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 3

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

3. Controladors

    3.1. Controladors i tipus

    3.2. Injección de dependències

3. Controladors

Controladors

- S'encarreguen de llevar-li la lògica a les vistes com accés a dades, validació de formularis.
- Per crear un controlador utilitzarem el comandament:

```
> php artisan make:controller PruebaController
```

- Generarà una classe buida a la `carpeta app/Http/Controllers`

Controladors

Controladors d'un sol mètode (invoke)

- Make:controller admet alguns paràmetres addicionals.
- Un d'ells és -i que permet crear la classe amb el mètode __invoke()
- Aquest mètode s'executa automàticament al invocar la classe.

```
> php artisan make:controller PruebaController -i
```

Controladors

Controladors d'un sol mètode (invoke)

- El comandament anterior crea una classe com la següent:

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PruebaController extends Controller
{
    ...
    public function __invoke(Request $request)
    {
        ...
    }
}
```

Controladors

Controladors d'un sol mètode (invoke)

- En el mètode invoke() es pot definir la lògica de dades que necessita o genera una vista:

```
public function __invoke(Request $request)
    {
        $datos = array(...);
        return view('miVista', compact('datos'));
    }
```

Controladors

Controladors d'un sol mètode (invoke)

- En les rutes, li indiquem el controlador.

```
use App\Http\Controllers\PruebaController;
...
Route::get('prueba', PruebaController::class)->name('prueba');
```

```
> php artisan make:controller PruebaController -r
```

Controladors

Controladors amb més d'un mètode

- Els mètodes amb un únic mètode no solen ser els més comuns. Normalment, necessitem tenir mètodes per a llistar, mostrar, eliminar, ...
- Existeix un altre paràmetre en make:controller , -r (resource), per crear un controlador amb les operacions principals:

`index`: mostrar llistat

`store`: emmagatzemar el form d'alta

`edit`: formulari d'edició

`destroy`: eliminar

`create`: mostrar un formulari d'alta

`show`: detall d'un recurs

`update`: emmagatzema form d'edit

Controladors

Controladors amb més d'un mètode

- Ara, per a poder invocar un mètode en concret des de la ruta ho farem de la següent forma:

```
use App\Http\Controllers\PruebaController;
...
Route::get('prueba', [PruebaController::class, 'index'])
   ->name('listado_prueba');
```

Controladors

Controladors amb més d'un mètode

- Si continuem amb el nostre exemple de la biblioteca, faríem:

```
> php artisan make:controller -r LibroController
```

- De moment implementaren els mètodes d'index i show:

Controladors

```
public function index()
{
    $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'));
}

public function show($id)
{
    return "Mostrando ficha de libro $id";
}
```

Controladors

Controladors amb més d'un mètode

Per últim, actualitzem les rutes:

```
use App\Http\Controllers\LibroController;
...
Route::get('libros', [LibroController::class, 'index']);
Route::get('libros/{id}', [LibroController::class, 'show']);
```

Controladors

Rutes, vistes i controladors

- A mesura que el projecte va creixent, anirem creant més vistes associades a controladors.
- Una bona pràctica és crear una carpeta per a cada *resource*.
- En el cas del nostre exemple, crearíem la subcarpeta `resources/views/libros`.
- Dins d'aquesta subcarpeta tindríem:
  - `index.blade.php`
  - `show.blade.php`
  - …

Controladors

Rutes, vistes i controladors

Ara, si volem renderitzar la vista show des d'un controlador o vista, faríem el següent:

```
public function show($id)
{
    return view('libros.show', compact('id'));
}
```

Controladors

Unificar totes les rutes d'un controlador

Podem agrupar tots els mètodes d'un controlador utilitzant el mètode `resource` de la classe `Route` en lloc d'utilitzar `get`.

```
use App\Http\Controllers\LibroController;
...
Route::resource('libros', LibroController::class);
```

Controladors

Unificar totes les rutes d'un controlador

- **IMPORTANT**: a través de la ruta `Route::resource` només s'inclouen els mètodes estàndard d'un controlador de recursos (`index, show, create, edit, etc`), però NO els que afegim a mà després en aquest controlador amb altres noms.
- També és possible indicar els mètodes que volem amb el mètode `only`

```
use App\Http\Controllers\LibroController;
...
Route::resource('libros', LibroController::class)
   ->only(['index', 'show']);
```

Controladors

Unificar totes les rutes d'un controlador

- Des del costat oposat, tenim disponible el mètode `except` per a indicar que es generen totes les rutes excepte aquelles per als mètodes indicats:

```
use App\Http\Controllers\LibroController;
...
Route::resource('libros', LibroController::class)
   ->except(['update', 'edit']);
```

Controladors

Reanoment les rutes

- Al crear les rutes automàtiques, genera els noms `create, edit,` ... Podem canviar aquests noms editant el proveïdor de serveis `AppServiceProvider`, ubicat en `app/Providers`.
- En el mètode `boot` utilitzarem el mètode `resourceVerbs` de la classe `Route`

```
Use Illuminate\Support\Facades\Route
…
public function boot()
{
    Route::resourceVerbs([
        'create' => 'crear',
        'edit' => 'editar'
    ]);
}
```

Controladors

Exemple biblioteca

- Continuant amb el nostre exemple de la biblioteca utilitzarem els mètodes create i edit.

```
public function create()
{
    return "Formulario de inserción de libros";
}

public function edit()
{
    return "Formulario de edición de libros";
}
```

Controladors

Exemple biblioteca

- Deixarem aquesta instrucció per gestionar totes les rutes:

```
Route::resource('libros', LibroController::class)
->only(['index', 'show', 'create', 'edit']);
```

Controladors

Exemple biblioteca

- Ara, el nostre projecte gestionarà les següents rutes:
  - http://127.0.0.1:8000/libros (cridarà a index)
  - http://127.0.0.1:8000/libros/3 (cridarà a show per al llibre 3)
  - http://127.0.0.1:8000/libros/crear (cridarà a create)
  - http://127.0.0.1:8000/libros/3/editar (cridarà a edit per al llibre 3)

- Podem consultar totes les rutes del nostre projecte amb el comandament:

```
php artisan route:list
```

Injecció de dependències

- Consisteix en un mecanisme que facilita recursos als diferents components de l'aplicació, i és una cosa que ja hem utilitzat, sense saber-ho, en els mètodes que s'han generat per als controladors.

Injecció de dependències

Injectar la petició de l'usuari

- Quan definim un mètode en un controlador que necessita processar una petició, se li passa com a paràmetre un objecte de tipus **Request**. Automàticament, Laravel processa el tipus de dada i obté l'objecte associat

```
class LibroController extends Controller
{
    ...
    public function store(Request $request)
    {
        ...
    }
}
```

Injecció de dependències

Injectar la resposta del servidor

- Igual que tenim un objecte **Request** per a obtindre dades de la petició, també existeix un **Response** per a gestionar la resposta. Laravel proporciona un mètode response al qual li podem passar diversos paràmetres:
  - El contingut de la resposta
  - El codi d'estat HTTP de resposta (si no s'especifica, per defecte és 200)
  - Un array amb les capçaleres de resposta (per defecte està buit).

Injecció de dependències

Injectar la petició de l'usuari

- Podem respondre així:

```
response("Mensaje de respuesta", 201)
   ->header('Cabecera1', 'Valor1')
   ->header('Cabecera2', 'Valor2');
```

```
return response()->json(['datos' => datos], 201)
   ->header('Cabecera1', 'Valor1')
   ...;
```

Injecció de dependències

Redireccions

- Podem fer **redireccions** en les respostes així:

```
redirect('/');
```

```
redirect()->route('inicio');
```

- Si volem **passar valor per sessió**:

```
redirect()->route('inicio')
   ->with('mensaje', 'Mensaje enviado correctamente');
```

Injecció de dependències

Redireccions

- Per accedir a les **sessions** fem:

```
@if(session()->has('mensaje'))
    {{ session('mensaje') }}
@endif
```

- Si fem una **redirecció** des d'un mètode d'un controlador hem de fer un `return`

```
    public function store(...)
    {
        ...
        return redirect()->route('libros.index');
    }
```

Injecció de dependències

Helpers

- Un helper és bàsicament una **funció d'utilitat** que podem voler utilitzar en diversos punts de la nostra web, i que necessitem tindre localitzada i compartida.
- Per exemple, imaginem que volem ressaltar en el nostre menú de navegació l'opció que tenim actualment visible.
- Suposem que la classe CSS per a identificar el menú actiu es diu `activo`. En aquest cas, per a un menú de diverses opcions com aquest, n'hi ha prou amb utilitzar el mètode `routeIs` de la petició (`request`) per a comprovar si la ruta coincideix amb cada menú

Injecció de dependències

Helpers

```
<nav>
    <ul>
        <li class="{{ request()->routeIs('inicio') ? 'activo' : '' }}">
            <a href="/">Inicio</a>
        </li>
        <li class="{{ request()->routeIs('contacto') ? 'activo' : ''
}}">
            <a href="/contacto">Contacto</a>
        </li>
        ...
    </ul>
</nav>
```

Injecció de dependències

Helpers

- Utilitzant l'arxiu `helpers.php` i ubicar-lo en la mateixa carpeta `app`, definim la funció:

```
function setActivo($nombreRuta)
{
    return request()->routeIs($nombreRuta) ? 'activo' : '';
}
```

Injecció de dependències

Helpers

- Des de la vista simplement cridem a la funció:

```
<nav>
    <ul>
        <li class="{{ setActivo('inicio') }}">
            <a href="/">Inicio</a>
        </li>
        <li class="{{ setActivo('contacto') }}">
            <a href="/contacto">Contacto</a>
        </li>
        ...
    </ul>
</nav>
```

Injecció de dependències

Helpers

- Si volem mantindre l'enllaç actiu per a qualsevol subruta, utilitzem `*`

```
<li class="{{ setActivo('libros.*') }}">
   <a href="{{ route('libros') }}">Libros</a>
</li>
```

Injecció de dependències

Helpers

- Per últim, per que Laravel carregue l'arxiu `helpers.php`, haurem d'indicar-ho a l'arxiu `composer.json`, a la secció autoload i afegir la secció `files`:

```
"autoload": {
    "classmap": [ ... ],
    "psr-4": { ... },
    "files": ["app/helpers.php"]
},
```

- Per últim executem:

```
composer dump-autoload
```

Exercicis

Exercici 1

1.1. Crea un **controlador de recursos** (opció -r) anomenat **PostController**, que ens servirà per a gestionar tota la lògica dels posts del blog.

1.2. Assigna automàticament amb el mètode **resource** cada ruta a la seua funció corresponent del controlador, en l'arxiu **routes/web.php**. Limita amb **only** les accions només a les funcions de **llistat (index), fitxa (show), creació (create) i edició (edit)**.

1.3. Utilitza el proveïdor de serveis **AppServiceProvider** per a "**castellanitzar**" les rutes de creació i edició, com en l'exemple que hem vist de llibres.

Exercicis

1.4. Canvia de nom les **vistes** de **llistat** i **fitxa** d'un post a **index.blade.php** i **show.blade.php**, dins de la seua carpeta **posts**, i fes que els mètodes corresponents del controlador de posts renderitzen aquestes vistes. Per als mètodes **create** i **edit**, simplement retorna un text pla indicant “Nou post” i “Edició de post”, per exemple.

1.5. Fes els **canvis addicionals** que siguen convenients (per exemple, en el menú de navegació) perquè els enllaços continuen funcionant, i prova que les quatre rutes (llistat, fitxa, creació i edició) funcionen adequadament.

Exercicis

Exercici 2

2.1. Fes que les funcions de **create** i **edit** del controlador de posts, en lloc de mostrar un missatge de text pla indicant que ací va un formulari, **redirigisquen** a la pàgina **d'inici**, usant la instrucció redirect.

2.2. Afig un **helper** al projecte que definisca una funció anomenada **fechaActual**. Rebrà com a paràmetre un format de data (per exemple, “d/m/i”) i traurà la data actual en aquest format. Utilitza-ho per a mostrar la data actual en format “d/m/l” en la plantilla base, sota la barra de navegació, alineada a la dreta.