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 4

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

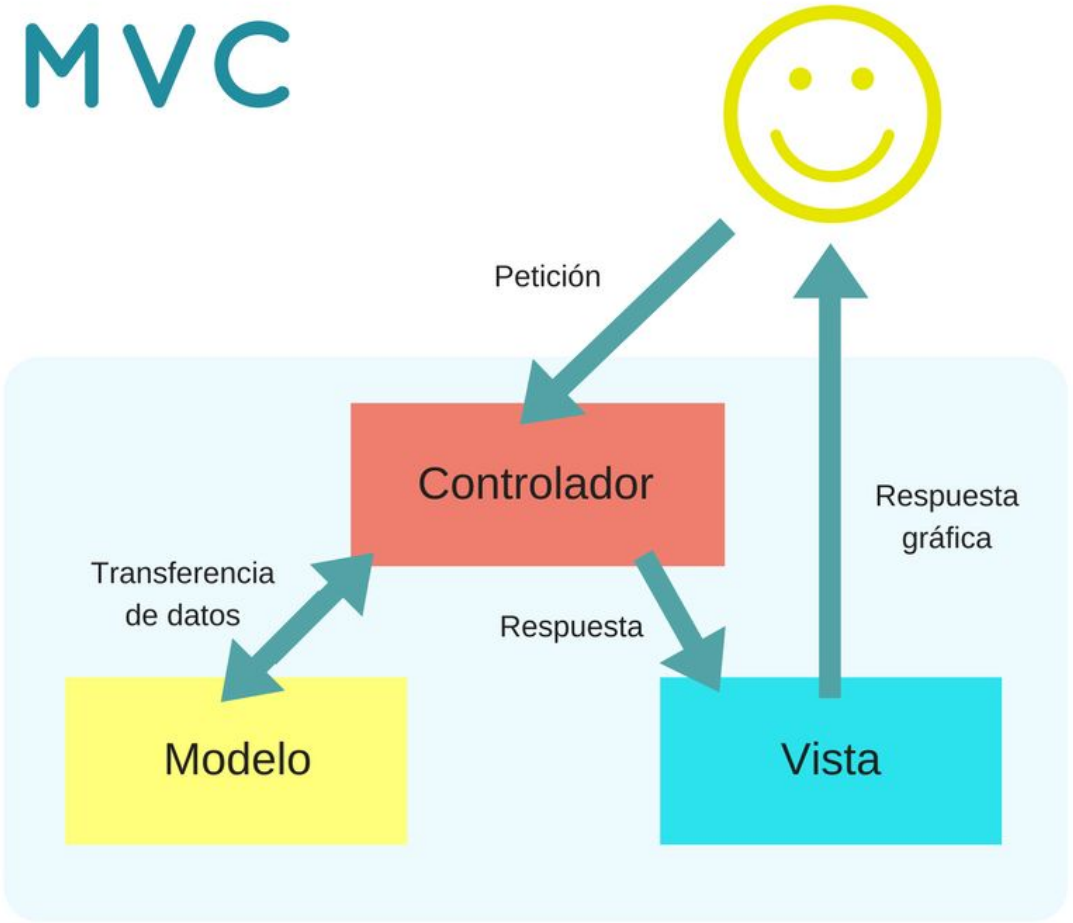
4. El model de dades
    - 4.1. Accés a bases de dades
    - 4.2. Les migracions
    - 4.3. Els models
    - 4.4. Relacions entre models
    - 4.5. Seeders i factories
    - 4.6. Query Builders i ús de dates

4. El model de dades

Accés a la base de dades

Fins ara hem vist les vistes i controladors del MVC.

El model de dades ofereix accés a la base de dades per obtenir les dades, però també creació de taules a partir del model.



Accés a la base de dades

Paràmetres de connexió

Es localitzen a l'arxiu .env. I podem configurar:

- **`DB_CONNECTION`**: tipus de SGBD a utilitzar
- **`DB_HOST`**: adreça o IP del SGBD (127.0.0.1 per a connexió local)
- **`DB_PORT`**: per a MySQL es 3306
- **`DB_DATABASE`**: nom de la base de dades
- **`DB_USERNAME`**: usuari
- **`DB_PASSWORD`**: pass

Accés a la base de dades

Creació de la base de dades

Des de Laravel podem crear taules i accedir a la informació, però no crear la base de dades.

Per fer-ho, haurem de fer-ho des de phpMyAdmin o qualsevol altre client de bases de dades.

Migracions

Estructura de les migracions

Les migracions permeten generar l'estructura completa de la base de dades.

A més crea un tipus de control de versions de la base de dades així com modificar-la d'una manera senzilla.

Per defecte hi ha una sèrie de migracions que podem trobar a `database/migrations`. Aquestes migracions per defecte podem borrar-les o deixar-les per si volem fer alguna modificació sobre alguna d'elles, per exemple, la d'usuaris.

Migracions

Estructura de les migracions

Les migracions han de tenir dos mètodes:

- `up`: permet afegir taules, columnes o índex.
- `down`: desfa les operacions realitzades en el mètode anterior.

```
public function up()
{
    Schema::create('usuarios', function(BluePrint $tabla) {
        $tabla->id(); //numèric i autoincrement
        $tabla->string('nombre');
        $tabla->string('email')->unique();
        ...
        $tabla->timestamps();
    });
}
```

Migracions

Estructura de les migracions

- També podem indicar la propietat `nullable()` per fer que un camp puga ser nul.
- També podem definir camps com `text()` o `longText()`.
- En la documentación oficial, tenim tota la informació sobre els possibles tipus de camps.
- Si necessitem tenir clau primària composta, ho farem així:

```
$table->primary(['campo1', 'campo2']);
```

Migracions

Creació de migracions

- Per poder crear una migració faríem alguna cosa així:

```
php artisan make:migration crear_tabla_prueba
```

- A l'hora de crear la migració, podem indicar si anem a crear una taula o modificar-la:

```
php artisan make:migration crear_tabla_pedidos --create=pedidos
php artisan make:migration nuevo_campo_usuario --table=usuarios
```

Migracions

Creació de migracions

- Un exemple per afegir un camp telèfon quedaria així:

```
public function up()
{
    Schema::table('usuarios', function(Blueprint $tabla) {
        $tabla->string('telefono')->nullable();
    });
}

public function down()
{
    Schema::table('usuarios', function(Blueprint $tabla) {
        $tabla->dropColumn('telefono');
    });
}
```

Migracions

Execució i borrat de migracions

- Per executar una migració farem:

```
php artisan migrate
```

- Si volem desfer les migracions:

```
php artisan rollback //desfà totes les migracions de l'últim lot
```

```
php artisan rollback --step=2//desfà les últimes 2
```

```
php artisan migrate:fresh //ELIMINA totes les migraciosn i torna a crear-les
```

Migracions

Migracions en el exemple de la biblioteca

- En primer lloc eliminem totes les migracions de la carpeta `database/migrations` a excepció de la taula `create_users_table`
- Modifiquem l'arxiu deixant-ho així:

```
public function up()
{
    Schema::create('usuarios', function(Blueprint $table) {
        $table->id();
        $table->string('login')->unique();
        $table->string('password');
        $table->timestamps();
    });
}
```

Migracions

Migracions en el exemple de la biblioteca

- Ara creem la nova migració per als llibres:

```
php artisan make:migration crear_tabla_libros --create=libros
```

```
public function up()
{
    Schema::create('libros', function(Blueprint $table) {
        $table->id();
        $table->string('titulo');
        $table->string('editorial')->nullable();
        $table->float('precio');
        $table->timestamps();
    });
}
```

Migracions

Migracions en el exemple de la biblioteca

- Carreguem les migracions

```
php artisan migrate
```

El model de dades

Creació del model

- Una vegada tenim creades les taules anem a crear el model i veure com podem manipular les dades utilitzant Eloquent.
- Com vam fer en les unitats anterior, la idea és crear un model per cada taula.
- Per fer-ho de la taula `libros`, farem:

```
php artisan make:model Libro
```

El model de dades

Creació del model

- Per convenció el models es creen en singular amb la primera lletra en majúscules.
- Els models es creen en la carpeta `app\Models`.
- L'estructura bàsica dels models és:

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Libro extends Model
{

}
```

El model de dades

Creació del model

- Automàticaments els models s'associen a una taula amb el mateix nom en minúscula i en plural (`libros` i `usuarios`)
- També podem definir-ho manualment així:

```
class Libro extends Model
{
   protected $table = 'mislibros';
}
```

El model de dades

Altres opcions per crear models

- Podem afegir alguns paràmetres a l'hora de crear els models

```
php artisan make:model Pelicula -m // crea el model i fa la migració
```

```
php artisan make:model Pelicula -mc // també crea el controlador
```

```
php artisan make:model Pelicula -mcr // el controlador té els mètodes propis
```

El model de dades

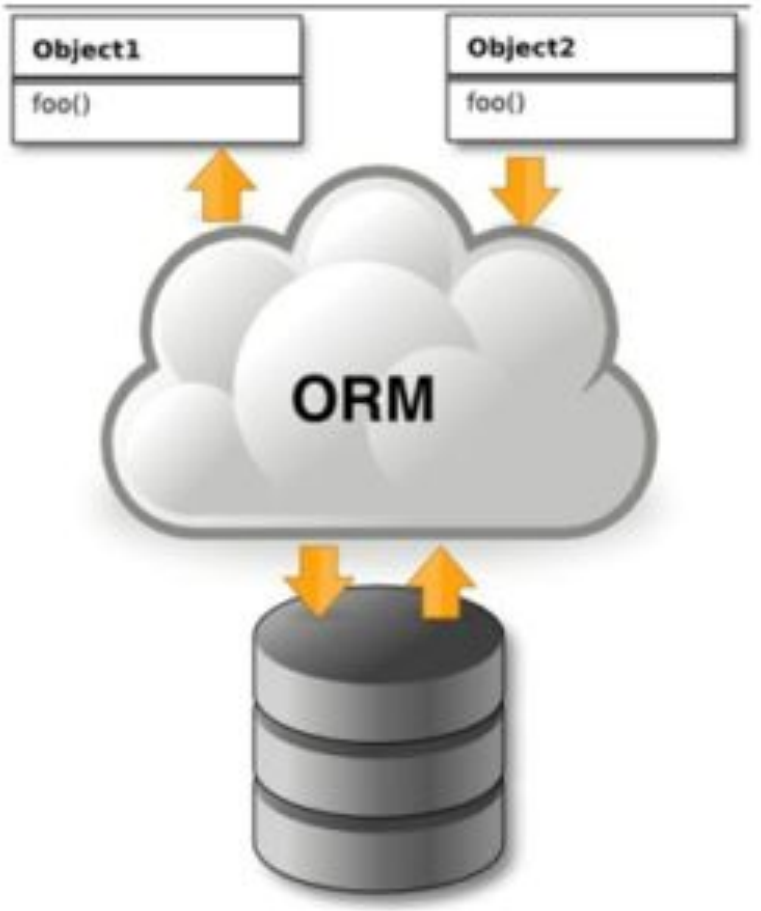
Seguir una nomenclatura uniforme

- Per al model `Libro`
  - Tenim un controlador associat: `LibroController`
  - Hem definit les vistes en `resources/views/libros`
  - El nom de les vistes tenen els noms dels mètodes associats:
    - `index.blade.php`
    - `show.blade.php`
    - ...

Operacions sobre el model

Introducció. Primers passos amb Eloquent

- Eloquent és l'ORM incorporat per defecte en Laravel.
- Un ORM és una tècnica de disseny que permet fer ús de les bases de dades utilitzant la programació orientada a objectes.
- Eloquent implementa les accions `save()`, `update()`, `delete()`, …



Operacions sobre el model

Select

- Una vegada creat el model, ja podem obtenir registres de la base de dades:

```
use App\Models\Libro;
...

class LibroController extends Controller
{
    public function index()
    {
        $libros = Libro::get();
        return view('libros.index', compact('libros'));
    }
}
```

Operacions sobre el model

Select

- A continuació, si volem mostrar els títols en la vista, podem fer el següent:

```
@forelse($libros as $libro)
   {{ $libro->titulo }}
@endforelse
```

Operacions sobre el model

Select

- Si volem filtrar la informació, podem fer ús de l'operador `where`:

```
$libros = Libro::where('precio', '<', 10)->get();
```

```
$libros = Libro::where('precio', '<', 10)
   ->where('precio', '>', 5)->get();
```

- Per ordenar:

```
$libros = Libro::orderBy('titulo')->get();
$libros = Libro::orderBy('titulo', 'DESC')->get();
```

Operacions sobre el model

Paginació dels resultats

- En el controlador:

```
public function index()
{
    $libros = Libro::paginate(5);
    return view('libros.index', compact('libros'));
}
```

Operacions sobre el model

Paginació dels resultats

- En la vista:

```
@forelse($libros as $libro)
   {{ $libro->titulo }}
@endforelse

{{ $libros->links() }}
```

Nota: para posar la paginació en castellà modificar l'arxiu bootstrap-5.blade.php

```
> php artisan vendor:publish --tag=laravel-pagination
```

Operacions sobre el model

Paginació dels resultats

- Si el volem ordenat

```
public function index()
{
    $libros = Libro::orderBy('titulo', ‘asc’)
        ->orderBy('editorial', ‘asc’)
        ->paginate(5);
    return view('libros.index', compact('libros'));
}
```

Operacions sobre el model

Paginació dels resultats

- Per defecte, per crear els botons s'utilitza el framework `Tailwind.css`.
- Si volem fer ús de Bootstrap, haurem d'afegir la següent línia en el mètode `boot` del provider `App\Providers\AppServiceProvider`

```
use Illuminate\Pagination\Paginator;
...
// Bootstrap 5
Paginator::useBootstrapFive();
```

Operacions sobre el model

Detall dels objectes individuals

- Mostrar un enllaç al detall des d'un llistat. Des de la plantilla Blade:

```
@forelse($libros as $libro)
   <li><a href="{{ route('libros.show', $libro) }}">
          {{ $libro->titulo }}
   </a></li>
@endforelse
```

Operacions sobre el model

Detall dels objectes individuals

- En el controlador tindríem:

```
class LibroController extends Controller
{
    ...

    public function show($id)
    {
        $libro = Libro::find($id);
        return view('libros.show', compact('libro'));
    }
}
```

Operacions sobre el model

Detall dels objectes individuals

- La ruta, seria alguna cosa així si no està definit el resource:

```
Route::get('/libros/{id}', [LibroController::class, 'show'])
->name('libros.show');
```

Operacions sobre el model

Detall dels objectes individuals

- Per últim, en el mètode `show()` del controlador, farem ús del mètode `find`() del model:

```
...
class LibroController extends Controller
{
    ...

    public function show($id)
    {
        $libro = Libro::find($id); //alternativament findOrFail → 404
        return view('libros.show', compact('libro'));
    }
}
```

Operacions sobre el model

Insert

- Per fer-ho, farem ús del mètode `save()` heretat d'Eloquent.
- Aquest codi normalment en el mètode `store()`

```
$libro = new Libro();
$libro->titulo = "El juego de Ender";
$libro->editorial = "Ediciones B";
$libro->precio = 8.95;
$libro->save();
```

Operacions sobre el model

Insert

- De manera alternativa, podem utilitzar el mètode create i passar-li totes les dades de la petició que arriben del formulari:

```
Libro::create($request->all());
```

Operacions sobre el model

Insert

- Per poder fer l'anterior s'han de complir dos premises:
  - Cada camp de la petició ha de tenir un nom associat amb el mateix nom en el model
  - Hem de crear en el mètode una propietat amb el nom `$fillable` indicant els camps que ens interessen.

```
class Libro extends Model
{
   protected $fillable = ['titulo', 'editorial', 'precio'];
}
```

Operacions sobre el model

Modificacions

- Les modificacions es realitzen en dos passos:
  - Obtenir l'objecte amb el mètode `findOrFail` a partir de la `id`
  - Modificar les propietats que volem i guardant amb `save`

```
$libroAModificar = Libro::findOrFail($id);
$libroAModificar->titulo="Otro título";
$libroAModificar->save();
```

Operacions sobre el model

Modificacions

- Alternativament, també podem fer:

```
Libro::findOrFail($id)->update($request->all());
```

- Aquest codi es sol posar en el mètode `update` del controlador.

Operacions sobre el model

Eliminar

- De manera similar a la modificació:

```
Libro::findOrFail($id)->delete();
```

- Aquest codi es sol posar en el mètode `destroy` del controlador.
- Després de borrar normalment es redirigeix a l'índex.

```
public function destroy($id)
{
    Libro::findOrFail($id)->delete();
    $libros = Libro::get();
    return view('libros.index', compact('libros'));
}
```

Operacions sobre el model

Eliminar

- Quant a la vista, em de crear un formulari. Quedaria així:

```
<form action="{{ route('libros.destroy', $libro) }}" method="POST">
    @method('DELETE')
    @csrf
    <button>Borrar</button>
</form>
```

Ampliació Bootstrap

Personalització de botons

1. Instal·lar icones bootstrap

```
npm install bootstrap-icons
```

2. Afegir llibreria a el nostre arxiu de configuració css: /resources/css/sass/app.scss

```
@import 'bootstrap-icons/font/bootstrap-icons.css';
```

Ampliació Bootstrap

Personalització de botons

3. Compilar

```
npm run build
```

4. Utilitzar icones

```
<i class="bi bi-pencil"></i>
```

Exercicis

Exercici 1

- Crea una base de dades anomenada blog en el teu servidor de bases de dades a través de phpMyAdmin. Modifica també l'arxiu `.env` del projecte per a accedir a aquesta base de dades amb les credencials adequades, similars a les de l'exemple de la biblioteca (canviant el nom de la base de dades).
- Elimina totes les migracions existents, excepte la de `create_users_table`. Edita aquesta migració de la taula usuaris per a deixar-la igual que l'exemple de la biblioteca (únicament amb els camps `login` i `password`, a més de la `id` i els `timestamps`).

Exercicis

Exercici 1

- Crea una nova migració anomenada `crear_taula_posts`, que crearà una taula anomenada `posts` amb aquests camps:
  - Id autonumèric
  - Títol del post (string)
  - Contingut del post (text)
  - Timestamps per a gestionar automàticament la data de creació o modificació del post
- Llança les migracions i comprova que es creen les taules corresponents amb els camps associats en la base de dades.

Exercicis

Exercici 2

- Modifica si no ho has fet encara el model `User` que ve per defecte perquè es diga `Usuari`, igual que hem fet en l'exemple de la biblioteca. Crea un nou model anomenat `Post` per als posts del nostre blog. Assegura't que tots dos models se situen en la carpeta `App\Models` del projecte.
- Després, modifica els mètodes del controlador `PostController` creat en sessions anteriors, d'aquesta manera:
  - El mètode `index` ha d'obtindre tots els posts de la taula, i mostrar la vista `posts.index` amb eixe llistat de posts.
  - La vista `posts.index`, per part seua, rebrà el llistat de posts i mostrarà els títols de cadascun, i un botó Veure per a mostrar la seua fitxa (`posts.show`).

Exercicis

Exercici 2

- Has de mostrar el llistat de posts ordenat per títol en ordre ascendent, i paginat de 5 en 5.
- El mètode show ha d'obtindre el post del id que es passarà com a paràmetre, i mostrar-lo en la vista `posts.show`.
- La vista `posts.show` rebrà l'objecte amb el post a mostrar, i mostrarem el títol, contingut i data de creació del post, amb el format que vulgues.

Exercicis

Exercici 2

- El mètode `destroy` eliminarà el post que el seu `id` rebrà com a paràmetre, i retornarà la vista `posts.index` amb el llistat actualitzat. Per a provar aquest mètode, recorda que has de definir un formulari en una vista (el pots fer per a cada post mostrat en la vista `posts.index`) que envie a la ruta `posts.destroy` usant un mètode `DELETE`.
- Els mètodes `create`, `edit`, `store` i `update` de moment els deixarem sense fer, fins que vegem com gestionar formularis.

Exercicis

Exercici 2

- Per a simular la `inserció` i la `modificació`, crearem dos mètodes addicionals en el controlador, que usarem de manera temporal:
  - Un mètode anomenat `nuevoPrueba`, que cada vegada que el cridem crearà un post amb un títol a l'atzar (per exemple, “Títol X”, sent X un enter aleatori), i un contingut a l'atzar (“Contingut X”).
  - Un mètode anomenat `editarPrueba`, que rebrà com a paràmetre un `id` i modificarà el títol i contingut del post altres generats aleatòriament, com en el punt anterior.
- Aquests dos mètodes ens serviran per a crear una sèrie de posts de prova que després ens serviran per a provar el llistat i la fitxa dels posts.

Exercicis

Exercici 2

- En l'arxiu `routes/web.php`, recorda afegir dues noves rutes temporals de tipus get per a provar aquestes insercions i modificacions. La primera pot apuntar a `/posts/nuevoPrueba`, per exemple, i la segona a `/posts/editarPrueba/{id}`. Recorda també eliminar o editar la restricció `only` de les rutes del controlador que vas establir la sessió anterior, perquè no sols permeta les rutes `index, show, create i edit`, i a més permeta la de `destroy`

Exercicis

Exercici 2

- **IMPORTANT**: els mètodes `nuevoPrueba` i `editarPrueba` que has creat en `PostController` NO són mètodes estàndard d'un controlador de recursos, i de cap manera estaran disponibles a través de `Route::resource` en `routes/web.php`. Per això has de definir a mà una ruta per a cadascun d'ells en eixe arxiu, a través de `Route::get`, i eixes rutes han de definir-se **ABANS** de la de recursos (`Route::resource`) perquè en cas contrari no s'aparellaran correctament.

Atribucions

Curs de Nacho Iborra

Laracast