució de codi en el servidor i en el client web**.

b) S'han reconegut els avantatges d'**unir totes dues tecnologies** en el procés de desenvolupament de programes.

c) S'han identificat les **tecnologies i frameworks relacionades amb la generació per part del servidor** de pàgines web amb guions embeguts.

d) S'han utilitzat aquestes **tecnologies i frameworks per a generar pàgines web** que incloguen **interacció** amb **l'usuari**.

e) S'han utilitzat aquestes tecnologies i frameworks, per a generar pàgines web que incloguen **verificació** de formularis.

f) S'han utilitzat aquestes tecnologies i frameworks per a generar pàgines web que incloguen **modificació dinàmica del seu contingut** i la seua estructura.

g) S'han aplicat aquestes tecnologies i frameworks en la **programació d'aplicacions web**.

Unitat 5. Frameworks. Laravel

2. Rutes i vistes

    2.1. Rutes amb Laravel

    2.2. Vistes amb Blade

    2.3. Estils i javascript

2. Rutes i vistes

Rutes en Laravel

Les rutes estableixen quina resposta donar a una petició.

```
/routes/

    web.php

    api.php
```

La funció **`get`** té dos paràmetres: la **ruta** i la **funció** que ha d'executar.

```
use Illuminate\Support\Facades\Route;

Route::get('/', function() {
    return view('welcome');
});
```

Rutes en Laravel

Rutes simples

Són aquelles que al accedir a una ruta ens retorna una resposta:

```
Route::get('fecha', function() {
    return date("d/m/y h:i:s");
});
```

Si ara accedim a http://127.0.0.1:8000/fecha ens retornarà la data i hora actual

Rutes en Laravel

Rutes amb paràmetres

És possible enviar paràmetres mitjançant la URL. Per capturar-los hem de fer-ho així:

```
Route::get('saludo/{nombre}', function($nombre) {
   return "Hola, " . $nombre;
});
```

En aquest cas el paràmetre és obligatori, si no el posem, ens donarà un error 404.

Rutes en Laravel

Rutes amb paràmetres

Per fer-ho opcional, hem de posar un **?**:

```
Route::get('saludo/{nombre?}', function($nombre = "Invitado") {
    return "Hola, " . $nombre;
});
```

Rutes en Laravel

Validació de paràmetres

Hi ha paràmetres que necessitem que tinguen uns **requisits**, per exemple, una ID hauria de ser un valor numèric.

Per fer-ho afegim la clausula **`where`**. Li pasem el **paràmetre** a avaluar i l'**expressió regular** per fer-ho.

```
Route::get('saludo/{nombre?}', function($nombre = "Invitado") {
    return "Hola, " . $nombre;
})->where('nombre', "[A-Za-z]+");
```

Rutes en Laravel

Rutes amb noms o *named routes*

En ocasions es molt útil posar-li un nom a les rutes, especialment quan forma part d'un enllaç i modificar-la suposaria modificar tots els llocs on apareix.

Per fer-ho, utilitzarem la funció **name**

```
Route::get('contacto', function() {
   return "Página de contacto";
})->name('ruta_contacto');
```

Rutes en Laravel

Rutes amb noms o *named routes*

Ara, si volem utilizar la ruta com a contacte, en lloc de posar açò:

```
echo '<a href="/contacto">Contacto</a>';
```

Posarem açò:

```
<a href="{{ route('ruta_contacto') }}">Contacto</a>
```

Rutes en Laravel

Combinació d'elements en rutes

Podem combinar diferents condicions where així com afegir un name a una mateixa ruta:

```
Route::get('saludo/{nombre?}/{id?}',
function($nombre="Invitado", $id=0)
{
   return "Hola $nombre, tu código es el $id";
})->where('nombre', "[A-Za-z]+")
  ->where('id', "[0-9]+")
  ->name('saludo');
```

URL	Respuesta
/saludo	*Hola Invitado, tu código es el 0*
/saludo/Nacho	*Hola Nacho, tu código es el 0*
/saludo/Nacho/3	*Hola Nacho, tu código es el 3*
/saludo/3	Error 404 (URL incorrecta)

Rutes en Laravel

Altres métodes de Route

Hem vist en els exemples anteriors l'ús de **get**.

Existeixen altres mètodes estàtics com **`Route::post`**, **`Route::put`** or **`Route::delete`** que veurem més endavant.

```
Route::get('saludo/{nombre?}/{id?}',
function($nombre="Invitado", $id=0)
{
   return "Hola $nombre, tu código es el $id";
})->where('nombre', "[A-Za-z]+")
  ->where('id', "[0-9]+")
  ->name('saludo');
```

Vistes en Laravel

Altres métodes de Route

Hem vist en els exemples anteriors l'ús de get.

Existeixen altres mètodes estàtics com `Route::post`, `Route::put` or `Route::delete` que veurem més endavant.

Vistes en Laravel

En els exemples vistos fins ara, les routes retornaven un text. El més normal és que retornen un HTML.

Fer-ho des de la ruta no és habitual ni recomanable.

Per fer-ho tenim les vistes que estan ubicades en **`resources/views`**.

```
Route::get('/', function() {
    return view('welcome');  //retorna la vista welcome
});
```

Vistes en Laravel

Podem definir la nostra pàgina d'inici de la següent forma:

1. En **`resources/views`** creem l'arxiu **`inicio.blade.php`**:

```
<html>
…
</html>
```

2. A continuació, definim la ruta:

```
Route::get('/', function() {
    return view('inicio');
});
```

Vistes en Laravel

Passar valors a les vistes

És possible passar valors a les llistes amb el mètode **`with`**, indicant el **`nom`** de la variable i el **`valor`**:

```
Route::get('/', function() {
    $nombre = "Nacho";
    return view('inicio')->with('nombre', $nombre);
});
```

Vistes en Laravel

Passar valors a les vistes

Posteriorment, en el arxiu php utilitzarem la variable de la següent forma:

```
<html>
    <head>
        <title>Inicio</title>
    </head>
    <body>
        <h1>Página de inicio</h1>
        <p>Bienvenido/a <?php echo $nombre; ?></p>
    </body>
</html>
```

Vistes en Laravel

Passar valors a les vistes

Alternativament al mètode anterior, també podem:

Passar un **array associatiu** de valors:

```
return view('inicio')->with(['nombre' => $nombre, ...]);
```

Passar l'array com un **segon argument de view**:

```
return view('inicio', ['nombre' => $nombre, ...]);
```

Vistes en Laravel

Passar valors a les vistes

Alternativament al mètode anterior, també podem:

Quan el nom de la variable i el paràmetre coincideixen, podem utilitzar la funció compact()

```
return view('inicio', compact('nombre', ...));
```

Si anem a tornar una vista amb poca lògica interna, podem retornar directament la la vista en la ruta:

```
Route::view('/', 'inicio', ['nombre' => 'Nacho']);
```

Vistes en Laravel

Primers passos amb el motor de plantilles Blade

Per poder fer ús de la potència del motor Blade, les nostres pàgines hauran de tenir una extensió `.blade.php`

Una de les coses que permet és substituir l'ús de:

```
Bienvenido/a <?php echo $nombre ?>
```

Per:

```
Bienvenido/a {{ $nombre }}
```

Important: per evitar atacs XSS quan es renderitza la pàgina s'utiliza una funció intermitja anomenada *e*. Si no volem que l'utilitze escriurem `{!! $nombre !!}`

Vistes en Laravel

Estructures de control de fluxe amb Blade

Foreach: similar a PHP

```
<ul>
    @foreach($elementos as $elemento)
        <li>{{ $elemento }}</li>
    @endforeach
</ul>
```

Vistes en Laravel

Estructures de control de fluxe amb Blade

Si volem comprovar si l'array està buit, podem utilitzar la directiva @if

```
<ul>
    @if($elementos)
        @foreach($elementos as $elemento)
            <li>{{ $elemento }}</li>
        @endforeach
    @else
        <li>No hay elementos que mostrar</li>
    @endif
</ul>
```

Vistes en Laravel

Estructures de control de fluxe amb Blade

També es pot comprovar si la variable està definida amb @isset

```
<ul>
    @isset($elementos)
        @foreach($elementos as $elemento)
            <li>{{ $elemento }}</li>
        @endforeach
    @else
        <li>No hay elementos que mostrar</li>
    @endisset
</ul>
```

Vistes en Laravel

Estructures de control de fluxe amb Blade

En els dos exemples anteriors hi ha un problema. En el primer cas, si la variable no està definida, error. En el segon, si l'array no té elements, no mostra res.

Solució: @forelse

```
<ul>
    @forelse($elementos as $elemento)
        <li>{{ $elemento }}</li>
    @empty
        <li>No hay elementos que mostrar</li>
    @endforelse
</ul>
```

Vistes en Laravel

Estructures de control de fluxe amb Blade

En els iteradors anteriors, tenim un objecte especial `$loop` que té algunes propietats interessants com:

- `index`: indica la posició
- `count`: nombre total d'elements
- `first i last`: indiquen si és el primer o l'últim element respectivament

Podem veure totes les propietats de `$loop` amb `var_dump`

```
<ul>
    @forelse($elementos as $elemento)
        <li>{{ $elemento }} {{ var_dump($loop) }} </li>
    @empty
        <li>No hay elementos que mostrar</li>
    @endforelse
</ul>
```

Vistes en Laravel

Exemple

Continuant amb l'exemple de la biblioteca anem a fer que al accedir a la ruta listado, ens retorne la vista `listado.blade.php`

```
Route::get('listado', function() {
    $libros = array(
        array("titulo" => "El juego de Ender",
              "autor" => "Orson Scott Card"),
        array("titulo" => "La tabla de Flandes",
              "autor" => "Arturo Pérez Reverte"),
        array("titulo" => "La historia interminable",
              "autor" => "Michael Ende"),
        array("titulo" => "El Señor de los Anillos",
              "autor" => "J.R.R. Tolkien")
    );

    return view('listado', compact('libros'));
})->name('listado_libros');
```

Vistes en Laravel

Exemple

En `listado.blade.php`quedarà així:

```
<html>
    <head>
        <title>Listado de libros</title>
    </head>
    <body>
        <h1>Listado de libros</h1>
        <ul>
        @forelse ($libros as $libro)
            <li>{{ $libro["titulo"] }} ({{ $libro["autor"] }})</li>
        @empty
            <li>No se encontraron libros</li>
        @endforelse
        </ul>
    </body>
</html>
```

Vistes en Laravel

Exemple

Per últim, en la pàgina d'inici (inicio.blade.php) posarem un enllaç al llistat de llibres:

```
<p>Bienvenido/a {{ $nombre }}</p>
<p><a href="{{ route('listado_libros') }}">Listado de libros</a></p>
```

Vistes en Laravel

Definir plantilles comuns

Per poder utilitzar una pàgina com a plantilla, crearem la pàgina plantilla.blade.php. Aquelles parts que el contingut siga variable, utilitzarem la secció `@yield`

```
<html>
    <head>
        <title>
        @yield('titulo')
        </title>
    </head>
    <body>
        <nav>
            <!-- ... Menú de navegación -->
        </nav>
        @yield('contenido')
    </body>
</html>
```

Vistes en Laravel

Definir plantilles comuns

Ara, en cada vista que volem fer ús de la plantilla utilitzarem la directiva **@extends**. Amb la directiva **@section** i el nom de la secció definim cadascú dels **@yields** de la plantilla.

```
@extends('plantilla')

@section('titulo', 'Inicio')

@section('contenido')
    <h1>Página de inicio</h1>
    <p>Bienvenido/a {{ $nombre }}</p>
@endsection
```

Vistes en Laravel

Definir plantilles comuns

Per a l'exemple del llistat de llibres quedaria així:

```
@extends('plantilla')

@section('titulo', 'Listado de libros')

@section('contenido')
    <h1>Listado de libros</h1>
    <ul>
    @forelse ($libros as $libro)
        <li>{{ $libro["titulo"] }} ({{ $libro["autor"] }})</li>
    @empty
        <li>No se encontraron libros</li>
    @endforelse
    </ul>
@endsection
```

Vistes en Laravel

Afegir vistes dins d'altres

Podem definir continguts parcials. Per fer-ho, normalment es fa en el directori `partials` (`resources/views/partial`)

Per crear la barra de navegació crearíem `resources/views/partial/nav.blade.php`

```
<nav>
    <a href="{{ route('inicio') }}">Inicio</a>
     |
    <a href="{{ route('listado_libros') }}">Listado de libros</a>
</nav>
```

Vistes en Laravel

Afegir vistes dins d'altres

Per afegir el menú de navegació en la plantilla utilitzarem la directiva @include. Faríem el següent:

```
<html>
    <head>
        <title>
        @yield('titulo')
        </title>
    </head>
    <body>
        @include('partials.nav')
        @yield('contenido')
    </body>
</html>
```

Vistes en Laravel

Estructurar vistes en carpetes

Quan l'aplicació és complexa, cal tenir una bona organització de plantilles. En el cas de la biblioteca, podríem crear la subcarpeta `libros` dins de `resources/views`.

Si ara ubiquem la vista listado.blade.php dins de la capeta libros, haurem d'actualitzar la vista així:

```
Route::get('listado', function() {
    ...
    return view('libros.listado', compact('libros'));
})->name('listado_libros');
```

Vistes en Laravel

Vistes per a pàgines d'error

Per definir pàgines d'error personalitzades, haurem d'afegir les vistes a resources/views/errors, per exemple: resources/views/errors/404.blade.php

```
@extends('plantilla')

@section('titulo', 'Error 404')

@section('contenido')
    <h1>Error</h1>
    <p>Documento no encontrado</p>
@endsection
```

Estils i Javascript

Infraestructura per arxius CSS i Javascript

Les dependències de llibreries Javascript estan definides a l'arxiu package.json. Algunes d'elles ja estan definides, com vite.

Quan creem el projecte és important tenir-les disponibles amb aquest comandament:

```
> npm install
```

Crearà una carpeta **node_modules** amb les dependències instal·lades. Similar a la carpeta **vendor** amb les dependències per a PHP.

Aquestes carpetes no haurien de pujar-se al repositori git.

Estils i Javascript

Infraestructura per arxius CSS i Javascript

Pel que fa al CSS tenim el directori `/resources/css/app.css` on podrem afegir els nostres estils o incloure noves llibreries externes.

```
body
{
    background-color: #CCC;
    font-family: Arial;
    text-align: justify;
}
```

Estils i Javascript

Generació automàtica de CSS i JS

Els arxius vistos anteriorment necessiten ser processats per afegir el codi a la nostra aplicació.

L'encarregat de fer-ho és l'eina vite, a l'arxiu vite.config.js.Ací haurem d'afegir els arxius ue necessitem processar:

```
export default defineConfig({
    plugins: [
        laravel({
            input: ['resources/css/app.css', 'resources/js/app.js'],
            refresh: true
        }),
    ],
});
```

Estils i Javascript

Generació automàtica de CSS i JS

A continuació, en les nostres vistes haurem d'afegir els arxius així:

```
<head>
    ...
    @vite(['resources/css/app.css', 'resources/js/app.js'])
</head>
```

Per últim, executarem el comandament:

```
> npm run build
```

Estils i Javascript

Afegir estils Bootstrap

Per poder utilitzar aquesta famosa llibreria al nostre projecte hem d'utilitzar una llibreria del servidor anomenda ui.

```
> composer require laravel/ui:*
```

Nota: * última versió

Una vegada afegida l'eina, utilitzem artisan per afegir-la al projecte Bootstrap

```
> php artisan ui bootstrap
```

Açò afegirà Bootstrap a l'arxiu package.json

Estils i Javascript

Afegir estils Bootstrap

A partir d'aquest moment, haurem d'afegir tots els nostres estils CSS a l'arxiu `app.scss`

```
 @import '~bootstrap/scss/bootstrap';

body
{
    background-color: #CCC;
    font-family: Arial;
    text-align: justify;
}
```

Estils i Javascript

Afegir estils Bootstrap

També haurem d'actualitzar totes les vistes que utilitzaven la directiva @vite

```
<!doctype html>
<head>
    ...
    @vite(['resources/sass/app.scss', 'resources/js/app.js'])
</head>
```

Per finalitzar

```
> npm install
> npm run build
```

Exercicis vistes i rutes

Exercici 1

Sobre el projecte blog d'exercicis anteriors, edita el fitxer routes/web.php i afig una nova ruta a la URL posts. En accedir a aquesta ruta (http://127:0.0.1:8000/posts), haurem de veure un missatge amb el text “Llistat de posts”.

Exercicis vistes i rutes

Exercici 2

Sobre el projecte blog anterior, afegirem aquests dos canvis:

- Afig una nova ruta parametritzada a posts/{id}, de manera que el paràmetre id siga numèric (és a dir, només continga dígits del 0 al 9) i obligatori. Fes que la ruta retorne el missatge “Fitxa del post XXXX”, sent XXXX la id que haja rebut com a paràmetre.

- Posa un nom a les tres rutes que hi ha definides fins ara: a la pàgina d'inici posa-li el nom “inici”, a la del llistat li direm “posts_llistat” i a la de fitxa que acabes de crear, li direm “posts_fitxa”.

Exercicis vistes i rutes

Exercici 3

Continuem amb el projecte blog. En aquest cas definirem una plantilla i una sèrie de vistes que la utilitzen.

- Començarem definint una plantilla anomenada `plantilla.blade.php` en la carpeta de vistes del projecte (`resources/views`). Defineix una capçalera amb una secció `yield` per al títol, i una altra per al contingut de la pàgina, com la de l'exemple que hem vist anteriorment.
- Defineix en un arxiu a part en la subcarpeta `partials`, anomenat `nav.blade.php`, una barra de navegació que ens permeta accedir a aquestes direccions de moment:
  - Pàgina d'inici
  - Llistat de posts

Exercicis vistes i rutes

Exercici 3

- Inclou la barra de navegació en la plantilla base que has definit abans.
- A partir de la plantilla base, defineix altres dues vistes en una subcarpeta posts, anomenades `posts/llistat.blade.php` i `posts/fitxa.blade.php`. Com a títol de cada pàgina posa un breu text del que són (per exemple, “Llistat posts” i “Fitxa post”), i com a contingut de moment deixa un encapçalat h1 que indique la pàgina en la qual estem: “Llistat de posts” o “Fitxa del post XXXX”, on XXXX serà l'identificador del post que haurem passat per la URL (i que hauràs de passar a la vista). Fes que les rutes corresponents de `routes/web.php` que ja has definit renderitzen aquestes vistes en lloc de retornar text pla.

Exercicis vistes i rutes

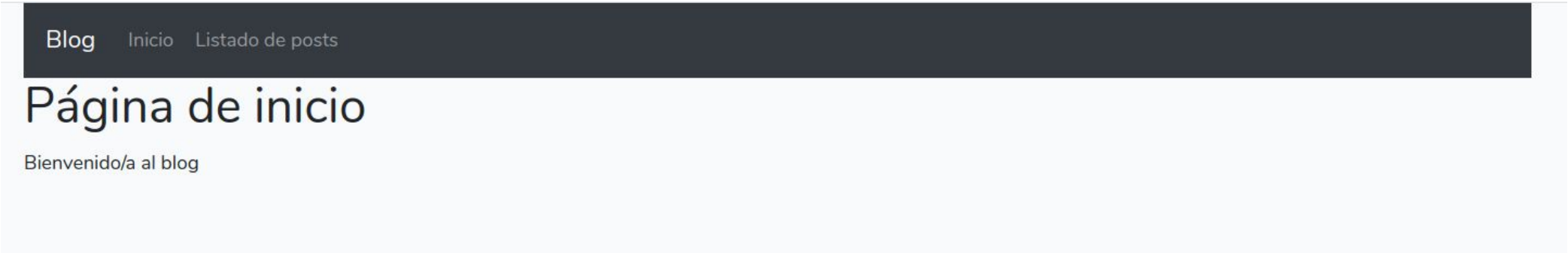
Exercici 4

- Sobre el mateix projecte blog que venim desenvolupant, incorpora ara els estils de Bootstrap seguint els passos vistos.
- Instal·la amb composer la llibreria laravel/ui, i utilitza-la per a incorporar Bootstrap al projecte
- Descàrrega Bootstrap amb npm install, i actualitza els arxius CSS i JavaScript amb npm run dev

Exercicis vistes i rutes

Exercici 4

- Incorpora els estils /css/app.css a la plantilla base del projecte, perquè els utilitzen totes les vistes que hereten d'ella.
- Edita l'arxiu partials/nav.blade.php per a modificar la barra de navegació i deixar-la amb un estil particular de Bootstrap. Pots consultar aquesta pàgina per a prendre idees d'alguns dissenys que pots aplicar en la barra de navegació.
- Canvia de nom l'arxiu welcome.blade.php a inici.blade.php i canvia-ho perquè també herete de la plantilla base. Afig algun text introductori com a contingut. Pot quedar-te més o menys així :



Atribucions

Curs de Nacho Iborra